mypy.ini

```
[mypy]
python_version = 3.8
ignore_missing_imports = False
mypy_path = timApp/modules/fields
disallow_untyped_calls = True
disallow_incomplete_defs = True
disallow_untyped_defs = True
no_implicit_optional = True
show_column_numbers = True
namespace_packages = True
show_error_codes = True
exclude = timApp/modules/cs/static
# warn_unused_ignores = True

# Ignore errors in tests.
[mypy-timApp.tests.*]
ignore_errors = True

# migrations contains mostly generated code.
[mypy-timApp.migrations.*]
ignore_errors = True

[mypy-timApp.modules.cs.*]
ignore_errors = True

[mypy-timApp.modules.imagex.*]
ignore_errors = True

[mypy-timApp.modules.svn.*]
ignore_errors = True

[mypy-tim_common.fileParams]
ignore_errors = True

[mypy-tim_common.marshmallow_dataclass]
ignore_errors = True

[mypy-tim_common.tim_server]
ignore_errors = True

# Ignore library errors.

[mypy-psycogreen.*]
ignore_missing_imports = True

[mypy-alembic]
ignore_missing_imports = True

[mypy-timApp.celery_sqlalchemy_scheduler.*]
ignore_errors = True

[mypy-tim_common.vendor.*]
ignore_errors = True

[mypy-html5lib]
ignore_missing_imports = True

```

```
[mypy-html5lib.*]
ignore_missing_imports = True

[mypy-celery.schedules]
ignore_missing_imports = True

[mypy-lxml]
ignore_missing_imports = True

[mypy-lxml.*]
ignore_missing_imports = True

[mypy-sqlalchemy]
ignore_missing_imports = True

[mypy-sqlalchemy.dialects]
ignore_missing_imports = True

[mypy-sqlalchemy.exc]
ignore_missing_imports = True

[mypy-sqlalchemy.orm]
ignore_missing_imports = True

[mypy-webargs.flaskparser]
ignore_missing_imports = True

[mypy-flask_wtf]
ignore_missing_imports = True

[mypy-isodate]
ignore_missing_imports = True

[mypy-bs4]
ignore_missing_imports = True

[mypy-sqlalchemy.dialects.postgresql]
ignore_missing_imports = True

[mypy-mailmanclient]
ignore_missing_imports = True

# The ignore_errors below should be gradually removed as the code is fixed.

[mypy-timApp.admin.migrate_to_postgre]
ignore_errors = True

[mypy-timApp.answer.answer]
ignore_errors = True

[mypy-timApp.answer.answer_models]
ignore_errors = True

[mypy-timApp.answer.feedbackanswer]
ignore_errors = True

[mypy-timApp.answer.routes]
ignore_errors = True

```

```
[mypy-timApp.auth.accesshelper]
ignore_errors = True

[mypy-timApp.auth.auth_models]
ignore_errors = True

[mypy-timApp.auth.saml]
ignore_errors = True

[mypy-timApp.auth.sessioninfo]
ignore_errors = True

[mypy-timApp.bookmark.bookmarks]
ignore_errors = True

[mypy-timApp.bookmark.routes]
ignore_errors = True

[mypy-timApp.defaultconfig]
ignore_errors = True

[mypy-timApp.document.attributeparser]
ignore_errors = True

[mypy-timApp.document.changelog]
ignore_errors = True

[mypy-timApp.document.changelogentry]
ignore_errors = True

[mypy-timApp.document.create_item]
ignore_errors = True

[mypy-timApp.document.docinfo]
ignore_errors = True

[mypy-timApp.document.docparagraph]
ignore_errors = True

[mypy-timApp.document.docsettings]
ignore_errors = True

[mypy-timApp.document.document]
ignore_errors = True

[mypy-timApp.document.documentparser]
ignore_errors = True

[mypy-timApp.document.documentparseroptions]
ignore_errors = True

[mypy-timApp.document.documents]
ignore_errors = True

[mypy-timApp.document.documentversion]
ignore_errors = True

[mypy-timApp.document.documentwriter]
ignore_errors = True
```

```
176
177 [mypy-timApp.document.editing.clipboard]
178 ignore_errors = True
179
180 [mypy-timApp.document.editing.documenteditresult]
181 ignore_errors = True
182
183 [mypy-timApp.document.editing.editrequest]
184 ignore_errors = True
185
186 [mypy-timApp.document.editing.proofread]
187 ignore_errors = True
188
189 [mypy-timApp.document.editing.routes]
190 ignore_errors = True
191
192 [mypy-timApp.document.editing.routes_clipboard]
193 ignore_errors = True
194
195 [mypy-timApp.document.exceptions]
196 ignore_errors = True
197
198 [mypy-timApp.document.post_process]
199 ignore_errors = True
200
201 [mypy-timApp.document.randutils]
202 ignore_errors = True
203
204 [mypy-timApp.document.routes]
205 ignore_errors = True
206
207 [mypy-timApp.document.timjsonencoder]
208 ignore_errors = True
209
210 [mypy-timApp.document.translation.routes]
211 ignore_errors = True
212
213 [mypy-timApp.document.translation.synchronize_translations]
214 ignore_errors = True
215
216 [mypy-timApp.document.translation.translation]
217 ignore_errors = True
218
219 [mypy-timApp.document.validationresult]
220 ignore_errors = True
221
222 [mypy-timApp.document.version]
223 ignore_errors = True
224
225 [mypy-timApp.document.yamlblock]
226 ignore_errors = True
227
228 [mypy-timApp.errorhandlers]
229 ignore_errors = True
230
231 [mypy-timApp.folder.folder]
232 ignore_errors = True
233
234 [mypy-timApp.folder.folder_view]
```

```
ignore_errors = True

[mypy-timApp.gamification.gamificationdata]
ignore_errors = True

[mypy-timApp.gamification.generateMap]
ignore_errors = True

[mypy-timApp.item.block]
ignore_errors = True

[mypy-timApp.item.copy_rights]
ignore_errors = True

[mypy-timApp.item.item]
ignore_errors = True

[mypy-timApp.item.manage]
ignore_errors = True

[mypy-timApp.item.partitioning]
ignore_errors = True

[mypy-timApp.item.routes]
ignore_errors = True

[mypy-timApp.item.routes_tags]
ignore_errors = True

[mypy-timApp.item.tag]
ignore_errors = True

[mypy-timApp.item.validation]
ignore_errors = True

[mypy-timApp.lecture.askedjson]
ignore_errors = True

[mypy-timApp.lecture.askedquestion]
ignore_errors = True

[mypy-timApp.lecture.lecture]
ignore_errors = True

[mypy-timApp.lecture.lectureanswer]
ignore_errors = True

[mypy-timApp.lecture.message]
ignore_errors = True

[mypy-timApp.lecture.routes]
ignore_errors = True

[mypy-timApp.lecture.useractivity]
ignore_errors = True

[mypy-timApp.markdown.dumboclient]
ignore_errors = True

```

```
[mypy-timApp.markdown.markdownconverter]
ignore_errors = True

[mypy-timApp.note.notes]
ignore_errors = True

[mypy-timApp.note.usernote]
ignore_errors = True

[mypy-timApp.notification.notification]
ignore_errors = True

[mypy-timApp.notification.notify]
ignore_errors = True

[mypy-timApp.notification.pending_notification]
ignore_errors = True

[mypy-timApp.plugin.plugin]
ignore_errors = True

[mypy-timApp.plugin.pluginControl]
ignore_errors = True

[mypy-timApp.plugin.routes]
ignore_errors = True

[mypy-timApp.plugin.taskid]
ignore_errors = True

[mypy-timApp.plugin.timtable.row_owner_info]
ignore_errors = True

[mypy-timApp.plugin.timtable.timTable]
ignore_errors = True

[mypy-timApp.plugin.timtable.timTableLatex]
ignore_errors = True

[mypy-timApp.plugin.qst.qst]
ignore_errors = True

[mypy-timApp.printing.documentprinter]
ignore_errors = True

[mypy-timApp.printing.pandoc_headernumberingfilter]
ignore_errors = True

[mypy-timApp.printing.pandoc_imagefilepathsfilter]
ignore_errors = True

[mypy-timApp.printing.pandoc_inlinestylesfilter]
ignore_errors = True

[mypy-timApp.printing.print]
ignore_errors = True

[mypy-timApp.printing.printeddoc]
ignore_errors = True
```

```
353
354 [mypy-timApp.readmark.readings]
355 ignore_errors = True
356
357 [mypy-timApp.readmark.readmarkcollection]
358 ignore_errors = True
359
360 [mypy-timApp.readmark.readparagraph]
361 ignore_errors = True
362
363 [mypy-timApp.readmark.readparagraphtype]
364 ignore_errors = True
365
366 [mypy-timApp.readmark.routes]
367 ignore_errors = True
368
369 [mypy-timApp.slide.routes]
370 ignore_errors = True
371
372 [mypy-timApp.slide.slidestatus]
373 ignore_errors = True
374
375 [mypy-timApp.tim]
376 ignore_errors = True
377
378 [mypy-timApp.tim_app]
379 ignore_errors = True
380
381 [mypy-timApp.tim_celery]
382 ignore_errors = True
383
384 [mypy-timApp.timdb.init]
385 ignore_errors = True
386
387 [mypy-timApp.timdb.sqa]
388 ignore_errors = True
389
390 [mypy-timApp.timdb.timdb]
391 ignore_errors = True
392
393 [mypy-timApp.timtypes]
394 ignore_errors = True
395
396 [mypy-timApp.upload.upload]
397 ignore_errors = True
398
399 [mypy-timApp.upload.uploadedfile]
400 ignore_errors = True
401
402 [mypy-timApp.user.consentchange]
403 ignore_errors = True
404
405 [mypy-timApp.user.groups]
406 ignore_errors = True
407
408 [mypy-timApp.user.hakaorganization]
409 ignore_errors = True
410
411 [mypy-timApp.user.newuser]
```

```
ignore_errors = True

[mypy-timApp.user.personaluniquecode]
ignore_errors = True

[mypy-timApp.user.preferences]
ignore_errors = True

[mypy-timApp.user.scimentity]
ignore_errors = True

[mypy-timApp.user.user]
ignore_errors = True

[mypy-timApp.user.usergroup]
ignore_errors = True

[mypy-timApp.user.usergroupmember]
ignore_errors = True

[mypy-timApp.user.users]
ignore_errors = True

[mypy-timApp.util.flask.ReverseProxied]
ignore_errors = True

[mypy-timApp.util.flask.cache]
ignore_errors = True

[mypy-timApp.util.flask.filters]
ignore_errors = True

[mypy-timApp.util.flask.responsehelper]
ignore_errors = True

[mypy-timApp.util.flask.search]
ignore_errors = True

[mypy-timApp.util.get_fields]
ignore_errors = True

[mypy-timApp.util.pdftools]
ignore_errors = True

[mypy-timApp.velp.annotation_model]
ignore_errors = True

[mypy-timApp.velp.annotations]
ignore_errors = True

[mypy-timApp.velp.velp]
ignore_errors = True

[mypy-timApp.velp.velp_folders]
ignore_errors = True

[mypy-timApp.velp.velp_models]
ignore_errors = True

```

```
[mypy-timApp.velp.velpgroups]
ignore_errors = True

[mypy-timApp.velp.velps]
ignore_errors = True
```

timApp/Dockerfile

```
FROM ubuntu:20.04

ENV APT_INSTALL="DEBIAN_FRONTEND=noninteractive apt-get -qq update && DEBIAN_FRONTEND=noninteractive apt-get -q install --no-install-recommends -y" \
    APT_CLEANUP="rm -rf /var/lib/apt/lists /dvisvgm-2.4 /usr/share/doc ~/.cache"

# Configure timezone and locale
RUN bash -c "${APT_INSTALL} locales tzdata && ${APT_CLEANUP}"
RUN locale-gen en_US.UTF-8 && bash -c "${APT_CLEANUP}"
ENV LANG=en_US.UTF-8 \
    LANGUAGE=en_US:en \
    LC_ALL=en_US.UTF-8
RUN echo "Europe/Helsinki" > /etc/timezone; dpkg-reconfigure -f noninteractive tzdata && bash -c "${APT_CLEANUP}"

# Install dependencies of texlive-full excluding packages that are not needed (such as documentation files).
# This almost-full installation of TeX Live is needed for the latex-pdf printing functionality, as
# TeX Live doesn't have an (MiKTeX/MacTeX-esque) auto-install functionality for missing LaTeX packages,
# i.e. the whole package archive needs to be pre-installed or the set of usable packages needs to be
# severely limited.
RUN bash -c "${APT_INSTALL} \
biber \
ca-certificates \
cm-super \
dvidvi \
dvipng \
feynmf \
fonts-texgyre \
fragmaster \
latex-cjk-all \
latexmk \
lcdf-typetools \
lmodern \
psutils \
purifyeps \
t1utils \
tex-gyre \
texlive-base \
texlive-bibtex-extra \
texlive-binaries \
texlive-extra-utils \
texlive-font-utils \
texlive-fonts-extra \
texlive-fonts-recommended \
texlive-formats-extra \
texlive-games \
texlive-humanities \
```

```
texlive-lang-arabic \
texlive-lang-chinese \
texlive-lang-cjk \
texlive-lang-cyrillic \
texlive-lang-czechslovak \
texlive-lang-english \
texlive-lang-european \
texlive-lang-french \
texlive-lang-german \
texlive-lang-greek \
texlive-lang-italian \
texlive-lang-japanese \
texlive-lang-korean \
texlive-lang-other \
texlive-lang-polish \
texlive-lang-portuguese \
texlive-lang-spanish \
texlive-latex-base \
texlive-latex-extra \
texlive-latex-recommended \
texlive-luatex \
texlive-metapost \
texlive-music \
texlive-pictures \
texlive-pstricks \
texlive-publishers \
texlive-science \
texlive-xetex \
wget \
&& ${APT_CLEANUP}"

# Update dvisvgm so that it supports converting PDFs to SVGs
RUN bash -c "${APT_INSTALL} gcc g++ libgs-dev libkpathsea-dev pkg-config ↩
    libfreetype6-dev make && ${APT_CLEANUP}"
RUN FILE=`mktemp`; wget ↩
    "https://github.com/mgieseki/dvisvgm/releases/download/2.10/dvisvgm-2.10.tar.gz" ↩
    -qO $FILE && \
 tar -xf $FILE && \
 cd dvisvgm-2.10 && \
 ./configure --enable-bundled-libs && \
 make -j4 && \
 make install && \
 cd / && \
 ${APT_CLEANUP} && \
 rm -rf /dvisvgm-2.10

# Install Python, pip and other necessary packages

RUN bash -c "${APT_INSTALL} openssh-server python3.8 && ${APT_CLEANUP}"

# lxml dependencies
# C-parser for PyYAML
# python-magic dependency
RUN bash -c "${APT_INSTALL} \
 git-core \
 libffi-dev \
 libmagic1 \
 libmagickwand-dev \
 libvoikko1 \
```

```
 libxml2-dev \
 libxmlsec1-dev \
 libxmlsec1-openssl \
 libxslt-dev \
 libyaml-dev \
 python3.8-dev \
 voikko-fi \
 zlib1g-dev \
 && ${APT_CLEANUP}"

RUN wget -q https://bootstrap.pypa.io/get-pip.py && python3.8 get-pip.py && rm ↩
    get-pip.py

ENV PIP_INSTALL="python3.8 -m pip install"
RUN bash -c "${PIP_INSTALL} wheel setuptools && ${APT_CLEANUP}"
RUN bash -c "${PIP_INSTALL} \
attrs \
autopep8 \
beautifulsoup4 \
bcrypt \
bleach \
celery[redis]==4.4 \
cffi \
cssselect \
docformatter \
filelock \
flask \
flask-apispec \
flask-assets \
flask-caching \
flask-compress \
flask-migrate \
flask-oidc \
flask-openid \
flask-sqlalchemy \
flask-testing \
flask-wtf \
gevent \
git+git://github.com/miracle2k/webassets.git \
gunicorn \
html5lib \
humanize \
isodate \
libsass \
lxml \
`# Use mailmanclient from GIT since version PyPI is not updated with a critical fix ↩
    yet` \
git+https://gitlab.com/mailman/mailmanclient.git@1c895c84b5744b2a79146c613885fe8d0f692ec3 ↩
    \
marshmallow \
marshmallow-enum \
marshmallow_union \
mmh3 \
mypy \
pandocfilters \
pillow \
psycogreen \
psycopg2-binary \
pyaml \
```

```
158 pylatex \
159 pypandoc \
160 python-dateutil \
161 python-magic \
162 python3-saml \
163 pytz \
164 recommonmark \
165 responses \
166 selenium==4.0.0b1 \
167 sphinx \
168 sqlalchemy\<1.4.0 \
169 sqlalchemy-utils \
170 typing-inspect \
171 voikko \
172 wand \
173 webargs==5.5 \
174 && ${PIP_INSTALL} requests --upgrade && ${APT_CLEANUP}"
175
176 # Install Pandoc
177 RUN FILE=`mktemp`; wget ↩
        "https://github.com/jgm/pandoc/releases/download/2.10.1/pandoc-2.10.1-1-amd64.deb" ↩
        -qO $FILE && \
178  dpkg -i $FILE && rm $FILE && bash -c "${APT_CLEANUP}"
179
180 # Set name and email for git.
181 RUN git config --global user.email "agent@docker.com" && git config --global ↩
        user.name "agent"
182
183 RUN mkdir /service
184
185 # Add user `agent` -- we don't want to run anything as root.
186 RUN useradd -M agent
187 RUN chown -R agent /service
188
189 RUN bash -c "${APT_INSTALL} curl pdftk gpg-agent poppler-utils && ${APT_CLEANUP} && ↩
        pdftk --version && curl --version"
190
191 # On current TIM production server, using latest versions of Node and NPM will cause
192 # ./npmi and ./js scripts to fail, causing the following symptoms:
193 #
194 # * ./npmi simply freezes right away, consuming memory until an out of memory ↩
        exception is thrown.
195 # * ./js does not freeze, but it throws a permission error after building scripts ↩
        when trying to copy them to the final
196 #   output directory.
197 #
198 # It has not yet been investigated whether it's because of Node or NPM update.
199 RUN (curl -sL https://deb.nodesource.com/setup_12.x | bash -) && bash -c ↩
        "${APT_INSTALL} nodejs && ${APT_CLEANUP}"
200 RUN npm i npm@6.14.8 -g && bash -c "${APT_CLEANUP}"
201
202 RUN mkdir /var/run/sshd
203 RUN echo 'root:test' | chpasswd
204 RUN sed -i 's/#PermitRootLogin prohibit-password/PermitRootLogin yes/' ↩
        /etc/ssh/sshd_config
205
206 # Flask-Testing does not let us configure host, so we do it here.
207 RUN sed -i "s/port=port, use_reloader=False/host='0.0.0.0', port=port, ↩
        use_reloader=False/" /usr/local/lib/python3.8/dist-packages/flask_testing/utils.py
```

```
# Fix newest Werkzeug incompatibility
RUN sed -i "s/from werkzeug import cached_property/from werkzeug.utils import cached_property/" /usr/local/lib/python3.8/dist-packages/flask_testing/utils.py

RUN wget -q http://mirrors.ctan.org/support/latexmk/latexmk.pl -O /usr/bin/latexmk

# Chromedriver (for running tests)
# Taken from https://github.com/SeleniumHQ/docker-selenium/blob/trunk/NodeChrome/Dockerfile
ARG CHROME_VERSION="google-chrome-stable"
RUN wget -q -O - https://dl-ssl.google.com/linux/linux_signing_key.pub | apt-key add - \
  && echo "deb http://dl.google.com/linux/chrome/deb/ stable main" >> /etc/apt/sources.list.d/google-chrome.list \
  && apt-get update -qqy \
  && apt-get -qqy install unzip \
    ${CHROME_VERSION:-google-chrome-stable} \
  && rm /etc/apt/sources.list.d/google-chrome.list \
  && rm -rf /var/lib/apt/lists/* /var/cache/apt/*

RUN wget -q https://raw.githubusercontent.com/SeleniumHQ/docker-selenium/trunk/NodeChrome/wrap_chrome_binar
    -O /usr/local/bin/wrap_chrome_binary && chmod +x /usr/local/bin/wrap_chrome_binary
RUN /usr/local/bin/wrap_chrome_binary

ARG CHROME_DRIVER_VERSION
RUN if [ -z "$CHROME_DRIVER_VERSION" ]; \
  then CHROME_MAJOR_VERSION=$(google-chrome --version | sed -E "s/.* ([0-9]+)(\.[0-9]+){3}.*/\1/") \
    && CHROME_DRIVER_VERSION=$(wget --no-verbose -O - "https://chromedriver.storage.googleapis.com/LATEST_RELEASE_${CHROME_MAJOR_VERSION}"); \
  fi \
  && echo "Using chromedriver version: "$CHROME_DRIVER_VERSION \
  && wget --no-verbose -O /tmp/chromedriver_linux64.zip https://chromedriver.storage.googleapis.com/$CHROME_DRIVER_VERSION/chromedriver_linux64.zip \
  && rm -rf /opt/selenium/chromedriver \
  && unzip /tmp/chromedriver_linux64.zip -d /opt/selenium \
  && rm /tmp/chromedriver_linux64.zip \
  && mv /opt/selenium/chromedriver /opt/selenium/chromedriver-$CHROME_DRIVER_VERSION \
  && chmod 755 /opt/selenium/chromedriver-$CHROME_DRIVER_VERSION \
  && ln -fs /opt/selenium/chromedriver-$CHROME_DRIVER_VERSION /usr/bin/chromedriver


WORKDIR /service

CMD python3 launch.py

EXPOSE 22
EXPOSE 5000
EXPOSE 5001
```

timApp/answer/routes.py

```
"""Answer-related routes."""
import json
import re
from collections import defaultdict
```

```
from dataclasses import dataclass, field
from datetime import datetime
from typing import Union, List, Tuple, Dict, Optional, Any, Callable, TypedDict, DefaultDict

import requests
from flask import Blueprint, session, current_app
from flask import Response
from flask import request
from marshmallow import validates_schema, ValidationError
from marshmallow.utils import missing
from sqlalchemy import func
from sqlalchemy.orm import lazyload
from webargs.flaskparser import use_args

from timApp.answer.exportedanswer import ExportedAnswer
from timApp.answer.backup import send_answer_backup_if_enabled
from tim_common.markupmodels import GenericMarkupModel
from timApp.answer.answer import Answer
from timApp.answer.answer_models import AnswerUpload
from timApp.answer.answers import get_existing_answers_info, save_answer, get_all_answers, \
    valid_answers_query, valid_taskid_filter
from timApp.auth.accesshelper import verify_logged_in, get_doc_or_abort, verify_manage_access, AccessDenied, \
    verify_admin, get_origin_from_request, is_allowed_ip, verify_ip_ok
from timApp.auth.accesshelper import verify_task_access, verify_teacher_access, verify_seeanswers_access, \
    has_teacher_access, \
    verify_view_access, get_plugin_from_request
from timApp.auth.accesstype import AccessType
from timApp.auth.login import create_or_update_user
from timApp.auth.sessioninfo import get_current_user_id, logged_in, user_context_with_logged_in, \
    get_other_session_users_objs
from timApp.auth.sessioninfo import get_current_user_object, get_current_user_group
from timApp.document.caching import clear_doc_cache
from timApp.document.docentry import DocEntry
from timApp.document.docinfo import DocInfo
from timApp.document.document import Document, dereference_pars
from timApp.document.hide_names import hide_names_in_teacher
from timApp.document.usercontext import UserContext
from timApp.document.viewcontext import ViewRoute, ViewContext, default_view_ctx, OriginInfo
from timApp.item.block import Block, BlockType
from timApp.markdown.dumboclient import call_dumbo
from tim_common.marshmallow_dataclass import class_schema
from timApp.notification.send_email import multi_send_email
from timApp.peerreview.peerreview_utils import has_review_access, get_reviews_for_user, is_peerreview_enabled
from timApp.plugin.containerLink import call_plugin_answer
from timApp.plugin.importdata.importData import MissingUser
from timApp.plugin.jsrunner import jsrunner_run, JsRunnerParams, JsRunnerError
from timApp.plugin.plugin import Plugin, PluginWrap, NEVERLAZY, TaskNotFoundException, find_task_ids, \
    CachedPluginFinder
from timApp.plugin.plugin import find_plugin_from_document
from timApp.plugin.pluginControl import pluginify
from timApp.plugin.pluginexception import PluginException
```

```
from timApp.plugin.plugintype import PluginType
from timApp.plugin.taskid import TaskId, TaskIdAccess
from timApp.tim_app import get_home_organization_group
from timApp.timdb.exceptions import TimDbException
from timApp.timdb.sqa import db
from timApp.user.groups import do_create_group, verify_group_edit_access
from timApp.user.user import User, UserInfo
from timApp.user.user import maxdate
from timApp.user.usergroup import UserGroup
from timApp.user.usergroupmember import UserGroupMember
from timApp.util.answerutil import period_handling
from timApp.util.flask.requesthelper import verify_json_params, get_option, get_consent_opt, RouteException, use_model, \
    get_urlmacros_from_request, NotExist, get_from_url
from timApp.util.flask.responsehelper import json_response, ok_response
from timApp.util.get_fields import get_fields_and_users, MembershipFilter, UserFields, RequestedGroups, \
    ALL_ANSWERED_WILDCARD, GetFieldsAccess
from timApp.util.logger import log_info
from timApp.util.utils import get_current_time, approximate_real_name
from timApp.util.utils import local_timezone
from timApp.util.utils import try_load_json, seq_to_str, is_valid_email
from tim_common.pluginserver_flask import value_or_default
from tim_common.utils import Missing

answers = Blueprint('answers',
                    __name__,
                    url_prefix='')


@answers.route("/savePoints/<int:user_id>/<int:answer_id>", methods=['PUT'])
def save_points(answer_id, user_id):
    answer, _ = verify_answer_access(
        answer_id,
        user_id,
        default_view_ctx,
        require_teacher_if_not_own=True,
    )
    tid = TaskId.parse(answer.task_id)
    d = get_doc_or_abort(tid.doc_id)
    points, = verify_json_params('points')
    try:
        plugin, _ = Plugin.from_task_id(
            answer.task_id,
            user_ctx=user_context_with_logged_in(None),
            view_ctx=default_view_ctx,
        )
    except PluginException as e:
        raise RouteException(str(e))
    a = Answer.query.get(answer_id)
    try:
        points = points_to_float(points)
    except ValueError:
        raise RouteException('Invalid points format.')
    try:
        a.points = plugin.validate_points(points) if not has_teacher_access(d) else points
    except PluginException as e:
        raise RouteException(str(e))
```

```
112     a.last_points_modifier = get_current_user_group()
113     db.session.commit()
114     return ok_response()
115
116
117 @dataclass
118 class AnswerIdModel:
119     answer_id: int
120
121
122 @dataclass
123 class ValidityModel(AnswerIdModel):
124     valid: bool
125
126
127 @dataclass
128 class DeleteCollabModel(AnswerIdModel):
129     user_id: int
130
131
132 @answers.route("/answer/saveValidity", methods=['PUT'])
133 @use_model(ValidityModel)
134 def save_validity(m: ValidityModel):
135     a, doc_id = verify_answer_access(
136         m.answer_id,
137         get_current_user_object().id,
138         default_view_ctx,
139         require_teacher_if_not_own=True,
140     )
141     verify_teacher_access(get_doc_or_abort(doc_id))
142     a.valid = m.valid
143     db.session.commit()
144     return ok_response()
145
146
147 @answers.route("/answer/delete", methods=['POST'])
148 @use_model(AnswerIdModel)
149 def delete_answer(m: AnswerIdModel):
150     """Deletes an answer.
151
152     This does not completely delete the answer but only removes user associations ↩
        from it,
153     so it is no longer visible in TIM.
154     """
155     a, doc_id = verify_answer_access(
156         m.answer_id,
157         get_current_user_object().id,
158         default_view_ctx,
159         require_teacher_if_not_own=True,
160     )
161     verify_teacher_access(get_doc_or_abort(doc_id))
162     verify_admin()
163     unames = [u.name for u in a.users_all]
164     a.users_all = []
165     db.session.commit()
166     u = get_current_user_object()
167     log_info(f'{u.name} deleted answer {a.id} (of {seq_to_str(unames)}) in task ↩
        {a.task_id}')
168     return ok_response()
```

```
169
170
171 @answers.route("/answer/deleteCollaborator", methods=['POST'])
172 @use_model(DeleteCollabModel)
173 def delete_answer_collab(m: DeleteCollabModel):
174     """Deletes an answer collaborator.
175     """
176     a, doc_id = verify_answer_access(
177         m.answer_id,
178         get_current_user_object().id,
179         default_view_ctx,
180         require_teacher_if_not_own=True,
181     )
182     verify_teacher_access(get_doc_or_abort(doc_id))
183     verify_admin()
184     collab_to_remove = User.get_by_id(m.user_id)
185     if not collab_to_remove:
186         raise RouteException(f'Answer {m.answer_id} does not have collaborator ↩
    {m.user_id}')
187     a.users_all.remove(collab_to_remove)
188     db.session.commit()
189     u = get_current_user_object()
190     log_info(f'{u.name} deleted collaborator {collab_to_remove.name} from answer ↩
    {a.id} in task {a.task_id}')
191     return ok_response()
192
193
194 def points_to_float(points: Union[str, float]):
195     if isinstance(points, float):
196         return points
197     if points == '':
198         return None
199     if points is None:
200         return None
201     return float(points)
202
203
204 @answers.route("/iframehtml/<plugintype>/<task_id_ext>/<int:user_id>")
205 @answers.route("/iframehtml/<plugintype>/<task_id_ext>/<int:user_id>/<int:answer_id>")
206 def get_iframehtml(plugintype: str, task_id_ext: str, user_id: int, answer_id: ↩
    Optional[int] = None):
207     """
208     Gets the HTML to be used in iframe.
209
210     :param plugintype: plugin type
211     :param task_id_ext: task id
212     :param user_id: the user whose information to get
213     :param answer_id: answer id from answer browser
214     :return: HTML to be used in iframe
215     """
216     try:
217         tid = TaskId.parse(task_id_ext)
218     except PluginException as e:
219         raise RouteException(f'Task id error: {e}')
220     d = get_doc_or_abort(tid.doc_id)
221     d.document.insert_preamble_pars()
222
223     ctx_user = User.get_by_id(user_id)
224     if not ctx_user:
```

```
225         raise RouteException('User not found')
226     vr = verify_task_access(
227         d,
228         tid,
229         AccessType.view,
230         TaskIdAccess.ReadWrite,
231         context_user=user_context_with_logged_in(ctx_user),
232         view_ctx=default_view_ctx,
233     )
234     plugin = vr.plugin
235     answer = None
236     if answer_id is not None:
237         answer, doc_id = verify_answer_access(
238             answer_id,
239             ctx_user.id,
240             default_view_ctx,
241             require_teacher_if_not_own=True,
242         )
243 
244     if plugin.type != plugintype:
245         raise RouteException(f'Plugin type mismatch: {plugin.type} != {plugintype}')
246 
247     users = [ctx_user]
248 
249     answerinfo = get_existing_answers_info(users, tid)
250 
251     info = plugin.get_info(users, answerinfo.count)
252 
253     state = try_load_json(answer.content) if answer else None
254 
255     answer_call_data = {'markup': plugin.values,
256                         'state': state,
257                         'taskID': tid.doc_task,
258                         'info': info,
259                         'iframehtml': True}
260 
261     vals = get_plug_vals(d, tid, user_context_with_logged_in(users[0]), ↩
    default_view_ctx)
262     if vals:
263         answer_call_data['markup']['fielddata'] = vals.to_json()
264 
265     jsonresp = call_plugin_answer_and_parse(answer_call_data, plugintype)
266 
267     if 'iframehtml' not in jsonresp:
268         return json_response({'error': 'The key "iframehtml" is missing in plugin ↩
    response.'}, 400)
269     result = jsonresp['iframehtml']
270     return result
271 
272 
273 def call_plugin_answer_and_parse(answer_call_data, plugintype):
274     plugin_response = call_plugin_answer(plugintype, answer_call_data)
275     try:
276         jsonresp = json.loads(plugin_response)
277     except ValueError as e:
278         raise PluginException(
279             'The plugin response was not a valid JSON string. The response was: ' + ↩
    plugin_response) from e
280     return jsonresp
```

```
281
282
283 def get_useranswers_for_task(user: User, task_ids: List[TaskId], answer_map):
284     """
285     Performs a query for latest valid answers by given user for given task
286     Similar to pluginControl.get_answers but without counting
287     :param user: user
288     :param task_ids: tasks to be queried
289     :param answer_map: a dict where to add each taskID: Answer
290     :return: {taskID: Answer}
291     """
292     col = func.max(Answer.id).label('col')
293     sub = (user
294            .answers
295            .filter(valid_taskid_filter(task_ids))
296            .add_columns(col)
297            .with_entities(col)
298            .group_by(Answer.task_id).subquery())
299     answs: List[Answer] = Answer.query.join(sub, Answer.id == sub.c.col).all()
300     for answer in answs:
301         if len(answer.users_all) > 1:
302             answer_map[answer.task_id] = answer
303         else:
304             asd = answer.to_json()
305             asd.pop('users')
306             answer_map[answer.task_id] = asd
307     return answs
308
309
310 def get_globals_for_tasks(task_ids: List[TaskId], answer_map):
311     col = func.max(Answer.id).label('col')
312     cnt = func.count(Answer.id).label('cnt')
313     sub = (valid_answers_query(task_ids)
314            .add_columns(col, cnt)
315            .with_entities(col, cnt)
316            .group_by(Answer.task_id).subquery()
317            )
318     answers: List[Tuple[Answer, int]] = (
319         Answer.query.join(sub, Answer.id == sub.c.col)
320             .with_entities(Answer, sub.c.cnt)
321             .all()
322     )
323     for answer in answers:
324         asd = answer.Answer.to_json()
325         answer_map[answer.Answer.task_id] = asd
326     return cnt, answers
327
328
329 @dataclass
330 class UserAnswersForTasksModel:
331     tasks: List[str]
332     user: int
333
334
335 UserAnswersForTasksSchema = class_schema(UserAnswersForTasksModel)
336
337
338 @answers.route("/userAnswersForTasks", methods=['POST'])
339 @use_args(UserAnswersForTasksSchema())
```

```
def get_answers_for_tasks(args: UserAnswersForTasksModel):
    """
    Route for getting latest valid answers for given user and list of tasks
    :return: {"answers": {taskID: Answer}, "userId": user_id}
    """
    tasks, user_id = args.tasks, args.user
    user = User.get_by_id(user_id)
    if user is None:
        raise RouteException('Non-existent user')
    verify_logged_in()
    try:
        doc_map = {}
        tids = []
        gtids = []
        for task_id in tasks:
            tid = TaskId.parse(task_id)
            if tid.doc_id not in doc_map:
                dib = get_doc_or_abort(tid.doc_id, f'Document {tid.doc_id} not found')
                verify_seeanswers_access(dib)
                doc_map[tid.doc_id] = dib.document
            if tid.is_global:
                gtids.append(tid)
            else:
                tids.append(tid)
        answer_map = {}
        if tids:
            get_useranswers_for_task(user, tids, answer_map)
        if gtids:
            get_globals_for_tasks(gtids, answer_map)
        return json_response({"answers": answer_map, "userId": user_id})
    except Exception as e:
        raise RouteException(str(e))


@dataclass
class JsRunnerMarkupModel(GenericMarkupModel):
    fields: Union[
        List[str], Missing] = missing  # This is actually required, but we cannot use ↩
    non-default arguments here...
    autoadd: Union[bool, Missing] = missing
    autoUpdateTables: Union[bool, Missing] = True
    creditField: Union[str, Missing] = missing
    defaultPoints: Union[float, Missing] = missing
    failGrade: Union[str, Missing] = missing
    fieldhelper: Union[bool, Missing] = missing
    gradeField: Union[str, Missing] = missing
    gradingScale: Union[Dict[Any, Any], Missing] = missing
    group: Union[str, Missing] = missing
    groups: Union[List[str], Missing] = missing
    includeUsers: Union[MembershipFilter, Missing] = ↩
    field(default=MembershipFilter.Current,
                                                        metadata={'by_value': True})
    selectIncludeUsers: bool = False
    paramFields: Union[List[str], Missing] = missing
    postprogram: Union[str, Missing] = missing
    preprogram: Union[str, Missing] = missing
    program: Union[str, Missing] = missing
    overrideGrade: bool = False
    showInView: bool = False
```

```
    confirmText: Union[str, Missing] = missing
    timeout: Union[int, Missing] = missing
    updateFields: Union[List[str], Missing] = missing
    nextRunner: Union[str, Missing] = missing
    timeZoneDiff: Union[int, Missing] = missing

    @validates_schema(skip_on_field_errors=True)
    def validate_schema(self, data, **_):
        if data.get('fields') is None:
            raise ValidationError('Missing data for required field.', field_name='fields')
        if data.get('group') is None and data.get('groups') is None:
            raise ValidationError("Either group or groups must be given.")


JsRunnerMarkupSchema = class_schema(JsRunnerMarkupModel)


@dataclass
class JsRunnerInputModel:
    nosave: Union[bool, Missing] = missing
    userNames: Union[List[str], Missing] = missing
    paramComps: Union[Dict[str, str], Missing] = missing
    includeUsers: Union[MembershipFilter, Missing] = field(default=missing, metadata={'by_value': True})


@dataclass
class RefFrom:
    docId: int
    par: str


AnswerData = Dict[str, Any]


@dataclass
class JsRunnerAnswerModel:
    input: JsRunnerInputModel
    ref_from: Optional[RefFrom] = None
    abData: Union[AnswerData, Missing] = missing


JsRunnerAnswerSchema = class_schema(JsRunnerAnswerModel)


@dataclass
class SendEmailModel:
    rcpts: str
    msg: str
    subject: str
    bccme: Union[bool, Missing, None] = missing


SendEmailSchema = class_schema(SendEmailModel)


@answers.route('/sendemail/', methods=['post'])
@use_args(SendEmailSchema())
```

```
454 def send_email(args: SendEmailModel):
455     """
456     Route for sending email
457     TODO: combine with multisendemail
458     :return:
459     """
460     rcpts, msg, subject, bccme = args.rcpts, args.msg, args.subject, args.bccme
461     curr_user = get_current_user_object()
462     if curr_user not in UserGroup.get_teachers_group().users and curr_user not in ↩
      get_home_organization_group().users:
463         raise AccessDenied("Sorry, you don't have permission to use this resource.")
464     curr_user = get_current_user_object()
465     if bccme:
466         bcc = curr_user.email
467     multi_send_email(
468         rcpt=rcpts,
469         subject=subject,
470         msg=msg,
471         mail_from=curr_user.email,
472         reply_to=curr_user.email,
473         bcc=bcc
474     )
475     return ok_response()
476
477
478 @answers.route("/multiSendEmail/<doc_id>", methods=['POST'])
479 def multisendemail(doc_id: int):
480     d = get_doc_or_abort(doc_id)
481     verify_teacher_access(d)
482     mail_from = get_current_user_object().email
483     bcc = ""
484     bccme = request.json.get('bccme', False)
485     if bccme:
486         bcc = mail_from
487     multi_send_email(
488         rcpt=request.json.get('rcpt'),
489         subject=request.json.get('subject'),
490         msg=request.json.get('msg'),
491         mail_from=mail_from,
492         reply_to=mail_from,
493         bcc=bcc
494     )
495     return ok_response()
496
497
498 @answers.route("/<plugintype>/<task_id_ext>/answer/", methods=['put'])
499 @answers.route("/<plugintype>/<task_id_ext>/answer", methods=['put'])
500 def post_answer(plugintype: str, task_id_ext: str):
501     """Saves the answer submitted by user for a plugin in the database.
502
503     :param plugintype: The type of the plugin, e.g. csPlugin.
504      TODO: No longer needed because it is checked from the document block's plugin ↩
      attribute.
505     :param task_id_ext: The extended task id of the form "22.palidrome.par_id".
506
507     """
508     answerdata, = verify_json_params('input')
509     answer_browser_data, answer_options = verify_json_params('abData', 'options', ↩
      require=False, default={})
```

```
    curr_user = get_current_user_object()
    verify_ip_ok(user=curr_user, msg='Answering is not allowed from this IP address.')
    return json_response(
        post_answer_impl(
            task_id_ext,
            answerdata,
            answer_browser_data,
            answer_options,
            curr_user,
            get_urlmacros_from_request(),
            get_other_session_users_objs(),
            get_origin_from_request()
        ).result)


@dataclass
class AnswerRouteResult:
    result: Dict[str, Any]
    plugin: Plugin


def post_answer_impl(
        task_id_ext: str,
        answerdata: AnswerData,
        answer_browser_data,
        answer_options,
        curr_user: User,
        urlmacros,
        other_session_users: List[User],
        origin: Optional[OriginInfo]
) -> AnswerRouteResult:
    receive_time = get_current_time()
    tid = TaskId.parse(task_id_ext)
    d = get_doc_or_abort(tid.doc_id)
    d.document.insert_preamble_pars()

    # It is rare but possible that the current user has been deleted (for example as ↩
    the result of merging 2 accounts).
    # We assume it's the case here, so we clear the session and ask to log in again.
    if curr_user.is_deleted:
        session.clear()
        raise AccessDenied('Please refresh the page and log in again.')

    force_answer = answer_options.get('forceSave', False)  # Only used in feedback ↩
    plugin.
    is_teacher = answer_browser_data.get('teacher', False)
    save_teacher = answer_browser_data.get('saveTeacher', False)
    should_save_answer = answer_browser_data.get('saveAnswer', True) and tid.task_name

    if save_teacher:
        verify_teacher_access(d, user=curr_user)
    users = None

    ctx_user = None
    if is_teacher:
        answer_id = answer_browser_data.get('answer_id', None)
        user_id = answer_browser_data.get('userId', None)

        if answer_id is not None:
```

```
567             answer = Answer.query.get(answer_id)
568             if not answer:
569                 raise PluginException(f'Answer not found: {answer_id}')
570             expected_task_id = answer.task_id
571             if expected_task_id != tid.doc_task:
572                 raise PluginException('Task ids did not match')
573 
574             # Later on, we may call users.append, but we don't want to modify the users of the existing
575             # answer. Therefore, we make a copy of the user list so that SQLAlchemy no longer associates
576             # the user list with the answer.
577             users = list(answer.users_all)
578             if not users:
579                 raise PluginException('No users found for the specified answer')
580             # For now global fields use current user in browser
581             # We set answerer user to be current user later so we ignore user mismatch in global case
582             if user_id not in (u.id for u in users) and not tid.is_global:
583                 raise PluginException('userId is not associated with answer_id')
584         elif user_id and user_id != curr_user.id and False:  # TODO: Vesa's hack to no need for belong teachers group
585             teacher_group = UserGroup.get_teachers_group()
586             if curr_user not in teacher_group.users:
587                 raise PluginException('Permission denied: you are not in teachers group.')
588         if user_id:
589             ctx_user = User.query.get(user_id)
590             if not ctx_user:
591                 raise PluginException(f'User {user_id} not found')
592             users = [ctx_user]  # TODO: Vesa's hack to save answer to student
593     try:
594         vr = verify_task_access(
595             d,
596             tid,
597             AccessType.view,
598             TaskIdAccess.ReadWrite,
599             context_user=UserContext(ctx_user or curr_user, curr_user),
600             view_ctx=ViewContext(ViewRoute.View, False, urlmacros=urlmacros, origin=origin),
601             allow_grace_period=True,
602         )
603         plugin = vr.plugin
604     except (PluginException, TimDbException) as e:
605         raise PluginException(str(e))
606 
607     if tid.is_points_ref:
608         return AnswerRouteResult(
609             result=handle_points_ref(answerdata, curr_user, d, plugin.ptype, tid),
610             plugin=plugin,
611         )
612 
613     get_task = isinstance(answerdata, dict) and answerdata.get("getTask", False) and plugin.ptype.can_give_task()
614     if not (should_save_answer or get_task) or is_teacher:
615         verify_seeanswers_access(d, user=curr_user)
616 
617     uploads = []
618 
```

```
    if not curr_user.logged_in and not plugin.known.anonymous:
        raise RouteException('You must be logged in to answer this task.')
    if plugin.known.useCurrentUser or plugin.task_id.is_global:  # For plugins that is saved only for current user
        users = [curr_user]

    if isinstance(answerdata, dict):
        file = answerdata.get('uploadedFile', '')
        trimmed_file = file.replace('/uploads/', '')
        type = answerdata.get('type', '')
        if trimmed_file and type == 'upload':
            # The initial upload entry was created in /pluginUpload route, so we need to check that the owner matches
            # what the browser is saying. Additionally, we'll associate the answer with the uploaded file later
            # in this route.
            block = Block.query.filter((Block.description == trimmed_file) &
                                       (Block.type_id == BlockType.Upload.value)).first()
            if block is None:
                raise PluginException(f'Non-existent upload: {trimmed_file}')
            verify_view_access(block, message="You don't have permission to touch this file.", user=curr_user)
            uploads = [AnswerUpload.query.filter(AnswerUpload.upload_block_id == block.id).first()]
            # if upload.answer_id is not None:
            #    raise PluginException(f'File was already uploaded: {file}')

        files: List[int] = answerdata.get('uploadedFiles', None)
        if files is not None:
            for file in files:
                trimmed_file = file["path"].replace('/uploads/', '')
                block = Block.query.filter((Block.description == trimmed_file) &
                                           (Block.type_id == BlockType.Upload.value)).first()
                if block is None:
                    raise PluginException(f'Non-existent upload: {trimmed_file}')
                verify_view_access(block, message="You don't have permission to touch this file.", user=curr_user)
                uploads.append(AnswerUpload.query.filter(AnswerUpload.upload_block_id == block.id).first())

    # Load old answers

    if users is None:
        users = [curr_user] + other_session_users

    answerinfo = get_existing_answers_info(users, tid)
    valid, _ = plugin.is_answer_valid(answerinfo.count, {})
    info = plugin.get_info(users, answerinfo.count, look_answer=is_teacher and not save_teacher, valid=valid)

    # Get the newest answer (state). Only for logged in users.
    state = try_load_json(answerinfo.latest_answer.content) if curr_user.logged_in and answerinfo.latest_answer else None

    preprocessor = answer_call_preprocessors.get(plugin.type)
    if preprocessor:
        preprocessor(answerdata, curr_user, d, plugin)
```

```
667
668     answer_call_data = {'markup': plugin.values,
669                         'state': state,
670                         'input': answerdata,
671                         'taskID': tid.doc_task,
672                         'info': info}
673     preoutput = ""
674     preprogram = plugin.values.get("preprogram", None)
675     if preprogram and plugin.type != "jsrunner":
676         params = JsRunnerParams(code=preprogram, data=answer_call_data)
677         answer_call_data, preoutput = jsrunner_run(params)
678
679     jsonresp = call_plugin_answer_and_parse(answer_call_data, plugin.type)
680
681     web = jsonresp.get('web')
682     if web is None:
683         raise PluginException(f'Got malformed response from plugin: {jsonresp}')
684     result = {'web': web}
685
686     if 'savedata' in jsonresp:
687         siw = answer_call_data.get("markup", {}).get("showInView", False)
688         add_group = None
689         if plugin.type == 'importData':
690             add_group = plugin.values.get('addUsersToGroup')
691         saveresult = save_fields(jsonresp, curr_user, d, allow_non_teacher=siw, ↩
    add_users_to_group=add_group)
692
693         # TODO: Could report the result to other plugins too.
694         if plugin.type == 'importData':
695             web['fieldresult'] = saveresult
696
697     def add_reply(obj, key, run_markdown=False):
698         if key not in plugin.values:
699             return
700         text_to_add = plugin.values[key]
701         if run_markdown:
702             dumbo_result = call_dumbo([text_to_add])
703             text_to_add = dumbo_result[0]
704         obj[key] = text_to_add
705
706     noupdate = False  #  if true do not send new id
707
708     resultmd = result['web'].get('md', None)
709     if resultmd:
710         result['web']['md'] = call_dumbo([resultmd])[0]
711
712     if not get_task:
713         add_reply(result['web'], '-replyImage')
714         add_reply(result['web'], '-replyMD', True)
715         add_reply(result['web'], '-replyHTML')
716     if 'save' in jsonresp and not get_task:
717         # TODO: RND_SEED: save used rnd_seed for this answer if answer is saved, ↩
    found from par.get_rnd_seed()
718         save_object = jsonresp['save']
719         tags = []
720         tim_info = jsonresp.get('tim_info', {})
721         if tim_info.get("noupdate", False):
722             noupdate = True
723         points = tim_info.get('points', None)
```

```
724         multiplier = plugin.points_multiplier()
725         if multiplier and points is not None:
726             points *= plugin.points_multiplier()
727         elif not multiplier:
728             points = None
729         # Save the new state
730         try:
731             tags = save_object['tags']
732         except (TypeError, KeyError):
733             pass
734 
735         def get_name_and_val(name1, name2=""):
736             """
737             Try with name1, -name1 amnd name2
738             return working name and value or "", None
739             """
740             name = name1
741             val = plugin.values.get(name, None)
742             if val:
743                 return name, val
744 
745             name = "-" + name1
746             val = plugin.values.get(name, None)
747             if val:
748                 return name, val
749 
750             if name2:
751                 name = name2
752                 val = plugin.values.get(name, None)  # old name
753             if val:
754                 return name, val
755 
756             name = ""
757             return name, val
758 
759         postprogram_name, postprogram = \
760             get_name_and_val("postprogram", "postProgram")
761 
762         postlibraries_name, postlibraries = get_name_and_val("postlibraries")
763 
764         postoutput = plugin.values.get("postoutput", 'feedback')
765 
766         if postprogram and postlibraries:
767             libs = ""
768             postlibraries_edit = plugin.values.get("postlibrariesEdit", {})
769             libnr = 0
770             for lib in postlibraries:
771                 try:
772                     content = get_from_url(lib)
773                     if content.startswith('{"error"'):
774                         web["error"] += lib + "\n" + content
775                         postprogram = ""
776                         break
777                     libedit = postlibraries_edit.get(libnr, None)
778                     if libedit:
779                         libdel = libedit.get("deleteAfter", None)
780                         if libdel:
781                             delpos = content.find(libdel)
782                             if delpos >= 0:
```

```
                                content = content[0:delpos]
                    libs += content
                except Exception as ex:
                    web["error"] += lib + "\n" + str(ex)
                    postprogram = ""
                libnr += 1
            if postprogram:
                postprogram = libs + \
                              "\n//=== END LIBRARIES ===\n" + \
                              postprogram

        def set_postoutput(result, output, postoutput):
            if not postoutput or (not output and not preoutput):
                return
            parts = postoutput.split(".")
            r = result
            lastkey = parts[-1]
            for p in parts[:-1]:
                if not p in r:
                    r[p] = {}
                r = r[p]
            r[lastkey] = r.get(lastkey, '') + str(preoutput) + str(output)

        def add_value(result, key, data):
            value = data.get(key, None)
            if value is None:
                return
            if value.startswith('md:'):
                value = call_dumbo([value[3:]])[0]
            result[key] = result.get(key, "") + value

        def postprogram_result(data, output):
            result["web"] = data.get("web", web)
            add_value(result, "error", data)
            add_value(result, "feedback", data)
            add_value(result, "topfeedback", data)
            if output.startswith('md:'):
                output = call_dumbo([output[3:]])[0]
            set_postoutput(result, output, postoutput)

        if (not is_teacher and should_save_answer) or ('savedata' in jsonresp):
            is_valid, explanation = plugin.is_answer_valid(answerinfo.count, tim_info)
            if vr.is_expired:
                fixed_time = receive_time - ↩
    d.document.get_settings().answer_submit_time_tolerance()
                if fixed_time < (vr.access.accessible_to or maxdate):
                    is_valid = True
                else:
                    is_valid = False
                    explanation = 'Your view access to this document has expired, so ↩
    this answer was saved but marked as invalid.'
            points_given_by = None
            if answer_browser_data.get('giveCustomPoints'):
                try:
                    points = plugin.validate_points(answer_browser_data.get('points'))
                except PluginException as e:
                    result['error'] = str(e)
                else:
                    points_given_by = get_current_user_group()
```

```
840
841             if postprogram:
842                 data = {
843                         'answer_call_data': answer_call_data,
844                         'points': points,
845                         'save_object': save_object,
846                         'tags': tags,
847                         'is_valid': is_valid,
848                         'force_answer': force_answer,
849                         'error': '',
850                         'web': web,
851                         }
852                 try:
853                     params = JsRunnerParams(code=postprogram, data=data)
854                     data, output = jsrunner_run(params)
855                     points = data.get("points", points)
856                     save_object = data.get("save_object", save_object)
857                     is_valid = data.get("is_valid", is_valid)
858                     force_answer = data.get("force_answer", force_answer)
859                     postprogram_result(data, output)
860                 except JsRunnerError as e:
861                     return AnswerRouteResult(
862                         result={'web': {'error': 'Error in JavaScript: ' + e.args[0]}},
863                         plugin=plugin,
864                     )
865
866             if points or save_object is not None or tags:
867                 a = save_answer(
868                     users,
869                     tid,
870                     save_object,
871                     points,
872                     tags,
873                     is_valid,
874                     points_given_by,
875                     force_answer,
876                     plugintype=plugin.ptype,
877                     max_content_len=current_app.config['MAX_ANSWER_CONTENT_SIZE'],
878                 )
879                 result['savedNew'] = a.id if a else None
880                 if a:
881                     send_answer_backup_if_enabled(a)
882             else:
883                 result['savedNew'] = None
884             if noupdate:
885                 result['savedNew'] = None
886
887             if not is_valid:
888                 result['error'] = explanation
889         elif save_teacher:
890             points = answer_browser_data.get('points', points)
891             points = points_to_float(points)
892             a = save_answer(
893                 users,
894                 tid,
895                 save_object,
896                 points,
897                 tags,
898                 valid=True,
```

```
                points_given_by=get_current_user_group(),
                saver=curr_user,
                plugintype=plugin.ptype,
                max_content_len=current_app.config['MAX_ANSWER_CONTENT_SIZE'],
            )
            # TODO: Could call backup here too, but first we'd need to add support for saver in export/import.
            result['savedNew'] = a.id if a else None
        else:
            result['savedNew'] = None
            if postprogram:
                data = {'answer_call_data': answer_call_data,
                        'points': points,
                        'save_object': save_object,
                        'tags': tags,
                        'is_valid': True,
                        'force_answer': force_answer,
                        'error': '',
                        'web': web,
                        }
                try:
                    params = JsRunnerParams(code=postprogram, data=data)
                    data, output = jsrunner_run(params)
                    points = data.get("points", points)
                    output += "\nPoints: " + str(points)
                    postprogram_result(data, output)
                except JsRunnerError as e:
                    return AnswerRouteResult(
                        result={'web': {'error': 'Error in JavaScript: ' + e.args[0]}},
                        plugin=plugin,
                    )
        if result['savedNew'] is not None and uploads:
            # Associate this answer with the upload entries
            for upload in uploads:
                upload.answer_id = result['savedNew']

    db.session.commit()

    for u in users:
        clear_doc_cache(d, u)

    try:
        if postprogram_name:
            result['web']['markup'].pop(postprogram_name)  # TODO: stdy why someone puts markup here
    except:
        pass

    return AnswerRouteResult(result=result, plugin=plugin)


def preprocess_jsrunner_answer(answerdata: AnswerData, curr_user: User, d: DocInfo, plugin: Plugin):
    """Executed before the actual jsrunner answer route is called.
    This is required to fetch the requested data from the database."""

    s = JsRunnerMarkupSchema()
    runnermarkup: JsRunnerMarkupModel = s.load(plugin.values)
    runner_req: JsRunnerAnswerModel = JsRunnerAnswerSchema().load({'input': answerdata})
```

```
955     groupnames = runnermarkup.groups
956     if groupnames is missing:
957         groupnames = [runnermarkup.group]
958     requested_groups = RequestedGroups.from_name_list(groupnames)
959     not_found_groups = sorted(list(set(groupnames) - set(g.name for g in ↩
        requested_groups.groups)
960                                    - {ALL_ANSWERED_WILDCARD}))  # Ensure the wildcard ↩
        is removed
961     if not_found_groups:
962         raise PluginException(f'The following groups were not found: {", ↩
        ".join(not_found_groups)}')
963     if runner_req.input.paramComps:  # TODO: add paramComps to the interface, so no ↩
        need to manipulate source code
964         preprg = runnermarkup.preprogram or ''
965         plugin.values["preprogram"] = f"gtools.params = ↩
        {json.dumps(runner_req.input.paramComps)};\n{preprg}"
966     siw = runnermarkup.showInView
967     markup_include_opt = value_or_default(runnermarkup.includeUsers, ↩
        MembershipFilter.Current)
968     if (not runnermarkup.selectIncludeUsers and
969             isinstance(runner_req.input.includeUsers, MembershipFilter) and
970             markup_include_opt != runner_req.input.includeUsers):
971         raise AccessDenied('Not allowed to select includeUsers option.')
972 
973     ensure_grade_and_credit(runnermarkup.program, runnermarkup.fields)
974 
975     answerdata['data'], answerdata['aliases'], _, _ = get_fields_and_users(
976         runnermarkup.fields,
977         requested_groups,
978         d,
979         curr_user,
980         default_view_ctx,
981         access_option=GetFieldsAccess.from_bool(siw),
982         member_filter_type=value_or_default(runner_req.input.includeUsers, ↩
        markup_include_opt),
983         user_filter=User.name.in_(runner_req.input.userNames) if ↩
        runner_req.input.userNames else None
984     )
985     answerdata.pop('paramComps', None)  # This isn't needed by jsrunner server, so ↩
        don't send it.
986     # plugin.values['timeZoneDiff'] = 3
987     tzd = plugin.values.get('timeZoneDiff', None)
988     if tzd is None:
989         localtz = local_timezone
990         localoffset = localtz.utcoffset(datetime.now())
991         tzd = localoffset.total_seconds() / 3600
992         plugin.values['timeZoneDiff'] = tzd
993     if runnermarkup.program is missing:
994         raise PluginException("Attribute 'program' is required.")
995 
996 
997 def ensure_grade_and_credit(prg, flds):
998     if not prg:
999         return
1000     if prg.find('grade') >= 0 or prg.find('Grade'):  # add grade to fields if missing
1001         grade_found = False
1002         credit_found = False
1003         for fld in flds:
1004             if fld.startswith('grade'):
```

```
                grade_found = True
            if fld.startswith('credit'):
                credit_found = True
            if grade_found and credit_found:
                break
        if not grade_found:
            flds.append('grade')
        if not credit_found:
            flds.append('credit')


answer_call_preprocessors: Dict[str, Callable[[AnswerData, User, DocInfo, Plugin], None]] = {
    'jsrunner': preprocess_jsrunner_answer,
}


def handle_points_ref(answerdata: AnswerData, curr_user: User, d: DocInfo, ptype: PluginType, tid: TaskId):
    verify_teacher_access(d, user=curr_user)
    given_points = answerdata.get(ptype.get_content_field_name())
    if given_points is not None:
        try:
            given_points = float(given_points)
        except ValueError:
            raise RouteException('Points must be a number.')
    a = curr_user.answers.filter_by(task_id=tid.doc_task).order_by(Answer.id.desc()).first()
    if a:
        a.points = given_points
        s = None
    else:
        a = Answer(
            content=json.dumps({ptype.get_content_field_name(): ''}),
            points=given_points,
            task_id=tid.doc_task,
            users_all=[curr_user],
            valid=True,
        )
        db.session.add(a)
        db.session.flush()
        s = a.id
    db.session.commit()
    return {'savedNew': s, 'web': {'result': 'points saved'}}


class JsrunnerGroups(TypedDict, total=False):
    set: Dict[str, List[int]]
    add: Dict[str, List[int]]
    remove: Dict[str, List[int]]


MAX_GROUPS_PER_CALL = 10


def handle_jsrunner_groups(groupdata: Optional[JsrunnerGroups], curr_user: User):
    if not groupdata:
        return
    groups_created = 0
```

```
    for op, group_set in groupdata.items():
        for name, uids in group_set.items():
            ug = UserGroup.get_by_name(name)
            if not ug:
                if op == 'set':
                    if groups_created >= MAX_GROUPS_PER_CALL:
                        raise RouteException(
                            f'Maximum of {MAX_GROUPS_PER_CALL} groups can be created per one jsrunner run.',
                        )
                    ug, _ = do_create_group(name)
                    groups_created += 1
                else:
                    raise RouteException(f'Group does not exist: {name}')
            else:
                verify_group_edit_access(ug, curr_user)
            users: List[User] = User.query.filter(User.id.in_(uids)).all()
            found_user_ids = set(u.id for u in users)
            missing_ids = set(uids) - found_user_ids
            if missing_ids:
                raise RouteException(f'Users not found: {missing_ids}')
            if op == 'set':
                ug.memberships_sel = [UserGroupMember(user=u, adder=curr_user) for u in users]
            elif op == 'add':
                for u in users:
                    u.add_to_group(ug, added_by=curr_user)
            elif op == 'remove':
                ug.memberships_sel = [ugm for ugm in ug.memberships_sel if ugm.user_id not in found_user_ids]
            else:
                raise RouteException(f'Unexpected group operation: {op}')


class UserFieldEntry(TypedDict):
    user: int
    fields: Dict[str, str]


def create_missing_users(users: List[MissingUser]) -> Tuple[List[UserFieldEntry], List[User]]:
    created_users = []
    for mu in users:
        ui = mu.user
        if ui.email is not None:
            # A+ may give users with invalid mails like '6128@localhost'. Just skip over those.
            if ui.email.endswith('@localhost'):
                continue
            if not is_valid_email(ui.email):
                raise RouteException(f'Invalid email: "{ui.email}"')
        if ui.username is None:
            ui.username = ui.email
        if ui.full_name is None and ui.email is not None:
            # Approximate real name with the help of email.
            # This won't be fully accurate, but we can't do better.
            ui.full_name = approximate_real_name(ui.email)
        u = create_or_update_user(ui, update_username=False)
        created_users.append(u)
```

```
    db.session.flush()
    fields = []
    for u, missing_u in zip(created_users, users):
        fields.append({'user': u.id, 'fields': missing_u.fields})
    return fields, created_users


MissingUserSchema = class_schema(MissingUser)


@dataclass
class FieldSaveResult:
    users_created: List[User] = field(default_factory=list)
    users_missing: List[UserInfo] = field(default_factory=list)
    fields_changed: int = 0
    fields_unchanged: int = 0
    fields_ignored: int = 0


class FieldSaveUserEntry(TypedDict):
    user: int
    fields: Dict[str, str]


class FieldSaveRequest(TypedDict, total=False):
    savedata: Optional[List[FieldSaveUserEntry]]
    ignoreMissing: Optional[bool]
    allowMissing: Optional[bool]
    createMissingUsers: Optional[bool]
    missingUsers: Optional[Any]
    groups: Optional[JsrunnerGroups]


def verify_user_create_right(curr_user: User):
    if curr_user.is_admin:
        return
    user_creators = UserGroup.get_user_creator_group()
    if user_creators not in curr_user.groups:
        raise AccessDenied('You do not have permission to create users.')


def save_fields(
        jsonresp: FieldSaveRequest,
        curr_user: User,
        current_doc: Optional[DocInfo] = None,
        allow_non_teacher: bool = False,
        add_users_to_group: Optional[str] = None,
) -> FieldSaveResult:
    save_obj = jsonresp.get('savedata')
    ignore_missing = jsonresp.get('ignoreMissing', False)
    allow_missing = jsonresp.get('allowMissing', False)
    ignore_fields = {}
    handle_jsrunner_groups(jsonresp.get('groups'), curr_user)
    missing_users = jsonresp.get('missingUsers')
    saveresult = FieldSaveResult()
    if save_obj is None:
        save_obj = []
    if missing_users:
        m_users: List[MissingUser] = MissingUserSchema().load(missing_users, many=True)
```

```
        if jsonresp.get('createMissingUsers'):
            verify_user_create_right(curr_user)
            new_fields, users = create_missing_users(m_users)
            save_obj += new_fields
            saveresult.users_created = users
        else:
            saveresult.users_missing = [mu.user for mu in m_users]
    if not save_obj:
        return saveresult
    tasks = set()
    doc_map: Dict[int, DocInfo] = {}
    user_map: Dict[int, User] = {u.id: u for u in User.query.filter(User.id.in_(x['user'] for x in save_obj)).all()}

    # We need this separate "add_users_to_group" parameter because the plugin may have reported missing users.
    # They are created above, so the plugin cannot report them with "groups" in jsonresp because the user IDs are not
    # known until now.
    if add_users_to_group:
        handle_jsrunner_groups({'add': {add_users_to_group: [k for k in user_map.keys()]}}, curr_user)

    for item in save_obj:
        task_u = item['fields']
        for tid in task_u.keys():
            tasks.add(tid)
            try:
                id_num = TaskId.parse(tid, require_doc_id=False, allow_block_hint=False, allow_custom_field=True)
            except PluginException:
                raise RouteException(f'Invalid task name: {tid.split(".")[1]}')
            if not id_num.doc_id:
                raise RouteException(f'Doc id missing: {tid}')
            if id_num.doc_id not in doc_map:
                doc_map[id_num.doc_id] = get_doc_or_abort(id_num.doc_id)
    task_content_name_map = {}
    for task in tasks:
        t_id = TaskId.parse(task, require_doc_id=True, allow_block_hint=False, allow_custom_field=True)
        if ignore_fields.get(t_id.doc_task, False):
            continue
        dib = doc_map[t_id.doc_id]
        # TODO: Return case-specific abort messages
        if not (curr_user.has_teacher_access(dib) or (allow_non_teacher and t_id.doc_id == current_doc.id) or (
                curr_user.has_view_access(dib) and dib.document.get_own_settings().get("allow_external_jsrunner",
                                                                                       False))):
            raise AccessDenied(f'Missing teacher access for document {dib.id}')
        try:
            vr = verify_task_access(
                dib,
                t_id,
                AccessType.view,
                TaskIdAccess.ReadWrite,
                UserContext.from_one_user(curr_user),
                default_view_ctx,
```

```
            )
            plugin = vr.plugin
        except TaskNotFoundException as e:
            if not allow_missing:
                if ignore_missing:
                    ignore_fields[t_id.doc_task] = True
                    continue
                raise RouteException(str(e))
            plugin = PluginType('textfield')  # assuming textfield type for fields that are not in the document
        except (PluginException, TimDbException) as e:
            raise RouteException(str(e))

        # TODO this 'if' seems unnecessary
        if t_id.task_name in ('grade', 'credit', 'completionDate'):
            task_content_name_map[task] = 'c'
            continue

        if t_id.field and t_id.field != "points" and t_id.field != "styles":
            if plugin.type == "numericfield" or plugin.type == "textfield":
                if t_id.field != plugin.get_content_field_name():
                    raise RouteException(f'Error saving to {task}: {t_id.field} is not an accepted field.')
            task_content_name_map[task] = t_id.field
        else:
            task_content_name_map[task] = plugin.get_content_field_name()

    parsed_task_ids = {
        key: TaskId.parse(key, require_doc_id=True, allow_block_hint=False, allow_custom_field=True)
        for user in save_obj for key in user['fields'].keys()
    }
    sq = (Answer.query
          .filter(Answer.task_id.in_([tid.doc_task for tid in parsed_task_ids.values() if not tid.is_global]))
          .join(User, Answer.users)
          .filter(User.id.in_(user_map.keys()))
          .group_by(User.id, Answer.task_id)
          .with_entities(func.max(Answer.id).label('aid'), User.id.label('uid'))
          .subquery())
    datas = Answer.query.join(sq, Answer.id == sq.c.aid).with_entities(sq.c.uid, Answer).all()
    global_answers = get_global_answers(parsed_task_ids)
    answer_map = defaultdict(dict)
    for uid, a in datas:
        answer_map[uid][a.task_id] = a
    for uid in user_map.keys():
        for a in global_answers:
            answer_map[uid][a.task_id] = a
    cpf = CachedPluginFinder(
        doc_map=doc_map,
        curr_user=UserContext.from_one_user(curr_user),
        view_ctx=default_view_ctx,
    )
    for user in save_obj:
        u_id = user['user']
        u = user_map.get(u_id)
        if not u:
            raise RouteException(f'User id {u_id} not found')
```

```
        user_fields = user['fields']
        task_map: DefaultDict[str, Dict[str, Any]] = defaultdict(dict)
        for key, value in user_fields.items():
            task_id = parsed_task_ids[key]
            if ignore_fields.get(task_id.doc_task, False):
                saveresult.fields_ignored += 1
                continue
            field = task_id.field
            if field is None:
                field = task_content_name_map[task_id.doc_task]
            task_map[task_id.doc_task][field] = value
        for taskid, contents in task_map.items():
            task_id = TaskId.parse(taskid, require_doc_id=False, allow_block_hint=False)
            if ignore_fields.get(task_id.doc_task, False):
                continue
            an: Answer = answer_map[u.id].get(task_id.doc_task)
            points = None
            content = {}
            new_answer = False
            if an:
                points = an.points
                content = json.loads(an.content)
            lastfield = "c"
            for field, value in contents.items():
                lastfield = field
                if field == 'points':
                    if value == "":
                        value = None
                    else:
                        try:
                            value = float(value)
                        except ValueError:
                            raise RouteException(f'Value {value} is not valid point value for task {task_id.task_name}')
                    if points != value:
                        new_answer = True
                    points = value
                elif field == "styles":
                    if isinstance(value, str):
                        try:
                            value = json.loads(value or "null")
                        except json.decoder.JSONDecodeError:
                            raise RouteException(
                                f'Value {value} is not valid style syntax for task {task_id.task_name}')
                    plug = cpf.find(task_id)
                    if not plug:
                        continue
                    if plug.allow_styles_field():
                        if not an or content.get(field) != value:
                            new_answer = True
                        if value is None:
                            content.pop(field, None)
                        else:
                            content[field] = value

                        # Ensure there's always a content field even when setting styles to an empty answer.
                        c_field = task_content_name_map[f'{task_id.doc_task}.{field}']
```

```
                        if c_field not in content:
                            content[c_field] = None
                elif field == "JSSTRING":  # TODO check if this should be ALL!  No this is for settings using string
                    if not an or json.dumps(content) != value:
                        new_answer = True
                    content = json.loads(value)  # TODO: shoud this be inside if
                else:
                    if not an or content.get(field, "") != value:
                        new_answer = True
                    content[field] = value
            if not new_answer:
                saveresult.fields_unchanged += 1
                continue
            if not content:
                content[task_content_name_map[f'{task_id.doc_task}.{lastfield}']] = None
            content = json.dumps(content)
            ans = Answer(
                content=content,
                points=points,
                task_id=task_id.doc_task,
                users=[u],
                valid=True,
                saver=curr_user,
            )
            saveresult.fields_changed += 1
            # If this was a global task, add it to all users in the answer map so we won't save it multiple times.
            if task_id.is_global:
                for uid in user_map.keys():
                    answer_map[uid][ans.task_id] = ans
            db.session.add(ans)
    return saveresult


def get_global_answers(parsed_task_ids: Dict[str, TaskId]) -> List[Answer]:
    sq2 = (Answer.query
           .filter(Answer.task_id.in_([tid.doc_task for tid in parsed_task_ids.values() if tid.is_global]))
           .group_by(Answer.task_id)
           .with_entities(func.max(Answer.id).label('aid'))
           .subquery())
    global_datas = Answer.query.join(sq2, Answer.id == sq2.c.aid).with_entities(Answer).all()
    return global_datas


def get_hidden_name(user_id):
    return 'Student %d' % user_id


def should_hide_name(d: DocInfo, user: User):
    # return True
    # return not user.has_teacher_access(d) and user.id != get_current_user_id()
    return user.id != get_current_user_id()


def maybe_hide_name(d: DocInfo, u: User):
    if should_hide_name(d, u):
```

```
1389         # NOTE! To anonymize user, do NOT assign to u's real_name, name, etc. ↩
     attributes here (or anywhere else either)
1390         # because it is
1391         #  1) dangerous (the anonymization would be persisted if db.session.commit() ↩
     was called after the assignment)
1392         #  2) not necessary because the hiding is done in User.to_json method.
1393         u.hide_name = True
1394
1395
1396 @answers.route("/taskinfo/<task_id>")
1397 def get_task_info(task_id):
1398     try:
1399         user_ctx = user_context_with_logged_in(None)
1400         plugin, d = Plugin.from_task_id(task_id, user_ctx=user_ctx, ↩
     view_ctx=default_view_ctx)
1401         verify_task_access(
1402             d,
1403             plugin.task_id,
1404             AccessType.view,
1405             TaskIdAccess.ReadOnly,
1406             allow_grace_period=True,
1407             context_user=user_ctx,
1408             view_ctx=default_view_ctx,
1409         )
1410         tim_vars = find_tim_vars(plugin)
1411     except PluginException as e:
1412         raise RouteException(str(e))
1413     return json_response(tim_vars)
1414
1415
1416 def find_tim_vars(plugin: Plugin):
1417     tim_vars = {
1418         'maxPoints': plugin.max_points(),
1419         'userMin': plugin.user_min_points(),
1420         'userMax': plugin.user_max_points(),
1421         'showPoints': plugin.show_points(),
1422         'deadline': plugin.deadline(),
1423         'starttime': plugin.starttime(),
1424         'answerLimit': plugin.answer_limit(),
1425         'triesText': plugin.known.tries_text(),
1426         'pointsText': plugin.known.points_text(),
1427     }
1428     return tim_vars
1429
1430
1431 def hide_points(a: Answer):
1432     j = a.to_json()
1433     j['points'] = None
1434
1435     # TODO: Hack for csPlugin
1436     c = a.content_as_json
1437     if isinstance(c, dict):
1438         c.pop('points', None)
1439         j['content'] = json.dumps(c)
1440
1441     if a.points is not None:
1442         j['points_hidden'] = True
1443     return j
1444
```

```
@answers.route('/exportAnswers/<path:doc_path>')
def export_answers(doc_path: str):
    d = DocEntry.find_by_path(doc_path, try_translation=False)
    if not d:
        raise RouteException('Document not found')
    verify_teacher_access(d)
    answer_list: List[Tuple[Answer, str]] = (
        Answer.query
            .filter(Answer.task_id.startswith(f'{d.id}.'))
            .join(User, Answer.users)
            .with_entities(Answer, User.email)
            .all()
    )
    return json_response([{
        'email': email,
        'content': a.content,
        'valid': a.valid,
        'points': a.points,
        'time': a.answered_on,
        'task': a.task_name,
        'doc': doc_path,
    } for a, email in answer_list])


@dataclass
class ImportAnswersModel:
    answers: List[ExportedAnswer]
    allow_missing_users: bool = False
    doc_map: Dict[str, str] = field(default_factory=dict)


@answers.route('/importAnswers', methods=['post'])
@use_model(ImportAnswersModel)
def import_answers(m: ImportAnswersModel):
    verify_admin()
    doc_paths = set(m.doc_map.get(a.doc, a.doc) for a in m.answers)
    docs = DocEntry.query.filter(DocEntry.name.in_(doc_paths)).all()
    doc_path_map = {d.path: d for d in docs}
    missing_docs = doc_paths - set(doc_path_map)
    if missing_docs:
        raise RouteException(f'Some documents not found: {missing_docs}')
    for d in docs:
        verify_teacher_access(d)
    filter_cond = Answer.task_id.startswith(f'{docs[0].id}.')
    for d in docs[1:]:
        filter_cond |= Answer.task_id.startswith(f'{d.id}.')
    existing_answers: List[Tuple[Answer, str]] = (
        Answer.query
            .filter(filter_cond)
            .join(User, Answer.users)
            .with_entities(Answer, User.email)
            .all()
    )
    existing_set = set((a.parsed_task_id.doc_id, a.task_name, a.answered_on, a.valid, ↩
    a.points, email) for a, email in
                       existing_answers)
    dupes = 0
    users = {u.email: u for u in User.query.filter(User.email.in_([a.email for a in ↩
```

```
    m.answers])).all()}
    requested_users = set(a.email for a in m.answers)
    missing_users = requested_users - set(users.keys())
    if missing_users and not m.allow_missing_users:
        raise RouteException(f'Email(s) not found: {seq_to_str(list(missing_users))}')
    m.answers.sort(key=lambda a: a.time)
    all_imported = []
    for a in m.answers:
        doc_id = doc_path_map[m.doc_map.get(a.doc, a.doc)].id
        if (doc_id, a.task, a.time, a.valid, a.points, a.email) not in existing_set:
            u = users.get(a.email)
            if not u:
                if not m.allow_missing_users:
                    raise Exception('Missing user should have been reported earlier')
                continue
            imported_answer = Answer(
                task_id=f'{doc_id}.{a.task}',
                valid=a.valid,
                points=a.points,
                content=a.content,
                answered_on=a.time,
            )
            imported_answer.users_all.append(u)
            db.session.add(imported_answer)
            all_imported.append(imported_answer)
        else:
            dupes += 1
    db.session.flush()

    # Sanity check: Make sure that the ids are in the same order as the timestamps of the answers - we currently rely on
    # the fact that the latest answer has the largest id.
    all_imported.sort(key=lambda a: a.id)
    for a, b in zip(all_imported, all_imported[1:]):
        if a.answered_on > b.answered_on:
            raise Exception('Import bug: Answer ids were in different order than answer timestamps. Imported nothing.')

    db.session.commit()
    return json_response({
        'imported': len(all_imported),
        'skipped_duplicates': dupes,
        'missing_users': list(missing_users),
    })


@answers.route("/getAnswers/<task_id>/<int:user_id>")
def get_answers(task_id: str, user_id: int):
    verify_logged_in()
    try:
        tid = TaskId.parse(task_id)
    except PluginException as e:
        raise RouteException(str(e))
    d = get_doc_or_abort(tid.doc_id)
    user = User.get_by_id(user_id)
    if user is None:
        raise RouteException('Non-existent user')
    curr_user = get_current_user_object()
    if user_id != get_current_user_id():
```

```
1559         if not verify_seeanswers_access(d, require=False):
1560             if not is_peerreview_enabled(d):
1561                 raise AccessDenied()
1562             if not has_review_access(d, curr_user, tid, user):
1563                 if has_review_access(d, curr_user, None, user):
1564                     return json_response([])
1565                 else:
1566                     raise AccessDenied()
1567 
1568     elif d.document.get_settings().get('need_view_for_answers', False):
1569         verify_view_access(d)
1570     try:
1571         p = find_plugin_from_document(d.document, tid, ↩
        user_context_with_logged_in(user), default_view_ctx)
1572     except TaskNotFoundException:
1573         p = None
1574     user_answers: List[Answer] = user.get_answers_for_task(tid.doc_task).all()
1575     if hide_names_in_teacher(d, context_user=user):
1576         for answer in user_answers:
1577             for u in answer.users_all:
1578                 maybe_hide_name(d, u)
1579     if p and not p.known.show_points() and not curr_user.has_teacher_access(d):
1580         user_answers = list(map(hide_points, user_answers))
1581     return json_response(user_answers)
1582 
1583 
1584 @answers.route("/allDocumentAnswersPlain/<path:doc_path>")
1585 def get_document_answers(doc_path):
1586     d = DocEntry.find_by_path(doc_path, fallback_to_id=True)
1587     pars = d.document.get_dereferenced_paragraphs(default_view_ctx)
1588     task_ids, _, _ = find_task_ids(pars, default_view_ctx, ↩
        user_context_with_logged_in(None))
1589     return get_all_answers_list_plain(task_ids)
1590 
1591 
1592 @answers.route("/allAnswersPlain/<task_id>")
1593 def get_all_answers_plain(task_id):
1594     return get_all_answers_list_plain([TaskId.parse(task_id)])
1595 
1596 
1597 def get_all_answers_list_plain(task_ids: List[TaskId]):
1598     all_answers, format_opt = get_all_answers_as_list(task_ids)
1599     if format_opt == 'json':
1600         return json_response(all_answers)
1601     jointext = "\n"
1602     print_opt = get_option(request, 'print', 'all')
1603     print_answers = print_opt == "all" or print_opt == "answers"
1604     if print_answers:
1605         jointext = ↩
        "\n\n------------------------------------------------------------------------------\n"
1606     text = jointext.join(all_answers)
1607     return Response(text, mimetype='text/plain')
1608 
1609 
1610 def get_all_answers_as_list(task_ids: List[TaskId]):
1611     verify_logged_in()
1612     format_opt = get_option(request, 'format', 'text')
1613     if not task_ids:
1614         return [], format_opt
```

```
    doc_ids = set()
    d = None
    for tid in task_ids:
        doc_ids.add(tid.doc_id)
        d = get_doc_or_abort(tid.doc_id)
        # Require full teacher rights for getting all answers
        verify_teacher_access(d)

    usergroup = get_option(request, 'group', None)
    age = get_option(request, 'age', 'max')
    valid = get_option(request, 'valid', '1')
    name_opt = get_option(request, 'name', 'both')
    sort_opt = get_option(request, 'sort', 'task')
    print_opt = get_option(request, 'print', 'all')
    period_opt = get_option(request, 'period', 'whenever')
    format_opt = get_option(request, 'format', 'text')
    consent = get_consent_opt()
    printname = name_opt == 'both'

    period_from, period_to = period_handling(task_ids, doc_ids, period_opt)

    if not usergroup:
        usergroup = None

    hide_names = name_opt == 'anonymous'
    if d:
        # Above, we're requiring teacher access to all documents, so it does not ↩
    matter which DocInfo we pass here.
        hide_names = hide_names or hide_names_in_teacher(d)
    all_answers = get_all_answers(task_ids,
                                  usergroup,
                                  hide_names,
                                  age,
                                  valid,
                                  printname,
                                  sort_opt,
                                  print_opt,
                                  period_from,
                                  period_to,
                                  format_opt,
                                  consent=consent)
    return all_answers, format_opt


class GraphData(TypedDict):
    data: List[Union[str, float, None]]
    labels: List[str]


@dataclass
class FieldInfo:
    data: UserFields
    aliases: Dict[str, str]
    fieldnames: List[str]
    graphdata: GraphData


def get_plug_vals(doc: DocInfo, tid: TaskId, user_ctx: UserContext, view_ctx: ↩
    ViewContext) -> Optional[FieldInfo]:
```

```
    d, plug = get_plugin_from_request(doc.document, tid, user_ctx, view_ctx)
    flds = plug.known.fields
    if not flds:
        return None

    data, aliases, field_names, _ = get_fields_and_users(
        flds,
        RequestedGroups([user_ctx.user.personal_group_prop]),
        doc,
        user_ctx.logged_user,
        view_ctx,
        add_missing_fields=True,
        access_option=GetFieldsAccess.from_bool(True),
    )
    df = data[0]['fields']
    da = []
    for fn in field_names:
        da.append(df.get(fn, 0))
    return FieldInfo(
        data=df,
        aliases=aliases,
        fieldnames=field_names,
        graphdata={'data': da, 'labels': field_names},
    )


@answers.route("/jsframe/userChange/<task_id>/<user_id>")
def get_jsframe_data(task_id, user_id):
    """
        TODO: check proper rights
    """
    tid = TaskId.parse(task_id)
    doc = get_doc_or_abort(tid.doc_id)
    # verify_seeanswers_access(doc)
    user = User.get_by_id(user_id)
    curr_user = get_current_user_object()
    try:
        vals = get_plug_vals(doc, tid, UserContext(user=user, logged_user=curr_user), ↩
    default_view_ctx)
        return json_response(vals)
    except Exception as e:
        raise RouteException(str(e))
        # return json_response({})


@dataclass
class GetStateModel:
    user_id: int
    answer_id: Optional[int] = None
    par_id: Optional[str] = None
    doc_id: Optional[int] = None
    review: bool = False
    task_id: Optional[str] = None


GetStateSchema = class_schema(GetStateModel)


@answers.route("/getState")
```

```
@use_args(GetStateSchema())
def get_state(args: GetStateModel):
    par_id = args.par_id
    user_id = args.user_id
    answer_id = args.answer_id
    review = args.review
    task_id = args.task_id
    answer = None
    user = User.get_by_id(user_id)
    if user is None:
        raise RouteException('Non-existent user')
    view_ctx = ViewContext(ViewRoute.View, False, origin=get_origin_from_request())
    if answer_id:
        answer = Answer.query.get(answer_id)
        if not answer:
            raise RouteException('Non-existent answer')
        tid = TaskId.parse(answer.task_id)
        d = get_doc_or_abort(tid.doc_id)
        doc_id = d.id
        if not has_review_access(d, get_current_user_object(), tid, user):
            try:
                answer, doc_id = verify_answer_access(
                    answer_id,
                    user_id,
                    view_ctx,
                    allow_grace_period=True,
                )
            except PluginException as e:
                raise RouteException(str(e))
        doc = Document(doc_id)
        tid = TaskId.parse(answer.task_id)
    elif task_id:
        tid = TaskId.parse(task_id)
        d = get_doc_or_abort(tid.doc_id)
        if get_current_user_id() != user_id:
            verify_seeanswers_access(d)
        else:
            verify_view_access(d)
        doc = d.document
    else:
        raise RouteException("Missing answer ID or task ID")

    doc.insert_preamble_pars()
    if par_id:
        tid.maybe_set_hint(par_id)
    user_ctx = user_context_with_logged_in(user)
    try:
        doc, plug = get_plugin_from_request(doc, task_id=tid, u=user_ctx, ↩
    view_ctx=view_ctx)
    except PluginException as e:
        raise RouteException(str(e))
    block = plug.par

    def deref():
        return dereference_pars([block], context_doc=doc, view_ctx=view_ctx)

    presult = pluginify(doc, deref(), user_ctx, view_ctx, custom_answer=answer, ↩
    task_id=task_id, do_lazy=NEVERLAZY,
                        pluginwrap=PluginWrap.Nothing)
```

```
    plug = presult.custom_answer_plugin
    html = plug.get_final_output()
    if review:
        block.final_dict = None
        presult2 = pluginify(doc, deref(), user_ctx, view_ctx, custom_answer=answer, task_id=task_id, do_lazy=NEVERLAZY,
                             review=review, pluginwrap=PluginWrap.Nothing)
        rplug = presult2.custom_answer_plugin
        rhtml = rplug.get_final_output()
        return json_response({'html': html, 'reviewHtml': rhtml})
    else:
        return json_response({'html': html, 'reviewHtml': None})


def verify_answer_access(
        answer_id: int,
        user_id: int,
        view_ctx: ViewContext,
        require_teacher_if_not_own=False,
        required_task_access_level: TaskIdAccess = TaskIdAccess.ReadOnly,
        allow_grace_period: bool = False,
) -> Tuple[Answer, int]:
    answer: Answer = Answer.query.get(answer_id)
    if answer is None:
        raise RouteException('Non-existent answer')
    tid = TaskId.parse(answer.task_id)

    if tid.is_global:
        return answer, tid.doc_id

    d = get_doc_or_abort(tid.doc_id)
    d.document.insert_preamble_pars()

    if verify_teacher_access(d, require=False):
        return answer, tid.doc_id

    user_ctx = user_context_with_logged_in(None)
    if user_id != get_current_user_id() or not logged_in():
        if require_teacher_if_not_own:
            verify_task_access(d, tid, AccessType.teacher, required_task_access_level, user_ctx, view_ctx)
        else:
            verify_task_access(d, tid, AccessType.see_answers, required_task_access_level, user_ctx, view_ctx)
    else:
        verify_task_access(
            d,
            tid,
            AccessType.view,
            required_task_access_level,
            allow_grace_period=allow_grace_period,
            context_user=user_ctx,
            view_ctx=view_ctx,
        )
        if not any(a.id == user_id for a in answer.users_all):
            raise AccessDenied("You don't have access to this answer.")
    return answer, tid.doc_id


```

```
@answers.route("/getTaskUsers/<task_id>")
def get_task_users(task_id):
    tid = TaskId.parse(task_id)
    d = get_doc_or_abort(tid.doc_id)
    if not verify_seeanswers_access(d, require=False):
        curr_user = get_current_user_object()
        if not is_peerreview_enabled(d):
            raise AccessDenied()
        reviews = get_reviews_for_user(d, curr_user)
        if not reviews:
            raise AccessDenied()
        users = list(r.reviewable for r in reviews if r.task_name == tid.task_name)
    else:
        usergroup = request.args.get('group')
        q = (
            User.query
                .options(lazyload(User.groups))
                .join(Answer, User.answers)
                .filter_by(task_id=task_id)
                .join(UserGroup, User.groups)
                .order_by(User.real_name.asc())
        )
        if usergroup is not None:
            q = q.filter(UserGroup.name.in_([usergroup]))
        users = q.all()
    if hide_names_in_teacher(d):
        for user in users:
            maybe_hide_name(d, user)
    return json_response(users)


@answers.route('/renameAnswers/<old_name>/<new_name>/<path:doc_path>')
def rename_answers(old_name: str, new_name: str, doc_path: str):
    d = DocEntry.find_by_path(doc_path, fallback_to_id=True)
    if not d:
        raise NotExist()
    verify_manage_access(d)
    force = get_option(request, 'force', False)
    for n in (old_name, new_name):
        if not re.fullmatch('[a-zA-Z0-9_-]+', n):
            raise RouteException(f'Invalid task name: {n}')
    conflicts = Answer.query.filter_by(task_id=f'{d.id}.{new_name}').count()
    if conflicts > 0 and not force:
        raise RouteException(f'The new name conflicts with {conflicts} other answers ↩
    with the same task name.')
    answers_to_rename = Answer.query.filter_by(task_id=f'{d.id}.{old_name}').all()
    for a in answers_to_rename:
        a.task_id = f'{d.id}.{new_name}'
    db.session.commit()
    return json_response({'modified': len(answers_to_rename), 'conflicts': conflicts})
```

timApp/defaultconfig.py

```
import logging
import multiprocessing
import os
import subprocess
from datetime import timedelta
from pathlib import Path
```

```
7
8  from celery.schedules import crontab
9
10 # NOTE: If you are a different organization (other than JYU), please don't modify ↩
       this file directly.
11 # This avoids merge conflicts. Override the values with prodconfig.py instead.
12
13 ALLOWED_DOCUMENT_UPLOAD_MIMETYPES = ['text/plain']
14 COMPRESS_DEBUG = True
15 COMPRESS_MIMETYPES = ['text/html', 'text/css', 'text/xml', 'application/json', ↩
       'application/javascript']
16 COMPRESS_MIN_SIZE = 50
17 DEBUG = False
18 FILES_PATH = '/tim_files'
19 LOG_DIR = "/service/tim_logs/"
20 LOG_FILE = "timLog.log"
21 LOG_LEVEL = logging.INFO
22 LOG_LEVEL_STDOUT = logging.INFO
23 LOG_PATH = os.path.join(LOG_DIR, LOG_FILE)
24 MAX_CONTENT_LENGTH = 50 * 1024 * 1024
25 PROFILE = False
26 SECRET_KEY = '85db8764yhfZz7-U.-y968buyn89b54y8y45tg'
27 PERMANENT_SESSION_LIFETIME = timedelta(days=14)
28 SQLALCHEMY_TRACK_MODIFICATIONS = False
29 IMMEDIATE_PRELOAD = False
30 LIBSASS_STYLE = "compressed"
31 LIBSASS_INCLUDES = ["node_modules/bootstrap-sass/assets/stylesheets",
32                     "node_modules/eonasdan-bootstrap-datetimepicker/src/sass",
33                     "static"]
34 TIM_NAME = os.environ.get('COMPOSE_PROJECT_NAME', 'tim')
35 TIM_HOST = os.environ.get('TIM_HOST', 'http://localhost')
36 DB_PASSWORD = 'postgresql'
37 DB_URI = f"postgresql://postgres:{DB_PASSWORD}@postgresql:5432/{TIM_NAME}"
38 SASS_GEN_PATH = Path('generated')
39 TEMPLATES_AUTO_RELOAD = True
40 SQLALCHEMY_DATABASE_URI = DB_URI
41 cpus = multiprocessing.cpu_count()
42
43 # If PG_MAX_CONNECTIONS is not defined (possible when running from IDE), we use a ↩
       default value that gives
44 # pool size 2.
45 PG_MAX_CONNECTIONS = os.environ.get('PG_MAX_CONNECTIONS')
46 max_pool_all_workers = int(PG_MAX_CONNECTIONS or cpus * 3 + 5) - 5
47 SQLALCHEMY_POOL_SIZE = (max_pool_all_workers // cpus) - 1
48 SQLALCHEMY_POOL_TIMEOUT = 15
49 SQLALCHEMY_MAX_OVERFLOW = (max_pool_all_workers - SQLALCHEMY_POOL_SIZE * cpus) // cpus
50 LAST_EDITED_BOOKMARK_LIMIT = 15
51 LAST_READ_BOOKMARK_LIMIT = 15
52
53 PLUGIN_COUNT_LAZY_LIMIT = 20
54 QST_PLUGIN_PORT = 5000
55 PLUGIN_CONNECT_TIMEOUT = 0.5
56
57 # When enabled, the readingtypes on_screen and hover_par will not be saved in the ↩
       database.
58 DISABLE_AUTOMATIC_READINGS = False
59 HELP_EMAIL = 'tim@jyu.fi'
60
61 # Default sender address for email.
```

```
MAIL_FROM = 'tim@jyu.fi'

ERROR_EMAIL = 'timwuff.group@korppi.jyu.fi'
WUFF_EMAIL = 'wuff@tim.jyu.fi'
NOREPLY_EMAIL = 'no-reply@tim.jyu.fi'
GLOBAL_NOTIFICATION_FILE = '/tmp/global_notification.html'

GIT_LATEST_COMMIT_TIMESTAMP = subprocess.run(["git", "log", "-1", "--date=format:%d.%m.%Y %H:%M:%S", "--format=%cd"],
                                             stdout=subprocess.PIPE).stdout.decode().strip()
GIT_BRANCH = subprocess.run(["git", "rev-parse", "--abbrev-ref", "HEAD"],
                            stdout=subprocess.PIPE).stdout.decode().strip()

CELERY_BROKER_URL = 'redis://redis:6379'
CELERY_RESULT_BACKEND = 'redis://redis:6379'
CELERY_IMPORTS = ('timApp.tim_celery',)
CELERYBEAT_SCHEDULE = {
    'update-search-files': {
        'task': 'timApp.tim_celery.update_search_files',
        'schedule': crontab(hour='*/12', minute='0'),
    },
    'process-notifications': {
        'task': 'timApp.tim_celery.process_notifications',
        'schedule': crontab(minute='*/5'),
    }
}
# This makes the log format a little less verbose by omitting the Celery task id (which is an UUID).
CELERYD_TASK_LOG_FORMAT = "[%(asctime)s: %(levelname)s/%(processName)s] %(task_name)s: %(message)s"
BEAT_DBURI = DB_URI

MAIL_HOST = "smtp.jyu.fi"
MAIL_SIGNATURE = "\n\n-- \nThis message was automatically sent by TIM"
WTF_CSRF_METHODS = ['POST', 'PUT', 'PATCH', 'DELETE']
WTF_CSRF_HEADERS = ['X-XSRF-TOKEN']
WTF_CSRF_TIME_LIMIT = None
MIN_PASSWORD_LENGTH = 10
PROXY_WHITELIST = [
    'korppi.jyu.fi',
    'plus.cs.aalto.fi',
    'gitlab.com',
    'gitlab.jyu.fi',
    'tim.jyu.fi',
]

# Whitelist of /getproxy domains that don't require login.
PROXY_WHITELIST_NO_LOGIN = {}

SISU_ASSESSMENTS_URL = 'https://s2s.apitest.jyu.fi/assessments/'
SISU_CERT_PATH = '/service/certs/sisu.pem'

SAML_PATH = '/service/timApp/auth/saml/dev'
HAKA_METADATA_URL = 'https://haka.funet.fi/metadata/haka_test_metadata_signed.xml'
HAKA_METADATA_FINGERPRINT = '811dd04e5bde0976be6c7aa6a62e2e633d3de37807642e6c532019674545d019'

# In production, copy these to prodconfig.py and remove the "_PROD" suffix.
```

```
116 SAML_PATH_PROD = '/service/timApp/auth/saml/prod'
117 HAKA_METADATA_URL_PROD = 'https://haka.funet.fi/metadata/haka-metadata.xml'
118 HAKA_METADATA_FINGERPRINT_PROD = ↩
        '70a9058262190cc23f8b0b14d6f0b7c0c74648e8b979bf4258eb7e23674a52f8'
119 # Fingerprint for the upcoming (1.12.2020) v5 certificate.
120 HAKA_METADATA_FINGERPRINT_NEW_PROD = ↩
        'a2c1eff331849cbfbfc920924861e03c8a56414ec003bf919e7f1b1a7dbc3169'
121
122 HOME_ORGANIZATION = 'jyu.fi'
123
124 LOAD_STUDENT_IDS_IN_TEACHER = False
125
126 HAS_HTTPS = TIM_HOST.startswith('https:')
127 SESSION_COOKIE_SAMESITE = 'None' if HAS_HTTPS else None  # Required for Aalto iframe ↩
        to work.
128 SESSION_COOKIE_SECURE = HAS_HTTPS  # Required by Chrome due to SameSite=None setting.
129
130 BOOKMARKS_ENABLED = True
131
132 # If False, only admins can create folders and documents.
133 ALLOW_CREATE_DOCUMENTS = True
134
135 EMAIL_REGISTRATION_ENABLED = True
136 HAKA_ENABLED = True
137
138 # If False, resetting password is not allowed.
139 PASSWORD_RESET_ENABLED = True
140
141 # When enabled, the email login and signup processes are unified so that:
142 #
143 # * only email is asked first
144 # * then the password is requested and TIM asks to check email if the user has not ↩
        logged in before.
145 SIMPLE_EMAIL_LOGIN = False
146
147 # Whether to use a Studyinfo message for help text after email is given.
148 # The point is to warn that TIM will only send the password if the account exists ↩
        (and password is null)
149 # and the email corresponds to the one in Studyinfo.
150 # This only makes sense with EMAIL_REGISTRATION_ENABLED = False.
151 SIMPLE_LOGIN_USE_STUDY_INFO_MESSAGE = False
152
153 LOG_HOST = False
154
155 MAX_ANSWER_CONTENT_SIZE = 200 * 1024  # bytes
156
157 SCIM_ALLOWED_IP = '127.0.0.1'
158
159 # Whether to allow creation of messages lists via GUI. At this moment requires ↩
        Mailman to be configured.
160 MESSAGE_LISTS_ENABLED = False
161 # Settings for mailmanclient-library. Set properly in production.
162 MAILMAN_URL = None
163 MAILMAN_USER = None
164 MAILMAN_PASS = None
165 # Settings for mailman-rest-events library. Set properly in production.
166 MAILMAN_EVENT_API_USER = None
167 MAILMAN_EVENT_API_KEY = None
168 # Link prefix to Postorius Web-UI. If used as is, directs to the mailing lists page.
```

```
MAILMAN_UI_LINK_PREFIX = "https://timlist.it.jyu.fi/postorius/lists/"
# Permitted file extensions allowed on message lists. If this grows large, maybe move to an external file and modify
# getting attachment file extensions from the file instead.
PERMITTED_ATTACHMENTS = ["doc", "docx", "htm", "html", "jpeg", "jpg", "pdf", "png", "ppt", "pptx", "tex", "txt", "xls",
                         "xlsx"]
# These names are reserved from the pool of names for message lists. If need arises, split into TIM and message
# channel specific reserved names.
RESERVED_NAMES = ["postmaster", "listmaster", "admin"]

# If true, prints all SQL statements with tracebacks.
DEBUG_SQL = False

MINIMUM_SCHEDULED_FUNCTION_INTERVAL = 3600

INTERNAL_PLUGIN_DOMAIN = 'tim'

# BACKUP_ANSWER_* variables are related to backing up answers by sending them to another host on the fly.

# When sending an answer to another host, use this secret for authentication.
BACKUP_ANSWER_SEND_SECRET = None

# When receiving an answer from another host, make sure that the given secret matches this one.
BACKUP_ANSWER_RECEIVE_SECRET = None

# In the receiving host, the filename where the answers will be stored, one JSON string per line.
BACKUP_ANSWER_FILE = 'answers.backup'

# The hosts where to back up the answers. Every entry should start with "https://".
BACKUP_ANSWER_HOSTS = None

# DIST_RIGHTS_* variables are related to distributing rights.

# A mapping of target identifiers to lists of hosts.
# Example:
# {
#     'some_exam': {
#         'hosts': ['https://machine1.example.com', 'https://machine2.example.com'],
#         'item': 'path/to/some/exam/here',
#     },
# }
DIST_RIGHTS_HOSTS = {

}

# When registering a right that is going to be distributed, make sure that the given secret matches this one.
DIST_RIGHTS_REGISTER_SECRET = None

# When sending a right to another host, send this secret.
DIST_RIGHTS_SEND_SECRET = None

# When receiving a right from the distributor host, make sure that the given secret matches this one.
```

```
DIST_RIGHTS_RECEIVE_SECRET = None

# A list of documents on this TIM instance that can register and distribute rights directly
DIST_RIGHTS_MODERATION_DOCS = [

]

# Map of items that should trigger rights distribution when unlocking the item.
DIST_RIGHTS_UNLOCK_TARGETS = {
    # 'path/to/item': ['some_target'],
}

# List of hosts to send /register calls.
DIST_RIGHTS_REGISTER_HOSTS = []

# When calling /register, send this secret.
DIST_RIGHTS_REGISTER_SEND_SECRET = None

# The group that is allowed to call /changeStartTime.
DIST_RIGHTS_START_TIME_GROUP = None

# Whether this host is the rights distributor.
DIST_RIGHTS_IS_DISTRIBUTOR = False

# The set of allowed IP networks. The following actions are restricted:
# * Login and email registration are denied for non-admins.
# * Answer route is blocked.
IP_BLOCK_ALLOWLIST = None

# The informational message to display in TIM header if the IP is outside the allowlist.
IP_BLOCK_MESSAGE = None

# If true, IPs that are:
# * outside allowed networks and
# * not in blocklist
# are not blocked but only logged.
IP_BLOCK_LOG_ONLY = False

# The set of documents for which the right is inherited from its containing folder.
INHERIT_FOLDER_RIGHTS_DOCS = {}
```

timApp/item/block.py

```
from __future__ import annotations

from enum import Enum
from typing import Optional, List, TYPE_CHECKING

from sqlalchemy import func
from sqlalchemy.orm.collections import attribute_mapped_collection

from timApp.auth.accesstype import AccessType
from timApp.auth.auth_models import BlockAccess
from timApp.item.blockassociation import BlockAssociation
from timApp.item.tag import Tag
from timApp.messaging.messagelist.messagelist_models import MessageListModel
from timApp.messaging.timMessage.internalmessage_models import InternalMessage, InternalMessageDisplay
```

```
15 from timApp.timdb.sqa import db
16 from timApp.user.usergroup import UserGroup
17 from timApp.user.usergroupdoc import UserGroupDoc
18 from timApp.util.utils import get_current_time
19
20 if TYPE_CHECKING:
21     from timApp.folder.folder import Folder
22
23
24 class Block(db.Model):
25     """The "base class" for all database objects that are part of the permission system."""
26     __tablename__ = 'block'
27     id = db.Column(db.Integer, primary_key=True)
28     """A unique identifier for the Block."""
29
30     latest_revision_id = db.Column(db.Integer)
31     """Old field that is not used anymore."""
32
33     type_id = db.Column(db.Integer, nullable=False)
34     """Type of the Block, see BlockType enum for possible types."""
35
36     description = db.Column(db.Text)
37     """Additional information about the Block. This is used for different purposes by different BlockTypes,
38     so it isn't merely a "description".
39     """
40
41     created = db.Column(db.DateTime(timezone=True), nullable=False, default=func.now())
42     """When this Block was created."""
43
44     modified = db.Column(db.DateTime(timezone=True), default=func.now())
45     """When this Block was last modified."""
46
47     docentries = db.relationship('DocEntry', back_populates='_block')
48     folder = db.relationship('Folder', back_populates='_block', uselist=False)
49     translation = db.relationship('Translation', back_populates='_block', uselist=False,
50                                   foreign_keys="Translation.doc_id")
51     answerupload = db.relationship('AnswerUpload', back_populates='block', lazy='dynamic')
52     accesses = db.relationship(
53         'BlockAccess',
54         back_populates='block',
55         lazy='joined',
56         cascade='all, delete-orphan',
57         collection_class=attribute_mapped_collection('block_collection_key'),
58     )
59     tags: List[Tag] = db.relationship('Tag', back_populates='block', lazy='select')
60     children = db.relationship('Block',
61                                secondary=BlockAssociation.__table__,
62                                primaryjoin=id == BlockAssociation.__table__.c.parent,
63                                secondaryjoin=id == BlockAssociation.__table__.c.child,
64                                lazy='select')
65     parents = db.relationship('Block',
66                               secondary=BlockAssociation.__table__,
67                               primaryjoin=id == BlockAssociation.__table__.c.child,
68                               secondaryjoin=id == BlockAssociation.__table__.c.parent,
69                               lazy='select')
70     notifications = db.relationship('Notification', back_populates='block', ↩
```

```
    lazy='dynamic')

    relevance = db.relationship('BlockRelevance', back_populates='_block', uselist=False)

    # If this Block corresponds to a group's manage document, indicates the group being managed.
    managed_usergroup: Optional[UserGroup] = db.relationship(
        'UserGroup',
        secondary=UserGroupDoc.__table__,
        lazy='select',
        uselist=False,
    )

    #  If this Block corresponds to a message list's manage document, indicates the message list
    #  being managed.
    managed_messagelist: Optional[MessageListModel] = db.relationship("MessageListModel", back_populates="block",
                                                                      lazy="select")

    internalmessage: Optional[InternalMessage] = db.relationship('InternalMessage', back_populates='block')
    internalmessage_display: Optional[InternalMessageDisplay] = db.relationship('InternalMessageDisplay',
                                                                                back_populates='display_block')

    def __json__(self):
        return ['id', 'type_id', 'description', 'created', 'modified']

    @property
    def owners(self) -> List[UserGroup]:
        return [o.usergroup for o in self.owner_accesses]

    @property
    def parent(self) -> Folder:
        if self.type_id == BlockType.Document.value:
            from timApp.document.docentry import DocEntry
            return DocEntry.find_by_id(self.id).parent
        elif self.type_id == BlockType.Folder.value:
            from timApp.folder.folder import Folder
            folder = Folder.get_by_id(self.id)
            return folder.parent

    def is_unpublished(self):
        from timApp.auth.sessioninfo import get_current_user_object
        u = get_current_user_object()
        return u.has_ownership(self) is not None and all(not o.is_large() for o in self.owners) and len(
            self.accesses) <= 1

    @property
    def owner_accesses(self):
        return [a for a in self.accesses.values() if a.type == AccessType.owner.value]

    def set_owner(self, usergroup: UserGroup):
        """Changes the owner group for a block.

```

```
121         :param usergroup: The new usergroup.
122 
123         """
124         self.accesses = {(usergroup.id, AccessType.owner.value): BlockAccess(
125             usergroup_id=usergroup.id,
126             type=AccessType.owner.value,
127             accessible_from=get_current_time(),
128         )}
129 
130     def add_rights(self, groups, access_type: AccessType):
131         for gr in groups:
132             key = (gr.id, access_type.value)
133             self.accesses[key] = BlockAccess(
134                 usergroup_id=gr.id,
135                 type=access_type.value,
136                 accessible_from=get_current_time(),
137             )
138 
139 
140 class BlockType(Enum):
141     Document = 0
142     Comment = 1
143     Note = 2
144     Answer = 3
145     Image = 4
146     Reading = 5
147     Folder = 6
148     File = 7
149     Upload = 8
150     ScheduledFunction = 9
151 
152     @staticmethod
153     def from_str(type_name: str) -> BlockType:
154         return BlockType[type_name.title()]
155 
156 
157 def insert_block(block_type: BlockType, description: Optional[str],
158                  owner_groups: Optional[List[UserGroup]] = None) -> Block:
159     """Inserts a block to database.
160 
161     :param description: The name (description) of the block.
162     :param owner_groups: The owner groups of the block.
163     :param block_type: The type of the block.
164     :returns: The id of the block.
165 
166     """
167     b = Block(description=description, type_id=block_type.value)
168     db.session.add(b)
169     if owner_groups:
170         for owner_group in owner_groups:
171             db.session.flush()
172             access = BlockAccess(block=b,
173                                  usergroup=owner_group,
174                                  type=AccessType.owner.value,
175                                  accessible_from=get_current_time())
176             b.accesses[(owner_group.id, AccessType.owner.value)] = access
177     return b
178 
179 
```

```
def copy_default_rights(item, item_type: BlockType, owners_to_skip: Optional[List[UserGroup]] = None):
    from timApp.user.userutils import grant_access
    from timApp.user.users import get_default_rights_holders
    default_rights: List[BlockAccess] = []
    folder = item.parent
    while folder is not None:
        default_rights += get_default_rights_holders(folder, item_type)
        folder = folder.parent
    for d in default_rights:
        if owners_to_skip and d.usergroup in owners_to_skip and d.access_type == AccessType.owner:
            continue
        grant_access(d.usergroup,
                     item,
                     d.atype.to_enum(),
                     accessible_from=d.accessible_from,
                     accessible_to=d.accessible_to,
                     duration_from=d.duration_from,
                     duration_to=d.duration_to,
                     duration=d.duration)
```

timApp/item/item.py

```
from __future__ import annotations

from itertools import accumulate
from typing import TYPE_CHECKING, Optional

from flask import current_app
from sqlalchemy import tuple_, func
from sqlalchemy.orm import defaultload

from timApp.auth.auth_models import BlockAccess
from timApp.auth.get_user_rights_for_item import get_user_rights_for_item
from timApp.item.block import Block, BlockType
from timApp.item.blockrelevance import BlockRelevance
from timApp.timdb.exceptions import TimDbException
from timApp.timdb.sqa import include_if_loaded


from timApp.util.utils import split_location, date_to_relative, cached_property

if TYPE_CHECKING:
    from timApp.folder.folder import Folder
    from timApp.user.user import User


class ItemBase:
    """An item that can be assigned permissions."""

    @property
    def owners(self):
        return self.block.owners if self.block else None

    @property
    def block(self) -> Block:
        # Relationships are not loaded when constructing an object with __init__.
        if not hasattr(self, '_block') or self._block is None:
```

```
36              self._block = Block.query.get(self.id)
37          return self._block
38
39      @property
40      def id(self):
41          """Returns the item id."""
42          raise NotImplementedError
43
44      @property
45      def last_modified(self):
46          return self.block.modified if self.block else None
47
48      @property
49      def parents(self):
50          return self.block.parents
51
52      @property
53      def children(self):
54          return self.block.children
55
56      @property
57      def relevance(self) -> BlockRelevance:
58          return self.block.relevance if self.block else None
59
60
61  class Item(ItemBase):
62      """An item that exists in the TIM directory hierarchy. Currently ↩
        :class:`~.Folder` and :class:`~.DocInfo`."""
63
64      @property
65      def id(self):
66          raise NotImplementedError
67
68      @property
69      def path(self):
70          """Returns the Document path, including the language part in case of a ↩
        translation."""
71          raise NotImplementedError
72
73      @property
74      def path_without_lang(self):
75          """Returns the Document path without the language part in case of a ↩
        translation."""
76          raise NotImplementedError
77
78      @property
79      def url(self):
80          return current_app.config['TIM_HOST'] + self.url_relative
81
82      def get_url_for_view(self, name: str):
83          return f'{current_app.config["TIM_HOST"]}/{name}/{self.path}'
84
85      def get_relative_url_for_view(self, name: str):
86          return f'/{name}/{self.path}'
87
88      @property
89      def url_relative(self):
90          return '/view/' + self.path
91
```

```
92      @property
93      def location(self):
94          folder, _ = split_location(self.path_without_lang)
95          return folder
96
97      @property
98      def title(self):
99          if self.block is None:
100             return 'All documents'
101         if not self.block.description:
102             return self.short_name
103         return self.block.description
104
105     @title.setter
106     def title(self, value):
107         self.block.description = value
108
109     @property
110     def short_name(self):
111         parts = self.path_without_lang.rsplit('/', 1)
112         return parts[len(parts) - 1]
113
114     def parents_to_root(self, include_root=True, eager_load_groups=False):
115         if not self.path_without_lang:
116             return []
117         path_tuples = self.parent_paths()
118         from timApp.folder.folder import Folder
119         if not path_tuples:
120             return [Folder.get_root()]
121
122         # TODO: Add an option whether to load relevance eagerly or not;
123         #  currently eager by default is better to speed up search cache processing
124         #  and it doesn't slow down other code much.
125         crumbs_q = (
126             Folder.query
127                 .filter(tuple_(Folder.location, Folder.name).in_(path_tuples))
128                 .order_by(func.length(Folder.location).desc())
129                 .options(defaultload(Folder._block).joinedload(Block.relevance))
130         )
131         if eager_load_groups:
132             crumbs_q = (
133                 crumbs_q
134                     .options(defaultload(Folder._block)
135                              .joinedload(Block.accesses)
136                              .joinedload(BlockAccess.usergroup))
137             )
138         crumbs = crumbs_q.all()
139         if include_root:
140             crumbs.append(Folder.get_root())
141         return crumbs
142
143     def parent_paths(self):
144         path_parts = self.path_without_lang.split('/')
145         paths = list(p[1:] for p in accumulate('/' + part for part in path_parts[:-1]))
146         return [split_location(p) for p in paths]
147
148     @cached_property
149     def parents_to_root_eager(self):
150         return self.parents_to_root(eager_load_groups=True)
```

```
151
152     @property
153     def parent(self) -> Folder:  # TODO rename this to parent_folder to distinguish ↩
    better from "parents" attribute
154         folder = self.location
155         from timApp.folder.folder import Folder
156         return Folder.find_by_path(folder) if folder else Folder.get_root()
157
158     @property
159     def public(self):
160         return True
161
162     def to_json(self, curr_user: Optional[User] = None):
163         if curr_user is None:
164             from timApp.auth.sessioninfo import get_current_user_object
165             curr_user = get_current_user_object()
166         return {'name': self.short_name,
167                 'path': self.path,
168                 'title': self.title,
169                 'location': self.location,
170                 'id': self.id,
171                 'modified': date_to_relative(self.last_modified) if ↩
    self.last_modified else None,
172                 'owners': self.owners,
173                 'rights': get_user_rights_for_item(self, curr_user),
174                 'unpublished': self.block.is_unpublished() if self.block else False,
175                 'public': self.public,
176                 # We only add tags if they've already been loaded.
177                 **include_if_loaded('tags', self.block),
178                 **include_if_loaded('relevance', self.block),
179                 }
180
181     def get_relative_path(self, path: str):
182         """Gets the item path relative to the given path.
183         The item must be under the path; otherwise TimDbException is thrown.
184         """
185         path = path.strip('/')
186         if not self.path.startswith(path + '/'):
187             raise TimDbException('Cannot get relative path')
188         return self.path.replace(path + '/', '', 1)
189
190     @staticmethod
191     def find_by_id(item_id):
192         b = Block.query.get(item_id)
193         if b:
194             if b.type_id == BlockType.Document.value:
195                 from timApp.document.docentry import DocEntry
196                 return DocEntry.find_by_id(item_id)
197             elif b.type_id == BlockType.Folder.value:
198                 from timApp.folder.folder import Folder
199                 return Folder.get_by_id(item_id)
200             else:
201                 raise NotImplementedError
202         return None
```

timApp/item/manage.py

```
1 """Routes for manage view."""
2 import re
```

```
3 from dataclasses import dataclass, field
4 from datetime import datetime, timedelta
5 from enum import Enum
6 from typing import Generator, List, Optional, Tuple
7
8 from flask import Blueprint, render_template
9 from flask import redirect
10 from flask import request
11 from isodate import Duration
12
13 from timApp.auth.accesshelper import verify_manage_access, verify_ownership, ↩
      verify_view_access, has_ownership, \
14     verify_edit_access, get_doc_or_abort, get_item_or_abort, get_folder_or_abort, ↩
      verify_copy_access, AccessDenied, \
15     get_single_view_access, has_edit_access
16 from timApp.auth.accesstype import AccessType
17 from timApp.auth.auth_models import AccessTypeModel, BlockAccess
18 from timApp.auth.sessioninfo import get_current_user_group_object
19 from timApp.auth.sessioninfo import get_current_user_object
20 from timApp.document.create_item import copy_document_and_enum_translations
21 from timApp.document.docentry import DocEntry
22 from timApp.document.docinfo import move_document, find_free_name, DocInfo
23 from timApp.folder.createopts import FolderCreationOptions
24 from timApp.folder.folder import Folder, path_includes
25 from timApp.item.block import BlockType, Block
26 from timApp.item.copy_rights import copy_rights
27 from timApp.item.item import Item
28 from timApp.item.validation import validate_item, ↩
      validate_item_and_create_intermediate_folders, has_special_chars
29 from timApp.timdb.sqa import db
30 from timApp.user.user import User, ItemOrBlock
31 from timApp.user.usergroup import UserGroup
32 from timApp.user.users import remove_default_access, get_default_rights_holders, ↩
      get_rights_holders, remove_access
33 from timApp.user.userutils import grant_access, grant_default_access
34 from timApp.util.flask.requesthelper import verify_json_params, get_option, ↩
      use_model, RouteException, NotExist
35 from timApp.util.flask.responsehelper import json_response, ok_response, ↩
      get_grid_modules
36 from timApp.util.logger import log_info
37 from timApp.util.utils import remove_path_special_chars, split_location, ↩
      join_location, get_current_time, \
38     cached_property, seq_to_str
39
40 manage_page = Blueprint('manage_page',
41                         __name__,
42                         url_prefix='')  # TODO: Better URL prefix.
43
44
45 @manage_page.route("/manage/<path:path>")
46 def manage(path):
47     if has_special_chars(path):
48         return redirect(remove_path_special_chars(request.path) + '?' + ↩
      request.query_string.decode('utf8'))
49     item = DocEntry.find_by_path(path, fallback_to_id=True)
50     if item is None:
51         item = Folder.find_by_path(path, fallback_to_id=True)
52         if item is None:
53             raise NotExist()
```

```

    verify_view_access(item)

    is_folder = isinstance(item, Folder)
    if not is_folder and has_edit_access(item):
        item.serialize_content = True
        item.changelog_length = get_option(request, 'history', 100)

    return render_template(
        'manage.jinja2',
        route='manage',
        translations=item.translations if not is_folder else None,
        item=item,
        js=['angular-ui-grid'],
        jsMods=get_grid_modules(),
        orgs=UserGroup.get_organizations(),
        access_types=AccessTypeModel.query.all(),
    )


@manage_page.route("/changelog/<int:doc_id>/<int:length>")
def get_changelog(doc_id, length):
    doc = get_doc_or_abort(doc_id)
    verify_manage_access(doc)
    return json_response({'versions': doc.get_changelog_with_names(length)})


class TimeType(Enum):
    always = 0
    range = 1
    duration = 2


@dataclass
class TimeOpt:
    type: TimeType
    duration: Optional[Duration] = None
    to: Optional[datetime] = None
    ffrom: Optional[datetime] = field(metadata={'data_key': 'from'}, default=None)
    durationTo: Optional[datetime] = None
    durationFrom: Optional[datetime] = None

    @cached_property
    def effective_opt(self):
        acc_to = None
        dur_from = None
        dur_to = None
        duration = None
        accessible_from = get_current_time()
        if self.type == TimeType.range:
            accessible_from = self.ffrom
            acc_to = self.to
        if self.type == TimeType.duration:
            accessible_from = None
            dur_from = self.durationFrom
            acc_to = self.to
            dur_to = self.durationTo
            duration = self.duration_timedelta
        return TimeOpt(
```

```
            type=self.type,
            duration=duration,
            to=acc_to,
            ffrom=accessible_from,
            durationFrom=dur_from,
            durationTo=dur_to,
        )

    @property
    def duration_timedelta(self):
        if not self.duration:
            return None
        if isinstance(self.duration, timedelta):
            return self.duration
        try:
            return self.duration.totimedelta(start=datetime.min)
        except (OverflowError, ValueError):
            raise RouteException('Duration is too long.')


class EditOption(Enum):
    Add = 'add'
    Remove = 'remove'


@dataclass
class PermissionEditModel:
    type: AccessType = field(metadata={'by_value': True})
    time: TimeOpt
    groups: List[str]
    confirm: Optional[bool]

    def __post_init__(self):
        if self.confirm and self.time.type == TimeType.range and self.time.ffrom:
            raise RouteException("Cannot require confirm with start time set")

    @cached_property
    def group_objects(self):
        return UserGroup.query.filter(UserGroup.name.in_(self.groups)).all()

    @property
    def nonexistent_groups(self):
        return sorted(list(set(self.groups) - set(g.name for g in self.group_objects)))


@dataclass
class PermissionSingleEditModel(PermissionEditModel):
    id: int


class DefaultItemType(Enum):
    document = 0
    folder = 1


@dataclass
class DefaultPermissionModel(PermissionSingleEditModel):
    item_type: DefaultItemType

```

```
172
173 @dataclass
174 class PermissionRemoveModel:
175     id: int
176     type: AccessType = field(metadata={'by_value': True})
177     group: int
178
179
180 @dataclass
181 class DefaultPermissionRemoveModel(PermissionRemoveModel):
182     item_type: DefaultItemType
183
184
185 @dataclass
186 class PermissionMassEditModel(PermissionEditModel):
187     ids: List[int]
188     action: EditOption = field(metadata={'by_value': True})
189
190
191 @manage_page.route("/permissions/add", methods=["PUT"])
192 @use_model(PermissionSingleEditModel)
193 def add_permission(m: PermissionSingleEditModel):
194     i = get_item_or_abort(m.id)
195     is_owner = verify_permission_edit_access(i, m.type)
196     accs = add_perm(m, i)
197     if accs:
198         a = accs[0]
199         check_ownership_loss(is_owner, i)
200         log_right(f'added {a.info_str} for {seq_to_str(m.groups)} in {i.path}')
201         db.session.commit()
202     return permission_response(m)
203
204
205 def permission_response(m: PermissionEditModel):
206     return json_response({'not_exist': m.nonexistent_groups})
207
208
209 def log_right(s: str):
210     u = get_current_user_object()
211     log_info(f'RIGHTS: {u.name} {s}')
212
213
214 def get_group_and_doc(doc_id: int, username: str) -> Tuple[UserGroup, DocInfo]:
215     i = get_item_or_abort(doc_id)
216     verify_permission_edit_access(i, AccessType.view)
217     g = UserGroup.get_by_name(username)
218     if not g:
219         raise RouteException('User not found')
220     return g, i
221
222
223 @manage_page.route("/permissions/expire/<int:doc_id>/<username>")
224 def expire_permission_url(doc_id: int, username: str):
225     g, i = get_group_and_doc(doc_id, username)
226     ba: Optional[BlockAccess] = BlockAccess.query.filter_by(
227         type=AccessType.view.value,
228         block_id=i.id,
229         usergroup_id=g.id,
230     ).first()
```

```
231     if not ba:
232         raise RouteException('Right not found.')
233     if ba.expired:
234         raise RouteException('Right is already expired.')
235     ba.accessible_to = get_current_time()
236     if ba.duration:
237         ba.duration = None
238         ba.duration_from = None
239         ba.duration_to = None
240     db.session.commit()
241     return ok_response()
242 
243 
244 @manage_page.route("/permissions/confirm/<int:doc_id>/<username>")
245 def confirm_permission_url(doc_id: int, username: str):
246     g, i = get_group_and_doc(doc_id, username)
247     m = PermissionRemoveModel(id=doc_id, type=AccessType.view, group=g.id)
248     return do_confirm_permission(m, i)
249 
250 
251 @manage_page.route("/permissions/confirm", methods=["PUT"])
252 @use_model(PermissionRemoveModel)
253 def confirm_permission(m: PermissionRemoveModel):
254     i = get_item_or_abort(m.id)
255     verify_permission_edit_access(i, m.type)
256     return do_confirm_permission(m, i)
257 
258 
259 def do_confirm_permission(m: PermissionRemoveModel, i: DocInfo):
260     ba: Optional[BlockAccess] = BlockAccess.query.filter_by(
261         type=m.type.value,
262         block_id=m.id,
263         usergroup_id=m.group,
264     ).first()
265     if not ba:
266         raise RouteException('Right not found.')
267     if not ba.require_confirm:
268         raise RouteException(f'{m.type.name} right for {ba.usergroup.name} does not ↩
    require confirmation or it was already confirmed.')
269     ba.do_confirm()
270     ug: UserGroup = UserGroup.query.get(m.group)
271     log_right(f'confirmed {ba.info_str} for {ug.name} in {i.path}')
272     db.session.commit()
273     return ok_response()
274 
275 
276 @manage_page.route("/permissions/edit", methods=["put"])
277 @use_model(PermissionMassEditModel)
278 def edit_permissions(m: PermissionMassEditModel):
279     groups = m.group_objects
280     nonexistent = set(m.groups) - set(g.name for g in groups)
281     if nonexistent:
282         raise RouteException(f'Non-existent groups: {nonexistent}')
283     items = Block.query.filter(Block.id.in_(m.ids)
284                                & Block.type_id.in_([BlockType.Document.value, ↩
    BlockType.Folder.value])).order_by(Block.id).all()
285     a = None
286     owned_items_before = set()
287     for i in items:
```

```
288         checked_owner = verify_permission_edit_access(i, m.type)
289         if checked_owner:
290             owned_items_before.add(i)
291         if m.action == EditOption.Add:
292             accs = add_perm(m, i)
293             if accs:
294                 a = accs[0]
295         else:
296             for g in groups:
297                 a = remove_perm(g, i, m.type) or a
298 
299     if m.type == AccessType.owner:
300         owned_items_after = set()
301         u = get_current_user_object()
302         for i in items:
303             if u.has_ownership(i):
304                 owned_items_after.add(i)
305         if owned_items_before != owned_items_after:
306             raise AccessDenied('You cannot remove ownership from yourself.')
307     if a:
308         action = 'added' if m.action == EditOption.Add else 'removed'
309         log_right(f'{action} {a.info_str} for {seq_to_str(m.groups)} in blocks: ↩
    {seq_to_str(list(str(x) for x in m.ids))}')
310         db.session.commit()
311     return permission_response(m)
312 
313 
314 def add_perm(
315         p: PermissionEditModel,
316         item: Item,
317 ) -> List[BlockAccess]:
318     if get_current_user_object().get_personal_folder().id == item.id:
319         if p.type == AccessType.owner:
320             raise AccessDenied('You cannot add owners to your personal folder.')
321     opt = p.time.effective_opt
322     accs = []
323     for group in p.group_objects:
324         a = grant_access(
325             group,
326             item,
327             p.type,
328             accessible_from=opt.ffrom,
329             accessible_to=opt.to,
330             duration_from=opt.durationFrom,
331             duration_to=opt.durationTo,
332             duration=opt.duration,
333             require_confirm=p.confirm,
334         )
335         accs.append(a)
336     return accs
337 
338 
339 @manage_page.route("/permissions/remove", methods=["PUT"])
340 @use_model(PermissionRemoveModel)
341 def remove_permission(m: PermissionRemoveModel):
342     i = get_item_or_abort(m.id)
343     had_ownership = verify_permission_edit_access(i, m.type)
344     ug: UserGroup = UserGroup.query.get(m.group)
345     if not ug:
```

```
346         raise RouteException('User group not found')
347     a = remove_perm(ug, i.block, m.type)
348     check_ownership_loss(had_ownership, i)
349 
350     log_right(f'removed {a.info_str} for {ug.name} in {i.path}')
351     db.session.commit()
352     return ok_response()
353 
354 
355 @dataclass
356 class PermissionClearModel:
357     paths: List[str]
358     type: AccessType = field(metadata={'by_value': True})
359 
360 
361 @manage_page.route("/permissions/clear", methods=["PUT"])
362 @use_model(PermissionClearModel)
363 def clear_permissions(m: PermissionClearModel):
364     for p in m.paths:
365         i = DocEntry.find_by_path(p, try_translation=True)
366         if not i:
367             i = Folder.find_by_path(p)
368         if not i:
369             raise RouteException(f'Item not found: {p}')
370         verify_ownership(i)
371         i.block.accesses = {
372             (ugid, permtype): v for (ugid, permtype), v in i.block.accesses.items() if
373             permtype != m.type.value
374         }
375     db.session.commit()
376     return ok_response()
377 
378 
379 @dataclass
380 class SelfExpireModel:
381     id: int
382 
383 
384 @manage_page.route("/permissions/selfExpire", methods=["post"])
385 @use_model(SelfExpireModel)
386 def self_expire_permission(m: SelfExpireModel):
387     i = get_item_or_abort(m.id)
388     acc = verify_view_access(i, require=False)
389     if not acc:
390         return ok_response()
391     acc = get_single_view_access(i)
392     acc.accessible_to = get_current_time()
393     log_right(f'self-expired view access in {i.path}')
394     db.session.commit()
395     return ok_response()
396 
397 
398 def remove_perm(group: UserGroup, b: Block, t: AccessType):
399     return remove_access(group, b, t)
400 
401 
402 def check_ownership_loss(had_ownership, item):
403     db.session.flush()
404     db.session.refresh(item)
```

```
405     if had_ownership and not has_ownership(item):
406         raise AccessDenied('You cannot remove ownership from yourself.')
407 
408 
409 @manage_page.route("/alias/<int:doc_id>", methods=["GET"])
410 def get_doc_names(doc_id):
411     d = get_doc_or_abort(doc_id)
412     verify_manage_access(d)
413     return json_response(d.aliases)
414 
415 
416 @manage_page.route("/alias/<int:doc_id>/<path:new_alias>", methods=["PUT"])
417 def add_alias(doc_id, new_alias):
418     d = get_doc_or_abort(doc_id)
419     verify_manage_access(d)
420     is_public, = verify_json_params('public', require=False, default=True)
421 
422     new_alias = new_alias.strip('/')
423 
424     validate_item_and_create_intermediate_folders(new_alias, BlockType.Document, ↩
    get_current_user_group_object())
425     d.add_alias(new_alias, is_public)
426     db.session.commit()
427     return ok_response()
428 
429 
430 @manage_page.route("/alias/<path:alias>", methods=["POST"])
431 def change_alias(alias):
432     alias = alias.strip('/')
433     new_alias, = verify_json_params('new_name')
434     new_alias = new_alias.strip('/')
435     is_public, = verify_json_params('public', require=False, default=True)
436 
437     doc = DocEntry.find_by_path(alias, try_translation=False)
438     if doc is None:
439         raise NotExist('The document does not exist!')
440 
441     verify_manage_access(doc)
442 
443     if alias != new_alias:
444         src_f = Folder.find_first_existing(alias)
445         if not get_current_user_object().can_write_to_folder(src_f):
446             raise AccessDenied("You don't have permission to write to the source ↩
    folder.")
447         validate_item_and_create_intermediate_folders(new_alias, BlockType.Document, ↩
    get_current_user_group_object())
448 
449     doc.name = new_alias
450     doc.public = is_public
451     if all(not a.public for a in doc.aliases):
452         raise RouteException('This is the only visible name for this document, so you ↩
    cannot make it invisible.')
453     db.session.commit()
454     return ok_response()
455 
456 
457 @manage_page.route("/alias/<path:alias>", methods=["DELETE"])
458 def remove_alias(alias):
459     alias = alias.strip('/')
```

```
460
461     doc = DocEntry.find_by_path(alias, try_translation=False)
462     if doc is None:
463         raise NotExist('The document does not exist!')
464
465     verify_manage_access(doc)
466
467     if len(doc.aliases) <= 1:
468         raise AccessDenied("You can't delete the only name the document has.")
469
470     f = Folder.find_first_existing(alias)
471     if not get_current_user_object().can_write_to_folder(f):
472         raise AccessDenied("You don't have permission to write to that folder.")
473
474     db.session.delete(doc)
475     db.session.commit()
476     return ok_response()
477
478
479 @manage_page.route("/rename/<int:item_id>", methods=["PUT"])
480 def rename_folder(item_id):
481     new_name = request.get_json()['new_name'].strip('/')
482
483     d = DocEntry.find_by_id(item_id)
484     if d:
485         raise AccessDenied('Rename route is no longer supported for documents.')
486
487     f = get_folder_or_abort(item_id)
488     verify_manage_access(f)
489
490     parent, _ = split_location(new_name)
491     parent_f = Folder.find_by_path(parent)
492
493     if parent_f is None:
494         # Maybe do a recursive create with permission checks here later?
495         raise AccessDenied("The location does not exist.")
496
497     if parent_f.id == item_id:
498         raise AccessDenied("A folder cannot contain itself.")
499
500     validate_item(new_name, BlockType.Folder)
501
502     f.rename_path(new_name)
503     db.session.commit()
504     return json_response({'new_name': new_name})
505
506
507 @manage_page.route("/permissions/get/<int:item_id>")
508 def get_permissions(item_id):
509     i = get_item_or_abort(item_id)
510     verify_manage_access(i)
511     grouprights = get_rights_holders(item_id)
512     return json_response({
513         'grouprights': grouprights,
514         'accesstypes': AccessTypeModel.query.all(),
515         'orgs': UserGroup.get_organizations(),
516     }, date_conversion=True)
517
518
```

```
519 @manage_page.route("/defaultPermissions/<object_type>/get/<int:folder_id>")
520 def get_default_document_permissions(folder_id, object_type):
521     f = get_folder_or_abort(folder_id)
522     verify_manage_access(f)
523     grouprights = get_default_rights_holders(f, BlockType.from_str(object_type))
524     return json_response({'grouprights': grouprights}, date_conversion=True)
525
526
527 @manage_page.route("/defaultPermissions/add", methods=["PUT"])
528 @use_model(DefaultPermissionModel)
529 def add_default_doc_permission(m: DefaultPermissionModel):
530     i = get_folder_or_abort(m.id)
531     verify_permission_edit_access(i, m.type)
532     opt = m.time.effective_opt
533     grant_default_access(
534         m.group_objects,
535         i,
536         m.type,
537         BlockType.from_str(m.item_type.name),
538         accessible_from=opt.ffrom,
539         accessible_to=opt.to,
540         duration_from=opt.durationFrom,
541         duration_to=opt.durationTo,
542         duration=opt.duration,
543     )
544     db.session.commit()
545     return permission_response(m)
546
547
548 @manage_page.route("/defaultPermissions/remove", methods=["PUT"])
549 @use_model(DefaultPermissionRemoveModel)
550 def remove_default_doc_permission(m: DefaultPermissionRemoveModel):
551     f = get_folder_or_abort(m.id)
552     verify_permission_edit_access(f, m.type)
553     ug = UserGroup.query.get(m.group)
554     if not ug:
555         raise NotExist('Usergroup not found')
556     remove_default_access(ug, f, m.type, BlockType.from_str(m.item_type.name))
557     db.session.commit()
558     return ok_response()
559
560
561 def verify_permission_edit_access(i: ItemOrBlock, perm_type: AccessType) -> bool:
562     """Verifies that the user has right to edit a permission.
563
564     :param i: The item to check for permission.
565     :param perm_type: The permission type.
566     :return: True if owner permission was checked, false if just manage access.
567
568     """
569     if perm_type == AccessType.owner:
570         verify_ownership(i)
571         return True
572     else:
573         verify_manage_access(i)
574         return False
575
576
577 @manage_page.route("/documents/<int:doc_id>", methods=["DELETE"])
```

```
578 def del_document(doc_id):
579     d = get_doc_or_abort(doc_id)
580     verify_ownership(d)
581     f = get_trash_folder()
582     move_document(d, f)
583     db.session.commit()
584     return ok_response()
585 
586 
587 def get_trash_folder() -> Folder:
588     trash_folder_path = f'roskis'
589     f = Folder.find_by_path(trash_folder_path)
590     if not f:
591         f = Folder.create(trash_folder_path, ↩
        owner_groups=UserGroup.get_admin_group(), title='Roskakori')
592     return f
593 
594 
595 @manage_page.route("/folders/<folder_id>", methods=["DELETE"])
596 def delete_folder(folder_id):
597     f = get_folder_or_abort(folder_id)
598     verify_ownership(f)
599     if f.location == 'users':
600         raise AccessDenied('Personal folders cannot be deleted.')
601     trash = get_trash_folder()
602     if f.location == trash.path:
603         raise RouteException('Folder is already deleted.')
604     trash_path = find_free_name(trash, f)
605     f.rename_path(trash_path)
606     db.session.commit()
607     return ok_response()
608 
609 
610 @manage_page.route("/changeTitle/<int:item_id>", methods=["PUT"])
611 def change_title(item_id):
612     item = get_item_or_abort(item_id)
613     verify_edit_access(item)
614     new_title, = verify_json_params('new_title')
615     item.title = new_title
616     db.session.commit()
617     return ok_response()
618 
619 
620 def get_copy_folder_params(folder_id):
621     f = get_folder_or_abort(folder_id)
622     verify_copy_access(f, message=f'Missing copy access to folder {f.path}')
623     dest, exclude = verify_json_params('destination', 'exclude')
624     compiled = get_pattern(exclude)
625     if path_includes(dest, f.path):
626         raise AccessDenied('Cannot copy folder inside of itself.')
627     return f, dest, compiled
628 
629 
630 @manage_page.route("/copy/<int:folder_id>", methods=["POST"])
631 def copy_folder_endpoint(folder_id):
632     f, dest, compiled = get_copy_folder_params(folder_id)
633     o = get_current_user_group_object()
634     nf = Folder.find_by_path(dest)
635     if not nf:
```

```
636         validate_item_and_create_intermediate_folders(dest, BlockType.Folder, o)
637         nf = Folder.create(dest, o, ↩
      creation_opts=FolderCreationOptions(apply_default_rights=True))
638     u = get_current_user_object()
639     copy_folder(f, nf, u, compiled)
640     db.session.commit()
641     return json_response(nf)
642
643
644 def get_pattern(exclude: str):
645     if not exclude:
646         exclude = 'a^'
647     try:
648         return re.compile(exclude)
649     except:
650         raise RouteException(f'Wrong pattern format: {exclude}')
651
652
653 @manage_page.route("/copy/<int:folder_id>/preview", methods=["POST"])
654 def copy_folder_preview(folder_id):
655     f, dest, compiled = get_copy_folder_params(folder_id)
656     preview_list = []
657     for i in enum_items(f, compiled):
658         preview_list.append({
659             'to': join_location(dest, i.get_relative_path(f.path)),
660             'from': i.path,
661         })
662     return json_response({
663         'preview': preview_list,
664         'dest_exists': Folder.find_by_path(dest) is not None,
665     })
666
667
668 def enum_items(folder: Folder, exclude_re) -> Generator[Item, None, None]:
669     for d in folder.get_all_documents(include_subdirs=False):
670         if not exclude_re.search(d.path):
671             yield d
672     for f in folder.get_all_folders():
673         if not exclude_re.search(f.path):
674             yield f
675             yield from enum_items(f, exclude_re)
676
677
678 def copy_folder(f_from: Folder, f_to: Folder, user_who_copies: User, exclude_re):
679     db.session.flush()
680     if not user_who_copies.can_write_to_folder(f_to):
681         raise AccessDenied(f'Missing edit access to folder {f_to.path}')
682     if not user_who_copies.has_copy_access(f_from):
683         raise AccessDenied(f'Missing copy access to folder {f_from.path}')
684     folder_opts = FolderCreationOptions(get_templates_rights_from_parent=False)
685     for d in f_from.get_all_documents(include_subdirs=False):
686         if exclude_re.search(d.path):
687             continue
688         if not user_who_copies.has_copy_access(d):
689             raise AccessDenied(f'Missing copy access to document {d.path}')
690         nd_path = join_location(f_to.path, d.short_name)
691         if DocEntry.find_by_path(nd_path):
692             raise AccessDenied(f'Document already exists at path {nd_path}')
693         nd = DocEntry.create(
```

```
            nd_path,
            title=d.title,
            folder_opts=folder_opts,
        )
        copy_rights(d, nd, new_owner=user_who_copies)
        nd.document.modifier_group_id = user_who_copies.get_personal_group().id
        for tr, new_tr in copy_document_and_enum_translations(d, nd, copy_uploads=True):
            copy_rights(tr, new_tr, new_owner=user_who_copies)
    for f in f_from.get_all_folders():
        if exclude_re.search(f.path):
            continue
        nf_path = join_location(f_to.path, f.short_name)
        nf = Folder.find_by_path(nf_path)
        if nf:
            pass
        else:
            nf = Folder.create(
                nf_path,
                title=f.title,
                creation_opts=folder_opts,
            )
            copy_rights(f, nf, new_owner=user_who_copies)
        copy_folder(f, nf, user_who_copies, exclude_re)
```

timApp/markdown/htmlSanitize.py

```
# -*- coding: utf-8 -*-
import lxml
import lxml.etree
from lxml.html import fromstring, tostring
from lxml.html.clean import Cleaner

TIM_SAFE_TAGS = [
    'a',
    'abbr',
    'acronym',
    'aside',
    'b',
    'blockquote',
    'button',
    'code',
    'em',
    'figcaption',
    'figure',
    'i',
    'li',
    'ol',
    'strong',
    'ul',
    'video',
    'p',
    'code',
    'div',
    'span',
    'br',
    'pre',
    'img',
    'h1',
    'h2',
```

```
    'h3',
    'h4',
    'h5',
    'h6',
    'h7',
    'hr',
    'input',
    'label',
    'table',
    'tbody',
    'thead',
    'tfoot',
    'td',
    'tr',
    'th',
    'caption',
    'colgroup',
    'col',
    'sub',
    'sup',
    'u',
    's',
    'tim-login-menu',
    'tim-plugin-loader',
    # plugin components:
    'cs-comtest-runner',
    'cs-comtest-runner-input',
    'cs-console',
    'cs-jsav-runner',
    'cs-jypeli-runner',
    'cs-jypeli-runner-input',
    'cs-parsons-runner',
    'cs-runner',
    'cs-runner-input',
    'cs-sage-runner',
    'tim-geogebra',
    'cs-simcir-runner',
    'cs-tauno-runner',
    'cs-tauno-runner-input',
    'cs-text-runner',
    'dropdown-runner',
    'feedback-runner',
    'drag-runner',
    'imagex-runner',
    'js-runner',
    'mcq',
    'mmcq',
    'pali-runner',
    'tim-multisave',
    'textfield-runner',
    'cbcountfield-runner',
    'cbfield-runner',
    'rbfield-runner',
    'numericfield-runner',
    'goaltable-runner',
    'jsframe-runner',
    'tim-video',
    'importdata-runner',
    'tim-table',
```

```
93
94      # raw AngularJS components:
95      'tim-rights-editor',
96      'tim-self-expire',
97      'tim-mark-all-as-read',
98      'tim-add-member',
99      'tim-goto-link',
100     'tim-graph-viz',
101     'tim-variables',
102     'tim-message-list-admin',
103     'tim-message-view',
104     'manage-read-receipt',
105 ]
106
107 TIM_SAFE_ATTRS_MAP = {'*': ['class', 'id', 'align'],
108                       'video': ['src', 'controls'],
109                       'abbr': ['title'],
110                       'acronym': ['title'],
111                       'img': ['src', 'width', 'height'],
112                       'a': ['href', 'title', 'target']}
113
114 TIM_SAFE_ATTRS = frozenset([
115     'abbr', 'accept', 'accept-charset', 'accesskey', 'action', 'align',
116     'alt', 'axis', 'border', 'cellpadding', 'cellspacing', 'char', 'charoff',
117     'charset', 'checked', 'cite', 'class', 'clear', 'cols', 'colspan',
118     'color', 'compact', 'coords', 'datetime', 'dir', 'disabled', 'enctype',
119     'for', 'frame', 'headers', 'height', 'href', 'hreflang', 'hspace', 'id',
120     'ismap', 'label', 'lang', 'longdesc', 'maxlength', 'media', 'method',
121     'multiple', 'name', 'nohref', 'noshade', 'nowrap', 'prompt', 'readonly',
122     'rel', 'rev', 'rows', 'rowspan', 'rules', 'scope', 'selected', 'shape',
123     'size', 'span', 'src', 'start', 'style', 'summary', 'tabindex', 'target', 'title',
124     'type', 'usemap', 'valign', 'value', 'vspace', 'width', 'controls', 'plugin',
125     'json', 'data-plugin', 'data-answer-id', 'answer-id', 'task-id', 'placeholder', ↩
    'data-html',
126
127     # tim-rights-editor
128     'item-id', 'allow-select-action', 'barcode-mode', 'restrict-rights', 'hide-remove',
129     'hide-edit', 'hide-expire', 'confirm-expire',
130     'force-duration', 'force-duration-start', 'force-duration-end', 'force-confirm',
131
132     # tim-self-expire
133     'button-text', 'confirm',
134
135     # tim-table
136     'bind-data',
137
138     # tim-add-member
139     'group',
140
141     # tim-goto-link
142     'auto-open',
143     'check-unsaved',
144     'close-at',
145     'countdown-text',
146     'is-button',
147     'max-wait',
148     'open-at',
149     'past-due-text',
150     'reset-time',
```

```
    'stop-after-countdown',
    'time-lang',
    'unauthorized-text',
    'unsaved-changes-text',
    'wait-text',

    # viz and vars:
    'usercode',
    'vizcmd',
    'height',
    'jsparams',

])

c_no_style = Cleaner(
    allow_tags=TIM_SAFE_TAGS,
    comments=False,
    forms=False,
    remove_unknown_tags=False,
    safe_attrs=TIM_SAFE_ATTRS,
)

c_with_styles = Cleaner(
    allow_tags=TIM_SAFE_TAGS + ['style'],
    comments=False,
    forms=False,
    remove_unknown_tags=False,
    safe_attrs=TIM_SAFE_ATTRS,
)


# NOTE: lxml cleaner is a LOT faster than bleach.
def sanitize_html(html_string: str, allow_styles: bool = False) -> str:
    cleaner = c_with_styles if allow_styles else c_no_style
    try:
        doc = fromstring(html_string)
        cleaner(doc)
        cleaned = tostring(doc, encoding='ascii').decode('ascii')
        return strip_div(cleaned)
    except lxml.etree.ParserError:  # Thrown if the HTML string is empty
        return ""
    except lxml.etree.XMLSyntaxError:  # Not yet sure why thrown
        return ""
    except ValueError:  # Thrown if XML has an encoding declaration
        return ""


def strip_div(s: str) -> str:
    if s.startswith('<div>') and s.endswith('</div>'):
        return s[5:-6]
    else:
        return s
```

timApp/static/scripts/tim/main.ts

```
import "./loadJQueryAndMomentGlobals";
import "reflect-metadata";

import {enableProdMode} from "@angular/core";
```

```
5  import angular from "angular";
6  import bootstrap from "bootstrap";
7  import "eonasdan-bootstrap-datetimepicker";
8  import $ from "jquery";
9  import * as answerbrowser from "tim/answer/answerbrowser3";
10 import * as userlistController from "tim/answer/userlistController";
11 import {timApp} from "tim/app";
12 import * as viewctrl from "tim/document/viewctrl";
13 import {ViewRangeNavigationComponent} from "tim/document/view-range-navigation.component";
14 import {environment} from "tim/environments/environment";
15 import {FrontPageComponent} from "tim/frontpage/front-page.component";
16 import * as loadMap from "tim/gamification/loadMap";
17 import * as manageCtrl from "tim/item/manageCtrl";
18 import * as rightsEditor from "tim/item/rightsEditor";
19 import * as markAllAsRead from "tim/ui/markAllAsRead";
20 import {BootstrapPanelComponent} from "tim/ui/bootstrap-panel.component";
21 import {LogoComponent} from "tim/ui/logo.component";
22 import {LoginMenuComponent} from "tim/user/login-menu.component";
23 import * as timRoot from "tim/timRoot";
24 import {SettingsComponent} from "tim/user/settings.component";
25 import {markAsUsed, ModuleArray, StringArray} from "tim/util/utils";
26 import {AnnotationComponent} from "tim/velp/annotation.component";
27 import * as velpSelection from "tim/velp/velpSelection";
28 import {staticDynamicImport} from "tim/staticDynamicImport";
29 import {AppModule} from "tim/app.module";
30 import {HeaderComponent} from "tim/header/header.component";
31 import {CreateItemComponent} from "tim/item/create-item.component";
32 import {TimAlertComponent} from "tim/ui/tim-alert.component";
33 import {createDowngradedModule, doDowngrade} from "tim/downgrade";
34 import {platformBrowserDynamic} from "@angular/platform-browser-dynamic";
35 import {setAngularJSGlobal} from "@angular/upgrade/static";
36 import {MarkupErrorComponent} from "tim/ui/markup-error.component";
37 import {LoadingComponent} from "tim/ui/loading.component";
38 import {VelpSummaryComponent} from "tim/velp/velp-summary.component";
39 import {DialogComponent} from "tim/ui/dialog.component";
40 import {CloseButtonComponent} from "tim/ui/close-button.component";
41 import {DialogContainerComponent} from "tim/ui/angulardialog/dialog-container.component";
42 import {AddMemberComponent} from "tim/ui/add-member.component";
43 import {FooterComponent} from "tim/footer.component";
44 import {SiteHeaderComponent} from "tim/header/site-header.component";
45 import {TimeLeftComponent} from "tim/ui/time-left.component";
46 import {CountdownComponent} from "tim/ui/countdown.component";
47 import {AccessCountdownComponent} from "tim/item/access-countdown.component";
48 import {GotoLinkComponent} from "tim/ui/goto-link.component";
49 import {SidebarMenuComponent} from "tim/sidebarmenu/sidebar-menu.component";
50 import BackspaceDisabler from "backspace-disabler";
51 import {DrawToolbarComponent} from "tim/plugin/drawToolbar";
52 import {DrawCanvasComponent} from "tim/plugin/drawCanvas";
53 import {DirectoryListComponent} from "tim/folder/directory-list.component";
54 import {TemplateListComponent} from "tim/document/editing/template-list.component";
55 import * as selfExpire from "tim/item/selfExpire";
56 import {HelpParContent} from "tim/document/editing/help-par-content.component";
57 import {DurationPickerComponent} from "tim/ui/duration-picker.component";
58 import {RelevanceEditComponent} from "tim/item/relevance-edit.component";
59 import {MessageListAdminComponent} from "tim/messaging/message-list-admin.component";
60 import {TimMessageViewComponent} from "tim/messaging/tim-message-view.component";
61 import {ManageReadReceiptComponent} from "tim/messaging/manage-read-receipt.component";
```

```
62 import {insertLogDivIfEnabled, timLogInit, timLogTime} from "./util/timTiming";
63 import {genericglobals} from "./util/globals";
64 import {ParCompiler} from "./editor/parCompiler";
65
66 BackspaceDisabler.disable();
67
68 if (environment.production) {
69     enableProdMode();
70 }
71
72 markAsUsed(
73     answerbrowser,
74     bootstrap,
75     loadMap,
76     manageCtrl,
77     rightsEditor,
78     selfExpire,
79     timRoot,
80     userlistController,
81     velpSelection,
82     viewctrl,
83     markAllAsRead
84 );
85
86 setAngularJSGlobal(angular);
87
88 function createDowngradedAppModule() {
89     const dg = createDowngradedModule((extraProviders) => {
90         const platformRef = platformBrowserDynamic(extraProviders);
91         return platformRef.bootstrapModule(AppModule);
92     });
93     doDowngrade(dg, "timHeader", HeaderComponent);
94     doDowngrade(dg, "createItem", CreateItemComponent);
95     doDowngrade(dg, "timAlert", TimAlertComponent);
96     doDowngrade(dg, "timMarkupError", MarkupErrorComponent);
97     doDowngrade(dg, "timLoading", LoadingComponent);
98     doDowngrade(dg, "annotation", AnnotationComponent);
99     doDowngrade(dg, "velpSummary", VelpSummaryComponent);
100    doDowngrade(dg, "timDialog", DialogComponent);
101    doDowngrade(dg, "timCloseButton", CloseButtonComponent);
102    doDowngrade(dg, "timDialogContainer", DialogContainerComponent);
103    doDowngrade(dg, "timAddMember", AddMemberComponent);
104    doDowngrade(dg, "timFooter", FooterComponent);
105    doDowngrade(dg, "timLoginMenu", LoginMenuComponent);
106    doDowngrade(dg, "timLogo", LogoComponent);
107    doDowngrade(dg, "bootstrapPanel", BootstrapPanelComponent);
108    doDowngrade(dg, "timStart", FrontPageComponent);
109    doDowngrade(dg, "timSiteHeader", SiteHeaderComponent);
110    doDowngrade(dg, "timSettings", SettingsComponent);
111    doDowngrade(dg, "timAccessCountdown", AccessCountdownComponent);
112    doDowngrade(dg, "timGotoLink", GotoLinkComponent);
113    doDowngrade(dg, "timTimeLeft", TimeLeftComponent);
114    doDowngrade(dg, "timCountdown", CountdownComponent);
115    doDowngrade(dg, "timSidebarMenu", SidebarMenuComponent);
116    doDowngrade(dg, "timDrawToolbar", DrawToolbarComponent);
117    doDowngrade(dg, "timDrawCanvas", DrawCanvasComponent);
118    doDowngrade(dg, "timIndex", DirectoryListComponent);
119    doDowngrade(dg, "timTemplateList", TemplateListComponent);
120    doDowngrade(dg, "timViewRangeNavigation", ViewRangeNavigationComponent);
```

```
121     doDowngrade(dg, "timHelpParContent", HelpParContent);
122     doDowngrade(dg, "timDurationPicker", DurationPickerComponent);
123     doDowngrade(dg, "timRelevanceEdit", RelevanceEditComponent);
124     doDowngrade(dg, "timMessageListAdmin", MessageListAdminComponent);
125     doDowngrade(dg, "timMessageView", TimMessageViewComponent);
126     doDowngrade(dg, "manageReadReceipt", ManageReadReceiptComponent);
127     return dg;
128 }
129
130 const downgradedModule = createDowngradedAppModule();
131
132 if (document.location) {
133     timLogInit(document.location.search.slice(1));
134 }
135
136 const themeNameMap: Record<string, string | undefined> = {
137     lighttheme: "theme-light",
138     reunukset: "theme-borders",
139 };
140
141 function applyThemeClasses() {
142     for (const [name, _] of Object.entries(
143         genericglobals().userPrefs.css_files
144     )) {
145         const classname = themeNameMap[name];
146         if (classname) {
147             document.body.classList.add(classname);
148         }
149     }
150 }
151
152 $(async () => {
153     timLogTime("DOM ready", "main.ts");
154     insertLogDivIfEnabled();
155     applyThemeClasses();
156     const jsmodules = genericglobals().JSMODULES;
157     const moduleLoads = [];
158     for (const mname of jsmodules) {
159         const m = staticDynamicImport(mname);
160         if (!m) {
161             continue;
162         }
163         moduleLoads.push(m);
164     }
165     const angularModules: string[] = [];
166     for (const m of moduleLoads) {
167         const loaded = (await m) as {moduleDefs: unknown};
168         const mods = loaded.moduleDefs;
169         if (ModuleArray.is(mods)) {
170             angularModules.push(...mods.map((mm) => mm.name));
171         }
172     }
173     const extraAngularModules = genericglobals().ANGULARMODULES;
174     if (StringArray.is(extraAngularModules)) {
175         angularModules.push(...extraAngularModules);
176     }
177     angular.bootstrap(
178         document,
179         [timApp.name, downgradedModule.name, ...angularModules],
```

```
        {strictDi: false}
    );
    timLogTime("Angular bootstrap done", "main.ts");
    ParCompiler.processAllMathDelayed($("body"), 1500);

    // For some reason, anchor link in URL doesn't work when loading a page for the first time.
    // This is a workaround for it.
    if (location.hash && !location.hash.includes("/")) {
        try {
            const id = decodeURIComponent(location.hash).slice(1);

            // Don't use jQuery selector here because id would have to be escaped
            // and then jQuery would have to be updated to get escapeSelector method.
            const element = document.getElementById(id);
            if (element) {
                // Both with and without setTimeout are needed to get smooth experience.
                // Firefox and Chrome behave slightly differently.
                element.scrollIntoView();
                setTimeout(() => element.scrollIntoView());
            }
        } catch {
            // location.hash may still be invalid after decoding
        }
    }
});
```

timApp/static/scripts/tim/plugin/tableForm.ts

```
/**
 * Defines the client-side implementation of a plugin for editing other plugins' answers in a formatted table
 */
import angular from "angular";
import * as t from "io-ts";
import {$http, $httpParamSerializer} from "tim/util/ngimport";
import {
    clone,
    defaultErrorMessage,
    maxContentOrFitContent,
    to,
} from "tim/util/utils";
import {
    ApplicationRef,
    ChangeDetectorRef,
    Component,
    DoBootstrap,
    ElementRef,
    NgModule,
    OnInit,
    ViewChild,
} from "@angular/core";

import {TimMessageComponent} from "tim/messaging/tim-message-send.component";
import {TimUtilityModule} from "tim/ui/tim-utility.module";
import {createDowngradedModule, doDowngrade} from "tim/downgrade";
import {BrowserModule, DomSanitizer} from "@angular/platform-browser";
import {HttpClient, HttpClientModule} from "@angular/common/http";
import {FormsModule} from "@angular/forms";
```

```
30 import {platformBrowserDynamic} from "@angular/platform-browser-dynamic";
31 import {AngularPluginBase} from "tim/plugin/angular-plugin-base.directive";
32 import {vctrlInstance} from "tim/document/viewctrlinstance";
33 import {showInputDialog} from "tim/ui/showInputDialog";
34 import {InputDialogKind} from "tim/ui/input-dialog.kind";
35 import {BsDropdownModule} from "ngx-bootstrap/dropdown";
36 import {TimepickerModule} from "ngx-bootstrap/timepicker";
37 import {DatetimePickerModule} from "tim/ui/datetime-picker/datetime-picker.component";
38 import {documentglobals} from "tim/util/globals";
39 import {ViewCtrl} from "../document/viewctrl";
40 import {widenFields} from "../util/common";
41 import {
42     GenericPluginMarkup,
43     getTopLevelFields,
44     IncludeUsersOption,
45     nullable,
46     withDefault,
47 } from "./attributes";
48 import {
49     CellAttrToSave,
50     CellToSave,
51     ClearSort,
52     colnumToLetters,
53     DataEntity,
54     DataViewSettingsType,
55     isPrimitiveCell,
56     TimTable,
57     TimTableComponent,
58     TimTableModule,
59 } from "./timTable";
60
61 const RunScriptModel = t.type({
62     script: nullable(t.string),
63     button: nullable(t.string),
64     all: nullable(t.boolean),
65     update: nullable(t.boolean),
66     interval: nullable(t.number),
67 });
68
69 interface RunScriptModelType extends t.TypeOf<typeof RunScriptModel> {}
70
71 interface RunScriptType extends RunScriptModelType {
72     handle?: number;
73     running?: number;
74 }
75
76 // interface RunScriptsType extends t.TypeOf<typeof t.array(t.union([t.string, ↩
    RunScriptModel]))> {}
77
78 const TableFormMarkup = t.intersection([
79     t.partial({
80         anonNames: nullable(t.boolean),
81         autosave: t.boolean,
82         hideButtonText: nullable(t.string),
83
84         hiddenColumns: t.array(t.number),
85         hiddenRows: t.array(t.number),
86         lockedFields: t.array(t.string),
87         maxWidth: t.string,
```

```
88          minWidth: t.string,
89          maxRows: t.string,
90          filterRow: t.boolean,
91          toolbarTemplates: t.array(t.object),
92
93          cbColumn: t.boolean,
94          nrColumn: t.boolean,
95          charRow: t.boolean,
96          groups: t.array(t.string),
97          report: nullable(t.boolean),
98          reportButton: nullable(t.string),
99          separator: nullable(t.string),
100         sortBy: nullable(
101             t.string
102         ) /* TODO! Username and task, or task and username -- what about points? */,
103         table: nullable(t.boolean),
104         removeUsersButtonText: nullable(t.string),
105         userListButtonText: nullable(t.string),
106         emailUsersButtonText: nullable(t.string),
107         forceUpdateButtonText: nullable(t.string),
108         fields: t.array(t.string),
109         showToolbar: t.boolean,
110         hide: t.partial({
111             editMenu: t.boolean,
112             insertMenu: t.boolean,
113         }),
114         sisugroups: t.string,
115         // runScripts: t.array(t.union([t.string, RunScriptModel])),
116         runScripts: t.array(t.union([t.string, RunScriptModel])),
117         dataView: nullable(DataViewSettingsType),
118     }),
119     GenericPluginMarkup,
120     t.type({
121         // all withDefaults should come here; NOT in t.partial
122         autoupdate: withDefault(t.number, 500),
123         cols: withDefault(t.number, 20),
124         autoUpdateFields: withDefault(t.boolean, true),
125         autoUpdateTables: withDefault(t.boolean, true),
126         fontSize: withDefault(t.string, "smaller"),
127         fixedColor: withDefault(t.string, "#f0f0f0"),
128         includeUsers: withDefault(IncludeUsersOption, "current"),
129         saveStyles: withDefault(t.boolean, true),
130         removeDocIds: withDefault(t.boolean, true),
131         taskBorders: withDefault(t.boolean, false),
132         singleLine: withDefault(t.boolean, true),
133         usernames: withDefault(t.boolean, true),
134         realnames: withDefault(t.boolean, true),
135         emails: withDefault(t.boolean, false),
136         maxCols: withDefault(t.string, maxContentOrFitContent()),
137         openButtonText: withDefault(t.string, "Avaa Taulukko/Raporttinäkymä"),
138         open: withDefault(t.boolean, true),
139         reportFilter: withDefault(t.string, ""),
140     }),
141 ]);
142
143 const Rows = t.record(
144     t.string,
145     t.record(t.string, t.union([t.string, t.null, t.number]))
146 );
```

```
147 const Styles = t.record(
148     t.string,
149     t.record(t.string, t.union([t.null, t.record(t.string, t.string)]))
150 );
151
152 interface IRowsType extends t.TypeOf<typeof Rows> {}
153
154 interface ITableFormUser {
155     id: number;
156     real_name: string;
157     email: string;
158 }
159
160 const TableFormAll = t.intersection([
161     t.partial({
162         aliases: t.record(t.string, t.string),
163         fields: t.array(t.string),
164         users: t.record(
165             t.string,
166             t.type({id: t.number, email: t.string, real_name: t.string})
167         ),
168         membershipmap: t.record(t.string, nullable(t.string)),
169         rows: Rows,
170         styles: Styles,
171     }),
172     getTopLevelFields(TableFormMarkup),
173     t.type({}),
174 ]);
175
176 const realNameColumn = "A";
177 const userNameColumn = "B";
178 const emailColumn = "C";
179 const membershipColumn = "D";
180 const realNameColIndex = 0;
181 const userNameColIndex = 1;
182 const emailColIndex = 2;
183 const memberShipColIndex = 3;
184 const sortLang = "fi";
185
186 @Component({
187     selector: "tableform-runner",
188     // changeDetection: ChangeDetectionStrategy.OnPush,
189     template: `
190         <div class="tableform" *ngIf="showTable">
191             <tim-markup-error *ngIf="markupError" ↩
    [data]="markupError"></tim-markup-error>
192             <h4 *ngIf="header" [innerHtml]="header"></h4>
193             <p *ngIf="stem" [innerHtml]="stem"></p>
194             <tim-table *ngIf="tableCheck()" [data]="data"
195                        [taskid]="getTaskId()"></tim-table>
196
197             <div class="hidden-print">
198                 <button class="timButton"
199                         *ngIf="tableCheck() && !autosave"
200                         (click)="saveText()">
201                     {{ buttonText() }}
202                 </button>
203                 <button class="timButton"
204                         *ngIf="reportCheck()"
```

```
205                     (click)="generateReport()">
206                 {{ reportButton() }}
207             </button>
208             <button class="timButton"
209                     (click)="closeTable()"
210                     *ngIf="hideButtonText">
211                 {{hideButtonText}}
212             </button>
213             <button class="timButton"
214                     (click)="forceUpdateTable()"
215                     *ngIf="forceUpdateButtonText">
216                 {{forceUpdateButtonText}}
217             </button>
218             <button class="timButton"
219                     (click)="removeUsers()"
220                     *ngIf="removeUsersButtonText && cbCount">
221                 {{removeUsersButtonText}}
222             </button>
223             <button class="timButton"
224                     (click)="listUsernames()"
225                     *ngIf="userListButtonText && cbCount">
226                 {{userListButtonText}}
227             </button>
228             <button class="timButton"
229                     (click)="emailUsers()"
230                     *ngIf="emailUsersButtonText && cbCount">
231                 {{emailUsersButtonText}}
232             </button>
233             <ng-container *ngIf="runScripts">
234                 <button class="timButton"
235                         *ngFor="let s of runScripts"
236                         [hidden]="!s.all && !cbCount"
237                         (click)="runJsRunner(s)">
238                     {{s.button}}
239                 </button>
240             </ng-container>
241             <button class="timButton"
242                     (click)="orderSisuGroups()"
243                     *ngIf="sisugroups && cbCount">
244                 Confirm groups
245             </button>
246         </div>
247         <div class="csRunDiv tableUsers" style="padding: 1em;" *ngIf="userlist"> ↩
    <!-- userlist -->
248             <tim-close-button (click)="userlist=''"></tim-close-button>
249             <p>Separator:
250                 <label><input type="radio" name="listsep" [(ngModel)]="listSep" ↩
    value="-"
251                               (change)="listUsernames()">-</label> 
252                 <label><input type="radio" name="listsep" [(ngModel)]="listSep" ↩
    value=","
253                               (change)="listUsernames()">,</label> 
254                 <label><input type="radio" name="listsep" [(ngModel)]="listSep" ↩
    value="|"
255                               (change)="listUsernames()">|</label> 
256                 <label><input type="radio" name="listsep" [(ngModel)]="listSep" ↩
    value=";"
257                               (change)="listUsernames()">;</label> 
258                 <label><input type="radio" name="listsep" [(ngModel)]="listSep" ↩
```

```
    value="\n"
                                    (change)="listUsernames()">\\n</label> 
                </p>
                <label><input type="checkbox" [(ngModel)]="listName" ↩
    (change)="listUsernames()">Name</label> 
                <label><input type="checkbox" [(ngModel)]="listUsername"
                              (change)="listUsernames()">Username</label> 
                <label><input type="checkbox" [(ngModel)]="listEmail" ↩
    (change)="listUsernames()">Email</label> 
                <br>
                <textarea id="userlist" [(ngModel)]="userlist" rows="10" ↩
    cols="60"></textarea>
                <button class="timButton"
                        (click)="copyList()">
                    Copy
                </button>
            </div>
            <tim-message-send [recipientList]="recipientList" ↩
    [docId]="currentDocumentID()"></tim-message-send>
            <pre *ngIf="result">{{result}}</pre>
            <pre *ngIf="error" [innerHtml]="error"></pre>
            <p *ngIf="footer" [innerText]="footer" class="plgfooter"></p>
        </div>
        <div class="tableOpener" *ngIf="!showTable">
            <button class="timButton"
                    [disabled]="loading"
                    (click)="openTable()">
                {{openButtonText}}
            </button>
            <tim-loading *ngIf="loading"></tim-loading>
        </div>
    `,
    styleUrls: ["./tableForm.scss"],
})
export class TableFormComponent
    extends AngularPluginBase<
        t.TypeOf<typeof TableFormMarkup>,
        t.TypeOf<typeof TableFormAll>,
        typeof TableFormAll
    >
    implements OnInit {
    public viewctrl?: ViewCtrl;
    result?: string;
    error?: string;
    private userfilter = "";
    data: TimTable & {userdata: DataEntity} = {
        hide: {edit: false, insertMenu: true, editMenu: true},
        hiddenRows: [],
        hiddenColumns: [],
        hideSaveButton: true,
        // lockCellCount: true,
        lockedCells: [],
        lockedColumns: [],
        table: {countRow: 0, countCol: 0, columns: []},
        // TODO: give rows (and maybe colums) in data.table
        task: true,
        userdata: {type: "Relative", cells: {}},
        nonUserSpecific: true,
        isPreview: false,
```

```
313     };
314     // TODO: Change row format to properly typed format (maybe userobject:IRowstype) ↩
        format
315     private rows!: IRowsType;
316     private styles?: t.TypeOf<typeof Styles>;
317     private fields!: string[];
318     private lockedFields!: string[];
319     private users!: Record<string, ITableFormUser>;
320     private membershipmap!: Record<string, string | null>;
321     private aliases!: Record<string, string>;
322     private realnames = false;
323     private usernames = false;
324     private emails = false;
325     showTable = false;
326     private tableFetched = false;
327     private rowKeys!: string[];
328     private userLocations: Record<string, string> = {};
329     private taskLocations: Record<string, string> = {};
330     private changedCells: string[] = []; // Use same type as data.userdata?
331     private clearStylesCells = new Set<string>();
332
333     runScripts?: RunScriptType[];
334
335     userlist: string = "";
336     listSep: string = "-";
337     listName: boolean = false;
338     listUsername: boolean = true;
339     listEmail: boolean = false;
340     private fixedColor: string = "#f0f0f0";
341     cbCount: number = 0;
342     @ViewChild(TimTableComponent)
343     timTable?: TimTableComponent;
344     recipientList = "";
345     loading = false;
346
347     currentDocumentID() {
348         return documentglobals().curr_item.id;
349     }
350
351     getDefaultMarkup() {
352         return {};
353     }
354
355     get autosave() {
356         return this.markup.autosave;
357     }
358
359     get hideButtonText() {
360         return this.markup.hideButtonText;
361     }
362
363     get openButtonText() {
364         return this.markup.openButtonText;
365     }
366
367     get removeUsersButtonText() {
368         return this.markup.removeUsersButtonText;
369     }
370
```

```
371     get userListButtonText() {
372         return this.markup.userListButtonText;
373     }
374
375     get emailUsersButtonText() {
376         return this.markup.emailUsersButtonText;
377     }
378
379     get forceUpdateButtonText() {
380         return this.markup.forceUpdateButtonText;
381     }
382
383     get xrunScripts() {
384         return this.markup.runScripts;
385     }
386
387     get sisugroups() {
388         return this.markup.sisugroups;
389     }
390
391     /**
392      * Used to define table view & relative save button in angular, true or false.
393      */
394     buttonText() {
395         return this.markup.buttonText ?? "Tallenna taulukko";
396     }
397
398     /**
399      * Used to define table view & relative save button in angular, true or false.
400      */
401     reportButton() {
402         return this.markup.reportButton ?? "Luo Raportti";
403     }
404
405     addHiddenIndex(i: number) {
406         if (!this.data.hiddenColumns) {
407             this.data.hiddenColumns = [i];
408         } else {
409             this.data.hiddenColumns.push(i);
410         }
411     }
412
413     checkToShow(param: boolean | undefined, i: number, def: boolean): boolean {
414         if (param == undefined) {
415             param = def;
416         }
417         if (param) {
418             return true;
419         }
420
421         this.addHiddenIndex(i);
422         return false;
423     }
424
425     // noinspection JSUnusedLocalSymbols
426     constructor(
427         el: ElementRef,
428         http: HttpClient,
429         domSanitizer: DomSanitizer,
```

```
        private cdr: ChangeDetectorRef
    ) {
        super(el, http, domSanitizer);
        // cdr.detach();
    }

    parseRunScripts(
        scripts: (RunScriptModelType | string)[],
        oldScripts: RunScriptType[] | undefined
    ) {
        if (oldScripts) {
            for (const sr of oldScripts) {
                if (sr.handle) {
                    window.clearInterval(sr.handle);
                }
            }
        }
        const runScripts = [];

        for (const r of scripts) {
            let s: string | undefined | null = "";
            const rs: RunScriptType = {
                script: "",
                button: null,
                all: null,
                update: null,
                interval: null,
                handle: undefined,
                running: 0,
            };
            if (typeof r === "string") {
                if (r.length == 0) {
                    continue;
                }
                s = r;
            } else {
                s = r.script;
                rs.button = r.button;
                rs.all = r.all;
                rs.update = r.update;
                rs.interval = r.interval;
            }
            let script = "";
            if (s) {
                const parts = s.split("=");
                script = parts[0].replace("!", "").replace("*", "");
                rs.script = script;
                rs.button = rs.button ?? (parts.length > 1 ? parts[1] : script);
                rs.all = rs.all ?? s.startsWith("*"); // compatible with old format: ↩
    "*name!=buttontext"
                rs.update = rs.update ?? s.includes("!");
            }
            if (script || rs.interval) {
                runScripts.push(rs);
                if (rs.interval && rs.interval > 0) {
                    // if (rs.interval < 10) rs.interval = 10; // at least 5 sec
                    rs.handle = window.setInterval(() => {
                        if (rs.running) {
                            return;
```

```
488                         }
489                         rs.running = 1;
490                         const tt:
491                             | TimTableComponent
492                             | undefined = this.getTimTable();
493                         if (!tt) {
494                             return;
495                         }
496                         if (tt.isPreview() || !this.showTable) {
497                             return;
498                         }
499                         this.runJsRunner(rs);
500                         rs.running = 0;
501                     }, rs.interval * 1000);
502                 }
503             }
504         }
505         return runScripts;
506     }
507
508     ngOnInit() {
509         super.ngOnInit();
510
511         if (this.markup.runScripts) {
512             this.runScripts = this.parseRunScripts(
513                 this.markup.runScripts,
514                 this.runScripts
515             );
516         }
517
518         const tid = this.getTaskId();
519         this.viewctrl = vctrlInstance; // TODO: Make an Angular service for getting ↩
    ViewCtrl.
520         if (this.viewctrl && tid) {
521             this.viewctrl.addTableForm(this, tid.docTask());
522         }
523         const table = this.data.table;
524         if (this.markup.fontSize) {
525             table.fontSize = this.markup.fontSize;
526         }
527         this.data.taskBorders = this.markup.taskBorders;
528         this.fixedColor = this.markup.fixedColor || this.fixedColor;
529
530         this.data.hiddenRows = this.markup.hiddenRows;
531         this.data.hiddenColumns = this.markup.hiddenColumns;
532
533         // Initialize hide-attribute
534         this.data.hide = {editMenu: true, insertMenu: true};
535         if (this.markup.hide) {
536             this.data.hide = this.markup.hide; // TODO: TimTablen oletukset tähän
537             const hide = this.markup.hide;
538             if (hide.editMenu === undefined) {
539                 this.data.hide.editMenu = true;
540             }
541             if (hide.insertMenu === undefined) {
542                 this.data.hide.insertMenu = true;
543             }
544         }
545         if (this.markup.showToolbar !== undefined) {
```

```
546             this.data.hide.toolbar = this.markup.showToolbar;
547         }
548 
549         this.userfilter = "";
550         this.realnames = this.checkToShow(
551             this.markup.realnames,
552             realNameColIndex,
553             true
554         );
555         this.usernames = this.checkToShow(
556             this.markup.usernames,
557             userNameColIndex,
558             true
559         );
560         this.emails = this.checkToShow(
561             this.markup.emails,
562             emailColIndex,
563             false
564         );
565         this.checkToShow(
566             this.markup.includeUsers !== "current",
567             memberShipColIndex,
568             false
569         );
570 
571         this.rows = this.attrsall.rows ?? {};
572         this.rowKeys = Object.keys(this.rows);
573         this.styles = this.attrsall.styles;
574         this.fields = this.attrsall.fields ?? [];
575         this.lockedFields = this.markup.lockedFields ?? [];
576         this.users = this.attrsall.users ?? {};
577         this.membershipmap = this.attrsall.membershipmap ?? {};
578         this.aliases = this.attrsall.aliases ?? {};
579 
580         this.setDataMatrix();
581 
582         this.data.saveCallBack = (cellsTosave) => this.cellChanged(cellsTosave);
583         if (this.markup.saveStyles) {
584             this.data.saveStyleCallBack = (cellsTosave) =>
585                 this.cellChanged(cellsTosave);
586         }
587         this.data.cbCallBack = (cbs, n, index) => this.cbChanged(cbs, n, index);
588 
589         if (this.markup.minWidth) {
590             this.data.minWidth = this.markup.minWidth;
591         }
592         if (this.markup.maxWidth !== undefined) {
593             this.data.maxWidth = this.markup.maxWidth;
594         }
595         if (this.markup.singleLine) {
596             this.data.singleLine = this.markup.singleLine;
597         }
598         if (this.markup.open) {
599             this.tableFetched = true;
600             this.showTable = this.markup.open;
601         }
602 
603         this.data.cbColumn = this.markup.cbColumn;
604         this.data.nrColumn = this.markup.nrColumn;
```

```
605         this.data.charRow = this.markup.charRow;
606         this.data.filterRow = this.markup.filterRow;
607         this.data.maxRows = this.markup.maxRows;
608         this.data.maxCols = this.markup.maxCols;
609         this.data.toolbarTemplates = this.markup.toolbarTemplates;
610         this.data.dataView = this.markup.dataView;
611         this.data.isPreview = this.isPreview();
612         // this.cdr.detectChanges();
613     }
614
615     /**
616      * Returns the TimTableComponent within the tableForm.
617      */
618     getTimTable() {
619         return this.timTable;
620     }
621
622     /**
623      * Sorts row key values (usernames) by their real name attribute in this.users
624      * @param a username to compare with b
625      * @param b username to compare with a
626      */
627     sortByRealName(a: string, b: string) {
628         if (!this.users) {
629             return 0;
630         }
631         try {
632             return this.users[a].real_name.localeCompare(
633                 this.users[b].real_name,
634                 sortLang
635             );
636         } catch (e) {
637             return 0;
638         }
639     }
640
641     sortByEmail(a: string, b: string) {
642         try {
643             return this.users[a].email.localeCompare(
644                 this.users[b].email,
645                 sortLang
646             );
647         } catch (e) {
648             return 0;
649         }
650     }
651
652     /**
653      * Clears tableForm rows and fetches new data to be put into rows
654      * Basically just a reset
655      */
656     public async updateTable() {
657         if (this.attrsall.markup.sisugroups) {
658             return;
659         }
660
661         // TODO: Save before reset?
662         interface TableFetchResponse {
663             aliases: Record<string, string>;
```

```
664             fields: string[];
665             users: Record<string, ITableFormUser>;
666             membershipmap: Record<string, string>;
667             rows: IRowsType;
668             styles: t.TypeOf<typeof Styles>;
669         }
670
671         let prom;
672         const tid = this.getTaskId();
673         if (!tid) {
674             this.error = "TaskId is missing.";
675             return;
676         }
677         this.loading = true;
678         if (this.isPreview()) {
679             prom = $http.get<TableFetchResponse>(
680                 "/tableForm/fetchTableDataPreview?" +
681                     $httpParamSerializer({
682                         taskid: tid.docTask(),
683                         fields: this.markup.fields,
684                         groups: this.markup.groups,
685                         removeDocIds: this.markup.removeDocIds,
686                     })
687             );
688         } else {
689             prom = $http.get<TableFetchResponse>(
690                 "/tableForm/fetchTableData?" +
691                     $httpParamSerializer({
692                         taskid: tid.docTask(),
693                     })
694             );
695         }
696         const r = await to(prom);
697         this.loading = false;
698         if (!r.ok) {
699             this.error = r.result.data.error;
700         } else {
701             const tableResponse = r.result;
702             // TODO: Generic reset function
703             this.aliases = tableResponse.data.aliases || {};
704             this.membershipmap = tableResponse.data.membershipmap;
705             this.rows = tableResponse.data.rows || {};
706             this.rowKeys = Object.keys(tableResponse.data.rows);
707             this.fields = tableResponse.data.fields || [];
708             this.users = tableResponse.data.users;
709             this.styles = tableResponse.data.styles || {};
710             this.userLocations = {};
711             this.taskLocations = {};
712             this.data.table.countCol = 0;
713             this.data.table.countRow = 0;
714             this.data.table.columns = [];
715             this.data.userdata.cells = {};
716             this.setDataMatrix();
717             this.reinitializeTimTable();
718         }
719     }
720
721     private reinitializeTimTable() {
722         const timtab = this.getTimTable();
```

```
723         if (timtab) {
724             timtab.reInitialize(ClearSort.No);
725             timtab.c();
726         }
727     }
728 
729     /**
730      * Queries new values for given fields and updates the table
731      * @param fields to be updated
732      */
733     public async updateFields(fields: string[]) {
734         if (this.attrsall.markup.sisugroups) {
735             return;
736         }
737 
738         try {
739             if (!this.tableFetched || !this.viewctrl) {
740                 return;
741             }
742             const fieldsToUpdate: string[] = [];
743             if (!this.markup.fields) {
744                 return;
745             }
746             const ownFields = widenFields(this.markup.fields);
747             for (const aliasfield of ownFields) {
748                 const field = aliasfield.split("=")[0].trim();
749                 const docField = this.viewctrl.docId + "." + field;
750                 // TODO: Double .includes call - maybe it's better to search for ↩
    fieldsToUpdate from somethign
751                 //  that already has the docID
752                 if (fields.includes(field) || fields.includes(docField)) {
753                     fieldsToUpdate.push(aliasfield);
754                 }
755             }
756             const tid = this.getTaskId();
757             if (!tid) {
758                 return;
759             }
760             const r = await to(
761                 $http.get<{
762                     rows: IRowsType;
763                     styles: t.TypeOf<typeof Styles>;
764                     fields: string[];
765                 }>(
766                     "/tableForm/updateFields?" +
767                         $httpParamSerializer({
768                             fields: fieldsToUpdate,
769                             taskid: tid.docTask(),
770                         })
771                 )
772             );
773             if (!r.ok) {
774                 return;
775             }
776             const tableResponse = r.result;
777             // TODO if response status != ok
778             const rows = tableResponse.data.rows || {};
779             const styles = tableResponse.data.styles || {};
780             const tableFields = tableResponse.data.fields || [];
```

```
781
782             // Find out which columns to update
783             const taskColumns: Record<string, string> = {};
784             for (const f of tableFields) {
785                 const extendedField = this.aliases[f] || f;
786                 for (const [key, value] of Object.entries(this.taskLocations)) {
787                     if (value == extendedField) {
788                         taskColumns[f] = key;
789                         break;
790                     }
791                 }
792             }
793
794             // TODO: Check if any value changed.  If not do not call reInitialize
795             for (const f of tableFields) {
796                 for (let y = 0; y < this.rowKeys.length; y++) {
797                     if (
798                         styles &&
799                         !angular.equals(styles, {}) &&
800                         styles[this.rowKeys[y]]
801                     ) {
802                         this.data.userdata.cells[taskColumns[f] + (y + 1)] = {
803                             cell: rows[this.rowKeys[y]][f],
804                             ...styles[this.rowKeys[y]][f],
805                         };
806                     } else {
807                         if (rows[this.rowKeys[y]]) {
808                             this.data.userdata.cells[
809                                 taskColumns[f] + (y + 1)
810                             ] = {cell: rows[this.rowKeys[y]][f]};
811                         }
812                     }
813                 }
814             }
815             this.reinitializeTimTable();
816         } catch (e) {
817             console.log(e);
818             this.error = "Error updating fields" + "\n" + e;
819         }
820     }
821
822     /**
823      * Transforms user/task combination defined in this.rows into cell format and ↩
        sets up the table
824      * TODO: generate rows/columns for this.data.table, possibly needed for more ↩
        easily maintained layout handling
825      */
826     setDataMatrix() {
827         try {
828             if (!this.data.lockedCells) {
829                 this.data.lockedCells = [];
830             }
831             if (!this.data.lockedColumns) {
832                 this.data.lockedColumns = [];
833             }
834             if (this.realnames) {
835                 this.rowKeys.sort((a, b) => this.sortByRealName(a, b));
836                 this.data.lockedColumns.push(realNameColumn);
837             } else if (this.usernames) {
```

```
838             } else {
839                 this.rowKeys.sort((a, b) => this.sortByEmail(a, b));
840             }
841             this.data.lockedColumns.push(userNameColumn);
842             this.data.lockedColumns.push(emailColumn);
843             // }
844             if (this.attrsall.markup.sisugroups) {
845                 // These require unique names, otherwise could just use empty strings ↩
    in place of "invisibleX".
846                 this.data.headers = [
847                     "Kuvaus",
848                     "Sisu-nimi",
849                     "invisible1",
850                     "invisible2",
851                 ];
852             } else {
853                 this.data.headers = [
854                     "Henkilön nimi",
855                     "Käyttäjänimi",
856                     "eMail",
857                     "Poistunut?",
858                 ];
859             }
860             this.data.headersStyle = {
861                 backgroundColor: this.fixedColor,
862                 "font-weight": "bold",
863             };
864 
865             if (this.fields) {
866                 this.data.table.countCol = this.fields.length + 3;
867             }
868             this.data.table.countRow = Object.keys(this.rows).length;
869             let y = 1;
870             for (const r of this.rowKeys) {
871                 this.data.userdata.cells[userNameColumn + y] = {
872                     cell: r,
873                     backgroundColor: this.fixedColor,
874                 };
875                 this.userLocations[y] = r;
876                 const userInfo = this.users[r];
877                 this.data.userdata.cells[realNameColumn + y] = {
878                     cell: userInfo.real_name,
879                     backgroundColor: this.fixedColor,
880                 };
881                 this.data.userdata.cells[emailColumn + y] = {
882                     cell: userInfo.email,
883                     backgroundColor: this.fixedColor,
884                 };
885                 for (const [map, col] of [
886                     [this.membershipmap, membershipColumn],
887                 ] as const) {
888                     if (map) {
889                         this.data.userdata.cells[col + y] = {
890                             cell: map[r],
891                             backgroundColor: this.fixedColor,
892                         };
893                     }
894                 }
895                 y++;
```

```
896             }
897             // TODO: Load default cell colors from tableForm's private answer?
898             const xOffset = memberShipColIndex + 1;
899             if (this.fields) {
900                 for (let x = 0; x < this.fields.length; x++) {
901                     const colheader = this.fields[x];
902                     const currentCol = colnumToLetters(x + xOffset);
903                     const expandedLockedFields = widenFields(this.lockedFields);
904                     if (expandedLockedFields.includes(colheader)) {
905                         this.data.lockedColumns.push(currentCol);
906                     }
907                     this.data.headers.push(colheader);
908                     /*
909                     this.data.userdata.cells[colnumToLetters(x + xOffset) + 1] = {
910                         cell: colheader,
911                         backgroundColor: this.fixedColor,
912                     };
913                     */
914 
915                     let contentalias;
916                     if (this.aliases && colheader in this.aliases) {
917                         contentalias = this.aliases[colheader];
918                     } else {
919                         contentalias = colheader;
920                     }
921                     this.taskLocations[
922                         colnumToLetters(x + xOffset)
923                     ] = contentalias;
924                     // this.data.lockedCells.push(colnumToLetters(x + xOffset) + 1);
925                     // y = 0;
926                     // for (const [u, r] of Object.entries(this.rows)) {
927                     //     if (r[this.attrsall.fields[x]]) {
928                     //         this.data.userdata.cells[colnumToLetters(x + xOffset) ↩
    + (y + 2)] = r[this.attrsall.fields[x]];
929                     //     }
930                     //     y++;
931                     // }
932                     for (y = 0; y < this.rowKeys.length; y++) {
933                         // this.data.userdata.cells[colnumToLetters(x + xOffset) + (y ↩
    + 1)] = this.rows[this.rowKeys[y]][this.attrsall.fields[x]];
934                         if (this.styles && !angular.equals(this.styles, {})) {
935                             this.data.userdata.cells[currentCol + (y + 1)] = {
936                                 cell: this.rows[this.rowKeys[y]][
937                                     this.fields[x]
938                                 ],
939                                 ...this.styles[this.rowKeys[y]][this.fields[x]],
940                             };
941                         } else {
942                             this.data.userdata.cells[currentCol + (y + 1)] = {
943                                 cell: this.rows[this.rowKeys[y]][
944                                     this.fields[x]
945                                 ],
946                             };
947                         }
948                     }
949                 }
950             }
951         } catch (e) {
952             console.log(e);
```

```
953             this.error = "Error in setDataMatrix" + "\n" + e;
954         }
955     }
956
957     /**
958      * Closes timTable's editor and saves the cell that is being currently edited
959      */
960     async saveText() {
961         const timTable = this.getTimTable();
962         if (timTable == null) {
963             return;
964         }
965         await timTable.saveAndCloseSmallEditor();
966         this.doSaveText();
967     }
968
969     /**
970      * Returns true value, if table attribute is true.
971      * Used to define table view & relative save button in angular, true or false.
972      */
973     tableCheck() {
974         // return (this.markup.table === true);
975         if (this.markup.table != undefined) {
976             return this.markup.table;
977         } else {
978             return true;
979         }
980     }
981
982     /**
983      * Returns true value, if report attribute is true.
984      * Used to define create report button in angular, true or false.
985      */
986     reportCheck() {
987         return this.markup.report == true;
988     }
989
990     /**
991      * String to determinate how usernames are filtered in report.
992      * Choises are username, username and full name and anonymous. Username as default.
993      */
994     sortBy() {
995         return this.markup.sortBy ?? "username";
996     }
997
998     /**
999      * Generates report based on the table.
1000      * Used if report is set to true and create report button is clicked.
1001      * Used to define table view & relative save button in angular, true or false.
1002      */
1003     generateReport() {
1004         // const dataTable = this.generateCSVTable();
1005         const taskId = this.pluginMeta.getTaskId();
1006         if (taskId == undefined) {
1007             return;
1008         }
1009         const timTable = this.getTimTable();
1010         if (timTable == null) {
1011             return;
```

```
        }

        const reportParams = {
            // TODO: support for relevant attrs (realnames, usernames, emails...)
            // TODO: get relevant user input from timTable (sort, filters, checkboxes...)
            // taskid: this.getTaskId()
            // TODO: use taskid? (less data to transfer because of plug.values, but dependant on task existence)
            docId: taskId.docId,
            fields: this.markup.fields,
            groups: this.markup.groups,
            removeDocIds: this.markup.removeDocIds,
            separator: this.markup.separator ?? ",",
            anonNames: this.markup.anonNames,
            realnames: this.markup.realnames,
            usernames: this.markup.usernames,
            emails: this.markup.emails,
            reportFilter: this.markup.reportFilter,
        };
        let filterParams;
        const selUsers = timTable.getCheckedRows(0, false);
        const users = TableFormComponent.makeUserArray(
            selUsers,
            userNameColIndex
        );

        if (selUsers.length > 0) {
            filterParams = {userFilter: users};
        } else {
            const filterFields: string[] = [];
            const filterValues: string[] = [];

            const xOffset = memberShipColIndex + 1;

            timTable.filters.forEach((value, index) => {
                if (!value) {
                    return;
                }
                switch (index) {
                    case realNameColIndex:
                        filterFields.push("realname");
                        break;
                    case userNameColIndex:
                        filterFields.push("username");
                        break;
                    case emailColIndex:
                        filterFields.push("email");
                        break;
                    case memberShipColIndex:
                        filterFields.push("membership");
                        break;
                    default:
                        filterFields.push(this.fields[index - xOffset]);
                }
                filterValues.push(value);
            });
            filterParams = {filterFields, filterValues};
        }
```

```
1069
1070        const win = window.open(
1071            "/tableForm/generateCSV?" +
1072                $httpParamSerializer({
1073                    ...reportParams,
1074                    ...filterParams,
1075                }),
1076            "WINDOWID"
1077        );
1078        if (win == null) {
1079            this.error = "Failed to open report window.";
1080        }
1081    }
1082
1083    /**
1084     * Make list of users colIndex.  Separate items by separators
1085     * @param users array of users
1086     * @param colIndex what index to use for list
1087     * @param preseparator what comes before evyry item
1088     * @param midseparator what comes between items
1089     */
1090    static makeUserList(
1091        users: string[][],
1092        colIndex: number,
1093        preseparator: string,
1094        midseparator: string
1095    ): string {
1096        let result = "";
1097        let sep = "";
1098        for (const r of users) {
1099            result += sep + preseparator + r[colIndex];
1100            sep = midseparator;
1101        }
1102        return result;
1103    }
1104
1105    static makeUserArray(users: string[][], colIndex: number): string[] {
1106        const result = [];
1107        for (const r of users) {
1108            result.push(r[colIndex]);
1109        }
1110        return result;
1111    }
1112
1113    /**
1114     * Removes selected users from the group
1115     */
1116    async removeUsers() {
1117        const timTable = this.getTimTable();
1118        if (timTable == null) {
1119            return;
1120        }
1121        const selUsers = timTable.getCheckedRows(0, true);
1122        let msg = "";
1123        for (const r of selUsers) {
1124            msg += r.join(", ") + "<br>";
1125        }
1126        if (msg == "") {
1127            return;
```

```
        }

        if (!this.markup.groups) {
            return;
        }
        const group = this.markup.groups[0];

        await showInputDialog({
            text:
                "<b>Really remove the following users from group:</b> " +
                group +
                "<br>\n<pre>\n" +
                msg +
                "\n</pre>",
            title: "Remove users from group " + group,
            isInput: InputDialogKind.ValidatorOnly,
            validator: async () => {
                const ulist = TableFormComponent.makeUserList(
                    selUsers,
                    1,
                    "",
                    ","
                );
                const r = await to(
                    $http.post<unknown>(`/groups/removemember/${group}`, {
                        names: ulist.split(","),
                    })
                );
                if (r.ok) {
                    return {ok: true, result: r.result.data} as const;
                } else {
                    return {ok: false, result: r.result.data.error} as const;
                }
            },
        });
        location.reload();
    }

    listUsernames() {
        const timTable = this.getTimTable();
        if (timTable == null) {
            return;
        }
        let preseparator = " - ";
        let midseparator = "\n";
        let sep = this.listSep;
        const colindex = 0;
        const selUsers = timTable.getCheckedRows(0, true);
        const ulist = [];
        let usep = "";
        for (const u of selUsers) {
            const un = u[userNameColIndex];
            let s = "";
            if (this.listName) {
                s = this.users[un].real_name;
                usep = ", ";
            }
            if (this.listUsername) {
                s += usep + un;
```

```
1187                usep = ", ";
1188            }
1189            if (this.listEmail) {
1190                s += usep + this.users[un].email;
1191                usep = ", ";
1192            }
1193            usep = "";
1194            ulist.push([s]);
1195        }
1196        // if ( this.listEmail ) { midseparator = "\n"; preseparator = "";  }
1197        if (sep == "") {
1198            sep = "\n";
1199        } // radio could not give \n?
1200        if (sep != "-") {
1201            midseparator = sep;
1202            preseparator = "";
1203        }
1204        this.userlist = TableFormComponent.makeUserList(
1205            ulist,
1206            colindex,
1207            preseparator,
1208            midseparator
1209        );
1210    }
1211
1212    copyList() {
1213        const ta = this.element.find("#userlist");
1214        ta.focus();
1215        ta.select();
1216        document.execCommand("copy");
1217        // TODO: myös iPad toimimaan, ks GeoGebra tai csPlugin jaa tee yleinen copy
1218    }
1219
1220    emailUsers() {
1221        const timTable = this.getTimTable();
1222        if (timTable == null) {
1223            return;
1224        }
1225        const selUsers = timTable.getCheckedRows(0, true);
1226        this.recipientList = TableFormComponent.makeUserList(
1227            selUsers,
1228            emailColIndex,
1229            "",
1230            "\n"
1231        );
1232    }
1233
1234    /**
1235     * Callback function to be noticed when check boxes are changed in table
1236     * @param cbs boolean list of cb-values
1237     * @param n number of visible checked cbs
1238     * @param index index of clicked cb, may be -1 if header row cb clicked
1239     */
1240    cbChanged(cbs: boolean[], n: number, index: number) {
1241        this.cbCount = n;
1242    }
1243
1244    /**
1245     * Callback function that gets called when timTable saves a cell
```

```
1246      * Collects information about which cells have changed and which ones want to ↩
      clear their style attributes
1247      * @param cellsToSave list of cells that needs to be saved
1248      */
1249     async cellChanged(cellsToSave: CellToSave[] | CellAttrToSave[]) {
1250         if (this.attrsall.markup.sisugroups) {
1251             return;
1252         }
1253         for (const c of cellsToSave) {
1254             const coli = c.col;
1255             const rowi = c.row;
1256             const changedStyle = c.key;
1257             if (changedStyle) {
1258                 if (changedStyle == "CLEAR") {
1259                     this.clearStylesCells.add(
1260                         colnumToLetters(coli) + (rowi + 1)
1261                     );
1262                 } else {
1263                     this.clearStylesCells.delete(
1264                         colnumToLetters(coli) + (rowi + 1)
1265                     );
1266                 }
1267             }
1268             this.changedCells.push(colnumToLetters(coli) + (rowi + 1));
1269         }
1270         if (this.markup.autosave) {
1271             await this.doSaveText();
1272         }
1273     }
1274
1275     async openTable() {
1276         if (!this.tableFetched) {
1277             await this.updateTable();
1278             this.tableFetched = true;
1279         }
1280         this.showTable = true;
1281     }
1282
1283     async forceUpdateTable() {
1284         this.timTable?.dataViewComponent?.startReset();
1285         this.timTable?.ngOnInit();
1286         this.ngOnInit();
1287         await this.updateTable();
1288         this.timTable?.clearSortOrder();
1289         this.timTable?.repeatLastSort();
1290         this.timTable?.handleChangeFilter();
1291         this.timTable?.c();
1292         this.tableFetched = true;
1293         this.timTable?.dataViewComponent?.endReset();
1294     }
1295
1296     closeTable() {
1297         this.showTable = false;
1298     }
1299
1300     /**
1301      * Transforms the cell format back to row format and saves the table input
1302      */
1303     async doSaveText() {
```

```
        // this.error = "... saving ...";
        if (this.changedCells.length == 0) {
            return;
        }
        const replyRows: Record<
            number,
            Record<string, string | null | Record<string, unknown>>
        > = {};
        const changedFields = new Set<string>();
        const changedFieldsForTables = new Set<string>();
        try {
            for (const coord of this.changedCells) {
                const alphaRegExp = new RegExp("([A-Z]*)");
                const alpha = alphaRegExp.exec(coord);
                if (alpha == null) {
                    continue;
                }
                const columnPlace = alpha[0];
                const numberPlace = coord.substring(columnPlace.length);
                if (
                    columnPlace === userNameColumn ||
                    columnPlace === realNameColumn || // TODO: Do we need this anymore?
                    columnPlace === emailColumn
                ) {
                    // TODO: Do we need this anymore?
                    continue;
                }
                const cell = this.data.userdata.cells[coord];
                let cellContent;
                let cellStyle: Record<string, unknown> | null = null;
                if (!isPrimitiveCell(cell)) {
                    cellContent = cell.cell;
                    if (this.markup.saveStyles) {
                        cellStyle = clone(cell);
                        delete cellStyle.cell;
                    }
                } else {
                    cellContent = cell;
                }
                if (cellContent === null) {
                    cellContent = "";
                } else if (
                    typeof cellContent === "boolean" ||
                    typeof cellContent === "number"
                ) {
                    cellContent = cellContent.toString();
                }
                // else if (typeof cellContent === "boolean") {
                //     throw new Error("cell was boolean?");

                if (
                    (this.markup.autoUpdateFields ||
                        this.markup.autoUpdateTables) &&
                    this.viewctrl
                ) {
                    const taskWithField = this.taskLocations[columnPlace].split(
                        "."
                    );
                    const docTask = taskWithField[0] + "." + taskWithField[1];
```

```
                    if (
                        this.viewctrl.selectedUser.name ==
                        this.userLocations[numberPlace]
                    ) {
                        // TODO: Should check for global / useCurrentUser fields here
                        changedFields.add(docTask);
                    }
                    changedFieldsForTables.add(docTask);
                }
                const userId = this.users[this.userLocations[numberPlace]].id;
                try {
                    replyRows[userId][
                        this.taskLocations[columnPlace]
                    ] = cellContent;
                } catch (e) {
                    replyRows[userId] = {};
                    replyRows[userId][
                        this.taskLocations[columnPlace]
                    ] = cellContent;
                }
                /* TODO: instead of iterating clearStylesCells could decide that ↩
    absence of any styles
                    (e.g primitivecell) would mean result in null style value being sent
                */
                if (this.clearStylesCells.has(columnPlace + numberPlace)) {
                    const taskWithField = this.taskLocations[columnPlace].split(
                        "."
                    );
                    const docTaskStyles =
                        taskWithField[0] + "." + taskWithField[1] + ".styles";
                    replyRows[userId][docTaskStyles] = null;
                } else if (
                    cellStyle != null &&
                    Object.keys(cellStyle).length != 0
                ) {
                    const taskWithField = this.taskLocations[columnPlace].split(
                        "."
                    );
                    const docTaskStyles =
                        taskWithField[0] + "." + taskWithField[1] + ".styles";
                    replyRows[userId][docTaskStyles] = cellStyle;
                }
            }
        } catch (e) {
            console.log(e);
            this.error = "Error in doSaveText" + "\n" + e;
        }
        const params = {
            input: {
                nosave: false,
                replyRows: replyRows,
            },
        };
        const url = this.pluginMeta.getAnswerUrl();
        const r = await to(
            $http.put<{web: {result: string; error?: string}}>(url, params)
        );
        this.loading = false;
        if (r.ok) {
```

```
            const data = r.result.data;
            this.error = data.web.error;
            // this.result = "Saved";
        } else {
            this.error = r.result.data.error ?? defaultErrorMessage;
        }
        const timtab = this.getTimTable();
        if (!timtab) {
            return;
        }
        timtab.confirmSaved();
        if (this.viewctrl) {
            if (this.markup.autoUpdateFields && changedFields.size > 0) {
                this.viewctrl.updateFields(Array.from(changedFields));
            }
            if (
                this.markup.autoUpdateTables &&
                changedFieldsForTables.size > 0
            ) {
                this.viewctrl.updateAllTables(
                    Array.from(changedFieldsForTables),
                    this.getTaskId()
                );
            }
        }
        this.clearStylesCells.clear();
        this.changedCells = [];
    }

    async orderSisuGroups() {
        const timTable = this.getTimTable();
        if (timTable == null || this.data.headers == null) {
            return;
        }
        const selUsers = timTable.getCheckedRows(0, true);
        const groups = TimTableComponent.makeSmallerMatrix(selUsers, [
            1,
            this.data.headers.indexOf("TIM-nimi"),
        ]);
        const params = groups.map(([sisuid, timname]) => ({
            externalId: sisuid,
            name: timname,
        }));
        this.loading = true;
        const r = await to(
            $http.post<{web: {result: string; error?: string}}>(
                "/sisu/createGroupDocs",
                params
            )
        );
        this.loading = false;
        if (r.ok) {
            timTable.confirmSaved();
            location.reload();
        } else {
            this.error = r.result.data.error;
        }
    }

```

```
    runJsRunner(runner: RunScriptType) {
        const timTable = this.getTimTable();
        if (timTable == null) {
            return;
        }
        const runnerName = runner.script;
        if (this.viewctrl && runnerName) {
            const selUsers = timTable.getCheckedRows(0, true);
            const users = TableFormComponent.makeUserArray(
                selUsers,
                userNameColIndex
            );
            this.viewctrl.runJsRunner(runnerName, users);
        }
        if (runner.update) {
            if (!timTable.isSomeCellBeingEdited()) {
                this.forceUpdateTable();
            }
        }
    }

    getAttributeType() {
        return TableFormAll;
    }
}

@NgModule({
    declarations: [TableFormComponent, TimMessageComponent],
    imports: [
        BrowserModule,
        HttpClientModule,
        FormsModule,
        TimUtilityModule,
        TimTableModule,
        BsDropdownModule.forRoot(),
        TimepickerModule.forRoot(),
        DatetimePickerModule,
    ],
    exports: [TimMessageComponent],
})
export class TableFormModule implements DoBootstrap {
    ngDoBootstrap(appRef: ApplicationRef) {}
}

export const moduleDefs = [
    doDowngrade(
        createDowngradedModule((extraProviders) =>
            platformBrowserDynamic(extraProviders).bootstrapModule(
                TableFormModule
            )
        ),
        "tableformRunner",
        TableFormComponent
    ),
];
```

timApp/static/scripts/tim/sidebarmenu/tabs/settings-tab.component.ts

```
import {Component, OnInit} from "@angular/core";
import {Users, UserService} from "tim/user/userService";
import {ViewCtrl} from "tim/document/viewctrl";
import {vctrlInstance} from "tim/document/viewctrlinstance";
import {LectureController} from "tim/lecture/lectureController";
import {
    DocumentOrFolder,
    IDocument,
    isRootFolder,
    redirectToItem,
} from "tim/item/IItem";
import {isDocumentGlobals, someglobals} from "tim/util/globals";
import {
    getCurrentViewRange,
    IViewRange,
    toggleViewRange,
} from "tim/document/viewRangeInfo";
import {getTypedStorage, IOkResponse, to2} from "tim/util/utils";
import {HttpClient} from "@angular/common/http";
import {getActiveDocument} from "tim/document/activedocument";
import {ITemplateParams} from "tim/printing/print-dialog.component";
import {
    ADMIN_GROUPNAME,
    IGroupWithSisuPath,
    TEACHERS_GROUPNAME,
} from "tim/user/IUser";
import {IDocSettings} from "tim/document/IDocSettings";
import {IRelevanceResponse} from "tim/item/relevance-edit.component";
import {showViewRangeEditDialog} from "tim/document/showViewRangeEditDialog";
import {showRelevanceEditDialog} from "tim/item/showRelevanceEditDialog";
import {showTagSearchDialog} from "tim/item/showTagSearchDialog";
import {showCourseDialog} from "tim/document/course/showCourseDialog";
import {showTagDialog} from "tim/item/showTagDialog";
import {showPrintDialog} from "tim/printing/showPrintDialog";
import {showInputDialog} from "tim/ui/showInputDialog";
import {InputDialogKind} from "tim/ui/input-dialog.kind";
import {showMergePdfDialog} from "tim/document/minutes/showMergePdfDialog";
import {showMessageDialog} from "tim/ui/showMessageDialog";
import * as t from "io-ts";
import {openScheduleDialog} from "tim/document/scheduling/openScheduleDialog";
import {showMessageListCreation} from "tim/messaging/showMessageListCreation.component";
import {getVisibilityVars, IVisibilityVars} from "tim/timRoot";

const DEFAULT_PIECE_SIZE = 20;

@Component({
    selector: "settings-tab",
    template: `
        <ng-template i18n="@@settingsTabTitle">Document settings</ng-template>
        <ng-container>
            <h5 i18n>Help</h5>
            <a i18n-title title="Open TIM-guide" href="/view/tim/TIM-ohjeet" ↩
    i18n>User guide</a>
        </ng-container>
        <ng-container *ngIf="users.isLoggedIn()">
```

```
55          <h5 i18n>Customize</h5>
56          <a href="/settings" i18n>Customize TIM</a>
57      </ng-container>
58      <ng-container *ngIf="showFolderSettings && showRelevance">
59          <h5 i18n>Folder settings</h5>
60          <button class="timButton btn-block"
61                  i18n-title title="Set item relevance value"
62                  (click)="openRelevanceEditDialog()"
63                  i18n>
64              Edit relevance (<span i18n-tooltip tooltip="Current relevance ↩
    value">{{currentRelevance}}</span>)
65          </button>
66      </ng-container>
67      <ng-container *ngIf="item && item.isFolder">
68          <h5 i18n>Search</h5>
69          <button class="timButton btn-block"
70                  i18n-title title="Search with tags"
71                  (click)="searchWithTags()"
72                  i18n>Search with tags
73          </button>
74      </ng-container>
75      <ng-container *ngIf="users.isLoggedIn() && item && !item.isFolder">
76          <h5 i18n>Document settings</h5>
77          <a i18n-title title="Toggle between showing full and partitioned document"
78             (click)="toggleViewRange()">
79              <ng-container *ngIf="isFullPage; else showInFull" i18n>Show page in ↩
    parts</ng-container>
80              <ng-template #showInFull i18n>Show page in full</ng-template>
81          </a>
82          <a class="same-line spaced" i18n-title title="Open document partitioning ↩
    settings"
83             (click)="openViewRangeMenu()">
84              <span class="glyphicon glyphicon-cog"></span>
85          </a>
86          <button *ngIf="vctrl && isFullPage && item.rights.editable"
87                  class="timButton btn-block"
88                  (click)="vctrl.editingHandler.editSettingsPars()"
89                  i18n>Edit settings
90          </button>
91          <button *ngIf="item && item.rights.manage"
92                  class="timButton btn-block"
93                  title="Set item relevance value" i18n-title
94                  (click)="openRelevanceEditDialog()"
95                  i18n>
96              Edit relevance (<span i18n-tooltip tooltip="Current relevance ↩
    value">{{currentRelevance}}</span>)
97          </button>
98          <button class="timButton btn-block"
99                  title="Mark all paragraphs of the document as read" i18n-title
100                 (click)="markAllAsRead()"
101                 i18n>Mark all as read
102         </button>
103         <button *ngIf="vctrl?.isTranslation()"
104                 class="timButton btn-block"
105                 title="Mark document as translated" i18n-title
106                 (click)="markTranslated()"
107                 i18n>Mark all as translated
108         </button>
109         <button *ngIf="docSettings?.exam_mode && item.rights.manage"
```

```
110                 class="timButton btn-block"
111                 title="Delete all read marks from all users who visited this ↩
    document" i18n-title
112                 (click)="markDocumentUnread()"
113                 i18n>Delete all read marks
114         </button>
115     </ng-container>
116     <ng-container *ngIf="lctrl.lectureSettings.inLecture">
117         <h5 i18n>Lecture settings</h5>
118         <div class="checkbox">
119             <label i18n><input type="checkbox" ↩
    [(ngModel)]="lctrl.lectureSettings.useWall"
120                                (ngModelChange)="lctrl.refreshWall()"> Show ↩
    wall</label>
121         </div>
122         <div *ngIf="!lctrl.isLecturer" class="checkbox">
123             <label i18n>
124                 <input type="checkbox" ↩
    [(ngModel)]="lctrl.lectureSettings.useQuestions"> Show questions
125             </label>
126         </div>
127         <div *ngIf="lctrl.isLecturer" class="checkbox">
128             <label i18n><input type="checkbox" ↩
    [(ngModel)]="lctrl.lectureSettings.useAnswers"> Show answers</label>
129         </div>
130     </ng-container>
131
132     <ng-container *ngIf="item && !item.isFolder">
133         <div>
134             <h5 class="same-line" i18n>Print document</h5>
135             <a class="same-line spaced" ↩
    href="https://tim.jyu.fi/view/tim/ohjeita/tulostusohje">
136                 <span class="glyphicon glyphicon-question-sign" title="Printing ↩
    help" i18n-title></span>
137             </a>
138         </div>
139         <button class="timButton btn-block"
140                 title="Print using LaTeX (best quality)" i18n-title
141                 (click)="printDocument()"
142                 i18n>Print document
143         </button>
144         <button class="timButton btn-block"
145                 title="Print via browser's print dialog" i18n-title
146                 (click)="cssPrint()"
147                 i18n>Browser print
148         </button>
149
150         <div>
151             <h5 class="same-line" i18n>Document tags</h5>
152             <a class="same-line spaced" ↩
    href="https://tim.jyu.fi/view/tim/ohjeita/opettajan-ohje#kurssikoodi">
153                 <span class="glyphicon glyphicon-question-sign"
154                       title="Teachers' help for course code" i18n-title></span>
155             </a>
156         </div>
157         <button *ngIf="item && item.rights.manage"
158                 class="timButton btn-block"
159                 title="Add or remove document tags" i18n-title
160                 (click)="addTag()"
```

```
                i18n>Edit tags
            </button>
            <button class="timButton btn-block"
                    title="Search with tags" i18n-title
                    (click)="searchWithTags()"
                    i18n>Search with tags
            </button>
            <button *ngIf="userBelongsToTeachersOrIsAdmin"
                    class="timButton btn-block"
                    title="Set document as a course main page" i18n-title
                    (click)="openCourseDialog()"
                    i18n>Set as course
            </button>

            <h5 *ngIf="isMinutesOrInvitation" i18n>Memo/Minutes</h5>
            <button *ngIf="enableCreateExtractsButton"
                    class="timButton btn-block"
                    title="Create extracts" i18n-title
                    (click)="createMinuteExtracts()"
                    i18n>Create extracts
            </button>
            <button *ngIf="enableCreateMinutesButton"
                    class="timButton btn-block"
                    title="Create minutes" i18n-title
                    (click)="createMinutes()"
                    i18n>Create minutes
            </button>
            <button *ngIf="isMinutesOrInvitation"
                    class="timButton btn-block"
                    title="Display attachments, check their validity, and merge them ↩
into single file." i18n-title
                    (click)="mergePdf()"
                    i18n>Merge attachments
            </button>
        </ng-container>

        <ng-container *ngIf="users.isGroupAdmin() || linkedGroups.length > 0">
            <h5 i18n>Groups</h5>
            <ng-container *ngIf="linkedGroups.length > 0">
                <h6 *ngIf="!sisugroupPath; else courseGroupsLink" i18n>Linked course ↩
groups</h6>
                <ng-template #courseGroupsLink i18n>Linked <a ↩
href="/view/{{sisugroupPath}}">course groups</a>
                </ng-template>
                <ul class="list-unstyled">
                    <li *ngFor="let group of linkedGroups"><a ↩
href="/view/{{group.path}}">{{group.title}}</a></li>
                </ul>
            </ng-container>
            <ng-container *ngIf="users.isGroupAdmin()">
                <button class="timButton btn-block"
                        title="Create a new group" i18n-title
                        (click)="createGroup()"
                        i18n>Create a new group
                </button>
                <a href="/view/groups" i18n>Browse existing groups</a>
            </ng-container>
            <ng-container *ngIf="users.isGroupAdmin() && !hideVars.messageListCreate">
                <h5>Message lists</h5>
```

```
                <button class="timButton btn-block"
                        title="Create a new message list"
                        (click)="createMessagelist()"
                >Create a new message list
                </button>
                <a href="/view/messagelists">Browse existing message lists</a>
                <br/>
                <a href="/view/archives">Browse archives</a>
            </ng-container>
        </ng-container>

        <ng-container *ngIf="docSettings?.cache && item?.rights?.manage">
            <h5 i18n>Cache</h5>
            <a href="/generateCache/{{ item?.path }}?caddy_nobuffering=1" target="_blank" i18n>Refresh cache</a>
        </ng-container>

        <ng-container *ngIf="users.canScheduleFunctions()">
            <h5 i18n>Scheduled functions</h5>
            <button (click)="openScheduleDialog()" class="timButton" i18n>Manage scheduled functions</button>
        </ng-container>

        <ng-template i18n="@@markAllReadFail">Could not mark the document as read.</ng-template>
        <ng-template i18n="@@markAllTranslatedConfirm">
            This will mark all paragraphs in this document as translated. Continue?
        </ng-template>
        <ng-template i18n="@@markAllUnreadConfirm">
            This document is in exam mode. Marking document unread will remove read marks from all users! Continue?
        </ng-template>
        <ng-template i18n="@@markAllUnreadAffectedCount">
            This will affect {{0}} users in total.
        </ng-template>
        <ng-template i18n="@@notInDocumentError">Not in a document</ng-template>
        <ng-template i18n="@@noKnroMacroError">The document has no 'knro' macro defined</ng-template>
    `,
})
export class SettingsTabComponent implements OnInit {
    hideVars: IVisibilityVars = getVisibilityVars();
    users: UserService = Users;
    showFolderSettings: boolean = false;
    showRelevance: boolean = true;
    currentRelevance?: number;
    vctrl?: ViewCtrl = vctrlInstance;
    lctrl: LectureController = LectureController.instance;
    isFullPage: boolean = true;
    linkedGroups: IDocument[] = [];
    sisugroupPath?: string;
    item?: DocumentOrFolder;
    docSettings?: IDocSettings;
    private currentViewRange?: IViewRange;
    private documentMemoMinutes: string | undefined;

    constructor(private http: HttpClient) {
        const globals = someglobals();
        this.item = globals.curr_item;
```

```
        this.docSettings = isDocumentGlobals(globals)
            ? globals.docSettings
            : undefined;
        this.documentMemoMinutes = isDocumentGlobals(globals)
            ? globals.memoMinutes
            : undefined;
        if (isDocumentGlobals(globals) && globals.linked_groups) {
            this.updateLinkedGroups(globals.linked_groups);
        }
    }

    ngOnInit(): void {
        void this.getCurrentRelevance();
        if (this.item) {
            this.showFolderSettings =
                this.users.isLoggedIn() && this.item.isFolder;
        }
        if (!this.item?.isFolder) {
            this.loadViewRangeSettings();
        }
    }

    /**
     * Open relevance edit dialog.
     */
    openRelevanceEditDialog() {
        if (this.item) {
            void showRelevanceEditDialog(this.item);
        }
    }

    /**
     * Opens tag search dialog.
     */
    searchWithTags() {
        void showTagSearchDialog();
    }

    /**
     * (Un)partition document (starting from the beginning) using user defined piece size.
     */
    async toggleViewRange() {
        if (!this.item) {
            return;
        }
        await toggleViewRange(
            this.item.id,
            getTypedStorage("pieceSize", t.number) ?? DEFAULT_PIECE_SIZE
        );
        this.currentViewRange = getCurrentViewRange();
        this.updateIsFullRange();
    }

    private updateIsFullRange() {
        this.isFullPage = this.currentViewRange?.is_full ?? true;
    }

    /**
```

```
     * Open dialog for editing view range settings.
     */
    openViewRangeMenu() {
        if (!this.item) {
            return;
        }
        void showViewRangeEditDialog(this.item);
        this.currentViewRange = getCurrentViewRange();
        this.updateIsFullRange();
    }

    /**
     * Marks all paragraphs of the document as read.
     */
    async markAllAsRead() {
        if (!this.item) {
            return;
        }
        const r = await to2(
            this.http.put(`/read/${this.item.id}`, {}).toPromise()
        );
        if (!r.ok) {
            await showMessageDialog(
                $localize`:@@markAllReadFail:Could not mark the document as read.`
            );
            return;
        }
        const doc = getActiveDocument();
        doc.hideReadMarks();
        doc.refreshSectionReadMarks();
    }

    async markDocumentUnread() {
        if (!this.item) {
            return;
        }
        const r = await to2(
            this.http
                .get<number>(`/read/${this.item.id}/groupCount`)
                .toPromise()
        );
        let message = $localize`:@@markAllUnreadConfirm:This document is in exam mode. Marking document unread will remove read marks from all users! Continue?`;
        if (r.ok) {
            message +=
                "\n" +
                $localize`:@@markAllUnreadAffectedCount:This will affect ${r.result}:INTERPOLATION: users in total.`;
        }
        await this.confirmPost(message, `/markAllUnread/${this.item.id}`);
    }

    async markTranslated() {
        await this.confirmPost(
            $localize`:@@markAllTranslatedConfirm:This will mark all paragraphs in this document as translated. Continue?`,
            `/markTranslated/${this.item!.id}`
        );
    }
```

```
384
385     private async confirmPost(message: string, url: string) {
386         if (!this.item) {
387             return;
388         }
389         const shouldMark = window.confirm(message);
390         if (!shouldMark) {
391             return;
392         }
393         const r = await to2(this.http.post<IOkResponse>(url, {}).toPromise());
394         if (r.ok) {
395             window.location.reload();
396         } else {
397             await showMessageDialog(r.result.error.error);
398         }
399     }
400
401     /**
402      * Opens print dialog.
403      */
404     async printDocument() {
405         if (!this.item) {
406             return;
407         }
408         const r = await to2(
409             this.http
410                 .get<ITemplateParams>(`/print/templates/${this.item.path}`)
411                 .toPromise()
412         );
413         if (r.ok) {
414             await showPrintDialog({document: this.item, params: r.result});
415         }
416     }
417
418     cssPrint() {
419         window.print();
420     }
421
422     /**
423      * Opens tag editing dialog.
424      */
425     addTag() {
426         if (!this.item) {
427             return;
428         }
429         void showTagDialog(this.item);
430     }
431
432     /**
433      * Opens 'Set as a course' -dialog.
434      */
435     async openCourseDialog() {
436         if (!this.item) {
437             return;
438         }
439         await showCourseDialog(this.item);
440         const r = await to2(
441             this.http
442                 .get<IGroupWithSisuPath[]>(
```

```
                    `/items/linkedGroups/${this.item.id}`
                )
                .toPromise()
        );
        if (r.ok) {
            this.updateLinkedGroups(r.result);
        } else {
            await showMessageDialog(r.result.error.error);
        }
    }

    private updateLinkedGroups(groups: IGroupWithSisuPath[]) {
        this.linkedGroups = [];
        for (const group of groups) {
            if (group.admin_doc) {
                this.linkedGroups.push(group.admin_doc);
            }
        }
        // TODO: Theoretically there can be multiple different courses.
        //  Should display a list in that case.
        const gr = groups.find((g) => g.sisugroup_path != null);
        if (gr?.sisugroup_path) {
            this.sisugroupPath = gr.sisugroup_path;
        }
    }

    /**
     * Checks whether user belongs to teachers or admins group.
     * @returns {boolean}
     */
    get userBelongsToTeachersOrIsAdmin() {
        return (
            this.users.belongsToGroup(ADMIN_GROUPNAME) ||
            this.users.belongsToGroup(TEACHERS_GROUPNAME)
        );
    }

    /**
     * Checks whether the side menu should have a button for creating extracts from ↩
    minutes in this document.
     * @returns {boolean} Whether the button for creating extracts should be displayed.
     */
    get enableCreateExtractsButton() {
        return (
            this.docSettings?.macros?.knro &&
            this.documentMemoMinutes == "minutes" &&
            this.item?.rights.manage
        );
    }

    createMinuteExtracts() {
        window.location.href = window.location.href.replace(
            "/view/",
            "/minutes/createMinuteExtracts/"
        );
    }

    /**
     * Checks whether the side menu should have a button for creating minutes in this ↩
```

```
    document.
501      * @returns {boolean} Whether the button for creating minutes should be displayed.
502      */
503     get enableCreateMinutesButton() {
504         return (
505             this.docSettings?.macros?.knro &&
506             this.documentMemoMinutes == "memo" &&
507             this.item?.rights.manage
508         );
509     }
510
511     /**
512      * Creates minutes from a IT faculty council meeting invitation
513      */
514     async createMinutes() {
515         if (!this.item) {
516             await showMessageDialog(
517                 $localize`:@@notInDocumentError:Not in a document`
518             );
519             return;
520         }
521
522         if (!this.docSettings?.macros?.knro) {
523             await showMessageDialog(
524                 $localize`:@@noKnroMacroError:The document has no 'knro' macro defined`
525             );
526             return;
527         }
528
529         const r = await to2(
530             this.http
531                 .post<{path: string}>("/minutes/createMinutes", {
532                     item_path: ↩
    `${this.item.location}/pk/pk${this.docSettings.macros.knro}`,
533                     item_title: `pk${this.docSettings.macros.knro}`,
534                     copy: this.item.id,
535                 })
536                 .toPromise()
537         );
538         if (r.ok) {
539             window.location.href = `/view/${r.result.path}`;
540         } else {
541             await showMessageDialog(r.result.error.error);
542         }
543     }
544
545     /**
546      * Checks if the document is faculty council minutes or a faculty council meeting ↩
    invitation.
547      * @returns {boolean} Whether the document is a faculty council meeting document.
548      */
549     get isMinutesOrInvitation() {
550         return (
551             this.docSettings?.macros?.knro &&
552             this.item?.rights.manage &&
553             (this.documentMemoMinutes == "minutes" ||
554                 this.documentMemoMinutes == "memo")
555         );
556     }
```

```
557
558     async mergePdf() {
559         if (!this.item) {
560             return;
561         }
562         await showMergePdfDialog({document: this.item});
563     }
564
565     async createGroup() {
566         const doc = await showInputDialog({
567             isInput: InputDialogKind.InputAndValidator,
568             defaultValue: "",
569             text: "Enter name of the usergroup",
570             title: "Create group",
571             validator: async (s) => {
572                 const r = await to2(
573                     this.http.get<IDocument>(`/groups/create/${s}`).toPromise()
574                 );
575                 if (r.ok) {
576                     return {ok: true, result: r.result};
577                 } else {
578                     return {ok: false, result: r.result.error.error};
579                 }
580             },
581         });
582         redirectToItem(doc);
583     }
584
585     async createMessagelist() {
586         await showMessageListCreation();
587     }
588
589     /**
590      * Fetches active relevance value. If root dir (id = -1), skip and hide relevance ↩
    dir.
591      */
592     private async getCurrentRelevance() {
593         if (this.item && !isRootFolder(this.item)) {
594             const r = await to2(
595                 this.http
596                     .get<IRelevanceResponse>(
597                         `/items/relevance/get/${this.item.id}`
598                     )
599                     .toPromise()
600             );
601             if (r.ok) {
602                 this.currentRelevance = r.result.relevance.relevance;
603             }
604         } else {
605             this.showRelevance = false; // Don't show in root folder.
606         }
607     }
608
609     /**
610      * Get piece size from local storage and current view range from document globals.
611      */
612     private loadViewRangeSettings() {
613         this.currentViewRange = getCurrentViewRange();
614         this.updateIsFullRange();
```

```
615     }
616
617     openScheduleDialog() {
618         if (!this.item) {
619             return;
620         }
621         openScheduleDialog(this.item);
622     }
623 }
```

timApp/static/scripts/tim/timRoot.ts

```
1  import {getUrlParams} from "tim/util/utils";
2  import {genericglobals, isDocumentGlobals, someglobals} from "tim/util/globals";
3  import {Users} from "tim/user/userService";
4  import {BookmarksComponent} from "tim/sidebarmenu/util/bookmarks.component";
5  import {setRoot} from "tim/rootinstance";
6  import {showMessageDialog} from "tim/ui/showMessageDialog";
7  import {timApp} from "./app";
8
9  export interface IVisibilityVars {
10     footer?: boolean;
11     siteheader?: boolean;
12     sidebar?: boolean;
13     header?: boolean;
14     bookmarks?: boolean;
15     getquestion?: boolean;
16     hamburger?: boolean;
17     index?: boolean;
18     lecturer?: boolean;
19     lecturetab?: boolean;
20     links?: boolean;
21     login?: boolean;
22     logo?: boolean;
23     search?: boolean;
24     settings?: boolean;
25     unilogo?: boolean;
26     velps?: boolean;
27     hakaLogin?: boolean;
28     emailLogin?: boolean;
29     signup?: boolean;
30     passwordRecovery?: boolean;
31     userMenuOptions?: boolean;
32     editLine?: boolean;
33     noteBadgeButton?: boolean;
34     headerNav?: boolean;
35     headerDocumentActions?: boolean;
36     scoreBoard?: boolean;
37     messageListCreate?: boolean;
38 }
39
40 function hideLinkStuff(hide: IVisibilityVars) {
41     hide.links = true;
42     hide.bookmarks = true;
43     hide.settings = true;
44     hide.hamburger = true;
45     hide.header = true;
46 }
47
```

```
48 function hideParsOnlyStuff(hide: IVisibilityVars) {
49     hide.bookmarks = true;
50     hide.getquestion = true;
51     hide.hamburger = true;
52     hide.index = true;
53     hide.lecturer = true;
54     hide.lecturetab = true;
55     hide.links = true;
56     hide.login = Users.isLoggedIn();
57     hide.logo = true;
58     hide.search = true;
59     hide.settings = true;
60     hide.unilogo = true;
61     hide.velps = true;
62     hide.header = true;
63     hide.sidebar = true;
64     hide.siteheader = Users.isLoggedIn();
65     hide.footer = true;
66 }
67
68 function hideTopButtonsStuff(hide: IVisibilityVars) {
69     hide.logo = true;
70     hide.search = true;
71     hide.unilogo = true;
72     hide.header = true;
73     // hide.login = true; // TODO: Should login be hidden or not?
74 }
75
76 function hideExamModeElements(hide: IVisibilityVars) {
77     hide.links = true;
78     hide.bookmarks = true;
79     hide.settings = true;
80     hide.headerNav = true;
81     hide.headerDocumentActions = true;
82     hide.search = true;
83     hide.userMenuOptions = true;
84     hide.editLine = true;
85     hide.noteBadgeButton = true;
86     hide.footer = true;
87 }
88
89 function hideSideMenu(hide: IVisibilityVars) {
90     hide.hamburger = true;
91     hide.bookmarks = true;
92     hide.index = true;
93     hide.settings = true;
94     hide.scoreBoard = true;
95 }
96
97 export function getVisibilityVars() {
98     const params = getUrlParams();
99     const g = someglobals();
100     let hide: IVisibilityVars = {};
101     if (isDocumentGlobals(g)) {
102         if (g.hideLinks) {
103             hideLinkStuff(hide);
104         }
105         if (g.parsOnly) {
106             hideParsOnlyStuff(hide);
```

```
        }
        if (g.hideTopButtons) {
            hideTopButtonsStuff(hide);
        }
        if (g.docSettings.login) {
            hide = {...hide, ...g.docSettings.login?.hide};
        }
        if (g.exam_mode) {
            hideExamModeElements(hide);
        }
        if (g.hide_sidemenu) {
            hideSideMenu(hide);
        }
    }

    // If bookmarks are globally disabled, don't show the tab.
    if (g.bookmarks == null) {
        hide.bookmarks = true;
    }

    if (!g.config.hakaEnabled) {
        hide.hakaLogin = true;
    }

    if (!g.config.emailRegistrationEnabled) {
        hide.signup = true;
    }

    if (!g.config.messageListsEnabled) {
        hide.messageListCreate = true;
    }

    if (params.get("hide_top_buttons")) {
        hideTopButtonsStuff(hide);
    }
    if (params.get("hide_links")) {
        hideLinkStuff(hide);
    }
    if (params.get("pars_only")) {
        hideParsOnlyStuff(hide);
    }
    return hide;
}

export class RootCtrl {
    public bookmarksCtrl: BookmarksComponent | undefined;
    private hide = getVisibilityVars();

    registerBookmarks(bookmarksCtrl: BookmarksComponent) {
        this.bookmarksCtrl = bookmarksCtrl;
    }

    $onInit() {
        setRoot(this);
        const g = genericglobals();
        if (g.config.hosts && Users.isLoggedIn()) {
            if (!g.config.hosts.allowed.includes(location.hostname)) {
                let message;
                for (const [key, val] of Object.entries(
```

```
                    g.config.hosts.warnings
                )) {
                    if (location.hostname.startsWith(key)) {
                        message = val;
                        break;
                    }
                }
                if (!message) {
                    message = g.config.hosts.defaultwarning;
                }
                const u = Users.getCurrent();
                const name = (
                    u.last_name ??
                    u.real_name?.split(" ")[0] ??
                    ""
                ).toLowerCase();
                message = message.replace(/LASTNAME/g, name);
                if (name) {
                    let letter;
                    if (name.length === 1) {
                        letter = name;
                    } else if (name.startsWith("k")) {
                        if (name.localeCompare("ko") < 0) {
                            letter = "ka";
                        } else {
                            letter = "ko";
                        }
                    } else {
                        letter = name[0];
                    }
                    message = message.replace(/LETTER/g, letter);
                    localStorage.clear(); // Make sure the dialog is shown in default position.
                    void showMessageDialog(message);
                }
            }
        }
    }
}

timApp.component("timRoot", {
    controller: RootCtrl,
    template: `<div ng-transclude style="display: flex; min-height: 100vh; flex-direction: column"></div>`,
    transclude: true,
});
```

timApp/static/scripts/tim/util/globals.ts

```
import {IBookmarkGroup} from "tim/bookmark/bookmark.service";
import {HeaderIndexItem} from "tim/sidebarmenu/services/header-indexer.service";
import {IDocScoreInfo} from "tim/sidebarmenu/services/scoreboard.service";
import {IDocSettings} from "../document/IDocSettings";
import {EditMode} from "../document/popup-menu-dialog.component";
import {IViewRange, IViewRangeUnnamed} from "../document/viewRangeInfo";
import {
    DocumentOrFolder,
    IDocument,
    IFolder,
```

```
11      IFullDocument,
12      IItem,
13      ITranslation,
14  } from "../item/IItem";
15  import {ILecture} from "../lecture/lecturetypes";
16  import {
17      IFullUser,
18      IGroup,
19      IGroupWithSisuPath,
20      IUser,
21      IUserListEntry,
22  } from "../user/IUser";
23  import {ICssFile, INotification, ISettings} from "../user/settings.component";
24
25  interface ILayout {
26      col_1_lg: number;
27      col_2_lg: number;
28      col_3_lg: number;
29      col_m_lg: number;
30      col_1_md: number;
31      col_2_md: number;
32      col_3_md: number;
33      col_m_md: number;
34      col_1_sm: number;
35      col_2_sm: number;
36      col_3_sm: number;
37      col_m_sm: number;
38      col_1_xs: number;
39      col_2_xs: number;
40      col_3_xs: number;
41      col_m_xs: number;
42  }
43
44  interface IHostConfig {
45      allowed: string[];
46      defaultwarning: string;
47      warnings: Record<string, string>;
48  }
49
50  interface IConfig {
51      minPasswordLength: number;
52      simpleLoginUseStudyInfoMessage: boolean;
53      gitLastestCommitTimestamp: string;
54      helpEmail: string;
55      gitBranch: string;
56      hakaEnabled: boolean;
57      emailRegistrationEnabled: boolean;
58      simpleEmailLogin: boolean;
59      hosts?: IHostConfig;
60      messageListsEnabled: boolean;
61  }
62
63  export type Locale = "fi" | "en-US";
64
65  export interface IGenericGlobals {
66      IS_TESTING: boolean;
67      current_user: IFullUser;
68      locale: Locale;
69      other_users: IUser[];
```

```
70      bookmarks: IBookmarkGroup[] | null;
71      ANGULARMODULES: unknown[];
72      JSMODULES: string[];
73      curr_item?: DocumentOrFolder;
74      userPrefs: ISettings;
75      homeOrganization: string;
76      config: IConfig;
77      layout: ILayout;
78      lectureInfo: {in_lecture: boolean; is_lecturer: boolean};
79  }
80
81  export interface IItemGlobals extends IGenericGlobals {
82      breadcrumbs: IFolder[];
83      curr_item: DocumentOrFolder;
84  }
85
86  export interface IFolderGlobals extends IItemGlobals {
87      items: IItem[];
88      curr_item: IFolder;
89  }
90
91  export interface IDocumentGlobals extends IItemGlobals {
92      parsOnly: boolean;
93      users: IUserListEntry[];
94      startIndex: number;
95      docVersion: [number, number];
96      curr_item: IDocument;
97      noBrowser: boolean;
98      allowMove: boolean; // TODO this doesn't come from server and should be removed ↩
        from globals
99      group: IGroup;
100     docSettings: IDocSettings;
101     editMode: EditMode | null;
102     hideLinks: boolean;
103     hideTopButtons: boolean;
104     index: HeaderIndexItem[];
105     lectureMode: boolean;
106     liveUpdates: number;
107     memoMinutes?: string;
108     showReviewTab: boolean; // needs functionality
109     noQuestionAutoNumbering: boolean;
110     notifications: unknown[];
111     readExpiry: string;
112     reqs: Record<string, unknown>; // TODO proper type
113     showIndex: boolean;
114     teacherMode: boolean;
115     translations: ITranslation[];
116     velpMode: boolean;
117     wordList: string[];
118     linked_groups: IGroupWithSisuPath[] | null;
119     current_view_range?: IViewRangeUnnamed | null;
120     nav_ranges?: IViewRange[];
121     exam_mode: boolean;
122     hide_sidemenu: boolean;
123     score_infos: IDocScoreInfo[] | null;
124     current_list_user?: IUser;
125     show_unpublished_bg: boolean;
126 }
127
```

```
128 export interface ILectureInfoGlobals extends IDocumentGlobals {
129     lecture: ILecture<string>;
130     inLecture: boolean;
131 }
132
133 export interface ISlideGlobals extends IDocumentGlobals {
134     background_url: string;
135     background_color: string;
136 }
137
138 export interface IManageGlobals extends IGenericGlobals {
139     orgs: IGroup[];
140     accessTypes: Array<unknown>; // TODO proper type
141     curr_item: IFullDocument | IFolder;
142 }
143
144 export interface ISettingsGlobals extends IGenericGlobals {
145     settings: ISettings;
146     css_files: Array<ICssFile>;
147     notifications: INotification[];
148     notificationLimit: number;
149 }
150
151 export type SomeGlobals =
152     | IGenericGlobals
153     | IFolderGlobals
154     | IDocumentGlobals
155     | ISlideGlobals
156     | IManageGlobals
157     | ISettingsGlobals
158     | ILectureInfoGlobals;
159
160 export function someglobals(): SomeGlobals {
161     return someGlobals();
162 }
163
164 export function itemglobals(): IItemGlobals {
165     return someGlobals();
166 }
167
168 export function genericglobals(): IGenericGlobals {
169     return someGlobals();
170 }
171
172 export function documentglobals(): IDocumentGlobals {
173     return someGlobals();
174 }
175
176 export function slideglobals(): ISlideGlobals {
177     return someGlobals();
178 }
179
180 export function folderglobals(): IFolderGlobals {
181     return someGlobals();
182 }
183
184 export function manageglobals(): IManageGlobals {
185     return someGlobals();
186 }
```

```
187
188 export function settingsglobals(): ISettingsGlobals {
189     return someGlobals();
190 }
191
192 export function lectureinfoglobals(): ILectureInfoGlobals {
193     return someGlobals();
194 }
195
196 export function isDocumentGlobals(g: SomeGlobals): g is IDocumentGlobals {
197     return "docSettings" in g;
198 }
199
200 function someGlobals<T extends IGenericGlobals>(): T {
201     return (window as unknown) as T;
202 }
```

timApp/templates/base.jinja2

```
1  {% import 'macros.jinja2' as m -%}
2
3  {% set lecture_mode = route == "lecture" %}
4  {% set is_lecture_info = request.endpoint == 'lecture.show_lecture_info' %}
5  {% set show_lecture_menu = lecture_info.in_lecture or lecture_mode or is_lecture_info %}
6
7  {# Set default column widths if they are not defined in a child template #}
8  {% set col_1_lg = col_1_lg|default(2) %}
9  {% set col_2_lg = col_2_lg|default(8) %}
10 {% set col_3_lg = col_3_lg|default(2) %}
11 {% set col_m_lg = col_m_lg|default(12) %} {# col_m_* must be even #}
12
13 {% set col_1_md = col_1_md|default(2) %}
14 {% set col_2_md = col_2_md|default(8) %}
15 {% set col_3_md = col_3_md|default(2) %}
16 {% set col_m_md = col_m_md|default(12) %}
17
18 {% set col_1_sm = col_1_sm|default(2) %}
19 {% set col_2_sm = col_2_sm|default(10) %}
20 {% set col_3_sm = col_3_sm|default(0) %}
21 {% set col_m_sm = col_m_sm|default(12) %}
22
23 {% set col_1_xs = col_1_xs|default(0) %}
24 {% set col_2_xs = col_2_xs|default(12) %}
25 {% set col_3_xs = col_3_xs|default(0) %}
26 {% set col_m_xs = col_m_xs|default(12) %}
27
28 <!DOCTYPE html>
29 <html lang="{{ locale }}">
30 <head>
31     {% block head %}
32         <base href="{{ request.path }}">
33         <meta charset="utf-8">
34         <meta http-equiv="X-UA-Compatible" content="IE=edge">
35         <meta name="viewport" content="width=device-width, initial-scale=1">
36         <!-- The above 3 meta tags *must* come first in the head; any other head ↩
       content must come *after* these tags -->
37         <meta name="description" content="TIM - The Interactive Material."/>
38         <meta name="keywords" content="tim,interactive,material"/>
39
```

```
40          <link href="{{ url_for('static', filename='images/favicon.ico') }}" ↩
    rel="shortcut icon" type="image/x-icon"/>
41          <meta name="google-site-verification" ↩
    content="GHYGje2NiYlQM66xyggsnzQht1OHSgtv_9baHbFIklU"/>
42          <script>
43              var homeOrganization = {{ config.HOME_ORGANIZATION|tojson }};
44              var IS_TESTING = {{ config.TESTING|tojson }};
45              var ANGULARMODULES = [];
46              var JSMODULES = [];
47
48              {% if show_lecture_menu %}JSMODULES.push("lectureMenu");{% endif %}
49              {% if is_lecture_info %}JSMODULES.push("lectureInfo");{% endif %}
50
51              var userPrefs = {{ prefs|tojson }};
52              var current_user = {{ current_user.to_json(full=True)|tojson }};
53              var other_users = {{ other_users|tojson }};
54              var bookmarks = {{ bookmarks|tojson }};
55              var locale = {{locale|tojson}};
56              var lectureInfo = {{lecture_info|tojson}};
57
58              var layout = {};
59              layout.col_1_lg = {{col_1_lg|tojson}};
60              layout.col_2_lg = {{col_2_lg|tojson}};
61              layout.col_3_lg = {{col_3_lg|tojson}};
62              layout.col_m_lg = {{col_m_lg|tojson}};
63              layout.col_1_md = {{col_1_md|tojson}};
64              layout.col_2_md = {{col_2_md|tojson}};
65              layout.col_3_md = {{col_3_md|tojson}};
66              layout.col_m_md = {{col_m_md|tojson}};
67              layout.col_1_sm = {{col_1_sm|tojson}};
68              layout.col_2_sm = {{col_2_sm|tojson}};
69              layout.col_3_sm = {{col_3_sm|tojson}};
70              layout.col_m_sm = {{col_m_sm|tojson}};
71              layout.col_1_xs = {{col_1_xs|tojson}};
72              layout.col_2_xs = {{col_2_xs|tojson}};
73              layout.col_3_xs = {{col_3_xs|tojson}};
74              layout.col_m_xs = {{col_m_xs|tojson}};
75
76              var config = {};
77              config.gitLastestCommitTimestamp = ↩
    {{config.GIT_LATEST_COMMIT_TIMESTAMP|tojson}};
78              config.helpEmail = {{config.HELP_EMAIL|tojson}};
79              config.gitBranch = {{config.GIT_BRANCH|tojson}};
80              config.hakaEnabled = {{config.HAKA_ENABLED|tojson}};
81              config.emailRegistrationEnabled = ↩
    {{config.EMAIL_REGISTRATION_ENABLED|tojson}};
82              config.simpleEmailLogin = {{config.SIMPLE_EMAIL_LOGIN|tojson}};
83              config.simpleLoginUseStudyInfoMessage = ↩
    {{config.SIMPLE_LOGIN_USE_STUDY_INFO_MESSAGE|tojson}};
84              config.minPasswordLength = {{config.MIN_PASSWORD_LENGTH|tojson}};
85              config.hosts = {{config.HOSTS|tojson}};
86              config.messageListsEnabled = {{config.MESSAGE_LISTS_ENABLED|tojson}};
87          </script>
88          {{ angularscripts|safe }}
89          {% if not override_theme %}
90              {{ m.scss(config.SASS_GEN_PATH / prefs.css_combined) }}
91          {% endif %}
92
93          <script type="text/x-mathjax-config">
```

```
94              MathJax.Hub.Config({
95                  skipStartupTypeset: true
96              });
97          </script>
98          <title>{% block title %}{% endblock %} - TIM</title>
99      {% endblock %}
100     <style type="text/css">
101         {{ prefs.custom_css }}
102     </style>
103 </head>
104 <body>
105 {% block body %}
106 {% if request.user_agent.browser == 'msie' %}
107 <div style="text-align: center; border: 2px solid red; padding: 1em;">
108     <p>TIM ei toimi Internet Explorer -selaimella. Siirry käyttämään modernia ↩
    selainta, kuten Firefoxia tai Chromea.</p>
109
110     <p>TIM does not work with Internet Explorer. Please switch to a modern browser ↩
    such as Firefox or Chrome.</p>
111 </div>
112 {% endif %}
113 <tim-root>
114 <{% block mainctrl %}div{% endblock %} style="flex: 1">
115     <div ng-init="$ctrl = $parent.$ctrl" class="container-fluid">
116         <div class="row hidden-print" ng-cloak>
117             <div class="col-lg-{{ col_1_lg }}
118                         col-md-{{ col_1_md }}
119                         col-xs-{{ col_1_xs }}
120                         col-sm-{{ col_1_sm }}">
121                 <tim-sidebar-menu></tim-sidebar-menu>
122             </div>
123             <div class="col-lg-{{ col_2_lg }}
124                         col-md-{{ col_2_md }}
125                         col-xs-{{ col_2_xs }}
126                         col-sm-{{ col_2_sm }}">
127                 <tim-site-header></tim-site-header>
128             </div>
129             <div class="col-lg-{{ col_3_lg }}
130                         col-md-{{ col_3_md }}
131                         col-xs-{{ col_3_xs }}
132                         col-sm-{{ col_3_sm }}">
133                 <div class="right-fixed-side">
134                     {% block rightside %}{% endblock %}
135                 </div>
136             </div>
137         </div>
138         <div class="row" ng-if="!$ctrl.hide.messages" ng-cloak>
139             <div class="col-lg-{{ col_2_lg }} col-lg-offset-{{ col_1_lg }}
140                         col-md-{{ col_2_md }} col-md-offset-{{ col_1_md }}
141                         col-xs-{{ col_2_xs }} col-xs-offset-{{ col_1_xs }}
142                         col-sm-{{ col_2_sm }} col-sm-offset-{{ col_1_sm }}">
143                 {% for n in global_notifications %}
144                     <div class="global-notification alert alert-info">
145                         <i class="glyphicon glyphicon-info-sign"></i> {{ n|safe }}
146                     </div>
147                 {% endfor %}
148                 <tim-message-view></tim-message-view>
149                 {% with messages = get_flashed_messages() %}
150                     {% if messages %}
```

```
                        {% for message in messages %}
                            <div class="alert alert-info">
                                <i class="glyphicon glyphicon-info-sign"></i> {{ message }}
                            </div>
                        {% endfor %}
                    {% endif %}
                {% endwith %}
            </div>
        </div>
        <div class="row" ng-if="!$ctrl.hide.header" ng-cloak>
            <div class="col-lg-{{ col_2_lg }} col-lg-offset-{{ col_1_lg }}
                        col-md-{{ col_2_md }} col-md-offset-{{ col_1_md }}
                        col-xs-{{ col_2_xs }} col-xs-offset-{{ col_1_xs }}
                        col-sm-{{ col_2_sm }} col-sm-offset-{{ col_1_sm }}}">
                <div id="header" class="hidden-print">
                    <tim-header></tim-header>
                </div>
            </div>
        </div>
        <div class="row">
            <div class="col-lg-{{ col_2_lg }} col-lg-offset-{{ col_1_lg }}
                        col-md-{{ col_2_md }} col-md-offset-{{ col_1_md }}
                        col-xs-{{ col_2_xs }} col-xs-offset-{{ col_1_xs }}
                        col-sm-{{ col_2_sm }} col-sm-offset-{{ col_1_sm }}}">
                <div class="row">
                    <div class="col-lg-{{ col_m_lg }} col-lg-offset-{{ (12 - col_m_lg) // 2 }}
                                col-md-{{ col_m_md }} col-md-offset-{{ (12 - col_m_md) // 2 }}
                                col-xs-{{ col_m_xs }} col-xs-offset-{{ (12 - col_m_xs) // 2 }}
                                col-sm-{{ col_m_sm }} col-sm-offset-{{ (12 - col_m_sm) // 2 }}">
                        <div class="material">
                            {% if show_lecture_menu %}
                                <tim-lecture-menu></tim-lecture-menu>
                            {% endif %}
                            {% block content %}{% endblock %}
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </div>
</{{ self.mainctrl() }}>
<tim-footer ng-if="!$parent.$ctrl.hide.footer"></tim-footer>
</tim-root>
{% endblock %}
<div class="device-xs visible-xs-block"></div>
<div class="device-sm visible-sm-block"></div>
<div class="device-md visible-md-block"></div>
<div class="device-lg visible-lg-block"></div>
<tim-dialog-container></tim-dialog-container>
</body>
</html>
```

timApp/tim.py

```
# -*- coding: utf-8 -*-
import time
import traceback
from dataclasses import dataclass
from typing import Optional, List
from urllib.parse import urlparse

import bs4
import requests
from bs4 import BeautifulSoup
from flask import Response, send_file
from flask import g
from flask import redirect
from flask import render_template
from flask import request
from flask import session
from flask_assets import Environment
from flask_wtf.csrf import generate_csrf
from requests.exceptions import MissingSchema, InvalidURL
from sqlalchemy import event
from werkzeug.middleware.profiler import ProfilerMiddleware

from timApp.admin.cli import register_clis
from timApp.admin.global_notification import global_notification
from timApp.admin.routes import admin_bp
from timApp.backup.backup_routes import backup
from timApp.answer.feedbackanswer import feedback
from timApp.answer.routes import answers
from timApp.auth.accesshelper import verify_edit_access, verify_logged_in
from timApp.auth.login import login_page
from timApp.auth.saml import saml
from timApp.auth.sessioninfo import get_current_user_object, get_other_users_as_list, ↩
    logged_in
from timApp.bookmark.bookmarks import Bookmarks
from timApp.bookmark.routes import bookmarks, add_to_course_bookmark
from timApp.defaultconfig import SECRET_KEY
from timApp.document.course.routes import course_blueprint
from timApp.document.course.validate import is_course
from timApp.document.create_item import get_templates_for_folder
from timApp.document.docentry import DocEntry
from timApp.document.editing.routes import edit_page
from timApp.document.editing.routes_clipboard import clipboard
from timApp.document.minutes.routes import minutes_blueprint
from timApp.document.routes import doc_bp
from timApp.document.translation.routes import tr_bp
from timApp.errorhandlers import register_errorhandlers
from timApp.gamification.generateMap import generateMap
from timApp.item.block import Block
from timApp.item.distribute_rights import dist_bp
from timApp.item.manage import manage_page
from timApp.item.routes import view_page
from timApp.item.routes_tags import tags_blueprint
from timApp.item.tag import Tag, GROUP_TAG_PREFIX
from timApp.lecture.lectureutils import get_current_lecture_info
from timApp.lecture.routes import lecture_routes
from timApp.messaging.messagelist.mailman_events import mailman_events
from timApp.messaging.messagelist.routes import messagelist
from timApp.modules.fields.cbcountfield import cbcountfield_route
from timApp.note.routes import notes
```

```
from timApp.notification.notify import notify
from timApp.plugin.importdata.importData import importData_plugin
from timApp.plugin.qst.qst import qst_plugin
from timApp.plugin.routes import plugin_bp
from timApp.plugin.tableform.tableForm import tableForm_plugin
from timApp.plugin.tape.tape import tape_plugin
from timApp.plugin.timmenu.timMenu import timMenu_plugin
from timApp.plugin.timtable.timTable import timTable_plugin
from timApp.plugin.userselect.userselect import user_select_plugin
from timApp.printing.print import print_blueprint
from timApp.readmark.routes import readings
from timApp.scheduling.scheduling_routes import scheduling
from timApp.sisu.scim import scim
from timApp.sisu.sisu import sisu
from timApp.messaging.timMessage.routes import timMessage
from timApp.tim_app import app
from timApp.timdb.sqa import db
from timApp.upload.upload import upload
from timApp.user.groups import groups
from timApp.user.settings.settings import settings_page
from timApp.user.usergroup import UserGroup
from timApp.util.flask.cache import cache
from timApp.util.flask.requesthelper import get_request_message, use_model, ↩
    RouteException, NotExist
from timApp.util.flask.responsehelper import json_response, ok_response, add_csp_header
from timApp.util.flask.search import search_routes
from timApp.util.logger import log_info, log_debug
from timApp.util.utils import get_current_time
from timApp.velp.annotation import annotations
from timApp.velp.velp import velps

cache.init_app(app)

blueprints = [
    admin_bp,
    annotations,
    answers,
    backup,
    clipboard,
    course_blueprint,
    dist_bp,
    doc_bp,
    edit_page,
    feedback,
    generateMap,
    global_notification,
    groups,
    saml,
    lecture_routes,
    login_page,
    manage_page,
    minutes_blueprint,
    notes,
    notify,
    plugin_bp,
    print_blueprint,
    readings,
    scim,
    search_routes,
```

```
117     settings_page,
118     sisu,
119     tags_blueprint,
120     tr_bp,
121     upload,
122     velps,
123     view_page,
124     scheduling,
125     mailman_events,
126 
127     # plugins
128     importData_plugin,
129     qst_plugin,
130     tableForm_plugin,
131     tape_plugin,
132     cbcountfield_route,
133     timMenu_plugin,
134     timTable_plugin,
135     user_select_plugin,
136     messagelist,
137     timMessage,
138 ]
139 
140 if app.config['BOOKMARKS_ENABLED']:
141     app.register_blueprint(bookmarks)
142 
143 for bp in blueprints:
144     app.register_blueprint(bp)
145 
146 
147 assets = Environment(app)
148 
149 register_errorhandlers(app)
150 register_clis(app)
151 
152 
153 @app.context_processor
154 def inject_custom_css() -> dict:
155     """Injects the user prefs variable to all templates."""
156     prefs = get_current_user_object().get_prefs()
157     return dict(prefs=prefs)
158 
159 
160 @app.context_processor
161 def inject_angular_scripts() -> dict:
162     """Provides the JavaScript files compiled by Angular."""
163     locale = get_locale()
164     try:
165         # Angular compiler modifies the base href of index.html to redirect the ↩
    scripts to correct location,
166         # but it does not work for TIM because the HTML is dynamically generated and ↩
    modifying base href would break
167         # other links. So we modify the script by hand.
168         # TODO: Cache the modified result.
169         return get_angularscripts(f'static/scripts/build/{locale}/index.html', ↩
    locale=locale)
170     except FileNotFoundError:
171         try:
172             return get_angularscripts(f'static/scripts/build/index.html')
```

```
173         except FileNotFoundError:
174             raise Exception(
175                 'TypeScript files have not been built (compiled JavaScript files are ↩
    missing).\n'
176                 'If this is a local development TIM instance, start the "bdw" NPM ↩
    script (in timApp/package.json) '
177                 'from your IDE.\n'
178                 'If this is not a local TIM instance, run "./js" from TIM root.'
179             )
180 
181 
182 def get_angularscripts(index_file: str, locale: Optional[str]=None):
183     with open(index_file) as f:
184         html_data = f.read()
185         bs = BeautifulSoup(html_data, 'lxml')
186         scripts: List[bs4.element.Tag] = [e for e in bs.find_all('script')]
187         n = BeautifulSoup("", 'lxml')
188         style = bs.find('link')
189         for s in scripts:
190             n.append(s)
191             if locale:
192                 s['src'] += f'?l={locale}'  # The parameter is only needed for cache ↩
    busting (for Chrome).
193 
194         # Only production config has extractCss enabled, so this will be None for a ↩
    non-prod build.
195         # TODO: this is possibly always True after upgrading to Angular 11.
196         if style:
197             n.append(style)
198         return dict(angularscripts=str(n))
199 
200 
201 KNOWN_LANGUAGES = [
202     'fi',
203     'en-US',
204 ]
205 
206 
207 def get_locale():
208     header_lang = request.accept_languages.best_match(KNOWN_LANGUAGES, default='en-US')
209     lng = request.cookies.get('lang')
210     if not lng:
211         if not logged_in():
212             return header_lang
213         u = get_current_user_object()
214         lng = u.get_prefs().language
215     if lng in KNOWN_LANGUAGES:
216         return lng
217     return header_lang
218 
219 
220 @app.context_processor
221 def inject_user() -> dict:
222     """Injects user-related info to all templates."""
223     r = dict(
224         current_user=get_current_user_object(),
225         lecture_info=get_current_lecture_info(),
226         other_users=get_other_users_as_list(),
227         locale=get_locale(),
```

```
    )
    if logged_in() and app.config['BOOKMARKS_ENABLED']:
        r['bookmarks'] = get_current_user_object().bookmarks.as_dict()
    return r


@app.route('/js/<path:path>')
def get_js_file(path: str):
    locale = get_locale()
    for f in [
        f'static/scripts/build/{locale}/{path}',
        f'static/scripts/build/{path}',
    ]:
        try:
            return send_file(f, conditional=True)
        except FileNotFoundError:
            pass
    raise NotExist('File not found')


@app.route('/empty')
def empty_response_route():
    return Response('', mimetype='text/plain')


@app.route("/ping")
def ping():
    return ok_response()


@dataclass
class GetProxyModel:
    url: str
    auth_token: Optional[str] = None
    raw: bool = False
    mimetype: Optional[str] = None


@app.route("/getproxy")
@use_model(GetProxyModel)
def getproxy(m: GetProxyModel):
    parsed = urlparse(m.url)
    if not parsed.scheme:
        raise RouteException('Unknown URL scheme')
    if parsed.scheme not in ('http', 'https'):
        raise RouteException(f'URL scheme not allowed: {parsed.scheme}')
    if parsed.netloc not in app.config['PROXY_WHITELIST']:
        raise RouteException(f'URL domain not whitelisted: {parsed.netloc}')
    if parsed.netloc not in app.config['PROXY_WHITELIST_NO_LOGIN']:
        verify_logged_in()
    headers = {}
    if m.auth_token:
        headers['Authorization'] = f'Token {m.auth_token}'
    try:
        r = requests.get(m.url, headers=headers)
    except (MissingSchema, InvalidURL):
        raise RouteException('Invalid URL')
    if m.raw:
        mimetype = r.headers['Content-Type']
```

```
287         if m.mimetype:
288             mimetype = m.mimetype
289         resp = Response(
290             r.content,
291             status=r.status_code,
292             mimetype=mimetype,
293         )
294         add_csp_header(resp, 'sandbox allow-scripts')
295         return resp
296 
297     return json_response({'data': r.text, 'status_code': r.status_code})
298 
299 
300 @app.route("/time")
301 def get_time():
302     return json_response({'time': get_current_time()}, date_conversion=True)
303 
304 
305 @app.route("/getTemplates")
306 def get_templates():
307     current_path = request.args.get('item_path', '')
308     d = DocEntry.find_by_path(current_path)
309     if not d:
310         raise NotExist()
311     verify_edit_access(d)
312     templates = get_templates_for_folder(d.parent)
313     return json_response(templates, date_conversion=True)
314 
315 
316 def update_user_course_bookmarks():
317     u = get_current_user_object()
318     for gr in u.groups:  # type: UserGroup
319         if gr.is_sisu_student_group:
320             docs = DocEntry.query.join(Block).join(Tag).filter(Tag.name == ↩
    GROUP_TAG_PREFIX + gr.name).with_entities(DocEntry).all()
321             if not docs:
322                 continue
323             if len(docs) > 1:
324                 continue
325             d: DocEntry = docs[0]
326             if not is_course(d):
327                 continue
328             if d.document.get_settings().sisu_require_manual_enroll():
329                 continue
330             b = Bookmarks(u)
331             add_to_course_bookmark(b, d)
332 
333 
334 @app.route("/en")
335 @app.route("/fi")
336 @app.route("/")
337 def start_page():
338     update_user_course_bookmarks()
339     db.session.commit()
340     return render_template(
341         'start.jinja2',
342     )
343 
344 
```

```
345 def install_sql_hook():
346     prev_exec_time = get_current_time()
347 
348     @event.listens_for(db.engine, 'before_execute')
349     def receive_before_execute(conn, clauseelement, multiparams, params):
350         nonlocal prev_exec_time
351         curr = get_current_time()
352         print(f'-------------------------------------TIMING: {curr} ({curr - ↩
    prev_exec_time})')
353         prev_exec_time = curr
354         for r in traceback.format_stack():
355             if r.startswith('  File "/service/') and not ↩
    receive_before_execute.__name__ in r:
356                 print(r, end='')
357         print(clauseelement)
358 
359 
360 if app.config['DEBUG_SQL']:
361     install_sql_hook()
362 
363 
364 @app.before_request
365 def preprocess_request():
366     session.permanent = True
367     g.request_start_time = time.monotonic()
368     if request.method == 'GET':
369         p = request.path
370         if '//' in p or (p.endswith('/') and p != '/'):
371             fixed_url = p.rstrip('/').replace('//', '/') + '?' + ↩
    request.query_string.decode()
372             return redirect(fixed_url)
373 
374 
375 def should_log_request():
376     p = request.path
377     if p.startswith('/static/'):
378         raise Exception('static files should be served by Caddy, not Flask')
379     if p == '/favicon.ico':
380         return False
381     return True
382 
383 
384 @app.after_request
385 def log_request(response):
386     if should_log_request():
387         status_code = response.status_code
388         log_info(get_request_message(status_code))
389         if request.method in ('PUT', 'POST', 'DELETE'):
390             log_debug(request.get_json(silent=True))
391     return response
392 
393 
394 @app.after_request
395 def close_db(response):
396     if hasattr(g, 'timdb'):
397         g.timdb.close()
398     return response
399 
400 
```

```
@app.after_request
def del_g(response):
    """For some reason, the g object is not cleared when running browser test, so we do it here."""
    if app.config['TESTING']:
        if hasattr(g, 'user'):
            del g.user
        if hasattr(g, 'viewable'):
            del g.viewable
        if hasattr(g, 'editable'):
            del g.editable
        if hasattr(g, 'teachable'):
            del g.teachable
        if hasattr(g, 'manageable'):
            del g.manageable
        if hasattr(g, 'see_answers'):
            del g.see_answers
        if hasattr(g, 'owned'):
            del g.owned
    return response


@app.after_request
def after_request(resp: Response):
    token = generate_csrf()
    resp.set_cookie(
        'XSRF-TOKEN',
        token,
        samesite=app.config['SESSION_COOKIE_SAMESITE'],
        secure=app.config['SESSION_COOKIE_SECURE'],
    )
    if not request.cookies.get('lang'):
        resp.set_cookie('lang', get_locale())
    return resp


@app.teardown_appcontext
def close_db_appcontext(_e):
    if not app.config['TESTING'] and hasattr(g, 'timdb'):
        g.timdb.close()


def init_app():
    if app.config['PROFILE']:
        app.wsgi_app = ProfilerMiddleware(app.wsgi_app, sort_by=('cumtime',), restrictions=[100])

    for var in [
        'DB_URI',
        'DEBUG',
        'MAIL_HOST',
        'PG_MAX_CONNECTIONS',
        'PLUGIN_CONNECT_TIMEOUT',
        'PROFILE',
        'SQLALCHEMY_MAX_OVERFLOW',
        'SQLALCHEMY_POOL_SIZE',
    ]:
        log_info(f'{var}: {app.config.get(var, "(undefined)")}')
    if not app.config['DEBUG']:
```

```
458         if app.config['SECRET_KEY'] == SECRET_KEY:
459             raise Exception('SECRET_KEY must not be the same as default SECRET_KEY ↩
    when DEBUG=False')
460     return app
461 
462 
463 def start_app() -> None:
464     init_app()
465     app.run(host='0.0.0.0',
466             port=5000,
467             use_evalex=False,
468             use_reloader=False,
469             threaded=True)
```

timApp/tim_app.py

```
1  """Creates the Flask application for TIM.
2 
3  Insert only configuration-related things in this file. Do NOT define routes here.
4 
5  """
6  import inspect
7  import mimetypes
8  import os
9  import sys
10 from typing import Optional
11 
12 from flask import Flask
13 from flask_migrate import Migrate
14 from flask_wtf import CSRFProtect
15 from sqlalchemy import func
16 from sqlalchemy.sql.ddl import CreateTable
17 from werkzeug.middleware.proxy_fix import ProxyFix
18 
19 from timApp.answer.answer import Answer, AnswerSaver
20 from timApp.answer.answer_models import AnswerTag, AnswerUpload, UserAnswer
21 from timApp.auth.auth_models import AccessTypeModel, BlockAccess
22 from timApp.celery_sqlalchemy_scheduler import IntervalSchedule, CrontabSchedule, ↩
    SolarSchedule, PeriodicTaskChanged, \
23     PeriodicTask
24 from timApp.document.docentry import DocEntry
25 from timApp.document.timjsonencoder import TimJsonEncoder
26 from timApp.document.translation.translation import Translation
27 from timApp.folder.folder import Folder
28 from timApp.item.block import Block
29 from timApp.item.blockassociation import BlockAssociation
30 from timApp.item.blockrelevance import BlockRelevance
31 from timApp.item.tag import Tag
32 from timApp.lecture.askedjson import AskedJson
33 from timApp.lecture.askedquestion import AskedQuestion
34 from timApp.lecture.lecture import Lecture
35 from timApp.lecture.lectureanswer import LectureAnswer
36 from timApp.lecture.lectureusers import LectureUsers
37 from timApp.lecture.message import Message
38 from timApp.lecture.question import Question
39 from timApp.lecture.questionactivity import QuestionActivity
40 from timApp.lecture.runningquestion import Runningquestion
41 from timApp.lecture.showpoints import Showpoints
42 from timApp.lecture.useractivity import Useractivity
```

```
from timApp.messaging.messagelist.messagelist_models import MessageListModel, MessageListMember, \
    MessageListExternalMember, MessageListTimMember, MessageListDistribution
from timApp.messaging.timMessage.internalmessage_models import InternalMessage, InternalMessageDisplay, \
    InternalMessageReadReceipt
from timApp.note.usernote import UserNote
from timApp.notification.notification import Notification
from timApp.notification.pending_notification import PendingNotification, DocumentNotification, CommentNotification
from timApp.plugin.timtable.row_owner_info import RowOwnerInfo
from timApp.printing.printeddoc import PrintedDoc
from timApp.readmark.readparagraph import ReadParagraph
from timApp.peerreview.peerreview import PeerReview
from timApp.sisu.scimusergroup import ScimUserGroup
from timApp.slide.slidestatus import SlideStatus
from timApp.timdb.sqa import db
from timApp.user.consentchange import ConsentChange
from timApp.user.hakaorganization import HakaOrganization
from timApp.user.newuser import NewUser
from timApp.user.personaluniquecode import PersonalUniqueCode
from timApp.user.user import User
from timApp.user.usergroup import UserGroup
from timApp.user.usergroupdoc import UserGroupDoc
from timApp.user.usergroupmember import UserGroupMember
from timApp.util.flask.filters import map_format, timdate, humanize_timedelta, humanize_datetime
from timApp.util.logger import setup_logging
from timApp.util.utils import datestr_to_relative, date_to_relative
from timApp.velp.annotation_model import Annotation
from timApp.velp.velp_models import Velp, VelpContent, VelpGroup, VelpGroupDefaults, VelpGroupLabel, \
    VelpGroupSelection, VelpGroupsInDocument, VelpInGroup, VelpLabel, VelpLabelContent, VelpVersion, \
    LabelInVelp, AnnotationComment


# All SQLAlchemy models must be imported in this module.
all_models = (
    AccessTypeModel,
    Annotation,
    AnnotationComment,
    Answer,
    AnswerSaver,
    AnswerTag,
    AnswerUpload,
    AskedJson,
    AskedQuestion,
    Block,
    BlockAccess,
    BlockAssociation,
    BlockRelevance,
    CommentNotification,
    ConsentChange,
    DocEntry,
    DocumentNotification,
    Folder,
    HakaOrganization,
    InternalMessage,
```

```
    InternalMessageReadReceipt,
    InternalMessageDisplay,
    LabelInVelp,
    Lecture,
    LectureAnswer,
    LectureUsers,
    Message,
    MessageListDistribution,
    MessageListMember,
    MessageListModel,
    MessageListTimMember,
    MessageListExternalMember,
    NewUser,
    Notification,
    PeerReview,
    PendingNotification,
    PersonalUniqueCode,
    PrintedDoc,
    Question,
    QuestionActivity,
    ReadParagraph,
    RowOwnerInfo,
    Runningquestion,
    ScimUserGroup,
    Showpoints,
    SlideStatus,
    Tag,
    Translation,
    User,
    Useractivity,
    UserAnswer,
    UserGroup,
    UserGroupDoc,
    UserGroupMember,
    UserNote,
    Velp,
    VelpContent,
    VelpGroup,
    VelpGroupDefaults,
    VelpGroupLabel,
    VelpGroupSelection,
    VelpGroupsInDocument,
    VelpInGroup,
    VelpLabel,
    VelpLabelContent,
    VelpVersion,

    CrontabSchedule,
    IntervalSchedule,
    PeriodicTask,
    PeriodicTaskChanged,
    SolarSchedule,
)

sys.setrecursionlimit(10000)
app = Flask(__name__)

# The autoescape setting needs to be forced because the template file extension used ↩
    in TIM is jinja2.
```

```
154 # The more accurate file extension helps IDEs recognize the file type better.
155 app.jinja_env.autoescape = True
156
157 app.jinja_env.trim_blocks = True
158 app.jinja_env.lstrip_blocks = True
159 app.config.from_pyfile('defaultconfig.py', silent=False)
160 app.config.from_envvar('TIM_SETTINGS', silent=True)
161 app.config.from_json('hosts.json', silent=True)
162 setup_logging(app)
163
164 # Compress(app)
165
166 # Disabling object expiration on commit makes testing easier
167 # because sometimes objects would expire after calling a route.
168 if app.config['TESTING']:
169     db.session = db.create_scoped_session({'expire_on_commit': False})
170
171 db.init_app(app)
172 db.app = app
173 migrate = Migrate(app, db)
174
175 csrf = CSRFProtect(app)
176
177 app.jinja_env.filters['map_format'] = map_format
178 app.jinja_env.filters['datestr_to_relative'] = datestr_to_relative
179 app.jinja_env.filters['date_to_relative'] = date_to_relative
180 app.jinja_env.filters['timdate'] = timdate
181 app.jinja_env.filters['timtimedelta'] = humanize_timedelta
182 app.jinja_env.filters['timreldatetime'] = humanize_datetime
183 app.jinja_env.add_extension('jinja2.ext.do')
184
185 mimetypes.add_type('text/plain', '.scss')
186
187 app.json_encoder = TimJsonEncoder
188
189 # Caddy sets the following headers:
190 # X-Forwarded-For: <ip>
191 # X-Forwarded-Proto: <http/https>
192 # In prod_multi, there are 2 Caddys - the one in the container and the one in the ↩
      host machine.
193 num_proxies = 2 if os.environ.get('COMPOSE_PROFILES') == 'prod_multi' else 1
194 app.wsgi_app = ProxyFix(
195     app.wsgi_app,
196     x_for=num_proxies,
197     x_proto=1,
198     x_host=0,
199     x_port=0,
200     x_prefix=0,
201 )
202
203
204 @app.shell_context_processor
205 def make_shell_context():
206     ctx = {
207         c.__name__: c for c in all_models
208     }
209     ctx['db'] = db
210     ctx['func'] = func
211     return ctx
```

```
212
213
214 def print_schema(bind: Optional[str] = None):
215     """Prints the database schema generated by the models.
216
217     :param bind: The bind to use.
218
219     """
220     models = inspect.getmembers(sys.modules[__name__], lambda x: inspect.isclass(x) ↩
        and hasattr(x, '__table__'))
221     eng = db.get_engine(app, bind)
222
223     for _, model_class in models:
224         print(CreateTable(model_class.__table__).compile(eng), end=';')
225     print()
226     sys.stdout.flush()
227
228
229 # print_schema()
230
231
232 def get_home_organization_group() -> UserGroup:
233     return UserGroup.get_organization_group(app.config['HOME_ORGANIZATION'])
```

timApp/user/user.py

```
1  import json
2  from dataclasses import dataclass, field
3  from datetime import datetime, timedelta, timezone
4  from enum import Enum
5  from typing import List, Tuple, Dict
6  from typing import Optional, Union, Set
7
8  from sqlalchemy import func
9  from sqlalchemy.orm import Query, joinedload, defaultload
10 from sqlalchemy.orm.collections import attribute_mapped_collection
11
12 from timApp.answer.answer import Answer
13 from timApp.answer.answer_models import UserAnswer
14 from timApp.auth.accesstype import AccessType
15 from timApp.auth.auth_models import BlockAccess
16 from timApp.document.docinfo import DocInfo
17 from timApp.document.timjsonencoder import TimJsonEncoder
18 from timApp.folder.createopts import FolderCreationOptions
19 from timApp.folder.folder import Folder
20 from timApp.item.block import Block
21 from timApp.item.item import ItemBase
22 from timApp.lecture.lectureusers import LectureUsers
23 from timApp.messaging.timMessage.internalmessage_models import ↩
       InternalMessageReadReceipt
24 from timApp.notification.notification import Notification
25 from timApp.timdb.exceptions import TimDbException
26 from timApp.timdb.sqa import db, TimeStampMixin, is_attribute_loaded
27 from timApp.user.hakaorganization import HakaOrganization, get_home_organization_id
28 from timApp.user.personaluniquecode import SchacPersonalUniqueCode, PersonalUniqueCode
29 from timApp.user.preferences import Preferences
30 from timApp.user.scimentity import SCIMEntity
31 from timApp.user.settings.theme import Theme
32 from timApp.user.special_group_names import ANONYMOUS_GROUPNAME, ANONYMOUS_USERNAME, ↩
```

```
    LOGGED_IN_GROUPNAME, \
33      SPECIAL_USERNAMES
34  from timApp.user.usergroup import UserGroup, get_logged_in_group_id, ↩
        get_anonymous_group_id
35  from timApp.user.usergroupmember import UserGroupMember, membership_current, ↩
        membership_deleted
36  from timApp.user.userutils import grant_access, get_access_type_id, \
37      create_password_hash, check_password_hash, check_password_hash_old
38  from timApp.util.utils import remove_path_special_chars, cached_property, ↩
        get_current_time
39
40  ItemOrBlock = Union[ItemBase, Block]
41  maxdate = datetime.max.replace(tzinfo=timezone.utc)
42
43  view_access_set = {t.value for t in [
44      AccessType.view,
45      AccessType.copy,
46      AccessType.edit,
47      AccessType.owner,
48      AccessType.teacher,
49      AccessType.see_answers,
50      AccessType.manage,
51  ]}
52
53  edit_access_set = {t.value for t in [
54      AccessType.edit,
55      AccessType.owner,
56      AccessType.manage,
57  ]}
58
59  manage_access_set = {t.value for t in [
60      AccessType.owner,
61      AccessType.manage,
62  ]}
63
64  owner_access_set = {t.value for t in [
65      AccessType.owner,
66  ]}
67
68  teacher_access_set = {t.value for t in [
69      AccessType.owner,
70      AccessType.manage,
71      AccessType.teacher,
72  ]}
73
74  seeanswers_access_set = {t.value for t in [
75      AccessType.owner,
76      AccessType.teacher,
77      AccessType.see_answers,
78      AccessType.manage,
79  ]}
80
81  copy_access_set = {t.value for t in [
82      AccessType.copy,
83      AccessType.edit,
84      AccessType.owner,
85      AccessType.manage,
86  ]}
87
```

```
access_sets = {
    AccessType.copy: copy_access_set,
    AccessType.edit: edit_access_set,
    AccessType.manage: manage_access_set,
    AccessType.owner: owner_access_set,
    AccessType.see_answers: seeanswers_access_set,
    AccessType.teacher: teacher_access_set,
    AccessType.view: view_access_set,
}

SCIM_USER_NAME = ':scimuser'


class Consent(Enum):
    CookieOnly = 1
    CookieAndData = 2


class UserOrigin(Enum):
    """Indicates how the user originally registered to TIM.

    Only Email, Korppi and Sisu are used so far; the others are speculative.
    """
    Email = 1
    Korppi = 2
    Sisu = 3
    Haka = 4
    OpenID = 5
    OpenIDConnect = 6
    Facebook = 7
    Google = 8
    Twitter = 9


@dataclass
class UserInfo:
    username: Optional[str] = None
    email: Optional[str] = None
    full_name: Optional[str] = None
    given_name: Optional[str] = None
    last_name: Optional[str] = None
    origin: Optional[UserOrigin] = None
    password: Optional[str] = None
    password_hash: Optional[str] = None
    unique_codes: List[SchacPersonalUniqueCode] = field(default_factory=list)

    def __post_init__(self):
        assert self.password is None or self.password_hash is None, 'Cannot pass both ↩
    password and password_hash to UserInfo'


def last_name_to_first(full_name: Optional[str]):
    """Converts a name of the form "Firstname Middlenames Lastname" to "Lastname ↩
    Firstname Middlenames".
    """
    if full_name is None:
        return None
    names = full_name.split(' ')
    if len(names) > 1:
```

```
        return f'{names[-1]} {" ".join(names[:-1])}'
    return full_name


def last_name_to_last(full_name: Optional[str]):
    """Converts a name of the form "Lastname Firstname Middlenames" to "Firstname Middlenames Lastname".
    """
    if full_name is None:
        return None
    names = full_name.split(' ')
    if len(names) > 1:
        return f'{" ".join(names[1:])} {names[0]}'
    return full_name


deleted_user_suffix = '_deleted'


def user_query_with_joined_groups() -> Query:
    return User.query.options(joinedload(User.groups))


class User(db.Model, TimeStampMixin, SCIMEntity):
    """A user account.

    A special user 'Anonymous user' denotes a user that is not logged in. Its id is 0.
    """
    __tablename__ = 'useraccount'
    id = db.Column(db.Integer, primary_key=True)
    """User identifier."""

    name = db.Column(db.Text, nullable=False, unique=True)
    """User name (not full name)."""

    given_name = db.Column(db.Text)
    last_name = db.Column(db.Text)

    real_name = db.Column(db.Text)
    """Real (full) name. This may be in the form "Lastname Firstname" or "Firstname Lastname"."""

    email = db.Column(db.Text, unique=True)
    """Email address."""

    prefs = db.Column(db.Text)
    """Preferences as a JSON string."""

    pass_ = db.Column('pass', db.Text)
    """Password hashed with bcrypt."""

    consent = db.Column(db.Enum(Consent), nullable=True)
    """Current consent for cookie/data collection."""

    origin = db.Column(db.Enum(UserOrigin), nullable=True)
    """How the user registered to TIM."""

    uniquecodes = db.relationship(
        'PersonalUniqueCode',
```

```
        back_populates='user',
        collection_class=attribute_mapped_collection('user_collection_key'),
    )

    internalmessage_readreceipt: Optional[InternalMessageReadReceipt] = db.relationship('InternalMessageReadReceipt',
      back_populates='user')

    @property
    def scim_display_name(self):
        return last_name_to_last(self.real_name)

    @property
    def scim_created(self):
        return self.created

    @property
    def scim_modified(self):
        return self.modified

    @property
    def scim_id(self):
        return self.name

    @property
    def scim_resource_type(self):
        return 'User'

    @property
    def scim_extra_data(self):
        return {'emails': [{'value': self.email}] if self.email else []}

    consents = db.relationship('ConsentChange', back_populates='user', lazy='select')
    notifications = db.relationship('Notification', back_populates='user', lazy='dynamic')
    notifications_alt = db.relationship('Notification')

    groups: List[UserGroup] = db.relationship(
        UserGroup,
        UserGroupMember.__table__,
        primaryjoin=(id == UserGroupMember.user_id) & membership_current,
        back_populates='users',
        lazy='select',
    )
    groups_dyn = db.relationship(
        UserGroup,
        UserGroupMember.__table__,
        primaryjoin=id == UserGroupMember.user_id,
        lazy='dynamic',
    )
    groups_inactive = db.relationship(
        UserGroup,
        UserGroupMember.__table__,
        primaryjoin=(id == UserGroupMember.user_id) & membership_deleted,
        lazy='dynamic',
    )
    memberships_dyn = db.relationship(
        UserGroupMember,
```

```
        foreign_keys="UserGroupMember.user_id",
        lazy='dynamic',
    )
    memberships: List[UserGroupMember] = db.relationship(
        UserGroupMember,
        foreign_keys="UserGroupMember.user_id",
    )
    active_memberships = db.relationship(
        UserGroupMember,
        primaryjoin=(id == UserGroupMember.user_id) & membership_current,
        collection_class=attribute_mapped_collection("UserGroupMember.usergroup_id"),
        # back_populates="group",
    )
    lectures = db.relationship('Lecture', secondary=LectureUsers.__table__,
                               back_populates='users', lazy='select')
    owned_lectures = db.relationship('Lecture', back_populates='owner', lazy='dynamic')
    owned_lectures_alt = db.relationship('Lecture')
    lectureanswers = db.relationship('LectureAnswer', back_populates='user', ↩
    lazy='dynamic')
    lectureanswers_alt = db.relationship('LectureAnswer')
    messages = db.relationship('Message', back_populates='user', lazy='dynamic')
    messages_alt = db.relationship('Message')
    questionactivity = db.relationship('QuestionActivity', back_populates='user', ↩
    lazy='select')
    useractivity = db.relationship('Useractivity', back_populates='user', lazy='select')
    answers = db.relationship('Answer', secondary=UserAnswer.__table__,
                              back_populates='users', lazy='dynamic')
    answers_alt = db.relationship('Answer', secondary=UserAnswer.__table__)
    annotations = db.relationship('Annotation', back_populates='annotator', ↩
    lazy='dynamic')
    annotations_alt = db.relationship('Annotation')
    velps = db.relationship('Velp', back_populates='creator', lazy='dynamic')
    velps_alt = db.relationship('Velp')

    def __repr__(self):
        return f'<User(id={self.id}, name={self.name}, email={self.email}, ↩
    real_name={self.real_name})>'

    @property
    def logged_in(self):
        return self.id > 0

    @property
    def is_deleted(self):
        return self.name.endswith(deleted_user_suffix) and ↩
    self.email.endswith(deleted_user_suffix)

    @property
    def group_ids(self):
        return set(g.id for g in self.groups)

    @cached_property
    def is_admin(self):
        for g in self.groups:
            if g.name == 'Administrators':
                return True
        return False

    @property
```

```
    def is_email_user(self):
        """Returns whether the user signed up via email and has not been "upgraded" to Korppi or Sisu user."""
        return '@' in self.name or self.name.startswith('testuser')

    @property
    def pretty_full_name(self):
        """Returns the user's full name."""
        if self.is_name_hidden:
            return f'User {self.id}'
        if self.given_name and self.last_name:
            return f'{self.given_name} {self.last_name}'
        if self.real_name is None:
            return '(real_name is null)'
        parts = self.real_name.split(' ')
        if len(parts) == 1:
            return self.real_name
        return ' '.join(parts[1:]) + ' ' + parts[0]

    @staticmethod
    def create_with_group(
            info: UserInfo,
            is_admin: bool = False,
            uid: Optional[int] = None,
    ) -> Tuple['User', UserGroup]:
        p_hash = create_password_hash(info.password) if info.password is not None else info.password_hash
        user = User(
            id=uid,
            name=info.username,
            real_name=info.full_name,
            last_name=info.last_name,
            given_name=info.given_name,
            email=info.email,
            pass_=p_hash,
            origin=info.origin,
        )
        db.session.add(user)
        user.set_unique_codes(info.unique_codes)
        group = UserGroup.create(info.username)
        user.groups.append(group)
        if is_admin:
            user.make_admin()
        return user, group

    @staticmethod
    def get_by_name(name: str) -> Optional['User']:
        return user_query_with_joined_groups().filter_by(name=name).first()

    @staticmethod
    def get_by_id(uid: int) -> Optional['User']:
        return user_query_with_joined_groups().get(uid)

    @staticmethod
    def get_by_email(email: str) -> Optional['User']:
        if email is None:
            raise Exception('Tried to find an user by null email')
        return user_query_with_joined_groups().filter_by(email=email).first()

```

```
369     @staticmethod
370     def get_by_email_case_insensitive(email: str) -> List['User']:
371         return ↩
        user_query_with_joined_groups().filter(func.lower(User.email).in_([email])).all()
372
373     @staticmethod
374     def get_by_email_case_insensitive_or_username(email_or_username: str) -> ↩
        List['User']:
375         users = User.get_by_email_case_insensitive(email_or_username)
376         if users:
377             return users
378         u = User.get_by_name(email_or_username)
379         if u:
380             return [u]
381         return []
382
383     @property
384     def email_name_part(self):
385         parts = self.email.split('@')
386         return parts[0]
387
388     @property
389     def is_special(self):
390         return self.name in SPECIAL_USERNAMES
391
392     def check_password(self, password: str, allow_old=False, update_if_old=True) -> ↩
        bool:
393         if not self.pass_:
394             return False
395         is_ok = check_password_hash(password, self.pass_)
396         if is_ok:
397             return True
398         if not allow_old:
399             return False
400         is_ok = check_password_hash_old(password, self.pass_)
401         if is_ok and update_if_old:
402             self.pass_ = create_password_hash(password)
403         return is_ok
404
405     def make_admin(self):
406         ag = UserGroup.get_admin_group()
407         if ag not in self.groups:
408             self.groups.append(ag)
409
410     def get_home_org_student_id(self):
411         home_org_id = get_home_organization_id()
412         puc = self.uniquecodes.get((home_org_id, 'studentID'), None)
413         return puc.code if puc is not None else None
414
415     def get_personal_group(self) -> UserGroup:
416         return self.personal_group_prop
417
418     @cached_property
419     def personal_group_prop(self) -> UserGroup:
420         group_to_find = self.name
421         if self.name == ANONYMOUS_USERNAME:
422             group_to_find = ANONYMOUS_GROUPNAME
423         for g in self.groups:
424             if g.name == group_to_find:
```

```
                return g
        raise TimDbException(f'Personal usergroup for user {self.name} was not found!')

    def derive_personal_folder_name(self):
        real_name = self.real_name
        if not real_name:
            real_name = "anonymous"
        basename = remove_path_special_chars(real_name).lower()
        index = ''
        while Folder.find_by_path('users/' + basename + index):
            index = str(int(index or 1) + 1)
        return basename + index

    def get_personal_folder(self) -> Folder:
        return self.personal_folder_prop

    @cached_property
    def personal_folder_prop(self) -> Folder:
        if self.logged_in:
            group_condition = UserGroup.name == self.name
        else:
            group_condition = UserGroup.name == ANONYMOUS_GROUPNAME
        folders: List[Folder] = Folder.query.join(
            BlockAccess, BlockAccess.block_id == Folder.id
        ).join(
            UserGroup, UserGroup.id == BlockAccess.usergroup_id
        ).filter(
            (Folder.location == 'users') &
            group_condition &
            (BlockAccess.type == AccessType.owner.value)
        ).with_entities(Folder).options(
            defaultload(Folder._block)
                .joinedload(Block.accesses)
                .joinedload(BlockAccess.usergroup)
        ).all()
        if len(folders) >= 2:
            raise TimDbException(f'Found multiple personal folders for user ↩
    {self.name}: {[f.name for f in folders]}')
        if not folders:
            f = Folder.create('users/' + self.derive_personal_folder_name(),
                              self.get_personal_group(),
                              title=f"{self.real_name}",
                              ↩
    creation_opts=FolderCreationOptions(apply_default_rights=True))
            db.session.commit()
            return f
        return folders[0]

    def get_prefs(self) -> Preferences:
        prefs = json.loads(self.prefs or '{}')
        try:
            return Preferences.from_json(prefs)
        except TypeError:
            return Preferences()

    def set_prefs(self, prefs: Preferences):
        css_files = prefs.css_files
        existing_css_files = {}
        for k, v in css_files.items():
```

```
482             t = Theme(k)
483             if t.exists() and v:
484                 existing_css_files[t.filename] = True
485         prefs.css_files = existing_css_files
486         self.prefs = json.dumps(prefs, cls=TimJsonEncoder)
487 
488     def get_groups(self, include_special=True) -> Query:
489         special_groups = [ANONYMOUS_GROUPNAME]
490         if self.logged_in:
491             special_groups.append(LOGGED_IN_GROUPNAME)
492         q = UserGroup.query.filter(
493             ↩
    UserGroup.id.in_(db.session.query(UserGroupMember.usergroup_id).filter_by(user_id=self.id))
494         )
495         if include_special:
496             q = q.union(
497                 UserGroup.query.filter(UserGroup.name.in_(special_groups))
498             )
499         return q
500 
501     def add_to_group(self, ug: UserGroup, added_by: Optional['User']) -> bool:
502         # Avoid cyclical importing.
503         from timApp.messaging.messagelist.messagelist_utils import ↩
    sync_message_list_on_add
504         existing: UserGroupMember = self.id is not None and ↩
    self.memberships_dyn.filter_by(group=ug).first()
505         if existing:
506             existing.membership_end = None
507             existing.adder = added_by
508             new_add = False
509         else:
510             self.memberships.append(UserGroupMember(group=ug, adder=added_by))
511             new_add = True
512         # On changing of group, sync this person to the user goup's message lists.
513         sync_message_list_on_add(self, ug)
514         return new_add
515 
516     @staticmethod
517     def get_scimuser() -> 'User':
518         u = User.get_by_name(SCIM_USER_NAME)
519         if not u:
520             u, _ = User.create_with_group(UserInfo(
521                 username=SCIM_USER_NAME,
522                 full_name='Scim User',
523                 email='scimuser@example.com',
524             ))
525         return u
526 
527     @staticmethod
528     def get_anon() -> 'User':
529         return User.get_by_id(0)
530 
531     def update_info(
532             self,
533             info: UserInfo,
534     ):
535         if info.username and self.name != info.username:
536             group = self.get_personal_group()
537             self.name = info.username
```

```
            group.name = info.username
        if info.given_name:
            self.given_name = info.given_name
        if info.last_name:
            self.last_name = info.last_name
        if info.full_name:
            self.real_name = info.full_name
        if info.email:
            self.email = info.email
        if info.password:
            self.pass_ = create_password_hash(info.password)
        elif info.password_hash:
            self.pass_ = info.password_hash
        self.set_unique_codes(info.unique_codes)

    def set_unique_codes(self, codes: List[SchacPersonalUniqueCode]):
        for c in codes:
            ho = HakaOrganization.get_or_create(name=c.org)
            if ho.id is None or self.id is None:
                db.session.flush()
            puc = PersonalUniqueCode(
                code=c.code,
                type=c.codetype,
                org_id=ho.id,
            )
            if puc.user_collection_key not in self.uniquecodes:
                self.uniquecodes[puc.user_collection_key] = puc

    def has_some_access(
            self,
            i: ItemOrBlock,
            vals: Set[int],
            allow_admin: bool = True,
            grace_period: timedelta = timedelta(seconds=0),
    ) -> Optional[BlockAccess]:
        if allow_admin and self.is_admin:
            return BlockAccess(block_id=i.id,
                               ↩
    accessible_from=datetime.min.replace(tzinfo=timezone.utc),
                               type=AccessType.owner.value,
                               usergroup_id=self.get_personal_group().id)
        if isinstance(i, ItemBase):
            b = i.block
        else:
            b = i
        if not b:
            return None
        now = get_current_time()
        best_access = None
        curr_group_ids = self.group_ids
        for a in b.accesses.values():  # type: BlockAccess
            if a.usergroup_id not in curr_group_ids:
                if self.logged_in and a.usergroup_id == get_logged_in_group_id():
                    pass
                elif a.usergroup_id == get_anonymous_group_id():
                    pass
                else:
                    continue
            if a.type not in vals:
```

```
596                 continue
597             to_time = a.accessible_to
598             if to_time is not None:
599                 to_time += grace_period
600             if (a.accessible_from or maxdate) <= now < (to_time or maxdate):
601                 # If the end time of the access is unrestricted, there is no better ↩
    access.
602                 if to_time is None:
603                     return a
604                 # If the end time of the access is restricted, there might be a ↩
    better access,
605                 # so we'll continue looping.
606                 if best_access is None or best_access.accessible_to < a.accessible_to:
607                     best_access = a
608         return best_access
609
610     def has_access(
611             self,
612             i: ItemOrBlock,
613             access: AccessType,
614             grace_period=timedelta(seconds=0),
615     ) -> Optional[BlockAccess]:
616         from timApp.auth.accesshelper import check_inherited_right
617         return check_inherited_right(self, i, access, grace_period) or ↩
    self.has_some_access(i, access_sets[access],
618                                                                                 ↩
          grace_period=grace_period)
619
620     def has_view_access(self, i: ItemOrBlock) -> Optional[BlockAccess]:
621         return self.has_some_access(i, view_access_set)
622
623     def has_edit_access(self, i: ItemOrBlock) -> Optional[BlockAccess]:
624         return self.has_some_access(i, edit_access_set)
625
626     def has_manage_access(self, i: ItemOrBlock) -> Optional[BlockAccess]:
627         return self.has_some_access(i, manage_access_set)
628
629     def has_teacher_access(self, i: ItemOrBlock) -> Optional[BlockAccess]:
630         return self.has_some_access(i, teacher_access_set)
631
632     def has_seeanswers_access(self, i: ItemOrBlock) -> Optional[BlockAccess]:
633         return self.has_some_access(i, seeanswers_access_set)
634
635     def has_copy_access(self, i: ItemOrBlock) -> Optional[BlockAccess]:
636         return self.has_some_access(i, copy_access_set)
637
638     def has_ownership(self, i: ItemOrBlock, allow_admin: bool = True) -> ↩
    Optional[BlockAccess]:
639         return self.has_some_access(i, owner_access_set, allow_admin)
640
641     def can_write_to_folder(self, f: Folder):
642         # not even admins are allowed to create new items in 'users' folder
643         if f.path == 'users':
644             return False
645         return self.has_edit_access(f)
646
647     def grant_access(self, block: ItemOrBlock,
648                      access_type: AccessType,
649                      accessible_from: Optional[datetime] = None,
```

```
                    accessible_to: Optional[datetime] = None,
                    duration_from: Optional[datetime] = None,
                    duration_to: Optional[datetime] = None,
                    duration: Optional[timedelta] = None,
                    require_confirm: Optional[bool] = None):
        return grant_access(group=self.get_personal_group(),
                            block=block,
                            access_type=access_type,
                            accessible_from=accessible_from,
                            accessible_to=accessible_to,
                            duration_from=duration_from,
                            duration_to=duration_to,
                            duration=duration,
                            require_confirm=require_confirm)

    def remove_access(self, block_id: int, access_type: str):
        BlockAccess.query.filter_by(block_id=block_id,
                                    usergroup_id=self.get_personal_group().id,
                                    type=get_access_type_id(access_type)).delete()

    def get_notify_settings(self, doc: DocInfo):
        n = self.notifications.filter_by(doc_id=doc.id).first()
        if not n:
            n = Notification(doc_id=doc.id,
                             user_id=self.id,
                             email_doc_modify=False,
                             email_comment_add=False,
                             email_comment_modify=False
                             )
            db.session.add(n)
        return n

    def set_notify_settings(self, doc: DocInfo, doc_modify: bool, comment_add: bool, ↩
    comment_modify: bool):
        n = self.get_notify_settings(doc)
        n.email_comment_add = comment_add
        n.email_doc_modify = doc_modify
        n.email_comment_modify = comment_modify
        if not any((doc_modify, comment_add, comment_modify)):
            db.session.delete(n)

    def get_answers_for_task(self, task_id: str):
        return ↩
    self.answers.options(joinedload(Answer.users_all)).order_by(Answer.id.desc()).filter_by(task_id

    @property
    def is_name_hidden(self):
        return getattr(self, 'hide_name', False)

    @property
    def basic_info_dict(self):
        if not self.is_name_hidden:
            info_dict = {
                'id': self.id,
                'name': self.name,
                'real_name': self.real_name,
                'email': self.email,
            }
        else:
```

```
707             info_dict = {
708                 'id': self.id,
709                 'name': f'user{self.id}',
710                 'real_name': f'User {self.id}',
711                 'email': f'user{self.id}@example.com',
712             }
713 
714         if is_attribute_loaded("uniquecodes", self):
715             info_dict['student_id'] = self.get_home_org_student_id()
716 
717         return info_dict
718 
719     def to_json(self, full: bool=False) -> Dict:
720         return {**self.basic_info_dict,
721                 'group': self.get_personal_group(),
722                 'groups': self.groups,
723                 'folder': self.get_personal_folder() if self.logged_in else None,
724                 'consent': self.consent,
725                 'last_name': self.last_name,
726                 } if full else self.basic_info_dict
727 
728     @cached_property
729     def bookmarks(self):
730         from timApp.bookmark.bookmarks import Bookmarks
731         return Bookmarks(self)
732 
733     def belongs_to_any_of(self, *groups: UserGroup):
734         return bool(set(groups) & set(self.groups))
735 
736 
737 def get_membership_end(u: User, group_ids: Set[int]):
738     relevant_memberships: List[UserGroupMember] = [m for m in u.memberships if ↩
    m.usergroup_id in group_ids]
739     membership_end = None
740     # If the user is not active in any of the groups, we'll show the lastly-ended ↩
    membership.
741     # TODO: It might be possible in the future that the membership_end is in the future.
742     if relevant_memberships and all(m.membership_end is not None for m in ↩
    relevant_memberships):
743         membership_end = (
744             max(m.membership_end for m in relevant_memberships)
745         )
746     return membership_end
747 
748 
749 def check_rights(hide_type: str, rights: dict):
750     """
751     Checks whether the user has the correct rights rights not to hide links or the ↩
    buttons in the top of the
752     page from them.
753 
754     :param hide_type What elements to hide in the document.
755     :param rights Which user roles the elements should be hidden from.
756     :return Should the elements be hidden from the user.
757     """
758     return {'view': not rights['editable'] and not rights['see_answers'],
759             'edit': not rights['see_answers'],
760             'see_answers': not rights['teacher'],
761             'teacher': not rights['manage']}.get(hide_type, False)
```

```
762
763
764 def get_owned_objects_query(u: User):
765     return ↩
      u.get_personal_group().accesses.filter_by(type=AccessType.owner.value).with_entities(
766         BlockAccess.block_id)
```

timApp/user/usergroup.py

```
1  from __future__ import annotations
2
3  from functools import lru_cache
4  from typing import List, Dict, Tuple, TYPE_CHECKING, Optional
5
6  import attr
7  from sqlalchemy.orm import joinedload
8  from sqlalchemy.orm.collections import attribute_mapped_collection
9
10 from timApp.auth.auth_models import BlockAccess
11 from timApp.messaging.messagelist.messagelist_models import MessageListTimMember
12 from timApp.messaging.timMessage.internalmessage_models import ↩
       InternalMessageDisplay, InternalMessageReadReceipt
13 from timApp.sisu.parse_display_name import parse_sisu_group_display_name
14 from timApp.sisu.scimusergroup import ScimUserGroup
15 from timApp.timdb.sqa import db, TimeStampMixin, include_if_exists, is_attribute_loaded
16 from timApp.user.scimentity import SCIMEntity
17 from timApp.user.special_group_names import ANONYMOUS_GROUPNAME, LOGGED_IN_GROUPNAME, \
18     ADMIN_GROUPNAME, GROUPADMIN_GROUPNAME, TEACHERS_GROUPNAME, SPECIAL_GROUPS, ↩
       FUNCTIONSCHEDULER_GROUPNAME
19 from timApp.user.usergroupdoc import UserGroupDoc
20 from timApp.user.usergroupmember import UserGroupMember, membership_current
21
22 if TYPE_CHECKING:
23     from timApp.item.block import Block
24
25 # Prefix is no longer needed because scimusergroup determines the Sisu (SCIM) groups.
26 SISU_GROUP_PREFIX = ''
27
28
29 def tim_group_to_scim(tim_group: str) -> str:
30     if not tim_group.startswith(SISU_GROUP_PREFIX):
31         raise Exception(f"Group {tim_group} is not a Sisu group")
32     return tim_group[len(SISU_GROUP_PREFIX):]
33
34
35 ORG_GROUP_SUFFIX = ' users'
36
37
38 class UserGroup(db.Model, TimeStampMixin, SCIMEntity):
39     """A usergroup. Each User should belong to a personal UserGroup that has the same ↩
       name as the User name. No one
40     else should belong to a personal UserGroup.
41
42     A User can additionally belong to any number of other UserGroups.
43
44     Two special groups named 'Logged-in users' and 'Anonymous users' denote the set ↩
       of all logged-in users and all
45     users including anonymous (not logged-in) ones, respectively.
46
```

```
    In database, the User 'Anonymous user' belongs to 'Anonymous users' group. Other than that,
    the two groups are empty from the database's point of view.
    """
    __tablename__ = 'usergroup'
    id = db.Column(db.Integer, primary_key=True)
    """Usergroup identifier."""

    name = db.Column(db.Text, nullable=False, unique=True)
    """Usergroup name (textual identifier)."""

    display_name = db.Column(db.Text, nullable=True)
    """Usergroup display name. Currently only used for storing certain Sisu course properties:
     - course code
     - period (P1...P5)
     - date range
     - group description in Sisu
    """

    @property
    def scim_display_name(self):
        return self.display_name

    users = db.relationship(
        'User',
        UserGroupMember.__table__,
        primaryjoin=(id == UserGroupMember.usergroup_id) & membership_current,
        secondaryjoin="UserGroupMember.user_id == User.id",
        back_populates="groups",
    )
    memberships = db.relationship(
        UserGroupMember,
        back_populates="group",
        lazy='dynamic',
    )
    memberships_sel = db.relationship(
        UserGroupMember,
        back_populates="group",
        cascade='all, delete-orphan',
    )
    current_memberships = db.relationship(
        UserGroupMember,
        primaryjoin=(id == UserGroupMember.usergroup_id) & membership_current,
        collection_class=attribute_mapped_collection("user_id"),
        back_populates="group",
    )
    accesses = db.relationship(
        'BlockAccess',
        back_populates='usergroup',
        lazy='dynamic',
    )
    accesses_alt: Dict[Tuple[int, int], BlockAccess] = db.relationship(
        'BlockAccess',
        collection_class=attribute_mapped_collection('group_collection_key'),
        cascade='all, delete-orphan',
    )
    readparagraphs = db.relationship('ReadParagraph', back_populates='usergroup', lazy='dynamic')
```

```
    readparagraphs_alt = db.relationship('ReadParagraph')
    notes = db.relationship('UserNote', back_populates='usergroup', lazy='dynamic')
    notes_alt = db.relationship('UserNote')

    admin_doc: Block = db.relationship(
        'Block',
        secondary=UserGroupDoc.__table__,
        lazy='select',
        uselist=False,
    )

    # For groups created from SCIM API
    external_id: ScimUserGroup = db.relationship('ScimUserGroup', lazy='select', uselist=False)

    messagelist_membership: MessageListTimMember = db.relationship("MessageListTimMember", back_populates="user_group")

    internalmessage_display: Optional[InternalMessageDisplay] = db.relationship('InternalMessageDisplay',
                                                                                back_populates='usergroup')
    internalmessage_readreceipt: Optional[InternalMessageReadReceipt] = db.relationship('InternalMessageReadReceipt',
                                                                                        back_populates='recipient')

    def __repr__(self):
        return f'<UserGroup(id={self.id}, name={self.name})>'

    @property
    def scim_created(self):
        return self.created

    @property
    def scim_modified(self):
        return self.modified

    @property
    def scim_id(self):
        return self.external_id.external_id if self.external_id else None

    @property
    def scim_resource_type(self):
        return 'Group'

    def is_anonymous(self) -> bool:
        return self.name == ANONYMOUS_GROUPNAME

    def is_large(self) -> bool:
        return self.name.endswith(ORG_GROUP_SUFFIX)

    def load_personal_user(self):
        """If this is a personal usergroup, loads the user object to personal_user attribute."""
        from timApp.user.user import User
        self.personal_user = User.get_by_name(self.name)

    def to_json(self):
```

```
155         r = {
156             'id': self.id,
157             'name': self.name,
158             **include_if_exists('personal_user', self),
159         }
160         if is_attribute_loaded('admin_doc', self) and self.admin_doc and ↩
    self.admin_doc.docentries:
161             r['admin_doc_path'] = self.admin_doc.docentries[0].path
162         return r
163 
164     @property
165     def is_personal_group(self):
166         self.load_personal_user()
167         return self.personal_user is not None
168 
169     @property
170     def pretty_full_name(self):
171         return self.name
172 
173     @property
174     def is_sisu(self):
175         return self.external_id is not None
176 
177     @property
178     def is_sisu_student_group(self):
179         return self.is_sisu and self.external_id.external_id.endswith('-students')
180 
181     @staticmethod
182     def create(name: str) -> 'UserGroup':
183         """Creates a new user group.
184 
185         :param name: The name of the user group.
186         :returns: The id of the created user group.
187 
188         """
189 
190         ug = UserGroup(name=name)
191         db.session.add(ug)
192         return ug
193 
194     @staticmethod
195     def get_by_external_id(name: str) -> 'UserGroup':
196         r = get_sisu_groups_by_filter(ScimUserGroup.external_id == name)
197         return r[0] if r else None
198 
199     @staticmethod
200     def get_by_name(name) -> 'UserGroup':
201         return UserGroup.query.filter_by(name=name).first()
202 
203     @staticmethod
204     def get_anonymous_group() -> 'UserGroup':
205         return UserGroup.query.filter_by(name=ANONYMOUS_GROUPNAME).one()
206 
207     @staticmethod
208     def get_admin_group() -> 'UserGroup':
209         return UserGroup.query.filter_by(name=ADMIN_GROUPNAME).one()
210 
211     @staticmethod
212     def get_groupadmin_group() -> 'UserGroup':
```

```
213         return UserGroup.query.filter_by(name=GROUPADMIN_GROUPNAME).one()
214
215     @staticmethod
216     def get_organization_group(org: str) -> 'UserGroup':
217         gname = org + ORG_GROUP_SUFFIX
218         return UserGroup.get_or_create_group(gname)
219
220     @staticmethod
221     def get_haka_group() -> 'UserGroup':
222         haka_group_name = 'Haka users'
223         return UserGroup.get_or_create_group(haka_group_name)
224
225     @staticmethod
226     def get_organizations() -> List['UserGroup']:
227         return UserGroup.query.filter(UserGroup.name.endswith(' users') & ↩
        UserGroup.name.notin_(SPECIAL_GROUPS)).all()
228
229     @staticmethod
230     def get_teachers_group() -> 'UserGroup':
231         return UserGroup.query.filter_by(name=TEACHERS_GROUPNAME).one()
232
233     @staticmethod
234     def get_user_creator_group() -> 'UserGroup':
235         user_creator_group_name = 'User creators'
236         return UserGroup.get_or_create_group(user_creator_group_name)
237
238     @staticmethod
239     def get_function_scheduler_group() -> 'UserGroup':
240         return UserGroup.get_or_create_group(FUNCTIONSCHEDULER_GROUPNAME)
241
242     @staticmethod
243     def get_or_create_group(group_name: str) -> 'UserGroup':
244         ug = UserGroup.get_by_name(group_name)
245         if not ug:
246             ug = UserGroup.create(group_name)
247             db.session.add(ug)
248         return ug
249
250     @staticmethod
251     def get_logged_in_group() -> 'UserGroup':
252         return UserGroup.query.filter_by(name=LOGGED_IN_GROUPNAME).one()
253
254
255 @lru_cache()
256 def get_logged_in_group_id() -> int:
257     return UserGroup.get_logged_in_group().id
258
259
260 @lru_cache()
261 def get_anonymous_group_id() -> int:
262     return UserGroup.get_anonymous_group().id
263
264
265 def get_usergroup_eager_query():
266     from timApp.item.block import Block
267     return (
268         UserGroup.query
269             .options(joinedload(UserGroup.admin_doc)
270                      .joinedload(Block.docentries))
```

```
271             .options(joinedload(UserGroup.current_memberships))
272     )
273
274
275 def get_sisu_groups_by_filter(f) -> List[UserGroup]:
276     gs: List[UserGroup] = (
277         get_usergroup_eager_query()
278             .join(ScimUserGroup)
279             .filter(f)
280             .all()
281     )
282     return gs
283
284
285 # When a SCIM group is deleted, the group name gets this prefix.
286 DELETED_GROUP_PREFIX = 'deleted:'
287
288
289 @attr.s(auto_attribs=True)
290 class UserGroupWithSisuInfo:
291     """Wrapper for UserGroup that reports the sisugroup path in to_json."""
292     ug: UserGroup
293
294     def to_json(self):
295         return {
296             **self.ug.to_json(),
297             'admin_doc': self.ug.admin_doc.docentries[0] if self.ug.admin_doc else None,
298             'sisugroup_path': parse_sisu_group_display_name(
299                 self.ug.display_name).sisugroups_doc_path if self.ug.display_name ↩
    else None
300         }
```

timApp/user/usergroupmember.py

```
1 from sqlalchemy import func
2
3 from timApp.timdb.sqa import db
4 from timApp.util.utils import get_current_time
5
6
7 class UserGroupMember(db.Model):
8     """Associates Users with UserGroups."""
9     __tablename__ = 'usergroupmember'
10    usergroup_id = db.Column(db.Integer, db.ForeignKey('usergroup.id'), ↩
    primary_key=True)
11    user_id = db.Column(db.Integer, db.ForeignKey('useraccount.id'), primary_key=True)
12    membership_end = db.Column(db.DateTime(timezone=True))
13    added_by = db.Column(db.Integer, db.ForeignKey('useraccount.id'))
14
15    user = db.relationship(
16        'User',
17        foreign_keys=[user_id],
18    )
19    adder = db.relationship(
20        'User',
21        foreign_keys=[added_by],
22    )
23    group = db.relationship(
24        'UserGroup',
```

```
25      )
26
27      def set_expired(self):
28          # Avoid cyclical importing.
29          from timApp.messaging.messagelist.messagelist_utils import ↩
        sync_message_list_on_expire
30          self.membership_end = get_current_time()
31          # When the membership is expired, update message lists the user has been part ↩
        of via the group.
32          sync_message_list_on_expire(self.user, self.group)
33
34
35  membership_current = ((UserGroupMember.membership_end == None) | (
36          func.current_timestamp() < UserGroupMember.membership_end))
37
38  membership_deleted = (func.current_timestamp() >= UserGroupMember.membership_end)
```