File: tulostettavat/tim.py

```
1  # -*- coding: utf-8 -*-
2  # Modified hajoviin
3  import logging
4  import os
5  import imghdr
6  import io
7  import re
8  import time
9  import datetime
10  from time import mktime
11  import posixpath
12  import threading
13
14  from flask import Flask, redirect, url_for, Blueprint
15  from flask import stream_with_context
16  from flask import render_template
17  from flask import send_from_directory
18  from werkzeug.utils import secure_filename
19  from flask.helpers import send_file
20  from bs4 import UnicodeDammit
21  from ReverseProxied import ReverseProxied
22
23  import containerLink
24  from routes.edit import edit_page
25  from routes.manage import manage_page
26  from routes.view import view_page
27  from routes.login import login_page
28  from timdb.timdbbase import TimDbException
29  import pluginControl
30  from containerLink import PluginException
31  from routes.settings import settings_page
32  from routes.common import *
33
34
35  app = Flask(__name__)
36  app.config.from_pyfile('defaultconfig.py', silent=False)
37  app.config.from_envvar('TIM_SETTINGS', silent=True)
38  # Compress(app)
39
40  app.register_blueprint(settings_page)
41  app.register_blueprint(manage_page)
42  app.register_blueprint(edit_page)
43  app.register_blueprint(view_page)
44  app.register_blueprint(login_page)
45  app.register_blueprint(Blueprint('bower',
46                                   __name__,
47
static_folder='static/scripts/bower_components',
48
static_url_path='/static/scripts/bower_components'))
49
50  print('Debug mode: {}'.format(app.config['DEBUG']))
51
52  KNOWN_TAGS = ['difficult', 'unclear']
53
54  #
current_app.logging.basicConfig(filename='timLog.log',level=logging.DEBU
G, format='%(asctime)s %(message)s')
```

```
55  formatter = logging.Formatter(
56      "{\"time\":%(asctime)s, \"file\": %(pathname)s, \"line\"
:%(lineno)d, \"messageLevel\":  %(levelname)s, \"message\":
%(message)s}")
57  if not os.path.exists(app.config['LOG_DIR']):
58      os.mkdir(app.config['LOG_DIR'])
59  handler = logging.FileHandler(app.config['LOG_PATH'])
60  handler.setLevel(logging.DEBUG)
61  handler.setFormatter(formatter)
62  app.logger.addHandler(handler)
63
64
65  def allowed_file(filename):
66      return '.' in filename and \
67             filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS
68
69
70  DOC_EXTENSIONS = ['txt', 'md', 'markdown']
71  PIC_EXTENSIONS = ['png', 'jpg', 'jpeg', 'gif']
72  ALLOWED_EXTENSIONS = set(PIC_EXTENSIONS + DOC_EXTENSIONS)
73  STATIC_PATH = "./static/"
74  DATA_PATH = "./static/data/"
75
76  LOG_LEVELS = {"CRITICAL": app.logger.critical,
77                "ERROR": app.logger.error,
78                "WARNING": app.logger.warning,
79                "INFO": app.logger.info,
80                "DEBUG": app.logger.debug}
81
82  # List to question that are being asked
83  __question_to_be_asked = []
84
85  # Dictionary for pull request to answers
86  __pull_answer = {}
87  # Dictionary to activities of different users
88  __user_activity = {}
89
90  # Logger call
91  @app.route("/log/", methods=["POST"])
92  def logMessage():
93      try:
94          message = request.get_json()['message']
95          level = request.get_json()['level']
96          LOG_LEVELS[level](message)
97      except KeyError:
98          app.logger.error("Failed logging call: " +
str(request.get_data()))
99
100
101  def error_generic(error, code):
102      if 'text/html' in request.headers.get("Accept", ""):
103          return render_template(str(code) + '.html',
message=error.description), code
104      else:
105          return jsonResponse({'error': error.description}, code)
106
107
108  @app.errorhandler(400)
109  def bad_request(error):
110      return error_generic(error, 400)
111
```

```
112
113  @app.errorhandler(403)
114  def forbidden(error):
115      return error_generic(error, 403)
116
117
118  @app.errorhandler(404)
119  def notFound(error):
120      return error_generic(error, 404)
121
122
123  @app.route('/diff/<int:doc_id>/<doc_hash>')
124  def documentDiff(doc_id, doc_hash):
125      timdb = getTimDb()
126      if not timdb.documents.documentExists(doc_id):
127          abort(404)
128      verifyEditAccess(doc_id, "Sorry, you don't have permission to
download this document.")
129      try:
130          doc_diff =
timdb.documents.getDifferenceToPrevious(DocIdentifier(doc_id, doc_hash))
131          return render_template('diff.html', diff_html=doc_diff)
132      except TimDbException as e:
133          abort(404, str(e))
134
135
136  @app.route('/download/<int:doc_id>/<doc_hash>')
137  def documentHistory(doc_id, doc_hash):
138      timdb = getTimDb()
139      if not timdb.documents.documentExists(doc_id):
140          abort(404)
141      verifyEditAccess(doc_id, "Sorry, you don't have permission to
download this document.")
142      try:
143          doc_data =
timdb.documents.getDocumentMarkdown(DocIdentifier(doc_id, doc_hash))
144          return Response(doc_data, mimetype="text/plain")
145      except TimDbException as e:
146          abort(404, str(e))
147
148
149  @app.route('/download/<int:doc_id>')
150  def downloadDocument(doc_id):
151      return documentHistory(doc_id, getNewest(doc_id).hash)
152
153
154  @app.route('/upload/', methods=['POST'])
155  def upload_file():
156      if request.method != 'POST':
157          return jsonResponse({'message': 'Only POST method is
supported.'}, 405)
158      if not loggedIn():
159          return jsonResponse({'message': 'You have to be logged in
to upload a file.'}, 403)
160      timdb = getTimDb()
161
162      doc = request.files['file']
163      folder = request.form['folder']
164      filename = posixpath.join(folder,
secure_filename(doc.filename))
165
```

```
166      userName = getCurrentUserName()
167      if not timdb.users.userHasAdminAccess(getCurrentUserId()) and
not timdb.users.isUserInGroup(userName,
168
"Timppa-projektiryhmä") and re.match(
169                              '^' + userName + '\/', filename) is
None:
170          return jsonResponse({'message': "You're not authorized to
write here."}, 403)
171
172      if not allowed_file(doc.filename):
173          return jsonResponse({'message': 'The file format is not
allowed.'}, 403)
174
175      if (filename.endswith(tuple(DOC_EXTENSIONS))):
176          content = UnicodeDammit(doc.read()).unicode_markup
177          if not content:
178              return jsonResponse({'message': 'Failed to convert the
file to UTF-8.'}, 400)
179          timdb.documents.importDocument(content, filename,
getCurrentUserGroup())
180          return "Successfully uploaded document"
181      else:
182          content = doc.read()
183          imgtype = imghdr.what(None, h=content)
184          if imgtype is not None:
185              img_id, img_filename = timdb.images.saveImage(content,
doc.filename, getCurrentUserGroup())
186              timdb.users.grantViewAccess(0, img_id)  # So far
everyone can see all images
187              return jsonResponse({"file": str(img_id) + '/' +
img_filename})
188          else:
189              doc.save(os.path.join(app.config['UPLOAD_FOLDER'],
filename))
190              return redirect(url_for('uploaded_file',
filename=filename))
191
192
193  @app.route('/images/<int:image_id>/<image_filename>')
194  def getImage(image_id, image_filename):
195      timdb = getTimDb()
196      if not timdb.images.imageExists(image_id, image_filename):
197          abort(404)
198      verifyViewAccess(image_id)
199      img_data = timdb.images.getImage(image_id, image_filename)
200      imgtype = imghdr.what(None, h=img_data)
201      f = io.BytesIO(img_data)
202      return send_file(f, mimetype='image/' + imgtype)
203
204
205  @app.route('/images')
206  def getAllImages():
207      timdb = getTimDb()
208      images = timdb.images.getImages()
209      allowedImages = [image for image in images if
timdb.users.userHasViewAccess(getCurrentUserId(), image['id'])]
210      return jsonResponse(allowedImages)
211
212
213  # Route to get info from lectures. Gives answers, and messages and
```

```
other necessary info.
@app.route('/getLectureInfo')
def get_lecture_info():
    if not request.args.get("lecture_id"):
        abort(400, "Bad request, missing lecture id")
    lecture_id = int(request.args.get("lecture_id"))
    messages = get_all_messages(lecture_id)
    timdb = getTimDb()
    answer_dicts =
timdb.lecture_answers.get_answers_to_questions_from_lecture(lecture_id)
    question_ids = []
    answerers = []
    for singleDict in answer_dicts:
        singleDict['user_name'] =
timdb.users.getUser(singleDict['user_id']).get("name")
        if singleDict['question_id'] not in question_ids:
            question_ids.append(singleDict['question_id'])
        if singleDict['user_name'] not in answerers:
            answerers.append(singleDict['user_name'])

    lecture_questions =
timdb.questions.get_multiple_questions(question_ids)

    is_lecturer = False
    current_user = getCurrentUserId()
    if timdb.lectures.get_lecture(lecture_id)[0].get("lecturer") ==
current_user:
        is_lecturer = True

    user_name = timdb.users.getUser(current_user).get("name")

    return jsonResponse(
        {"messages": messages, "answerers": answerers, "answers":
answer_dicts, "questions": lecture_questions,
         "isLecturer": is_lecturer,
         "userName": user_name})


# Route to get all the messages from some lecture.
# Tulisi hakea myös kaikki aukiolevat kysymykset, joihin käyttäjä
ei ole vielä vastannut.
@app.route('/getAllMessages')
def get_all_messages(param_lecture_id=-1):
    if not request.args.get("lecture_id") and param_lecture_id is
-1:
        abort(400, "Bad request, missing lecture id")
    timdb = getTimDb()
    if request.args.get("lecture_id"):
        lecture_id = int(request.args.get("lecture_id"))
    else:
        lecture_id = param_lecture_id

    # Prevents previously asked question to be asked from user and
new questions from people who just came to lecture
    current_user = getCurrentUserId()
    for triple in __question_to_be_asked:
        if triple[0] == lecture_id and current_user not in
triple[2]:
            triple[2].append(current_user)

    messages = timdb.messages.get_messages(lecture_id)
```

```
265      if len(messages) > 0:
266          list_of_new_messages = []
267          for message in messages:
268              user = timdb.users.getUser(message.get('user_id'))
269              time_as_time = datetime.datetime.fromtimestamp(
270                  mktime(time.strptime(message.get("timestamp"), "%Y-
%m-%d %H:%M:%S.%f")))
271              list_of_new_messages.append(
272                  {"sender": user.get('name'),
273                   "time": time_as_time.strftime('%H:%M:%S'),
274                   "message": message.get('message')})
275
276          # When using this same method just to get the messages for
lectureInfo
277          if param_lecture_id is not -1:
278              return list_of_new_messages
279
280          return jsonResponse(
281              {"status": "results", "data": list_of_new_messages,
"lastid": messages[-1].get('msg_id'),
282               "lectureId": lecture_id})
283
284      # When using this same method just to get the messages for
lectureInfo
285      if param_lecture_id is not -1:
286          return []
287
288      return jsonResponse({"status": "no-results", "data": [],
"lastid": -1, "lectureId": lecture_id})
289
290
291  # Gets updates from some lecture. Checks updates in 1 second
frequently and answers if there is updates.
292  @app.route('/getUpdates')
293  def get_updates():
294      if not request.args.get('client_message_id') or not
request.args.get("lecture_id") or not request.args.get(
295              'doc_id') or not request.args.get('is_lecturer'):
296          abort(400, "Bad requst")
297      client_last_id = int(request.args.get('client_message_id'))
298
299      use_wall = False
300      use_quesitions = False
301      if request.args.get('get_messages') == "true":
302          session['use_wall'] = True
303          use_wall = True
304      else:
305          session['use_wall'] = False
306
307      if request.args.get('get_questions') == "true":
308          session['use_questions'] = True
309          use_quesitions = True
310      else:
311          session['use_questions'] = False
312
313      helper = request.args.get("lecture_id")
314      if len(helper) > 0:
315          lecture_id = int(float(helper))
316      else:
317          lecture_id = -1
318
```

```
319      timdb = getTimDb()
320      step = 0
321
322      doc_id = int(request.args.get('doc_id'))
323
324      if not check_if_lecture_is_running(lecture_id):
325          timdb.lectures.delete_users_from_lecture(lecture_id)
326          clean_dictionaries_by_lecture(lecture_id)
327          return get_running_lectures(doc_id)
328
329      list_of_new_messages = []
330      last_message_id = -1
331
332      lecturers = []
333      students = []
334
335      time_now = str(datetime.datetime.now().strftime("%H:%M:%S"))
336      __user_activity[getCurrentUserId(), lecture_id] = time_now
337
338      lecture = timdb.lectures.get_lecture(lecture_id)
339
340      current_user = getCurrentUserId()
341      lecture_ending = 100
342
343      # Jos poistaa tämän while loopin, muuttuu long pollista
perinteiseksi polliksi
344      while step <= 10:
345
346          # Gets new messages if the wall is in use.
347          if use_wall:
348              last_message =
timdb.messages.get_last_message(lecture_id)
349              if last_message:
350                  last_message_id = last_message[-1].get('msg_id')
351                  if last_message_id != client_last_id:
352                      messages =
timdb.messages.get_new_messages(lecture_id, client_last_id)
353                      messages.reverse()
354
355                      for message in messages:
356                          user =
timdb.users.getUser(message.get('user_id'))
357                          time_as_time =
datetime.datetime.fromtimestamp(
358
mktime(time.strptime(message.get("timestamp"), "%Y-%m-%d %H:%M:%S.%f")))
359                          list_of_new_messages.append(
360                              {"sender": user.get('name'),
361                               "time":
time_as_time.strftime('%H:%M:%S'),
362                               "message": message.get('message')})
363                      last_message_id = messages[-1].get('msg_id')
364
365          # Gets new questions if the questions are in use.
366          if use_quesitions:
367              for pair in __question_to_be_asked:
368                  if pair[0] == lecture_id and current_user not in
pair[2]:
369                      question_json =
timdb.questions.get_question(pair[1])[0].get("questionJson")
370                      pair[2].append(getCurrentUserId())
```

```
                    lecture_ending =
check_if_lecture_is_ending(current_user, timdb, lecture_id)
                    return jsonResponse(
                        {"status": "results", "data":
list_of_new_messages, "lastid": last_message_id,
                         "lectureId": lecture_id, "question": True,
"questionId": pair[1],
                         "questionJson": question_json,
                         "isLecture": True, "lecturers": lecturers,
"students": students,
                         "lectureEnding": lecture_ending})

        if len(list_of_new_messages) > 0:
            if len(lecture) > 0 and lecture[0].get("lecturer") ==
current_user:
                lecture_ending =
check_if_lecture_is_ending(current_user, timdb, lecture_id)
                lecturers, students = get_lecture_users(timdb,
lecture_id)
            return jsonResponse(
                {"status": "results", "data": list_of_new_messages,
"lastid": last_message_id,
                 "lectureId": lecture_id, "isLecture": True,
"lecturers": lecturers, "students": students,
                 "lectureEnding": lecture_ending})

        # Myös tämä sleep kannattaa poistaa.
        time.sleep(1)
        step += 1

    if len(lecture) > 0 and lecture[0].get("lecturer") ==
current_user:
        lecture_ending = check_if_lecture_is_ending(current_user,
timdb, lecture_id)

    return jsonResponse(
        {"status": "no-results", "data": ["No new messages"],
"lastid": client_last_id, "lectureId": lecture_id,
         "isLecture": True, "lecturers": lecturers, "students":
students, "lectureEnding": lecture_ending})


# Checks if the lecture is about to end. 1 -> ends in 1 min. 5 ->
ends in 5 min. 100 -> goes on atleast for 5 mins.
def check_if_lecture_is_ending(current_user, timdb, lecture_id):
    lecture = timdb.lectures.get_lecture(lecture_id)
    lecture_ending = 100
    if len(lecture) > 0 and lecture[0].get("lecturer") ==
current_user:
        time_now = datetime.datetime.now()
        ending_time = datetime.datetime.fromtimestamp(
            mktime(time.strptime(lecture[0].get("end_time"), "%Y-
%m-%d %H:%M")))
        time_left = str(ending_time - time_now)
        splitted_time = time_left.split(",")
        if len(splitted_time) == 1:
            h, m, s = splitted_time[0].split(":")
            hours_as_min = int(h) * 60
            if hours_as_min + int(m) < 1:
                lecture_ending = 1
            elif hours_as_min + int(m) < 5:
```

```
                lecture_ending = 5

    return lecture_ending


# Route to add message to database.
@app.route('/sendMessage', methods=['POST'])
def send_message():
    timdb = getTimDb()
    new_message = request.args.get("message")
    lecture_id = int(request.args.get("lecture_id"))

    new_timestamp = str(datetime.datetime.now())
    msg_id = timdb.messages.add_message(getCurrentUserId(),
lecture_id, new_message, new_timestamp, True)
    return jsonResponse(msg_id)


# Route to render question
@app.route('/lecture/question')
def show_question():
    return render_template('question.html')


# Route to render question
@app.route('/question')
def show_question_without_view():
    return render_template('question.html')


# Route to get specific question
@app.route('/getQuestion')
def get_quesition():
    doc_id = request.args.get('doc_id')
    par_index = request.args.get('par_index')
    timdb = getTimDb()
    question = timdb.questions.get_paragraphs_question(doc_id,
par_index)
    return jsonResponse(question)


# Route to get all questions
@app.route('/getQuestions', methods=['GET'])
def get_questions():
    timdb = getTimDb()
    questions = timdb.questions.get_questions()
    return jsonResponse(questions)


# Route to get add question to database
@app.route('/addQuestion/', methods=['POST'])
def add_question():
    # TODO: Only lecturers should be able to create questions.
    # verifyOwnership(doc_id)
    question_id = None
    if (request.args.get('question_id')):
        question_id = int(request.args.get('question_id'))
    question_title = request.args.get('question_title')
    answer = request.args.get('answer')
    doc_id = int(request.args.get('doc_id'))
    par_index = int(request.args.get('par_index'))
```

```
475      questionJson = request.args.get('questionJson')
476      timdb = getTimDb()
477      if not question_id:
478          questions = timdb.questions.add_questions(doc_id,
par_index, question_title, answer, questionJson)
479      else:
480          questions = timdb.questions.update_question(question_id,
doc_id, par_index, question_title, answer, questionJson)
481      return jsonResponse(questions)
482
483
484  # Route to check if the current user is in some lecture in specific
document
485  @app.route('/checkLecture', methods=['GET'])
486  def check_lecture():
487      if not request.args.get('doc_id'):
488          abort(400)
489
490      doc_id = int(request.args.get('doc_id'))
491      timdb = getTimDb()
492      current_user = getCurrentUserId()
493      is_in_lecture, lecture_id, =
timdb.lectures.check_if_in_lecture(doc_id, current_user)
494      lecture = timdb.lectures.get_lecture(lecture_id)
495      lecturers = []
496      students = []
497      if lecture:
498          lecture_code = lecture[0].get("lecture_code")
499          if lecture[0].get("lecturer") == current_user:
500              is_lecturer = True
501              lecturers, students = get_lecture_users(timdb,
lecture_id)
502          else:
503              is_lecturer = False
504
505          if "use_wall" in session:
506              use_wall = session['use_wall']
507          else:
508              use_wall = False
509
510          if "use_questions" in session:
511              use_question = session['use_wall']
512          else:
513              use_question = False
514
515          return jsonResponse({"isInLecture": is_in_lecture,
"lectureId": lecture_id, "lectureCode": lecture_code,
516                               "isLecturer": is_lecturer,
"startTime": lecture[0].get("start_time"),
517                               "endTime": lecture[0].get("end_time"),
"lecturers": lecturers, "students": students,
518                               "useWall": use_wall, "useQuestions":
use_question})
519      else:
520          return get_running_lectures(doc_id)
521
522
523  # Route to start lecture that's start time is in future
524  @app.route("/startFutureLecture", methods=['POST'])
525  def start_future_lecture():
526      if not request.args.get('lecture_code') or not
```

```
request.args.get("doc_id"):
527           abort(400)
528
529       timdb = getTimDb()
530       lecture_code = request.args.get('lecture_code')
531       doc_id = int(request.args.get("doc_id"))
532       verifyOwnership(doc_id)
533       lecture = timdb.lectures.get_lecture_by_code(lecture_code,
doc_id)
534       time_now = str(datetime.datetime.now().strftime("%Y-%m-%d
%H:%M"))
535       lecture = timdb.lectures.update_lecture_starting_time(lecture,
time_now)
536       timdb.lectures.join_lecture(lecture.get("lecture_id"),
getCurrentUserId(), True)
537       students, lecturers = get_lecture_users(timdb,
lecture.get("lecture_id"))
538       return jsonResponse({"isLecturer": True, "lectureCode":
lecture_code, "startTime": lecture.get("start_time"),
539                            "endTime": lecture.get("end_time"),
"lectureId": lecture.get("lecture_id"),
540                            "students": students,
541                            "lecturers": lecturers})
542
543
544   # Route to get all the lectures from document
545   @app.route('/getAllLecturesFromDocument', methods=['GET'])
546   def get_all_lectures():
547       if not request.args.get('doc_id'):
548           abort(400)
549
550       doc_id = int(request.args.get('doc_id'))
551       timdb = getTimDb()
552
553       lectures =
timdb.lectures.get_all_lectures_from_document(doc_id)
554       time_now = str(datetime.datetime.now().strftime("%Y-%m-%d
%H:%M"))
555       current_lectures = []
556       past_lectures = []
557       future_lectures = []
558       for lecture in lectures:
559           lecture_info = {"lecture_id": lecture.get("lecture_id"),
"lecture_code": lecture.get('lecture_code'),
560                           "target": "/showLectureInfo/" +
str(lecture.get("lecture_id")),
561                           "is_access_code": not
(lecture.get("password") == "")};
562           if lecture.get("start_time") <= time_now <
lecture.get("end_time"):
563               current_lectures.append(lecture_info)
564           elif lecture.get("end_time") <= time_now:
565               past_lectures.append(lecture_info)
566           else:
567               future_lectures.append(lecture_info)
568
569       return jsonResponse(
570           {"currentLectures": current_lectures, "futureLectures":
future_lectures, "pastLectures": past_lectures})
571
572
```

```
573  # Route to get show lecture info of some specific lecture
574  @app.route('/showLectureInfo/<int:lecture_id>', methods=['GET'])
575  def show_lecture_info(lecture_id):
576      timdb = getTimDb()
577      lecture = timdb.lectures.get_lecture(lecture_id)
578      if len(lecture) <= 0:
579          abort(400)
580
581      lecture = lecture[0]
582      return render_template("lectureInfo.html",
583                             docId=lecture.get("doc_id"),
584                             lectureId=lecture_id,
585                             lectureCode=lecture.get("lecture_code"),
586
lectureStartTime=lecture.get("start_time"),
587                             lectureEndTime=lecture.get("end_time"))
588
589
590  # Route to get show lecture info of some specific lecture
591  @app.route('/showLectureInfoGivenName/', methods=['GET'])
592  def show_lecture_info_given_name():
593      timdb = getTimDb()
594      lecture =
timdb.lectures.get_lecture_by_name(request.args.get('lecture_code'),int(
request.args.get('doc_id')))
595      if len(lecture) <= 0:
596          abort(400)
597
598      lecture = lecture[0]
599      return jsonResponse({"docId": lecture.get("doc_id"),
"lectureId": lecture.get("lecture_id"),
600                           "lectureCode": lecture.get("lecture_code"),
"lectureStartTime": lecture.get("start_time"),
601                           "lectureEndTime": lecture.get("end_time")})
602
603
604  # Gets users from specific lecture
605  # returns 2 lists of dictionaries.
606  # TODO: Think if it would be better to return only one
607  def get_lecture_users(timdb, lecture_id):
608      lecture = timdb.lectures.get_lecture(lecture_id)
609      lecturers = []
610      students = []
611      users = timdb.lectures.get_users_from_leture(lecture_id)
612
613      if len(__user_activity) <= 0:
614          for user in users:
615              __user_activity[user.get("user_id"), lecture_id] = ""
616
617      for user in users:
618          if lecture[0].get("lecturer") == user.get("user_id"):
619              if (user.get("user_id"), lecture_id) in
__user_activity:
620                  lecturer = {"name":
timdb.users.getUser(user.get('user_id')).get("name"),
621                              "active":
__user_activity[user.get("user_id"), lecture_id]}
622              else:
623                  lecturer = {"name":
timdb.users.getUser(user.get('user_id')).get("name"), "active": ""}
624              lecturers.append(lecturer)
```

```
625
626         else:
627             if (user.get("user_id"), lecture_id) in
__user_activity:
628                 student = {"name":
timdb.users.getUser(user.get('user_id')).get("name"),
629                            "active":
__user_activity[user.get("user_id"), lecture_id]}
630             else:
631                 student = {"name":
timdb.users.getUser(user.get('user_id')).get("name"), "active": ""}
632             students.append(student)
633
634     return lecturers, students
635
636
637 # Checks if some lecture is running or not.
638 def check_if_lecture_is_running(lecture_id):
639     timdb = getTimDb()
640     time_now = str(datetime.datetime.now().strftime("%Y-%m-%d
%H:%M"))
641     return timdb.lectures.check_if_lecture_is_running(lecture_id,
time_now)
642
643
644 # Gets all lectures that are currently running. Also gives the ones
that are in the future
645 def get_running_lectures(doc_id):
646     timdb = getTimDb()
647     time_now = str(datetime.datetime.now().strftime("%Y-%m-%d
%H:%M"))
648     lecture_code = "Not running"
649     list_of_lectures = timdb.lectures.get_document_lectures(doc_id,
time_now)
650     current_lecture_codes = []
651     future_lectures = []
652     is_lecturer = hasOwnership(doc_id)
653     for lecture in list_of_lectures:
654         if lecture.get("start_time") <= time_now <
lecture.get("end_time"):
655             current_lecture_codes.append({"lecture_code":
lecture.get("lecture_code"),
656                                           "is_access_code": not
(lecture.get("password") == "")})
657         else:
658             future_lectures.append(
659                 {"lecture_code": lecture.get("lecture_code"),
660                  "lecture_start": lecture.get("start_time")})
661     return jsonResponse(
662         {"isLecturer": is_lecturer, "lectures":
current_lecture_codes, "futureLectures": future_lectures,
663          "lectureCode": lecture_code})
664
665
666 # Route to create lecture.
667 @app.route('/createLecture', methods=['POST'])
668 def create_lecture():
669     if not request.args.get("doc_id") or not
request.args.get("start_date") or not request.args.get(
670             "end_date") or not request.args.get("lecture_code"):
671         abort(400, "Missing parameters")
```

```
672      lecture_id = -1
673      if request.args.get("lecture_id"):
674          lecture_id = int(request.args.get("lecture_id"))
675      doc_id = int(request.args.get("doc_id"))
676      verifyOwnership(doc_id)
677      timdb = getTimDb()
678      start_time = request.args.get("start_date")
679      end_time = request.args.get("end_date")
680      lecture_code = request.args.get("lecture_code")
681      password = request.args.get("password")
682      if not password:
683          password = ""
684      current_user = getCurrentUserId()
685      if not timdb.lectures.check_if_correct_name(doc_id,
lecture_code, lecture_id):
686              abort(400, "Can't create two or more lectures with the
same name to the same document.")
687      if lecture_id <0:
688          lecture_id = timdb.lectures.create_lecture(doc_id,
current_user, start_time, end_time, lecture_code, password, True)
689      else:
690          timdb.lectures.update_lecture(lecture_id, doc_id,
current_user, start_time, end_time, lecture_code, password)
691
692      current_time = datetime.datetime.now().strftime("%Y-%m-%d
%H:%M")
693
694      if start_time <= current_time <= end_time:
695          timdb.lectures.join_lecture(lecture_id, current_user, True)
696      return jsonResponse({"lectureId": lecture_id})
697
698
699  # Route to end lecture
700  @app.route('/endLecture', methods=['POST'])
701  def end_lecture():
702      if not request.args.get("doc_id") or not
request.args.get("lecture_id"):
703          abort(400)
704
705      doc_id = int(request.args.get("doc_id"))
706      lecture_id = int(request.args.get("lecture_id"))
707      verifyOwnership(doc_id)
708      timdb = getTimDb()
709      timdb.lectures.delete_users_from_lecture(lecture_id)
710
711      now = datetime.datetime.now().strftime("%Y-%m-%d %H:%M")
712      timdb.lectures.set_end_for_lecture(lecture_id, str(now))
713
714      clean_dictionaries_by_lecture(lecture_id)
715
716      return get_running_lectures(doc_id)
717
718
719  # Cleans dictionaries from lecture that isn't runnin anymore
720  def clean_dictionaries_by_lecture(lecture_id):
721      entries_to_remove = []
722      pulls_to_remove = []
723      for pair in __question_to_be_asked:
724          if pair[0] == lecture_id:
725              __question_to_be_asked.remove(pair)
726
```

```
727      for actitivty in __user_activity:
728          if actitivty[1] == lecture_id:
729              entries_to_remove.append(actitivty)
730
731      for actitivty in entries_to_remove:
732          del __user_activity[actitivty]
733
734      for pullRequest in __pull_answer:
735          if pullRequest[1] == lecture_id:
736              pulls_to_remove.append(pullRequest)
737
738      for pullRequest in pulls_to_remove:
739          del __pull_answer[pullRequest]
740
741
742  # Route to extend lecture
743  @app.route('/extendLecture', methods=['POST'])
744  def extend_lecture():
745      if not request.args.get("doc_id") or not
request.args.get("lecture_id") or not request.args.get("new_end_time"):
746          abort(400)
747      doc_id = int(request.args.get("doc_id"))
748      lecture_id = int(request.args.get("lecture_id"))
749      new_end_time = request.args.get("new_end_time")
750      verifyOwnership(doc_id)
751      timdb = getTimDb()
752      timdb.lectures.extend_lecture(lecture_id, new_end_time)
753      return jsonResponse("")
754
755
756  # Route to delete lecture.
757  @app.route('/deleteLecture', methods=['POST'])
758  def delete_lecture():
759      if not request.args.get("doc_id") or not
request.args.get("lecture_id"):
760          abort(400)
761      doc_id = int(request.args.get("doc_id"))
762      verifyOwnership(doc_id)
763      lecture_id = int(request.args.get("lecture_id"))
764      timdb = getTimDb()
765      timdb.messages.delete_messages_from_lecture(lecture_id, True)
766      timdb.lectures.delete_users_from_lecture(lecture_id, True)
767
768      timdb.lectures.delete_lecture(lecture_id, True)
769
770      clean_dictionaries_by_lecture(lecture_id)
771
772      return get_running_lectures(doc_id)
773
774
775  # Route to join lecture. Checks that the given password is correct.
776  @app.route('/joinLecture', methods=['POST'])
777  def join_lecture():
778      if not request.args.get("doc_id") or not
request.args.get("lecture_code"):
779          abort(400, "Missing parameters")
780      timdb = getTimDb()
781      doc_id = int(request.args.get("doc_id"))
782      lecture_code = request.args.get("lecture_code")
783      password_quess = request.args.get("password_quess")
784      lecture_id = timdb.lectures.get_lecture_by_code(lecture_code,
```

```
doc_id)
785      current_user = getCurrentUserId()
786      lecture = timdb.lectures.get_lecture(lecture_id)
787      if lecture[0].get("password") != password_quess:
788          return jsonResponse({"correctPassword": False})
789      timdb.lectures.join_lecture(lecture_id, current_user, True)
790
791      time_now = str(datetime.datetime.now().strftime("%H:%M:%S"))
792      __user_activity[getCurrentUserId(), lecture_id] = time_now
793
794      lecturers = []
795      students = []
796      if lecture[0].get("lecturer") == current_user:
797          is_lecturer = True
798          lecturers, students = get_lecture_users(timdb, lecture_id)
799      else:
800          is_lecturer = False
801      return jsonResponse(
802          {"correctPassword": True, "inLecture": True, "lectureId":
lecture_id, "isLecturer": is_lecturer,
803           "lectureCode": lecture_code, "startTime":
lecture[0].get("start_time"),
804           "endTime": lecture[0].get("end_time"), "lecturers":
lecturers, "students": students})
805
806
807  # Route to leace lecture
808  @app.route('/leaveLecture', methods=['POST'])
809  def leave_lecture():
810      timdb = getTimDb()
811      lecture_id = int(request.args.get("lecture_id"))
812      doc_id = int(request.args.get("doc_id"))
813      timdb.lectures.leave_lecture(lecture_id, getCurrentUserId(),
True)
814      del __user_activity[getCurrentUserId(), lecture_id]
815      return get_running_lectures(doc_id)
816
817
818  @app.route('/uploads/<filename>')
819  def uploaded_file(filename):
820      return send_from_directory(app.config['UPLOAD_FOLDER'],
filename)
821
822
823  @app.route("/getDocuments")
824  def getDocuments():
825      versions = 1
826      if request.args.get('versions'):
827          ver_str = request.args.get('versions')
828          if re.match('^\d+$', ver_str) is None:
829              return "Invalid version argument."
830          else:
831              ver_int = int(ver_str)
832              if ver_int > 10:
833                  # DoS prevention
834                  return "Version limit is currently capped at 10."
835              else:
836                  versions = ver_int
837
838      timdb = getTimDb()
839      docs = timdb.documents.getDocuments(historylimit=versions)
```

```
840      allowedDocs = [doc for doc in docs if
timdb.users.userHasViewAccess(getCurrentUserId(), doc['id'])]
841
842      req_folder = request.args.get('folder')
843      if req_folder is not None and len(req_folder) == 0:
844          req_folder = None
845      finalDocs = []
846
847      for doc in allowedDocs:
848          fullname = doc['name']
849
850          if req_folder:
851              if not fullname.startswith(req_folder + '/'):
852                  continue
853              docname = fullname[len(req_folder) + 1:]
854          else:
855              docname = fullname
856
857          if '/' in docname:
858              continue
859
860          uid = getCurrentUserId()
861          doc['name'] = docname
862          doc['fullname'] = fullname
863          doc['canEdit'] = timdb.users.userHasEditAccess(uid,
doc['id'])
864          doc['isOwner'] =
timdb.users.userIsOwner(getCurrentUserId(), doc['id']) or
timdb.users.userHasAdminAccess(uid)
865          doc['owner'] = timdb.users.getOwnerGroup(doc['id'])
866          finalDocs.append(doc)
867
868      return jsonResponse(finalDocs)
869
870
871  @app.route("/getFolders")
872  def getFolders():
873      root_path = request.args.get('root_path')
874      timdb = getTimDb()
875      folders = timdb.folders.getFolders(root_path,
getCurrentUserGroup())
876      allowedFolders = [f for f in folders if
timdb.users.userHasViewAccess(getCurrentUserId(), f['id'])]
877      uid = getCurrentUserId()
878
879      for f in allowedFolders:
880          f['isOwner'] = timdb.users.userIsOwner(uid, f['id']) or
timdb.users.userHasAdminAccess(uid)
881          f['owner'] = timdb.users.getOwnerGroup(f['id'])
882
883      return jsonResponse(allowedFolders)
884
885
886  @app.route("/getJSON/<int:doc_id>/")
887  def getJSON(doc_id):
888      timdb = getTimDb()
889      verifyViewAccess(doc_id)
890      try:
891          texts =
timdb.documents.getDocumentBlocks(getNewest(doc_id))
892          doc = timdb.documents.getDocument(doc_id.id)
```

```
            return jsonResponse({"name": doc['name'], "text": texts})
        except IOError as err:
            print(err)
            return "No data found"


@app.route("/getJSON-HTML/<int:doc_id>")
def getJSON_HTML(doc_id):
    timdb = getTimDb()
    verifyViewAccess(doc_id)
    try:
        newest = getNewest(doc_id)
        blocks = timdb.documents.getDocumentAsHtmlBlocks(newest)
        doc = timdb.documents.getDocument(doc_id)
        return jsonResponse({"name": doc['name'], "text": blocks})
    except ValueError as err:
        print(err)
        return "[]"
    except TimDbException as err:
        print(err)
        return "[]"


def createItem(itemName, itemType, createFunction):
    if not loggedIn():
        return jsonResponse({'message': 'You have to be logged in
to create a {}.'.format(itemType)}, 403)

    if itemName.startswith('/') or itemName.endswith('/'):
        return jsonResponse({'message': 'The {} name cannot start
or end with /.'.format(itemType)}, 400)

    if re.match('^(\d)*$', itemName) is not None:
        return jsonResponse(
            {'message': 'The {} name can not be a number to avoid
confusion with document id.'.format(itemType)}, 400)

    timdb = getTimDb()

    userName = getCurrentUserName()

    if timdb.documents.getDocumentId(itemName) is not None or
timdb.folders.getFolderId(itemName) is not None:
        return jsonResponse({'message': 'Item with a same name
already exists.'}, 403)

    if not canWriteToFolder(itemName):
        return jsonResponse(
            {'message': 'You cannot create {}s in this folder. Try
users/{} instead.'.format(itemType, userName)}, 403)

    itemId = createFunction(itemName)
    return jsonResponse({'id': itemId, 'name': itemName})


@app.route("/createDocument", methods=["POST"])
def createDocument():
    jsondata = request.get_json()
    docName = jsondata['doc_name']
    timdb = getTimDb()
    createFunc = lambda docName:
```

```
timdb.documents.createDocument(docName, getCurrentUserGroup())
948       return createItem(docName, 'document', createFunc)
949
950
951   @app.route("/createFolder", methods=["POST"])
952   def createFolder():
953       jsondata = request.get_json()
954       folderName = jsondata['name']
955       ownerId = jsondata['owner']
956       timdb = getTimDb()
957       createFunc = lambda folderName:
timdb.folders.createFolder(folderName, ownerId)
958       return createItem(folderName, 'folder', createFunc)
959
960
961   @app.route("/getBlock/<int:docId>/<int:blockId>")
962   def getBlockMd(docId, blockId):
963       timdb = getTimDb()
964       verifyViewAccess(docId)
965       block = timdb.documents.getBlock(getNewest(docId), blockId)
966       return jsonResponse({"text": block})
967
968
969   @app.route("/getBlockHtml/<int:docId>/<int:blockId>")
970   def getBlockHtml(docId, blockId):
971       timdb = getTimDb()
972       verifyViewAccess(docId)
973       block = timdb.documents.getBlockAsHtml(getNewest(docId),
blockId)
974       return block
975
976
977   @app.route("/<plugin>/<path:fileName>")
978   def pluginCall(plugin, fileName):
979       try:
980           req = containerLink.call_plugin_resource(plugin, fileName)
981           return Response(stream_with_context(req.iter_content()),
content_type=req.headers['content-type'])
982       except PluginException:
983           abort(404)
984
985
986   @app.route("/index/<int:docId>")
987   def getIndex(docId):
988       timdb = getTimDb()
989       verifyViewAccess(docId)
990       index = timdb.documents.getIndex(getNewest(docId))
991       return jsonResponse(index)
992
993
994   @app.route("/postNote", methods=['POST'])
995   def postNote():
996       jsondata = request.get_json()
997       noteText = jsondata['text']
998       access = jsondata['access']
999       sent_tags = jsondata.get('tags', {})
1000       tags = []
1001       for tag in KNOWN_TAGS:
1002           if sent_tags[tag]:
1003               tags.append(tag)
1004       doc_id = jsondata['docId']
```

```
1005      doc_ver = request.headers.get('Version')
1006      paragraph_id = jsondata['par']
1007      verifyCommentRight(doc_id)
1008      timdb = getTimDb()
1009      group_id = getCurrentUserGroup()
1010      timdb.notes.addNote(group_id, doc_id, doc_ver,
int(paragraph_id), noteText, access, tags)
1011      # TODO: Handle error.
1012      return "Success"
1013
1014
1015  @app.route("/editNote", methods=['POST'])
1016  def editNote():
1017      verifyLoggedIn()
1018      jsondata = request.get_json()
1019      group_id = getCurrentUserGroup()
1020      doc_id = int(jsondata['docId'])
1021      doc_ver = request.headers.get('Version')
1022      paragraph_id = int(jsondata['par'])
1023      noteText = jsondata['text']
1024      access = jsondata['access']
1025      note_index = int(jsondata['note_index'])
1026      sent_tags = jsondata.get('tags', {})
1027      tags = []
1028      for tag in KNOWN_TAGS:
1029          if sent_tags[tag]:
1030              tags.append(tag)
1031      timdb = getTimDb()
1032
1033      if not (timdb.notes.hasEditAccess(group_id, doc_id,
paragraph_id, note_index)
1034              or timdb.users.userIsOwner(getCurrentUserId(),
doc_id)):
1035          abort(403, "Sorry, you don't have permission to edit this
note.")
1036
1037      timdb.notes.modifyNote(doc_id, doc_ver, paragraph_id,
note_index, noteText, access, tags)
1038      return "Success"
1039
1040
1041  @app.route("/deleteNote", methods=['POST'])
1042  def deleteNote():
1043      verifyLoggedIn()
1044      jsondata = request.get_json()
1045      group_id = getCurrentUserGroup()
1046      doc_id = int(jsondata['docId'])
1047      paragraph_id = int(jsondata['par'])
1048      note_index = int(jsondata['note_index'])
1049      timdb = getTimDb()
1050
1051      if not (timdb.notes.hasEditAccess(group_id, doc_id,
paragraph_id, note_index)
1052              or timdb.users.userIsOwner(getCurrentUserId(),
doc_id)):
1053          abort(403, "Sorry, you don't have permission to remove
this note.")
1054
1055      timdb.notes.deleteNote(doc_id, paragraph_id, note_index)
1056      return "Success"
1057
```

```
1058
1059  @app.route("/questions/<int:doc_id>")
1060  def getQuestions(doc_id):
1061      verifyOwnership(doc_id)
1062      timdb = getTimDb()
1063      questions = timdb.questions.get_doc_questions(doc_id)
1064      return jsonResponse(questions)
1065
1066
1067  @app.route("/getLectureWithName", methods=['POST'])
1068  def getLectureWithName(lecture_code,doc_id):
1069      verifyOwnership(doc_id)
1070      timdb = getTimDb()
1071      lecture =
timdb.lectures.get_lecture_by_code(lecture_code,doc_id)
1072      return jsonResponse(lecture)
1073
1074
1075  @app.route("/askQuestion", methods=['POST'])
1076  def ask_question():
1077      if not request.args.get('doc_id') or not
request.args.get('question_id') or not request.args.get('lecture_id'):
1078          abort(400, "Bad request")
1079      doc_id = int(request.args.get('doc_id'))
1080      question_id = int(request.args.get('question_id'))
1081      lecture_id = int(request.args.get('lecture_id'))
1082
1083      verifyOwnership(doc_id)
1084
1085      if lecture_id < 0:
1086          abort(400, "Not valid lecture id")
1087
1088      timdb = getTimDb()
1089      if not
json.loads(timdb.questions.get_question(question_id)[0].get("questionJso
n"))["TIMELIMIT"]  :
1090          question_timelimit = 0
1091      else:
1092          question_timelimit =
int(json.loads(timdb.questions.get_question(question_id)[0].get("questio
nJson"))["TIMELIMIT"])
1093      threadToStopQuestion =
threading.Thread(target=stop_question_from_running,
1094                                               args=(lecture_id,
question_id, question_timelimit))
1095
1096      threadToStopQuestion.start()
1097
1098      verifyOwnership(int(doc_id))
1099      __question_to_be_asked.append((lecture_id, question_id, []))
1100
1101      return jsonResponse("")
1102
1103
1104  def stop_question_from_running(lecture_id, question_id,
question_timelimit):
1105      if question_timelimit == 0:
1106          return
1107
1108      # Adding extra time to limit so when people gets question a
bit later than others they still get to answer
```

```
1109      # TODO: If current implementation changes the way that the
question last 10 seconds and after that you can't
1110      # TODO: answer. Remove this part
1111      extra_time = 3
1112      time.sleep(question_timelimit + extra_time)
1113
1114      for question in __question_to_be_asked:
1115          if question[0] == lecture_id and question[1] ==
question_id:
1116              __question_to_be_asked.remove(question)
1117
1118
1119  @app.route("/getQuestionById", methods=['GET'])
1120  def get_question():
1121      if not request.args.get("question_id"):
1122          abort("400")
1123      # doc_id = int(request.args.get('doc_id'))
1124      question_id = int(request.args.get('question_id'))
1125
1126      # verifyOwnership(doc_id)
1127      timdb = getTimDb()
1128      question = timdb.questions.get_question(question_id)
1129      return jsonResponse(question[0])
1130
1131  @app.route("/deleteQuestion", methods=['POST'])
1132  def delete_question():
1133      if not request.args.get("question_id") or not
request.args.get('doc_id'):
1134          abort("400")
1135
1136      doc_id = int(request.args.get('doc_id'))
1137      question_id = int(request.args.get('question_id'))
1138
1139      verifyOwnership(doc_id)
1140      timdb = getTimDb()
1141      timdb.questions.delete_question(question_id)
1142
timdb.lecture_answers.delete_answers_from_question(question_id)
1143
1144      return jsonResponse("")
1145
1146
1147  # Tämän muuttaminen long polliksi vaatii threadien poistamisen
1148  @app.route("/getLectureAnswers", methods=['GET'])
1149  def get_lecture_answers():
1150      if not request.args.get('question_id') or not
request.args.get('doc_id') or not request.args.get('lecture_id'):
1151          abort(400, "Bad request")
1152
1153      verifyOwnership(int(request.args.get('doc_id')))
1154      question_id = int(request.args.get('question_id'))
1155      lecture_id = int(request.args.get('lecture_id'))
1156
1157      __pull_answer[question_id, lecture_id] = threading.Event()
1158
1159      for pull in __pull_answer:
1160          question, lecture = pull
1161          if lecture == lecture_id and question != question_id:
1162              __pull_answer[pull].set()
1163
1164      if not request.args.get("time"):
```

```
1165          time_now = str(datetime.datetime.now().strftime("%Y-%m-%d
%H:%M:%S:%f"))
1166      else:
1167          time_now = request.args.get('time')
1168
1169      __pull_answer[question_id, lecture_id].wait()
1170
1171      timdb = getTimDb()
1172      answers =
timdb.lecture_answers.get_answers_to_question(question_id, time_now)
1173      if len(answers) <= 0:
1174          return jsonResponse({"noAnswer": True})
1175
1176      latest_answer = answers[-1].get("answered_on")
1177
1178      return jsonResponse({"answers": answers, "questionId":
question_id, "latestAnswer": latest_answer})
1179
1180
1181  @app.route("/answerToQuestion", methods=['PUT'])
1182  def answer_to_question():
1183      if not request.args.get("question_id") or not
request.args.get('answers') or not request.args.get('lecture_id'):
1184          abort(400, "Bad request")
1185
1186      timdb = getTimDb()
1187
1188      question_id = int(request.args.get("question_id"))
1189      answer = request.args.get("answers")
1190      whole_answer = answer
1191      lecture_id = int(request.args.get("lecture_id"))
1192      time_now = str(datetime.datetime.now().strftime("%Y-%m-%d
%H:%M:%S:%f"))
1193
1194      question_ended = True
1195      for question in __question_to_be_asked:
1196           if question[0] == lecture_id and question[1] ==
question_id:
1197               question_ended = False
1198
1199      if question_ended:
1200          return jsonResponse({"questionLate": "The question has
already finished. Your answer was not saved."})
1201
1202
1203      single_answers = []
1204      answers = answer.split('|')
1205      for answer in answers:
1206          single_answers.append(answer.split(','))
1207
1208      question_json =
json.loads(timdb.questions.get_question(question_id)[0].get("questionJso
n"))
1209
1210      points = 0.0
1211      for oneAnswer in single_answers:
1212          question_number = 0
1213          for oneLine in oneAnswer:
1214              header_number = 0
1215              if question_json['TYPE'] == "matrix" or
question_json['TYPE'] == "true-false":
```

```
                for header in question_json['DATA']['HEADERS']:
                    if header['text'] == oneLine:
                        try:
                            points += float(
question_json['DATA']['ROWS'][question_number]['COLUMNS'][header_number]
['points'])
                            break
                        except ValueError:
                            points += 0
                    header_number += 1
            else:
                for row in question_json['DATA']['ROWS']:
                    if row['text'] == oneLine:
                        try:
                            points +=
float(row['COLUMNS'][0]['points'])
                            break
                        except ValueError:
                            points += 0

        question_number += 1

    timdb.lecture_answers.add_answer(getCurrentUserId(),
question_id, lecture_id, whole_answer, time_now, points)

    __pull_answer[question_id, lecture_id].set()

    return jsonResponse("")


@app.route("/notes/<int:doc_id>")
def getNotes(doc_id):
    verifyViewAccess(doc_id)
    timdb = getTimDb()
    group_id = getCurrentUserGroup()
    doc_ver = timdb.documents.getNewestVersionHash(doc_id)
    notes = [note for note in timdb.notes.getNotes(group_id,
doc_id, doc_ver)]
    for note in notes:
        note['editable'] = note['UserGroup_id'] == group_id or
timdb.users.userIsOwner(getCurrentUserId(), doc_id)
        note['private'] = note['access'] == 'justme'
        tags = note['tags']
        note['tags'] = {}
        for tag in KNOWN_TAGS:
            note['tags'][tag] = tag in tags
    return jsonResponse(notes)


@app.route("/read/<int:doc_id>", methods=['GET'])
def getReadParagraphs(doc_id):
    verifyReadMarkingRight(doc_id)
    timdb = getTimDb()
    doc_ver = timdb.documents.getNewestVersionHash(doc_id)
    readings = timdb.readings.getReadings(getCurrentUserGroup(),
doc_id, doc_ver)
    for r in readings:
        r.pop('doc_ver', None)
    return jsonResponse(readings)

```

```
1270
1271  @app.route("/read/<int:doc_id>/<int:specifier>", methods=['PUT'])
1272  def setReadParagraph(doc_id, specifier):
1273      verifyReadMarkingRight(doc_id)
1274      timdb = getTimDb()
1275      blocks =
timdb.documents.getDocumentAsBlocks(getNewest(doc_id))
1276      doc_ver = timdb.documents.getNewestVersionHash(doc_id)
1277      if len(blocks) <= specifier:
1278          return jsonResponse({'error': 'Invalid paragraph
specifier.'}, 400)
1279      timdb.readings.setAsRead(getCurrentUserGroup(), doc_id,
doc_ver, specifier)
1280      return "Success"
1281
1282
1283  def parse_task_id(task_id):
1284      # Assuming task_id is of the form "22.palindrome"
1285      pieces = task_id.split('.')
1286      if len(pieces) != 2:
1287          abort(400, 'The format of task_id is invalid. Expected
exactly one dot character.')
1288      doc_id = int(pieces[0])
1289      task_id_name = pieces[1]
1290      return doc_id, task_id_name
1291
1292
1293  @app.route("/<plugintype>/<task_id>/answer/", methods=['PUT'])
1294  def saveAnswer(plugintype, task_id):
1295      timdb = getTimDb()
1296
1297      doc_id, task_id_name = parse_task_id(task_id)
1298      verifyViewAccess(doc_id)
1299      if not 'input' in request.get_json():
1300          return jsonResponse({'error': 'The key "input" was not
found from the request.'}, 400)
1301      answerdata = request.get_json()['input']
1302
1303      answer_browser_data = request.get_json().get('abData', {})
1304      is_teacher = answer_browser_data.get('teacher', False)
1305      if is_teacher:
1306          verifyOwnership(doc_id)
1307
1308      # Load old answers
1309      oldAnswers = timdb.answers.getAnswers(getCurrentUserId(),
task_id)
1310
1311      # Get the newest answer (state). Only for logged in users.
1312      state = oldAnswers[0]['content'] if loggedIn() and
len(oldAnswers) > 0 else None
1313
1314      markup = getPluginMarkup(doc_id, plugintype, task_id_name)
1315      if markup is None:
1316          return jsonResponse(
1317              {'error': 'The task was not found in the document. ' +
str(doc_id) + ' ' + task_id_name},
1318              404)
1319      if markup == "YAMLERROR: Malformed string":
1320          return jsonResponse({'error': 'Plugin markup YAML is
malformed.'}, 400)
1321
```

```
1322      answerCallData = {'markup': markup, 'state': state, 'input':
answerdata, 'taskID': task_id}
1323
1324      pluginResponse = containerLink.call_plugin_answer(plugintype,
answerCallData)
1325
1326      try:
1327          jsonresp = json.loads(pluginResponse)
1328      except ValueError:
1329          return jsonResponse(
1330              {'error': 'The plugin response was not a valid JSON
string. The response was: ' + pluginResponse}, 400)
1331
1332      if 'web' not in jsonresp:
1333          return jsonResponse({'error': 'The key "web" is missing in
plugin response.'}, 400)
1334
1335      if 'save' in jsonresp:
1336          saveObject = jsonresp['save']
1337          tags = []
1338          tim_info = jsonresp.get('tim_info', {})
1339          points = tim_info.get('points', None)
1340
1341          # Save the new state
1342          try:
1343              tags = saveObject['tags']
1344          except (TypeError, KeyError):
1345              pass
1346          if not is_teacher:
1347              timdb.answers.saveAnswer([getCurrentUserId()],
task_id, json.dumps(saveObject), points, tags)
1348          else:
1349              if answer_browser_data.get('saveTeacher', False):
1350                  answer_id = answer_browser_data.get('answer_id',
None)
1351                  if answer_id is None:
1352                      return jsonResponse({'error': 'Missing
answer_id key'}, 400)
1353                  expected_task_id =
timdb.answers.get_task_id(answer_id)
1354                  if expected_task_id != task_id:
1355                      return jsonResponse({'error': 'Task ids did
not match'}, 400)
1356                  users = timdb.answers.get_users(answer_id)
1357                  if len(users) == 0:
1358                      return jsonResponse({'error': 'No users found
for the specified answer'}, 400)
1359                  if not getCurrentUserId() in users:
1360                      users.append(getCurrentUserId())
1361                  points = answer_browser_data.get('points', points)
1362                  if points == "":
1363                      points = None
1364                  timdb.answers.saveAnswer(users, task_id,
json.dumps(saveObject), points, tags)
1365
1366      return jsonResponse({'web': jsonresp['web']})
1367
1368
1369  @app.route("/answers/<task_id>/<user>")
1370  def get_answers(task_id, user):
1371      verifyLoggedIn()
```

```
1372      timdb = getTimDb()
1373      doc_id, task_id_name = parse_task_id(task_id)
1374      if not timdb.documents.documentExists(doc_id):
1375          abort(404, 'No such document')
1376      user_id = timdb.users.getUserByName(user)
1377      if user_id != getCurrentUserId():
1378          verifyOwnership(doc_id)
1379      if user_id is None:
1380          abort(400, 'Non-existent user')
1381      answers = timdb.answers.getAnswers(user_id, task_id)
1382      return jsonResponse(answers)
1383
1384
1385  @app.route("/getState")
1386  def get_state():
1387      timdb = getTimDb()
1388      doc_id, par_id, user, state = unpack_args('doc_id', 'par_id',
'user', 'state', types=[int, int, str, str])
1389      if not timdb.documents.documentExists(doc_id):
1390          abort(404, 'No such document')
1391      user_id = timdb.users.getUserByName(user)
1392      if user_id != getCurrentUserId():
1393          verifyOwnership(doc_id)
1394      if user_id is None:
1395          abort(400, 'Non-existent user')
1396      if not timdb.documents.documentExists(doc_id):
1397          abort(404, 'No such document')
1398      if not hasViewAccess(doc_id):
1399          abort(403, 'Permission denied')
1400
1401      version = request.headers['Version']
1402      block = timdb.documents.getBlockAsHtml(DocIdentifier(doc_id,
version), par_id)
1403
1404      texts, jsPaths, cssPaths, modules =
pluginControl.pluginify([block],
1405
user,
1406
timdb.answers,
1407
doc_id,
1408
user_id,
1409
custom_state=state)
1410      return jsonResponse(texts[0])
1411
1412
1413  def getPluginMarkup(doc_id, plugintype, task_id):
1414      timdb = getTimDb()
1415      doc_markdown =
timdb.documents.getDocumentAsHtmlBlocks(getNewest(doc_id))
1416      for block in doc_markdown:
1417          if ('plugin="{}"'.format(plugintype) in block and "<pre"
in block and 'id="{}"'.format(task_id) in block):
1418              markup = pluginControl.get_block_yaml(block)
1419              return markup
1420      return None
1421
1422
```

```
1423  @app.route("/")
1424  @app.route("/view/")
1425  def indexPage():
1426      timdb = getTimDb()
1427      possible_groups =
timdb.users.getUserGroupsPrintable(getCurrentUserId())
1428      return render_template('index.html',
1429                             userName=getCurrentUserName(),
1430                             userId=getCurrentUserId(),
1431                             userGroups=possible_groups)
1432
1433
1434  def startApp():
1435      # TODO: Think if it is truly necessary to have threaded=True
here
1436      app.wsgi_app = ReverseProxied(app.wsgi_app)
1437      app.run(host='0.0.0.0', port=5000, use_reloader=False,
threaded=True)
```