File: tulostettavat/static/scripts/controllers/answerToQuestionController.js

```
1  /**
2   * Created by hajoviin on 22.4.2015
3   * FILL WITH SUITABLE TEXT
4   * @module answerToQuestionController
5   * @author Matias Berg
6   * @author Bek Eljurkaev
7   * @author Minna Lehtomäki
8   * @author Juhani Sihvonen
9   * @author Hannu Viinikainen
10   * @licence MIT
11   * @copyright 2015 Timppa project authors
12   */
13
14  var angular;
15
16  var timApp = angular.module('timApp');
17  timApp.controller('AnswerToQuestionController', ['$scope',
'$rootScope', function ($scope, $rootScope) {
18      "use strict";
19      $scope.questionHeaders = [];
20      $scope.answerTypes = [];
21      $scope.dynamicAnswerSheetControl = {};
22      $scope.isLecturer = false;
23      $scope.questionTitle = "";
24
25      $scope.$on("setQuestionJson", function (event, args) {
26          $scope.questionId = args.questionId;
27          $scope.isLecturer = args.isLecturer;
28          $scope.questionJson = args.questionJson;
29          $scope.questionTitle = args.questionJson.TITLE;
30
31          $scope.dynamicAnswerSheetControl.createAnswer();
32      });
33
34      /**
35       * FILL WITH SUITABLE TEXT
36       * @memberof module:answerToQuestionController
37       */
38      $scope.answer = function () {
39          $scope.dynamicAnswerSheetControl.answerToQuestion();
40          if($scope.isLecturer) {
41              $rootScope.$broadcast("lecturerAnswered");
42          }
43
44      };
45
46      /**
47       * FILL WITH SUITABLE TEXT
48       * @memberof module:answerToQuestionController
49       */
50      $scope.close = function () {
51          $scope.dynamicAnswerSheetControl.closeQuestion();
52      };
53
54      /**
55       *
56       */
```

```
57      $scope.showAnswers = function () {
58          $scope.$emit('showAnswers', true);
59      };
60  }]);
```

File: tulostettavat/static/scripts/controllers/lectureInfoController.js

```
1   /**
2    * Created by hajoviin on 11.5.2015.
3    * Handles the controls of lecture info page.
4    * @module lectureInfoController
5    * @author Matias Berg
6    * @author Bek Eljurkaev
7    * @author Minna Lehtomäki
8    * @author Juhani Sihvonen
9    * @author Hannu Viinikainen
10   * @licence MIT
11   * @copyright 2015 Timppa project authors
12   */
13
14  var angular, docId, lectureId, lectureCode, lectureStartTime,
lectureEndTime;
15
16  var timApp = angular.module('timApp');
17  timApp.controller('LectureInfoController', ['$scope', '$http',
'$window', function ($scope, $http, $window) {
18      "use strict";
19      $scope.docId = docId;
20      $scope.lectureId = lectureId;
21      $scope.lectureCode = "Lecture info: " + lectureCode;
22      $scope.lectureStartTime = lectureStartTime;
23      $scope.lectureEndTime = lectureEndTime;
24      $scope.msg = "";
25      $scope.answers = "";
26      $scope.userName = "";
27      $scope.dynamicAnswerShowControls = [];
28      $scope.dynamicAnswerShowControl = {};
29      $scope.index = 0;
30      $scope.isLecturer = false;
31      $scope.answerers = [];
32      $scope.showPoints = false;
33      $scope.points = [];
34
35      /**
36       * Sends http request to get info about the specific lecture.
37       * @memberof module:lectureInfoController
38       */
39      $scope.getLectureInfo = function () {
40          $http({
41              url: '/getLectureInfo',
42              type: 'GET',
43              params: {lecture_id: $scope.lectureId}
44          })
45              .success(function (answer) {
46
47                  angular.forEach(answer.messages, function (msg) {
48                      $scope.msg = $scope.msg + msg.sender + " <" +
msg.time + ">: " + msg.message + "\n";
```

```
                });
                $scope.answers = answer.answers;
                for (var i = 0; i < answer.questions.length; i++) {
                    $scope.dynamicAnswerShowControls.push({});
                    $scope.points.push(0);
                }
                $scope.questions = answer.questions;
                $scope.isLecturer = answer.isLecturer;
                $scope.answerers = answer.answerers;
                $scope.userName = answer.userName;
            })
            .error(function () {
                $window.console.log("fail");
            });
    };

    /*
     Gets the lecture info when loading page.
     */
    $scope.getLectureInfo();

    /**
     * Sends http request to delete the lecture.
     * @memberof module:lectureInfoController
     */
    $scope.deleteLecture = function () {
        var confirmAnswer = $window.confirm("Do you really want to
delete this lecture?");
        if (confirmAnswer) {
            $http({
                url: '/deleteLecture',
                method: 'POST',
                params: {'doc_id': $scope.docId, lecture_id:
$scope.lectureId}
            })
                .success(function () {
                    window.history.back();
                })
                .error(function () {
                    $window.console.log("Failed to delete the
lecture");
                });
        }
    };

    /**
     * Draws charts from the answer of the current lecture.
     * @param userName Which users answers to shows. If undefined
shows from every user.
     * @memberof module:lectureInfoController
     */
    $scope.drawCharts = function (userName) {
        for (var p = 0; p < $scope.points.length; p++) {
            $scope.points[p] = 0;
        }
         $scope.showPoints = true;
         var user;
         if (typeof userName === 'undefined') {
             user = "";
         } else {
             user = userName;
```

```
106          }
107          var questionIndexes = [];
108          for (var i = 0; i <
$scope.dynamicAnswerShowControls.length; i++) {
109
$scope.dynamicAnswerShowControls[i].createChart(JSON.parse($scope.questi
ons[i].questionJson));
110              questionIndexes.push($scope.questions[i].question_id);
111          }
112
113          for (var j = 0; j < $scope.answers.length; j++) {
114              if (($scope.isLecturer && user === "") ||
$scope.answers[j].user_name === user) {
115
$scope.dynamicAnswerShowControls[questionIndexes.indexOf($scope.answers[
j].question_id)]
116                      .addAnswer([{"answer":
$scope.answers[j].answer}]);
117
$scope.points[questionIndexes.indexOf($scope.answers[j].question_id)] +=
$scope.answers[j].points;
118              }
119
120          }
121
122
123          if ($scope.answers.length <= 0) {
124              var elem = $("#infoBox");
125              elem.empty();
126              elem.append("No answers from this lecture");
127          }
128      };
129  }]);
```

File: tulostettavat/static/scripts/controllers/questionPreviewController.js

```
1  /**
2   * FILL WITH SUITABLE TEXT
3   * @module questionPreviewController
4   * @author Matias Berg
5   * @author Bek Eljurkaev
6   * @author Minna Lehtomäki
7   * @author Juhani Sihvonen
8   * @author Hannu Viinikainen
9   * @licence MIT
10   * @copyright 2015 Timppa project authors
11   */
12
13  var angular, docId, lectureId, lectureCode, lectureStartTime,
lectureEndTime;
14
15  var timApp = angular.module('timApp');
16  timApp.controller('QuestionPreviewController', ['$scope', '$window',
'$http', '$rootScope',
17      function ($scope, $window, http, $rootScope) {
18          //TODO parse json and set values from rows and columns to
scope variables
```

```
19          //TODO edit questionPreview.html to repeat rows and columns
20          "use strict";
21
22          /**
23           * FILL WITH SUITABLE TEXT
24           * @memberof module:questionPreviewController
25           */
26          $scope.editQuestion = function () {
27              $scope.close();
28              $rootScope.$broadcast("editQuestion", {"question_id":
$scope.qId, "json": $scope.json});
29
30          };
31
32          /**
33           * FILL WITH SUITABLE TEXT
34           * @memberof module:questionPreviewController
35           */
36          $scope.ask = function () {
37              http({
38                  url: '/askQuestion',
39                  method: 'POST',
40                  params: {
41                      lecture_id: $scope.lectureId,
42                      question_id: $scope.qId,
43                      doc_id: $scope.docId,
44                      buster: new Date().getTime()
45                  }
46              })
47                  .success(function () {
48                      $rootScope.$broadcast('askQuestion', {"json":
$scope.json, "questionId": $scope.qId});
49                      $scope.$emit('closeQuestionPreview');
50                  })
51                  .error(function (error) {
52                      $scope.$emit('closeQuestionPreview');
53                      $window.console.log(error);
54                  });
55          };
56
57          /**
58           * FILL WITH SUITABLE TEXT
59           * @memberof module:questionPreviewController
60           */
61          $scope.close = function () {
62              $scope.$emit('closeQuestionPreview');
63          };
64
65          /**
66           * FILL WITH SUITABLE TEXT
67           * @memberof module:questionPreviewController
68           */
69          $scope.deleteQuestion = function () {
70              var confirmDi = $window.confirm("Are you sure you want
to delete this question?");
71              if (confirmDi) {
72                  http({
73                      url: '/deleteQuestion',
74                      method: 'POST',
75                      params: {question_id: $scope.qId, doc_id:
$scope.docId}
```

```
                })
                    .success(function () {
                        $scope.$emit('closeQuestionPreview');
                        $window.console.log("Deleted question");
                        location.reload();
                    })
                    .error(function (error) {

                        $scope.$emit('closeQuestionPreview');
                        $window.console.log(error);
                    });

            }
        };
    }
]);
```

File: tulostettavat/static/scripts/controllers/showStatisticsToQuestionController.js

```
/**
 * Created by hajoviin on 6.5.2015.
 * FILL WITH SUITABLE TEXT
 * @module showStatisticsToQuestionController
 * @author Matias Berg
 * @author Bek Eljurkaev
 * @author Minna Lehtomäki
 * @author Juhani Sihvonen
 * @author Hannu Viinikainen
  * @licence MIT
  * @copyright 2015 Timppa project authors
  */

var angular;

var timApp = angular.module('timApp');
timApp.controller('ShowStatisticsToQuestionController', ['$scope', '$http', function ($scope) {
    "use strict";
    $scope.dynamicAnswerShowControl = {};
    $scope.canvas = "";
    $scope.questionTitle = "";
    $scope.lecturerAnswered = false;

    $scope.$on("closeAnswerSheetForGood", function() {
        $scope.$emit('closeAnswerShow');
       $scope.dynamicAnswerShowControl.close();
    });
    /**
     * Closes statistic window
     * @memberof module:showStatisticsToQuestionController
     */
    $scope.close = function () {
        $scope.$emit('closeAnswerShow');
        if($scope.lecturerAnswered) {
            $scope.dynamicAnswerShowControl.close();
        }
```

```
37      };
38
39      $scope.$on("lecturerAnswered", function () {
40         $scope.lecturerAnswered = true;
41      });
42
43      /**
44       * Adds answer to statistic directive
45       * @memberof module:showStatisticsToQuestionController
46       */
47      $scope.$on("putAnswers", function (event, answer) {
48          $scope.dynamicAnswerShowControl.addAnswer(answer.answers);
49      });
50
51      /**
52       * Creates chart based on question json.
53       * @memberof module:showStatisticsToQuestionController
54       */
55      $scope.$on("createChart", function (event, question) {
56          $scope.lecturerAnswered = false;
57          $scope.dynamicAnswerShowControl.createChart(question);
58          $scope.questionTitle = question.QUESTION;
59      });
60
61  }]);
62
```

File: tulostettavat/static/scripts/createLectureCtrl.js

```
1   /**
2    * Lecture creation controller which is used to handle and validate
the form data.
3    * @module createLectureCtrl
4    * @author Matias Berg
5    * @author Bek Eljurkaev
6    * @author Minna Lehtomäki
7    * @author Juhani Sihvonen
8    * @author Hannu Viinikainen
9    * @licence MIT
10    * @copyright 2015 Timppa project authors
11    */
12
13  var angular;
14
15  var timApp = angular.module('timApp');
16
17  timApp.controller("CreateLectureCtrl", ['$scope', "$http",
"$window",
18
19      function ($scope, $http, $window) {
20          "use strict";
21          $scope.showLectureCreation = false;
22          $scope.useDate = false;
23          $scope.useDuration = false;
24          $scope.dateChosen = false;
25          $scope.durationChosen = false;
26          $scope.durationHour = "";
27          $scope.durationMin = "";
```

```
28          $scope.lectureId = null;
29          $scope.dateCheck = false;
30          $scope.dueCheck = false;
31          $scope.error_message = "";
32          $scope.lectureCode = "";
33          $scope.password = "";
34          $scope.startDate = "";
35          $scope.startHour = "";
36          $scope.startMin = "";
37          $scope.earlyJoining = true;
38          var date = new Date();
39          $scope.editMode = false;
40          $scope.startDate = date.getDate() + "." + (date.getMonth() +
1) + "." + date.getFullYear();
41          angular.element('#startDate').datepicker({dateFormat:
'dd.m.yy'}); //calendar for start date initalized
42          angular.element('#endDate').datepicker({dateFormat:
'dd.m.yy'}); //calendar for start date initalized
43
44          /**
45           * Function called when new lecture form button is pressed.
46           * @memberof module:createLectureCtrl
47           */
48          $scope.$on('initLectureFormVals', function () {
49              $scope.initLectureForm();
50          });
51
52          $scope.$on('editLecture', function (event,data) {
53              $scope.lectureCode = data.lecture_name;
54              $scope.lectureId = data.lecture_id;
55              var splittedStart = data.start_date.split(" ");
56              var splittedStartTime = splittedStart[1].split(":");
57              var splittedStartDate = splittedStart[0].split("-");
58              $scope.startMin = splittedStartTime[1];
59              $scope.startHour = splittedStartTime[0];
60              $scope.startDate = splittedStartDate[2] + "." +
splittedStartDate[1] + "." + splittedStartDate[0];
61              var splittedEnd = data.end_date.split(" ");
62              var splittedEndTime = splittedEnd[1].split(":");
63              var splittedEndDate = splittedEnd[0].split("-");
64              $scope.enableDate2();
65              $scope.endMin = splittedEndTime[1];
66              $scope.endHour = splittedEndTime[0];
67              $scope.endDate = splittedEndDate[2] + "." +
splittedEndDate[1] + "." + splittedEndDate[0];
68              if(data.password !== undefined){
69                  $scope.password = data.password;
70              }
71              $scope.editMode = data.editMode;
72          });
73
74          /**
75           * Lecture form initialized to have the current date and
time as default starting time.
76           * @memberof module:createLectureCtrl
77           */
78          $scope.initLectureForm = function () {
79              $window.console.log("Lecture initialized.");
80              var date = new Date();
81              $scope.startDate = date.getDate() + "." +
(date.getMonth() + 1) + "." + date.getFullYear();
```

```
82              $scope.startHour = $scope.leftPadder(date.getHours(),
2);
83              $scope.startMin = $scope.leftPadder(date.getMinutes(),
2);
84              $scope.dueCheck = true;
85              $scope.earlyJoining = true;
86              $scope.enableDue2();
87          };
88
89          $scope.populateLectureForm = function(name) {
90              $scope.lectureCode = name;
91          };
92
93          /**
94           * This function is called if end date is selected. Sets
boolean values to reflect this choice and sets
95           * the default end time to 2 hours ahead of start time.
96           * @memberof module:createLectureCtrl
97           */
98          $scope.enableDate2 = function () {
99              $window.console.log($scope.startDate);
100              $scope.dateCheck = true;
101              $scope.dueCheck = false;
102              $scope.endDate = $scope.startDate;
103              $scope.endHour =
$scope.leftPadder(parseInt($scope.startHour) + 2, 2);
104              $scope.endMin = $scope.leftPadder($scope.startMin, 2);
105
106              $scope.useDate = true;
107              $scope.useDuration = false;
108              $scope.durationHour = "";
109              $scope.durationMin = "";
110          };
111
112          /**
113           * Function for enabling fields and buttons for "Duration"
and disabling them for "Use date". This function
114           * is called when duration is chosen and is chosen by
default.
115           * @memberof module:createLectureCtrl
116           */
117          $scope.enableDue2 = function () {
118              $scope.dateCheck = false;
119              $scope.dueCheck = true;
120              $scope.useDuration = true;
121              $scope.useDate = false;
122              $scope.endDay = "";
123              $scope.endMonth = "";
124              $scope.endYear = "";
125              $scope.endHour = "";
126              $scope.endMin = "";
127              $scope.durationHour = "02";
128              $scope.durationMin = "00";
129          };
130
131          /**
132           * Remove border from given element.
133           * @param element ID of the field whose border will be
removed.
134           * @memberof module:createLectureCtrl
135           */
```

```
136         $scope.defInputStyle = function (element) {
137             if (element !== null || !element.isDefined) {
138                 angular.element("#"+element).css("border", "");
139             }
140         };
141 
142         /**
143          * Checks if the value is between 0-23
144          * @param element The value of the element to be validated.
145          * @param val The ID of the input field so user can be
notified of the error.
146          * @memberof module:createLectureCtrl
147          */
148         $scope.isHour = function (element, val) {
149             if (element === "" || isNaN(element) || element > 23 ||
element < 0) {
150                 $scope.errorize(val, "Hour has to be between 0 and
23.");
151             }
152         };
153 
154         /**
155          * Checks if the value is between 0-59
156          * @param element The value of the element to be validated.
157          * @param val The ID of the input field so user can be
notified of the error.
158          * @memberof module:createLectureCtrl
159          */
160         $scope.isMinute = function (element, val) {
161             if (element === "" || isNaN(element) || element > 59 ||
element < 0) {
162                 $scope.errorize(val, "Minutes has to be between 0
and 59.");
163             }
164         };
165 
166         /**
167          * Checks if the number given is positive.
168          * @param element The value of the element to be validated.
169          * @param val The ID of the input field so user can be
notified of the error.
170          * @memberof module:createLectureCtrl
171          */
172         $scope.isPositiveNumber = function (element, val) {
173             if (element === "" || isNaN(element) || element < 0) {
174                 $scope.errorize(val, "Number has to be positive.");
175             }
176         };
177 
178         /**
179          * Function for validating that the date is of the format
dd.mm.yyyy.
180          * @param element The value of the element to be validated.
181          * @param val The ID of the input field so user can be
notified of the error.
182          * @memberof module:createLectureCtrl
183          */
184         $scope.isDateValid = function (element, val) {
185             var reg = new RegExp("^(0?[1-9]|[12][0-
9]|3[01])[.]((0?[1-9]|1[012])[.](19|20)?[0-9]{2})*$");
186             if (!reg.test(element)) {
```

```
187                 $scope.errorize(val, "Date is not of format
dd.mm.yyyy or date is invalid.");
188             }
189         };
190 
191         /**
192          * Translates date object to string in the form yyyy-mm-dd
hh:mm
193          * @param input JavaScript date object
194          * @param include_time Boolean value whether to include the
time or not
195          * @returns {string} yyyy-mm-dd( hh:mm)
196          * @memberof module:createLectureCtrl
197          */
198         $scope.dateObjectToString = function (input, include_time)
{
199             var output = $scope.leftPadder(input.getFullYear(), 4)
+ "-" +
200                 $scope.leftPadder(input.getMonth() + 1, 2) + "-" +
201                 $scope.leftPadder(input.getDate(), 2);
202             if (include_time) {
203                 output += " " + $scope.leftPadder(input.getHours(),
2) + ":" +
204                 $scope.leftPadder(input.getMinutes(), 2);
205             }
206             return output;
207         };
208         /**
209          * Creates a new date object with the specified date and
time
210          * @param date_to_be_validated Date as a string in either
seperated by dots or dashes.
211          * @param time_hours
212          * @param time_mins
213          * @param splitter Date seperator.
214          * @returns {Date} Returns a JavaScript date object.
215          * @memberof module:createLectureCtrl
216          */
217         $scope.translateToDateObject = function
(date_to_be_validated, time_hours, time_mins, splitter) {
218             var parms = date_to_be_validated.split(splitter);
219             var dd;
220             var mm;
221             var yyyy;
222 
223             if (splitter === ".") {
224                 dd = parseInt(parms[0], 10);
225                 mm = parseInt(parms[1], 10);
226                 yyyy = parseInt(parms[2], 10);
227             } else {
228                 dd = parseInt(parms[2], 10);
229                 mm = parseInt(parms[1], 10);
230                 yyyy = parseInt(parms[0], 10);
231             }
232             var hours = parseInt(time_hours,10);
233             var mins = parseInt(time_mins,10);
234             return new Date(yyyy, mm - 1, dd, hours, mins, 0);
235         };
236 
237         /**
238          * Function for creating a new lecture and validation.
```

```
239           * @memberof module:createLectureCtrl
240           */
241          $scope.submitLecture = function () {
242              if($scope.lectureId === undefined || $scope.lectureId
=== null) {
243                  $scope.editMode = false;
244              }
245              $scope.removeErrors();
246              $scope.isDateValid($scope.startDate, "startDate");
247              if($scope.error_message<=0) {
248                  $scope.start_date =
$scope.translateToDateObject($scope.startDate, $scope.startHour,
$scope.startMin,".");
249                  if ($scope.endDate !== undefined, $scope.useDate) {
250                      $scope.isDateValid($scope.endDate, "endDate");
251                      $scope.end_date =
$scope.translateToDateObject($scope.endDate, $scope.endHour,
$scope.endMin, ".");
252                  } else {
253                      $scope.end_date = "";
254                  }
255              }
256
257              /*This checks that "lecture code"-field is not empty.*/
258              if ($scope.lectureCode === "") {
259                  $scope.errorize("lCode", "Lecture name must be
entered!");
260              }
261
262              /*This checks that either "Use date" or "Duration" is
chosen for ending time.*/
263              if ($scope.dateCheck === false && $scope.dueCheck ===
false) {
264                  $scope.errorize("endInfo", "A date or duration must
be chosen.");
265              }
266              /*Checks that hours in starting and ending time are
between 0 and 23.
267               Checks that minutes in starting and ending time are
between 0 and 59*/
268              $scope.isHour($scope.startHour, "startHour");
269              $scope.isMinute($scope.startMin, "startMin");
270
271              if ($scope.start_date !== undefined) {
272                  var now_hours = date.getHours();
273                  var now_minutes = date.getMinutes();
274                  var now_date_object =
$scope.translateToDateObject($scope.dateObjectToString(date, false),
now_hours, now_minutes, "-");
275                  var lecture_starting_in_past = $scope.start_date -
now_date_object < 0;
276                  var lecture_ending_in_past;
277
278                  /* Checks that are run if end date is used*/
279                  if ($scope.useDate && $scope.end_date !==
undefined) {
280                      $scope.isHour($scope.endHour, "endHour");
281                      $scope.isMinute($scope.endMin, "endMin");
282                      if ($scope.end_date - $scope.start_date <
120000) {
283                          $scope.errorize("endDateDiv", "Lecture has
```

```
to last at least two minutes.");
                    }
                    $scope.endDateForDB =
$scope.dateObjectToString($scope.end_date, true);
                }

                /* Check that are run if duration is used. */
                if ($scope.useDuration) {
                    if ($scope.durationHour === "") {
                        $scope.durationHour = "00";
                    }
                    if ($scope.durationMin === "") {
                        $scope.durationMin = "00";
                    }
                    $scope.isPositiveNumber($scope.durationHour,
"durationHour");
                    $scope.isPositiveNumber($scope.durationMin,
"durationMin");
                    if ($scope.durationHour.length <= 0 &&
$scope.durationMin.length <= 0) {
                        $scope.errorize("durationDiv", "Please give
a duration.");
                    }
                    $scope.end_date = new
Date($scope.start_date.getTime());

$scope.end_date.setHours($scope.start_date.getHours() +
parseInt($scope.durationHour));

$scope.end_date.setMinutes($scope.start_date.getMinutes() +
parseInt($scope.durationMin));
                    if ($scope.end_date - $scope.start_date <
120000) {
                        $scope.errorize("durationDiv", "Lecture has
to last at least two minutes.");
                    }
                    $scope.endDateForDB =
$scope.dateObjectToString($scope.end_date, true);
                }
                var alert_message = "";
                lecture_ending_in_past = $scope.end_date -
now_date_object <= 0;
                /* Confirmations if lecture starts and/or ends
before the current date */
                if (lecture_starting_in_past && !$scope.editMode) {
                    alert_message += "Are you sure you want the
lecture to start before now? ";
                }
                if (lecture_ending_in_past && !$scope.editMode) {
                    alert_message += "Are you sure that the lecture
ends in the past or now and will not run?";
                }
                $window.console.log($scope.error_message.length);
                if (alert_message !== "" &&
$scope.error_message.length <= 0) {
                    if (!$window.confirm(alert_message)) {
                        if (lecture_starting_in_past) {
                            $scope.errorize("startInfo", "Please
select another date and time.");
                        }
                        if (lecture_ending_in_past) {
```

```
                            $scope.errorize("endInfo", "Please
select another date or duration.");
                        }
                    }
                }
            }
            /* If no errors save the lecture to the database */
            if ($scope.error_message <= 0) {
                if($scope.earlyJoining){
                    $scope.start_date = new Date($scope.start_date
- 900000); // adds 15 minutes
                }
                $scope.startDateForDB =
$scope.dateObjectToString($scope.start_date, true);
                $http({
                    url: '/createLecture',
                    method: 'POST',
                    params: {
                        'lecture_id': $scope.lectureId,
                        'doc_id': $scope.docId,
                        'lecture_code': $scope.lectureCode,
                        'password': $scope.password,
                        'start_date': $scope.startDateForDB,
                        'end_date': $scope.endDateForDB
                    }
                })
                    .success(function (answer) {
                        if($scope.editMode) {
                            $window.console.log("Lecture " +
answer.lectureId + " updated.");
                        }
                        else {
                            $window.console.log("Lecture created: "
+ answer.lectureId);
                        }
                        $scope.$parent.useWall = true;
                        $scope.$parent.useAnswers = true;
                        $scope.clearForm();
                        $scope.$emit("closeLectureForm", true);
                    })
                    .error(function (answer) {
                        $scope.error_message += answer.error;
                        $window.console.log(answer);
                    });
            }
        };

        /**
         * Changes the border of the element to red.
         * @param div_val The ID of the input field so user can be
notified of the error.
         * @param error_text Error text that will be printed if the
error occurs.
         * @memberof module:createLectureCtrl
         */
        $scope.errorize = function (div_val, error_text) {
            angular.element("#" + div_val).css('border', "1px solid
red");
            if (error_text.length > 0) {
                $scope.error_message += error_text + "<br />";
            }
```

```
        };

        /**
         * Calls defInputStyle for all the form elements.
         * @memberof module:createLectureCtrl
         */
        $scope.removeErrors = function () {
            $scope.error_message = "";

            var elementsToRemoveErrorsFrom = [
                "lCode",
                "startDate",
                "startHour",
                "startMin",
                "startInfo",
                "endInfo",
                "endDate",
                "endHour",
                "endMin",
                "endDateDiv",
                "durationDiv",
                "durationHour",
                "durationMin"
            ];
            for (var i = 0; i < elementsToRemoveErrorsFrom.length;
i++) {
                if (elementsToRemoveErrorsFrom[i] !== undefined) {

$scope.defInputStyle(elementsToRemoveErrorsFrom[i]);
                }
            }
        };

        /**
         *  Resets all the values of the form.
         *  @memberof module:createLectureCtrl
         */
        $scope.clearForm = function () {
            $scope.lectureCode = "";
            $scope.password = "";
            $scope.startDate = "";
            $scope.endDate = "";
            $scope.startHour = "";
            $scope.startMin = "";
            $scope.durationMin = "";
            $scope.durationHour = "";
            $scope.endHour = "";
            $scope.endMin = "";
            $scope.showLectureCreation = false;
            $scope.removeErrors();
            $scope.useDate = false;
            $scope.useDuration = true;
            $scope.dateChosen = false;
            $scope.durationChosen = false;
            $scope.dateCheck = false;
            $scope.lectureId = null;
        };

        /**
         * Function for adding preceding zeroes to a number to make
it of desired length.
```

```
436           * @param number Number to be padded.
437           * @param size How long should the number be.
438           * @returns {string} Returns the number in the padded
length.
439           * @memberof module:createLectureCtrl
440           */
441          $scope.leftPadder = function (number, size) {
442              var paddedNumber = "" + number;
443              var len = paddedNumber.length;
444              while (len < size) {
445                  paddedNumber = "0" + paddedNumber;
446                  len++;
447              }
448              return paddedNumber;
449          };
450
451          /**
452           * Function for cancelling the lecture creation.
453           * @memberof module:createLectureCtrl
454           */
455          $scope.cancelCreation = function () {
456              $scope.editMode = false;
457              $scope.$emit("closeLectureForm");
458              $scope.clearForm();
459          };
460      }
461  ]);
```

File: tulostettavat/static/scripts/directives/dynamicAnswerSheet.js

```
1   /**
2    * Created by localadmin on 25.5.2015.
3    * Directive for dynamic answer sheet. Sheet to answer lecture
questions.
4    * @module dynamicAnswerSheet
5    * @author Matias Berg
6    * @author Bek Eljurkaev
7    * @author Minna Lehtomäki
8    * @author Juhani Sihvonen
9    * @author Hannu Viinikainen
10   * @licence MIT
11   * @copyright 2015 Timppa project authors
12   */
13
14  var angular;
15
16  var timApp = angular.module('timApp');
17  timApp.directive('dynamicAnswerSheet', ['$interval', '$compile',
'$rootScope', function ($interval, $compile, $rootScope) {
18      "use strict";
19      return {
20          restrict: 'E',
21          replace: "true",
22          scope: {
23              control: '='
24          },
25          transclude: true,
26          link: function ($scope, $element) {
```

```
27              var promise;
28              var timeLeft;
29              var barFilled;
30              $scope.internalControl = $scope.control || {};
31              $scope.internalControl.createAnswer = function () {
32                  $scope.json = $scope.$parent.questionJson;
33                  var htmlSheet = "<div class = 'answerSheet'>";
34                  if ($scope.json.TYPE !== "true-false") {
35                      htmlSheet += "<h2>" + $scope.json.QUESTION +
"<h2>";
36                  }
37                  if ($scope.json.TIMELIMIT !== "") {
38                      htmlSheet += "<progress value='0' max='" +
$scope.json.TIMELIMIT + "' id='progressBar'>";
39                      htmlSheet += "</progress>";
40                      htmlSheet += "<span id='progressLabel'>" +
$scope.json.TIMELIMIT + "</span>";
41                  }
42
43                  htmlSheet += "<table id='answer-sheet-table'>";
44
45                  if ($scope.json.TYPE === "true-false") {
46                      $scope.json.DATA.HEADERS[0] = {"type": "header",
"id": 0, "text": "True"};
47                      $scope.json.DATA.HEADERS[1] = {"type": "header",
"id": 1, "text": "False"};
48                  }
49
50                  if ($scope.json.DATA.HEADERS.length > 0 &&
!($scope.json.DATA.HEADERS[0].text === "" &&
$scope.json.DATA.HEADERS.length === 1)) {
51                      htmlSheet += "<tr>";
52                      if($scope.json.DATA.HEADERS.length > 1) {
53                          htmlSheet += "<th></th>";
54                      }
55                      angular.forEach($scope.json.DATA.HEADERS,
function (header) {
56                          htmlSheet += "<th class='answer-button'>" +
header.text + "</th>";
57                      });
58                      htmlSheet += "</tr>";
59                  }
60
61                  angular.forEach($scope.json.DATA.ROWS, function
(row) {
62                      htmlSheet += "<tr>";
63                      if ($scope.json.TYPE === "matrix" ||
$scope.json.TYPE === "true-false") {
64                          if (row.text.length >= 1) {
65                              htmlSheet += "<td>" + row.text +
"</td>";
66                          }
67                      }
68                      var header = 0;
69                      //TODO: Needs correct JSON to be made better way
70                      for (var i = 0; i < row.COLUMNS.length; i++) {
71                          var group;
72                          if ($scope.json.TYPE === "matrix" ||
$scope.json.TYPE === "true-false") {
73                              if (row.COLUMNS[i].answerFieldType ===
"text") {
```

```
                                    group = "group" + i;
                                    htmlSheet += "<td><label> <textarea
id='textarea-answer' name='" + group + "'"
                                    if($scope.json.DATA.HEADERS[0].text
=== "" && $scope.json.DATA.HEADERS.length === 1 &&
$scope.json.DATA.ROWS.length ===1) {
                                        htmlSheet +=
"style='height:200px'";
                                    }
                                    htmlSheet +=
"></textarea></label></td>";
                                    header++;
                                } else {
                                    group = "group" +
row.text.replace(/[^a-zA-Z0-9]/g, "");
                                    htmlSheet += "<td class='answer-
button'><label> <input type='" + row.COLUMNS[i].answerFieldType + "'
name='" + group + "'" +
                                    " value='" +
$scope.json.DATA.HEADERS[header].text + "'" +
                                    "></label></td>";
                                    header++;
                                }
                            } else {
                                group = "group" + row.type.replace(/[^a-
zA-Z0-9]/g, "");
                                htmlSheet += "<td class='answer-
button'><label> <input type='" + row.COLUMNS[i].answerFieldType + "'
name='" + group + "'" +
                                " value='" + row.text + "'" +
                                ">" + row.text + "</label></td>";
                            }
                        }
                        htmlSheet += "</tr>";
                    });


                    htmlSheet += "</div>";
                     $element.append(htmlSheet);
                     $compile($scope);

                     var fakeTime = $scope.json.TIMELIMIT * 1000;
                     timeLeft = $scope.json.TIMELIMIT;
                     barFilled = 0;
                     var timeBetween = 100;
                     var intervalTimes = fakeTime / timeBetween;
                     $scope.valChange = fakeTime / 1000 / intervalTimes;
                     $scope.progressElem = $("#progressBar");
                     $scope.progressText = $("#progressLabel");
                     $scope.start = function () {
                         promise =
$interval($scope.internalControl.updateBar, timeBetween, intervalTimes);
                     };
                     $scope.start();

                }
                ;

                /**
                 * FILL WITH SUITABLE TEXT
                 * @memberof module:dynamicAnswerSheet
```

```
122              */
123             $scope.internalControl.updateBar = function () {
124                 //TODO: Problem with inactive tab.
125                 timeLeft -= $scope.valChange;
126                 barFilled += $scope.valChange;
127                 $scope.progressElem.attr("value", (barFilled));
128 
129                 $scope.progressText.text(timeLeft.toFixed(0) + "
s");
130                 if (Math.abs(barFilled -
$scope.progressElem.attr("max")) < 0.02) {
131 
132                     if (!$scope.$parent.isLecturer) {
133                         $scope.internalControl.answerToQuestion();
134                     } else {
135                         clearInterval(promise);
136                         $scope.progressText.text("Time's up");
137                     }
138                 }
139 
140             };
141 
142             /**
143              * FILL WITH SUITABLE TEXT
144              * @memberof module:dynamicAnswerSheet
145              */
146             $scope.internalControl.answerToQuestion = function () {
147                 var answers = [];
148                 $interval.cancel(promise);
149                 if (angular.isDefined($scope.json.DATA.ROWS)) {
150                     var groupName = "";
151                     if ($scope.json.TYPE === "matrix" ||
$scope.json.TYPE === "true-false") {
152                         for (var i = 0; i <
$scope.json.DATA.ROWS.length; i++) {
153                             var answer = [];
154                             var matrixInputs;
155                             groupName = "group" +
$scope.json.DATA.ROWS[i].text.replace(/[^a-zA-Z0-9]/g, '');
156 
157                             if
($scope.json.DATA.ROWS[0].COLUMNS[0].answerFieldType === "text") {
158                                 matrixInputs = $('textarea[name='
+"group"+i + ']');
159                                 for (var c = 0; c <
matrixInputs.length; c++) {
160 
answer.push(matrixInputs[c].value);
161                                 }
162 
163                                 answers.push(answer);
164                                 continue;
165                             }
166 
167                             matrixInputs = $('input[name=' +
groupName + ']:checked');
168 
169                             for (var k = 0; k <
matrixInputs.length; k++) {
170                                 answer.push(matrixInputs[k].value);
171                             }
```

```
                                if (matrixInputs.length <= 0) {
                                    answer.push("undefined");
                                }
                                answers.push(answer);
                            }
                        }
                        else {
                            groupName = "group" +
$scope.json.DATA.ROWS[0].type.replace(/[^a-zA-Z0-9]/g, '');
                            var checkedInputs = $('input[name=' +
groupName + ']:checked');
                            for (var j = 0; j < checkedInputs.length;
j++) {
                                answers.push(checkedInputs[j].value);
                            }

                            if (checkedInputs.length <= 0) {
                                answers.push("undefined");
                            }
                        }
                    }

                    if (angular.isDefined($scope.json.DATA.COLUMNS)) {
                        angular.forEach($scope.json.DATA.COLUMNS,
function (column) {
                            var groupName = "group" +
column.Value.replace(/ /g, '');
                            answers.push($('input[name=' + groupName +
']:checked').val());
                        });
                    }

                    $element.empty();
                    $scope.$emit('answerToQuestion', {answer: answers,
questionId: $scope.$parent.questionId});
                    clearInterval(promise);
                };

                /**
                 * FILL WITH SUITABLE TEXT
                 * @memberof module:dynamicAnswerSheet
                 */
                $scope.internalControl.closeQuestion = function () {
                    clearInterval(promise);
                    $element.empty();
                    $scope.$emit('closeQuestion');
                    $rootScope.$broadcast('closeAnswerSheetForGood');
                };
            }

        };
    }
    ])
    ;
```

File: tulostettavat/static/scripts/directives/popUpDialog.js

```
1  /**
```

```
 * FILL WITH SUITABLE TEXT
 * @module popUpDialog
 * @author Matias Berg
 * @author Bek Eljurkaev
 * @author Minna Lehtomäki
 * @author Juhani Sihvonen
 * @author Hannu Viinikainen
 * @licence MIT
  * @copyright 2015 Timppa project authors
  */

timApp.directive('popUpDialog', function () {
    var toDragOrNot = function(){
        var width = (window.innerWidth > 0) ? window.innerWidth :
screen.width;
        if(width> 500)
        {
            return "tim-draggable-fixed";
        }
    };
    return {
        restrict: 'E',
        template: "<div class='pop-up' ng-show='show'
id='popUpBack'>"+
        "<div class='pop-up-overlay'></div> " +
        "<div class='pop-up-dialog' " + toDragOrNot() + " ng-
mousedown='checkDown($event)' ng-mouseup='checkUp($event)'
style='top:0px;'>" +
        "<div class='pop-up-dialog-content' ng-transclude></div>" +
        "</div>" +
        "</div>",

        scope: {
            show: '='
        },
        replace: true,
        transclude: true,

        link: function ($scope, $element) {

            /**
             * FILL WITH SUITABLE TEXT
             * @memberof module:popUpDialog
             * @param e
             */
            $scope.checkDown = function (e) {
                $scope.mouseDownX = e.clientX;
                $scope.mouseDownY = e.clientY;
                var window = $element.find("popUpBack");

                window.context.style.position = "absolute";
                window.context.style.bottom = 'auto';

            };

            /**
             * FILL WITH SUITABLE TEXT
             * @memberof module:popUpDialog
             */
            $scope.checkUp = function () {
                var window = $element.find("popUpBack");
```

```
                window.context.style.position = "fixed";
            };
        }

    };
});
```

File: tulostettavat/static/scripts/directives/showChartDirective.js

```
/**
 * Created by hajoviin on 13.5.2015.
 * FILL WITH SUITABLE TEXT
 * @module showChartDirective
 * @author Matias Berg
 * @author Bek Eljurkaev
 * @author Minna Lehtomäki
 * @author Juhani Sihvonen
 * @author Hannu Viinikainen
  * @licence MIT
  * @copyright 2015 Timppa project authors
  */



/*global $:false */
/*global Chart:false */
var angular;

var timApp = angular.module('timApp');
timApp.directive('showChartDirective', ['$compile', function
($compile) {
    "use strict";
    return {
        restrict: 'E',
        scope: {
            canvas: '@',
            control: '='
        },
        link: function ($scope, $element) {
            $scope.internalControl = $scope.control || {};
            $scope.canvasId = "#" + $scope.canvas || "";
            $scope.isText = false;

            //TODO: If more than 12 choices this will break.
Refactor to better format.
            var basicSets = [
                {
                    label: "Answers",
                    fillColor: "rgba(0,220,0,0.2)",
                    strokeColor: "rgba(0,220,0,1)",
                    pointColor: "rgba(0,220,0,1)",
                    pointStrokeColor: "#fff",
                    pointHighlightFill: "#fff",
                    pointHighlightStroke: "rgba(0,220,0,1)",
                    data: []
                },
                {
                    label: "Answers",
```

```
                    fillColor: "rgba(220,0,0,0.2)",
                    strokeColor: "rgba(220,0,0,1)",
                    pointColor: "rgba(220,0,0,1)",
                    pointStrokeColor: "#fff",
                    pointHighlightFill: "#fff",
                    pointHighlightStroke: "rgba(220,0,0,1)",
                    data: []
                },

                {
                    label: "Answers",
                    fillColor: "rgba(0,0,220,0.2)",
                    strokeColor: "rgba(0,0,220,1)",
                    pointColor: "rgba(0,0,220,1)",
                    pointStrokeColor: "#fff",
                    pointHighlightFill: "#fff",
                    pointHighlightStroke: "rgba(0,0,220,1)",
                    data: []
                },
                {
                    label: "Answers",
                    fillColor: "rgba(0,220,220,0.2)",
                    strokeColor: "rgba(0,220,220,1)",
                    pointColor: "rgba(0,220,220,1)",
                    pointStrokeColor: "#fff",
                    pointHighlightFill: "#fff",
                    pointHighlightStroke: "rgba(0,220,220,1)",
                    data: []
                },
                {
                    label: "Answers",
                    fillColor: "rgba(220,0,220,0.2)",
                    strokeColor: "rgba(220,0,220,1)",
                    pointColor: "rgba(220,0,220,1)",
                    pointStrokeColor: "#fff",
                    pointHighlightFill: "#fff",
                    pointHighlightStroke: "rgba(0,0,220,1)",
                    data: []
                },
                {
                    label: "Answers",
                    fillColor: "rgba(220,220,220,0.2)",
                    strokeColor: "rgba(220,220,220,1)",
                    pointColor: "rgba(220,220,220,1)",
                    pointStrokeColor: "#fff",
                    pointHighlightFill: "#fff",
                    pointHighlightStroke: "rgba(220,220,220,1)",
                    data: []
                },
                 {
                    label: "Answers",
                    fillColor: "rgba(165,220,0,0.2)",
                     strokeColor: "rgba(165,220,0,1)",
                     pointColor: "rgba(165,220,0,1)",
                     pointStrokeColor: "#fff",
                     pointHighlightFill: "#fff",
                     pointHighlightStroke: "rgba(165,220,0,1)",
                     data: []
                 },
                  {
                     label: "Answers",
```

```
                    fillColor: "rgba(220,165,0,0.2)",
                    strokeColor: "rgba(220,165,0,1)",
                    pointColor: "rgba(220,165,0,1)",
                    pointStrokeColor: "#fff",
                    pointHighlightFill: "#fff",
                    pointHighlightStroke: "rgba(220,165,0,1)",
                    data: []
                },
                 {
                    label: "Answers",
                    fillColor: "rgba(0,165,220,0.2)",
                    strokeColor: "rgba(220,165,0,1)",
                    pointColor: "rgba(220,165,0,1)",
                    pointStrokeColor: "#fff",
                    pointHighlightFill: "#fff",
                    pointHighlightStroke: "rgba(220,165,0,1)",
                    data: []
                },
                 {
                    label: "Answers",
                    fillColor: "rgba(220,0,165,0.2)",
                    strokeColor: "rgba(220,0,165,1)",
                    pointColor: "rgba(220,0,165,1)",
                    pointStrokeColor: "#fff",
                    pointHighlightFill: "#fff",
                    pointHighlightStroke: "rgba(220,0,165,1)",
                    data: []
                },
                 {
                    label: "Answers",
                    fillColor: "rgba(30,0,75,0.2)",
                    strokeColor: "rgba(30,0,75,1)",
                    pointColor: "rgba(30,0,75,1)",
                    pointStrokeColor: "#fff",
                    pointHighlightFill: "#fff",
                    pointHighlightStroke: "rgba(30,0,75,1)",
                    data: []
                },
                 {
                    label: "Answers",
                    fillColor: "rgba(75,75,180,0.2)",
                    strokeColor: "rgba(75,75,180,1)",
                    pointColor: "rgba(75,75,180,1)",
                    pointStrokeColor: "#fff",
                    pointHighlightFill: "#fff",
                    pointHighlightStroke: "rgba(75,75,180,1)",
                    data: []
                }
            ];

            /**
             * FILL WITH SUITABLE TEXT
             * @memberof module:showChartDirective
             * @param question FILL WITH SUITABLE TEXT
             */
            $scope.internalControl.createChart = function
(question) {
                $scope.ctx =
$($scope.canvasId).get(0).getContext("2d");
                $scope.x = 10;
                $scope.y = 20;
```

```
                    if (typeof
question.DATA.ROWS[0].COLUMNS[0].answerFieldType !== "undefined" &&
question.DATA.ROWS[0].COLUMNS[0].answerFieldType === "text") {
                        $scope.isText = true;
                        return;
                    }
                    $scope.isText = false;
                    var labels = [];
                    var emptyData = [];
                    if (angular.isDefined(question.DATA.ROWS)) {
                        angular.forEach(question.DATA.ROWS, function
(row) {
                            labels.push(row.text);
                            emptyData.push(0);
                        });
                    }

                    if (angular.isDefined(question.DATA.COLUMNS)) {
                        angular.forEach(question.DATA.COLUMNS, function
(column) {
                            angular.forEach(column.ROWS, function (row)
{
                                labels.push(row.Value);
                                emptyData.push(0);
                            });
                        });
                    }

                    labels.push("No answer");
                    emptyData.push(0);


                    var usedDataSets = [];


                    if (question.TYPE === "true-false") {
                        question.DATA.HEADERS[0] = {"type": "header",
"id": 0, "text": "True"};
                        question.DATA.HEADERS[1] = {"type": "header",
"id": 1, "text": "False"};
                    }

                    if (question.TYPE === "matrix" || question.TYPE ===
"true-false") {
                        for (var i = 0; i <
question.DATA.ROWS[0].COLUMNS.length; i++) {
                            usedDataSets.push(basicSets[i]);
                            usedDataSets[i].data = emptyData;
                        }

                        for (i = 0; i < question.DATA.HEADERS.length;
i++) {
                            usedDataSets[i].label =
question.DATA.HEADERS[i].text;
                        }
                    } else {
                        usedDataSets.push(basicSets[0]);
                        usedDataSets[0].data = emptyData;
                    }


```

```
218                 var data = {
219                     labels: labels,
220                     datasets: usedDataSets
221                 };
222 
223                 $scope.answerChart = new
Chart($scope.ctx).Bar(data, {
224                     multiTooltipTemplate: "<%= datasetLabel %> -
<%= value %>"
225                 });
226 
227                 $compile($scope);
228             };
229 
230             /**
231              * FILL WITH SUITABLE TEXT
232              * @memberof module:showChartDirective
233              * @param answers FILL WITH SUITABLE TEXT
234              */
235             $scope.internalControl.addAnswer = function (answers) {
236                 if (!angular.isDefined(answers)) {
237                     return;
238                 }
239                 $scope.ctx.font = "20px Georgia";
240 
241                 for (var answerersIndex = 0; answerersIndex <
answers.length; answerersIndex++) {
242                     var onePersonAnswers =
answers[answerersIndex].answer.split("|");
243                     var datasets;
244                     if (!$scope.isText) {
245                         datasets = $scope.answerChart.datasets;
246                     }
247                     for (var a = 0; a < onePersonAnswers.length;
a++) {
248                         var singleAnswers =
onePersonAnswers[a].split(',');
249                         for (var sa = 0; sa < singleAnswers.length;
sa++) {
250                             var singleAnswer = singleAnswers[sa];
251 
252                             if ($scope.isText) {
253                                 $scope.ctx.fillText(singleAnswer,
$scope.x, $scope.y);
254                                 $scope.y += 20;
255                                 continue;
256                             }
257                             if (datasets.length === 1) {
258                                 for (var b = 0; b <
datasets[0].bars.length; b++) {
259                                     if (datasets[0].bars[b].label
=== singleAnswer) {
260                                         datasets[0].bars[b].value
+= 1;
261                                     }
262                                 }
263                             } else {
264                                 for (var d = 0; d <
datasets.length; d++) {
265                                     if (datasets[d].label ===
singleAnswer) {
```

```
                                        datasets[d].bars[a].value
+= 1;
                                    }
                                }
                            }

                            if (singleAnswer === "undefined") {
                                var helperBars =
$scope.answerChart.datasets[0].bars;
                                helperBars[helperBars.length -
1].value += 1;
                            }
                        }
                    }
                }

                if (!$scope.isText) {
                    $scope.answerChart.update();
                }

            };

            /**
             * FILL WITH SUITABLE TEXT
             * @memberof module:showChartDirective
             */
            $scope.internalControl.close = function () {
                $scope.ctx.clearRect(0, 0,
$($scope.canvasId)[0].width, $($scope.canvasId)[0].height);
                if (typeof $scope.answerChart !== "undefined") {
                    $scope.answerChart.destroy();
                }
                $($scope.canvasId).remove(); // this is my <canvas>
element
                $('#chartDiv').append('<canvas id=' +
$scope.canvasId.substring(1) + ' width="400" height="300"><canvas>');
                $element.empty();
            };
        }
    };
 }])
 ;
```

File: tulostettavat/static/scripts/lectureController.js

```
/**
 * Created by hajoviin on 24.2.2015.
 * Contoller that handles lectures.
 * @module lectureController
 * @author Matias Berg
 * @author Bek Eljurkaev
 * @author Minna Lehtomäki
 * @author Juhani Sihvonen
 * @author Hannu Viinikainen
  * @licence MIT
  * @copyright 2015 Timppa project authors
  */

```

```
14  var angular;
15
16  var timApp = angular.module('timApp');
17
18  //TODO: Painike, josta voisi hakea kysymyksiä.
19  //TODO: Button, to get questions and wall.
20  timApp.controller("LectureController", ['$scope', '$controller',
"$http", "$window", '$rootScope', '$timeout',
21
22      function ($scope, controller, http, $window, $rootScope,
$timeout) {
23          "use strict";
24          $scope.lectureStartTime = "";
25          $scope.lectureEndTime = "";
26          $scope.lectureName = "";
27          $scope.msg = "";
28          $scope.newMsg = "";
29          $scope.wallName = "Wall";
30          $scope.messageName = true;
31          $scope.messageTime = true;
32          $scope.newMessagesAmount = 0;
33          $scope.newMessagesAmountText = "";
34          $scope.showPoll = true;
35          $scope.polling = true;
36          $scope.requestOnTheWay = false;
37          $scope.showWall = false;
38          $scope.canStart = true;
39          $scope.canStop = false;
40          $scope.showLectureCreation = false;
41          $scope.lectures = [];
42          $scope.futureLectures = [];
43          $scope.futureLecture = "";
44          $scope.chosenLecture = "";
45          $scope.passwordQuess = "";
46          $scope.pollingLectures = [];
47          $scope.useDate = false;
48          $scope.useDuration = false;
49          $scope.dateChosen = false;
50          $scope.durationChosen = false;
51          $scope.durationHour = "";
52          $scope.durationMin = "";
53          $scope.isLecturer = false;
54          $scope.lectureId = null;
55          $scope.showAnswerWindow = false;
56          $scope.showStudentAnswers = false;
57          $scope.studentTable = [];
58          $scope.lecturerTable = [];
59          $scope.gettingAnswers = false;
60          $scope.answeredToLectureEnding = false;
61          $scope.showLectureEnding = false;
62          $scope.extendTime = "15";
63          $scope.lectureEnded = false;
64          $scope.showLectureForm = false;
65          $scope.showLectureOptions = false;
66          $scope.useQuestions = true;
67          $scope.useWall = true;
68          $scope.useAnswers = true;
69          $scope.wallMessages = [];
70
71          var header = $('#header');
72          header.css("display", "none");
```

```
73
74          var wall = $('#wall');
75          var htmlMessageList = $('#wallMessageList');
76
77          /**
78           * Makes http request to check if the current user is in
lecture.
79           * @memberof module:lectureController
80           */
81          $scope.checkIfInLecture = function () {
82              http({
83                  url: '/checkLecture',
84                  method: 'GET',
85                  params: {'doc_id': $scope.docId, 'buster': new
Date().getTime()}
86              })
87                  .success(function (answer) {
88                      if (answer.isInLecture) {
89                          $scope.showLectureView(answer);
90                      } else {
91                          $scope.showBasicView(answer);
92                      }
93                  });
94          };
95
96          /*
97           Checks if the user is in lecture when loading page.
98           */
99          $scope.checkIfInLecture();
100
101
102          /*
103           Listener of askQuestion event. Opens the question view and
answer view for the lecturer.
104           */
105          $scope.$on("askQuestion", function (event, data) {
106              $scope.showStudentAnswers = true;
107              if ($scope.useAnswers) {
108                  // Because of dynamic creation needs to wait 1ms to
ensure that the directice is made(maybe?)
109                  $timeout(function () {
110                      $rootScope.$broadcast("createChart",
data.json);
111                  }, 1);
112
113                  var answer = {"questionId": data.questionId};
114                  $scope.getLectureAnswers(answer);
115              }
116
117              $rootScope.$broadcast("setQuestionJson", {
118                  questionJson: data.json,
119                  questionId: data.questionId,
120                  isLecturer: $scope.isLecturer
121              });
122              $scope.showAnswerWindow = true;
123          });
124
125          /*
126           Event listener for getLectureId. Emits the lectureId back
127           */
128          $scope.$on('getLectureId', function () {
```

```
129             $scope.$emit('postLectureId', $scope.lectureId);
130         });
131 
132         /*
133          Event listener for getLecture. Emits the boolean value if
the user is in lecture
134          */
135         $scope.$on('getInLecture', function () {
136             $scope.$emit('postInLecture', $scope.inLecture);
137         });
138 
139         /*
140          Event listener for getIsLecturer event. Emits value if the
user is lecturer
141          */
142         $scope.$on('getIsLecturer', function () {
143             $scope.$emit('postIsLecturer', $scope.isLecturer);
144         });
145 
146         $scope.$on('showAnswers', function (x) {
147             $scope.showStudentAnswers = x;
148         });
149 
150         /*
151          Event listener for closeLectureForm. Closes lecture form.
152          */
153         $scope.$on('closeLectureForm', function (event, showWall) {
154             $scope.showLectureForm = false;
155             if (showWall) {
156                 $scope.useWall = true;
157             }
158             $scope.checkIfInLecture();
159         });
160 
161         /*
162          Event listener for closeQuestion. Closes quesition pop-up.
163          */
164         $scope.$on('closeQuestion', function () {
165             $scope.showAnswerWindow = false;
166         });
167 
168 
169         /*
170          Event listener for answerToQuesion. Closes the answer pop-
up and sends answer to server.
171          */
172         $scope.$on("answerToQuestion", function (event, answer) {
173             $scope.showAnswerWindow = false;
174             var mark = "";
175             var answerString = "";
176             angular.forEach(answer.answer, function (singleAnswer)
{
177                 answerString += mark + singleAnswer;
178                 mark = "|";
179             });
180 
181             http({
182                 url: '/answerToQuestion',
183                 method: 'PUT',
184                 params: {
185                     'question_id': answer.questionId,
```

```
                    'lecture_id': $scope.lectureId,
                    'answers': answerString,
                    'buster': new Date().getTime()
                }
            })
                .success(function (answer) {
                    if(angular.isDefined(answer.questionLate)) {
                        $window.alert(answer.questionLate);
                    }
                })
                .error(function () {
                    $window.console.log("Failed to answer to
question");
                });
        });


        /*
         Event window for closeAnswerShow. Closes pop-up to show
answers and stops gettin more of them.
         */
        //TODO: Also send event to server to remove getting answers
for this question.
        $scope.$on("closeAnswerShow", function () {
            $scope.showStudentAnswers = false;
            $scope.gettingAnswers = false;
        });

        /**
         * Depending on what users wants to see on the wall, makes
the msg to correct form. Able to show the
         * name of the sender, time and message. Sender and time
are optional.
         * @memberof module:lectureController
         */
        $scope.showInfo = function () {
            $scope.msg = "";
            var i = 0;
            if ($scope.messageName && $scope.messageTime) {
                for (i = 0; i < $scope.wallMessages.length; i++) {
                    $scope.msg += $scope.wallMessages[i].sender;
                    $scope.msg += " <" +
$scope.wallMessages[i].time + ">: ";
                    $scope.msg += $scope.wallMessages[i].message +
"\r\n";
                }
            }

            if (!$scope.messageName && $scope.messageTime) {
                for (i = 0; i < $scope.wallMessages.length; i++) {
                    $scope.msg += " <" +
$scope.wallMessages[i].time + ">: ";
                    $scope.msg += $scope.wallMessages[i].message +
"\r\n";
                }
            }

            if ($scope.messageName && !$scope.messageTime) {
                for (i = 0; i < $scope.wallMessages.length; i++) {
                    $scope.msg += $scope.wallMessages[i].sender +
": ";
```

```
237                     $scope.msg += $scope.wallMessages[i].message +
"\r\n";
238                 }
239             }
240
241             if (!$scope.messageName && !$scope.messageTime) {
242                 for (i = 0; i < $scope.wallMessages.length; i++) {
243                     $scope.msg += ">" +
$scope.wallMessages[i].message + "\r\n";
244                 }
245             }
246         };
247
248         /**
249          * Clears the password input when changing the lecture from
current lectures list.
250          * @memberof module:lectureController
251          */
252         $scope.clearChange = function () {
253             $scope.passwordQuess = "";
254         };
255
256         /**
257          * Puts chosen lecture options to use.
258          * @memberof module:lectureController
259          * @param useQuestions Whether or not to display questions?
260          * @param useWall Whether or not to display the wall?
261          */
262         $scope.useOptions = function (useQuestions, useWall) {
263             $scope.showLectureView($scope.lectureAnswer);
264             $scope.useWall = useWall;
265             $scope.useQuestions = useQuestions;
266             $scope.showLectureOptions = false;
267
268         };
269
270         /**
271          * Ask lecture options from students that are coming to
lecture.
272          * @memberof module:lectureController
273          */
274         $scope.lectureOptions = function () {
275             if (!$scope.isLecturer) {
276                 $scope.showLectureOptions = true;
277             }
278             $scope.joinLecture();
279         };
280
281         /**
282          * Method to join selected lecture. Checks that lecture is
chosen and sends http request to server.
283          * @memberof module:lectureController
284          * @param name Name of the lecture to be joined.
285          */
286         $scope.joinLecture = function (name, code_required) {
287             if(code_required) $scope.passwordQuess =
$window.prompt("Please enter a password:", "");
288             if ($scope.chosenLecture === "" && name === "") {
289                 $window.alert("Choose lecture to join");
290                 return;
291             }
```

```
292
293             var lectureName = "";
294             if (angular.isDefined(name)) {
295                 lectureName = name;
296                 $('#currentList').slideToggle();
297             } else {
298                 lectureName = $scope.chosenLecture.lecture_code;
299             }
300
301             http({
302                 url: '/joinLecture',
303                 method: 'POST',
304                 params: {
305                     'doc_id': $scope.docId,
306                     'lecture_code': lectureName,
307                     'password_quess': $scope.passwordQuess
308                 }
309             })
310                 .success(function (answer) {
311                     $scope.passwordQuess = "";
312                     var input = $("#passwordInput");
313                     if (!answer.correctPassword) {
314                         $window.alert("Wrong access code!");
315                     } else {
316                         input.removeClass('errorBorder');
317                         input.attr("placeholder", "Access code");
318                         if ($scope.isLecturer) {
319                             $scope.showLectureView(answer);
320                             $scope.useWall = true;
321                             $scope.useQuestions = true;
322                         } else {
323                             $scope.showLectureOptions = true;
324                             $scope.lectureAnswer = answer;
325                         }
326                     }
327                 });
328         };
329
330         /**
331          * Checks where the mouse is when clicking the wall. Sets
style to absolute for the draggable directive.
332          * @param e Event of the mouse click.
333          * @memberof module:lectureController
334          */
335         $scope.checkDown = function (e) {
336             $scope.mouseDownX = e.clientX;
337             $scope.mouseDownY = e.clientY;
338             wall.position("absolute");
339             wall.css("bottom", "auto");
340             $scope.wallMoved = true;
341
342         };
343
344         /**
345          * Checks where th mouse is when releasing the wall. If the
mouse hasn't moved enough, closes the wall.
346          * @param e Event of the mouse release.
347          * @memberof module:lectureController
348          */
349         $scope.checkUp = function (e) {
350             var mouseUpX = e.clientX;
```

```
351             var mouseUpY = e.clientY;
352             wall.position("fixed");
353 
354             if (Math.abs(Math.sqrt(Math.pow(($scope.mouseDownX -
mouseUpX), 2) + Math.pow(($scope.mouseDownY - mouseUpY), 2))) < 10) {
355                 $scope.wallMoved = false;
356             }
357         };
358 
359         /**
360          * Hides the wall if the wall hasn't moved.
361          * @memberof module:lectureController
362          */
363         $scope.hideWall = function () {
364             if (!$scope.wallMoved) {
365                 $scope.hide();
366             }
367         };
368 
369         /**
370          * Toggle the lecture creation form.
371          * @memberof module:lectureController
372          */
373         $scope.toggleLecture = function () {
374             $('#currentList').hide();
375             $('#futureList').hide();
376             $scope.showLectureForm = true;
377             $rootScope.$broadcast("initLectureFormVals");
378         };
379 
380         /**
381          * Starts lecture that is in future lecture list.
382          * @memberof module:lectureController
383          */
384         $scope.startFutureLecture = function () {
385             http({
386                 url: '/startFutureLecture',
387                 method: 'POST',
388                 params: {'doc_id': $scope.docId, 'lecture_code':
$scope.futureLecture.lecture_code}
389             })
390                 .success(function (answer) {
391                     $scope.showLectureView(answer);
392                 });
393         };
394 
395         /**
396          * Change the usage of wall.
397          * @param wallUsage Whether wall should be displayed or
not.
398          * @memberof module:lectureController
399          */
400         $scope.changeUsingWall = function (wallUsage) {
401             $scope.useWall = wallUsage;
402         };
403 
404         /**
405          * Change the usage of getting lecture questions.
406          * @param questionUsage Whether questions should be
displayed or not.
407          * @memberof module:lectureController
```

```
408          */
409         $scope.changeUsingQuestions = function (questionUsage) {
410             $scope.useQuestions = questionUsage;
411         };
412 
413         /**
414          * Changes the usage of getting answers from students.
415          * @param answerUsage Whether to get answers from students
or not.
416          * @memberof module:lectureController
417          */
418         $scope.changeUsingAnswers = function (answerUsage) {
419             $scope.useAnswers = answerUsage;
420         };
421 
422         /**
423          * Initializes the window to be lecture view a.k.a. the
current user in in lecture.
424          * @param lecture The lecture to be shown.
425          * @memberof module:lectureController
426          */
427         $scope.showLectureView = function (lecture) {
428             $scope.isLecturer = lecture.isLecturer;
429 
430             $scope.lectureName = lecture.lectureCode;
431             $scope.wallName = "Wall - " + lecture.lectureCode;
432             if ($scope.wallName.length > 30) {
433                 $scope.wallName = $scope.wallName.substring(0, 30)
+ "...";
434             }
435             $scope.lectureStartTime = "Started: " +
lecture.startTime;
436             $scope.lectureEndTime = "Ends: " + lecture.endTime;
437             $scope.inLecture = true;
438             $scope.lectureId = lecture.lectureId;
439             $scope.polling = true;
440             $scope.msg = "";
441             $scope.showWall = true;
442             $scope.useWall = lecture.useWall;
443             $scope.useQuestions = lecture.useQuestions;
444 
445             $scope.getAllMessages();
446 
447 
448             if ($scope.isLecturer) {
449                 $rootScope.$broadcast("getQuestions");
450                 $scope.canStop = true;
451                 $scope.addPeopleToList(lecture.students,
$scope.studentTable);
452                 $scope.addPeopleToList(lecture.lecturers,
$scope.lecturerTable);
453             }
454         };
455 
456         /**
457          * Adds people entities to give list.
458          * @param people name: String, active:String
459          * @param peopleList List of people in the lecture.
460          * @memberof module:lectureController
461          */
462         $scope.addPeopleToList = function (people, peopleList) {
```

```
463             var oldUser = false;
464             for (var i = 0; i < people.length; i++) {
465                 for (var index = 0; index < peopleList.length;
index++) {
466                     if (peopleList[index].name === people[i].name)
{
467                         oldUser = true;
468                         peopleList[index].active =
people[i].active;
469                         break;
470                     }
471                 }
472 
473                 if (!oldUser) {
474                     var student = {
475                         name: people[i].name,
476                         active: people[i].active
477                     };
478                     peopleList.push(student);
479                 }
480             }
481         };
482 
483         /**
484          * Initializes the window to be basic view a.k.a. view
where user is not in lecture.
485          * @param answer The lecture to be processed.
486          * @memberof module:lectureController
487          */
488         $scope.showBasicView = function (answer) {
489 
490             $scope.isLecturer = answer.isLecturer;
491             if ($scope.isLecturer) {
492                 $rootScope.$broadcast("getQuestions");
493                 $scope.canStart = true;
494                 $scope.canStop = false;
495             }
496             $scope.wallMessages = [];
497             $scope.useWall = false;
498             $scope.polling = false;
499             $scope.inLecture = false;
500             $scope.lectureId = -1;
501             $scope.lectureName = "Not running";
502             $scope.showStudentAnswers = false;
503             $scope.showAnswerWindow = false;
504             $scope.lecturerTable = [];
505             $scope.studentTable = [];
506             $scope.lectures = [];
507             $scope.futureLectures = [];
508 
509 
510             var addLecture = true;
511             for( var i = 0; i < answer.lectures.length; i++) {
512                 $scope.lectures.push(answer.lectures[i]);
513             }
514 
515             for(  i = 0; i < answer.futureLectures.length; i++) {
516 
$scope.futureLectures.push(answer.futureLectures[i]);
517             }
518 
```

```
519             if ($scope.lectures.length > 0) {
520                 $scope.chosenLecture = $scope.lectures[0];
521             }
522 
523             if ($scope.futureLectures.length > 0) {
524                 $scope.futureLecture = $scope.futureLectures[0];
525             }
526         };
527 
528         /**
529          * Extends the lecture based on the time selected in pop-up
to extend lecture.
530          * Currently extends to the old lecture ending time. Other
option is to extend
531          * from the current moment(needs to be implemented).
532          * @memberof module:lectureController
533          */
534         $scope.extendLecture = function () {
535             var dateTime = $scope.lectureEndTime.split(" ");
536 
537             //TODO: Use javascript date object.
538             var YearMonthDay = dateTime[1].split("-");
539             var endYear = parseInt(YearMonthDay[0]);
540             var endMonth = parseInt(YearMonthDay[1]);
541             var endDay = parseInt(YearMonthDay[2]);
542 
543             var hoursMins = dateTime[2].split(":");
544             var endHour = parseInt(hoursMins[0]);
545             var endMinutes = parseInt(hoursMins[1]);
546 
547 
548             endMinutes += parseInt($scope.extendTime);
549 
550             if (endMinutes >= 60) {
551                 endHour += 1;
552                 endMinutes -= 60;
553                 if (endHour >= 24) {
554                     endDay += 1;
555                     endHour -= 24;
556                     switch (endMonth) {
557                         case 1:
558                         case 3:
559                         case 5:
560                         case 7:
561                         case 8:
562                         case 10:
563                         case 12:
564                             if (endDay > 31) {
565                                 endDay = 1;
566                                 endMonth += 1;
567                             }
568                             break;
569                         case 2:
570                             if (endDay > 28) {
571                                 endDay = 1;
572                                 endMonth += 1;
573                             }
574                             break;
575                         default:
576                             if (endDay > 30) {
577                                 endDay = 1;
```

```
                                endMonth += 1;
                            }
                        }
                        if (endMonth > 12) {
                            endMonth = 1;
                            endYear += 1;
                        }
                    }
                }


                var endTimeDate = endYear + "-" +
$scope.leftPadder(endMonth, 2) + "-" + $scope.leftPadder(endDay, 2) + "
" +
                    $scope.leftPadder(endHour, 2) + ":" +
$scope.leftPadder(endMinutes, 2);
                $scope.lectureEndTime = "Ends: " + endTimeDate;
                $scope.showLectureEnding = false;
                $scope.lectureEnded = false;
                $scope.answeredToLectureEnding = true;

                http({
                    url: '/extendLecture',
                    method: 'POST',
                    params: {'doc_id': $scope.docId, lecture_id:
$scope.lectureId, new_end_time: endTimeDate}
                })
                    .success(function () {
                        $scope.answeredToLectureEnding = false;
                        $window.console.log("Lecture extended");
                    })
                    .error(function () {
                        $window.console.log("Failed to delete the
lecture");
                    });
            };

            /**
             * Closes the lecture view and sets answered to
lectureEnding to true to prevent multiple quesitions from this.
             * @memberof module:lectureController
             */
            $scope.continueLecture = function () {
                $scope.answeredToLectureEnding = true;
                $scope.showLectureEnding = false;
            };

            /**
             * NOT IMPLEMENTED YET
             * @memberof module:lectureController
             */
            $scope.editLecture = function (lecture_code) {
                $('#currentList').hide();
                $('#futureList').hide();
                http({
                    url: '/showLectureInfoGivenName',
                    method: 'GET',
                    params: {'lecture_code': lecture_code, 'doc_id':
$scope.docId}
                })
                    .success(function(lecture) {
```

```
                    $rootScope.$broadcast("editLecture",
{"lecture_id": lecture.lectureId, "lecture_name": lecture.lectureCode,
"start_date": lecture.lectureStartTime, "end_date":
lecture.lectureEndTime, "password": "", "editMode": true});
                    $scope.showLectureForm = true;
                })
                .error(function() {
                    $window.console.log("Failed to fetch
lecture.");
                });
        };

        /**
         * Sends http request to end the lecture.
         * @memberof module:lectureController
         */
        $scope.endLecture = function () {
            $scope.showLectureEnding = false;

            // TODO: Change to some better confirm dialog.
            var confirmAnswer = $window.confirm("Do you really want
to end this lecture?");
            if (confirmAnswer) {
                http({
                    url: '/endLecture',
                    method: 'POST',
                    params: {'doc_id': $scope.docId, lecture_id:
$scope.lectureId}
                })
                    .success(function (answer) {
                        $scope.showBasicView(answer);
                        $scope.chosenLecture = "";
                        $scope.msg = "";
                        $window.console.log("Lecture ended, not
deleted");

                    })
                    .error(function () {
                        $window.console.log("Failed to delete the
lecture");
                    });
            }
        };

        /**
         * Sends http request to delete the lecture.
         * @memberof module:lectureController
         */
        $scope.deleteLecture = function () {
            http({
                url: '/deleteLecture',
                method: 'POST',
                params: {'doc_id': $scope.docId, lecture_id:
$scope.lectureId}
            })
                .success(function (answer) {
                    $scope.showBasicView(answer);
                    $scope.lectures.splice($scope.lectureId, 1);
                    $scope.chosenLecture = "";
                    $scope.msg = "";
                    $window.console.log("Lecture deleted");
```

```
684
685                 })
686                 .error(function () {
687                     $window.console.log("Failed to delete the
lecture");
688                 });
689         };
690
691         /**
692          * Sends http request to leave the lecture.
693          * @memberof module:lectureController
694          */
695         $scope.leaveLecture = function () {
696             //TODO: better confirm dialog
697             var confirmAnswer = false;
698             if ($scope.isLecturer) {
699                  confirmAnswer = $window.confirm("Do you really
want to leave this lecture?");
700             }
701
702             if (!$scope.isLecturer || confirmAnswer) {
703                 $scope.msg = "";
704                 http({
705                     url: '/leaveLecture',
706                     method: "POST",
707                     params: {'lecture_id': $scope.lectureId,
'doc_id': $scope.docId}
708                 })
709                     .success(function (answer) {
710                         $scope.showBasicView(answer);
711                     });
712             }
713         };
714
715         /**
716          * Hides the wall. After this you can only see the topbar
of the wall.
717          * @memberof module:lectureController
718          */
719         $scope.hide = function () {
720             $scope.showWall = !$scope.showWall;
721
722             if (!$scope.showWall) {
723                 $scope.wallHeight = wall.height();
724                 wall.height(28);
725                 $scope.newMessagesAmount = 0;
726                 $scope.newMessagesAmountText = "(" +
$scope.newMessagesAmount + ")";
727             } else {
728                 wall.height($scope.wallHeight);
729             }
730         };
731
732         /**
733          * Shows lecture creation. //TODO: Something is missing
from here
734          */
735         $scope.modifyLecture = function () {
736             $scope.showLectureCreation = true;
737         };
738
```

```
739          /**
740           * Sends http request to send a message.
741           * @param message The message to be sent.
742           * @returns {boolean} Whether the message was sent
successfully.
743           * @memberof module:lectureController
744           */
745          $scope.sendMessageEvent = function (message) {
746              if (message.trim() === "") {
747                  $window.alert("Can't send empty messages");
748                  return false;
749              }
750
751              http({
752                  url: '/sendMessage',
753                  method: 'POST',
754                  params: {'message': message, lecture_id:
$scope.lectureId}
755              })
756                  .success(function () {
757                      $scope.newMsg = "";
758                      //TODO: Fix this to scroll bottom without
cheating.
759                      var wallArea = $('#wallArea');
760                      wallArea.animate({scrollTop:
wallArea[0].scrollHeight * 10}, 1000);
761
762                  })
763                  .error(function () {
764                      $window.console.log("Can't send message or
something");
765                  });
766
767          };
768
769          /**
770           * Sends http request to get all the messages from the
current lecture.
771           * @memberof module:lectureController
772           */
773          $scope.getAllMessages = function () {
774              $scope.msg = "";
775              http({
776                  url: '/getAllMessages',
777                  type: 'GET',
778                  params: {lecture_id: $scope.lectureId, 'buster':
new Date().getTime()}
779              })
780                  .success(function (answer) {
781                      angular.forEach(answer.data, function (msg) {
782                          $scope.wallMessages.push(msg);
783                          if ($scope.messageName) {
784                              $scope.msg += msg.sender + " ";
785                          }
786                          if ($scope.messageTime) {
787                              $scope.msg += "<" + msg.time + ">: ";
788                          }
789
790                          $scope.msg += msg.message + "\n";
791                      });
792
```

```
                    //TODO: Fix this to scroll bottom without
cheating.
                    var wallArea = $('#wallArea');
                    wallArea.animate({scrollTop:
wallArea[0].scrollHeight * 10}, 1000);

                    $scope.lastID = answer.lastid;
                    $scope.requestDocId = $scope.lectureId;

                    if
($scope.pollingLectures.indexOf(answer.lectureId) === -1) {
                        $scope.startLongPolling($scope.lastID);

$scope.pollingLectures.push(answer.lectureId);
                    }
                });
        };

        /**
         * Gets answers from the current lecture to current
question.
         * @param answer Lecture to get the answers from.
         * @memberof module:lectureController
         */
        $scope.getLectureAnswers = function (answer) {
            $scope.gettingAnswers = true;
            http({
                url: '/getLectureAnswers',
                type: 'GET',
                params: {
                    'question_id': answer.questionId,
                    'doc_id': $scope.docId,
                    'lecture_id': $scope.lectureId,
                    'time': answer.latestAnswer,
                    'buster': new Date().getTime()
                }
            })
                .success(function (answer) {
                    $rootScope.$broadcast("putAnswers", {"answers":
answer.answers});
                    if ($scope.gettingAnswers &&
!angular.isDefined(answer.noAnswer)) {
                        $scope.getLectureAnswers(answer);
                    }

                })
                .error(function () {
                    $window.console.log("Couldn't get answers");
                });
        };

        /**
         * Starts long polling for the updates.(messages,
questions, lecture ending)
         * @param lastID Last id which was received.
         * @memberof module:lectureController
         */
        $scope.startLongPolling = function (lastID) {
            function message_longPolling(lastID) {
                var timeout;

```

```
                if (lastID === null) {
                    lastID = -1;
                }

                $scope.requestOnTheWay = true;
                http({
                    url: '/getUpdates',
                    type: 'GET',
                    params: {
                        'client_message_id': lastID,
                        'lecture_id': $scope.lectureId,
                        'doc_id': $scope.docId,
                        'is_lecturer': $scope.isLecturer, //
Tarkista mielummin serverin päässä
                        'get_messages': $scope.useWall,
                        'get_questions': $scope.useQuestions,
                        'buster': new Date().getTime()
                    }
                })
                    .success(function (answer) {

                        if (!answer.isLecture) {
                            $scope.showBasicView(answer);
                            return;
                        }

$scope.pollingLectures.splice($scope.pollingLectures.indexOf(answer.lect
ureId), 1);
                        if (answer.lectureId !== $scope.lectureId)
{
                            return;
                        }

                        if (answer.lectureEnding !== 100) {
                            if (answer.lectureEnding === 1 &&
!$scope.lectureEnded) {
                                $scope.showLectureEnding = true;
                                $scope.lectureEnded = true;
                            }
                            if (!$scope.answeredToLectureEnding) {
                                $scope.showLectureEnding = true;
                            }
                        }

                        $scope.addPeopleToList(answer.students,
$scope.studentTable);
                        $scope.addPeopleToList(answer.lecturers,
$scope.lecturerTable);


                        if (answer.question && !$scope.isLecturer)
{
                            $scope.showAnswerWindow = true;

$rootScope.$broadcast("setQuestionJson", {
                                questionJson:
JSON.parse(answer.questionJson),
                                questionId: answer.questionId,
                                isLecturer: $scope.isLecturer
                            });
                        }
```

```
897
898                             $scope.requestOnTheWay = false;
899                             $window.clearTimeout(timeout);
900                             if ($scope.polling) {
901
902
$scope.pollingLectures.push(answer.lectureId);
903                                 // Odottaa sekunnin ennen kuin pollaa
uudestaan.
904                                 timeout = setTimeout(function () {
905                                     message_longPolling(answer.lastid);
906                                 }, 1000);
907
908                                 if (answer.status === 'results') {
909                                     angular.forEach(answer.data,
function (msg) {
910                                         $scope.wallMessages.push(msg);
911                                         if ($scope.messageName &&
$scope.messageTime) {
912                                             $scope.msg += msg.sender;
913                                             $scope.msg += " <" +
msg.time + ">: ";
914                                             $scope.msg += msg.message +
"\r\n";
915                                         }
916
917                                         if (!$scope.messageName &&
$scope.messageTime) {
918                                             $scope.msg += " <" +
msg.time + ">: ";
919                                             $scope.msg += msg.message +
"\r\n";
920
921                                         }
922
923                                         if ($scope.messageName &&
!$scope.messageTime) {
924                                             $scope.msg += msg.sender +
": ";
925                                             $scope.msg += msg.message +
"\r\n";
926
927                                         }
928
929                                         if (!$scope.messageName &&
!$scope.messageTime) {
930                                             $scope.msg += ">" +
msg.message + "\r\n";
931
932                                         }
933                                     });
934                                     $scope.lastID = answer.lastid;
935                                     var wallArea = $('#wallArea');
936
wallArea.scrollTop(wallArea[0].scrollHeight);
937                                 } else {
938                                     $window.console.log("Sending new
poll.");
939
940                                 }
941                             } else {
```

```
                                $window.console.log("Got answer but not
polling anymore.");
                            }
                        })
                        .error(function () {
                            $scope.requestOnTheWay = false;
                            $window.clearTimeout(timeout);
                            //Odottaa 30s ennen kuin yrittää uudelleen
errorin jälkeen.
                            timeout = setTimeout(function () {
                                message_longPolling();
                            }, 30000);
                        });
                }

                message_longPolling(lastID);
            };

            /**
             * Event for pressing enter while writing message. Sends
message.
             * @param event They key press.
             * @memberof module:lectureController
             */
            $scope.chatEnterPressed = function (event) {
                if (event.which === 13) {
                    $scope.sendMessageEvent($scope.newMsg);
                }
            };

            /**
             * Event when pressing enter while writing password for
lecture. Tries to join lecture
             * @param event The key press.
             * @memberof module:lectureController
             */
            $scope.passEnterPressed = function (event) {
                if (event.which === 13) {
                    $scope.joinLecture();
                }
            };

            /**
             * Left padder that adds 0 to left side of number.
             * @param number Given number to add 0s
             * @param size how many characters the padded value should
be.
             * @returns {string} The string of the length specified
padded with zeroes.
             * @memberof module:lectureController
             */
            $scope.leftPadder = function (number, size) {
                var paddedNumber = "" + number;
                var len = paddedNumber.length;
                while (len < size) {
                    paddedNumber = "0" + paddedNumber;
                    len++;
                }
                return paddedNumber;

            };
```

```
997      }
998  ])
999  ;
```

File: tulostettavat/static/scripts/sidebarMenuCtrl.js

```
1  /**
2   * FILL WITH SUITABLE TEXT
3   * @module sidebarMenuCtrl
4   * @author Matias Berg
5   * @author Bek Eljurkaev
6   * @author Minna Lehtomäki
7   * @author Juhani Sihvonen
8   * @author Hannu Viinikainen
9   * @licence MIT
10   * @copyright 2015 Timppa project authors
11   */
12
13  timApp.controller("SidebarMenuCtrl", ['$scope', "$http", "$window",
14
15      function ($scope, $http, $window) {
16          $scope.currentLecturesList = [];
17          $scope.futureLecturesList = [];
18          $scope.pastLecturesList = [];
19          $scope.lectureQuestions = [];
20          $scope.materialQuestions = [];
21          $scope.indexSidebarState = 'autohidden';
22          $scope.lecturesSidebarState = 'autohidden';
23          $scope.questionSidebarState = 'autohidden';
24          $scope.peopleSidebarState = 'autohidden';
25          $scope.settingsSidebarState = 'autohidden';
26    $scope.indexIconState = 'noClick';
27    $scope.lectureIconState = 'noClick';
28    $scope.questionIconState = 'noClick';
29    $scope.peopleIconState = 'noClick';
30    $scope.settingsIconState = 'noClick';
31
32          /**
33           * FILL WITH SUITABLE TEXT
34           * @memberof module:sidebarMenuCtrl
35           */
36          $scope.showSidebar = function () {
37              $('.menu').slideToggle();
38              $('#futureList').hide();
39              $('#currentList').hide();
40              $scope.indexSidebarState = 'hidden';
41              $scope.lecturesSidebarState = 'hidden';
42              $scope.questionSidebarState = 'hidden';
43              $scope.peopleSidebarState = 'hidden';
44              $scope.settingsSidebarState = 'hidden';
45          };
46
47          var w = angular.element($window);
48
49          /**
50           * FILL WITH SUITABLE TEXT
51           * @memberof module:sidebarMenuCtrl
52           */
```

```
53          w.bind('resize', function () {
54              $scope.indexSidebarState = 'hidden';
55              $scope.lecturesSidebarState = 'hidden';
56              $scope.questionSidebarState = 'hidden';
57              $scope.peopleSidebarState = 'hidden';
58              $scope.settingsSidebarState = 'hidden';
59      $scope.indexIconState = 'noClick';
60      $scope.lectureIconState = 'noClick';
61      $scope.questionIconState = 'noClick';
62      $scope.peopleIconState = 'noClick';
63      $scope.settingsIconState = 'noClick';
64          });
65
66          /**
67           * FILL WITH SUITABLE TEXT
68           * @memberof module:sidebarMenuCtrl
69           */
70          $scope.toggleIndex = function () {
71              var visible = angular.element('.index-
sidebar').is(":visible");
72              if (visible) {
73                  $scope.indexSidebarState = 'hidden';
74       $scope.indexIconState = 'noClick';
75              } else {
76                  $scope.indexSidebarState = 'open';
77                  $scope.lecturesSidebarState = 'hidden';
78                  $scope.questionSidebarState = 'hidden';
79                  $scope.peopleSidebarState = 'hidden';
80                  $scope.settingsSidebarState = 'hidden';
81       $scope.indexIconState = 'clicked';
82       $scope.lectureIconState = 'noClick';
83       $scope.questionIconState = 'noClick';
84       $scope.peopleIconState = 'noClick';
85       $scope.settingsIconState = 'noClick';
86              }
87          };
88
89          /**
90           * FILL WITH SUITABLE TEXT
91           * @memberof module:sidebarMenuCtrl
92           */
93          $scope.toggleLectures = function () {
94              var visible = angular.element('.lectures-
sidebar').is(":visible");
95              if (visible) {
96                  $scope.lecturesSidebarState = 'hidden';
97       $scope.lectureIconState = 'noClick';
98
99              } else {
100                 $scope.indexSidebarState = 'hidden';
101                 $scope.lecturesSidebarState = 'open';
102                 $scope.questionSidebarState = 'hidden';
103                 $scope.peopleSidebarState = 'hidden';
104                 $scope.settingsSidebarState = 'hidden';
105      $scope.indexIconState = 'noClick';
106      $scope.lectureIconState = 'clicked';
107      $scope.questionIconState = 'noClick';
108      $scope.peopleIconState = 'noClick';
109      $scope.settingsIconState = 'noClick';
110
111                 $http({
```

```
                    url: '/getAllLecturesFromDocument',
                    method: 'GET',
                    params: {'doc_id': $scope.docId}
                })
                    .success(function (lectures) {
                        $scope.currentLecturesList =
lectures.currentLectures;
                        $scope.futureLecturesList =
lectures.futureLectures;
                        $scope.pastLecturesList =
lectures.pastLectures;
                    })
                    .error(function () {
                        console.log("Couldn't fetch the lectures");
                    })


            }
        };

        /**
         * FILL WITH SUITABLE TEXT
         * @memberof module:sidebarMenuCtrl
         */
        $scope.toggleQuestions = function () {
            var visible = angular.element('.questions-
sidebar').is(":visible");

            $scope.lectureQuestions = [];
            if (visible) {
                $scope.questionSidebarState = 'hidden';
    $scope.questionIconState = 'noClick';

            } else {
                $scope.indexSidebarState = 'hidden';
                $scope.lecturesSidebarState = 'hidden';
                $scope.questionSidebarState = 'open';
                $scope.peopleSidebarState = 'hidden';
                $scope.settingsSidebarState = 'hidden';
    $scope.indexIconState = 'noClick';
    $scope.lectureIconState = 'noClick';
    $scope.questionIconState = 'clicked';
    $scope.peopleIconState = 'noClick';
    $scope.settingsIconState = 'noClick';

                $http({
                    url: '/questions/' + $scope.docId,
                    method: 'GET'
                })
                    .success(function (questions) {
                        for (var i = 0; i < questions.length; i++)
{
                            var question = {
                                "questionId":
questions[i].question_id,
                                "questionTitle":
(JSON.parse(questions[i].questionJson)).TITLE
                            };
                            $scope.lectureQuestions.push(question);
                        }
                    })
```

```
                    .error(function () {
                        console.log("Couldn't fetch the
questions");
                    })
            }
        };

        /**
         * FILL WITH SUITABLE TEXT
         * @memberof module:sidebarMenuCtrl
         */
        $scope.togglePeople = function () {
            var visible = angular.element('.people-
sidebar').is(":visible");
            if (visible) {
                $scope.peopleSidebarState = 'hidden';
    $scope.peopleIconState = 'noClick';

            } else {
                $scope.indexSidebarState = 'hidden';
                $scope.lecturesSidebarState = 'hidden';
                $scope.questionSidebarState = 'hidden';
                $scope.peopleSidebarState = 'open';
                $scope.settingsSidebarState = 'hidden';
    $scope.indexIconState = 'noClick';
    $scope.lectureIconState = 'noClick';
    $scope.questionIconState = 'noClick';
    $scope.peopleIconState = 'clicked';
    $scope.settingsIconState = 'noClick';
            }
        };

        /**
         * FILL WITH SUITABLE TEXT
         * @memberof module:sidebarMenuCtrl
         */
        $scope.toggleSettings = function () {
            var visible = angular.element('.settings-
sidebar').is(":visible");
            if (visible) {
                $scope.settingsSidebarState = 'hidden';
    $scope.settingsIconState = 'noClick';
            } else {
                $scope.indexSidebarState = 'hidden';
                $scope.lecturesSidebarState = 'hidden';
                $scope.questionSidebarState = 'hidden';
                $scope.peopleSidebarState = 'hidden';
                $scope.settingsSidebarState = 'open';
    $scope.indexIconState = 'noClick';
    $scope.lectureIconState = 'noClick';
    $scope.questionIconState = 'noClick';
    $scope.peopleIconState = 'noClick';
    $scope.settingsIconState = 'clicked';
            }
        };

        /**
         * FILL WITH SUITABLE TEXT
         * @memberof module:sidebarMenuCtrl
         */
        $scope.autoHideSidebar = function () {
```

```
            if ($scope.indexSidebarState === 'open') {
                $scope.indexSidebarState = 'autohidden';
            }
            if ($scope.lecturesSidebarState === 'open') {
                $scope.lecturesSidebarState = 'autohidden';
            }
            if ($scope.questionSidebarState === 'open') {
                $scope.questionSidebarState = 'autohidden';
            }
            if ($scope.peopleSidebarState === 'open') {
                $scope.peopleSidebarState = 'autohidden';
            }
            if ($scope.settingsSidebarState === 'open') {
                $scope.settingsSidebarState = 'autohidden';
            }
        };
    }
])
;
```

File: tulostettavat/static/scripts/smallMenuCtrl.js

```
/**
 * FILL WITH SUITABLE TEXT
 * @module smallMenuCtrl
 * @author Matias Berg
 * @author Bek Eljurkaev
 * @author Minna Lehtomäki
 * @author Juhani Sihvonen
 * @author Hannu Viinikainen
 * @licence MIT
  * @copyright 2015 Timppa project authors
  */

timApp.controller("SmallMenuCtrl", ['$scope', '$window', '$http',
    function ($scope, $window, $http) {
  $scope.currentLecturesList = [];
        $scope.futureLecturesList = [];

        /**
         * FILL WITH SUITABLE TEXT
         * @memberof module:smallMenuCtrl
         */
        var ready = function() {
   $('#currentList').hide();
   $('#futureList').hide();
  };

        /**
         * FILL WITH SUITABLE TEXT
         * @memberof module:smallMenuCtrl
         */
  $scope.openCurrentLectureMenu = function() {
   $('#currentList').slideToggle();
   $('#futureList').hide();
   $('.menu').hide();

    $http({
```

```
                    url: '/getAllLecturesFromDocument',
                    method: 'GET',
                    params: {'doc_id': $scope.docId, 'buster':
Date.now()}
                })
                    .success(function (lectures) {
                        $scope.currentLecturesList =
lectures.currentLectures;

                    })
                    .error(function () {
                        console.log("Couldn't fetch the lectures");
                    })

    };

          /**
           * FILL WITH SUITABLE TEXT
           * @memberof module:smallMenuCtrl
           */
    $scope.openFutureLectureMenu = function() {
     $('#futureList').slideToggle();
     $('#currentList').hide();
     $('.menu').hide();

      $http({
                    url: '/getAllLecturesFromDocument',
                    method: 'GET',
                    params: {'doc_id': $scope.docId}
                })
                    .success(function (lectures) {
                        $scope.futureLecturesList =
lectures.futureLectures;
                    })
                    .error(function () {
                        console.log("Couldn't fetch the lectures");
                    })

    };

          /**
           * FILL WITH SUITABLE TEXT
           * @memberof module:smallMenuCtrl
           */
    $scope.selectCurrentLecture = function() {

    };

    var w = angular.element($window);
    w.bind('resize', function () {
     $('#currentList').hide();
     $('#futureList').hide();
    });
      }
  ]);
```

File: tulostettavat/static/scripts/view_html.js

```
1  var katex, $, angular, modules, version, refererPath, docId, docName,
rights, startIndex, users, teacherMode, lectureMode;
2
3  /*global $:false */
4
5  var timApp = angular.module('timApp');
6  timApp.controller("ViewCtrl", [
7      '$scope',
8      '$http',
9      '$q',
10      '$upload',
11      '$injector',
12      '$compile',
13      '$window',
14      '$document',
15      '$rootScope',
16      function (sc, http, q, $upload, $injector, $compile, $window,
$document, $rootScope) {
17          "use strict";
18          http.defaults.headers.common.Version = version.hash;
19          http.defaults.headers.common.RefererPath = refererPath;
20          sc.docId = docId;
21          sc.docName = docName;
22          sc.rights = rights;
23          sc.startIndex = startIndex;
24          sc.users = users;
25          sc.teacherMode = teacherMode;
26          sc.lectureMode = lectureMode;
27          sc.selectedUser = sc.users[0];
28          sc.noteClassAttributes = ["difficult", "unclear",
"editable", "private"];
29          sc.editing = false;
30          sc.questionShown = false;
31          sc.firstTimeQuestions = true;
32          var DEFAULT_BUTTON_CLASS = "timButton defaultButton";
33          var NOTE_ADD_BUTTON_CLASS = "timButton addNote";
34          var NOTE_ADD_BUTTON = "." + NOTE_ADD_BUTTON_CLASS.replace("
", ".");
35          var EDITOR_CLASS = "editorArea";
36          var EDITOR_CLASS_DOT = "." + EDITOR_CLASS;
37          var PAR_ADD_BUTTON_CLASS = "timButton addPar";
38          var PAR_ADD_BUTTON = "." + PAR_ADD_BUTTON_CLASS.replace(" ",
".");
39          var PAR_EDIT_BUTTON_CLASS = "timButton editPar";
40          var PAR_EDIT_BUTTON = "." + PAR_EDIT_BUTTON_CLASS.replace("
", ".");
41          var QUESTION_ADD_BUTTON_CLASS = "timButton addQuestion";
42          var QUESTION_ADD_BUTTON = "." +
QUESTION_ADD_BUTTON_CLASS.replace(" ", ".");
43          var PAR_CLOSE_BUTTON_CLASS = "timButton menuClose";
44          var PAR_CLOSE_BUTTON = "." +
PAR_CLOSE_BUTTON_CLASS.replace(" ", ".");
45
46          sc.processAllMath = function ($elem) {
47              $elem.find('.math').each(function () {
48                  sc.processMath(this);
49              });
50          };
```

```
51
52          sc.processMath = function (elem) {
53              var $this = $(elem);
54              var math = $this.text();
55              var hasDisplayMode = false;
56              if (math[1] === '[') {
57                  hasDisplayMode = true;
58              }
59              else if (math[1] !== '(') {
60                  return;
61              }
62              katex.render(math.slice(2, -2), elem, {displayMode:
hasDisplayMode});
63          };
64
65
66          sc.changeUser = function (user) {
67              sc.$broadcast('userChanged', {user: user});
68          };
69
70          sc.getParIndex = function ($par) {
71              return $par.index() + sc.startIndex;
72          };
73
74          sc.getElementByParIndex = function (index) {
75              return $("#pars").children().eq(index - sc.startIndex);
76          };
77
78          sc.toggleParEditor = function ($par, options) {
79              var url;
80              if ($par.hasClass("new")) {
81                  url = '/newParagraph/';
82              } else {
83                  url = '/postParagraph/';
84              }
85              var par_id = sc.getParIndex($par);
86              var attrs = {
87                  "save-url": url,
88                  "extra-data": JSON.stringify({
89                      docId: sc.docId,
90                      par: par_id
91                  }),
92                  "options": JSON.stringify({
93                      showDelete: options.showDelete,
94                      showImageUpload: true,
95                      destroyAfterSave: true
96                  }),
97                  "after-save": 'addSavedParToDom(saveData,
extraData)',
98                  "after-cancel": 'handleCancel(extraData)',
99                  "after-delete": 'handleDelete(saveData, extraData)',
100                  "preview-url": '/preview/' + sc.docId,
101                  "delete-url": '/deleteParagraph/' + sc.docId + "/"
+ par_id
102              };
103              if (options.showDelete) {
104                  attrs["initial-text-url"] = '/getBlock/' + sc.docId
+ "/" + par_id;
105              }
106              sc.toggleEditor($par, options, attrs);
107          };
```

```
108
109         sc.toggleEditor = function ($par, options, attrs) {
110             if ($par.children(EDITOR_CLASS_DOT).length) {
111                 $par.children().remove(EDITOR_CLASS_DOT);
112                 sc.editing = false;
113             } else {
114                 $(EDITOR_CLASS_DOT).remove();
115
116                 var createEditor = function (attrs) {
117                     var $div = $("<pareditor>", {class:
EDITOR_CLASS}).attr(attrs);
118                     $compile($div[0])(sc);
119                     $par.append($div);
120                     sc.editing = true;
121                 };
122
123                 if (options.showDelete) {
124                     $(".par.new").remove();
125                 }
126                 createEditor(attrs);
127             }
128         };
129
130         sc.showQuestionById = function (questionId) {
131             var question = $("#" + questionId);
132             sc.showQuestion(question);
133
134         };
135
136         sc.showQuestion = function (question) {
137             sc.json = "No data";
138             sc.qId = question[0].getAttribute('id');
139
140             http({
141                 url: '/getQuestionById',
142                 method: 'GET',
143                 params: {'question_id': sc.qId, 'buster': new
Date().getTime()}
144             })
145                 .success(function (data) {
146                     sc.json = JSON.parse(data.questionJson);
147
148                 })
149
150                 .error(function () {
151                     $window.console.log("There was some error
creating question to database.");
152                 });
153
154
155             sc.lectureId = -1;
156             sc.inLecture = false;
157
158             sc.$on('postLectureId', function (event, response) {
159                 sc.lectureId = response;
160             });
161
162             sc.$on('postInLecture', function (event, response) {
163                 sc.inLecture = response;
164             });
165
```

```
166             $rootScope.$broadcast('getLectureId');
167             $rootScope.$broadcast('getInLecture');
168             sc.showQuestionPreview = true;
169             sc.$digest();
170         };
171 
172         sc.toggleNoteEditor = function ($par, options) {
173             if (!sc.rights.can_comment) {
174                 return;
175             }
176             var url,
177                 data;
178             if (options.isNew) {
179                 url = '/postNote';
180                 data = {
181                     access: 'everyone',
182                     tags: {
183                         difficult: false,
184                         unclear: false
185                     }
186                 };
187             } else {
188                 url = '/editNote';
189                 data = options.noteData;
190                 if (!data.editable) {
191                     $window.alert('You cannot edit this note.');
192                     return;
193                 }
194             }
195 
196             var par_id = sc.getParIndex($par),
197                 attrs = {
198                     "save-url": url,
199                     "extra-data": JSON.stringify(angular.extend({
200                         docId: sc.docId,
201                         par: par_id
202                     }, data)),
203                     "options": JSON.stringify({
204                         showDelete: !options.isNew,
205                         showImageUpload: false,
206                         tags: [
207                             {name: 'difficult', desc: 'The text is
difficult to understand'},
208                             {name: 'unclear', desc: 'The text is
unclear'}
209                         ],
210                         choices: {
211                             desc: [{
212                                 desc: 'Show note to:',
213                                 name: 'access',
214                                 opts: [
215                                     {desc: 'Everyone', value:
'everyone'},
216                                     {desc: 'Just me', value:
'justme'}
217                                 ]
218                             }]
219                         },
220                         destroyAfterSave: true
221                     }),
222                     "after-save": 'handleNoteSave(saveData,
```

```
extraData)',
                    "after-cancel": 'handleNoteCancel(extraData)',
                    "after-delete": 'handleNoteDelete(saveData,
extraData)',
                    "preview-url": '/preview/' + sc.docId,
                    "delete-url": '/deleteNote',
                    "editor-text": data.content
                };
            sc.toggleEditor($par, options, attrs);
        };

        sc.forEachParagraph = function (func) {
            $('.paragraphs .par').each(func);
        };

        // Event handlers

        var ua = $window.navigator.userAgent,
            eventName = (ua.match(/iPad/i)) ? "touchstart" :
"click";

        sc.addEvent = function (className, func) {
            $document.on(eventName, className, func);
        };

        sc.showEditWindow = function (e, $par) {
            sc.toggleParEditor($par, {showDelete: true});
        };

        sc.addEvent(PAR_EDIT_BUTTON, function (e) {
            var $par = $(e.target).parent().parent().parent();
            $(".par.new").remove();
            sc.toggleActionButtons(e, $par, false, false, null);
            sc.showEditWindow(e, $par, null);
        });

        sc.addEvent("#defaultEdit", function (e) {
            var $par = $(e.target).parent().parent().parent();
            sc.toggleActionButtons(e, $par, false, false, null);
            sc.defaultAction = sc.showEditWindow;
        });

        sc.showAddParagraphAbove = function (e, $par) {
            var $newpar = $("<div>", {class: "par new"})
                .append($("<div>", {class: "parContent"}).html('New
paragraph'));
            $par.before($newpar);
            sc.toggleParEditor($newpar, {showDelete: false});
        };

        sc.showAddParagraphBelow = function (e, $par) {
            var $newpar = $("<div>", {class: "par new"})
                .append($("<div>", {class: "parContent"}).html('New
paragraph'));
            $par.after($newpar);
            sc.toggleParEditor($newpar, {showDelete: false});
        };

        sc.addEvent(PAR_ADD_BUTTON, function (e) {
            var $par = $(e.target).parent().parent().parent();
            $(".par.new").remove();
```

```
279             sc.toggleActionButtons(e, $par, false, false, null);
280             var $newpar = $("<div>", {class: "par new"})
281                 .append($("<div>", {class: "parContent"}).html('New
paragraph'));
282 
283             if ($(e.target).hasClass("above")) {
284                 $par.before($newpar);
285             } else if ($(e.target).hasClass("below")) {
286                 $par.after($newpar);
287             }
288 
289             sc.toggleParEditor($newpar, {showDelete: false});
290         });
291 
292         // Event handler for "Add question below"
293         // Opens pop-up window to create question.
294         sc.addEvent(QUESTION_ADD_BUTTON, function (e) {
295             var $par = $(e.target).parent().parent().parent();
296             sc.toggleQuestion();
297             sc.toggleActionButtons(e, $par, false, false, null);
298             sc.par = $par;
299             sc.$apply();
300         });
301 
302         // Shows question window
303         sc.toggleQuestion = function () {
304             sc.questionShown = !sc.questionShown;
305         };
306 
307         $.fn.slideFadeToggle = function (easing, callback) {
308             return this.animate({opacity: 'toggle', height:
'toggle'}, 'fast', easing, callback);
309         };
310 
311         sc.addEvent("#defaultPrepend", function (e) {
312             var $par = $(e.target).parent().parent().parent();
313             sc.toggleActionButtons(e, $par, false, false, null);
314             sc.defaultAction = sc.showAddParagraphAbove;
315         });
316 
317         sc.addEvent("#defaultAppend", function (e) {
318             var $par = $(e.target).parent().parent().parent();
319             sc.toggleActionButtons(e, $par, false, false, null);
320             sc.defaultAction = sc.showAddParagraphBelow;
321         });
322 
323         sc.doNothing = function (e, $par) {
324             sc.toggleActionButtons(e, $par, false, false, null);
325         };
326 
327         sc.addEvent(PAR_CLOSE_BUTTON, function (e) {
328             var $par = $(e.target).parent().parent().parent();
329             $(".par.new").remove();
330             sc.toggleActionButtons(e, $par, false, false, null);
331         });
332 
333         sc.addEvent("#defaultClose", function (e) {
334             var $par = $(e.target).parent().parent().parent();
335             sc.toggleActionButtons(e, $par, false, false, null);
336             sc.defaultAction = sc.doNothing;
337         });
```

```
338
339         sc.handleCancel = function (extraData) {
340             var $par = sc.getElementByParIndex(extraData.par);
341             if ($par.hasClass("new")) {
342                 $par.remove();
343             }
344             sc.editing = false;
345         };
346
347         sc.handleDelete = function (data, extraData) {
348             var $par = sc.getElementByParIndex(extraData.par);
349             http.defaults.headers.common.Version = data.version;
350             $par.remove();
351             sc.editing = false;
352         };
353
354         sc.addSavedParToDom = function (data, extraData) {
355             var $par = sc.getElementByParIndex(extraData.par),
356                 len = data.texts.length;
357             http.defaults.headers.common.Version = data.version;
358             for (var i = len - 1; i >= 0; i--) {
359                 var $newpar = $("<div>", {class: "par"})
360                     .append($("<div>", {class:
"parContent"}).html($compile(data.texts[i].html)(sc)));
361                 var readClass = "unread";
362                 if (i === 0 && !$par.hasClass("new")) {
363                     $par.find(".notes").appendTo($newpar);
364                     if ($par.find(".read, .modified").length > 0) {
365                         readClass = "modified";
366                     }
367                 }
368                 $par.after($newpar.append($("<div>",
369                     {class: "readline " + readClass, title: "Click
to mark this paragraph as read"})));
370                 sc.processMath($newpar[0]);
371             }
372             $par.remove();
373             sc.editing = false;
374         };
375
376         sc.addEvent(".readline", function () {
377             var par_id = sc.getParIndex($(this).parents('.par'));
378             var oldClass = $(this).attr("class");
379             $(this).attr("class", "readline read");
380             http.put('/read/' + sc.docId + '/' + par_id + '?_=' +
Date.now())
381                 .success(function (data, status, headers, config) {
382                     // No need to do anything here
383                 }).error(function () {
384                     $window.alert('Could not save the read
marking.');
385                     $(this).attr("class", oldClass);
386                 });
387         });
388
389         sc.showNoteWindow = function (e, $par) {
390             sc.toggleNoteEditor($par, {isNew: true});
391         };
392
393         sc.addEvent(NOTE_ADD_BUTTON, function (e) {
394             var $par = $(e.target).parent().parent().parent();
```

```
395             sc.toggleActionButtons(e, $par, false, false, null);
396             sc.showNoteWindow(e, $par, null);
397         });
398 
399         sc.addEvent("#defaultAdd", function (e) {
400             var $par = $(e.target).parent().parent().parent();
401             sc.toggleActionButtons(e, $par, false, false, null);
402             sc.defaultAction = sc.showNoteWindow;
403         });
404 
405         sc.handleNoteCancel = function () {
406             sc.editing = false;
407         };
408 
409         sc.handleNoteDelete = function () {
410             sc.getNotes();
411             sc.editing = false;
412         };
413 
414         sc.handleNoteSave = function () {
415             sc.getNotes();
416             sc.editing = false;
417         };
418 
419         sc.addEvent('.paragraphs .parContent', function (e) {
420             if (sc.editing) {
421                 return;
422             }
423 
424             sc.$apply();
425 
426             var $target = $(e.target);
427             var tag = $target.prop("tagName");
428 
429             // Don't show paragraph menu on these specific tags or
class
430             var ignoredTags = ['BUTTON', 'INPUT', 'TEXTAREA', 'A'];
431             if (ignoredTags.indexOf(tag) > -1 ||
$target.parents('.no-popup-menu').length > 0) {
432                 return;
433             }
434 
435             var $par = $(this).parent();
436             var coords = {left: e.pageX - $par.offset().left, top:
e.pageY - $par.offset().top};
437             var toggle1 = $par.find(".actionButtons").length === 0;
438             var toggle2 = $par.hasClass("lightselect");
439 
440             $(".par.selected").removeClass("selected");
441             $(".par.lightselect").removeClass("lightselect");
442             $(".actionButtons").remove();
443             sc.toggleActionButtons(e, $par, toggle1, toggle2,
coords);
444         });
445 
446         sc.addEvent(".noteContent", function () {
447             sc.toggleNoteEditor($(this).parent().parent().parent(),
{isNew: false, noteData: $(this).parent().data()});
448         });
449 
450         sc.addEvent(".questionAdded", function () {
```

```
451             sc.showQuestion($(this));
452             sc.par = ($(this.parentNode.parentNode));
453         });
454
455         // Note-related functions
456
457         sc.showOptionsWindow = function (e, $par, coords) {
458             var default_width = $par.outerWidth() / 16;
459             var button_width = $par.outerWidth() / 4 - 1.7 *
default_width;
460             var $actionDiv = $("<div>", {class: 'actionButtons'});
461             var $span;
462             if (sc.rights.can_comment) {
463                 $span = $("<span>");
464                 $span.append($("<button>", {class:
NOTE_ADD_BUTTON_CLASS, text: 'Comment/note', width: button_width}));
465                 $span.append($("<button>", {
466                     id: 'defaultAdd',
467                     class: DEFAULT_BUTTON_CLASS,
468                     text: 'Default',
469                     width: default_width
470                 }));
471                 $actionDiv.append($span);
472             }
473             if (sc.rights.editable) {
474                 $span = $("<span>");
475                 $span.append($("<button>", {class:
PAR_EDIT_BUTTON_CLASS, text: 'Edit', width: button_width}));
476                 $span.append($("<button>", {
477                     id: 'defaultEdit',
478                     class: DEFAULT_BUTTON_CLASS,
479                     text: 'Default',
480                     width: default_width
481                 }));
482                 $actionDiv.append($span);
483
484                 $span = $("<span>");
485                 $span.append($("<button>", {
486                     class: PAR_ADD_BUTTON_CLASS + ' above',
487                     text: 'Add paragraph above',
488                     width: button_width
489                 }));
490                 $span.append($("<button>", {
491                     id: 'defaultPrepend',
492                     class: DEFAULT_BUTTON_CLASS,
493                     text: 'Default',
494                     width: default_width
495                 }));
496                 $actionDiv.append($span);
497
498                 $span = $("<span>");
499                 $span.append($("<button>", {
500                     class: PAR_ADD_BUTTON_CLASS + ' below',
501                     text: 'Add paragraph below',
502                     width: button_width
503                 }));
504                 $span.append($("<button>", {
505                     id: 'defaultAppend',
506                     class: DEFAULT_BUTTON_CLASS,
507                     text: 'Default',
508                     width: default_width
```

```
509                 }));
510                 $actionDiv.append($span);
511 
512                 if (sc.lectureMode) {
513                     $span = $("<span>");
514                     $span.append($("<button>", {
515                         class: QUESTION_ADD_BUTTON_CLASS,
516                         text: 'Create question',
517                         width: button_width
518                     }));
519                     $span.append($("<button>", {
520                         id: 'createQuestion',
521                         class: DEFAULT_BUTTON_CLASS,
522                         text: 'Default',
523                         width: default_width
524                     }));
525                     $actionDiv.append($span);
526                 }
527 
528                 $span = $("<span>");
529                 $span.append($("<button>", {class:
PAR_CLOSE_BUTTON_CLASS, text: 'Close menu', width: button_width}));
530                 $span.append($("<button>", {
531                     id: 'defaultClose',
532                     class: DEFAULT_BUTTON_CLASS,
533                     text: 'Default',
534                     width: default_width
535                 }));
536                 $actionDiv.append($span);
537 
538             }
539             $actionDiv.offset(coords);
540             $actionDiv.css('position', 'absolute'); // IE needs
this
541             $par.prepend($actionDiv);
542         };
543 
544         sc.toggleActionButtons = function (e, $par, toggle1,
toggle2, coords) {
545             if (!sc.rights.editable && !sc.rights.can_comment) {
546                 return;
547             }
548             if (toggle2) {
549                 // Clicked twice successively
550                 var clicktime = new Date().getTime() -
sc.lastclick;
551                 //console.log(clicktime);
552                 $par.addClass("selected");
553 
554                 if (clicktime < 500) {
555                     // Double click
556                     sc.defaultAction(e, $par, coords);
557                 }
558                 else {
559                     // Two clicks
560                     sc.showOptionsWindow(e, $par, coords);
561                 }
562             } else if (toggle1) {
563                 // Clicked once
564                 $par.addClass("lightselect");
565                 sc.lastclick = new Date().getTime();
```

```
            } else {
                $window.console.log("This line is new: " + $par);
                $par.children().remove(".actionButtons");
                $par.removeClass("selected");
                $par.removeClass("lightselect");
            }
        };

        sc.getNoteHtml = function (notes) {
            var $noteDiv = $("<div>", {class: 'notes'});
            for (var i = 0; i < notes.length; i++) {
                var classes = ["note"];
                for (var j = 0; j < sc.noteClassAttributes.length;
j++) {
                    if (notes[i][sc.noteClassAttributes[j]] ||
notes[i].tags[sc.noteClassAttributes[j]]) {
                        classes.push(sc.noteClassAttributes[j]);
                    }
                }
                $noteDiv.append($("<div>", {class: classes.join("
")})
                    .data(notes[i])
                    .append($("<div>", {class: 'noteContent', html:
notes[i].htmlContent})));
            }
            return $noteDiv;
        };

        sc.getQuestionHtml = function (questions) {
            var questionImage = '../../../static/images/show-
question-icon.png';
            var $questionsDiv = $("<div>", {class: 'questions'});

            // TODO: Think better way to get the ID of question.
            for (var i = 0; i < questions.length; i++) {
                var img = new Image(30, 30);
                img.src = questionImage;
                img.title = questions[i].question_title;
                var $questionDiv = $("<span>", {
                    class: 'questionAdded', html: img, id:
questions[i].question_id
                });
                $questionsDiv.append($questionDiv);
            }
            return $questionsDiv;
        };


        sc.getQuestions = function () {
            var rn = "?_=" + Date.now();

            http.get('/questions/' + sc.docId + rn)
                .success(function (data) {
                    var pars = {};
                    var questionCount = data.length;
                    for (var i = 0; i < questionCount; i++) {
                        var pi = data[i].par_index;
                        if (!(pi in pars)) {
                            pars[pi] = {questions: []};
                        }

```

```
                            pars[pi].questions.push(data[i]);
                        }

                        sc.forEachParagraph(function (index) {
                                if
($(this).children().hasClass("questions")) {
                                    var children = $(this).children();
                                    for (var i = 0; i <
children.length; i++) {
                                        if (children[i].className ==
"questions") {
                                            children[i].remove();
                                        }
                                    }
                                }

                                var parIndex = index + sc.startIndex;
                                if (parIndex in pars) {
                                    var $questionsDiv =
sc.getQuestionHtml(pars[parIndex].questions);
                                    $(this).append($questionsDiv);

                                }
                            }
                        );

                    });
            };


            sc.getNotes = function () {
                var rn = "?_=" + Date.now();

                http.get('/notes/' + sc.docId + rn).success(function
(data) {
                    $('.notes').remove();
                    var pars = {};

                    var noteCount = data.length;
                    for (var i = 0; i < noteCount; i++) {
                        var pi = data[i].par_index;
                        if (!(pi in pars)) {
                            pars[pi] = {notes: []};

                        }
                        if (!('notes' in pars[pi])) {
                            pars[pi].notes = [];
                        }
                        pars[pi].notes.push(data[i]);
                    }
                    sc.forEachParagraph(function (index) {
                        var parIndex = index + sc.startIndex;
                        if (parIndex in pars) {
                            var $notediv =
sc.getNoteHtml(pars[parIndex].notes);
                            var $this = $(this);
                            $this.append($notediv);
                            sc.processAllMath($this);
                        }
                    });

```

```
676             }).error(function () {
677                 $window.alert("Could not fetch notes.");
678             });
679         };
680 
681         sc.getReadPars = function () {
682             if (!sc.rights.can_mark_as_read) {
683                 return;
684             }
685             var rn = "?_=" + Date.now();
686             http.get('/read/' + sc.docId + rn).success(function
(data) {
687                 var readCount = data.length;
688                 $('.readline').remove();
689                 var pars = {};
690                 for (var i = 0; i < readCount; i++) {
691                     var readPar = data[i];
692                     var pi = data[i].par_index;
693                     if (!(pi in pars)) {
694                         pars[pi] = {};
695                     }
696                     pars[pi].readStatus = readPar.status;
697                 }
698                 sc.forEachParagraph(function (index) {
699                     var parIndex = index + sc.startIndex;
700                     var classes = ["readline"];
701                     if (parIndex in pars && 'readStatus' in
pars[parIndex]) {
702                         classes.push(pars[parIndex].readStatus);
703                     } else {
704                         classes.push("unread");
705                     }
706                     var $div = $("<div>", {
707                         class: classes.join(" "),
708                         title: "Click to mark this paragraph as
read"
709                     });
710                     $(this).append($div);
711                 });
712             }).error(function () {
713                 $window.alert("Could not fetch reading info.");
714             });
715         };
716 
717         sc.setHeaderLinks = function () {
718             $(".par h1, .par h2, .par h3, .par h4, .par h5, .par
h6").each(function () {
719                 var $par = $(this).parent();
720                 $par.append($("<a>", {
721                     text: '#',
722                     href: '#' + $(this).attr('id'),
723                     class: 'headerlink',
724                     title: 'Permanent link'
725                 }));
726             });
727         };
728 
729         // Index-related functions
730 
731         sc.totext = function (str) {
732             if (str.indexOf('{') > 0) {
```

```
                return str.substring(0, str.indexOf('{')).trim();
            }
            return str;
        };

        sc.tolink = function (str) {
            if (str.indexOf('{') >= 0 && str.indexOf('}') > 0) {
                var ob = str.indexOf('{');
                var cb = str.indexOf('}');
                return str.substring(ob + 1, cb);
            }
            return "#" + str.replace(/^(\d)+(\.\d+)*\.? /,
"").replace(/[^\d\wÃ¥Ã¤Ã¶\.\- ]/gi, "").trim().replace(/ +/g, '-
').toLowerCase();
        };

        sc.findIndexLevel = function (str) {
            for (var i = 0; i < str.length; i++) {
                if (str.charAt(i) !== '#') {
                    return i;
                }
            }

            return 0;
        };

        sc.getIndex = function () {
            http.get('/index/' + sc.docId).success(function (data)
{
                var parentEntry = null;
                sc.indexTable = [];

                for (var i = 0; i < data.length; i++) {
                    var lvl = sc.findIndexLevel(data[i]);
                    if (lvl < 1 || lvl > 3) {
                        continue;
                    }

                    var astyle = "a" + lvl;
                    var txt = data[i].substr(lvl);
                    txt = txt.trim().replace(/\\#/g, "#");
                    var entry = {
                        text: sc.totext(txt),
                        target: sc.tolink(txt),
                        style: astyle,
                        level: lvl,
                        items: [],
                        state: ""
                    };

                    if (lvl === 1) {
                        if (parentEntry !== null) {
                            if ("items" in parentEntry &&
parentEntry.items.length > 0) {
                                parentEntry.state = 'col';
                            }
                            sc.indexTable.push(parentEntry);
                        }

                        parentEntry = entry;
                    }
```

```
                    else if (parentEntry !== null) {
                        if (!("items" in parentEntry)) {
                            // For IE
                            parentEntry.items = [];
                        }
                        parentEntry.items.push(entry);
                    }

                }

                if (parentEntry !== null) {
                    if (parentEntry.items.length > 0) {
                        parentEntry.state = 'col';
                    }
                    sc.indexTable.push(parentEntry);
                }
            }).error(function () {
                $window.alert("Could not fetch index entries.");
            });
        };

        sc.invertState = function (state) {
            if (state === 'exp') {
                return 'col';
            }
            if (state === 'col') {
                return 'exp';
            }
            return state;
        };

        sc.clearSelection = function () {
            if ($document.selection) {
                $document.selection.empty();
            }
            else if ($window.getSelection) {
                $window.getSelection().removeAllRanges();
            }
        };

        sc.invertStateClearSelection = function (event, state) {
            if (event.which !== 1) {
                // Listen only to the left mouse button
                return state;
            }
            if (event.target.className === 'a2' ||
event.target.className === 'a3') {
                // Do not collapse/expand if a subentry is clicked
                return state;
            }

            var newState = sc.invertState(state);
            if (newState !== state) {
                sc.clearSelection();
            }
            return newState;
        };

        if (sc.lectureMode) {
            sc.$on("getQuestions", function () {
                if (sc.firstTimeQuestions) {
```

```
                    sc.getQuestions();
                    sc.firstTimeQuestions = false;
                }
            });

            sc.$on("closeQuestionPreview", function () {
                sc.showQuestionPreview = false;
                //sc.clearQuestion();
            });
        }

        // Load index, notes and read markings
        sc.setHeaderLinks();
        sc.indexTable = [];
        sc.getIndex();
        sc.getNotes();
        sc.getReadPars();
        sc.processAllMath($('body'));
        sc.defaultAction = sc.showOptionsWindow;
    }
])
;

/**
 * Controller for creating and editing questions
 * @module questionController
 * @author Matias Berg
 * @author Bek Eljurkaev
 * @author Minna Lehtomäki
 * @author Juhani Sihvonen
 * @author Hannu Viinikainen
 * @licence MIT
 * @copyright 2015 Timppa project authors
 */

timApp.controller("QuestionController", ['$scope', '$http',
'$window', '$rootScope', function (scope, http, $window, $rootScope) {
    "use strict";
    $(function () {
        $('#calendarStart').datepicker({dateFormat: 'dd.m.yy'});
    });

    scope.putBackQuotations = function(x){
        return x.replace(/&quot;/g, '"');
    };

    scope.$on("editQuestion", function (event, data) {
            var id = data.question_id;
            var json = data.json;

            if (id) scope.question.question_id = id;
            if (json["TITLE"]) scope.question.title =
scope.putBackQuotations(json["TITLE"]);
            if (json["QUESTION"]) scope.question.question =
scope.putBackQuotations(json["QUESTION"]);
            if (json["TYPE"]) scope.question.type = json["TYPE"];
            if (json["MATRIXTYPE"]) scope.question.matrixType =
json["MATRIXTYPE"];
            if (json["ANSWERFIELDTYPE"])
scope.question.answerFieldType = (json["ANSWERFIELDTYPE"]);

```

```
906
907             var jsonData = json["DATA"];
908             var jsonHeaders = jsonData["HEADERS"];
909             var jsonRows = jsonData["ROWS"];
910
911             var columnHeaders = [];
912             for (var i = 0; i < jsonHeaders.length; i++) {
913                 columnHeaders[i] = {
914                     id: i,
915                     type: jsonHeaders[i].type,
916                     text:
scope.putBackQuotations(jsonHeaders[i].text)
917                 };
918             }
919             scope.columnHeaders = columnHeaders;
920
921             var rows = [];
922             for (var i = 0; i < jsonRows.length; i++) {
923                 rows[i] = {
924                     id: jsonRows[i].id,
925                     text:
scope.putBackQuotations(jsonRows[i].text),
926                     type: jsonRows[i].type,
927                     value: jsonRows[i].value
928                 };
929
930
931                 var jsonColumns = jsonRows[i]["COLUMNS"];
932                 var columns = [];
933                 for (var j = 0; j < jsonColumns.length; j++) {
934                     columns[j] = {
935                         id: j,
936                         rowId: i,
937                         text: jsonColumns[j].text,
938                         //points: jsonColumns[j].points,
939                         type: jsonColumns[j].type,
940                         answerFiledType:
jsonColumns[j].answerFieldType
941                     };
942                     if (jsonColumns[j].points) {
943                         columns[j].points = jsonColumns[j].points;
944                     } else {
945                         columns[j].points = "";
946                     }
947
948                 }
949
950                 rows[i].columns = columns;
951
952             }
953             scope.rows = rows;
954
955             if (json["TIMELIMIT"] && json["TIMELIMIT"] > 0) {
956                 var time = json["TIMELIMIT"];
957                 scope.question.endTimeSelected = true;
958                 if (time > 3600) {
959                     scope.question.timeLimit.hours =
Math.floor(time / 3600);
960                     time = time % 3600;
961                 } else {
962                     scope.question.timeLimit.hours = 0;
```

```
963                 }
964
965                 if (time > 60) {
966                     scope.question.timeLimit.minutes =
Math.floor(time / 60);
967                     time = time % 60;
968                 } else {
969                     scope.question.timeLimit.minutes = 0;
970                 }
971
972                 if (time > 0) {
973                     scope.question.timeLimit.seconds = time;
974                 } else {
975                     scope.question.timeLimit.seconds = 0;
976                 }
977
978             } else {
979                 scope.question.endTimeSelected = false;
980             }
981
982             scope.toggleQuestion();
983
984         }
985     );
986
987
988     scope.question = {
989         title: "",
990         question: "",
991         matrixType: "",
992         answerFieldType: "",
993         timeLimit: {hours: "0", minutes: "0", seconds: "30"},
994         endTimeSelected: true
995
996     };
997
998
999     scope.rows = [];
1000      scope.columns = [];
1001      scope.columnHeaders = [];
1002      scope.answerDirection = "horizontal";
1003      scope.question.timeLimit.seconds = 30;
1004      scope.error_message = "";
1005      scope.answerFieldTypes = [
1006
1007          {label: "Text area", value: "textArea"},
1008          {label: "Radio Button horizontal", value: "radiobutton-
horizontal"},
1009          {label: "Checkbox", value: "checkbox"}
1010      ];
1011
1012      /**
1013       * A function for creating a matrix.
1014       * @memberof module:questionController
1015       * @param rowsCount The number of rows to create for the
matrix.
1016       * @param columnsCount The number of columns to create for new
matrix.
1017       * @param type The answer type of the matrix.
1018       */
1019      scope.createMatrix = function (type) {
```

```
1020          var rowsCount = 0;
1021          var columnsCount = 0;
1022          if (type === 'matrix' || type === 'true-false') {
1023              rowsCount = 2;
1024              columnsCount = 2;
1025          } else {
1026              rowsCount = 2;
1027              columnsCount = 1;
1028          }
1029
1030          if (scope.rows.length < 1) {
1031              for (var i = 0; i < rowsCount; i++) {
1032                  scope.addRow(i);
1033              }
1034          }
1035
1036
1037          if (type === 'radio-vertical' || 'true-false')
scope.question.answerFieldType = 'radio';
1038          if (type === 'checkbox-vertical')
scope.question.answerFieldType = 'checkbox';
1039          if (type === 'matrix') {
1040              scope.question.answerFieldType = 'matrix';
1041          }
1042
1043          for (var i = 0; i < scope.rows.length; i++) {
1044              if (scope.rows[i].columns.length > columnsCount)
scope.rows[i].columns.splice(columnsCount,
scope.rows[i].columns.length);
1045              if (scope.rows[i].columns.length < columnsCount)
scope.addCol(scope.rows[0].columns.length);
1046              for (var j = 0; j < scope.rows[i].columns.length; j++)
{
1047                  scope.rows[j].columns.answerFieldType =
scope.question.answerFieldType;
1048              }
1049          }
1050
1051          scope.columnHeaders = [];
1052          if (type === 'matrix') {
1053              for (var i = 0; i < scope.rows[0].columns.length; i++)
{
1054                  scope.columnHeaders[i] = {
1055                      id: i,
1056                      text: "",
1057                      type: 'header'
1058                  };
1059              }
1060          }
1061
1062
1063          /*        if (scope.question.type != type ||
scope.rows.length <= 0) {
1064           scope.question.type = type;
1065
1066           if (type === 'radio-vertical' || 'true-false')
scope.question.answerFieldType = 'radio';
1067           else if (type === 'checkbox-vertical')
scope.question.answerFieldType = 'checkbox';
1068           else if (type === 'matrix')
scope.question.answerFieldType = 'matrix';
```

```
1069
1070
1071
1072            var i;
1073            if (scope.rows.length > 0) {
1074            scope.columnHeaders.splice(0,
scope.columnHeaders.length);
1075            for (i = 0; i < scope.rows.length; i++) {
1076            scope.rows[i].columns.splice(0,
scope.rows[i].columns.length);
1077            }
1078
1079            for (i = 0; i < columnsCount; i++) {
1080            scope.addCol(i);
1081            }
1082
1083            } else {
1084
1085            var columnHeaders = [];
1086            for (i = 0; i < rowsCount; i++) {
1087            var columns = [];
1088            columnHeaders = [];
1089            for (var j = 0; j < columnsCount; j++) {
1090            columnHeaders.push({type: "header", id: j, text: ""});
1091            columns[j] = {
1092            id: j,
1093            rowId: i,
1094            text: '',
1095            points: '',
1096            type: "answer",
1097            answerFiledType: scope.question.answerFieldType
1098            };
1099            }
1100            scope.rows[i] = {
1101            id: i,
1102            text: '',
1103            type: 'question',
1104            value: '',
1105            columns: columns
1106            };
1107
1108            }
1109            scope.columnHeaders = columnHeaders;
1110
1111            }
1112            scope.columnHeaders.splice(columnsCount,
scope.columnHeaders.length);
1113            }*/
1114      };
1115
1116      /*    */
1117      /**
1118       * A function handling rowClick
1119       * @memberof module:questionController
1120       * @param index FILL WITH SUITABLE TEXT
1121       */
1122      /*
1123       scope.rowClick = function (index) {
1124       scope.addRow(index);
1125       };*/
1126
```

```
1127      /**
1128       * A function to add a column to an existing matrix.
1129       * @memberof module:questionController
1130       * @param loc The index in the matrix where to add the new
column.
1131       */
1132      scope.addCol = function (loc) {
1133          var location = loc;
1134          if (loc === -1) {
1135              location = scope.rows[0].columns.length;
1136              loc = scope.rows[0].columns.length;
1137          }
1138          scope.columnHeaders.splice(loc, 0, {type: "header", id:
loc, text: ""});
1139          //add new column to columns
1140          for (var i = 0; i < scope.rows.length; i++) {
1141
1142              scope.rows[i].columns.splice(loc, 0, {
1143                  id: location,
1144                  rowId: i,
1145                  text: '',
1146                  points: '',
1147                  type: "answer",
1148                  answerFiledType: scope.question.answerFieldType
1149              });
1150          }
1151
1152
1153      };
1154
1155      /**
1156       * The function adds a row to an existing matrix
1157       * @memberof module:questionController
1158       * @param loc The index in the matrix where to add the new
row.
1159       */
1160      scope.addRow = function (loc) {
1161
1162          scope.CreateColumnsForRow = function (location) {
1163              var columns = [];
1164              if (scope.rows.length > 0) {
1165                  for (var j = 0; j < scope.rows[0].columns.length;
j++) {
1166                      columns[j] = {
1167                          id: j,
1168                          rowId: location,
1169                          type: "answer",
1170                          value: '',
1171                          answerFiledType:
scope.question.answerFieldType,
1172                          points: ""
1173                      };
1174
1175                  }
1176              }
1177              return columns;
1178          };
1179
1180          var location = loc;
1181          if (loc === -1) {
1182              location = scope.rows.length;
```

```
            loc = scope.rows.length;
        }

        var columns = scope.CreateColumnsForRow(location);
        scope.rows.splice(loc, 0,
            {
                id: location,
                text: "",
                type: "question",
                value: "",
                columns: columns
            });

        for (var i = 0; i < scope.rows.length; i++) {
            scope.rows[i].id = i;
        }


    };

    /**
     * A function to delete a row from a matrix.
     * @memberof module:questionController
     * @param indexToBeDeleted The index of the row to be deleted.
     */
    scope.delRow = function (indexToBeDeleted) {
        scope.error_message = "";
        if (scope.rows.length > 1) {
            if (indexToBeDeleted === -1) {
                scope.rows.splice(-1, 1);
            }
            else {
                scope.rows.splice(indexToBeDeleted, 1);
            }
        } else {
            scope.errorize("", "You cannot have an empty table.");
        }

    };

    /**
     * A function to delete a column from a matrix.
     * @memberof module:questionController
     * @param indexToBeDeleted Index of the column to be deleted.
     */
    scope.delCol = function (indexToBeDeleted) {
        for (var i = 0; i < scope.rows.length; i++) {
            if (indexToBeDeleted === -1) {
                scope.rows[i].columns.splice(-1, 1);
            }
            else {
                scope.rows[i].columns.splice(indexToBeDeleted, 1);
            }
        }
        if (indexToBeDeleted === -1) {
            scope.columnHeaders.splice(-1, 1);
        }
        else {
            scope.columnHeaders.splice(indexToBeDeleted, 1);
        }

```

```
1244      };
1245
1246      /**
1247       * A function to reset the question values.
1248       * @memberof module:questionController
1249       */
1250      scope.clearQuestion = function () {
1251          scope.question = {
1252              title: "",
1253              question: "",
1254              matrixType: "",
1255              answerFieldType: "",
1256              timeLimit: {hours: "0", minutes: "0", seconds: "30"},
1257              endTimeSelected: true
1258          };
1259
1260          scope.rows = [];
1261          scope.answer = "";
1262          scope.columnHeaders = [];
1263          //scope.toggleQuestion();
1264      };
1265
1266      /**
1267       * A function to close question edition form.
1268       * @memberof module:questionController
1269       */
1270      scope.close = function () {
1271          scope.removeErrors();
1272          scope.clearQuestion();
1273          if (scope.questionShown) scope.toggleQuestion();
1274      };
1275
1276      /**
1277       * The function replaces linebreaks with HTML code.
1278       * @memberof module:questionController
1279       * @param val The input string
1280       * @returns {*} The reformatted line.
1281       */
1282      scope.replaceLinebreaksWithHTML = function (val) {
1283          var output = val.replace(/(?:\r\n|\r|\n)/g, '<br />');
1284          output = output.replace(/"/g, '&quot;');
1285          return output.replace(/\\/g, "\\\\");
1286      };
1287
1288      /**
1289       * The function to highlight the source of the errors for a
given ID.
1290       * @memberof module:questionController
1291       * @param div_val ID of the element to be errorized.
1292       * @param error_text Description of the occured error.
1293       */
1294      scope.errorize = function (div_val, error_text) {
1295          angular.element("#" + div_val).css('border', "1px solid
red");
1296          if (error_text.length > 0) {
1297              scope.error_message += error_text + "<br />";
1298          }
1299      };
1300
1301      /**
1302       * The function to highlight the source of the errors for a
```

```
given class.
1303         * @memberof module:questionController
1304         * @param div_val Class of the element to be errorized.
1305         * @param error_text Description of the occured error.
1306         */
1307        scope.errorizeClass = function (div_val, error_text) {
1308            angular.element("." + div_val).css('border', "1px solid
red");
1309            if (error_text.length > 0) {
1310                scope.error_message += error_text + "<br />";
1311            }
1312        };
1313
1314        /**
1315         * Removes border of a given element.
1316         * @memberof module:questionController
1317         * @param element ID of the field whose border will be
removed.
1318         */
1319        scope.defInputStyle = function (element) {
1320            if (element !== null || !element.isDefined) {
1321                angular.element("#" + element).css("border", "");
1322            }
1323        };
1324
1325        /**
1326         * Calls defInputStyle for all the form elements.
1327         * @memberof module:questionController
1328         */
1329        scope.removeErrors = function () {
1330            scope.error_message = "";
1331            var elementsToRemoveErrorsFrom = [
1332                "questionName",
1333                "questionTiming",
1334                "questionStart",
1335                "questionTimer",
1336                "qType",
1337                "matrix",
1338                "durationSec",
1339                "durationHour",
1340                "durationMin",
1341                "durationDiv"
1342            ];
1343            for (var i = 0; i < elementsToRemoveErrorsFrom.length;
i++) {
1344                if (elementsToRemoveErrorsFrom[i] !== undefined) {
1345
scope.defInputStyle(elementsToRemoveErrorsFrom[i]);
1346                }
1347            }
1348            angular.element(".rowHeading").css("border", "");
1349        };
1350
1351        /**
1352         * Function for checking if the row headings are empty.
1353         * @memberof module:questionController
1354         * @param rows The array of rows to be checked.
1355         * @returns {boolean} Whether or not the row headings are
empty.
1356         */
1357        scope.rowHeadingsEmpty = function (rows) {
```

```
1358          for (var i = 0; i < rows.length; i++) {
1359              if (rows[i].text === "" || rows[i].text === null) {
1360                  return true;
1361              }
1362          }
1363          return false;
1364      };
1365
1366      /**
1367       * Checks if a value is a positive number and makes the
appropriate errors if this is not the case.
1368       * @memberof module:questionController
1369       * @param element The value to be checked.
1370       * @param val The id of the value, which is used in case the
number is not positive.
1371       */
1372      scope.isPositiveNumber = function (element, val) {
1373          if (element === "" || isNaN(element) || element < 0) {
1374              scope.errorize(val, "Number has to be positive.");
1375          }
1376      };
1377
1378      /**
1379       * Validates and saves the question into the database.
1380       * @memberof module:questionController
1381       */
1382      scope.createQuestion = function () {
1383
1384          scope.removeErrors();
1385          if (scope.question.question === undefined ||
scope.question.question.trim().length === 0 || scope.question.title ===
undefined || scope.question.title.trim().length === 0) {
1386              scope.errorize("questionName", "Both title and
question are required for a question.");
1387          }
1388          if (scope.question.type === undefined) {
1389              scope.errorize("qType", "Question type must be
selected.");
1390          } else if (scope.question.type === "matrix" &&
(scope.question.matrixType === undefined || scope.question.matrixType
=== "")) {
1391              scope.errorize("check", "Answer type must be
selected.");
1392          } else if ((scope.question.type === "radio-vertical" ||
1393              scope.question.type === "checkbox-vertical" ||
1394              scope.question.type === "true-false") &&
1395              scope.rowHeadingsEmpty(scope.rows)) {
1396              scope.errorizeClass("rowHeading", "All rows must be
filled in.");
1397          }
1398          if (scope.rows.length > 0) {
1399              if ((scope.question.type === "radio-vertical" ||
scope.question.type === "checkbox-vertical") && scope.rows.length < 2) {
1400                  scope.errorize("matrix", "You must have at least
two choices.");
1401              }
1402          } else if (scope.question.type !== undefined) {
1403              scope.errorize("matrix", "You must have at least one
row.");
1404          }
1405          var timeLimit = "";
```

```
1406          if (scope.question.endTimeSelected) {
1407              if (scope.question.timeLimit.hours === "") {
1408                  scope.question.timeLimit.hours = 0;
1409              }
1410              if (scope.question.timeLimit.minutes === "") {
1411                  scope.question.timeLimit.minutes = 0;
1412              }
1413              if (scope.question.timeLimit.seconds === "") {
1414                  scope.question.timeLimit.seconds = 0;
1415              }
1416              scope.isPositiveNumber(scope.question.timeLimit.hours,
"durationHour");
1417
scope.isPositiveNumber(scope.question.timeLimit.minutes, "durationMin");
1418
scope.isPositiveNumber(scope.question.timeLimit.seconds, "durationSec");
1419              timeLimit = 0;
1420              timeLimit = parseInt(timeLimit) +
parseInt(scope.question.timeLimit.seconds);
1421              if (scope.question.timeLimit.hours) {
1422                  timeLimit = parseInt(timeLimit) +
(scope.question.timeLimit.hours * 60 * 60);
1423              }
1424              if (scope.question.timeLimit.minutes) {
1425                  timeLimit = parseInt(timeLimit) +
(scope.question.timeLimit.minutes * 60);
1426              }
1427              if (timeLimit <= 0) {
1428                  scope.errorize("durationDiv", "Please enter a
duration greater then zero or for unending question uncheck the duration
box.");
1429              }
1430          } else {
1431              timeLimit = "";
1432          }
1433
1434          if (scope.error_message !== "") {
1435              return;
1436          }
1437
1438          if (scope.question.type === 'matrix') {
1439
1440              if (scope.question.matrixType === "radiobutton-
horizontal" || scope.question.matrixType === "radiobutton-vertical") {
1441                  scope.question.answerFieldType = "radio";
1442              }
1443
1444              if (scope.question.matrixType === "textArea") {
1445                  scope.question.answerFieldType = "text";
1446              }
1447              if (scope.question.matrixType === "checkbox") {
1448                  scope.question.answerFieldType = "checkbox";
1449              }
1450          }
1451          var doc_id = scope.docId;
1452          var $par = scope.par;
1453          var par_index = scope.getParIndex($par);
1454
1455          //TODO use  JSON.stringify
1456
1457
```

```
1458          scope.question.question =
scope.replaceLinebreaksWithHTML(scope.question.question);
1459          scope.question.title =
scope.replaceLinebreaksWithHTML(scope.question.title);
1460
1461          var questionJson = '{"QUESTION": "' +
scope.question.question + '", "TITLE": "' + scope.question.title + '",
"TYPE": "' + scope.question.type + '", "ANSWERFIELDTYPE": "' +
scope.question.answerFieldType + '", "MATRIXTYPE": "' +
scope.question.matrixType + '", "TIMELIMIT": "' + timeLimit + '",
"DATA": {';
1462
1463          var testJson = JSON.stringify(scope.question);
1464          testJson += JSON.stringify(scope.columnHeaders);
1465
1466          questionJson += '"HEADERS" : [';
1467          var i;
1468          if (scope.question.type === "matrix") {
1469              for (i = 0; i < scope.columnHeaders.length; i++) {
1470                  questionJson += '{';
1471                  questionJson += '"type":"' +
scope.columnHeaders[i].type + '",';
1472                  questionJson += '"id":"' +
scope.columnHeaders[i].id + '",';
1473                  questionJson += '"text":"' +
scope.replaceLinebreaksWithHTML(scope.columnHeaders[i].text) + '"';
1474                  questionJson += '},';
1475              }
1476              if (i > 0) {
1477                  questionJson = questionJson.substring(0,
questionJson.length - 1);
1478              }
1479              questionJson += ']';
1480              questionJson += ',';
1481          } else {
1482              questionJson += "],";
1483          }
1484
1485          questionJson += '"ROWS": [';
1486          for (i = 0; i < scope.rows.length; i++) {
1487              questionJson += '{';
1488              questionJson += '"id":"' + scope.rows[i].id + '",';
1489              questionJson += '"type":"' + scope.rows[i].type +
'",';
1490              questionJson += '"text":"' +
scope.replaceLinebreaksWithHTML(scope.rows[i].text) + '",';
1491              questionJson += '"COLUMNS": [';
1492              for (var j = 0; j < scope.rows[i].columns.length; j++)
{
1493                  questionJson += '{';
1494                  questionJson += '"id":"' +
scope.rows[i].columns[j].id + '",';
1495                  questionJson += '"rowId":"' +
scope.rows[i].columns[j].rowId + '",';
1496                  questionJson += '"type":"' +
scope.rows[i].columns[j].type + '",';
1497                  if (scope.question.answerFieldType !== "text") {
1498                      questionJson += '"points":"' +
scope.rows[i].columns[j].points + '",';
1499                  } else {
1500                      questionJson += '"points":"' + "" + '",';
```

```
1501                 }
1502                 questionJson += '"answerFieldType":"' +
scope.question.answerFieldType + '"';
1503                 questionJson += '},';
1504             }
1505 
1506             if (j > 0) {
1507                 questionJson = questionJson.substring(0,
questionJson.length - 1);
1508             }
1509             questionJson += ']';
1510             questionJson += '},';
1511 
1512         }
1513         if (i > 0) {
1514             questionJson = questionJson.substring(0,
questionJson.length - 1);
1515         }
1516         questionJson += ']';
1517         questionJson += '}}';
1518 
1519         var rn = "?_=" + Date.now();
1520 
1521         http({
1522             method: 'POST',
1523             url: '/addQuestion/' + rn,
1524             params: {
1525                 'question_id': scope.question.question_id,
1526                 'question_title': scope.question.title,
1527                 'answer': "test", //answerVal,
1528                 'par_index': par_index,
1529                 'doc_id': doc_id,
1530                 'questionJson': questionJson
1531             }
1532         })
1533             .success(function () {
1534                 $window.console.log("The question was successfully
added to database");
1535                 scope.removeErrors();
1536                 //scope.clearQuestion();
1537                 //TODO: This can be optimized to get only the new
one.
1538                 scope.$parent.getQuestions();
1539                 scope.close();
1540             })
1541             .error(function () {
1542                 $window.console.log("There was some error creating
question to database.");
1543             });
1544     };
1545 
1546     scope.deleteQuestion = function () {
1547         var confirmDi = $window.confirm("Are you sure you want to
delete this question?");
1548         if (confirmDi) {
1549             http({
1550                 url: '/deleteQuestion',
1551                 method: 'POST',
1552                 params: {question_id: scope.qId, doc_id:
scope.docId}
1553             })
```

```
                .success(function () {
                    $window.console.log("Deleted question done!");
                    //location.reload();
                    scope.close();
                    //scope.clearQuestion();
                    scope.getQuestions();
                })
                .error(function (error) {

                    $window.console.log(error);
                    scope.getQuestions();

                });

        }
    };
}]);
```