Unit Test Plan

Project Manager: Sebastiaan Candel

Quality Assurance Manager: Ronald van Zon

Senior Management:
Mark van den Brand, MF 7.0.96
Lou Somers, MF 7.145

Advisor:
Erik Scheffers, MF 7.146

Customer:
Lex Lemmens, HG -01.27

Authors:
Peter van Heck (0649886)
Peter Koymans (0748876)
Kay Lukas (0758084)
Astrid Pieterse (0743073)
Robbert Raats (0675661)
Willem Sonke (0752229)
Roby Visser (0654428)

October 26, 2012 Eindhoven
Abstract

This document is the Unit Test Plan (UTP) of the kroket group. This project is part of the Software Engineering Project (2IP35) and is one of the assignments at Eindhoven University of Technology. The document complies with the UTP from the Software Engineering Standard, as set by the European Space Agency [1].

This document provides the main guidance for the Unit Test (UT) during the Detailed Design (DD) phase for the kroket application. It describes the environment needed to perform the UT. When this environment is set up, all test cases must be executed according to their corresponding test procedures. After a test has been performed a report needs to be written.
Contents

Document Status Sheet 4

Document Change Records 5

1 Introduction 6
   1.1 Purpose ................................................................. 6
   1.2 Overview ............................................................... 6
   1.3 List of definitions .................................................. 6
   1.4 List of references ................................................... 6

2 Test plan 8
   2.1 Test items .............................................................. 8
   2.2 Features to be tested ............................................... 8
   2.3 Test deliverables .................................................... 8
   2.4 Testing tasks ........................................................ 9
   2.5 Environmental needs ............................................... 9
   2.6 Test case pass/fail criteria ........................................ 9

3 Testcase specification 10
   3.1 Client side tests .................................................... 10
      3.1.1 Event handlers .................................................. 10
      3.1.2 Authentication ................................................... 14
      3.1.3 Onload events ..................................................... 16
      3.1.4 Searching for packages .......................................... 17
      3.1.5 Search for subjects .............................................. 18
      3.1.6 Save and load schedule ......................................... 20
      3.1.7 Helper functions .................................................. 21
      3.1.8 Checking the schedule .......................................... 27
      3.1.9 English language version ...................................... 29
   3.2 Server side tests ..................................................... 30
      3.2.1 Testing adding of subject and packet information .......... 30
      3.2.2 Testing adding of subject and packet information .......... 35
      3.2.3 Testing retrieving information from OWIS .................... 37
      3.2.4 Testing schedule related tasks ................................ 37
      3.2.5 Testing recommendation related tasks ........................ 43
Kroket – UNIT TEST PLAN

3.2.6 Testing package related tasks .............................................. 45
3.2.7 Testing security related tasks .............................................. 47
3.2.8 Testing account related tasks .............................................. 48
3.2.9 Keyword generation .......................................................... 60
3.2.10 Authentication of the user ............................................... 62
3.2.11 Testing adding of subject and packet information .................. 67
3.2.12 Requesting additional information of a subject ...................... 71
3.2.13 Converting nested lists of subject codes to nested list of standard formats 74
3.2.14 Searching on subjects .................................................... 76
3.2.15 Checking whether the schedule is valid .............................. 81

4 Test procedures 88
4.1 Javascript testing ............................................................ 88
4.2 Server side testing ........................................................... 89

5 Test reports 91
5.1 Javascript tests ............................................................... 91
5.2 Python tests ................................................................. 100
Document Status Sheet

General

Document title  Unit Test
Identification  Documentatie.SVVP.UT.0.1.0
Authors  Peter Koymans, Astrid Pieterse, Robbert Raats
Document status  Final

Document history

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Author</th>
<th>Reason of change</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0.0</td>
<td>10-9-2012</td>
<td>Peter Koymans, Astrid Pieterse, Robbert Raats</td>
<td>Initial version</td>
</tr>
<tr>
<td>0.0.1</td>
<td>18-9-2012</td>
<td>Astrid Pieterse</td>
<td>First version for internal review</td>
</tr>
<tr>
<td>0.0.2</td>
<td>19-9-2012</td>
<td>Astrid Pieterse</td>
<td>Second version for internal review</td>
</tr>
<tr>
<td>0.1.0</td>
<td>26-10-2012</td>
<td>Peter Koymans</td>
<td>Final version for internal review</td>
</tr>
</tbody>
</table>
Document Change Records

General
Date 26-10-2012
Document title Unit Test
Identification Documentatie.SVVP.UT.0.1.0

Document history

<table>
<thead>
<tr>
<th>Sections</th>
<th>Reason of change</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>Internal review and improvements</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 Purpose

The UTP document describes the plan for testing the developed software units against the detailed design, defined in the DDD [2]. The unit tests make sure that kroket complies with the design in the Detailed Design (DD) phase of the kroket project as described in the ESA software engineering standard [1].

1.2 Overview

Chapter 2 gives an overview of all items be tested, and the general criteria for the UT. Chapter 3 specifies how tests are defined, chapter 4 specifies the test procedures and chapter 5 specifies how test results are reported.

1.3 List of definitions

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DD</td>
<td>Detailed Design</td>
</tr>
<tr>
<td>DDD</td>
<td>Detailed Design Document</td>
</tr>
<tr>
<td>ESA</td>
<td>European Space Agency</td>
</tr>
<tr>
<td>KROKET</td>
<td>Kies niet Roekeloos maar Objectief Keuzevakken Efficiënt en Tevreden</td>
</tr>
<tr>
<td>SCMP</td>
<td>Software Configuration Management Plan</td>
</tr>
<tr>
<td>SVVP</td>
<td>Software Validation and Verification Plan</td>
</tr>
<tr>
<td>Trac</td>
<td>Project management and bug/issue tracking system.</td>
</tr>
<tr>
<td>UT</td>
<td>Unit Test</td>
</tr>
<tr>
<td>UTP</td>
<td>Unit Test Plan</td>
</tr>
</tbody>
</table>

1.4 List of references


2. kroket group, Detailed Design Document (DDD).
Kroket – UNIT TEST PLAN

3. KROKET group, Software Configuration Management Plan (SCMP).
4. KROKET group, Software Validation and Verification Plan (SVVP).
5. KROKET group, Unit Test Plan (UTP).

7
Chapter 2

Test plan

2.1 Test items

The software to be tested is the kroket application. Information about the detailed design of kroket can be found in the DDD [2].

2.2 Features to be tested

kroket must meet the design as stated in the DDD [2]. Each component should adhere to the interfaces given in the DDD [2].

2.3 Test deliverables

Before the testing starts, the following documents must be delivered:

- SVVP [4].
- DDD [2].
- This document.
- UT input data.

After completing the testing, the following documents must be delivered:

- UT output data.
- Problem reports (if any).
2.4 Testing tasks

Before any testing in the UT phase can take place, the following tasks need to be done:

- Designing the unit tests.
- Tracing all test cases to components.
- All components mentioned in the DDD [2] need to be covered by test cases.
- Creation of the UT input data.
- Ensuring that all environmental needs for the UT have been satisfied.

When these tasks have been done a UT can be performed according to the procedures described in chapter 4.

2.5 Environmental needs

The following resources are needed in order to perform the UT:

- A computer connected to the Internet.
- A tablet connected to the Internet.
- A server running the kroket application.

See also the constraints described in the DDD [2].

2.6 Test case pass/fail criteria

Every test should describe the criteria that should be met to pass a specific test. An overall UT pass can only be achieved when all tests described in chapter 3 have been performed and passed.
Chapter 3

Testcase specification

Each testcase specification contains the following information:

• Unique testcase identifier.
• Test items, the items to be tested.
• Input specification, the input for the testcase.
• Output specification, the expected output from the testcase.
• Environmental needs, the test environment.

Detailed test information for each testcase will follow.

3.1 Client side tests

For the client side tests, you always need to have one of the following web browsers, so this is an environmental need for all testcases.

• Chrome version 20 or above
• Firefox version 13 or above
• Internet Explorer version 9 or above
• Safari version 5 or above
• Opera version 12 or above

3.1.1 Event handlers

UT1

Test items setYear declared in studyplanner.js.
Kroket – UNIT TEST PLAN

Input specifications

1. Call `setYear` with argument ‘2014’.

Output specifications

1. The `startYear` variable should be equal to 2014.

Environmental needs None.

UT2

Test items Tests `addYear` and `removeYear` in `studyplanner.js`.

Input specifications

1. Build the table and call `addYear` three times.
2. After calling `addYear` three times, call `removeYear` twice.
3. After three times `addYear` and two times `removeYear`, call `removeYear` once.

Output specifications

1. The table should now have 6 rows.
2. The table should now have 4 rows.
3. The table should now have 3 rows.

Environmental needs None.

UT3

Test items Function `fillInCourses` in `studyplanner.js`.

Input specifications

1. 2 courses, one in year 1 quartile 3 and one in year 2 quartile 4 from the user’s major. There are no courses in the schedule.
2. 2 courses, one in year 1 quartile 3 and one in year 2 quartile 4 from the user’s major. One of the two courses is already in the user’s schedule.
Output specifications

1. The two courses should be added to the schedule.
2. Only the course that was not yet in the schedule, should have been added.

Environmental needs None.

UT4

Test items Function updateMajorErrorbar in studyplanner.js.

Input specifications

1. Set the user’s major to unknown, and call updateMajorErrorBar(true).
2. Set the major to “Software science”, and call updateMajorErrorBar(true).
3. Set the user’s major to unknown, and call updateMajorErrorBar(false).

Output specifications

1. The error bar saying you do not have chosen a major yet should be shown (id major-error-bar).
2. The major error bar should no longer be shown, because we now have chosen a major.
3. The major error bar should not be visible, because the deepening/broadening buttons are not pressed (parameter false).

Environmental needs None.

UT5

Test items Function saveMajor in studyplanner.js.

Input specifications

1. We choose major “Test major” (with target group 1234) in the dialog in which you should fill in your major. Furthermore, 2013-2014 is selected as the first year.

Output specifications

1. The major should now be 1234, the majorName should be “Test major” and the startYear should be 2013.
Kroket – UNIT TEST PLAN

Environmental needs  None.

UT6

Test items  colorCourses in studyplanner.js.

Input specifications
1. Set the schedule to contain one major subject and one non-major subject. Call colorCourses.

Output specifications
1. The major course should have the class major.

Environmental needs  None.

UT7

Test items  removeCourse in studyplanner.js.

Input specifications  There is a course in the schedule and the delete icon of the subject is pressed by the javascript.

Output specifications  The subject is no longer in the schedule.

Environmental needs  None.

UT8

Test items  showSubjectInformation in studyplanner.js.

Input specifications  Initialize a new course object, using makeSubject. Call showSubjectInformation with this course.

Output specifications  The subject information corresponding to the course should be shown.

Environmental needs  None.

UT9
CHAPTER 3. TESTCASE SPECIFICATION

Test items checkDoubleCourses in studyplanner.js.

Input specifications A cell with 2 courses that are both given in timeslot A. Then checkDoubleCourses is called.

Output specifications true, there is a conflict.

Environmental needs None.

UT10

Test items emptySchedule in studyplanner.js.

Input specifications There is a subject in the user’s schedule, and then emptySchedule is called.

Output specifications All subjects are removed from the schedule.

Environmental needs None.

UT11

Test items setRecommendedSubjects (which is a helper function for recommending subjects)

Input specifications setRecommendSubjects is called data containing information about two subjects.

Output specifications The subjects should be added to the div with id recommended-subjects-container.

Environmental needs None.

3.1.2 Authentication

UT12

Test items isValidUsername declared in authentication.js.
Kroket – UNIT TEST PLAN

Input specifications  Execute the function isValidUsername with each of the following parameters.

1. SomeName
2. Some Name
3. Somename123
4. Some-Name
5. SomeName*
6. Names
7. Name
8. ThisNameIsReallyLong
9. ThisReallyIsALongName

Output specifications  The results of above function calls should be:

1. true, this is a valid username.
2. false, spaces are not allowed in a username.
3. true
4. true
5. false, a '*' in a username is not allowed.
6. true
7. false, this username is too short. A username should contain at least 5 characters.
8. true
9. false, this username is too long. A username may contain at most 20 characters.

Environmental needs  None.

UT13

Test items  isValidPassword declared in authentication.js.
CHAPTER 3. TESTCASE SPECIFICATION

Input specifications  Execute the function `isValidPassword` with each of the following parameters.

1. password
2. passwrd
3. Thisisaverylongpasswordthatisreallyreallylong
4. hééééééé
5. containsatab	

Output specifications  The results of above function calls should be:

1. true, this is a valid (but not very good) password.
2. false, this password is too short.
3. true
4. false, only ASCII characters are allowed.
5. false, ASCII control characters are not allowed.

Environmental needs  None.

3.1.3  Onload events

UT14

Test items  `getMajors` declared in load.js.

Input specifications  The function `getMajors` will ask the majors from the server. In this testcase the server is simulated on the client side by mockjax. For this testcase, the returned majors are “a major”, with target group 200, and “Another major”, with target group 201.

Output specification  The majors returned by the server (“A major” and “Another major”) should be added to the dropdown menu with id major.

Environmental needs  Uses mockjax to simulate the server, so None.

UT15

Test items  `getMajors` and `receiveMajors`, both declared in load.js.
Kroket – UNIT TEST PLAN

Input specifications  getMajors will be called, it will ask data from the real server. receiveMajors will be called with 3 fake majors: major A, major B and major C, with target groups 1, 2 and 3.

Output specifications  getMajors should receive majors from the server, with a name of type string. receiveMajors should show the majors in the dropdown menu with id “major”.

Environmental needs  Requires the server to be online and working. To make the test useful, the server should return at least one major, however this is not strictly necessary.

UT16

Test items  testBuildTable declared in load.js.

Input specifications  Calls buildTable with parameter 4, meaning a table of 4 years should be build.

Output specifications  The table body (id tbody) should have 4 rows (tr), one for each year. Also each cell should have an element with class addCourse.

Environmental needs  None.

3.1.4  Searching for packages

UT17

Test items  coherentSearchButtonWaiting as declared in searchPackages.js. The function will disable (or enable) the search button, depending on if a search is performed.

Input specifications  Call the function with each of the following arguments:

1. true
2. false

Output specifications  This should give the following results:

1. The search button should be disabled, meaning the attribute disabled of this button should be set to disabled.
2. The search button should not have the attribute disabled, meaning this attribute is not defined.
CHAPTER 3. TESTCASE SPECIFICATION

Environmental needs  None.

UT18

Test items  showCoherentPackages declared in searchPackages.js.

Input specifications  The function is called with each of the following parameters:

1. [] (an empty list)
2. [{name: ‘test’, target_group: 1}, {name: ‘test2’, target_group: 2}], 2 coherent packages with names “test” and “test2”.

Output specifications  The result of this function will be shown in the div with id search-results.

1. “No results found” should be shown, because there were no search results.
2. 2 packages must be shown. The first packet that was added should have name “test”.

Environmental needs  None.

3.1.5  Search for subjects

UT19

Test items  getPostDataSearch declared in searchSubjects.js.

Input specification  The search fields are filled in such that:

• subject-name has value “test” meaning as a search term we give “test”.

• year-1-button is pressed, meaning we want to search for subjects in year 1.

• broadening-button is pressed, meaning we want to search for broadening subjects.

• timeslot-B-button is pressed, we search for subjects given in timeslot B.

• quartile-3-button

Then the function getPostDataSearch. Also the person’s major is set to 1375 and the name of this major is “Software Science”.

18
Kroket – UNIT TEST PLAN

Output specification  A dictionary with the following fields:

- major is 1375
- year is 1
- searchTerm is “test”
- broadening is “y”
- timeSlot is “B”
- quartile is 3
- difficulty is “Basic”

Environmental needs  None.

UT20

Test items  showSearchResults declared in searchSubjects.js

Input specifications  Calls showSearchResults with the following input:

1. [{code:'code', name:'test', ects:5, planning:[{year:1, quartile:4, timeSlot:'A', target_group:'1111'}]]

2. [] An empty list

Output specifications

1. The subject should have been added to the search results, meaning a div with data with key courseInfo. In this data, the code should be equal to “code”.

2. The search results should contain no courses.

Environmental needs  None.

UT21

Test items  keywordHighlight as declared in searchSubjects.js. This will highlight the search term in the search results (subject name, subject description, etc).

Input specifications  We set the search term (lastsearchTerm) to the word “test”.

Output specifications  The output should be the following string:
“<span class='keyword-highlighting'>test</span>ing”
CHAPTER 3. TESTCASE SPECIFICATION

Environmental needs  None.

UT22

Test items  keywordRemoveHighlight as declared in searchSubjects.js.

Input specifications  We add the following string to the description of a subject:
“ <span class="keyword-highlighting">test</span>ing”

Output specifications  This string should be changed into “test”.

Environmental needs  None.

3.1.6  Save and load schedule

UT23

Test items  saveClick. This is the function that is called if the user clicks “save”. Declared in storage.js.

Input specifications  We run the following two tests:
1. A schedule is saved by pressing “save”. The save name (currentScheduleName) is set to “save name” by the script.
2. “save” is clicked again. (This clicking is automatically done by the javascript.)

Output specifications  The following is expected form the GUI:
1. A save name is asked. (Meaning the modal with id saveModal is shown.)
2. No save name is asked, because it is already known.

Environmental needs  None.

UT24

Test items  showSchedule, declared in storage.js.

Input specification  The major is set to 1, and the majorName is set to “hi”. We give the following parameter to showSchedule: A schedule with some courses in it, in the right format. It has 4 years. And the majorCourses (courses belonging to the person’s major) are also set.
Kroket – UNIT TEST PLAN

Output specification  The resulting schedule has 4 years and contains the right courses. Subjects that are also in the major have the class major (in the GUI they will be colored dark-blue). The button to add a year should be enabled, just like the button to remove a year.

Environmental needs  None.

3.1.7  Helper functions

UT25

Test items  packageTypeToString declared in utils.js.

Input specifications  This function is called with the following input:

1. 1
2. 3
3. 2

Output specifications  The expected output of the function is:

1. “USE package”
2. “Elective package”
3. “Major”

Environmental needs  None.

UT26

Test items  showDisabler and hideDisabler, both declared in utils.js.

Input specifications

1. showDisabler(‘‘test’’) is called.
2. hideDisabler() is called.

Output specifications

1. The loading screen (a div element with id schedule-disabler) is shown and contains the text “test”.
2. The loading screen is no longer shown.
CHAPTER 3. TESTCASE SPECIFICATION

Environmental needs  None.

UT27

Test items  Tests setModified in utils.js.

Input specification  The function is called twice
1. First with true
2. Then with false

Output specification
1. The boolean modifiedSchedule should now be true.
2. modifiedSchedule should now be false.

Environmental needs  None.

UT28

Test items  disableCourses in utils.js

Input specifications  The function is called under the following circumstances.
1. In the search results, we have the course “test”, which is not in the schedule.
2. In the search results we now have the course “test”, which is also in the schedule.
3. There are two subjects in the search results, both called “test”, and they are both in the schedule.

Output specifications  The results should be:
1. The course “test” should not be disabled.
2. The course “test” in the search results should be disabled.
3. Both courses in the search results should be disabled.

Environmental needs  None.

UT29

Test items  filterTimeSlot in utils.js.
Kroket – UNIT TEST PLAN

Input specifications
1. “A”
2. “A;B”

Output specifications
1. “A”
2. “A and B”

Environmental needs  None

UT30

Test items  getSchedule in utils.js.

Input specifications  We call this function with a few different schedules.
1. An empty schedule.
2. A schedule with the course “test” in quartile 3 in year 2.
3. A schedule with two subjects with code “test” in each quartile and 3 subjects in quartile 3 year 2.

Output specifications  The output should be:
1. [[],[],[],[],[],[],[],[],[],[],[],[]] An array with 12 empty arrays.
2. [[],[],[],[],[],[],’test’] An array with 12 arrays, where the seventh array contains the course “test”.
3. [['test’],['test’],['test’],['test’],['test’],['test’],['test’],['test’],['test’],['test’],['test’],['test’]]

Environmental needs  None.

UT31

Test items  setSubjectInfo in utils.js.

Input specifications
1. A dictionary where content is “This is content”, studygoal is “Some studygoal” and evaluation is 8.2.
CHAPTER 3. TESTCASE SPECIFICATION

Output specifications

1. The description as shown in the GUI (id subject-info-description) should now be equal to “This is content”, the studygoal should show the text “Some studygoal” and the evaluation should be 8.2.

Environmental needs  None.

UT32

Test items  highlight in utils.js. Used to highlight the table cells where a subject can be placed.

Input specifications  For testing, highlight is called with the following parameter:

1. A course with lastYear is 2012, quartile is 3 and timeSlot is “A”. The start year of the user is set to 2012.

Output specifications

1. The cell in the table at quartile 3, year 2012 (thus the first year) should now be highlighted, and it should also be the only highlighted cell in the table.

Environmental needs  None.

UT33

Test items  removeHighlight in utils.js.

Input specifications

1. All cells of the table where subjects can be placed (id table and class tableInnerCell) are highlighted, before removeHighlight is called.

Output specifications

1. No cell is highlighted (no cell has class highlight).

Environmental needs  None.

UT34

Test items  checkDoubleCoursesRecursive in utils.js.
Kroket – UNIT TEST PLAN

**Input specifications** The function is called with each of the following parameters. It will always be called with index 0.

1. Two courses that are both given in only timeslot A.
2. One course that is given in timeslot A, and another one that is given in timeslot B.
3. Two courses can both be followed in either timeslot A or C.
4. Two courses, one needs both timeslot A and timeslot C, the other one is given in either timeslot A or timeslot C.
5. Five courses: two given in timeslot A or C, one in B or A, one in B or D and one in timeslot E.

**Output specifications**

1. false
2. true
3. true
4. false
5. true

**Environmental needs** None.

UT35

---

**Test items** `makePackage` in `utils.js`

**Input specifications**

1. A dictionary where `remarks` is an empty string, `target_group` is 1431, `type` is 3 and `name` is “Smart Environment”.

**Output specifications**

1. A `div` element with text “Smart Environment” (not taking any additional spaces into account) and with data with key `target_group`, which is equal to 1431.

**Environmental needs** None.

UT36
CHAPTER 3. TESTCASE SPECIFICATION

Test items  makeSubjects in utils.js.

Input specifications

1. A dictionary with code “2IP90”, name “programming”, priorKnowledge is an empty string, ects is 5, difficulty is an empty string, planning is an array of 3 different plannings.

Output specifications

1. A div element with data with key courseInfo, which contains the above information about the course. The div element should show the code and the name of this subject.

Environmental needs  None.

UT37

Test items  noConflict in utils.js, a function used by checkDoubleCoursesRecursive

Input specifications

1. Three subjects, one with current timeslot A, one with slot B and one in slot C.
2. Two subjects, both with current timeslot A.

Output specifications

1. false, no problems, all subjects given in different timeslots.
2. true, the courses are in the same timeslot.

Environmental needs  None.

UT38

Test items  noCurrConflict in utils.js.

Input specifications  The function is called with each of the following pairs of parameters.

1. “A;B” and “A” (these both represent timeslots)
2. “C;B” and “A”
Kroket – UNIT TEST PLAN

Output specifications This should give the following output:
1. false, this is a conflict, both subjects are given in timeslot A.
2. true, no conflict indeed.

3.1.8 Checking the schedule

For testing these functions, we simulate the server. This means, when the function posts a request to the server, this request is not sent to the server, but the response is defined by us. We have 2 different responses.

failingTestData is a dictionary where major, use, coherent and overlap have status “FAIL”. major has one missing subject: “2WA20”, the use-package is “A package” and overlap has 2 overlapping subjects, which are “2WA30” and “2IP65”.

In passingTestData all tests by the server have status “PASS”. There is a use package named “A package” and there are 2 coherent packages. Of course there is no overlap and there are no missing major subjects.

UT39

Test items checkMajor in validator.js.

Input specifications
1. passingTestData
2. failingTestData

Output specifications
1. The field that shows if the major is correct (id validator-major) has class success.
2. The field that shows if the major is correct (id validator-major) has class error.

Environmental needs None.

UT40

Test items checkUse

Input specifications
1. passingTestData
2. failingTestData
Output specifications

1. The field that shows if there are enough use-packages (id validator-use) has class success.

2. The field that shows if there are enough use-packages (id validator-use) has class error.

Environmental needs  None.

UT41

Test items  checkCoherent

Input specifications

1. passingTestData

2. failingTestData

Output specifications

1. The field that shows if there are enough coherent packages (id validator-elective) has class success.

2. The field that shows if there are enough coherent packages (id validator-elective) has class error.

Environmental needs  None.

UT42

Test items  checkOverlap

Input specifications

1. passingTestData

2. failingTestData

Output specifications

1. The field that shows if there are overlapping packages (id validator-overlap) has class success.

2. The field that shows if there are overlapping packages (id validator-overlap) has class error.
Kroket – UNIT TEST PLAN

Environmental needs  None.

UT43

Test items  countEcts in validator.js.

Input specifications

1. Three subjects, each subject has 5 ECTS.
2. One subject with 180 ECTS.
3. Two subjects with 90 ECTS each.
4. Two subjects with 100 ECTS each.
5. 36 courses with 5 ECTS each.
6. An empty array.

Output specifications

1. 15
2. 180
3. 180
4. 200
5. 180
6. 0, no courses means no ECTS.

Environmental needs  None.

3.1.9  English language version

UT44

Test items  Tests if some of the strings in translation.js are defined (for the English translation).
CHAPTER 3. TESTCASE SPECIFICATION

Input specifications  Asks for the following strings:

1. _._.year The translation of the word “year”.

2. _._.aScheduleOfNameXDoesntExist Text saying a schedule of name ... does not exist.

3. Some random not existing string.

Output specifications

1. This string should be defined (and equal to “year”)

2. This string should be defined.

3. This string should not be defined.

Environmental needs  None.

3.2 Server side tests

3.2.1 Testing adding of subject and packet information

This tests the module addData.py on a unitesting level. To make testing possible some additional classes are imported. First the unittest-class from Django which basically adds the same functionality as the unittest-class of Python. Further we need to prepare some data to be able to query, therefore the models are imported. addData and parser are the two classes we want to test and categorize is needed to fill the database with correct data (after a course is added its keywords need to be found). More detailed information can be found in DDD [2] chapter 7.10.

A new and empty database is created when these tests are run. This means that before every test, some data is added to the database. We will describe this in the input specifications. The output specifications contains all the assertions that we test for. Before a test is run, the test-database is cleared by the function setUp. Note that for all test cases a properly working database should be available.

UT45

Test items  clearDB

Input specifications  Every table, concerning subject and packet information, is filled with an item.
Kroket – UNIT TEST PLAN

Output specifications To make sure the delete function works, first some items are added to the database. We also check that there is indeed at least one item in the table after adding the data.

1. The table CoursePlanning contains more than 0 objects.
2. The table SubjectFollowUpRelation contains more than 0 objects.
3. The table SubjectPriorKnowledgeRelation contains more than 0 objects.
4. The table KeywordSubjectRelation contains more than 0 objects.
5. The table KeywordPacketRelation contains more than 0 objects.
6. The table SubjectPacket contains more than 0 objects.
7. The table Keyword contains more than 0 objects.
8. The table Subject contains more than 0 objects.

After deleting it is only checked if there are no items left in the tables.

1. The table CoursePlanning contains 0 objects.
2. The table SubjectFollowUpRelation contains 0 objects.
3. The table SubjectPriorKnowledgeRelation contains 0 objects.
4. The table KeywordSubjectRelation contains 0 objects.
5. The table KeywordPacketRelation contains 0 objects.
6. The table SubjectPacket contains 0 objects.
7. The table Keyword contains 0 objects.
8. The table Subject contains 0 objects.

Environmental needs None.

UT46

Test items parseDoelgroep

Input specifications parseDoelgroep() has been called to make sure all information from Doelgroep.csv is added.
CHAPTER 3. TESTCASE SPECIFICATION

Output specifications
1. The major with target group="1365" has name “Industrial Engineering & Management Science”.
2. The major with target group="1365" has type SubjectPacket.Major.
3. The major with target group="1375" has name “Software Science”.
4. The major with target group="1375" has type SubjectPacket.Major.
5. The major with target group="1414" has name “Statistics and research methodology”.
6. The major with target group="1414" has type SubjectPacket.COHERENT.

Environmental needs None.

UT47

Test items  addCourse

Input specifications
1. The subject “1BV00” is added with difficulty “Basic”.
2. The subject “8WA00” is added with difficulty “”.
3. The non existing subject “NonEx” is added with difficulty “Basic”.

Output specifications
1. The subject with subjectcode “1BV00” has name “Business modeling”.
2. The subject with subjectcode “1BV00” has difficulty “Basic”.
3. The subject with subjectcode “1BV00” is worth 5 ECTS.
4. The subject with subjectcode “1BV00” has certain string as content.
5. The subject with subjectcode “1BV00” has an empty string as study goal.
6. The subject with subjectcode “8WA00” has name “Heart and lung”.
7. The subject with subjectcode “8WA00” has difficulty “”.
8. The subject with subjectcode “8WA00” is worth 5 ECTS.
9. The subject with subjectcode “8WA00” has “N.A.” as content.
10. The subject with subjectcode “8WA00” has a certain string as study goal.
11. There is a subject with subjectcode “NonEx”.

32
Kroket – UNIT TEST PLAN

Environmental needs  owinfo.tue.nl should be reachable.

UT48

Test items  addCoursePlanning

Input specifications
1. The course “oLCB4” is added with difficulty “”.
2. The major “Industrial Engineering & Management Science” is added.
3. The course “oSAB1” is added with difficulty “”.
4. The major “Building and Planning” is added.

Output specifications
1. The subject with subjectcode “oLCB4” is planned in year 1.
2. The subject with subjectcode “oLCB4” is planned in timeslot “A”.
3. The subject with subjectcode “oLCB4” is planned in quartile 3.
4. The subject with subjectcode “oSAB1” is scheduled 0 times.

Environmental needs  None.

UT49

Test items  getAdditioanalInfo

Input specifications  The course “2IP35” is added with difficulty “”.

Output specifications
1. The subject with subjectcode “2IP35” has url “http://wwwis.win.tue.nl/2IP35”.
2. The subject with subjectcode “2IP35” has department “Mathematics and Computer Science”.
3. The subject with subjectcode “2IP35” has a certain string as study goal.

Environmental needs  The ‘GeefVakGegevens’ function in biztalk should be reachable.

UT50
Chapter 3. Testcase Specification

Test items  `owisCourses`

Input specifications  The course “2lP35” is added with difficulty “”.

Output specifications  Before OWIS is requested:

1. The subject with subjectcode “2IP35” has url “”.

After OWIS is requested for more information:

1. The subject with subjectcode “2IP35” has url “http://wwwis.win.tue.nl/2IP35”.

2. The subject with subjectcode “2IP35” has department “Mathematics and Computer Science”.

3. The subject with subjectcode “2IP35” has a certain string as study goal.

Environmental needs  The ‘GeefVakGegevens’ function in biztalk should be reachable.

UT51

Test items  `addSoftwareScienceMajor`

Input specifications

1. The course “2lO90” is added with difficulty “”.

2. The major “Software Science” is added with target group 1375.

Output specifications

1. The subject with subjectcode “KROKo8” has name “Business information systems”.

2. The subject with subjectcode “2IP35” is planned in year 3 for the major with target group 1375.

3. The subject with subjectcode “2IP35” is planned in quartile 1 for the major with target group 1375.

4. The subject with subjectcode “2IP35” is planned in timeslot “?” for the major with target group 1375.

5. The subject with subjectcode “2lO90” is planned in the major with target group 1375.

Environmental needs  None.
3.2.2 Testing adding of subject and packet information

This tests the module addData.py on a unit testing level. To make testing possible some additional classes are imported. First the unittest-class from Django which basically adds the same functionality as the unittest-class of python. Further we need to prepare some data to be able to query, therefore the models are imported. addData and parser are two classes we want to test and categorize is also needed to fill the database with correct data (after a course is added its keywords need to be found). More detailed information can be found in DDD [2] chapter 7.10.

A new and empty database is created when these tests are run. This means that before every test, some data is added to the database. We will describe this in the input specifications. The output specifications contains all the assertions that we test for. Before a test is run, the test-database is cleared by the function setUp. Note that for all test cases a properly working database should be available.

UT52

Test items createTuples

Input specifications

1. The courses “oLABo”, “8TA03”, “2DE30” and “4CA00” are added.
2. The major “Industrial Engineering & Management Science” is added with target group 1365.
3. The course “oLABo” is added to the major with target group 1365 in year 2012, quartile 3 and timeslot “A”.
4. The user “aap” is created with password “noot”.
5. The shedule “mies” is created with the four added courses.

Output specifications

1. There is a tuple (course8TA03, course2DE30, 1365) in the list tempDB.
2. There is a tuple (course2DE30, course8TA03, 1365) in the list tempDB.
3. There is a tuple (course8TA03, course4CA00, 1365) in the list tempDB.
4. There is a tuple (course4CA00, course8TA03, 1365) in the list tempDB.
5. There is a tuple (course2DE30, course4CA00, 1365) in the list tempDB.
6. There is a tuple (course4CA00, course2DE30, 1365) in the list tempDB.
7. There is no tuple (courseoLABo, course8TA03, 1365) in the list tempDB.
8. There is no tuple (courseoLABo, course2DE30, 1365) in the list tempDB.
9. There is no tuple (course4CA00, courseoLABo, 1365) in the list tempDB.
10. There is no tuple (course8TA03, course2DE30, 1367) in the list tempDB.

Environmental needs  None.

UT53

Test items  deleteOldTable

Input specifications
1. tempDB is assigned the predefined list.
2. tempDB is processed into the database

Output specifications
1. The RelatedSubject table contains more then 0 objects before deleting
2. The RelatedSubject table contains exactly 0 objects after deleting

Environmental needs  None.

UT54

Test items  getNrOfDoubles

Input specifications  tempDB is assigned the predefined list.

Output specifications
1. getNrOfDoubles(“oLABo”, “8TA03”, 1365) returns 3.
2. getNrOfDoubles(“oLABo”, “8TA03”, 1400) returns 1.
3. getNrOfDoubles(“oLABo”, “2DE30”, 1400) returns 1.
4. getNrOfDoubles(“8TA03”, “oLABo”, 1365) returns 3.
5. getNrOfDoubles(“4CA00”, “4CA00”, 1431) returns 4.
6. getNrOfDoubles(“4CA00”, “4CA00”, 1431) returns 0 directly after the previous is asked.

The last assertion checks if the items are deleted after the function has been called.
Kroken – UNIT TEST PLAN

Environmental needs None.

UT55

Test items deleteDoubles

Input specifications tempDB is assigned the predefined list.

Output specifications

1. “oLABo” is related to “8TA03”.
2. “oLABo” is related to “2DE30”.
3. “8TA03” is related to “oLABo”.
4. “2DE30” is related to “oLABo”.
5. “4CA00” is not related to “oLABo”.

The last assertion checks if the items are deleted after the function has been called.

Environmental needs None.

3.2.3 Testing retrieving information from OWIS

There is no module for testing biztalk.py. This is due to the fact that this is all already tested in “test_GetAditionallInfo” in the addData_tester.py file. The function getAdditionalInfo in the addData.py file retrieves the data from biztalk and applies that to a subject, which would be exactly how we test this function.

3.2.4 Testing schedule related tasks

All tasks related to the schedule fall into this unittest. Each procedure is tested separately, which implies that for every test a new database is needed. Therefore, we assume that a working, empty database is available for each test.

The following methods are tested, all have the prefix /query/schedule:

- /save/saveschedule
- /load.loadschedule
- /list/userschedules
- /delete/removeschedule
CHAPTER 3. TESTCASE SPECIFICATION

- /rename/renameScheduleName
- /validate/validate

The following methods are also used, and assumed that they work. Hence this unittest is not meant to test these methods.

- /query/user/authenticate/register
- /query/user/authenticate/login
- /query/user/authenticate/logout
- /schedule/daily_jobs/subject_updater/addData/clearDB

Tested items are listed with only their method name. Their location in file and directory can be retrieved by using the previous list.

Only relevant information is shown in each variable name. For example, what is needed to create an account is not relevant. However, a name of the account is used to save a schedule, hence that is shown, but a password is not. Testing account registration is done in other unittests.

UT56

Test items load, save

Input specifications

1. Account a
   - (String) "Aeris", name of the account

2. Schedule s
   - (String) "My schedule", name of the schedule
   - (String) "-1", id of the major of the schedule
   - (String) "2012", begin year of the schedule
   - (JSON) "["2IP35","","","","","2IL15"]", codes of subjects in the schedule

Output specifications Registers an account a, a logs in, saves a schedule s, a logs out, a logs in, loads s, a logs out.

1. (String) Loaded major of the schedule s is equal to the major of the schedule of the input.
Kroket – UNIT TEST PLAN

2. (String) Loaded begin year of the schedule is equal to the begin year of the schedule of the input.

3. (JSON) Loaded codes of subjects in the schedule is equal to codes of subjects in the schedule of the input.

UT57

Test items  load,save

Input specifications

1. Account a
   - (String) “Aeris”, name of account 1

2. Account b
   - (String) “Cloud”, name of account 2

3. Schedule s
   - (String) “My schedule”, name of the schedule
   - (String) “-1”, id of the major of the schedule
   - (String) “2012”, begin year of the schedule
   - (JSON) “[“2IP35”],[“IL15”]”, codes of subjects in the schedule

Output specifications  Registers account a and b, a logs in, saves schedule s, a logs out, b logs in, loads s, b logs out.

1. (Bool) Load returns FALSE, as there is no schedule s saved on account b named “Empty” (schedule is saved on account a).

UT58

Test items  load,save

Input specifications

1. Account a
   - (String) “Aeris”, name of the account

2. Schedule s
CHAPTER 3. TESTCASE SPECIFICATION

- (String) "My schedule", name of the schedule
- (String) "-1", id of the major of the schedule
- (String) "2012", begin year of the schedule
- (JSON) "["2IP35"],["IL15"]", codes of subjects in the schedule

3. Schedule t

- (String) "Empty", name of schedule

Output specifications  Registers account a, a logs in, saves schedule s, a logs out, a logs in, loads t, a logs out.

1. (Bool) Load returns FALSE, as there is no schedule t saved which is named "Empty".

UT59

Test items  load, save

Input specifications

1. Account a

- (String) "Aeris", name of the account

2. Schedule s

- (String) "My schedule", name of the schedule
- (String) "-1", id of the major of the schedule
- (String) "2012", begin year of the schedule
- (JSON) "["2IP35"],["IL15"]", codes of subjects in the schedule

3. Schedule t

- (String) "My schedule", name of schedule 2
- (String) "-1", id of the major of schedule 2
- (String) "3483", begin year of schedule 2
- (JSON) "["IL15"]", codes of subjects in schedule
Output specifications Registers account a, a logs in, saves schedule s, a logs in, saves t, a logs out, a logs in, loads t.

1. (String) Loaded major of the schedule is equal to the major of schedule t of the input.
2. (String) Loaded begin year of the schedule is equal to the begin year of schedule t of the input.
3. (JSON) Loaded codes of subjects in the schedule of the schedule is equal to codes of subjects in schedule t of the input.

UT60

Test items save, delete

Input specifications

1. Account a
   - (String) ”Aeris”, name of the account

2. Schedule s
   - (String) ”My schedule”, name of the schedule
   - (String) ”-1”, id of the major of the schedule
   - (String) ”2012”, begin year of the schedule
   - (JSON) "["21P5","35","","","","21I5"]", codes of subjects in the schedule

3. Schedule t
   - (String) ”My schedule”, name of schedule
   - (String) ”-1”, id of the major of schedule
   - (String) ”3483”, begin year of schedule
   - (JSON) "["","","","","","","","","","""]", codes of subjects in schedule

Output specifications Registers account a and b, a logs in, saves schedule s and t, a logs out, a logs in, deletes s, a logs out.

1. (Integer) The number of schedules named ”My schedule” on account a should be 0.
2. (Integer) The number of schedules named ”Emtpy” on account a should be 1.

UT61

Test items save, load, rename
CHAPTER 3. TESTCASE SPECIFICATION

Input specifications

1. Account a
   - (String) "Aeris", name of the account

2. Account b
   - (String) "Cloud", name of the account

3. Schedule s
   - (String) "My schedule", name of the schedule
   - (String) "Strife", rename of the schedule
   - (String) "-1", id of the major of the schedule
   - (String) "2012", begin year of the schedule
   - (JSON) ":[["2IP35"],["1"],["1"],["2IL15"]", codes of subjects in the schedule

4. Schedule t
   - (String) "My schedule", name of schedule
   - (String) "-1", id of the major of schedule
   - (String) "3483", begin year of schedule
   - (JSON) "[["1","1","1","1","1","1"]", codes of subjects in schedule

Output specifications  Registers account a and b, a logs in, saves schedule s and t, a logs out, a logs in, renames s, loads s, a logs out.

1. (Integer) The number of schedules named "My schedule" on account a should be 0.

2. (String) Loaded major of the schedule is equal to the major of schedule s of the input.

3. (String) Loaded begin year of the schedule is equal to the begin year of schedule s of the input.

4. (JSON) Loaded codes of subjects in the schedule of the schedule is equal to codes of subjects in schedule s of the input.

UT62

Test items  save, load, userschedules
Kroket – UNIT TEST PLAN

Input specifications

1. Account a
   - (String) "Aeris", name of the account

2. Schedule s
   - (String) "My schedule", name of the schedule
   - (String) "1", id of the major of the schedule
   - (String) "2012", begin year of the schedule
   - (JSON) "[["2IP35"],["1"],["2IL15"]], codes of subjects in the schedule

3. Schedule t
   - (String) "My schedule", name of schedule
   - (String) "1", id of the major of schedule
   - (String) "3483", begin year of schedule
   - (JSON) "[["1"],["2"],["3"]], codes of subjects in schedule

Output specifications

Registers account a, a logs in, saves schedule s and t, a logs out, a logs in, loads list, a logs out.

1. (Bool) Loaded list should contain name "Empty".

2. (Bool) Loaded list should contain name "My Schedule".

3. (Integer) Loaded list should have a length of 2.

3.2.5 Testing recommendation related tasks

All tasks related to the recommendation fall into this unittest. Each procedure is tested separately, which implies that for every test a new database is needed. Therefore, we assume that a working, empty database is available for each test.

The following method is tested: /query/recommendation/subject/view

The following methods are also used, and assumed that they work. Hence this unittest is not meant to test these methods.

- /query/user/authenticate/register
- /query/user/authenticate/login
- /query/user/authenticate/logout
CHAPTER 3. TESTCASE SPECIFICATION

- /schedule/daily_jobs/subject_updater/addData/clearDB

The tested item is listed with only their method name. Their location in file and directory can be retrieved by using the previous list.

UT63

Test items view

Input specifications

1. The following tuples are added to the database. The tuples are based on predefined schedules.

   - ("4GA10", "0HV40", 3, 1368) in the database.
   - ("4GA10", "4GA30", 3, 1368) in the database.
   - ("0HV40", "4GA10", 3, 1368) in the database.
   - ("0HV40", "4GA30", 3, 1368) in the database.
   - ("4GA30", "4GA10", 3, 1368) in the database.
   - ("4GA30", "0HV40", 3, 1368) in the database.
   - ("4GA10", "0HV40", 1, 1375) in the database.
   - ("4GA10", "4GA30", 1, 1375) in the database.
   - ("0HV40", "4GA10", 1, 1375) in the database.
   - ("0HV40", "4GA30", 1, 1375) in the database.
   - ("4GA30", "4GA10", 1, 1375) in the database.
   - ("4GA30", "0HV40", 1, 1375) in the database.

2. Schedule s:

   - (JSON) "[]["4GA10"]" schedule to get recommendations from.

Output specifications  One query is fired. The result should match the expectation.

1. (Bool) This should be true since subject with code "4GA30" is in the result of recommendations.

2. (Bool) This should be true since subject with code "0HV40" is in the result of recommendations.

3. (Integer) This should be 2 since the earlier mentioned subjects are the only ones related to "4GA10".
3.2.6 Testing package related tasks

All tasks related to the packages fall into this unittest. Each procedure is tested separately, which implies that for every test a new database is needed. Therefore, we assume that a working, empty database is available for each test.

The following methods are tested, all have the prefix /query/package:

- /subjects/searchpackage
- /search/searchpackage

The following method is also used: we assume that it works. Hence this unittest is not meant to test these methods. /schedule/daily_jobs/subject_updater/addData/clearDB

UT64

Test items /search/searchpackage

Input specifications

1. Packet p
   - (String) “Medic Master”, name of the packet.
   - (String) “Become the medic who knows to use any method.”, remarks of the packet.
   - (Integer) 1377, id of the packet

2. Packet q
   - (String) “Medical Guru”, name of the packet.
   - (String) “To become the omniscient one in medical knowledge.”, remarks of the packet.
   - (Integer) 1378, id of the packet

3. Packet r
   - (String) “Drug Dojo, name of the packet.
   - (String), “Every kind of medication will be known to you.”, remarks of the packet.
   - (Integer) 1379, id of the packet
CHAPTER 3. TESTCASE SPECIFICATION

Output specifications  Numerous searches are performed. Their respective expectations are compared with the search results.

1. (list) The search to “medical” should return the list containing only id 1378.

2. (list) The search to “med” should return the list containing id 1377, 1378 and 1379.

3. (list) The search to “method knowledge willpower” should return the list only containing id 1378.

4. (list) The search to “blasphemy” should return an empty list.

5. (list) The search to “on in you” should return the list only containing id 1379.

UT65

Test items /subjects/searchpackage

Input specifications

1. (undefined) Packet labeled as id 1377 with subject named “Medical stuff”.

2. (undefined) Packet labeled as id 1378 with subject named “Medicine Chief”.

3. Packet p
   • (Integer) 1377, id of the packet.
   • (Subject) “Medical stuff”, name of a subject in packet.

4. Packet q
   • (Integer) 1378, id of the packet
   • (Subject) “Medical Chief”, name of a subject in packet.

5. Packet r
   • (Integer) 1379, id of the packet

Output specifications  A query is fired. It should return all subjects in the respective packet.

1. (Subject) When all subjects are retrieved from packet with id 1377 only the subject “Medical stuff” should be retrieved.

2. (Integer) The number of subjects retrieved from packet with id 1377 should be 1.
### 3.2.7 Testing security related tasks

All tasks related to the security fall into this unittest. Each procedure is tested separately, which implies that for every test a new database is needed. Therefore, we assume that a working, empty database is available for each test.

The following method is tested: `/security/check_credentials`

It should be noted that security contains more methods than the only one previous mentioned. However, these other methods depend on requests, which are heavily used by the client. Also, these methods are dependent on the request by the client, hence they are tested by their respective unittest.

**UT66**

---

**Test items**  check_credentials

**Input specifications**

1. (undefined) Account with username “Aeris” and password “Gainsborough”.
2. (undefined) Account with username “Yuffie” and password “Kisaragi”.
3. (undefined) Account with password “Sephiroth” and password “Sephiroth”.

**Output specifications**

1. (Bool) Tries the combination of username “Aeris” with password “Gainsborough”. It should return true.
2. (Bool) Tries the combination of username “Aeris” with password “Kisaragi”. It should return false.
3. (Bool) Tries the combination of username “Aeris” with password “Sephiroth”. It should return false.
4. (Bool) Tries the combination of username “Aeris” with password “Aeris”. It should return false.
5. (Bool) Tries the combination of username “Aeris” with password “Yuffie”. It should return false.
6. (Bool) Tries the combination of username “Yuffie” with password “Gainsborough”. It should return false.
7. (Bool) Tries the combination of username “Yuffie“ with password “Kisaragi“. It should return true.

8. (Bool) Tries the combination of username “Yuffie“ with password “Sephiroth“. It should return false.

9. (Bool) Tries the combination of username “Yuffie“ with password “Aeris“. It should return false.

10. (Bool) Tries the combination of username “Yuffie“ with password “Yuffie“. It should return false.

11. (Bool) Tries the combination of username “Sephiroth“ with password “Gainsborough“. It should return false.

12. (Bool) Tries the combination of username “Sephiroth“ with password “Kisaragi“. It should return false.

13. (Bool) Tries the combination of username “Sephiroth“ with password “Sephiroth“. It should return true.

14. (Bool) Tries the combination of username “Sephiroth“ with password “Aeris“. It should return false.

15. (Bool) Tries the combination of username “Sephiroth“ with password “Yuffie“. It should return false.

3.2.8 Testing account related tasks

All tasks related to the schedule fall into this unittest. Each procedure is tested separately, which implies that for every test a new database is needed. Therefore, we assume that a working, empty database is available for each test.

The following methods are tested, all have the prefix /query/user:

- /change/changeProperties
- /authenticate/login
- /authenticate/logout
- /authenticate/info
- /authenticate/register

UT67

Test items  login
Kroket – UNIT TEST PLAN

Input specifications


2. (list) ["", "1234", "abcd", "Spatie s", "unicode\u4500", "!B@CDEF A", "Seven", "CORRECT_USERNAME", "Piet R\u00E9mer"], list of invalid usernames.

3. (list) ["t93KCd4SA", "abcddefg", "My birth date", "This is a very long string. This is allowed as we only require that the string is at least 8 characters long and contains only ASCII CHARACTERS", "09112002", "Voornaam", "SASXER54", "Amsterdam", "01234567"], list of valid passwords.


Output specifications

1. (Bool) Every i\textsuperscript{th} element of lists of valid usernames and valid passwords are combined. Login should return false, as the accounts are not registered.

2. (Bool) Every i\textsuperscript{th} element of lists of invalid usernames and invalid passwords are combined. Login should return false, as the accounts are not registered.

UT68

Test items login, logout, register, info

Input specifications


2. (list) ["t93KCd4SA", "abcddefg", "My birth date", "This is a very long string. This is allowed as we only require that the string is at least 8 characters long and contains only ASCII CHARACTERS", "09112002", "Voornaam", "SASXER54", "Amsterdam", "01234567"], list of valid passwords.

Output specifications

1. (Bool) Every i\textsuperscript{th} element of lists of valid usernames and valid passwords are combined. Login should return false, as the account is not registered.

2. (Bool) Every i\textsuperscript{th} element of lists of invalid usernames and invalid passwords are combined. Login should return false, as the accounts are not registered.

3. (Bool) Every i\textsuperscript{th} element of lists of valid usernames and valid passwords are combined. Login should return false, as only the password is sent.

4. (Bool) Every i\textsuperscript{th} element of lists of valid usernames and valid passwords are combined. Login should return false, as only the username is sent.
CHAPTER 3. TESTCASE SPECIFICATION

4. (Bool) Every $i^{th}$ element of lists of valid usernames and valid passwords are combined. Login should return false, as nothing is sent.

5. (Bool) Every $i^{th}$ element of lists of valid usernames and valid passwords are combined. Login should return false, as no POST request is made.

6. (Bool) Every $i^{th}$ element of lists of valid usernames and valid passwords are combined. Login should return false, as no AJAX request is made.

7. (Bool) Every $i^{th}$ element of lists of valid usernames and valid passwords are combined. Login should return true, as the account is registered, and username and password is sent as a POST and AJAX request.

8. (Bool) Every $i^{th}$ element of lists of valid usernames and valid passwords are combined. Login should return false, as the account is deleted.

9. (Bool) Every $i^{th}$ element of lists of valid usernames and valid passwords are combined. Logout should return true, as the account is deleted.

UT69

Test items  login, logout, register, info

Input specifications

1. (list) ["", "1234", "abcd", "Spatie s", u"unicode_\u4500", "!B@CDEF A", Seven ", "CORRECT_USERNAME", u"Piet R\u00E9mer"], list of invalid usernames.

2. (list) ["CORRECT_PASSWORD", "LAl", "12", "345", u"ABCED", u"LONG ENOUGH BUT GREEK: β", "7", u"Invalid Password é", "9.00"], list of invalid passwords.

Output specifications

1. (Bool) Every $i^{th}$ element of lists of valid usernames and valid passwords are combined. Login should return false, as the account is not registered.

2. (Bool) Every $i^{th}$ element of lists of valid usernames and valid passwords are combined. Login should return false, as only the password is sent.

3. (Bool) Every $i^{th}$ element of lists of valid usernames and valid passwords are combined. Login should return false, as only the username is sent.

4. (Bool) Every $i^{th}$ element of lists of valid usernames and valid passwords are combined. Login should return false, as nothing is sent.

5. (Bool) Every $i^{th}$ element of lists of valid usernames and valid passwords are combined. Login should return false, as no POST request is made.
Kroket – UNIT TEST PLAN

6. (Bool) Every \(i^{th}\) element of lists of valid usernames and valid passwords are combined. Login should return false, as no AJAX request is made.

7. (Bool) Every \(i^{th}\) element of lists of valid usernames and valid passwords are combined. Login should return false, as the account name or password contain invalid characters.

8. (Bool) Every \(i^{th}\) element of lists of valid usernames and valid passwords are combined. Login should return false, as the account is deleted.

9. (Bool) Every \(i^{th}\) element of lists of valid usernames and valid passwords are combined. Logout should return true, as the account is deleted.

UT70

Test items login, logout, register, info

Input specifications


2. (list) [“t93KCd4SA”, “abcddfeg”, “My birth date”, “This is a very long string. This is allowed as we only require that the string is at least 8 characters long and contains only ASCII CHARACTERS”, “09112002”, “Voornaam”, “SASXER54”, “Amsterdam”, “01234567”], list of valid passwords.

Output specifications

1. (Bool) Every \(i^{th}\) element of lists of valid usernames and valid passwords are combined. Logout should return false, as the account is not logged in.

2. (Bool) Every \(i^{th}\) element of lists of valid usernames and valid passwords are combined. Login should return true, as the account is registered, and username and password is sent as a POST and AJAX request.

3. (Bool) Every \(i^{th}\) element of lists of valid usernames and valid passwords are combined. Logout should return false, as no AJAX request is made.

4. (Bool) Every \(i^{th}\) element of lists of valid usernames and valid passwords are combined. Info should return true, as the account is registered and a POST and AJAX request.

5. (Bool) Every \(i^{th}\) element of lists of valid usernames and valid passwords are combined. Login should return false, as no POST request is made.
CHAPTER 3. TESTCASE SPECIFICATION

6. (Bool) Every $i^{th}$ element of lists of valid usernames and valid passwords are combined. Info should return true, as the account is registered and a POST and AJAX request.

7. (Bool) Every $i^{th}$ element of lists of valid usernames and valid passwords are combined. Login should return false, as no POST nor AJAX request is made.

8. (Bool) Every $i^{th}$ element of lists of valid usernames and valid passwords are combined. Info should return true, as the account is registered and a POST and AJAX request.

9. (Bool) Every $i^{th}$ element of lists of valid usernames and valid passwords are combined. Logout should return true, as the account is registered, and username and password is sent as a POST and AJAX request, and logged in.

10. (Bool) Every $i^{th}$ element of lists of valid usernames and valid passwords are combined. Info should return false, as the account is registered and a POST and AJAX request, but logged out.

UT71

Test items logout, info

Input specifications


2. (list) ["t93KCd4SA", "abcddefg", "My birth date", "This is a very long string. This is allowed as we only require that the string is at least 8 characters long and contains only ASCII CHARACTERS", "09112002", "Voornaam", "SASXER54", "Amsterdam", "01234567"], list of valid passwords.

Output specifications

1. (Bool) Every $i^{th}$ element of lists of valid usernames and valid passwords are combined. Logout should return true, as the account is registered, and username and password is sent as a POST and AJAX request, and logged in.

2. (Bool) Every $i^{th}$ element of lists of valid usernames and valid passwords are combined. Info should return false, as the account is registered and AJAX request, but logged out.

UT72
Kroket – UNIT TEST PLAN

Test items  login, register

Input specifications


2. (list) ["t93KCd4SA", "abcddefg", "My birth date", "This is a very long string. This is allowed as we only require that the string is at least 8 characters long and contains only ASCII CHARACTERS", "09112002", "Voornaam", "SASXER54", "Amsterdam", "01234567"], list of valid passwords.

Output specifications

1. (Bool) Every i\textsuperscript{th} element of lists of valid usernames and valid passwords are combined. Register should return true, as the account is not registered, and username and password is sent as a POST and AJAX request.

2. (Bool) Every i\textsuperscript{th} element of lists of valid usernames and valid passwords are combined. Login should return true, as the account is registered, and username and password is sent as a POST and AJAX request.

3. (Bool) Every element of valid usernames is combined with every element of valid passwords. Register should return true, as the account is not registered, and username and password is sent as a POST and AJAX request.

UT\textsuperscript{73}

Test items  login, register

Input specifications

1. (list) ["", "1234", "abcd", "Spatie s", u"unicode\_u4500", "!B@CDEF A", Seven ", "CORRECT_USERNAME", u"Piet R\u100E9m"], list of invalid usernames.

2. (list) ["CORRECT_PASSWORD", "LAI", "12", "345", u"ABCED", u"LONG ENOUGH BUT GREEK: β", "γ", u"Invalid Password é", "9.00"], list of invalid passwords.

Output specifications

1. (Bool) Every i\textsuperscript{th} element of lists of invalid usernames and invalid passwords are combined. Register should return false, as the account name or password contain invalid characters.

2. (Bool) Every i\textsuperscript{th} element of lists of invalid usernames and invalid passwords are combined. Login should return false, as the account is not registered.
CHAPTER 3. TESTCASE SPECIFICATION

UT74

Test items  login, register

Input specifications


2. (list) ["", "1234", "abcd", "Spatie s", u"unicode\u4500", "!B@CDEF A", Seven", "CORRECT_USERNAME", u"Piet R\u00E9mer"], list of invalid usernames.

3. (list) ["t93KCd4SA", "abcddefg", "My birth date", "This is a very long string. This is allowed as we only require that the string is at least 8 characters long and contains only ASCII CHARACTERS", "09112002", "Voornaam", "SASXER54", "Amsterdam", "01234567"], list of valid passwords.

4. (list) ["CORRECT_PASSWORD", "LAI"", "12", "345", u"ABCED", u"LONG ENOUGH BUT GREEK: β", "7", u"Invalid Password é”, "9.00"], list of invalid passwords.

Output specifications

1. (Bool) Every $i$th element of lists of valid usernames and valid passwords are combined. Register should return false, as only the password is sent.

2. (Bool) Every $i$th element of lists of valid usernames and valid passwords are combined. Register should return false, as only the username is sent.

3. (Bool) Every $i$th element of lists of valid usernames and valid passwords are combined. Register should return false, as nothing is sent.

4. (Bool) Every $i$th element of lists of valid usernames and valid passwords are combined. Login should return false, as the account is not registered.

5. (Bool) Every $i$th element of lists of valid usernames and valid passwords are combined. Register should return false, as only the password is sent.

6. (Bool) Every $i$th element of lists of valid usernames and valid passwords are combined. Register should return false, as only the username is sent.

7. (Bool) Every $i$th element of lists of valid usernames and valid passwords are combined. Register should return false, as nothing is sent.

8. (Bool) Every $i$th element of lists of invalid usernames and invalid passwords are combined. Register should return false, as only the password is sent.

9. (Bool) Every $i$th element of lists of invalid usernames and invalid passwords are combined. Register should return false, as only the username is sent.
Kroket – UNIT TEST PLAN

10. (Bool) Every $i^{th}$ element of lists of invalid usernames and invalid passwords are combined. Register should return false, as nothing is sent.

11. (Bool) Every $i^{th}$ element of lists of invalid usernames and invalid passwords are combined. Login should return false, as the account is not registered.

12. (Bool) Every $i^{th}$ element of lists of invalid usernames and invalid passwords are combined. Register should return false, as only the password is sent.

13. (Bool) Every $i^{th}$ element of lists of invalid usernames and invalid passwords are combined. Register should return false, as only the username is sent.

14. (Bool) Every $i^{th}$ element of lists of invalid usernames and invalid passwords are combined. Register should return false, as nothing is sent.

UT75

Test items login, info

Input specifications


2. (list) ["t93Kcd4SA", "abcdefefg", "My birth date", "This is a very long string. This is allowed as we only require that the string is at least 8 characters long and contains only ASCII CHARACTERS", "09112002", "Voornaam", "SASXER54", "Amsterdam", "01234567"], list of valid passwords.

Output specifications

1. (Bool) Every $i^{th}$ element of lists of valid usernames and valid passwords are combined. Login should return true, as the account is registered and a POST and AJAX request.

2. (Bool) Every $i^{th}$ element of lists of valid usernames and valid passwords are combined. Info should return true, as the account is registered and logged in.

3. (Bool) Every $i^{th}$ element of lists of valid usernames and valid passwords are combined. Info should return false, as the account is logged out.

UT76

Test items info
CHAPTER 3. TESTCASE SPECIFICATION

Input specifications


2. (list) ["", "1234", "abcd", "Spatie s", u"unicode\unicode4500", "!B@CDEF A", Seven", "CORRECT_USERNAME", u"Piet R\unicode90E9mer"], list of invalid usernames.

3. (list) ["193KCd4SA", "abcdfefg", "My birth date", "This is a very long string. This is allowed as we only require that the string is at least 8 characters long and contains only ASCII CHARACTERS", "09112002", "Voornaam", "SASXER54", "Amsterdam", "01234567"], list of valid passwords.

4. (list) ["CORRECT_PASSWORD", "LAI", "12", "345", u"ABCED", u"LONG ENOUGH BUT GREEK: β", "7", u"Invalid Password é", "9.00"], list of invalid passwords.

Output specifications

1. (Bool) Every $i^{th}$ element of lists of valid usernames and valid passwords are combined. Info should return false, as the account is logged out.

2. (Bool) Every $i^{th}$ element of lists of invalid usernames and invalid passwords are combined. Info should return false, as the account is not registered.

UT77

Test items  login, info

Input specifications


2. (list) ["193KCd4SA", "abcdfefg", "My birth date", "This is a very long string. This is allowed as we only require that the string is at least 8 characters long and contains only ASCII CHARACTERS", "09112002", "Voornaam", "SASXER54", "Amsterdam", "01234567"], list of valid passwords.

Output specifications

1. (Bool) Every $i^{th}$ element of lists of valid usernames and valid passwords are combined. Info should return false, as no AJAX request is made.

2. (Bool) Every $i^{th}$ element of lists of invalid usernames and invalid passwords are combined. Info should return false, as no AJAX request is made.

UT78
Kroket – UNIT TEST PLAN

Test items  changeProperties, info, logout

Input specifications

2. (list) ["t93KCd4SA", "abcdfeq", "My birth date", "This is a very long string. This is allowed as we only require that the string is at least 8 characters long and contains only ASCII CHARACTERS", "09112002", "Voornaam", "SASXER54", "Amsterdam", "01234567"], list of valid passwords.

Output specifications
1. (Bool) Every i\textsuperscript{th} element of lists of valid usernames and valid passwords are combined. changeProperties should return true, as the account is logged in.

2. (Bool) Every i\textsuperscript{th} element of lists of invalid usernames and invalid passwords are combined. Info should return false, as no AJAX request is made.

3. (Bool) Every i\textsuperscript{th} element of lists of valid usernames and valid passwords are combined. Logout should return true, as the account was logged in.

UT79

Test items  changeProperties, info, logout

Input specifications

2. (list) ["t93KCd4SA", "abcdfeq", "My birth date", "This is a very long string. This is allowed as we only require that the string is at least 8 characters long and contains only ASCII CHARACTERS", "09112002", "Voornaam", "SASXER54", "Amsterdam", "01234567"], list of valid passwords.

Output specifications
1. (Bool) Every i\textsuperscript{th} element of lists of valid usernames and valid passwords are combined. changeProperties should return false, as the wrong password of the account is sent.

2. (Bool) Every i\textsuperscript{th} element of lists of invalid usernames and invalid passwords are combined. Info should return false, as the account is not logged in.

UT80
CHAPTER 3. TESTCASE SPECIFICATION

Test items changeProperties, info, logout

Input specifications


2. (list) ["t93KCd4SA", "abcdfeg", "My birth date", "This is a very long string. This is allowed as we only require that the string is at least 8 characters long and contains only ASCII CHARACTERS", "09112002", "Voornaam", "SASXER54", "Amsterdam", "01234567"], list of valid passwords.

Output specifications

1. (Bool) Every i\textsuperscript{th} element of lists of valid usernames and valid passwords are combined. changeProperties should return true, as the password belonging to the account is sent, together with a valid new password.

2. (Bool) Every i\textsuperscript{th} element of lists of invalid usernames and invalid passwords are combined. Info should return true, as the account is logged in.

3. (Bool) Every i\textsuperscript{th} element of lists of invalid usernames and invalid passwords are combined. Logout should return true, as the account was logged in.

UT81

Test items changeProperties, info, logout, login

Input specifications


2. (list) ["t93KCd4SA", "abcdfeg", "My birth date", "This is a very long string. This is allowed as we only require that the string is at least 8 characters long and contains only ASCII CHARACTERS", "09112002", "Voornaam", "SASXER54", "Amsterdam", "01234567"], list of valid passwords.

3. (list) ["CORRECT_PASSWORD", "LAI","12","345","ABCED","LONG ENOUGH BUT GREEK: β","7","Invalid Password é","9.00"], list of invalid passwords.
Output specifications

1. (Bool) Every \(i\)th element of lists of valid usernames and valid passwords are combined. changeProperties should return false, as the password belonging to the account is sent, but an unvalid new password.

2. (Bool) Every \(i\)th element of lists of invalid usernames and invalid passwords are combined. Logout should return true, as the account was logged in.

3. (Bool) Every \(i\)th element of lists of invalid usernames and invalid passwords are combined. Info should return false, as the account log in try is done with the invalid new password.

4. (Bool) Every \(i\)th element of lists of invalid usernames and invalid passwords are combined. Login should return true, as the account was logged out.

Test items  changeProperties, info, logout

Input specifications


2. (list) ["t93KCd4SA", "abcdgef2", "My birth date", "This is a very long string. This is allowed as we only require that the string is at least 8 characters long and contains only ASCII CHARACTERS", "09112002", "Voornaam", "SASXER54", "Amsterdam", "01234567"], list of valid passwords.

Output specifications

1. (Bool) Every \(i\)th element of lists of valid usernames and valid passwords are combined. changeProperties should return false, as the account is not logged in.

2. (Bool) Every \(i\)th element of lists of valid usernames and valid passwords are combined. changeProperties should return false, as the old and new password is not sent.

3. (Bool) Every \(i\)th element of lists of valid usernames and valid passwords are combined. changeProperties should return false, as the new password is not sent.

4. (Bool) Every \(i\)th element of lists of valid usernames and valid passwords are combined. changeProperties should return false, as it is not an AJAX request.
CHAPTER 3. TESTCASE SPECIFICATION

3.2.9 Keyword generation

This tests the module categorize.py on a unittesting level. To make testing possible some additional classes are imported. First the unittest-class from Django which basically adds the same functionality as the unittest-class of python. Further we need to prepare some data to be able to query, therefore the models are imported. To have some test data we import KroketApp.query.subject.subjectTestSet, which gives us a set of predefined test models, which we can use to test the keyword generation. All subject codes used in this document have a corresponding subject defined in subjectTestSet.

A new and empty database is created when these tests are run which means that before every test first some data is added on which the check runs. After a test is run, the test-database is cleared by the function tearDown (if needed). In setUp the database is filled with the data from the subjectTestSet.

UT83

Test items rankStringNLTK

Input specifications The following strings are ranked (in the given order) with rankStringNLTK:

1. "A noun will be a keyword" with an empty ranking.

2. "Parsing a new sentence should not clear the dictionary, but just append the keywords found into the dictionary" with the ranking of the previous string.

Output specifications Per input string the following assertions are made (the number identifies the input specification):

1. There are 2 keywords in the rankings, and these are 'noun' and 'keyword'.

2. There are at least 5 keywords in the ranking, and at least these are in the ranking: 'new','sentence';'dictionary'; 'keyword'; 'noun'.

Environmental needs None.

UT84

Test items rankEnglish

Input specifications The subjects corresponding to the following subject codes are ranked (in the given order) with rankEnglish:

1. OHV30

2. 2IT60
Kroket – UNIT TEST PLAN

3. 4MA00
4. 5EIA0
5. 8QA01
6. 2AS00

Output specifications Per subject we check if the subject code is a keyword; Per subject the function should relate the following keywords to the subject (the number identifies the input specification):

1. \{ “consumer”, “behavior”, “social”, “psychology”, “condition”, “boundary”, “handbook”, “cognitive”, “stateoftheart”, “humantechnology” \}
3. \{ “structure”, “properties”, “materials”, “development”, “science”, “static” \}
4. \{ “computation”, “aim”, “electrical”, “engineering”, “stack”, “program”, “processor” \}
5. \{ “image”, “analysis”, “pathology”, “project”, “question”, “image”, “software” \}
6. \{ “statistical”, “dataanalysis”, “aim”, “course”, “research”, “variation”, “hypothesis” \}

Environmental needs None.

UT85

Test items updateSubDB

Input specifications The input parameters to this function are a Subject and a dictionary with keywords. The following input parameters are tested:

4. Subject: 5EIA0; Dictionary keys: (“computation”, “aim”, “electrical”, “engineering”, “stack”, “program”, “processor”); All dictionary values are 10.
CHAPTER 3. TESTCASE SPECIFICATION

5. Subject: 8QA01; Dictionary keys: {“image”, “analysis”, “pathology”, “project”, “question”, “image”, “software”}; All dictionary values are 10.

6. Subject: 2AS00; Dictionary keys: {“statistical”, “dataanalysis”, “aim”, “course”, “research”, “variation”, “hypothesis”}; All dictionary values are 10.

**Output specifications**  Per subject we check if all dictionary keys are associated with the subject.

**Environmental needs**  None.

**UT86**

---

**Test items**  rankPacket

**Input specifications**  The input parameters are the SubjectPacket corresponding to the following target_group identifier: 6, 7.

**Output specifications**  Per SubjectPacket the function should relate the following keywords to the subject (the number identifies the input specification):


7. ⟨“analysis”, “society”, “image”, “analysis”, “pathology”, “project”, “question”, “image”, “software”⟩

**Environmental needs**  None.

### 3.2.10 Authentication of the user

This tests the module addData.py on a unit testing level. To make testing possible some additional classes are imported. First the unittest-class from Django which basically adds the same functionality as the unittest-class of python. Further we need to prepare some data so we need to be able to query, therefore the models are imported. addData and parser are two classes we want to test and categorize is also needed to fill the database with correct data (after a course is added it’s keywords need to be find). More detailed information can be found in DDD [2] chapter 7.10.

A new and empty database is created when these tests are run which means that before every test first some data is added on which the check runs. These explained as input. The output are all the different assertions. Before a test is run, the test-database is cleared by the function setUp. This means that for all test cases a database should be available which works properly.
Kroket – UNIT TEST PLAN

UT87

Test items clearDB

Input specifications Every table, concerning subject and packet information is filled with an item.

Output specifications To make sure the delete function works, first some items are added to the database. This is to make sure an already empty table does not always conclude the deleting went correctly. Therefore it is also important that after adding the data a check is done that there is indeed at least one item in the table.

1. The table CoursePlanning contains more than 0 objects.
2. The table SubjectFollowUpRelation contains more than 0 objects.
3. The table SubjectPriorKnowledgeRelation contains more than 0 objects.
4. The table KeywordSubjectRelation contains more than 0 objects.
5. The table KeywordPacketRelation contains more than 0 objects.
6. The table SubjectPacket contains more than 0 objects.
7. The table Keyword contains more than 0 objects.
8. The table Subject contains more than 0 objects.

After deleting it is only checked if there are no items left in the tables, if so everything is deleted.

1. The table CoursePlanning contains 0 objects.
2. The table SubjectFollowUpRelation contains 0 objects.
3. The table SubjectPriorKnowledgeRelation contains 0 objects.
4. The table KeywordSubjectRelation contains 0 objects.
5. The table KeywordPacketRelation contains 0 objects.
6. The table SubjectPacket contains 0 objects.
7. The table Keyword contains 0 objects.
8. The table Subject contains 0 objects.

Environmental needs None.

UT88
CHAPTER 3. TESTCASE SPECIFICATION

Test items  parseDoelgroep

Input specifications  parseDoelgroep() has been called to make sure all information from Doelgroep.csv are added.

Output specifications

1. The major with target group="1365" has name “Industrial Engineering & Management Science”.
2. The major with target group="1365" has type SubjectPacket.Major.
3. The major with target group="1375" has name “Software Science”.
4. The major with target group="1375" has type SubjectPacket.Major.
5. The major with target group="1414" has name “Statistics and research methodology”.
6. The major with target group="1414" has type SubjectPacket.COHERENT.

Environmental needs  None.

UT89

Test items  addCourse

Input specifications

1. The subject “1BV00” is added with difficulty “Basic”.
2. The subject “8WA00” is added with difficulty “”.
3. The non existing subject “NonEx” is added with difficulty “Basic”.

Output specifications

1. The subject with subjectcode “1BV00” has name “Heart and lung”.
2. The subject with subjectcode “1BV00” has difficulty “”.
3. The subject with subjectcode “1BV00” is worth 5 ECTS.
4. The subject with subjectcode “1BV00” has “N.A.” as content.
5. The subject with subjectcode “1BV00” has a certain string as study goal.
6. There is a subject with subjectcode “NonEx”.
Kroket – UNIT TEST PLAN

Environmental needs  None.

UT90

Test items  addCoursePlanning

Input specifications
  1. The course “oLCB4” is added with difficulty “”.
  2. The major “Industrial Engineering & Management Science” is added.
  3. The course “oSAB1” is added with difficulty “”.
  4. The major “Building and Planning” is added.

Output specifications
  1. The subject with subjectcode “oLCB4” is planned in year 1.
  2. The subject with subjectcode “oLCB4” is planned in timeslot “A”.
  3. The subject with subjectcode “oLCB4” is planned in quartile 3.
  4. The subject with subjectcode “oSAB1” scheduled 0 times.

Environmental needs  None.

UT91

Test items  getAdditionalInfo

Input specifications  The course “2IP35” is added with difficulty “”.

Output specifications
  1. The subject with subjectcode “2IP35” has url “http://www.is.win.tue.nl/2IP35”.
  2. The subject with subjectcode “2IP35” has department “Mathematics and Computer Science”.
  3. The subject with subjectcode “2IP35” has a certain string as study goal.

Environmental needs  None.

UT92
CHAPTER 3. TESTCASE SPECIFICATION

Test items  owisCourses

Input specifications  The course “2IP35” is added with difficulty “”.

Output specifications  Before OWIS is requested:

1. The subject with subjectcode “2IP35” has url “”.

After OWIS is requested for more information:

1. The subject with subjectcode “2IP35” has url “http://wwwis.win.tue.nl/2IP35”.

2. The subject with subjectcode “2IP35” has department “Mathematics and Computer Science”.

3. The subject with subjectcode “2IP35” has a certain string as study goal.

Environmental needs  None.

UT93

Test items  addSoftwareScienceMajor

Input specifications

1. The course “2IO90” is added with difficulty “”.

2. The major “Software Science” is added with target group 1375.

Output specifications

1. The subject with subjectcode “KROKo8” has name “Business information systems”.

2. The subject with subjectcode “2IP35” is planned in year 3 for the major with target group 1375.

3. The subject with subjectcode “2IP35” is planned in quartile 1 for the major with target group 1375.

4. The subject with subjectcode “2IP35” is planned in timeslot “?” for the major with target group 1375.

5. The subject with subjectcode “2IO90” is planned in the major with target group 1375.

Environmental needs  None.
Kroket – UNIT TEST PLAN

3.2.11 Testing adding of subject and packet information

This tests the module addData.py on a unit testing level. To make testing possible some additional classes are imported. First the unittest-class from Django which basically adds the same functionality as the unittest-class of Python. Further we need to prepare some data so we need to be able to query, therefore the models are imported. addData and parser are two classes we want to test and categorize is also needed to fill the database with correct data (after a course is added it’s keywords need to be find). More detailed information can be found in DDD [2] chapter 7.10.

A new and empty database is created when these tests are run which means that before every test first some data is added on which the check runs. These explained as input. The output are all the different assertions. Before a test is run, the test-database is cleared by the function setUp. This means that for all test cases a database should be available which works properly.

UT94

Test items  clearDB

Input specifications  Every table, concerning subject and packet information is filled with an item.

Output specifications  To make sure the delete function works, first some items are added to the database. This is to make sure an already empty table does not always conclude the deleting went correctly. Therefore it is also important that after adding the data a check is done that there is indeed at least one item in the table.

1. The table CoursePlanning contains more than 0 objects.
2. The table SubjectFollowUpRelation contains more than 0 objects.
3. The table SubjectPriorKnowledgeRelation contains more than 0 objects.
4. The table KeywordSubjectRelation contains more than 0 objects.
5. The table KeywordPacketRelation contains more than 0 objects.
6. The table SubjectPacket contains more than 0 objects.
7. The table Keyword contains more than 0 objects.
8. The table Subject contains more than 0 objects.

After deleting it is only checked if there are no items left in the tables, if so everything is deleted.

1. The table CoursePlanning contains 0 objects.
2. The table SubjectFollowUpRelation contains 0 objects.

3. The table SubjectPriorKnowledgeRelation contains 0 objects.

4. The table KeywordSubjectRelation contains 0 objects.

5. The table KeywordPacketRelation contains 0 objects.

6. The table SubjectPacket contains 0 objects.

7. The table Keyword contains 0 objects.

8. The table Subject contains 0 objects.

Environmental needs None.

UT95

Test items parseDoelgroep

Input specifications parseDoelgroep() has been called to make sure all information from Doelgroep.csv are added.

Output specifications

1. The major with target group="1365" has name “Industrial Engineering & Management Science”.

2. The major with target group="1365" has type SubjectPacket.Major.

3. The major with target group="1375" has name “Software Science”.

4. The major with target group="1375" has type SubjectPacket.Major.

5. The major with target group="1414" has name “Statistics and research methodology”.

6. The major with target group="1414" has type SubjectPacket.COHERENT.

Environmental needs None.

UT96

Test items addCourse
Kroket – UNIT TEST PLAN

Input specifications

1. The subject “1BVoo” is added with difficulty “Basic”.
2. The subject “8WAoo” is added with difficulty “”.
3. The non existing subject “NonEx” is added with difficulty “Basic”.

Output specifications

1. The subject with subjectcode “1BVoo” has name “Heart and lung”.
2. The subject with subjectcode “1BVoo” has difficulty “”.
3. The subject with subjectcode “1BVoo” is worth 5 ECTS.
4. The subject with subjectcode “1BVoo” has “N.A.” as content.
5. The subject with subjectcode “1BVoo” has a certain string as study goal.
6. There is a subject with subjectcode “NonEx”.

Environmental needs None.

UT97

Test items addCoursePlanning

Input specifications

1. The course “oLCB4” is added with difficulty “”.
2. The major “Industrial Engineering & Management Science” is added.
3. The course “oSAB1” is added with difficulty “”.
4. The major “Building and Planning” is added.

Output specifications

1. The subject with subjectcode “oLCB4” is planned in year 1.
2. The subject with subjectcode “oLCB4” is planned in timeslot “A”.
3. The subject with subjectcode “oLCB4” is planned in quartile 3.
4. The subject with subjectcode “oSAB1” is scheduled 0 times.
CHAPTER 3. TESTCASE SPECIFICATION

Environmental needs  None.

UT98

Test items  getAdditionalInfo

Input specifications  The course “2IP35” is added with difficulty “”.

Output specifications
1. The subject with subjectcode “2IP35” has url “http://wwwis.win.tue.nl/2IP35”.
2. The subject with subjectcode “2IP35” has department “Mathematics and Computer Science”.
3. The subject with subjectcode “2IP35” has a certain string as study goal.

Environmental needs  None.

UT99

Test items  owisCourses

Input specifications  The course “2IP35” is added with difficulty “”.

Output specifications  Before OWIS is requested:
1. The subject with subjectcode “2IP35” has url “”.

After OWIS is requested for more information:
1. The subject with subjectcode “2IP35” has url “http://wwwis.win.tue.nl/2IP35”.
2. The subject with subjectcode “2IP35” has department “Mathematics and Computer Science”.
3. The subject with subjectcode “2IP35” has a certain string as study goal.

Environmental needs  None.

UT100

Test items  addSoftwareScienceMajor
Input specifications

1. The course “IO90” is added with difficulty “”.
2. The major “Software Science” is added with target group 1375.

Output specifications

1. The subject with subjectcode “KROK08” has name “Business information systems”.
2. The subject with subjectcode “2IP35” is planned in year 3 for the major with target group 1375.
3. The subject with subjectcode “2IP35” is planned in quartile 1 for the major with target group 1375.
4. The subject with subjectcode “2IP35” is planned in timeslot “?” for the major with target group 1375.
5. The subject with subjectcode “2IO90” is planned in the major with target group 1375.

Environmental needs  None.

3.2.12 Requesting additional information of a subject

This tests the module query/subject/info.py:info on a unittesting level. To make testing possible some additional classes are imported. First the unittest-class from Django which basically adds the same functionality as the unittest-class of python. Furthermore we need to prepare some data to be able to query, therefore the models are imported.

A new and empty database is created when these tests are run. This means that before every test, some data is added to the database. This is done by requesting the test set from subjectTestSet and saving those models to the database. After a test is run, the test-database is cleared by the function tearDown.

UT101

Test items  info.info

Input specifications  The following requests are made to info.info:

1. A POST request with parameter code the string ‘‘DOESNOTEXIST’’ and a AJAX-header is sent with the request.
2. A POST request with no parameter code and a AJAX-header is sent with the request.
3. A GET request with parameter code as a string ‘‘0HV30’’ (this subject does exist) and a AJAX-header is sent with the request.
CHAPTER 3. TESTCASE SPECIFICATION

4. A GET request with parameter code as a string ‘‘0HV30’’ (this subject does exist) and no AJAX-header is sent with the request.

5. A GET request with parameter code as a string ‘‘DOESNOTEXIST’’ and a AJAX-header is sent with the request.

6. A GET request with no parameter code and a AJAX-header is sent with the request.

Output specifications Per input the following assertions are made (the number identifies the input specification):

1. The response should be a HTTPResponse with status code 403.
2. The response should be a HTTPResponse with status code 403.
3. The response should be a HTTPResponse with status code 403.
4. The response should be a HTTPResponse with status code 403.
5. The response should be a HTTPResponse with status code 403.
6. The response should be a HTTPResponse with status code 403.

Environmental needs None.

UT102

Test items info.info

Input specifications A brute force check is made on all subjects in the database. For each subject the subject code is posted to the info.info with AJAX-headers.

Output specifications For each brute force attempt the following properties are checked:

- the response should be a JSON-encoded dictionary.
- In the resulting dictionary the key ‘code’ should be associated with the subject code.
- In the resulting dictionary the key ‘name’ should be associated with the subject its name field.
- In the resulting dictionary the key ‘ects’ should be associated with the subject its ects field.
Kroket – UNIT TEST PLAN

- In the resulting dictionary the key ‘difficulty’ should be associated with the subject its niveau field.
- In the resulting dictionary the key ‘lastYear’ should be associated with the subject its last_year field.
- In the resulting dictionary the key ‘remarks’ should be associated with the subject its remarks field.
- In the resulting dictionary the key ‘studyGoal’ should be associated with the subject its study_goal field.
- In the resulting dictionary the key ‘content’ should be associated with the subject its content field.
- In the resulting dictionary the key ‘weeklyContent’ should be associated with the subject its weekly_content field.
- In the resulting dictionary the key ‘videoUrl’ should be associated with the subject its video_url field.
- In the resulting dictionary the key ‘videoMaterialDescription’ should be associated with the subject its video_mat_desc field.
- In the resulting dictionary the key ‘educationType’ should be associated with the subject its education_type field.
- In the resulting dictionary the key ‘examinationType’ should be associated with the subject its examination_type field.
- In the resulting dictionary the key ‘department’ should be associated with the subject its department field.
- In the resulting dictionary the key ‘subdepartment’ should be associated with the subject its subdepartment field.
- In the resulting dictionary the key ‘planning’ should be equal to a list with dictionaries, without respecting to the order. For each courseplanning associated with the subject the following dictionary should be in the list:
  - A key ‘year’ with as value the field year of the course planning;
  - A key ‘quartile’ with as value the field year of the course planning;
  - A key ‘timeSlot’ with as value the field timeSlot of the course planning;
  - A key ‘targetGroup’ with as value the field target_group identifier to which the subject is associated;
  - A key ‘packet’ with as value the name of the packet for which the subject is scheduled;
  - A key ‘packetType’ with as value the type of the packet for which the subject is scheduled;
- A key ‘mandatory’ with as value if this subject is mandatory to the corresponding packet.

- In the resulting dictionary the key ‘priorKnowledge’ should associated with one of the predefined strings (in which all subjects should be enumerated, which are related to the subject with SubjectPriorKnowledgeRelation, using the separator ‘;’, we don’t have any restrictions on the order in this string).

- In the resulting dictionary the key ‘followUp’ should associated with one of the predefined strings (in which all subjects should be enumerated, which are related to the subject with SubjectPriorKnowledgeRelation, using the separator ‘;’, we don’t have any restrictions on the order in this string)

**Environmental needs** None.

### 3.2.13 Converting nested lists of subject codes to nested list of standard formats

This tests the module query/subject/info.py:convert on a unittesting level. This function converts an arbitrary deeply nested list of subject code to an nested list (with the same structure) of subjects in standard format. To make testing possible some additional classes are imported. First the unittest-class from Django which basically adds the same functionality as the unittest-class of python. Furthermore we need to prepare some data to be able to query, therefore the models are imported.

A new and empty database is created when these tests are run. This means that before every test, some data is added to the database. This is done by requesting the test set from subjectTestSet and saving those models to the database. After a test is run, the test-database is cleared by the function tearDown.

**UT103**

**Test items** info.convert

**Input specifications** The following requests are made to info.info:

1. A POST request with no parameter subjects and a AJAX-header is sent with the request.

2. A GET request with as parameter subjects the JSON list: ‘[‘0HV30’]’ and a AJAX-header is sent with the request.

3. A GET request with as parameter subjects the JSON list: ‘[‘0HV30’]’ and no AJAX-header is sent with the request.

4. A GET request with as parameter subjects the JSON list: ‘[‘0HV30’]’ and a AJAX-header is sent with the request.
Kroket – UNIT TEST PLAN

5. A POST request with as parameter subjects the JSON list: ['0HV30'] and no AJAX-header is sent with the request.

6. A GET request with no parameter subjects and a AJAX-header is sent with the request.

**Output specifications**  Per input the following assertions are made (the number identifies the input specification):

1. The response should be a HTTPResponse with status code 403.
2. The response should be a HTTPResponse with status code 403.
3. The response should be a HTTPResponse with status code 403.
4. The response should be a HTTPResponse with status code 403.
5. The response should be a HTTPResponse with status code 403.
6. The response should be a HTTPResponse with status code 403.

**Environmental needs**  None.

**UT104**

**Test items**  info.info

**Input specifications**  A couple of request are made with different subjects parameter. These are JSON-encoded lists defined as follows:

1. [[[['0HV30', '0HV30'], '0HV30'], '2IT60', ['4MA00'], [[[['5EIA0']]]]]
2. ['2AS00', '5EIA0', '4MA00', '8QA01', 'DOESNOTEXISTS']
3. ['NOTEXISTANT']
4. [['0HV30', '4MA00'], ['5EIA0', '2AS00'], [], ['8QA01']]

**Output specifications**  For all lists we check if the response is in JSON format. After parsing the JSON-encoded response, the response should be equal to (the number identifies the input specification, standardFormat is the standardFormat representation of a subject):

1. [[[standardFormat('0HV30'), standardFormat('0HV30')], standardFormat('0HV30')], standardFormat('2IT60'), [standardFormat('4MA00')], [[[[standardFormat('5EIA0')]]]]
2. [standardFormat('2AS00'), standardFormat('5EIA0'), standardFormat('4MA00'), standardFormat('8QA01')]
3. []
CHAPTER 3. TESTCASE SPECIFICATION

4. [[standardFormat('OHV30'), standardFormat('4MA00')], [standardFormat('5EIA0'), standardFormat('2AS00')], [], [standardFormat('8QA01')]]

Environmental needs  None.

3.2.14 Searching on subjects

This tests the module query/subject/search.py on a unittesting level. To make testing possible some additional classes are imported. First the unittest-class from Django which basically adds the same functionality as the unittest-class of python. Furthermore we need to prepare some data to be able to query, therefore the models are imported.

A new and empty database is created when these tests are run. This means that before every test, some data is added to the database. This is done by requesting the test set from subjectTestSet and saving those models to the database. After a test is run, the test-database is cleared by the function tearDown.

UT105

Test items  search.search

Input specifications  The following requests are made to info.info:

1. A GET request with no parameters and an AJAX-header.
2. A GET request with no parameters and no AJAX-header.
3. A POST request with a parameter ‘range’ with value ‘invalid’ and an AJAX-header.
4. A POST request with a parameter ‘range’ with value ‘1000-5132a’ and an AJAX-header.

Output specifications  Per input the following assertions are made (the number identifies the input specification):

1. The response should be a HTTPResponse with status code 403.
2. The response should be a HTTPResponse with status code 403.
3. The response should be a HTTPResponse with status code 403.
4. The response should be a HTTPResponse with status code 403.

Environmental needs  None.

UT106
Kroket – UNIT TEST PLAN

Test items  search.search

Input specifications  Will search on all subject codes of all subjects in the database, by giving as searchTerm the subject code of a certain subject.

Output specifications  For each brute force attempt the following properties are checked:

- The response should be a JSON-encoded dictionary, list with only element.
- The first dictionary of resulting list should have the keys ‘code’, ‘name’, ‘ects’, ‘difficulty’, ‘lastYear’, ‘planning’, ‘priorKnowledge’.
- In the first dictionary of resulting list the key ‘code’ should be associated with the subject code.
- In the first dictionary of resulting list the key ‘name’ should be associated with the subject its name field.
- In the first dictionary of resulting list the key ‘ects’ should be associated with the subject its ects field.
- In the first dictionary of resulting list the key ‘difficulty’ should be associated with the subject its niveau field.
- In the first dictionary of resulting list the key ‘lastYear’ should be associated with the subject its last_year field.
- In the first dictionary of resulting list the key ‘planning’ should be equal to a list with dictionaries, without respecting to the order. For each courseplanning associated with the subject the following dictionary should be in the list:
  - A key ‘year’ with as value the field year of the course planning;
  - A key ‘quartile’ with as value the field year of the course planning;
  - A key ‘timeSlot’ with as value the field timeSlot of the course planning;
  - A key ‘target_group’ with as value the field target_group identifier to which the subject is associated;
  - A key ‘packet’ with as value the name of the packet for which the subject is scheduled;
  - A key ‘packetType’ with as value the type of the packet for which the subject is scheduled;
  - A key ‘mandatory’ with as value if this subject is mandatory to the corresponding packet.
- In the resulting dictionary the key ‘priorKnowledge’ should associated with one of the predefined strings (in which all subjects should be enumerated, which are related to the subject with SubjectPriorKnowledgeRelation, using the separator ‘;’, we don’t have any restrictions on the order in this string).
CHAPTER 3. TESTCASE SPECIFICATION

Environmental needs  None.

UT107

Test items  search.search

Input specifications  Will search on the following parameters:

- ‘range’ as value ‘100-10’
- ‘range’ as value ‘-20-3’
- ‘range’ as value ‘1000000-1000010’
- ‘range’ as value ‘0-5’
- ‘range’ as value ‘1-6’

Output specifications  Per input the following assertions are made (the number identifies the input specification):

1. The response should have a status code 403-Forbidden;
2. The response should have a status code 403-Forbidden;
3. The response should have a status code 200-OK;
4. The response should have 6 elements;
5. The last 5 elements of the previous response should be equal to the first 5 elements of the current response. They should also be in the same order;

Environmental needs  None.

UT108

Test items  search.search

Input specifications  Will search on the following parameters:

- ‘quartile’ as value ‘1’
- ‘quartile’ as value ‘2’
Kroket – UNIT TEST PLAN

Output specifications  Per input the following assertions are made (the number identifies the input specification):

1. The response should be a JSON list of subjects and the subject codes should be equal to ‘8QA01’;

2. The response should be a JSON list of subjects and the subject codes should be equal to ‘0HV30’, ‘4MA00’, ‘8QA01’;

Environmental needs  None.

UT109

Test items  search.search

Input specifications  Will search on the following parameters:

- ‘timeSlot’ as value ‘A’
- ‘timeSlot’ as value ‘C’

Output specifications  Per input the following assertions are made (the number identifies the input specification):

1. The response should be a JSON list of subjects and the subject codes should be equal to ‘0HV30’, ‘4MA00’;

2. The response should be a JSON list of subjects and the subject codes should be equal to ‘8QA01’;

Environmental needs  None.

UT110

Test items  search.search

Input specifications  Will search on the following parameters:

- ‘year’ as value ‘2’
- ‘year’ as value ‘3’
Output specifications  Per input the following assertions are made (the number identifies the input specification):

1. The response should be a JSON list of subjects and the subject codes should be equal to ‘5EIA0’;
2. The response should be a JSON list of subjects and the subject codes should be equal to ‘4MA00’, ‘8QA01’;

Environmental needs  None.

UT111

Test items  search.search

Input specifications  Will search on the following parameters:

- ‘difficulty’ as value ‘Basic’
- ‘difficulty’ as value ‘Intermediate’

Output specifications  Per input the following assertions are made (the number identifies the input specification):

1. The response should be a JSON list of subjects and the subject codes should be equal to ‘0HV30’;
2. The response should be a JSON list of subjects and the subject codes should be equal to empty list;

Environmental needs  None.

UT112

Test items  search.search

Input specifications  Will search on the following parameters:

- ‘target_group’ as value ‘6’

Output specifications  Per input the following assertions are made (the number identifies the input specification):

1. The response should be a JSON list of subjects and the subject codes should be equal to ‘0HV30’;
Kroket – UNIT TEST PLAN

Environmental needs  None.

3.2.15 Checking whether the schedule is valid

This tests the module query/schedule/validate.py on a unit testing level. To make testing possible some additional classes are imported. First the unittest-class from Django which basically adds the same functionality as the unittest-class of python. Further we need to prepare some data to be able to query, therefore the models are imported.

A new and empty database is created when these tests are run. This means that before every test, some data is added to the database. After a test is run, the test-database is cleared by the function tearDown, which in turn calls clearData() (if needed).

In setUp the database is filled with the data from the loadData(). These function define some instances of SubjectPackages, Subject, CoursePlannings, etc.

UT113

Test items  validate.validate

Input specifications  For each major m in the database (these are created by loadData), and the corresponding set of mandatory subjects S, the following is tested:

1. A POST request with parameter schedule as a JSON list of subject codes of S, and major m, to validate.validate. But no AJAX-header is sent with the request.

2. A POST request with major m to validate.validate and a AJAX-header is sent with the request. But no schedule parameter is sent.

3. A POST request with parameter schedule as a JSON list of subject codes of S and a AJAX-header is sent with the request. But no major parameter is sent.

4. A GET request with parameter schedule as a JSON list of subject codes of S, major m, to validate.validate and an AJAX-header is sent with the request.

5. A GET request with no parameters, and no AJAX-header.

Output specifications  Per input string the following assertions are made (the number identifies the input specification):

1. The response should be a HTTPResponse with status code 403.

2. The response should be a HTTPResponse with status code 403.

3. The response should be a HTTPResponse with status code 403.

4. The response should be a HTTPResponse with status code 403.

5. The response should be a HTTPResponse with status code 403.
CHAPTER 3. TESTCASE SPECIFICATION

Environmental needs  None.

UT114

Test items  validate.validate

Input specifications  For a certain major $m$, we brute-force on combinations of coherent packages of length 2 to 3 and USE packages of length 1 to 2 with the restriction that the packages have no subject in common. The major parameter will be the target group identifier of $m$ and the schedule parameter will be the set of all mandatory subjects belonging to the major, and the set of all subjects belonging to the USE/coherent packages. Furthermore, all headers and methods are as required.

Output specifications  For each brute force attempt the follow output should be given:

- The response should have status code 200;
- The response should be in JSON format and in the format specified by the function specification;
- The resulting dictionary should indicate a PASS on the USE requirements;
- The resulting dictionary should indicate that the USE packages in the schedule are equal to the chosen subset of USE packages;
- The resulting dictionary should indicate a PASS on the Coherent requirements;
- The resulting dictionary should indicate that the Coherent packages in the schedule are equal to the chosen subset of Coherent packages;
- The resulting dictionary should indicate a PASS on the major requirements;
- The resulting dictionary should indicate that there are no missing major subjects;
- The resulting dictionary should indicate a PASS on the overlap requirements;
- The resulting dictionary should indicate that there are no packages overlapping;

Environmental needs  None.

UT115

Test items  validate.validate
Kroket – UNIT TEST PLAN

Input specifications  For a certain major m, we brute-force on combinations of coherent packages of length 2 to 3 and USE packages of length 1 to 2. With the restriction that some package have subjects in common. The major parameter will be the target group identifier of m and the schedule parameter will be the set of all mandatory subjects belonging to the major, and the set of all subjects belonging to the USE/coherent packages. Furthermore, all headers and methods are as required.

Output specifications  For each brute force attempt the follow output should be given:

- The response should have status code 200;
- The response should be in JSON format and in the format specified by the function specification;
- The resulting dictionary should indicate a PASS on the USE requirements;
- The resulting dictionary should indicate that the USE packages in the schedule are equal to the chosen subset of USE packages;
- The resulting dictionary should indicate a PASS on the Coherent requirements;
- The resulting dictionary should indicate that the Coherent packages in the schedule are equal to the chosen subset of Coherent packages;
- The resulting dictionary should indicate a PASS on the major requirements;
- The resulting dictionary should indicate that there are no missing major subjects;
- The resulting dictionary should indicate a FAIL on the overlap requirements;
- The resulting dictionary should indicate that the chosen packages which have subjects in common are overlapping packages, and no other packages are overlapping packages.

Environmental needs  None.

UT116

Test items  validate.validate

Input specifications  For a certain major m, we brute-force on combinations of coherent packages of length 0 to 1 and USE packages of length 1 to 2. With the restriction that the packages have no subject in common. The major parameter will be the target group identifier of m and the schedule parameter will be the set of all mandatory subjects belonging to the major, and the set of all subjects belonging to the USE/coherent packages. Furthermore, all headers and methods are as required.
CHAPTER 3. TESTCASE SPECIFICATION

Output specifications  For each brute force attempt the follow output should be given:

- The response should have status code 200;
- The response should be in JSON format and in the format specified by the function specification;
- The resulting dictionary should indicate a PASS on the USE requirements;
- The resulting dictionary should indicate that the USE packages in the schedule are equal to the chosen subset of USE packages;
- The resulting dictionary should indicate a FAIL on the Coherent requirements;
- The resulting dictionary should indicate that the Coherent packages in the schedule are equal to the chosen subset of Coherent packages;
- The resulting dictionary should indicate a PASS on the major requirements;
- The resulting dictionary should indicate that there are no missing major subjects;
- The resulting dictionary should indicate a PASS on the overlap requirements;
- The resulting dictionary should indicate that there are no packages overlapping;

Environmental needs  None.

UT117

Test items  validate.validate

Input specifications  For a certain major m, we brute-force on combinations of coherent packages of length 2 to 3 and no USE packages. With the restriction that the packages have no subject in common. The major parameter will be the target group identifier of m and the schedule parameter will be the set of all mandatory subjects belonging to the major, and the set of all subjects belonging to the USE/coherent packages. Furthermore, all headers and methods are as required.

Output specifications  For each brute force attempt the follow output should be given:

- The response should have status code 200;
- The response should be in JSON format and in the format specified by the function specification;
- The resulting dictionary should indicate a FAIL on the USE requirements;
- The resulting dictionary should indicate that there are no USE packages
Kroket – UNIT TEST PLAN

• The resulting dictionary should indicate a PASS on the Coherent requirements;
• The resulting dictionary should indicate that the Coherent packages in the schedule are equal to the chosen subset of Coherent packages;
• The resulting dictionary should indicate a PASS on the major requirements;
• The resulting dictionary should indicate that there are no missing major subjects;
• The resulting dictionary should indicate a PASS on the overlap requirements;
• The resulting dictionary should indicate that there are no packages overlapping;

Environmental needs  None.

UT118

Test items  validate.validate

Input specifications  For a certain major m, we brute-force on combinations of coherent packages of length 2 to 3 and USE packages of length 1 to 2. With the restriction that the packages have no subject in common. We then brute-force on all subsets of the major courses, with the extra requirement that all but 3 major courses are chosen in the schedule, furthermore we include in the schedule parameter the set of all subjects belonging to the USE/coherent packages. Furthermore, all headers and methods are as required.

Output specifications  For each brute force attempt the follow output should be given:
• The response should have status code 200;
• The response should be in JSON format and in the format specified by the function specification;
• The resulting dictionary should indicate a PASS on the USE requirements;
• The resulting dictionary should indicate that the USE packages in the schedule are equal to the chosen subset of USE packages;
• The resulting dictionary should indicate a PASS on the Coherent requirements;
• The resulting dictionary should indicate that the Coherent packages in the schedule are equal to the chosen subset of Coherent packages;
• The resulting dictionary should indicate a FAIL on the major requirements;
• The resulting dictionary should indicate that the missing major subjects are equal to the all major subjects minus the chosen major subjects;
• The resulting dictionary should indicate a PASS on the overlap requirements;
• The resulting dictionary should indicate that there are no packages overlapping;
Environmental needs  None.

UT119

Test items  validate.checkOverlap

Input specifications  We brute force on all non overlapping packages.

Output specifications  The resulting dictionary should have a key status with value 'PASS' and a key 'overlap' with as value an empty list.

Environmental needs  None.

UT120

Test items  validate.checkOverlap

Input specifications  We brute force on all combinations of packages where at least two packages that have two subjects in common.

Output specifications  The resulting dictionary should have a key status with value 'FAIL' and a key 'overlap' with as value the chosen overlapping packages.

Environmental needs  None.

UT121

Test items  validate.checkOverlap

Input specifications  We brute force on all subsets of coherent packages, with as extra requirement that we have chosen at least 2 packages in common.

Output specifications  The resulting dictionary should have a key status with value 'PASS' and a key 'packages' with as value the chosen packages.

Environmental needs  None.

UT122

Test items  validate.checkCoherent
Kroket – UNIT TEST PLAN

**Input specifications**  We brute force on all subsets of (coherent and USE packages, with as extra requirement that we have chosen at most 1 coherent package. Then the input is defined as the set of all subjects belong to one of the chosen packages.

**Output specifications**  The resulting dictionary should have a key status with value 'FAIL' and on key 'packages' with a list corresponding to the chosen subset of coherent packages.

**Environmental needs**  None.

---

**UT123**

**Test items**  validate.checkUSE

**Input specifications**  We brute force on all subsets of USE packages, with as extra requirement that we have chosen at least 1 subsets.

**Output specifications**  The resulting dictionary should have a key status with value 'PASS' and a key 'packages' with as value the chosen packages.

**Environmental needs**  None.

---

**UT124**

**Test items**  validate.checkUSE

**Input specifications**  We brute force on all subsets of (coherent and USE packages, with as extra requirement that we have chosen at most 1 USE package. Then the input is defined as the set of all subjects belong to one of the chosen packages.

**Output specifications**  The resulting dictionary should have a key status with value 'FAIL' and on key 'packages' with a list corresponding to the chosen subset of USE packages.

**Environmental needs**  None.
Chapter 4

Test procedures

The author responsible for the class is also responsible for the testcases. Furthermore, a
test procedure needs to be created for each component. A test procedure should execute
all testcases for the classes in the module. The test procedure may also execute additional
testcases: these testcases can combine several testcases in order to test the interaction be-
tween classes in a module. Only if every testcase in the test procedure is successful, the
component can be merged with other components.

Note that a similar abstraction can be used to create system-wide testcases. Each test
procedure contains the following information:

- Unique test procedure identifier.
- Purpose: the purpose of the test procedure and the test cases the procedure executes.
- Procedure steps: describes how to log, set up, start, proceed, measure, shut down,
  restart, stop the test.

Detailed test procedure information for each test procedure will follow.

4.1 Javascript testing

Test procedure identifier  UTprocedure1

Purpose  These tests are used to test all javascript, which is the client-side part of the
website. This procedures will execute all javascript tests.

Procedure steps  All javascript functions will be tested using qUnit.

The map test should be placed inside the map called media, such that the tests can use
the html and the javascript that is placed inside the media map. If you want to change this,
you will have to change some urls inside test\index.html. To make sure the right part of
the HTML from the webpage is added to the test page, you need to mark the start of the
HTML that should be added with the line <!--TEST from here--> and end it again with
the line <!--END TEST--> . You can use this multiple times within the code, make sure not
to include any scripts that should not be executed for the tests. We have already done this, do not remove the tags when editing the HTML page.

Now the tests can be run by adding \test to the website’s url. (At this moment, the address to go is http://kroket.win.tue.nl:8085/media/test.)

4.2 Server side testing

To run tests on the server it is mandatory that the following packages are installed on the server:

- Python nose, can be installed by the command: pip install nose;
- Django-nose, can be installed by the command: pip install django_nose;
- Django nose coverage, can be installed by the command: pip install coverage.

Test procedure identifier  UTprocedure2

Purpose  These tests are used to test all python scripts, which is the server-side part of the website.

Procedure steps  The server side tests can only be run with the terminal of the server. To execute all server sided tests, one must be in the directory where manage.py resides in. This is the main directory of the Django project. Once there, execute shell script runtest by entering:

    . runtest

This shell script generates an output on the terminal, which is the same as nosetests.xml in the same working directory. Additionally, it creates an HTML page which also shows the coverage. This file can be found in tests_results/index.html from the current working directory.

Test procedure identifier  UTprodecure3

Purpose  Run a specific test case for the server side python scripts.

Procedure steps  Open the terminal in the server and set the current directory to the django directory. Then execute the following command:

    python manage.py test {project name}.({filepath to file of unittests}):{class name of unittest}.{method name of test}

For example:

single test: python manage.py test
    KroketApp.tests.query.user.testAuthentication:LoginTestCase.test_ValidLogin
CHAPTER 4. TEST PROCEDURES

**single unittest**: python manage.py test
   KroketApp.tests.query.user.testAuthentication:LoginTestCase

**unittests in one file**: python manage.py test
   KroketApp.tests.query.user.testAuthentication

**unittests in one directory (recursive)**: python manage.py test KroketApp.tests.query
Chapter 5

Test reports

A test report should be generated after each execution of a test procedure. It contains the following items:

- Unique test report identifier.
- Description, the items tested.
- The result of the unittest. In case of failure, the problem is also described.

Any bugs must be documented in Trac, see SCMP [3] for details.

5.1 Javascript tests

The results of the test can be found by visiting the subfolder test/ of the website. For example, when the url of the application is http://kroket.win.tue.nl:8085/media/, then test results can be viewed by visiting http://kroket.win.tue.nl:8085/media/test/.

In the table on the following pages the results of tests are stated.
<table>
<thead>
<tr>
<th>Class name</th>
<th>Name</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base tests</td>
<td>Some basic tests.</td>
<td>passed all</td>
</tr>
<tr>
<td>1</td>
<td>True should pass.</td>
<td>pass</td>
</tr>
<tr>
<td>2</td>
<td>Simple arithmetic: (1 + 1 = 2).</td>
<td>pass</td>
</tr>
<tr>
<td>3</td>
<td>Simple arithmetic: (1 + 1 \neq 3).</td>
<td>pass</td>
</tr>
<tr>
<td>Loading tests</td>
<td>Test asking all majors from a simulated (mocked) server.</td>
<td>passed all</td>
</tr>
<tr>
<td>1</td>
<td>There were two majors in the simulation.</td>
<td>pass</td>
</tr>
<tr>
<td>2</td>
<td>Name of the first major.</td>
<td>pass</td>
</tr>
<tr>
<td>3</td>
<td>Name of the second major.</td>
<td>pass</td>
</tr>
<tr>
<td>4</td>
<td>First major should be in the #major list.</td>
<td>pass</td>
</tr>
<tr>
<td>Loading tests</td>
<td>Test asking all majors from the actual server.</td>
<td>passed all</td>
</tr>
<tr>
<td>1</td>
<td>Major ... should be in the data.</td>
<td>pass</td>
</tr>
<tr>
<td>2</td>
<td>Major A should be in the data.</td>
<td>pass</td>
</tr>
<tr>
<td>3</td>
<td>Major B should be in the data.</td>
<td>pass</td>
</tr>
<tr>
<td>4</td>
<td>Major C should be in the data.</td>
<td>pass</td>
</tr>
<tr>
<td>5</td>
<td>Major A should be in the #major dropdown list.</td>
<td>pass</td>
</tr>
<tr>
<td>6</td>
<td>Major B should be in the #major dropdown list.</td>
<td>pass</td>
</tr>
<tr>
<td>7</td>
<td>Major C should be in the #major dropdown list.</td>
<td>pass</td>
</tr>
<tr>
<td>8</td>
<td>Only strings should be returned, no other weird values.</td>
<td>pass</td>
</tr>
<tr>
<td>Loading tests</td>
<td>Building the table.</td>
<td>passed all</td>
</tr>
<tr>
<td>1</td>
<td>Building a table with 4 years should give a table with 4 rows. Number of rows was 4.</td>
<td>pass</td>
</tr>
<tr>
<td>2</td>
<td>Each table cell should have 1 .addCourse element inside.</td>
<td>pass</td>
</tr>
<tr>
<td>Authentication tests</td>
<td>Test isValidUsername().</td>
<td>passed all</td>
</tr>
<tr>
<td>1</td>
<td>Should be considered valid: ‘SomeName’.</td>
<td>pass</td>
</tr>
<tr>
<td>2</td>
<td>Should not be considered valid (spaces): ‘Some Name’.</td>
<td>pass</td>
</tr>
<tr>
<td>3</td>
<td>Should be considered valid: ‘SomeName123’.</td>
<td>pass</td>
</tr>
<tr>
<td>4</td>
<td>Should be considered valid: ‘Some-Name’.</td>
<td>pass</td>
</tr>
<tr>
<td>5</td>
<td>Should not be considered valid (strange characters): ‘SomeName*’.</td>
<td>pass</td>
</tr>
<tr>
<td>6</td>
<td>Should be considered valid: ‘Names’.</td>
<td>pass</td>
</tr>
<tr>
<td>Test Case</td>
<td>Description</td>
<td>Result</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
<td>--------</td>
</tr>
<tr>
<td>7</td>
<td>Should not be considered valid (too short): ‘Name’.</td>
<td>pass</td>
</tr>
<tr>
<td>8</td>
<td>Should be considered valid: ‘ThisNameIsReallyLong’.</td>
<td>pass</td>
</tr>
<tr>
<td>9</td>
<td>Should not be considered valid (too long): ‘ThisReallyIsALongName’.</td>
<td>pass</td>
</tr>
<tr>
<td>Authentication tests</td>
<td>Test isValidPassword().</td>
<td>passed all</td>
</tr>
<tr>
<td>1</td>
<td>Should be considered valid: ‘password’.</td>
<td>pass</td>
</tr>
<tr>
<td>2</td>
<td>Should not be considered valid (too short): ‘password’.</td>
<td>pass</td>
</tr>
<tr>
<td>3</td>
<td>Should be considered valid: ‘Thisisaverylongpasswordthatisreallyreallylong’.</td>
<td>pass</td>
</tr>
<tr>
<td>4</td>
<td>Should not be considered valid (extended ASCII range): ‘hééééééé’.</td>
<td>pass</td>
</tr>
<tr>
<td>5</td>
<td>Should not be considered valid (ASCII control characters): ‘containsatab’.</td>
<td>pass</td>
</tr>
<tr>
<td>Validator tests</td>
<td>Test checkMajor().</td>
<td>passed all</td>
</tr>
<tr>
<td>1</td>
<td>When passing the major check, #validator-major should have class .success.</td>
<td>pass</td>
</tr>
<tr>
<td>2</td>
<td>When failing the major check, #validator-major should have class .error.</td>
<td>pass</td>
</tr>
<tr>
<td>Validator tests</td>
<td>Test checkUse().</td>
<td>passed all</td>
</tr>
<tr>
<td>1</td>
<td>When passing the USE check, #validator-use should have class .success.</td>
<td>pass</td>
</tr>
<tr>
<td>2</td>
<td>When failing the USE check, #validator-use should have class .error.</td>
<td>pass</td>
</tr>
<tr>
<td>Validator tests</td>
<td>Test checkCoherent().</td>
<td>passed all</td>
</tr>
<tr>
<td>1</td>
<td>When passing the coherent check, #validator-elective should have class .success.</td>
<td>pass</td>
</tr>
<tr>
<td>2</td>
<td>When failing the coherent check, #validator-elective should have class .error.</td>
<td>pass</td>
</tr>
<tr>
<td>Validator tests</td>
<td>Test countEcts().</td>
<td>passed all</td>
</tr>
<tr>
<td>1</td>
<td>Drie vakken met elk 5 ECTS zou te weinig moeten zijn, slechts 15 ECTS.</td>
<td>pass</td>
</tr>
<tr>
<td>2</td>
<td>Één vak met 180 ECTS, dat is raar maar moet 180 ECTS zijn.</td>
<td>pass</td>
</tr>
<tr>
<td>3</td>
<td>Twee vakken met 90 ECTS zou goed moeten zijn, namelijk 180 ECTS.</td>
<td>pass</td>
</tr>
<tr>
<td>4</td>
<td>Twee vakken met 100 ECTS is 200 ECTS.</td>
<td>pass</td>
</tr>
<tr>
<td>Validator tests</td>
<td>Test checkOverlap()</td>
<td>passed all</td>
</tr>
<tr>
<td>-----------------</td>
<td>---------------------</td>
<td>------------</td>
</tr>
<tr>
<td>1</td>
<td>When passing the overlap check, #validator-overlap should not be visible.</td>
<td>pass</td>
</tr>
<tr>
<td>2</td>
<td>When failing the overlap check, #validator-overlap should be visible.</td>
<td>pass</td>
</tr>
<tr>
<td>Utils tests</td>
<td>Test packageTypeToNull: Get the package type from an int.</td>
<td>passed all</td>
</tr>
<tr>
<td>1</td>
<td>A type 1 package is a USE package.</td>
<td>pass</td>
</tr>
<tr>
<td>2</td>
<td>A type 3 package is an Elective package.</td>
<td>pass</td>
</tr>
<tr>
<td>3</td>
<td>A type 2 package is a major.</td>
<td>pass</td>
</tr>
<tr>
<td>Utils tests</td>
<td>Test setScheduleLoading</td>
<td>passed all</td>
</tr>
<tr>
<td>1</td>
<td>showDisabler should set the loading screen visible.</td>
<td>pass</td>
</tr>
<tr>
<td>2</td>
<td>showDisabler with text “test” should set the text to “test”.</td>
<td>pass</td>
</tr>
<tr>
<td>3</td>
<td>hideDisabler should hide the loading screen.</td>
<td>pass</td>
</tr>
<tr>
<td>Utils tests</td>
<td>Test setModified</td>
<td>passed all</td>
</tr>
<tr>
<td>1</td>
<td>setModified(true) should change modifiedSchedule to true.</td>
<td>pass</td>
</tr>
<tr>
<td>2</td>
<td>setModified(false) should change modifiedSchedule to false.</td>
<td>pass</td>
</tr>
<tr>
<td>Utils tests</td>
<td>Test disableCourses</td>
<td>passed all</td>
</tr>
<tr>
<td>1</td>
<td>The course should not have been disabled.</td>
<td>pass</td>
</tr>
<tr>
<td>2</td>
<td>The course should have been disabled, it is in the schedule and in the results.</td>
<td>pass</td>
</tr>
<tr>
<td>3</td>
<td>The course should have been disabled, it is in the schedule and in the results.</td>
<td>pass</td>
</tr>
<tr>
<td>Utils tests</td>
<td>Test filterTimeSlot: change “,” into the word “and”, for display of a timeslot in the GUI.</td>
<td>passed all</td>
</tr>
<tr>
<td>1</td>
<td>A timeslot should not be changed if it doesn’t contain a semicolon.</td>
<td>pass</td>
</tr>
<tr>
<td>2</td>
<td>Timeslot A:B should return A and B.</td>
<td>pass</td>
</tr>
<tr>
<td>Utils tests</td>
<td>Test getSchedule: should return array of courses currently in the schedule.</td>
<td>passed all</td>
</tr>
<tr>
<td>1</td>
<td>The schedule is empty, so getschedule should give empty arrays.</td>
<td>pass</td>
</tr>
<tr>
<td>2</td>
<td>one course in the schedule.</td>
<td>pass</td>
</tr>
<tr>
<td>Test</td>
<td>Description</td>
<td>Result</td>
</tr>
<tr>
<td>------</td>
<td>-----------------------------------------------------------------------------</td>
<td>--------</td>
</tr>
<tr>
<td></td>
<td>2 courses in each cell except for year 2 Q 3.</td>
<td>pass</td>
</tr>
<tr>
<td></td>
<td><strong>Utils tests</strong> Test setSubjectInfo</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Setting the subject info with content “This is content” should show this in the subject information.</td>
<td>pass</td>
</tr>
<tr>
<td>2</td>
<td>Setting the evaluation to “8.2” should show this in the evaluation tab in subject info.</td>
<td>pass</td>
</tr>
<tr>
<td>3</td>
<td>Setting the subject info with studygoal “Some studygoal” should show this in the subject information.</td>
<td>pass</td>
</tr>
<tr>
<td></td>
<td><strong>Utils tests</strong> Test highlight: highlight available timeslots</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1 cell should be highlighted</td>
<td>pass</td>
</tr>
<tr>
<td>2</td>
<td>The cell in year 2012, quartile 3 should be highlighted.</td>
<td>pass</td>
</tr>
<tr>
<td></td>
<td><strong>Utils tests</strong> Test removeHighlight</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>All highlights should be removed.</td>
<td>pass</td>
</tr>
<tr>
<td></td>
<td><strong>Utils tests</strong> Test checkDoubleCoursesRecursive</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2 courses that can only be given in timeslot A, should give a conflict.</td>
<td>pass</td>
</tr>
<tr>
<td>2</td>
<td>1 course in timeslot B, and another in timeslot A, no problems</td>
<td>pass</td>
</tr>
<tr>
<td>3</td>
<td>1 course in timeslot A or C and another in timeslot A or C, no problems</td>
<td>pass</td>
</tr>
<tr>
<td>4</td>
<td>1 course in timeslot A and C and another in timeslot A or C, not possible</td>
<td>pass</td>
</tr>
<tr>
<td>5</td>
<td>4 courses, filled schedule, but it will fit</td>
<td>pass</td>
</tr>
<tr>
<td></td>
<td><strong>Utils tests</strong> Test makePackage</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>The package text must be Smart Environment</td>
<td>pass</td>
</tr>
<tr>
<td>2</td>
<td>Target group should be stored in the data of the package.</td>
<td>pass</td>
</tr>
<tr>
<td></td>
<td><strong>Utils tests</strong> Test makeSubject</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Name of the course should be properly set.</td>
<td>pass</td>
</tr>
<tr>
<td>2</td>
<td>This is a course, so is should have the class course.</td>
<td>pass</td>
</tr>
<tr>
<td>3</td>
<td>Code of the course is 2IP90</td>
<td>pass</td>
</tr>
<tr>
<td>4</td>
<td>Subject code should be shown in the div</td>
<td>pass</td>
</tr>
<tr>
<td></td>
<td><strong>Utils tests</strong> Test noConflict</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>3 courses in different timeslots is no conflict</td>
<td>pass</td>
</tr>
<tr>
<td></td>
<td>2 courses in the same timeslot is a conflict</td>
<td>pass</td>
</tr>
<tr>
<td>---</td>
<td>------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td><strong>Utils tests</strong></td>
<td>Test noCurrConflict</td>
<td>passed all</td>
</tr>
<tr>
<td>1</td>
<td>One course in timeslot A and B, and another in A, should give a conflict.</td>
<td>pass</td>
</tr>
<tr>
<td>2</td>
<td>One course in timeslot C and B, and another in A, should give no conflict.</td>
<td>pass</td>
</tr>
<tr>
<td><strong>Storage tests</strong></td>
<td>Test asking for save name.</td>
<td>passed all</td>
</tr>
<tr>
<td>1</td>
<td>No currentScheduleName, so save name should be asked.</td>
<td>pass</td>
</tr>
<tr>
<td>2</td>
<td>There is a currentScheduleName, so save name should not be asked.</td>
<td>pass</td>
</tr>
<tr>
<td><strong>Storage tests</strong></td>
<td>Test showSchedule(data).</td>
<td>passed all</td>
</tr>
<tr>
<td>1</td>
<td>There should only be one schedule table.</td>
<td>pass</td>
</tr>
<tr>
<td>2</td>
<td>The schedule table should have 4 rows (header is not counted here, since that is a thead).</td>
<td>pass</td>
</tr>
<tr>
<td>3</td>
<td>Year 1, quartile 1, should have 1 subject.</td>
<td>pass</td>
</tr>
<tr>
<td>4</td>
<td>Year 1, quartile 2, should have 2 subjects.</td>
<td>pass</td>
</tr>
<tr>
<td>5</td>
<td>Year 1, quartile 3, should have no subjects.</td>
<td>pass</td>
</tr>
<tr>
<td>6</td>
<td>Year 4, quartile 2, should have 1 subject.</td>
<td>pass</td>
</tr>
<tr>
<td>7</td>
<td>Year 1, quartile 1, should not have the .conflictingQuartile class (also check the test for checkDoubleCoursesRecursive if this fails).</td>
<td>pass</td>
</tr>
<tr>
<td>8</td>
<td>Year 1, quartile 2, should have the .conflictingQuartile class (also check the test for checkDoubleCoursesRecursive if this fails).</td>
<td>pass</td>
</tr>
<tr>
<td>9</td>
<td>Year 1, quartile 3, should not have the .conflictingQuartile class (also check the test for checkDoubleCoursesRecursive if this fails).</td>
<td>pass</td>
</tr>
<tr>
<td>10</td>
<td>Year 1, quartile 1, should have '2IP90' as the first subject's code.</td>
<td>pass</td>
</tr>
<tr>
<td>11</td>
<td>Year 1, quartile 1, should have 'Programming' as the first subject's name.</td>
<td>pass</td>
</tr>
<tr>
<td>12</td>
<td>Year 1, quartile 1, should have a close button (.close) on the first subject.</td>
<td>pass</td>
</tr>
<tr>
<td>13</td>
<td>Year 1, quartile 1, should have .major on the first subject.</td>
<td>pass</td>
</tr>
<tr>
<td>14</td>
<td>Year 1, quartile 2, should not have .major on the first subject.</td>
<td>pass</td>
</tr>
<tr>
<td>15</td>
<td>The 'Add year' button should not be disabled.</td>
<td>pass</td>
</tr>
<tr>
<td></td>
<td>The ‘Remove year’ button should not be disabled.</td>
<td>pass</td>
</tr>
<tr>
<td>---</td>
<td>------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>Search subjects tests</td>
<td>Test getPostdata</td>
<td>passed all</td>
</tr>
<tr>
<td>1</td>
<td>Posted major should be 1375.</td>
<td>pass</td>
</tr>
<tr>
<td>2</td>
<td>Year that was posted should be 1.</td>
<td>pass</td>
</tr>
<tr>
<td>3</td>
<td>Searchterm should be test.</td>
<td>pass</td>
</tr>
<tr>
<td>4</td>
<td>Looking for broadening subjects.</td>
<td>pass</td>
</tr>
<tr>
<td>5</td>
<td>Timeslot clicked is B.</td>
<td>pass</td>
</tr>
<tr>
<td>6</td>
<td>Quartile was 3.</td>
<td>pass</td>
</tr>
<tr>
<td>7</td>
<td>Difficulty was basic.</td>
<td>pass</td>
</tr>
<tr>
<td>Search subjects tests</td>
<td>Test showSearchResults</td>
<td>passed all</td>
</tr>
<tr>
<td>1</td>
<td>Subject should have been added to the search results.</td>
<td>pass</td>
</tr>
<tr>
<td>2</td>
<td>No courses in search results if there were no results.</td>
<td>pass</td>
</tr>
<tr>
<td>Search subjects tests</td>
<td>Test showSearchResults</td>
<td>passed all</td>
</tr>
<tr>
<td>1</td>
<td>Keyword is test, word is testing, should highlight the word “test”.</td>
<td>pass</td>
</tr>
<tr>
<td>Search subjects tests</td>
<td>Test removeKeywordHighlight</td>
<td>passed all</td>
</tr>
<tr>
<td>1</td>
<td>testing (test was highlighted) should turn into testing (no highlight) after removeKeywordHighlight.</td>
<td>pass</td>
</tr>
<tr>
<td>Search packages tests</td>
<td>Test coherentSearchButtonWaiting</td>
<td>passed all</td>
</tr>
<tr>
<td>1</td>
<td>Search button should be disabled after calling the function.</td>
<td>pass</td>
</tr>
<tr>
<td>2</td>
<td>Search button must be enabled again.</td>
<td>pass</td>
</tr>
<tr>
<td>Search packages tests</td>
<td>Test showCoherentPackages</td>
<td>passed all</td>
</tr>
<tr>
<td>1</td>
<td>When no search results are returned, the string “No results found” should be displayed.</td>
<td>pass</td>
</tr>
<tr>
<td>2</td>
<td>2 packages should be added to the search-results.</td>
<td>pass</td>
</tr>
<tr>
<td>3</td>
<td>First packet added had name “test”.</td>
<td>pass</td>
</tr>
<tr>
<td>Studyplanner tests</td>
<td>Test setYear</td>
<td>passed all</td>
</tr>
<tr>
<td>1</td>
<td>Start year set to 2014.</td>
<td>pass</td>
</tr>
<tr>
<td>Studyplanner tests</td>
<td>Test addYear and removeYear</td>
<td>passed all</td>
</tr>
<tr>
<td>1</td>
<td>After three times addYear the year count should be 6.</td>
<td>pass</td>
</tr>
<tr>
<td>2</td>
<td>The table should now contain 6 rows.</td>
<td>pass</td>
</tr>
<tr>
<td>3</td>
<td>After additionally two times removeYear the year count should be 4.</td>
<td>pass</td>
</tr>
<tr>
<td>Studyplanner tests</td>
<td>Test fillInCourses</td>
<td>passed all</td>
</tr>
<tr>
<td>--------------------</td>
<td>--------------------</td>
<td>------------</td>
</tr>
<tr>
<td>1</td>
<td>2 courses should be added to the schedule.</td>
<td>pass</td>
</tr>
<tr>
<td>2</td>
<td>And this course should be in year 1, quartile 3.</td>
<td>pass</td>
</tr>
<tr>
<td>3</td>
<td>0 courses should be added to the schedule.</td>
<td>pass</td>
</tr>
<tr>
<td>4</td>
<td>1 course was already in the schedule, one should be added.</td>
<td>pass</td>
</tr>
<tr>
<td>5</td>
<td>Course that could have been added in year 1, q 3, was already elsewhere in the schedule, so it should not be added.</td>
<td>pass</td>
</tr>
<tr>
<td>6</td>
<td>Filling in no courses in an empty schedule, schedule stays empty.</td>
<td>pass</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Studyplanner tests</th>
<th>Test updateMajorErrorBar</th>
<th>passed all</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>We have no major, and we ‘pressed’ the deepening/broadening button, the search button should be disabled.</td>
<td>pass</td>
</tr>
<tr>
<td>2</td>
<td>Now we have a major, so #major-error-bar should not be shown.</td>
<td>pass</td>
</tr>
<tr>
<td>3</td>
<td>No major anymore, but the deepening/broadening button was not pressed, so #major-error-bar should not be shown.</td>
<td>pass</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Studyplanner tests</th>
<th>Test saveMajor</th>
<th>passed all</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Set the major ID to ‘1234’.</td>
<td>pass</td>
</tr>
<tr>
<td>2</td>
<td>Set the major name to ‘Test major’.</td>
<td>pass</td>
</tr>
<tr>
<td>3</td>
<td>Set the first year to ‘2013’.</td>
<td>pass</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Studyplanner tests</th>
<th>Test colorCourses</th>
<th>passed all</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 major course was added.</td>
<td>pass</td>
</tr>
<tr>
<td>2</td>
<td>The subject with code “test2” should have class major.</td>
<td>pass</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Studyplanner tests</th>
<th>Test removeCourse</th>
<th>passed all</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>One course in the schedule.</td>
<td>pass</td>
</tr>
<tr>
<td>2</td>
<td>No courses in the schedule after the remove.</td>
<td>pass</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Studyplanner tests</th>
<th>Test showSubjectInformation</th>
<th>passed all</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The subject code must be shown.</td>
<td>pass</td>
</tr>
<tr>
<td>2</td>
<td>The subject name should be shown.</td>
<td>pass</td>
</tr>
<tr>
<td>3</td>
<td>The difficulty level should be shown.</td>
<td>pass</td>
</tr>
<tr>
<td></td>
<td>The number of ects should be shown.</td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>-----------------------------------</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Prior knowledge should be shown.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Studyplanner tests</td>
<td>Test checkDoubleCourses</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>the same courses are given in the same timeslot, so the cell should have class conflictingQuartile.</td>
<td>pass</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>pass</td>
</tr>
<tr>
<td>Studyplanner tests</td>
<td>Test emptySchedule</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Added one course to the schedule.</td>
<td>pass</td>
</tr>
<tr>
<td>2</td>
<td>No courses in schedule after emptySchedule.</td>
<td>pass</td>
</tr>
<tr>
<td>Studyplanner tests</td>
<td>Test setRecommendedSubjects</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Two courses should have been added.</td>
<td>pass</td>
</tr>
<tr>
<td>2</td>
<td>First course should have code “code”.</td>
<td>pass</td>
</tr>
<tr>
<td>Translation tests</td>
<td>Test whether the some translated strings are available.</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>.year should be defined.</td>
<td>pass</td>
</tr>
<tr>
<td>2</td>
<td>.aScheduleOfNameXDoesntExist should be defined.</td>
<td>pass</td>
</tr>
<tr>
<td>3</td>
<td>.blahblah should not be defined.</td>
<td>pass</td>
</tr>
</tbody>
</table>
5.2 Python tests

A test report should be generated after each execution of a test procedure. It contains the following items:

- Class name of the testcase.
- Test name.
- A description of the error if the test failed.
- Time the whole testcase took.

These are generated in XUnit format in the file nosetests.xml. In the folder tests_results/ a coverage report is generated in HTML format. This can be viewed using any browser.
<table>
<thead>
<tr>
<th>Class name</th>
<th>Test name</th>
<th>Result</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>query.major.testMajor.testMajor</td>
<td>test_list</td>
<td>pass</td>
<td>11.5</td>
</tr>
<tr>
<td>query.major.testMajor.testMajor</td>
<td>test_package</td>
<td>pass</td>
<td>4.0</td>
</tr>
<tr>
<td>query.package.testPackage.testPackage</td>
<td>test_package</td>
<td>pass</td>
<td>4.3</td>
</tr>
<tr>
<td>query.package.testPackage.testPackage</td>
<td>test_search</td>
<td>pass</td>
<td>4.4</td>
</tr>
<tr>
<td>query.recommendation.testRecommendation.testRecommendation</td>
<td>test_recommendations</td>
<td>pass</td>
<td>80.5</td>
</tr>
<tr>
<td>query.schedule.testSchedule.testSchedule</td>
<td>test_all_schedules</td>
<td>pass</td>
<td>8.3</td>
</tr>
<tr>
<td>query.schedule.testSchedule.testSchedule</td>
<td>test_delete_schedule</td>
<td>pass</td>
<td>8.2</td>
</tr>
<tr>
<td>query.schedule.testSchedule.testSchedule</td>
<td>test_load_non_existing_schedule</td>
<td>pass</td>
<td>8.0</td>
</tr>
<tr>
<td>query.schedule.testSchedule.testSchedule</td>
<td>test_load_non_related_schedule</td>
<td>pass</td>
<td>8.3</td>
</tr>
<tr>
<td>query.schedule.testSchedule.testSchedule</td>
<td>test_overwrite_schedule</td>
<td>pass</td>
<td>8.2</td>
</tr>
<tr>
<td>query.schedule.testSchedule.testSchedule</td>
<td>test_rename_schedule</td>
<td>pass</td>
<td>8.4</td>
</tr>
<tr>
<td>query.schedule.testValidate.CoherentTestCase</td>
<td>test_fail</td>
<td>pass</td>
<td>13.6</td>
</tr>
<tr>
<td>query.schedule.testValidate.CoherentTestCase</td>
<td>test_pass</td>
<td>pass</td>
<td>5.8</td>
</tr>
<tr>
<td>query.schedule.testValidate.OverlapTestCase</td>
<td>test_fail</td>
<td>pass</td>
<td>103.3</td>
</tr>
<tr>
<td>query.schedule.testValidate.OverlapTestCase</td>
<td>test_pass</td>
<td>pass</td>
<td>17.6</td>
</tr>
<tr>
<td>query.schedule.testValidate.USETestCase</td>
<td>test_fail</td>
<td>pass</td>
<td>4.6</td>
</tr>
<tr>
<td>query.schedule.testValidate.USETestCase</td>
<td>test_pass</td>
<td>pass</td>
<td>5.6</td>
</tr>
<tr>
<td>query.schedule.testValidate.Validate TestCase</td>
<td>test_failCoherent</td>
<td>pass</td>
<td>4.4</td>
</tr>
<tr>
<td>query.schedule.testValidate.Validate TestCase</td>
<td>test_failMajor</td>
<td>pass</td>
<td>253.3</td>
</tr>
<tr>
<td>query.schedule.testValidate.Validate TestCase</td>
<td>test_failOverlap</td>
<td>pass</td>
<td>57.9</td>
</tr>
<tr>
<td>query.schedule.testValidate.Validate TestCase</td>
<td>test_failUSE</td>
<td>pass</td>
<td>7.0</td>
</tr>
<tr>
<td>query.schedule.testValidate.Validate TestCase</td>
<td>test_invalidUSE</td>
<td>pass</td>
<td>3.9</td>
</tr>
<tr>
<td>query.schedule.testValidate.Validate TestCase</td>
<td>test_passAll</td>
<td>pass</td>
<td>53.8</td>
</tr>
<tr>
<td>query.subject.testInfo.ConvertTestCase</td>
<td>test_invalidRequest</td>
<td>pass</td>
<td>0.6</td>
</tr>
<tr>
<td>query.subject.testInfo.ConvertTestCase</td>
<td>test_validRequest</td>
<td>pass</td>
<td>0.7</td>
</tr>
<tr>
<td>query.subject.testInfo.InfoTestCase</td>
<td>test_invalidRequest</td>
<td>pass</td>
<td>0.4</td>
</tr>
<tr>
<td>query.subject.testInfo.InfoTestCase</td>
<td>test_validRequest</td>
<td>pass</td>
<td>0.9</td>
</tr>
<tr>
<td>query.subject.testSearch.SearchTestCase</td>
<td>test_basicSearchCodeTerm</td>
<td>pass</td>
<td>49.7</td>
</tr>
<tr>
<td>Test Case</td>
<td>Method</td>
<td>Pass Score</td>
<td></td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>---------------------------------------------</td>
<td>------------</td>
<td></td>
</tr>
<tr>
<td>query.subject.testSearch.SearchTestCase</td>
<td>test_searchTermFormat</td>
<td>44.3</td>
<td></td>
</tr>
<tr>
<td>query.subject.testSearch.SearchTestCase</td>
<td>test_difficulty</td>
<td>43.9</td>
<td></td>
</tr>
<tr>
<td>query.subject.testSearch.SearchTestCase</td>
<td>test_invalidRequest</td>
<td>44.7</td>
<td></td>
</tr>
<tr>
<td>query.subject.testSearch.SearchTestCase</td>
<td>test_quartile</td>
<td>45.3</td>
<td></td>
</tr>
<tr>
<td>query.subject.testSearch.SearchTestCase</td>
<td>test_range</td>
<td>42.7</td>
<td></td>
</tr>
<tr>
<td>query.subject.testSearch.SearchTestCase</td>
<td>test_targetGroup</td>
<td>33.1</td>
<td></td>
</tr>
<tr>
<td>query.subject.testSearch.SearchTestCase</td>
<td>test_timeSlot</td>
<td>33.1</td>
<td></td>
</tr>
<tr>
<td>query.subject.testSearch.SearchTestCase</td>
<td>test_year</td>
<td>33.4</td>
<td></td>
</tr>
<tr>
<td>query.user.testAuthenticate.InfoTestCase</td>
<td>test_InvalidRequestInfo</td>
<td>0.3</td>
<td></td>
</tr>
<tr>
<td>query.user.testAuthenticate.InfoTestCase</td>
<td>test_LoggedInInfo</td>
<td>1.5</td>
<td></td>
</tr>
<tr>
<td>query.user.testAuthenticate.InfoTestCase</td>
<td>test_LoggedOutInfo</td>
<td>1.3</td>
<td></td>
</tr>
<tr>
<td>query.user.testAuthenticate.LoginTestCase</td>
<td>test_InvalidLoginTest</td>
<td>1.2</td>
<td></td>
</tr>
<tr>
<td>query.user.testAuthenticate.LoginTestCase</td>
<td>test_ValidLogin</td>
<td>8.2</td>
<td></td>
</tr>
<tr>
<td>query.user.testAuthenticate.RegisterTestCase</td>
<td>test_BasicLogouts</td>
<td>0.9</td>
<td></td>
</tr>
<tr>
<td>query.user.testAuthenticate.RegisterTestCase</td>
<td>test_InvalidPost</td>
<td>1.7</td>
<td></td>
</tr>
<tr>
<td>query.user.testAuthenticate.RegisterTestCase</td>
<td>test_InvalidRequests</td>
<td>1.4</td>
<td></td>
</tr>
<tr>
<td>query.user.testAuthenticate.RegisterTestCase</td>
<td>test_ValidCredentialsRegisters</td>
<td>4.6</td>
<td></td>
</tr>
<tr>
<td>query.user.testAuthenticate.RegisterTestCase</td>
<td>test_WrongUsernamePasswordRegisters</td>
<td>0.2</td>
<td></td>
</tr>
<tr>
<td>query.user.testChange.ChangePropertiesTestCase</td>
<td>test_invalidQuery</td>
<td>4.0</td>
<td></td>
</tr>
<tr>
<td>query.user.testChange.ChangePropertiesTestCase</td>
<td>test_negSuccessPasswordQuery</td>
<td>92.9</td>
<td></td>
</tr>
<tr>
<td>query.user.testChange.ChangePropertiesTestCase</td>
<td>test_negSuccessEmptyQuery</td>
<td>4.1</td>
<td></td>
</tr>
<tr>
<td>query.user.testChange.ChangePropertiesTestCase</td>
<td>test_successEmptyQuery</td>
<td>1.9</td>
<td></td>
</tr>
<tr>
<td>query.user.testChange.ChangePropertiesTestCase</td>
<td>test_successPasswordQuery</td>
<td>13.2</td>
<td></td>
</tr>
<tr>
<td>addData.tester.testAddData</td>
<td>test_AddCourse</td>
<td>7.4</td>
<td></td>
</tr>
<tr>
<td>addData.tester.testAddData</td>
<td>test_AddCoursePlanning</td>
<td>4.3</td>
<td></td>
</tr>
<tr>
<td>addData.tester.testAddData</td>
<td>test_AddSoftwareScienceMajor</td>
<td>4.6</td>
<td></td>
</tr>
<tr>
<td>addData.tester.testAddData</td>
<td>test_ClearDB</td>
<td>3.6</td>
<td></td>
</tr>
<tr>
<td>addData.tester.testAddData</td>
<td>test_GetAdditionalInfo</td>
<td>10.7</td>
<td></td>
</tr>
<tr>
<td>addData.tester.testAddData</td>
<td>test_OwisCourses</td>
<td>6.8</td>
<td></td>
</tr>
<tr>
<td>addData.tester.testAddData</td>
<td>test_ParseDoelgroep</td>
<td>0.2</td>
<td></td>
</tr>
<tr>
<td>addData.tester.testAddData</td>
<td>test_ParseProgramma</td>
<td>325.9</td>
<td></td>
</tr>
<tr>
<td>Test Case</td>
<td>Method</td>
<td>Result</td>
<td>Score</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>-----------------------</td>
<td>--------</td>
<td>-------</td>
</tr>
<tr>
<td>recommender_job.tester.testRecommender_job</td>
<td>test_CreateTuples</td>
<td>pass</td>
<td>8.5</td>
</tr>
<tr>
<td>recommender_job.tester.testRecommender_job</td>
<td>test_DeleteDoubles</td>
<td>pass</td>
<td>0.1</td>
</tr>
<tr>
<td>recommender_job.tester.testRecommender_job</td>
<td>test_DeleteOldTable</td>
<td>pass</td>
<td>0.0</td>
</tr>
<tr>
<td>recommender_job.tester.testRecommender_job</td>
<td>test_GetNrOfDoubles</td>
<td>pass</td>
<td>0.0</td>
</tr>
<tr>
<td>testCategorize.testPacket</td>
<td>test_rankPacket</td>
<td>pass</td>
<td>28.8</td>
</tr>
<tr>
<td>testCategorize.testSubject</td>
<td>test_rankEnglish</td>
<td>pass</td>
<td>27.0</td>
</tr>
<tr>
<td>testCategorize.testSubject</td>
<td>test_updateSubDB</td>
<td>pass</td>
<td>1.0</td>
</tr>
<tr>
<td>testCategorize.testRankStringNLTK</td>
<td>test_rankStringNLTK</td>
<td>pass</td>
<td>0.2</td>
</tr>
<tr>
<td>testSecurity.testSecurity</td>
<td>test_credentials</td>
<td>pass</td>
<td>0.6</td>
</tr>
<tr>
<td>Module</td>
<td>statements</td>
<td>missing</td>
<td>excluded</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------</td>
<td>---------</td>
<td>----------</td>
</tr>
<tr>
<td>KroketApp</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>KroketApp.admin</td>
<td>71</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>KroketApp.categorize</td>
<td>135</td>
<td>25</td>
<td>0</td>
</tr>
<tr>
<td>KroketApp.manager(^1)</td>
<td>9</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>KroketApp.models(^1)</td>
<td>113</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>KroketApp.query</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>KroketApp.query.major</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>KroketApp.query.major.list</td>
<td>11</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>KroketApp.query.major.subjects</td>
<td>21</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>KroketApp.query.package</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>KroketApp.query.package.search</td>
<td>65</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>KroketApp.query.package.subjects</td>
<td>16</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>KroketApp.query.recommendation</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>KroketApp.query.recommendation.subject</td>
<td>35</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>KroketApp.query.schedule</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>KroketApp.query.schedule.delete</td>
<td>17</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>KroketApp.query.schedule.list</td>
<td>10</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>KroketApp.query.schedule.load</td>
<td>64</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>KroketApp.query.schedule.rename</td>
<td>18</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>KroketApp.query.schedule.save</td>
<td>58</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>KroketApp.query.schedule.validate</td>
<td>66</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>KroketApp.query.subject</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>KroketApp.query.subject.info</td>
<td>50</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>KroketApp.query.subject.search</td>
<td>94</td>
<td>9</td>
<td>0</td>
</tr>
<tr>
<td>KroketApp.query.urls</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>KroketApp.query.user</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>KroketApp.query.user.authenticate</td>
<td>58</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>KroketApp.query.user.change</td>
<td>26</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

\(^1\)These files define the database structure, which is not possible to test using Django.
| KroketApp.recommmodels\(^1\)                  | 12   | 11   | 0    | 8%   |
| KroketApp.savemodels\(^1\)                  | 64   | 40   | 0    | 38%  |
| KroketApp.schedule                         | 0    | 0    | 0    | 100% |
| KroketApp.schedule.daily.jobs              | 0    | 0    | 0    | 100% |
| KroketApp.schedule.daily.jobs.recommender_job | 61   | 0    | 0    | 100% |
| KroketApp.schedule.daily.jobs.subject_updater | 0    | 0    | 0    | 100% |
| KroketApp.schedule.daily.jobs.subject_updater.addData | 157  | 20   | 0    | 87%  |
| KroketApp.schedule.daily.jobs.subject_updater.biztalk | 137  | 17   | 0    | 88%  |
| KroketApp.schedule.daily.jobs.subject_updater.parser | 64   | 0    | 0    | 100% |
| KroketApp.security                         | 32   | 2    | 0    | 94%  |
| KroketApp.urls                            | 3    | 0    | 0    | 100% |
| **Total**                                  | 1469 | 267  | 0    | 82%  |

\(^1\)These files define the database structure, which is not possible to test using Django.