Unit Test Plan

Version 1.0

Project Team
D.P. van den Berg 0949036
R.T. van Bergen 0938857
D.J.C. Dekker 0936100
S. van den Eerenbeemt 0954445
J. Mols 0851883
B.F. Rongen 0858160
B.W.M. van Rooijen 0895073
R.P. Schellekens 0944330
A.A. Vast 0854060
G. Walravens 0904152
S. Wessel 0941508

Project Managers
S.P.O. Oostveen
A. Rajaraman

Project Supervisor
dr. N. Zannone

Customer
dr. L. Genga
Abstract
This document contains a description of the unit test plan to be used with the Delta extension of the APD tool. The unit tests contained in this document reflect the software requirements specified in the Delta Software Requirements Document [2]. This document complies with the ESA software standards [1].
# Contents

1 Introduction .......................................................... 7
  1.1 Purpose .................................................................. 7
  1.2 Overview .................................................................. 7
  1.3 List of definitions and abbreviations ...................... 7
    1.3.1 List of definitions ........................................... 7
    1.3.2 List of abbreviations ....................................... 7
  1.4 List of references .................................................. 7

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 ....................................................... 8
  2.5 Environmental needs ............................................ 8
  2.6 Test case pass or fail criteria .................................. 9

3 Test case specifications .............................................. 10
  3.1 APD Facade tests .................................................. 10
  3.2 Database tests ..................................................... 13
    3.2.1 Database setup tests ....................................... 13
    3.2.2 Database credentials tests ............................... 13
    3.2.3 Database manager tests .................................. 14
  3.3 Registration tests .................................................. 14
    3.3.1 Basic registration tests .................................... 14
    3.3.2 Registration input validation tests ..................... 17
    3.3.3 Consent tests ................................................ 20
  3.4 Password reset tests .............................................. 22
    3.4.1 Basic password reset tests ............................... 23
    3.4.2 Password reset link creation tests ..................... 24
    3.4.3 Password reset token tests .............................. 25
    3.4.4 Password reset token retrieval tests .................. 26
  3.5 Login tests .......................................................... 28
  3.6 User tests ........................................................... 29
    3.6.1 Basic user tests .............................................. 30
    3.6.2 User database query tests ............................... 34
    3.6.3 User input validation tests ................................ 37
    3.6.4 User account information tests ......................... 37
    3.6.5 Edit user account information tests ................... 38
    3.6.6 Change user password tests ............................. 39
    3.6.7 User deletion tests ......................................... 39
    3.6.8 Ownership transfer tests .................................. 40
  3.7 Administrator tests ............................................... 41
    3.7.1 Basic administrator tests .................................. 41
    3.7.2 Administrator database queries tests .................. 41
    3.7.3 Deleting user accounts .................................... 42
  3.8 Project tests ........................................................ 42
    3.8.1 Basic project tests .......................................... 42
    3.8.2 Project database queries tests .......................... 45
    3.8.3 Project creation tests ...................................... 47
    3.8.4 Project input validation tests ............................ 52
    3.8.5 Project deletion tests ..................................... 52
  3.9 Experiment tests .................................................. 53
    3.9.1 Basic experiment tests ..................................... 53
    3.9.2 Experiment database queries tests ...................... 56
    3.9.3 Experiment creation tests ................................ 59
    3.9.4 Experiment input validation tests ....................... 60
3.9.5 Experiment deletion tests ................................................................. 61
3.9.6 Retrieving and downloading experiment files tests .......................... 62
3.9.7 Experiment status tests ................................................................. 63
3.10 Experiment Phase tests ...................................................................... 64
  3.10.1 Basic experiment phase tests ......................................................... 64
  3.10.2 Experiment phase validation tests ................................................. 65
  3.10.3 Experiment phase input validation tests ........................................ 66
3.11 Experiment result tests ....................................................................... 67
  3.11.1 Subgraph expansion tests ............................................................. 67
  3.11.2 Experiment results retrieval tests ................................................ 68
3.12 Utility tests ....................................................................................... 70
  3.12.1 Form validation utility tests ......................................................... 70
  3.12.2 Mail utility tests ......................................................................... 71
  3.12.3 Error handling utility tests ......................................................... 73
  3.12.4 Response utility tests .................................................................. 73
  3.12.5 Timezone utility tests .................................................................. 75
  3.12.6 File extension validation tests ..................................................... 75
  3.12.7 Context Switch tests ................................................................. 78
3.13 Settings tests ..................................................................................... 78
4 Test procedures ..................................................................................... 80
  4.1 Unit test procedure .......................................................................... 80
    4.1.1 Purpose ...................................................................................... 80
    4.1.2 Procedure steps ........................................................................ 80
5 Test Coverage ....................................................................................... 81
  5.1 Server code ...................................................................................... 81
6 Test reports .......................................................................................... 85
Document Status

General

Title of document | Unit Test Plan
Document identifier | Delta.UTP/1.0
Version | 1.0
Authors
- D.P. van den Berg
- R.T. van Bergen
- D.J.C. Dekker
- S. van den Eerenbeemt
- J. Mols
- B.F. Rongen
- B.W.M. van Rooijen
- R.P. Schellekens
- A.A. Vast
- G. Walravens
- S. Wessel

History

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Author(s)</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>2018-07-05</td>
<td>D.P. van den Berg</td>
<td>Auto-generated</td>
</tr>
<tr>
<td></td>
<td></td>
<td>J. Mols</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>B.F. Rongen</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>A.A. Vast</td>
<td></td>
</tr>
<tr>
<td>Section</td>
<td>Reason</td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>-------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Added test case specifications</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
1 Introduction

1.1 Purpose

This Unit Test Plan (UTP) document has the purpose of specifying how the Delta extension for the APD tool needs to be tested such that all parts of the software work as intended. The extension is comprised of multiple units, and these units are grouped to structure the tests in the UTP. The results of the unit tests will be described in this document.

1.2 Overview

After Section 1, Section 2 will describe a test plan which serves as a general guide for the setup and completion of the unit tests, as well as a general guide for the completion of the test plan in this document. The requirements for the test plan will be described, including test deliverables, testing tasks and environmental needs. Section 3 contains a detailed specification of each individual test case. Section 4 expands upon the test procedures each of which has a purpose and a set of steps that are described. In Section 5 the test coverage results for the tests in Section 3 are highlighted. Lastly, Section 6 offers the test reports obtained by the completion of this Unit Test Plan.

1.3 List of definitions and abbreviations

1.3.1 List of definitions

For any definitions specific to this product, the list of definitions in Section 1.3.1 of the URD [3] can be referenced.

1.3.2 List of abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>APD</td>
<td>Anomalous Process Discovery</td>
</tr>
<tr>
<td>ATP</td>
<td>Acceptance Test Plan</td>
</tr>
<tr>
<td>SRD</td>
<td>Software Requirements Document</td>
</tr>
<tr>
<td>STD</td>
<td>Software Transfer Document</td>
</tr>
<tr>
<td>URD</td>
<td>User Requirements Document</td>
</tr>
</tbody>
</table>

1.4 List of references


2 Test plan

2.1 Test items

The software to be tested is the Delta extension of the APD tool. This extension consists of the
website that all users can access, the user management system, and the interface between the web-
site and the APD tool. Since project sharing, modular experiments, and custom subgraph discovery
components have not been implemented, no test cases for these proposed parts are included. Fur-
ther details on the extension, including the parts that are not implemented, can be found in the
URD [3] and SRD [2].

2.2 Features to be tested

The features to be tested are all the implemented features of the Delta extension, except the user
interface elements. These elements include the web pages and the scripts that are running on the
front-end.

2.3 Test deliverables

Before the unit test plan can be executed, the following documents and products need to be deliv-
ered:

- UTP section 1 through 4 (this document).
- The final version of the Delta extension of the APD tool.

After completing the test procedure the following documents must be delivered:

- UTP section 5.
- Problem reports (if any).

2.4 Testing tasks

Before testing, the following tasks need to be performed:

- Designing unit tests for each of the features to be tested.
- Defining and creating input data for the unit tests where necessary.
- Ensure all the environmental needs are met for the tests.

Once the above tasks have been completed, the tests can be performed according to the procedures
described in chapter 4.

2.5 Environmental needs

To run the unit tests, the following environmental requirements need to be met:

- Linux 14.04 or newer is installed.
- PHP 5.5 with the Graphviz (libgv-php5, ≥ 2.36.0) and MySQL (php5-mysqlnd ≥ 5.5.9) ex-
tensions is installed.
- The PHPUnit testing framework is available as a PHAR package, version ≥ 4.8.36.
- The source code of the Delta extension and the APD tool.
2.6 Test case pass or fail criteria

The pass criteria for each unit test are described along with the test. If these criteria are not met, the test fails. The overall unit test plan passes only if all individual tests pass.
3 Test case specifications

For the test case specification, we will use the following template.

```
Example description, describing what is tested.

**Test subject:** exampleTestedNameSpace\exampleTestedMethod(exampleVar1 : string, exampleVar2 : exampleObject)

Precondition: Example precondition
Input: Example input
Output: Example output
```

First, the identifier of the test is given, together with the test's namespace and method name. Then, a general description is provided, which summarizes what is being tested. Each test case specification also includes four properties: the test subject, precondition, input and output. The test subject describes the method (and namespace of the method, if the method is not global), together with the parameters of the method with given types. The precondition describes the environmental needs necessary to run the test case. The input property describes the input necessary to test the subject, whereas the output describes the behaviour that ought to be observed in order to pass the test case.

### 3.1 APD Facade tests

In this section, the unit tests that test the APD Facade are described. These tests check whether experiments are correctly started and whether experiment results are correctly retrieved, both with and without expansion of nodes.

```
Test subject: \apd\EsSubFacade\getSubgraphExtractionResult(experiment : Experiment)
Precondition: The phase anomalous subgraph discovery phase has been run on the input experiment
Input: An existing experiment
Output: A non-null result starting with '<svg'
```

```
Test subject: \apd\EsSubFacade\getSubgraphExtractionResult(experiment : Experiment)
Precondition: None
Input: A non-existing experiment
Output: Null
```

```
Test subject: \apd\EsSubFacade\getSubgraphExtractionResult(experiment : Experiment)
Precondition: None
Input: A non-existing experiment and an invalid pattern type
Output: Throws an InvalidArgumentException
```
Tests whether an exception is thrown when an invalid pattern type is given while trying to acquire the partial order discovery results.

**Test subject:** \apd\EsubFacade\getSubgraphExtractionResult(experiment : Experiment)

**Precondition:** None

**Input:** A non-existing experiment and an invalid pattern type

**Output:** Throws an InvalidArgumentException

---

Tests the acquiring of partial order discovery results on an existing experiment.

**Test subject:** \apd\EsubFacade\getPartialOrderDiscoveryResult(experiment : Experiment, patternType : int, supportThreshold : float, expandedNodes : string[])

**Precondition:** The partial order discovery phase has been run on the input experiment

**Input:** An existing experiment

**Output:** A non-null result starting with '<svg'

---

Tests the acquiring of partial order discovery results on a non-existing experiment.

**Test subject:** \apd\EsubFacade\getPartialOrderDiscoveryResult(experiment : Experiment, patternType : int, supportThreshold : float, expandedNodes : string[])

**Precondition:** None

**Input:** A non-existing experiment

**Output:** Null

---

Tests whether an exception is thrown when trying to start partial order discovery when anomalous subgraph discovery is not finished.

**Test subject:** \apd\EsubFacade\startPartialOrderDiscoveryPhase(experiment : Experiment, orThreshold : float, fThreshold : float)

**Precondition:** The anomalous pattern discovery phase has not been run on the input experiment

**Input:** An existing experiment

**Output:** An IllegalStateException is thrown

---

Tests the running of anomalous subgraph discovery phase.

**Test subject:** \apd\EsubFacade\startSubgraphExtractionPhase(experiment : Experiment)

**Precondition:** None

**Input:** An existing experiment

**Output:** The subgraph extraction phase is initialized, indicated by the 'log.txt' file being created

---

Tests the running of partial order discovery phase.

**Test subject:** \apd\EsubFacade\startPartialOrderDiscoveryPhase(experiment : Experiment, orThreshold : float, fThreshold : float)

**Precondition:** The anomalous subgraph discovery phase for the given experiment is finished

**Input:** An existing experiment

**Output:** The partial order discovery phase is initialized, indicated by the 'rules_log.txt' file being created
Tests the post processing of the anomalous subgraph discovery phase.

**Test subject:** \apd\EsubFacadeTest\testExpandSubgraphNodeProcessor()

**Precondition:** The subgraph discovery phase has been run on the given experiment

**Input:** An existing experiment

**Output:** A directory delta, expanded.subs and expandedSubs.dot are created in the experiment folder

---

Tests the acquiring of an expanded subgraph.

**Test subject:** \apd\EsubFacade\getNodeExpandedGraph(experiment : Experiment, patterns : string[]|null)

**Precondition:** The subgraph discovery phase has been run on the input experiment

**Input:** An existing experiment and the name of one subgraph

**Output:** A result starting with '<svg' that contains only one 'class="cluster"' substring

---

Tests the acquiring of multiple expanded subgraphs.

**Test subject:** \apd\EsubFacade\getNodeExpandedGraph(experiment : Experiment, patterns : string[]|null)

**Precondition:** The subgraph discovery phase has been run on the input experiment

**Input:** An existing experiment and the name of two subgraphs

**Output:** A result starting with '<svg' that contains two 'class="cluster"' substrings

---

Tests the acquiring of all expanded subgraphs.

**Test subject:** \apd\EsubFacade\getNodeExpandedGraph(experiment : Experiment, patterns : string[]|null)

**Precondition:** The subgraph discovery phase has been run on the input experiment

**Input:** An existing experiment with 73 patterns that can be expanded

**Output:** A result starting with '<svg' that contains 73 occurrences of the 'class="cluster"' substring

---

Tests the acquiring of a pattern.

**Test subject:** \apd\EsubFacade\getPartialSubgraph(experiment : Experiment, patterns : string[]|null)

**Precondition:** The subgraph discovery phase has been run on the input experiment

**Input:** An existing experiment and the name of one subgraph

**Output:** A result starting with '<svg' that contains one occurrence of the 'cluster.' substring

---

Tests the acquiring of multiple patterns.

**Test subject:** \apd\EsubFacade\getPartialSubgraph(experiment : Experiment, patterns : string[]|null)

**Precondition:** The subgraph discovery phase has been run on the input experiment

**Input:** An existing experiment and the name of two subgraphs

**Output:** A result starting with '<svg' that contains two occurrences of the 'cluster.' substring
3.2 Database tests

In this section, unit tests are described that test whether the database manager works correctly, and whether the database credentials are correct. Also, tests are included that test whether queries can be successfully executed.

3.2.1 Database setup tests

UT-16  
DatabaseTest\testDatabaseObjectNotNull()

Tests whether the database is globally available.

Test subject: Global database variable
Precondition: None
Input: None
Output: The global database variable

UT-17  
DatabaseTest\testDatabaseObject()

Tests the preparing of database queries.

Test subject: \DatabaseManager\prepare(string)
Precondition: The global database object is not null
Input: Valid SQL query
Output: A zero error code for success

3.2.2 Database credentials tests

UT-18  
DatabaseCredentialsTest\testGetDatabaseCredentialsHost()

Tests the retrieval of the host name of the database credentials.

Test subject: \DatabaseCredentials\getHost()
Precondition: None
Input: None
Output: The host of the DatabaseCredentials object

UT-19  
DatabaseCredentialsTest\testGetDatabaseCredentialsUsername()

Tests the retrieval of the username of the database credentials.

Test subject: \DatabaseCredentials\getUsername()
Precondition: None
Input: None
Output: The username of the DatabaseCredentials object

UT-20  
DatabaseCredentialsTest\testGetDatabaseCredentialsPassword()

Tests the retrieval of the password of the database credentials.

Test subject: \DatabaseCredentials\getPassword()
Precondition: None
Input: None
Output: The password of the DatabaseCredentials object
3.2.3 Database manager tests

UT-22 DatabaseManagerTest \testConnectionPrepareValid()

Tests the preparation of a valid SQL statement.

Test subject: \DatabaseManager\prepare(sql : string)
Precondition: None
Input: A valid SQL statement
Output: A non null statement object

UT-23 DatabaseManagerTest \testConnectionPrepareInvalid()

Tests the preparation and execution of an invalid SQL statement.

Test subject: \DatabaseManager\prepare(sql : string)
Precondition: None
Input: A valid SQL statement
Output: False

3.3 Registration tests

In this section, registration unit tests are described. In particular, registration input validation is tested, as well as the privacy consent.

3.3.1 Basic registration tests

UT-24 RegistrationRequestTest \testUsernameIncorrect()

Tests the creation of a user with an invalid username.

Test subject: createUserRequest(username : string, email : string, password : string, response : Response )
Precondition: The given username is incorrect
Input: A username, an email address, a password, a new Response object
Output: An error Response object indicating no user was created

UT-25 RegistrationRequestTest \testUsernameExists()

Tests the creation of a user with an invalid username.

Test subject: createUserRequest(username : string, email : string, password : string, response : Response )
Precondition: The given username is already taken
Input: A username, an email address, a password, a new Response object
Output: An error Response object indicating no user was created
Tests the creation of a user with an invalid email address.

**Test subject:** `createUserRequest(username : string, email : string, password : string, response : Response)`

**Precondition:** The given email address is incorrect

**Input:** A username, an email address, a password, a new Response object

**Output:** An error Response object indicating no user was created

Tests the creation of a user with an invalid email address.

**Test subject:** `createUserRequest(username : string, email : string, password : string, response : Response)`

**Precondition:** The given email address is already taken

**Input:** A username, an email address, a password, a new Response object

**Output:** An error Response object indicating no user was created

Tests the creation of a user with an invalid password.

**Test subject:** `createUserRequest(username : string, email : string, password : string, response : Response)`

**Precondition:** The given password is too long

**Input:** A username, an email address, a password, a new Response object

**Output:** An error Response object indicating no user was created

Tests the creation of a user.

**Test subject:** `createNewUser(username : string, email : string, password : string, response : Response)`

**Precondition:** The given input is correct

**Input:** A username, an email address, a password, a new Response object

**Output:** A successful Response object

Tests that no new user is created when given an invalid username.

**Test subject:** `createNewUser(username : string, email : string, password : string, response : Response)`

**Precondition:** The given username is too long

**Input:** A username, an email address, a password, a new Response object

**Output:** An error Response object indicating no new user was created

Tests that a new user is created when given valid account data.

**Test subject:** `createUserRequest(username : string, email : string, password : string, response : Response)`

**Precondition:** The given input is correct

**Input:** A username, an email address, a password, a new Response object

**Output:** A successful Response object
Tests that no new user is registered when no consent is given.

**Test subject:** \(\text{createUserRequest} (\text{username} : \text{string}, \text{email} : \text{string}, \text{password} : \text{string}, \text{response} : \text{Response})\)

**Precondition:** The given input is correct

**Input:** A username, an email address, a password and a boolean indicating no consent was given

**Output:** A successful Response object

Tests the registering of a new user.

**Test subject:** \(\text{registrationRequest} (\text{username} : \text{string}, \text{email} : \text{string}, \text{password} : \text{string}, \text{consentCheck} : \text{boolean})\)

**Precondition:** The given input is correct

**Input:** A username, an email address, a password

**Output:** A successful Response object

Tests the creation of a user profile.

**Test subject:** \(\text{createProfileEntry} (\text{username} : \text{string}|\text{null}, \text{firstName} : \text{string}|\text{null}, \text{lastName} : \text{string}|\text{null}, \text{organization} : \text{string}|\text{null}, \text{role} : \text{string}|\text{null}, \text{country} : \text{string}|\text{null}, \text{city} : \text{string}|\text{null}, \text{address} : \text{string}|\text{null}, \text{heardAbout} : \text{string}|\text{null})\)

**Precondition:** A user exists with the given username

**Input:** a username, a first name, a last name, an organization, a role, a country, a city, an address, how they heard about the tool

**Output:** A successful Response object

Tests the creation of a user profile with empty profile data.

**Test subject:** \(\text{createProfileEntry} (\text{username} : \text{string}|\text{null}, \text{firstName} : \text{string}|\text{null}, \text{lastName} : \text{string}|\text{null}, \text{organization} : \text{string}|\text{null}, \text{role} : \text{string}|\text{null}, \text{country} : \text{string}|\text{null}, \text{city} : \text{string}|\text{null}, \text{address} : \text{string}|\text{null}, \text{heardAbout} : \text{string}|\text{null})\)

**Precondition:** There exists a user with the given username

**Input:** A username, eight null values

**Output:** A successful Response object

Tests whether sending the registration mail succeeds.

**Test subject:** \(\text{registrationMail} (\text{mail} : \text{string}, \text{username} : \text{string})\)

**Precondition:** None

**Input:** An email and a username

**Output:** A successful Response object
3.3.2 Registration input validation tests

UT-37 RegistrationInputCheckTest\testUsernameIncorrect()

Tests the validation of a username.

Test subject: \checkUsername(username : string, response : Response)
Precondition: None
Input: An invalid username and a new Response object
Output: False

UT-38 RegistrationInputCheckTest\testUsernameExists()

Tests the validation of a username.

Test subject: \checkUsername(username : string, response : Response)
Precondition: There exists a user with the given username
Input: A valid username and a new Response object
Output: False

UT-39 RegistrationInputCheckTest\testUsernameCorrect()

Tests the validation of a username.

Test subject: \checkUsername(username : string, response : Response)
Precondition: None
Input: A valid username and a new Response object
Output: True

UT-40 RegistrationInputCheckTest\testEmailIncorrect()

Tests the validation of an email address.

Test subject: \checkEmail(email : string, response : Response)
Precondition: None
Input: An invalid email and a new Response object
Output: False

UT-41 RegistrationInputCheckTest\testEmailExists()

Tests the validation of an email address.

Test subject: \checkEmail(email : string, response : Response)
Precondition: There exists a user with the given email
Input: A valid email and a new Response object
Output: False

UT-42 RegistrationInputCheckTest\testEmailCorrect()

Tests the validation of an email address.

Test subject: \checkEmail(email : string, response : Response)
Precondition: None
Input: A valid email and a new Response object
Output: True
UT-43 RegistrationInputCheckTest\|test\|Data\|Validation\|Username\|Too\|Long() Tests the validation of all required account data.

**Test subject:** `dataValidation(username: string, email: string, password: string, response: Response)`

**Precondition:** None

**Input:** A username which is longer than 64 characters, a valid email, a valid password and a new Response object

**Output:** False

UT-44 RegistrationInputCheckTest\|test\|Data\|Validation\|Username\|Empty() Tests the validation of all required account data.

**Test subject:** `dataValidation(username: string, email: string, password: string, response: Response)`

**Precondition:** None

**Input:** An empty username, a valid email, a valid password and a new Response object

**Output:** False

UT-45 RegistrationInputCheckTest\|test\|Data\|Validation\|Username\|Contains\|Special() Tests the validation of all required account data.

**Test subject:** `dataValidation(username: string, email: string, password: string, response: Response)`

**Precondition:** None

**Input:** A username with a special character, a valid email, a valid password and a new Response object

**Output:** False

UT-46 RegistrationInputCheckTest\|test\|Data\|Validation\|Username\|Exists() Tests the validation of all required account data.

**Test subject:** `dataValidation(username: string, email: string, password: string, response: Response)`

**Precondition:** There exists a user with the given username.

**Input:** A valid username, a valid email, a valid password and a new Response object

**Output:** False

UT-47 RegistrationInputCheckTest\|test\|Data\|Validation\|Email\|Too\|Long() Tests the validation of all required account data.

**Test subject:** `dataValidation(username: string, email: string, password: string, response: Response)`

**Precondition:** None

**Input:** A valid username, an email which is longer than 64 characters, a valid password and a new Response object

**Output:** False
Tests the validation of all required account data.

Test subject:  

Precondition:  None
Input:  A valid username, an empty email, a valid password and a new Response object
Output:  False

Tests the validation of all required account data.

Test subject:  

Precondition:  None
Input:  A valid username, an invalid email, a valid password and a new Response object
Output:  False

Tests the validation of all required account data.

Test subject:  

Precondition:  There exists a user with the given email
Input:  A valid username, a valid email, a valid password and a new Response object
Output:  False

Tests the validation of all required account data.

Test subject:  

Precondition:  None
Input:  A valid username, a valid email, an invalid password and a new Response object
Output:  False

Tests the validation of all required account data.

Test subject:  

Precondition:  None
Input:  A valid username, a valid email, a valid password and a new Response object
Output:  True
3.3.3 Consent tests

**UT-53**

<table>
<thead>
<tr>
<th>Test subject:</th>
<th><code>query\consentTextById(id: int)</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Precondition:</td>
<td>The database is initialized and connected and the id is present in the database table</td>
</tr>
<tr>
<td>Input:</td>
<td>A valid consent text entry id</td>
</tr>
<tr>
<td>Output:</td>
<td>A ConsentText object</td>
</tr>
</tbody>
</table>

Tests the retrieval of the consent text.

**UT-54**

<table>
<thead>
<tr>
<th>Test subject:</th>
<th><code>query\latestConsentText()</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Precondition:</td>
<td>The database is initialized and connected and the id is present in the consent text database table</td>
</tr>
<tr>
<td>Input:</td>
<td>None</td>
</tr>
<tr>
<td>Output:</td>
<td>The most recently inserted consent text as a ConsentText object</td>
</tr>
</tbody>
</table>

Tests the retrieval of the most recently inserted consent text.

**UT-55**

<table>
<thead>
<tr>
<th>Test subject:</th>
<th><code>query\giveProfileConsent(user: User, method: string, text: ConsentText)</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Precondition:</td>
<td>The database is initialized and connected and the consent entry id is present in the consent text database table and the user exists</td>
</tr>
<tr>
<td>Input:</td>
<td>A user and a valid consent text entry id</td>
</tr>
<tr>
<td>Output:</td>
<td>The profile consent is added in the database</td>
</tr>
</tbody>
</table>

Tests the marking of consent for a user.

**UT-56**

<table>
<thead>
<tr>
<th>Test subject:</th>
<th><code>query\giveProfileConsentWithNullText(user: User, method: string, text: ConsentText)</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Precondition:</td>
<td>The database is initialized and connected, the user exists and the user has an existing consent entry</td>
</tr>
<tr>
<td>Input:</td>
<td>A user</td>
</tr>
<tr>
<td>Output:</td>
<td>The latest consent for the user is unchanged</td>
</tr>
</tbody>
</table>

Tests the marking of consent for a user.

**UT-57**

<table>
<thead>
<tr>
<th>Test subject:</th>
<th><code>query\revokeProfileConsent(user: User, method: string, text: ConsentText)</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Precondition:</td>
<td>The database is initialized and connected and the user exists</td>
</tr>
<tr>
<td>Input:</td>
<td>A user and a valid consent entry id</td>
</tr>
<tr>
<td>Output:</td>
<td>The consent is marked as revoked for the given user</td>
</tr>
</tbody>
</table>
ConsentQueriesTest\testRevokeProfileConsentWithNullText()

Tests the revoking of consent for a user.

**Test subject:** `\query\revokeProfileConsent(user : User, method : string, text : ConsentText)`

**Precondition:** The database is initialized and connected and the user exists

**Input:** A user  
**Output:** The default consent is revoked for the given user

ConsentQueriesTest\testHasGivenProfileConsentAfterAdd()

Tests the retrieval of consent for a user.

**Test subject:** `\query\hasGivenProfileConsent(user : User)`

**Precondition:** The database is initialized and connected, the user exists and the user has given consent

**Input:** A user  
**Output:** True

ConsentQueriesTest\testHasGivenProfileConsentAfterRevoke()

Tests the retrieval of consent for a user.

**Test subject:** `\query\hasGivenProfileConsent(user : User)`

**Precondition:** The database is initialized and connected, the user exists and the user has revoked his consent

**Input:** A user  
**Output:** False

ConsentQueriesTest\testLatestProfileConsentText()

Tests the retrieval of the latest consent text for which the user has consented.

**Test subject:** `\query\latestProfileConsentText(user : User)`

**Precondition:** The database is initialized and connected, the user exists and has given consent

**Input:** A user  
**Output:** A ConsentText object of the most recently added consent for the user

ConsentQueriesTest\testConsentTextByIdProblem()

Tests the retrieval of a consent text.

**Test subject:** `\query\consentTextById(id : int)`

**Precondition:** The database is initialized and connected and no consent text with the given id exists

**Input:** An invalid consent text entry id  
**Output:** Null

ConsentQueriesTest\testLatestProfileConsentTextProblem()

Tests the retrieval of the latest consent text for which the user has consented.

**Test subject:** `\query\latestProfileConsentText(user : User)`

**Precondition:** The database is initialized and connected, the user exists and the user has not given any consent

**Input:** A valid user  
**Output:** Null
Tests the retrieval of a consent text’s id.

**Test subject:** `ConsentText::getId()`

**Precondition:** There exists a `ConsentText` object with valid id, valid text and a valid creation date

**Input:** None

**Output:** The id of the `ConsentText` object

Tests the retrieval of a consent text.

**Test subject:** `ConsentText::getText()`

**Precondition:** There exists a `ConsentText` object with valid id, valid text and a valid creation date

**Input:** None

**Output:** The text of the `ConsentText` object

Tests the retrieval of a consent text’s creation date.

**Test subject:** `ConsentText::getCreationDate()`

**Precondition:** There exists a `ConsentText` object with valid id, valid text and a valid creation date

**Input:** None

**Output:** The creation date of the `ConsentText` object

Tests the textual representation of a consent text object.

**Test subject:** `ConsentText::__toString()`

**Precondition:** There exists a `ConsentText` object with valid id, valid text and a valid creation date

**Input:** None

**Output:** The text field of the `ConsentText` object

Tests the key-value representation of a consent text object.

**Test subject:** `ConsentText::toArray()`

**Precondition:** There exists a `ConsentText` object with valid id, valid text and a valid creation date

**Input:** None

**Output:** An array containing the valid values of all fields in the `ConsentText` object

### 3.4 Password reset tests

In this section, password reset tests are described. In particular, unit tests testing the password reset link creation are described, as well as tests regarding the password reset token and the retrieval of a password reset token.
3.4.1  Basic password reset tests

**UT-69**  
ForgotPasswordRequestTest\testForgotPasswordInvalidInput()

Tests the creation of a password reset link when given an invalid email or username.

**Test subject:** `createForgotPasswordLink(usernameOrEmail : String )`

**Precondition:** None

**Input:** An invalid email or username

**Output:** An error Response object indicating the email or username is invalid

**UT-70**  
ForgotPasswordRequestTest\testForgotPasswordNoUserWithEmail()

Tests the creation of a password reset link when given an email for which no user exists.

**Test subject:** `createForgotPasswordLink(usernameOrEmail : String )`

**Precondition:** There does not exist a user with the given email

**Input:** A valid email

**Output:** An error Response object indicating that there does not exists a user with the given email

**UT-71**  
ForgotPasswordRequestTest\testForgotPasswordNoUserWithUsername()

Tests the creation of a password reset link when given a username for which no user exists.

**Test subject:** `createForgotPasswordLink(usernameOrEmail : String )`

**Precondition:** There does not exist a user with the given username

**Input:** A valid username

**Output:** An error Response object indicating that there does not exists a user with the given username

**UT-72**  
ForgotPasswordRequestTest\testForgotPasswordWithEmail()

Tests the creation of a password reset link when given a valid email.

**Test subject:** `createForgotPasswordLink(usernameOrEmail : String )`

**Precondition:** There exists a user with the given email

**Input:** A valid email

**Output:** A successful Response object with the reset link as data

**UT-73**  
ForgotPasswordRequestTest\testForgotPasswordWithUsername()

Tests the creation of a password reset link when given a valid username.

**Test subject:** `createForgotPasswordLink(usernameOrEmail : String )`

**Precondition:** There exists a user with the given username

**Input:** A valid username

**Output:** A successful Response object with the reset link as data

**UT-74**  
ResetPasswordRequestTest\testTokenNotExists()

Tests the changing of a password through password reset.

**Test subject:** `resetPassword(token : string, password : string )`

**Precondition:** The given token does not exist in the database

**Input:** A token that does not exist and a valid password

**Output:** An error Response object indicating the token has expired
Tests the changing of a password through password reset.

**Test subject:** \( \text{resetPassword}(\text{token}: \text{string}, \text{password}: \text{string}) \)

**Precondition:** The given token exists in the database and is expired

**Input:** An expired token and a valid password

**Output:** An error Response object indicating the token has expired

---

Tests the changing of a password through password reset.

**Test subject:** \( \text{resetPassword}(\text{token}: \text{string}, \text{password}: \text{string}) \)

**Precondition:** The given token exists in the database and is already used

**Input:** An already used token and a valid password

**Output:** An error Response object indicating the token has expired

---

Tests the changing of a password through password reset.

**Test subject:** \( \text{resetPassword}(\text{token}: \text{string}, \text{password}: \text{string}) \)

**Precondition:** The given token exists in the database

**Input:** A valid token and multiple invalid passwords

**Output:** An error Response object indicating the password is invalid

---

Tests the changing of a password through password reset.

**Test subject:** \( \text{resetPassword}(\text{token}: \text{string}, \text{password}: \text{string}) \)

**Precondition:** The given token exists in the database and is not expired or used

**Input:** A valid token and a valid password

**Output:** A successful Response object

---

Tests the changing of a password through password reset.

**Test subject:** \( \text{resetPassword}(\text{token}: \text{string}, \text{password}: \text{string}) \)

**Precondition:** The given token exists in the database and is not expired or used

**Input:** A valid token and a valid password

**Output:** An error Response object indicating the token is expired

---

### 3.4.2 Password reset link creation tests

 Tests the creation of a password reset link when the logged in user is not an administrator.

**Test subject:** \( \text{createResetLink}(\text{userId}: \text{int}) \)

**Precondition:** The logged in user is not an administrator and there exists a user with the given user id

**Input:** A valid user id

**Output:** An error Response object indicating that the user does not have the correct permissions
Tests the creation of a password reset link for a non-existing user.

**Test subject:** `createResetLink(userId: int)`

**Precondition:** The logged in user is an administrator and there does not exists a user with the given user id

**Input:** A valid user id

**Output:** An error Response object indicating that the user does not exist

Tests the creation of a password reset link for a user.

**Test subject:** `createResetLink(userId: int)`

**Precondition:** The logged in user is an administrator and there exists a user with the given user id

**Input:** A valid user id

**Output:** A successful Response object with a password reset link as data

Tests the generation of a password reset link.

**Test subject:** `generateLink(user: User, response: Response)`

**Precondition:** None

**Input:** A user and a Response object

**Output:** A successful Response object with a password reset link as data

Tests the generation of a token.

**Test subject:** `generateToken()`

**Precondition:** None

**Input:** None

**Output:** A token of length 32

Tests the sending of an email with a password reset link.

**Test subject:** `passwordResetMail(resetLink: string, user: User, admin: null|User)`

**Precondition:** None

**Input:** A password reset link and a user

**Output:** An email containing the password reset link was sent to the user

### 3.4.3 Password reset token tests

**UT-86**

Tests the creation of a token.

**Test subject:** `query\newToken(token: string, user: User)`

**Precondition:** The database is initialized and connected and the token does not exists and the user with the given use id exists

**Input:** A valid token and a valid user id

**Output:** The token object
Tests the retrieval of a token.

**Test subject:** `query.tokenByToken(token: string)`

**Precondition:** The database is initialized and connected and a token with the given token exists

**Input:** A valid token

**Output:** The token object

Tests the retrieval of an invalid token.

**Test subject:** `query.tokenByToken(token: string)`

**Precondition:** The database is initialized and connected and no token with the given token exists

**Input:** A valid token

**Output:** Null

Tests the consuming of a token.

**Test subject:** `query.markTokenAsUsed(token: Token)`

**Precondition:** The database is initialized and connected and an unused token with the given token exists

**Input:** A valid token

**Output:** The token object with updated boolean flag

### 3.4.4 Password reset token retrieval tests

Tests the retrieval of a token’s user.

**Test subject:** `Token.getUser()`

**Precondition:** There exists a Token object with a valid token, user, creation date and boolean used flag and there exists a user associated to that token

**Input:** None

**Output:** The User object

Tests the retrieval of a token’s creation date.

**Test subject:** `Token.getCreationDate()`

**Precondition:** There exists a Token object with a valid token, user, creation date and boolean used flag

**Input:** None

**Output:** The creation date of the Token object
UT-92

Tests the retrieval of a token's contained token.

**Test subject:** `Token::getToken()`  

**Precondition:** There exists a Token object with a valid token, user, creation date and boolean used flag  

**Input:** None  

**Output:** The token contained in the called Token object

UT-93

Tests if a token is already used.

**Test subject:** `Token::isUsed()`  

**Precondition:** There exists a Token object with a valid token, user, creation date and boolean used flag  

**Input:** None  

**Output:** false

UT-94

Tests if a token is already used.

**Test subject:** `Token::isUsed()`  

**Precondition:** There exists a Token object with a valid token, user, creation date and boolean used flag  

**Input:** None  

**Output:** true

UT-95

Tests the retrieval of a token's link.

**Test subject:** `Token::getLink()`  

**Precondition:** There exists a Token object with a valid token, user, creation date and boolean used flag  

**Input:** None  

**Output:** A string representing a link to the password reset page

UT-96

Tests if a token is expired.

**Test subject:** `Token::isExpired()`  

**Precondition:** There exists a Token object with a valid token, user, creation date and boolean used flag  

**Input:** None  

**Output:** False

UT-97

Tests if a token is expired.

**Test subject:** `Token::isExpired()`  

**Precondition:** There exists a Token object with a valid token, user, creation date older than the expiration date and boolean used flag  

**Input:** None  

**Output:** True
3.5 Login tests

In this section, unit tests with respects to user login are described.

UT-98 LoginRequestTest\testLoginEmptyForm()

Tests the procedure of logging in when an empty username and password are given.
Test subject: \requestLogin(username:string, password:string)
Precondition: None
Input: An empty username and password
Output: An error Response object indicating that the username and password are empty

UT-99 LoginRequestTest\testLoginNullValues()

Tests the procedure of logging in when no input is given.
Test subject: \requestLogin(username:string, password:string)
Precondition: None
Input: None
Output: An error Response object indicating the username and password are empty

UT-100 LoginRequestTest\testLoginNoUsername()

Tests the procedure of logging in when an empty username is given.
Test subject: \requestLogin(username:string, password:string)
Precondition: None
Input: An empty username and a valid password
Output: An error Response object indicating the username is empty

UT-101 LoginRequestTest\testLoginNoPassword()

Tests the procedure of logging in when an empty password is given.
Test subject: \requestLogin(username:string, password:string)
Precondition: There exists a user with the given username
Input: A username and an empty password
Output: An error Response object indicating the password is empty

UT-102 LoginRequestTest\testLoginUsernameInvalid()

Tests the procedure of logging when given an invalid username.
Test subject: \requestLogin(username:string, password:string)
Precondition: None
Input: An invalid username and a password
Output: An error Response object indicating the username is invalid

UT-103 LoginRequestTest\testLoginWrongUsername()

Tests the procedure of logging in when there does not exists a user with the given username.
Test subject: \requestLogin(username:string, password:string)
Precondition: There does not exist a user with the given username
Input: A username and a password
Output: An error Response object indicating there does not exists a user with the given username
Tests the procedure of logging in when the password is incorrect.

Test subject: \requestLogin(username: string, password: string)
Precondition: There exists a user with the given username and the given password is not the same as the user's password
Input: A username and an incorrect password
Output: An error Response object indicating that the password is incorrect

Tests the procedure of logging in.

Test subject: \requestLogin(username: string, password: string)
Precondition: There exists a user with the given username and the given password is the same as the user's password
Input: A username and a password
Output: A successful Response object and the user is now logged in

Tests the check for if a user is logged in with a user that is not logged in.

Test subject: \isUserLoggedIn()
Precondition: The user is not logged in
Input: None
Output: An error Response object indicating that the user is not logged in

Tests the check for if a user is logged in with a logged in user.

Test subject: \isUserLoggedIn()
Precondition: The user is logged in and the user is not an admin
Input: None
Output: An error Response object indicating that the user is logged in and is not an admin

Tests the check for if a user is logged in with a logged in administrator.

Test subject: \isUserLoggedIn()
Precondition: The user is logged in and the user is an admin
Input: None
Output: An error Response object indicating that the user is logged in and is an admin

3.6 User tests

In this section, unit tests are described with respects to user accounts. In particular, database queries are tested, as well as user input validation, user account information, the editing of user account information, changing the password of a user and user deletion.
## 3.6.1 Basic user tests

**UT-109**  
**UserTest\testUserGetId()**  
Tests the retrieval of a user’s id.  
**Test subject:** `\User\getId()`  
**Precondition:** There exists a User object with a valid id, a valid username, a valid email and a valid administrator status  
**Input:** None  
**Output:** The id of the User object

**UT-110**  
**UserTest\testUserGetUsername()**  
Tests the retrieval of a user’s username.  
**Test subject:** `\User\getUsername()`  
**Precondition:** There exists a User object with a valid id, a valid username, a valid email and a valid administrator status  
**Input:** None  
**Output:** The username of the User object

**UT-111**  
**UserTest\testUserGetEmail()**  
Tests the retrieval of a user’s email address.  
**Test subject:** `\User\getEmail()`  
**Precondition:** There exists a User object with a valid id, a valid username, a valid email and a valid administrator status  
**Input:** None  
**Output:** The email of the User object

**UT-112**  
**UserTest\testUpdate()**  
Tests the updating of a user’s email address.  
**Test subject:** `\User\updateEmail(email:string)`  
**Precondition:** There exists a User object with a valid id, a valid username, a valid email, and a valid administrator status  
**Input:** A valid username and a valid email  
**Output:** The updated email of the User object

**UT-113**  
**UserTest\testDelete()**  
Tests the deletion of a user.  
**Test subject:** `\User\delete()`  
**Precondition:** There exists a User object with a valid id, a valid username, a valid email, and a valid administrator status  
**Input:** None  
**Output:** Null
UT-114 UserTest\testPasswordHash()

Tests the retrieval of a user’s hashed password.

**Test subject:** User\passwordHash()

**Precondition:** There exists a User object with a valid id, a valid username, a valid email, and a valid administrator status

**Input:** None

**Output:** The hashed password of the user

UT-115 UserTest\testUpdatePasswordHash()

Tests the updating of a user’s password.

**Test subject:** User\updatePassword(newPasswordHash : string)

**Precondition:** There exists a User object with a valid id, a valid username, a valid email, and a valid administrator status

**Input:** A valid password

**Output:** The updated password hash of the user

UT-116 UserTest\testGetProfile()

Tests the retrieval of a user’s profile data.

**Test subject:** User\getProfile()

**Precondition:** There exists a User object with a valid id, a valid username, a valid email, and a valid administrator status

**Input:** None

**Output:** The UserProfile object associated with the user

UT-117 UserTest\testGiveProfileConsent()

Tests the affirming of a user’s consent.

**Test subject:** User\giveProfileConsent(method : string, text : ConsentText)

**Precondition:** There exists a User object with a valid id, a valid username, a valid email, and a valid administrator status

**Input:** None

**Output:** True

UT-118 UserTest\testRevokeProfileConsent()

Tests the revoking of a user’s consent.

**Test subject:** User\revokeProfileConsent(method : string, text : ConsentText)

**Precondition:** There exists a User object with a valid id, a valid username, a valid email, and a valid administrator status and the user’s consent is added to the database

**Input:** None

**Output:** False

UT-119 UserTest\testHasNotGivenProfileConsent()

Tests whether a user has given consent to a privacy notice.

**Test subject:** User\hasGivenProfileConsent()

**Precondition:** There exists a User object with a valid id, a valid username, a valid email, and a valid administrator status

**Input:** None

**Output:** False
**UT-120**

*UserTest\testHasGivenProfileConsent()*

Tests whether a user has given consent to a privacy notice.

**Test subject:** \User\hasGivenProfileConsent()

**Precondition:** There exists a User object with a valid id, a valid username, a valid email, and a valid administrator status and the user’s consent is added to the database

**Input:** None

**Output:** True

---

**UT-121**

*UserTest\testLatestProfileConsentText()*

Tests the retrieving of the latest privacy notice that a user has consented with.

**Test subject:** \User\latestProfileConsentText()

**Precondition:** There exists a User object with a valid id, a valid username, a valid email, and a valid administrator status and the user has given consent

**Input:** None

**Output:** The ConsentText for which the user has consented

---

**UT-122**

*UserTest\testToArray()*

Tests the key-value representation of a user object.

**Test subject:** \User.toArray()

**Precondition:** There exists a User object

**Input:** None

**Output:** An array containing the field of the User object

---

**UT-123**

*UserProfileTest\testUserProfileGetUserId()*

Tests the retrieval of a user profile's id.

**Test subject:** \UserProfile\getUserId()

**Precondition:** There exists a UserProfile object with valid fields

**Input:** None

**Output:** The user id of the UserProfile object

---

**UT-124**

*UserProfileTest\testUserProfileGetFirstName()*

Tests the retrieval of the first name in a user profile.

**Test subject:** \UserProfile\getFirstName()

**Precondition:** There exists a UserProfile object with valid fields

**Input:** None

**Output:** The first name of the UserProfile object

---

**UT-125**

*UserProfileTest\testUserProfileGetLastName()*

Tests the retrieval of the last name in a user profile.

**Test subject:** \UserProfile\getLastName()

**Precondition:** There exists a UserProfile object with valid fields

**Input:** None

**Output:** The last name of the UserProfile object
Tests the retrieval of the organization in a user profile.

**Test subject:** `UserProfile\getOrganization()`

**Precondition:** There exists a `UserProfile` object with valid fields

**Input:** None

**Output:** The organization of the `UserProfile` object

Tests the retrieval of the organizational role in a user profile.

**Test subject:** `UserProfile\getRole()`

**Precondition:** There exists a `UserProfile` object with valid fields

**Input:** None

**Output:** The role of the `UserProfile` object

Tests the retrieval of the home country in a user profile.

**Test subject:** `UserProfile\getCountry()`

**Precondition:** There exists a `UserProfile` object with valid fields

**Input:** None

**Output:** The country of the `UserProfile` object

Tests the retrieval of the home city in a user profile.

**Test subject:** `UserProfile\getCity()`

**Precondition:** There exists a `UserProfile` object with valid fields

**Input:** None

**Output:** The city of the `UserProfile` object

Tests the retrieval of the home address in a user profile.

**Test subject:** `UserProfile\getAddress()`

**Precondition:** There exists a `UserProfile` object with valid fields

**Input:** None

**Output:** The address of the `UserProfile` object

Tests the retrieval of how a user heard about the tool in a user profile.

**Test subject:** `UserProfile\getHeardAbout()`

**Precondition:** There exists a `UserProfile` object with valid fields

**Input:** None

**Output:** The 'heard about' value of the `UserProfile` object
Tests the updating of a user profile.

**Test subject:** `UserProfile::update(firstName : string, lastName : string, organization : string, role : string, country : string, city : string, address : string, heardAbout : string)`

**Precondition:** There exists a UserProfile object with valid fields

**Input:** All valid input fields

**Output:** The updated UserProfile object

Tests the deleting of a user profile.

**Test subject:** `UserProfile::delete()`

**Precondition:** There exists a UserProfile object with valid fields

**Input:** None

**Output:** Null

Tests the key-value representation of a user profile object.

**Test subject:** `UserProfile::toArray()`

**Precondition:** There exists a UserProfile object with valid fields

**Input:** None

**Output:** The UserProfile object transformed to an array format

Tests the textual representation of a user profile object.

**Test subject:** `UserProfile::__toString()`

**Precondition:** There exists a UserProfile object with valid fields

**Input:** None

**Output:** The UserProfile object transformed to a string format

### 3.6.2 User database query tests

Tests the retrieval of a user profile.

**Test subject:** `UserProfileQueries::getUserProfileByUser(user : User)`

**Precondition:** The database is initialized and connected and the given user exists

**Input:** A user object

**Output:** The user profile object belonging to the given user

Tests the modification of a user profile.

**Test subject:** `UserProfileQueries::updateUserProfile(profile : UserProfile)`

**Precondition:** The database is initialized and connected and a profile with the given profile's id exists.

**Input:** A new UserProfile object and a user

**Output:** The user profile of the user is updated
Tests the deletion of a user profile from the database.

**Test subject:** `deleteUserProfile(userProfile : UserProfile)`

**Precondition:** The database is initialized and connected and the given user exists

**Input:** A valid user

**Output:** The user profile of the given user is removed

---

Tests the retrieval of a user by id.

**Test subject:** `userById(id : int )`

**Precondition:** The database is initialized and connected and a user with the given id exists

**Input:** A valid user id

**Output:** The user object is returned

---

Tests the retrieval of a non-existing user by id.

**Test subject:** `userById(id : int )`

**Precondition:** The database is initialized and connected and no user with the given id exists

**Input:** An invalid user id

**Output:** null

---

Tests the retrieval of a user by username.

**Test subject:** `userByUsername(username : string)`

**Precondition:** The database is initialized and connected and a user with the given username exists

**Input:** A valid username

**Output:** The user object is returned

---

Tests the retrieval of a non-existing user by username.

**Test subject:** `userByUsername(username : string)`

**Precondition:** The database is initialized and connected and no user with the given username exists

**Input:** An invalid username

**Output:** null

---

Tests the retrieval of a user by email.

**Test subject:** `userByEmail(email : string)`

**Precondition:** The database is initialized and connected and a user with the given email exists

**Input:** A valid email

**Output:** The user object is returned
**UT-144**

UserQueriesTest\testUserByEmailInvalidEmail()

Tests the retrieval of a non-existing user by email.

**Test subject:** \query\userByEmail(email : string)

**Precondition:** The database is initialized and connected and no user with the given email exists

**Input:** An invalid email

**Output:** null

**UT-145**

UserQueriesTest\testUpdatePasswordHash()

Tests the modification of the password hash.

**Test subject:** \query\updatePasswordHash(user : User, passwordHash : string)

**Precondition:** The database is initialized and connected and the given user exists

**Input:** A user object and a new password hash

**Output:** The password hash is updated in the database

**UT-146**

UserQueriesTest\testDeleteUser()

Tests the deletion of a user from the database.

**Test subject:** \query\deleteUser(user : User)

**Precondition:** The database is initialized and connected and the given user exists

**Input:** A user object

**Output:** The user is removed from the database

**UT-147**

UserQueriesTest\testPasswordHash()

Tests the retrieval of the password hash for a user.

**Test subject:** \query\passwordHash(user : User)

**Precondition:** The database is initialized and connected and the given user exists

**Input:** A user object

**Output:** The user's password hash

**UT-148**

UserQueriesTest\testAllUsers()

Tests the retrieval of all users and their information.

**Test subject:** \query\allUsers()

**Precondition:** The database is initialized and connected and there exist 3 users

**Input:** None

**Output:** All users data

**UT-149**

UserQueriesTest\testNoUsers()

Tests the retrieval of all users when no users exist.

**Test subject:** \query\allUsers()

**Precondition:** The database is initialized and connected and there exists no user

**Input:** None

**Output:** None
3.6.3 User input validation tests

**UT-150**  
**UsernameRequestTest\testUsernameCorrect()**
Tests the uniqueness check for a username.

- **Precondition:** No user exists with the given username
- **Input:** A valid username
- **Output:** A successful Response object

**UT-151**  
**UsernameRequestTest\testUsernameTooLong()**
Tests the uniqueness check for an invalid username.

- **Precondition:** The given username is too long
- **Input:** A username which is longer than 64 characters
- **Output:** An error Response object indicating something went wrong

**UT-152**  
**UsernameRequestTest\testUsernameUnique()**
Tests the uniqueness check for a non-unique username.

- **Precondition:** There exists a user with the given username
- **Input:** A valid username
- **Output:** An error Response object indicating that a user with the given username already exists

3.6.4 User account information tests

**UT-153**  
**AccountInformationRequestTest\testUserNotInDatabaseAllInformation()**
Tests the retrieval of all information for a non-existing user.

- **Precondition:** The user with the given id does not exist in the database
- **Input:** A valid user id
- **Output:** An empty array

**UT-154**  
**AccountInformationRequestTest\testGetAllInformation()**
Tests the retrieval of all information for a user.

- **Precondition:** The user with the given id does not exist in the database
- **Input:** A valid user id
- **Output:** Array containing both the user and the profile information of the user
3.6.5 Edit user account information tests

UT-155 EditAccountRequestTest\testEditProfileInformationNonExistingUser()
Tests the modification of a user profile for a non-existing user.

Test subject: editProfileInformation(profile:Object)
Precondition: The database is initialized and connected, the user with the given user id does not exist in the database.
Input: A user profile containing a non-existing user id
Output: A successful Response object

UT-156 EditAccountRequestTest\testEditProfileInformation()
Tests the modification of a user profile.

Test subject: editProfileInformation(profile:Object)
Precondition: The database is initialized and connected, the user with the given user id exists in the database.
Input: A user profile containing a user id
Output: A successful Response object and the profile of the user is updated

UT-157 EditAccountRequestTest\testEditAccountInformationInvalidEmail()
Tests the modification of the account information of a user with an invalid email.

Test subject: editAccountInformation(user:User)
Precondition: The user with the given id exists in the database.
Input: User object containing invalid user information
Output: An error Response object indicating that the email is invalid

UT-158 EditAccountRequestTest\testEditAccountInformation()
Tests the modification of the account information of a user.

Test subject: editAccountInformation(user:User)
Precondition: The user with the given id exists in the database.
Input: User object containing valid new user information
Output: A successful Response object and the account information of the user is updated

UT-159 EditAccountRequestTest\testEditAccountInformationEmailExists()
Tests the modification of the account information of a user with an email that is already associated with another user.

Test subject: editAccountInformation(user:User)
Precondition: The user with the given id exists in the database and there exists a user with the new email.
Input: User object containing valid new user information
Output: An error Response object indicating that there already exists a user with the given email.
3.6.6 Change user password tests

UT-160 EditPasswordRequestTest\testNonExistingUser()
Tests the modification of the password of a non-existing user.
Test subject: \requestEditPassword(userId:int, oldPassword:string, newPassword:string)
Precondition: The user with the given id does not exist in the database
Input: A user id, the old password and a new password
Output: An error Response object indicating that the user does not exist

UT-161 EditPasswordRequestTest\testWrongOldPassword()
Tests the modification of the password of a user when the old password is incorrect
Test subject: \requestEditPassword(userId:int, oldPassword:string, newPassword:string)
Precondition: The user with the given user id exists in the database and has a password that is different from the given password
Input: A user id, a incorrect old password and a new password
Output: An error Response object indicating that the old password was incorrect

UT-162 EditPasswordRequestTest\testInvalidNewPassword()
Tests the modification of the password of a user with an invalid new password
Test subject: \requestEditPassword(userId:int, oldPassword:string, newPassword:string)
Precondition: The user with the given user id exists in the database
Input: A user id, the old password and an invalid new password
Output: An error Response object indicating the new password is invalid

UT-163 EditPasswordRequestTest\testValidPasswordEdit()
Tests the modification of the password of a user.
Test subject: \requestEditPassword(userId:int, oldPassword:string, newPassword:string)
Precondition: The user with the given user id exists in the database
Input: A user id, the old password and a new password
Output: A successful Response object and the password of the user is updated

3.6.7 User deletion tests

UT-164 DeleteUserRequestTest\testRequestDeleteUserNotActiveUser()
Tests the deletion of a user when that user is not logged in.
Test subject: \requestDeleteUser(userId:int)
Precondition: The database is initialized, the user associated with the given id is not logged in
Input: A user id
Output: An error Response object indicating the user is not logged in
Tests the deletion of a user when there does not exist an administrator.

**Test subject:** `requestDeleteUser(userId : int )`

**Precondition:** The database is initialized, the user associated with the given id is logged in and there does not exist an administrator

**Input:** A user id

**Output:** An error Response object indicating that no new owner could be found

### 3.6.8 Ownership transfer tests

Tests the retrieval of a new owner for a project.

**Test subject:** `getNewOwner()`

**Precondition:** The database is initialized and there exists an administrator.

**Input:** None

**Output:** The user object of the admin with the oldest account

Tests the retrieval of a new owner for a project when no administrators exist.

**Test subject:** `getNewOwner()`

**Precondition:** The database is initialized and there does not exist an administrator.

**Input:** None

**Output:** Null

Tests the retrieval of a new owner when given a project.

**Test subject:** `getNewOwner()`

**Precondition:** The database is initialized and there exists an administrator

**Input:** A project

**Output:** The user object of the admin with the oldest account

Tests the transferring of project ownership.

**Test subject:** `transferAllOwnership(from : User, to : User )`

**Precondition:** The database is initialized

**Input:** The user to transfer all projects from, the user to transfer all projects to

**Output:** All projects have their owner changed from the first user to the second user

Tests the transferring of project ownership on user deletion.

**Test subject:** `requestDeleteUser(userId : int )`

**Precondition:** The database is initialized, the user associated with the given id is logged in

**Input:** A user id

**Output:** A successful Response object and all projects have their owner changed from the user with the given id to the oldest admin
3.7 Administrator tests

In this section, the unit tests that test administrator specific functionality are described. In particular, tests that handle the deletion of user accounts by an administrator are described.

3.7.1 Basic administrator tests

**UT-171** \(UserTest\).testUserIsAdministrator()

Tests if a user is an administrator.

**Test subject:** \(User.isAdministrator()\)

**Precondition:** There exists a User object with a valid id, a valid username, a valid email and a valid administrator status

**Input:** None

**Output:** The administrator status of the User object

3.7.2 Administrator database queries tests

**UT-172** \(FetchAllUsersRequestTest\).testNoAdmin()

Tests the retrieval of all users when the logged in user is not an administrator.

**Test subject:** \(fetchAllUsers()\)

**Precondition:** The logged in user is not an administrator

**Input:** None

**Output:** An error Response object indicating that the logged in user is not an administrator

**UT-173** \(FetchAllUsersRequestTest\).testAdminWithUsers()

Tests the retrieval of all users when the logged in user is an administrator.

**Test subject:** \(fetchAllUsers()\)

**Precondition:** The logged in user is an administrator

**Input:** None

**Output:** A successful Response object with an array of users as data

**UT-174** \(UserQueriesTest\).testUserAdmins()

Tests the retrieval of all admins.

**Test subject:** \(query\admins()\)

**Precondition:** The database is initialized and connected

**Input:** None

**Output:** The objects of all users with admin status.

**UT-175** \(FetchProjectsRequestTest\).testAdminAllProjects()

Tests the retrieval of projects when the user is an administrator.

**Test subject:** \(requestProjects(user: User)\)

**Precondition:** The given user exists and is an administrator

**Input:** A user

**Output:** An array with all the projects in the database
3.7.3 Deleting user accounts

**UT-176** DeleteUserByAdminRequestTest\testDeleteNotAdmin()

Tests the deletion of a user when the logged in user is not an administrator.

**Test subject:** `deleteUserByAdmin(userId : int)`

**Precondition:** There exists a user associated with the input user id and the logged in user is not an administrator.

**Input:** A valid user id

**Output:** False, indicating that the user does not have permission

**UT-177** DeleteUserByAdminRequestTest\testDeleteNoUser()

Tests the deletion of a non-existing user.

**Test subject:** `deleteUserByAdmin(userId : int)`

**Precondition:** There does not exist a user with the given user id.

**Input:** A user id

**Output:** False, indicating that the user does not exist

**UT-178** DeleteUserByAdminRequestTest\testDeleteUserByAdmin()

Tests the deletion of a user.

**Test subject:** `deleteUserByAdmin(userId : int)`

**Precondition:** There exists a user with the associated user id, there exists a project associated to that user and the logged in user is an administrator.

**Input:** A valid user id

**Output:** A successful Response object and the user is deleted

3.8 Project tests

In this section, unit tests with respects to projects are described. Unit tests regarding database queries are described, as well as unit tests regarding project creation and deletion, and project input validation.

3.8.1 Basic project tests

**UT-179** ProjectTest\testProjectGetId()

Tests the retrieval of a project’s id.

**Test subject:** `Project\getId()`

**Precondition:** There exists a Project object with a valid id, a valid name, a valid owner, and a valid creation date.

**Input:** None

**Output:** The id of the Project object
Tests the retrieval of a project's name.

**Test subject:** Project\getName()

**Precondition:** There exists a Project object with a valid id, a valid name, a valid owner, and a valid creation date

**Input:** None

**Output:** The name of the Project object

Tests the retrieval of a project's owner.

**Test subject:** Project\getOwner()

**Precondition:** There exists a Project object with a valid id, a valid name, a valid owner, and a valid creation date

**Input:** None

**Output:** The owner of the Project object

Tests the updating of a project's name.

**Test subject:** Project\update(newProjectName : string)

**Precondition:** There exists a Project object with a valid id, a valid name, a valid owner, and a valid creation date

**Input:** A valid project name

**Output:** The updated project name

Tests the deletion of a project.

**Test subject:** Project\delete()

**Precondition:** There exists a Project object with a valid id, a valid name, a valid owner, and a valid creation date

**Input:** None

**Output:** Null

Tests the textual representation of a project object.

**Test subject:** Project\_toString()

**Precondition:** There exists a Project object with a valid id, a valid name, a valid owner, and a valid creation date

**Input:** None

**Output:** The Project object transformed to a string format

Tests the retrieval of a project's legacy name.

**Test subject:** Project\legacyProjectName()

**Precondition:** There exists a Project object with a valid id, a valid name, a valid owner, and a valid creation date

**Input:** None

**Output:** The string 'project' appended with the project id
Tests the retrieval of a project's folder name.

**Test subject:** \Project\folderName()

**Precondition:** There exists a Project object with a valid id, a valid name, a valid owner, and a valid creation date

**Input:** None

**Output:** The string 'project' appended with the project id

Tests the retrieval of the name of a petrinet corresponding to a project.

**Test subject:** \Project\petriNetFileName()

**Precondition:** There exists a Project object with a valid id, a valid name, a valid owner, and a valid creation date

**Input:** None

**Output:** The string 'project' appended with the project id and '_petriNet.pnml'

Tests the retrieval of the name of a log file corresponding to a project.

**Test subject:** \Project\eventLogFileName()

**Precondition:** There exists a Project object with a valid id, a valid name, a valid owner, and a valid creation date

**Input:** None

**Output:** The string 'project' appended with the project id and '.xes'

Tests the retrieval of a project's creation date.

**Test subject:** \Project\getCreationDate()

**Precondition:** There exists a Project object with a valid id, a valid name, a valid owner, and a valid creation date

**Input:** None

**Output:** The creation date of the Project object

Tests the retrieval of all experiments of a project.

**Test subject:** \Project\getExperiments()

**Precondition:** There exists a Project object with a valid id, a valid name, a valid owner, and a valid creation date

**Input:** None

**Output:** An array containing all experiments associated with the given project

Tests the key-value representation of a project object.

**Test subject:** \Project\toArray()

**Precondition:** There exists a Project object with a valid id, a valid name, a valid owner, and a valid creation date

**Input:** None

**Output:** An array containing all the valid fields of the given Project object
Tests the key-value representation of a project object.

**Test subject:** `Project.toArray()`  
**Precondition:** There exists a Project object with a valid id, a valid name, a valid owner, and a valid creation date  
**Input:** None  
**Output:** An array containing all the valid fields of the given Project object

Tests the deletion of a project’s files.

**Test subject:** `Project.deleteFiles()`  
**Precondition:** None  
**Input:** None  
**Output:** All the files associated to this project are deleted

### 3.8.2 Project database queries tests

Tests the retrieval of a project by id.

**Test subject:** `query/projectById(id:int)`  
**Precondition:** The database is initialized and connected and a project with the given id exists  
**Input:** A valid project id  
**Output:** A project object with the given project id

Tests the retrieval of a non-existing project by id.

**Test subject:** `query/projectById(id:int)`  
**Precondition:** The database is initialized and connected and no project with the given id exists  
**Input:** A project id  
**Output:** Null

Tests the retrieval of a project by name.

**Test subject:** `query/projectByName(name:string)`  
**Precondition:** The database is initialized and connected and a project with the given name exists  
**Input:** A valid project name  
**Output:** A project object with the given project name

Tests the retrieval of a non-existing project by name.

**Test subject:** `query/projectByName(name:string)`  
**Precondition:** The database is initialized and connected and no project with the given name exists  
**Input:** A project name  
**Output:** Null
Tests the retrieval of all projects for which the user is an owner.

Test subject: \`\query\projectsByOwner(owner: \User)\`

Precondition: The database is initialized and connected and the given user exists and the user owns projects

Input: A valid user

Output: The projects owned by the user

Tests the retrieval of all projects for a non-existing user.

Test subject: \`\query\projectsByOwner(owner: \User)\`

Precondition: The database is initialized and connected and the given user does not exist

Input: A non-existing user

Output: An empty array

Tests the modification of a project name.

Test subject: \`\query\updateProject(project: \Project)\`

Precondition: The database is initialized and connected, a project with the given id exists and the given user exists

Input: A new project name

Output: A project object which contains the updated name

Tests the deletion of a project from the database.

Test subject: \`\query\deleteProject(project: \Project)\`

Precondition: The database is initialized and connected and the given project exists

Input: A project object

Output: The project is removed from the database, fetching the deleted project’s id returns null

Tests the modification of the project owner.

Test subject: \`\query\updateProjectOwner(project: \Project, user: \User)\`

Precondition: The database is initialized and connected and the given project exists and the project has a different owner than the given user

Input: A project object and a user object

Output: The updated project with its new owner

Tests the retrieval of all projects.

Test subject: \`\query\allProjects()\`

Precondition: The database is initialized and connected

Input: None

Output: All projects from the database
UT-204 FetchProjectsRequestTest\testUserNoProjects()

Tests the retrieval of projects for a user which does not have access to any projects.

**Test subject:** `requestProjects(user: User)`

**Precondition:** The given user exists but does not have access to any projects

**Input:** A user

**Output:** An empty array

UT-205 FetchProjectsRequestTest\testUserOneProject()

Tests the retrieval of projects for a user which has access to only one project.

**Test subject:** `requestProjects(user: User)`

**Precondition:** The given user exists and only has access to one project

**Input:** A user

**Output:** An array with exactly the one project the given user has access to

UT-206 FetchProjectsRequestTest\testUserMultipleProjects()

Tests the retrieval of projects for a user.

**Test subject:** `requestProjects(user: User)`

**Precondition:** The given user exists and has access to multiple projects

**Input:** A user

**Output:** An array with the projects the user has access to

3.8.3 Project creation tests

UT-207 ProjectCreationRequestTest\testProjectNameEmpty()

Tests the validation of project creation when an empty project name was given.

**Test subject:** `validateInput(projectName: string, eventLog: string, processModel: string, response: Response)`

**Precondition:** None

**Input:** An empty project name, an event log name, a process model name and a Response object

**Output:** False

UT-208 ProjectCreationRequestTest\testDataValidationProjectNameTooLong()

Tests the validation of project creation when a project name which is too long was given.

**Test subject:** `validateInput(projectName: string, eventLog: string, processModel: string, response: Response)`

**Precondition:** There does not exist a project with the given name

**Input:** A project name over 64 characters, an event log name, a process model name and a Response object

**Output:** False
**UT-209** ProjectCreationRequestTest\testDataValidationEventLogEmpty()

Tests the validation of project creation when an empty event log filename was given.

**Test subject:** `validateInput(projectName : string, eventLog : string, processModel : string, response : Response)`

**Precondition:** There does not exist a project with the given name

**Input:** A project name, an empty event log name, a process model name and a Response object

**Output:** False

**UT-210** ProjectCreationRequestTest\testDataValidationEventLogNoExtension()

Tests the validation of project creation when an event log without extension was given.

**Test subject:** `validateInput(projectName : string, eventLog : string, processModel : string, response : Response)`

**Precondition:** There does not exist a project with the given name

**Input:** A project name, an event log name without extension, a process model name and a Response object

**Output:** False

**UT-211** ProjectCreationRequestTest\testDataValidationEventLogWrongExtension()

Tests the validation of project creation when an event log with an incorrect extension was given.

**Test subject:** `validateInput(projectName : string, eventLog : string, processModel : string, response : Response)`

**Precondition:** There does not exist a project with the given name

**Input:** A project name, an event log name with an incorrect extension, a process model name and a Response object

**Output:** False

**UT-212** ProjectCreationRequestTest\testDataValidationProcessModelEmpty()

Tests the validation of project creation when an empty process model filename was given.

**Test subject:** `validateInput(projectName : string, eventLog : string, processModel : string, response : Response)`

**Precondition:** There does not exist a project with the given name

**Input:** A project name, an event log name, an empty process model name and a Response object

**Output:** False

**UT-213** ProjectCreationRequestTest\testDataValidationProcessModelNoExtension()

Tests the validation of project creation when a process model with no extension was given.

**Test subject:** `validateInput(projectName : string, eventLog : string, processModel : string, response : Response)`

**Precondition:** There does not exist a project with the given name

**Input:** A project name, an event log name, a process model name without an extension and a Response object

**Output:** False
Tests the validation of project creation when a process model with an incorrect extension was given.

Test subject: \validateInput(projectName : string, eventLog : string, processModel : string, response : Response )
Precondition: There does not exist a project with the given name
Input: A project name, an event log name, a process model name with an incorrect extension and a Response object
Output: False

Tests the validation of project creation.

Test subject: \validateInput(projectName : string, eventLog : string, processModel : string, response : Response )
Precondition: There does not exist a project with the given name
Input: A project name, an event log name, a process model name and a Response object
Output: True

Tests the validation of project creation when there already exists a project with the given project name.

Test subject: \validateInput(projectName : string, eventLog : string, processModel : string, response : Response )
Precondition: There exists a project with the given name
Input: A project name, an event log name, a process model name and a Response object
Output: False

Tests whether an exception is thrown when a database error is encountered.

Test subject: \createNewProject(projectName : string, response : Response )
Precondition: There exists a project with the given name
Input: A project name and a Response object
Output: SQLException

Tests the creation of a project when given an empty project name.

Test subject: \createProjectRequest(projectName : string, eventLog : string, processModel : string )
Precondition: None
Input: An empty project name, an event log name and a process model name
Output: An error Response object indicating that the project name is empty

Tests the retrieval of the event log filename.

Test subject: \getEventLogName(project : Project )
Precondition: The given project exists
Input: A project
Output: The name of the event log associated with the given project
Tests the creation of a project folder when the project does not contain the correct files.

**Test subject:** `createProjectFolder(project: Project)`  
**Precondition:** The given project does not contain the correct files  
**Input:** A project  
**Output:** False

Tests the uploading of an event log when the project contains no files.

**Test subject:** `uploadEventLog(project: Project, projectFolder: string)`  
**Precondition:** The given project contains no files  
**Input:** A project and a folder name  
**Output:** False

Tests the uploading of a process model when the project contains no files.

**Test subject:** `uploadProcessModel(project: Project, projectFolder: string)`  
**Precondition:** The given project contains no files  
**Input:** A project and a folder name  
**Output:** False

Tests the creation of a project when no event log was uploaded.

**Test subject:** `createNewProject(projectName: string, response: Response)`  
**Precondition:** No event log was uploaded  
**Input:** A project name and a Response object  
**Output:** False
Tests the creation of a project folder when no process model was uploaded.

**Test subject:** `createProjectFolder(project: Project)`

**Precondition:** The given project object does not contain a process model

**Input:** A project

**Output:** False

Tests the creation of a project folder.

**Test subject:** `createProjectFolder(project: Project)`

**Precondition:** The project has an event log and process model

**Input:** A project

**Output:** True

Tests the creation of a project

**Test subject:** `createNewProject(projectName: string, response: Response)`

**Precondition:** A process model and event log were uploaded

**Input:** A project name and a Response object

**Output:** True

Tests the creation of a project when no files were uploaded.

**Test subject:** `createProjectRequest(projectName: string, eventLog: string, processModel: string)`

**Precondition:** No files were uploaded with the given names

**Input:** A project name, an event log name and a process model name

**Output:** An error Response object indicating there was an error during folder creation

Tests the creation of a project

**Test subject:** `createProjectRequest(projectName: string, eventLog: string, processModel: string)`

**Precondition:** An event log and process model were uploaded

**Input:** A project name, an event log name and a process model name

**Output:** A successful Response object and the project is created

Tests the creation of a 'files' directory.

**Test subject:** `setupProjectDirectory(projectFolder: string)`

**Precondition:** None

**Input:** The path to a project folder

**Output:** True
3.8.4  Project input validation tests

UT-232  ProjectNameValidationTest\testProjectNameTooLong()
Tests the validation of a project name.
**Test subject:** `validateProjectName(projectName: string, response: Response)`
**Precondition:** None
**Input:** A project name that is longer than 64 characters and a new Response object
**Output:** False

UT-233  ProjectNameValidationTest\testProjectNameEmpty()
Tests the validation of a project name.
**Test subject:** `validateProjectName(projectName: string, response: Response)`
**Precondition:** None
**Input:** An empty project name and a new Response object
**Output:** False

UT-234  ProjectNameValidationTest\testProjectNameCorrect()
Tests the validation of a project name.
**Test subject:** `validateProjectName(projectName: string, response: Response)`
**Precondition:** None
**Input:** A valid project name and a new Response object
**Output:** True

UT-235  ProjectNameValidationTest\testProjectNameExists()
Tests the validation of a project name.
**Test subject:** `validateProjectName(projectName: string, response: Response)`
**Precondition:** There exists a project with the given project name
**Input:** A valid project name and a new Response object
**Output:** False

3.8.5  Project deletion tests

UT-236  ProjectDeletionRequestTest\testValidateOwner()
Test the deletion of a non-existing project
**Test subject:** `createProjectDeletionRequest(projectId: int)`
**Precondition:** there does not exist a project with the given id
**Input:** A project id
**Output:** An error Response object

UT-237  ProjectDeletionRequestTest\testDeleteProject()
Tests the deletion of a project.
**Test subject:** `createProjectDeletionRequest(projectId: int)`
**Precondition:** The logged in user owns a project with the given id
**Input:** A project id
**Output:** A successful Response object, all project files are deleted and all files of the experiments of the project are deleted
3.9 Experiment tests

In this section, descriptions of unit tests are included that test the experiment related methods. These include the methods related to the experiment object-relational mapping (ORM) model, database queries, experiment creation and deletion, experiment status related methods, experiment file download requests and experiment file template model, which describes all files related to a given experiment.

3.9.1 Basic experiment tests

**UT-238** ExperimentTest\testExperimentById()

Tests the retrieval of an experiment’s id.

**Test subject:** \Experiment\getId()

**Precondition:** There exists an Experiment object with a valid id, a valid name, a valid project, a valid creation and a valid creation date

**Input:** None

**Output:** The id of the Experiment object

**UT-239** ExperimentTest\testExperimentGetName()

Tests the retrieval of an experiment’s name.

**Test subject:** \Experiment\getName()

**Precondition:** There exists an Experiment object with a valid id, a valid name, a valid project, a valid creation and a valid creation date

**Input:** None

**Output:** The experiment name of the Experiment object

**UT-240** ExperimentTest\testGetProject()

Tests the retrieval of the project of an experiment.

**Test subject:** \Experiment\getProject()

**Precondition:** There exists an Experiment object with a valid id, a valid name, a valid project, a valid creation and a valid creation date and there exists a project associated with the given Experiment object

**Input:** None

**Output:** The Project object associated with the Experiment object

**UT-241** ExperimentTest\testGetCreator()

Tests the retrieval of an experiment’s creator.

**Test subject:** \Experiment\getCreator()

**Precondition:** There exists an Experiment object with a valid id, a valid name, a valid project, a valid creation and a valid creation date and there exists a user associated with the given Experiment object

**Input:** None

**Output:** The User object associated with the Experiment object
**UT-242**  
**ExperimentTest** \testGetExperimentCreationDate()  
Tests the retrieval of an experiment’s creation date.  
**Test subject:** \Experiment\getCreationDate()  
**Precondition:** There exists an Experiment object with a valid id, a valid name, a valid project, a valid creation and a valid creation date  
**Input:** None  
**Output:** The creation date of the Experiment object

**UT-243**  
**ExperimentTest** \testExperimentFolderName()  
Tests the retrieval of an experiment’s folder name.  
**Test subject:** \Experiment\folderName()  
**Precondition:** There exists an Experiment object with a valid id, a valid name, a valid project, a valid creation and a valid creation date  
**Input:** None  
**Output:** The folder name of the Experiment object

**UT-244**  
**ExperimentTest** \testExperimentTimestampsFileName()  
Tests the retrieval of the name of the timestamps file of an experiment.  
**Test subject:** \Experiment\timestampsFileName()  
**Precondition:** There exists an Experiment object with a valid id, a valid name, a valid project, a valid creation and a valid creation date  
**Input:** None  
**Output:** The file name of the timestamps file of the experiment

**UT-245**  
**ExperimentTest** \testGetExperimentString()  
Tests the textual representation of an experiment object.  
**Test subject:** \Experiment\__toString()  
**Precondition:** There exists an Experiment object with a valid id, a valid name, a valid project, a valid creation and a valid creation date  
**Input:** None  
**Output:** A string starting with 'Experiment' appended with the id, name, creator and project of the experiment

**UT-246**  
**ExperimentTest** \testToArrayNoStatus()  
Tests the key-value representation of an experiment object where no phases have been started on the experiment.  
**Test subject:** \Experiment\toArray()  
**Precondition:** There exists an Experiment object with a valid id, a valid name, a valid project, a valid creation and a valid creation date  
**Input:** None  
**Output:** An array containing all valid fields of the given Experiment object
Tests the key-value representation of an experiment object where some phases of the experiment have been completed.

**Test subject:** \Experiment\toArray()

**Precondition:** There exists an Experiment object with a valid id, a valid name, a valid project, a valid creation and a valid creation date, on which several phases have been run

**Input:** None

**Output:** An array containing all valid fields of the given Experiment object

Tests the retrieval of the anomalous subgraph discovery phase status on an experiment where the anomalous subgraph discovery phase has not been started yet.

**Test subject:** \Experiment\getLatestAnomalousSubgraphDiscoveryStatusEntry()

**Precondition:** The database is initialized and connected and there does not exist a status for anomalous subgraph discovery

**Input:** None

**Output:** Null

Tests the retrieval of the partial order discovery phase status on an experiment where the partial order discovery phase has not been started yet.

**Test subject:** \Experiment\getLatestPartialOrderDiscoveryStatusEntry()

**Precondition:** The database is initialized and connected and there does not exists a status for anomalous partial order

**Input:** None

**Output:** Null

Tests the retrieval of the anomalous subgraph discovery phase status on an experiment.

**Test subject:** \Experiment\getLatestAnomalousSubgraphDiscoveryStatusEntry()

**Precondition:** The database is initialized and connected and there exists a status for anomalous subgraph discovery

**Input:** None

**Output:** The latest status for anomalous subgraph discovery

Tests the retrieval of the partial order discovery phase status on an experiment.

**Test subject:** \Experiment\getLatestPartialOrderDiscoveryStatusEntry()

**Precondition:** The database is initialized and connected and there exists a status for partial order discovery

**Input:** None

**Output:** The latest status for partial order discovery
3.9.2  Experiment database queries tests

**UT-252  ExperimentQueriesTest\testExperimentsByProjectValid()**

Tests the retrieval of all experiments for a project.

**Test subject:** \query\experimentsByProject(project : Project )

**Precondition:** The database is initialized and connected and the given project exists

**Input:** A valid project

**Output:** An array of all experiments for the given project

**UT-253  ExperimentQueriesTest\testExperimentsByProjectInvalid()**

Tests the retrieval of all experiments for a non-existing project.

**Test subject:** \query\experimentsByProject(project : Project )

**Precondition:** The database is initialized and connected and the given project does not exist

**Input:** A non-existing project

**Output:** An empty array

**UT-254  ExperimentQueriesTest\testExperimentsByOwnerValid()**

Tests the retrieval of all experiments for which a user is owner.

**Test subject:** \query\experimentsByOwner(owner : User )

**Precondition:** The database is initialized and connected and the given user exists

**Input:** A valid user, which is the owner of an experiment

**Output:** An array of all experiments that are owned by the given user

**UT-255  ExperimentQueriesTest\testExperimentsByOwnerInvalid()**

Tests the retrieval of all experiments for a non-existing user.

**Test subject:** \query\experimentsByOwner(owner : User )

**Precondition:** The database is initialized and connected and the given user does not exist

**Input:** A non-existing user, which is the owner of an experiment

**Output:** An empty array

**UT-256  ExperimentQueriesTest\testDeleteExperiment()**

Tests the deletion of an experiment from the database.

**Test subject:** \query\deleteExperiment(experiment : Experiment )

**Precondition:** The database is initialized and connected and the given experiment exists

**Input:** A valid experiment

**Output:** Null when trying to fetch the experiment after removal

**UT-257  ExperimentQueriesTest\testAddStatus()**

Tests the addition of a status to an experiment.

**Test subject:** \query\addStatus(phaseId : int, statusCode : int, experiment : Experiment )

**Precondition:** The database is initialized and connected and the given experiment exists

**Input:** A valid experiment and a StatusCode

**Output:** A valid StatusEntry object when fetching the status after adding the status
Tests whether an exception is thrown when trying to add a status to a non-existing experiment.

**Test subject:** `\query\addStatus(phaseId : int, statusCode : int, experiment : Experiment)

**Precondition:** The database is initialized and connected and the given experiment does not exist

**Input:** A non-existing experiment and valid StatusCode

**Output:** An SQLException is thrown

Tests the retrieval of an invalid status.

**Test subject:** `\query\statusEntryById(id : int)

**Precondition:** The database is initialized and connected

**Input:** An invalid status entry id

**Output:** Null

Tests the retrieval of a non-existing experiment.

**Test subject:** `\query\experimentById(id : int)

**Precondition:** The database is initialized and connected and no experiment with the given id exists

**Input:** a non-existing experiment id

**Output:** Null

Tests whether an exception is thrown when given a non-existing project while creating a new experiment.

**Test subject:** `\query\newExperiment(name : string, project : Project, creator : User)

**Precondition:** The database is initialized and connected and the given project does not exist

**Input:** An experiment name, a non-existing project and a valid user

**Output:** An SQLException is thrown

Tests whether an exception is thrown when given a non-existing user while creating a new experiment.

**Test subject:** `\query\newExperiment(name : string, project : Project, creator : User)

**Precondition:** The database is initialized and connected and the given user does not exist

**Input:** An experiment name, a valid project and a non-existing user

**Output:** An SQLException is thrown

Tests the retrieval of the current status for a non-existing experiment.

**Test subject:** `\query\experimentCurrentStatus(experiment : Experiment)

**Precondition:** The database is initialized and connected and the given experiment does not exist

**Input:** A non-existing experiment

**Output:** Null
UT-264  ExperimentQueriesTest\testAddStatusInvalidStatusCode()

Tests whether an exception is thrown when given an invalid status while adding a status to an experiment.

Test subject: \query\addStatus(phaseId : int, statusCode : int, experiment : Experiment)
Precondition: The database is initialized and connected and the given experiment exists
Input: A valid phase id, an invalid status code and a valid experiment
Output: An SQLException is thrown

UT-265  ExperimentQueriesTest\testAddStatusInvalidPhaseId()

Tests whether an exception is thrown when given an invalid phase while adding a status to an experiment.

Test subject: \query\addStatus(phaseId : int, statusCode : int,experiment : Experiment)
Precondition: The database is initialized and connected and the given experiment exists
Input: An invalid phase id, a valid status code and a valid experiment
Output: An SQLException is thrown

UT-266  ExperimentQueriesTest\testAddStatusInvalidExperiment()

Tests whether an exception is thrown when adding a status to a non-existing experiment.

Test subject: \query\addStatus(phaseId : int, statusCode : int,experiment : Experiment)
Precondition: The database is initialized and connected and the given experiment does not exist
Input: A valid phase id, a valid status code and a non-existing experiment
Output: An SQLException is thrown

UT-267  ExperimentQueriesTest\testGetDescriptionInvalidProjectPhase()

Tests the retrieval of the status description for an invalid status.

Test subject: \query\experimentStatusDescription(statusCode : int)
Precondition: The database is initialized and connected
Input: An invalid status code id
Output: Null

UT-268  ExperimentQueriesTest\testChangeExperimentCreator()

Tests the modification of the owner of an experiment.

Test subject: \query\updateExperimentCreator(experiment : Experiment, user : User)
Precondition: The database is initialized and connected and the given experiment exists and the experiment has a different creator than the user which is used as input
Input: An experiment object and a user object
Output: The updated experiment with its new creator

UT-269  ExperimentQueriesTest\testExperimentPhaseName()

Tests the retrieval of the name for a phase.

Test subject: \query\experimentPhaseName(phaseId : int)
Precondition: The database is initialized and connected and the experiment_phase table is populated
Input: A valid phase id
Output: The name of the phase
Tests the retrieval of the latest status of an experiment.

Test subject: \query\experimentLatestStatus(experiment : Experiment, phaseId : int)
Precondition: The database is initialized and connected and there does not exist a status for the given phase
Input: A valid phase id and a valid experiment
Output: Null

Tests the retrieval of the latest status of an experiment.

Test subject: \query\experimentLatestStatus(experiment : Experiment, phaseId : int)
Precondition: The database is initialized and connected and there exists a status for the given phase
Input: A valid phase id and a valid experiment
Output: The latest status for the given phase

Tests the retrieval of experiments for a non-existing project.

Test subject: \fetchExperiments(projectId : int)
Precondition: There does not exist a project with the given id
Input: A valid project id
Output: An empty array

Tests the retrieval of experiments for a project.

Test subject: \fetchExperiments(projectId : int)
Precondition: There exists a project with the given id and this project contains experiments
Input: A valid project id
Output: An array containing the experiments of the project

3.9.3 Experiment creation tests

Tests the creation of an experiment.

Test subject: \createNewExperiment(experimentName : string, projectId : int)
Precondition: There exists a project with the given id
Input: A valid experiment name and a project id
Output: True

Tests the creation of an experiment when given a non-existing project.

Test subject: \createNewExperiment(experimentName : string, projectId : int)
Precondition: There does not exist a project with the given id
Input: A valid experiment name and a project id
Output: False
Tests the creation of an experiment when given an empty name.

Test subject: `createNewExperiment(experimentName: string, projectId: int)`
Precondition: There exists a project with the given id
Input: A project id
Output: An SQLException indicating the query failed

Tests the creation of an experiment when given an invalid name.

Test subject: `createExperimentRequest(experimentName: string, projectId: int)`
Precondition: There exists an experiment with the given id
Input: An invalid experiment name and a project id
Output: An error Response object indicating the experiment name is invalid

Tests the creation of an experiment when given a non-existing project.

Test subject: `createNewExperiment(experimentName: string, projectId: int)`
Precondition: There does not exist a project with the given id
Input: A valid experiment name and a project id
Output: An error Response object indicating that the project does not exist

Tests whether createExperimentRequest() succeeds when given a valid experiment name and project id.

Test subject: `createExperimentRequest(experimentName: string, projectId: int)`
Precondition: There exists a project with the given id
Input: A valid experiment name, and project id
Output: True, indicating the experiment was created

Tests the validation of an experiment name with an invalid experiment name.

Test subject: `validateExperimentName(name: string, response: Response)`
Precondition: None
Input: A null string and a new Response object
Output: False

Tests the validation of an experiment name with an invalid experiment name.

Test subject: `validateExperimentName(name: string, response: Response)`
Precondition: None
Input: A empty experiment name and a new Response object
Output: False
UT-282 | ExperimentValidationTest\testExperimentNameTooLong()
---|---
Tests the validation of an experiment name with an invalid experiment name.
**Test subject:** \validateExperimentName(name: string, response: Response)
**Precondition:** None
**Input:** A string with a length larger than 64 characters and a new Response object
**Output:** False

UT-283 | ExperimentValidationTest\testExperimentNameCorrect()
---|---
Tests the validation of an experiment name.
**Test subject:** \validateExperimentName(name: string, response: Response)
**Precondition:** None
**Input:** A string that is a valid experiment name and a new Response object
**Output:** True

### 3.9.5 Experiment deletion tests

UT-284 | ExperimentTest\testDeleteFiles()
---|---
Tests the deletion of an experiment’s files.
**Test subject:** \Experiment\deleteFiles()
**Precondition:** None
**Input:** None
**Output:** All the files associated to this experiment are deleted

UT-285 | ExperimentTest\testDeleteTimestampsFile()
---|---
Tests the deletion of the status timestamps of an experiment.
**Test subject:** \Experiment\deleteTimestampsFile()
**Precondition:** There exists a timestamps file for the experiment
**Input:** None
**Output:** True, and the timestamps file of the experiment is deleted

UT-286 | ExperimentTest\testDeleteTimestampsFileFail()
---|---
Tests the deletion of the status timestamps of an experiment.
**Test subject:** \Experiment\deleteTimestampsFile()
**Precondition:** There does not exist a timestamps file for the experiment
**Input:** None
**Output:** False

UT-287 | ExperimentTest\testDeleteDatabase()
---|---
Tests the deletion of an experiment’s database.
**Test subject:** \Experiment\deleteDatabase()
**Precondition:** The database is initialised, there is an experiment database for the experiment
**Input:** None
**Output:** The database of the experiment is deleted
3.9.6 Retrieving and downloading experiment files tests

**UT-288** DownloadLogRequestTest\testWrongExtension()

Tests the downloading of a file when given an incorrect filename.

**Test subject:** DownloadFileTemplate(experimentId : int, template : string)

**Precondition:** The experiment exists

**Input:** An experiment id, an incorrect filename

**Output:** An error Response object indicating that the file does not exist

**UT-289** DownloadLogRequestTest\testCorrectFile()

Tests the downloading of a file.

**Test subject:** DownloadFileTemplate(experimentId : int, template : string)

**Precondition:** The experiment exists

**Input:** An experiment id, a correct filename

**Output:** A successful Response object

**UT-290** DownloadLogRequestTest\testNoExperiment()

Tests the downloading of a file when the experiment does not exist.

**Test subject:** DownloadFileTemplate(experimentId : int, template : string)

**Precondition:** There is no experiment with the given id

**Input:** An experiment id, a correct filename

**Output:** An error Response object indicating that the experiment does not exist

**UT-291** ExperimentFileTemplatesTest\testTemplateFromKey()

Tests the retrieval of an experiment file template with a valid key.

**Test subject:** ExperimentFileTemplates\templateFromKey(key : string)

**Precondition:** The key is an existing key

**Input:** A key

**Output:** The string for the key

**UT-292** ExperimentFileTemplatesTest\testTemplateFromKeyNonExistingKey()

Tests the retrieval of an experiment file template with an invalid key.

**Test subject:** ExperimentFileTemplates\templateFromKey(key : string)

**Precondition:** The key does not exist

**Input:** A key

**Output:** Null
3.9.7 Experiment status tests

**UT-293** StatusEntryTest\testGetExperiment()

Tests the retrieval of the experiment of a status entry.

**Test subject:** StatusEntry\getExperiment()

**Precondition:** There exists a StatusEntry object with a valid id, a valid experiment, a valid status and a valid creation date and there exists an experiment associated with the given StatusEntry object

**Input:** None

**Output:** The Experiment object associated with the StatusEntry object

**UT-294** StatusEntryTest\testGetStatus()

Tests the retrieval of a status of a status entry.

**Test subject:** StatusEntry\getStatus()

**Precondition:** There exists a StatusEntry object with a valid id, a valid experiment, a valid status and a valid creation date

**Input:** None

**Output:** The Status object associated with the StatusEntry object

**UT-295** StatusEntryTest\testGetStatusId()

Tests the retrieval of the id of a status entry.

**Test subject:** Status\getId()

**Precondition:** There exists a StatusEntry object with a valid id, a valid experiment, a valid status and a valid creation date

**Input:** None

**Output:** The id of the Status object

**UT-296** StatusEntryTest\testGetStatusCreationDate()

Tests the retrieval of the creation date of a status entry.

**Test subject:** StatusEntry\getCreationDate()

**Precondition:** There exists a StatusEntry object with a valid id, a valid experiment, a valid status and a valid creation date

**Input:** None

**Output:** The creation date of the StatusEntry object

**UT-297** StatusEntryTest\testGetDescription()

Tests the retrieval of the description of a status entry.

**Test subject:** Status\getDescription()

**Precondition:** There exists a Status object with a valid id and a valid description

**Input:** None

**Output:** The description of the Status object
Tests the textual representation of the status entry object.

**Test subject:** `StatusEntry::toString()`  
**Precondition:** There exists a StatusEntry object with a valid id, a valid experiment, a valid status and a valid creation date  
**Input:** None  
**Output:** The StatusEntry object transformed to a string format

Tests the retrieval of the project belonging to the experiment of a status entry.

**Test subject:** `Experiment::getProject()`  
**Precondition:** There exists an Experiment object with a valid id, a valid name, a valid project, a valid creation and a valid creation date  
**Input:** None  
**Output:** The Project object which contains this experiment

Tests the retrieval of the id of a status entry.

**Test subject:** `Status::getId()`  
**Precondition:** There exists a Status object with a valid id and a valid description  
**Input:** None  
**Output:** The id of the Status object

### 3.10 Experiment Phase tests

This section describes unit tests that test functionality of individual experiment phases. This includes the methods that are related to the phase object-relational mapping model, validation of the input of phases and the running of phases.

#### 3.10.1 Basic experiment phase tests

Tests the retrieval of a phase's id.

**Test subject:** `Phase::getId()`  
**Precondition:** There exists a Phase object with a valid id and a valid name  
**Input:** None  
**Output:** The id of the Phase object

Tests the retrieval of a phase's name.

**Test subject:** `Phase::getName()`  
**Precondition:** There exists a Phase object with a valid id and a valid name  
**Input:** None  
**Output:** The name of the Phase object
3.10.2 Experiment phase validation tests

**UT-303** PartialOrderDiscoveryRequestTest\testValidateExperimentInvalid()
Tests the validation of an experiment id for which no experiment exists
Test subject: validateExperimentId(id:int, response:Response)
Precondition: There does not exist an experiment with the given id
Input: An experiment id and a Response object
Output: False

**UT-304** PartialOrderDiscoveryRequestTest\testValidateExperimentValid()
Tests the validation of an experiment id
Test subject: validateExperimentId(id:int, response:Response)
Precondition: There exists an experiment with the given id
Input: An experiment id and a new Response object
Output: True

**UT-305** PartialOrderDiscoveryRequestTest\testValidationNoExperiment()
Tests the validation of the partial order input with an experiment id for which no experiment exists.
Test subject: validatePartialOrderInput(experimentId:int, orThreshold:float, uniFB01Threshold:float, response:Response)
Precondition: There does not exist an experiment with the given id
Input: An experiment id, a threshold, a support and a Response object
Output: False

**UT-306** PartialOrderDiscoveryRequestTest\testValidationValid()
Tests the validation of the partial order input.
Test subject: validatePartialOrderInput(experimentId:int, orThreshold:float, uniFB01Threshold:float, response:Response)
Precondition: There exists an experiment with the given id
Input: An experiment id, a threshold, a support and a Response object
Output: True

**UT-307** PartialOrderDiscoveryRequestTest\testValidationNoNumeric()
Tests the validation of the partial order input when given incorrect input.
Test subject: validatePartialOrderInput(experimentId:int, orThreshold:float, uniFB01Threshold:float, response:Response)
Precondition: There exists an experiment with the given id
Input: An experiment id, a threshold parameter, a support parameter and a Response object
Output: False

**UT-308** PartialOrderDiscoveryRequestTest\testRequestValidationNoExperiment()
Tests the procedure of starting partial order discovery when the given experiment does not exist.
Test subject: startPartialOrderDiscoveryRequest(experimentId:int, orThreshold:float, uniFB01Threshold:float)
Precondition: There does not exist and experiment with the given id
Input: An experiment id, a threshold and a support
Output: An error Response object indicating that the experiment does not exist
Tests the procedure of starting partial order discovery when given an invalid threshold parameter.

**Test subject:** 
\texttt{startPartialOrderDiscoveryRequest(experimentId: int, orThreshold: float, uniFB01Threshold: float)}

**Precondition:** There exists an experiment with the given id

**Input:** An experiment id, a non-numeric threshold and a support

**Output:** An error Response object indicating that the input is not numeric

Tests the procedure of starting partial order discovery when given an invalid support parameter

**Test subject:** 
\texttt{startPartialOrderDiscoveryRequest(experimentId: int, orThreshold: float, uniFB01Threshold: float)}

**Precondition:** There exists an experiment with the given id

**Input:** An experiment id, a threshold and a non-numeric support

**Output:** An error Response object indicating that the input is not numeric

Tests the procedure of starting partial order discovery.

**Test subject:** 
\texttt{startPartialOrderDiscoveryRequest(experimentId: int, orThreshold: float, uniFB01Threshold: float)}

**Precondition:** There exists an experiment with the given id

**Input:** An experiment id, a threshold and a support

**Output:** A successful Response object

Tests the procedure of starting partial order discovery when anomalous subgraph discovery has not been finished for the given experiment

**Test subject:** 
\texttt{startPartialOrderDiscoveryRequest(experimentId: int, orThreshold: float, uniFB01Threshold: float)}

**Precondition:** There exists an experiment with the given id and anomalous subgraph discovery is not finished for this experiment.

**Input:** An experiment id, a threshold and a support

**Output:** An error Response object indicating anomalous subgraph discovery is not finished for the given experiment

### 3.10.3 Experiment phase input validation tests

Tests the validation of a pattern type.

**Test subject:** 
\texttt{validatePatternType(pattern: string)}

**Precondition:** None

**Input:** A valid pattern type and a new Response object

**Output:** True
PartialOrderInputValidationTest\testPatternIncorrect()

Tests the validation of a pattern type.

**Test subject:** validatePatternType(pattern: string)

**Precondition:** None

**Input:** An invalid pattern type and a new Response object

**Output:** False

PartialOrderInputValidationTest\testPatternInput()

Tests the validation of partial order discovery parameters.

**Test subject:** validatePartialOrderInput(experimentId: int, orThreshold: float, fThreshold: float, response: Response)

**Precondition:** None

**Input:** Valid pattern type and support and a new Response object

**Output:** True

PartialOrderInputValidationTest\testPatternInputIncorrect()

Tests the validation of partial order discovery parameters.

**Test subject:** validatePartialOrderInput(experimentId: int, orThreshold: float, fThreshold: float, response: Response)

**Precondition:** None

**Input:** Invalid pattern type and support and a new Response object

**Output:** False

### 3.11 Experiment result tests

In this section, unit tests with respects to experiment results are described. Subgraph expansion is tested, as well as experiment result retrieval.

#### 3.11.1 Subgraph expansion tests

PartialOrderInputValidationTest\testNoExperiment()

Tests the retrieval of expanded subgraph results for a non-existing experiment

**Test subject:** getExpandedSubgraphResultRequest(experimentId: int, clusters: string[])

**Precondition:** There does not exist an experiment with the given id

**Input:** An experiment id

**Output:** An error Response object indicating the experiment does not exist

PartialOrderInputValidationTest\testGetResult()

Tests the retrieval of expanded subgraph results for an experiment

**Test subject:** getExpandedSubgraphResultRequest(experimentId: int, clusters: string[])

**Precondition:** There exists an experiment with the given id and the anomalous subgraph discovery has been run for this experiment

**Input:** An experiment id

**Output:** A successful Response object with an 'svg' as data
Tests the retrieval of expanded subgraph results for an experiment.

**Test subject:** `getExpandedSubgraphResultRequest(experimentId : int, clusters : string[])`

**Precondition:** There exists an experiment with the given id and the anomalous subgraph discovery has been run for this experiment.

**Input:** An experiment id and an array with clusters

**Output:** A successful Response object with an 'svg' as data

Tests the retrieval of expanded subgraph results for an experiment for which the anomalous subgraph discovery has not been run.

**Test subject:** `getExpandedSubgraphResultRequest(experimentId : int, clusters : string[])`

**Precondition:** There exists an experiment with the given id and the anomalous subgraph discovery has not been run for this experiment.

**Input:** An experiment id

**Output:** An error Response object indicating that no results for this experiment exist

### 3.11.2 Experiment results retrieval tests

Tests the retrieval of anomalous subgraph discovery results with a non-existing experiment.

**Test subject:** `getSubgraphResultRequest(experimentId : int)`

**Precondition:** There does not exist an experiment with the given id

**Input:** An experiment id

**Output:** An error Response object indicating no results were returned

Tests the retrieval of anomalous subgraph discovery results with an experiment which does not contain anomalous subgraph discovery results.

**Test subject:** `getSubgraphResultRequest(experimentId : int)`

**Precondition:** No anomalous subgraph discovery results exist for the given experiment id

**Input:** An experiment id

**Output:** An error Response object indicating no results were returned

Tests the retrieval of anomalous subgraph discovery results.

**Test subject:** `getSubgraphResultRequest(experimentId : int)`

**Precondition:** There exists an experiment with the given id, the experiment contains results for anomalous subgraph discovery

**Input:** An experiment id

**Output:** A successful Response object
PartialOrderResultRequestTest\testIncorrectExperimentId()

Tests the retrieval of partial order discovery results when given a non-existing experiment.

**Test subject:** `getPartialOrderResultRequest (experimentId : int, pattern : int, support : float, expandedNodes : string[])`

**Precondition:** There does not exist an experiment with this id

**Input:** An experiment id, a pattern parameter and a support parameter

**Output:** An error Response object indicating that the experiment does not exist

PartialOrderResultRequestTest\testNoResults()

Tests the retrieval of partial order discovery results when no results exist for the given experiment.

**Test subject:** `getPartialOrderResultRequest (experimentId : int, pattern : int, support : float, expandedNodes : string[])`

**Precondition:** There exists an experiment with the given id, the experiment has no results for partial order discovery

**Input:** An experiment id, a pattern parameter and a support parameter

**Output:** An error Response object indicating that no results exist for the given experiment

PartialOrderResultRequestTest\testResults()

Tests the retrieval of partial order discovery results.

**Test subject:** `getPartialOrderResultRequest (experimentId : int, pattern : int, support : float, expandedNodes : string[])`

**Precondition:** There exists an experiment with the given id and the experiment has results for partial order discovery

**Input:** An experiment id, a pattern parameter and a support parameter

**Output:** A successful Response object with an 'svg' as data

PartialOrderResultRequestTest\testInputWrongPattern()

Tests the retrieval of partial order discovery results when an invalid pattern type was given.

**Test subject:** `getPartialOrderResultRequest (experimentId : int, pattern : int, support : float, expandedNodes : string[])`

**Precondition:** None

**Input:** An experiment id, an invalid pattern type and a support parameter

**Output:** An error Response object indicating there was an input error

PartialOrderResultRequestTest\testInputWrongSupport()

Tests the retrieval of partial order discovery results when an invalid support parameter was given.

**Test subject:** `getPartialOrderResultRequest (experimentId : int, pattern : int, support : float, expandedNodes : string[])`

**Precondition:** None

**Input:** An experiment id, a pattern parameter and an invalid support parameter

**Output:** An error Response object indicating there was an input error

PartialSubgraphResultRequestTest\testNoExepriment()

Tests the retrieval of patterns for a non-existing experiment.

**Test subject:** `getPartialSubgraphResultRequest (experimentId : int, clusters : string[])`

**Precondition:** There does not exist an experiment with the given experiment id

**Input:** An experiment id

**Output:** An error Response object indicating the the experiment does not exist
PartialSubgraphResultRequestTest\testNoResult()

Tests the retrieval of patterns for an experiment which has no partial order discovery results.

**Test subject:** \getPartialSubgraphResultRequest\(\text{experimentId}:\text{int}, \text{clusters}:\text{string}\[\])

**Precondition:** An experiment with the given experiment id exists and does not have partial order discovery results

**Input:** An experiment id

**Output:** An error Response object indicating that the experiment contained no results

PartialSubgraphResultRequestTest\testGetResultWithCluster()

Tests the retrieval of patterns for an experiment

**Test subject:** \getPartialSubgraphResultRequest\(\text{experimentId}:\text{int}, \text{clusters}:\text{string}\[\])

**Precondition:** An experiment with the given experiment id exists and does have partial order discovery results

**Input:** An experiment id and an array of clusters

**Output:** A successful Response object with an ‘svg’ as data

3.12 Utility tests

In this section, basic utilities are tested. In particular, utilities like form validation, mail, error handling and file extension validation are tested.

3.12.1 Form validation utility tests

PartialSubgraphResultRequestTest\testUsernameIncorrect()

Tests the validation of a username.

**Test subject:** \validateUsername\(\text{name}:\text{string}, \text{response}:\text{Response}\)

**Precondition:** None

**Input:** An invalid username and a new Response object

**Output:** False

PartialSubgraphResultRequestTest\testEmailIncorrect()

Tests the validation of an email address.

**Test subject:** \validateEmail\(\text{email}:\text{string}, \text{response}:\text{Response}\)

**Precondition:** None

**Input:** An invalid email and a new Response object

**Output:** False

PartialSubgraphResultRequestTest\testPasswordIncorrect()

Tests the validation of a password.

**Test subject:** \validatePassword\(\text{pass}:\text{string}, \text{response}:\text{Response}\)

**Precondition:** None

**Input:** An invalid password and a new Response object

**Output:** False
Tests the validation of a username.

**Test subject:** `validateUsername(name: string, response: Response)`

**Precondition:** None

**Input:** A valid username and a new Response object

**Output:** True

Tests the validation of an email address.

**Test subject:** `validateEmail(email: string, response: Response)`

**Precondition:** None

**Input:** A valid email and a new Response object

**Output:** True

Tests the validation of a password.

**Test subject:** `validatePassword(pass: string, response: Response)`

**Precondition:** None

**Input:** A valid password and a new Response object

**Output:** True

3.12.2 Mail utility tests

Tests the sending of an email.

**Test subject:** `Mailer.sendMail(receiver: string, subject: string, body: string, bcc: null|string[], alternativeBody: null|string)`

**Precondition:** None

**Input:** A receiver mail, a subject, a body and an array containing a bbc mail

**Output:** A successful Response object

Tests the sending of an email.

**Test subject:** `Mailer.sendMail(receiver: string, subject: string, body: string, bcc: null|string[], alternativeBody: null|string)`

**Precondition:** None

**Input:** An invalid receiver mail, a subject, a body and an array containing a bbc mail

**Output:** An error Response object indicating the receiver mail was invalid

Tests the sending of an email.

**Test subject:** `Mailer.sendMail(receiver: string, subject: string, body: string, bcc: null|string[], alternativeBody: null|string)`

**Precondition:** None

**Input:** A receiver mail, a subject, a body and an array containing an invalid bbc mail

**Output:** An error Response object indicating the bbc mail was invalid
UT-341  
MailerTest\testSendMailFromTemplate()

Tests the sending of an email from a template.

**Test subject:** \Mailer\sendMailFromTemplate(*receiver*: string,  *template*: MailTemplate,  *bcc*: null|string[],  *alternativeBody*: null|string)

**Precondition:** None
**Input:** A receiver mail, a mail template and an array containing a bbc mail
**Output:** A successful Response object

UT-342  
MailFactoryTest\testMailSingleton()

Tests the retrieval of a mail factory instance.

**Test subject:** \MailFactory\instance()

**Precondition:** None
**Input:** None
**Output:** The MailFactory instance

UT-343  
MailFactoryTest\testMailFactory()

Tests the construction of a mail factory.

**Test subject:** \MailFactory\__construct()

**Precondition:** None
**Input:** None
**Output:** The mail templates

UT-344  
MailTemplateTest\testConstructFilePathDoesNotExist()

Test whether an exception is thrown when an invalid mail template path is given on construction.

**Test subject:** \MailTemplate\__construct(*subject*: string,  *path*: string)

**Precondition:** The path to the mail template does not exist
**Input:** A mail subject and an invalid path to a mail template
**Output:** An InvalidArgumentException

UT-345  
MailTemplateTest\testGetSubject()

Tests the retrieval of the mail template's subject.

**Test subject:** \MailTemplate\getSubject()

**Precondition:** None
**Input:** A mail subject and a path to a mail template
**Output:** The mail subject as defined in the constructor

UT-346  
MailTemplateTest\testGetContent()

Tests the retrieval of the mail template's content.

**Test subject:** \MailTemplate\getContent()

**Precondition:** The template has a <code>name</code> parameter
**Input:** A path to a mail template
**Output:** The template content
Tests the editing of the email template’s arguments.

**Test subject:** `MailTemplate множество аргументов`<br>
**Precondition:** The template has a `<code>name</code>` parameter<br>
**Input:** A path to a mail template and a parameter key value map<br>
**Output:** The template content with the parameter set

### 3.12.3 Error handling utility tests

**UT-348**

Tests the key-value representation of the error data object.

**Test subject:** `ErrorData объекта ошибки`<br>
**Precondition:** There exists an ErrorData object with a valid name and a valid message<br>
**Input:** None<br>
**Output:** A JSON string containing the valid values of the fields of the ErrorData object

**UT-349**

Tests the retrieval of the error message.

**Test subject:** `ErrorData сообщение ошибки`<br>
**Precondition:** There exists an ErrorData object with a valid name and a valid message<br>
**Input:** None<br>
**Output:** The message of the error

### 3.12.4 Response utility tests

**UT-350**

Tests the retrieval of the success value of a response.

**Test subject:** `Response успеха`<br>
**Precondition:** There exists a Response object<br>
**Input:** None<br>
**Output:** True

**UT-351**

Tests the retrieval of the success value of a response.

**Test subject:** `Response успеха`<br>
**Precondition:** There exists a Response object and an ErrorData object is added to the Response object<br>
**Input:** None<br>
**Output:** False
Tests the JSON representation of the response object.

**Test subject:** `Response::toJson()`  
**Precondition:** There exists a `Response` object  
**Input:** None  
**Output:** A JSON formatted string including the fields of the `Response` object

Tests the retrieval of errors contained in a response.

**Test subject:** `Response::getErrors()`  
**Precondition:** There exists a `Response` object with an error  
**Input:** None  
**Output:** The error of the `Response` object

Tests the adding of data to a response.

**Test subject:** `Response::addData(key: string, object: string)`  
**Precondition:** There exists a `Response` object  
**Input:** A key and a User object  
**Output:** None

Tests the retrieval of data contained in a response.

**Test subject:** `Response::getData()`  
**Precondition:** There exists a `Response` object and data has been added to the `Response` object  
**Input:** A key  
**Output:** The object associated with the given key

Tests the deletion of data from a response.

**Test subject:** `Response::removeData(key: string)`  
**Precondition:** There exists a `Response` object and the this response contains data  
**Input:** A key  
**Output:** Null

Tests the JSON representation of a response object.

**Test subject:** `Response::toJson()`  
**Precondition:** There exists a `Response` object and an object and key were added to the `Response` object  
**Input:** A `Response` object  
**Output:** A JSON formatted string including the fields of the `Response` object and the added data
3.12.5 Timezone utility tests

**UT-358**  
Tests the retrieval of a time zone instance.  
**Test subject:** `usedTimezone()`  
**Precondition:** None  
**Input:** None  
**Output:** An instance of the `DateTimeZone` class

3.12.6 File extension validation tests

**UT-359**  
Tests the validation of a given file extension.  
**Test subject:** `validateExtensionInput(fileName : string, extension : string[], response : Response)`  
**Precondition:** None  
**Input:** A filename with an extension, an array with the same extension and a new Response object  
**Output:** True

**UT-360**  
Tests the validation of a given file extension.  
**Test subject:** `validateExtensionInput(fileName : string, extension : string[], response : Response)`  
**Precondition:** None  
**Input:** An empty filename, an array with an extension and a new Response object  
**Output:** False

**UT-361**  
Tests the validation of a given file extension.  
**Test subject:** `validateExtensionInput(fileName : string, extension : string[], response : Response)`  
**Precondition:** None  
**Input:** A filename with an extension, an empty array and a new Response object  
**Output:** False

**UT-362**  
Tests the validation of a given file extension.  
**Test subject:** `validateExtensionInput(fileName : string, extension : string[], response : Response)`  
**Precondition:** None  
**Input:** A null filename, an array with an extension and a new Response object  
**Output:** False
Tests the validation of a given file extension.

**Test subject:** `validateExtensionInput(fileName : string, extension : string[], response : Response)`

**Precondition:** None

**Input:** A filename with an extension, a null array and a new Response object

**Output:** False

Tests the validation of a given file extension.

**Test subject:** `validateFileExtension(fileName : string, extension : string[], response : Response)`

**Precondition:** None

**Input:** A filename with an extension, an array with the same extension and a new Response object

**Output:** True

Tests the validation of a given file extension.

**Test subject:** `validateFileExtension(fileName : string, extension : string[], response : Response)`

**Precondition:** None

**Input:** A filename with an extension, an array with the same extension and other extensions and a new Response object

**Output:** True

Tests the validation of a given file extension.

**Test subject:** `validateFileExtension(fileName : string, extension : string[], response : Response)`

**Precondition:** None

**Input:** A filename with an extension, an array without the same extension and a new Response object

**Output:** False

Tests the validation of a given file extension.

**Test subject:** `validateFileExtension(fileName : string, extension : string[], response : Response)`

**Precondition:** None

**Input:** A filename with an extension, an array with other extensions and a new Response object

**Output:** False
UT-368

FileExtensionValidationTest\testNoExtension()

Tests the validation of a given file extension.

Test subject: validateFileExtension(fileName : string, extension : string[], response : Response)
Precondition: None
Input: A filename without an extension, an array with an extension and a new Response object
Output: False

UT-369

FileExtensionValidationTest\testExtraDotsSingle()

Tests the validation of a given file extension.

Test subject: validateFileExtension(fileName : string, extension : string[], response : Response)
Precondition: None
Input: A filename with an extension and extra dots, an array with the same extension and a new Response object
Output: True

UT-370

FileExtensionValidationTest\testExtraDotsDouble()

Tests the validation of a given file extension.

Test subject: validateFileExtension(fileName : string, extension : string[], response : Response)
Precondition: None
Input: A filename with an extension and extra dots, an array with the same extension and other extensions and a new Response object
Output: True

UT-371

FileExtensionValidationTest\testNoFilename()

Tests the validation of a given file extension.

Test subject: validateFileExtension(fileName : string, extension : string[], response : Response)
Precondition: None
Input: An empty filename, an array with an extension and a new Response object
Output: False

UT-372

FileExtensionValidationTest\testNoExtensionSingle()

Tests the validation of a given file extension.

Test subject: validateFileExtension(fileName : string, extension : string[], response : Response)
Precondition: None
Input: A filename with an extension, an array with only an empty string and a new Response object
Output: False
Tests the validation of a given file extension.

**Test subject:** `validateFileExtension(fileName : string, extension : string[], response : Response)`

**Precondition:** None

**Input:** A filename with an extension, an empty array and a new Response object

**Output:** False

### 3.12.7 Context Switch tests

**UT-374**

Tests the initial usage of the context switch and the changing of the working directory.

**Test subject:** `\utils\execution\ContextSwitch\begin()`

**Precondition:** None

**Input:** None

**Output:** The working directory is changed

**UT-375**

Tests the closing of the context switch and the changing to the old working directory.

**Test subject:** `\utils\execution\ContextSwitch\end()`

**Precondition:** None

**Input:** None

**Output:** The working directory is changed back to the old working directory

### 3.13 Settings tests

In this section, global settings are tested. In particular, settings with respects to the database, the mail server and password hashing are tested.

**UT-376**

Tests the retrieval of a settings instance.

**Test subject:** `\Settings\instance()`

**Precondition:** None

**Input:** None

**Output:** The Settings instance

**UT-377**

Tests the retrieval of an email sender name from the settings.

**Test subject:** `\Settings\emailSenderName()`

**Precondition:** None

**Input:** None

**Output:** A non-null and non empty sender name
Tests the retrieval of an email server name from the settings.

**Test subject:** Settings
**emailSMTPServerName()**
**Precondition:** None
**Input:** None
**Output:** A non-null and non empty sender name

Tests the retrieval of an email user name from the settings.

**Test subject:** Settings
**emailUsername()**
**Precondition:** None
**Input:** None
**Output:** A non-null and non empty sender name

Tests the retrieval of an email password from the settings.

**Test subject:** Settings
**emailPassword()**
**Precondition:** None
**Input:** None
**Output:** A non-null and non empty username

Tests the retrieval of an encryption method from the settings.

**Test subject:** Settings
**emailSMTPEncryptionMethod()**
**Precondition:** None
**Input:** None
**Output:** A valid textual representation of the encryption method

Tests the retrieval of a server port digit from the settings.

**Test subject:** Settings
**emailSMTPServerPort()**
**Precondition:** None
**Input:** None
**Output:** A port digit
4 Test procedures

4.1 Unit test procedure

4.1.1 Purpose

The purpose of this procedure is to execute all the unit tests implemented by Delta. The output does not include coverage or test duration.

4.1.2 Procedure steps

The following procedure is run on the Linux machine, as explained in Section 2.5.

1. Create a file named `phpunit.xml` with the XML content as described in Listing 1.
2. In the `phpunit.xml` file, replace `[@sourcepath@]` with the path to the root folder of the tool.
3. Let the path to the `phpunit.xml` configuration be referred to as `<confpath>`. In the directory where the PHPUnit PHAR package is located (named `phpunit.phar`), run PHPUnit using the configuration file using the following command.

   ```bash
   php5 phpunit.phar -c <confpath>/phpunit.xml
   ``

4. Observe the output printed by PHPUnit.

```xml
<phpunit
  color="true"
  stopOnError="false"
  stopOnFailure="false"
  stopOnIncomplete="false"
  stopOnSkipped="false"
  stopOnRisky="false"
  verbose="true">
  <logging>
    <log type="coverage-html" target="[@sourcepath@]/Esub/test/results/coverage"/>
  </logging>

  <testsuites>
    <testsuite name="AllTests">
      <directory suffix=".php">[@sourcepath@]/Esub/test</directory>
    </testsuite>
  </testsuites>

  <filter>
    <whitelist addUncoveredFilesFromWhitelist="true">
      <directory suffix=".php">[@sourcepath@]/Esub/delta</directory>
      <exclude>
        <directory>[@sourcepath@]/Esub/delta/handlers</directory>
      </exclude>
    </whitelist>
  </filter>
</phpunit>
```

Listing 1: PHPUnit configuration file
5 Test Coverage

In this chapter screenshots of the test coverage results that have been acquired using the PHPUnit testing framework are given. Following the steps described in the previous chapter the full results can be obtained. However, the screenshots in this chapter should provide a complete overview.

5.1 Server code

PHPUnit is used to generate coverage information for the server code. This coverage is shown in the sections below.

Root folder

The root folder contains all the PHP server code. Figure 1 shows the coverage of the root folder. The apd folder contains the interface between the Delta extension and the original APD tool. We ensured a high coverage for database, models, request, settings, utils and validation as these are the core features of the Delta extension. Finally, note that the handlers folder is excluded from coverage as these files do not contain testable logic. The logic of these handlers is placed in the corresponding requests.

<table>
<thead>
<tr>
<th></th>
<th>Lines</th>
<th>Code Coverage</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Total</td>
<td>68.66%</td>
<td>1052 / 1024</td>
<td>52.71%</td>
<td>164 / 176</td>
<td>66.21%</td>
</tr>
<tr>
<td>apd</td>
<td>79.60%</td>
<td>180 / 228</td>
<td>62.00%</td>
<td>16 / 24</td>
<td>33.33%</td>
</tr>
<tr>
<td>database</td>
<td>92.43%</td>
<td>523 / 566</td>
<td>96.91%</td>
<td>10 / 11</td>
<td>50.00%</td>
</tr>
<tr>
<td>exceptions</td>
<td>100.00%</td>
<td>0 / 0</td>
<td>100.00%</td>
<td>0 / 0</td>
<td>100.00%</td>
</tr>
<tr>
<td>handlers</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>model</td>
<td>100.00%</td>
<td>250 / 250</td>
<td>100.00%</td>
<td>114 / 114</td>
<td>100.00%</td>
</tr>
<tr>
<td>requests</td>
<td>92.04%</td>
<td>391 / 427</td>
<td>100.00%</td>
<td>0 / 0</td>
<td>0 / 0</td>
</tr>
<tr>
<td>settings</td>
<td>100.00%</td>
<td>10 / 10</td>
<td>100.00%</td>
<td>0 / 0</td>
<td>0 / 0</td>
</tr>
<tr>
<td>utils</td>
<td>92.66%</td>
<td>66 / 73</td>
<td>91.67%</td>
<td>11 / 12</td>
<td>75.00%</td>
</tr>
<tr>
<td>validation</td>
<td>100.00%</td>
<td>101 / 101</td>
<td>100.00%</td>
<td>0 / 0</td>
<td>0 / 0</td>
</tr>
<tr>
<td>reset-password.php</td>
<td>0.00%</td>
<td>0 / 0</td>
<td>100.00%</td>
<td>0 / 0</td>
<td>0 / 0</td>
</tr>
</tbody>
</table>

Figure 1: The coverage of the root folder

Apd

The coverage of the apd folder can be seen in Figure 2. The esub folder contains some post processing features used to obtain better results from the original APD tool. There are some files in this folder that are directly coupled to the original APD tool. Therefore these files are untestable, explaining the relatively low coverage seen in Figure 3.

<table>
<thead>
<tr>
<th></th>
<th>Lines</th>
<th>Code Coverage</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Total</td>
<td>79.60%</td>
<td>160 / 229</td>
<td>62.65%</td>
<td>15 / 26</td>
<td>33.33%</td>
</tr>
<tr>
<td>esub</td>
<td>63.37%</td>
<td>64 / 101</td>
<td>66.71%</td>
<td>6 / 7</td>
<td>0.00%</td>
</tr>
<tr>
<td>ApdFacade.php</td>
<td>100.00%</td>
<td>0 / 0</td>
<td>100.00%</td>
<td>0 / 0</td>
<td>0 / 0</td>
</tr>
<tr>
<td>EslFacade.php</td>
<td>90.62%</td>
<td>116 / 128</td>
<td>52.14%</td>
<td>3 / 17</td>
<td>0.00%</td>
</tr>
<tr>
<td>ExperimentPhaseFinishedListener.php</td>
<td>100.00%</td>
<td>0 / 0</td>
<td>100.00%</td>
<td>0 / 0</td>
<td>0 / 0</td>
</tr>
<tr>
<td>PatternType.php</td>
<td>100.00%</td>
<td>0 / 0</td>
<td>100.00%</td>
<td>0 / 0</td>
<td>100.00%</td>
</tr>
</tbody>
</table>

Figure 2: The coverage of the apd folder
Figure 3: The coverage of the esub folder

Database

The coverage of the database folder can be seen in Figure 4. Here the database.php file only contains the creation and setup of the database. This setup of the database is tested using a global variable, which did not increase the coverage.

Figure 4: The coverage of the database folder

Exceptions

The coverage of the exceptions folder can be seen in Figure 5.

Figure 5: The coverage of the exception folder

Model

The coverage of the model folder can be seen in Figure 6.

Requests

The coverage of the requests folder can be seen in Figure 7. The lower coverage of some files is caused by the fact that authentication checks are performed. When these fail a PHP exit statement is called, stopping the execution of the tests.

Session

The coverage of the session folder can be seen in Figure 8. Authenticated-user.php and http-error.php both contain PHP exit statements. These exit statements will stop the execution when included in the tests, and are therefore not covered.
The coverage of the utils folder can be seen in Figure 9. The coverage of both the execution and mail folder can be seen in Figure 10 and Figure 11. Reproducing an error that can occur during mailing turned out to be too difficult. Therefore these errors were not tested, which results in a lower coverage.
Figure 9: The coverage of the utils folder

Figure 10: The coverage of the execution folder

Figure 11: The coverage of the mail folder

Validation

The coverage of the validation folder can be seen in Figure 12.

Figure 12: The coverage of the validation folder
6 Test reports

On the following pages the test report generated by PhpStorm is given. All back-end tests have been run and all test cases passed. For each test case the time it took to complete is shown on the right. The total time for running all tests in each file is also shown.

All tests: 382 total, 382 passed

<table>
<thead>
<tr>
<th>All</th>
<th>43.07 s</th>
</tr>
</thead>
<tbody>
<tr>
<td>apdEsubFacadeTest</td>
<td>27.89 s</td>
</tr>
<tr>
<td>testASDResult</td>
<td>passed</td>
</tr>
<tr>
<td>testASDResultNoFiles</td>
<td>passed</td>
</tr>
<tr>
<td>testPODRResultInvalidPatternTypeWithoutThreshold</td>
<td>passed</td>
</tr>
<tr>
<td>testPODRResultInvalidSupportWithThreshold</td>
<td>passed</td>
</tr>
<tr>
<td>testPODRResult</td>
<td>passed</td>
</tr>
<tr>
<td>testPODRResultNoFiles</td>
<td>passed</td>
</tr>
<tr>
<td>testNoStartPhaseWhenNotFinished</td>
<td>passed</td>
</tr>
<tr>
<td>testStartASDPhase</td>
<td>passed</td>
</tr>
<tr>
<td>testStartPODPhase</td>
<td>passed</td>
</tr>
<tr>
<td>testExpandSubgraphNodeProcessor</td>
<td>passed</td>
</tr>
<tr>
<td>testGetNodeExpandedGraphSingle</td>
<td>passed</td>
</tr>
<tr>
<td>testGetNodeExpandedGraphDouble</td>
<td>passed</td>
</tr>
<tr>
<td>testGetNodeExpandedGraphFull</td>
<td>passed</td>
</tr>
<tr>
<td>testGetPartialSubgraphSingle</td>
<td>passed</td>
</tr>
<tr>
<td>testGetPartialSubgraphDouble</td>
<td>passed</td>
</tr>
<tr>
<td>DatabaseCredentialsTest</td>
<td>0 ms</td>
</tr>
<tr>
<td>testGetDatabaseCredentialsHost</td>
<td>passed</td>
</tr>
<tr>
<td>testGetDatabaseCredentialsUsername</td>
<td>passed</td>
</tr>
<tr>
<td>testGetDatabaseCredentialsPassword</td>
<td>passed</td>
</tr>
<tr>
<td>testGetDatabaseCredentialsDatabaseName</td>
<td>passed</td>
</tr>
<tr>
<td>DatabaseManagerTest</td>
<td>0 ms</td>
</tr>
<tr>
<td>testConnectionPrepareValid</td>
<td>passed</td>
</tr>
<tr>
<td>testConnectionPrepareInvalid</td>
<td>passed</td>
</tr>
<tr>
<td>ConsentQueriesTest</td>
<td>160 ms</td>
</tr>
<tr>
<td>testConsentTextById</td>
<td>passed</td>
</tr>
<tr>
<td>testLatestConsentText</td>
<td>passed</td>
</tr>
<tr>
<td>testGiveProfileConsent</td>
<td>passed</td>
</tr>
<tr>
<td>testGiveProfileConsentWithNullText</td>
<td>passed</td>
</tr>
<tr>
<td>testRevokeProfileConsent</td>
<td>passed</td>
</tr>
<tr>
<td>testRevokeProfileConsentWithNullText</td>
<td>passed</td>
</tr>
<tr>
<td>testHasGivenProfileConsentAfterAdd</td>
<td>passed</td>
</tr>
<tr>
<td>testHasGivenProfileConsentAfterRevoke</td>
<td>passed</td>
</tr>
<tr>
<td>testLatestProfileConsentText</td>
<td>passed</td>
</tr>
<tr>
<td>testConsentTextByIdProblem</td>
<td>passed</td>
</tr>
<tr>
<td>testLatestProfileConsentTextProblem</td>
<td>passed</td>
</tr>
<tr>
<td>DatabaseTest</td>
<td>0 ms</td>
</tr>
<tr>
<td>-----------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>testDatabaseObjectNotNull</td>
<td>passed 0 ms</td>
</tr>
<tr>
<td>testDatabaseObject</td>
<td>passed 0 ms</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>ExperimentQueriesTest</td>
<td>150 ms</td>
</tr>
<tr>
<td>testExperimentsByProjectValid</td>
<td>passed 10 ms</td>
</tr>
<tr>
<td>testExperimentsByProjectInvalid</td>
<td>passed 0 ms</td>
</tr>
<tr>
<td>testExperimentsByOwnerValid</td>
<td>passed 20 ms</td>
</tr>
<tr>
<td>testExperimentsByOwnerInvalid</td>
<td>passed 0 ms</td>
</tr>
<tr>
<td>testDeleteExperiment</td>
<td>passed 10 ms</td>
</tr>
<tr>
<td>testAddStatus</td>
<td>passed 20 ms</td>
</tr>
<tr>
<td>testAddStatusInvalid</td>
<td>passed 0 ms</td>
</tr>
<tr>
<td>testInvalidStatusEntryId</td>
<td>passed 0 ms</td>
</tr>
<tr>
<td>testInvalidExperimentId</td>
<td>passed 0 ms</td>
</tr>
<tr>
<td>testNewExperimentInvalidProject</td>
<td>passed 0 ms</td>
</tr>
<tr>
<td>testNewExperimentInvalidCreator</td>
<td>passed 0 ms</td>
</tr>
<tr>
<td>testInvalidExperimentCurrentStatus</td>
<td>passed 0 ms</td>
</tr>
<tr>
<td>testAddStatusInvalidStatusCode</td>
<td>passed 0 ms</td>
</tr>
<tr>
<td>testAddStatusInvalidPhaseId</td>
<td>passed 0 ms</td>
</tr>
<tr>
<td>testAddStatusInvalidExperiment</td>
<td>passed 0 ms</td>
</tr>
<tr>
<td>testGetDescriptionInvalidProjectPhase</td>
<td>passed 0 ms</td>
</tr>
<tr>
<td>testChangeExperimentCreator</td>
<td>passed 20 ms</td>
</tr>
<tr>
<td>testExperimentPhaseName</td>
<td>passed 0 ms</td>
</tr>
<tr>
<td>testLatestStatusEmpty</td>
<td>passed 0 ms</td>
</tr>
<tr>
<td>testLatestStatus</td>
<td>passed 70 ms</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>ProjectQueriesTest</td>
<td>50 ms</td>
</tr>
<tr>
<td>testProjectByIdValid</td>
<td>passed 0 ms</td>
</tr>
<tr>
<td>testProjectByIdInvalid</td>
<td>passed 0 ms</td>
</tr>
<tr>
<td>testProjectByNameValid</td>
<td>passed 0 ms</td>
</tr>
<tr>
<td>testProjectByNameInvalid</td>
<td>passed 0 ms</td>
</tr>
<tr>
<td>testProjectsByOwnerValid</td>
<td>passed 0 ms</td>
</tr>
<tr>
<td>testProjectsByOwnerInvalid</td>
<td>passed 0 ms</td>
</tr>
<tr>
<td>testUpdateProject</td>
<td>passed 10 ms</td>
</tr>
<tr>
<td>testDeleteProject</td>
<td>passed 10 ms</td>
</tr>
<tr>
<td>testChangeProjectOwner</td>
<td>passed 20 ms</td>
</tr>
<tr>
<td>testFetchProjectsAsAdmin</td>
<td>passed 10 ms</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>ResetTokenQueriesTest</td>
<td>30 ms</td>
</tr>
<tr>
<td>testNewToken</td>
<td>passed 10 ms</td>
</tr>
<tr>
<td>testTokenByToken</td>
<td>passed 10 ms</td>
</tr>
<tr>
<td>testNoTokenByToken</td>
<td>passed 0 ms</td>
</tr>
<tr>
<td>testMarkTokenAsUsed</td>
<td>passed 10 ms</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>UserQueriesTest</td>
<td>100 ms</td>
</tr>
<tr>
<td>testUserAdmins</td>
<td>passed 10 ms</td>
</tr>
<tr>
<td>testUserById</td>
<td>passed 10 ms</td>
</tr>
<tr>
<td>testUserByIdInvalidId</td>
<td>passed 0 ms</td>
</tr>
<tr>
<td>testUserByUsername</td>
<td>passed 10 ms</td>
</tr>
</tbody>
</table>

---

**Delta Unit Test Plan 86**
<table>
<thead>
<tr>
<th>Test Case</th>
<th>Status</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>testUserByUsernameInvalidUsername</td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td>testUserByEmail</td>
<td>passed</td>
<td>20 ms</td>
</tr>
<tr>
<td>testUserByEmailInvalidEmail</td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td>testUpdatePasswordHash</td>
<td>passed</td>
<td>10 ms</td>
</tr>
<tr>
<td>testDeleteUser</td>
<td>passed</td>
<td>10 ms</td>
</tr>
<tr>
<td>testPasswordHash</td>
<td>passed</td>
<td>10 ms</td>
</tr>
<tr>
<td>testAllUsers</td>
<td>passed</td>
<td>20 ms</td>
</tr>
<tr>
<td>testNoUsers</td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td>UserProfileQueriesTest</td>
<td></td>
<td>20 ms</td>
</tr>
<tr>
<td>testUserProfileByUser</td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td>testUpdateUserProfile</td>
<td>passed</td>
<td>10 ms</td>
</tr>
<tr>
<td>testDeleteUserProfile</td>
<td>passed</td>
<td>10 ms</td>
</tr>
<tr>
<td>ConsentTextTest</td>
<td></td>
<td>0 ms</td>
</tr>
<tr>
<td>testConsentTextGetId</td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td>testConsentTextGetText</td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td>testConsentTextGetCreationDate</td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td>testToString</td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td>testToArray</td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td>ErrorDataTest</td>
<td></td>
<td>0 ms</td>
</tr>
<tr>
<td>testToJson</td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td>testGetMessage</td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td>ExperimentTest</td>
<td></td>
<td>260 ms</td>
</tr>
<tr>
<td>testExperimentById</td>
<td>passed</td>
<td>10 ms</td>
</tr>
<tr>
<td>testExperimentGetName</td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td>testGetProject</td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td>testGetCreator</td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td>testGetExperimentCreationDate</td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td>testGetExperimentFolderName</td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td>testGetTimestampsFileName</td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td>testGetExperimentString</td>
<td>passed</td>
<td>30 ms</td>
</tr>
<tr>
<td>testToArrayNoStatus</td>
<td>passed</td>
<td>10 ms</td>
</tr>
<tr>
<td>testToArrayWithStatus</td>
<td>passed</td>
<td>60 ms</td>
</tr>
<tr>
<td>testLatestAnomalousSubgraphDiscoveryStatusEmpty</td>
<td></td>
<td>0 ms</td>
</tr>
<tr>
<td>testLatestPartialOrderDiscoveryStatusEmpty</td>
<td></td>
<td>0 ms</td>
</tr>
<tr>
<td>testLatestAnomalousSubgraphDiscoveryStatus</td>
<td></td>
<td>60 ms</td>
</tr>
<tr>
<td>testLatestPartialOrderDiscoveryStatus</td>
<td></td>
<td>60 ms</td>
</tr>
<tr>
<td>testDeleteFiles</td>
<td>passed</td>
<td>20 ms</td>
</tr>
<tr>
<td>testDeleteTimestampsFile</td>
<td>passed</td>
<td>10 ms</td>
</tr>
<tr>
<td>testDeleteTimestampsFileFail</td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td>testDeleteDatabase</td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td>ExperimentFileTemplatesTest</td>
<td></td>
<td>0 ms</td>
</tr>
<tr>
<td>testTemplateFromKey</td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td>testTemplateFromKeyNonExistingKey</td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td>PhaseTest</td>
<td></td>
<td>0 ms</td>
</tr>
<tr>
<td>Test Case</td>
<td>Status</td>
<td>Time</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------</td>
<td>-------</td>
</tr>
<tr>
<td>testGetId</td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td>testGetName</td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ProjectTest</td>
<td></td>
<td>190 ms</td>
</tr>
<tr>
<td>testProjectGetId</td>
<td>passed</td>
<td>10 ms</td>
</tr>
<tr>
<td>testProjectGetName</td>
<td>passed</td>
<td>10 ms</td>
</tr>
<tr>
<td>testProjectGetOwner</td>
<td>passed</td>
<td>10 ms</td>
</tr>
<tr>
<td>testUpdate</td>
<td>passed</td>
<td>20 ms</td>
</tr>
<tr>
<td>testDelete</td>
<td>passed</td>
<td>20 ms</td>
</tr>
<tr>
<td>testToString</td>
<td>passed</td>
<td>10 ms</td>
</tr>
<tr>
<td>testLegacyName</td>
<td>passed</td>
<td>10 ms</td>
</tr>
<tr>
<td>testProjectFolder</td>
<td>passed</td>
<td>10 ms</td>
</tr>
<tr>
<td>testPetriNetFile</td>
<td>passed</td>
<td>10 ms</td>
</tr>
<tr>
<td>testEventLog</td>
<td>passed</td>
<td>10 ms</td>
</tr>
<tr>
<td>testCreationDate</td>
<td>passed</td>
<td>10 ms</td>
</tr>
<tr>
<td>testGetExperiments</td>
<td>passed</td>
<td>10 ms</td>
</tr>
<tr>
<td>testToArrayNoOwnership</td>
<td>passed</td>
<td>10 ms</td>
</tr>
<tr>
<td>testToArrayOwnership</td>
<td>passed</td>
<td>10 ms</td>
</tr>
<tr>
<td>testDeleteProjectFiles</td>
<td>passed</td>
<td>30 ms</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ResponseTest</td>
<td></td>
<td>0 ms</td>
</tr>
<tr>
<td>testSuccess</td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td>testFail</td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td>testError</td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td>testGetErrors</td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td>testAddData</td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td>testGetData</td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td>testRemoveData</td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td>testAddDataJson</td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>StatusEntryTest</td>
<td></td>
<td>60 ms</td>
</tr>
<tr>
<td>testGetExperiment</td>
<td>passed</td>
<td>10 ms</td>
</tr>
<tr>
<td>testGetStatus</td>
<td>passed</td>
<td>10 ms</td>
</tr>
<tr>
<td>testGetStatusId</td>
<td>passed</td>
<td>10 ms</td>
</tr>
<tr>
<td>testGetStatusCreationDate</td>
<td>passed</td>
<td>10 ms</td>
</tr>
<tr>
<td>testGetDescription</td>
<td>passed</td>
<td>10 ms</td>
</tr>
<tr>
<td>testGetStatusString</td>
<td>passed</td>
<td>10 ms</td>
</tr>
<tr>
<td>testGetProject</td>
<td>passed</td>
<td>10 ms</td>
</tr>
<tr>
<td>testGetId</td>
<td>passed</td>
<td>10 ms</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TokenTest</td>
<td></td>
<td>0 ms</td>
</tr>
<tr>
<td>testGetUser</td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td>testGetCreationDate</td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td>testGetToken</td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td>testNotIsUsed</td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td>testIsUsed</td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td>testIsNotExpired</td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td>testIsExpired</td>
<td>passed</td>
<td>0 ms</td>
</tr>
</tbody>
</table>
# User Test

<table>
<thead>
<tr>
<th>Test Name</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>testUserGetId</td>
<td>90 ms</td>
</tr>
<tr>
<td>testUserGetUsername</td>
<td>passed</td>
</tr>
<tr>
<td>testUserGetEmail</td>
<td>0 ms</td>
</tr>
<tr>
<td>testUserIsAdministrator</td>
<td>passed</td>
</tr>
<tr>
<td>testUpdate</td>
<td>10 ms</td>
</tr>
<tr>
<td>testDelete</td>
<td>passed</td>
</tr>
<tr>
<td>testPasswordHash</td>
<td>10 ms</td>
</tr>
<tr>
<td>testUpdatePasswordHash</td>
<td>passed</td>
</tr>
<tr>
<td>testGetProfile</td>
<td>passed</td>
</tr>
<tr>
<td>testGiveProfileConsent</td>
<td>passed</td>
</tr>
<tr>
<td>testRevokeProfileConsent</td>
<td>passed</td>
</tr>
<tr>
<td>testHasNotGivenProfileConsent</td>
<td>passed</td>
</tr>
<tr>
<td>testHasGivenProfileConsent</td>
<td>passed</td>
</tr>
<tr>
<td>testLatestProfileConsentText</td>
<td>passed</td>
</tr>
<tr>
<td>testToArray</td>
<td>passed</td>
</tr>
</tbody>
</table>

# UserProfile Test

<table>
<thead>
<tr>
<th>Test Name</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>testUserProfileGetUserId</td>
<td>passed</td>
</tr>
<tr>
<td>testUserProfileGetFirstName</td>
<td>passed</td>
</tr>
<tr>
<td>testUserProfileGetLastName</td>
<td>passed</td>
</tr>
<tr>
<td>testUserProfileGetOrganization</td>
<td>passed</td>
</tr>
<tr>
<td>testUserProfileGetRole</td>
<td>passed</td>
</tr>
<tr>
<td>testUserProfileGetCountry</td>
<td>passed</td>
</tr>
<tr>
<td>testUserProfileGetCity</td>
<td>passed</td>
</tr>
<tr>
<td>testUserProfileGetAddress</td>
<td>passed</td>
</tr>
<tr>
<td>testUserProfileGetHeardAbout</td>
<td>passed</td>
</tr>
<tr>
<td>testUpdate</td>
<td>passed</td>
</tr>
<tr>
<td>testDelete</td>
<td>passed</td>
</tr>
<tr>
<td>testToString</td>
<td>passed</td>
</tr>
</tbody>
</table>

# Account Information Request Test

<table>
<thead>
<tr>
<th>Test Name</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>testUserNotInDatabaseAllInformation</td>
<td>passed</td>
</tr>
<tr>
<td>testGetAllInformation</td>
<td>passed</td>
</tr>
</tbody>
</table>

# Delete User By Admin Request Test

<table>
<thead>
<tr>
<th>Test Name</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>testDeleteNotAdmin</td>
<td>passed</td>
</tr>
<tr>
<td>testDeleteNoUser</td>
<td>passed</td>
</tr>
<tr>
<td>testDeleteUserByAdmin</td>
<td>passed</td>
</tr>
</tbody>
</table>

# Delete User Request Test

<table>
<thead>
<tr>
<th>Test Name</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>testGetNewOwner</td>
<td>30 ms</td>
</tr>
<tr>
<td>testGetNewOwnerNoAdmins</td>
<td>30 ms</td>
</tr>
<tr>
<td>testGetNewOwnerWhenGivenProject</td>
<td>20 ms</td>
</tr>
<tr>
<td>testTransferAllOwnership</td>
<td>40 ms</td>
</tr>
<tr>
<td>testRequestDeleteUserSuccess</td>
<td>40 ms</td>
</tr>
<tr>
<td>testRequestDeleteUserNotActiveUser</td>
<td>30 ms</td>
</tr>
<tr>
<td>testRequestDeleteUserNoNewOwner</td>
<td>40 ms</td>
</tr>
<tr>
<td>Test Case</td>
<td>Time (ms)</td>
</tr>
<tr>
<td>------------------------------</td>
<td>-----------</td>
</tr>
<tr>
<td>DownloadLogRequestTest</td>
<td>20</td>
</tr>
<tr>
<td>testWrongExtension</td>
<td>10</td>
</tr>
<tr>
<td>testCorrectFile</td>
<td>10</td>
</tr>
<tr>
<td>testNoExperiment</td>
<td>0</td>
</tr>
<tr>
<td>EditAccountRequestTest</td>
<td>30</td>
</tr>
<tr>
<td>testEditProfileInformationNonExistingUser</td>
<td>0</td>
</tr>
<tr>
<td>testEditProfileInformation</td>
<td>10</td>
</tr>
<tr>
<td>testEditAccountInformationInvalidEmail</td>
<td>0</td>
</tr>
<tr>
<td>testEditAccountInformation</td>
<td>10</td>
</tr>
<tr>
<td>testEditAccountInformationEmailExists</td>
<td>10</td>
</tr>
<tr>
<td>EditPasswordRequestTest</td>
<td>820</td>
</tr>
<tr>
<td>testNonExistingUser</td>
<td>0</td>
</tr>
<tr>
<td>testWrongOldPassword</td>
<td>140</td>
</tr>
<tr>
<td>testInvalidNewPassword</td>
<td>410</td>
</tr>
<tr>
<td>testValidPasswordEdit</td>
<td>270</td>
</tr>
<tr>
<td>ExpandedSubgraphResultRequestTest</td>
<td>190</td>
</tr>
<tr>
<td>testNoExperiment</td>
<td>0</td>
</tr>
<tr>
<td>testGetResult</td>
<td>120</td>
</tr>
<tr>
<td>testGetResultWithCluster</td>
<td>50</td>
</tr>
<tr>
<td>testNoResult</td>
<td>20</td>
</tr>
<tr>
<td>ExperimentCreationRequestTest</td>
<td>20</td>
</tr>
<tr>
<td>testCreateExperiment</td>
<td>10</td>
</tr>
<tr>
<td>testCreateExperimentNoProject</td>
<td>0</td>
</tr>
<tr>
<td>testCreateExperimentNameNull</td>
<td>0</td>
</tr>
<tr>
<td>testRequestExperimentNameFail</td>
<td>0</td>
</tr>
<tr>
<td>testRequestExperimentCreationFail</td>
<td>0</td>
</tr>
<tr>
<td>testRequestExperimentCreation</td>
<td>10</td>
</tr>
<tr>
<td>FetchAllUsersRequestTest</td>
<td>80</td>
</tr>
<tr>
<td>testNoAdmin</td>
<td>10</td>
</tr>
<tr>
<td>testAdminWithUsers</td>
<td>70</td>
</tr>
<tr>
<td>FetchExperimentRequestTest</td>
<td>20</td>
</tr>
<tr>
<td>testInvalidProject</td>
<td>0</td>
</tr>
<tr>
<td>testExperimentsFetching</td>
<td>20</td>
</tr>
<tr>
<td>FetchProjectsRequestTest</td>
<td>200</td>
</tr>
<tr>
<td>testUserNoProjects</td>
<td>20</td>
</tr>
<tr>
<td>testUserOneProject</td>
<td>40</td>
</tr>
<tr>
<td>testUserMultipleProjects</td>
<td>40</td>
</tr>
<tr>
<td>testAdminAllProjects</td>
<td>100</td>
</tr>
<tr>
<td>ForgotPasswordRequestTest</td>
<td>20</td>
</tr>
<tr>
<td>testForgotPasswordInvalidInput</td>
<td>0</td>
</tr>
<tr>
<td>testForgotPasswordNoUserWithEmail</td>
<td>0</td>
</tr>
<tr>
<td>testForgotPasswordNoUserWithUsername</td>
<td>0</td>
</tr>
<tr>
<td>Test Case</td>
<td>Status</td>
</tr>
<tr>
<td>--------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td><code>testForgotPasswordWithEmail</code></td>
<td>passed</td>
</tr>
<tr>
<td><code>testForgotPasswordWithUsername</code></td>
<td>passed</td>
</tr>
</tbody>
</table>

**LoginRequestTest**

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Status</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>testLoginEmptyForm</code></td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td><code>testLoginNullValues</code></td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td><code>testLoginNoUsername</code></td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td><code>testLoginNoPassword</code></td>
<td>passed</td>
<td>70 ms</td>
</tr>
<tr>
<td><code>testLoginUsernameInvalid</code></td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td><code>testLoginWrongUsername</code></td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td><code>testLoginWrongPassword</code></td>
<td>passed</td>
<td>70 ms</td>
</tr>
<tr>
<td><code>testLoginSuccessfulLogin</code></td>
<td>passed</td>
<td>70 ms</td>
</tr>
</tbody>
</table>

**PartialOrderDiscoveryRequestTest**

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Status</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>testValidateExperimentInvalid</code></td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td><code>testValidateExperimentValid</code></td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td><code>testValidationNoExperiment</code></td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td><code>testValidationValid</code></td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td><code>testValidationNoNumeric</code></td>
<td>passed</td>
<td>10 ms</td>
</tr>
<tr>
<td><code>testRequestValidationNoExperiment</code></td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td><code>testRequestValidationStringThreshold</code></td>
<td>passed</td>
<td>10 ms</td>
</tr>
<tr>
<td><code>testRequestValidationStringSupport</code></td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td><code>testRequestValidationValid</code></td>
<td>passed</td>
<td>10 ms</td>
</tr>
<tr>
<td><code>testSubgraphNotFinished</code></td>
<td>passed</td>
<td>10 ms</td>
</tr>
</tbody>
</table>

**PartialOrderResultRequestTest**

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Status</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>testIncorrectExperimentId</code></td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td><code>testNoResults</code></td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td><code>testResults</code></td>
<td>passed</td>
<td>160 ms</td>
</tr>
<tr>
<td><code>testInputWrongPattern</code></td>
<td>passed</td>
<td>10 ms</td>
</tr>
<tr>
<td><code>testInputWrongSupport</code></td>
<td>passed</td>
<td>0 ms</td>
</tr>
</tbody>
</table>

**PartialSubgraphResultRequestTest**

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Status</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>testNoExperiment</code></td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td><code>testNoResult</code></td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td><code>testGetResultWithCluster</code></td>
<td>passed</td>
<td>50 ms</td>
</tr>
</tbody>
</table>

**PasswordResetLinkCreationRequestTest**

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Status</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>testCreateLinkNotAdmin</code></td>
<td>passed</td>
<td>10 ms</td>
</tr>
<tr>
<td><code>testCreateLinkNoUser</code></td>
<td>passed</td>
<td>20 ms</td>
</tr>
<tr>
<td><code>testCreateLink</code></td>
<td>passed</td>
<td>20 ms</td>
</tr>
<tr>
<td><code>testGenerateLink</code></td>
<td>passed</td>
<td>10 ms</td>
</tr>
<tr>
<td><code>testGenerateToken</code></td>
<td>passed</td>
<td>10 ms</td>
</tr>
<tr>
<td><code>testSendResetMail</code></td>
<td>passed</td>
<td>1.96 s</td>
</tr>
</tbody>
</table>

**ProjectCreationRequestTest**

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Status</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>testProjectNameEmpty</code></td>
<td>passed</td>
<td>10 ms</td>
</tr>
<tr>
<td><code>testDataValidationProjectNameTooLong</code></td>
<td>passed</td>
<td>10 ms</td>
</tr>
<tr>
<td><code>testDataValidationEventLogEmpty</code></td>
<td>passed</td>
<td>10 ms</td>
</tr>
<tr>
<td><code>testDataValidationEventLogNoExtension</code></td>
<td>passed</td>
<td>10 ms</td>
</tr>
<tr>
<td>Test Case</td>
<td>Result</td>
<td>Time</td>
</tr>
<tr>
<td>--------------------------------------------------------------------------</td>
<td>--------</td>
<td>-------</td>
</tr>
<tr>
<td>testDataValidationEventLogWrongExtension</td>
<td>passed</td>
<td>10 ms</td>
</tr>
<tr>
<td>testDataValidationProcessModelEmpty</td>
<td>passed</td>
<td>10 ms</td>
</tr>
<tr>
<td>testDataValidationProcessModelNoExtension</td>
<td>passed</td>
<td>10 ms</td>
</tr>
<tr>
<td>testDataValidationProcessModelWrongExtension</td>
<td>passed</td>
<td>10 ms</td>
</tr>
<tr>
<td>testDataValidationCorrect</td>
<td>passed</td>
<td>10 ms</td>
</tr>
<tr>
<td>testProjectNameExists</td>
<td>passed</td>
<td>10 ms</td>
</tr>
<tr>
<td>testDatabaseError</td>
<td>passed</td>
<td>10 ms</td>
</tr>
<tr>
<td>testInvalidInputOnRequest</td>
<td>passed</td>
<td>10 ms</td>
</tr>
<tr>
<td>testGetEventLogXes</td>
<td>passed</td>
<td>10 ms</td>
</tr>
<tr>
<td>testCreateProjectIncorrectFiles</td>
<td>passed</td>
<td>10 ms</td>
</tr>
<tr>
<td>testUploadEventLogWithNoFile</td>
<td>passed</td>
<td>10 ms</td>
</tr>
<tr>
<td>testUploadEventLog</td>
<td>passed</td>
<td>10 ms</td>
</tr>
<tr>
<td>testUploadProcessModelWithNoFile</td>
<td>passed</td>
<td>10 ms</td>
</tr>
<tr>
<td>testUploadProcessModel</td>
<td>passed</td>
<td>10 ms</td>
</tr>
<tr>
<td>testCreateFolderNoEventLog</td>
<td>passed</td>
<td>20 ms</td>
</tr>
<tr>
<td>testCreateFolderNoProcessModel</td>
<td>passed</td>
<td>10 ms</td>
</tr>
<tr>
<td>testCreateFolder</td>
<td>passed</td>
<td>10 ms</td>
</tr>
<tr>
<td>testCreateNewProject</td>
<td>passed</td>
<td>20 ms</td>
</tr>
<tr>
<td>testCreateProjectNoFiles</td>
<td>passed</td>
<td>20 ms</td>
</tr>
<tr>
<td>testCreateProjectOnRequest</td>
<td>passed</td>
<td>20 ms</td>
</tr>
<tr>
<td>testCreateNewProject</td>
<td>passed</td>
<td>10 ms</td>
</tr>
<tr>
<td>testCreateFolder</td>
<td>passed</td>
<td>10 ms</td>
</tr>
<tr>
<td>testCreateProjectOnRegistrationRequest</td>
<td>passed</td>
<td>10 ms</td>
</tr>
<tr>
<td>testSetupDirectory</td>
<td>passed</td>
<td>10 ms</td>
</tr>
</tbody>
</table>

**ProjectDeletionRequestTest**

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Result</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>testValidateOwner</td>
<td>passed</td>
<td>140 ms</td>
</tr>
<tr>
<td>testDeleteProject</td>
<td>passed</td>
<td>0 ms</td>
</tr>
</tbody>
</table>

**RegistrationRequestTest**

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Result</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>testUsernameIncorrect</td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td>testUsernameExists</td>
<td>passed</td>
<td>10 ms</td>
</tr>
<tr>
<td>testEmailIncorrect</td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td>testEmailExists</td>
<td>passed</td>
<td>10 ms</td>
</tr>
<tr>
<td>testPasswordTooLong</td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td>testCreateUser</td>
<td>passed</td>
<td>290 ms</td>
</tr>
<tr>
<td>testCreateUserError</td>
<td>passed</td>
<td>270 ms</td>
</tr>
<tr>
<td>testCreateUserOnRequest</td>
<td>passed</td>
<td>270 ms</td>
</tr>
<tr>
<td>testRegisterWithoutConsent</td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td>testCreateUserOnRegistrationRequest</td>
<td>passed</td>
<td>270 ms</td>
</tr>
<tr>
<td>testCreateProfileEntry</td>
<td>passed</td>
<td>270 ms</td>
</tr>
<tr>
<td>testCreateProfileEntryNullValues</td>
<td>passed</td>
<td>280 ms</td>
</tr>
<tr>
<td>testRegistrationMail</td>
<td>passed</td>
<td>2.10 s</td>
</tr>
</tbody>
</table>

**ResetPasswordRequestTest**

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Result</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>testTokenNotExists</td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td>testTokenExpired</td>
<td>passed</td>
<td>10 ms</td>
</tr>
<tr>
<td>testTokenUsed</td>
<td>passed</td>
<td>10 ms</td>
</tr>
<tr>
<td>testInvalidPassword</td>
<td>passed</td>
<td>10 ms</td>
</tr>
<tr>
<td>testResetPassword</td>
<td>passed</td>
<td>530 ms</td>
</tr>
<tr>
<td>testResetPasswordTwice</td>
<td>passed</td>
<td>800 ms</td>
</tr>
<tr>
<td>Test Suite</td>
<td>Result</td>
<td>Time</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>--------</td>
<td>-------</td>
</tr>
<tr>
<td><strong>SubgraphResultRequestTest</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>testIncorrectExperimentId</td>
<td>passed</td>
<td>50 ms</td>
</tr>
<tr>
<td>testNoResults</td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td>testResults</td>
<td>passed</td>
<td>50 ms</td>
</tr>
<tr>
<td><strong>UserLoggedInRequestTest</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>testUserNotLoggedln</td>
<td>passed</td>
<td>10 ms</td>
</tr>
<tr>
<td>testUserLoggedlnNoAdmin</td>
<td>passed</td>
<td>10 ms</td>
</tr>
<tr>
<td>testUserLoggedlnAdmin</td>
<td>passed</td>
<td>10 ms</td>
</tr>
<tr>
<td><strong>UsernameRequestTest</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>testUsernameCorrect</td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td>testUsernameTooLong</td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td>testUsernameUnique</td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td><strong>SettingsTest</strong></td>
<td></td>
<td>0 ms</td>
</tr>
<tr>
<td>testSingleton</td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td>testMailName</td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td>testMailServerName</td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td>testMailUsername</td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td>testMailPassword</td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td>testMailEncryptionMethod</td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td>testMailPort</td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td><strong>ContextSwitchTest</strong></td>
<td></td>
<td>0 ms</td>
</tr>
<tr>
<td>testChangeWorkingDirectory</td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td>testChangeWorkingDirectoryBack</td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td><strong>MailFactoryTest</strong></td>
<td></td>
<td>0 ms</td>
</tr>
<tr>
<td>testMailSingleton</td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td>testMailFactory</td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td><strong>MailTemplateTest</strong></td>
<td></td>
<td>0 ms</td>
</tr>
<tr>
<td>testConstructFilePathDoesNotExist</td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td>testGetSubject</td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td>testGetContent</td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td>testSetArgument</td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td><strong>MailerTest</strong></td>
<td></td>
<td>4.17 s</td>
</tr>
<tr>
<td>testSendMail</td>
<td>passed</td>
<td>2.09 s</td>
</tr>
<tr>
<td>testSendMailInvalidReceiver</td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td>testSendMailInvalidBbc</td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td>testSendMailFromTemplate</td>
<td>passed</td>
<td>2.08 s</td>
</tr>
<tr>
<td><strong>TimezoneTest</strong></td>
<td></td>
<td>0 ms</td>
</tr>
<tr>
<td>testUsedTimezone</td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td><strong>ExperimentValidationTest</strong></td>
<td></td>
<td>0 ms</td>
</tr>
<tr>
<td>testNoExperimentName</td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td>testExperimentNameEmptyString</td>
<td>passed</td>
<td>0 ms</td>
</tr>
</tbody>
</table>
**FileExtensionValidationTest**

- testExtensionInputValidation: passed 0 ms
- testExtensionInputValidationEmptyString: passed 0 ms
- testExtensionInputValidationEmptyArray: passed 0 ms
- testExtensionInputValidationNullString: passed 0 ms
- testExtensionInputValidationNullArray: passed 0 ms
- testCorrectExtensionSingle: passed 0 ms
- testCorrectExtensionDouble: passed 0 ms
- testWrongExtensionSingle: passed 0 ms
- testWrongExtensionDouble: passed 0 ms
- testNoExtension: passed 0 ms
- testExtraDotsSingle: passed 0 ms
- testExtraDotsDouble: passed 0 ms
- testNoFilename: passed 0 ms
- testNoExtensionSingle: passed 0 ms
- testNoExtensionArray: passed 0 ms

**FormValidationTest**

- testUsernameIncorrect: passed 0 ms
- testEmailIncorrect: passed 0 ms
- testPasswordIncorrect: passed 0 ms
- testUsernameCorrect: passed 0 ms
- testEmailCorrect: passed 0 ms
- testPasswordCorrect: passed 0 ms

**PartialOrderInputValidationTest**

- testPattern: passed 0 ms
- testPatternIncorrect: passed 0 ms
- testPatternInput: passed 0 ms
- testPatternInputIncorrect: passed 0 ms

**ProjectNameValidationTest**

- testProjectNameTooLong: passed 0 ms
- testProjectNameEmpty: passed 0 ms
- testProjectNameCorrect: passed 0 ms
- testProjectNameExists: passed 10 ms

**RegistrationInputCheckTest**

- testUsernameIncorrect: passed 0 ms
- testUsernameExists: passed 10 ms
- testUsernameCorrect: passed 0 ms
- testEmailIncorrect: passed 0 ms
- testEmailExists: passed 10 ms
- testEmailCorrect: passed 0 ms
- testDataValidationUsernameTooLong: passed 0 ms
- testDataValidationUsernameEmpty: passed 0 ms
<table>
<thead>
<tr>
<th>Test Case</th>
<th>Status</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>testDataValidationUsernameContainsSpecial</td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td>testDataValidationUsernameExists</td>
<td>passed</td>
<td>10 ms</td>
</tr>
<tr>
<td>testDataValidationEmailTooLong</td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td>testDataValidationEmailEmpty</td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td>testDataValidationEmailInvalidFormat</td>
<td>passed</td>
<td>0 ms</td>
</tr>
<tr>
<td>testDataValidationEmailExists</td>
<td>passed</td>
<td>10 ms</td>
</tr>
<tr>
<td>testPasswordIncorrect</td>
<td>passed</td>
<td>10 ms</td>
</tr>
<tr>
<td>testPasswordCorrect</td>
<td>passed</td>
<td>0 ms</td>
</tr>
</tbody>
</table>

Generated by PhpStorm on 5-7-18 16:47