TECHNISCHE UNIVERSITÄT
CHEMNITZ

Entwicklerhandbuch

Softwaretechnikpraktikum

Fakultät für Informatik
Professur Softwaretechnik

Eingereicht von: Gruppe 5
Einreichungsdatum: 10.03.2023

Betreuerin: Prof. Dr. Janet Siegmund
Betreuer: Dominik Gorgosch

1 Volere Snow Cards

Mit den Snow Cards haben wir die Requirements unseres Systems erarbeitet. Dazu haben wir uns die einzelnen, von der Aufgabenstellung gegebenen Systemteile hergenommen und überlegt, wie diese funktionieren sollten. Dieser Schritt ist wichtig, damit man im Vorhinein bereits Erwartungen festlegt und im weiteren Verlauf damit sicherstellen kann, dass sich unsere Arbeit in die richtige Richtung bewegt. Des Weiteren bekommt man dadurch einen guten ersten Überblick über die Gesamtheit des Projektes.

Login System

Requirement-ID: **1.1** Requirement Type: **functional** Event: **login**

Description: **button press triggers the login process**

Rationale: **pressing the login button starts the login process**

Originator: **user, customer, administrator**

Fit Criterion: **shibboleth system prompts for user credentials**

Customer Satisfaction: **5** Customer Dissatisfaction: **5**

Dependencies: **-** Conflicts: **-**

Materials: **shibboleth documentation**

History: **created: 09.11.2022**

Requirement-ID: **1.2** Requirement Type: **functional** Event: **login**

Description: **user has the option to change his/her password**

Rationale: **when forgetting his/her password the user can request the system to send an e-mail to change the password**

Originator: **user, administrator**

Fit Criterion: **all users can successfully change their passwords**

Customer Satisfaction: **5** Customer Dissatisfaction: **3**

Dependencies: **-** Conflicts: **-**

Materials: **-**

History: **created: 09.11.2022**

Requirement-ID: **1.3** Requirement Type: **non-functional** Event: **login**

Description: **the front page has an intuitive design**

Rationale: **lowering the hurdles for the user to start using the platform**

Originator: **user**

Fit Criterion: **a test group of users spots the login button in less than 3 seconds on average**

Customer Satisfaction: **2** Customer Dissatisfaction: **3**

Dependencies: **-** Conflicts: **-**

Materials: `https://m3.material.io`

History: **created: 09.11.2022**

Requirement-ID: **1.4** Requirement Type: **non-functional** Event: **login**

Description: **the front page has an appealing design**

Rationale: **making a good first impression and don't annoy the users**

Originator: **user, customer**

Fit Criterion: **a test group of potential users considers creating a user account**

Customer Satisfaction: **2** Customer Dissatisfaction: **3**

Dependencies: **-** Conflicts: **-**

Materials: **-**

History: **created: 09.11.2022**

Exercise System

Requirement-ID: **2.1** Requirement Type: **functional** Event: **exercise**

Description: **hints for the programming exercises**

Rationale: **useful to avoid the user getting stuck and give additional information**

Originator: **user, customer**

Fit Criterion: **administrator should be able to add hints to an exercise**

Customer Satisfaction: **4** Customer Dissatisfaction: **3**

Dependencies: **-** Conflicts: **-**

Materials: **-**

History: **created: 10.11.2022**

Requirement-ID: **2.2** Requirement Type: **functional** Event: **exercise**

Description: **search function for exercises with type filter**

Rationale: **user can search for specific tasks to have a more individual learning experience**

Originator: **user, customer**

Fit Criterion: **all exercises can be found and filtered by type**

Customer Satisfaction: **4** Customer Dissatisfaction: **2**

Dependencies: **-** Conflicts: **-**

Materials: **-**

History: **created: 10.11.2022**

Requirement-ID: **2.3** Requirement Type: **functional** Event: **exercise**

Description: **log various metrics while the user solves the exercises**

Rationale: **statistics can be created to track the learning process**

Originator: **user, customer**

Fit Criterion: **metrics like time and how often an exercise was solved are logged and can be proofed with test**

Customer Satisfaction: **4** Customer Dissatisfaction: **3**

Dependencies: **-** Conflicts: **-**

Materials: **-**

History: **created: 10.11.2022**

Requirement-ID: **2.4** Requirement Type: **non-functional** Event: **exercise**

Description: **syntax highlighting in all exercises for both Python and Java**

Rationale: **syntax highlighting leads to a more readable and understandable source code, especially for beginners**

Originator: **user, customer**

Fit Criterion: **syntax is correctly highlighted in various scenarios**

Customer Satisfaction: **2** Customer Dissatisfaction: **4**

Dependencies: **-** Conflicts: **-**

Materials: **-**

History: **created 09.11.2022**

Requirement-ID: **2.5** Requirement Type: **non-functional** Event: **exercise**

Description: **prevent the code validation system from running malicious code**

Rationale: **malicious code can lead to system crashes or remote code executions**

Originator: **customer**

Fit Criterion: **malicious code entered into the validation system does not cause the server to crash or undermines system integrity**

Customer Satisfaction: **2** Customer Dissatisfaction: **3**

Dependencies: **-** Conflicts: **-**

Materials: **-**

History: **created: 14.11.2022**

Administration System

Requirement-ID: **3.1** Requirement Type: **functional** Event: **administration**

Description: **the platform shall conform to data protection requirements**

Rationale: **the administration system shall not allow access to sensitive user information**

Originator: **user, data protection officer**

Fit Criterion: **federal data protection officer approves the design**

Customer Satisfaction: **1** Customer Dissatisfaction: **3**

Dependencies: **-** Conflicts: **3.2**

Materials: `https://gdpr-info.eu/`

History: **created: 05.11.2022**

Requirement-ID: **3.2** Requirement Type: **functional** Event: **administration**

Description: **the administration system shall provide insights into required metrics**

Rationale: **metrics are necessary to create learning schedules**

Originator: **customer**

Fit Criterion: **all the required metrics can be analyzed in the administration system**

Customer Satisfaction: **5** Customer Dissatisfaction: **5**

Dependencies: **-** Conflicts: **3.1**

Materials: **-**

History: **created: 05.11.2022**

Requirement-ID: **3.3** Requirement Type: **functional** Event: **administration**

Description: **the administration system shall provide the ability to create tasks**

Rationale: **creating tasks is necessary to expand the platform**

Originator: **customer, administrator**

Fit Criterion: **after a year of service the platform has more than just the sample exercises**

Customer Satisfaction: **5** Customer Dissatisfaction: **5**

Dependencies: **-** Conflicts: **-**

Materials: **-**

History: **created: 07.11.2022**

Requirement-ID: **3.4** Requirement Type: Event: **administration non-functional**

Description: **the administration system shall be intuitive to use**

Rationale: **an unintuitive administration system drives the administrators away**

Originator: **administrator**

Fit Criterion: **an administrator shall perform a task in less then 15 minutes after reading the user manual**

Customer Satisfaction: **3** Customer Dissatisfaction: **4**

Dependencies: **-** Conflicts: **-**

Materials: **-**

History: **created: 07.11.2022**

Requirement-ID: **3.5** Requirement Type: Event: **administration non-functional**

Description: **the administration system shall work performant**

Rationale: **long response times lower the user experience**

Originator: **administrator**

Fit Criterion: **the administration system returns results after a maximum of one second**

Customer Satisfaction: **3** Customer Dissatisfaction: **4**

Dependencies: **-** Conflicts: **-**

Materials: **-**

History: **created: 07.11.2022**

Evaluation System

Requirement-ID: **4.1** Requirement Type: **functional** Event: **evaluation**

Description: **the user receives an evaluation after submitting a solution**

Rationale: **the user needs to see an evaluation to understand the scoring**

Originator: **customer**

Fit Criterion: **validation with test data**

Customer Satisfaction: **1** Customer Dissatisfaction: **5**

Dependencies: **-** Conflicts: **-**

Materials: **-**

History: **created: 05.11.2022**

Requirement-ID: **4.2** Requirement Type: **functional** Event: **evaluation**

Description: **the evaluation system saves results to the database**

Rationale: **learning process is evident and comparison the previous solutions is possible**

Originator: **customer**

Fit Criterion: **validation with test data**

Customer Satisfaction: **2** Customer Dissatisfaction: **4**

Dependencies: **-** Conflicts: **-**

Materials: **-**

History: **created: 05.11.2022**

Requirement-ID: **4.3** Requirement Type: **functional** Event: **evaluation**

Description: **the evaluation system shall provide a sample solution**

Rationale: **user needs to see a correct solution for learning effect**

Originator: **user, customer**

Fit Criterion: **validation with wrong solutions**

Customer Satisfaction: **5** Customer Dissatisfaction: **4**

Dependencies: **-** Conflicts: **-**

Materials: **-**

History: **created: 05.11.2022**

Requirement-ID: **4.4** Requirement Type: **non-functional** Event: **evaluation**

Description: **the evaluation system handles malicious user input**

Rationale: **the evaluation system is available most of the time**

Originator: **customer, administrator**

Fit Criterion: **the evaluation system crashes only at 1% of cases when presented with specially crafted or randomly generated user input**

Customer Satisfaction: **2** Customer Dissatisfaction: **4**

Dependencies: **-** Conflicts: **-**

Materials: **-**

History: **created: 05.11.2022**

Requirement-ID: **4.5** Requirement Type: **non-functional** Event: **evaluation**

Description: **the evaluation system shall perform tasks with low latency**

Rationale: **high user experience and satisfaction is ensured**

Originator: **user, customer**

Fit Criterion: **in tests the evaluation system returns after at least 1 second**

Customer Satisfaction: **1** Customer Dissatisfaction: **4**

Dependencies: **-** Conflicts: **-**

Materials: `https://www.selenium.dev/`

History: **created: 05.11.2022**

2 CRC-Cards

Die CRC-Cards, dienen dazu, das System, entsprechend der Requirements, in verschiedene Bereiche/Klassen zu unterteilen und die Interaktionen zwischen diesen festzulegen. Dazu haben wir uns anhand unserer ersten Vorstellungen und der Snow Cards überlegt, wie man das System so unterteilen kann, dass es möglichst übersichtlich bleibt und möglichst einfach und effizient umzusetzen ist. Damit haben wir den Grundstein für die Implementierung des Projektes gelegt.

Database	-
create, read, update, delete exercise data create, read, update, delete user data	AbstractExercise AbstractUser

Server	-
serve web pages and related files provide API interface translate API requests into SQL queries	Database AbstractUser AbstractExercise

AbstractUser	-
reflect database structure provide database interface define API route	Database Server

User	AbstractUser
solve exercises	AbstractExercise

Administrator	AbstractUser
manipulate exercise data evaluate solutions	Database ExerciseCreator AbstractExercise User LearningProgress

AbstractExercise	-
reflect database structure provide database interface define API route	Database Server

GapTextExercise	AbstractExercise

SyntaxExercise	AbstractExercise

ParsonsPuzzleExercise	AbstractExercise

FindTheBugExercise	AbstractExercise

DocumentationExercise	AbstractExercise

OutputExercise	AbstractExercise

ProgrammingExercise	AbstractExercise

ExerciseTracker	-
track solving time	AbstractExercise

ExerciseEvaluator	-
evaluate solution store solution attempt and solving time	SourceCodeEvaluator AbstractExercise ExerciseTracker Database Administrator

SourceCodeEvaluator	-
parse user code execute user code in sandbox	ExerciseEvaluator AbstractExercise ExerciseTracker Database Administrator

LearningProgress	-
assemble learning progress create graphical representation	Database User AbstractExercise

ExerciseCreator	-
create exercise with hints and solution	Database AbstractExercise

HomePage	-
provide login	AbstractUser LoginArea

LoginArea	-
display components	TextField LoginButton SignUpButton ForgotPasswordButton

TextField	-
accept user input obfuscate password input	

LoginButton	-
trigger login process	Server

SignUpButton	-
trigger user creation process	Server

ForgotPasswordButton	-
trigger password reset process	Server

3 Testing

Die Tests sind einer der wichtigsten Aspekte, der erfolgreichen Umsetzung eines Programmierprojektes. Sie stellen sicher, dass Funktionalität („Bugfreiheit“) bei

zunehmender Komplexität und voranschreitenden Veränderungen des Quellcodes gewährleistet bleibt. Das gibt sowohl dem Benutzer, als auch dem Entwickler eines Systems die Sicherheit, dass dieses zu einem gewissen Grad benutzbar ist. Um die Testfälle zu erarbeiten haben wir uns zunächst darüber Gedanken gemacht, unter welchen Bedingungen unsere Web-App benutzbar sein soll und mit welchen Sonderfällen sie dadurch Konfrontiert werden könnte. Mit diesen Informationen haben wir dann Testfälle erarbeitet, die im Späteren dann implementiert wurden, um automatisches Testen zu ermöglichen.

Evaluator Tests

GapTextExercise tests

`test_evaluate_gap_text`

- tests whether solutions for gap text exercises get correctly evaluated
- gap text exercises are exercises where one has to fill in blanks

SyntaxExercise tests

`test_evaluate_syntax`

- tests whether solutions for syntax exercises get correctly evaluated
- syntax exercises are exercises where one has to point out syntax errors in a given code snippet

ParsonsPuzzleExercise tests

`test_evaluate_parsons_puzzle`

- tests whether solutions for parsons puzzle exercises get correctly evaluated
- parsons puzzle exercises are exercises where one has to arrange code blocks in the correct order

FindTheBugExercise tests

`test_evaluate_find_the_bug`

- tests whether solutions for find the bug exercises get correctly evaluated
- find the bug exercises are exercises where one has to point out bugs in a given code snippet

DocumentationExercise tests

`test_evaluate_documentation`

- tests whether solutions for documentation exercises get correctly evaluated
- documentation exercises are exercises where one has to describe what a given code snippet does
- this exercise type can not be evaluated automatically, instead we mark them as `pending` and an admin evaluates the solution

OutputExercise tests

`test_evaluate_output`

- tests whether solutions for output exercises get correctly evaluated
- output exercises are exercises where one has to describe the output of a given code snippet
- this exercise type can not be evaluated automatically, instead we mark them as `pending` and an admin evaluates the solution

ProgrammingExercise tests

`test_evaluate_programming`

- tests whether four different versions of one code example get correctly evaluated
- the test cases are two Python tests and two Java tests, both in a correct and an incorrect version

Exercise Tests

GET tests

`test_get_existing`

- tests whether the system correctly returns an exercise requested by ID
- input: HTTP request with `{"exercise_id": 1}`
- expected output: all fields of the exercise and HTTP status `200`

`test_get_non_existing`

- tests whether the system can tell that an exercise does not exist
- input: HTTP request with `{"exercise_id": -2}`
- expected output: `{}` with HTTP status `200`

`test_get_existing_user`

- tests whether the system correctly returns an exercise when authenticated with an user token
- input: HTTP request with `{"exercise_id": 1}` and an user token
- expected output: all fields of the exercise and HTTP status `200`

`test_get_existing_no_login`

- tests whether the system rejects the attempt to query exercise information without being logged in
- input: HTTP request with `{"exercise_id": 1}` but without user token
- expected output: `{"message": "Login required"}` with HTTP status `401`

POST tests

`test_post_success`

- tests whether it is possible to create an exercise with an admin token
- input: HTTP request with

```
{
    "exercise_title": "My Exercise",
    "exercise_description" : "This is a good Test
       example!",
    "exercise_type": 1,
    "exercise_content":"1+1="
}
```

- expected output:

```
{
    "exercise_id": <exercise_id>,
    "exercise_title": "My Exercise",
    "message": "The exercise was created
       successfully"
}
```

  with HTTP status `201`

`test_post_no_access`

- tests whether the system rejects the attempt to create an exercise without an admin token
- input:

```
{
    "exercise_title": "My Exercise",
    "exercise_description" : "This is a good Test
       example!",
    "exercise_type": 1,
    "exercise_content":"1+1="
}
```

- expected output: `{"message": "Login required"}` with HTTP status `401` or `{"message": "No Access"}` with HTTP status `403`

`test_post_existing`

- tests whether the system rejects the attempt to create a duplicate exercise
- input: HTTP request with `exercise_title` which already exists
- expected output: `{"message": "An exercise with this title already exists"}` with HTTP status `409`

`test_post_without_req_arg`

- tests whether the system rejects the attempt to create an exercise with incomplete data
- input: HTTP request with missing fields
- expected output: response about the missing field with HTTP status `400`

PUT tests

`test_put_existing`

- tests whether a request with an admin token can change exercise information
- input: HTTP request with new `exercise_content` field
- expected output: `{"message": "Successfully changed exercise with exercise_id <exercise_id>"}` with HTTP status `200`

`test_put_existing_user`

- tests whether the system rejects the attempt to change exercise information with an user token
- input: HTTP request with new `exercise_content` field and an user token
- expected output: `{"message": "No Access"}` with HTTP status `403`

`test_put_existing_no_login`

- tests whether the system rejects the attempt to change exercise information without any token at all
- input: HTTP request with new `exercise_content` field without any token
- expected output: `{"message": "Login required"}` with HTTP status `401`

`test_put_without_req_arg`

- tests whether the system rejects the attempt to change exercise information with incomplete fields
- input: HTTP request with missing `exercise_id` field
- expected output: `{"message": "exercise_id is missing"}` with HTTP status `400`

`test_put_non_existing`

- tests whether the system rejects the attempt to change information of an exercise which does not exist
- input: HTTP request with new exercise information and an ID which does not exist
- expected output: `{"message": "Exercise with exercise_id <exercise_id> does not exist"}` with HTTP status `404`

DELETE tests

`test_delete_existing`

- tests whether it is possible to delete an exercise
- this test also tests about insufficient privileges (i.e. using an user token or no token)
- input: HTTP requests with admin token, with user token or without a token but always with an existing exercise ID
- expected output: depending on the request `{"message": "Login required"}` with HTTP status `401` when the token is missing, `{"message": "No Access"}` with HTTP status `403` when an user token is provided or `{"message": "Successfully deleted exercise with exercise_id <exercise_id>"}` with HTTP status `200` when the request was successful

`test_delete_non_existing`

- tests whether it is not possible to delete an exercise which does not exist
- input: HTTP request with `exercise_id: -2`
- expected output: `{"message": "Exercise with exercise_id <exercise_id> does not exist"}` with HTTP status `404`

Java Sandbox Tests

`test_init_constructor`

- tests correct initialization of an instance of the ExecuteJava class

`test_jvm_connection`

- tests whether a connection to the JWM is present

`test_correct_user_code`

- tests behavior when presented with correct user code
- user code is correct if there are no syntax, compilation or runtime errors and the code works as intended

`test_wrong_user_code`

- tests behavior when presented with wrong user code
- there are no errors, but the code does not produce the desired output

`test_timeout`

- tests whether the system aborts long running evaluations after a fixed amount of time

`test_not_compilable`

- tests behavior when user code results in a syntax or compilation error

`test_not_executable`

- tests behavior when user code encounters a runtime error

Login Tests

`test_login_success`

- tests whether login with correct credentials is possible
- input: HTTP request with correct user name and word
- expected output: `{"message": "Welcome <user_name>!"}` with HTTP status `200` and a session cookie

`test_login_fail`

- tests whether login fails when presented with wrong word or unknown user
- input: HTTP requests with either a wrong word or an user which does not exist
- expected output: `{"message": "Incorrect user name or password"}` with HTTP status `401` but without a session cookie

`test_login_without_req_arg`

- tests whether the system prevents a login with missing fields
- input: HTTP requests with missing `user_name` and `user_` respectively
- expected output: message about missing field and HTTP status `400` with no session cookie

Python Sandbox Tests

`test_correct_user_code`

- tests user code which can be executed and produces the desired output

`test_wrong_user_code`

- tests user code which can be executed but does not produce the expected output

`test_timeout`

- tests whether the system aborts evaluations which run longer than a certain amount of time

`test_not_compilable`

- tests user code wich produces a compiler error
- despite python being an interpreted language is is possible to precompile certain parts of a script

`test_not_executable`

- tests user code which produces a runtime error

Solution Tests

GET tests

`test_get_existing_by_id`

- tests whether it is possible to get a solution based in its ID

`test_get_existing_by_user_id`

- tests whether it is possible to get a solution based on the ID of the user who submitted the solution

`test_get_existing_by_exercise_id`

- tests whether it is possible to get a solution based on the ID of the exercise the solution was submitted for

`test_get_existing_by_date`

- tests whether it is possible to get a solution based on the date it was submitted on

`test_get_existing_by_duration`

- tests whether it is possible to get a solution based on duration the user needed to solve the exercise

`test_get_existing_by_correct`

- tests whether it is possible to get a solution based on state whether the solution is correct or not

`test_get_existing_by_pending`

- tests whether is is possible to get a solution based on its state whether manual evaluation is pending

`test_get_non_existing_by_id`

- tests whether the system handles requesting a non existing solution ID

`test_get_non_existing_by_user_id`

- tests whether the system handles requesting a solution from user who has not submitted as solution yet

`test_get_non_existing_by_exercise_id`

- tests whether the system handles requesting a solution for an exercise which has not got a solution submitted for

`test_get_non_existing_by_date`

- tests whether the system handles requesting a solution from a date when no solutions were submitted

`test_get_non_existing_by_duration`

- tests whether the system handles requesting a solution which did not need the requested amount of time to solve

`test_get_non_existing_by_correct`

- tests whether the system handles requesting a solution when there are no solutions which have the requested status

`test_get_non_existing_by_pending`

- tests whether the system handles requesting a solution when there are no solutions which have the requested status

`test_get_restrict_page_size`

- tests whether restricting the number of returned items works

POST tests

`test_create_as_user`

- tests whether it is possible to submit a solution with a user token

`test_create_as_admin`

- tests whether it is possible to submit a solution with an admin token

`test_create_without_token`

- tests whether it is possible to submit a solution without a token
- only logged in users and admins can submit solutions

`test_create_without_user_id`

- tests whether it is possible to submit a solution which does not reference the user who submitted the solution

`test_create_without_exercise_id`

- tests whether it is possible to submit a solution which does not reference the exercise it was submitted for

`test_create_with_date_in_past`

- tests whether it is possible to submit a solution with a start date in the past
- as the database entry gets created right after submitting the solution the start of the working time has to be close to the current and can not be far in the past

`test_create_with_negative_duration`

- tests whether it is possible to submit a solution with a negative amount of time the user needed to solve the exercise
- negative time intervals are not possible in reality

`test_create_with_empty_text`

- tests whether it is possible to submit a solution without the solution text
- providing no solution to an exercise is useless

`test_create_with_malformed_text`

- tests whether it is possible to submit a solution where the solution text does not adhere to the schema
- there is a XML schema which defines how the solution for every exercise type should be defined

PUT tests

`test_change_existing`

- tests whether it is possible to change the fields of an existing solution

`test_change_non_existing`

- tests whether it is possible to change the fields of a non existing solution
- when trying to change the fields a solution which does not exist an error code gets returned

`test_change_as_user`

- tests whether it is possible to change the fields of a solution with an user token
- only admins are allowed to change solution fields

`test_change_as_admin`

- tests whether it is possible to change the fields of a solution with an admin token

`test_change_without_token`

- tests whether it is possible to change the fields of a solution without a token
- changing a solution requires an admin token

`test_change_to_empty_user`

- tests whether it is possible to change a solution so it does not reference an user

`test_change_to_empty_exercise`

- tests whether it is possible to change a solution so it does not reference an exercise

`test_change_to_date_in_past`

- tests whether it is possible to change a solution so that the start date of it is in the past
- has to be possible as changing solution fields always happens after the solution was created

`test_change_to_negative_duration`

- tests whether it is possible to change a solution so that its amount of time the user needed to solve the exercise is negative
- negative time intervals are not possible in reality

`test_change_to_empty_text`

- tests whether it is possible to change a solution so that its solution text is empty
- providing no solution to an exercise is useless

`test_change_to_malformed_text`

- tests whether it is possible to change a solution so that its solution text does not adhere to the schema
- there is a XML schema which defines how the solution for every exercise type should be defined

DELETE tests

`test_delete_existing`

- tests whether it is possible to delete an existing solution

`test_delete_non_existing`

- tests whether the system handles deleting a non existing solution
- when deleting a solution which does not exist an error code gets returned

`test_delete_as_user`

- tests whether it is possible to delete a solution with an user token
- only admins are allowed to delete solutions

`test_delete_as_admin`

- tests whether it is possible to delete a solution with an admin token

`test_delete_without_token`

- tests whether it is possible to delete a solution without a token
- deleting a solution requires an admin token

User Tests

GET Tests

`test_get_existing_by_id`

- tests whether the system finds and returns an existing user correctly based on the users ID
- input: HTTP request with `user_id=2`
- expected output: `{"user_id": 2, "user_name": "tuser", "user_mail": "tuser@example.com", "user_role": "User"}` and HTTP status `200`

`test_get_existing_by_name`

- tests whether the system finds and returns an existing user correctly based on the users name
- input: HTTP request with `user_name = "tuser"`
- expected output: `{"user_id": 2, "user_name": "tuser", "user_mail": "tuser@example.com", "user_role": "User"}` and HTTP status `200`

`test_get_existing_by_mail`

- tests whether the system finds and returns an existing user correctly based on the users mail address
- input: HTTP request with `user_mail = "tuser@example.com"`
- expected output: `{"user_id": 2, "user_name": "tuser", "user_mail": "tuser@example.com", "user_role": "User"}` and HTTP status `200`

`test_get_existing_by_role`

- tests whether the system finds and returns an existing user correctly based on the users role
- input: HTTP request with `user_role = 3`
- expected output: `{"user_id": 2, "user_name": "tuser", "user_mail": "tuser@example.com", "user_role": "User"}` and HTTP status `200`

`test_get_non_existing_by_id`

- tests how the system handles an request with an ID wich is not present in the database
- input: HTTP request with `user_id = 25`
- expected output: `{}` and HTTP status `200`

`test_get_non_existing_by_name`

- tests how the system handles an request with a name which is not present in the database
- input: HTTP request with `user_name = "nuser"`
- expected output: `{}` and HTTP status `200`

`test_get_non_existing_by_mail`

- tests how the system handles an request with a mail address which is not present in the database
- input: HTTP request with `user_mail = "nuser@example.com"`
- expected output: `{}` and HTTP status `200`

`test_get_non_existing_by_role`

- tests how the system handles an request with a role which is not present in the database
- input: HTTP request with `user_role = 4`
- expected output: `{"message": {"user_role": "4 is not a valid UserRole"}}` and HTTP status `400`

`test_restrict_page_size`

- tests whether the system correctly returns only the amount of elements which is specified in `user_limit` parameter
- input: HTTP request with `user_limit = 5`
- expected output: response contains exactly 5 users and HTTP status `200`

POST Tests

`test_create_user`

- tests whether the system correctly creates a new user
- input: HTTP request with

```
{
    "user_name": "Josslin Aloj",
    "user_mail": "Josslin.Aloj@example.com",
    "user_pass": "ian80"
}
```

- expected output:

```
{
    "message": "The user was created successfully",
    "user_id": <user_id>,
    "user_name": "Josslin Aloj",
    "user_mail": "Josslin.Aloj@example.com",
    "user_role": "User"
}
```

  and HTTP status 201

`test_create_admin`

- tests whether the system correctly creates a new admin
- input: HTTP request with

```
{
    "user_name": "Thurmon Uli",
    "user_mail": "Thurmon.Uli@example.com",
    "user_pass": "Pi5ta",
    "user_role": 2
}
```

- expected output:

```
{
    "message": "The user was created successfully",
    "user_id": <user_id>,
    "user_name": "Thurmon Uli",
    "user_mail": "Thurmon.Uli@example.com",
    "user_role": 2
}
```

  and HTTP status 201

`test_create_user_without_token`

- tests whether it is possible to create a user with out a token (without being logged in)
- necessary to allow new users to create a account (they don't have a login yet, so they can't request a token)
- input: HTTP request with `{"user_name": "Kon Archy", "user_mail": "Kon.Archy@example.com", "user_pass": "Fip5k"}`
- expected output:

```
{
    "message": "The user was created successfully",
    "user_id": <user_id>,
    "user_name": "Kon Archy",
    "user_mail": "Kon.Archy@example.com",
    "user_role": "User"
}
```

  and HTTP status `201`

`test_create_admin_without_token`

- tests whether the system rejects the attempt to create an admin without any token
- input: HTTP request with `{"user_name": "Detleff Deric", "user_mail": "Detleff.Deric@example.com", "user_pass": "eNg9h", "user_role": 2}`
- expected output: `{"message": "Login required"}` and HTTP status `401`

`test_create_admin_with_user_token`

- tests whether the system rejects the attempt to create an admin without being the SAdmin (the SAdmin token is send in the request header)
- input: HTTP request with `{"user_name": "Eldrige Ernesto", "user_mail": "Eldrige.Ernesto@example.com", "user_pass": "Xah8e", "user_role": 2}`
- expected output: `{"message": "No Access"}` and HTTP status `403`

`test_create_duplicate_user`

- tests whether the system rejects a new user with an existing mail (mail addresses have to be unique system wide)
- input: two HTTP requests with `{"user_name": "Geoffrey Tretan", "user_mail": "Geoffrey.Tretan@example.com", "user_pass": "Abeo0"}`
- expected output: `{"message": "A user with this mail already exists"}` and HTTP status `409`

`test_create_user_with_empty_name`

- tests whether the system rejects the attempt to create an account with an empty name
- input: HTTP request with `{"user_name": "", "user_mail": "Gofried.Dietbald@example "user_pass": "koh6P"}`
- expected output: `{"message": "user_name must not be empty"}` and HTTP status `400`

`test_create_user_with_empty_mail`

- tests whether the system rejects the attempt to create an account with an empty mail
- input: HTTP request with `{"user_name": "Rane Loewe", "user_mail": "", "user_pass": "Lohw3"}`
- expected output: `{"message": "user_mail must not be empty"}` and HTTP status `400`

`test_create_user_with_empty_password`

- tests whether the system rejects the attempt to create an account with an empty password
- input: HTTP request with `{"user_name": "Pepin Kuefer", "user_mail": "Pepin.Kuefer@example.com", "user_pass": ""}`
- expected output: `{"message": "user_pass must not be empty"}` and HTTP status `400`

`test_create_user_with_long_name`

- tests whether the system can handle a long name
- `<long_name>` is a string of 1024 random characters
- input: HTTP request with `{"user_name": <long_name>, "user_mail": "long.mail@example.com", "user_pass": "Eiph9"}`
- expected output:

```
{
    "message": "The user was created successfully",
    "user_id": <user_id>,
    "user_name": <long_name>,
    "user_mail": "long.mail@example.com",
    "user_role": 3
}
```

  and HTTP status `201`

`test_create_user_with_strange_name`

- tests whether the system can handle strange names consisting of unicode characters
- `<strange_name>` is a string of 14 unicode characters
- input: HTTP request with `{"user_name": <strange_name>, "user_mail": "strange.mail@example.com", "user_pass": "ai2Ou"}`
- expected output:

```
{
    "message": "The user was created successfully",
    "user_id": <user_id>,
    "user_name": <strange_name>,
    "user_mail": "strange.mail@example.com"
    "user_role": 3
}
```

  and HTTP status `201`

PUT Tests

`test_change_mail_to_existing`

- tests whether the system rejects the attempt to change the mail to an already existing one
- input: HTTP request with `"user_mail": "double.mail@example.com"`
- expected output: `{"message": "A user with this mail already exists"}` and HTTP status `409`

`test_change_to_empty_name`

- tests whether the system rejects the attempt to change to an empty name
- input: HTTP request with `"user_name": ""`
- expected output: `{"message": "user_name must no be empty"}` and HTTP status `400`

`test_change_to_empty_mail`

- tests whether the system rejects the attempt to change to an empty mail
- input: HTTP request with `"user_mail": ""`
- expected output: `{"message": "user_mail must not be empty"}` and HTTP status `400`

`test_change_to_empty_password`

- tests whether the system rejects the attempt to change to an empty pass
- input: HTTP request with `"user_pass": ""`
- expected output: `{"message": "user_pass must not be empty"}` and HTTP status `400`

`test_change_admin_elevation`

- tests whether the system rejects the attempt to raise the user role to Admin without a SAdmin token
- input HTTP request with `"user_role": 2` but no SAdmin token in header
- expected output: `{"message": "No Access"}` and HTTP status `403`

`test_change_sadmin_elevation`

- tests whether the system rejects the attempt to raise the user role to SAdmin without a SAdmin token
- input: HTTP request with `"user_role": 1` but no SAdmin token in header
- expected output: `{"message": "No Access"}` and HTTP status `403`

DELETE Tests

`test_delete_existing`

- tests whether the system correctly deletes an existing user
- input: HTTP request with a previously created used ID
- expected output: `{"message": "Successfully deleted user with user_id <user_id>"}` and HTTP status `200`

`test_delete_non_existing`

- tests whether the system handles deleting a non existing user
- input: HTTP request whit a user ID which does not exist
- expected output: `{"message": "User with user_id <user_id> does not exist"}` and HTTP status `404`

`test_delete_self_as_admin`

- tests whether a request with an admin token can delete an admin account
- input: HTTP request with admin token
- expected output: `{"message": "Successfully deleted user with user_id <user_id>}` and HTTP status `200`

`test_delete_self_as_user`

- tests whether a request with a user token can delete the account associated with the token
- input: HTTP request with user token
- expected output: `{"message": "Successfully deleted user with user_id <user_id>"}` and HTTP status `200`

`test_delete_other_as_admin`

- tests whether a request with an admin token can delete another account
- input: HTTP request with admin token
- expected output: `{"message": "Successfully deleted user with user_id <user_id>"}` and HTTP status `200`

`test_delete_other_as_user`

- tests whether the system rejects the attempt of a request with a user token to delete another account
- input: HTTP request with user token
- expected output: `{"message": "No Access}` and HTTP status `403`