TECHNISCHE UNIVERSITÄT
CHEMNITZ

Testing

Softwaretechnikpraktikum - Meilenstein 3

Fakultät für Informatik
Professur Softwaretechnik

Eingereicht von: Gruppe 5
Einreichungsdatum: 15.01.2023

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

Zusammenfassung

Für den 3. Meilenstein haben wir damit weitergemacht uns wöchentlich im ganzen Team zu treffen und auszutauschen, was jeder erreicht hat und was insgesamt noch gemacht werden muss. Dadurch, dass wir uns in ein Frontend und ein Backend Team aufgeteilt haben, hatten wir Schwierigkeiten bei der Verteilung der Aufgaben des 3. Meilensteins. Das lag daran, dass in unserem Frontend nichts mit Unit-Test getestet werden kann. Allerdings konnten wir die API-Schnittstellen Unit-testen. Deshalb haben wir beschlossen, dass das Frontend-Team sich Testfälle überlegt und das Backend-Team diese für die Klassen, die sie geschrieben haben, implementiert. Diese Vorschläge für Tests waren auch gute Anregungen, allerdings hat sich das Backend-Team auch viele Tests selbst überlegt und umgesetzt. Logischerweise wussten die Backend-Teammitglieder noch besser, was deren Code macht, und was getestet werden kann. Weiterhin kam dann noch die Anfrage, ob wir Dummy-Tests für noch nicht implementierte Methoden und Klassen schreiben könnten. Dies wurde auch für die entsprechenden geplanten Methoden umgesetzt und sowohl vorher als auch hinterher in den Meetings besprochen.

Für die weitere Umsetzung unseres Projekts sind wir bei der vorherigen Gruppeneinteilung geblieben, da sich schon jeder in seiner Rolle eingefunden und spezialisiert hat und es ein sehr großer Aufwand gewesen wäre die Rollen zu tauschen. Dadurch sind wir bedeutend schneller vorangekommen. Da Max und Simon sich um die Struktur und Leitung von Backend beziehungsweise Frontend kümmern und damit mehr Aufwand als die restlichen Teammitglieder haben, haben wir als Gruppe beschlossen, dass die beiden keinen Meilenstein-Vortrag halten müssen, um sie zu entlasten.

Im Backend ist der Stand der Dinge, dass sämtliche POST-, GET-, PUT- und DELETE-Requests implementiert sind und funktionieren. Weiterhin funktioniert sowohl das Login als auch der Registrierungsprozess. Außerdem ist eine Sandbox für die Programmieraufgaben erstellt worden. Aufgabe des Frontends ist es nun diese Funktionen von der Website aus nutzbar zu machen. Zum Beispiel wurde schon Login- und Registrierungs-Seite umgesetzt. Die Home-Page wurde noch einmal überarbeitet und schicker gemacht. Auch wurde eine Lösung für das Syntax-Highlighting bei den Programmieraufgaben gefunden und befindet sich gerade in der Bearbeitung.

Während der Implementierung traten auch immer wieder Probleme und Bugs auf, für die wir GitHub-Issues erstellten und versuchten diese schnell zu lösen. Zum Beispiel funktionierte das Routing auf der Website anfangs nicht richtig, da wir zunächst das Routing-System von sveltekit verwendeten. Der Python-Flask Server kooperierte damit nicht richtig, was auch schon in anderen Programmierprojekten ein Problem war. Dennoch konnten wir letztendlich das Problem durch die Verwendung eines anderen Routing-Systems lösen.

Eine weitere Herausforderung trat auf als Änderungen am Backend vorgenommen und gemerged wurde, die jedoch dafür sorgte, dass Fehler auftraten und der Server nicht mehr startete. Somit konnten alle Entwickler, die von diesem develop-Branch abspalten wollten, ihre Entwicklung nicht mehr testen. Deshalb haben wir es uns nun als Ziel gesetzt, dass der Server auf dem develop Branch immer funktionieren soll. Das wollen wir dadurch erreichen, dass Bugfixes so schnell wie möglich vorgenommen werden und neu gemergede Funktionen sofort getestet werden.

Allgemein stellen wir immer wieder fest, dass Kommunikation sehr wichtig ist und dass gerade dafür unsere Meetings sehr wertvoll sind.

Testing

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`