Assignment 2: Unit Testing

Overview: In conventional software development, unit tests are often the first tests to be written: they are on the lowest level of the testing pyramid, meaning that they are great in number but small in scope. A solid base of unit tests is essential to ensure the reliability of all components.
General notes: This assignment counts towards Lab 1. Deliver solutions to all exercises as defined in the *deliverables* to pass this assignment. When using material outside of the official course material (lecture, slides, tutorials), state the source and critically reflect on its usage. In case of questions, please contact Zeth Danielsson (zeda21@student.bth.se).
Work distribution: At the beginning of each assignment submission, state how the work was distributed among the two team members.

1. Mocking

Mocking is an essential technique in testing and fulfills a specific purpose during unit testing. *Explain the concept of mocking and its application in unit tests.*
Deliverables: The submission to this exercise must contain all of the following:

1. An explanation of mocking.
2. An explanation of the purpose that mocking fulfills in unit testing.

2. Unit testing

In the current version of the EduTask system, the login functionality does not yet require a password. Instead, users can authenticate to the system just by entering an email address of a registered account. This decision has been made to deliver an illustrative prototype earlier and postpone the security-critical password authentication to a later sprint in the development. However, it should at least be ensured that the authentication using just the email address works as intended. *Design, implement, and evaluate unit tests for the `get_user_by_email` function in the file `backend/src/controllers/usercontroller.py`. Follow best practices for unit test design and utilize mocking where applicable.* [1]
Deliverables: The submission to this exercise must contain all of the following:

1. A list of test cases for the method, which has been derived by selecting an appropriate oracle and applying the test design technique.
2. An implementation of these test cases using `Pytest`. Provide a link to the file(s) in your forked repository that contains your test code.
3. A screenshot showing the output of your test code execution.
4. An interpretation of the coverage of the test cases.

Recall that you only need to *detect failure(s)*. If you want, you can optionally *localize the defect(s)* that caused the failure (though this is typically outside of the testing scope) by investigating the code and writing a short explanation about why the failures occurred. However, please to not *correct the defect(s)* yet - this will be part of a later assignment.

[1] You may assume that all Python-native libraries and functions, including for example `print` and the `re` library, are not separate units, but parts of the system under test. You do not need to mock these dependencies.