MTAT.03.295 – Agile Software Development

Regular Exam – 04 January 2022

Part 2. Practice

Notes:

- **You can access any resource on the web**
- **You are not allowed to communicate with anyone during the exam in any way (except with the lecturer).**
- **You MUST create a PRIVATE repository at https://gitlab.cs.ut.ee, and add the teaching staff there as collaborators (with admin rights). You must submit the link to your repository on Moodle (Exam 1: Part 2 (Practice)) with the solution. (Username: orleny85, email: orlenyslp@ut.ee, Username: scott, email: ezequiel.scott@ut.ee)**
- **Your solution must be in the "master" branch of your repository**
- **Please do not make any commits to the repository after the end of the exam. Commits made after the end of the exam (13:45 – EEST time) will not be considered.**
- **IMPORTANT: You MUST commit to the repository each time that you complete a task or requirement in the exam. You can find the minimal list of commits messages that we expect. You are welcome to add other commits, for example, if you fix a task completed before. For each commit missing, you will get a penalization of 20% of the points received corresponding to the task. Note that commits whose messages do not correspond to the task will get 20% penalization. For example, if your commit message says, "Task 2, Completed", be sure that the code submitted corresponds to task 2; however, other parts of the implementation may be updated too.**
- **IMPORTANT: We strongly suggest you not to share your code before the end of the exam to avoid misunderstandings. In the case of projects in which it is evident that the students committed fraud, all the involved students will get 0 points in the exam, meaning that they all fail it. If we observe suspicious behavior either in the implementation or the commits, the students involved will be called for an online interview with the teaching staff. Examples of suspicious behavior can be all the commits happening at the end of the exam or with a non-realistic timeline, too many similarities between projects, etcetera. During the interview, the students will be asked about the code they submitted. Note that, during the interview, we will downgrade the mark of the student for each question not answered correctly, meaning that the student may fail the exam in the meeting.**

Exam Scenario: Loan Request

In this project, you are asked to implement an initial system to handle loan requests. To that end, in the initial release, you must consider the following requirements. The system must store the set of users that may request a loan in a database. Specifically, the system must include the full name, the email address, which should be unique, and the monthly income for each user. To request a loan, users must introduce their email, the loan amount, i.e., the amount of money to borrow, and the loan term, i.e., the number of years to return the borrowed money. Note that only registered users may request a loan. Thus, requests from invalid or non-existing emails are automatically rejected. Also, users may request only once for a loan. Thus, if a user had an accepted loan before, the system must reject any new requests.

Once the loan is submitted, the system initially calculates the interest ratio according to the number of years the user will take to pay it back, i.e., loan term. If the time is between 1 and 5 years, the interest ratio is 2.5%. If it is between 5 and 10 years, the ratio is 3.5%, and between 10 and 15 years, the interest ratio is 5%. Note that the loan term should be a floating number between 1 and 15. Otherwise, the request is automatically rejected. Next, the system calculates the total amount to pay by adding to the loan amount the corresponding fees from the interest ratio. Then, the system accepts the request if the monthly amount to pay, i.e., including the interest ratios, does not exceed the 40% of the user's monthly income. Note that the monthly amount is the result of dividing the total amount by the loan term. If none of the rejection conditions hold, the system accepts the loan. Accordingly, it stores a new loan with the calculated total amount (loan amount plus the interest fees), the loan term (number of years to be paid), and the current date (as the beginning of the agreement). In case of

rejection, the system must always display the following message: "*Sorry, you are not eligible for the loan*". If accepted, the system notifies the user with a success message which must include the calculated interest ratio, the monthly amount the user must pay, and the number of years to pay the loan.

You do not need to implement any user authentication/authorization or user registration. Thus, you must implement only the operation request loan according to the restrictions above.

The task breakdown and corresponding marks are as follows:

- **Task 1 (3 points) [BDD]**: Specification of a Gherkin user story describing the case in which a loan is created successfully. As a minimum, your feature must include the clauses ***Given***, ***And***, ***When***, and ***Then***. Your user story must specify that some users are registered and that at least one loan exists in the databases (providing a table with their data). This is followed by the information required to request a new loan and the action that triggers such creation. Finally, the corresponding success message, including the interest ratio, the monthly amount to be paid, and the number of years to complete the payment.
- **Task 2 (3 points) [BDD – steps implementation]**: Implementation of the white_bread steps described by the user story in Task 1.
- **Task 3 (1 point)**: Setup of the application routes to support operation to request a loan.
- **Task 4 (5 points)**: Setup & implementation of models (via migrations) and seed the database with some initial data based on your models. The seeding of the database must include some registered users and at least one accepted loan related to one of the registered users.
- **Task 5 (10 points)**: Implementation of controllers. You must implement the operations to render the templates to provide the input data for requesting a loan. Indeed, you must implement the request/validation/creation of new loans, including the messages to notify the acceptance/rejection (as described in requirements in this exam).
- **Task 6 (3 points)**: Implementation of the views and templates to introduce the input data for requesting a loan. You do not need to create/render a new template to display the messages. For that, you can write in the flash as we did during the course practical sessions.
- **Task 7 (5 points): [TDD]** Unit/Integration tests for either model/controllers checking the requirements. Specifically, your tests must consist of at least:
    - **T-7.1 (1.5 points)** A negative case in which the loan is rejected because the user that makes the request already has a previously accepted loan. Thus, the system displays an error. Note that your test must setup the input with valid data, so none of the other rejection conditions holds.
    - **T-7.2 (3.5 points)** A positive case in which the loan is accepted and stored in the system. The test must verify the total amount to pay (i.e., the loan amount plus interest ratio) is calculated correctly. Besides, the test must check that the total amount, the loan term, and the current date are stored in the database in the new loan associated with the requesting user.

Minimum Commits – Messages (NOT necessarily in the same order). Note that each message must correspond to a different commit.

- Initial Commit
- Task 1 Completed
- Task 2 Completed
- Task 3 Completed
- Task 4 Completed
- Task 5, Validations Completed
- Task 5, Creation/update of maintenance task Completed
- Task 6 Completed
- Task 7, T-7.1 Completed
- Task 7, T-7.2 Completed