MTAT.03.295 – Agile Software Development

Regular Exam – 14 December 2021

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 (13:00 – 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: Bike Maintenance

After the high demand of the Tartu Smarter Bike Share system, the city government of Tartu now requires tracking the maintenance of the bikes. In this project, you are asked to either extend the Bike-Sharing System or implement a new system to handle the bike maintenance. To that end, in the initial release, you must consider the following requirements.
The system should store a maintenance record for each bike, including the bike identifier, the corresponding maintenance status, and reference all the maintenance tasks performed on that bike. The maintenance status of a bike can be OPERATIONAL (the bike works properly), INOPERATIVE (the bike is broken and needs to be fixed), SCHEDULED (the bike that is scheduled for maintenance) and UNFIXABLE (if the bike is broken and it can't be fixed). A maintenance task includes the date it is scheduled, and the name and phone number of the technician who will perform the maintenance task. To create a maintenance task, a user introduces the bike identifier, the scheduled date, and the technician details. Then, the system validates that none of the input fields are empty. Similarly, the system validates the bike exists in the database and its maintenance status is OPERATIONAL or INNOPERATIVE, according to its maintenance plan. In case of failure, the system should display an error message. The message should contain what attribute is empty or the identifier of the bike that is not found, depending on the error. If no errors exist and a maximum of 10 maintenance tasks have been performed on that bike. Then, the system should update the status in the maintenance record of the bike as UNFIXABLE, no maintenance task is created, and a message is displayed notifying the user. Finally, suppose the maximum of 10 maintenance

tasks has not been reached. In that case, the system updates the status to SCHEDULED, a new maintenance task is created and updated in the database. Besides, a success message is displayed to the user, which should include the bike identifier and the date on which the maintenance task will be performed.

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 maintenance task is created successfully for a given bike. As a minimum, your feature must include the clauses ***Given***, ***And***, ***When***, and ***Then***. Your user story must specify that the maintenance record and some maintenance tasks exist in the databases (providing a table with their data). Followed by the information required to create a new maintenance task and the action which triggers such creation. Finally, the corresponding message to display to the user, including the bike identifier and the date received as input.
- **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 the creation of the maintenance task.
- **Task 4 (5 points)**: Setup & implementation of models (via migrations) and seeding the database with some initial data based on your models. The seeding of the database must include both maintenance record and maintenance task.
- **Task 5 (10 points)**: Implementation of controllers. You must implement the operations to render the templates to provide the input data for creating a maintenance task. Indeed, you must implement the validation/creation of new maintenance tasks, including the messages to notify the creation/rejection (as described in requirements in this exam).
- **Task 6 (3 points)**: Implementation of the views and templates to introduce the input data for creating maintenance tasks. 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: **T1 (1.5 points)** A negative case in which the bike identifier is not found. Thus, the system displays an error. **T2 (3.5 points)** A positive case in which the maintenance task is created successfully. The test must check that the status of the maintenance record is updated to SCHEDULED. Besides, a new maintenance task is stored in the database associated with the maintenance record corresponding to the bike identifier provided as input.

Note that you do not need to implement any user authentication, so anyone can perform the operation for creating maintenance tasks.

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, T1 Completed
- Task 7, T2 Completed