Rate University Application

1. The Problem

It should be possible for students to give feedback and rate the courses they attend. This will improve the quality of the courses in the future and also help students choose the courses to register based on the feedback given by previous students.

2. Scenarios

Ben, is studying computer science at UPT. He needs to find information about available courses. Based on the information and course schedule, Ben can find the courses that he is interested in. Ben wants to be sure before registering for a course and he reads all recent feedback (description and rating) given for a specific course by students who attended the course before. He is mostly interested in top-rated courses since he believes he will get more benefits from them.

Ben wants also to leave feedback for courses he attended recently so that he can share his experience with other students.

3. Functional / Non-Functional Requirements

The following functional requirements (FR) and nonfunctional requirements (NFR) have to be addressed in the project *

FR1: Create a new account and sign in.

FR2: Search for available courses: A student can see all courses. He is able to join a course. He can also drop a course.

FR3: Check course details: A student can see details about a course such *as a description of the course, the course lecturer, the course times, the location of the lecture hall,* and also the number of registered students.

FR4: Leave feedback: The student can leave feedback for a course. The feedback consists of a description (not more than 1000 characters) and a rating (from 1 to 5). It is possible only to leave feedback for the courses where the student is already registered. A student can only leave one feedback for a course.

FR5: View feedback: A student can view the list of feedback for any course (whether he/she is registered or not). The list should be ordered by date (recent feedbacks are on top of the list). Also, the average rating is calculated and displayed.

FR6: View top-rated courses: A student can view the 8 top-rated courses.

FR7: See course calendar: A student can see all courses in a calendar

FR8: Remove old feedback: The system should automatically remove any feedback that has been created more than 1 year ago.

**** The functional requirements capture what it is that the system should do, whereas, the nonfunctional requirements capture properties that the system should have***

NFR1: Usability: The app should be intuitive to use and the user interface should be easy to understand. All interactions should be completed in less than three clicks.

NFR2: Target platform: The app has to be developed in Java.

NFR3: The version control system must be Git and the hosting platform must be GitHub.

Additional constraints:

- The application can be created as a **web / desktop** application

4. Target Environment

The application should be demonstrated in Java.

For this assignment, you have to build teams of **4 members that work together**.

In this application we have only the user (student) role implemented. Data that must be added by an administrator (such as courses) should be inserted directly into the database. In this way, you can use the data to test your application.

Part I (Requirement specification: AGILE METHODOLOGIES (SCRUM) - SPECIFY USER STORIES)

Consider requirements specification ***"Rate University Application"*** *(pages 1,2 in this document)*

1. Role Product Owner: Create the **product backlog --** *Write all necessary user stories in order to create the rate-university application (Document 1)*
2. Role SCRUM Master: For each sprint, specify the **sprint backlog. (**Stories that depend on each other should not be inserted in the same sprint – except for the cases when these stories are so small that can be implemented in the same sprint but in not overlapping times) *(Document 2)*

Suppose that the developer team is composed of 5 developers. Each sprint lasts three weeks.

Remember that:

- a user story is implemented by one developer
- should be implementable in one sprint (including also the time needed to test the story).
- during the same sprint one developer can implement more than one story – in cases when the stories are small.

Each story should have a point number. In order to specify this number, you should consider the complexity and the time needed to implement the story. One day is converted to 0.5 points.

(Read the uploaded document: "TaskAgile - Example Agile Requirements.pdf". This document can help you write user stories)

Part II (Implementing/Testing Software)

Consider requirements specification ***"Rate University Application"***

- Implement the application using Java as a programming language and a relational database

  Apply what you have learned about software architecture, software design, and internal quality.

  The code should not only work but also it should be written in a way that is easily maintainable (readable, understandable…). Apply the refactoring patterns discussed in the lectures.

- Write Unit/Integration tests using the Junit testing framework in order to achieve not less than 50% of code coverage

  The tests should be maintainable (have a good internal quality)