Lab/Project 3 [LAPR3] – Project Assignment 2018/2019

V1.0, December 10th

Abstract

A software company needs to develop a software service and/or application that supports a bicycle sharing business. This service should allow managing users, bicycles, bicycle parks and bicycle's pickup and return process.

Table of Contents

1 LAPR's Goals and Context

In this project, students should be able to apply concepts of analysis, modelling and object-oriented programming to develop a Java service that supports the management of a bicycle sharing business.

In compliance with good practices learned and applied during the first part of the semester, namely on the course units of Applied Physics (FSIAP), Data Structures (ESINF) and Databases (BDDAD), an iterative and incremental development process must be used. An agile methodology based on Scrum must be applied to manage each team's work during each one-week sprint.

To increase software's maintainability, analysis best practices, Object-Oriented (OO) software design must be adopted and implementation should follow a Test-Driven Development (TDD) approach.

Technical documentation must be produced during the project by using Javadoc documentation and the Readme.MD file in your repository.

2 Problem Description

A bicycle sharing company needs a software service and/or application that allows managing its bicycles, bicycle parks and users.

The company owns several bicycle parks, scattered around town, that are used for parking bicycles when those are not in use.

There are different kinds of bicycles to suit different types of costumers. There are road, mountain and electrically assisted bicycles.

For users to gain access to the sharing service, the user registration has a cost. From there on, depending on the time each bicycle is loaned, a fee can be applied.

To pick-up bicycles, users must request the application to unlock one of the available bicycles at the park. When dropping-off a bicycle, the user should be notified that the bicycle was properly left locked.

There is also a system for crediting points based on certain user actions. For example, at the moment, when a user picks-up a bicycle and leaves it on park situated in a place with a higher elevation/altitude points are credited to the user.

User's monthly cost is issued to the user, by using invoices. Users' points may be used to partially pay the invoice. After paying the invoice, users should get a receipt.

3 Minimum Viable Product

The goal of this assignment is to achieve a Minimum Viable Product in incremental steps. For this purpose, the work will be divided in three Sprints:

- Sprint 1 – December 10th to December 14th
- Sprint 2 – December 17th to December 21st
- Sprint 3 – January 3rd to January 9th

For each Sprint, a description for a minimum viable product is provided to students. At the end of each sprint, each team should be able to have all the requirements fulfilled.

Teams should be able to write their own user stories on the backlog, size them and assign them over to each team member.

3.1 Sprint 1

An administrator should be capable of managing the bicycles and parks of the business. This implies adding, removing or updating existing parks as well as bicycles.

Each park has a geographical location using Decimal Degrees (DD)[1]. For example, given "Trindade Metro Station", its representation in DD corresponds to 41.152699, -8.609267.

Bicycles can have different types (mountain, road and electrically assisted). Each bicycle is identified by a unique number in the system.

Each park has a maximum capacity for different types of bicycles. For example, road and mountain bicycles may be parked on the same slots, but electrically assisted bicycles may only be parked on places that allow charging the battery.

Users should interact with the application to register in the system. Every registration requires an initial cost, a credit card number and the user's height and weight.

Given a user's location, the application should return a list of the nearest bicycle parks. Users can check on the application for available bicycles at a given park.

Users may use the application to check how far is another park.

[1] https://support.google.com/maps/answer/18539?co=GENIE.Platform%3DDesktop&hl=en

The application should allow users to request a bicycle unlock at a given park. At any time, only one bicycle may be unlocked by a user. When requesting an electrically assisted bicycle, the system should suggest bicycles that have a higher battery charge level.

The application should be able to calculate the amount of electrical energy required to travel from one park to another when using electrically assisted bicycles. When a destination park is given, this operation should be used to suggest a bicycle with enough range plus 10%, to get to the destination park as opposite to suggesting bicycles with the highest battery charge.

Users may use the application to check if a destination park has any free parking places for their currently loaned bicycle.

When parking a bicycle, users should receive an e-mail stating that the bicycle is correctly locked and for how long it was taken.

Also, when requested, the application should be able to return a projection for the total amount of calories burnt between two bicycle parks, according to the bicycle used and considering the altitude/elevation between the initial and final point.

4 Non-Functional Requirements

This section describes some of the non-functional requirements.

4.1 Users & Roles

All users must be registered in the system.

4.2 Other requirements

Business logic validation should take place when registering or updating data.

All developed services and applications should persist the data in a remote SGBD.

To maximize interoperability with other existing and/or developing systems, the software main core should be written in Java.

The class structure must be designed to allow easy application maintenance and addition of new features and following good OO design practices.

The following development requirements must be met to achieve the highest possible grading:

- JUnit Code Coverage should be above 95%;
- PIT Mutation Testing Coverage should be above 90%;
- A maximum of 5% Code Duplication is allowed;
- A maximum of 5-day Technical Debt is allowed.

The following development requirements must be met for the project to be graded:

- JUnit Code Coverage should be above 85%;
- PIT Mutation Testing Coverage should be above 80%.

Software products that do not compile on Jenkins, have an automatic grading of 0 (zero).

Software products that show up as “Failed” on SonarQube, have an automatic grading of 0 (zero).

No User Interface (UI) is requested for the system administration. A UI may be developed for the users’ application, but it is not necessary. In both cases, requirements should be fully tested by using Unit Tests.

5 Development Process

The software development process should be iterative and incremental while adopting good design practices (e.g. GRASP and SOLID principles), coding standards and using a Scrum approach.

5.1 Unit Testing

The implementation process must follow a TDD (Test Driven Development) approach. Unit tests should be developed to validate all domain classes.

Unit tests are based on application design (e.g. SDs & CD). The final evaluation of the project will include an analysis of the quality of testing and the use of a test-driven development approach.

Code coverage and mutation coverage are based on unit testing and is performed using quality assurance tools.

5.2 Tasks/Issues & Planning

For this project, task creation, division and planning must be used with Jira issues. Each user story should be created, sized and assigned to team members. For each user story, three issues/task must be created. Each task should focus on:

- **Analysis:** This task should focus on the Use Case Diagram and System Sequence Diagram for each user story. If necessary, possible changes to the Domain Model;
- **Design:** This task should focus on the Class Diagram, Sequence Diagram and Entity-Relationship Diagram;
- **Implementation:** This task should focus on implementing Test Cases and Code.
- **Review:** This task should focus on reviewing the implementation.

For example, for a user story, the following task/issues should be created:

- User Registration [**Analysis**]
- User Registration [**Design**]
- User Registration [**Implementation**]
- User Registration [**Review**]

6 Deliverables

This section describes all the deliverables necessary for the project. Failure to comply with these may invalidate the project's assessment.

6.1 Weekly Delivery

Every week, until each next Monday (except for the first week), students should fill in and submit a self/peer-assessment enquiry.

6.2 Final Delivery

Your project should always be up to date on Bitbucket.

The version that will be assessed in the end is the one with the commit time closest to the deadline. Nevertheless, **all team members** should submit the work on Moodle.

At the end of the project (January 9th, 11.55 p.m.), students must submit on LAPR's Moodle a final delivery that should contain the following zip files:

1. The project repository (all folders in the repository), in a single ZIP file named **LAPR-YYYY-GXXX-REPOSITORY[2].zip,** containing the following technical documentation:
    a. The Project Report should be written in the project's Readme.md file.
    b. Requirements Engineering:
        i. Use Case Diagram;
        ii. System Sequence Diagram (SSD);
    c. Engineering Analysis:
        i. Domain Model (DM);
    d. Engineering Design:
        i. Class Diagram (CD);
        ii. Sequence Diagram (SD);
        iii. Entity-Relationship Diagram (ER);

[2] Where YYYY stands for year, e.g. 2016 for 2016-2017 school year and XXX stands for group number e.g. 001.

7 Assessment

Assessment is performed everyday while students are in class and receive immediate feedback. The project's final assessment takes place on January 10th and 11th according to a schedule that will be provided on Moodle.

7.1 Self/Peer-Assessment

Every week, until each next Monday (except for the first week), students should fill and submit a self/peer-assessment enquiry.

7.2 Issues Assessment

This assessment is performed according to criteria that will be available on Moodle.

7.3 Commit Messages Assessment

Bitbucket will be connected to Jira's issue management system. Therefore, Git commit messages should include a description, a keyword and the task/issue number according to Jira Smart Commits[3]. An example commit message is presented next, where <ISSUE_KEY> should be replaced by the issue ID on Jira: **"<ISSUE_KEY> #close User Registration [Analysis]"**

GIT commit messages are rated using a [0-5] grading system:

- 5 – All commits have a well-formed commit message
- 4 – At least 90% of commits have a well-formed commit message
- 3 – About 50% of commits have a well-formed commit message
- 2 – Not Applicable
- 1 – About 25% of commits have a well-formed commit message
- 0 – Less than 25% of commits have a well-formed commit message

7.4 Code Quality Assessment

Code quality assessment is continually performed on your project. It takes into consideration the following measures:

[3] https://confluence.atlassian.com/fisheye/using-smart-commits-960155400.html

- Code Duplication;
- Technical Debt;
- JUnit Code Coverage;
- PIT Mutation Testing Coverage.

On each day, from December 17th to January 9th, the last commit pushed to the repository must leave the project in a state that enables it to be built with success on Jenkins and have the "Passes" state on Sonarqube.

Failure to comply with this rule, induces a group penalty of 0,125 points per day on a scale of 0 to 20 on your project final grade, up to a maximum of 3 points.

7.5 Plagiarism

Any attempt of copy or plagiarism (using third-party code) not explicitly mentioned in the report, will be heavily penalized and may lead to project annulment. Failure to comply with these policies and procedures will result in disciplinary action.

8 Relevant Hyperlinks

- Bitbucket
  - https://bitbucket.org/
- Jenkins
  - https://jenkins.dei.isep.ipp.pt/
    - Note: login with student number (e.g. 1010101)
- SonarQube
  - https://sonarqube.dei.isep.ipp.pt

9 Revision History

V1.0	Initial release.

Notes: new changes are underlined and removed content are strikethrough.