Lab/Project 3 [LAPR3] – Project Assignment 2019/2020 V1.6, December 15th

Abstract

A software company needs to develop a product that supports ride-sharing businesses. This service should allow managing users, bicycles, electric scooters, parks and 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 ride-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 is used. An agile methodology based on Scrum must be applied to manage each team's work during each one-week sprints.

To increase software's maintainability and 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 ride-sharing company needs a product that allows managing its bicycles, electric scooters, parks and users.

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

The company provides manual bicycles with three gears and electric scooters (i.e. city and off-road) to suit different types of costumers.

Registration on the ride-sharing service has an initial cost. From there on, depending on the time each vehicle is loaned, a fee can be applied.

To pick-up vehicles, users must request the application to unlock one of the available at a park. When dropping-off, the user should be notified that the vehicle was properly left locked. When users do not properly leave their loaned vehicles at available parks, they are penalised.

There is also a system for crediting points based on certain user actions. For example, at the moment, when a user picks-up an electric scooter and leaves it in less that 15 minutes, points are credited to the user; 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, etc..

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~~ two Sprints:

- Sprint 1 – December 9th to December 13th
- Sprint 2 – December 16th to January 8th ~~December 20th~~
- ~~Sprint 3 – January 2nd to January 8th~~

For each Sprint, a description for a minimum viable product is provided to students. Students should come up with the user stories and perform story mapping. 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 vehicles and parks of the business. This implies adding, removing or updating existing parks as well as vehicles.

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

Scooters can have different types (city and off-road) and bicycles can have different sizes (e.g. 15", 17", 19", etc.). Each vehicle is identified by a unique number or QR Code[2] in the system. Each park has a maximum capacity for the different types of vehicles.

Users should register themselves on the application. Every registration requires a credit card, has an initial cost, and requires the user's height, weight and gender.

Given a user's location, the application should return a list of the nearest parks. Users can check on the application for available vehicles at a given park. Users may use the application to check how far another park is.

The application should allow users to request a vehicle unlock at a given park. At any time, only one vehicle may be unlocked by a user. Besides, when requesting electric scooters and no

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

destination is provided, the system should recommend scooters that have the highest 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 scooters. When a destination park is given, this operation should be used to recommend a scooter with enough range plus 10%, to get to the destination park as opposite to suggesting scooters 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 vehicle.

When parking a vehicle user must receive an e-mail stating that the vehicle 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 parks, according to the bicycle used and considering the altitude/elevation between the initial and final point.

3.1.1 Example User Stories

As an administrator I want to enter the maximum number of bicycles allowed when adding a new park to the system, so that I can ensure an adequate distribution of vehicles across parks.

As a non-registered user, I want to be able to register myself on the product, providing my name, email, gender, height and weight, credit card number, validity date and CCV, so that I can use it to book vehicles and pay for their usage.

3.1.2 Example Epics

As an Administrator I want to create a parking system so that I can make it available for renting vehicles.

As a non-registered user, I want to register on the system, so that I can rent vehicles.

3.2 Sprint 2

When users return a vehicle, the park availability should be updated, and user accounts can be charged. When the vehicle was used for more than 1h, accounts should be charged by the minute. After the gratuitous period (1h), each following hour costs 1,5€.

Users are given 5 points every time they park a vehicle on a park located 25m higher than the one they picked it, and 15 points when the park is 50m higher.

Each user can request an up to date locking/unlocking history of vehicles, with information concerning locking and unlocking date and time, vehicle and parks.

To better handle routing between parks, the administrator should be allowed to load touristic points of interest to the application. As such, users may now be suggested with different routes between two different parks. When requesting a vehicle, the user should be given more choices about the routes that may be chosen from, for example:

- The shortest route between two parks;
- The most energetically efficient route between two parks;
- The shortest route between two parks that goes by at least a certain number of interest points, which can be specified by the user.
- The more energy efficient route between two parks, that goes by at least a certain number of interest points, which can be specified by the user.

When a user requests a route between two locations, and provides a certain number of interest points in-between, more than one route can be suggested to the user (up to a maximum of X – where X is set by the user). The user also specifies if results should be sorted in ascending or descending order for total distance, energy efficiency, etc..

On one of the cities where the system is already deployed, the parks have been updated and the electrical charging circuit is already working. Each park has a maximum charging capacity that can be evenly split by each charging point. At the moment, all parks are provided with a 16A current and 220v tension. Most vehicles can charge up to 1kWh, but it may differ from one vehicle to another, depending on each vehicles' charging system characteristics. Each charging point is capable of charging up to 3kWh.

Another vehicle sharing company is interested in buying this software and use it in Chicago, also known as the "Windy City". For this reason, it is necessary to consider wind as a factor when travelling between two locations. Administrators should be allowed to add wind support information between two different locations. Such information is characterised by the average wind speed between two locations and the wind direction from one point to the other. The wind sensors used for collecting wind speed and direction are quite accurate, providing the wind direction in degrees. In order to understand how the wind affects travelling from one location to another, it is necessary to calculate the bearing between location A and location B to check if the wind is facing forward, backwards or sideways. When no wind information is provided between two locations, you should assume there is no wind resistance between those two locations.

Invoices should be generated on the 5th of each month. Before issuing invoices, the system should check if users have any accumulated bonus points. Every 10 points can be exchanged for one euro. For example, if a user has to pay 1,50€ but has 25 points, the invoice should be issued with 0,50€ and 10 points should be removed from the user accumulated points. No negative or fractional points are allowed.

An administrator should be able to get a report for a specific park, stating the charging status for each vehicle and an estimate projection for how long it would take for each vehicle to reach 100% charge under the existing circumstances.

An administrator should be able to get a report, stating which electric vehicles do not have enough capacity to perform an estimated trip of over X Km (rounded to the unit) over a flat road. For example:

- 7Km, Scooter 3, Scooter 15, Scooter 5, Scooter 9
- 10Km, Scooter 1, Scooter 12, Scooter 17, Scooter 19

An administrator should be able to get a report stating which vehicles are unlocked at a given moment and by whom.

When dealing with batch input information (such as parks, users, etc.) the system should use the given formats and keep a transaction state for each batch of information.

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.

4.3 Database

The database is the main repository of the application and should always reflect an integrity state. Failure to comply with this will penalize the project grading.

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 Relational Model (Normalised);
- **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 using ITP Metrics website.

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 8th, 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[3].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. Relational Model (Normalised);

[3] Where YYYY stands for year, e.g. 2019 for 2019-2020 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[4]. 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:

[4] 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 15th 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 “Passed” state on Sonarqube. For night-time teams, the deadline is 3 a.m. next day. Failure to comply with this rule, induces a group penalty of 0,12 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
        - Note: login with student number (e.g. 1010101)

9 Revision History

V1.0	Initial release.
V1.1	Changes to section 3.2.
V1.2	Fix typo on section 2.
V1.3	Add section 3.3.
V1.4	Add section 6. Deliverables. Add section 7. Assessment. Renumber sections 3.1 Example User Stories and 3.2. Example Epics to 3.1.1 and 3.1.2 Renumber sections 6, Relevant Hyperlinks and 7, Revision History to 8 and 9. Update Table of contents.
V1.5	Update second paragraph of section 3.1. Add section 4.3 Database.
V1.6	Changes to section 3. Add section 3.2 Sprint 2.

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