Lab/Project 3 [LAPR3] – Project Assignment 2019/2020
V2.0, January 3rd

Abstract

A software company needs to develop a product that supports the delivery of pharmaceutical products. In this case, it will mostly be used to deliver self-injecting SARS-CoV-2 vaccines to clients. This service should allow the management of clients, products, orders, deliveries, etc.

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 pharmaceutical company as well as providing a home delivery.

In compliance with good practices learned and applied during the first part of the semester, namely on the course units of Applied Physics (FSIAP), Computer Architecture (ARQCP), 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 pharmacy company needs a solution for managing its products, clients, orders and deliveries.

Clients must register to make online orders.

A fee is charged for home deliveries.

For each purchase, an invoice must be issued and sent by e-mail to the client.

Clients earn credits when using the home delivery system.

Deliveries are performed by couriers using an electric scooter.

On every courier run, more than one order may be delivered. Yet, there's a maximum payload that the courier can carry.

The pharmacy has a parking space where electric scooters can be charged.

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 – January 4th to January 8th
- Sprint 2 – January 11th to January 15th
- Sprint 3 – January 18th to January 22nd

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 adding, removing or updating the list of electric scooters.

Each electric scooter is identified by a unique number or QR Code[1] in the system. The pharmacy park for electric scooters has a limited capacity.

Clients must register themselves on the application. Every registration is free of charge but requires the client's address.

Each client must have and address with a geographical location using Decimal Degrees (DD)[2]. For example, for the "Trindade Metro Station" it corresponds to 41.152699, -8.609267 DD.

A delivery run can be performed for one or more orders from one or different clients.

The application should be able to estimate the amount of energy required to perform a delivery run. In order to reduce costs, the application should suggest the route that spends as little energy as possible, according to predefined physics models that can be applied. Each delivery run should spend as little energy as possible and you should implement best known algorithms for this task.

Electric scooters charge the battery when are correctly docked to a parking space. Furthermore, when an electric scooter is parked, the courier receives an e-mail notification stating that the vehicle is correctly locked and received an estimate time for full charge. Unfortunately, the embedded system on each parking slot has scarce resources therefore you should use low-level implementation for this.

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

Persistence for the system is mandatory, so you should be able to deliver a system that uses a relational database for persistency.

When an order cannot be entirely fulfilled, the system should notify the client, and remove the item that does not have enough stock. An invoice/receipts should be issued for every order.

The system should allow updating stocks when items are received at the pharmacy.

Clients earn credits from each online order.

3.1.1 Example User Stories

As an administrator I want to enter the maximum number of electric scooters when creating a new pharmacy on the system, so that I can set the park limit.

As a non-registered user, I want to be able to register myself on the system, providing my name, email, home address and GPS location, credit card number, validity date and CCV, so that I can use it to make online orders and pay for their delivery.

3.1.2 Example Epics

As an Administrator I want to be able to manage the courier and electric scooters so that I can offer online orders and home deliveries to clients.

As a non-registered user, I want to register on the system, so that I can make online orders.

3.1.3 Non-Functional Requirements

The act of parking the vehicles must be simulated, because we are not yet developing the physical part of the system or interacting with real vehicles. For the time being, each park only has one parking slot.

To simulate the act of parking a vehicle in a slot, a file named "lock_[datetime].data" must be placed under a (configurable) folder that can be read by your application. The name of this file should have the datetime replaced by "yyyy_month_day_hour_minute_second" (e.g., "lock_2020_12_32_12_34_54.data") where year should always have four digits and the remaining elements should always have two digits. For each one of these files, another flag file named "lock_[datetime].data.flag" must be created, and the application should only take into account the "*.data" file when the flag/file exists. Both files should be disposed by the system after correct file handling.

The simulation for a not-well parked vehicle is performed by placing a file where necessary information is missing from that file.

The output should also be written to a file. The resulting file should be named "estimate_[datetime].data" and a flag "estimate_[datetime].data.flag". This file should be handled by your Java application in order to send the notification.

Appropriate data structure should be used for this application. This application is composed by C and Assembly modules compiled using a Makefile. The C component is solely responsible for interacting with the file system, while everything else should be handles by Assembly modules.

For the time being, the park has an unlimited charging capacity and only one charging space.

There can multiple couriers working for the pharmacy, and multiple vehicles available. Delivery runs should take into account the weight of the delivery (which may change at each point), the courier and the scooter characteristics. A directed graph is required because there are one-way streets.

3.2 Sprint 2

The company is expanding its branches and now has multiple pharmacies in their network. The administrator should be capable of managing these pharmacies registries.

Furthermore, each park may have multiple parking spaces. Some have charging capabilities while others do not.

Because some deliveries were taking a long time, the pharmacy decided to acquire some drones for deliveries. The drones can carry multiple orders at one time, if those do not exceed the maximum payload. An administrator should be capable of adding, removing or updating the list of drones. In order to automatically charge the drones, drone parks have been installed on some pharmacies. These parks also support drone charging on some spaces.

Each client is notified when each delivery run starts.

Clients can use their credits to "pay" for the online delivery.

When there is no stock for an item, the pharmacy should automatically verify if any of the nearby pharmacies have that item available and back-order it. Delivery for this order can only start when all the items are in the same pharmacy. An automatic transfer note must be issued by the providing pharmacy and a delivery note should be issued by the receiving pharmacy.

3.2.1 Non-Functional Requirements

For the time being, drones take off and land vertically, to and from an altitude of 150 meters and that there are no physical barriers/restrictions (e.g., buildings, mountains, etc.) between two possible travelling points.

You should now take into consideration elevation and road condition for your land deliveries. For air deliveries, you should take into consideration that drones always park (10 meters) from the ground.

When a delivery drone finishes the last delivery, it should return to the starting park. If that park is out-of-reach because of battery charge, then the drone should choose a nearby park to charge in order to reach the starting park.

In order to reduced wasted memory, your C and Assembly application should handle data using dynamic memory allocation.

3.3 Sprint 3

The company wishes to simulate the running costs of each delivery approach (land vs. air), in order to choose which is the most profitable to use while delivering orders as fast as possible and providing the best service for their clients. As a result, the pharmacy administrator wants to understand which is the most energy efficient type of delivery (land/air), given some specific orders, initial and final points, you should compare the land/air deliveries.

When performing an online order, the order should be forwarded to the pharmacy that is nearer to the client's location.

Restrictions have been applied to each park, and now each park has a limited power capacity that is shared by all parking spaces on that park. Therefore, when a new vehicle is parked, the charging capacity is evenly split between both devices, which alters the estimates of devices that were currently being charged. These changes should activate notifications.

3.3.1 Non-Functional Requirements

If a drone cannot perform a whole run on a charge, it may land on a nearby pharmacy park that has an available charging spot to perform a mid-run recharge. Furthermore, couriers might also use mid-parks for recharging.

It has been noted that winds have a great impact on air as well as land deliveries. You should now take into account wind velocity and direction on your routes.

4 Non-Functional Requirements

This section describes some of the non-functional requirements that must be taken into consideration when implementing your project.

All users must be registered in the system.

Business logic validation should take place when registering or updating data. The database is the main repository for the application and should always reflect an integrity state. All data should be persisted on a remote SGBD. Failure to comply with this will penalize the project grading.

To maximize interoperability with other existing and/or developing systems, the software main core should be written in Java. The software for the docking solutions for powered devices should be developed in Assembly.

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 as long as the system provides the necessary API to interact with the system and provide a clear demonstration of its purpose. In both cases, requirements should be fully tested using Unit Tests and demonstration scenarios should be implemented.

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., Sequence Diagrams (SDs) & Class Diagrams (CDs)). 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, prioritized, estimated and assigned to team members. For each user story, three tasks 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**].

5.3 User Interface

Your application does not require a Text or Graphical User Interface. If you wish, you can provide one to better support user interaction or demonstrations. Nevertheless, all inputs and outputs should be performed using text files.

You should have test case scenarios to automatically demonstrate a logical sequence of the requirements.

5.4 Database

One database for each group has been created on the server previously used on the Databases course. These databases have the following name "LAPR3-GXX" where XX should be replaced by your group number. The password is "qwerty" and should be updated, as soon as possible, to one of your choice.

For development purposes, you are also allowed to use your local Oracle EX installation, to avoid any other kind of problems when using DEI's VPN to access the database server. All the scripts for creating/updating your database schema should be exported to your project's git repository.

6 Deliverables

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

6.1 Weekly Delivery

Every week, until each next Monday, 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 25th, 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., 2029 for 2020-2021 school year and XXX stands for group number e.g. 001.

7 Assessment

On the last four weeks of the project, assessment is performed everyday while students are in class and receive immediate feedback. The project's final assessment takes place on January's last week, according to a schedule that will be provided on Moodle.

7.1 Self/Peer-Assessment

Every week, until each next Monday, 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:

- Code Duplication;
- Technical Debt;
- JUnit Code Coverage;

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

- PIT Mutation Testing Coverage.

On each day, from January 11th to January 26th, 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. Failure to comply with this rule, induces a group penalty of 0,20 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

V0.1	Work in progress.
V1.0	Initial release.
V2.0	Fix sprint dates on chapter 3. Fix typo and add text to section 3.1. Add sub-section 3.1.3. Add sections 3.2, 3.3, 5.3 and 5.4. Fix dates and penalties in section 7.4.

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