SOEN 6461-SS: Software Design Methodologies
Fall 2019 - Project Description

Dr. Constantinos Constantinides, P.Eng.

September 9, 2019

Contents

1 Introduction

This is a group project. Please read the entire document very carefully.

2 Ground Rules

These are the ground rules for the project. Please read this section very carefully:

1. This an assessment exercise. You may not seek any assistance while expecting to receive credit. **You must work strictly within your team and seek no assistance for this project (from the instructor, the markers, fellow classmates and other teams or external help)**. Failure to do so will result in penalties or no credit.

2. **The objective of this project is to adopt the concepts taught in class and deploy them in order to build a medium-size web application**. This means that if you develop an application that may seem to provide the overall functionality but does not follow the methodologies, the techniques and the concepts taught in class, you will receive a penalty or no credit. For example, you may not use Wordpress or any other similar environment to do this project. You may not deploy a framework to support object-relational mapping or concurrency. These requirements must be designed and implemented by your team.

3 Objectives

In this project you will adopt an iterative process to develop a web application. The application is to be developed in an object-oriented environment. You are free to deploy any programming language(s) you wish. The expectations of the project are described in the current document.

You are currently provided with a set of requirements, described in the following section. You must form a team of 5 - 6 members, and assign a team leader. The position of the team leader may be rotated among team members, but not during a development iteration.

4 Requirements

This section contains the initial set of functional and non-functional requirements of the project.

4.1 Overview of the system

Consider a company that rents out vehicles. Your assignment is to develop a web application to handle the operations. A database holds records of vehicles and records of its clients as follows:

1. A record for a car contains a type (sedan, SUV, truck), make, model, year, color, and licence plate number (unique alphanumeric in the format `ABC 123`). Only the licence plate number is unique. An example record is shown below:

   ```
   Type:           SUV
   Make:           Jeep
   Model:          Wrangler
   Year:           2019
   Color:          Black
   Licence plate:  XCB 468
   ```

2. A client record consists of a first and last name, driver licence (unique alphanumeric, in the format `A-1234-123456-12`) together with its expiration date, and a phone number. Only the driver licence is unique.

4.2 Functional requirements

4.2.1 Viewing and searching the catalogs

Clerks should be able to view the contents of a catalog, either in a random order, or by creating a result set through a selection of filtering criteria, e.g. "View all SUV models not more than 3 years old ." Clerks may additionally chose the order by which they view a result set: random order, or sort according to some criteria, e.g. "View sorted by year." From a given result set, a clerk may chose an item to view in detail. In this view, the system should provide an indication on whether the vehicle is available or is rented out. The clerk can subsequently proceed to the next item in detail view, or go back to the (possibly filtered) initial result set view.

4.2.2 Renting, reserving and returning vehicles

A clerk can manage a client record (create new, modify, delete), create a rental of a vehicle for a given client, create or cancel a reservation of a vehicle for a given client, or handle the return of a given vehicle.

The system maintains a record for every transaction (rental, reservation, or return) that includes client and vehicle information, together with a timestamp. In the case of a rent or a reservation, the record will include the due date for each item. Administrators can access, view and search the history of transactions per client, per vehicle, or per due date. For example, 'Show all vehicles currently out', 'When is a given vehicle due', 'Is a given vehicle available on a given date or over certain dates?', or 'What vehicles were due yesterday (and have they all been returned)?.'

4.2.3 Administering the database

An administrator can manage a vehicle record (create new, modify, delete). An administrator can additionally view the contents of the catalog and perform searches, but cannot perform a rental, a reservation, or a return.

4.3 Non-functional requirements

The two main non-functional requirements are concurrency and persistence.

4.3.1 Concurrency

The system would allow at most one administrator being logged in at any time. The system should allow possibly multiple clerks being logged in performing their tasks. We can divide system operations into two categories:

Write operation A write operation accesses in order to modify the contents of a shared resource. A client who wishes to perform a write operation is referred to as a 'writer.'

Read operation A read operation accesses in order to view (but does not modify) the contents of a shared resource. A client who wishes to perform a read operation is referred to as a 'reader.'

The following properties are relevant to any concurrent system:

Safety A shared resource must not be simultaneously accessed by a writer and reader. Furthermore, a shared resource must not be simultaneously accessed by more than one writer.

Liveness A client who wishes to obtain access to a shared resource will eventually be allowed access.

Fairness Requests for access must succeed infinitely often. No client will wait for ever to be serviced. Furthermore, in this system we want writers to have priority over readers.

4.3.2 Persistence

Persistence must be provided by a relational database and you must design and implement object-relational mapping from scratch.

4.3.3 Other

You must design and implement the application in an object-oriented environment with an appropriate architectural style that supports quality characteristics such as separation of concerns, modularity, reusability and adaptability.

5 Development

This section describes in detail what needs to be done: A high-level development plan, the artifacts to produce (including the running application), and finally some more technical considerations such as object-relational mapping.

5.1 Plan and schedule

There will be a total of four iterations in the project, and one in-class live demonstration of your running software. With the exception of the preliminary iteration, all other iterations are timeboxed into three weeks, by which time your iteration is concluded and your artifacts are under assessment. An iteration ends at 17:00 on its last day. The demonstration should be held for no more than 15 minutes. The development plan with respect to the iterations and the demonstration together with the corresponding weights is shown below:

Iteration 1 (9 Sep - 15 Sep). Weight: -5% : Set up your environment (See 6.1) and send invitations to our markers. Team leaders should create a README file in each environment with the team information (names and email addresses). The team formation is finalized at the end of this iteration. **After this deadline, no changes are allowed without approval from the instructor**. Each team leader should send invitations for both environments to our TAs.

Iteration 2 (16 Sep - 6 Oct). Weight: 10% : Clerk functionality.

Iteration 3 (7 Oct - 27 Oct). Weight: 10% : Administrator functionality.

Iteration 4 (28 Oct - 18 Nov). Weight: 20% : The system supports *concurrency* and *persistence*.

Live demonstration (See dates in Course Outline). Weight: 10%

What is the meaning of **-5%**? You will not receive credit for forming a team and setting up your environment, but you will receive a penalty if you fail to do that on time.

5.2 Artifacts and running application

Your software system will consist of the following artifacts that must be developed in line with an iterative process:

1. A *Software Requirements Specification* (template available on course website),

2. A *Software Architecture Document* (template available on course website),

3. Code repository (contains application code, as well as testing, configuration, installation and other supporting implementation), and finally

4. A running version to be demonstrated in class.

Your code must be readable, and must follow the naming conventions of the language(s) involved. You may consider the following references:

- Top 15+ Best Practices for Writing Super Readable Code (`https://code.tutsplus.com/tutorials/top-15-best-practices-for-writing-super-readable-code--net-8118`).
- Code Conventions for the Java Programming Language (`http://www.oracle.com/technetwork/java/javase/documentation/codeconventions-135099.html`).
- PHP Coding Standards (`https://make.wordpress.org/core/handbook/best-practices/coding-standards/php/`).
- PEP 8 – Style Guide for Python Code (`https://www.python.org/dev/peps/pep-0008/`).
- JavaScript Style Guide and Coding Conventions (`https://www.w3schools.com/js/js_conventions.asp`).
- C# Coding Conventions (C# Programming Guide) (`https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/inside-a-program/coding-conventions`).

5.3 Object-relational mapping

You may not use any built-in object-relational mapping technologies that come with some frameworks (see 6.2).

6 Organization and management

This section describes the preparation of your development platforms, the expectations on the team formation, recommended technologies, and your overall responsibilities.

6.1 Development platforms

You must organize your development as follows:

Google Docs For the preparation of the SRS and SAD you must use **Google Docs** that supports **Collaborative Document Development**. Please note that you must maintain only two documents (SRS and SAD) that will evolve over time, iteration by iteration. The folder should be kept with Link Sharing set to off; this prohibits anonymous changes. All team members should be invited to the folder via an email invite (with permission settings set to organize, add, & edit) and should be logged in to their accounts when editing to properly log all changes.

GitHub For implementation, you must deploy **Version Control** and **Issue Tracking** which can be provided by **GitHub**. Your GitHub repository must be private (GitHub provides free private repositories when you request their free student plan).

Please send invitations for both environments to our markers at

- chengcheng.study@gmail.com, and
- petia2397@gmail.com

Team formations will be posted on the course website and each team will be assigned a number. Please use this number in all communication with the instructor. You may not change a team and you may not break your team or restructure teams, unless you discuss the matter and obtain permission from the instructor (also, see 6.4).

Resources

- Google Docs (`https://www.google.ca/docs/about/`
- GitHub (`https://github.com/`
- Student Developer Pack (`https://education.github.com/pack`)

6.2 Recommended technologies

At the back-end, you must maintain a relational database management system such as one of the following:

- SQLite (`https://www.sqlite.org/`
- MySQL (`https://www.mysql.com/`
- PostgreSQL (`https://www.postgresql.org/`

To develop your server-side application, we can recommend the following languages, listed below together with applicable frameworks. Note that this list is not exhaustive.

- Java with Spring (`https://spring.io/`).
- C# with ASP.NET (`https://www.asp.net/mvc`).
- Python with Flask (`http://flask.pocoo.org/`).
- PHP with Laravel (`https://laravel.com/`).
- node.js (Javascript) with Express (`https://expressjs.com/`).

6.3 Responsibilities

As this is an academic exercise (as opposed to an industrial assignment) where the objective is to learn, **you are all expected to participate in all activities (requirements analysis, architecture and design, coding, testing) on a relatively equal weight**. Failure to do so will result in penalties to the individual(s) involved as well as to the team leader.

For each member of your team, we will monitor the following: a) number of commits, b) number of assigned tasks, c) the total number of tasks, and d) the degree of importance (of committed tasks), where the latter is defined as follows:

0: Unimportant commits, e.g. comments.

1: Minor commits, e.g. changing the names of functions or variables.

2: Important commits, such as adding new functionality into classes.

3: Very important commits, such as adding a class.

In other words, the notion of "relatively equal weight" of workload does not exclusively depend on the number of your commits, but on all four factors (a - d) described above.

6.4 Slacking off and voting out

Slackers will be penalized. Accommodating a slacker will result in the team leader receiving a penalty.

As a team, you have the power to vote a team member out by majority vote. In the case of a tie vote, the team leader has the privilege of a casting vote. In the case where the alleged slacker is the current team leader, then the team leader loses their privilege of a casting vote and you (the entire team) must arrange a meeting with the instructor. Note that your vote is not a final decision but a recommendation that you (the entire team) must present to the instructor in the presence of the member you wish out. A team member can be outed only upon approval of the instructor.

7 Assessment

This section provides all necessary information about your assessment.

7.1 Marking

Note that even though this is a group project, marking is not assigned collectively. Based on the criteria described in this document, individual penalties will result in different people from the same team possibly receiving a different project mark.

7.2 Monitoring your progress

We will monitor your environments and the artifacts that you produce and at the end of each iteration and we will assess your progress. Please do not submit either a hard copy or an electronic copy of your artifacts.