Project Overview: CO-OP-ERATOR

1 Scope of the Project

The scope of the software engineering group project of the Winter 2019 edition of the ECSE321 course is to develop a software system for managing co-op terms for co-op programs offered at McGill University. In teams of four or five students, you will gather requirements, design a multi-tier software solution to satisfy those requirements, implement the system, validate that the system is satisfying the requirements, and develop a release pipeline to automate the software delivery process. Viewpoints developed by multiple teams will be integrated with each other via service calls.

Your application (viewpoint) must support the scenarios described in the user story *for one stakeholder*, but it should also *integrate with the viewpoints provided for other stakeholders by other teams* (via service calls). 3-4 teams are expected to collaborate as a development unit to deliver student, employer, co-op administrator and an academic manager (i.e., co-op program manager + co-op term instructor) viewpoint.

All functionality of the system needs to be accessible via the web frontend for respective stakeholders. In addition, a mobile (Android) frontend shall allow to execute the most important functionality for the given stakeholder, i.e. it shall have both read and write access to the backend via RESTful service calls. Secure login with authorization can be omitted (but it counts as an extra). External systems or service mentioned above (e.g. *myFuture*, *myCourses*) are not required to be integrated.

2 Technological Constraints

Your project should adhere to the following technological constraints:

1. For each sprint, your team must
   1.1. Provide project backlog using GitHub Projects or ZenHub technologies.
   1.2. Use issues in GitHub to track *development*, *release engineering* and *documentation* tasks.
   1.3. Define milestones of the project for each deliverable and assign all issues created during a sprint to its corresponding milestone.
   1.4. Provide documentation (e.g. meeting minutes with key decisions, effort table, models, supplementary images) using the wiki pages of the GitHub repository.
2. Starting from Sprint 1 (Database), your team must
   2.1. Use UML Lab to create a domain model.
   2.2. Implement a persistence layer using a Postgres database.
   2.3. Use the ORM technology Hibernate to map objects to database concepts.
   2.4. Create a Spring/Spring Boot project.
   2.5. Configure a build system using Gradle.
   2.6. Use a Continuous Integration process using Travis CI to build and test the database layer.
3. Starting from Sprint 2 (Backend), your team must
   3.1. Implement RESTful web service using Java Spring Boot.
   3.2. Provide a suite of unit tests for the backend using JUnit by separating it from the database.
   3.3. Deploy the project as a Heroku application in addition to the constraints above.
   3.4. Integrate services (functionality) developed by other teams.
4. Starting from Sprint 3 (Web), your team must
   4.1. Implement the web frontend using Vue.js.
   4.2. Integrate web frontend with backend services (including services developed by fellow teams)
   4.3. Provide continuous integration of the web frontend using npm build, Travis CI
   4.4. Deploy the web frontend as a separate Heroku application.
5. For Sprint 4 (Android), your team must
   5.1. Implement the mobile frontend using the Android SDK but without the need for continuous integration and deployment for the Android frontend.

A team may choose a technology different from the recommended ones in case of items 2.4, 2.5, and 4.1, but no technical support will be provided. All other technological constraints need to be respected.

Deliverable 3 – Web Frontend, Architecture Modeling

This deliverable consists of the following parts:

1 Architecture Modeling

You need to provide an **architecture model** (in the form of a block diagram or UML Composite Structure diagram) that highlights the main functional components of your system, their hierarchy and their interconnections with other components. You need to include the diagram / figure of the architecture in the wiki of your team together with a **brief description of the key functional components** (max 1 paragraph each).

2 Implementation of Web Frontend

You need to provide a **graphical user interface** for the web frontend using HTML, CSS and JavaScript technology (preferably using Vue.js). In addition to functional correctness, usability and design of the web frontend will also be evaluated. Having a uniform design along all viewpoints is also a plus, thus you are allowed to collaborate with your fellow teams on the CSS. In addition, end-to-end test cases can be implemented (e.g. using Nightwatch) for bonus score.

3 Integration of Web Frontend with Backend Services

You need to **integrate your web frontend** with the **existing backend services**. You need to issue asynchronous calls to backend services via the RESTful API developed in Sprint 2, and correctly process the results. Your web frontend may also need to make calls to backend services developed by fellow teams. You are required to update the backend services as needed – your backend will be evaluated also as part of Sprint 3.

4 Build System, Continuous Integration and Delivery

A build system (using npm build) is needed to automate the process of compiling and packaging all code developed for the web frontend. Travis CI should compile and build the project and upon successful completion, it should deploy a new version of your web frontend using Heroku. Note that this step requires to create a *new Heroku application without a database* (i.e. a second Heroku application in addition to the existing one used for the backend and database). Since no test cases are required to be developed for the web frontend, test cases of the frontend are not required to be integrated to Travis CI either.

Every push to the GitHub repository should trigger a build job on Travis CI to initiate continuous integration and delivery for both the **backend** and the **web frontend**. The build system and the CI specification must conform to the Technological Constraints above.

5 Project Management and Project Report

A key aspect of agile software development is the use of a project backlog to coordinate development and project documentation activity. Each group is expected to **continuously use** of the **issue tracking** features on GitHub and an **agile project board** in order to create and manage the project backlog. Each issue needs to have an assignee to trace core responsibilities within the team. Each commit made to the Github repository should refer to a corresponding backlog issue.

The team should provide a welcome page that introduces the team and describes the main scope of the project in the README.md file in the root of each team's repository. In addition, the README.md file should contain an overview table with names, team roles and individual efforts (in hours) with separated entries for each deliverable.

Project Deliverable 3 shall be also accompanied with a succinct *project report* as *part of the project wiki* which records the meeting minutes and the key design decisions taken by the team. This project report

should be navigable from the README.md file of the project. Altogether, the team should comply with all Technological Constraints above.

Submission

The project will be carried out with the same teams (of FOUR or FIVE students) as you worked with for Deliverable 1 and 2. For Deliverable 3, your team is required to submit a commit link (i.e. a URL representing your last GitHub commit that counts as a deliverable) by ***Sunday, March 24th, 2019 23:30***.

Each team member must make contributions to the deliverable. A team member who does not contribute to the deliverable receives a mark of 0 for the deliverable. A team member may optionally email a confidential statement of work to the instructor before the due date of the deliverable. A statement of work first lists in point form the parts of the deliverable to which the team member contributed. In addition, the statement of work also describes whether the work load was distributed fairly evenly among the team members. A statement of work may be used to adjust the mark of a team member who is not contributing sufficiently to the deliverable. It is not necessary to send a statement of work, if a team distributed the work for the deliverable fairly evenly and each team member contributed sufficiently.

Marking Scheme (8% of total score)

Part of Assignment	***Marks***
Updates of backend Implementation (business methods, RESTful services, units tests)	5
Architecture model	10
Implementation of web frontend GUI (incl. design + usability)	25
Integration of web frontend with backend services (incl. asynchronous backend calls of RESTful API)	25
Build system + Continuous integration (npm build, Travis CI, Heroku for the web frontend; Gradle + Travis CI + Heroku for the backend)	20
Project management (e.g. backlog, issues, sprint planning, teamwork report, quality of documentation)	15
Total Marks:	100
The total mark may be adjusted based on the actual contributions of a team member to the deliverable.	