McGill

Department of Electrical and Computer Engineering
ECSE 321 Introduction to Software Engineering
Fall 2021

Project Deliverable 1:
Requirements, Domain Model, and Database Design

This deliverable consists of the following parts:

1 Requirements Model

You should identify a list of *functional and non-functional system requirements* focusing on not more than the 15 most important requirements. Each requirement should

(1) have a *unique ID*,
(2) have a *textual description* which follows best practices of formulating requirements (e.g., user story or other requirement templates),
(3) be *registered as an issue in the project backlog*.

These key requirements need to be refined by use cases and actors by preparing a *use case diagram* to define who has responsibility in initiating a specific use case or is participating in it. Each *use case* must be properly named and described in a natural language. The *actor(s)* involved in a use case must also be stated together with their connection to the appropriate use case(s). The use case diagram must be documented as part of the project wiki.

Each team member needs to provide a *detailed specification of 1 use case* (selected from the 5 most important use cases) with the *main flow of the use case*, and as many *alternative/exceptional flows* as needed. This should also be documented in the project wiki (e.g., as a hierarchically numbered list).

2 Domain Model

The deliverable should also contain a *domain model* defined in the form of a *class diagram*. The model itself should be stored in the GitHub repository of the team, and the *UML class diagram should be included in the documentation wiki*. The team should provide a *brief (textual) rationale* for the key decisions made while creating the domain model.

3 Persistence Layer

To persist the data when the server shuts down (gracefully or otherwise), data must be stored within a database hosted on a Postgres DBMS instance associated with a Heroku app. A persistence layer needs to be developed to access this data from a Java application using Object-Relational Mapping (ORM) technologies like Hibernate. The persistence layer should be compliant with the Domain Model, and it should be able to correctly persist data.

4 Testing of Persistence Layer

A test suite of the persistence layer needs to be developed to try one read and one write operation for each class of your domain model (i.e., two test cases for each class). The test suite should demonstrate that your application can read and write (1) objects, (2) attribute values of objects, and (3) references between objects. The test suite should allow to remove all contents of the database (clean). Test cases are required to demonstrate the persistence of objects of **all** domain model types, and at least one attribute and at least one reference of each object.

5 Build System and Continuous Integration

A build system is needed to automate the process of compiling and packaging the code. The project should successfully compile, build, and run the test suite of the persistence layer while conforming to the Technological Constraints. Every push to the GitHub repository should ***trigger a build job on the main branch***, which uses the build system to compile the package and push the new version of the application. The build system and the CI specification must conform to the Technological Constraints.

6 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 make use of the issue tracking features on GitHub to create and manage the project backlog. Each issue needs to have an assignee to trace core responsibilities within the team. 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 1 shall be accompanied with a succinct *project report* as *part of the project wiki* which records the meeting minutes, the key design decisions taken by the team, and the success spectrum exercise. This project report should be navigable from the README.md file. Altogether, the team should comply with all the Technological Constraints.

Submission

Your team will be assigned a private repository within GitHub, which should be used for this project. The course staff also has access to your GitHub repository. For Deliverable 1, your team is required to submit a commit link (i.e., a URL representing your last GitHub commit that counts as a deliverable) by ***Monday, October 18 at 11:59pm***. 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.

Marking Scheme (10% of total score)

Component	***Points***
Requirements model: functional and non-functional system requirements, use-case diagram(s), detailed use case specifications	20
Domain model (in UML)	10
Implementation of persistence layer (database persistence, adherence to domain model)	15
Test suite of persistence layer	15
Build system + Continuous integration	15
Project management (e.g., backlog, issues, sprint planning, teamwork report, documentation quality)	20
Code style	5
Total Marks:	100

Note: The total mark may be adjusted based on the actual contributions of a team member to the deliverable.

Detailed Marking Scheme

Requirements model	**8**
Functional requirements are properly formulated	2
Non-functional requirements are properly formulated	1
Requirements are identifiable	1
Requirement specifications systematically follow a template (e.g., ID, user stories, etc.)	2
Requirements registered as issues in project backlog	1
Requirements are verifiable	1
Use Case Diagram	**7**
UC diagram is syntactically well-formed	1
UC diagram is easy to understand (e.g., not a single messy diagram with all UCs)	1
Use cases are properly named (Verb + Subject)	1
Actors are properly named (Roles played in the system)	1
External subsystems included as actors	1
UC-Actor assignment is semantically meaningful	2
Detailed use case specifications	**5**
UC specifications are numbered hierarchically	1
Main flow for UCs is defined	1
Alternate flows for UCs are defined	2
Steps in scenarios are systematically formulated (Who does what?)	1
Domain Model	**10**
Class diagram of domain is easy to understand	0.5
Rationale of key decisions are documented	0.5
Classes are consistently used	2
Attributes are consistently used	1
Enumerations are consistently used	1
Generalization hierarchy is properly used (incl. abstract classes)	1
Containment hierarchy is elaborated (is there composition? not flat hierarchy?)	1
Multiplicities are appropriate (e.g., not too generous, not too restrictive)	1
Associations are appropriate (e.g., not only bidirectional)	2
Persistence Layer	**15**
JPA classes are compliant with domain model in UML	3
JPA annotations are consistently used	2
DAO implementation exists for CRUD methods of classes (auto generated or manually written) (Needs to be evaluated separately for each class)	10
Testing of Persistence Layer	**15**
Read test cases exist for each class	5
Write test cases exist for each class	5
Test suite demonstrates that application can read and write - objects	1
Test suite demonstrates that application can read and write - attributes	1
Test suite demonstrates that application can read and write - references	1
Database contents cleared / reverted after test method	2
Build system and CI specification	**15**
Testing documentation (testing summary, required configs, Heroku app address)	3
A .GitHub Actions.yml build file for the main branch is present (attempted CI)	2
The main branch build succeeds on GitHub Actions - project is properly built	2
The main branch build succeeds on GitHub Actions - tests are executed correctly	3
The project builds locally with Gradle (gradle build -xtest or ./gradlew build -xtest)	2
Tests run and pass locally	3
Project Management and Project Report	**20**
Project deliverable and success spectrum provided on wiki	2
Backlog is maintained in GitHub Projects	2
Issues are created at the beginning of sprint	2
Issues are assigned to milestones	2
Issues are assigned to responsible person	2
Lifecycle of issues is continuously managed in GitHub	2
Project Report has welcome page (introduction to group, scope of project)	2
Documentation of team operation (minutes of meeting recorded, design decisions recorded)	2
Individual roles are detailed	2
Individual efforts are summarized in the table	2
Code style	**5**
There are comments for persistence layer methods	1
There are comments for nontrivial code blocks	1
Code is free from unused variables and unused imports	1
Methods are not too long (approx. 20 lines)	1
Naming of variables, methods follow conventions	1
TOTAL	**100**