FIT5171 Project Assignment 2
Integration Testing & Regression Testing

Yuan-Fang Li and Yan Liu

Due 23:59, Sunday 28th April, 2019

1 Project description

In this assignment, you will build on the source code you developed in Assignment 1 as well as a new code base provided to you, and add functionality to it. You will need to develop a number of components and perform unit testing as well as integration testing.

The project will continue to use the build management tool Apache Maven and online repository GitHub. Moreover, we will be adding a number of new tools and systems to work with.

You will use the Jenkins[1] continuous integration server to automatically execute the test suite whenever some code change is committed into your repository. In other words, we will perform automated *regression testing*. In addition, we will use the Sonar code quality monitoring system[2] for monitoring code quality and test coverage.

2 Problem description

In this assignment you will need to develop (and test!) essential functionality for the web site `http://www.allaboutrockets.com`. In general, you will develop components (Java interfaces and classes) to accommodate *requirements changes*, implement some of interesting *new functionality*, and *thoroughly* test them.

To assist your development, a new and updated code base will be provided. The code base includes changes to the `model` classes, as well as code for new functionality of *data access* and *mining*. The new code base is available in a read-only SVN repository, `https://github.com/yanliu19/FIT5171_assignment_2.git`.

You need to `check out` the code base from this repository.

You need to integrate the given code base with your own code base from assignment 1.

[1] `https://jenkins.io/`
[2] `http://www.sonarsource.org/`.

2.1 New Functionality

You will need to implement and *thoroughly test* the following new functionality.

2.1.1 Validation.

At some stage, certain attributes of some classes need to bevalidated in different ways. The following are some examples of what should be validated.

User When some attribute of a user is being set, some conditions will need to be satisfied.

1. A user needs to have a valid email address.
2. A user needs to have a non-empty password.
3. A user needs to have a non-empty last name.

Rocket When a rocket is being created, it needs to satisfy the following conditions.

1. It must have a valid `manufacturer` value, which points to a `LaunchServiceProvider` object
2. It must have a valid, non-empty `name` attribute value.
3. It must have a valid, non-empty `country` attribute value.

2.1.2 Data Access.

We will store the data in a *graph database* engine called Neo4j[3]. A graph database allows a flexible data model without needing to define a schema upfront, or at all. As a result, data can be quite freely modified in the graph.

In our application, the graph comprises of nodes representing the various types of entities: rockets, launches, launch service providers, etc.; and edges represent their relationships. For example, a rocket is connected to the launches for which it is the launch vehicle, and to the launch service provider that manufactures it. A Neo4j visualisation of such a small graph is shown in Figure 1

It is usually a good idea to abstract away the details of accessing the underlying database, and this is usually done through the **data access object** (DAO) pattern. In our project, the `dataaccess` package contains a `DAO` interface for accessing model objects. With an appropriate implementation, a `DAO` object allows us to create, read, update and delete (CRUD) domain objects.

A Neo4j-based implementation, `Neo4jDAO`, implements the `DAO` interface and makes use of an *object-graph mapping* (OGM) abstraction layer to further hide the graph database details away. This is done through annotations on the model classes and the use of a `Session` object. Details can be found in the code base and on the Neo4e web site[4].

[3] `https://neo4j.com/`

[4] `https://neo4j.com/docs/ogm-manual/3.0/`

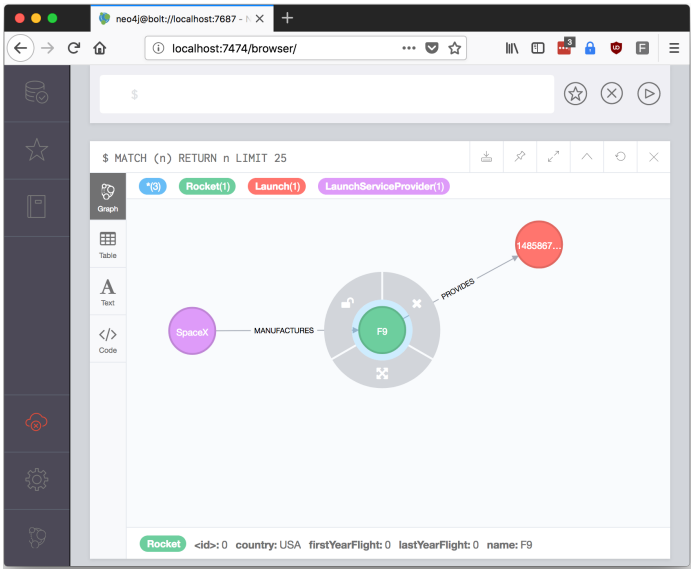

Figure 1: A small toy example graph.

2.1.3 Mining.

The `mining` package contains a simple data mining class that discovers interesting properties about `Rockets`, `Launchess` and `LaunchServiceProviders`. These properties may include the most active rockets, the most reliable launch service providers, the most recent launches (example given), etc. For each of these tasks, you may need to come up with your own definition. For example, you need to precisely define what it means for some launch service provider to be "busiest".

Obviously the above functionality requires access to some/all of the domain entities, which will be stored in a Neo4j graph database. To simplify development and testing, we will be *mocking* the actual database layer, using a Java mocking library called mockito[5].

Essentially, a mock object emulates the behaviours of a real object so that we can use it as a *driver* or a *stub*. Documentation and examples are available at the mockito web site. Examples of its usage can be found in `RocketMinerUnitTest.java` to help get you started.

Specifically, you will need to develop *and test* Java methods to support the functionality, including:

Workhorse The top-k active rockets with the most number of completed launches.

BestPerformed The top-k launch service providers with the most reliable launch record.

[5] `http://site.mockito.org/`

Dominant The dominant country in an orbit who has launched the most rockets.

Exorbitance The top-k most expensive launches.

Hotshots The top-k launch service providers that has the highest sales revenue in a year.

2.2 Testing

Apply a combination of appropriate unit and integration testing techniques to *thoroughly* test the newly updated code that you combine and develop. In your report, you need to (1) articulate the particular integration testing techniques you use, (2) provide rationale for using them in the report, and (3) discuss pros and cons of these techniques in their application in your code base.

2.3 Regression Testing

Depending on the availability of the Jenkins server (at the time of writing it is not yet available), you need to set up the continuous integration environment to enable automated regression testing. In achieving this you need to complete the following setup tasks.

- **Maven.** Modify your project's Maven `artifactId` in the file `pom.xml`. Your new `artifactId` will need to be in the format of "`fit5171_2019_XX`", where `XX` is your project group number.
- **Jenkins.** Complete the configuration according to the instructions given on Moodle. Log onto our continuous integration server (will be provided soon). Modify the settings of your group's project according to the above guide on Moodle. Make sure Jenkins automatically executes all tests in your code base whenever you commit some changes to your Subversion repository.
- **SonarQube.** A tool for continuously code quality monitoring. Instructions for setting up the SonarQube server will be uploaded soon. Curious students are referred to SonarLint[6], a plugin supported on most IDEs such as IntelliJ.

3 Assessment

This assignment carries **15 marks**. The assessment will be based on the submitted files as well as a demo conducted in week-8 tutorial classes.

[6] https://www.sonarlint.org/features/

Table 1: Sample summary of self assessment.

Member	**Percentage of work**	**Main contributions**
Marge Simpson	70%	Local setup, functionality extensions, JUnit test cases
Homer Simpson	30%	Test strategy

3.1 Submission

This assessment will be based on the submitted artifacts, including (1) your new code base, and (2) a short, *maximum 4-page* written report to be submitted by **23:59, 28th April, 2019** via Moodle. Only one member from each group needs to submit the file. Please submit a single **.zip** file containing both the code base as well as the report. Indicate clearly the members in the group, as well as each member's contributions in this assignment.

3.1.1 Self & peer assessment.

Even though the project assignments are group-based, assessment is individual-based. Hence, in your report, please include a brief statement of each member's contribution. It can be as simple as Table 1. Of course you are welcome to include more detailed information.

We will also use CATME for peer assessment. More details will be announced later.

3.2 Demo

A demo will be conducted in week-8 tutorials. Each group will demonstrate to the tutor their setup on a laptop. Each group will also need to give a code walkthrough to the tutor, showing your understanding of the code base, your extensions, and your tests. Also demonstrate to the tutor that you can run all tests automatically through Maven, both locally and remotely (provided the Jenkins server has been made available).

3.3 Assessment Breakdown

This assignment accounts for 15% of the total assessment. You will be assessed in the following areas.

12 marks Development & integration testing.

- You have successfully integrated the code bases (provided and your own from assignment 1) with no compilation errors.
- You have successfully developed the new validation and mining functionality specified above using the TDD approach.

- You have performed adequate unit testing and integration testing appropriately.
- In your testing process you have applied mocking appropriately.

3 marks Automated regression testing

- You have correctly set up the remote regression testing environment, i.e., the integration of the git repository and the corresponding Jenkins *job*.
- You have correctly set up the project on Jenkins so that *automated* regression testing on your project is enabled.

Note: the completion of this part depends on the availability of the Jenkins server, which is not yet ready. If the server does not become available on time, this part will be skipped, and the marks will be scaled.

The short report should be jointly authored by the group members. The report should be single-column, single-spaced with at least a 11pt font size for the main text. The report should (1) document functionality you implemented, (2) describe your integration testing approach, and (3) show evidence of your regression testing framework (screen captures, etc.). The report should not exceed **4 pages** in length.

Given the limited space, please be *succinct* and focus on the most interesting/important aspects of your development and testing.

4 Extra Credit: Maximum 3 Marks (Optional)

As in assignment 1, this assignment can be extended in many different ways. Extra credit may be given for extended functionality and adequate testing.

For example, you can further extend the `model` classes to add more features and validate them. You can also validate extensions you have implemented in assignment 1.

You can also extend the MINING layer by adding additional interesting functionality.