QA Coursework project 1

Due 24th June 2019

## Philip Mainwaring

# Introduction

The following report describes the creation of a simple REST based web application with CRUD functionality. This application presents the culmination of the opening weeks at QA consulting and the introduction to OOP. In this report I will detail how the project was planned, designed, implemented and tested. A review of the final product will also be given as a conclusion to this report.

Contents

[Introduction 1](#_Toc12143570)

[Planning and Problems Encountered 1](#_Toc12143571)

[Diagrams 2](#_Toc12143572)

[CI Software Used 2](#_Toc12143573)

[Programming and Software Development 3](#_Toc12143574)

[Systems Integration and Build 3](#_Toc12143575)

[Conclusions and Future Practice 3](#_Toc12143576)

[Appendix 1 4](#_Toc12143577)

# Planning and Problems Encountered

An agile method was followed using Sprints (please see Trello board) and test driven development produced from behaviour driven development. This was achieved by the initial construction of user stories (Appendix 1). The user stories allowed for user requirements to be produced and prioritised which were then established into J Unit, Mockito and Selenium test plans (please see Trello board and Test Class Diagram).

I wanted to use BDD and TDD methods as I have no experience of either and despite a promising start with the J Unit tests, I was forced to rebuild the project twice over because of problems adding and modifying pom.xml, beans.xml and persistence.xml files too late in development. I was also unsure how much time to assign to tasks such as Mockito testing as these were yet to be delivered during the course.

The prioritisation of tasks ensured a viable product was delivered at the end of the process but a number of tasks were not incorporated due to problems encountered during the second and third sprints. The most costly of these was persisting date types from the web site to the database which took around 9 hours to correctly implement. The problems arose from converting dates to and from a JSON format across the system and throughout testing.

Further problems occurred following the decision to implement inheritance along with a many to one relationship between the User and Activity entities. The CRUD functionality proved difficult to implement without explicitly defining a sub class of Activity. The design of the application was intentionally geared towards extending the model to incorporate other lifelogging activities. Hiking and Kayaking methods and scripts have been separated so that either one can be used as the basis for a third or fourth category and implemented alongside the existing classes.

# Diagrams

Attached to this document are diagrams showing the class structure for the main classes as well as a separate class diagram showing the replicated format for the testing classes and methods. An entity relationship diagram has been included to show the design view of the persisted tables, however it must be noted that a link table is also generated to combine the users and the Activities. This is managed by using an ArrayList in the java objects. Testing documentation has also been included here. A site map diagram has not been made for the front end as only one HTML page has been used in this iteration of the application.

A Trello board has also been included to present the categorisation of work tasks into sprints. This also shows the work left to complete. By preference, I would not use this software again as the functionality is very limited. I was only able to use simple checklists and the colour coding did not allow for a key to be created to label yellow tasks as complete and red tasks as yet to do or in progress. In short this product failed to work as an effective way of managing sprints, a series of colour coded sticky notes proved more useful as a Kanban board.

# CI Software Used

The following software choices have been mostly prechosen for this assignment.

Eclipse IDE – This proved an effective IDE with useful plugin features and a relatively gentle learning curve. The object diagrams were made from plugins and a number of testing packages were included after being downloaded from the Eclipse marketplace. This resulted in a better developed application with a higher standard of code.

Maven build tool – the maven build tool was useful when adding packages but led to a few setbacks during development when problems occurred that had no obvious solutions. In both of these occasions, new projects were made and all the Java code was copied to new classes and the faults disappeared.

Git – I have some experience of this version control tool but I am far more comfortable in how to use it and why to use it following this project. Setting up branches and tracking back changes has been invaluable when problems have occurred or when merges need to be compared after forgetting to pull at the start of a session.

Jenkins – I struggled to initially set up the CI server, however after working through problems I have a better idea of how this software works and I can see that it is a very useful application.

Google Cloud – This is a much better virtualisation tool than IAAS products I have used before and is incredibly intuitive. I feel very confident using this in the future.

Trello - (comments above)

Brackets – This is my preferred HTML editor due to the amount of plugins it offers. I decided to use this as I am more comfortable with the software from previous web design projects.

# Programming and Software Development

I have attempted to follow the best practices throughout the development of the project. Due to the TDD approach used, the SOLID principles were adhered to throughout development. Examples can be seen in the use of interfaces throughout the three tier architecture to transfer JSON data across the system as well as appropriate use of abstraction and dependency inversion through the Activity and Hiking/ Kayaking classes.

The finished product has achieved 81.5% testing coverage. I would have appreciated more time to implement selenium tests as I was very comfortable building these tests, however they required the front end to be complete before they could be built. The majority of functionality intended for the first iteration of this software has been implemented, however features such as reporting and password security have been delayed till the next stage of development.

Key features were changed during development, however as some of them only became apparent when the website interface was being linked, care was taken not to alter functionality. An example of this can be seen in adding the activity type attribute to the activity abstract class. This was needed to simplify updates to the Hiking and Kayaking classes. As the classes share the update method, there was a need to differentiate the objects without writing separate methods and duplicating the code. An additional attribute allowed the functionality to be used for both classes.

# Systems Integration and Build

The use of a version control system was very beneficial in this project. I managed to use a series of branches and merge them back into the master branch as features were implemented. Due to problems during development and a few occasions when I forgot to pull the latest version down to my local repository, I needed to force push new updates to the global repository. I made use of the merge conflict tool during this process and reverted back to a previous version when I had problems. I have made use of a git ignore file but also added a folder of documentation and testing reports for my own personal use during the project. As can be seen in the class diagrams, the code for this project is stored appropriately in packages with correct naming conventions.

Testing (as can been seen in the Surefire and Jacoco reports) surpassed 80% coverage for the java section of the program. This should reflect the TDD approach used as no additional testing was needed to raise the coverage level above the necessary standard. In reflection of this process, the few Mockito tests that did not run and were not completed, ultimately revealed problems that occurred in the front end. One example is updating the user email even though the email is a key field. The email field has been changed to read only in this stage of development as no fix was found for this problem.

I am happy with the end result of this project as CRUD functionality has been extensively tested in the back end and operates in all of the front end (with the exception of user emails). CRUD functionality is easily achieved and intuitive for Users, Hiking and Kayaking activities. I would have appreciated more time to implement final Mockito tests and a range of Selenium tests.

# Conclusions and Future Practice

I am happy with the result of this project as I have managed to create a fully functional CRUD application with OOP based back end. I have struggled with OOP processes in the past due to a background in SQL and functional programming. I made the decision to avoid SQL based solutions to problems and only have one line of SQL specific code in my program (discounting test data inserts). I am very proud that I have gained confidence in Java code and I know I will be able to plan and design OOP based programs in the future. I am eager to gain more experience of CI systems and cloud services as these look to have many advantages.

I have managed to achieve most of the user goals I set out at the start of the project and I am confident that if I had another week to spend on this project, I would be able to implement almost all of the features. Time and experience have been the main constraints for this project, however I expect these to be less influential after more time completing the training course.

# Appendix 1

User Stories

V1

1. I want to be able to store lifelogging activity record after completing an activity.
   1. Time
   2. Distance
   3. Name
   4. Category
   5. Image Directory or LL link
   6. Video Directory or LL link
   7. (User ID)
2. I want to be able to view all of my completed activities by category or by date descending.
3. If I add an entry by mistake I want to be able to edit or delete the activity.
4. I want to be able to make new categories of lifelogging activity.
5. I want to store links to lifelogging data/ images/ videos converted by a third party app.

V2

1. After completing more than one of the same activity I want to be able to see a report of progress made. This should include
   1. Time spent completing activities (per week, month).
   2. Frequency of activity over time period (week, month).
   3. Total activity and activity variety over time period (week, month).
2. I would like to store sample lifelogging images and videos along with each activity.

V3

1. I do not want to use a third party application to convert lifelogging images to video. I want to
   1. Set the time period for each image to show.
   2. Automatically remove blurred images.
   3. Store the completed video in a separate directory to be backed up externally.