

Report 2

Work Integrated Learning – SEB701

Francesco Ferraioli – n8323143 – francesco.ferraioli@connect.qut.edu.au

Software Engineering (EN40)

MonicIT (60 Days Placement)

Executive Summary

Executive Summary

Table of Contents

1.0 Work Place Background 4

2.0 Work Activities 4

2.1 Major Activity and Reflection 1 4

2.2 Major Activity and Reflection 2 5

2.3 Major Activity and Reflection 3 5

2.4 Major Activity and Reflection 4 5

2.5 Major Activity and Reflection 5 5

3.0 Conclusion 5

References 6

Appendicies 6

Appendix A: Work Log 6

Appendix B: Certificate Of Time Worked 6

Appendix C: Reflective Notes 7

# 1.0 Work Place Background

Work Place Background

# 2.0 Work Activities

## 2.1 Major Activity and Reflection 1

Major Activity and Reflection 1 – Airport model

##### Situation

One of LMS2’s main customer facing features is the booking form. The booking form allows a customer to book one or more trips with a limousine company. The booking form allows for automatic trip quotes, but it requires a number of things to be able to calculate it. This includes: trip distance and time, the pick up time, the payment method and the vehicle they wish to book for the trip. Using Google API, calculating the total distance and total time of the trip is a simple task, as it just requires the longitude and latitude of the pick up and drop off locations as inputs. However, many of the customers who wish to book a trip with a limousine, wish to either be picked up from the airport, or dropped off from the airport.

##### Task

Instead of the customers having to look up the address for their departure or arrival airports, supplying the customer with a list of airports with prepopulated addresses was certainly the better option. My task was to implement a way for our clients to manage the airports they wish to display on their booking form. To allow this feature, each client must be able to manage their own list of airports. This means that a new entity must be introduced into the system, the entity of airports.

##### Action

After speaking to both the boss and the lead engineer for the project, we settled on the schema for the airport table. The model will simply be a class that can be used to access the database table and records. The views will simply need a form to create and update airports and an index to list all the airport. Rails has a built-in scaffolding command to automatically generate a generic template for all this functionality.

After running the scaffolding command I did a bit of customization of the generic templates to match the current system look and feel. Furthermore, I finalized the CRUD operations for the airport model. The last implementation task was to add the airport functionality to the booking form, displaying a combo box, which lists all the available airports for the client. I then moved on to writing unit tests for the model functions as well as end to end tests to ensure the UI side of things acted as expected.

##### Results

The client could now easy create, read, update and delete their airports, through the views available for managing this entity. Furthermore, the customer facing booking form would list out the airports on the combo box. After a bit of tweaking, the unit tests and the end to end tests all passed.

##### Learnt

I learnt a lot of things during this task as this was the first rails entity I was implemented. I learnt about some important rails commands, including rails generate scaffold command which assisted me a lot in this task. I learnt a lot about the rails framework and how it functions with the database. Furthermore, I learnt how to write unit tests for rails models.

## 2.2 Major Activity and Reflection 2

Major Activity and Reflection 2 – Booking form validation

##### Situation

As most of the forms on the web use validation to ensure that the input from the users is valid, the booking form was in need of validation. The booking form is the most complex form of LMS2 and certainly the most frequently used form, and thus validation needs to work flawlessly.

##### Task

My task was to implement validation on the booking form. Validation needed to be done client side, for a number of reasons. First of all, doing things client side makes the response time much faster. Moreover, doing the validation client side means that possible invalid data from the users will never be sent to the server. The task was to be implemented using JQuery’s Validator Module.

##### Action

I began the task by reading some of the documentation on JQuery Validator Module that the lead developer gave me to read. Some of the documentation wasn’t as well documented as expected, so I began doing m own research and then began implementing the JQuery code to validate the form. Whenever I needed help I would ask one of the lead developers who had previously used the Module. After some struggling, validation for one trip was completed. However, the form allows for multiple trips in the one form. I began enhancing the code to validate all the trips.

Validation code can be very messy at times, especially to cover edge cases, however, I wanted to ensure the code was clear and maintainable. When enhancing the code to incorporate multiple trips I decided to do some refactoring and eliminate duplication of code.

The last step of this task was to compile some end to end UI tests to test this functionality. As previously state, validation needs to be flawless, as a bug in validation can cause very unwanted outcomes. I began writing multiple validation tests, covering both the simple validation, to the edge cases. Edge cases includes multiple trips and also extra selection validation like luggage and baby seats.

##### Results

The booking form now had validation, not only for the single trip, but the validation code would validate all the selected trips. All the manual tests that I ran while implementing the validation code all passed. Writing the validations end to end UI tests ended up being more difficult that expected. The tests were found to be fairly unstable. First of all, not all inputs would output a error message on invalid input, some would just simply add a red border around the input to highlight it. This was harder to test than checking for an error message on the page. Secondly, some error messages were the same and thus when checking for an error message on the page could provide a false positive as the message could be on the page for a different input than the one we expect it to be for. Lastly, timing issues arose, especially when dealing with Google Maps not loading fast enough or sometimes not loading at all.

##### Learnt

I learnt a lot about the JQuery Validator Module. First of all I learnt how to use it, but more importantly, after having used it so much, I learnt some of the best practices when using this module. I also enhanced a lot of my JQuery skills, especially surrounding DOM manipulation. I learnt some complex RSpec to assist me with the issues I was having with the validation UI tests. I also learnt more about how Google Maps works and how to work with it in the UI tests.

## 2.3 Major Activity and Reflection 3

Major Activity and Reflection 3 – ID vs Access Token

##### Situation

The booking form was using the database table IDs – which were simply incrementing numbers – to update bookings in the database. The IDs were being stored in the ID of most of the elements in the form, to assist with validation and form submission. The issue is that element IDs can be easily altered. This implies that an attacker could easily increment or decrement the ID stored in the elements and potentially view or update another person’s booking details. This is a major security vulnerability of the system as it causes threats to both confidentiality and the integrity of the system.

##### Task

My task was to add another property to bookings and trips model called access token and then to change the booking form to access the database via the access tokens instead of the ID’s. The access tokens obviously need to be unique, and that was certainly a major part of the task. Moreover, the access tokens need to be randomly generated. Randomly generating the access tokens implies that an attacker could not easily guess another persons access token and thus have access to their booking, like the incrementing ID’s used to do.

##### Action

I started off by discussing with the lead developer the structure of the access tokens as well as the best way to randomly generate them. We decided that the structure of the access tokens would be a string of 16 characters in length and we also wrote down some pseudo code for the randomly generating algorithm. I then used a rails command to generate a new migration in which I added the access tokens property in both the booking and trip models. After doing so I started implementing the code to randomly generate the access tokens when creating both a booking and a trip.

The next step, and probably most complex step, was to refactor the whole booking form to use the access tokens instead of the IDs. This included not only changing the front end to output the access tokens as the elements ID, but also modifying the backend calls to expect an access token instead of an ID.

The last step was to fix all of the failing end to end UI tests. These sort of tests are very dependent on element ID as the web drivers requires them to identify which element to interact with. The refactoring this task required me to undertake, involved changing many element’s IDs which meant that a lot of the tests would now fail as elements could not be found.

##### Results

After all the work was complete, the booking form was no longer posing the extremely dangerous vulnerabilities it was posing before. The entire booking form was refactored to use the access tokens instead of IDs to communicate to the back end. Furthermore, as previously stated, this work caused many of the UI tests to fail due to element ID mismatch, but after some work on fixing the tests, the build was passing again.

##### Learnt

This task certainly opened my eyes on the importance of security when implementing software systems. I learnt a lot about security vulnerabilities and threats, but more importantly, I learnt about the best practices for implementing vulnerability free code and ways for attackers to cause damage to a software system, especially regarding web based systems.

## 2.4 Major Activity and Reflection 4

Major Activity and Reflection 4 – Generic Template

##### Situation

LMS2 was beginning to be a very complex system, with many different models that needed managing for the application to work. The models were being managed through the web, using CRUD calls to the back end, via various forms and links. Each model had their own separate views for these CRUD operations. All these views did very basic and very similar things, simply allow the client CRUD functionality for the model, however they each had their own HTML templates and JavaScript code. This made LMS2 very hard to maintain.

##### Task

My task was to implement generic templates, both for the HTML and the JavaScript that will be used by all the models that simply need functionality to CRUD the database. This would make the system much more maintainable as a whole and would allow for consistency throughout the system.

##### Action

I started off by having a look at how the system was currently managing the CRUD operations for these models, this involved both the client side (HTML, JavaScript) as well as the server side (Ruby Controllers). I wrote down the similarities that I found, as well as the slight differences between them. It was necessary that the generic templates would incorporate all the similarities whilst also allowing for the differences.

I then began implementing the generic HTML templates. The template needed to include a table, which would list out all the objects available for that client of the model, along with all its properties. Furthermore, the table should have a column for links to update or delete the related object. Moreover, a link also needed to be present to allow for the creation of new objects of the model.

After the HTML was complete, I commenced writing the JavaScript to run the events on the UI as well as send of AJAX request to the back end for CRUD functionality. Once both the HTML and JavaScript generic templates were complete, my next and final task was to refactor each of the models to use the generic templates instead of their own views and scripts.

##### Results

After implementing the generic templates and refactoring all the models to use the generic templates, a lot of duplicate HTML and JavaScript was able to be removed from the system. A total of 14324 lines of both HTML and JavaScript were removed from the system as a result. These changes made the system much more maintainable and certainly much more consistent for the users.

##### Learnt

Learnt

## 2.5 Major Activity and Reflection 5

Major Activity and Reflection 5 – Best In Place

##### Situation

Situation

##### Task

Task

##### Action

Action

##### Results

Results

##### Learnt

Learnt

# 3.0 Conclusion

Conclusion

# References

**There are no sources in the current document.**

# Appendicies

## Appendix A: Work Log

## Appendix B: Certificate Of Time Worked

## Appendix C: Reflective Notes

|  |  |
| --- | --- |
| Situation | Booking form needed to support Airport selection |
| Task | Implement Airport model view and controller |
| Action | 1. Discussed with boss and lead engineer about the schema for the airport table 2. Implemented Airport model 3. Implemented Airport views and controller 4. Finalized Airport CRUD 5. Integrated Airport selection into booking form 6. Wrote unit test for Airport model functions |
| Result | * Airport CRUD was very easy to implement with Rails assisted scaffolding command * A client could view, create, edit and delete airports which they want to support as pick up or drop of locations * A customer using the booking form could select airports that the client supports * Airport model Unit tests pass |
| Learnt | * A model is equivalent to a table in the database * Rails makes implementing new CRUD very easy for developer with all the supporting commands * Leant a lot of new powerful rails command * Updating the schema for a model (table) is also easily done * Controllers work closely with the views as they respond to requests |

|  |  |
| --- | --- |
| Situation | Validation was needed for booking form |
| Task | Implement Validation using JQuery for booking form |
| Action | 1. Did some research on JQuery validator 2. Began implementing validation for inputs for one trip 3. Enhanced the validation to incorporate validation for multiple trips in one booking 4. Wrote UI tests for validation |
| Result | * JQuery validator was not documented heavily * Was helped out by senior developer * Trip validation was in place in booking form * UI tests were a bit flaky especially due to Google Maps issues |
| Learnt | * Sometimes useable documentation isn’t available and experience is much more valuable * JQuery makes it easy to enhance features * Leant a lot of JQuery functionality, including JQuery DOM manipulation * UI tests are difficult to write at times, especially when replying on third party tools |

|  |  |
| --- | --- |
| Situation | Booking form was using incrementing ID to access DB – Security Issue |
| Task | Modify booking form to use Access Tokens instead |
| Action | 1. Investigated use of ID throughout the booking form 2. Added access token both to trip and booking model 3. Implemented functions to calculate unique access tokens 4. Refactored booking form to work with access token instead of ID 5. Fixed failing tests |
| Result | * ID was used heavily in booking form * Refactoring took a long time * A lot of tests began to fail due to element ID mismatch * Fixing tests also took a long time |
| Learnt | * Exposing incrementing ID to customer facing tools is a big security vulnerability as attackers can easily increment or decrement the ID stored in the page to alter unauthorized resources * Refactoring a big page takes a long time * Using element ID for testing is not the best approach and tests will fail upon changes. |

|  |  |
| --- | --- |
| Situation | A generic template was needed for CRUD operations of many models for consistency |
| Task | Create a generic template that all models can use to implement CRUD operations |
| Action | 1. Discuss with boss, what the general UI should look like 2. Began implementing generic and modular HTML templates for views 3. Began implementing generic and modular JavaScript to run the views 4. Began refactoring model CRUD operations using the generic and modular templates and scripts |
| Result | * My work resulted in a lot of code and mark up to be removed as generic templates took care of it * Code was much more maintainable, as a lot of the models were using the same code for simple CRUD operations * CRUD operations was enhanced and made simpler for the user due to the changes |
| Learnt | * Generic, modular and reusable code is very important in large software systems * Rails makes this easy by the use of partials * How to use partials * How to write generic and modular code and how to use it correctly * Making generic code makes it much easier and quicker to implement enhancements and will be applied to all models |

|  |  |
| --- | --- |
| Situation | Functionality for updating models needed enhancements |
| Task | Integrating ruby gem ‘Best In Place’ to views |
| Action | 1. Read up on ‘Best In Place’ documentation 2. Began integrating ‘Best In Place’ into application 3. Added ‘Best In Place’ features to views for easy updating of model fields 4. Wrote helper functions to test ‘Best In Place’ functionality |
| Result | * Best In Place needed extra configuration than expected to work with current version of Rails * Users can now update model properties by a simple click now thanks to ‘Best In Place’ instead of having to open a form for the whole model * Tests were added to test the newly added ‘Best In Place’ functionality |
| Learnt | * Ruby gems are very powerful and easy to add to an application, even if extra configuration is needed * Learnt how to use the ‘Best In Place’ gem to provide the user with a simply way of updating model properties * New functionality always means new tests and adding helper functions can really help in test cleanliness and maintainability |