

Report 2

Work Integrated Learning – SEB701

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

Software Engineering (EN40)

MonicIT (60 Days Placement)

EXECUTIVE SUMMARY

This report provides a brief overview of my experiences with industry based experience at MonicIT during my last thirty days of work experience. MonicIT is a small and new business located in Brisbane CBD. MonicIT was a perfect place to undertake my industry based experience as, being a fairly new company, there were many new projects that I could take part of, even without a great deal of experience. Furthermore, MonicIT is a very small business, consisting of a small developer team, both senior and junior developers. This meant that I was able to get a lot of mentoring from the senior developers.

TABLE OF CONTENTS

1.0 Work Place Background	4
2.0 Work Activities.....	5
2.1 Major Activity and Reflection 1	5
2.2 Major Activity and Reflection 2.....	7
2.3 Major Activity and Reflection 3.....	9
2.4 Major Activity and Reflection 4.....	11
2.5 Major Activity and Reflection 5.....	13
3.0 Conclusion	15
References	15
Appendices	16
Appendix A: Work Log	16
Appendix B: Certificate Of Time Worked	19
Appendix C: Reflective Notes.....	20

1.0 WORK PLACE BACKGROUND

This report covers 30 days of my time, while working part time as a junior software engineer at MonicIT. During my work experience as a junior developer, I was provided with the chance to work on many different projects. After working my first 30 days at MonicIT, they decided to undertake a complete overhaul of their major system called Limousine Management System (LMS). They called the new project LMS2. One of the senior developers and myself were the first to start working on LMS2. At that stage I was simply writing automated test scripts for end to end UI tests but once I moved into this new project, I commenced implementing the ruby on rails application for LMS2.

2.0 WORK ACTIVITIES

2.1 MAJOR ACTIVITY AND REFLECTION 1

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.

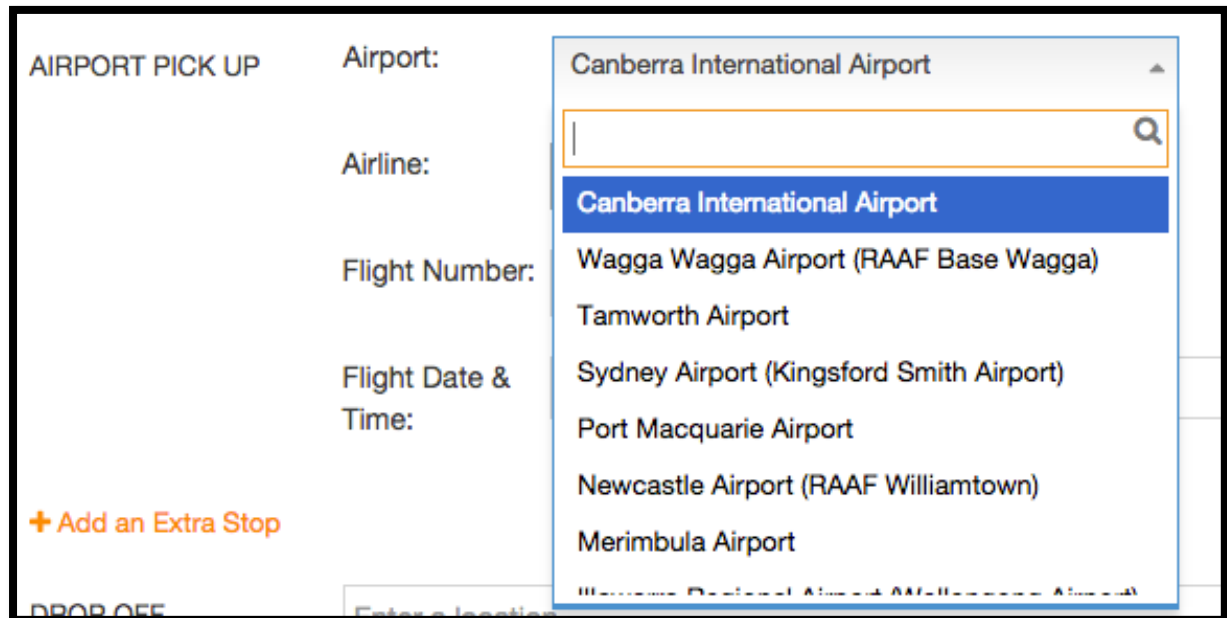
ACTION

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 easily 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, as illustrated in Figure 1. After a bit of tweaking, the unit tests and the end to end tests all passed.



The screenshot shows a flight booking form with the following fields and options:

- AIRPORT PICK UP** (Section Header)
- Airport:** A dropdown menu is open, showing a list of airports. The selected option is "Canberra International Airport".
- Airline:** A text input field with a search icon.
- Flight Number:** A text input field.
- Flight Date & Time:** A text input field.
- + Add an Extra Stop** (Button)
- DROP OFF** (Section Header)
- Enter a location** (Text input field)

The dropdown menu for the Airport field contains the following options:

- Canberra International Airport (Selected)
- Wagga Wagga Airport (RAAF Base Wagga)
- Tamworth Airport
- Sydney Airport (Kingsford Smith Airport)
- Port Macquarie Airport
- Newcastle Airport (RAAF Williamtown)
- Merimbula Airport

Figure 1: Airport Selection Combo Box

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

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 (Dziejma, 2004). 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 my 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 stated, 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 include multiple trips and also extra selection validation like luggage and baby seats.

RESULTS

The booking form now had validation as displayed in Figure 2. Validation was present 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 validation end to end UI tests ended up being more difficult than expected. The tests were found to be fairly unstable. First of all, not all inputs would output an 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.



PICK UP

Figure 2: Validation on Input

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

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 (Chong, 2007) (Bishop, 2004).

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 person's 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

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

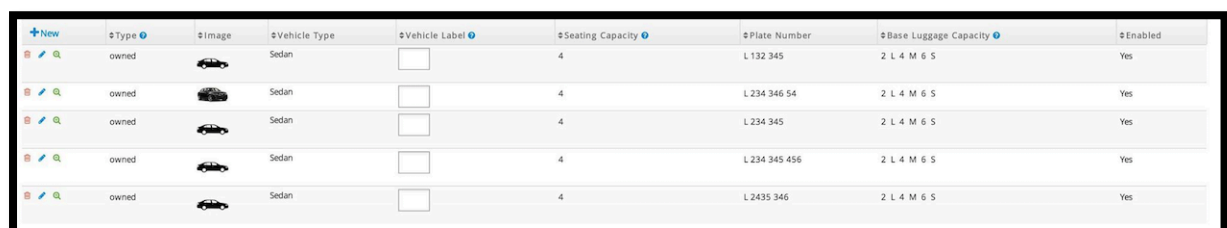
My task was to implement generic templates, both for the HTML and the JavaScript that will be used by all the models that simply need functionality to CRUD the database. This would make the system much more maintainable as a whole and would allow for consistency throughout the system.

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. Figure 3 depicts the described need.

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.























+ New	Type	Image	Vehicle Type	Vehicle Label	Seating Capacity	Plate Number	Base Luggage Capacity	Enabled
  	owned		Sedan	<input type="text"/>	4	L 132 345	2 L 4 M 6 S	Yes
  	owned		Sedan	<input type="text"/>	4	L 234 346 54	2 L 4 M 6 S	Yes
  	owned		Sedan	<input type="text"/>	4	L 234 345	2 L 4 M 6 S	Yes
  	owned		Sedan	<input type="text"/>	4	L 234 345 456	2 L 4 M 6 S	Yes
  	owned		Sedan	<input type="text"/>	4	L 2435 346	2 L 4 M 6 S	Yes

Figure 3: Generic Table (Vehicles)

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 54324 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

The task certainly opened my eyes to the real benefits of using generic template and to make use of reusable code when working with such large systems (Bishop, 2004). I learnt about the use of partials in the rails framework and how they can be used to make HTML templates much neater and can help to improve the maintainability of the system.

2.5 MAJOR ACTIVITY AND REFLECTION 5

SITUATION

The updating of models was frequently done but the way LMS2 allowed updating of models made this simple job a very tedious task. If a client wished to update a single property of a model they would need to click the edit link which would open up a form and load all of its properties into the form and then make the change and submit the form. Upon submitting the form, the whole model would be updated in the database.

TASK

My task was to modify the current procedure used in LMS2 for updating models to allow for a much smoother and more pleasant user experience.

ACTION

After conducting some research I found a rails gem called 'Best In Place'. This gem added helper functions that allowed for updating of model properties to be done 'in place' which also means that only that property would be updated, not the whole model.

I continued my research, focusing on that particular gem. After familiarizing myself with how the gem works, I began integrating the system to use 'Best In Place'. I modified the views to make use of the helper functions provided by 'Best In Place' which simply generated the necessary HTML for the functionality to work. Furthermore, the gem provided a JQuery widget which would handle events and fire AJAX requests for updating the property. I needed to hook the elements up to this widget for the full functionality to be complete. Thanks to the documentation I found on this particular gem, all of these were found to be simple tasks.

Following on from the integration of the gem into the current system, as always, I needed to write end to end UI tests to test the functionality. I started off by creating some helper functions which would assist in testing the 'Best In Place' functionality. I finished off by writing some simple tests that used the helper functions and that would ensure that the system was functioning as expected.

RESULTS

As I was integrating the 'Best In Place' gem into LMS2, I found that there was a mismatch in the version between the gem and the rails version that LMS2 was using. However, after some configuration modifications, the issue was fixed.

Once all the implementation was completed, the client could now easily update a property of a model by simply clicking on a property and editing it on the spot. The edit link however was not removed and the functionality to edit the whole model through the form was still in place.

LEARNT

I learnt about the real power of rails gem and how truly easy it is to add and remove these gems from a rails applications. I learnt a lot about the 'Best In Place' gem, not only how to use it, but also how it works. Moreover, I learnt how to write helper functions in RSpec and the best place to put them to allow them to be accessible in all the test suites.

3.0 CONCLUSION

My industry related work experience for the second thirty days gave me real world experience in the field that I want to work in, especially thanks to the introduction of the new project LMS2. Working on this new project as a developer gave me great experience in implementing web based applications, specifically ruby on rails. I learnt many valuable skills and technology during this time which will assist me through my whole career as a software engineer, especially in the work of web based development, which is the type of work I am interest in and enjoy thoroughly.

REFERENCES

- Bishop, M. (2004). Introduction to computer security.
- Chong, S. (2007, August). Enforcing Confidentiality and Integrity in Web Applications.
- Coleman, D. (1995). The application of software maintainability models in industrial software systems.
- Dziejma, A. (2004). System and method for a form validation engine.

APPENDICIES

APPENDIX A: WORK LOG

Day	Date	Hours	Type of work	Description of work
31	Monday, 20/05/2013	5	Implementation/ Autonomous Testing	Wrote more test to cover colleagues functionality Fix build
32	Wednesday, 22/05/2013	8	Implementation/ Autonomous Testing	Wrote some more tests to expose known bugs
33	Thursday, 23/05/2013	8	Implementation/ Autonomous Testing	Found bug whilst implementing and wrote test for it
34	Monday, 27/05/2013	5	Implementation/ Autonomous Testing	Began learning about the back end of a Ruby on Rails application
35	Wednesday, 29/05/2013	8	Implementation/ Autonomous Testing	Created a model and a controller class for the object Airport Fix build
36	Thursday, 30/05/2013	8	Implementation/ Autonomous Testing	Spent the day fixing the build
37	Monday, 03/06/2013	5	Implementation/ Autonomous Testing	Implemented validation on a form using jQuery's validator
38	Wednesday, 05/06/2013	8	Implementation/ Autonomous Testing	Wrote functionality to separate trip costs based on payment type
39	Thursday, 06/06/2013	8	Implementation/ Autonomous Testing	Documenting current features

40	Monday, 10/06/2013	5	Implementation/ Autonomous Testing	Implemented a form for emailing trip quotes Fix build
41	Wednesday, 12/06/2013	8	Implementation/ Autonomous Testing	Wrote test for email functionality
42	Thursday, 13/06/2013	8	Implementation/ Autonomous Testing	Finished confirmation page Wrote validation functionality for longitude and latitude presence from google maps Added functionality to include Subject when sending email of quotes
43	Monday, 17/06/2013	5	Implementation/ Autonomous Testing	Added UI enhancements to validation Fixed build
44	Wednesday, 19/06/2013	8	Implementation/ Autonomous Testing	Wrote functionality for user to log support tickets
45	Thursday, 20/06/2013	8	Implementation/ Autonomous Testing	Wrote test for functionality revolving logging a support ticket Fix build
46	Monday, 24/06/2013	5	Implementation/ Autonomous Testing	Wrote test for different stories Found a lot of bugs
47	Wednesday, 26/06/2013	8	Implementation/ Autonomous Testing	Worked on fixing the bugs and the build
48	Thursday, 27/06/2013	8	Implementation/ Autonomous Testing	Worked on autonomous tests to make them more stable

49	Monday, 01/07/2013	5	Implementation/ Autonomous Testing	Changed the booking form to use trip quote access tokens instead of trip id (security issue)
50	Wednesday, 03/07/2013	8	Implementation/ Autonomous Testing	Began working on a very big task which involved me making generic templates for others to use for CRUD
51	Thursday, 04/07/2013	8	Implementation/ Autonomous Testing	Continued working on the generic templates for CRUD
52	Monday, 08/07/2013	5	Implementation/ Autonomous Testing	Started implementing pages that use the templates
53	Wednesday, 10/07/2013	8	Implementation/ Autonomous Testing	Continued implementing pages using the generic templates Had a meeting with the team where I showed them how to use the generic templates
54	Thursday, 11/07/2013	8	Implementation/ Autonomous Testing	Worked on an automatic reload system for the generic pages when CRUD actions effect other sections
55	Monday, 15/07/2013	5	Implementation/ Autonomous Testing	Worked on more CRUD pages Started writing tests of the CRUD pages
56	Wednesday, 17/07/2013	8	Implementation/ Autonomous Testing	Introduced the "edit in place" idea to the team and started implementing it. Edit in place allows a user to edit an object without the use of a form

57	Thursday, 18/07/2013	8	Implementation/ Autonomous Testing	Continued work on integrating "edit in place" into the web app
58	Monday, 22/07/2013	5	Implementation/ Autonomous Testing	Started writing some initial tests for the "edit in place" functionality
59	Wednesday, 24/07/2013	8	Implementation/ Autonomous Testing	Worked with a colleague on some major refactoring and clean up of code
60	Thursday, 25/07/2013	8	Implementation/ Autonomous Testing	Worked with a colleague on optimization for some of the functionality

APPENDIX B: CERTIFICATE OF TIME WORKED

Attached at the end of this document

APPENDIX C: REFLECTIVE NOTES

Situation	Booking form needed to support Airport selection
Task	Implement Airport model view and controller
Action	<ol style="list-style-type: none"> 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	<ul style="list-style-type: none"> - 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	<ul style="list-style-type: none"> - 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	<ol style="list-style-type: none">1. Did some research on JQuery validator2. Began implementing validation for inputs for one trip3. Enhanced the validation to incorporate validation for multiple trips in one booking4. Wrote UI tests for validation
Result	<ul style="list-style-type: none">- 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	<ul style="list-style-type: none">- 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 relying 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	<ol style="list-style-type: none">1. Investigated use of ID throughout the booking form2. Added access token both to trip and booking model3. Implemented functions to calculate unique access tokens4. Refactored booking form to work with access token instead of ID5. Fixed failing tests
Result	<ul style="list-style-type: none">- 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	<ul style="list-style-type: none">- 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	<ol style="list-style-type: none">1. Discuss with boss, what the general UI should look like2. Began implementing generic and modular HTML templates for views3. Began implementing generic and modular JavaScript to run the views4. Began refactoring model CRUD operations using the generic and modular templates and scripts
Result	<ul style="list-style-type: none">- 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	<ul style="list-style-type: none">- 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	<ol style="list-style-type: none">1. Read up on 'Best In Place' documentation2. Began integrating 'Best In Place' into application3. Added 'Best In Place' features to views for easy updating of model fields4. Wrote helper functions to test 'Best In Place' functionality
Result	<ul style="list-style-type: none">- 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	<ul style="list-style-type: none">- 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

Situation	LMS2 needed to provide functionality for a client to issue a support ticket
Task	Implement the functionality for the support ticket issuing
Action	<ol style="list-style-type: none">1. Discussed this task with the lead developer and we settled on a mock up2. Commenced implementing link and form for support ticket3. Implemented functionality to email support ticket to MonicIT4. Wrote end to end UI test for this feature
Result	<ul style="list-style-type: none">- A link was present on the website which would open a modal with a form to submit the support ticket- Upon submission the details entered by the client would be sent to MonicIT- UI tests passed but were a bit unstable as checking emails is difficult in automated tests
Learnt	<ul style="list-style-type: none">- Form submission without persisting storage on database- How to use various ruby gem to send emails- How to test emails with RSpec

Situation	The external booking form requires functionality to allow customers to receive their quotes via email
Task	Implement the functionality to allow customers to receive their quotes via email in the booking form
Action	<ol style="list-style-type: none">1. A mock up was ready for the UI side of things and I started working on that2. Created the form to allow the customer to enter their email details3. Implemented functionality on the back end to gather the quotes and send the email to the customer.4. Implement links on the email to allow them to return to the booking form and book their trip.5. Wrote end to end UI test for this feature
Result	<ul style="list-style-type: none">- A form was present on the external booking form to allow the customer to enter in their details.- Upon submission of the form, the selected quotes would be sent via email to the customer.- Links were available on the email which would allow the customer to choose a quote and finalize the booking back on the booking form.- UI tests passed but were a bit unstable as checking emails
Learnt	<ul style="list-style-type: none">- Enhanced knowledge of how to send email to include links.- Learnt how to pass in extra parameters to a http request for a page through a link.- How UI tests work in terms of parallelism and how to write tests well and write ones which are vulnerable to fail due to parallelism issues.