Title Page

Insert you own title page here.

Make sure you include the names of all students in your project team.

Comments on this example report...

This report is good but received a B- grade because:

a) it is too short – several sections need further explanation

b) the student has not included a personal paragraph of individual learnings

c) the list of references is way too short

Note that this report was not from a team – it was a student working individually

---

Sections:

---

1. Introduction:

In creating this report I have focused on presenting the learning process that I have been through during the internship at Aderant, rather than specific details of the code produced (due to confidentiality). From my perspective, the majority of the time was spent learning the technology stack and the use of its components in order to implement the deliverables / functionality required.

2. Background:

Aderant: the company

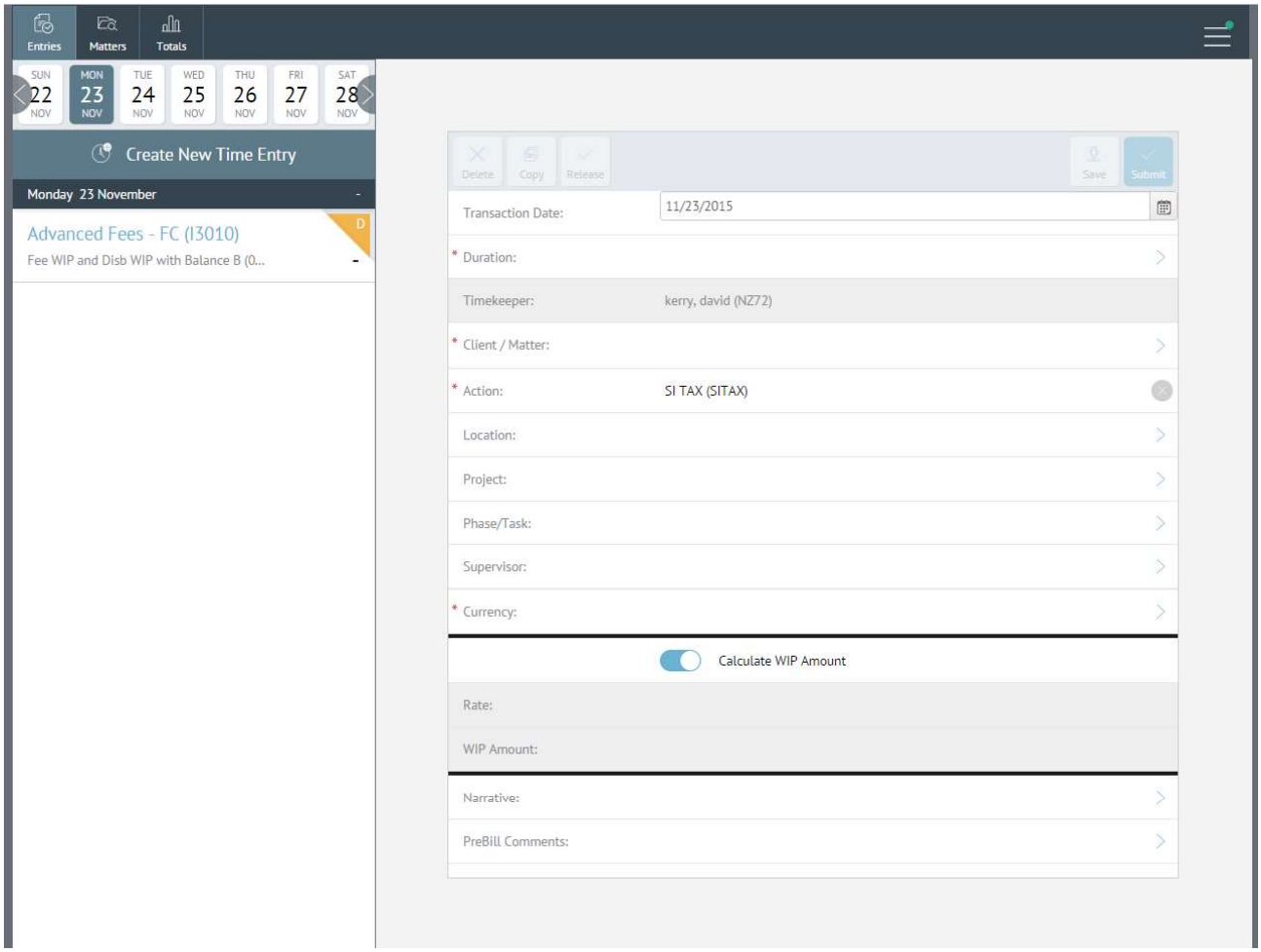
Aderant has a large suite of programs based around accounting for lawyers from day-to-day expenses, through to timekeeping for hours on customers cases, to compiling billing invoices. The product suite is targeted at larger companies where a set of applications can be deployed to each staff member. The software is designed to be highly customisable to the needs of any firm, where the tailoring is done 'in-house' by the law firms' own internal IT departments.

'On The Go (OTG)': a subset of the product suite

With the common use of smart phones there has been the ability to record various accounting information while not at the office, or even on a computer. This allows the users to update their accounts remotely, it could be directly after an event or task, or during transit when other work is not easily done. There is a subset of Aderant's application suite called 'On The Go' which is a set of Web apps which are mobile versions of their desktop counterparts. This is the team that three Massey University interns, myself included, joined for one semester.

Expenses: the product

To establish where the project was heading we can start with the most established 'On the Go' application which is called 'On The Go Time'. The purpose of this product is to collect the correct billing information to be able to account for, and bill the clients for the time that is spent working on each case for a client. This application has gone through a recent design phase from the ground up, and has an 'up to date' style, interface and functionality.



On The Go – Time. This was the 'template' for expenses to adopt.

One of the earliest products for mobile was the 'On The Go Expenses' application. The purpose of this application is to record expenses that are incurred as part of day-to-day business. This may be a taxi fare or a restaurant bill.

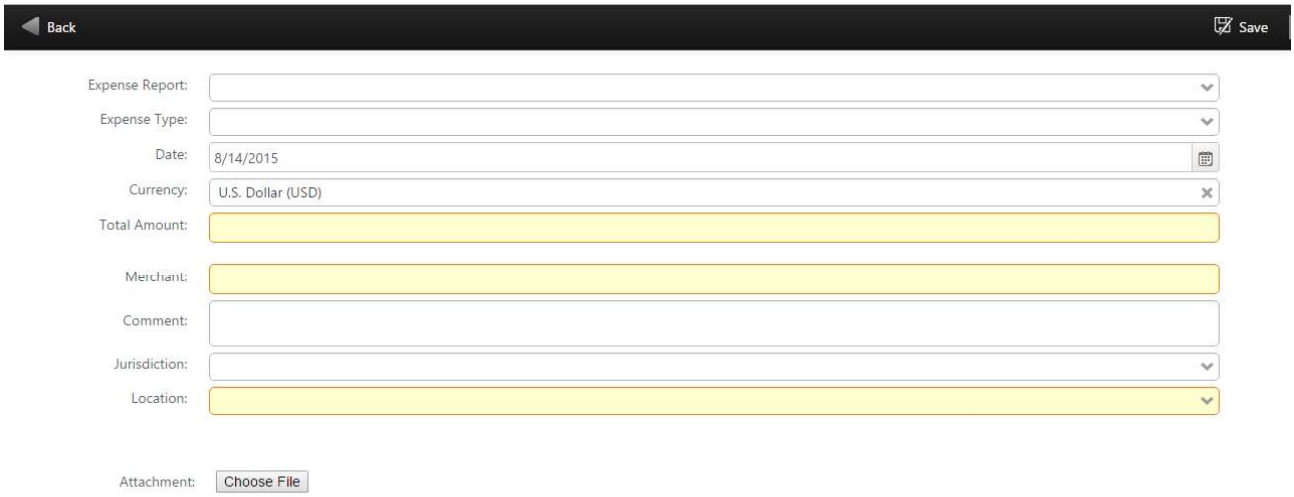
The expense 'receipt' can be associated to an 'Expense Report', given a date, an amount, a currency etc. At the most basic level this allows the company to reimburse for work expenses, but there is additional functionality that allows the expense to be billed to a client as a 'billable split' (a component of the total bill to be 'split' between the client and the firm, for example) for a proportion of the amount, or a 'non-billable split' where the bill (or a portion of it) is not billed to a client. The 'splits' in turn can be customised for any firm specific fields to be captured.

Our task was to bring the Expenses app in line with the more, up to date 'Time' interface and styling. Also additional functionality assigned as 'stories' (from the 'Agile' terminology) were to be implemented.

3. OTG Expenses: at the beginning

The original 'OTG Expenses'. This was the initial page, showing a history of the existing recipts.

When the original application was written, the 'expenses' app was targeted to a smaller screen device for the smart phones of the time. While allowing for some variation in screen size through utilisation of CSS (Cascading Style Sheets), the page layout is not dynamic; the content only expands and contracts to fill the page. The navigation of the app is provided by modifying the visibility of the underlying HTML DOM (Document Object Model) elements.



- A 'smart-form' showing the primary receipt fields. The fields shown are completely configurable by the firm.

As part of the customisability, fields on the forms may be added, modified or removed by the firm. These changes are applied on a central server which are then requested and updated by requests from the relevant applications. These are known as 'smartforms' and allow the applications to be modified by the firm as required.

A large part of the content in the app, including the smartforms, are asynchronously transferred and displayed once the elements are received. This allows the initial loading time to be kept to a minimum, and caching will help the speed of retrevial greatly once the initial load is complete and content is available locally on the device.

4. Getting Started:

Aderant, as many software companies have done, has adoped the 'Agile' methodology (Martin, 2003). We joined into the 'OTG' team 'scrums' (team meetings) and had our tasks set out as 'bugs' and 'user stories' (functionality to be added). 'Sprints' (revolving planning blocks to allocate intended work) were not planned in our time there, most likely as it did not make sense with our part-time schedule.

Getting up to speed was the first task as there was much to learn before any work could really begin. We were tasked with fixing existing bugs on the Expenses backlog. This developed familiarity with The IDE (Visual Studio), which I had come across this previously in a Web Development paper (158.358), however we had used very little of the IDE's functionality. Being set up in a commercial environment, and making the most of the environment, a lot more of the IDE is in use, such as the backlog for our (intern) team which we needed to get accustomed to.

Looking at the technology stack, albeit briefly as we have been asked to not disclose any of the specific technologies, it is quite complex. There are a lot of libraries and tools going into creating the complete application and gaining familiarity with the main tools was crucial to being able to later work on the product. It is not unusual to see a source file that contains three different languages or third party libraries in use within the same file.

The scale of the project was initially overwhelming. The project comprised several hundred files, and previously the largest program I had encountered was in the tens of files. At first it was difficult to know where to even begin looking to work on a issue. After recognising the 'Model View Controller' structure (Leff & Rayfield, 2001) of the project the files and their locations became much more intuitive. This design pattern is replicated over the higher level libraries as well.

Making our first code 'Check In', I had not encountered source control in my studies so far at Massey, but had some familiarity of the concepts and need for it from previous work. Keeping up to date was difficult as we were only in the office one and a half days a week, and every Thursday there would be three days' worth of changes to integrate. This was much better once we were moved to a separate and static branch, allowing us to have a stable base to work from at our 'part-time' pace.

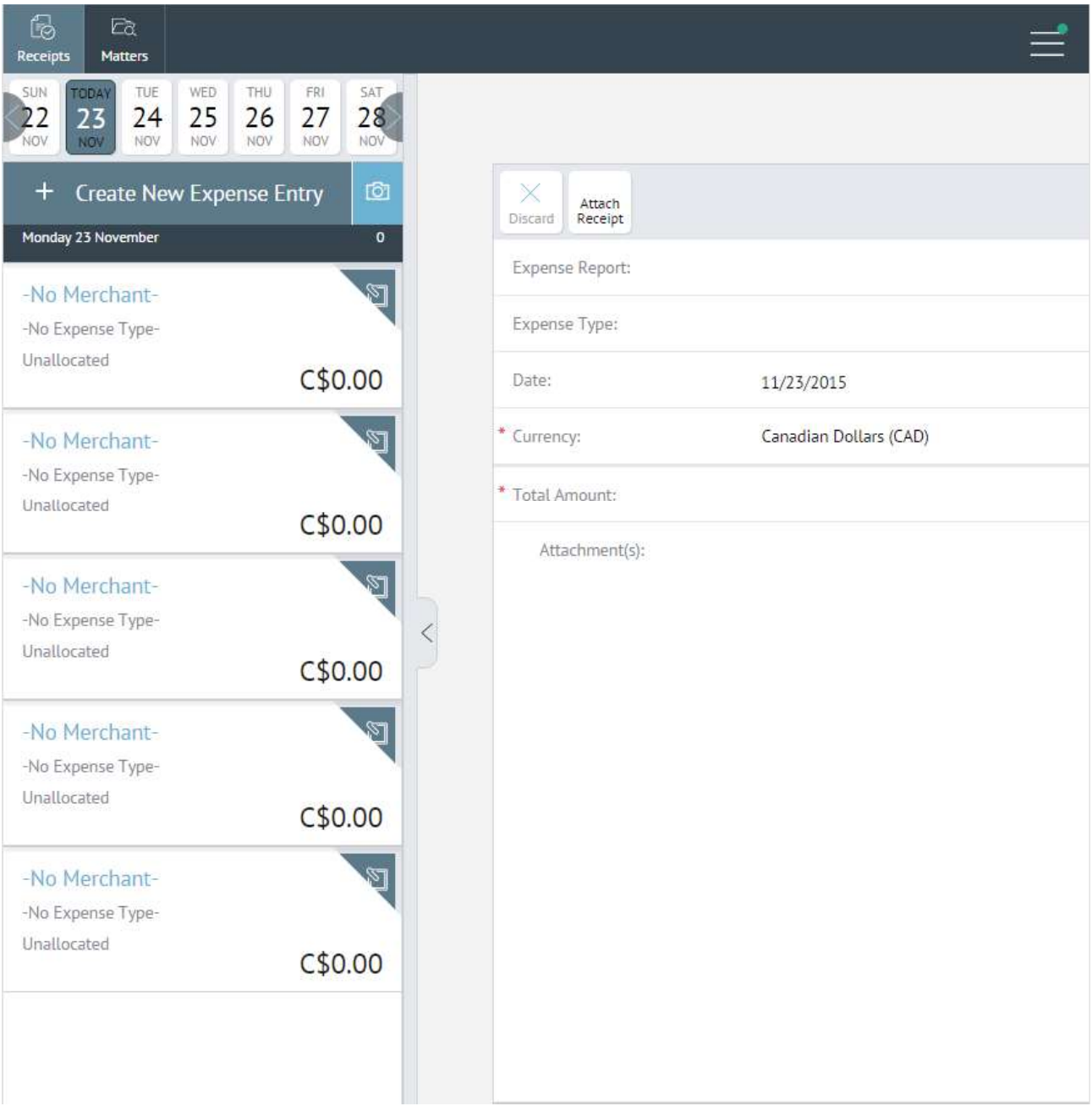
Continuous/automated testing has been implemented on the project, which runs a series of test scripts as part of the build process. It was great to see this in place and the very quick feedback on issues/errors in the code was helpful. Having a build 'fail' immediately allows you to resolve the issue quickly while the current task is still present in mind. Later in the project, I modified the test scripts to test some of the new functionality which was valuable to have some exposure to.

5. Implementation:

Split-view:

One of the first specific tasks I took on was to get the 'split-view layout' into the Expenses app. This breaks up the visible screen into four quadrants that are automatically sized and/or hidden as necessary for the device or as the window size changes. There are several layers of libraries in the project and this component was from the next layer up in the 'On The Go' (OTG) library. As 'Time' already had this functionality, the code for the split-view was removed from 'Time' and put up a layer into the 'On The Go' library. Then the calls to the relevant functions were relocated from local objects to point at the split-view components in the 'OTG' library. Once this was done we were able

to call the same functions and utilise the splitview components from within the Expenses application. I was able to later replicate this process to transfer some of the remaining styling and functionality regarding to screen width to get the medium screen size working in Expenses, where the left side slides in and out of view.



The 'middle sized' view from the split view. The left window is currently expanded, partially hiding the right frame.

It was great to see some OO code reuse in action, rather than understanding it from a purely theoretical standpoint. As the code base uses the Model-View-Controller paradigm, extracting objects and components was not too complex as they are kept fairly modular. This allowed the functionality to be transferred first, then the styling moved second, and finally inconsistencies checked and a general cleanup done, such as local variables that became redundant.

Navigation multiple views issue:

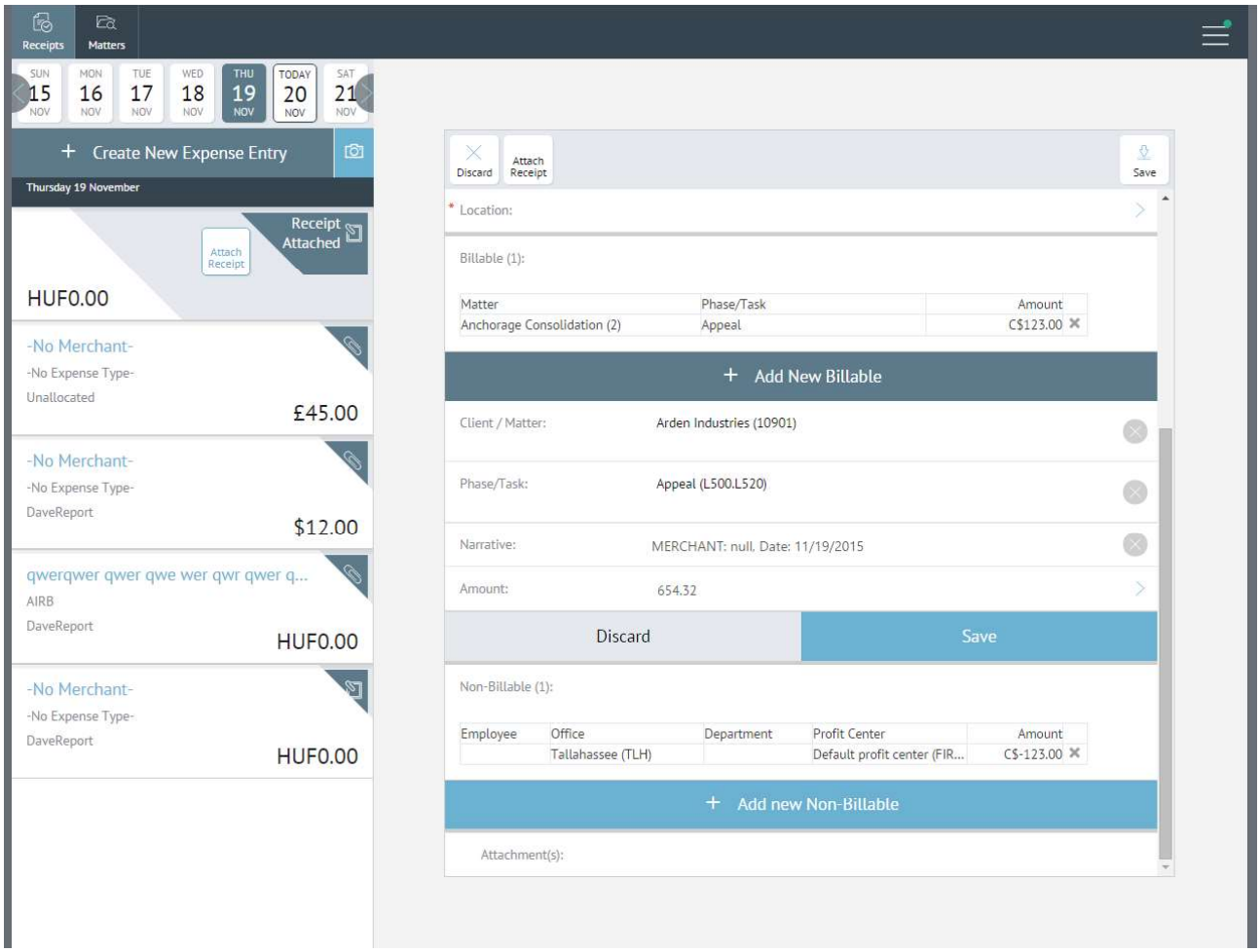
One of the problems that came up while working on the navigation of expenses is the extra navigation options on the 'Desktop' view. On a phone the navigation is purely linear. For example, when adding a 'billable split' to a form there is only one navigation point from this, which is to go back to the receipt that this billable is attached to. In doing so, the user can be prompted to save or discard their changes. However on the desktop, there is a receipt list on the left side and the user can directly go to another receipt while editing a 'billable' item. This causes a problem as the 'receipt' and the 'billable' are saved independantly. This would mean that a modal window would need to be made with a range of options:

- Discard All
- Discard Billable and Save Receipt
- Save Receipt and Billable.

This is not in line with the rest of the clear and intuitive interface. Two solutions were advised by the User Interface (UI) designers:

1. Disable all other navigation options while entering data to a billable
2. Have the billable expand 'inline' rather than on another view.

The inline expansion was the preferable option, as it was cleaner and simpler, and setting a 'disabled' state and visible style to the rest of the page seemed time consuming.



Inline billable and non-billable splits, all are visible and editable from the main receipt view.

I put the billable and non-billable components inline to the form; surprisingly a quick test showed it worked straight away (at least in the first test instance). The code is modular enough that it allowed these components to be active at the same time. Some 'navigation' cleanup was needed as the original functionality was driven by binary visibility flags of the billable and non-billable components. Also, some changes were made to the 'save' function to allow saving of the reciept as well as the billable split and non-billable split. Lastly, I added buttons to allow the addition of these components to the form as well as saving and discarding, and some cleanup to hide the tables if there were zero billables/non-billables.

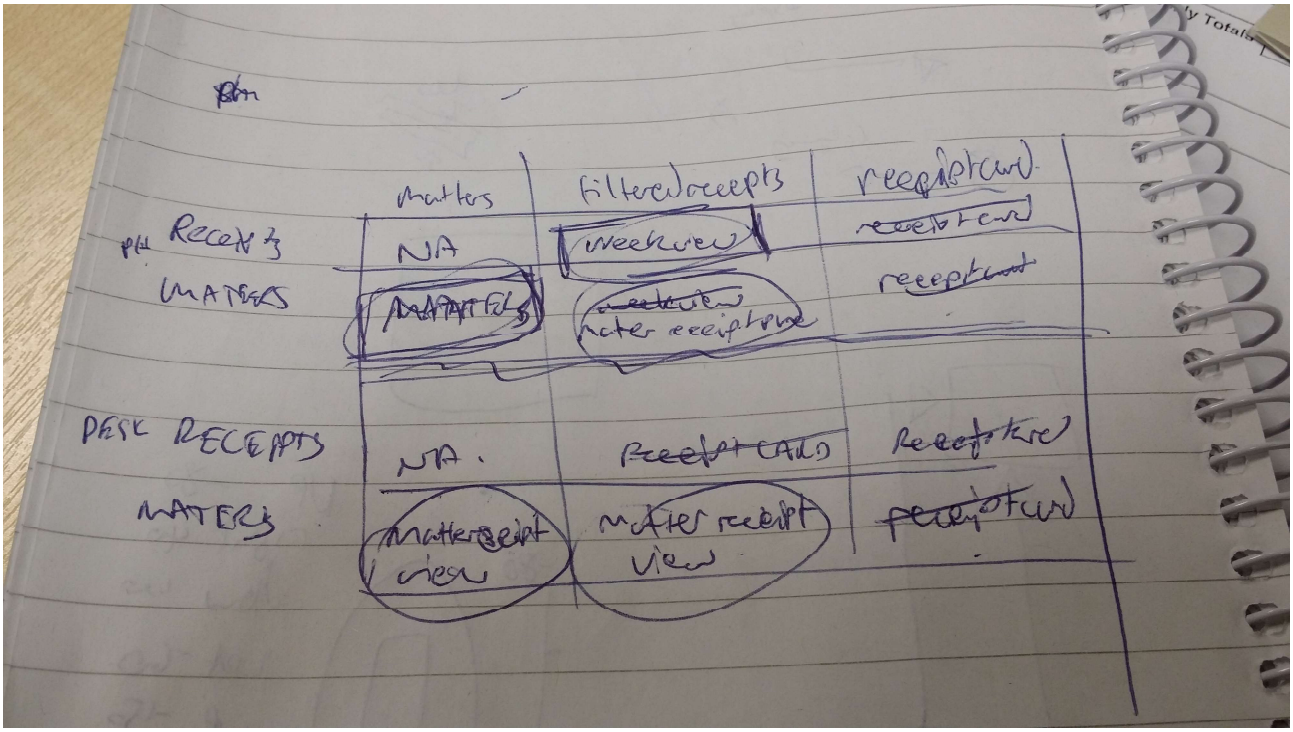
CSS details:

Especially toward the end of the internship, once the main functional changes were done, a surprising amount of time on the project was spent finding small discrepancies in alignment, or styling the components on the page. I found CSS quite challenging to get right, mostly because some functional details felt counter-intuitive. For example, even when an element is specifically sized, the size will be overridden by *any* of the elements that are contained inside it. This makes it difficult to locate the source of an oversize element display is not correct.

Another complexity is sub-pixel sizes and the rounding of fractional components. As the rounding is not completely consistent, it is difficult to programatically size a component from the window size, as sub pixel rounding can make a component one pixel larger or smaller than it needs to be, which can have flow-on effects. Such effects are the addition of an unwanted scroll bar, which in turn compresses the components horizontally, or causes other components to be misaligned due to their relationship to the oversized component.

Visibility control:

Some of the navigation became quite complex even for something as simple as visibility of a component. The intersection of a lot of variables required making a matrix to ensure correct boolean logic of display elements.



A table to verify the visibility state of the receipt view

If the device is considered a phone, in that the available screen size is small, then all views need to be seen on the right pane as this is the only pane it has. If it is not a phone, then only a subset of the views are shown on the right pane, as the higher level navigation views are always shown on the left pane.

The next variable is the two 'modes' that can be selected. The first is to navigate chronologically by day where a week is shown, and all the receipts are shown for the selected day. The second is to view the receipts by their assigned 'matter'. Both of these views have to navigate to a receipt view at the end point. An additional variable was added to keep track of the previous view, to be able to return along the correct navigation branch.

With the matrix completed it was now simpler to define when the views should be visible or not and to make the statements that applied the correct visibility state.

Asynchronicity:

There are a lot of asynchronous actions going on in the application. For example a basic receipt is shown by default, but in the background requests are being sent to populate additional fields, or expand inline components, such as a billable split smart-form. The project relies heavily on 'callbacks', to allow the actions to happen in a correct sequence. This adds a layer of complexity, and potentially the need to pass functions as parameters through many intermediary functions, where the order of execution is not always clear and the beginnings of understating of the term 'callback hell' (McKenna, 2012).

An example of a point where this became difficult is the inline expansion of the billable/non-billable smart forms. Ideally, once the form has loaded the content will be visible on the screen. If the screen is small or the smartform already fills the page the expanded form will be off the bottom of the page. There is an existing function that does this called scrollIntoView(). However calling this directly after the AJAX call to populate the form inline does not work as there will not be any content, so the scroll downward will be minimal and the form once loaded will still be partially off the page.

Intuitively you can put a callback on the Ajax call itself, but this will also be executed too early and the only solution is to place the scroll function in the code that populates the smart-form inline.

To me, this clearly illustrates the difference in implementation of the 'event driven' paradigm, as opposed to the 'procedural' approach. The next line of code that you need to put in may be in an entirely different part of the program.

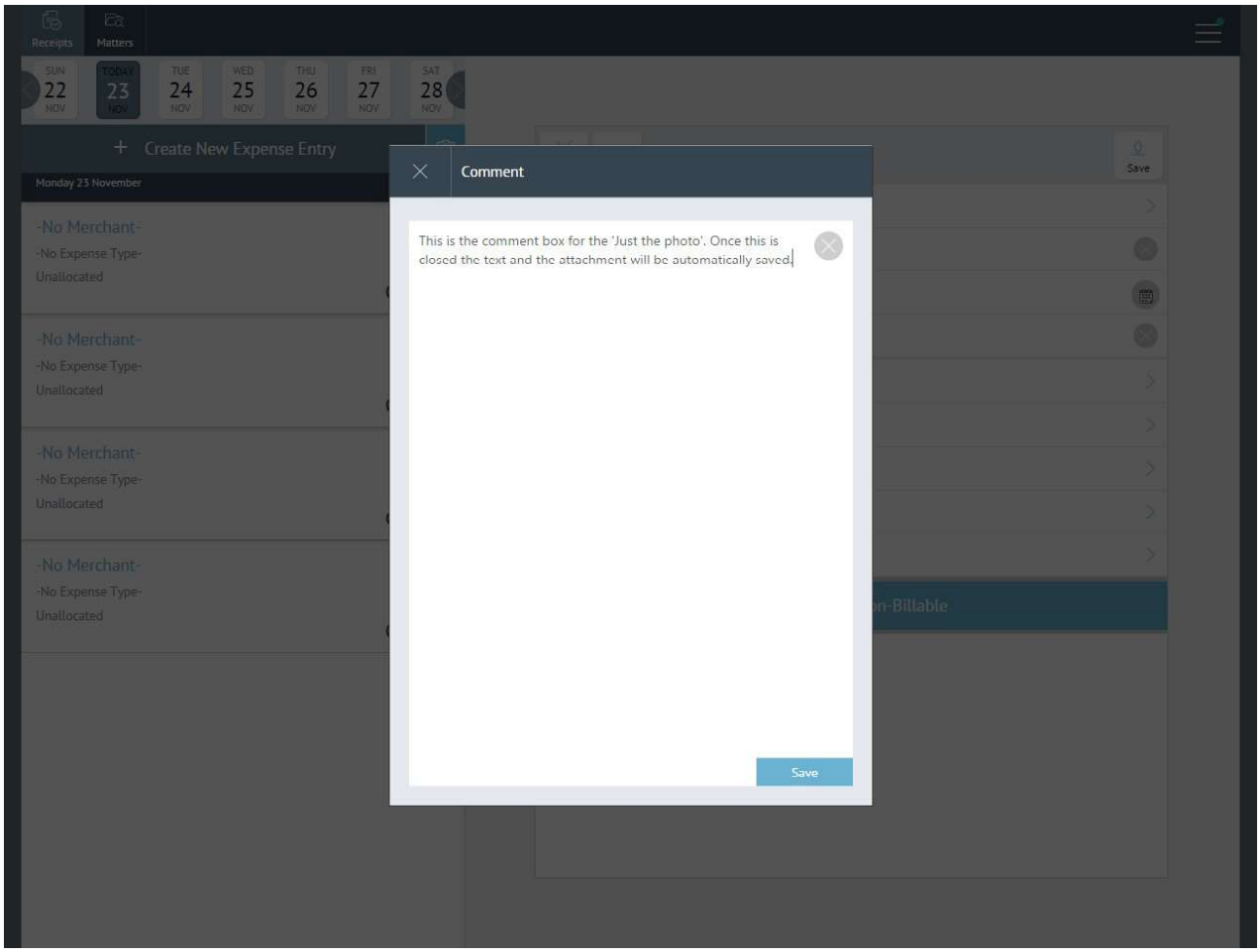
SVG icons:

For the left view there are icons that show the state of the attachment, either no attachment or that at least one is attached. I created these using an online tool to have a single paperclip indicating no attachment, or a paperclip with a photo to show the 'attached' state. Compared to graphical images, the SVG icons will take up significantly less bandwidth and have the additional capability to have styles applied. This means that if in future the colour palette for the application changes, the icons can be all modified to have a new colour scheme with code as opposed to regenerating all the icons with new colours from scratch.

Just the photo:

One of the stories in our tasks was to add a comment to the 'Just the photo' function. Just the photo allows the user to create a new receipt, for the selected day that contains 'Just the photo' and save it automatically. The rest of the fields can be filled in later, or by an assistant. The addition of a comment allows the user to give more context to photo, while still being a 'quick' feature.

There were several hurdles to overcome to get this working. On a phone, the receipt does not actually exist until you select one to view or create a new one, so there is no field to populate the comment text into. This was worked around by writing directly into the data object rather than applying the text to the comment section on the smart form.



'Just The Photo' comment box: implemented and functional.

A corner case example:

There were a lot of corner cases to be handled, one of which was resizing the window on the desktop. As the window size changes, the split view layout changes between the full desktop view, with both frames visible. The medium size is where the left frame slides on/off the left of the page. The smallest or 'phone' view is where just a single view is shown. When changing between the sizes, corrections need to be made to keep the view consistent. When the user sizes down the screen while editing a receipt on the desktop the receipt should remain visible (as opposed to going to a navigation view), which was not the outcome initially. Another is the 'back' navigation bar that appears when editing a receipt on a phone, this must be hidden as the view is sized up to the desktop as the navigation tools are always visible in the desktop.

6. Going forward and conclusion:

The majority of the functionality work is done. However, there is still a long way to go in testing for and making the 'corner cases' work as expected. Creating a new expense on another day will be saved correctly and appear on the correct day on the filtered list. The continuous QA will find more issues, especially once the cross platform testing starts and the software is run on different browsers and devices. I will be continuing doing this work further with my employment at Aderant.

The internship was a great opportunity to get familiar with programming in a production environment. It was informative to see a lot of the concepts that until now had been mostly theoretical. Things such as code reuse, lambda expressions, untyped languages, functions as variables, callbacks, asynchronous actions and general web programming. Going straight into a large project was quite challenging, but having the 'OTG Time' application as a general template was hugely helpful to see what we were aiming for, and as a reference for approaches on how to implement the changes.

Bibliography:

Leff, A, Rayfield, J.T. (2001) *Web-application development using the Model/View/Controller design pattern.* Publication *Enterprise Distributed Object Computing Conference, 2001 Proceedings Fifth IEEE international* Pg 118-127

McKenna, B. (2012). Roy: A Statically Typed, Functional Language for JavaScript. *IEEE Internet Computing*, (3), 86-91.

Martin, R. C. (2003). *Agile software development: principles, patterns, and practices*. Prentice Hall PTR.