# Cardiff University

Final Report

---

*Personal Finance Manager*

---

**Author:** *Thomas Whiddett*

**Student Number:** *C1130231*

**Supervisor:** *David W Walker*

**Moderator:** *Kirill Sidorov*

**Module Code:** *CM3203*

**Module Name:** *One Semester Project*

**Credits Due:** *40*

# ACKNOWLEDGEMENTS

I would like to thank Professor David Walker for his support and advice throughout the project. I would also like to thank my friends and family for their support during my time at Cardiff University.

# CONTENTS

**Contents** ... iii

**1 INTRODUCTION** ... 5
- **1.1 outline** ... 5
- **1.2 Aims and objectives** ... 5
  - 1.2.1 Aims ... 5
  - 1.2.2 Objectives ... 5
- **1.3 Initial assumptions** ... 6

**2 Background** ... 7
- **2.1 Wider context and the problem that has been identified** ... 7
- **2.2 Existing solutions** ... 7
  - 2.2.1 Quicken ... 7
  - 2.2.2 Mint ... 7
  - 2.2.3 You Need A Budget (YNAB) ... 8
  - 2.2.4 Money Dance ... 8
  - 2.2.5 Accountz ... 8
  - 2.2.6 Microsoft Excel Spreadsheet Template ... 8
  - 2.2.7 Summary ... 8
- **2.3 Potential use cases and stakeholders** ... 9
- **2.4 Theory associated with problem area and tools to be used** ... 10

**3 Specification and Design** ... 11
- **3.1 Overview** ... 11
- **3.2 Requirement Specification** ... 11
  - 3.2.1 User requirements ... 11
  - 3.2.2 Functional and Non-Functional Requirements ... 13
- **3.3 Design** ... 19
  - 3.3.1 Database design ... 19
  - 3.3.2 UI design ... 21
  - 3.3.3 Finalising design details ... 33
  - 3.3.4 TEST Cases ... 36

**4 Implementation** ... 43
- **4.1 General Structure** ... 44
  - 4.1.1 Database setup ... 45
  - 4.1.2 Templates ... 48
  - 4.1.3 Error Reporting ... 49
- **4.2 Login and Register** ... 51
- **4.3 Overview** ... 52
  - 4.3.1 Accounts, Transactions and charts ... 53
  - 4.3.2 CSV Upload ... 54
- **4.4 Reports** ... 56
  - 4.4.1 Customisable Reports ... 57
- **4.5 Student Finance** ... 60
- **4.6 Problems Encountered** ... 62

**5 Results and Evaluation** ... 64
- **5.1 Test CASes** ... 64
- **5.2 Strengths and Weaknesses** ... 68

**6 Future Work** ... 70

**7 Conclusions** ........ **71**
**8 Reflection on Learning** ........ **72**
**Bibliography** ........ **73**

# 1 INTRODUCTION

## 1.1 OUTLINE

Managing personal finances can be a daunting process but it should not be overlooked. Without proper organisation and budgeting, debt can easily be incurred. A vast majority of people own multiple bank accounts, whether it's a current account, savings account or credit card. They can also be faced with phone bills and subscriptions for services such as Netflix or Spotify. All of which can make it increasing difficult to keep track of your finances. Credit cards and loans also require special attention to ensure repayments are made before the deadline. Different users potentially have varied needs and levels of understanding financial management. Take a young person heading off to university, they often move away from home, forcing them to be more responsible. However it may also be the first time they have had the independence and responsibility to manage their finances. With rent, living costs and various other fees to consider they are likely to have a stricter budget and hence must be managed correctly.

This project will begin by exploring existing financial management products and identify any drawbacks, before developing the project requirements. These requirements will set the basis for the rest of the project. The approach will be adapted accordingly and will progress to formulating a design for the system. The implementation process will be documented before concluding with an evaluation of the finished product and a reflection over the course of the project.

## 1.2 AIMS AND OBJECTIVES

### 1.2.1 AIMS

The core aim of this project is to develop a system that will aid the user in managing their personal finances. The sub aims are to produce a piece of software that is both functional and aesthetically pleasing.

A breakdown of how this will be achieved is provided by the objectives below.

### 1.2.2 OBJECTIVES

1. Research existing personal finance management software to gain an insight into their functionality and why they are unsuitable.

2. Identify a user base to focus on and a new solution to the problem.

3. Explore the available tools and technologies to assist in implementing the solution.

4. Define a detailed requirement list for the system.

5. Design the system to meet the requirements identified in step 4.

6. Implement and develop the designed system, identifying any problems encountered along the way.

7. Analyse the results of testing the system.
8. Review the project and evaluate how it could be developed further.

## 1.3 INITIAL ASSUMPTIONS

Initial thoughts are to implement a web application for ease of access however this will present some questions about security. After uploading their finances, some form of categorisation would be recommended to help produce a helpful overview and report. Assigning categories to transaction would also provide a means to identify spending habits and trends.

# 2 BACKGROUND

## 2.1 WIDER CONTEXT AND THE PROBLEM THAT HAS BEEN IDENTIFIED

Having identified users owning multiple account, this often means individuals having their money spread out and sometimes across various banks. This makes it increasingly difficult to see an overview of their finances. With the rise of online banking and bank statements providing an itemised list of transactions, it is becoming easier. However, it is still difficult to identify spending habits or periods when expenses were higher than they should have been. This presents a problem when it comes to budgeting. A slip up or lapse in your budget could go by unnoticed with small increases in expenses each week, but this will eventually add up and could take you by surprise. Combining accounts and transactions into a central system can help overcome this difficulty. Having access to your financial information in a central location could provide key information and statistics, for example, your net worth. The use of reports and infographics can also provide extremely helpful information from numerous accounts across various time periods. This is exactly what is required to keep track of spending habits and ensure a well organised budget.

There are existing products that cover these features, which will be discussed in further detail in the next section. They can range from a simple customised spreadsheet to a fully featured specialised package with a monthly subscription. Depending on the users needs, these products will have a range of advantages and disadvantages that will be identified and discussed.

## 2.2 EXISTING SOLUTIONS

There are currently various existing applications for managing a users personal finance. By reviewing some of these, drawbacks can be identified and developed in this project to address the problem. The existing solutions range from a simple spreadsheet to standalone applications and web based applications. The products that will be discussed further include, Quicken, Mint, You Need A Budget (YNAB), Money Dance, Accountz and a Microsoft Excel Spreadsheet template for personal finance management.

### 2.2.1 QUICKEN

Quicken is a standalone application for use on both Windows and Macintosh, it also offers a mobile application [1]. It's a general all round financial management tool, with lots of features including, categorisation, budgeting and bill tracking, that connects directly to your bank account. However there is not a free version, the cheapest package starts at $39.99 for the starter version, which offers the most basic features. While the premium version costs $104.99 that includes additional features such as investment tracking. Being a pay only service, many users may decide to use an alternate system with a free version available.

### 2.2.2 MINT

Mint offers very similar features to Quicken and is also available on both desktop and mobile devices [2]. Their service is also offered for free which makes it very accessible to users both on the go or through the web. It connects directly to your bank account,

and so pulls the users transactions straight from their account. Features include an overview of spending based on these transactions, bill reminders, a budgeting system and advice on reducing fees and saving money. They do also offer a credit score check. However their main drawback is only being available in the US.

### 2.2.3 YOU NEED A BUDGET (YNAB)

You Need A Budget is a budgeting focused application that provides a simple and clean user interface to create and manage a budget [3]. You can add an account simply for tracking the balance for said account, or as a budgeting account. The money and transactions in this account will make up the budget. Features include adding scheduled transactions, categorising transactions and setting a budget amount for each category. Based on the transactions in your account, YNAB will inform the user how much of the budget is remaining. Similar to Quicken and Mint, YNAB links to your bank account to gather the transactions. They offer a 34 day free trial, but from there a $5 per month subscription or a single $50 a year payment is required.

### 2.2.4 MONEY DANCE

Money Dance is a standalone application, which similar to the previously discussed applications, connects to online banking to download the transactions [4]. Alternatively to this, it also allows the user to add transactions without the need to connect to a bank. Being able to import files from other sources, such as .CSV, makes this process much quicker. They also provide features to view a summary of the finances with graphs and reports as a visual aid, and investment tracking that downloads current prices automatically. Their mobile application is free to download; however it does require a full version of the desktop app to function that costs $49.99.

### 2.2.5 ACCOUNTZ

Accountz is another standalone desktop app based in the UK, but you can access the data anywhere using their mobile application [5]. This again is free but requires the paid version of the desktop application, costing £39.99 as a one off payment. Features include investment tracking, importing data from external sources, such as online bank accounts in a CSV or QIF file format. They also offer budgeting and reporting based on pre-defined categories that can be assigned to transactions. However this seems rather restricted as the available categories and reporting options seems fairly limited.

### 2.2.6 MICROSOFT EXCEL SPREADSHEET TEMPLATE

Finally Microsoft Office Excel offers various templates for financial management [6]. They are rather simple, making use of the standard Excel spreadsheet. Transactions are added manually and are used to generate simple reports. These are lacking any advanced features but for what is provided it is easy to use for low-level financial management.

### 2.2.7 SUMMARY

While the majority of the existing solutions are fairly easy to use, some are quite complex and offer a wide range of advanced features, such as investment and stock tracking. Mint, for example, also provides recommendations on how to save money based on users transactions. These additional features sometimes take away from the basic functionality or an application will specialise in one area. YNAB for example

focuses on planning and creating a budget and consequently lacks in variation for the reports. While it provides great functionality with a simple and clean user interface for budgeting, it is rather difficult to see a detailed overview of your accounts.

None of the explored applications provide a means to view a break down of loans or tax. The only information available is viewing a previous transaction where the money was paid into your account or debited. Some did provide the option to add upcoming or scheduled transactions such as direct debits, but details of a loan including the total amount received or the total amount repaid, are not easily accessible. Another feature that seems to be overlooked in these existing products is incorporating payslips from a job, be it part time or full time employment. As part of a payslip, details are provided for the total amount earned for the current tax year, the total tax paid, bonus received, any deductions such as loan repayments, or mortgages [7] [8]. This kind of information isn't available in the applications explored. Finally, while most have some form of classification method, or tags for the transactions, it is still difficult to drill down into specific categories. For example, electricity bills as part of the bills parent category.

These drawbacks present further development in the initial problem of personal finance management. The existing solutions that have been explored cater for the general user and may focus on a specific section of financial management. For a student who is faced with the prospect of managing their finances for the first time, they would have to utilise more than one existing application. Not only does a student have to keep to a strict budget, they also have to keep track of their student loan throughout their time at university, and any income earned in a part time job. The project will continue to address the problem based on the needs of a student.

## 2.3 POTENTIAL USE CASES AND STAKEHOLDERS

*Expand on use cases identified in initial plan and incorporate users needs from the drawbacks of the existing solutions above.*

As mentioned above, this project is going to focus on a main use case for students. They are most likely needing to keep to a strict budget. This will depend on various factors such as any income from loans or part time work and numerous expenses including rent and living costs. Student loans play a big part in their financial management. Sticking to a budget may depend on when instalments from the student loan are due to deposit into your account. Therefore keeping a record of what has been received so far, what is still to come and instalment dates are important. Loan repayments will also become crucial once finishing university. This will tie into their employment as repayments are generally deducted from their pay, once the student is earning above a certain threshold. So keeping a running tally of the total amount left to pay will be beneficial.

The project will not only be a useful tool for students but for the general user wanting to keep track of their finances and employment. This will be a secondary use case, for the likes of self employed working professionals. Being self-employed requires strict records regarding tax, to be able to file their own tax return. Therefore having a clear overview of their earnings for the tax year and a breakdown of the tax paid will prove essential.

## 2.4 THEORY ASSOCIATED WITH PROBLEM AREA AND TOOLS TO BE USED

To accomplish the goals of the project and produce a practical product, some background research into tax, payslips and student loans was required. By understanding what information can be provided on a payslip will help when designing how the system will work and how to process the employment records. Similarly understanding how tax is deducted and handled for someone's earnings and details of student loans will have to be taken into consideration in the design phase. Every payslip will have the earnings and any tax that has been paid. However there are various other income and deductions that could be present. These include gross pay, total amount after deductions, bonuses and commission, sick pay and maternity, paternity and adoption pay. Deductions could consist of the following, tax, national insurance, pensions, student loan payments and child maintenance. A few miscellaneous options could also be deducted or repaid such as parking permits and trade union subscriptions. A payslip will also show the tax code and period and as well as a summary of the current financial year (from 6th April to 5th April) [7] [8].

The types of finance available from a student loan are tuition fee loan, maintenance loan and maintenance grant. A tuition fee grant and course grant is available for part time students. Bursaries and scholarships are also available from the university or college, however these and the grants do not have to be paid back. Repayment depends on what plan you are on. Plan 1 is for students who started before 1st September 2012 and plan 2 is for those who started on or after 1st September 2012. For plan 1 repayments start when the student has an income above £17,495, although this may change each year. For plan 2 they start when the income is above £21,000. Interest is also charged on the loan and starts being added from the first loan instalment. The rates are 0.9% for plan 1 and the Retail Price Index (RPI) plus 3% for plan 2, currently 3.9%. However for plan 2, this initial interest rate only applies while studying. After leaving your course it changes to a variable rate that is dependent on income. The rates are RPI for £21,000 or less, RPI plus up to 3% between £21,000 and £41,000 and RPI plus 3% for over £41,000. The interest amount is added every month to the total owed [9].

The tools that are going to be used throughout this project are HTML and CSS for generic website coding. JavaScript and JQuery will be used for client side validation, visual effects and for the use of AJAX. This will be used to create asynchronous calls to the database and thus allowing the system to process data requests without having to constantly refresh the page. Sass is going to be used as a pre-processor for CSS as it provides extra functionality such as variables, nested rules and generally allows for the faster creation of stylesheets. PHP and MySQL are going to be used for back end scripting and database functionality. PHP is commonly used for server side programming and works very well with HTML, AJAX and MySQL. This combination will provide functionality between client side code and the database with PHPMyAdmin handling the database and MySQL. Finally for responsive web design the project will follow the general design set out by Ethan Marcotte book, Responsive Web Design [10]. This will make use of percentage and relative length sizing, as well as media queries to achieve a clean responsive design.

# 3 SPECIFICATION AND DESIGN

## 3.1 OVERVIEW

One of the objectives identified in the introduction was for a clean, easy to use interface. Managing finances can be a laborious task so an efficient interface will help speed up this process. This will be the main focus of the requirements, as well as a responsive design. For a student on the go, having access to the app on their portable devices will be necessary to keep up to date with their finances. This will also enable them to check their budget while out and about, which could affect their decisions if they discover they do not have many funds available.

Also mentioned in the initial assumptions was to create a web application. The reasoning behind this is to accommodate the user across various devices. The advantages of a web app over a standalone app is that it can be accessed anywhere, providing they have an internet connection. Publically available Wi-Fi is becoming more and more accessible, while this does pose a potential security risk; there are a few measures that can be taken to reduce this threat. Such as using HTTPS and Secure Sockets Layer (SSL). HTTPS is a protocol that uses the SSL for secure communication over a network that can protect against man in the middle attacks. This is achieved by encrypting page requests to a web server and the data returned from this server. Building a standalone application would mean developing both a desktop and mobile version, whereas a web app with a responsive design would function in both environments.

The next steps will identify user requirements based on the main use case of a student. Not having a physical client to draw these requirements from meant exploring and specifically defining what will be required from the main user. This can be seen throughout the next section where the students needs will be explored, followed by creating an initial high-level functionality of the system.

## 3.2 REQUIREMENT SPECIFICATION

### 3.2.1 USER REQUIREMENTS

With students being the main use case for the project, this section will attempt to identify any tasks or scenarios that a student may come across when faced with managing their personal finances. Firstly starting off with student loans, then employment, budgeting and finishing with general everyday tasks that will require financial management.

**Student Loans:**

- Applying for a student loan and receiving confirmation of the final amount to be received over the year.
- Checking instalment amount and the date this will be received.
- Checking how much of the student loan has been received so far.
- Once finishing the course, to see the total loan received throughout the whole period of university.

- How much needs to be paid back after interest has been added.
- How much has been paid so far and what is left to pay.

**Employment:**

- See a breakdown of their payslip,
    - Total earned for this period.
    - Total tax paid this period.
    - Any other income or deductions of importance.
- See an overview of total amount earned this financial year.
- See an overview of total tax paid this financial year.
- View the pay rate and hours worked, if it is a part time job.
- Keep a record of hours worked to check correct amount is received.
- Check correct amount of tax has been paid at the end of financial year.

**Budgeting:**

- Set an amount limit for each month.
- Set an amount limit for individual categories, such as rent and bills, or food shopping.
- See how much of the budget is remaining.
- See an overview of what has been spent.
- Predict income over a period to help determine budget limits.

**Everyday Tasks:**

- View previous transactions.
- Check bank balances.
- Confirm a transaction has been processed, for example confirming money transfer between accounts or that employment earnings have been paid.

These scenarios will form the basis of developing a solution. A system can be devised to function around each main area identified above. While the scenarios identify every possible task, the reality is that a user may not want to create a budget, or may not have a job and so would not need to record any employment information. The everyday tasks, like viewing account balances or previous transactions, are ones that would be required by every user of the system. Therefore an overview should be provided that will show any accounts added, along with their balances and recent transactions. Any budgets, student loans or employment records should also be displayed if they exist. Further sections of the system will allow creation, editing and analysis of student loans, employment and budgets. The functional and non-functional requirements below will look into each of these sections in detail.

### 3.2.2 FUNCTIONAL AND NON-FUNCTIONAL REQUIREMENTS

The requirements listed below have been split into sections relating to a part of the system. These are General requirements, Overview, Accounts and Reporting, Student Loan, Employment and Budgeting. Following on from the requirements a design will be developed.

#### GENERAL

| Functional Requirements | Acceptance Criteria |
|---|---|
| A user must be able to register an account | Demonstration of:<br>• User registering using their name, email address and creating a password.<br>• Confirmation of registration.<br>• Being able to log into the system. |
| User must be able to log into the system. | Demonstration of:<br>• User entering their email address and password.<br>• Being logged into the system and presented with the home page. |
| User must be able to log out at any time. | Demonstration of:<br>• Locating and clicking the log out button.<br>• Be logged out of the system. |
| The user must be able to delete their account and consequently any data associated with it. | Demonstration of:<br>• Clicking delete account and confirming deletion.<br>• Account having been successfully deleted.<br>• The account details should not be able to log the user into the system. |

| Non-Functional Requirements | Acceptance Criteria |
|---|---|
| The system must be secure. Each transaction must be uploaded and stored securely. | Demonstration of:<br>• HTTPS enabled for secure access. |
| Only a user with the correct email | Demonstration of: |

| | |
|---|---|
| address and password combination must be allowed to log into the system. | • In-correct login attempt.<br>• Failed attempt to access application pages without logging in. |
| The system must have a clean and intuitive design, so that the user is able to quickly and easily navigate the application. | Demonstration of:<br>• Accessible navigation on both desktop and mobile devices. |
| The application must be responsive so that it can be used easily on a variety of devices. | Demonstration of:<br>• Application working on mobile device.<br>• Displayed appropriately. |

**OVERVIEW AND REPORTS**

| Functional Requirements | Acceptance Criteria |
|---|---|
| The system must provide a means for the user to add a bank account. | Demonstration of:<br>• User being able to specify account name and balance<br>• The account should be displayed in the accounts section with the specified details. |
| A user must be able to edit or delete bank accounts that have been added. | Demonstration of:<br>• Editing part of a created account such as account name.<br>• Successful name change.<br>• Clicking delete bank account option.<br>• Confirmed deletion and account should no longer be visible in accounts section. |
| System must allow the user to manually enter individual transactions as well as uploading transactions from an external source (i.e. downloaded from banking website in the form of .CSV .QIF .OFX). | Demonstration of:<br>• Entering individual transaction details and uploading .CSV file.<br>• Specified transaction details should appear in the transaction list. |
| System must allow transactions to be edited so the user can change, description, date, amount or category | Demonstration of:<br>• User changing transaction details |

| | |
|---|---|
| of the transaction. | such as amount.<br>• This change should be accepted and visible in the transaction list. |
| A user must also be allowed to delete any uploaded transactions. | Demonstration of:<br>• A user selecting a transaction to delete and confirming.<br>• Transaction should be removed from the transaction list and therefore deleted. |
| Each transaction must be allocated to a category, either from a pre-defined list or by creating a new one. | Demonstration of:<br>• Assigning a category to a transaction.<br>• Upon submission this should be saved and visible in the transaction list. |
| The system must be able to produce reports of the uploaded transactions represented by infographics | Demonstration of:<br>• Various graphs and charts representing transactions for each account, |
| A user must be able to select the criteria to determine which transactions will be displayed in the report. This could be by data range, category type, and specific amount. | Demonstration of:<br>• A user selecting criteria for the data to be displayed.<br>• The result should be displayed with inforgraphics. |

**STUDENT LOAN**

| Functional Requirements | Acceptance Criteria |
|---|---|
| The user must be able to add a student loan. | Demonstration of:<br>• User being able to specify student loan name and instalments.<br>• The loan should be displayed with the details and an overview of the loan amount. |
| A user must be able to edit or delete a student loan that has been added. | Demonstration of:<br>• Editing details of the student loan, such as a name change. |

| | |
|---|---|
| | • Successful name change.<br>• Clicking delete student loan.<br>• Confirmed deletion and loan should no longer be available. |
| The system must allow the user to add instalments for the loan, of a specific type, i.e. maintenance loan, tuition fee etc. | Demonstration of:<br>• Entering instalment date, amount and type.<br>• Specified instalment should be visible in the loan overview and be displayed as a previous or upcoming instalment depending on the date. |
| A user must be able to edit or delete existing instalments. | Demonstration of:<br>• User changing instalment details such as date.<br>• This change should be accepted and visible in the instalments list.<br>• Deleting an instalment.<br>• The instalment should be completely removed from the loan. |
| The user must be able to see an overview of the loan, including the total loan amount and the next instalment date. | Demonstration of:<br>• Key loan details should be visible from the student loan section, such as total amount received.<br>• Upon adding extra instalments, these details must be updated. |

**EMPLOYMENT**

| Functional Requirements | Acceptance Criteria |
|---|---|
| The system must provide the functionality for the user to add a job, including a name and pay rate. | Demonstration of:<br>• User creating a job account.<br>• User being able to specify a name for the job and its starting pay rate or salary. |
| A user must be able to edit or delete a job that has been added. | Demonstration of:<br>• Editing details of the job, such as a |

| | |
|---|---|
| | pay rate<br>• Successful change of the pay rate, with confirmation.<br>• Clicking delete job.<br>• Confirmed deletion, the job and any associated details must no longer be visible. |
| The system must allow the user to add payslips to the job account, this should include key details such as hours worked, gross pay, deductions, final net pay, etc. | Demonstration of:<br>• Entering payslip and required details.<br>• Upon adding a payslip, the total earned and tax paid for the current financial year should be updated.<br>• A record of the payslip should be available in the job overview. |
| A user must be able to edit or delete existing payslips. | Demonstration of:<br>• User changing payslip details such as net pay.<br>• This change should be accepted and visible within the payslip overview.<br>• Deleting a payslip record.<br>• The payslip and associated details should be removed from the job account. |
| The system should allow the user to enter a record of hours worked. | Demonstration of:<br>• User entering date and hours worked.<br>• This should update the job overview with the added hours. |

**BUDGETING**

| Functional Requirements | Acceptance Criteria |
|---|---|
| The user must be able to create a budget for the current academic year. | Demonstration of:<br>• A budget being created.<br>• The user being present with the next steps to set up the budget for the year. |

| | |
|---|---|
| A user must be able to enter a limit to pre-defined categories relating to various income and expenses. None of which are required, but cover a wide enough base to account for most options. | Demonstration of:<br>• Setting a limit for certain categories. Such as food or travel limits. |
| The user must be able to define an account the budget will refer to. | Demonstration of:<br>• A user selecting an account that has already been added.<br>• The transactions from this account will be used to calculate the remaining budget. |
| The system must display the budget, amount consumed so far and the remaining funds. | Demonstration of:<br>• The user being able to visibly see the budget limit and the remaining funds.<br>• Adding a transaction should update these amounts. |
| A user must be able to edit or delete the budget and any limits set. | Demonstration of:<br>• User changing limits set on various categories, this will update the budget.<br>• Changing of budget, such as the associated account.<br>• Deleting the limit set for a category should remove that category from the budget.<br>• Deleting the budget should remove the budget and any limits set within the budget. It should not affect the transactions within the associated account. |

Figure 1 below, shows a diagram representing the system at a high level. This has been created based on the requirements above but is likely to change during the design process. The main pages of the application are highlighted in green.

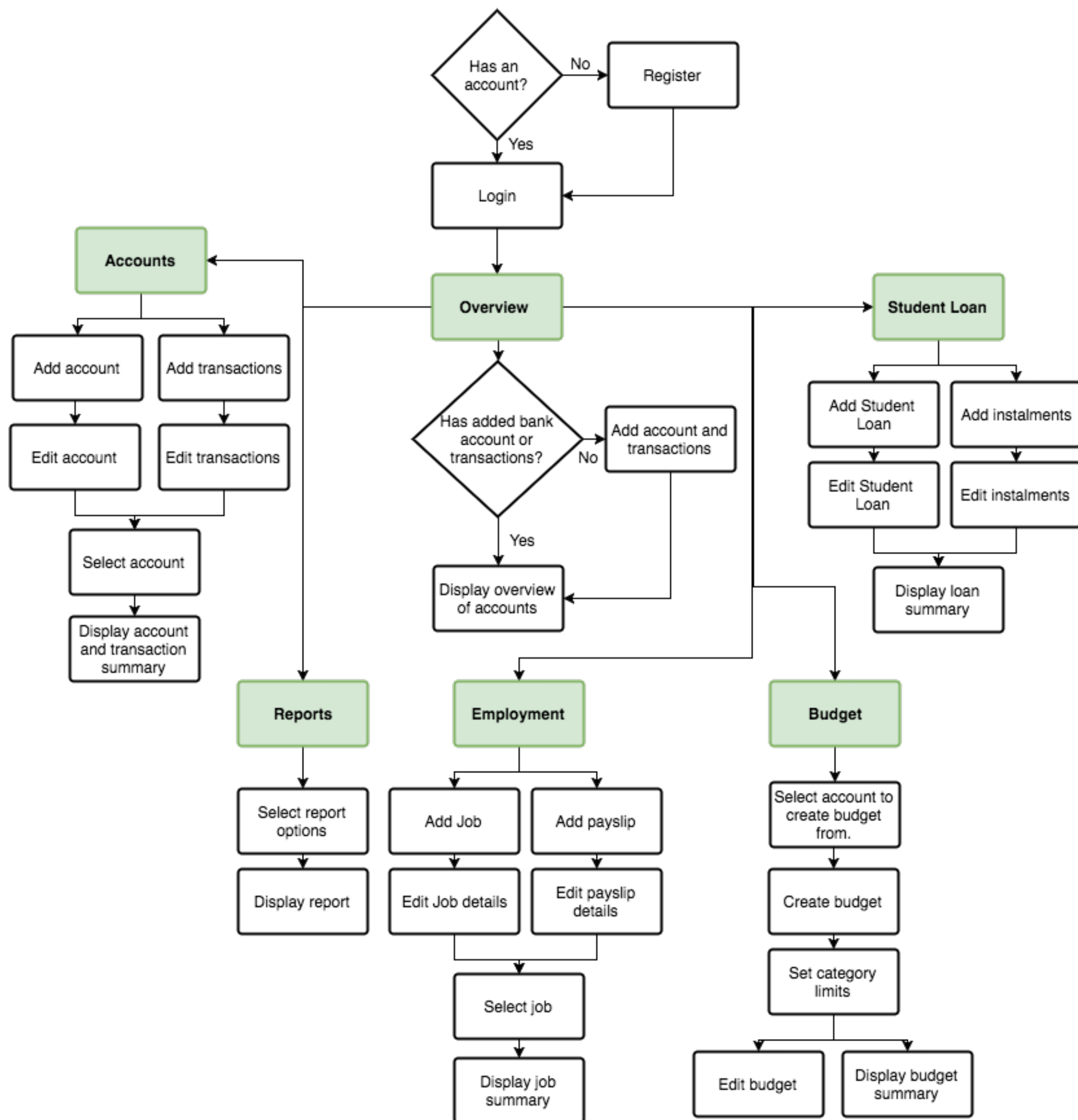


**Figure 1 - An abstract diagram to represent components of the system**

## 3.3 DESIGN

### 3.3.1 DATABASE DESIGN

The decision to use a relational database rather than NoSQL for example was mainly due to existing experience with relational databases. The content naturally presents itself in a structured manner with relationships between users and their associated accounts, loans etc. The benefits of indexing data and complex SQL queries available with relational databases will also play a huge part in representing the stored data to the user through the system. However, if the system were to significantly grow in size then a NoSQL alternative may have to be considered due to scalability and performance issues that could occur. This can be explored later in future work section of the project.

The database design has been developed from the requirements to create a system architecture that should represent each area of the system. Initially student loans and

jobs were going to be stored as an account within the *'accounts'* table. However as the complexity of each section increased the amount of data to be stored grew. This led to each major area being built up of multiple tables to store data specific to its area.

Figure 2 shows the Entity Relationship Diagram (ERD) for the database design. Each major section is colour coded accordingly, white for users, blue for accounts and transactions, yellow for student loans, green for employment and purple for budgets.

Once a user registers, their details will be stored in the *"users"* table. An active state is also created; this will initially be set to false to force the user to confirm their account through email activation. However this feature will only be implemented if time constraints permit. A users details will be accessed through a login request.

A user will be able to add multiple bank accounts, with the name, start date, opening balance and overdraft/credit limit set upon creation. A boolean *'credit_card'* field will specify if an account is a credit card as it will have to be handled accordingly. Within the account, multiple transactions can be added, specifying a transaction date, description and amount. Each transaction must be assigned a category, with the available options defined within the categories table. A child category will be referenced back to its parent category through its parents id. If a parent category doesn't exists its value will be null. Split transactions stem from an initial transaction, where the total amount from each split transaction must equal the amount set in the initial transaction. Finally a running balance is calculated within the *"accounts"* table, its value is updated each time a transaction is added. This will allow quicker access for reports rather than having to query and calculate the value every time.

As part of the *"student_loans"* table, a running total of each maintenance loan, maintenance grant, tuition fees and total loan is calculated when an instalment is added. When an instalment is added the user must specify the year period (e.g. 2015/2016), date, amount and instalment type. The types are predefined within the *"instalment_types"* table as maintenance loan, maintenance grant etc.

Similarly each job is broken down for each payslip received, updating the net pay and tax paid each time one is added. As seen in section 2.4, payslips can consist of various deductions and income types, including bonus pay and student loan deductions. To allow these details to be retrieved at a later date, they are stored in a separate table along with the associated payslip id. This is similar to the "split_transactions" table in that the total amount of each sub item of a payslip must add up to the total pay within the *"payslips"* table. The *"tax_periods"* table will represent each month within the financial year, defined as *"M1"* through to *"M12"*. This will allow a faster lookup when it comes to displaying the payslip details. As part of recording hours, the *"job_hours"* table will store records of days worked, from start to finish time, with the *"hours"* field being calculated from the start and finish times.

Finally the *"budgets"* table allows an overall budget to be set for the defined length. This amount should equal the total allowance set within the *"budget_items"* table. The *"budgets"* table is also connected to the *"accounts"* table so that a budget can be created for a specific account, which can then be used to calculate the remaining allowance from the transactions within the defined account. The *"budget_items"* table is linked to the

categories table so that an allowance can be set for a specific category and hence the remaining allowance can be calculated from the transactions added to the associated category and account.

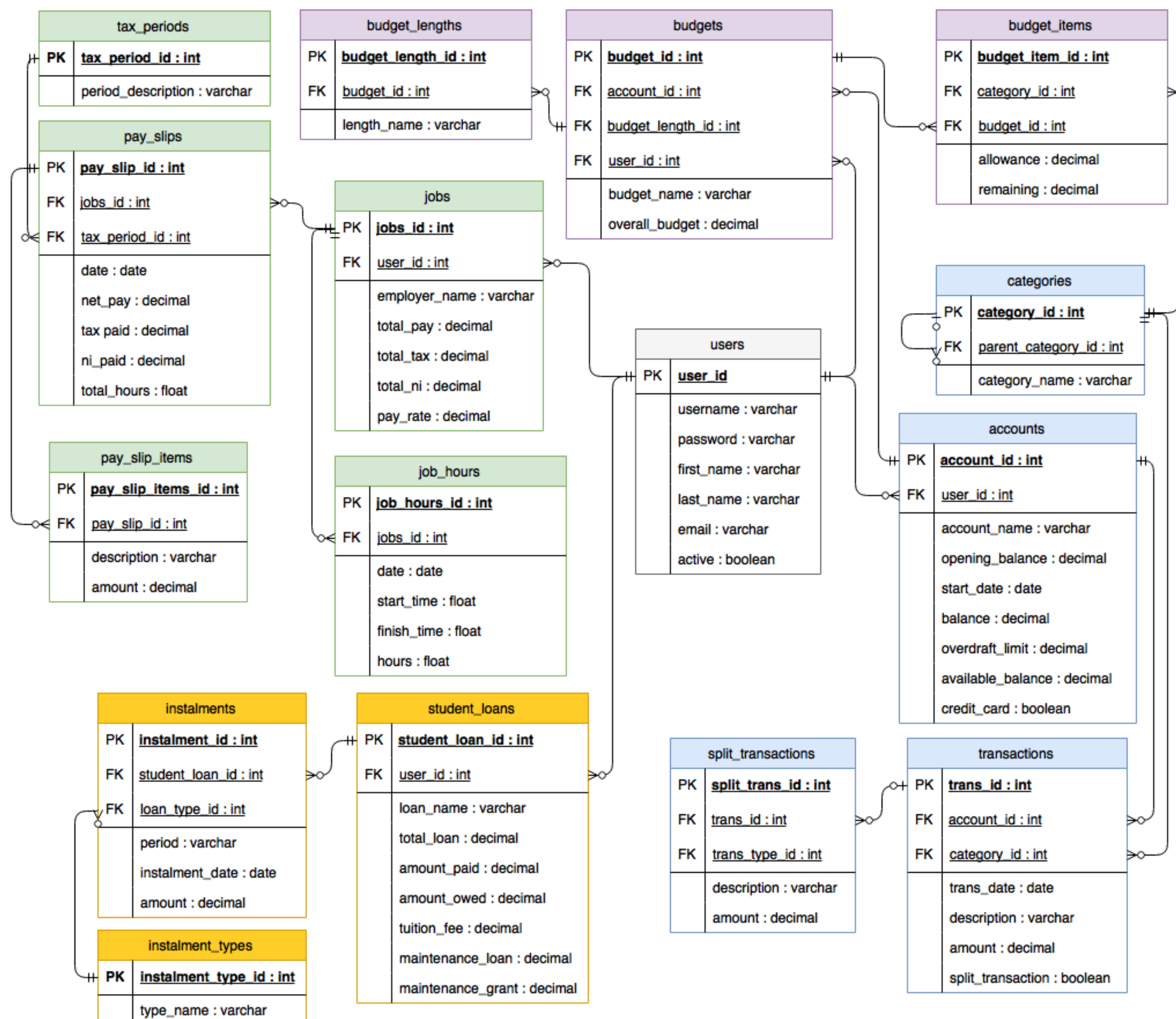

**Figure 2 - Entity Relationship Diagram (ERD) to represent the database**

### 3.3.2 UI DESIGN

With a database design complete the user interface can be developed to provide a clean responsive experience to the users. As per the objectives set out in the introduction. To achieve a functional responsive layout, a mobile first design approach has been used. Usually starting with a full featured desktop design, the developer has to gradually degrade the features and remove content to allow it to function on mobile devices. However with a mobile first approach this is flipped on its head. By developing a platform that performs perfectly and efficiently on a mobile device, the developer can then gradually add features to make it friendlier for larger devices. The enhancement process over degrading ensures designs are consistent throughout and alleviates situations where the mobile version is seemingly neglected.

Firstly a product name and logo had to be developed. Initial thoughts were to incorporate the pound symbol into the logo or name. So inspecting the alphabet for letters that could be merged with the pound symbol resulted in the letters “B”, “C”, “E”, “F” and “O”. After some research into words and phrases relating to money that

included one of these letters, the name “Bread” was settled on. The relation to money comes from cockney rhyming slang “bread and honey” meaning “money”. But also tracing back to the bible, it can be linked with the expression “earning a crust” referring to being able to afford to buy bread. Figure 3 shows some initial design concepts and incorporating the pound symbol into some of the identified letters in the alphabet. The final development of the logo is show in Figure 4.

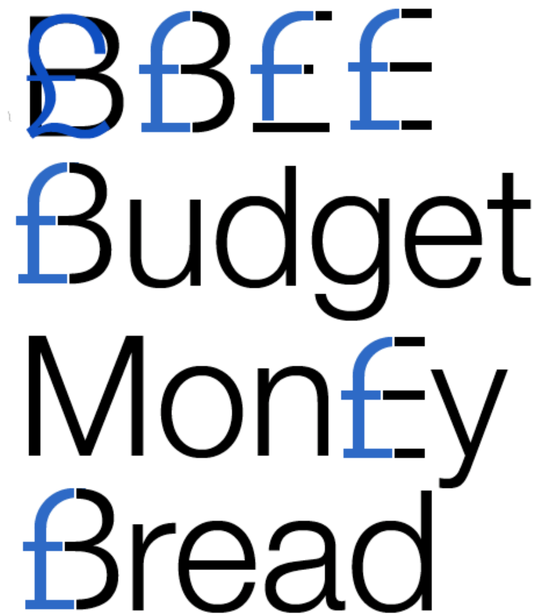


**Figure 3 - Initial logo design concepts**

**Figure 4 - Final development of logo**

The following figures depict the user interface designs. They have been created using Balsamiq and show both the mobile first design as well as a desktop version. Each area of the system has been designed so that it is easily accessible from every page while maintaining a consistent structure. Consistency is key to achieving an easy to use interface and will also help during implementation as the core structure can be set up as a template, this will help reduce redundant code.

### LOGIN AND REGISTRATION

The login and registration pages are both fairly simple, with one directing to the other as needed. Initially both were on the same page but the design looked cluttered and with further thought it seemed unnecessary to include the registration form every time the user wishes to login. In Figure 6, the first name and last name fields have been omitted and removed from the users table in the database. The reasoning for this action is because it is not relevant or required and hence wasted space within the database. Figure 7 shows the original login and registration page.

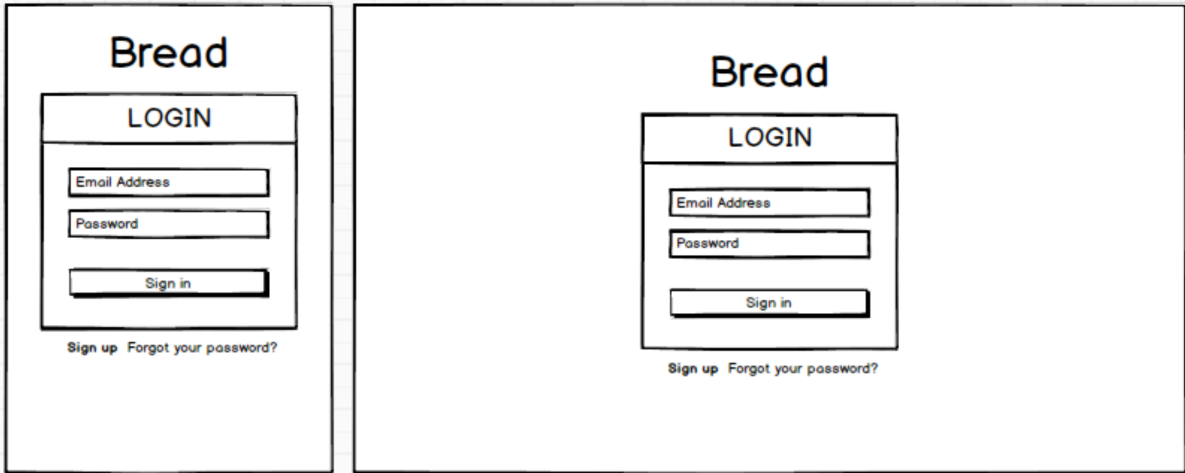


**Figure 5 - Login design**

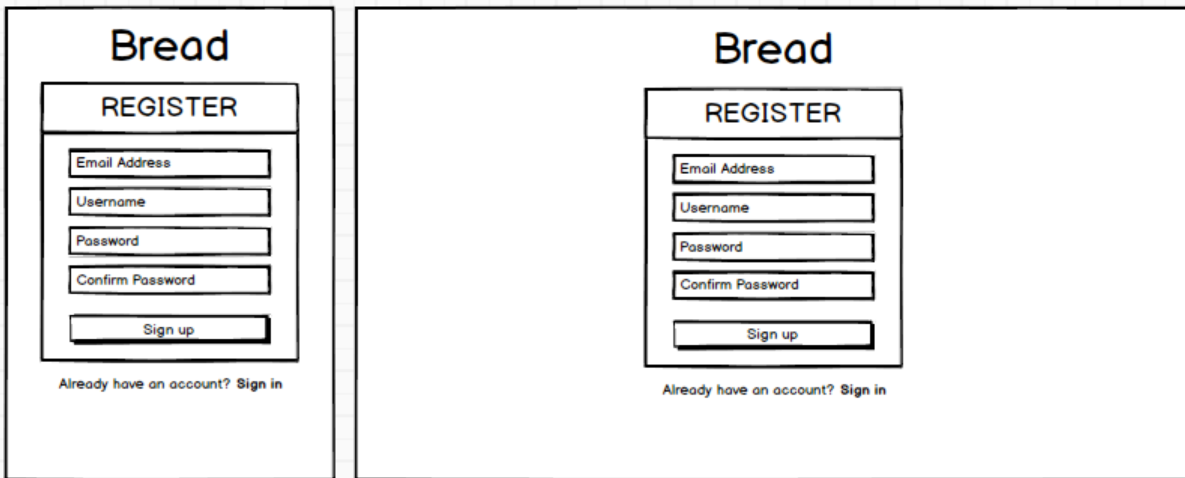


**Figure 6 - Registration design**

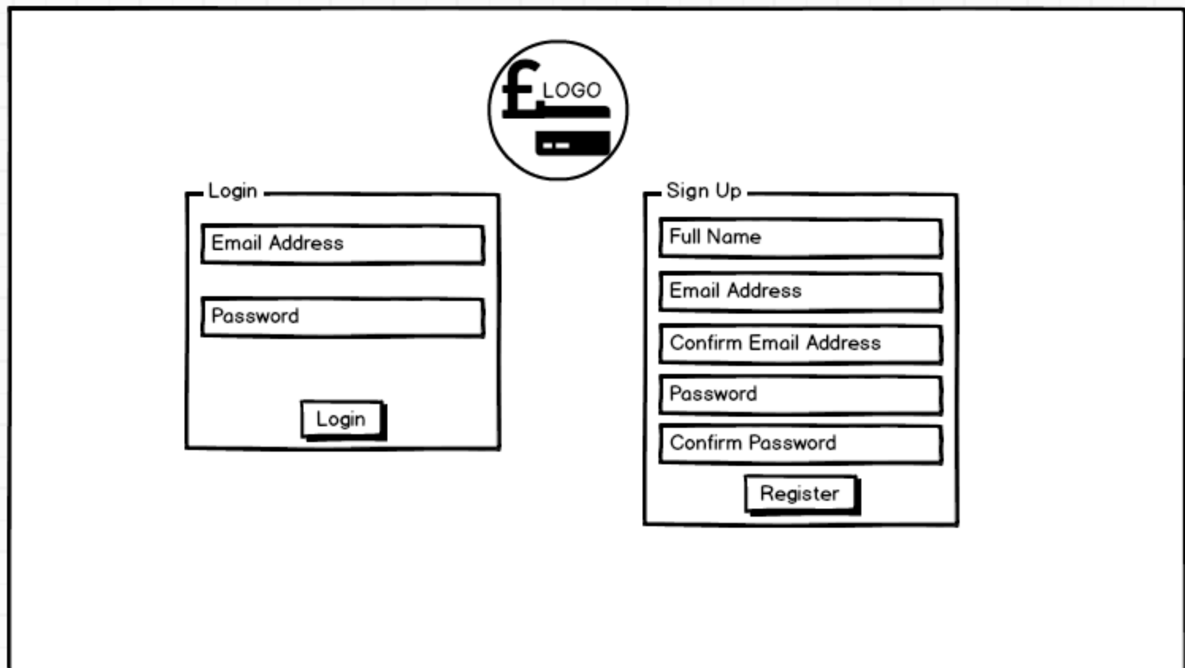


**Figure 7 - Original login and registration design**

**OVERVIEW**

Key points to note within Figures 8 and 9 are the navigation menu and its responsive design. On a mobile view the menu is collapsed into a dropdown, this can be seen in Figure 8. While the desktop view takes advantage of the extra screen space to expand into a fully visible navigation menu. Also shown in both figures is the in-line editing for existing transactions. Clicking the "gear" icon actives the row for editing and presents a further three options "delete", "submit" and "cancel". This will delete the entire row, submit any changes made or cancel the edit respectively. On the larger desktop version, an overview for every account can be displayed. On the mobile version the select account dropdown will update the pie chart accordingly, as will swiping left or right on the graph itself.

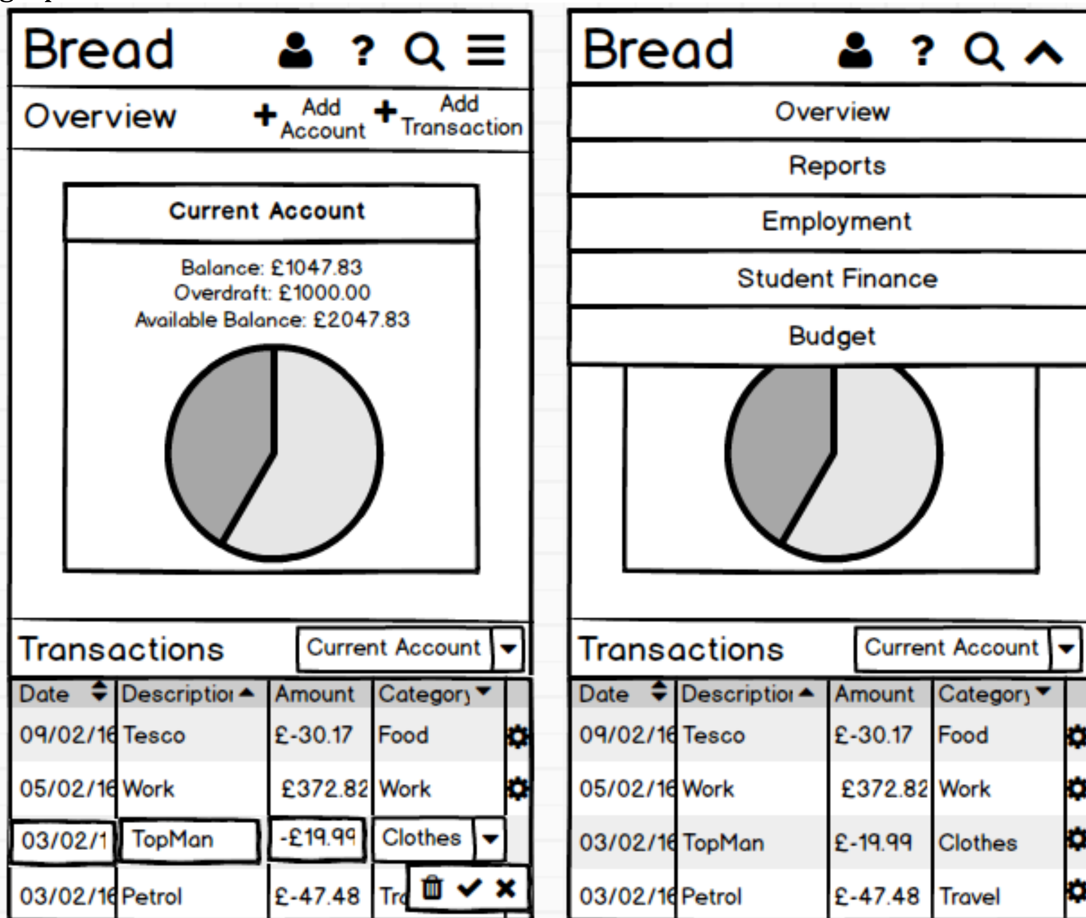


**Figure 8 - Mobile design for Overview page**

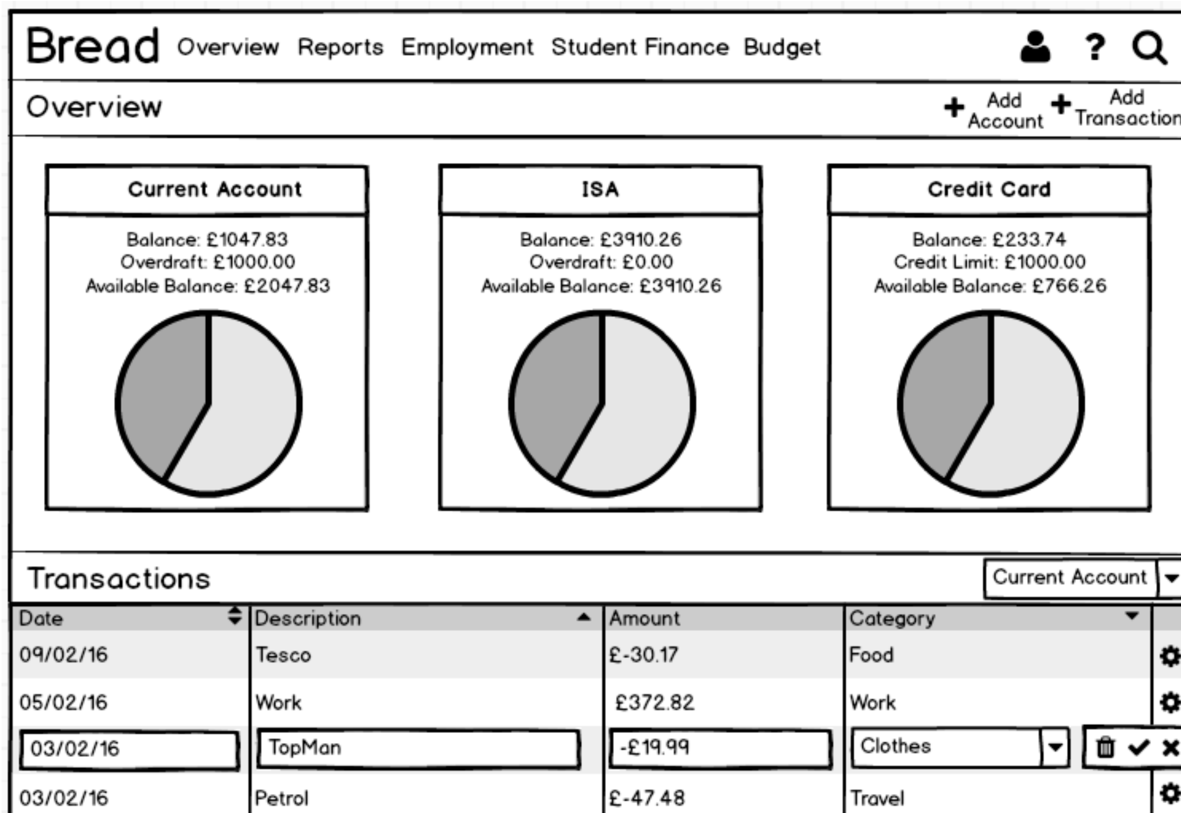


**Figure 9 - Desktop version of the Overview page**

The original design for the navigation and the overview page can be seen in Figure 10. You can see the initial thoughts to use sidebar navigation. While this would work well on both desktop and mobile platforms, it is not the most functional. On smaller devices the navigation would collapse in a similar manner to the method shown in Figure 8. This placement meant that the navigation would have to collapse into the top left corner and consequently push the logo over. This is not in keeping with the requirements of a clean and simple design as the main logo would be inconsistent across different devices. Due to this reason and because it consumes extra screen real estate, the navigation was moved to a top horizontal bar. The design in Figure 9 allows the core content of the application to fill the entire width of the display.

Within the original design the "Accounts" page was displaying the same content as the overview. So it makes sense to remove the page entirely. Any information required about an account or its transactions are available from either the Overview or Reports page in the designs shown in Figure 8 and 9. The original design also showed multiple tables being displayed at once. This was disregarded for two reasons, firstly as it is confusing for the user, imagine if there were 6 accounts with 6 tables showing their transactions, it would be a very messy and confusing page. Secondly you can only show a limited amount of transactions from each account using this method. Therefore, the logical decision was to switch to a single table where the user could select which account transactions to display.

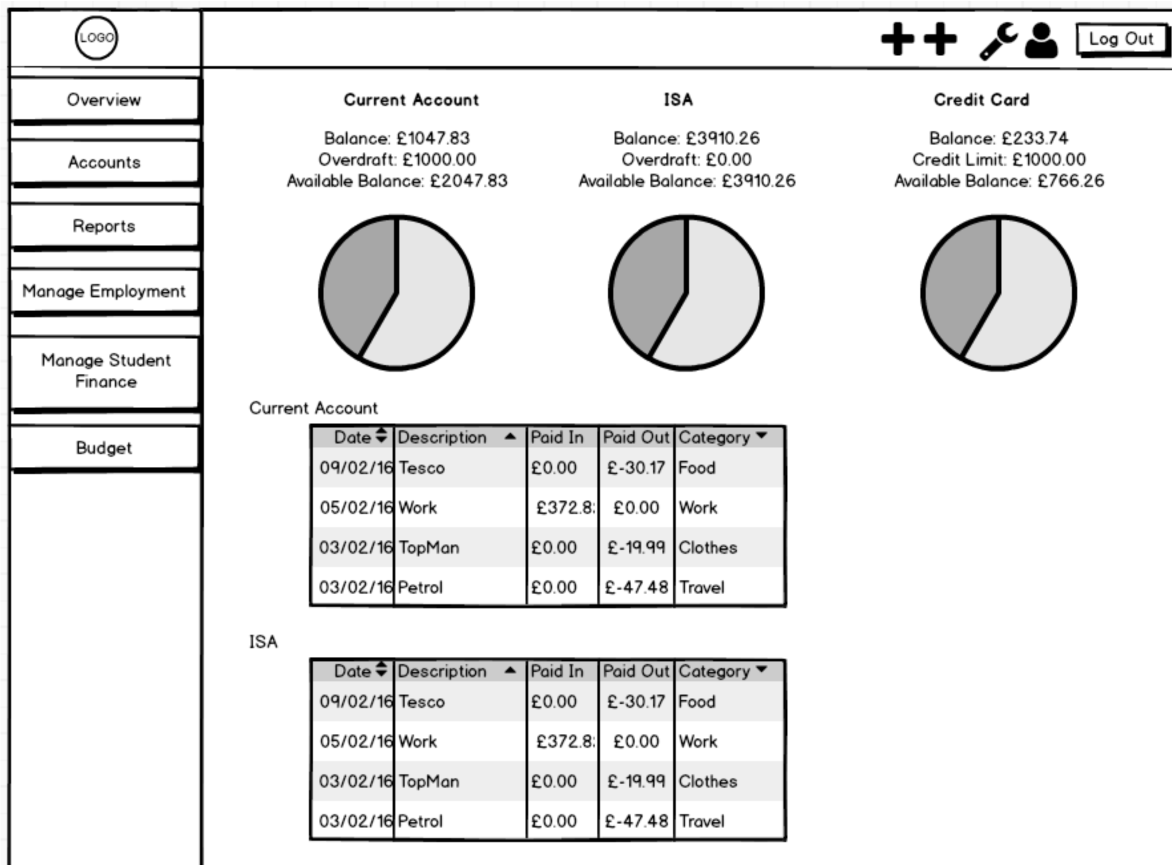


**Figure 10 - Original navigation and overview page designs**

## MODALS

From the overview and reports pages, the user is able to add an account and transactions to the system. This is achieved by clicking the appropriate button and following the steps within the pop up modal. Figure 11 shows the modal design for adding an account. Figures 12 through 15 shows modal designs for adding transactions. There are multiple methods available. Firstly adding individual transactions (Figure 12), adding multiple transactions (Figure 13) and uploading transactions through a CSV file (Figure 14). Figure 15 shows the second step of uploading transactions, the system reads the uploaded CSV file and presents the content to the user for verification and any alterations, before submitting to the database. The user must specify the account to upload the transactions to and also select a category for each transaction.

**Figure 11 - Modal design for adding an account**



**Figure 12 - Modal design for adding a single transaction**

Figure 13 - Modal design for adding multiple transactions

Figure 14 - Modal design for uploading a CSV file

VERIFY CSV CONTENT

Select an account to upload to

Select Account

Please select a category for each transaction

| | | | |
|---|---|---|---|
| 20/01/16 | Tescos | -£34.72 | Select Category |
| 19/01/16 | Student Loans | £ 840.24 | Select Category |
| 19/01/16 | Co-Op | -£13.18 | Select Category |
| 18/01/16 | National Rail | -£34.99 | Select Category |
| 17/01/16 | Tescos | -£10.82 | Select Category |

Upload Transactions

**Figure 15 - Modal design for verification of the CSV file**

## REPORTS

Similar to the Overview screen, only one graph is shown on the mobile view but swiping left or right will display the secondary bar chart for the income and expense comparison. The four selection dropdowns will update the pie chart and transaction list according to the value selected. Figure 16 shows the Report design.

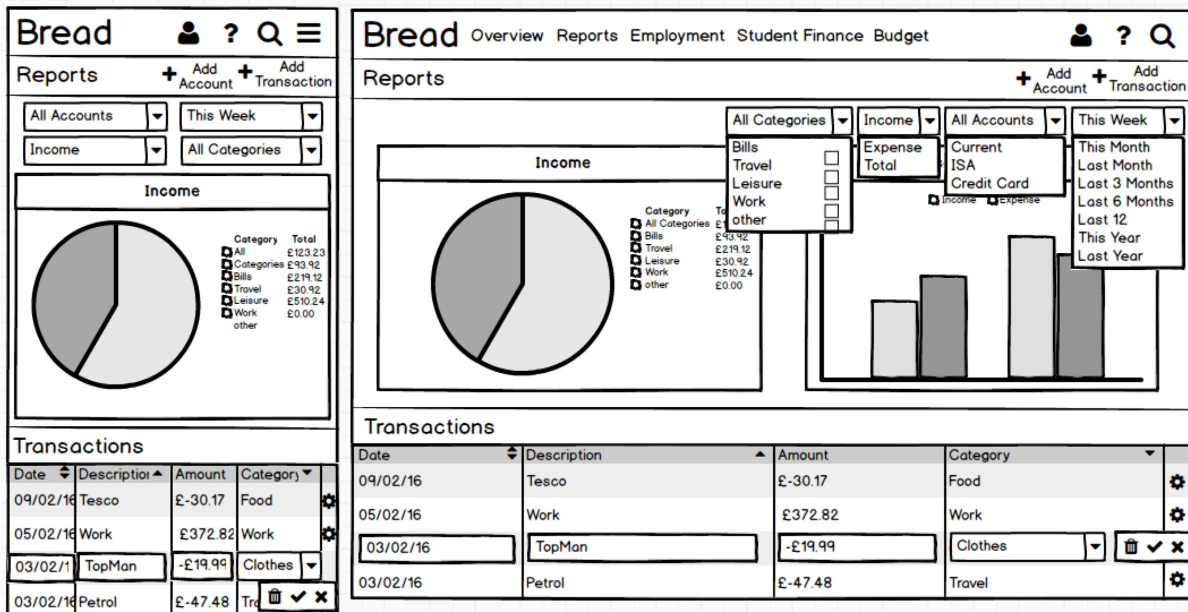


**Figure 16 - Design for the mobile and desktop Report screen**

## STUDENT FINANCE

Figure 17 shows the initial setup for a student loan account, this will be shown when the user first visits the page or when they do not have a loan account setup. In practice the Add instalments form would be disabled until the student loan set up form has been submitted. Only then can an instalment be added to the created loan. Therefore clicking “Create student loan” will disable the loan set up for and enable the add instalments

form, providing the request is successful. Similarly if clicking “Add instalments” is successful, both forms will be removed and the overview will be displayed. This is shown in Figure 18. An overview of the entire loan is displayed along with a summary of each academic year since the loan was created. The add instalments form will be available at all times, to allow the user to add more instalments.

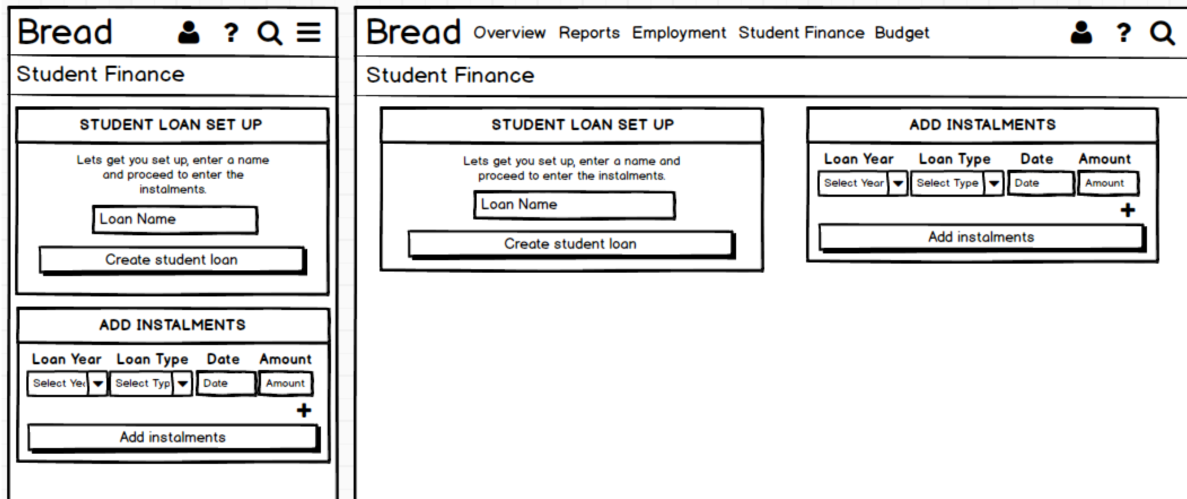


**Figure 17 - Design showing the mobile and desktop version of setting up the Student Finance page**

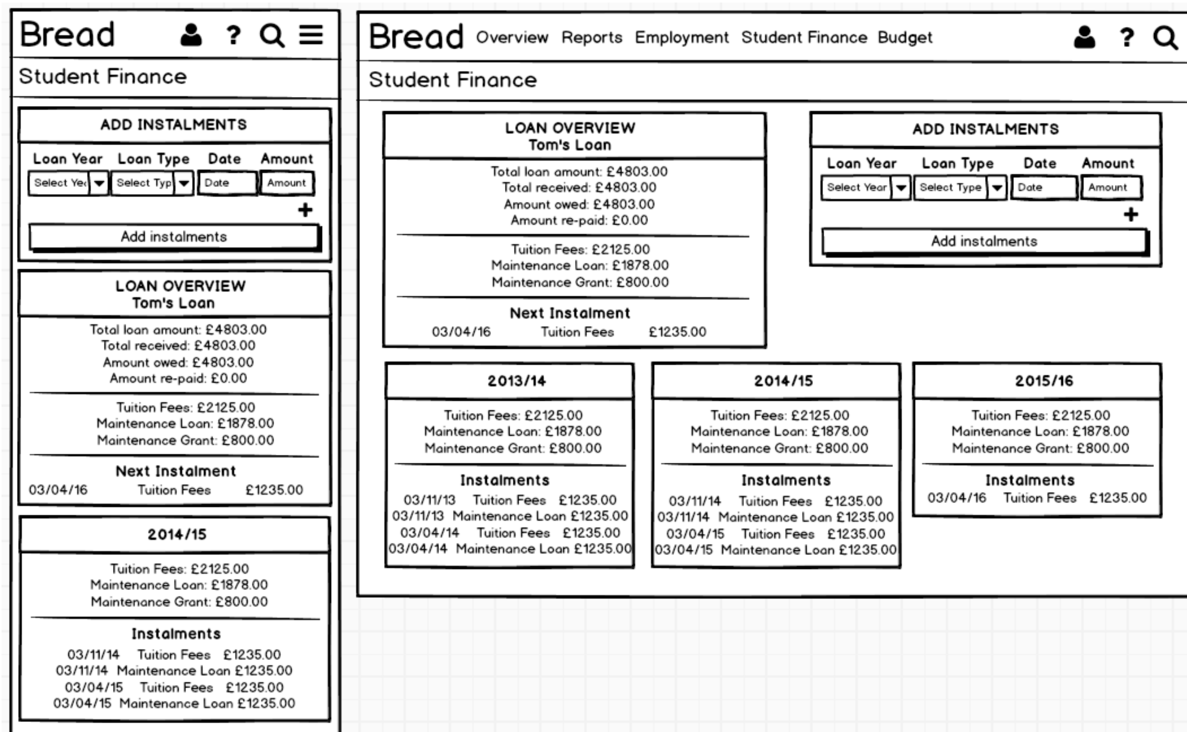


**Figure 18 - Design for the overview of the Student Finance page**

## EMPLOYMENT

Figure 19 represents the designs for the employment setup. The design functions in the same manner as the Student Finance setup. Figure 20 shows the Employment overview screen. A user can add multiple jobs and multiple payslips for each job. Upon adding a payslip the main totals (net pay, tax, NI and hours) are recorded but a user can break the details of the payslip down further. For example adding student loan deductions, bonuses or pensions to the description and amount fields. Each amount added must

sum to the net pay originally specified. Figure 20 shows these totals for the selected job and financial year, as well as showing each payslip that has been added. Expanding the payslip will reveal the extra details. Finally, any hours that are entered are shown to allow the user to confirm recorded hours with hours actually paid for.

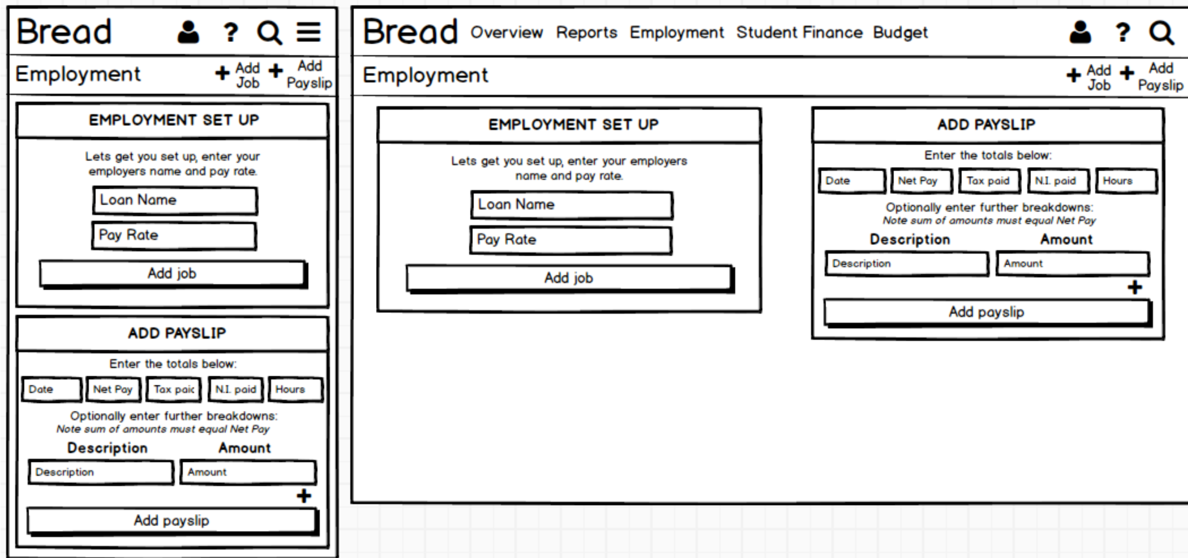


**Figure 19 - Employment setup designs**

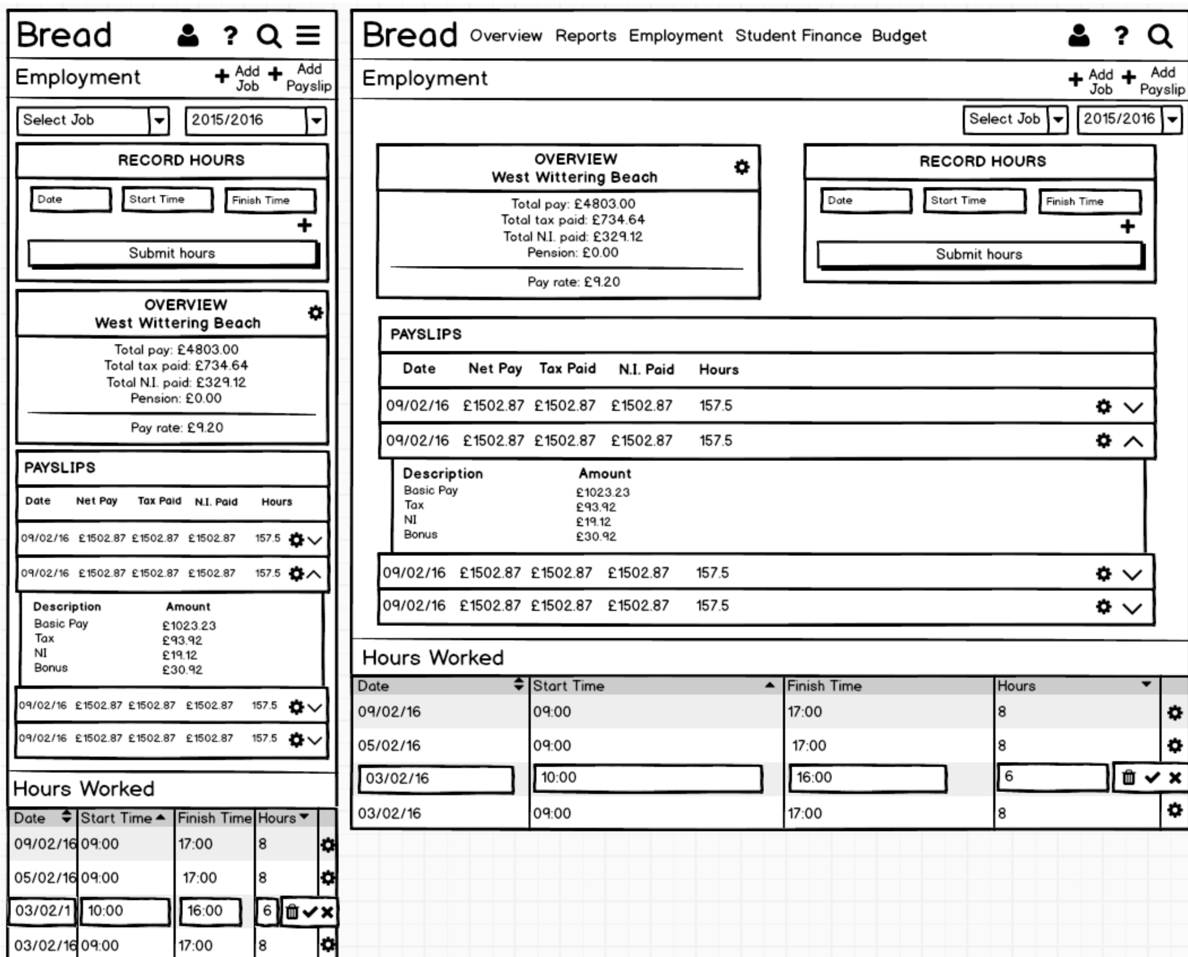


**Figure 20 - Employment overview designs**

## BUDGETS

Lastly Figures 21 and 22 show the designs for budgeting. The functionality is the same as previous pages. An initial setup to select the associated account, budget length and limit. These values are shown in the overview with a series of inputs to allow the user to add a budget to specific categories. The remaining budget is calculated from the transactions within the selected account, hence also displaying these transactions for easy access.

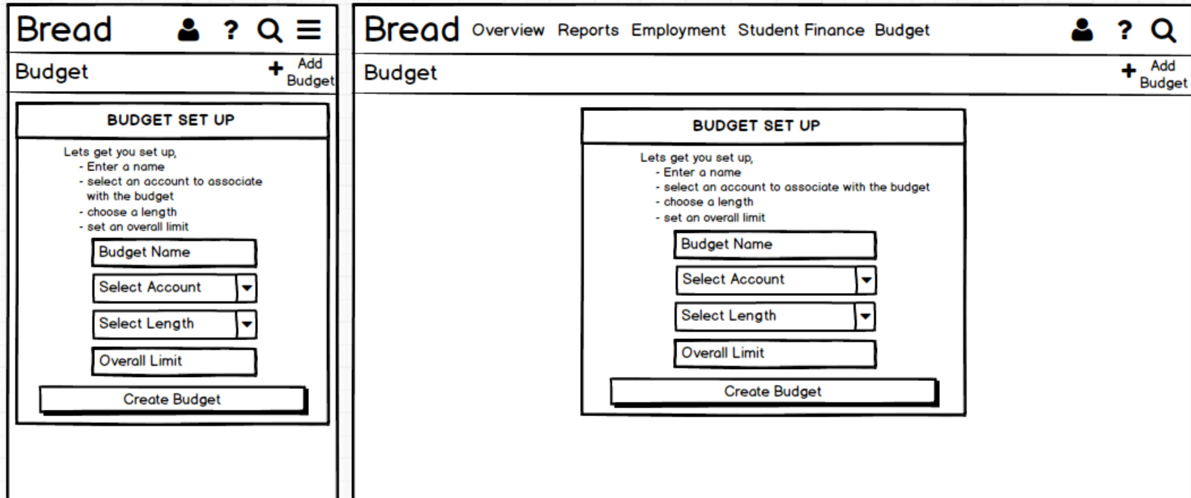


Figure 21 - Designs for the budget setup page

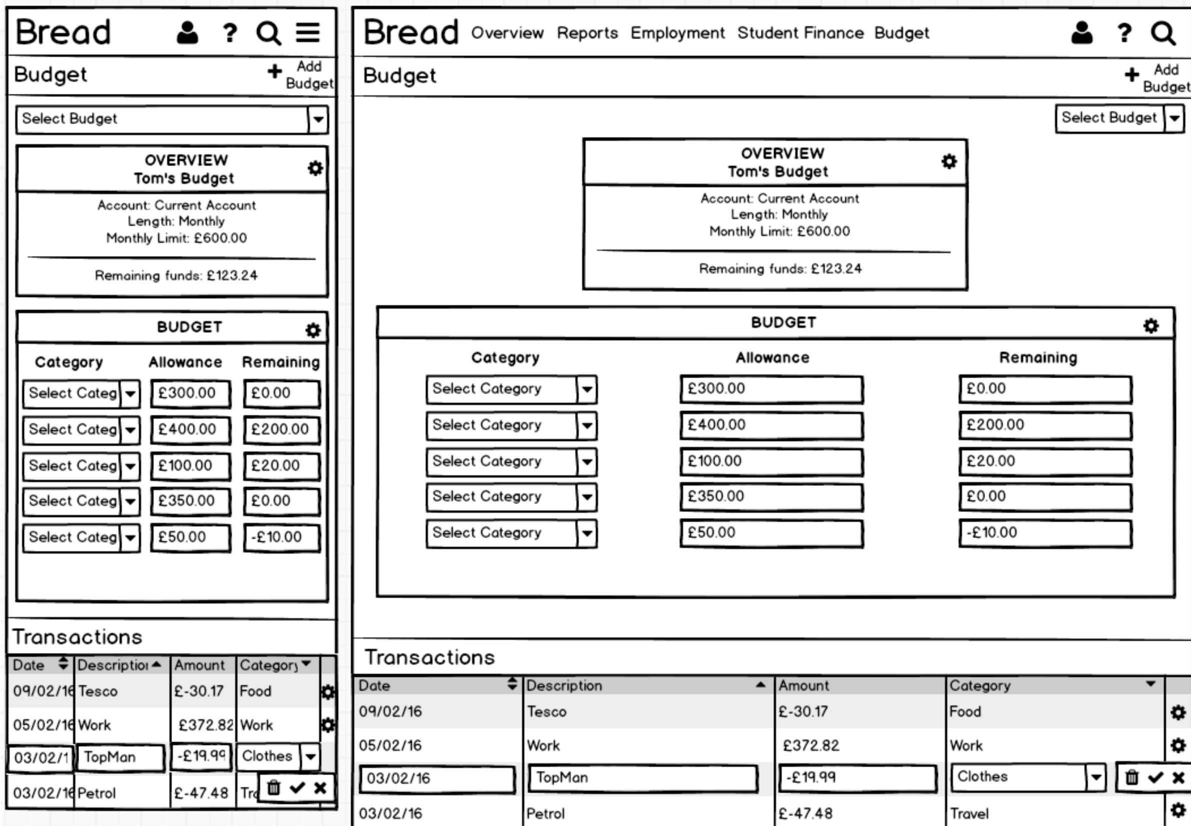


Figure 22 - Designs for the budget overview page

**GENERAL**

Figure 23 shows the FAQ's and the menu for user settings and sign out options. The other two icons to the right of the settings are for a FAQ section that will address general questions and answers on the majority of the features from each section. The search icon is provided to search for specific accounts or transactions with specific categories, however this will have to be a development for future work.

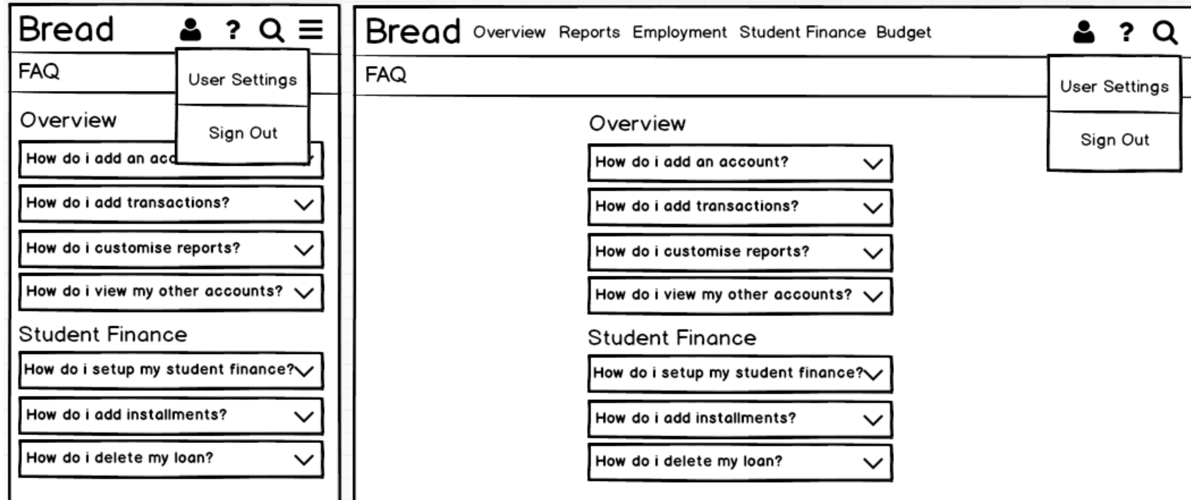


**Figure 23 - FAQ page design, also showing user settings menu**

### 3.3.3 FINALISING DESIGN DETAILS

The final designs that need to be completed are an overall colour scheme that can be used throughout the whole application, a definitive list of categories and an array of colours that will provide a wide range for the graphs and charts.

Rather than designing a colour scheme for each individual page, a master template demonstrated on one page will set a precedent for the rest of the system and will provide a good representation of the final product. Earlier in the report Figure 4 introduced the logo for the application and a couple of colours, a dark blue background with a bright green highlight for the pound symbol. These two colours provide a good contrast to be used throughout the system. The dark blue will function perfectly as a background for main headers, with various monochromatic shades of the blue for secondary headers. Again the green will provide a simple contrast for hover effects and button feedback, this will help create a simple and clean design. Figure 24 shows a mock up of the Overview page using these colours. This design was created using a visual design package called Sketch [11] and makes use of their predefined template sizes for both desktop and mobile devices. This has resulted in the layout being slightly altered to fit the smaller dimensions, however this will not affect the actual designs or implementation.

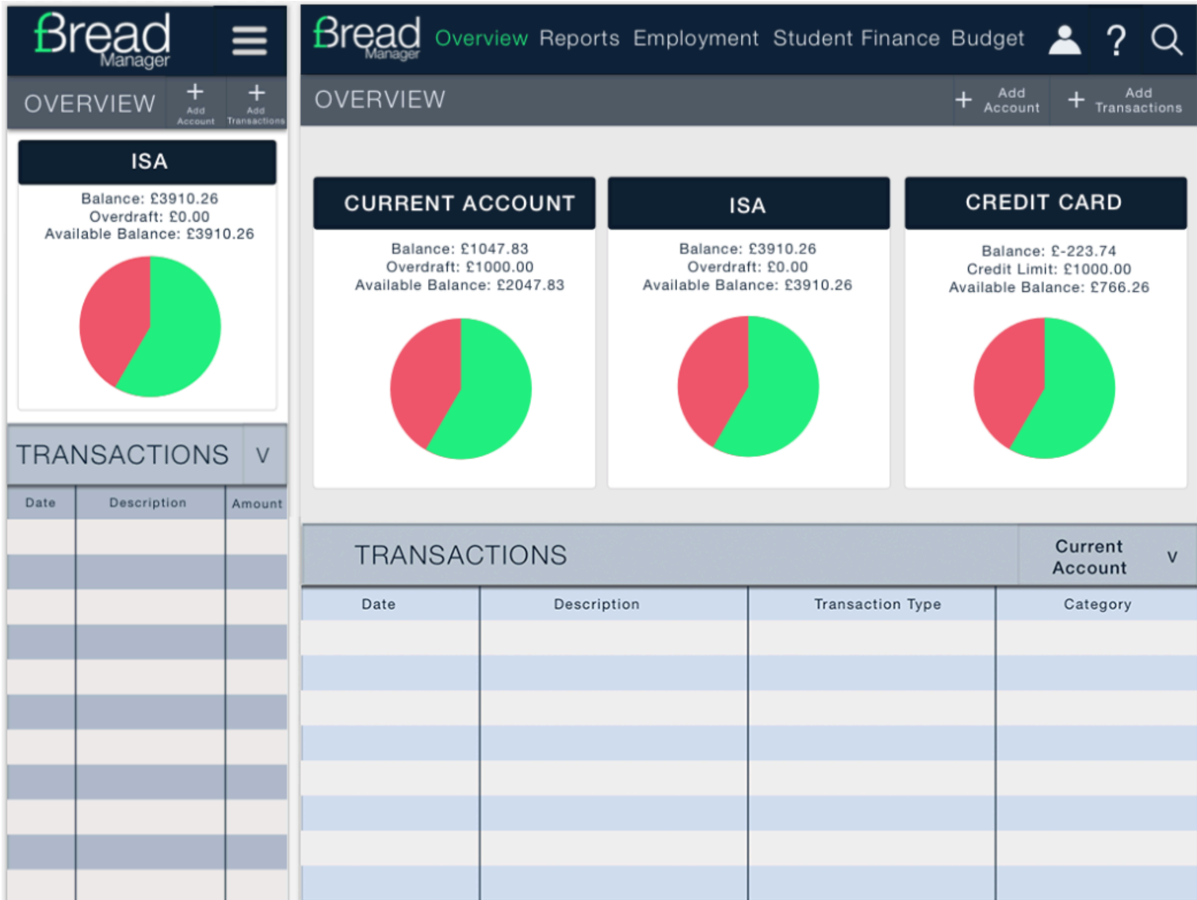


**Figure 24 - Mock up of Overview designs using colour template**

Table 1 includes a list of initial categories that will be available to users to assign to their transactions. The list is an attempt to cover a vast majority of categories a student would require. The list is likely to develop over the course of implementation and testing.

**Table 1 - Table listing possible categories, and sub categories**

| Category Name | Category Name |
|---|---|
| Bills<br>• Electric<br>• Gas<br>• Water<br>• Internet<br>• Phone Bill | Car<br>• Fuel<br>• Car Insurance<br>• Service<br>• MOT<br>• Road Tax<br>• Miscellaneous |
| Food<br>• Eating out<br>• Groceries<br>• Takeaway | Subscriptions<br>• Spotify<br>• ITunes<br>• Netflix |

| | |
|---|---|
| | • Amazon Prime |
| Education<br>• Books<br>• Course Costs<br>• Society | Money<br>• Cash Withdrawal<br>• Transfer<br>• Credit Card<br>• Parents |
| House<br>• Rent<br>• Council Tax<br>• Home Insurance<br>• Parking<br>• TV Licence | Leisure<br>• Night out<br>• Drinks<br>• Cinema<br>• Theatre<br>• Football<br>• Gym |
| Shopping<br>• Clothes<br>• Shoes<br>• Electronics<br>• Music<br>• Sport Gear<br>• Computing<br>• Books / Magazines | Student Finance<br>• Tuition Fee<br>• Maintenance Loan<br>• Maintenance Grant<br>• Tuition Fee Grant<br>• Course Grant<br>• Bursaries |
| Travel<br>• Train<br>• Bus<br>• Taxi<br>• Flight<br>• Boat | Work<br>• Basic Pay<br>• Bonus<br>• Commission<br>• Overtime<br>• Holiday Pay<br>• Sick Pay<br>• Other<br>• Tax<br>• NIC<br>• Pension<br>• Student Loan |
| Holiday<br>• Accommodation | |

- Travel
- Spending

Regarding the array of colours, it would be a difficult task to find a different colour for every category, especially to avoid colours that are indistinguishable from each other. Therefore having a different colour for each parent category with sub-categories being a different shade of its parent colour, would allow a wider range of colours. Table 2 shows the initial list of base colours and their hexadecimal value.

**Table 2 - Table of colours for graphs and charts**

| Colour | Hexadecimal |
|---|---|
| **Green** | #37EF86 |
| **Red** | #EE596C |
| **Yellow** | #FFEB00 |
| **Light Blue** | #76D7EA |
| **Light Purple** | #C9A0DC |
| **Orange** | #FF6037 |
| **Light Green** | #66FF66 |
| **Pink** | #FC80A5 |
| **Blue** | #0066FF |
| **Brown** | #87421F |
| **Purple** | #652DC1 |
| **Peach** | #FFCBA4 |
| **Grey** | #C8C8CD |
| **Light Yellow** | #FFFF9F |

### 3.3.4 TEST CASES

Testing will be completed during the implementation section, with the majority being conducted on the go during development. As a new feature is implemented, it will be tested accordingly to ensure its functionality. A final test will be conducted at the end to confirm it is still working as expected. The test cases defined here will be the basis for these final tests. They will cover the main requirements defined in section 3.2 while also assessing the functionality of the application across a variety of different browsers on a

variety of different operating systems. Not every test case will be defined here, as there will be in excess of 50 plus cases, hence the only the core tests will be defined.

| **Test 1:** | Can the user register an account. |
|---|---|
| **Environment:** | OS X |
| **Browser:** | Safari |
| **Pre-Conditions:** | On the Login page |
| **Process:** | 1. Select 'Sign Up' from the bottom of the login form.<br>2. User enters email address 'twhiddett@gmail.com' in the email field.<br>3. User enters username 'twhiddett' in the username field.<br>4. User enters password 'password' in the password field.<br>5. User re-enters password 'password' in confirm password field.<br>6. Select the 'Sign Up' button. |
| **Expected Outcome:** | The entered details should be stored within the 'Users' table in the database. The details will be used to login. |
| **Actual Outcome:** | Pass / Fail |

| **Test 2:** | Can the user log into the application |
|---|---|
| **Environment:** | OS X |
| **Browser:** | Safari |
| **Pre-Conditions:** | User must be registered and on the login page. |
| **Process:** | 1. Input the username 'twiddett' in the username field.<br>2. Input the password 'password' in the password field.<br>3. Select 'Sign In' button. |
| **Expected Outcome:** | The user is redirected to the home page (Overview page). |
| **Actual Outcome:** | Pass / Fail |

| **Test 3:** | Can the user create an account. |
|---|---|
| **Environment:** | OS X |

| | |
|---|---|
| **Browser:** | Safari |
| **Pre-Conditions:** | User must be logged in and on the overview page |
| **Process:** | 1. Select 'Add Account' button just under the navigation bar.<br>2. Input the account name 'ISA' in the 'Account Name' field<br>3. Input the start date '20/04/16' in the 'Start Date' field.<br>4. Input the opening balance '£1234.56' in the 'Opening Balance' field.<br>5. Select 'Add Account' button. |
| **Expected Outcome:** | The account details are saved in the accounts table.<br>A confirmation message is received telling the user the account was successfully added.<br>The account is displayed in the overview page. |
| **Actual Outcome:** | Pass / Fail |

| **Test 4:** | Can the user upload transactions to an account. |
|---|---|
| **Environment:** | OS X |
| **Browser:** | Safari |
| **Pre-Conditions:** | User must have created an account and be on the overview page. |
| **Process:** | 1. Select 'Add Transactions'.<br>2. Select 'Upload.<br>3. Select 'Choose File'.<br>4. User selects a CSV file on their system.<br>5. Select 'Upload Transactions'.<br>6. 'Verify CSV Content' Modal is displayed with the each transaction displayed in a row of inputs.<br>7. Select account 'ISA' from the 'select account' dropdown.<br>8. Select a category for each transaction.<br>9. Select 'Upload Transactions'. |
| **Expected Outcome:** | The CSV file is verified and uploaded to the transactions table.<br>A confirmation message is displayed to confirm upload success.<br>The uploaded transactions are visible in the 'Transactions' section of the overview page. |

| Actual Outcome: | Pass / Fail |
|---|---|

| Test 5: | The user can customise the display of different reports. |
|---|---|
| **Environment:** | OS X |
| **Browser:** | Safari |
| **Pre-Conditions:** | An account must have been created with transactions added.<br>Must be on reports page. |
| **Process:** | 1. Select 'Income' from transaction type selector.<br>2. Select 'ISA' from account selector.<br>3. Select 'Last 6 Months' from date range selector.<br>4. Select 'All categories' from category selector. |
| **Expected Outcome:** | The pi chart will display the spread of transactions by the total of each category.<br>The bar chart will display income vs. expense values for each month within the last 6 months.<br>The 'Transactions' section will be populated with the transactions that meet the defined criteria. |
| **Actual Outcome:** | Pass / Fail |

| Test 6: | Can the user edit individual transactions. |
|---|---|
| **Environment:** | OS X |
| **Browser:** | Safari |
| **Pre-Conditions:** | An account must have been created and populated with transactions.<br>Be on the Overview or Reports page. |
| **Process:** | 1. Selects the gear icon next to the transaction to be edited.<br>2. Change the description to 'Co-Op food shop'.<br>3. Select the category 'Groceries' from the category selector.<br>4. Click the tick icon to submit the change. |
| **Expected Outcome:** | The description of the selected transaction now read 'Co-Op food shop'.<br>The category for the selected transaction is not 'Groceries'. |

| | |
|---|---|
| | The values are changed in the transactions table in the database.<br>The updated transaction is displayed within the 'Transactions' section. |
| **Actual Outcome:** | Pass / Fail |

| | |
|---|---|
| **Test 7:** | Can the user delete a transaction. |
| **Environment:** | OS X |
| **Browser:** | Safari |
| **Pre-Conditions:** | An account must have been created and populated with transactions.<br>Be on the Overview or Reports page. |
| **Process:** | 1. Select the gear icon next to the transaction to be deleted.<br>2. Select the trash icon.<br>3. Select 'Confirm delete' from the confirmation pop up. |
| **Expected Outcome:** | The select transaction has been deleted from the transactions table.<br>The selected transaction is no longer visible in the 'Transactions' section on the page. |
| **Actual Outcome:** | Pass / Fail |

| | |
|---|---|
| **Test 8:** | Test whether the user can create a student loan |
| **Environment:** | OS X |
| **Browser:** | Safari |
| **Pre-Conditions:** | Be on the overview page. |
| **Process:** | 1. Select 'Student Finance' from the navigation menu.<br>2. Input loan name 'Tom's Loan' in the loan name field.<br>3. Select 'Create Student Loan' button.<br>4. Select '2015/2016' from loan period selector.<br>5. Select 'Maintenance Loan' from the loan type selector.<br>6. Input date '28/04/16' in the date field.<br>7. Input '1836.84' in the amount field.<br>8. Select the plus icon. |

| | |
|---|---|
| | 9. Select '2015/2016' from the loan period selector.<br>10. Select 'Tuition Fee' from the loan type selector.<br>11. Input date '28/04/16' in the date field.<br>12. Input '1528.28' in the amount field.<br>13. Select 'Add instalments' button. |
| **Expected Outcome:** | A loan called 'Tom's loan is created in the loans table.<br>The two instalments are stored in the instalments table.<br>The 'Student Finance' page now displays the overview of 'Tom's loan'.<br>The instalments are displayed. |
| **Actual Outcome:** | Pass / Fail |

| | |
|---|---|
| **Test 9:** | Test whether the user can create a budget. |
| **Environment:** | OS X |
| **Browser:** | Safari |
| **Pre-Conditions:** | Be on the overview page and has created an account. |
| **Process:** | 1. Select 'Budget' from the navigation menu.<br>2. Input budget name 'university budget' in the loan name field.<br>3. Select account 'ISA' from the accounts selector.<br>4. Select length 'monthly' from the length selector.<br>5. Input '600.00' in the overall limit field.<br>6. Select 'Create budget' button.<br>7. Select the category 'rent' from the category selector.<br>8. Input '380.00' in the allowance field.<br>9. Select the category 'bills' from the category selector.<br>10. Input '50.00' in the allowance field.<br>11. Select the category 'food' from the category selector.<br>12. Input '120.00' in the allowance field.<br>13. Select the category 'travel' from the category selector.<br>14. Input '50.00' in the allowance field. |
| **Expected Outcome:** | A budget called 'university budget' is stored in the budgets table.<br>The budgets page displays budget overview and the 4 categories |

| | |
|---|---|
| | budgeted for.<br>The transactions section will display the transactions for the current month.<br>The remaining funds are shown. |
| **Actual Outcome:** | Pass / Fail |

| | |
|---|---|
| **Test 10:** | Can the user log out. |
| **Environment:** | OS X |
| **Browser:** | Safari |
| **Pre-Conditions:** | Must be logged in. |
| **Process:** | 1. Select the user icon.<br>2. Select 'Log out' button. |
| **Expected Outcome:** | The user is logged out and redirected to the Login page. |
| **Actual Outcome:** | Pass / Fail |

| | |
|---|---|
| **Test 11:** | Does the application display correctly in Google Chrome. |
| **Environment:** | OS X |
| **Browser:** | Google Chrome |
| **Pre-Conditions:** | - |
| **Process:** | 1. Open the application in Google Chrome.<br>2. Login.<br>3. Select 'Reports' page from the navigation menu. |
| **Expected Outcome:** | The display is consistent between browsers. |
| **Actual Outcome:** | Pass / Fail |

# 4 IMPLEMENTATION

This section will follow the steps taken to produce a final implementation based on the designs from previous sections. The first steps were to set up a development environment. MAMP (which stands for Macintosh, Apache, MySQL and PHP) is a piece of software that runs a local server environment on your computer [12]. This allows server side languages such as PHP to be processed as well as connecting to MySQL databases. This is achieved through an Apache and MySQL server that MAMP provides, which allows for development and testing as if the application is hosted online. Sequel Pro or phpMyAdmin can be used with MAMP to create and manage the database. Initially phpMyAdmin was used but was later swapped out for Sequel Pro, a free desktop application for database management. This decision was made after a weeks work experience at a web development agency and the reasoning is purely a personal preference over the features that Sequel Pro offers and its superior user interface.

Using PHP to access a MySQL database means using one of three API's, 'mysql', 'mysqli' or 'PDO' (PHP Data Objects). Upon starting development the 'mysql' extension was used without realising that it has been depreciated in PHP versions 5 and above, and removed in PHP versions 7 and above. Unfortunately this was not realised until half way through development. This presented two options, to change every instance of the depreciated extension to either 'mysqli' or 'PDO'. During the work experience mentioned above, 'PDO' was introduced to me and was practiced during a project over that week. The main advantages of 'PDO' over 'mysqli' is its support for various other database drivers, not just MySQL, and the ability to name and bind parameters and use prepared statements. The initial database setup using 'mysql' was not object oriented and so there was a lot of duplicated code. While 'mysqli' does support both procedural and OOP, the decision to use 'PDO' was taken based on the advantages listed above. This did result in having to completely re-write the database setup, configuration and how functions interact with the database, but it was a necessary step towards a cleaner and improved application. The affects meant not fully completing every feature and a less than rigorous testing phase, but this will be discussed in section 5.0. The online manual for 'PDO' was extremely helpful in processing this change [13].

As mentioned in the background, Sass is being used as a pre-processor for CSS. It extends the original CSS to make use of helpful features. Some of these include variables, imports and mixins. Variables allow for global variables to be set and accessed through a variable name. This proves useful when defining fonts or colours, as only one line of code needs to be altered if these are changed. Imports keep the code structured and organised by having separate files for resets, variables, etc. and importing them into the main code. Mixins help reduce repeated code by defining set rules for styles such as borders and shadows, which can be called within the main file. All of these features help to produce CSS efficiently and once Sass is installed it can be setup to watch a Sass file and automatically compile it into a useable CSS file. The online documentation [14] and Sass for Web Designers book [15] were used as reference for the correct setup and usage.

Parts of the designs indicate various icons to represent the user options or the navigation dropdown on smaller devices. Font Awesome has been used as it provides a wide range of scalable, customisable icons that can be used throughout the application

[16]. Browsers have their own default styles for HTML, so to help avoid any conflicts or issues across different browsers, a reset stylesheet is going to be implemented. Eric Meyer's reset [17] is used during the project and is also recommended by Ethan in the Responsive Web Design book [10]. Google Fonts is being used to provide the Open Sans font. While this does mean the font files will have to be downloaded to the users computer when the page loads, the difference in response time should be negligible. You can select the range of styles to be included, so by only selecting the styles that will be used will reduce the load time. The Font is also cached in the browser after it is first downloaded which will also reduce the page load time.

As the application will be storing potentially sensitive information about the users financial transactions, it is essential security measures are taken to ensure the application is secure. This involves providing adequate authentication during the login process. A user should not be able to access the application without logging in. The implementation of this will be discussed in section 4.2. Secure connection between the web server and client should also be employed. An SSL (Secure Socket Layers) certificate is used for this purpose, using a public / private key pair, communications sent between the server and the client will be encrypted and signed by the SSL certificate. This would allow the application to be accessed through HTTPS. Self-signed certificates could be set up for development on the local environment, but has not been implemented.

Finally JavaScript and JQuery will be used for client side validation, interactive content and Ajax to request data from the server and update the page without reloading. The asynchronous requests will provide a more interactive experience on the application, especially when customising the transactions to be displayed for reports. Similar to Google Fonts, JQuery and JQuery UI is being served through Google API's. JQuery UI is only being used for the calendar dropdown feature for date entry. "JavaScript & JQuery – Interactive Front-End Web Development" by Jon Duckett [18] and the JQuery online documentation [19] provided extremely helpful reference during the development. The book also provided the template for a pop out modal.

Now that the development environment has been setup, the report will continue to explain key sections of the implementation process and any problems encountered. The full source code is provided separately but snippets of key areas will be included in this report.

## 4.1 GENERAL STRUCTURE

This section will describe the core structure of the application, which consists of the database, php template and error reporting. The database section will discuss how it has been implemented based on the design in section 3.3.1. The template combines common code, such as the navigation, into one file so that it can be included within each subsequent page. Both will be explained in further detail below.

Figure 25 shows the file directory structure. The root will contain files for each page of the application. 'Stylesheets' contains a 'Sass' subdirectory where the Sass code will be written, this is then complied into a CSS file and stored within the 'stylesheets' directory. The 'js' directory contains any JavaScript or jQuery files that will be used and

the 'fonts' folder contains the Font Awesome source code. The template files will be kept within the 'includes' directory, each of these files will be included by a core initiation file. This is part of the 'core' directory and will be explained in more detail during section 4.1.2. The 'core' folder also consists of four subdirectories, 'ajax' for any server code to be executed using ajax, 'database' for the database configuration, 'functions' which will contain general functions to be used throughout the application, and finally 'classes' that contains the object oriented classes used in the application. This file structure is key to keeping the code organised.

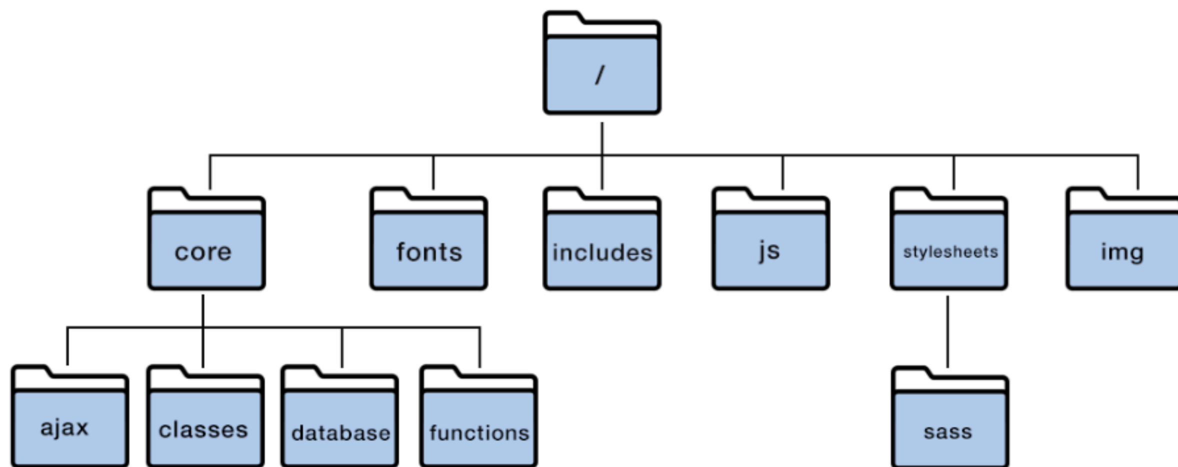


**Figure 25 - File directory structure**

### 4.1.1 DATABASE SETUP

A separate configuration file is used to define the database connection details. Using the PHP function 'define()' will create a constant variable that can not be changed once set but can be accessed throughout the application. This has been done for the database name, username, password and hostname. The following code is from the 'config.php' file.

```
// name of the database
define('DB_NAME', 'bread');
// database username
define('DB_USER', 'root');
// database password
define('DB_PASSWORD', 'root');
// hostname
define('DB_HOST', '127.0.0.1');
```

Throughout the application, multiple requests to the database are going to be made for entering or retrieving data. To save having to duplicate code each time access to the database is required, a database class has been created. The singleton design pattern is used here to ensure one and only one instance of the database is available. When the database class is first called and constructed, the database connection is saved to a private 'instance' variable. In this way when the database is called again, the instance of the connection will be used. This means that the application does not have to keep opening a new connection to the database for each page. The class also contains various methods that can be called to access the database. These methods can be seen in the

following code, note the actual content of the methods are not shown as it is available within the source code.

```
class Database {
private static $_instance = null;
private $_dbh, $_query, $_results, $_error = false, $_count = 0;

private function __construct() {
        try {
                $path = $_SERVER['DOCUMENT_ROOT'];
                include "{$path}/core/database/config.php";
                $this->_dbh = new PDO('mysql:host=' . DB_HOST . ';dbname='
. DB_NAME, DB_USER, DB_PASSWORD, array(PDO::MYSQL_ATTR_INIT_COMMAND => "SET
NAMES 'utf8'"));
                $this->_dbh->setAttribute(
PDO::MYSQL_ATTR_USE_BUFFERED_QUERY, true );
                $this->_dbh->setAttribute( PDO::ATTR_ERRMODE,
PDO::ERRMODE_EXCEPTION );
        } catch(PDOException $e) {
                die($e->getMessage());
        }
}

public static function getInstance() {
        if(!isset(self::$_instance)) {
                self::$_instance = new Database();
        }
        return self::$_instance;
}

public function query($sql, $params = array()) {}
public function queryAction($sql, $values = array()) {}
public function advancedAction($action, $columns, $table, $where, $extras =
'') {}
public function action($action, $columns, $table, $where = array()) {}
public function get($columns, $table, $where = array()) {}
public function advancedGet($columns, $table, $where, $extras = '') {}
public function insert($table, $fields = array()) {}
public function update($table, $fields, $where, $id) {}
public function results() {}
public function first() {}
public function error() {}
public function count() {}
}
```

The construct method uses the constant variables defined in the configuration file to connect to the database. The following code represents how the database class is called from within another class, for example the Users class.

```
$this->_db = Database::getInstance();
```

The Users class calls the ‘getInstance()’ method to store an instance of the database connection so that it can be used within this class.  If an instance does not exist then a new connection is created. The other methods shown retrieve, store and delete data within the database. For example to create a user from within the Users class, the following code is used.

```
$this->_db->insert('users', $fields)
```

The database handler ‘_db’ that holds the instance of the database is used to call the insert method. The two arguments passed through are the table name to insert data to and an array of key-value pairs for the data to be entered into the associated field. Within the insert method the fields array is split into keys and values to create an SQL statement that is then passed to the query method along with the fields array.

```
public function insert($table, $fields = array()) {
        $keys = array_keys($fields);
        $values = '';
        $x = 1;
        foreach ($fields as $field) {
                $values .= '?';
                if($x < count($fields)) {
                        $values .= ', ';
                }
                $x++;
        }
        $sql = "INSERT INTO $table (`" . implode('`, `', $keys) . "`)
VALUES ({$values})";
        if(!$this->query($sql, $fields)->error()) {
                return true;
        }
        return false;
}
```

The query method uses the PDO prepare function to prepare the SQL statement and then binds each value within the fields array to the SQL statement. This process is very important as it helps prevent SQL injections. By preparing the SQL statement and then binding the parameters separately it is impossible for SQL injection to occur.

```
public function query($sql, $params = array()) {
        $this->_error = false;
        if($this->_query = $this->_dbh->prepare($sql)) {
                $x = 1;
                if(count($params)) {
                        foreach ($params as $param) {
                                $this->_query->bindValue($x, $param);
                                $x++;
                        }
                }
                if(!$this->_query->execute()) {
```

```
                        $this->_error = true;
                }
        }
        return $this;
}
```

This process is the basic functionality for every method within the Database class. However the 'advancedGet', 'advancedAction' and 'queryAction' were added further into the implementation to accommodate for complex SQL statements that are required for the reports.

### 4.1.2 TEMPLATES

The main reasoning for setting up a template system for the application is to reduce repeated code. The header, footer, navigation and initiation file will be consistent across every web page so it does not make sense to repeat this. PHP include and require statements can be used to accomplish this. The opening 'html' and 'body' tags as well as the 'head' tag are placed within a separate 'header.php' file. The same is done for the closing 'html' and 'body' tags, navigation and modal html content. They are separated into 'footer.php', 'nav.php' and 'modal.php' respectively. These are then included on each page using the following code:

```
<?php
$path = $_SERVER['DOCUMENT_ROOT'];

require_once "{$path}/core/init.php";

$user = new User();
if(!$user->isLoggedIn()) {
        Redirect::to('login.php');
}

$categories = $user->getCategories();
$accounts = $user->getAccounts();

include "{$path}/includes/overall/header.php";
include "{$path}/includes/nav.php"; ?>
```

'require_once' has been used for the initiation file as it contains code that is crucial to the application and subsequent inclusions of the same code would throw and error. An instance of the User class is also created on each page. The constructor for the user class will attempt to find the user using a session that stores the user id, if this is successful then a 'isLoggedIn' flag is set to true. This session is set when the user logs into the application. So if the user has not logged in then no session will be set and the logged in flag will be set to false. As part of the authentication process, every page of the application will perform a check to see if the user is logged in. If this fails then the user will be redirect to the login page. This prevents unauthorised access to the application. The 'init.php' file is used to initialise the entire system. Sessions are started and the class files are loaded. The following code demonstrates this:

```
<?php
```

```php
session_start();
date_default_timezone_set('Europe/London');
$path = $_SERVER['DOCUMENT_ROOT'];
#Autoloader with annonymous function
spl_autoload_register(function($class) {
        $path = $_SERVER['DOCUMENT_ROOT'];
        require_once "{$path}/classes/{$class}.php";
});
require_once "{$path}/functions/sanitise.php";
```

The PHP 'spl_autoload_register' has been used to load the class as and when they are declared. PHP will put each class into a queue and only call the class when its explicitly declared. For example, when a new instance of the User class is declared as follows:

```php
$user = new User();
```

The autoloader will load the file with the class name 'User.php'. This effectively removes the need to include redundant classes that are not used on a specific page. The default time zone is also set within the 'init.php' file as the 'datetime' object is used for part of the report customisation settings. This will be explained further into the report. The last file included within 'init.php' is the sanitise function, seen below.

```php
function escape($string) {
        return htmlentities($string, ENT_QUOTES, 'UTF-8');
};
```

Sanitising and escaping is a very important concept as it prevents SQL injection and cross-site scripting attacts (XSS). Sanitising is done during data input, through PDO prepared statements and binding, and escaping is done when outputting data. Just sanitising on input is not enough, if the output is not escaped the user can inject HTML to the URL which in turn can load scripts and cause various problems. The 'htmlentities()' function converts characters to HTML entities, the second argument specifies that double quotes should also be escaped, while the third argument specifies the encoding, 'UTF-8' is used throughout the application and database. This escape function is used within the application and 'json_encode()' is used to escape JavaScript.

### 4.1.3 ERROR REPORTING

As part of the development and testing process the following code was used to ensure all errors would be displayed.

```php
#temp error settings to display errors for testing and debugging
ini_set('display_errors', 1);
ini_set('display_startup_errors', 1);
error_reporting(E_ALL);
```

As part of the application, success and error messages will be set when submitting forms. These messages are stored in a session and then displayed appropriately after form submissions. The following code shows the 'success' and 'error' sessions being initialised and then output process.

```
if( !isset($_SESSION['success'])){
        $_SESSION['success'] = array();
}
if( !isset($_SESSION['errors'])){
        $_SESSION['errors'] = array();
}
```

```
<section id="message-alert">
<?php
if(Session::exists('success')) {
        foreach(Session::get('success') as $key => $msg){
                echo '<div class="alert success"><p>' .
Session::flash('success', $key) . '</p><a href="#" class="alert-
close"><span class="fa fa-close fa-fw"></span></a></div>';
        }
}

if(Session::exists('errors')) {
        foreach(Session::get('errors') as $key => $msg){
                echo '<div class="alert error"><p>' .
Session::flash('errors', $key) . '</p><a href="#" class="alert-close"><span
class="fa fa-close fa-fw"></span></a></div>';
        }
}
?>
</section>
```

Figure 26 shows an example of an error message displayed after a failed login attempt.

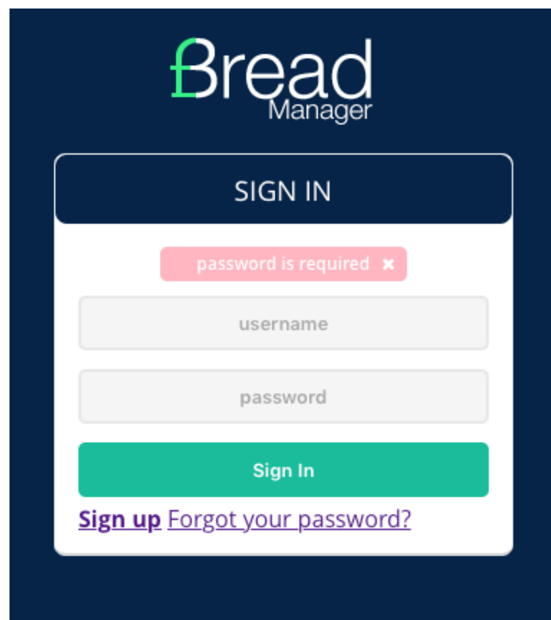


**Figure 26 - An example of the error message displayed after a failed login attempt**

Figure 27 - Shows the registration form

## 4.2 LOGIN AND REGISTER

The login and registration process is key to the security of the application. It is important that only authenticated users can access the system. As mentioned in section 4.1.2, each page checks to see if the user is logged in and will redirect back to the login page if not. However this is not the only security concern. The users password must be stored and retrieved securely without knowing the actual password. In the past this could be achieved by creating a salt and using a hashing function, such as md5, both the hash and salt would be stored in the database so that login could be verified. However this will pose various security problems, such as using the same salt and attacks to intercept the salt such as XSS or SQL injection. PHP now provides 'password_hash()' function, this will take the password string as input and encrypt with a randomly generated salt for each password, providing an encrypted string up to 255 characters long. This will prevent the same passwords having the same hash. It uses the bcrypt algorithm that is changed over time as newer and stronger algorithms are added to PHP, providing some future proofing. The code below shows this being implemented.

```
$user->create(array(
        'username' => Input::get('username'),
        'password' => password_hash(Input::get('password'),
PASSWORD_DEFAULT),
        'email' => Input::get('email')
));

Session::add('success', 'You have been registered and can now login');
Redirect::to('index.php');
```

The counterpart to this is 'password_verify()', which simply takes two arguments, the password to verify and the hash to compare with. The hash from the 'password_hash()' function contains all the information needed for 'password_verify()' to verify a

password. The function returns true or false depending if the password match. This code is used to verify and login the user:

```
public function login($username = null, $password = null) {
#finds username, if it does not exist then returns false
$user = $this->find($username);
if($user) {
        #checks password and stores user_id as _Session 'user_id'
        if(password_verify($password, $this->data()->password)) {
                Session::put($this->_sessionName, $this->data()->user_id);
                return true;
        }
}
return false;
}
```

## 4.3 OVERVIEW

The overview section uses various Ajax requests to get a users accounts and transactions to then plot them on a HTML canvas element. Both of these will be explained in the next section, followed by the process of uploading a CSV file. The code referenced in this section is contained within 'overview.js'.

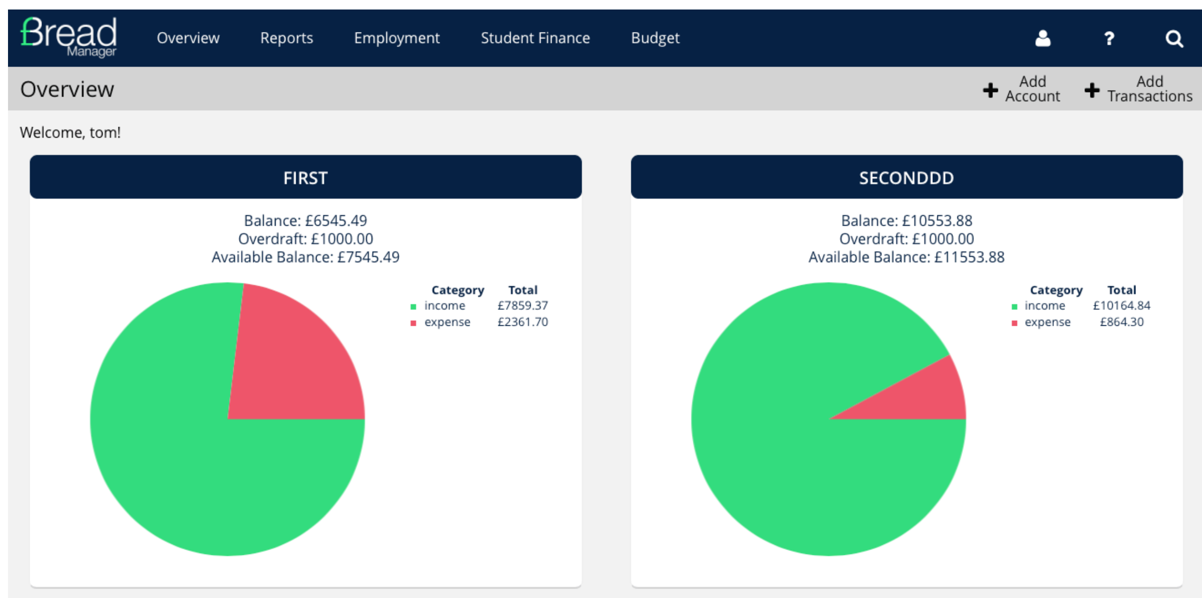


**Figure 28 - Shows the overview page with the pie charts for two accounts**

| Trans Date | Description | Amount | Category |
|---|---|---|---|
| 14/04/2016 | debit | £-1000.00 | Bills |
| 14/04/2016 | debit1 | £-1000.00 | Bills |
| 13/04/2016 | test | £2000.00 | Gas |
| 04/01/2016 | EE & T-MOBILE | £-55.10 | Food |
| 04/01/2016 | STUDENT LOANS CO | £500.00 | Books |
| 04/01/2016 | EE & T-MOBILE | £-51.10 | Gas |
| 04/01/2016 | STUDENT LOANS CO | £900.12 | Water |
| 04/01/2016 | EE & T-MOBILE | £-51.10 | Water |

**Figure 29 - showing the transaction list for the first account**

### 4.3.1 ACCOUNTS, TRANSACTIONS AND CHARTS

Once a user has added an account and transactions to their account, a pie chart is displayed for each account showing the total income, expenses and available balances. Figures 28 and 29 show the pie charts and transactions list respectively. PHP was used to generate a container for each account so that Ajax could be used to display the content. JQuery's Ajax method provides greater control over the requests, with a range of settings to apply. Mainly 'type', to define a HTTP 'GET' or 'POST' request, a 'url' to point to the document that will process the request, 'data' to send data along with the request, 'dataType' to define the type of data to be returned, and finally a success and error function that is called if the request is successful or fails. The following code shows this for retrieving user accounts:

```
$.ajax({
type: "GET",
dataType: "json",
url: "/core/ajax/getAccounts.php",
success: function(json) {}
error: function(xhr, textStatus, errorThrown) {
        alert(xhr.responseText);
}
```

The error functions are currently setup to pop up an alert with the error message. This was to aid debugging but would be implemented with functional error reporting in a final development. The success function code is not show as it is too large, however parts will be sectioned out below. Upon success it loops through each account and executes another Ajax request to retrieve the associated transactions. As shown below:

```
$.ajax({
type: "POST",
data: { accountId: item.account_id },
dataType: "json",
url: "/core/ajax/getTransactions.php",
success: function(json) {}
```

This sums the total of each income and expense, which is then passed as an object to a 'plotChart' function. Here the total amount is calculated so that the angle for each arc of the pie chart can be calculated. Looping over the transactions object to create another object, where each value is associated with a name, value, color, start and end angle.

```
for(var key in data) {
        pieData[k] = {
                name: key,
                value: data[key],
                color: colors[k],
                startAngle: 2 * Math.PI * lastPos,
                endAngle: 2 * Math.PI * (lastPos + (data[key]/myTotal))
        };
lastPos += data[key]/myTotal;
k++;
}
```

This data is then used to draw the pie chart. The code below demonstrates this process:

```
for (var i = 0; i < pieData.length; i++) {
        ctx.beginPath();
        ctx.moveTo(center[0], center[1]);
        ctx.arc(center[0], center[1], radius, pieData[i].startAngle,
pieData[i].endAngle, false);
        ctx.lineTo(center[0], center[1]);
        ctx.closePath();
        ctx.fillStyle = pieData[i].color;
        ctx.fill();

        //set up legend
        $('<tr>').append(
        $('<td>').append($('<span style="background-color:' +
pieData[i].color + '">')),
        $('<td>').text(pieData[i].name),
        $('<td>').text("£" + pieData[i].value.toFixed(2))
        ).appendTo($table);
}
```

Firstly the canvas radius and centre are calculated. The path point is set to the centre of the circle, 'ctx.arc()' draws the sector from the center, out to the radius, from the starting angle to the end angle as previously defined. This path is then filled with the assigned color and repeated for each part of the pie chart. A legend is created and appended to the canvas.

### 4.3.2 CSV UPLOAD

As per the requirements, the user can upload bulk transaction through a CSV file. This is accessed through the add transactions modal. The code is referenced from 'modal-init.js' which contains an event listener that executes when the upload input type is changed, i.e. when a file is selected. Both client-side and server-side validation ensure one and only one file has been selected and checks it is a CSV file less than 2MB in size.

The File API is then used to read the content of the file and display each record to the user so that they can verify and change any data if necessary. As seen in the code below, once the file has been selected, each row is split by a new line and subsequently each row split by a comma to obtain an array of fields for each row.

```
var files = evt.target.files; // FileList object
var f = files[0];
var reader = new FileReader();
// Read the contents of the file
reader.onload = (function(theFile) {
return function(e) {
    var article = document.createElement('article');
    // get data from csv file and split at each line break
    var data = e.target.result;
    var row = data.split("\n");

    for (i = 0; i < row.length-1; i++) {
        var item = row[i].split(','); // split row by comma
        var dateParts = item[0].split('-');
        var newDate = dateParts[2]+'/'+dateParts[1]+'/'+dateParts[0];
        }
}
```

Within the for loop above each item is displayed in an input element and appended to the modal. Some CSV files downloaded from a bank account very rarely have an extra field for a transaction record. This is caused by the transaction record having a double description name. Usually the file has three columns, the first being the transaction date, then the description and finally column three for the amount. For a transaction with a double description this structure changed to four columns, the transaction date, description name 1, description name 2 and finally the amount in column four. This only affected the offending row, so to overcome this problem a simple if statement was used to check the length of the row. If the length equalled four then column three would be ignored, and column four is placed in the html as the amount.

Figures 30 and 31 show the modal to upload a CSV file and the subsequent modal to verify the contents before upload.

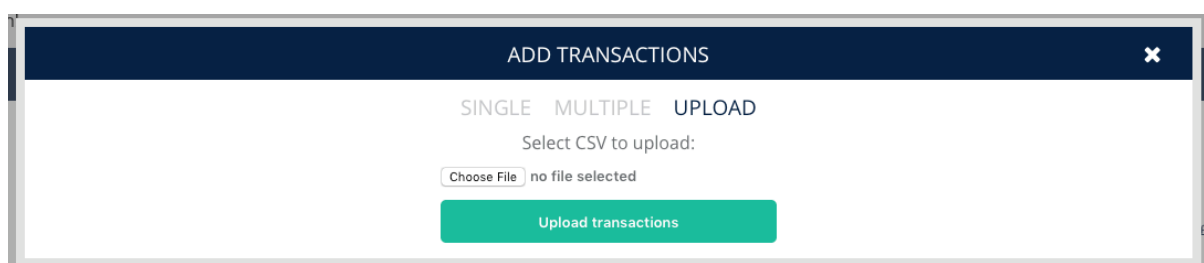


**Figure 30 - Upload CSV modal**

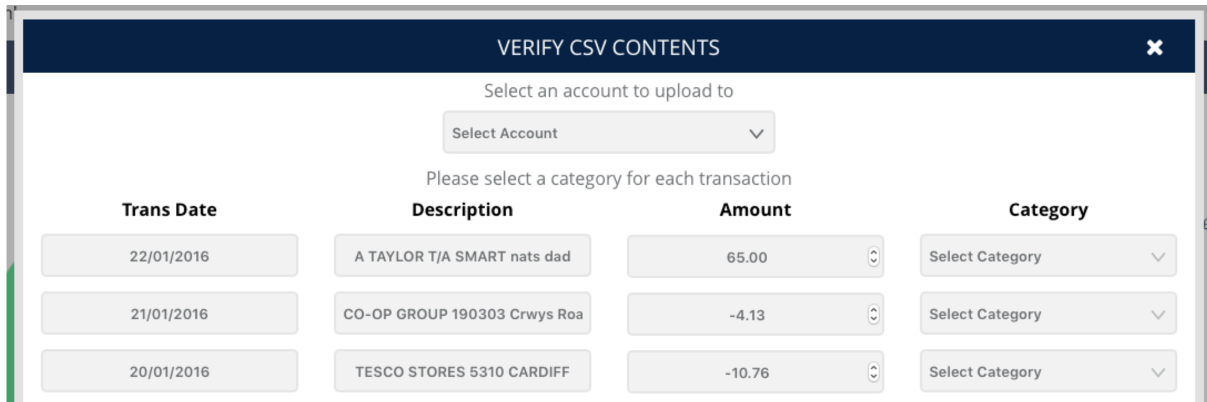


**Figure 31 - Verify CSV modal showing the details that can be edited before submission**

## 4.4 REPORTS

The reports page provides an in depth look into the users transactions. They are presented with four categories to define which transactions are displayed. Firstly to select which account will be used, a specific account can be selected or all accounts can be used. The next option is to select either income or expenses or both. Next, specific parent categories can be selected or all of them can be displayed. Finally a date range can be choosen. Various options are available including, the current week, month or year; figure 32 shows the full list. The next section will show how these were implemented. The pie chart uses the same code as described in section 4.3.1 and the bar chart uses very similar code, so will not be covered in detail but they can both been seen in figure 33 below. The modals to add accounts and transactions are also accessible from this page, but again the key details have been discussed previously.

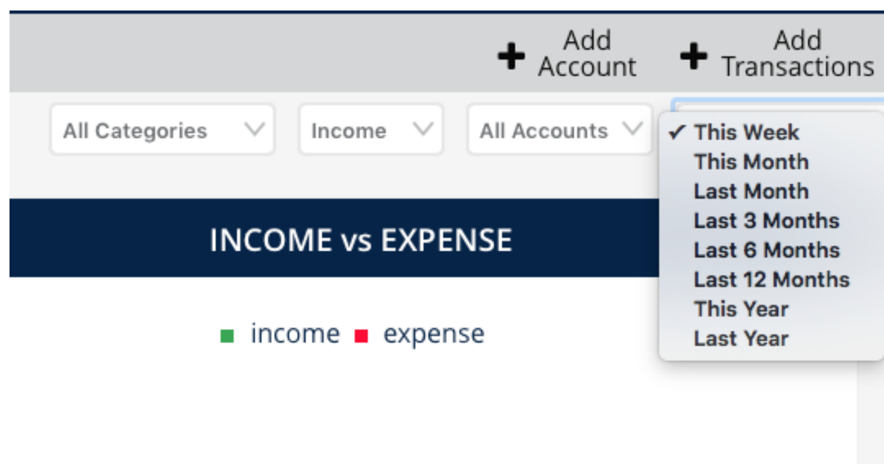


**Figure 32 - The full list of possisble date ranges, shown alongside the other selctions**

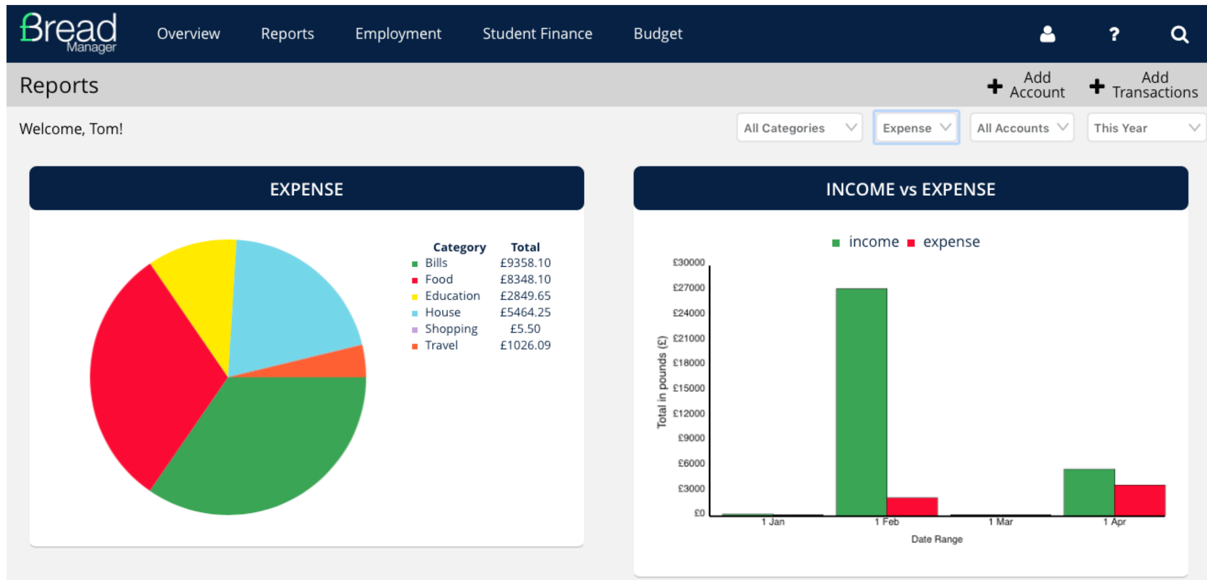


**Figure 33 - Shows the two charts for expenses from the year so far**

### 4.4.1 CUSTOMISABLE REPORTS

The category, transactions type and date range selections are all pre defined options whereas the accounts selection is populated using an Ajax request to fetch all of the users accounts. All four are connected to an event listener; this calls three functions to update the charts and transaction list every time one of the selections is changed. The values of each selection are passed onto these functions to define the selection criteria, however the date criterion requires a range between two dates. This range is calculated using the following code within 'reports.js':

```
var now = new Date();
var thisWeek = new Date();
var thisMonth = new Date();
var lastMonth = new Date();
var last3Months = new Date();
var last6Months = new Date();
var last12Months = new Date();
var thisYear = new Date();
var lastYear = new Date();

thisWeek.setDate(now.getDate() - 7);
thisMonth.setDate(1);
lastMonth.setMonth(now.getMonth() - 1, 1);
last3Months.setMonth(now.getMonth() - 3);
last6Months.setMonth(now.getMonth() - 6);
last12Months.setMonth(now.getMonth() - 12);
thisYear.setMonth(0, 1);
lastYear.setFullYear(now.getFullYear() - 1, 0, 1);

dateRanges = [thisWeek, thisMonth, lastMonth, last3Months, last6Months,
last12Months, thisYear, lastYear];
```

Firstly a variable for each of the possible options is instantiated as a date. The values are then calculated using JQuey's date methods by subtracting the appropriate number of days, months or years from todays date. This will ensure the selected range is always based from todays date. These values are then stored in an array so they can be accessed easily. The code below shows a switch statement for the date range selected by the user. Each case assigns a start date and end date using the array of dates created above. For example, if the user selects the option 'last3Months' then a start date is set as the first day of the month, three months before todays date and an end date as todays date. The date range is set to 'thisWeek' by defualt.

```
switch(dateRange) {
        case '4':
                startDate = dateRanges[4];
                endDate = now;
                break;
        default:
                startDate = dateRanges[0];
                endDate = now;
                break;
}
```

While plotting the Income vs Expense bar chart, a problem arose involving getting the transactions and plotting the data once received. The bar chart uses a for loop and a series of Ajax requests within the loop to retrieve the transactions for each bar group of income and expense to then store in a data array. So for example, a date range of last week would require 7 loops of Ajax requests to obtain the income and expense for each day of the week, adding the data for each day to the data array after every loop. The chart could only be plot once the data array was complete. As Ajax uses an asynchronous processing model, its success function would execute the plotData() function every time the request was successfully completed and so was executing with incomplete data. To solve this an if statement was created with the condition to only execute once a variable was equal to the number of requests required for a complete data set. The for loop initialising the Ajax requests can be seen below. Followed by the Ajax request itself and the plotData() function.

```
case '0':
// for loop calculates number of days between date range
for (var i = 0; currDate <= endDate; i++) {
        dates[i] = [days[currDate.getDay()] + " " + currDate.getDate(),
currDate.getTime()];
        currDate.setDate(currDate.getDate() + 1);
}
// for loop to call ajax request for transactions
for (var i = 0; i < dates.length-1; i++) {
        var date1 = new Date(dates[i][1]);
        var date2 = new Date(dates[i+1][1]);
        // ajax request
        getTrans(type, accountId, date1.toString(), date2.toString())
}
break;
```

```
function getTrans(type, accountId, startDate, endDate){
$.ajax({
        type: "GET",
        data: { type: type, accountId: accountId, startDate: startDate,
endDate: endDate, categoryId: categoryId },
        dataType: "json",
        url: "/core/ajax/getReportTransactions.php",
        success: function(json) {
                var income = 0;
                var expense = 0;
                $.each(json, function(j, item) {
                        var value = parseFloat(item.amount);
                        if(value < 0) {
                                expense += value;
                        } else {
                                income += value;
                        }
                });
                expense *= -1;
                tempData.push([income, expense]);
// plotData function to execute only after all ajax requests finished
                plotData();
        },
        error: function(xhr, textStatus, errorThrown) {
                alert(xhr.responseText);
        }
});
};
```

```
function plotData() {
// if statement solution
if(ready >= dates.length-1){
        for (var i = 0; i < dates.length-1; i++) {
                barChartData[dates[i][0]] = [tempData[i][0],
tempData[i][1]];
        }

        var canvas = "canvas2";
        var container = "#report-graph-2";
        $(container).find('.legend').empty();
        $(container).find('.legend').append($('<table>').append($('<tr>').a
ppend(
                $('<td>').append($('<span style="background-color:' +
colors[0] + '">')),
                $('<td>').text("income"),
                $('<td>').append($('<span style="background-color:' +
colors[1] + '">')),
        $('<td>').text("expense")
        )));
```

```
        plotBarData(canvas, barChartData);
}
// incrments variable for if condition
ready++;
}
```

## 4.5 STUDENT FINANCE

The student finance system has been set up so that a user can only have one student loan account as the majority of students will only ever have one. This could be expanded in the future to allow for students on second degrees that are able to receive a second student loan. The setup process is shown in the figures 34 and 35. The student loan setup module is active while the add instalment module is deactivated so no instalments can be added before an account is created. Once a loan name is submitted, an Ajax request is executed to create the account, this will active the add instalments module and deactivate the loan setup.



**Figure 34 - The student loan setup, showing instalments as disabled**



**Figure 35 - The second step of student loan setup, Instalments is active and setup is deactivated**

Any number of instalments can be added by clicking the 'plus' icon. This will add another row of inputs, similarly clicking the 'minus' icon will remove a row of inputs. The code below shows how this is implemented.

```
// add installment input row
$(document).on('click', '.add-installment-btn', function() {
	$container = $(this).closest('form').find('table');
	$(this).parent().prepend($removeBtn);
	var $row = $('<tr>').append(
	$('<td>').append($periodSelect),
	$('<td>').append($loanTypeSelect),
	$('<td>').append('<input type="date" class="loan-calendar"
name="installment_date[]" placeholder="Installment Date" required/>'),
	$('<td>').append('<input type="number" name="amount[]"
placeholder="Amount" required/>')
	);
	$container.append($row);
});

// remove installment input row
$(document).on('click', '.remove-installment-btn', function() {
	$lastRow = $(this).closest('form').find('table tr:last');
	$lastRow.remove();
	if ($(this).closest('form').find('tr').size() <= 2) {
		$removeBtn.detach();
	}
});
```

An if statement checks the number of rows remaining after clicking the 'minus' icon, if this value is less than or equal to two then the button is removed so that no more rows can be removed. The two remaining rows are the header and one input row. After clicking 'add installments' another Ajax request is submitted. During this process The instalments have to be added to their corresponding totals within the 'loan' database. The code below demonstrates this:

```
$current_total = $this->data()->total_loan;
$new_total = $fields['amount'] + $current_total;
$this->update(array('total_loan' => $new_total));
$loan_type_id = $fields['loan_type_id'];

if ($loan_type_id != 2) {
	$current_owed = $this->data()->amount_owed;
	$new_owed = $fields['amount'] + $current_owed;
	$this->update(array('amount_owed' => $new_owed));
}
if ($loan_type_id == 1) {
	$maintenance_loan = $this->data()->maintenance_loan;
	$new_maintenance_loan = $fields['amount'] + $maintenance_loan;
	$this->update(array('maintenance_loan' => $new_maintenance_loan));
} elseif ($loan_type_id == 2) {
```

```
        $maintenance_grant = $this->data()->maintenance_grant;
        $new_maintenance_grant = $fields['amount'] + $maintenance_grant;
        $this->update(array('maintenance_grant' =>
$new_maintenance_grant));
} elseif ($loan_type_id == 3) {
        $tuition_fee = $this->data()->tuition_fee;
        $new_tuition_fee = $fields['amount'] + $tuition_fee;
        $this->update(array('tuition_fee' => $new_tuition_fee));
}
```

Firstly, the current total is fetched from the database. The sum of the instalment amount and current total is submitted back to the database, using the database update method. A series of if statements check the 'loan_type_id' so that the accumulated loan_type total is updated correctly. After the instalments have been submitted the overall summary of the loan and yearly breakdown is displayed, shown in figure 36.

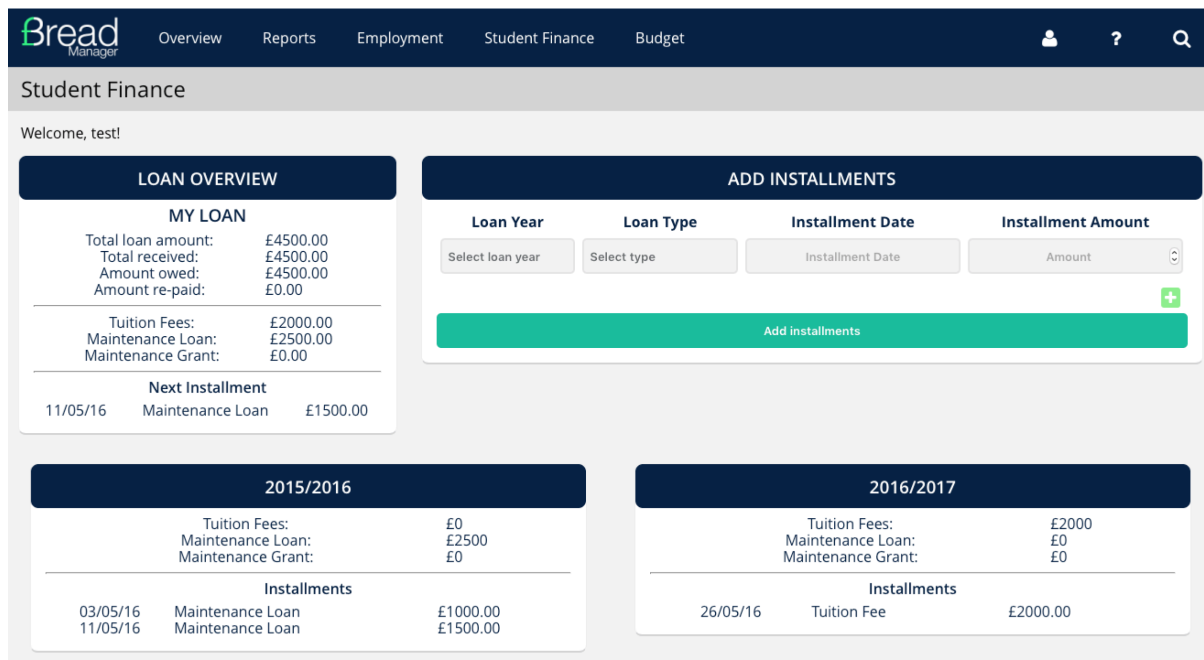


**Figure 36 - Student Loan overview**

## 4.6 PROBLEMS ENCOUNTERED

Having identified a few problems specific to areas in the above sections, other problems encountered will be discussed here. Most were fairly minor and were identified during testing throughout the development. Examples include:

- Incorrect dates being stored in transactions table as 0000-00-00. This was due to the format that the date was being submitted to the database. A simple change of syntax in php and the JQuery datepicker solved this issue.
- Problems closing the modal using the 'close' icon button. This was caused by a low level scope for the event listener, such that it was not being recognised. Changing to JQuerys 'on() method applied to the document solved this.

```
$(document).on('click', '.alert-close', function() {}
```

- Font weights appearing differently across different browsers. This is down to the browsers font rendering engine. Using CSS to alter 'text-rendering' or 'font-smoothing' may help but ultimately their default values differ between browsers. So difference may still occur.

As mentioned at the start of section 4, the main problem encountered was using a depreciated database extension. This has subsequently had knock on effects for the rest of the implementation. The original plan set out in the initial report states the implementation should have been finished by the end of week 9. However the mistake during this process delayed this task by an extra week and also resulted in two sections, Employment and Budgeting, not being implemented in the final development. They would have been completed within another week, but due to time constraints it was decided the risk was not worthwhile delaying the report any longer.

# 5 RESULTS AND EVALUATION

The aim identified at the start of the project has partially been achieved. A system has been created to allow the user to manage their personal finances. However various requirements have not been met. This is mainly due to the Budgeting and Employment sections not being implemented. This has been described in section 4.6 above. The objectives set out in introduction have largely been met as well. They identified objectives for the progression of the project, which have been followed through to the end. While some requirements may not have been accomplished, those relating to what has been implemented have largely been achieved. Functionality testing during the development process has helped this achievement. Each feature that was implemented was tested for redirects, validation and wrong inputs. For example, adding transactions to an account. This process involved constant trials on incorrect or unwanted data to produce strict validation rules that only allow required data. Regular expression have been used to ensure only the correct date format is accepted, similarly with an email address, only strings with the accepted format will be allowed. The test cases proposed in section 3.3.4 set out functional testing for various requirements. These will be discussed in the next section.

## 5.1 TEST CASES

The test cases defined in the specification and design section were chosen as they covered the main functionality proposed. Some test cases across the different sections would be very similar because of the features they both provide. For example, Displaying and editing transactions within the transaction list on both the Overview and Reports pages would yield practically the same tests. Test case 11 has been selected to represent the other browsers. The two main browsers available were Safari and Google Chrome on a mac, so have been tested on both. Internet explorer has been tested to the extent of using Safari developer tools to simulate using IE. This section will continue by evaluating the results of the test cases.

| **Test 1:** | Can the user register an account. |
|---|---|
| **Actual Outcome:** | User successfully registered an account with username 'twhiddett' and email 'twhiddett@gmail.com'.<br>Details are stored in the database. |

| **Test 2:** | Can the user log into the application |
|---|---|
| **Actual Outcome:** | User successfully logged in and redirect to the overview page. |

| **Test 3:** | Can the user create an account. |
|---|---|
| **Actual Outcome:** | Successfully added the account with provided details. |

| **Test 4:** | Can the user upload transactions to an account. |
|---|---|
| **Actual Outcome:** | Successfully selected CSV file to upload, verified the contents and selected an account and categories for each transaction. |

| **Test 5:** | The user can customise the display of different reports. |
|---|---|
| **Actual Outcome:** | All the transactions are shown as expected with the results show in both the pie chart and bar chart. |

| **Test 6:** | Can the user edit individual transactions. |
|---|---|
| **Actual Outcome:** | Successful edit of transaction, from category 'food' to 'groceries'. |

| **Test 7:** | Can the user delete a transaction. |
|---|---|
| **Actual Outcome:** | Transaction successfully deleted, no longer displayed within the database or the transactions list.<br><br>However directly after deletion, the remaining transactions were duplicated within the transactions list. This is only a visual bug caused by the table not clearing properly. After a page refresh they are displayed correctly. |

| **Test 8:** | Test whether the user can create a student loan |
|---|---|
| **Actual Outcome:** | Student loan successfully created and an overview displayed. |

| **Test 9:** | Test whether the user can create a budget. |
|---|---|
| **Actual Outcome:** | Failed test as the Budgeting has not been implemented. |

| **Test 10:** | Can the user log out. |
|---|---|
| **Actual Outcome:** | User successfully logged out and redirected to the login screen. |

| **Test 11:** | Does the application display correctly in Google Chrome. |
|---|---|
| **Actual Outcome:** | The application displays correctly in Google Chrome as well as internet explorer through Safari developer tools. The only noticeable difference is the font rendering which was previously mentioned in section 4.6. |

Test 12 and 13 have been added to include features that have been overlooked in the initial test case. Mainly compatibility testing. Mobile devices need to be tested to observe the responsive design and its functionality.

| **Test 12:** | Can the user delete their account and subsequently all associated accounts and transactions. |
|---|---|
| **Environment:** | OS X |
| **Browser:** | Safari |
| **Pre-Conditions:** | Must be logged in and have created an account with transactions. |
| **Process:** | 1. Select the user icon.<br>2. Select 'Update Details' button.<br>3. Select 'Delete Account'<br>4. Select 'Are you sure?' |
| **Expected Outcome:** | The user is logged out of the application, they cannot log back in and their accounts have been deleted from the database. |
| **Actual Outcome:** | User is logged out and a confirmation message is displayed at the login page. The users account transactions and student loan have successfully been removed from the database. |

| **Test 13:** | Is the application responsive. |
|---|---|
| **Environment:** | OS X |
| **Browser:** | Google Chrome – Device mode |
| **Pre-Conditions:** | Must be logged in. |
| **Process:** | 1. Within Google chrome access the developer tools and toggle device mode.<br>2. Select responsive to customise screen size to various sizes.<br>3. Select iPhone 6 Plus and Galaxy S5. |
| **Expected Outcome:** | The application should scale accordingly and function on a mobile device. |
| **Actual Outcome:** | The application scales down to size, however on the smaller screens of a mobile device the navigation bar is too congested as is the transaction list. The application functions very well on an iPad. Figure 37 shows an iPhone, figure 38 shows a Galaxy S5 |

| | device and figures 39 and 40 show an iPad version. |
|---|---|

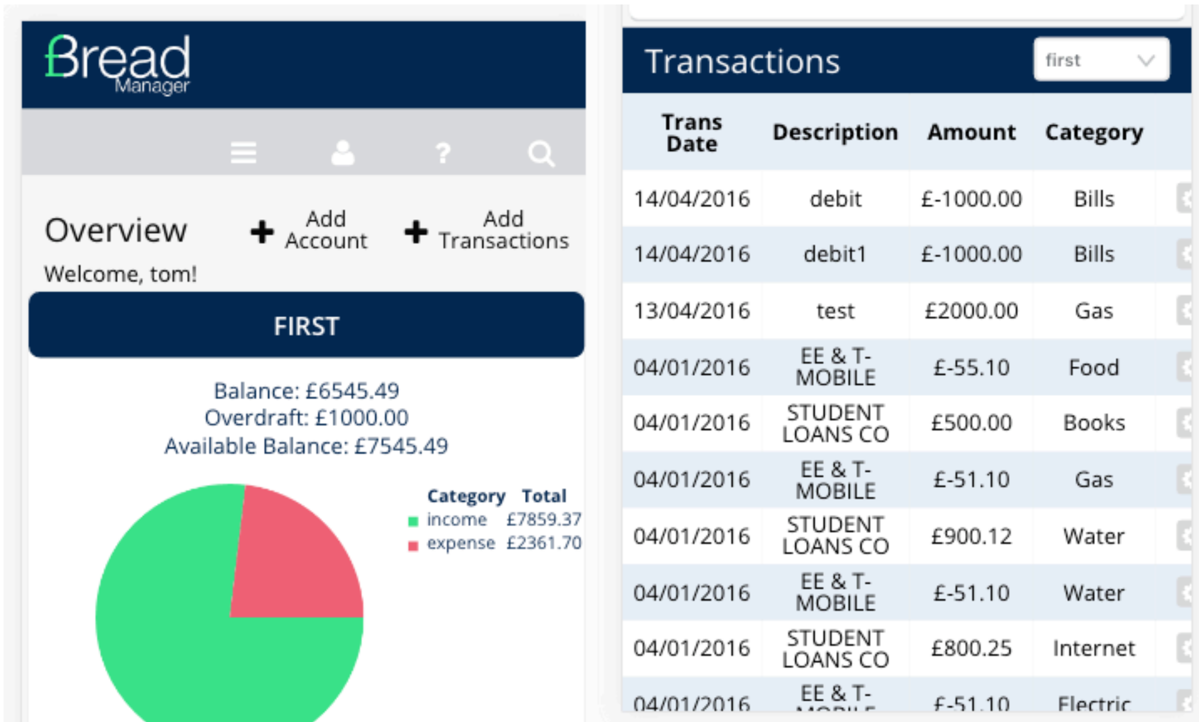


**Figure 37 - Test 13 Shows the application displayed on an iPhone 6 Plus**

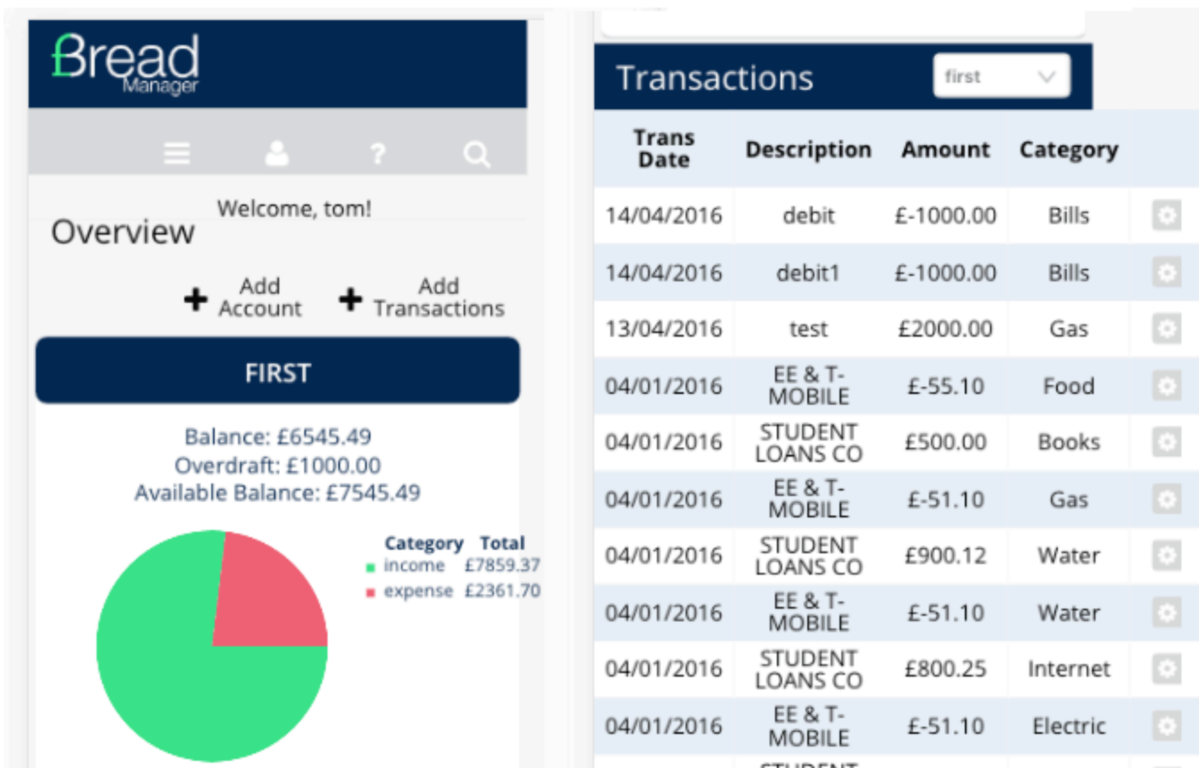


**Figure 38 - Test 13 Show the application displayed on a Galaxy S5**

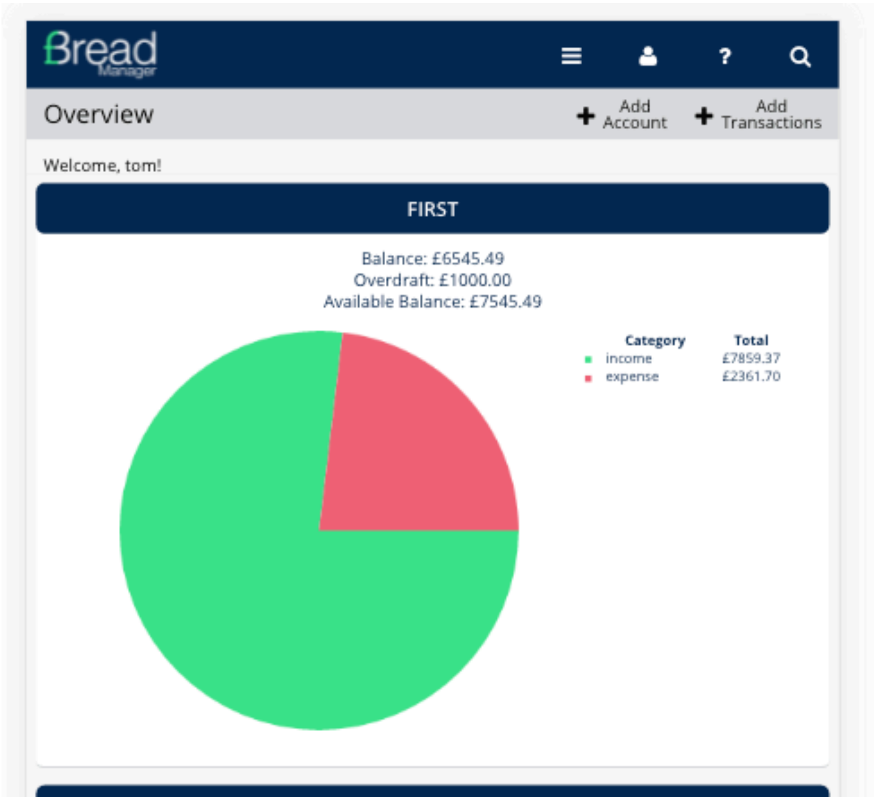


**Figure 39 - Test 13 shows the overview chart displayed on an iPad**

Transactions

| Trans Date | Description | Amount | Category |
|---|---|---|---|
| 14/04/2016 | debit | £-1000.00 | Bills |
| 14/04/2016 | debit1 | £-1000.00 | Bills |
| 13/04/2016 | test | £2000.00 | Gas |
| 04/01/2016 | EE & T-MOBILE | £-55.10 | Food |
| 04/01/2016 | STUDENT LOANS CO | £500.00 | Books |
| 04/01/2016 | EE & T-MOBILE | £-51.10 | Gas |
| 04/01/2016 | STUDENT LOANS CO | £900.12 | Water |
| 04/01/2016 | EE & T-MOBILE | £-51.10 | Water |
| 04/01/2016 | STUDENT LOANS CO | £800.25 | Internet |

**Figure 40 - Test 13 shows the overview transactions on an iPad**

## 5.2 STRENGTHS AND WEAKNESSES

Reflecting on the results from the tests has revealed some strengths and weaknesses within the application. Areas where the system is successful include the reporting and student loan. Mainly the flexibility to customise and produce various reports. The available options for selecting a date range and parent categories provide a good range of possibilities. The infographics supporting these options provide a clear visual aid of income and expenses over a period of time. While the pie chart allows further insight into spending habits by assessing areas of high expenditure. The student loan interface

may be fairly straightforward but having access to all the information in one location will be beneficial. Another feature that is invaluable is the ability to import CSV files. This makes the application so much more practical and will save a lot of time. Furthermore the layout and structure also strengthen the application. Every feature is within a few clicks and is easy to navigate between different sections. However, there is room for improvements, but these will be discussed in the next section.

Areas of weakness within the application include the responsive architecture. While it is responsive, unfortunately the implementation is lacking on smaller devices. Further thought needs to be put in to the transaction list. It may be a case of making the font smaller to help fit all the columns onto the screen or change the method of editing transactions. The navigation can be fixed by moving the various icons into the responsive navigation dropdown. With more time this would have been implemented. Another area of weakness is the account overview. The available balance is helpful but in hindsight the piecharts are not exactly meaningful. Displaying the 5 most recent transactions would have been more beneficial. Lastly, the application feels too ridged; there is no animation that is needed to enhance the user interaction.

Overall the application has proved a success, fulfilling the overall aim to provide a tool to manage personal finance. Of the three sections implemented, all of their functional requirements have been met as well as most of the general functional and non-functional requirements. The only non-functional requirement that has fallen short is the responsiveness for a variety of devices.

The approach throughout the project has worked well, although completing an in-depth risk analysis would have supported some leeway in the project plan. This would have potentially reduced the knock on affect of restructuring the database half way through the implementation. The chosen programing languages presented a challenge, having not used Ajax in the past. Therefore some research into various frameworks, such as angular.js, could have eased the learning curve required.

An alternative approach might have produced a better solution that met all the requirements. However it would not have developed the experience gained throughout this process. Therefore the chosen approach is still considered to have been appropriate for the project.

# 6 FUTURE WORK

Firstly the future work would consist of finishing the uncompleted features, the employment and budgeting sections. Implementing a planner for future savings could further expand the budget system. While also including a savings calculator, with two modes. One to save up for an item with a given cost, the calculator would then estimate how long it would take to save up for the item based on a set savings rate. Secondly to save up for an item by a given date, the calculator would then calculate how much needs to be saved each week or month.

The other sections can also be developed by slightly improving some features. For instance, rather than clicking a button to edit a row of transitions, the user could click on the field itself to begin editing. Once focus is lost from the field, an Ajax request can attempt to submit the changes. The reporting could also be improved upon by selecting multiple categories; this could effectively be used to compare different categories. Enhancing the piecharts by clicking on a segment that is a parent category, this would then progress to its subcategories, displaying their breakdown on the piechart.

Some of these could be implemented using existing frameworks. Chart.js provides various chart types with customisable animations and a responsive design. Angular.js is an example of a framework that would allow for a more fluid web application.

As mentioned in section 5.2 the user interaction needs to be developed to incorporate some animations and to improve the overall functionality of the application. A touch friendly design on mobile devices could implement swiping between different sections of the page. The final future work would be to explore further security threats and the measures needed to protect against them.

# 7 CONCLUSIONS

The project set out with the core aim to produce a piece of software that would assist a user in managing their personal finances. Secondary aims were to develop the system to be a functional and aesthetically pleasing product. As part of this process, requirements were defined that would guide the project towards completing these goals. These have been evaluated and so it can be concluded that the overall aims have been achieved.

Some challenges were encountered along the way that set the project behind, but in the end a solution has been produced that satisfies the requirements. While the proposed design was not completed, the sections that were, accomplished the goals they were designed for. An alternative approach may have produced a different outcome but it would not have developed the experienced gained in overcoming these challenges. Improvements have been identified for both the completed and unfinished designs, so that the project can be developed in future work.

# 8 REFLECTION ON LEARNING

This project has certainly provided me with some valuable experience both in the software development lifecycle and in developing my programming skills. Having little experience with Ajax and the basics of PHP, I feel the project has allowed me to challenge my abilities and push myself to quickly learn new languages and approaches to problem solving.

Programming the whole application from scratch rather than using a framework such as angular.js, may have been a bit ambitious. But it has allowed me to actually learn about the core language and understand various functions rather than learning how a framework implements the same functionally. Although I would have liked to have implemented more of the designed system, the mistake I made by using the depreciated mysql extension rather than the mysqli will teach me to be more cautious and conduct thorough research before rushing into the development.

Initially I was worried that my weeks work experience was going to hamper the time I had to spend on the project. But it actually turned out to be very helpful. They introduced me to PDO databases, while it was only the basic concepts to create a database connection, bind and prepare statements and the basic functionality. It prompted me to research it further and consequently implement it into the project. This did involved a quick and steep learning curve to get to grips with the more advanced features but it totally paid off as the database object setup ended up being so flexible.

The project plan was affected and delayed after having to restructure the database. But I feel I could have created an updated work plan so I had clearer goals that could be achieved each week. This may have helped get back on track and may have also allowed for further development and testing.

Ultimately the project has provided me with a developed skillset that I can use to take with me in future projects and further in my career.

# REFERENCES


[1] Quicken Inc. *"Quicken® personal finance, money management, budgeting,"* 2016. [Online]. Available: http://www.quicken.com. [Accessed: 30th Jan. 2016]

[2] Intuit. *"Mint: Money, bill pay, credit score & investing,"* 2016. [Online]. Available: https://www.mint.com. [Accessed: 30 Jan. 2016]

[3] You Need A Budget. *"YNAB. Personal finance software to take total control of your money,"* 2016. [Online]. Available: http://www.youneedabudget.com. [Accessed: 31 Jan. 2016]

[4] Money Dance. *"Personal finance manager for Mac, windows, and Linux,"* 2015. [Online]. Available: http://moneydance.com. [Accessed: 30 Jan. 2016]

[5] Accountz. *"Accounting software | bookkeeping software,"* 2016. [Online]. Available: http://www.accountz.com. [Accessed: 31 Jan. 2016]

[6] Microsoft Office. *"Financial management Templates,"* 2016. [Online]. Available: https://templates.office.com/en-ca/Financial%20Management. [Accessed: 31 Jan. 2016]

[7] GOV.UK. *"Running payroll - Payments,"* 2016. [Online]. Available: https://www.gov.uk/running-payroll/payments. [Accessed: 06 Feb. 2016]

[8] GOV.UK. *"Running payroll - Deductions,"* 2016. [Online]. Available: https://www.gov.uk/running-payroll/deductions. [Accessed: 06 Feb. 2016]

[9] GOV.UK. *"Student finance,"* 2016. [Online]. Available: https://www.gov.uk/student-finance-for-existing-students. [Accessed: 06 Feb. 2016]

[10] Ethan Marcotte, "*Responsive Web Design*, 2nd ed.", 2014.

[11] Sketch. *"Professional Digital Design for Mac,"* 2016. [Online]. Available: https://www.sketchapp.com. [Accessed: 14 Feb. 2016]

[12] *"MAMP & MAMP PRO"*. [Online]. Available: https://www.mamp.info/en/. [Accessed: 25 Apr. 2016]

[13] The PHP Group.. [Online]. Available: http://php.net/manual/en/book.pdo.php. [Accessed: 26 Mar. 2016]

[14] Sass. *"Sass documentation"*. [Online]. Available: http://sass-lang.com/documentation/file.SASS_REFERENCE.html. [Accessed: 24 Feb. 2016]

[15] Dan Cederholm, "*Sass for web designers*".: A Book Apart, 2013.

[16] Dave Gandy. *"Font Awesome"*. [Online]. Available: http://fontawesome.io. [Accessed: 24 Feb. 2016]

[17] Eric Meyer. *"CSS tools: Reset CSS"*. [Online]. Available: http://meyerweb.com/eric/tools/css/reset/. [Accessed: 24 Feb. 2016]

[18] Jon Duckett, "*JavaScript and JQuery: Interactive front-end web development*".: John Wiley & Sons, 2014.

[19] Ajax. *"Ajax Documentation,"* 02 Mar. 2016. [Online]. Available: http://api.jquery.com. [Accessed: ]