UNIVERSITY
OF HULL

Personal Financial Planning and Modelling Package

Final Report

Submitted for the BSc/MEng in

Computer Science

June 2020

by

Benjamin M A Stanley

Word count: 14372

Abstract

This project attempts to examine the existing financial planning application solutions that are currently available, to take certain features for inspiration, and develop an application that will allow even the most novice users the ability to gain control of their finances. The system developed will allow the user to input income and expenditure transactions, and thus have a breakdown of these into lists and charts with category visualisation. The system should also incorporate the use of roles, thus leading to the implementation of admin capabilities, which the normal user should not have access to. This project will cover the range of existing development technologies available, along with design and user interface issues.

Acknowledgements

I would like to take this opportunity to thank Dr. Bing Wang for his continued project supervision and overall support, and I would also like to thank Dr. Neil Gordon for invaluable advice at the mid-project review.

Additionally, I would also like to thank my closest friends – Jamie, Harriet, and Owen, for making my time at university enjoyable and helping me through everything.

Contents

1 Introduction

1.1 Report Structure

This report will provide an overview for the development of a personal financial planning solution. It will outline the task at hand, the objectives and requirements that the success of the project will be based upon, and the implementation of a solution. The existing applications in the current market will be examined, along with an in-depth analysis of the available technologies for application development, along with the methodology behind the eventual choice of implementation. The design section entails all aspects of user interface and system use, ranging from initial design prototypes to the UML class and activity diagrams, to the final User Interface. The techniques that are incorporated throughout the project will be discussed, with a thorough evaluation of the project software against the initial requirements, and a conclusion regarding the success of the project as a whole.

1.2 Background/Aims and Objectives

A system was required to be designed and created with the basis of being a financial planning and modelling application. The original specification for this specific project, provided by the University of Hull was as follows:

> *Aim of this project is to build a system which will allow an individual to record, plan and model their personal financial affairs which could include income, expenditure, budgeting, savings, investments, and tax. The precise area to be covered by this project may be chosen by the student.*
>
> *For example, the system could be essentially a record keeping system for financial transactions or a financial tool to compare products. For example, it could perform the calculations required to compare various mobile phone contracts or credit card offers.*
>
> *In the initial part of the project the student would investigate existing financial tools and then write a detailed specification of the system to be built.*
>
> *(University of Hull, 2019)*

This brief was open to interpretation, and the decision was made that the niche that was going to be targeted involved allowing a user the ability to track all their everyday transactions. This would include sources of income (wages, investment returns, or even perhaps birthday money) and expenditures e.g. Leisure, Taxes or other.

Regarding the overall aims of this project, the main goal of the project was as stated above, to develop and build a free to use system. This proposed system will allow an individual to either manually enter data, or have auto-generation of data via the use of a bank API, and the user signing in via the application. From this, the application will provide a useful breakdown for the user of all income/expenditure transactions, with the ability to filter over a time period, or have a category specific breakdown so they could see the how much they were spending and how much was allocated to each category. The application would also benefit from a savings feature, with the user providing an amount they would like to allocate towards this every week or month for example.

Further goals include providing an attractive system design, with a non-overwhelming layout, allowing for a larger potential target market, specifically those who may be daunted by the use of some existing financial planning applications. The solution should also be highly accessible and compatible with many devices/users.

The development choices and available technologies for this are discussed in section 2, and the specific software requirements as a result of the development choices are discussed in section 3. Overall, the decision was made to produce a web application, that is available on a range of applications, that the user manually enters data into, and is saved into a database under their respective user account which they are required to register.

2 Literature Review

2.1 Problem Context

In today's society, there is an astonishing amount of people who consider themselves or are considered to be not in control of their finances. Studies undertaken by the Money Advice Service and other sources support this view.

It was revealed that in 2015, 4 out of 10 adults were not in control of their finances, that 1 in 5 could not read a bank statement, and 1 in 3 could not calculate the impact of a 2% annual interest rate on £100 savings (The Money Advice Service, 2015). This shows the current financial literacy of the general public as a whole, and the fact that they can be daunted by finance jargon means a gap in the market for a more simple and non-overwhelming application.

Research in 2019 by Neyber also discovered that 1 in 5 young people admitted that their finances are out of control. (Neyber, 2019)

Another study undertaken in 2017 showed that 1 in 6 people in the UK were burdened with financial difficulties, and that there were over 8 million adults living with debt problems. (The Money Advice Service, 2017).

In the current climate, there is definitely higher demand for financial planning applications, one source stating that there was an increase of 90% market share for finance apps in 2020 (BusinessWire, 2020). This shows room for the proposed project software in this market, and the large target market that it can be designed for.

There is also the additional benefit of improving mental health when an individual is under control of their finances. An article from AgeUk claims that in people over the age of 65, roughly a third of those who had regained control of their finances had become "generally calmer and happier as a result" of this. Despite this, 1 in 6 of this age group have nobody to talk to about financial issues, and thus the incorporation of a financial planning application that they could use without being too overwhelming could prove invaluable. (AgeUk, 2018)

As seen in the sources above, there is a large need from a large target market, and the implementation of a financial planning solution has potential to provide benefits for both ends of the age spectrum.

2.2 Existing Solutions

Regarding the financial planning applications that currently are available, there are a number that range in capabilities, accessibility, and price range. There are a number of features shown that the implementation of in this project would be beneficial. Ideas for appropriate UI can be seen, and a specific market niche can be gauged from what solutions currently exist. (thebalance.com, 2020)

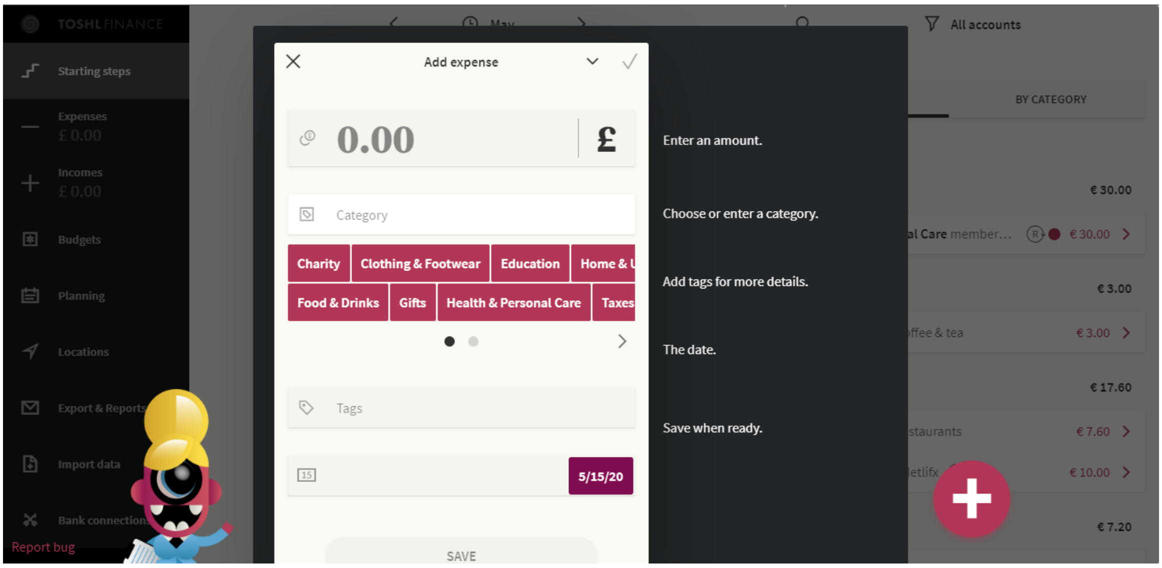
2.2.1 Toshl

This financial planning application has implementations for a web application and mobile applications for each operating system - IOS, Android and Windows Phone. This application provides a well formatted and designed layout of a users' income and expenditures into various categories, with an attractive UI. The approach of incorporating animated characters makes the application less mundane to use and instils a more enjoyable/light-hearted take on managing your finances. Useful features include the option for manually entering data and also connecting with banks and credit cards. This platform enables syncing of user data between the web application and any other devices, and has the ability to add receipt photos, reminders, locations, and repeats for transactions. This app also has multiple currency handling, which is a unique feature relative to the other existing solutions.

Platform(s): Web, Android, IOS, Windows Phone

Price: FREE

Rating (/10): 9



Enter an amount.

Choose or enter a category.

Add tags for more details.

The date.

Save when ready.

Figure 1: Toshl Web Application (Toshl, n.d.)

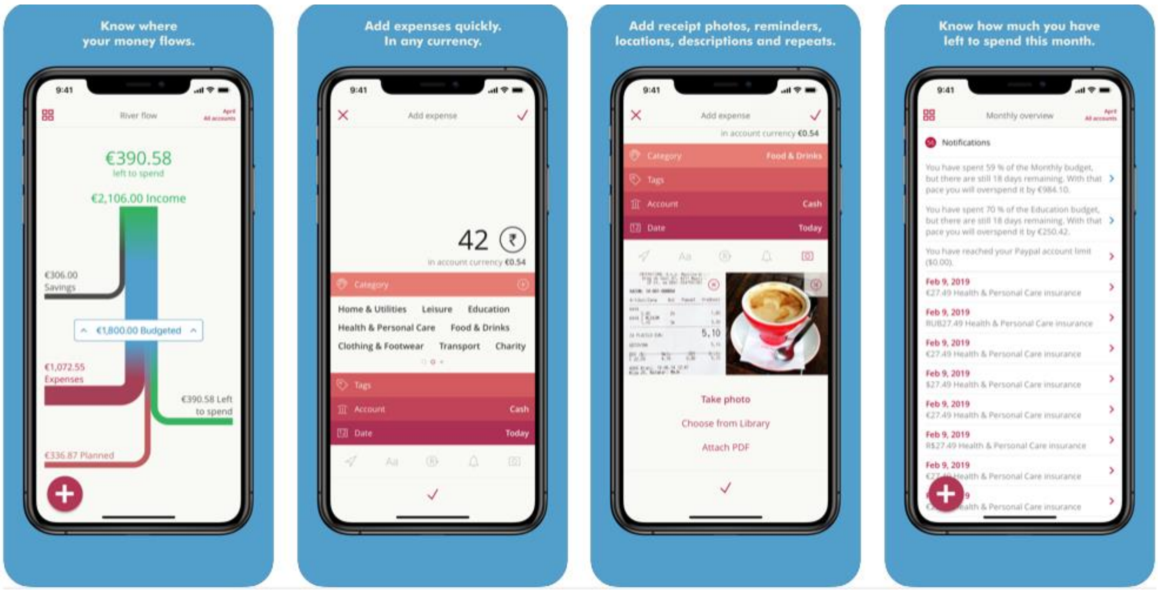

Know where your money flows.
Add expenses quickly. In any currency.
Add receipt photos, reminders, locations, descriptions and repeats.
Know how much you have left to spend this month.
€390.58
left to spend
€2,106.00 Income
€1,800.00 Budgeted
42
Take photo
Choose from Library
Attach PDF
Monthly overview
Notifications

Figure 2: Toshl Mobile Application (Toshl Inc, 2020)

2.2.2 Quicken Quicken

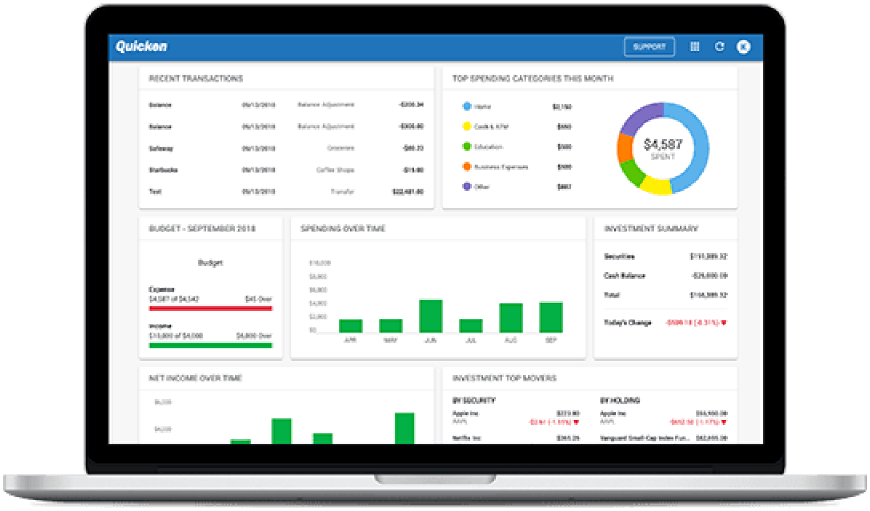


Figure 3: Quicken Web Application (Quicken, n.d.)

The Quicken financial solution is one of the more established and long running examples available, with 17 million users over the past 30 years (Quicken, n.d.) and has a simultaneously professional UI yet simple to understand and non-overwhelming layout. Key features including the ability to view balances, budgets, accounts, and transactions, furthermore, providing an analysis of your investments, which can be a useful tool. The app allows integration for exporting of data to Excel which can then be further modified if required. This app allows the user to pay bills directly, which is a feature few competitors analysed have. Windows, Mac, iOS, and Android applications are available, and devices are compatible so data can be synced between them via the user account. The app implementations are free to download, however these require a quicken account/subscription to be in place which will cost significantly more than other solutions available, with various plans starting at a hefty $34.99 (£28.90). This solution can be considered to provide the best financial overview out of these options discussed, however it has a cost element, which will affect the selling point for some. Thus, despite the vast range of features that this application offers and its usability, the high price may potentially be discouraging to a large number of customers, who may look to alternative free solutions.

Platform(s): Windows, MacOS, iOS, Android

Price: $34.99 basic plan

Rating (/10): 7

2.2.3 Mint

Mint pulls your transactions upon connection with a bank (required) and automatically categorises them to give you a visual breakdown of your spending. The main goal of this application is providing a good financial overview to the user. One unique factor with this solution is that it can show the user their real time credit score, with a breakdown as to what factors are affecting this and how they are doing so. This app is similar to Quicken as it can also track investments, along with bank accounts and credit cards.

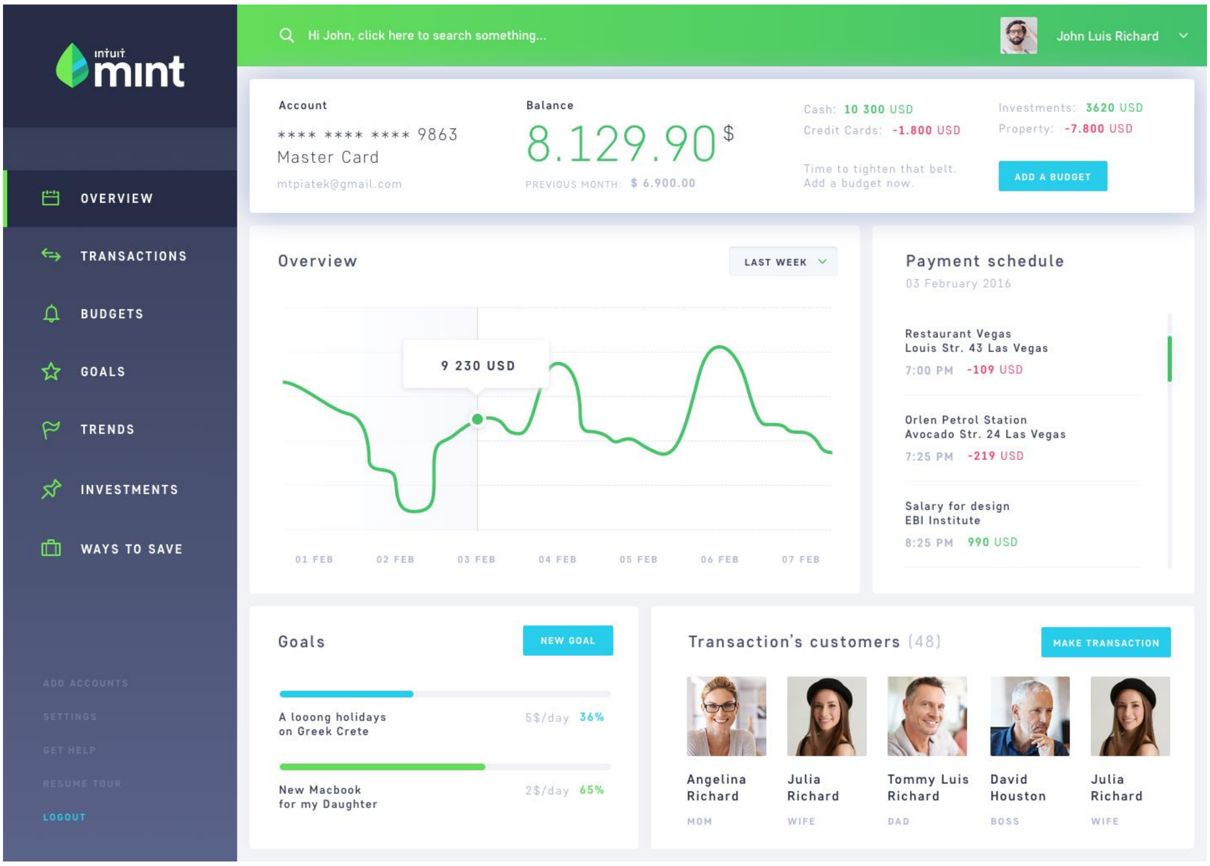
This app is free of charge and can provide tips tailored to the user based on their activity, making it an attractive product to potential users. It can show how much money you have across multiple accounts/cards at any given time. Although it cannot pay bills directly like other solutions, it does have a feature that will send alerts when bills are due or funds are low, meaning the user can re-prioritise spending.

Downfalls include this application requiring a connection to a bank, which although makes the application fluid and autogenerates the financial breakdown, some users may have an issue with or are unable to do this. This application also has geographical restrictions due to its banking connection reliance and is only available in US/Canada as it only has the ability to link with banks in these countries.

Platform(s): Web, IOS, Android, Windows Phone

Price: FREE

Rating (/10): 8



Account
**** **** **** 9863
Master Card
mtpiatek@gmail.com

Balance
8.129.90 $
PREVIOUS MONTH: $ 6.900.00

Cash: 10 300 USD
Credit Cards: -1.800 USD
Time to tighten that belt.
Add a budget now.

Investments: 3620 USD
Property: -7.800 USD

Overview

Payment schedule
03 February 2016

Restaurant Vegas
Louis Str. 43 Las Vegas
7:00 PM -109 USD

Orlen Petrol Station
Avocado Str. 24 Las Vegas
7:25 PM -219 USD

Salary for design
EBI Institute
8:25 PM 990 USD

Goals

A looong holidays on Greek Crete 5$/day 36%

New Macbook for my Daughter 2$/day 65%

Transaction's customers (48)

Angelina Richard	Julia Richard	Tommy Luis Richard	David Houston	Julia Richard
MOM	WIFE	DAD	BOSS	WIFE

Figure 4: Mint Web Application (Mint, n.d.)

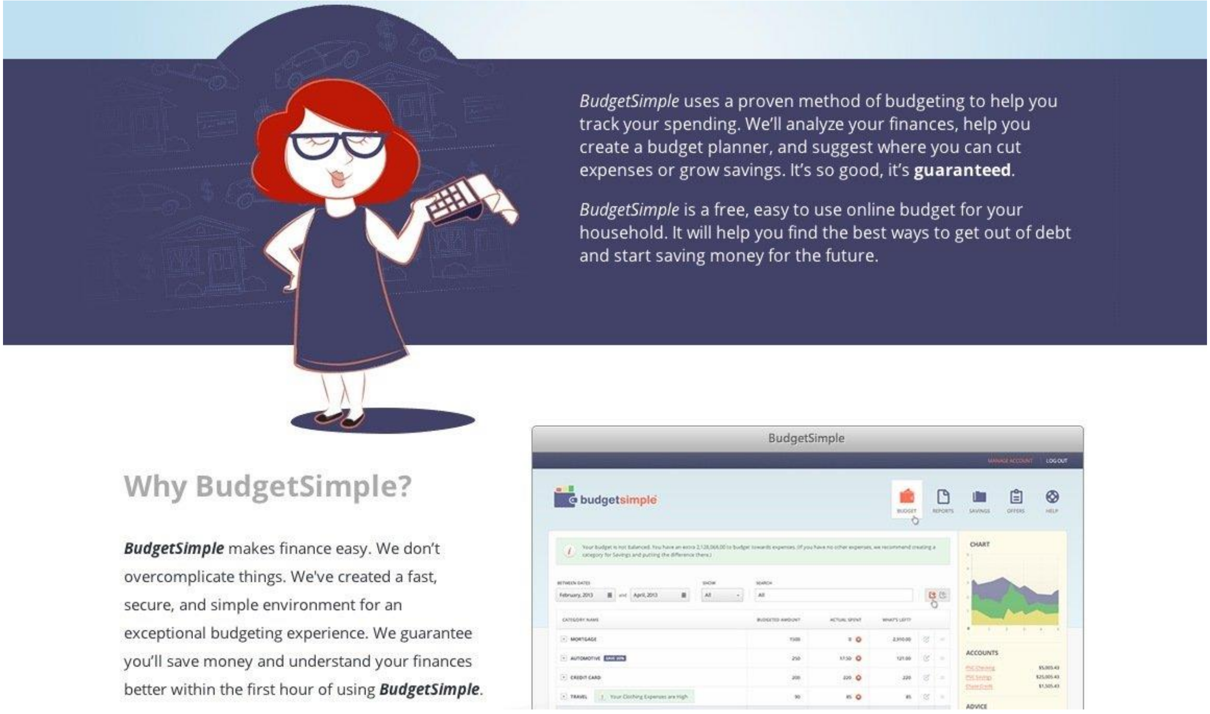
2.2.4 BudgetSimple

BudgetSimple is a web application that shows you where your money has been going, but also shows/suggests to you where your money should be going, dividing expenses into categories like gas, rent, car payments, movie nights. The application allows the user to add goals they can work towards such as a gym membership and equates that to trade-offs needed to meet your goals. The application stands by the motto that you can enjoy the things that matter to you while still working towards long term goals. BudgetSimple can also identify any potential expenses or monthly recurring payments that slip under the radar. Despite only being available as a web application and not yet having mobile implementations, this application is free to use and has an attractive and understandable UI that will not overwhelm novice users, which is something that will be looking to be incorporated for this project solution.

Platform(s): Web

Price: FREE

Rating (/10): 7



BudgetSimple uses a proven method of budgeting to help you track your spending. We'll analyze your finances, help you create a budget planner, and suggest where you can cut expenses or grow savings. It's so good, it's **guaranteed**.

BudgetSimple is a free, easy to use online budget for your household. It will help you find the best ways to get out of debt and start saving money for the future.

Why BudgetSimple?

BudgetSimple makes finance easy. We don't overcomplicate things. We've created a fast, secure, and simple environment for an exceptional budgeting experience. We guarantee you'll save money and understand your finances better within the first hour of using ***BudgetSimple***.

Figure 5: BudgetSimple Web Application (BudgetSimple, n.d.)

2.2.5 YNAB (You Need A Budget)

YNAB is by far one of the best options and financial planning applications available. This solution has a unique approach in improving the users' financial literacy whilst they use the application, providing tutorials on potentially daunting financial topics, whilst still maintaining the core features of budgeting and understanding expenditures. However, this application does not include any features for tracking investments like some of the other solutions. The company take pride in their top of the range solution, claiming that "on average, new budgeters save $600 in their first two months and more than $6,000 their first year." (You Need A Budget, 2020)

This app although not free of charge, does provide a 34-day free trial and offers two plans including an $11.99 monthly recurring subscription, or an annual plan costing $84 one-off payment. This app has implementations for web, iOS, Android and even Amazon Alexa. YNAB even has an API that developers can use for various integrations, which is a unique feature, setting it apart from its competitors. This is one avenue that could also be explored in future regarding implementation with this project. (You Need A Budget, 2018)

Platform(s): Web, iOS, Android, Amazon Alexa

Price: 34-day free trial, $11.99 p/month, $84 p/year

Rating (/10): 9

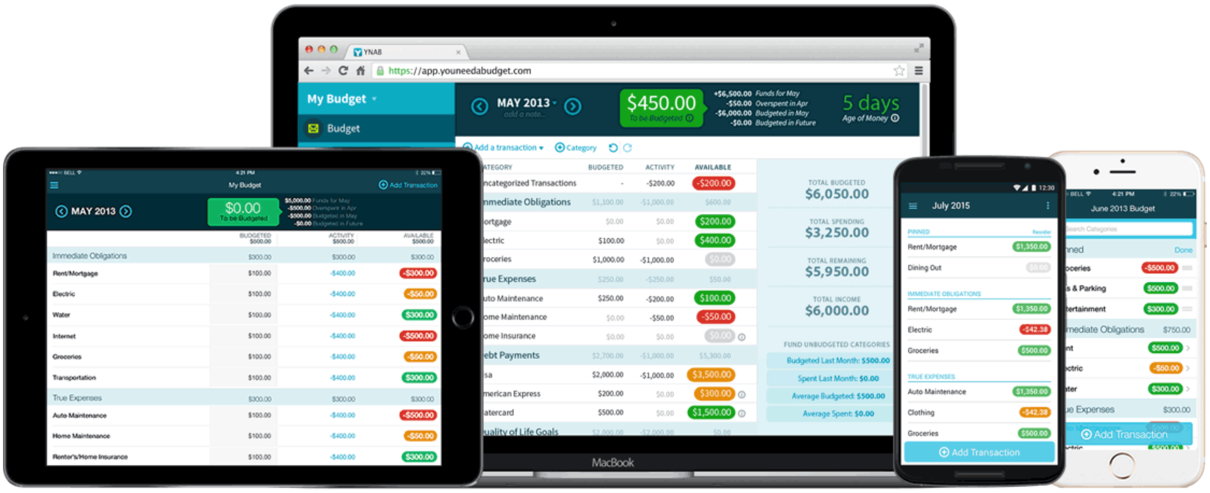


Figure 6: YNAB UI (You Need A Budget, 2020)

2.2.6 Prism

This solution is one that is more tailored to the finance niche of bill payment. It can be considered better than the other solutions in comparison at this, and due to the free price tag, is an attractive proposition to potential customers. This app has 11,000 billers which is far more than any other application of the same type (Prism, 2020). Upon adding your bills and payments to the application, you will automatically be sent reminders regarding payments, furthermore, payments can be made directly via the app. Despite having a user-friendly UI, catering from novice to expert, this solution is only available on mobile platforms meaning there is no web app implementation. Despite only being available on two mobile operating systems, iOS and Android have the biggest combined market share by far (as discussed below in section 2.3.1) and so this is less of an issue. This app could be further improved however by increased compatibility and the incorporation of a web application.

Platform(s): iOS, Android

Price: FREE

Rating (/10): 7

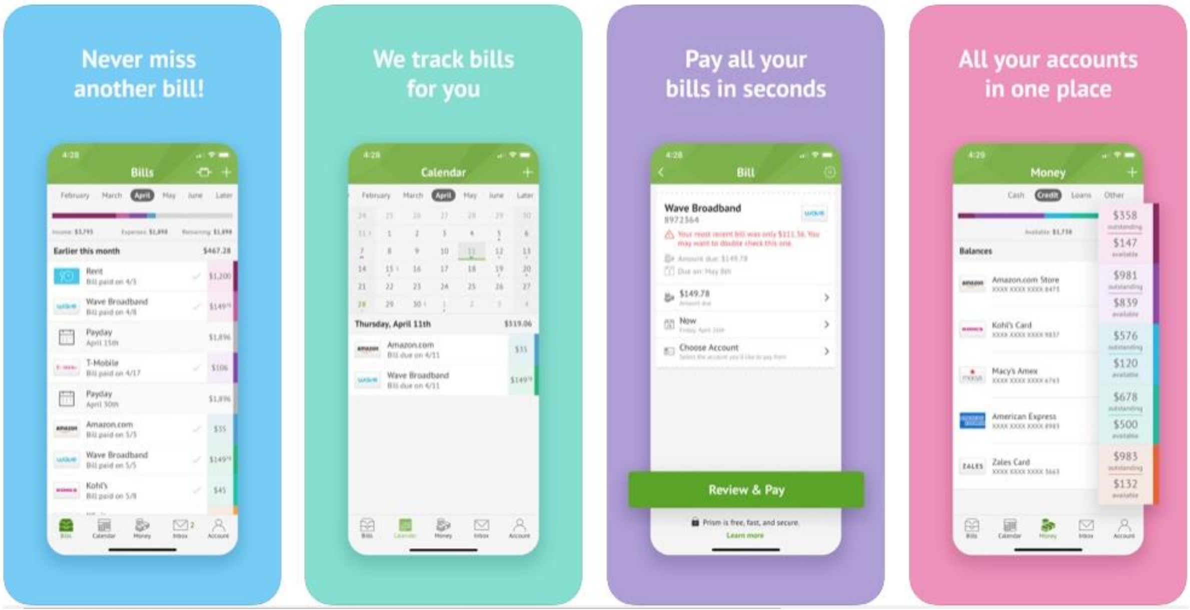


Figure 7: Prism Mobile Application (Mobilligy, 2020)

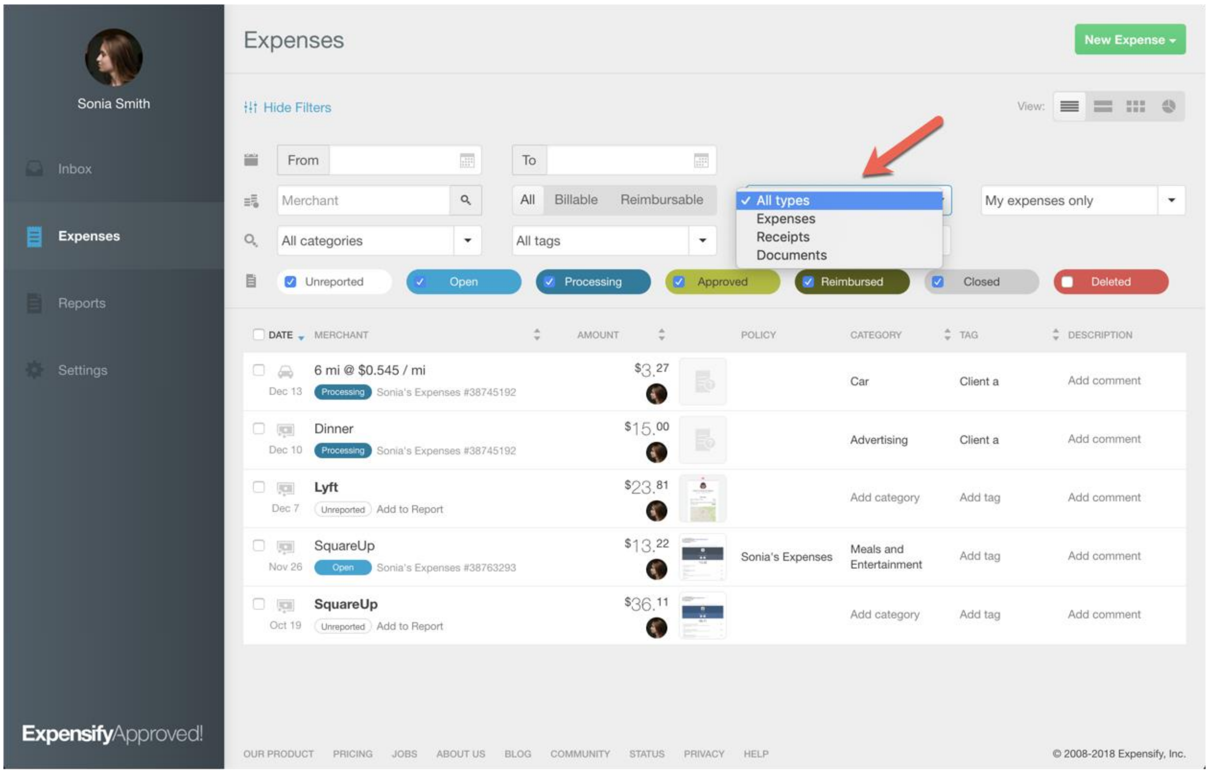
2.2.7 Expensify

This application is one which is more appropriate and suited for a business traveller individual, providing easy creation of expense reports, and methods for adding logs and receipt photos. This app offers a free trial and a free basic option, however if you require some of the more advanced features then a premium subscription is available for $4.99 a month. Expensify has a clean UI, with the side nav bar on the left of the screen seen in figure 8 being a classy touch to segregate complicated information into appropriate categories, and thus limiting potential overwhelming of users.

Platform(s): Web, Android, IOS, Windows phone, Blackberry

Price: Free Trial, Free Basic Version, $4.99 p/month full version

Rating (/10): 7



☑ Unreported ☑ Open ☑ Processing ☑ Approved ☑ Reimbursed ☑ Closed ☐ Deleted

DATE	MERCHANT	AMOUNT	POLICY	CATEGORY	TAG	DESCRIPTION
Dec 13	6 mi @ $0.545 / mi Processing Sonia's Expenses #38745192	$3.27		Car	Client a	Add comment
Dec 10	Dinner Processing Sonia's Expenses #38745192	$15.00		Advertising	Client a	Add comment
Dec 7	Lyft Unreported Add to Report	$23.81		Add category	Add tag	Add comment
Nov 26	SquareUp Open Sonia's Expenses #38763293	$13.22	Sonia's Expenses	Meals and Entertainment	Add tag	Add comment
Oct 19	SquareUp Unreported Add to Report	$36.11		Add category	Add tag	Add comment

Figure 8: Expensify Web Application (Expensify, 2018)

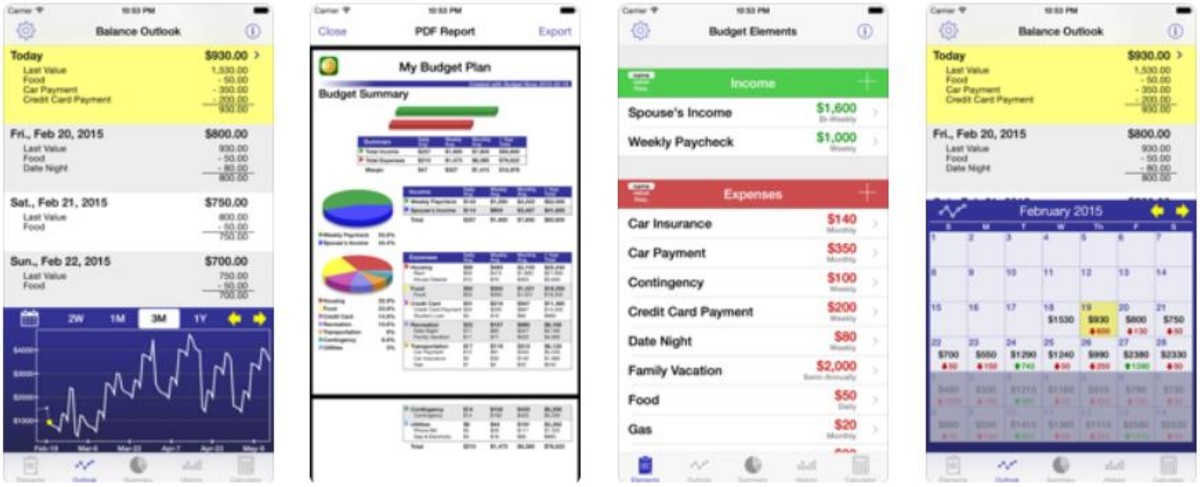
2.2.8 Budget Boss

This app despite being available on iOS only and furthermore not having the most aesthetically pleasing UI, has its own unique twist as it learns your spending habits from the data inputted. This can then be used for predicting future spending and providing a financial forecast for a certain point in time. From this, you can then adjust your spending to adapt this how you see fit. This app is in general limited and restricted in the number of customers who would be inclined to use it due to its poorly designed and unattractive UI. Furthermore, despite having some advanced features, as this solution is only available on iOS and only available in the United States app store, this makes it a low-quality application.

Platform(s): iOS

Price: $0.99

Rating (/10): 6



Balance Outlook

Today $930.00
Last Value 1,530.00
Food - 50.00
Car Payment - 350.00
Credit Card Payment - 200.00
930.00

Fri., Feb 20, 2015 $800.00
Last Value 930.00
Food - 50.00
Date Night - 80.00
800.00

Sat., Feb 21, 2015 $750.00
Last Value 800.00
Food - 50.00
750.00

Sun., Feb 22, 2015 $700.00
Last Value 750.00
Food - 50.00
700.00

Budget Elements	
Spouse's Income	$1,600
Weekly Paycheck	$1,000
Car Insurance	$140
Car Payment	$350
Contingency	$100
Credit Card Payment	$200
Date Night	$80
Family Vacation	$2,000
Food	$50
Gas	$20

Figure 9: BudgetBoss Mobile Application (Damian Toohey, 2017)

2.2.9 Overall Solution Review

This table shows an overall breakdown of the existing solutions that have been analysed.

Name	Platform(s)	USP / Breakdown	Price	Rating (/10)
Toshl	Web, Android, iOS, Windows Phone	High quality UI. Multiple currency handling	FREE	9
Quicken	Windows, MacOS, iOS, Android	Best overall solution, app integration/compatibility	$34.99 basic plan	7
Mint	Web, iOS, Android, Windows Phone	More popular solution, requires bank connection	FREE	8
BudgetSimple	Web	Simple breakdown of financial activity/no overcomplication	FREE	7
YNAB	Web, iOS, Android, Amazon Alexa	Platform versatility. API for further integration	34-day free trial $11.99 p/month Or $84 p/year	9
Prism	iOS, Android	Best solution for bill payment	FREE	7
Expensify	Web, iOS, Android, Windows Phone, Blackberry	Best solution for managing expenses	Free Trial Free Basic Version Or $4.99 p/month full version	7
BudgetBoss	iOS	Financial forecasting, Low quality UI	£0.82 (offers in-app purchases)	6

Some features of importance taken from this analysis include how vital an attractive but non-overwhelming UI is. Furthermore, the ability for an application to be highly accessible and available on a number of devices is something which should be considered.

2.3 Available Technologies

This section is a breakdown of the application types, what platform the application will be developed for, and the various tools that will be available for use when developing the solution.

2.3.1 Potential Application Types

As seen above in the 2.2 existing solutions section, there are 3 main types of implementations in the form of **Web**, **Desktop** (MacOS, Windows) and **Mobile** (iOS, Android, Windows Phone). There are pros and cons to be considered with the development of each.

Web

A web application is one which can potentially be both device compatible and OS compatible as it is not native to a system. A web app is responsive in nature and will adapt to the device that it is being viewed on i.e. a desktop or a mobile device, and hence can often be very similar in nature to a mobile application when viewed on a mobile device. These applications are viewed and accessed via the internet browser and do not require to be installed on the user device, and do not need app store approval before being launched. Web apps are also centrally updated and easily maintained, meaning that changes only need to be made at one point e.g. following a bug fix. This type of application also has the advantage of being significantly easier and quicker to develop and make available in comparison to mobile applications.

Despite the numerous advantages of a web app implementation, there are still some drawbacks that must be considered. For example, as it does not involve being downloaded on the device, it relies fully on internet connection to run, and hence costs more on bandwidth. For application discovery, this cannot rival the mobile platform, where there are dedicated app stores (Apple App store & google play store) which hold a database containing all the applications available for download to the user.

It can also be noted regarding accessibility of a web application, with a study being performed in 2019 by the International Telecommunication Union (ITU) that an estimated 51.2% of the global population had internet connectivity (Itu.int, 2019).

Desktop

A desktop application (i.e. software) can be designed with an operating system in mind, the main three being Windows, MacOS and Linux (StatCounter, 2020). The application would have to be specifically designed for these systems and thus would not be compatible with mobile OS's. This would hence restrict the availability of the application significantly. Despite this, the desktop platform does have some advantages, such as it being more appropriate for more complex tasks/features, with further resources available in general. This platform can be portable i.e. laptop or desktop tablet, however, does not match the levels of portability that comes with mobile devices. Regarding bugs and fixes, the software would have already been deployed, hence would require an update and re-deployment to all devices with the software already installed. This type of application would also not face reliance on internet connectivity to the same level of a web app and likely will be able to function offline.

<u>Mobile</u>

This platform has a far greater portability than desktop applications due to the smaller nature of the devices, and gives you the ability to work offline, performing faster than a web application. Most people nowadays are in possession of a smartphone due to the availability and low prices of the broad selection of mobile devices available to us. Downsides to this would include the application requiring a different implementation for each mobile OS, having to completely re-design and develop the app. Bug fixes on mobile, similar to desktop, requires updates/redeployment to all devices, however mobile devices can be formatted so that the user auto-updates an application upon having internet connection. These devices can also lower processing capabilities and may not deal with complex tasks/displaying in depth information in a suitable way as well as on desktop. There is also potential for cross device compatibility issues. Mobile applications also require app store approval which can be long and expensive compared to web apps.

The two mobile OS's with the majority combined market share available in today's market include Apple's iOS and Google's Android. Regarding the UK mobile OS market, at the time of writing (April 2020), these two combined have a 99.77% market share, with iOS (52.29%) slightly edging Android (47.48%). The next closest competitor is Samsung with 0.1%, with Windows (0.08%), BlackBerry OS (0.02%) and Unknown (0.01%) making up the rest. These landslide statistics can be visualised in the following graph from StatCounter.

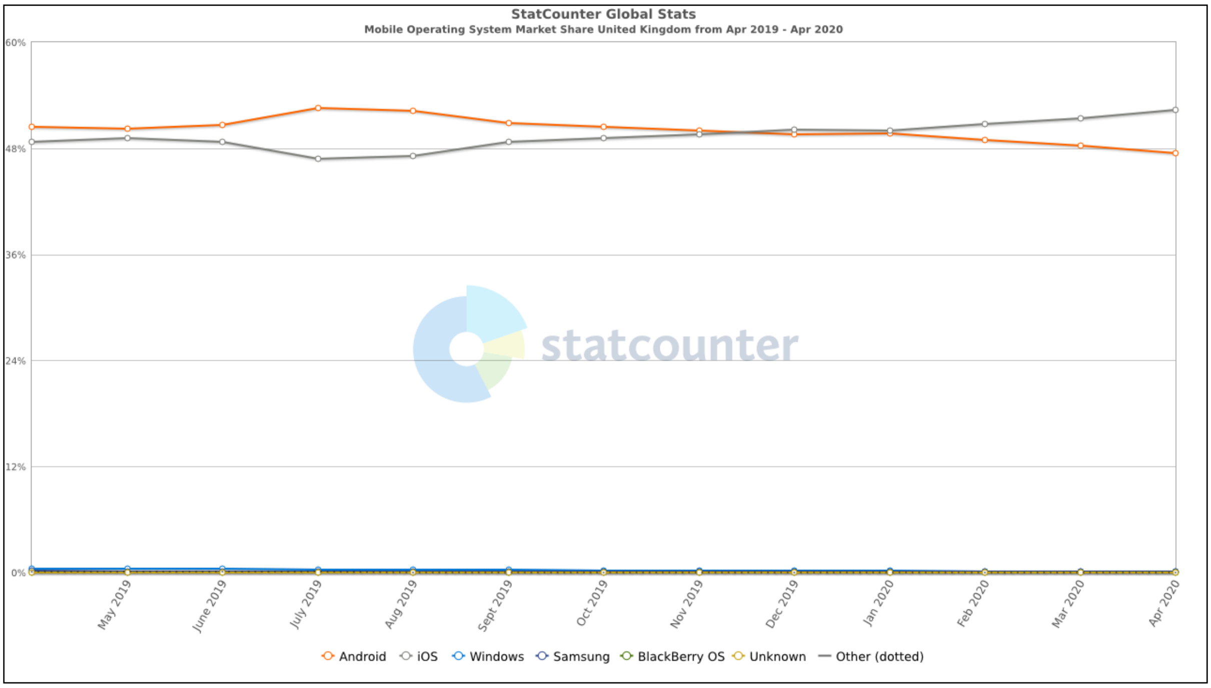


Figure 10: UK Mobile OS Market Share (StatCounter, 2020)

The global market share of mobile OS's however is a completely different picture. At the time of writing, iOS only holds 28.79% compared to Android having a substantial market share at 70.68%. The rest of the market is filled in a similar way to the UK equivalent, with Samsung having 0.17%, Windows having 0.07% and Unknown being 0.09%. The only noticeable other difference here is

KaiOS holding a 0.12% market share in the global market. A visualisation of these figures can be seen below with a graph published by Statcounter.

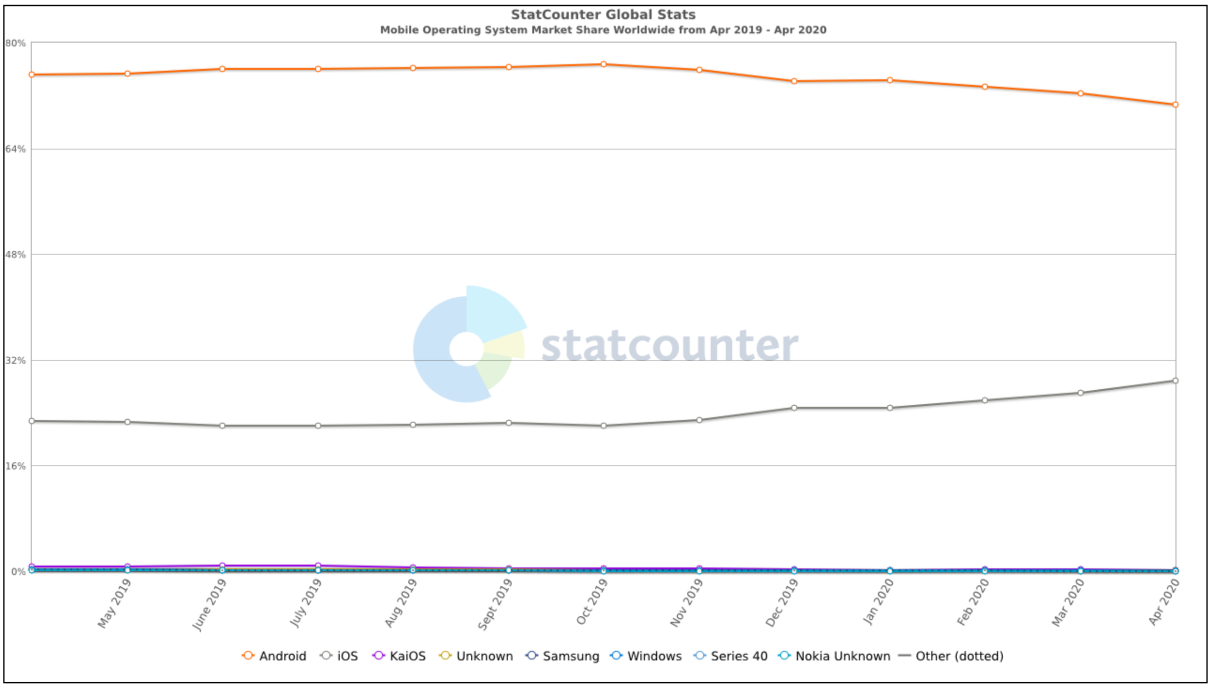


Figure 11: Worldwide Mobile OS Market Share (StatCounter, 2020)

Conclusion

In conclusion to the choice of application type decided for this project, there is no clear outright general best option and it depends on the actual application operations and how it will function to determine the most appropriate type. This is why it would be most suitable to develop a web application for this project as discussed below.

Although there is a strong argument for implementation of a mobile application because of the heavy everyday usage of smartphones, especially an Android implementation due to its vast market share as demonstrated above. It would be more suitable to implement a web application due to the following reasons, including its ability to be responsive and essentially act like a mobile application, with the only difference being its reliance on internet connection.

The web application implementation will be better for displaying more complex information and graphs to give a good financial visualisation to the user. This may potentially be slower than a mobile application but will be more appropriate for displaying more complex information. Its reliance on internet connection and bandwidth for a better display usability and responsiveness is considered a worthy trade off. This is because the application being created here is not considered to be fully reliant on speed of performance to judge its success (provided it does not damage the integrity of the product and ruin user experience).

The use of a web app may also be better for future improvements of the application such as bank integration and auto-generation of the financial breakdown – similar to that used in some of the previous existing applications, one example being 'Mint' (See section 2.2.3).

A web app will provide the greatest availability as it is not native to any particular system, being accessible on any device with a web browser and internet access. We know from the Office of National Statistics that around 99% of adults (age 16-44) in the UK have access to and are using the internet regularly (Prescott, C., 2019). This application type has the lowest overhead for testing and can be updated and deployed far quicker and easier than mobile and desktop applications. Any updates or bug fixes can easily be rolled out due to the centralised nature of a web application. It is for these reasons that this type of application is the most appropriate for development with this project.

2.3.2 Technical Development

As the decision was made to implement a Web application rather than a desktop or mobile application for this project, the options must be explored for how this could be developed. There are many web application frameworks that could potentially be used for the development of this project, making the application easier to maintain and scale. One popular example at the time of writing is ASP.NET, which can be used with any common language runtime, most likely C#. This framework can be used to produce dynamic web pages, apps, and services.

ASP, which stands for Active Server Pages (Microsoft 2014), is also directly in competition with some other server-side web technologies including Laravel, Node.JS, Django, Ruby on Rails and Cold Fusion (Mozilla, 2019).

The following table depicts the frameworks available for development of this web application, the language that they use, and the experience of using each option.

Web (Application) Framework	Developed using	Experience
ASP.NET	Any Common Language Runtime e.g. C#	Good
Laravel	PHP	Limited
Node.JS	JavaScript	Good
Django	Python	Good
Ruby on Rails	Ruby	Limited
Cold Fusion	ColdFusion Markup Language (CFML)	Limited

As seen above, it would be sensible to use one of the options with good prior knowledge on, and ASP.NET seems the most appropriate given the authors extensive experience with C#, and the frameworks versatility and options it provides.

2.3.3 System Structure

Given the choice of using ASP.NET, there are many different development models that can be used including:

- **Web Pages** – A single page model, like classic ASP and PHP.
- **MVC** – The Model View Controller pattern is a design pattern that allows the decoupling of the model (data) layer, the controller (commands) layer, and the view (output) layer.
- **Web Forms** – An event driven application model, creation of dynamic websites using drag and drop, with a large library of controls.
- **Web API** – Similar to MVC but without the view i.e. the API returns only data and does not provide a user interface.

(w3schools, n.d.)

Due to the nature of this project and the human interaction required with the application in question, it is most appropriate to use the MVC development model which can be visualised in figure 12 below. This architectural pattern separates the application into the three components which work together to display information via methods working on data to the user.

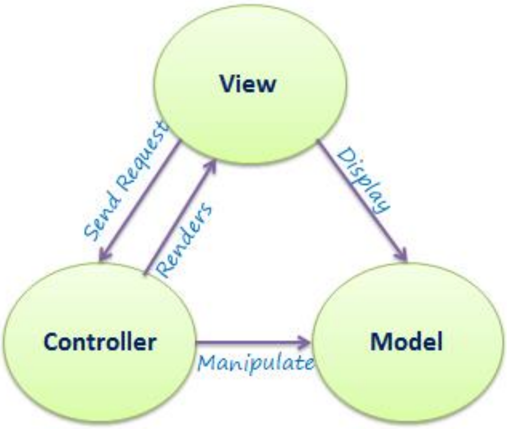


Figure 12: MVC Architecture (TutorialsTeacher, n.d.)

Regarding the MVC components, the breakdown of each is as follows:

- **Model** – This layer represents the format of the data i.e. the classes/entities that exist and the relationship between them.
- **View** – This is the user interface and is the way in which the information (model data) is displayed to the user. The view is usually created in ASP.NET MVC via the use of HTML, CSS and syntax.
- **Controller** – This can essentially be considered as the methods which handles the user request and performs functions on the model in order to produce the view that the user sees.

(TutorialsTeacher, n.d.)

2.3.4 User Interface (UI)

Given the decision to create this application in the form of an ASP.NET MVC Web App, this requires the design and creation of a user interface for client interaction. With this application, a good knowledge of HTML (Hypertext Markup Language), CSS (Cascading Style Sheets) and JavaScript is essential.

One option that is available for facilitating an easy UI design and creation is the use of Bootstrap (Bootstrap, 2019). This is an open source front-end framework that includes CSS and JavaScript which can easily be combined with html to format responsive and mobile friendly, browser friendly web pages. This framework has full support for Chrome, Firefox, Opera and Microsoft Edge on Android, iOS and Windows 10 Mobile devices and Mac and Windows desktop. This also has support for Safari, Android Browser and IE where applicable (Bootstrap, n.d.).

Bootstrap can create web pages that remain consistent and have a uniform layout site wide and regardless of the browser/device being used. Bootstrap defines CSS mostly for use with layout, divisions, and tables, with various JavaScript components being available for use. The way in which

Bootstrap works is that the developer can choose either a fluid container which always fills the width of the page, or a fixed container using one of four pre-defined widths dependant on the size of screen that the site is being viewed on. Bootstrap uses a 12 column grid division system to organize layouts, with divisions being displayed side by side on a screen that is large enough, or collapse to display underneath the previous division on a device with a smaller viewing screen e.g. a mobile device.

The views in an ASP.NET MVC application come in the form of .cshtml files. Although these files contain the base HTML, CSS and JS used for a web page and UI design, these views can also include the use of C# via the incorporation of Razor mark-up. The incorporation of server-based code into the web pages can be easily facilitated by prefixing any C# code in the view with the @ symbol. This will cause the following code segment to be evaluated and rendered in the html output.

(Microsoft, 2020)

2.3.5 Data Storage and Integration

The use of Entity Framework (Anderson & Dykstra 2014) and a Code First, with the creation of model classes first will result in EF creating the database for us based on the entities we provide and the relationship between them. The use of EF allows us to communicate with database as though it were C# objects, making it easy to incorporate seamless interaction between the application and the database.

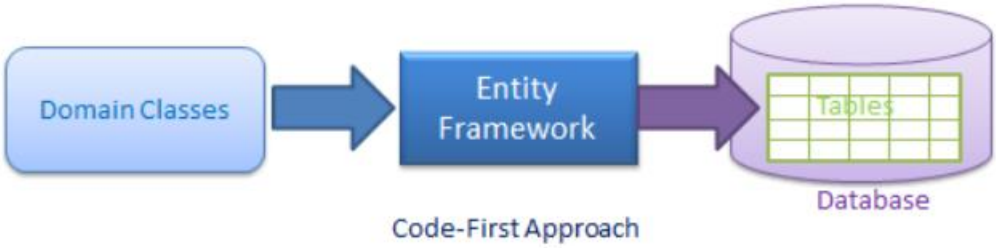


Figure 13: Entity Framework Code-First (Entity Framework Tutorial, n.d.)

2.3.6 Hosting

Regarding hosting of the web application, there are essentially two options that can be considered, consisting of self-hosting and cloud hosting (Molnar & Schechter 2010). Both these options have pros and cons and the most suitable option depends on the project itself that needs hosting.

Self-hosting

This option usually offers the user far more control in comparison to cloud-hosting. This option is more appropriate for organisations who deal with sensitive information (Pearson, 2013) and for potentially larger organisations. The information therefore would be held in-house however, meaning that the organisation performing the hosting would themselves be accountable for aspects of maintenance and security and furthermore would be responsible for uptime of the server.

Cloud-hosting

This option of hosting offers far less control and customisability than self-hosting. However, in the majority of cases can be a far more attractive proposition. Cloud hosting would be appropriate for organisations/applications that do not necessarily hold sensitive information. This would also be a better option for smaller organisations, especially as they would not be responsible for uptime and maintenance. Security is also a huge issue to be considered with cloud-hosting, the argument being that it is "inherently less secure than the self-hosted infrastructure" and is not "appropriate for high stakes applications such as health care" (Qureshi, A., 2017).

Options

A few potential options for hosting consist of the application being hosted via Windows Server, Microsoft SQL Server or Microsoft IIS.

The solution that is likely to be used for hosting is Microsoft Azure. Due to the integration with Visual Studio, the publishing to this platform is made easy provided the user has a Microsoft Account (which provides a free Azure hosting). This process is relatively straightforward, with an abundance of documentation and instructions for dealing with this.

2.4 Ethical Issues

There are certainly a number of ethical issues to consider with the development of a financial planning application. As the application will involve the storage of user transaction information, the solution must comply with the General Data Protection Regulation (GDPR). This requires the application to only keep the necessary information attached to a user, and that all personal data is processed securely with 'appropriate technical and organisational measures' (GDPR, 2019).

The development of this application must also be compliant with the British Computer Society's (BCS) Code of Conduct. In this, it is stated that 'due regard for public health, privacy, security and wellbeing of others and the environment' is essential. (Cdn.bcs.org, 2019).

2.5 Security

As this application will be dealing with user login details, which may be duplicated for other existing web apps or social media, it is important that the password is not easily visible to anyone who can view the database.

One simple method that can be used to massively increase the security of this application would be hashing the passwords before saving them in the database, so that they are not saved in the original plaintext form. Hashing involves an 'approach for maintaining a users' password-related information that is later used for authentication'. (Hatzivasilis, G. et al., 2015).

This will on one hand lead to a potentially slower application i.e. more processing time when dealing with passwords and checking against the database, however this would massively increase the programs resistance against brute force attacks, and it is even possible to detect password-cracking attempts. (Juels & Rivest, 2013).

2.6 Conclusion

Upon evaluation of existing solutions and existing technologies that are at the expense of a developer, the conclusion has been made to create a ASP.NET MVC Web application using Entity Framework and Azure. This option has been chosen as this is the most accessible option and caters for a large market, including mobile devices using the internet browser, where it could effectively act like a mobile application. The type of application being developed here is more appropriate to a desktop web app as the information will be able to be displayed easier, with incorporation of graphs and tables.

Features that are likely to be included from analysis of previous examples include the use of account management so that implementation on other devices may be possible in the future. The application must contain a simple and attractive UI with suitable visualisation of data that will provide the user and in-depth analysis of their income/expenditures, whilst not being overwhelming to the novice user.

One perspective that does not currently exist in the examples analysed is the incorporation of family budgeting. A suggestion for a system like this could include users being able to join a family plan, with the admin in said group being able to view the spending of all users. Of course, this would have ethical considerations with what data is accessible and would be hard to implement with real time banking data.

3 Requirements

Regarding the requirements, these are very similar in comparison to the PID's objectives, with the slight adjustment of being less focused on a mobile application implementation and more on a responsive and mobile friendly web application with emphasis on database integration.

3.1 Product Requirements

The product requirements for this software can be broken down into Functional Requirements and Non-Functional Requirements. The former covers the main functionality of the software and can usually be well portrayed by use cases. The latter includes requirements that are not directly linked to the functionality of the software, consisting of things such as reliability, scalability, security and integration. (Pathak et al., 2006)

3.2 Functional Requirements

1. Add Account
   - As a user, I must be able to register an account which will be created if all required account information is provided. This account should be given the normal user role (As opposed to admin).
2. Add Income/Expenditure (Transaction)
   - As a logged in user, I must be able add a new transaction in the form of an income or an expenditure, which will be created if the required information is provided, otherwise displaying relevant error messages.
3. View Transactions
   - As a logged in user, I must be able to see an overview of all the transactions that are associated with my account in an easy to navigate and attractive UI table. This table should include all the transaction variables and appropriate information.
4. Transaction Category breakdown
   - As a logged in user, I must be able to select a specific transaction category to see a further breakdown of their income/expenditure respective to this specific category.
5. Transaction Visualisation Breakdown
   - As a logged in user, I must be able to have a breakdown of all my transactions into a form of graph/chart visualisation.
6. Edit Transactions
   - As a logged in user, I must be able to select a transaction from their list of transactions and edit and save back this new information.
7. Admin Functions
   - As a user with Admin role, I should be able to view all users and all the respective transactions for each user.
   - As a user with Admin role, I should have the ability to delete user accounts (Admin account cannot be deleted).
   - As a user without Admin role, I must not have access to the admin functions, and any attempt to access these when not logged in or only logged in as a non-admin user should result in a redirect to the login page.

3.3 Non-Functional Requirements

1. Database Storage

- The user account information and transactions should be appropriately stored in a database which can be accessed on demand to retrieve required information.

2. Responsive layout
   - Web app should be responsive to the device/browser being used to access the app, and provide a site-wide uniform layout regardless of the screen size the application is being viewed on.
3. Concurrency handling
   - The application should use optimistic concurrency handling to ensure that the database does not enter an inconsistent state upon two instances editing the same transaction at once, and provide appropriate error messages for this.
4. Security
   - The password for a user account should not be stored in the database as plain text as this could lead to a number of security issues. The password should be hashed before being stored in the database as an unidentifiable sequence of characters.
5. GDPR
   - The application should require only the necessary information and should handle it appropriately, and comply with GDPR regulations regarding data handling and storage. The GDPR requires that personal data is stored and processed securely by means of ‘appropriate technical and organisational measures’ (GDPR, 2019).

4 Design

4.1 Software Structure

The initial design concept of the application involved the use of separate forms/views which contained buttons that would link to different views which in turn provided further actions/displays. Below in figure 14 can be seen an early whiteboard prototype sketch of this mentioned format.

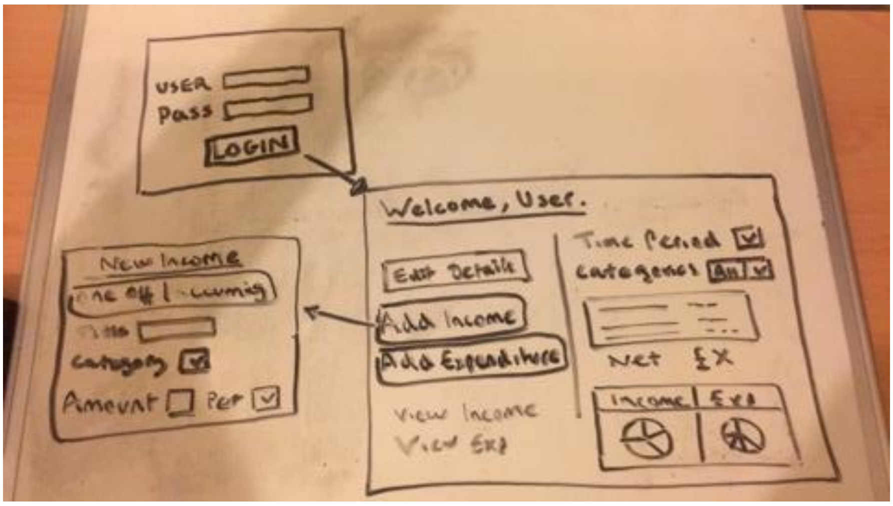


Figure 14: Early stages design concept

This shows the start-up view as the login screen, and upon successful login, the user would be displayed a view that can be considered a home screen, with a customised welcome message on the top header of the page. If a user with the name Owen was logged in for example, it would retrieve the user's name or perhaps email so that it displayed "Welcome Owen" or a similar variation of this. This concept provides buttons for new views to edit account details, add incomes and expenditures and also view incomes and expenditures. This home screen also contains a brief overview of the incomes vs expenditures, with the ability to sort via a time period or categories e.g. to see how much you spent on groceries.

The projected classes that would be involved in this application can be seen in figure 15 and include a UserAccount which would include variables to be displayed on various views, an *ICollection* of the Transaction class, having a 1 UserAccount to 0 to Many Transactions.

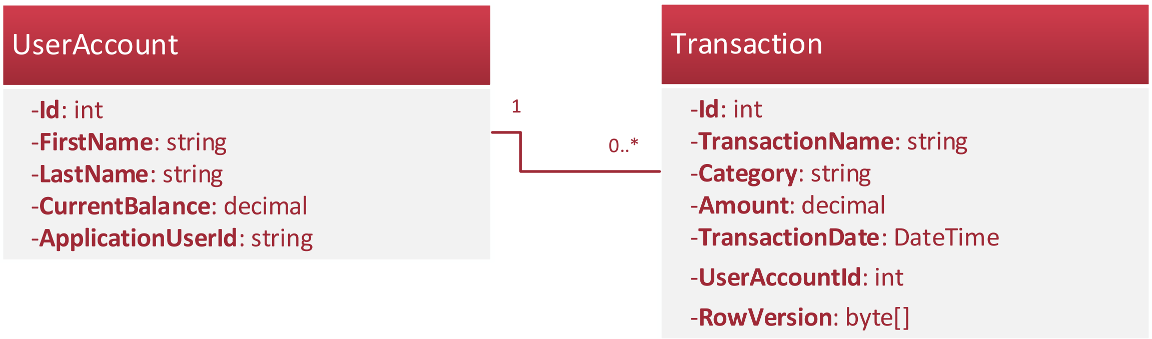


Figure 15: Application class entities & Variables

The UserAccount member variable '*ApplicationUserId*' acts as a foreign key to the *ApplicationUser* table.

The Transaction class contains various features of an income or expenditure transaction, with a *UserAccountId* to link the Transaction to a respective virtual UserAccount. This also contains a RowVersion variable which is decorated with the Timestamp data annotation in order to facilitate optimistic concurrency handling which is discussed in the implementation section of this report.

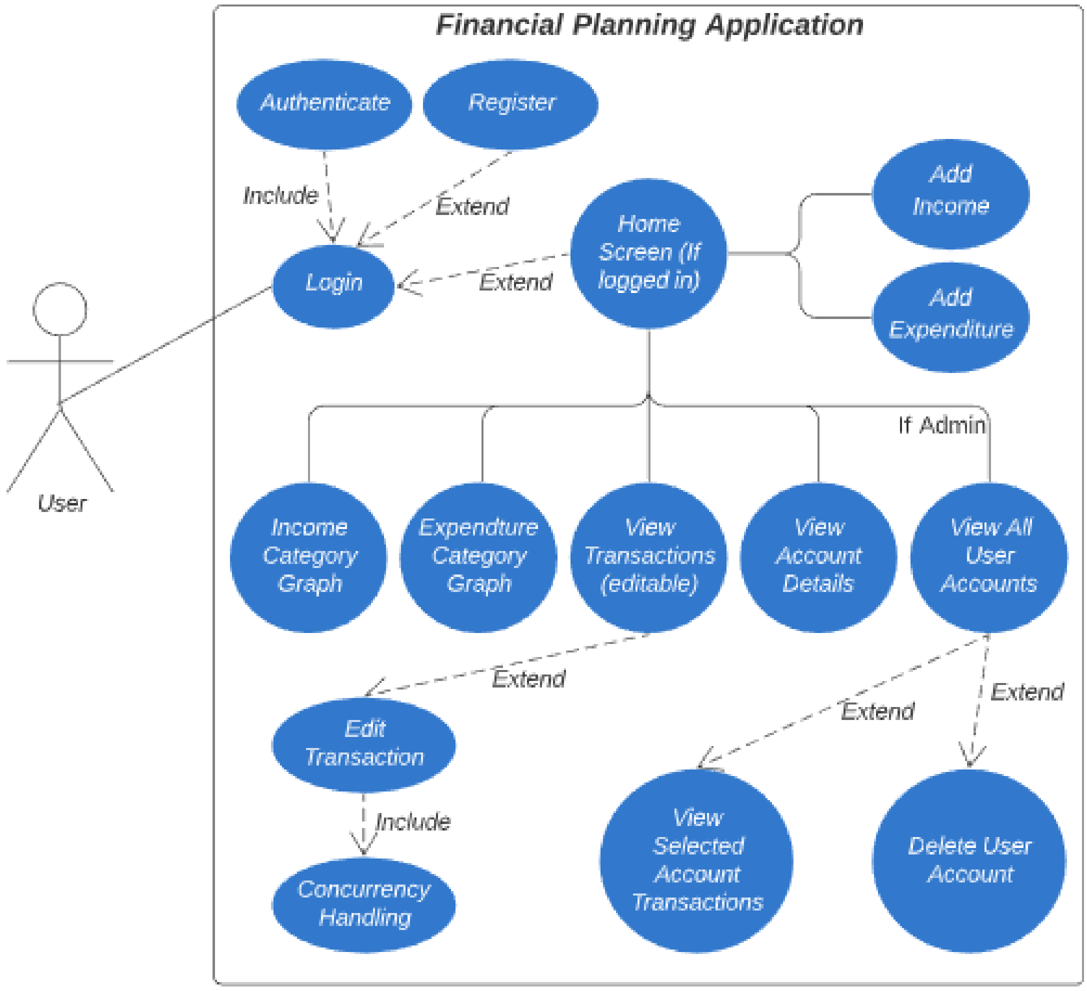
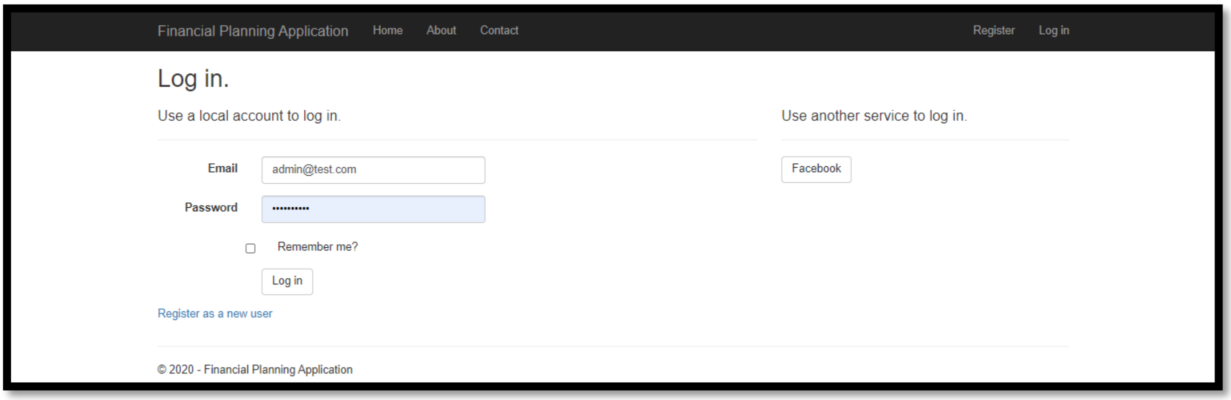


Figure 16: Use case diagram



Log in.

Use a local account to log in.

Email admin@test.com

Password ••••••••••

□ Remember me?

Log in

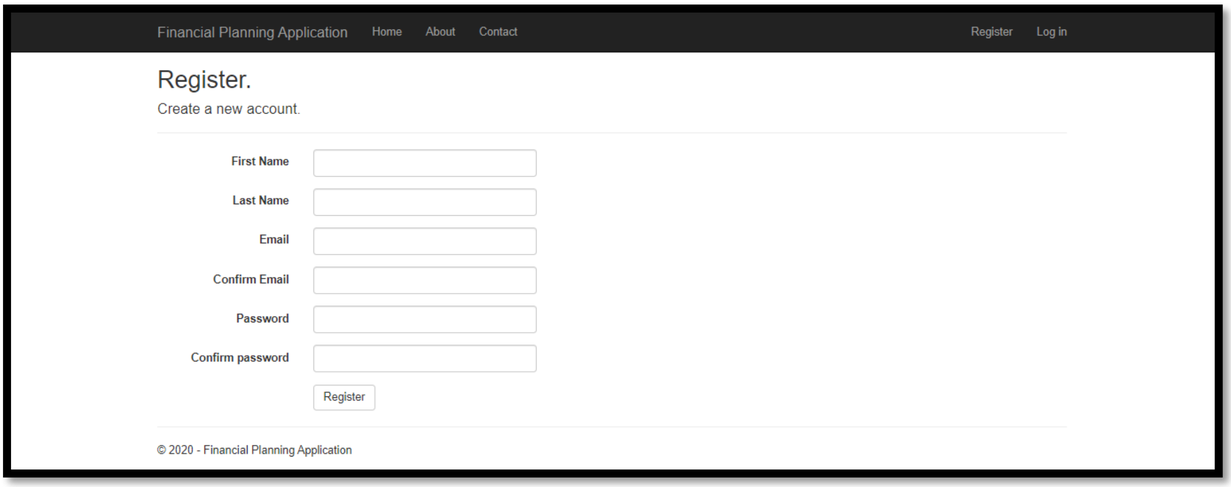
Register as a new user

Use another service to log in.

Facebook

© 2020 - Financial Planning Application

Figure 17: Application login view

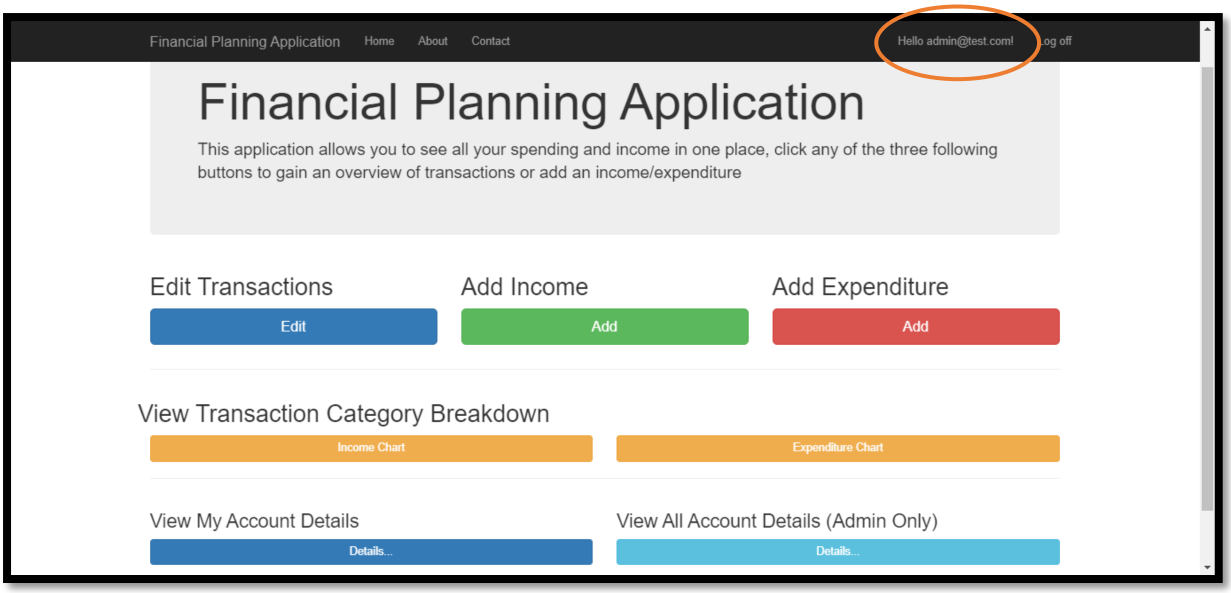


Register.

Create a new account.

© 2020 - Financial Planning Application

Figure 18: Application Register view



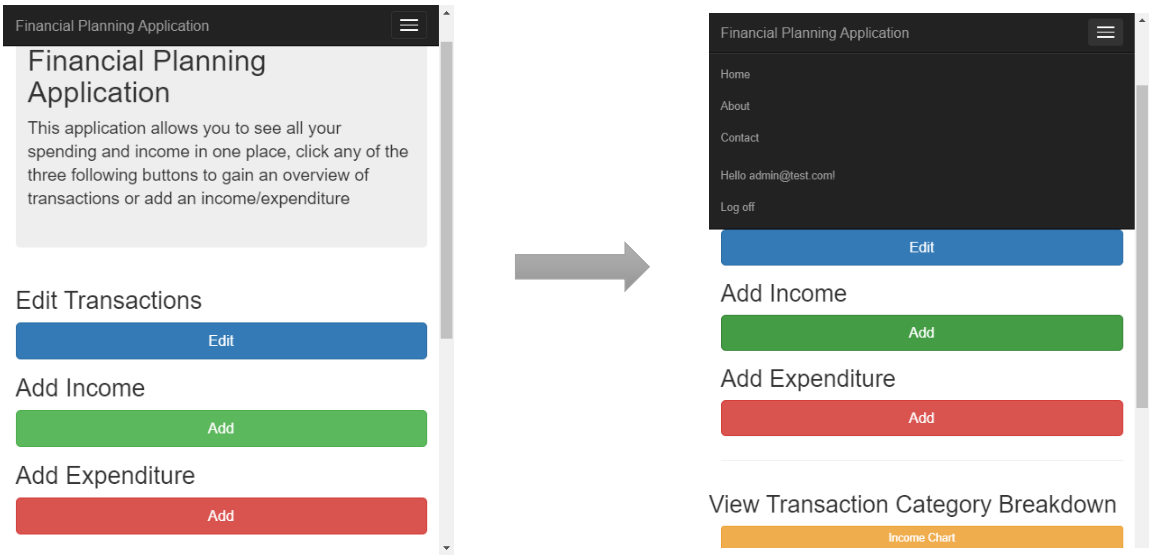
Financial Planning Application

This application allows you to see all your spending and income in one place, click any of the three following buttons to gain an overview of transactions or add an income/expenditure

Figure 19: Application home page

Figure 19 above showcases the final UI design of the home screen, highlighted is the responsive partial view top header nav bar which displays the welcome message “Hello admin@test.com!”, which was the user currently logged in at the time. This screen is only available to a logged in user, and any attempt to access it will result in the user being redirected back to the login screen seen in figure 17, or alternatively they can gain access by registering as a user, as seen in figure 18.

The responsive nature of the UI via the use of the mobile friendly Bootstrap grid system can be seen below in figure 20, with a resized small window causing the various buttons previously side by side on the above screenshot, now collapsed to be above and below. The nav bar has also been changed so that the links appear via a drop-down list instead of all displayed directly on the nav bar side by side.



Financial Planning Application

Financial Planning Application

This application allows you to see all your spending and income in one place, click any of the three following buttons to gain an overview of transactions or add an income/expenditure

Edit Transactions

Edit

Add Income

Add

Add Expenditure

Add

Financial Planning Application

Home

About

Contact

Hello admin@test.com!

Log off

Edit

Add Income

Add

Add Expenditure

Add

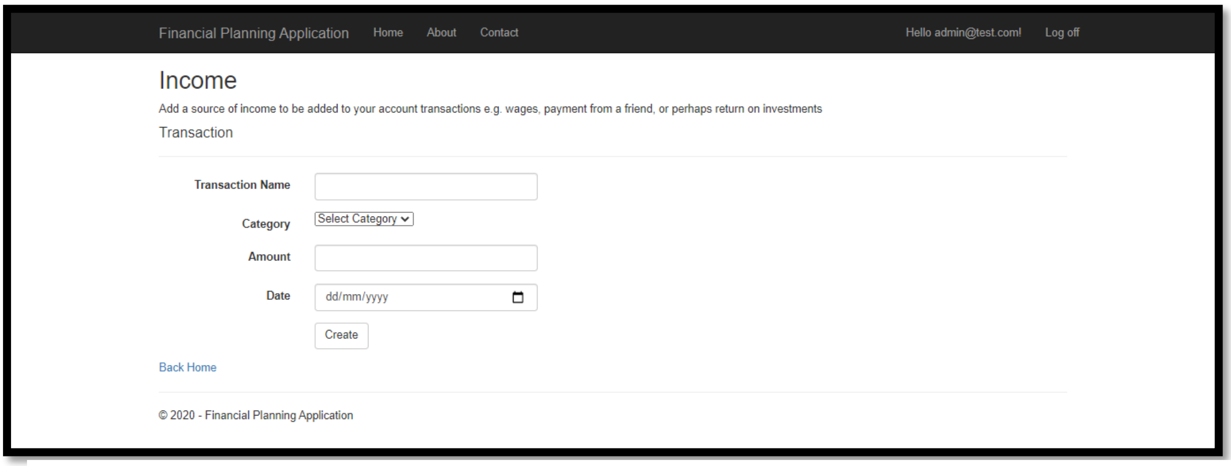
View Transaction Category Breakdown

Income Chart

Figure 20: Application Responsive layout

There are various Bootstrap themes that are freely available from sites like Bootswatch (Bootswatch, n.d.). These can be used as a foundation for the style and colour of things such as buttons, nav bars and tables, and can be implemented by replacing the original bootstrap.css file in the project with that of the chosen theme. However, the original theme that comes loaded with the ASP.NET MVC web app is suitable in this case, and so the incorporation of a new themes is not needed.

Upon clicking the button to add a new income, a view is loaded in the form of a Transaction create template. The Expenditure form is identical to this, with the one difference being the Category drop down containing different options for the user to select. Upon successful creation of a new transaction, this will be added to the database and attached to the instance of the current user logged in.



Financial Planning Application Home About Contact Hello admin@test.com! Log off

Income

Add a source of income to be added to your account transactions e.g. wages, payment from a friend, or perhaps return on investments

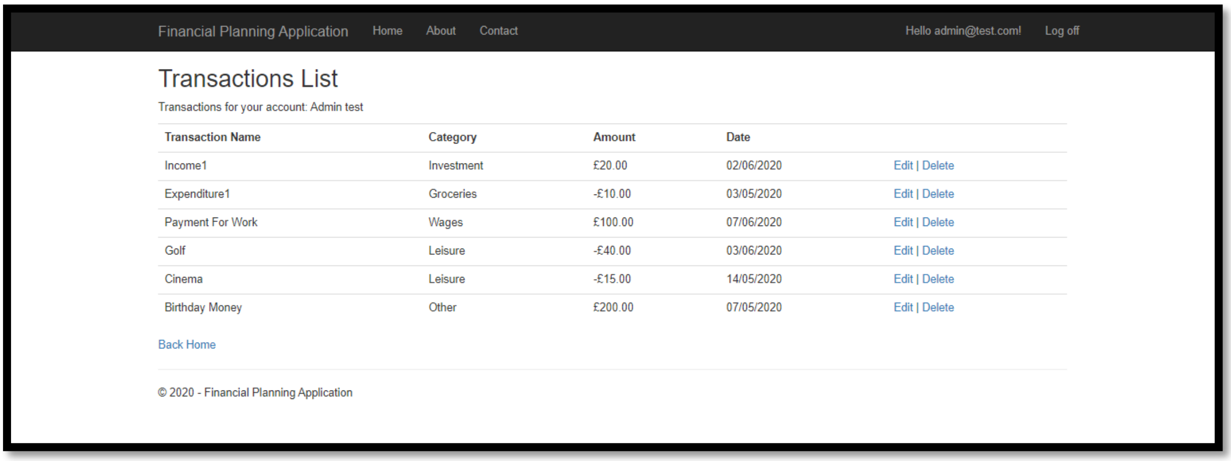
Transaction

Transaction Name	
Category	Select Category
Amount	
Date	dd/mm/yyyy
	Create

Back Home

© 2020 - Financial Planning Application

Figure 21: Add income view



Transactions List

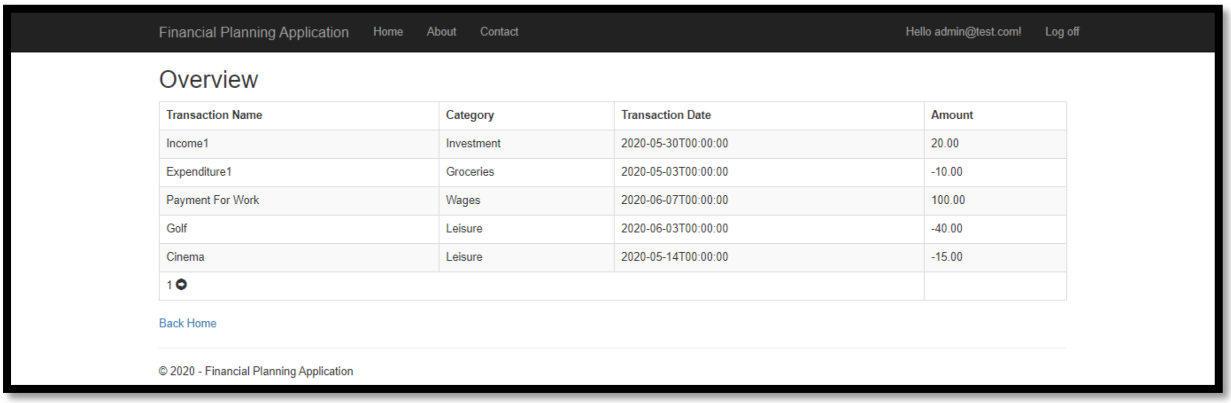
Transactions for your account: Admin test

Transaction Name	Category	Amount	Date	
Income1	Investment	£20.00	02/06/2020	Edit | Delete
Expenditure1	Groceries	-£10.00	03/05/2020	Edit | Delete
Payment For Work	Wages	£100.00	07/06/2020	Edit | Delete
Golf	Leisure	-£40.00	03/06/2020	Edit | Delete
Cinema	Leisure	-£15.00	14/05/2020	Edit | Delete
Birthday Money	Other	£200.00	07/05/2020	Edit | Delete

Back Home

© 2020 - Financial Planning Application

Figure 22: User specific transaction list



Overview

Transaction Name	Category	Transaction Date	Amount
Income1	Investment	2020-05-30T00:00:00	20.00
Expenditure1	Groceries	2020-05-03T00:00:00	-10.00
Payment For Work	Wages	2020-06-07T00:00:00	100.00
Golf	Leisure	2020-06-03T00:00:00	-40.00
Cinema	Leisure	2020-05-14T00:00:00	-15.00
1 ➔			

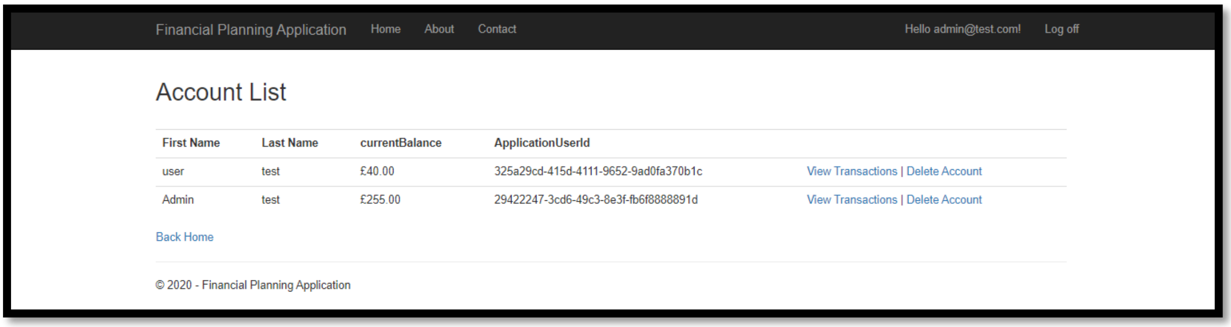
Back Home

© 2020 - Financial Planning Application

Figure 23: Admin overview of other users' transactions

The transactions overview of the logged in user can be seen in the above screenshot (figure 22). The currently logged in user is the only one who has access to the transaction list that enables them to select and edit a single transaction. However, the view in figure 23 shows the view where an Admin can access the transaction overview for any user. If a user is non-admin, they can only see the respective overview in figure 22 for their own account. The transactions are split up into pages and include all the appropriate information, omitting all the information which is not necessary for the client to see such as RowVersion.

The Admin only overview of all the accounts is as seen in figure 23, the admin account is the only account with access to this view, and hence is the only role that is able to delete accounts. If there is an attempt to delete the admin account, this will be unsuccessful, and the same form will be returned with an appropriate error message visible. It is also via this view that the Admin is able to see an overview of any users transactions.



Account List

First Name	Last Name	currentBalance	ApplicationUserId	
user	test	£40.00	325a29cd-415d-4111-9652-9ad0fa370b1c	View Transactions | Delete Account
Admin	test	£255.00	29422247-3cd6-49c3-8e3f-fb6f8888891d	View Transactions | Delete Account

Back Home

© 2020 - Financial Planning Application

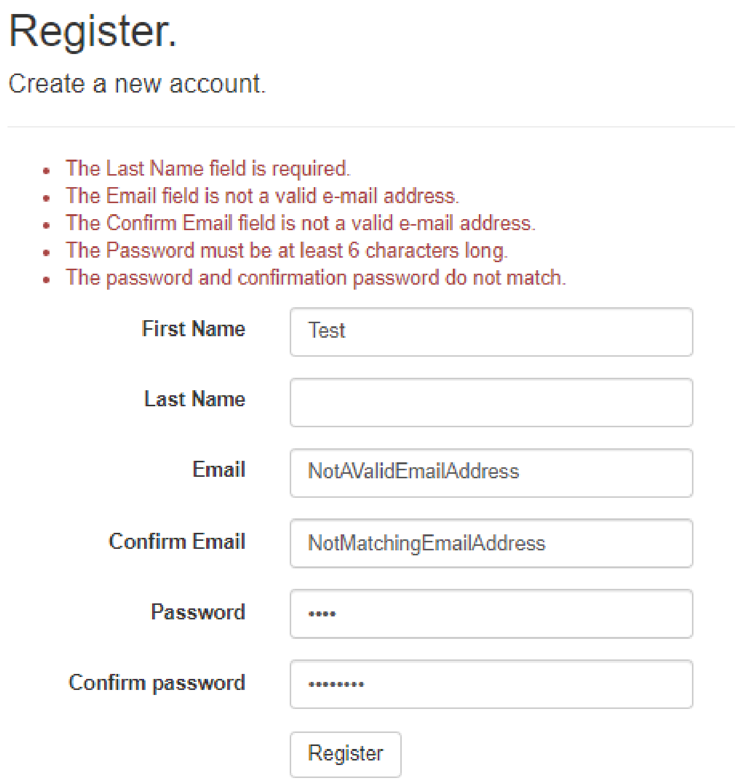
Figure 24: Admin view of all user accounts

As seen throughout the various screenshots of the application, the use of Bootstrap enables the application to have a consistent site-wide uniform layout and appearance, with the nav bar displaying the current logged in user on all views, or register/log in buttons if no user is logged in. Each view also has a 'Back Home' link at the bottom of each view, and the Nav Bar can also be used for shortcuts. However, the main functionality comes from the links available in the home screen, available to any logged in user.

5 Implementation and Testing

5.1 User Accounts

Upon attempting to create a new account via the register view, the user will have to provide all the required fields. If a field has been left empty, or is not valid, or a confirm field does not match the original field (Email and Password applicable), then the form will be redisplayed with appropriate error messages informing the user of this. This can be seen in Figure 25 where the register form has been left in an invalid state. The last name field has been left empty, the email address does not match the criteria for a string in email format, the password is not of required length/complexity, and furthermore the confirm fields for email and password do not match their respective fields. The appropriate error messages for this instance have been displayed to the user above the form, informing them of how they can make changes to successfully create an account.



Register.

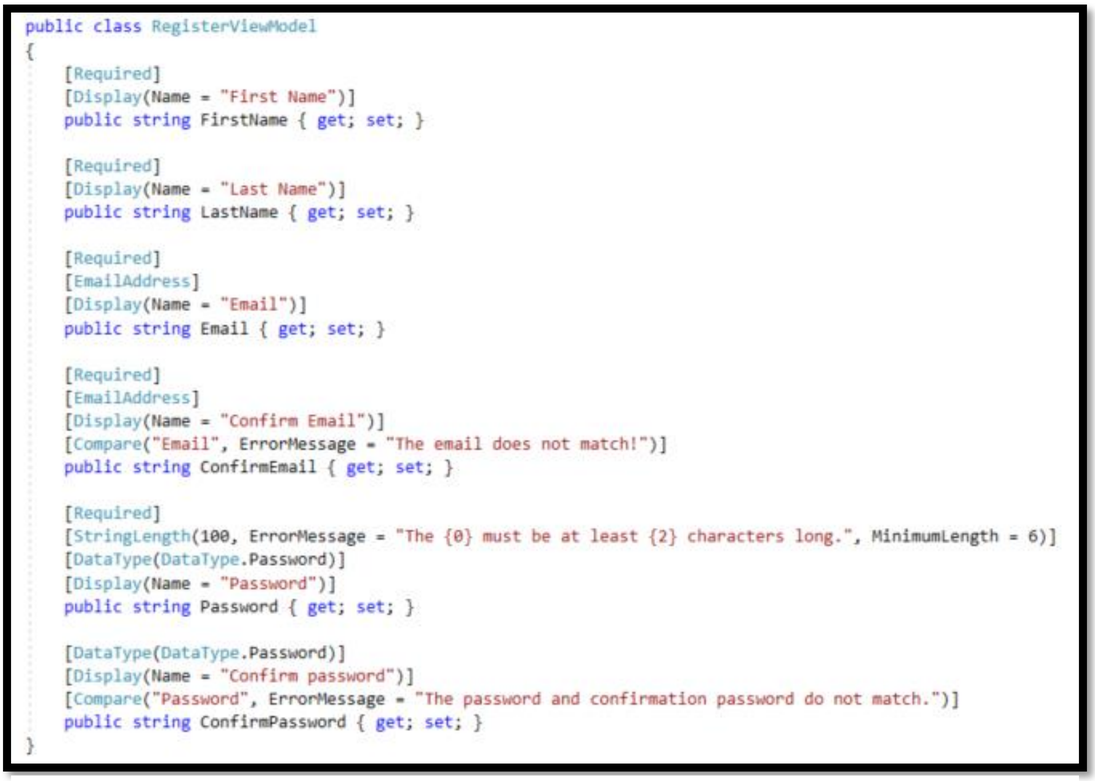
Create a new account.

- The Last Name field is required.
- The Email field is not a valid e-mail address.
- The Confirm Email field is not a valid e-mail address.
- The Password must be at least 6 characters long.
- The password and confirmation password do not match.

First Name	Test
Last Name	
Email	NotAValidEmailAddress
Confirm Email	NotMatchingEmailAddress
Password	••••
Confirm password	••••••••

Figure 25: Register View Error Feedback

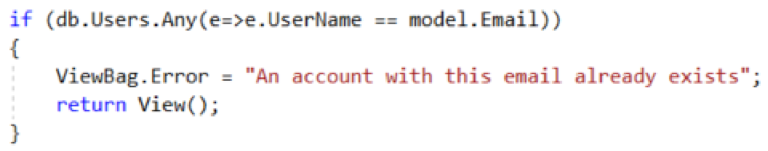
This is controlled by the various data annotations above the member variables in the *RegisterViewModel* class seen in Figure 26 below. This class is in the form of a view model and not a model, meaning it will only be displayed in views and not saved directly to the database, which will be discussed further down.



```
public class RegisterViewModel
{
    [Required]
    [Display(Name = "First Name")]
    public string FirstName { get; set; }

    [Required]
    [Display(Name = "Last Name")]
    public string LastName { get; set; }

    [Required]
    [EmailAddress]
    [Display(Name = "Email")]
    public string Email { get; set; }

    [Required]
    [EmailAddress]
    [Display(Name = "Confirm Email")]
    [Compare("Email", ErrorMessage = "The email does not match!")]
    public string ConfirmEmail { get; set; }

    [Required]
    [StringLength(100, ErrorMessage = "The {0} must be at least {2} characters long.", MinimumLength = 6)]
    [DataType(DataType.Password)]
    [Display(Name = "Password")]
    public string Password { get; set; }

    [DataType(DataType.Password)]
    [Display(Name = "Confirm password")]
    [Compare("Password", ErrorMessage = "The password and confirmation password do not match.")]
    public string ConfirmPassword { get; set; }
}
```

Figure 26: RegisterViewModel class

One additional step if the model is in a valid state, the system takes the user provided email and checks to see if an account with this already exists in the database, if it does, the view is returned with an appropriate error message informing the user of this. This is handled slightly differently than the data annotations, as the database has to be checked in this case, thus the error message is added to the ViewBag, and the same view is returned with this, as shown in the following lines of code.



```
if (db.Users.Any(e=>e.UserName == model.Email))
{
    ViewBag.Error = "An account with this email already exists";
    return View();
}
```

Figure 27: Checking if email already exists

The successful registration of an account with the *RegisterViewModel* in a valid state will in turn lead to an *ApplicationUser* being created via the *UserManager*, with the email and hashed password being stored. Then a *UserAccountModel* instance is created, with the appropriate variables, and an *ApplicationUserId* to link it to the appropriate instance of the *ApplicationUser*.

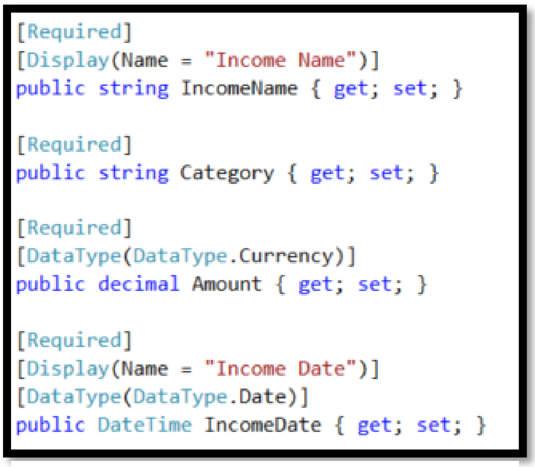
This new User that is created is saved to the *UserAccounts* DbSet via the *ApplicationDbContext*. The *SaveChanges*() method must be called in order to commit these changes and the newly created user to the database, and the interaction to the database via the use of EF will be discussed further down.

5.2 Transactions

5.2.1 Adding Transactions

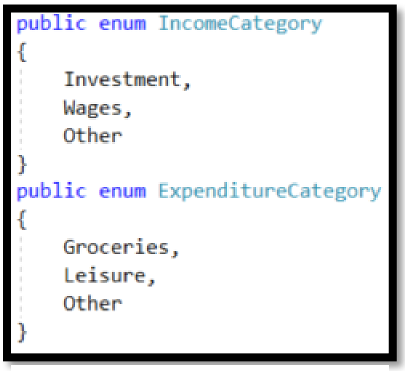
The template and view for adding transactions is in essence very similar to that of the user creation/register view in that it contains view models for both an income transaction and an expenditure transaction, along with a base *TransactionModel* class that is directly saved to the database.

For demonstration purposes here, the process of the addition of an income transaction will be explained, the view model for this being seen below. However, an expenditure transaction is almost identical to this, with the one difference being the Enum list of transaction categories available for selection. The lists for each of these can also be seen below, with the contents of each being very basic options, which could be expanded upon in future or following client requests.



```
[Required]
[Display(Name = "Income Name")]
public string IncomeName { get; set; }

[Required]
public string Category { get; set; }

[Required]
[DataType(DataType.Currency)]
public decimal Amount { get; set; }

[Required]
[Display(Name = "Income Date")]
[DataType(DataType.Date)]
public DateTime IncomeDate { get; set; }
```

Figure 28: IncomeViewModel class

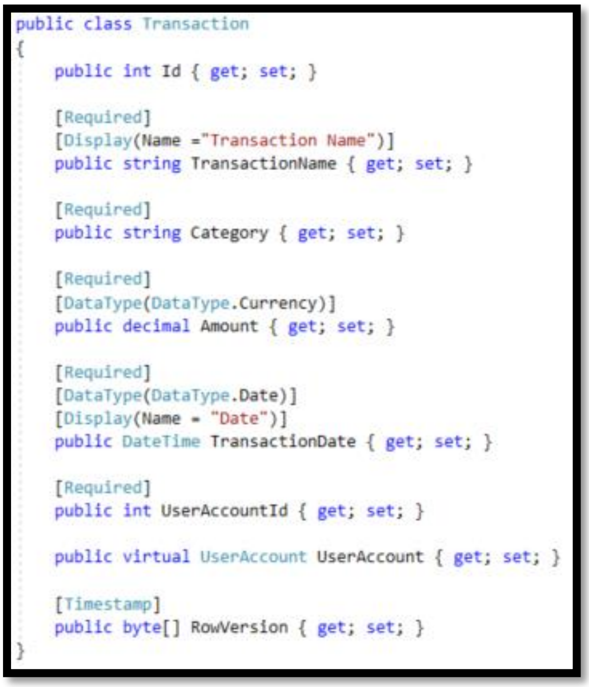


```
public enum IncomeCategory
{
    Investment,
    Wages,
    Other
}
public enum ExpenditureCategory
{
    Groceries,
    Leisure,
    Other
}
```

Figure 29: Enum categories for income/expenditure

This *IncomeViewModel* maintains a similar style to the *RegisterViewModel*, with data annotations providing information regarding which fields are required for a valid model state, data types for the amount and date variables, and Display names, to create a more attractive and user friendly interface.

The actual transaction model can be seen below in figure 30, and relative to the view models, includes further variables for *UserAccountId* and *UserAccount* (which is virtual) to link the transaction to an existing user, more specifically the currently logged in user, and *RowVersion* which has the *TimeStamp* attribute and is hence used for concurrency handling which is discussed later on.

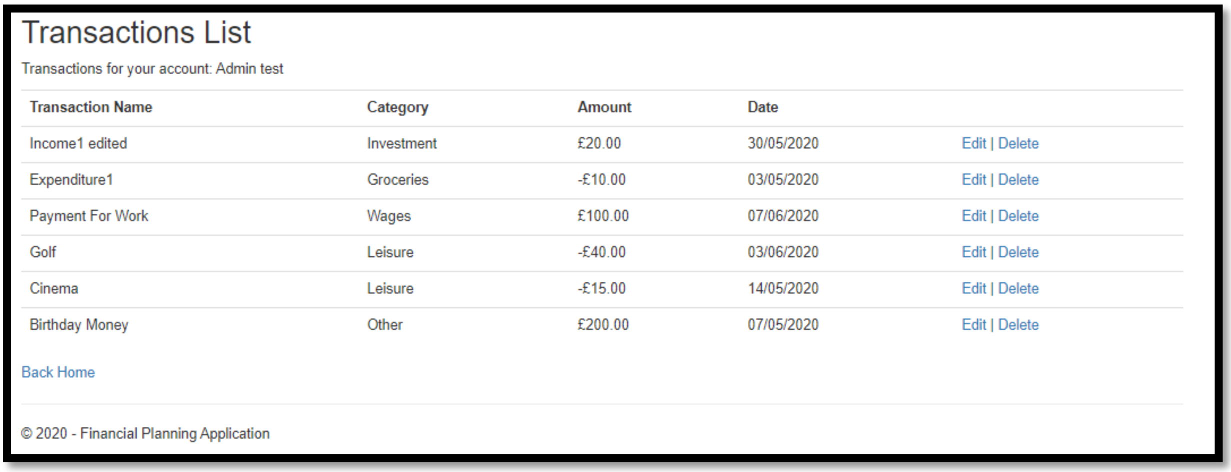


```
public class Transaction
{
    public int Id { get; set; }

    [Required]
    [Display(Name ="Transaction Name")]
    public string TransactionName { get; set; }

    [Required]
    public string Category { get; set; }

    [Required]
    [DataType(DataType.Currency)]
    public decimal Amount { get; set; }

    [Required]
    [DataType(DataType.Date)]
    [Display(Name = "Date")]
    public DateTime TransactionDate { get; set; }

    [Required]
    public int UserAccountId { get; set; }

    public virtual UserAccount UserAccount { get; set; }

    [Timestamp]
    public byte[] RowVersion { get; set; }
}
```

Figure 29: Transaction model class

5.2.2 Viewing/Editing Transactions

A logged in user has the ability to display all of their account transactions in a list displaying all the appropriate information, as seen in figure 31. They are also able to select a transaction from this list and load this data into a new view to edit this record. Should the transaction model be in a valid state, with all the appropriate fields provided, this transaction can then be altered, with the new changes being saved back to the database.



Transactions List

Transactions for your account: Admin test

Transaction Name	Category	Amount	Date	
Income1 edited	Investment	£20.00	30/05/2020	Edit | Delete
Expenditure1	Groceries	-£10.00	03/05/2020	Edit | Delete
Payment For Work	Wages	£100.00	07/06/2020	Edit | Delete
Golf	Leisure	-£40.00	03/06/2020	Edit | Delete
Cinema	Leisure	-£15.00	14/05/2020	Edit | Delete
Birthday Money	Other	£200.00	07/05/2020	Edit | Delete

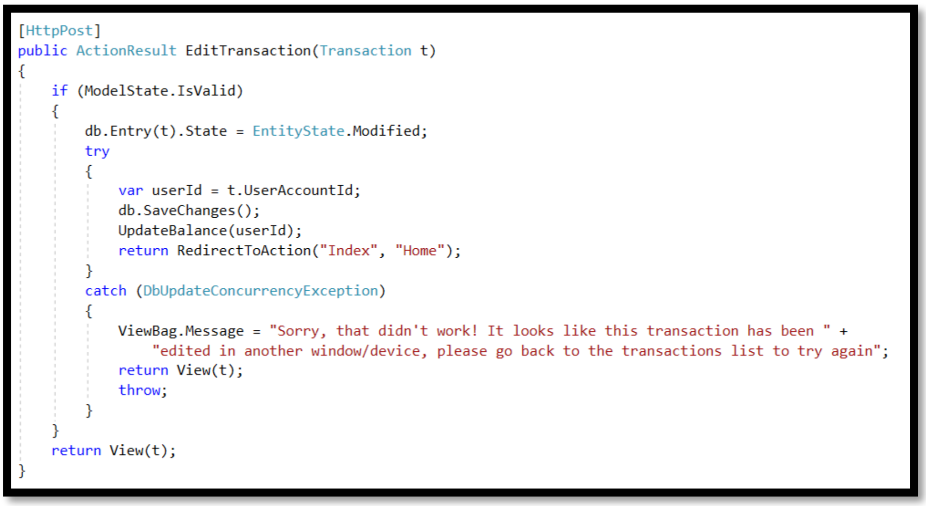
Back Home

© 2020 - Financial Planning Application

Figure 30: Transaction list breakdown for logged in user

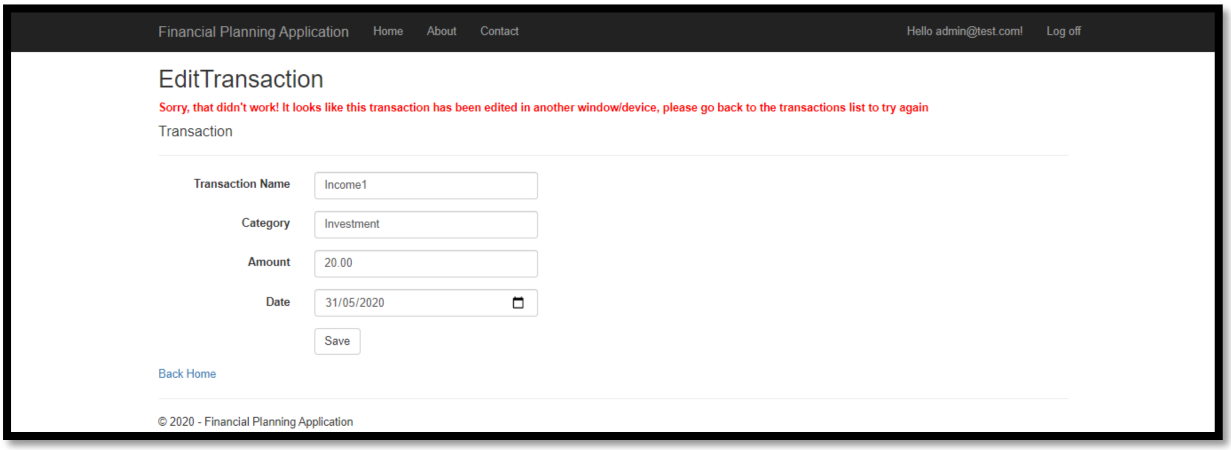
There is however potential for concurrency issues with the editing of transactions. If for example, a transaction record is loaded up on two different windows, and one commits changes, this will cause the database to become in an inconsistent state. The application will be attempting to edit a record that technically does not exist in that exact format. This will cause an exception and will prevent the application from working fluently.

The way that this can be combatted is via the use of Optimistic Concurrency Handling (Lee et al., 2002). The inclusion of the RowVersion attribute in the Transaction model allows for the facilitation of this. Upon one instance of a record being edited, entity framework will automatically change the RowVersion of this. If the record open in the second window is attempted to be edited, before changes are committed, it will check to ensure that the RowVersion of the record is the most up to date version and matches that of the one in the current database. If the RowVersion is different, Entity Framework can tell that the record has been altered by someone else, and thus we can return back the view with an appropriate error message. Thus, the concurrency issue being handled, whilst informing the user of this in understandable language as seen in figure 32 and figure 33.



```
[HttpPost]
public ActionResult EditTransaction(Transaction t)
{
    if (ModelState.IsValid)
    {
        db.Entry(t).State = EntityState.Modified;
        try
        {
            var userId = t.UserAccountId;
            db.SaveChanges();
            UpdateBalance(userId);
            return RedirectToAction("Index", "Home");
        }
        catch (DbUpdateConcurrencyException)
        {
            ViewBag.Message = "Sorry, that didn't work! It looks like this transaction has been " +
                "edited in another window/device, please go back to the transactions list to try again";
            return View(t);
            throw;
        }
    }
    return View(t);
}
```

Figure 31: EditTransaction method with concurrency handling



EditTransaction

Sorry, that didn't work! It looks like this transaction has been edited in another window/device, please go back to the transactions list to try again

Transaction

Transaction Name	Income1
Category	Investment
Amount	20.00
Date	31/05/2020

© 2020 - Financial Planning Application

Figure 32: Concurrency handling returned edit transaction view

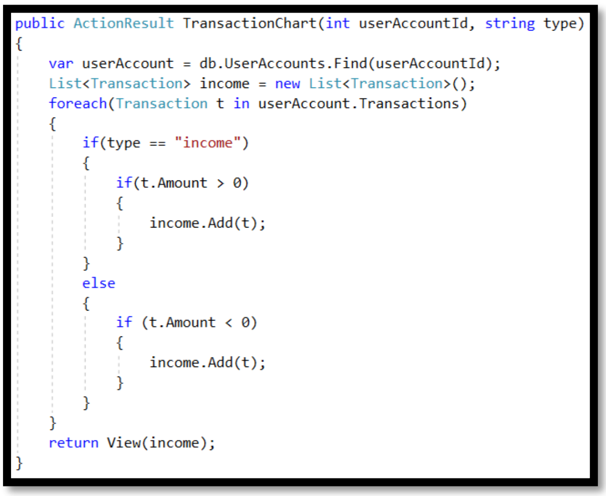
An attempt was made to incorporate a concurrency filter to display information on what the open transaction had been changed to. This would work in the following way – if the transaction values had been changed to X, Y and Z since the user opened a edit transaction form, the form would be reloaded and display an error message similar to the following:

"Sorry, that didn't work! It looks like this transaction has been edited in another window/device, the current values for this transaction are X, Y and Z. If you wish to update this transaction to the new values, please click 'save' again".

However, this proved too difficult to implement due to access issues with the transaction Id and foreign key values in the database. This meant the method was unable to bind the transaction attributes in order to feedback this proposed advanced information to the user.

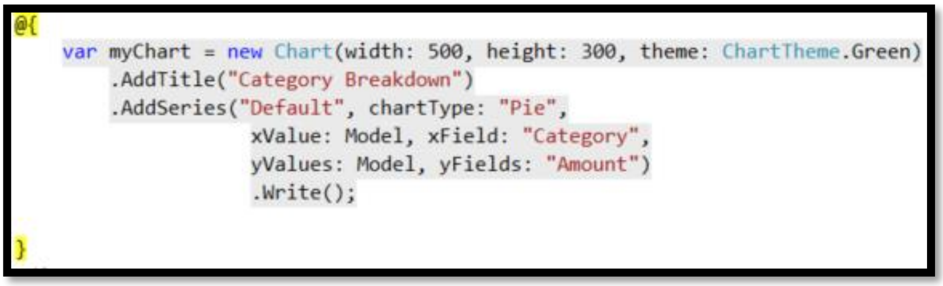
5.2.3 Transaction Category Breakdown

The application home/index page contains two buttons in the second row (on desktop full screen) which respectively contain links to an income chart and an expenditure chart. These charts are in the form of pie charts, and upon clicking of the link, a new tab will open displaying this chart. The chart will contain transaction information that is specific to the logged in user. If the user clicks the income chart button, the transactions from this user are filtered through, and all income transactions are added to a new list, which is then passed into the view which displays a pie chart of this respective data. The code for creation of the pie chart can be seen in figure 34 and figure 35, and a screenshot from an example created pie chart can be seen in figure 36.



```
public ActionResult TransactionChart(int userAccountId, string type)
{
    var userAccount = db.UserAccounts.Find(userAccountId);
    List<Transaction> income = new List<Transaction>();
    foreach(Transaction t in userAccount.Transactions)
    {
        if(type == "income")
        {
            if(t.Amount > 0)
            {
                income.Add(t);
            }
        }
        else
        {
            if (t.Amount < 0)
            {
                income.Add(t);
            }
        }
    }
    return View(income);
}
```

Figure 33: Method for loading chart in new window



```
@{
    var myChart = new Chart(width: 500, height: 300, theme: ChartTheme.Green)
        .AddTitle("Category Breakdown")
        .AddSeries("Default", chartType: "Pie",
                    xValue: Model, xField: "Category",
                    yValues: Model, yFields: "Amount")
                    .Write();

}
```

Figure 34: Razor syntax within the view for rendering the chart

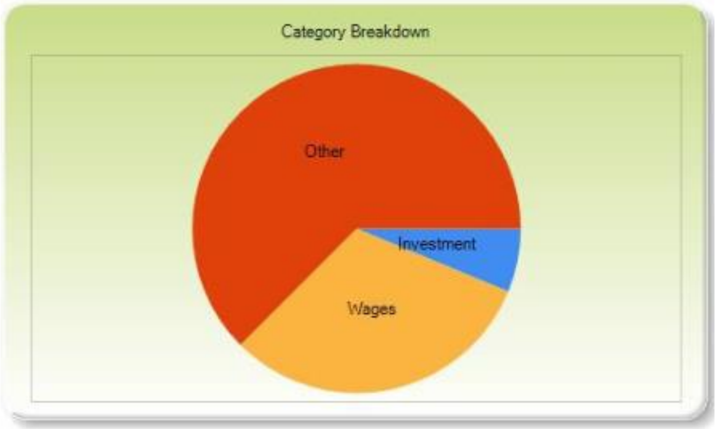


Figure 35: Example data transaction category breakdown

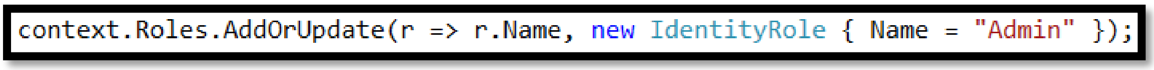
5.2.4 Deleting Transactions

The user also has the ability to delete a transaction record as displayed in the edit Transactions section. The user can select from the transaction list, and upon selecting a transaction to be deleted, it is retrieved from the list and removed from the database. The user account balance is then updated with the difference being the amount that the deleted transaction had, the changes are saved to the database via the *applicationDbContext*, and the user is redirected to the home page.

The Admin role has the ability to view any users transactions and can delete the selected user, however they are not able to directly interact with transactions of different users – as discussed below.

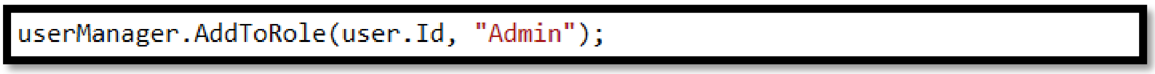
5.3 Admin Functionality and Roles

Within the seed method of the migrations *configuration.cs* file are the following two lines of code. The first line is the creation of a new *IdentityRole* and assigns the Admin status to it. The second line is using the built in *userManager* in order to add the previously created user to the admin role via its Id.



```
context.Roles.AddOrUpdate(r => r.Name, new IdentityRole { Name = "Admin" });
```

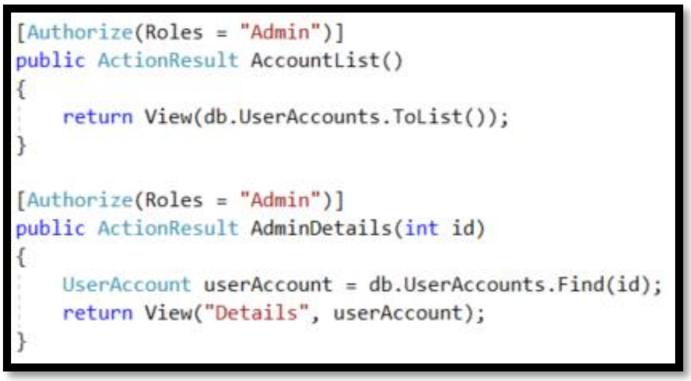
Figure 36: Creating an "Admin" role



```
userManager.AddToRole(user.Id, "Admin");
```

Figure 37: Assigning newly created user Admin status

This Role can then be used for levels of authorization on different methods/controllers. Figure 39 shows two methods in the *UserAccountController* which are decorated with the *[Authorize(Roles=”Admin”)]* attribute. This means that these methods can only be accessed by admin users.

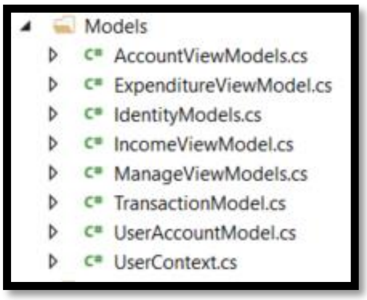


```
[Authorize(Roles = "Admin")]
public ActionResult AccountList()
{
    return View(db.UserAccounts.ToList());
}

[Authorize(Roles = "Admin")]
public ActionResult AdminDetails(int id)
{
    UserAccount userAccount = db.UserAccounts.Find(id);
    return View("Details", userAccount);
}
```

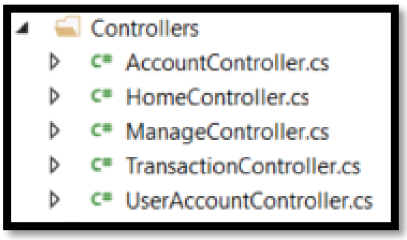
Figure 38: Method examples with Admin authorization

Any attempt to reach these methods will prompt the user to be directed towards the login page, giving them the opportunity to login as an Admin. The same redirect principle can be applied to the standalone [Authorize] attribute which decorates nearly all controllers and methods. The only methods which this does affect include the *About* and *Contact* methods within the *HomeController* which respectively correlate to the about and contact pages on the site. This attribute will mean that any attempt to access these pages whilst not being logged in to a user or admin account will cause a redirect to the login screen. Of course, the user can gain access to this by registering an account if they do not have credentials for an existing one. The about and contact pages are designed to be accessed even without the user being logged in, so that anyone can have access to this information. The About section specifically contains login information for test accounts that can be used for application demonstrations.

5.4 Models and Controllers



```
Models
    AccountViewModels.cs
    ExpenditureViewModel.cs
    IdentityModels.cs
    IncomeViewModel.cs
    ManageViewModels.cs
    TransactionModel.cs
    UserAccountModel.cs
    UserContext.cs
```



```
Controllers
    AccountController.cs
    HomeController.cs
    ManageController.cs
    TransactionController.cs
    UserAccountController.cs
```

Figure 39: Model folder

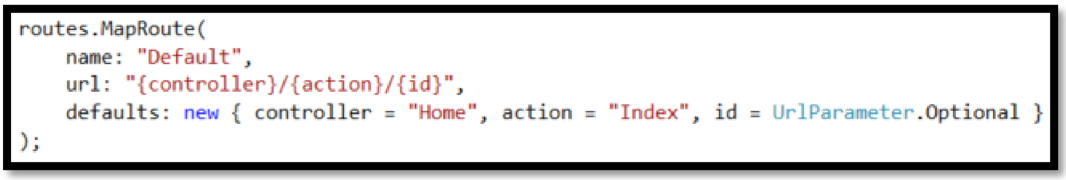
Figure 40: Controllers folder

The nature of this project being an MVC based, meant the inclusion of models and controllers which work together to perform functions on data to result in views that would be displayed to the user.

As shown in figure 40 which displays the contents of the models folder, the application, which effectively holds all the classes that will be stored in the database and displayed on the views. The *AccountViewModel* holds all the view models for login features, including view models for registration, external login, and password resets. The Home folder contains three views, consisting of About, Contact and the main index page. The manage view contains views such as changing passwords and managing logins. The shared folder contains the base view design that is present on all views, and also contains the view file that displays the personalised nav bar at the top of each view. Transaction contains views for both expenditure and income, and UserAccount includes views for a list of all the accounts, lists of all transactions for a specific account, and editing a transaction.

Most of these views are based on methods within controllers, sharing matching folder names (i.e. a home controller method will look for a view with the same name as the method under the same named folder in the views folder – unless explicitly stated otherwise). These controllers contain methods that will perform actions on the database sets for the most part and then determine which output or view will be displayed to the user.

The first section of the route represents the controller that the task will be directed to, and the second part of the route represents the method that will be called. Various parameters can also be passed to some of the routes and methods, which can then determine which actions will be performed and which views will be displayed to the user. This route format is declared in the *RouteConfig* file with the following lines of code:



```
routes.MapRoute(
    name: "Default",
    url: "{controller}/{action}/{id}",
    defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional }
);
```

Figure 41: Route Configuration

It is possible to add a new customised route, however this was not required for this application.

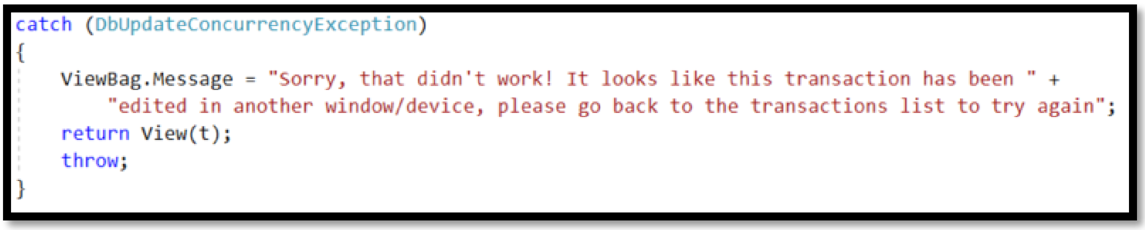
Below are some of the main routes that are used by the application, the authorization that they require, and the view/output that they produce.

Route	**Authorization**	**Output**
/Home/ /Home/Index	User logged in	Home screen displaying buttons for main application actions.
/Home/About	None	About section containing login details for test accounts.
/Home/Contact	None	Contact details page – left as default created by Visual Studio, can be adjusted for deployment into a real-world scenario.
/Account/Login	None	The main login screen, users are directed here if attempt to access a route that requires a logged in user.
/Account/Register	None	This allows a user to create and account given required fields are provided.
/Transaction/Income	User logged in	Allows the user to create a new income transaction.
/Transaction/Expenditure	User logged in	Allows the user to create a new expenditure transaction.
/UserAccount/Overview	User logged in Or Admin logged in	Gives the user an un-editable but paged layout of all transactions. Can be used by Admin to view other accounts transactions.

/UserAccount/TransactionsList	User logged in	Gives the user a list of their account transactions that they can chose from to edit or delete.
/UserAccount/EditTransaction	User logged in	Loads in the selected Transaction and allows the user to edit and save back this record to the database, with optimistic concurrency handling to prevent the database being put into an inconstant state.
/UserAccount/DeleteTransaction	User logged in	Allows the user to delete a transaction record following selection from the transactions list.
/UserAccount/Details	User logged in	A new view displaying the user's details and the current balance calculated by all the transaction records they have existing in the system.
/UserAccount/AccountList	Admin User logged in	Allows the admin user to view a list of all the user accounts that currently exist in the system. From here they can load up a specific users' transactions or delete a user. Admin account cannot be deleted.
/UserAccount/DeleteAccount	Admin User logged in	The route to delete an account from the above route.

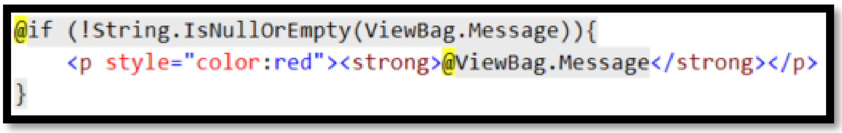
These routes can be accessed directly, including the relevant parameters such as *transactionId* or *UserId* where appropriate, however the more appropriate method for reaching these routes would include the use of the buttons and links available throughout the views, which would in turn call controller methods.

Information can be passed between controllers and hence views in three main ways. The first way is by the inclusion of a parameter in the route URL which the method can use directly. Another way information can be passed into views is via the use of ViewBag before loading up a new view. This is a dynamic property type that can be useful for transferring temporary data, as the information attached to this is removed once the view loads. One example of this would be the optimistic concurrency handling for when two instances of a transaction are attempted to be edited at the same time. In the case of this exception being caught, an appropriate message is added to the ViewBag, and the EditTransaction view with the same record is loaded with the appropriate display message. This is rendered on the page with the use of razor syntax to check if the error message exists, as shown below in figure 43 and figure 44.



```
catch (DbUpdateConcurrencyException)
{
    ViewBag.Message = "Sorry, that didn't work! It looks like this transaction has been " +
        "edited in another window/device, please go back to the transactions list to try again";
    return View(t);
    throw;
}
```

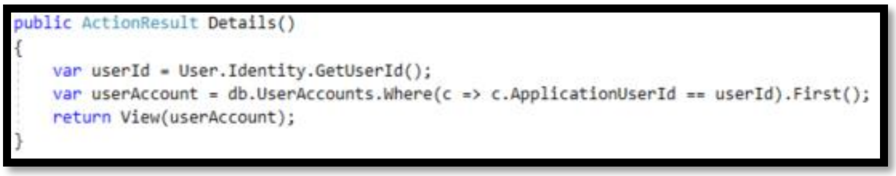
Figure 42: Concurrency Exception Error Message



```
@if (!String.IsNullOrEmpty(ViewBag.Message)){
    <p style="color:red"><strong>@ViewBag.Message</strong></p>
}
```

Figure 43: Edit Transaction View (displaying potential error message)

The final way that information can be accessed is by acquiring the *UserId* of the currently logged in user, and then querying the database with this parameter in order to obtain the user account. This method can be seen in the Details method of the *UserAccountController*, as seen below in figure 45.



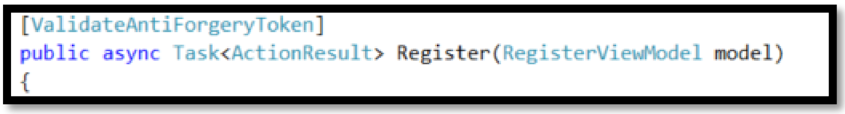
```
public ActionResult Details()
{
    var userId = User.Identity.GetUserId();
    var userAccount = db.UserAccounts.Where(c => c.ApplicationUserId == userId).First();
    return View(userAccount);
}
```

Figure 44: Details Method (brief overview of logged in users information)

5.5 Asynchronous Methods

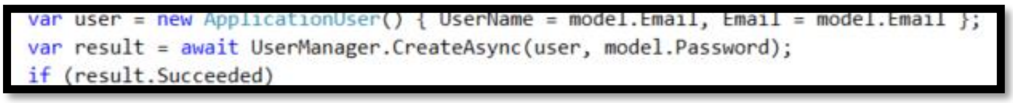
The use of the async modifier and the await expression can be incorporated into methods to ensure that tasks are performed asynchronously. This type of method means that if it is involved in a long process, then the system will not wait for it to be completed before executing further code. Thus, async methods can prevent blocking code, and can be used for network requests, disk access, and purposeful delays for a length of time. (Davies, A., 2012).

One example of an async method incorporation in this solution is shown below in figure 46 and figure 47 (both segments from the same method), which demonstrates the register method, which includes the required async modifier and the await expression in order for the user creation to be performed asynchronously.



```
[ValidateAntiForgeryToken]
public async Task<ActionResult> Register(RegisterViewModel model)
{
```

Figure 45: Async Register method (async Task)



```
var user = new ApplicationUser() { UserName = model.Email, Email = model.Email };
var result = await UserManager.CreateAsync(user, model.Password);
if (result.Succeeded)
```

Figure 46: Async Register method (await)

5.6 Bootstrap (Views) and UI

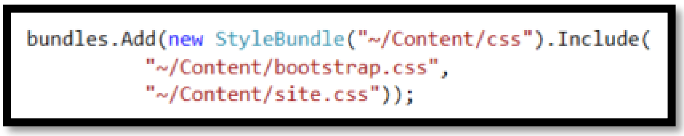
As shown in the design section of this report, the decision was made to incorporate the use of an open source front end CSS framework in the form of Bootstrap. This framework enables a site wide uniform layout which is essential for maintaining application integrity and professionalism.

Bundling and Minification

The incorporation of Bootstrap is assisted by the use of the creation of bundles in *BundleConfig*.cs which handles the CSS and JavaScript and can combine multiple files into one file. The motivation behind this is the idea that fewer files means fewer HTTP requests, and thus can cause direct improvements to application performance.

Minification can perform a number of optimisations to these files, such as deleting whitespace and comments and renaming variables to one letter where possible. The extent to which these two methods can aid performance is quite astonishing. One example from Microsoft shows the difference on a sample program with these two methods applied versus without. The file requests decreased by 256%, the KB sent and received decreased by 266% and 36% respectively, and the load time improved by 53%. (Microsoft, 2012)

The minified Bootstrap files are pre-loaded in the ASP.NET MVC solution, and the implementation of these can be performed by changing the file in the bundle to bootstrap.min.css, however it is currently set to the normal un-minified version as shown below in figure 48.

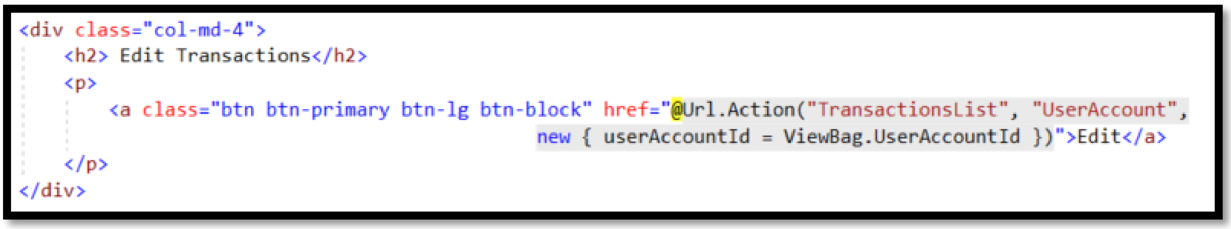


```
bundles.Add(new StyleBundle("~/Content/css").Include(
          "~/Content/bootstrap.css",
          "~/Content/site.css"));
```

Figure 47: Bundling of bootstrap CSS file

Bootstrap Grid System

The bootstrap file contains set styles for the divisions, classes, and buttons, that can be implemented in the view html. The home index view is created via the incorporation of the bootstrap 12 division grid system. The first row contains three nested division classes with a value set to “col-md-4”, one example seen in figure 49.



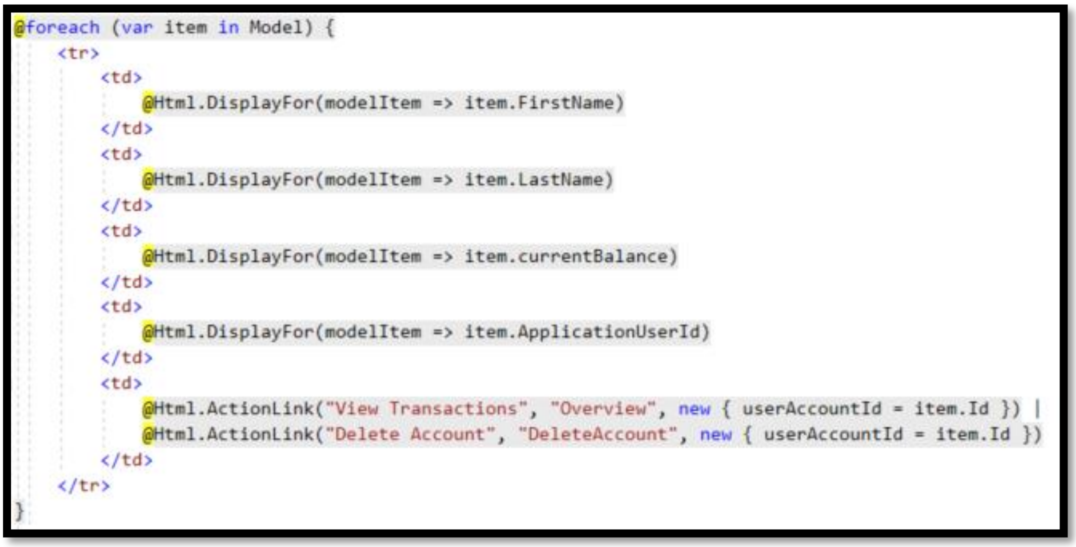
```
<div class="col-md-4">
    <h2> Edit Transactions</h2>
    <p>
        <a class="btn btn-primary btn-lg btn-block" href="@Url.Action("TransactionsList", "UserAccount",
                                                         new { userAccountId = ViewBag.UserAccountId })">Edit</a>
    </p>
</div>
```

Figure 48: Nested division tags on the index view (Bootstrap grid use)

These 3 divisions each with a value of “col-md-4” stretch out to fill the 12 divisions. The same technique is used for the second and third rows on the index page. As these rows only contain 2 divisions, these divisions are assigned the value of “col-md-6” so that combined they fill the 12 divisions for each row.

Razor

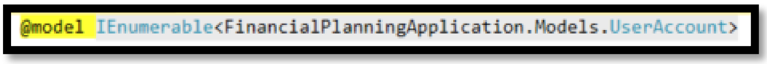
A significant amount of the Views within this application involved the incorporation of razor syntax within the html for assisting with data loading and display. The razor syntax can be initiated by the @ symbol within the view file, and so the system knows that the following code will be C# code and not html. A large use of the razor mark-up was the displaying of database records in tables, one example being the AccountList view.



```
@foreach (var item in Model) {
    <tr>
        <td>
            @Html.DisplayFor(modelItem => item.FirstName)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.LastName)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.currentBalance)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.ApplicationUserId)
        </td>
        <td>
            @Html.ActionLink("View Transactions", "Overview", new { userAccountId = item.Id }) |
            @Html.ActionLink("Delete Account", "DeleteAccount", new { userAccountId = item.Id })
        </td>
    </tr>
}
```

Figure 49: Use of Razor syntax to access and display model values

As shown in figure 50, when a view is loaded up with a model passed to it, Razor syntax and C# code can be used to loop through all the User records in the model (which in this case is an Enum of the *UserAccounts* – see figure 51). Razor can then be used to access each of the variables in every userAccount record, and a new table row is created for each individual record. Furthermore, the ActionLinks in the last column of each row can be used to redirect to methods, passing in the id of the UserAccount on the selected row as a parameter.



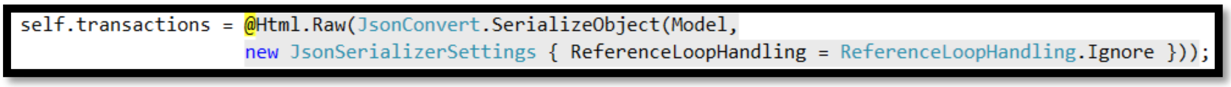
```
@model IEnumerable<FinancialPlanningApplication.Models.UserAccount>
```

Figure 50: Enum model type for the view

Knockout

Knockout is a JavaScript framework that makes use of the MVVM pattern (Model-View-View Model) which allows the binding of HTML elements against a model. This can be used for creating a clean user interface display, and in this application, is used for the overview view. This data binding results in an automatic UI refresh, causing it to be responsive in nature and meaning the view shows an up to date model state. (Knockout, n.d.)

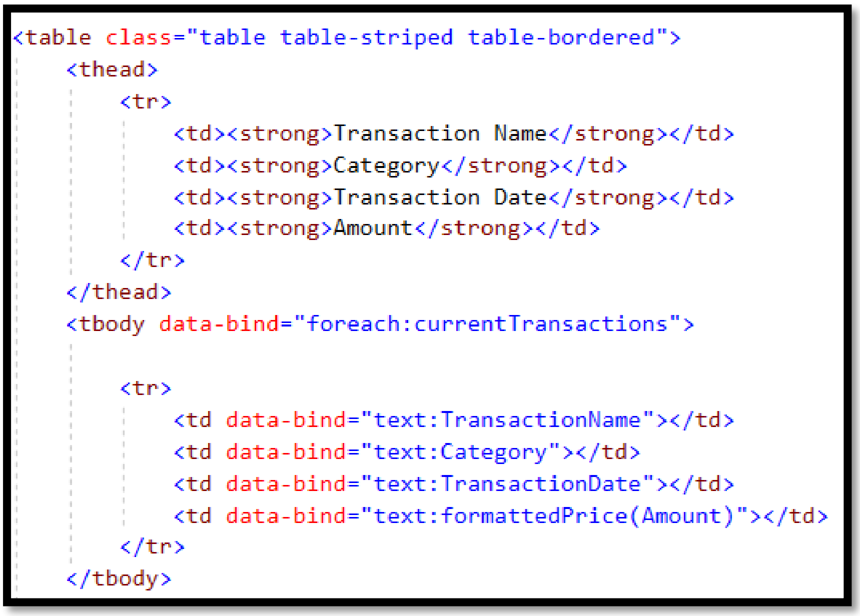
The overview view within the application contains use of the knockout framework. The transactions model is Serialized with JSON and then linked to *self.transactions*, as shown in the lines of code below in figure 52. The transactions are split into pages, and then the bindings are applied.



```
self.transactions = @Html.Raw(JsonConvert.SerializeObject(Model,
                    new JsonSerializerSettings { ReferenceLoopHandling = ReferenceLoopHandling.Ignore }));
```

Figure 51: Serializing the transaction model data

The table can then iterate through each transaction in the current page and display the up to date model information for all the transactions on that current page. This use of data binding means the data is always up to date, and automatic refreshes of the UI will be applied if any changes are made.

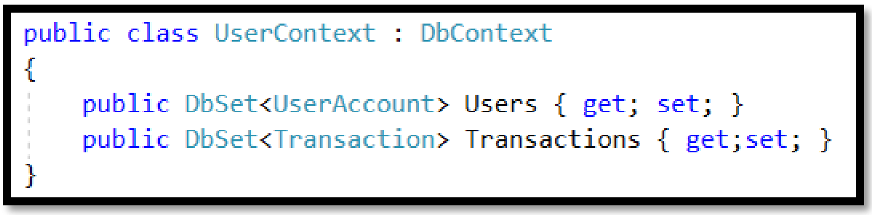


```
<table class="table table-striped table-bordered">
    <thead>
        <tr>
            <td><strong>Transaction Name</strong></td>
            <td><strong>Category</strong></td>
            <td><strong>Transaction Date</strong></td>
            <td><strong>Amount</strong></td>
        </tr>
    </thead>
    <tbody data-bind="foreach:currentTransactions">

        <tr>
            <td data-bind="text:TransactionName"></td>
            <td data-bind="text:Category"></td>
            <td data-bind="text:TransactionDate"></td>
            <td data-bind="text:formattedPrice(Amount)"></td>
        </tr>
    </tbody>
```

Figure 52: Displaying the transaction bindings in a table

5.7 Database

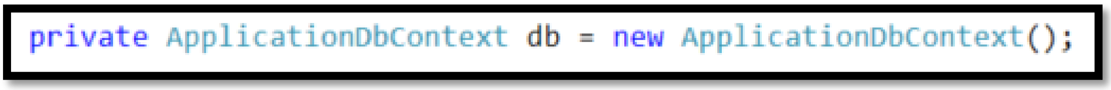
Regarding storage of user information for this web application, a Database with various tables was used to facilitate this. The code-first approach with the use of Entity Framework, which is an Object-Relational Mapping (ORM) Framework was essential to this. This allowed communication with the database as though interacting with C# objects. In order to communicate with the database, a database context was needed, with DbSets created for each entity that needed modelling to a table. This declaration of DbSets can be seen in figure 54. These DBSets of entities are linked via the use of virtual member variables e.g. each UserAccount in Users has a virtual ICollection of Transactions, which entity framework is smart enough to link via the incorporation of lazy loading.



```
public class UserContext : DbContext
{
    public DbSet<UserAccount> Users { get; set; }
    public DbSet<Transaction> Transactions { get;set; }
}
```

Figure 53: DbSets declaration

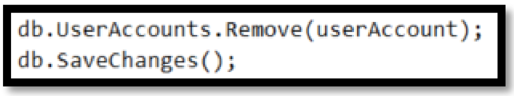
At the top of each controller, the ApplicationDbContext was declared with the following line of code or similar:



```
private ApplicationDbContext db = new ApplicationDbContext();
```

Figure 54: ApplicationDbContext declaration (in each controller)

This would allow access to the database and specifically the DbSets. Then, one of the many CRUD operations could be performed, these being create, retrieve, update, and delete. One example of this can be seen in the removal of an account by an Admin in figure 56.



```
db.UserAccounts.Remove(userAccount);
db.SaveChanges();
```

Figure 55: CRUD Delete operation (remove user)

Entity Framework pushes these changes to a database which is configured on Microsoft SQL Server via the use of a connection string for naming the database. A check is done to see if there is an existing database with this name, and if not, one is automatically created with the code-first technique using the classes/entities to create the specific tables.

Should any changes be made to the classes that the DbSets are based on, these can be translated easily into an updated database via the use of migrations. The way in which a migration works is that it takes the DbSets entities and its class variables, and relative to the existing database, it makes an SQL script to directly interact with the database to make these changes.

The creation of migrations also means a configuration.cs file, within which is a seed method. This method gets called whenever an Update-Database function is called (usually to update the database to the latest created migration) and can be useful for populating the database with records and assigning user roles. This can also be useful for testing, as it provides a consistent base of records that the system can be tested against, with the expected outcomes of tests easy to calculate.

All of these functions are performed in the Package Manager Console which is built into visual studio. Each migration has up and down methods, the up being the changes that will be made to the database when the existing migration is committed and the down method being the changes that would be required in order for a previous migration state to be reverted to. This is particularly useful if a corrupt migration is made and can be achieved by updating the database and targeting a specific migration that you wish to rollback to.

5.8 Hosting

Local testing can be done on the software by running the application on localhost, however in order to make this web app into its namesake, it needs to be readily available and accessible for anyone who has internet connection. This can be achieved by hosting the site along with the database. The option chosen for this project included the use of Azure due to its connectivity with Visual Studio and ease of publishing with a Microsoft account. Microsoft allows free credits to users with an account you can trial host a website with a database for free.

This can be configured by completing a simple set up wizard and logging in to a Microsoft account.

5.9 Testing

The principle of separation of concerns and the organisation of controller logic into individual actions makes MVC ideal for the unit testing of code. Unit testing has no universal set definition and is often subjective. However, as a general rule, it can be considered to be the testing of methods/functions, a class or group of related classes, or just a small segment of code. A class that includes a significant amount of code should not be held accountable by just one unit test however, and multiple unit tests for a class may be required to achieve good code coverage. These tests themselves should be appropriately short and should test small bits of logic in isolation.

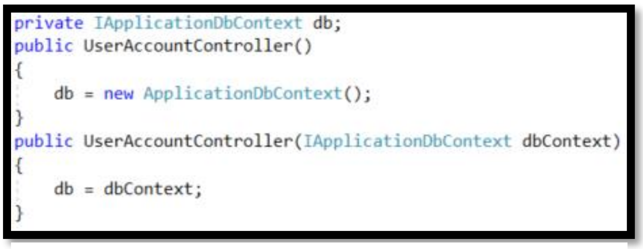
It is often a difficult task to convince developers of the worthwhileness of unit testing, however there are certainly a number of benefits of unit testing including the following:

- It can confirm that your code works the way you would expect it to, which is often not the case.
- It can help to determine when code is complete.
- It can make the changing of code less risky. When changes are made to a project, it is possible that certain features can become broken. However, a level of security will be instilled when the developer has written unit tests that can easily detect this.
- They can also help regarding the documentation of code. Well written tests should be clear in their purpose and should communicate what we want the code to accomplish.

Regarding unit tests in ASP.NET, the format usually follows an Arrange, Act, and Assert pattern. In the arrange step, we prepare the items that we wish to use for the specified test. In the Act step, various methods or procedures are called, and then in the Assert step tests to see if the expected outcome is matching to the actual outcome. Thus, giving us knowledge on whether a unit in our code is functioning as desired. Unit tests should be lightweight, fast, and should not rely on dependencies such as databases. The testing of methods that involve communication with a database are more appropriately called integration tests, as it would be testing of a unit with another component. However, in order to facilitate the testing of these units, we can mock a dependency (in this case the database) to ensure testing is still done in isolation. The testing for this application was performed via the use of Visual Studio Testing Framework (Microsoft, 2019).

The creation of a mock database context allows us to perform tests using any in memory objects/test data that we wish to construct. Thus, the tests will be able to be completed regardless of the environment and the database.

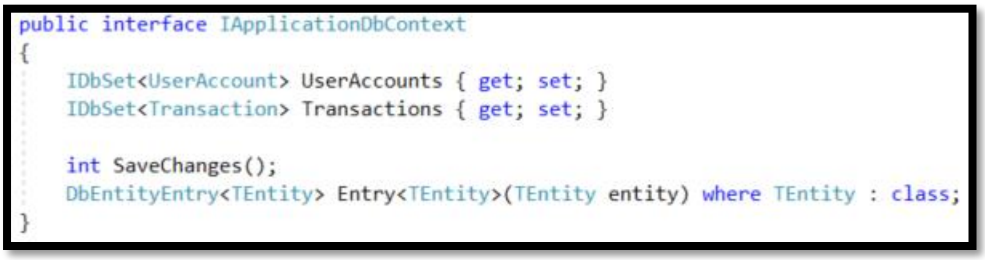
The way in which we achieve this is by changing any instances of *DbSet* to *IDbSet*, ApplicationDbContext to *IApplicationDbContext*, defining this interface, and making sure the ApplicationDbContext inherits this interface. This means that any db declaration could be connecting to either real database data or fake in-memory data. We can create two constructors which handle the assignment for both these uses. This means that every controller which included a db context was changed to the following (or similar):



```
private IApplicationDbContext db;
public UserAccountController()
{
    db = new ApplicationDbContext();
}
public UserAccountController(IApplicationDbContext dbContext)
{
    db = dbContext;
}
```

Figure 56: Option to use IApplicationDbContext

The decision was made to move ApplicationDbContext from IdentityModels.cs into its own file, and creation of an interface which contained the IDbSets for both UserAccounts and Transactions. A fake DbSet was taken from Github Gist (GitHub, 2011) that works with an in memory list instead of with the database. The interface included IDbSets and also a method for SaveChanges and a DBEntityEntry, as seen in figure 58.

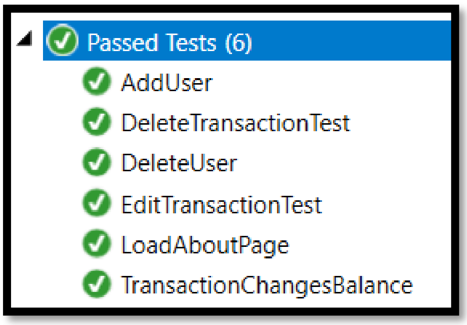


```
public interface IApplicationDbContext
{
    IDbSet<UserAccount> UserAccounts { get; set; }
    IDbSet<Transaction> Transactions { get; set; }

    int SaveChanges();
    DbEntityEntry<TEntity> Entry<TEntity>(TEntity entity) where TEntity : class;
}
```

Figure 57: IApplicationDbContext interface with IDbSets

There implementation of the mock database definitely caused some issues, however some unit tests that were performed, with a description of the test goal can be seen in the table below. A screenshot of the passing tests can also be seen in figure 59.

Test Method Name	Description
AddUser	This test was designed to see whether a user could be created and successfully added and saved to the database.
DeleteTransaction	This test was designed to check the ability of a user to delete a specific transaction, whilst not affecting the rest of the User.
DeleteUser	This test was designed to see if a user can be successfully deleted from the database.
EditTransactionTest	This test checks whether a transaction can be successfully edited in one or more values and saved back to the database.
LoadAboutPage	This is a generic test which checks to ensure the correct loading of the view.
TransactionChangesBalance	This check tests to see that a user's current balance is updated after a new transaction is created and added to that specific user's transaction list.



Passed Tests (6)
- ✓ AddUser
- ✓ DeleteTransactionTest
- ✓ DeleteUser
- ✓ EditTransactionTest
- ✓ LoadAboutPage
- ✓ TransactionChangesBalance

Figure 58: Passing Unit Tests

6 Evaluation

The success of a project, specifically creation of software, can for the most part be measured by its completion against the requirements criteria. Below is an analysis of the project final state against each of the requirements stated, whether they were completed and to what degree.

Functional Requirements

The completion of functional requirements can be considered more definitive as it measures against the functional actions of the software. 6 out of 7 of the main functional requirements were completed to a satisfactory degree.

Requirement	Completed	Comments
1.1 Add Account	✓	The user has the ability to create an account, with appropriate error handling for invalid fields i.e. invalid email address or a short password. New user accounts are successfully saved to the database.
1.2 Add Income/Expenditure	✓	The application successfully allows the user to create a new income or expenditure transaction – provided all required fields have appropriate values. One feature that could be improved upon is the options that the user has to chose from for the transaction category, however this would be easy to implement for real world deployment.
1.3 View Transactions	✓	The user has two views for displaying the transactions, one is produced via a ToList display of the model and the other is a read only display via the incorporation of knockout.js and data binded so that real time data is displayed.
1.4 Transaction Category Breakdown	✗	Unfortunately, this feature was not implemented as ideally would have been. This could be done by having a drop-down box above the transactions list, and selection of a category would filter all transactions and show only the ones relevant to that category.
1.5 Transaction Visualisation Breakdown	✓	This feature was implemented in the form of a simple pie chart, and could be displayed with a specific users' income transactions or expenditure transactions, showing them which categories the most funds (highest transaction combined amounts) were being allocated to.
1.6 Edit Transactions	✓	The user has the ability to select a transaction from a list and either delete or edit this. See non-functional requirements concurrency handling for further detail on editing transactions.

1.7 Admin Functions	✓	The admin user has access to an overview of all existing accounts and can also view a selected accounts transactions, or delete a selected account. The admin cannot directly interact with another users transactions – which could be implemented. The non-admin users do not have access to these functions due to the [Authorize(Roles=”Admin”)] attribute, which fills the requirement criteria.

The non-functional requirements are more open to interpretation of completion relative to the functional requirements. However, for the most part, the project can be considered to have completed all of these to a satisfactory degree.

Non-Functional Requirements

Requirement	Completed	Comments
2.1 Database Storage	✓	The project uses code-first entity framework development to interact with the database and save changes when appropriate.
2.2 Responsive Layout	✓	The use of Bootstrap and a grid system layout enabled the device to be responsive to the screen viewing size, and is compatible with most browsers and devices (including mobile devices).
2.3 Concurrency Handling	✓	The application incorporates optimistic concurrency handling and deals appropriately with transactions being edited at the same time, with a relevant error message being displayed following this.
2.4 Security	✓	The main security issue is the storing of the user password, which is hashed before it is stored so it does not appear in the database in plaintext, and instead is in the form of a sequence of unidentifiable characters.
2.5 GDPR	✓	This requirement can to some degree be subjective, however, aside from user account creation, including name and email address, which are used for login and customisation purposes, no further information is taken. This means that the application is only acquiring and storing the required user information which complies with GDPR regulations. Due to the application not running in sync with a bank API and real time data, the user has the choice as to what transaction data they enter into the system, and are under no obligations to provide anything they do not wish to.

7 Conclusion

7.1 Project Evaluation

Regarding the project as a whole, the system developed incorporated all the base requirements that were desired, and the attractive UI remained uniform and easy to navigate for even the novice user. This fits in with the goals of providing an application for a larger demographic of those who only wanted a brief overview of income and expenditures, without becoming too overwhelmed, which is definitely the case with some of the existing solutions that are available.

However, the time management and task planning could have been implemented better, or more rigorously adhered to. Too much time was spent researching the topic of development including existing solutions and the development options that were available for producing the proposed application. In hindsight, It would have been more appropriate to have chosen a development option/technique earlier on and ensured a more perfected application via the chosen method.

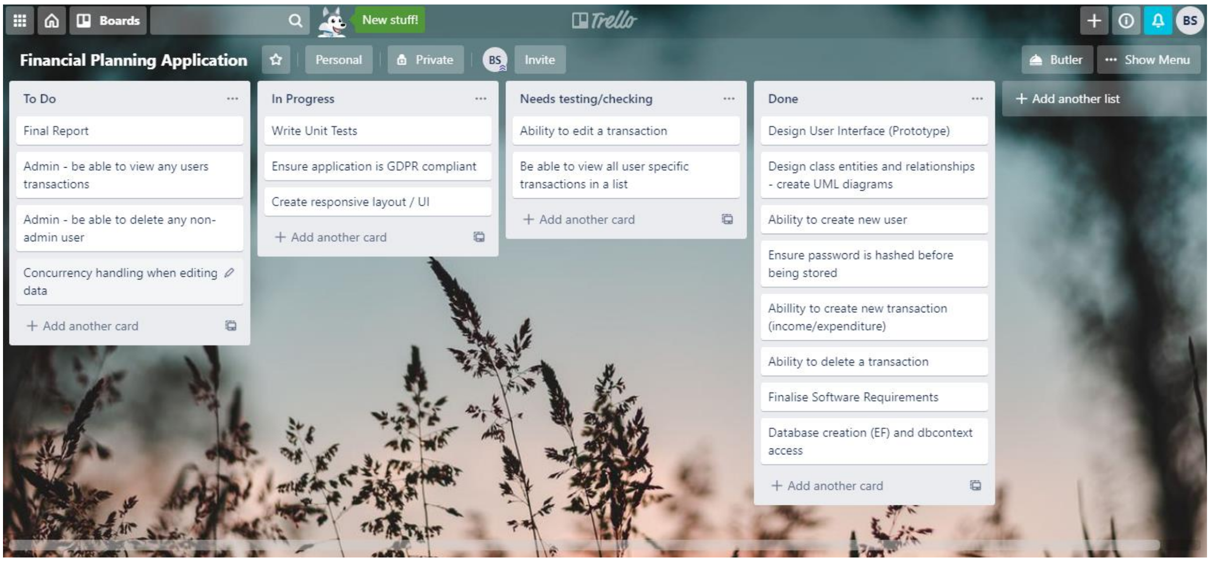
The objective of a web app creation can be considered a success, with it being available/compatible on most devices and browsers, with the availability and ease of deployment to the Azure hosting platform highly significant in achieving this.

The objective of a mobile implementation was not completed, however this was a secondary objective and thus desirable, but not essential to the project success. This objective also can be partially considered achieved however, due the final app implementation being available on most mobile browsers, and hence effectively acting like a mobile application, with the only difference being a slight decrease in accessibility and a reliance on internet access to use the application.

Overall, the project can be considered a success in terms of delivering its base functionality along with an attractive yet responsive UI, compatible with most device browsers. Furthermore, the use of Entity Framework to communicate from C# objects to database entities worked as well as anticipated, along with the Azure hosting and its easy integration due to its links with Visual Studio. The main downside to the project would be that it did not satisfy all the functional requirements, and the inclusion of more advanced base application features including the viewing of the transaction list by category or time period, and the option to allocate a certain amount of funds to a savings section.

7.2 Project/Time Management

As stated in the Project Initiation Document, the project operation and development methodology included was going to be based on an agile-like approach. This method was considered more favourable for development relative to a waterfall-based approach due to several reasons. The project in question does not explicitly have a client that the software is being designed for, and instead for a potential target market, thus meaning that requirements are not fixed at the beginning of the project. Agile is more accommodating of changes down the timeline and is also more capable of producing rapid development. Of course, this process was only agile-like and not explicitly agile due to it not being a team/group project and no reason for group meetings to discuss progress. However, the incorporation of an agile board to manage task completion provided a useful visualisation and organisation of the task backlog. A screenshot of it in use can be seen in Figure 60.



To Do
- Final Report
- Admin - be able to view any users transactions
- Admin - be able to delete any non-admin user
- Concurrency handling when editing data
- + Add another card

In Progress
- Write Unit Tests
- Ensure application is GDPR compliant
- Create responsive layout / UI
- + Add another card

Needs testing/checking
- Ability to edit a transaction
- Be able to view all user specific transactions in a list
- + Add another card

Done
- Design User Interface (Prototype)
- Design class entities and relationships - create UML diagrams
- Ability to create new user
- Ensure password is hashed before being stored
- Abillity to create new transaction (income/expenditure)
- Ability to delete a transaction
- Finalise Software Requirements
- Database creation (EF) and dbcontext access
- + Add another card

Figure 59: Use of Trello board for task management

However, as evident, the use of the agile board only utilised a high-level breakdown of tasks involved in the software development, and would have benefited significantly by improving upon this. There was also a distinct lack of following the time management and task allocation plan, which was stated in the PID, causing project disrupts, and generally inhibiting software development progress. If the project was to be done again, this is one area that would have definitely been done differently. A time plan would have been followed far more rigorously, and any deviation from the plan would have resulted in a re-allocation of resources. The original time plan from the PID can be seen in Appendix A.

Regarding time management and project supervision, weekly meetings with Dr. Bing Wang were attended to check on current progress and to provide insight and advice on what to focus on following the meetings. The in-person meetings were useful and continued until university teaching was suspended due to COVID-19. After this, project supervision was carried out via communication platforms including discord, skype and email.

7.3 What Has Been Learnt

Regarding what has been learned throughout the course of this project, along with much improved time management capabilities, a vast amount of coding/development experience has been gained. This project has seen a drastic increase in knowledge of various frameworks and coding techniques. The skills that have been acquired or improved throughout the course of this project include but are not restricted to the following:

- ASP.NET Development.
- MVC Development pattern (use of models, views, and controllers to perform functions on data and display in an attractive UI to the user).
- Server-side programming/back-end scripting, routing, HTTP requests, and the creation of dynamic web pages for interaction with the user.
- Entity Framework and a code-first approach for database creation and communication between C# objects and database entities, along with migrations and database table relationships.

- Client-Side scripting, JavaScript, and various libraries including knockout. Although less JavaScript was used in the end project, substantial research went into learning this as one of the options for implementing dynamic features.
- Asynchronous methods (using await) and optimistic concurrency handling (using RowVersion).
- A heightened awareness for the importance of time management and resource planning (did not go as fluently throughout this project, which stresses its significance for future projects).

7.4 Further Work

Regarding further work and future application development, there are several more advanced features that the implementation of would be desirable and would make the application a more attractive solution for managing a user’s finances.

Regarding these features, there are some that derive from the objectives/requirements that were unfortunately not met, and some which may be possible in future given less time/resource restrictions. These include but are not restricted to the following:

- Real time error message providing e.g. upon a user typing an email into the register form in its respective field, the form would be able to inform the user if the email was either in an invalid form or if it already existed in the system – all without having to click register and submit the form and reload the page.
- Implementation of viewing by categories – the user would be able to view all of their transactions from all categories, or filter these by selecting a category which would display only the transactions relevant to that. This was included as one of the main project/software functional requirements and was unfortunately not implemented, thus would be one of the first features to be introduced given more time to work on the project.
- Incorporation of more advanced graph visualisation for breakdown of transactions – this feature was incorporated into the final application, however its inclusion was very basic due to time/resource restrictions and would likely be improved given further time for project development.
- One long term, slightly less realistic, and potentially very difficult goal of the project given no time/cost restrictions would be the incorporation of a bank API to facilitate the use of real time data. This however would incur several further security issues and considerations, and thus was not reasonable for completion within the original timeframe. Alike some of the existing solutions examined in the literature review, the implementation of this would greatly improve the applications performance and user experience and thus in an ideal world would be included.
- A mobile app implementation could also be a future goal of the project and was considered in the project initiation document as a hopeful secondary objective. However, due to the nature of the final application that was produced and the incorporation of Bootstrap to create a mobile-browser friendly application means that this implementation is technically not needed. This is due to the coverage that Bootstrap provides, catering for and being compatible with most desktop and mobile browser implementations. Despite this, a mobile application would certainly be welcomed and very desirable due to its increased accessibility to the user i.e. having an application shortcut and the option for saving data locally to a mobile device instead of having the application open in a browser tab.

References

AgeUk, 2019 - Being in control of finances has positive effect on mental well-being says Age UK. Available at: https://www.ageuk.org.uk/latest-press/articles/2018/november/being-in-control-of-finances-has-positive-effect-on-mental-well-being-says-age-uk/ [Accessed December 19, 2019]

Anderson, R. & Dykstra, T., 2014. Getting Started with Entity Framework 6 Code First using MVC 5, Microsoft Corporation.

Bootstrap, 2019. Bootstrap. Available at: http://getbootstrap.com/ [Accessed December 20, 2019].

Bootstrap, n.d. Browsers and devices. Available at: https://getbootstrap.com/docs/4.5/getting-started/browsers-devices/). [Accessed December 22, 2019]

Bootstrap, n.d. Grid System. Available at: https://getbootstrap.com/docs/4.5/layout/grid/ [Accessed December 22, 2019].

Bootswatch, n.d. Bootswatch. Available at: https://bootswatch.com/ [Accessed May 3, 2020]

BudgetSimple, n.d. Available at: https://www.budgetsimple.com/ [Accessed April 10, 2020]

BusinessWire, 2019. Finance Apps See 90% Increase in Install Market Share Worldwide According to AppsFlyer. Available at: (https://www.businesswire.com/news/home/20200421005181/en/Finance-Apps-90-Increase-Install-Market-Share) [Accessed April 15, 2020]

Cdn.bcs.org. (2019). BCS Code of conduct. [online] Available at: https://cdn.bcs.org/bcs-orgmedia/2211/bcs-code-of-conduct.pdf [Accessed 21 Oct. 2019].

Damian Toohey, 2017. BudgetBoss. Mobile app. Version 2.9. Available from: https://apps.apple.com/us/app/budget-boss/id606136295 [Accessed April 10, 2020]

Davies, A., 2012. Async in C# 5.0. " O'Reilly Media, Inc.".

Entity Framework Tutorial, n.d. Available at: https://www.entityframeworktutorial.net/code-first/what-is-code-first.aspx [Accessed Feb 2, 2020]

Expensify, 2018. Available at: https://community.expensify.com/discussion/752/a-new-way-to-view-expenses-receipts-and-documents-in-expensify [Accessed April 11, 2020]

General Data Protection Regulation (GDPR). (2019). Recital 78 - Appropriate Technical and Organisational Measures | General Data Protection Regulation (GDPR). [online] Available at: https://gdpr-info.eu/recitals/no-78/ [Accessed 21 Oct. 2019].

GitHub, 2011, FakeDbSet. Available at: https://gist.github.com/lukewinikates/1309447) [Accessed May 15, 2020]

Hatzivasilis, G., Papaefstathiou, I. and Manifavas, C., 2015. Password Hashing Competition-Survey and Benchmark. IACR Cryptology ePrint Archive, 2015, p.265.

Itu.int. (2019). Statistics. [online] Available at: https://www.itu.int/en/ITUD/Statistics/Pages/stat/default.aspx [Accessed 21 Oct. 2019].

Juels, A. and Rivest, R.L., 2013, November. Honeywords: Making password-cracking detectable. In Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security (pp. 145-160).

Knockout, n.d. Knockout. Available at: https://knockoutjs.com/ [Accessed May 2, 2020]

Lee, V.C., Lam, K.W. and Son, S.H., 2002. Concurrency control using timestamp ordering in broadcast environments. The Computer Journal, 45(4), pp.410-422.).

Microsoft, 2012 – Bundling and Minification. Available at: (https://docs.microsoft.com/en-us/aspnet/mvc/overview/performance/bundling-and-minification [Accessed May 25, 2020]

Microsoft, 2014. ASP.NET Web Pages Using the Razor Syntax. Available at: https://docs.microsoft.com/en-us/aspnet/web-pages/overview/getting-started/introducing-razor-syntax-c [Accessed May 2, 2020]

Microsoft, 2019, Unit Testing Framework.

Microsoft, 2020 – Razor syntax reference for ASP.NET Core https://docs.microsoft.com/en-us/aspnet/core/mvc/views/razor?view=aspnetcore-3.1 [Accessed May 2, 2020].

Mint, n.d. Available at: https://www.mint.com/ [Accessed April 12, 2020]

Mobilligy, 2020. Prism. Mobile app. version 2.12.283. Available from: https://apps.apple.com/us/app/mobilligy/id522138897 [Accessed April 12, 2020]

Molnar, D. & Schechter, S., 2010. Self-Hosting vs. Cloud Hosting: Accounting for the Security Impact of Hosting in the Cloud. Self, pp.1–18.

Mozilla, 2019 – Server-side web frameworks. Available from: https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Web_frameworks [Accessed April 5, 2020]

Neyber, 2019 - One in five young people say their finances are out of control. Available from: http://www.moneynet.co.uk/one-in-five-young-people-say-their-finances-are-out-of-control/ [Accessed April 15, 2020]

Pathak, J., Basu, S. and Honavar, V., 2006, December. Modelling web services by iterative reformulation of functional and non-functional requirements. In International Conference on Service-Oriented Computing (pp. 314-326). Springer, Berlin, Heidelberg.)

Pearson, S., 2013. Privacy, Security and Trust in Cloud Computing. Privacy and Security for Cloud Computing.

Prescott, C., 2019. Office for National Statistics: Internet Users 2019. Available at: https://www.ons.gov.uk/businessindustryandtrade/itandinternetindustry/bulletins/internetusers/2019#:~:text=Virtually%20all%20adults%20aged%2016,up%20from%2090%25%20in%202018. [Accessed March 20, 2020]

Prism, 2020 – Prism Supports More Billers Than Any Other App. Available at: https://www.prismmoney.com/directory [Accessed April 10, 2020]

Quicken, n.d. Available at:(https://www.quicken.com/) [Accessed April 10, 2020]

Quicken, n.d. Available at: https://www.quicken.com/compare [Accessed April 10, 2020]

Qureshi, A., 2017. Analysis of Security in Cloud Hosted Service & Self Hosted Services.

StatCounter, 2020. Desktop Operating System Market Share Worldwide – May 2020. Available at: https://gs.statcounter.com/os-market-share/desktop/worldwide [Accessed June 1, 2020]

StatCounter, 2020. Desktop Operating System Market Share United Kingdom – May 2020. Available at: https://gs.statcounter.com/os-market-share/mobile/united-kingdom [Accessed June 1, 2020]

StatCounter, 2020. Mobile Operating System Market Share Worldwide - May 2020. Available at: https://gs.statcounter.com/os-market-share/mobile/worldwide [Accessed June 1, 2020]

TheBalance, 2020 - The 8 Best Personal Finance Software Options of 2020. Available at: https://www.thebalance.com/best-personal-finance-software-4171938 [Accessed 10 December 2019]

The Money Advice Service, 2015. Press release: Four out of 10 adults are not in control of their finances – new strategy launched to improve UK's financial capability. Available at: (https://www.moneyadviceservice.org.uk/en/corporate/four-out-of-10-adults-are-not-in-control-of-their-finances-new-strategy-launched-to-improve-uks-financial-capability). [Accessed April 25, 2020]

The Money Advice Service, 2017. One in six people in the UK burdened with financial difficulties. Available at: https://www.moneyadviceservice.org.uk/en/corporate/one-in-six-people-in-the-uk-burdened-with-financial-difficulties [Accessed April 25, 2020]

Toshl, n.d. Toshl. Available at: https://toshl.com/ [Accessed April 9, 2020]

Toshl Inc, 2020. Toshl. Mobile App. Version 3.4.6. Available at: https://apps.apple.com/us/app/toshl-finance-best-budget/id921590251 [Accessed April 9, 2020]

TutorialsTeacher, n.d. – ASP.NET MVC Architecture. Available at: https://www.tutorialsteacher.com/mvc/mvc-architecture [Accessed March 27, 2020]

University of Hull, 2019. Honours Stage Project Specification

(w3schools, n.d.) – ASP and ASP.NET Tutorials. Available at: https://www.w3schools.com/asp/default.asp [Accessed March 25, 2020]

You Need a Budget, 2020 – Website home page. Available at: https://www.youneedabudget.com/ [Accessed April 12, 2020]

(You Need A Budget, 2018) – Introducing YNAB's API. Available at: https://www.youneedabudget.com/introducing-ynabs-api/ [Accessed April 12, 2020]

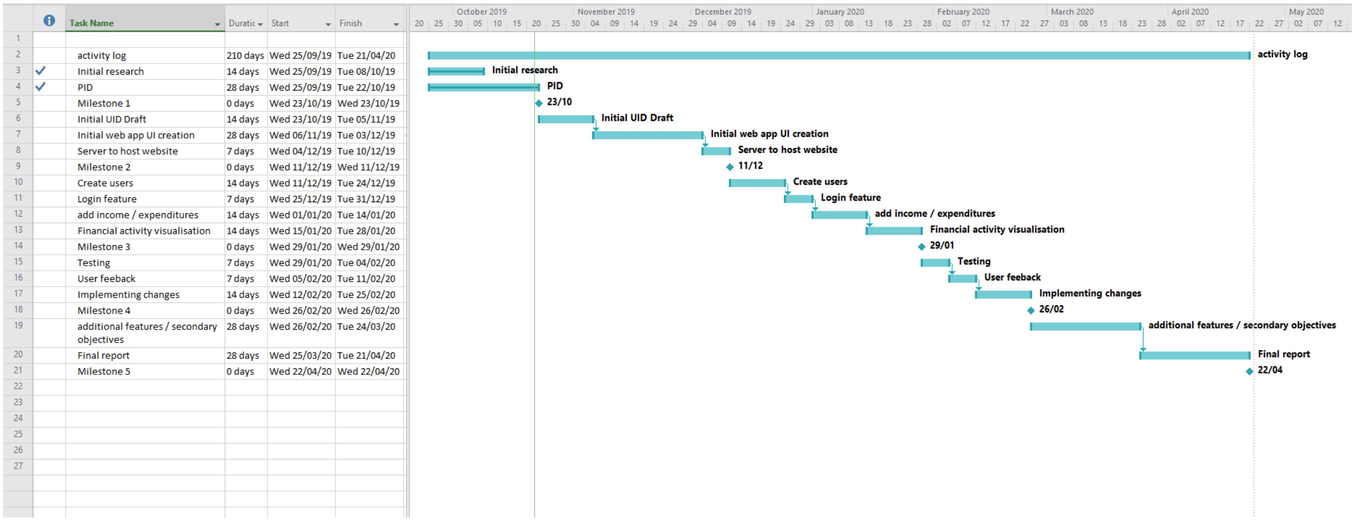
(You Need A Budget, 2020) YNAB 4 to NEW YNAB Upgrade. Available at: https://www.youneedabudget.com/landing/ynab4_upgrade/ [Accessed April 12, 2020]

Appendix A – Original time plan

Task		**Description**	**Expected duration (weeks)**	
Activity Log		Continuously update the activity log on progress and decision making over the course of the project.	Continuous	
Initial research		Reading around the topic of choice, finding books, reports and websites for sourcing information.	4	
PID		Project initiation document – including a rough specification for the project, with methodologies, tools, risk analysis and time planning.		
Milestone 1				
Initial UID Draft		Creation of a user interface design draft – either drawn on paper, whiteboard, or a digital drawing application.	2	
Initial web app UI creation		Coded implementation of a website, with the UI layout completed via use of html and CSS. Includes website theme and general design which will remain consistent over the application.	4	
Server to host website		Organise a server to host the website so that can be accessed by others (potential users).	1	
Milestone 2				
Implement working base features:	Create users	Ability to add users, with relevant information saved about the user including first name, last name(s), address, contact details, D.O.B. and potentially more.	2	7
	Login feature	Ability to login to a user account using a valid username and password combination.	1	
	Add income /expenditures	Give the user the ability to add an income or expenditure either in the form of a one-off transaction or a recurring one.	2	
	Financial activity visualisation	Have a visualisation of user financial activity including implementations of charts and graphs and breakdown by spending category.	2	
Milestone 3				
Testing		Perform testing of the application to ensure that all features run as expected with minimal bugs. This testing will be continuous, however should be completed thoroughly before allowing users to give feedback on the application.	Continuous	
User feedback		Achieve feedback from users who are given free reign of the application. Obtain useful information regarding what works well, what does not work well, what could be improved, what should be removed, and any potential features that the user would like to be implemented.	1	
Implementing changes		Incorporating any changes suggested by the user that would benefit the application upon implementation.	2	
Milestone 4				

Implementation of any additional features / secondary objectives	Given a surplus of time following all primary objectives being achieved, any remaining secondary objectives or any additional features that could be implemented in this remaining time period may be completed.	4
Final report	Delivering the final report, which should be continuously added to over the course of the project. And should break down and provide a thorough review and analysis of the entire project from start to finish. Due week 11 of semester 2.	4 / continuous
Total effort	Project completion with the application developed and the final report completed. Total number of expected weeks for achieving this.	29
Milestone 5 – Completed project		

Appendix B – Original Schedule Gantt Chart



	ℹ	Task Name	Duratic	Start	Finish
1					
2		activity log	210 days	Wed 25/09/19	Tue 21/04/20
3	✓	Initial research	14 days	Wed 25/09/19	Tue 08/10/19
4	✓	PID	28 days	Wed 25/09/19	Tue 22/10/19
5		Milestone 1	0 days	Wed 23/10/19	Wed 23/10/19
6		Initial UID Draft	14 days	Wed 23/10/19	Tue 05/11/19
7		Initial web app UI creation	28 days	Wed 06/11/19	Tue 03/12/19
8		Server to host website	7 days	Wed 04/12/19	Tue 10/12/19
9		Milestone 2	0 days	Wed 11/12/19	Wed 11/12/19
10		Create users	14 days	Wed 11/12/19	Tue 24/12/19
11		Login feature	7 days	Wed 25/12/19	Tue 31/12/19
12		add income / expenditures	14 days	Wed 01/01/20	Tue 14/01/20
13		Financial activity visualisation	14 days	Wed 15/01/20	Tue 28/01/20
14		Milestone 3	0 days	Wed 29/01/20	Wed 29/01/20
15		Testing	7 days	Wed 29/01/20	Tue 04/02/20
16		User feeback	7 days	Wed 05/02/20	Tue 11/02/20
17		Implementing changes	14 days	Wed 12/02/20	Tue 25/02/20
18		Milestone 4	0 days	Wed 26/02/20	Wed 26/02/20
19		additional features / secondary objectives	28 days	Wed 26/02/20	Tue 24/03/20
20		Final report	28 days	Wed 25/03/20	Tue 21/04/20
21		Milestone 5	0 days	Wed 22/04/20	Wed 22/04/20
22					
23					
24					
25					
26					
27					

Appendix C – Risk Analysis Table

Risk	**Severity (Low – 1 Medium – 2 High – 3)**	**Likelihood (Very low – 1 Very high – 5)**	**Significance (Severity x Likelihood)**	**Steps to avoid / contingency planning**
Hard drive failure / data wiped	3	2	6	Perform regular backups / incorporate use of a version control system so that in the event of data loss, a checkout from a repository could be performed.
Potential security risks of using real time data	3	1	3	Although this risk could have serious implications, it is unlikely that this application will incorporate the use of real time data.
RSI or eye strain when using computer terminal	1	2	2	Use of appropriate posture, working environment and screen brightness. Take regular breaks / avoid working for long intervals to avoid these potential risks.
Supervisor / customer / client departure	2	1	2	Ensure you get in contact with the appropriate person of authority as soon as possible after an incident like this occurs, and maintain a constant dialogue, updating upon the current scenario.
Illness / unforeseen circumstances	1	4	4	Given then event of illness or likewise, a deadline extension or mitigating circumstances should be applied for.
Underestimation of task completion time	3	3	9	All primary objectives should be achieved a suitable time prior to the deadline. It should be ensured not to fall behind schedule as dependencies could push back the project further. The estimated dates in 'Tasks and Milestones' are also potentially overestimates due to Parkinson's law.