

National College of Ireland

Bachelor of Science (Honours) in Computing

Software Development

2021/2022

Alan Mellowes

18467362

18467362@student.ncirl.ie

High availability Fintech Application

Technical Report

Contents

Executive Summary	4
1.0 Introduction	4
1.1. Background	4
1.2. Aims.....	5
1.3. Technology	6
1.4. Structure	8
2.0 System.....	9
2.1. Requirements.....	9
2.1.1. Functional Requirements.....	9
2.1.2 Non-Functional Requirements	15
1.1.1.1. Use Case Diagram	16
1.1.1.2. Requirement 1: Send money to a saved contact	16
1.1.1.3. Description & Priority.....	16
1.1.1.4. Use Case	16
1.1.1.5. Requirement 2: Send money to a acquaintance via PayPal API	19
1.1.1.6. Description & Priority.....	19
1.1.1.7. Use Case	19
1.1.1.8. Requirement 3: Search for cryptocurrencies	22
1.1.1.9. Description & Priority.....	22
1.1.1.10. Use Case	22
1.1.1.11. Requirement 4: Add money to vault.....	24
1.1.1.12. Description & Priority.....	24
1.1.1.13. Use Case	24
1.1.1.14. Requirement 5: send money to a friend via GooglePay	27
1.1.1.15. Description & Priority.....	27
1.1.1.16. Use Case	27
1.1.1.17. Requirement 5: Register new account.....	30
1.1.1.18. Description & Priority.....	30
1.1.1.19. Use Case	30
1.1.1.20. Requirement 8: Keep track of money spent	32
1.1.1.21. Description & Priority.....	32
1.1.1.22. Use Case	32
1.1.1.23. Requirement 9: Log out	34
1.1.1.24. Description & Priority.....	34
1.1.1.25. Use Case	34

3.0 Data Requirements	36
3.1 User Requirements	36
3.2 Environmental Requirements	36
3.3 Usability Requirements	37
4.0 Design & Architecture	38
4.1 Class diagram	39
4.2 Flowchart	44
4.3 User Personas	45
4.3.1 Persona 1	46
4.3.2 Persona	47
4.3.2.1 Scenario 2 – Ian Mac	47
5.0 Implementation	48
5.1 Failover Clustering	48
5.2 PayPal Activity	51
5.3 Google Pay Activity	52
5.4 Stripe Activity	53
5.5 Daily analytics	55
5.6 Step Activity	60
6.0 Graphical User Interface (GUI)	64
6.1 GUI – Wireframes	64
7.0 Testing	77
8.0 Evaluation	94
9.0 Conclusions	95
10.0 Further Development or Research	96
11.0 References	97
12.0 Appendices	99
3.4 Project Proposal	99
1.0 Objectives	100
2.0 Background	100
3.0 State of the Art	101
4.0 Technical Approach	101
5.0 Technical Details	102
6.0 Special Resources Required	103
7.0 Project Plan	103
8.0 Testing	104
9.0 References	105

2.2.	Ethics Approval Application (only if required)	105
3.5	Reflective Journals	105

Executive Summary

The objective of this report is to provide an application intended to be used by users worldwide. The aim of my project is to provide an application that presents users with multiple unique features you would normally need a variety of applications for. My application looks at running through required steps to implementing a highly availability application with a secure SQL database. I have developed an application using android studio that allows users to send money, make payments, manage their savings, and monitor cryptocurrencies. The main scope of my application is the fintech application side with similar functionality to Revolut, it will look at dealing with transactions, savings monitoring and investment management. In my report I will talk about how I would go about providing high availability if the funding was made available. I have implemented a secure database to simply compliment my application.

Having used multiple applications for these different features I believe that this application has potential in today's market. The purpose of this report is to simply dive into some technical details revolving my application. In the report I go through some important questions about my project and what my overall goal is. As well as running through some use cases and describing some snippets of code to explain how I implemented some of my applications core functionalities.

1.0 Introduction

1.1. Background

The fintech sector is an ever-progressive platform, fintech statistics show from case studies performed in Australia, the UK, and the US, have shown us that the fintech statistics have risen from 16% in 2015, rising to 31% in 2017 and reaching up to 60% in 2019. In a popular technology article, it was mentioned that *"The global **fintech market** was worth \$127.66 billion in 2018, and it is expected to reach \$309.98 billion at a CAGR Of 24.8% through 2022."* [prenewswire] [2019]. It is also estimated that 75% of millennials in the US switched to digital banking, I expect that this will lead to massive growth opportunities for digital finance services.

To this day, companies still seem to be experiencing issues with providing high availability and highly secure systems. High availability, database security and developing have always been areas of interest to me. I believe that implementing these three factors to a good standard is necessary when creating a mobile application for an end user. When a system or database fails, organisations require high availability protection. They require high availability to keep their systems up and running and to mitigate loss of revenue, brand damage and unhappy customers. The high availability method I plan on describing how to implement is known as 'Failover clustering'. Failover clustering is a collection of separate servers that collaborate to boost the availability of systems and applications. If a service fails, the services hosted on that node can be migrated automatically to a different, available node.

There are many reasons as to why database security is important seen as sensitive and personal data is on the line. Firebase help secure your data with Firebase Security Rules that are flexible and extensible.

1.2. Aims

"The word "fintech" is simply a combination of the words "financial" and "technology". It describes the use of technology to deliver financial services and products to consumers". [Central bank, 2022] The helped me define the scope of my project, I aim to develop a fully functionable fintech application that deals with payments, investment management, savings monitoring whilst with the help of possible future funding, providing high availability whilst also ensuring the customers data is stored securely in a Firebase database. For my application to be fully functionable it is necessary that it is possible to deal with transactions, an example of this would be sending money to someone or paying for a product. My aim for the savings monitoring feature is to provide the user with access to a vault in which they can insert and monitor their virtual money. I aim on providing the user with daily analytics to keep track of what they are spending their money on.

A user can insert money on multiple occasions, it does not have to be one transaction. The money inserted in this vault will be added up together and categorised by different items of choice, the total balance will then be compared to the savings goal in which the user originally entered. For example, when a user first opens a vault, they will be prompted to insert a value which corresponds to their overall savings goal. Let us say that a user has a goal of \$1000. Whenever the user adds money to this vault, it will be added to the original amount that was in the vault and be presented to the user.

When dealing with the investment management feature, my aim is to provide the user with the top 100 cryptocurrencies and their current market price. Let us say the user is invested in Bitcoin, navigating to the investment management page the user will be able to see what the current market price is for the coin as well as having the ability to search for other cryptocurrencies.

I cannot stress the importance of high availability, essentially it is one or more databases that fail over together. I aim to provide high availability in the future by walking through the steps of implementing failover clustering. My justification is that failover clusters have a multitude of practical uses, including highly available clustered roles and highly available or continuously available file sharing storage. The only reason I can't provide windows server failover clustering is cost. To provide this feature you must own a standard version of Windows Server which is priced around 600 euro.

1.3. Technology

Technologies to be used:

- Java
- Cryptocurrency API
- Google Pay API
- PayPal API
- Stripe API
- Failover Clusters
- Firebase database
- Detectors
- Sensors
- Espresso testing framework

When it comes to developing my mobile application, I have been using Android studio. My reasoning is, Android Studio is considered the official IDE for android development. I decided Java would be most suitable for my application, android studio's source code is in Java which also led me to pick this as my IDE. From researching a variety of programming languages, I found Java to be the most suitable. This was due to a variety of reasons; Java is a highly regarded language for banking applications where security is amongst one of the main concerns. Seen as I plan to deal with everyday banking tasks such as transactions and money storage, I figured this was a suitable candidate.

Stripe API

On the topic of banking, Stripe is a payment processing software and application programming interface I plan to implement as it is compatible with mobile applications. The Stripe API is organized around REST. Their API accepts form-encoded request bodies, delivers JSON-encoded replies, and employs standard HTTP response codes, authentication, and verbs. [*stripe docs, 2020*]

Database

The database I use for my application will look at storing sensitive data such as personal user details. This is an important topic and should be considered to not only prevent hacks but to also ensure the customer feels like their data is in safe hands instead of feeling vulnerable which is the opposite of what we want. Now more than ever, it is important for businesses to comprehend the risks and consequences at hand when dealing with data breaches. Losing customer trust is one of the long-term repercussions of a data breach, in the event of a data breach, afterwards the company will have to deal with all sorts of implications. The company will be under pressure as their brand reputation is at stake. If customers believe that the company is neglecting to take additional steps to prevent future security breaches, they may seek for an alternative programme that offers the same functionality but with improved security. From January to September 2020, it was estimated that around 36 billion records were compromised, according to these statistics it has been found that cybercriminals are still the main drivers behind these breaches. It can be said that Database security is the business of

the entire organization as all people use the data held in the organizations database and any loss or corruption would affect the day-to-day operation of the organization and the performance of the people.

Encryption provides a way to encode data so that only authorized users can view the data in an unencrypted state overall limiting the exposure of sensitive data.

Failover cluster

A high-availability cluster, also known as a failover cluster, ensures that if a fault causes one system to fail, another system can be smoothly utilised to preserve the application's availability. The cluster I plan on documenting will consist of 4 nodes which are essentially servers. One of the machines will be a domain controller, another will be a storage server whilst the other two are the two failover nodes. On these machines it is essential that windows server standard edition is installed. Before a failover, each member of an active high availability cluster actively processes data.

PayPal API

I have demonstrated how to utilise the Native Checkout SDK to develop a safe and secure payment method for my mobile application. I've incorporated a Pay Now checkout option, which enables the user to transfer funds to another user's PayPal account. The SDK supports integrations on both the client and server sides. [*developer.paypal, 2022*]

Google Pay API

To submit a request for payment and receive a response, Google Pay supports generic Unified Payments Interface (UPI) API calls. As I mentioned I intend on this application being suitable for users worldwide, this feature is only applicable for users with an Indian Google pay account. I have implemented Google Pay payments with the help of the official documentation. [*Google Pay, 2021*]

Espresso Framework

As my programme involves extensive user interaction with the user interface, I wanted to write UI unit tests to analyse the numerous application components with which the user will interact. To assist me with testing my user interface I have implemented Espresso and ran tests for multiple test cases. [*developer.android, 2021*]

Cryptocurrency API

With the aid of the aid of the official documentation of CoinMarketCap's cryptocurrency API I have managed to retrieve the top 100 cryptocurrencies in today's market and display the results to the user. The CoinMarketCap API is a set of RESTful JSON endpoints with exceptional performance. [*CoinMarketCap, 2021*]

1.4. Structure

This report is a technical report which explores every detail of the mobile application I am developing. The report consists of 12 main sections however there are also a variety of subsections. In section 1 of the report, I answer some important questions about my project. I discussed the goals of my project, why I decided to undertake it, the technology I plan on using to achieve the tasks I have set out to do as well as how I plan on using specific technologies.

In section 2 I have listed all functional and non-functional requirements in order whilst also providing multiple use case diagrams as well as a description of what is being carried out in the use case. In this section I also go on to describe the design, system architecture and components used followed up by describing the main algorithms used in my project. I provided some screenshots of my applications most visited screens in which I designed using Adobe XD.

Section 3 of my report goes onto exploring user requirements and limitations, environment requirements and limitations, as well as usability requirements and limitations.

In section 4 of the report, I designed a class diagram using lucid chard to show what methods are contained in each activity as well as giving a general idea of how the activity relate to each other. A more detailed flow of my application is depicted in the flowchart diagram I have designed. I have also created two user personas to assist me in creating a system that users require, rather than a system that users desire.

Section 5 of the report goes into detail about how I implemented some of my applications core functionality. The implementation section is extremely important as I also provide the necessary steps that should be followed to implement windows server failover clustering. This section includes snippets of code along with explanations as what the code does.

Section 6 simply shows the stages of my applications user interface design. I originally did some general wireframes to get an idea of page layout and overall app structure. I later moved to AdobeXD and designed a blue/white colour schemed navigable and consistent user interface.

In section 7 I explored different types of testing. I ran through manual tests in which I documented after each merge was performed. I learn how to implement automated testing using Espressos testing framework. These tests can be run through android studio by running the testing activities instead of the application.

Section 8 covers an evaluation of what my application successfully achieved in comparison to the original proposal.

In section 9 I concluded my findings and provided a list of the advantages and disadvantages of my project. I also underlined the strength and weakness of my application.

Section 10 explores what future work I would like to include. I mention that with the help of some funding I would be able to successfully implement windows server failover clustering.

2.0 System

2.1. Requirements

2.1.1. Functional Requirements

User requirement Make payment using Stripe API

<i>Main Task</i>	A user wishes to upgrade their current card. The user goes to the upgrade card activity and activates the Stripe API. The user then completes their payment.
<i>Subtask</i>	User has to be logged into wave and PayPal
<i>Functional system requirement</i>	<ol style="list-style-type: none">1. User opens wave2. User logs into their Wave account3. User selects 'payments' activity on the home screen.4. Users selects the 'Upgrade Card' activity.5. User clicks on the card being sold for 10 euro6. User is presented with Stripe API7. User enters card details8. User confirms payment
<i>Description</i>	Users can upgrade to a better credit card. The card is being sold for 10 euro and offers incentives such as Free travel insurance and unlimited withdrawals without any fees being applied.

User requirement Send money via PayPal API

<i>Main Task</i>	A user is presented with multiple options of payment. They decided they would like to send money to a contact via PayPal. They choose the PayPal payment method, enter their login details, recipients name, and payment amount. The user then confirms the payment.
<i>Subtask</i>	User has to be logged into wave and PayPal
<i>Functional system requirement</i>	<ol style="list-style-type: none">1. User opens wave2. User logs into their Wave account3. User selects 'payments' activity on the home screen.4. Users selects the 'PayPal' payment method.5. User enters recipient's email6. User enters amount they wish to send7. User logs into PayPal account8. User confirms payment
<i>Description</i>	Users can make a payment to another person using PayPal API

User requirement Send money via Google API

<i>Main Task</i>	A user is presented with multiple options of payment. They decided they would like to send money to a contact via Google Pay. They choose the Google Pay payment method, enter their login details, recipients UPID, and payment amount. The user then confirms the payment.
<i>Subtask</i>	User has to be logged into wave and own an Indian Google Pay account
<i>Functional system requirement</i>	<ol style="list-style-type: none">1. User opens wave2. User logs into their Wave account3. User selects 'payments' activity on the home screen.4. Users selects the 'Google Pay' payment method.5. User enters recipient's details6. User enters amount they wish to send7. User presses pay and gets redirected to Google Pays API8. User confirms payment
<i>Description</i>	Users from India can make a payment to another person from India using Google Pay API.

User requirement Create an account stored in Firebase database

<i>Main Task</i>	If the user wishes to access the applications features, it is necessary that they have an existing account. Creating an account gives the user the option to send money, purchase items, keep track of their savings, and monitor cryptocurrencies
<i>Subtask</i>	User must have a valid email address.
<i>Functional system requirement</i>	<ol style="list-style-type: none">1. User opens wave2. User clicks register button3. User fills out form.4. Users completes submission.
<i>Description</i>	User can create an account

User requirement Login

<i>Main Task</i>	If the user wishes to access the applications features, it is necessary that they have an existing account. When the user logs in they are presented with the opportunity to send money, purchase items, keep track of their savings, and monitor cryptocurrencies
<i>Subtask</i>	User must have a valid email address.
<i>Functional system requirement</i>	<ol style="list-style-type: none">1. User opens wave2. User clicks login button3. User fills out form.4. Users submits credentials.
<i>Description</i>	User can login to an existing account

User requirement Add money to savings vault

<i>Main Task</i>	The user heads to the savings activity. The user decided to create a vault in which they enter the amount of money they would like to save. The user can add virtual money.
<i>Subtask</i>	User must be logged in.
<i>Functional system requirement</i>	<ol style="list-style-type: none">1. User opens wave2. User logs into their Wave account3. User clicks savings activity4. Users clicks on 'Vault' activity5. User inputs amount6. User selects option on what they are saving for
<i>Description</i>	User can add money to a savings vault

User requirement Remove money to savings vault

<i>Main Task</i>	The user heads to the savings activity. The user decided to create a vault in which they enter the amount of money they would like to save. The user has entered the wrong value. The user deletes the item they created.
<i>Subtask</i>	User must be logged in.
<i>Functional system requirement</i>	<ol style="list-style-type: none">1. User opens wave2. User logs into their Wave account3. User clicks savings activity4. Users clicks on 'Vault' activity5. User deletes item in vault.
<i>Description</i>	User can withdraw from savings vault

User requirement Update money in savings vault

<i>Main Task</i>	The user heads to the savings activity. The user decided to create a vault in which they enter the amount of money they would like to save. The user has entered the wrong value. The user decides to update the item they originally entered.
<i>Subtask</i>	User must be logged in.
<i>Functional system requirement</i>	<ol style="list-style-type: none">1. User opens wave2. User logs into their Wave account3. User clicks savings activity4. Users clicks on 'Vault' activity5. User updates item in vault.
<i>Description</i>	User can update money in savings vault.

User requirement Keep track of money spent today

<i>Main Task</i>	The user heads to the savings activity. The user wants to budget their money. The user heads to the 'Money spent today' activity. The user enters money they have spent today. The user can apply CRUD functionality to items in this activity.
<i>Subtask</i>	User must be logged in.
<i>Functional system requirement</i>	<ol style="list-style-type: none">1. User opens wave2. User logs into their Wave account3. User clicks savings activity4. User clicks on 'Money spent today' activity5. User creates item6. User enters details, how much money they spent etc.7. User can perform CRUD functionality.
<i>Description</i>	User can budget their spendings. This activity allows the user to monitor any money they have spent today.

User requirement Keep track of money spent this week

<i>Main Task</i>	The user heads to the savings activity. The user wants to budget their money. The user heads to the 'Money spent this week' activity. User analyses data.
<i>Subtask</i>	User must be logged in.
<i>Functional system requirement</i>	<ol style="list-style-type: none">1. User opens wave2. User logs into their Wave account3. User clicks savings activity4. User clicks on 'Money spent this week' activity5. User analyses data
<i>Description</i>	User can budget their spendings. This activity allows the user to monitor their spendings from this week.

User requirement Keep track of money this month

<i>Main Task</i>	The user heads to the savings activity. The user wants to budget their money. The user heads to the 'Money spent this month' activity. User analyses data.
<i>Subtask</i>	User must be logged in.
<i>Functional system requirement</i>	<ol style="list-style-type: none">1. User opens wave2. User logs into their Wave account3. User clicks savings activity4. User clicks on 'Money spent this month' activity5. User analyses data.
<i>Description</i>	User can budget their spendings. This activity allows the user to monitor their spendings from this month.

User requirement Search for a cryptocurrency

<i>Main Task</i>	The user heads to the cryptocurrency activity. The user searches for a cryptocurrency.
<i>Subtask</i>	User must be logged in.
<i>Functional system requirement</i>	<ol style="list-style-type: none">1. User opens wave2. User logs into their Wave account3. User clicks cryptocurrency activity4. User searches for a cryptocurrency5. User analyses current price.
<i>Description</i>	Users can monitor the top 100 cryptocurrency prices. The cryptocurrencies are retrieving using coinmarketcaps API.

User requirement Monitor daily spending analytics

<i>Main Task</i>	The user heads to the daily analytics activity. The user presses the pie chart to analyse what they have spent their money on.
<i>Subtask</i>	User must be logged in. User must have logged daily spendings.
<i>Functional system requirement</i>	<ol style="list-style-type: none">1. User opens wave2. User logs into their Wave account3. User clicks savings activity4. User clicks the daily spending analytics activity5. User clicks pie chart6. User analyses daily spendings
<i>Description</i>	Users is presented with a pie chart which depicts what they have spent their money on in the last 24 hours.

User requirement Nearest ATM

<i>Main Task</i>	The user heads settings activity. The user clicks the map activity.
<i>Subtask</i>	User must be logged in. User must have Google maps downloaded.
<i>Functional system requirement</i>	<ol style="list-style-type: none">1. User opens wave2. User logs into their Wave account3. User clicks settings activity4. User clicks the daily spending analytics activity5. User clicks map activity6. User locates nearest ATM
<i>Description</i>	Users that may wish to withdraw money from an ATM but are unsure as to where the nearest ATM is. The user can click the map icon which will direct them to the nearest ATM.

User requirement Health activity

<i>Main Task</i>	The user heads to settings activity. The user presses the health activity and monitors their current step count.
<i>Subtask</i>	User must be logged in. User must have accepted app permissions.
<i>Functional system requirement</i>	<ol style="list-style-type: none">1. User opens wave2. User logs into their Wave account3. User clicks settings activity4. User clicks the health activity5. User monitors their step count
<i>Description</i>	The health of my users is important. This activity helps users feel good about their achievements as it returns the total count of steps they have made since they have opened the app, as well as returning the step count since they first installed my app.

2.1.2 Non-Functional Requirements

A non-functional requirement describes how a system should behave and what its functional limitations are. Non-functional requirements server a great deal of importance *“Well you can develop application only with Functional requirement but without NFR the product will be buggy, non-reliable and incomplete.” [Vishwas Ng, 2019].*

Non-functional	Rationale
Emails should have a maximum latency of one minute.	When a user creates an account for the first time, they should expect to receive a verification email within 1 minutes of doing so. This principle also applies when a user resets their password.
Each request should be processed in 7 seconds or less.	This non-functional requirement applies to many processes in my application. Whether the user is creating an account, signing out, making a payment, or anything else, all of the following processes should take no more than 7 seconds to complete.
Security	The application will be secure. This ranges from makings sure payments are secure to storing user login credentials. The firebase database provides security features that protect any data uploaded to the application.
Availability	The term "application availability" refers to the ability of people to successfully user my application. Signing into an account you have registered in the firebase database can be used to gain access, if my application shows the expected content, it means it is available. We can improve overall application availability by carrying out tests to ensure that my application is capable of handling multiple scenarios created by the user.
Recoverability	Any application can cause disaster if it is not managed and secured properly. Using a disaster recovery method allows my application to relocate temporarily in the event of a security breach or natural disaster. A disaster recovery app ensures that a business can continue operations until it is safe to return to its original location or a new permanent location.
Capacity	This refers to the number of users who are online at the same time. This refers to the number of users my application can handle while each user performs actions such as paying, creating an account, and so on.
Scalability	This refers to my application's ability to handle an increase in workload when adding resources. This app should be scalable across all Android devices.

1.1.1.1. Use Case Diagram

1.1.1.2. Requirement 1: Send money to a saved contact

01 MakePayment	Purchase a better credit card	Make payment card using Stripe API	1
-------------------	----------------------------------	--	---

Identifier: 01.MakePayment

1.1.1.3. Description & Priority

Priority 1 (highest)

This requirement ("Purchase a better credit card") is considered one of the most important requirements within my application. This is due to it being one of the core functions, which is why the priority is so high.

The function 'MakePayment' allows the user to purchase an upgraded credit card. The rationale behind this, is that the upgraded credit card offers the user unlimited withdrawals with now fees applied as well as free travel insurance. The transaction is handled by Stripes API. The user is required to enter there credit/debit card information to make the payment of 10 euro.

1.1.1.4. Use Case

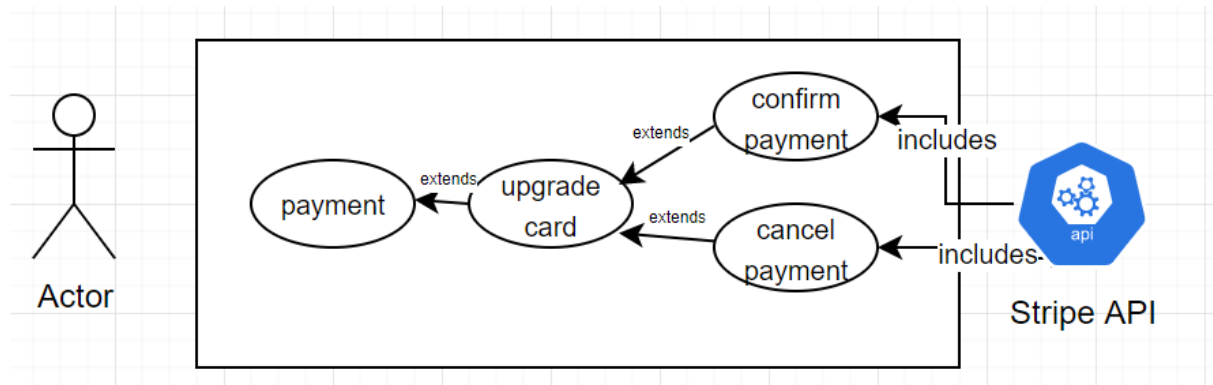
Scope

The scope of this use case is to purchase a credit card using the Stripe API. This will all be done within the 'payment' page which is navigated to by using the cardview.

Description

Users can upgrade to a better credit card. The card is being sold for 10 euro and offers incentives such as Free travel insurance and unlimited withdrawals without any fees being applied.

Use Case Diagram



[FIG 1] [Upgrade card with Stripe API]

Flow Description

Precondition

The user must have access to the internet.

The user must be signed in.

Activation

This use case starts when an <Actor> is on the payments page and clicks the card view for upgrading to a new credit card.

Main flow

1. The <Actor> opens wave
2. The <Actor> logs into their wave account
3. The <Actor> is presented with the home page.
4. The <Actor> selects the 'payment' activity.
5. The <Actor> selects the card view assigned to upgrading their credit card.
6. The <Stripe API> prompts the <Actor> to enter their credit card details.
7. The <Actor> enters their credit card details.
8. The <Actor> confirms their payment.
9. The <Stripe API> handles the transaction.

Alternate flow

A1: Accidental selection

1. The <Actor> opens wave
2. The <Actor> logs into their wave account
3. The <Actor> is presented with the home page.
4. The <Actor> selects the 'payment' activity.
5. The <Actor> selects the card view assigned to upgrading their credit card.
6. The <Stripe API> prompts the <Actor> to enter their credit card details.
7. The <Actor> cancels their payment.

A2: Insufficient funds

1. The <Actor> opens wave
2. The <Actor> logs into their wave account
3. The <Actor> is presented with the home page.
4. The <Actor> selects the 'payment' activity.
5. The <Actor> selects the card view assigned to upgrading their credit card.
6. The <Stripe API> prompts the <Actor> to enter their credit card details.
7. The <Actor> enters their credit card details.
8. The <Actor> confirms their payment.
9. The <Stripe API> fails to handle the transaction due to insufficient funds.
10. The application notifies the user that the transaction has been cancelled.

Termination

The system presents the message "insufficient funds"

Post condition

The system goes into a wait state

1.1.1.5. Requirement 2: Send money to a acquaintance via PayPal API

02 sendMoneyPayPal	Allows the user to send money as they wish	Send money to a contact using PayPal API	1
-----------------------	--	--	---

Identifier: 02.sendMoneyPayPal

1.1.1.6. Description & Priority

Priority 1 (highest)

This requirement (“Send money to a new person”) is considered one of the most important requirements within my application. This is due to it being one of the core functions, which is why the priority is so high.

The function ‘sendMoneyPayPal’ allows the user to send money to another PayPal account as they wish. The user now has the option to send money to anyone considering they have their PayPal details.

1.1.1.7. Use Case

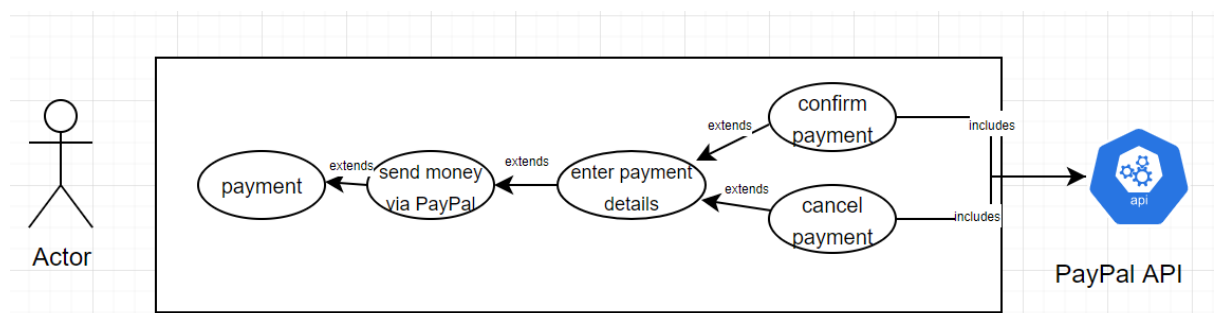
Scope

The scope of this use case is to send money a new person. This will all be done within the ‘payment’ page which is navigated to by using the cardview.

Description

The user may can send money to another PayPal account using the PayPal API.

Use Case Diagram



[FIG 2] [PayPal payment use case]

Flow Description

Precondition

The user must have access to the internet.

The user must be signed into their wave account.

The user must be signed into their PayPal account.

Activation

This use case starts when an <Actor> is on the payments page and decides to press the send money to a new person via PayPal card view.

Main flow

1. The <Actor> opens wave
2. The <Actor> logs into their wave account.
3. <Actor> selects payments activity on home screen.
4. The <Actor> selects the 'PayPal' activity.
5. The <Actor> fills out payment details.
6. The <Actor> confirms presses pay button to send money.
7. <PayPal API> presents user with login page.
8. The <Actor> logs into their PayPal account.
9. <PayPal API> presents user with review of payment.
10. The <Actor> confirms the payment

Alternate flow

A1: Accidental selection

1. The <Actor> opens wave
2. The <Actor> logs into their wave account.
3. <Actor> selects payments activity on home screen.
4. The <Actor> selects the 'PayPal' activity.
5. The <Actor> fills out payment details.
6. The <Actor> confirms presses pay button to send money.
7. <PayPal API> presents user with login page.
8. The <Actor> logs into their PayPal account.
9. <PayPal API> presents user with review of payment.
10. The <Actor> cancels the payment.
11. <PayPal API> closes and presents user with the previous activity.

A2: Insufficient funds

1. The <Actor> opens wave
2. The <Actor> logs into their wave account.
3. <Actor> selects payments activity on home screen.
4. The <Actor> selects the 'PayPal' activity.
5. The <Actor> fills out payment details.
6. The <Actor> confirms presses pay button to send money.
7. <PayPal API> presents user with login page.
8. The <Actor> logs into their PayPal account.
9. <PayPal API> presents user with review of payment.
10. The <Actor> confirms the payment
11. <PayPal API> notifies user that they have insufficient funds.
12. <PayPal API> closes and presents user with the previous

Termination

The system presents the message "insufficient funds"

Post condition

The system goes into a wait state

1.1.1.8. Requirement 3: Search for cryptocurrencies

03 searchCrypto	Search for cryptocurrencies	the user may search through the list of cryptocurrencies pulled by API	1
--------------------	--------------------------------	---	---

Identifier: 03.searchCrypto

1.1.1.9. Description & Priority

Priority 1 (highest)

This requirement (“Search for cryptocurrencies”) is considered one of the most important requirements within my application. This is due to it being one of the core functions, which is why the priority is so high.

The function “searchCrypto” provides the user with the functionality of search for the cryptocurrency they are curious about. The list consists of 100 cryptocurrencies which are retrieved using coinmarketcaps API. It would be inefficient for the user to scroll through all 100 cryptocurrencies just to look for one in particular, that is why I implemented this search requirement.

1.1.1.10. Use Case

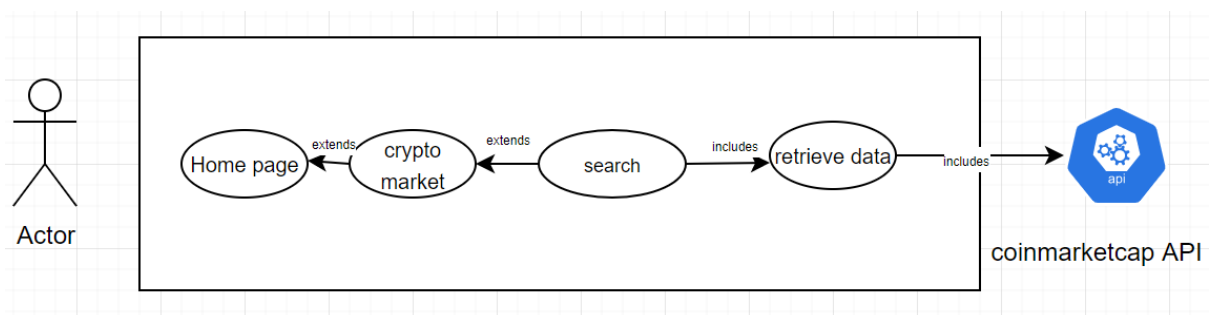
Scope

The scope of this use case is to search through a list of 100 cryptocurrencies and retrieve results which match the user’s input. This will all be done within the “cryptocurrencies” page which is navigated to by using the card view.

Description

The user can monitor the top 100 cryptocurrencies in today’s market using coinmarketcap’s API. The user has the option to search for the cryptocurrency by its name.

Use Case Diagram



[FIG 3] [Search use case]

Flow Description

Precondition

The user must have access to the internet.

The user must be logged in.

Activation

This use case starts when an <Actor> is on the investment management page and decides to search for cryptocurrencies filtered by their text input.

Main flow

1. The <Actor> navigates to investment management page
2. The <coinmarketcap API> retrieves the top 100 cryptocurrencies in today's market.
3. The <Actor> presses search and types in text input.
4. The <System> filters cryptocurrencies by name and returns results which contain the same text as users' input.

Termination

The system presents the message "unable to load data"

Post condition

The system goes into a wait state

1.1.1.11. Requirement 4: Add money to vault

04 CRUDMoney	Add money to vault	The user can apply CRUD functionality within their savings vault	1
--------------	--------------------	--	---

Identifier: 04.CRUDMoney

1.1.1.12. Description & Priority

Priority 1 (highest)

This requirement (“Add money to vault”) is considered one of the most important requirements within my application. This is due to it being one of the core functions, which is why the priority is so high.

The functions involved in ‘CRUDMoney’ simply allows the user to add money to their already existing savings vault as well as allowing the users to update and delete the stored information. When adding money to the savings vault the user can enter information such as the amount, what the money is being saved for, and a personal note. At the top of the screen the user is presented with the total balance within the savings vault.

1.1.1.13. Use Case

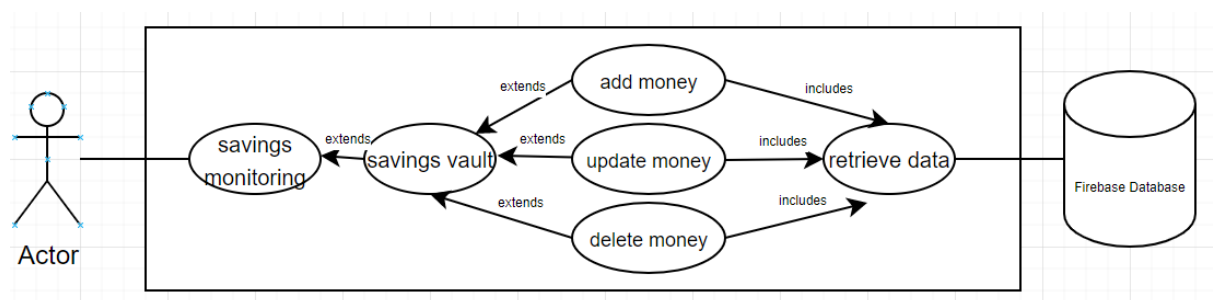
Scope

The scope of this use case is to perform CRUD functionality within their savings vault. This will all be done within the savings monitoring page which is navigated to by using the card view located on the home screen.

Description

The user can add money to their already existing savings vault as well as updating or deleting any existing funds.

Use Case Diagram



[FIG 4] [CRUD vault use case]

Flow Description

Precondition

The user must have access to the internet.

The user must be logged in.

Activation

This use case starts when an <Actor> is on the savings monitoring page and decides to press the add money button.

Main flow

1. The <Actor> opens wave.
2. The <Actor> logs into their wave account
3. The <Actor> clicks the savings activity on the home screen.
4. The <Actor> clicks on the 'Vault' card view.
5. The <Actor> adds money
6. The <Firebase database> stores the information inputted by the <Actor>
7. The <System> presents the total amount stored in the savings vault.

Alternate flow

A1 : Accidental selection

1. The <Actor> presses add money to vault button.
2. The system prompts the <Actor> asking to enter amount and confirm.
3. The <Actor> presses the cancel option
1. The use case continues at position 3 of the main flow.

A2: update item

1. The <Actor> opens wave.
2. The <Actor> logs into their wave account
3. The <Actor> clicks the savings activity on the home screen.
4. The <Actor> clicks on the 'Vault' card view.
5. The <Actor> adds money
6. The <Firebase database> stores the information inputted by the <Actor>
7. The <System> presents the total amount stored in the savings vault.
8. The <Actor> selects an item in the vault.
9. The <Actor> updates the details presented
10. The <Firebase database> updates the information selected by the <Actor>
11. The <System> presents the total amount stored in the savings vault.

A2: Delete an item

1. The <Actor> opens wave.
2. The <Actor> logs into their wave account
3. The <Actor> clicks the savings activity on the home screen.
4. The <Actor> clicks on the 'Vault' card view.
5. The <Actor> adds money
6. The <Firebase database> stores the information inputted by the <Actor>
7. The <System> presents the total amount stored in the savings vault.
8. The <Actor> selects an item in the vault.
9. The <Actor> deletes the selected item.
10. The <Firebase database> removes the information selected by the <Actor>
11. The <System> presents the total amount stored in the savings vault.

Termination

The user cancels the transaction.

Post condition

The system goes into a wait state

1.1.1.14. Requirement 5: send money to a friend via GooglePay

05 sendMoneyPayPal	Allows the user to send money as they wish	Send money to a contact using GooglePay API	1
--------------------	--	---	---

Identifier: 02.sendMoneyGooglePay

1.1.1.15. Description & Priority

Priority 1 (highest)

This requirement (“Withdraw money from vault”) is considered one of the most important requirements within my application. This is due to it being one of the core functions, which is why the priority is so high.

The function ‘sendMoneyGooglePay’ allows the user to send money to another Google Pay account as they wish. The user now has the option to send money to anyone considering they have their Google Pay details.

1.1.1.16. Use Case

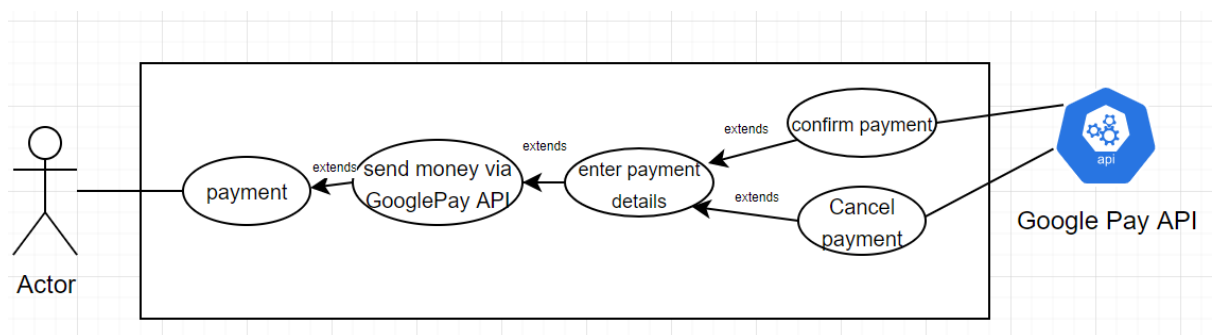
Scope

The scope of this use case is to send money a new person. This will all be done within the ‘payment’ page which is navigated to by using the card view.

Description

The user can make a payment to a friend, but it is required that they are from India.

Use Case Diagram



[FIG 5] [Google Pay use case]

Flow Description

Precondition

The user must have access to the internet.

The user must be logged in.

The user must own an Indian Google Pay account.

Activation

This use case starts when an <Actor> is on the payments page and decides to press the send money to a new person via PayPal card view.

Main flow

- i. The <Actor> opens wave
- ii. The <Actor> logs into their wave account.
- iii. <Actor> selects payments activity on home screen.
- iv. The <Actor> selects the 'GooglePay' activity.
- v. The <Actor> fills out payment details.
- vi. The <Actor> confirms presses pay button to send money.
- vii. <Google Pay API> presents user with login page.
- viii. The <Actor> logs into their Google Pay account.
- ix. <GooglePay API> presents user with review of payment.
- x. The <Actor> confirms the payment

Alternate flow

A1 : Accidental selection

1. The <Actor> opens wave
2. The <Actor> logs into their wave account.
3. <Actor> selects payments activity on home screen.
4. The <Actor> selects the 'Google Pay activity.
5. The <Actor> fills out payment details.
6. The <Actor> confirms presses pay button to send money.
7. <GooglePay API> presents user with login page.
8. The <Actor> logs into their Google Pay account.
9. <GooglePay API> presents user with review of payment.
10. The <Actor> cancels the payment.
11. <GooglePay API> closes and presents user with the previous activity.

A2: Insufficient funds

1. The <Actor> opens wave
2. The <Actor> logs into their wave account.
3. <Actor> selects payments activity on home screen.
4. The <Actor> selects the 'GooglePay activity.
5. The <Actor> fills out payment details.
6. The <Actor> confirms presses pay button to send money.
7. <GooglePay API> presents user with login page.
8. The <Actor> logs into their PayPal account.
9. <GooglePay API> presents user with review of payment.
10. The <Actor> confirms the payment
11. <GooglePay API> notifies user that they have insufficient funds.
12. <GooglePay API> closes and presents user with the previous

Termination

The system presents the message "insufficient funds"

Post condition

The system goes into a wait state

1.1.1.17. Requirement 5: Register new account

06 registerAccount	Register new account	A new user will be able to register a new account so they can attain access to the app	1
--------------------	----------------------	--	---

Identifier: 06.registerAccount

1.1.1.18. Description & Priority

Priority 1 (highest)

This requirement ("Register new account") is considered one of the most important requirements within my application. This is due to it being one of the core functions, which is why the priority is so high.

The function 'registerAccount' simply allows new users to register an account to they can gain access to the applications features. The registration details are stored in Firebase.

1.1.1.19. Use Case

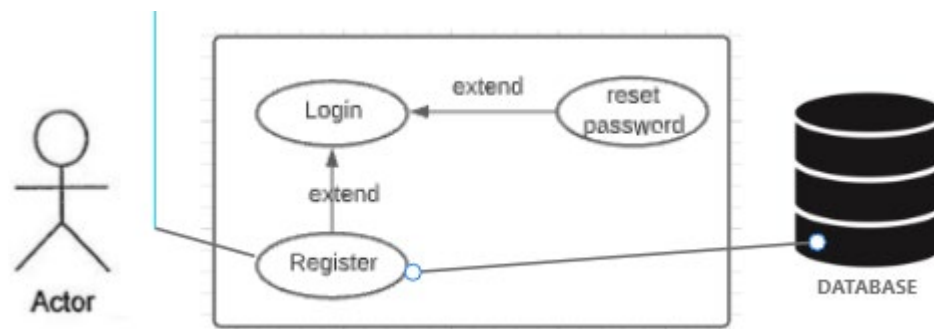
Scope

The scope of this use case is to allow a new user to register an account. This will all be done within the registration page which is navigated to clicking register on the welcome page.

Description

The user will be able to register a new account so they can gain access to the applications features.

Use Case Diagram



[FIG 6] [register use case]

Flow Description

Precondition

The user must have access to the internet.

The user must have a valid email address.

Activation

This use case starts when an <Actor> is on the registration page and decides to press the register button.

Main flow

1. The <Actor> navigates to savings registration page
2. The <Actor> inputs information and selects register button.
3. The system checks if users' information entered reaches certain expectations and if so the <Actor> account will be created.
4. The <firebase database> stored the users login credentials.
5. The system allows access to <Actor> and presents them with home features page within application.

Alternate flow

A1: Accidental selection

1. The <Actor> presses register button
2. The system prompts the <Actor> letting them know passwords don't match.
3. The <Actor> presses the register button
4. The system prompts the <Actor> letting them know email does not meet criteria.
5. The <Actor> presses the register button
6. The system prompts the <Actor> saying password must be minimum of 6 characters

Termination

The system presents the message "account with email already exists"

Post condition

The system goes into a wait state

1.1.1.20. Requirement 8: Keep track of money spent

08 resetPassword	Reset current password	A current user can reset their current password if they wish to change it	2
---------------------	---------------------------	---	---

Identifier: 08.resetPassword

1.1.1.21. Description & Priority

Priority 2 (medium)

This requirement (“Reset current password”) isn’t consider a necessary requirement but it will definitely benefit users who forget their passwords. The function “resetPassword” simply allows existing users to reset their current password if forgotten.

1.1.1.22. Use Case

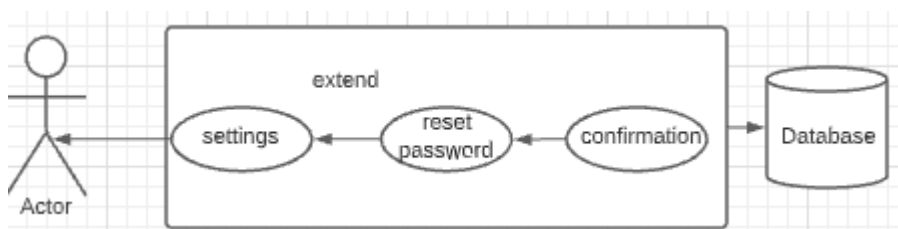
Scope

The scope of this use case is to allow an existing user to reset their password. This will all be done within the settings page which is navigated to clicking register on the card view found on the features page.

Description

An existing user can reset their current password if they wish to change it.

Use Case Diagram



[FIG 7] [Reset password]

Flow Description

Precondition

The user must have access to the internet.

Activation

This use case starts when an <Actor> is on the settings page and decides to press the reset password button.

Main flow

1. The <Actor> navigates to settings page
2. The <Actor> presses reset password.
3. The system prompts the <Actor> and awaits confirmation.

4. The <Actor> enters new password and confirms.
5. The system updates new password in database.

Alternate flow

A1 : Accidental selection

7. The <Actor> presses reset password button
8. The system prompts the <Actor> and awaits confirmation.
9. The <Actor> presses the cancel button

Termination

The system presents the message “Cannot retrieve account”

Post condition

The system goes into a wait state

1.1.1.23. Requirement 9: Log out

09 Logout	Log out of current account	A user may decide to logout of their account	2
--------------	-------------------------------	---	---

Identifier: 09.logOut

1.1.1.24. Description & Priority

Priority 2 (medium)

This requirement ("Logout") isn't considered a necessary requirement, but it will allow users to logout of their account if they don't wish to stay signed in.

1.1.1.25. Use Case

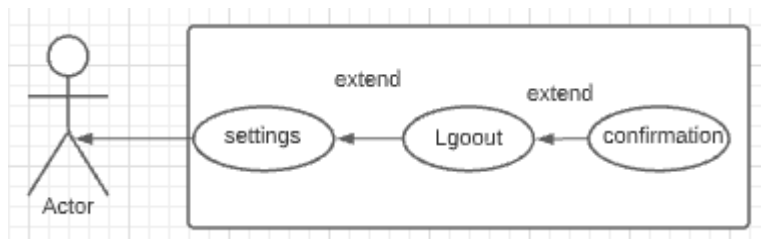
Scope

The scope of this use case is to allow an existing user to logout of their account. This will all be done within the settings page which is navigated to clicking register on the card view found on the features page.

Description

An existing user can logout of their account.

Use Case Diagram



[FIG 8] [Logout use case]

Flow Description

Precondition

The user must have access to the internet.

Activation

This use case starts when an <Actor> is on the settings page and decides to press the logout button.

Main flow

1. The <Actor> navigates to settings page
2. The <Actor> presses logout button.
3. The system prompts the <Actor> and awaits confirmation.
4. The <Actor> confirms.
5. The <Actor> is logged out and set back to welcome page.

Alternate flow

A1 : Accidental selection

1. The <Actor> navigates to settings page
2. The <Actor> presses logout button.
3. The system prompts the <Actor> and awaits confirmation.
4. The <Actor> cancels.
5. Back to step 1

Termination

The system presents the message “Cannot retrieve account”

Post condition

The system goes into a wait state

3.0 Data Requirements

Data will be stored in a firebase database. SQL can be used for programming and is also designed for managing data help in a relational database management system. In my case, I will be using Microsoft SQL Server management studio. Data will reside on a server but also has the option to failover to backup server which ensures high availability.

3.1 User Requirements

1. User must have internet access.
2. User must be using an android device
3. Users should create an account.
4. Users should be notified if account with email already exists.
5. User should be notified if they failed to meet the password requirements.
6. User should be able to login to their account.
7. Users should remain logged in.
8. User should be notified if they enter the wrong password
9. Users should be able to make in app purchases using their credit card.
10. Users should be able to send money to an acquaintance using their PayPal account.
11. Users should be able to send money to an acquaintance using their Google Pay account.
12. Users who would like info regarding current state of cryptocurrency market.
13. Users who would like to be able to process transactions.
14. Users demand a navigable application interface and a mobile application which is responsive with good standard graphical UI design.

3.2 Environmental Requirements

An Android emulator and device are required to execute and test the application while it is being developed.

1. App should include a splash activity.
2. App should include a welcome page.
3. App should include a login page.
4. App should include a registration page.
5. App should be free to download.
6. App should include a Payment option page.
7. App should include a PayPal payment option.
8. App should include a Google Pay payment option.
9. App should include a Stripe payment activity.
10. App should present a navigable user interface.
11. App should include a health activity.

12. App should include a map activity to assist the user with locating the nearest ATM.
13. App should include a savings vault.
14. App should include a cryptocurrency page with the latest prices.
15. App should assist users with daily, weekly, and monthly budgeting.
16. App should provide analytics on the users spendings.

3.3 Usability Requirements

“The usability of a user interface refers to the fluency or ease with which a user is able to interact with a system without ‘thinking’ about it.”

Usability is essentially how much something allows users to do what is necessary for the user to do, as simple and pleasantly as possible. It is considered important for many reasons, let us say a user is trying to send money to a friend but the functionality within my application keeps failure. The user will become frustrated as they cannot achieve their goals. The user is not going to be patient with this issue they will more than likely look for an alternative application to cater their needs. A designs usability varies on how well its characteristics accommodate user’s needs.

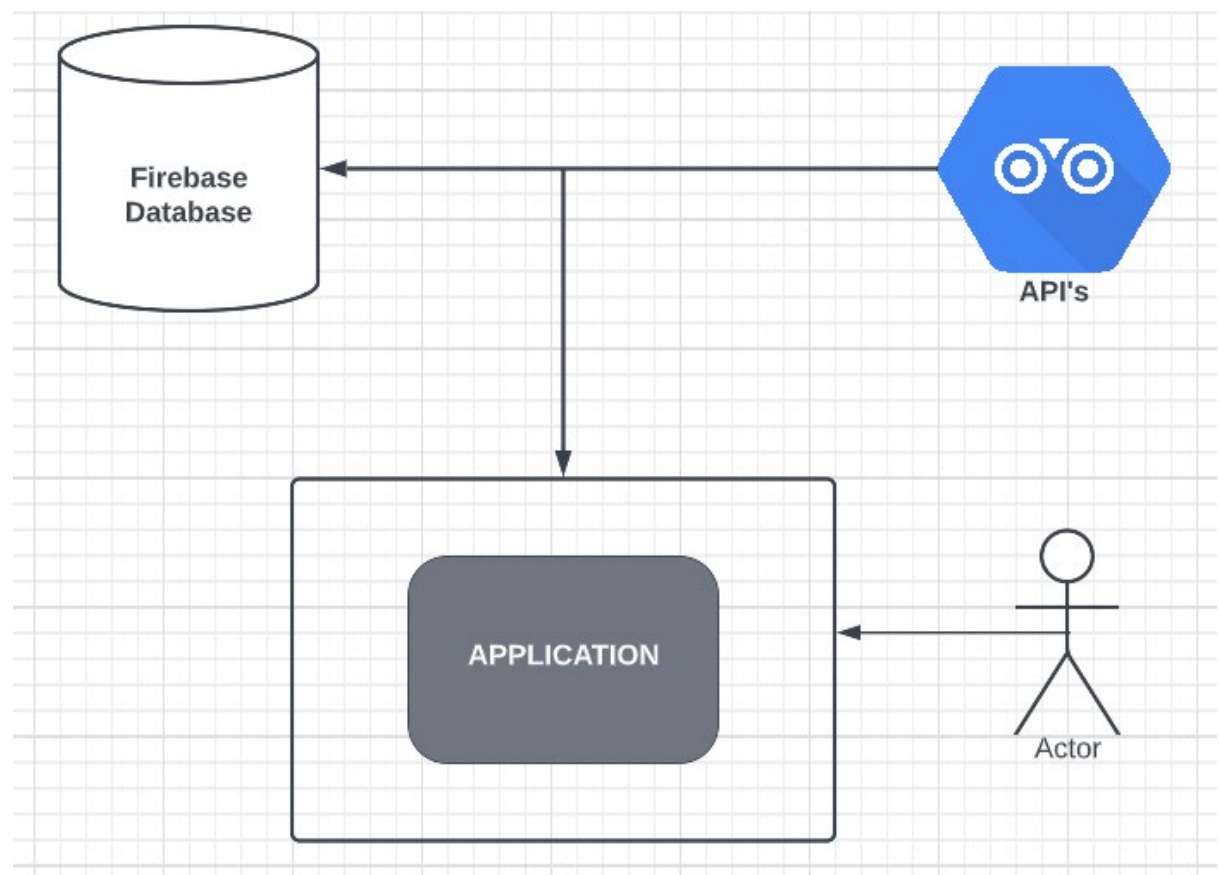
“While the ISO definition has three aspects, Nielsen divides usability into five elements, so-called attributes, which can be measured and used to specify usability objectives.” [Tuomo Sippola, 2017]. They are Effectiveness, Efficiency, Engaging, Error Tolerance and Easy to Learn.

- Effectiveness
 - This refers to how users remember a system; when users return to the system after a time of inactivity, it is deemed effective if they regain proficiency.
- Efficiency
 - This is all about how quickly a user can complete tasks once they have mastered a system.
- Engaging
 - This encompasses aesthetics, the use of suitable layouts, usability, and understandable language. All these parameters are presented in order to offer the suitable user interface.
- Error Tolerance
 - There is a significant likelihood that you will never completely eliminate system errors. To aid in reducing errors, we examine the frequency, severity, and ease of recovery of user mistakes.
- Easy to Learn
 - Systems must be simple to understand, as this impacts the user's initial opinion of the system.

4.0 Design & Architecture

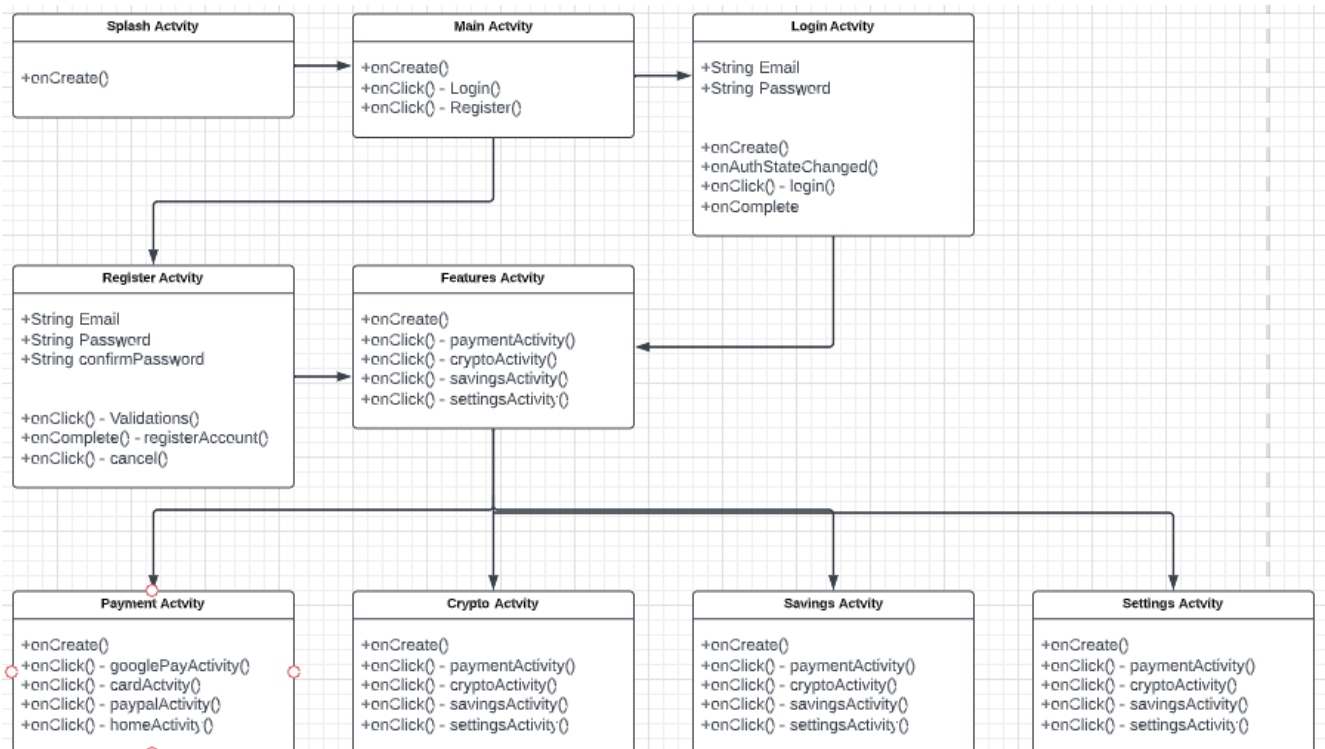
The application will be stored locally on Android-based mobile devices, according with System Architecture Diagram. All user information will be stored in a centralized NoSQL Firebase database. The application has access to multiple API's. The application has activities that utilize the following:

- PayPal API
- Google Pay API
- Stripe API
- Cryptocurrency API



[FIG 9] [Design and architecture diagram]

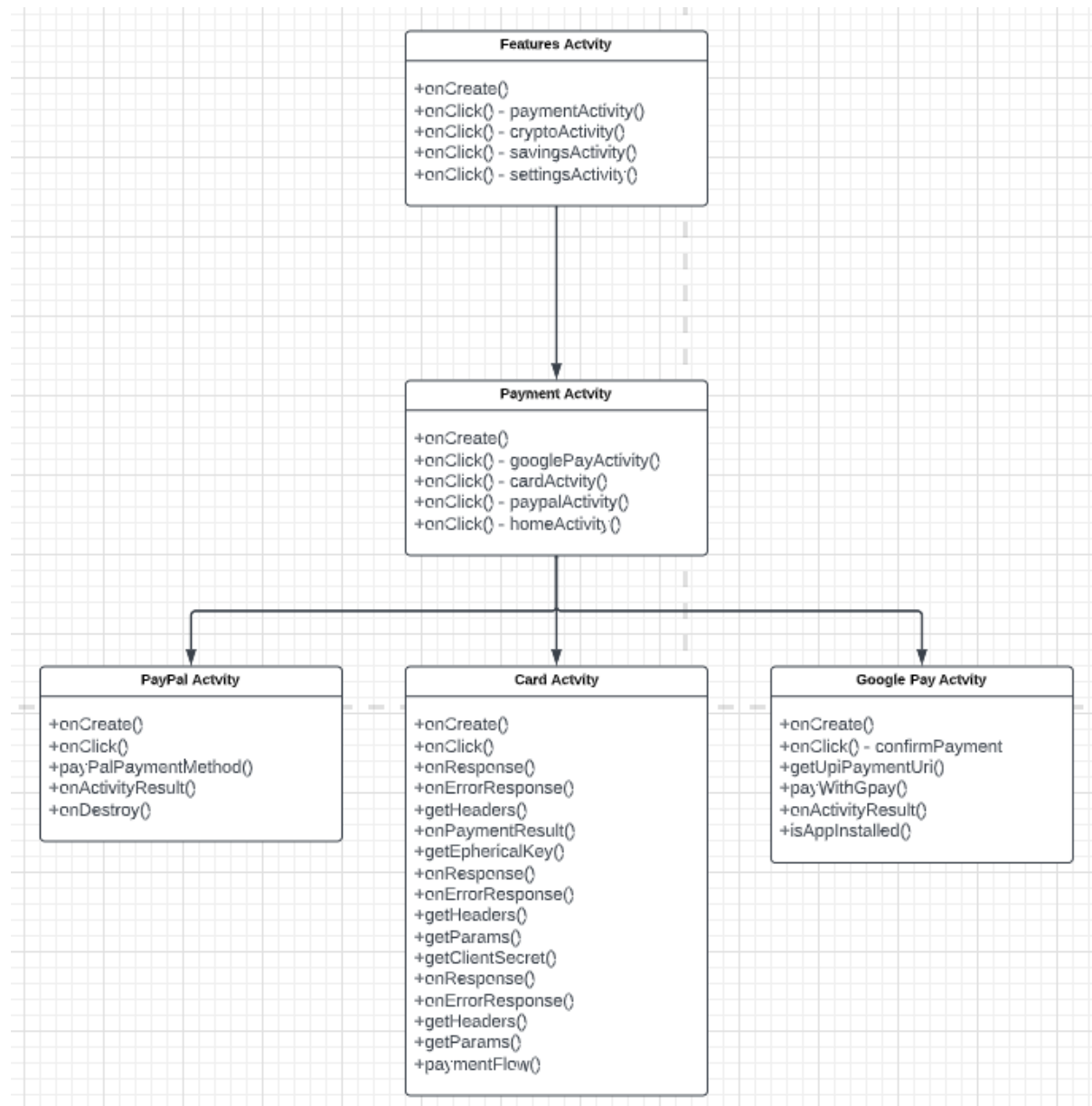
4.1 Class diagram



[FIG 10] [Application Class diagram 1]

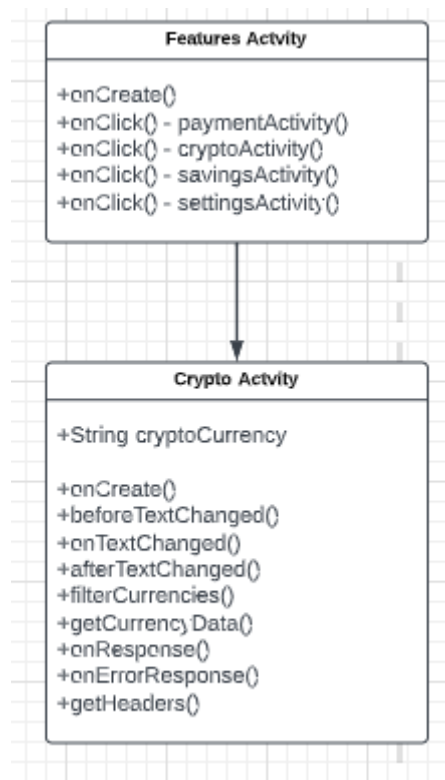
Couldn't fit all the activities in a single snapshot so I have broken it up and continued from the snapshot above.

Payment Activity



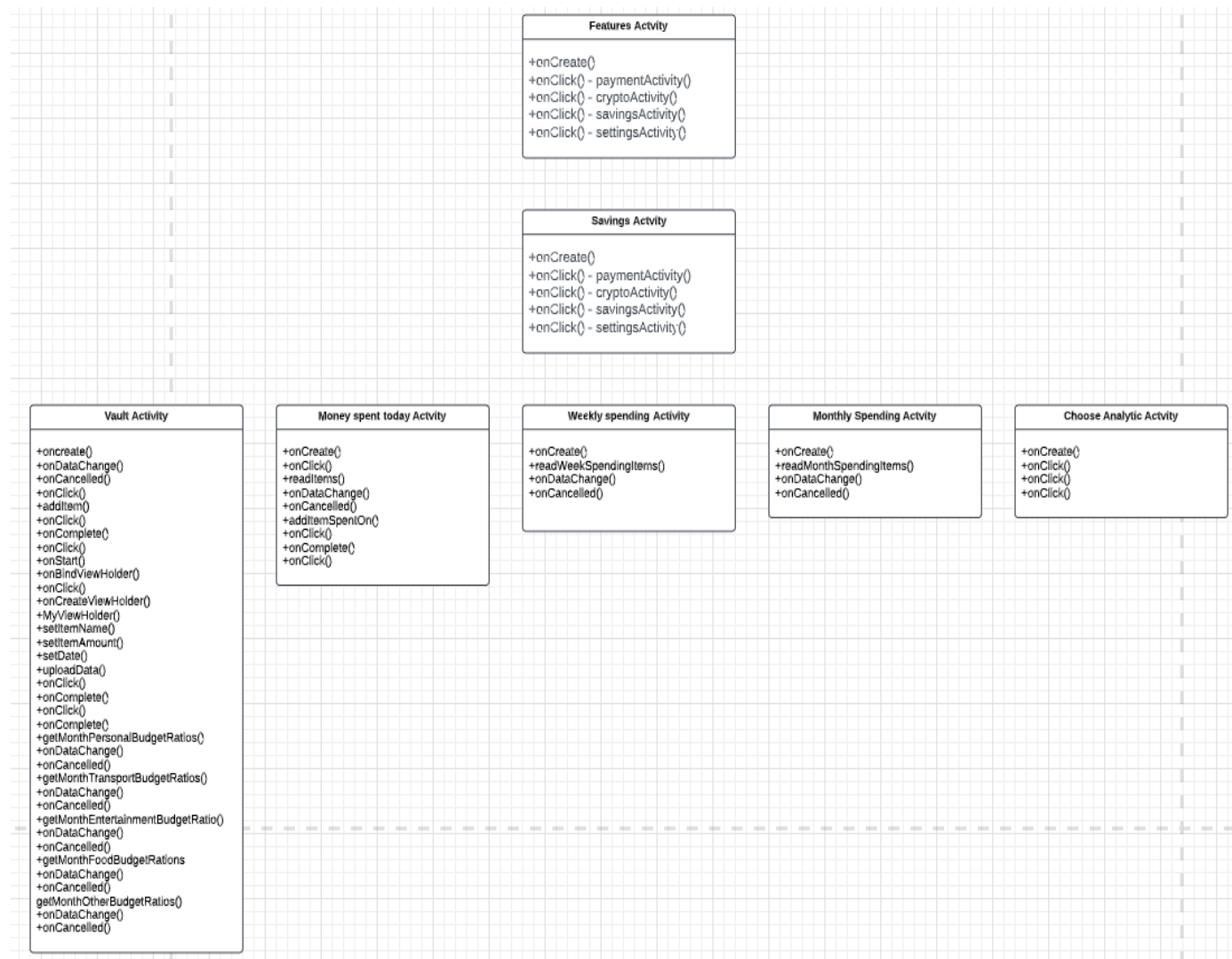
[FIG 11] [Application Class diagram 2]

Cryptocurrency Activity



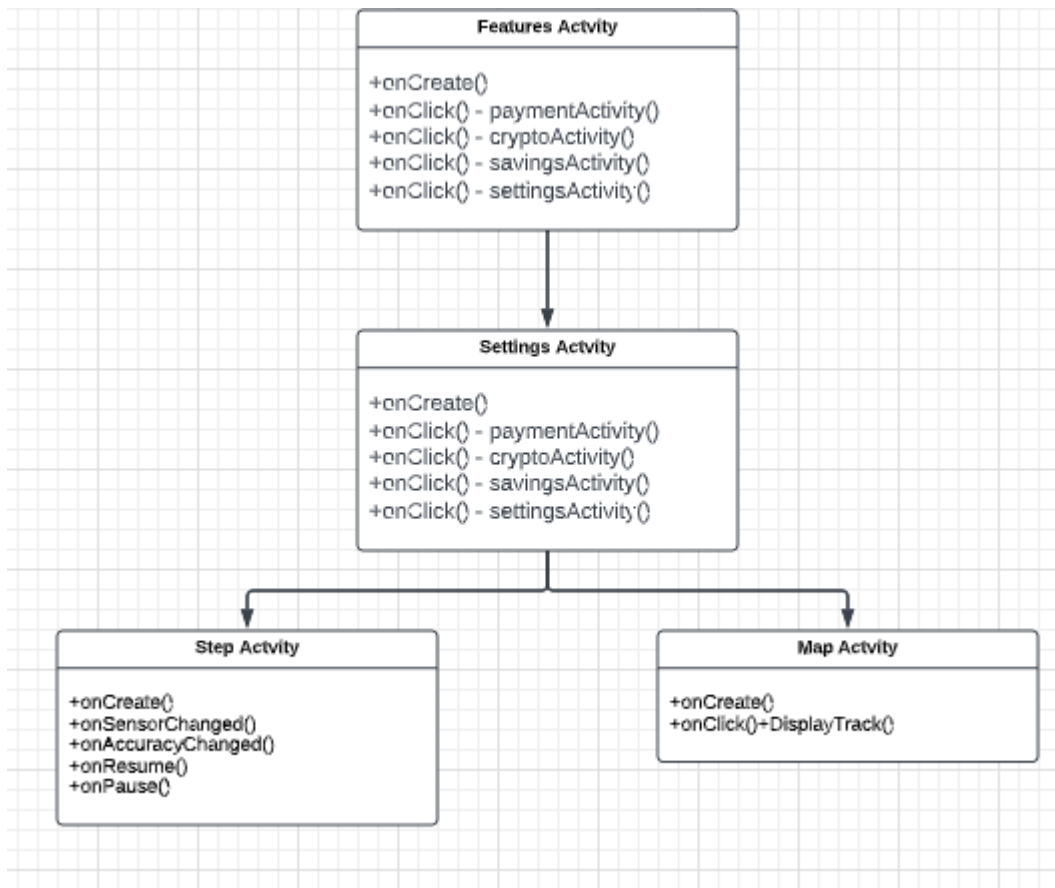
[FIG 12] [Application Class diagram 3]

Savings Activity



[FIG 13] [Application Class diagram 4]

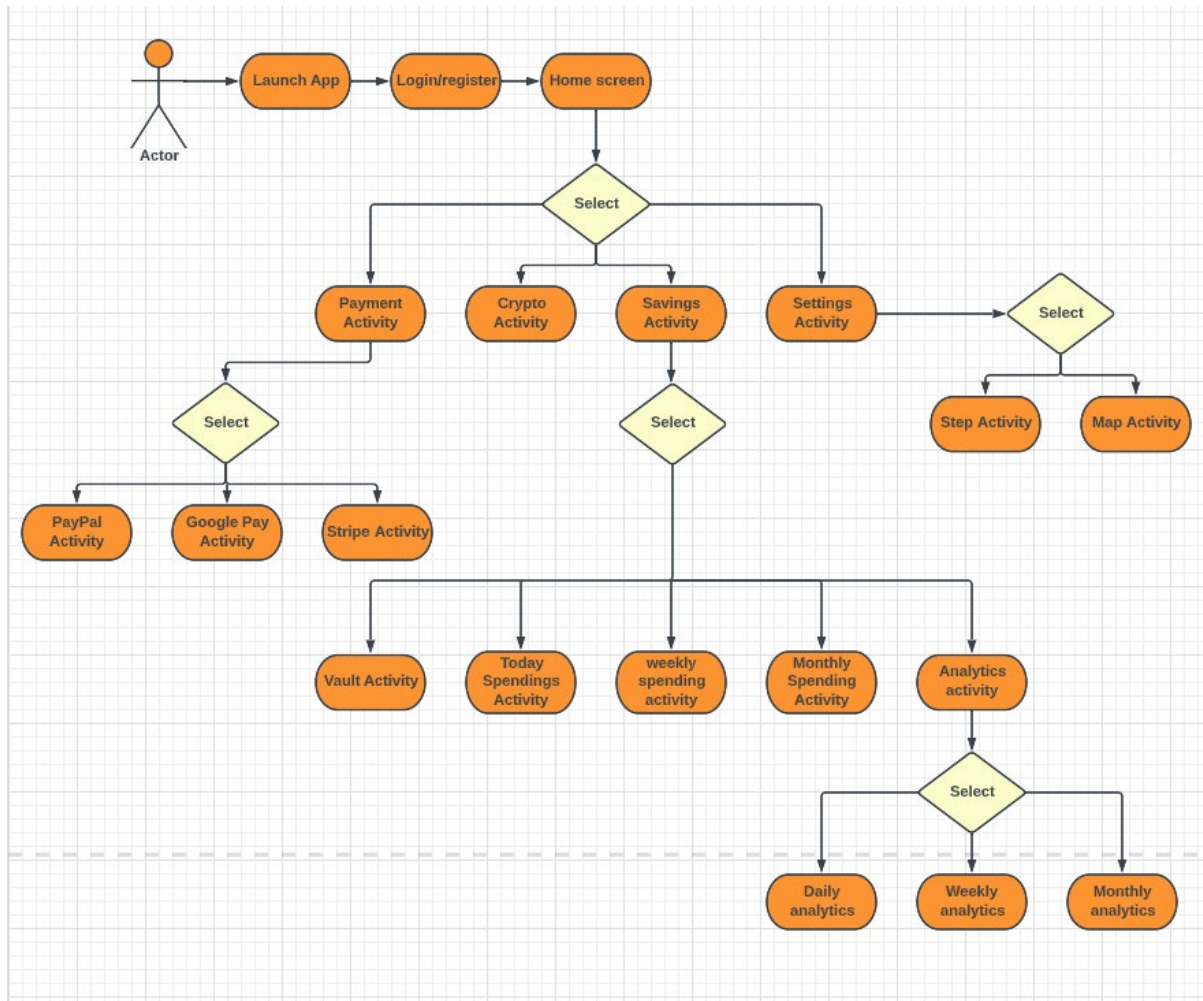
Settings Activity



[FIG 14] [Application Class diagram 5]

4.2 Flowchart

A flowchart is an excellent tool that I have implemented for visualizing a complex process. I have used a flowchart as shown below to help depict the processes, sequences, and decisions involved in my system.



[FIG 15] [Flowchart diagram]

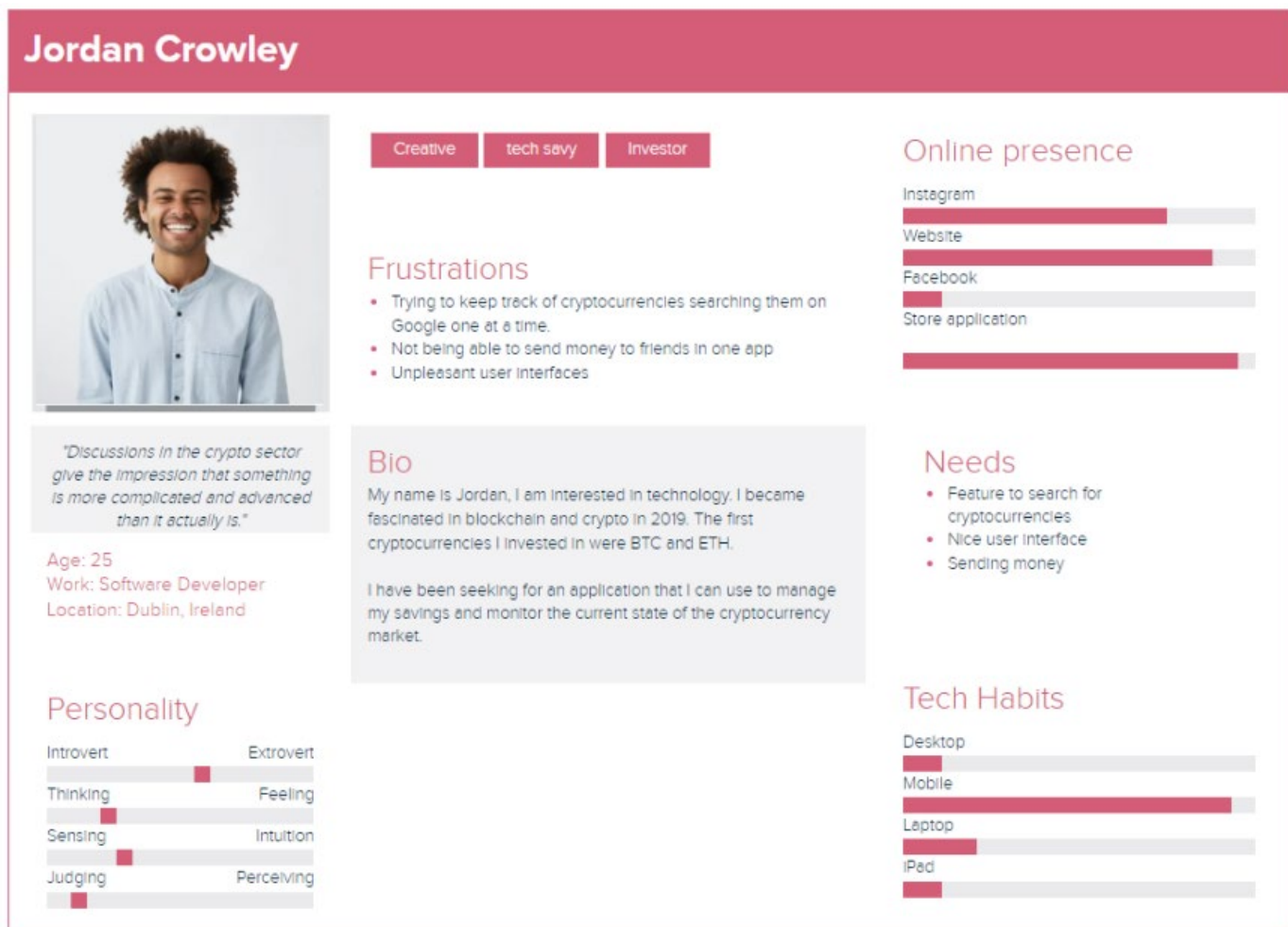
4.3 User Personas

“A user persona is a fictional representation of your ideal customer.” [Raven Veal, 2021]. User personas are archetypal users whose goals and characteristics reflect the needs of a larger population. Personas aided me in developing a system that consumers require, as opposed to one that they desire.

Investors and tech-savvy individuals are my two primary user groups, and both of my personas fall under these categories. I hosted a brainstorming session and conducted extensive research to build my personas. In conclusion, I feel that an effective persona should:

- Personas are not fictitious depictions of a target user's beliefs. Every aspect of a persona's description must be based on actual data.
- Not various user roles, but real user patterns are reflected in personas.

4.3.1 Persona 1



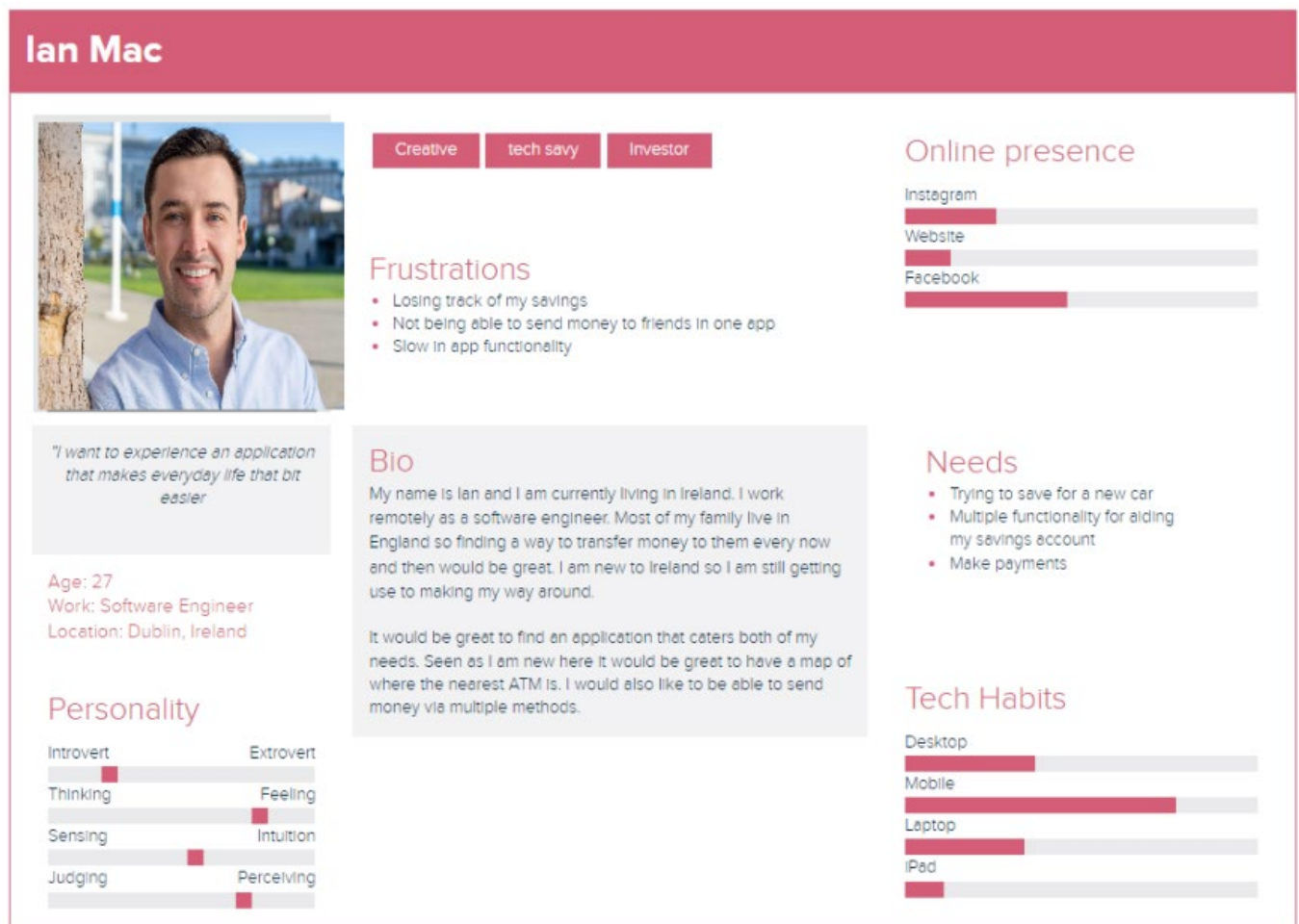
[FIG 16] [User persona one]

4.3.1.1 Scenario 1 – Jordan Crowley

Monitoring Cryptocurrencies: Jordan Crowley is a Software Developer who is from Dublin, Ireland. Jordan is fed up with searching for cryptocurrencies individually on Google. Jordan would love an application that assists him with checking the current price of cryptocurrencies in today's market. For Jordans needs to be full catered, it is necessary that he can filter the list with a search filter.

Action: Jordan opens 'Wave' and signs into the account he previously made. Jordan then clicks on the cryptocurrency option in the home screens menu. Jordan takes a quick look at the top 100 cryptocurrencies but can find the cryptocurrency he is looking for. Jordan uses the search functionality at the top of the page to search for his desire cryptocurrency.

4.3.2 Persona



[FIG 17] [User persona two]

4.3.2.1 Scenario 2 – Ian Mac

Adding money to savings vault:

It is a Friday afternoon and Ian has received his monthly pay. Ian is saving for a car and wants to save as much money as efficiently as possible. Straight away Ian opens the 'Wave' app. He logs into the account has had for a week now. Ian navigates to the savings vault activity using the navigable user interface provided. Ian decides that he wants to save 500 euro for his car.

5.0 Implementation

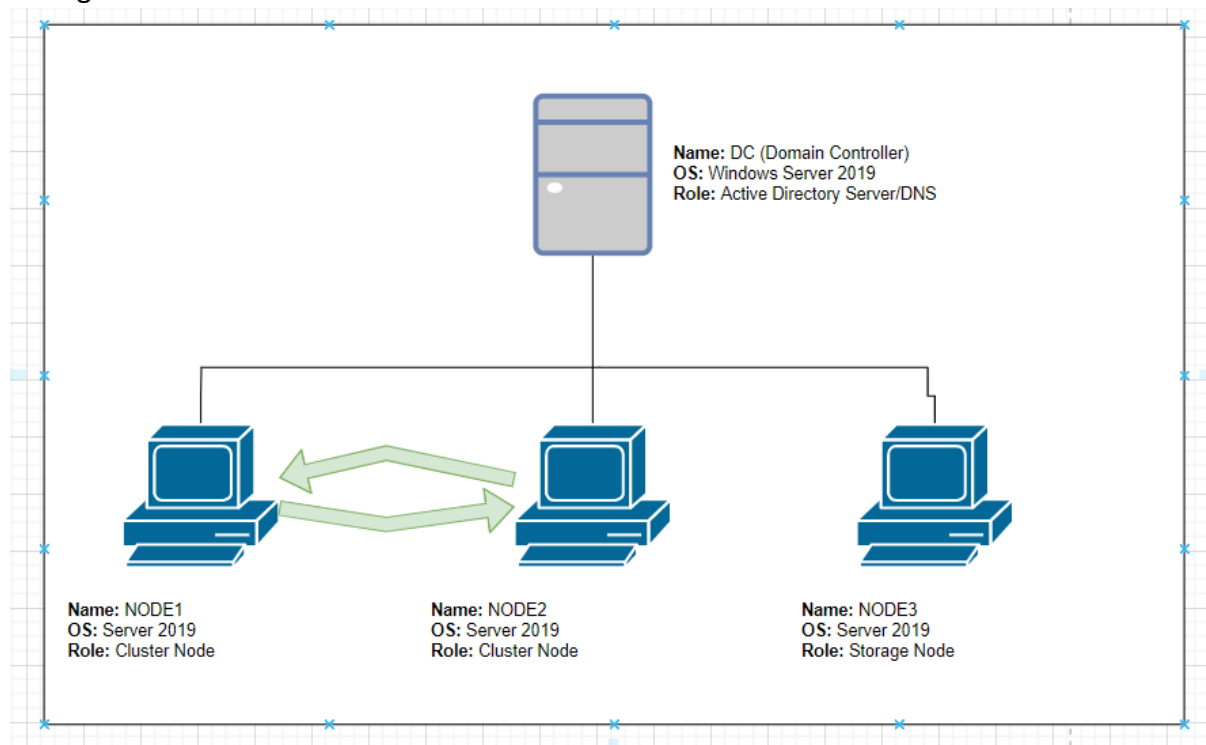
5.1 Failover Clustering

A Windows Server Failover Cluster (WSFC) is a collection of individual servers that collaborate to improve application as well as service availability. WSFC services and capabilities are used by SQL Server to support Always On availability groups and SQL Server Failover Cluster Instances. In summary, if a cluster node or service fails, the services hosted on that node can be moved to another available node automatically or manually. *“Failover clusters also provide Cluster Shared Volume (CSV) functionality that provides a consistent, distributed namespace that clustered roles can use to access shared storage from all nodes.”* [doc.microsoft.com] [2022]. Throughout my two years as working as a database administrator I have dealt with configuring multiple windows server failover clusters, below I walk through the key steps that need to be followed to successfully implement the disaster recovery solution.

Key terms that will be mentioned in this implementation:

- **Node** - A server that is going to be involved in the WSFC.
- **Domain Controller**- Used for creating the cluster object and joining the member servers to the domain.

This 4-node setup is what I would use to ensure that my application is highly available. The first node will serve as the domain controller, followed by two nodes that will participate in the failover cluster, and finally by my storage node, which will be configured to store our storage disks.



[FIG 18] [Failover cluster setup]

The following events are taking place on the storage machine.

Step 1: First and foremost, I would start by installing the iSCSI target server role. This is accomplished by going to server manager -> manage -> Add roles and features -> server roles -> enable iSCSI Target Server. -> install.

Step two: Go to File and Storage Services -> iSCSI -> *Here we have the option to create an iSCSI virtual disc* -> select C: drive -> name it -> allocate the disk 5GB -> I need to create the iSCSI target -> go to access servers, this is where we will add my future failover cluster nodes. -> Add nodes using their IP addresses. -> Create. The first two steps are then repeated to create two more discs with the same parameters.

Step three: connect the virtual disks I previously created to my nodes which are going to become failover cluster, these are the disks that are going to be used as the shared storage.

The following events are taking place on NODE1

Step four: Navigate to the server's server manager on NODE1. -> open tools -> iSCSI initiator -> iSCSI initiator properties window will appear, where I will need to locate and connect the iSCSI target server that I have configured on my storage server. -> Navigate to the discover portal and enter the IP address of the storage server. -> Navigate to the targets tab and connect the initiator to the target.

Step five: Step 4 must be repeated on my NODE2 server.

Step six: Still on NODE1, open disc management -> here we can see that I have three offline discs stored on my storage server. I'll bring the disks online and initialise them. -> Next, I'll create a simple volume on each disk. -> The three volumes will no longer be accessible on my NODE1 server.

The following events are taking place on NODE2

Step seven: Rather than repeating the previous steps, I will simply open disc management -> bring the discs online -> without creating a volume on them. -> The disks are now available on NODE2, and share storage is configured.

The following events are taking place on NODE1 and NODE2

Step eight: At this point, I will install the failover clustering feature on both nodes. -> first, I'll return to server manager -> Add Roles and Features -> features tab -> select failover clustering -> install. -> repeat on NODE2.

The following events are taking place on NODE1

Step 10: Open the failover clustering manager -> tools -> failover cluster manager -> validate configuration, which will highlight any errors that need to be corrected before we start the failover cluster. -> Add two validation nodes to the wizard -> Run all tests -> These tests would be done offline in a production environment because they take the disks offline.

Step ten: Using the create cluster wizard, create a cluster. -> Add two nodes to the cluster -> Do not add eligible storage. -> Add three disks to the failover cluster. -> Add a role to the

cluster; this role will be the file server role, which creates a highly available share so that if a user connects to the share and something happens to one of the nodes in my failover cluster, the other node will pick the share disks in the shares to ensure that there is no downtime for the users who are using these shares.

The following events are taking place on NODE1

Step twelve: Witness disk configuration -> Choose disc 1 as the witness disk in failover cluster manager -> configure cluster Quorum settings -> select a quorum witness and configure a disk witness.

The following events are taking place on NODE2

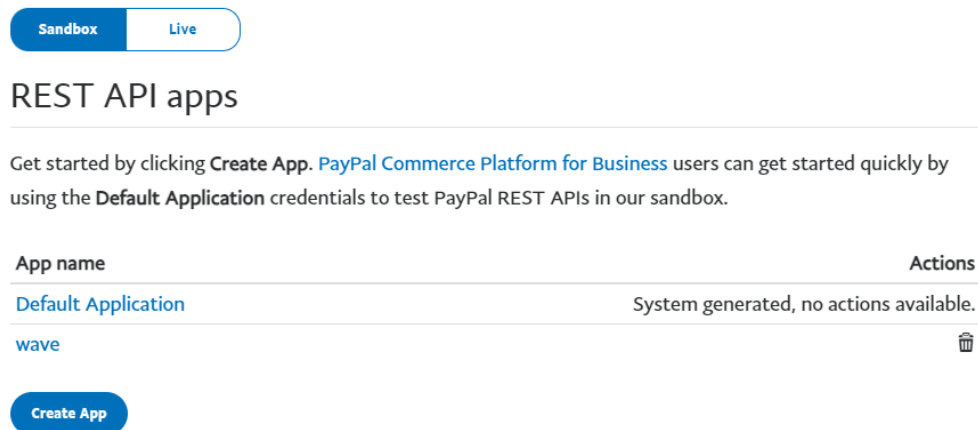
Step thirteen: Performing the failover. Power off the server, NODE2 will failover to NODE1.

Step fourteen: There will be some slowness on the client's machine at first, but there will be no downtime.

5.2 PayPal Activity

“The Native Checkout SDK provides an in-context mobile experience that keeps your existing server-side integration intact.” [developer.paypal, 2022] Following the preparation of the UI for the PayPal activity, I created an app on the PayPal developer website using my account. To work with Android Studio, I needed a client ID, which I found on the 'My apps and Credentials' page. I went into my sandbox account to see my test accounts because I was running this in a test environment rather than production. The client ID is saved in my application's PayPalClientID activity.

My apps & credentials



[FIG 19] [PayPal Sandbox application]

In order to work with the PayPal API, I had to add the following dependency to my build Gradle:

```
implementation 'com.paypal.sdk:paypal-android-sdk:2.15.3'
```

On the PayPal card view, I've added a onClick listener that, when triggered, redirects the user to the PayPal checkout page. I've added the following code to make sure the application knows the following transaction is being performed in a test environment:

```
.environment(PayPalConfiguration.ENVIRONMENT_SANDBOX)
```

In the code in the Payment activity, I mentioned a class 'PaymentActivity' which doesn't actually exist. This activity is created in the PayPal dependency I have added.

```
Intent intent = new Intent(this, PaymentActivity.class);
```

5.3 Google Pay Activity

When a user triggers the payment option in the merchant app, the following code shows how to display Google Pay as a way of payment.

```
private static Uri getUpiPaymentUri(String name, String upiId, String note, String amount){  
    return new Uri.Builder()  
        .scheme("upi")  
        .authority("pay")  
        .appendQueryParameter("pa",upiId) //Payee address or business virtual payment address  
        .appendQueryParameter("pn",name) //Payee name https://developers.google.com/pay/india/  
        .appendQueryParameter("tn",note) //transaction note. It is the description appearing i  
        .appendQueryParameter("am",amount) //transaction amount  
        .appendQueryParameter("cu","INR") //Currency code. (This should be set in the details  
        .build();  
}
```

[FIG 20] [getUpiPaymentUri code screenshot]

I have also added an additional method to simply determine whether the Google pay app is installed on the user's device. This method is later called in the 'payWithGPay' method, if the user has Google Pay installed it will open up the payment activity, if the app is not found on the user's device it will prompt the user with a message notifying them that the transaction cannot be complete as they do not have Google Pay installed.

```
public static boolean isAppInstalled(Context context, String packageName){  
    try{  
        context.getPackageManager().getApplicationInfo(packageName, 0);  
        return true;  
    }catch (PackageManager.NameNotFoundException e){  
        return false;  
    }  
}
```

[FIG 21] [isAppInstalled code screenshot]

5.4 Stripe Activity

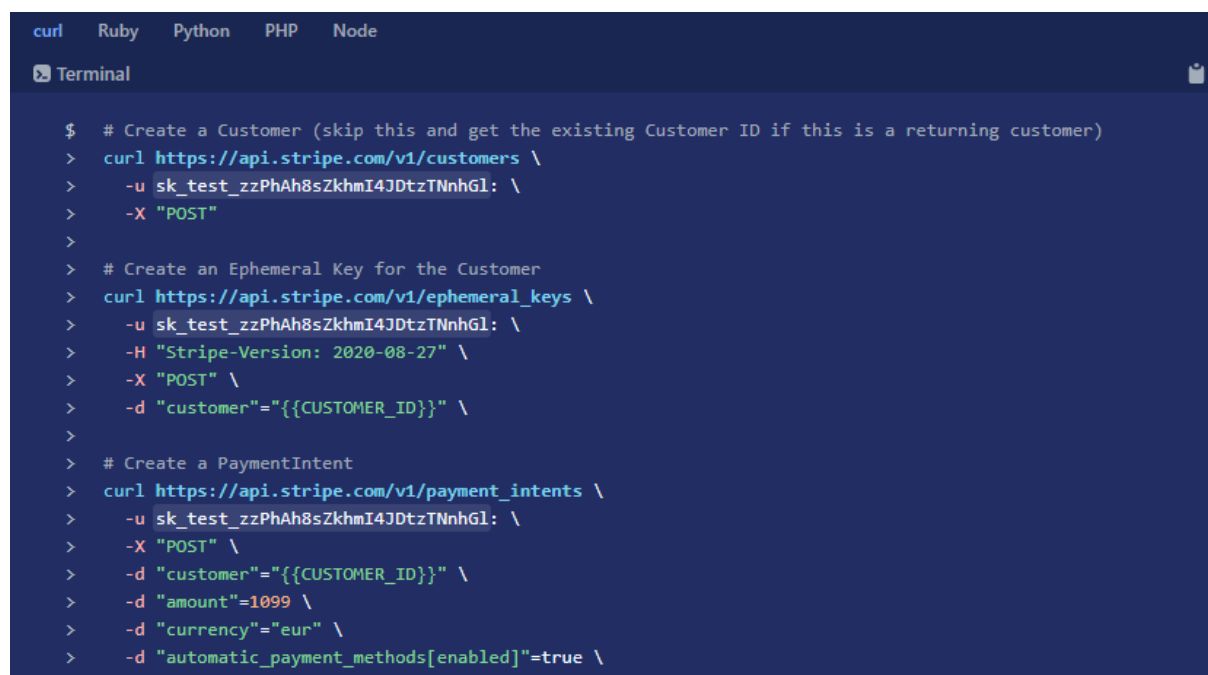
Building this activity was aided by the official Stripe documentation [Stripe docs, 2022]. The first step when working towards adding stripe payments was to add the following dependencies to my build Gradle file:

```
implementation 'com.stripe:stripe-java:20.77.0'  
implementation 'com.stripe:stripe-android:17.2.0'
```

Additionally, I have added the dependency below to post requests to the server:

```
implementation 'com.android.volley:volley:1.1.0'
```

The next step is to add an endpoint, I have used postman to test API calls. Stripe mentions that for security reasons, my app will not be able to create the necessary objects and that I will need to add an endpoint on my server. I followed the following steps in my postman environment. The steps were provided as part of the Stripe official documentation.



```
curl  Ruby  Python  PHP  Node  
Terminal  
  
$ # Create a Customer (skip this and get the existing Customer ID if this is a returning customer)  
> curl https://api.stripe.com/v1/customers \  
>   -u sk_test_zzPhAh8sZkhnI4JDtzTNnhG1: \  
>   -X "POST"  
>  
> # Create an Ephemeral Key for the Customer  
> curl https://api.stripe.com/v1/ephemeral_keys \  
>   -u sk_test_zzPhAh8sZkhnI4JDtzTNnhG1: \  
>   -H "Stripe-Version: 2020-08-27" \  
>   -X "POST" \  
>   -d "customer"="{{CUSTOMER_ID}}" \  
>  
> # Create a PaymentIntent  
> curl https://api.stripe.com/v1/payment_intents \  
>   -u sk_test_zzPhAh8sZkhnI4JDtzTNnhG1: \  
>   -X "POST" \  
>   -d "customer"="{{CUSTOMER_ID}}" \  
>   -d "amount"=1099 \  
>   -d "currency"="eur" \  
>   -d "automatic_payment_methods[enabled]"=true \  
>
```

[FIG 22] [Code for postman environment]

The snapshot below relates to step one in the snapshot above. When the call is made, I received customerID which is being called by the key 'id'. When I successfully received the customerID I then proceed to get the ephemeral key.

```
StringRequest stringRequest = new StringRequest(Request.Method.POST,
    url: "https://api.stripe.com/v1/customers",
    new Response.Listener<String>() {
        @Override
        public void onResponse(String response) {
            try {
                JSONObject object = new JSONObject(response);
                customerID = object.getString("id");

                Toast.makeText(context: cardActivity.this, customerID, Toast.LENGTH_SHORT).show();

                getEphericalKey(customerID);
            } catch (JSONException e) {
                e.printStackTrace();
            }
        }
    })
```

[FIG 23] [onRepsonse code screenshot]

5.5 Daily analytics

The following methods are important in this activity because they are used to calculate total spending in each category. In my code, I've made it so that if a user hasn't spent money in a specific category, the activity won't be displayed. For example, if the 'getTotalWeelTransportExpenses' function returns a value of 0, the transport category will not be displayed to the user.

```
getTotalWeekTransportExpenses();
getTotalWeekFoodExpenses();
getTotalWeekPersonal();
getTotalWeekEntertainmentExpenses();
getTotalWeekOtherExpenses();
getTotalDaySpending();
```

All the methods have similar code so I will walk through the 'getTotalWeelTransportExpenses' method. At the start of the code, I am concatenating the current date with the 'transport' item. I am doing this because in the VaultActivity I have the following code:

```
String itemNday = budgetItem+date;
```

This will return the total amount spent on a certain item on the present day.

The database reference is retrieving the total expenses for the user that is currently logged in.

```
private void getTotalWeekTransportExpenses() {
    MutableDateTime epoch = new MutableDateTime();
    epoch.setDate(0);
    DateTime now = new DateTime();

    DateFormat dateFormat = new SimpleDateFormat( pattern: "dd-MM-yyyy");
    Calendar cal = Calendar.getInstance();
    String date = dateFormat.format(cal.getTime());
    String itemNday = "Transport" + date;

    String strURL = "https://wave-cbfdd-default-rtdb.europe-west1.firebaseio.com/";
    DatabaseReference reference = FirebaseDatabase.getInstance(strURL).getReference( path: "expenses").child("mYetIoEY7hMEDhw15jt0epLsP0Z2");

    Query query = reference.orderByChild("itemNday").equalTo(itemNday);
    query.addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot snapshot) {
            if (snapshot.exists()) {
                int totalAmount = 0;
                for (DataSnapshot ds : snapshot.getChildren()) {
                    Map<String, Object> map = (Map<String, Object>) ds.getValue();
                    Object total = map.get("amount");
                    int pTotal = Integer.parseInt(String.valueOf(total));
                    totalAmount += pTotal;
                    analyticsTransportAmount.setText("Spent " + totalAmount);
                }
                personalRef.child("dayTrans").setValue(totalAmount);
            } else {
                LinearLayoutTransport.setVisibility(View.GONE);
            }
        }

        @Override
        public void onCancelled(@NonNull DatabaseError error) {

        }
    });
}
```

[FIG 24] [getTotalWeekTransportExpenses code screenshot]

Daily analytics continued

The next step in my code is key for loading the pie chart. The following code returns the amount that has been spent in that day. For example, with transport I am checking the value stored in the node “dayTrans”, I am then converting the value into an integer. If the variable does not exist, the variable “traTotal” is set to 0.

```
//load graph
private void loadGraph() {
    personalRef.addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot snapshot) {
            if (snapshot.exists()) {

                int traTotal;
                if (snapshot.hasChild( path: "dayTrans")) {
                    traTotal = Integer.parseInt(snapshot.child("dayTrans").getValue().toString());
                } else {
                    traTotal = 0;
                }

                int foodTotal;
                if (snapshot.hasChild( path: "dayFood")) {
                    foodTotal = Integer.parseInt(snapshot.child("dayFood").getValue().toString());
                } else {
                    foodTotal = 0;
                }

                int entTotal;
                if (snapshot.hasChild( path: "dayEnt")) {
                    entTotal = Integer.parseInt(snapshot.child("dayEnt").getValue().toString());
                } else {
                    entTotal = 0;
                }

                int otherTotal;
                if (snapshot.hasChild( path: "dayOther")) {
                    otherTotal = Integer.parseInt(snapshot.child("dayOther").getValue().toString());
                } else {
```

[FIG 25] [loadGraph code screenshot]

Daily Analytics continued

Before implementing the pie chart, it is necessary to install the following dependency which I have added to my build Gradle:

```
implementation 'com.github.AnyChart:AnyChart-Android:1.1.2'
```

To then set the pie chart I simply add an entry for each of the items. For each item I have assigned them to the value of each of their totals.

```
Pie pie = AnyChart.pie();
List<DataEntry> data = new ArrayList<>();
data.add(new ValueDataEntry(x "Transport", traTotal));
data.add(new ValueDataEntry(x "Food", foodTotal));
data.add(new ValueDataEntry(x "Entertainment", entTotal));
data.add(new ValueDataEntry(x "Other", otherTotal));

pie.data(data);

pie.title("Daily Analytics");

pie.labels().position("outside");

pie.legend().title().enabled(true);
pie.legend().title()
    .text("Items spent on")
    .padding(0d, 0d, 10d, 0d);

pie.legend()
    .position("center-bottom")
    .itemsLayout(LegendLayout.HORIZONTAL)
    .align(Align.CENTER);

anyChartView.setChart(pie);
} else {
    Toast.makeText(context DailyAnalyticsActivity.this, text "Child does not exist", Toast.LENGTH_SHORT).show();
}
}
```

[FIG 25] [pie chart code screenshot]

Registration page

In the code in the snippet below, it is related to the registration page. Basically, what I am doing is passing some rules. The user can't register an account if the input boxes are empty, the passwords do not match, the user provides an invalid email, or the password is less than 6 characters.

```
if(!(ConfirmPassword.equals(password))){  
    editTextPasswordConfirm.setError("Passwords don't match!");  
    editTextPasswordConfirm.requestFocus();  
    return;  
}  
  
if(!Patterns.EMAIL_ADDRESS.matcher(email).matches()){ //checks if us  
    editTextEmail.setError("Invalid Email!");  
    editTextEmail.requestFocus();  
    return;  
}  
  
if(password.length() < 6) { //firebase don't except passwords < 6  
    editTextPassword.setError("Min password should be 6 characters");  
    editTextPassword.requestFocus();  
    return;  
}
```

[FIG 26] [password validations code screenshot]

Cryptocurrency Activity

In the code in the snippet below, we can see the code I used to fetch the data from coinmarketcap using their API. In the code the first thing I did was create the method called 'getCurrencyData'. Looking at this method on a high level, I created a string for URL in which I pasted the URL for coinmarketcap's API, I used volley as it is an HTTP library that makes networking for android apps easier. I made a GET request which was followed by a try catch statement for error handling. Essentially in the try I created a for loop that calls data from JSON file which gets all objects inside of the data array, this is how I got the name, price, and symbol of each cryptocurrency.

```
private void getCurrencyData(){
    PB.setVisibility(View.VISIBLE);
    String url = "https://pro-api.coinmarketcap.com/v1/cryptocurrency/listings/latest";
    RequestQueue requestQueue = Volley.newRequestQueue( context: this);
    JsonObjectRequest jsonObjectRequest = new JsonObjectRequest(Request.Method.GET, url, null, new Response.Listener<JsonObject>() {
        @Override
        public void onResponse(JsonObject response) {
            PB.setVisibility(View.GONE);
            try {
                JSONArray dataArray = response.getJSONArray( name: "data");
                for(int i=0; i<dataArray.length();i++){
                    JSONObject dataObj = dataArray.getJSONObject(i);
                    String name = dataObj.getString( name: "name");
                    String symbol = dataObj.getString( name: "symbol");
                    JSONObject quote = dataObj.getJSONObject("quote");
                    JSONObject USD = quote.getJSONObject("USD");
                    double price = USD.getDouble( name: "price");

                    currencyRVModalArrayList.add(new CurrencyRVModal(name,symbol,price));
                }
                currencyAdapter.notifyDataSetChanged();
            }catch (JSONException e){
                e.printStackTrace();
                Toast.makeText( context: cryptoActivity.this, text: "Failed to extract", Toast.LENGTH_SHORT).show();
            }
        }
    }, new Response.ErrorListener() {
        @Override
        public void onErrorResponse(VolleyError error) {
            PB.setVisibility(View.GONE);
            Toast.makeText( context: cryptoActivity.this, text: "Failed to retrieve data", Toast.LENGTH_SHORT).show();
        }
    }){
        @Override
        public Map<String, String> getHeaders() throws AuthFailureError {
            HashMap<String, String> headers = new HashMap<>();
            headers.put("X-CMC_PRO_API_KEY", "db9dec82-25d7-4435-93e4-6c88fb483c6f");
            return headers;
        }
    };
    requestQueue.add(jsonObjectRequest);
}
```

[FIG 27] [getCurrencyData code screenshot]

5.6 Step Activity

Detector

The sensor's constant is 'TYPE STEP DETECTOR,' and it triggers an event every time the user takes a step. The latency is expected to be less than 2 seconds. One of the main checks I perform in the snapshot is to see if the step sensor is available. To use both sensors, the screen must be kept awake using the code 'getWindow' (). This is ensured by `addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);`

Counter

The sensor's constant is 'TYPE STEP COUNTER,' and the unit of measurement is number of steps. The step counter keeps track of how many steps you've taken since the last reboot while the sensor was turned on. The latency of the step counter is greater than that of the step detector. The step counter is expected to have a latency of up to 10 seconds. Despite having a longer latency, the counter is more accurate than the step detector.

```
@Override
protected void onCreate(@Nullable Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_step);

    getWindow().addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON); //keeps our screen on always
    textViewStepCounter = findViewById(R.id.step);
    textViewStepDetector = findViewById(R.id.step2);

    circularProgressBar = findViewById(R.id.progress_circular);
    circularProgressBar2 = findViewById(R.id.progress_circular2);

    sensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);

    if(sensorManager.getDefaultSensor(Sensor.TYPE_STEP_COUNTER)!=null){
        mStepCounter = sensorManager.getDefaultSensor(Sensor.TYPE_STEP_COUNTER);
        isCounterSensorPresent = true;
    }else{
        textViewStepCounter.setText("Counter Sensor not available");
        isCounterSensorPresent = false;
    }
    if(sensorManager.getDefaultSensor(Sensor.TYPE_STEP_DETECTOR)!=null){
        mStepDetector = sensorManager.getDefaultSensor(Sensor.TYPE_STEP_DETECTOR);
        isDetectorSensorPresent = true;
    }else{
        textViewStepDetector.setText("Counter Detector not available");
        isDetectorSensorPresent = false;
    }
}
```

[FIG 28] [step detector code screenshot]

The method called `onSensorChanged()` is used for each time the event is triggered, when it is, the value will be added to the previous value of the step detector and it will continue from there. The results are represented by setting the number of detected steps to a text view.

```
@Override
public void onSensorChanged(SensorEvent sensorEvent) {
    if(sensorEvent.sensor == mStepCounter){
        stepCount = (int) sensorEvent.values[0];

        textViewStepCounter.setText(String.valueOf(stepCount));
        circularProgressBar.setProgress(stepCount);
    }else if(sensorEvent.sensor == mStepDetector){
        stepDetect = (int) (stepDetect+sensorEvent.values[0]);

        textViewStepDetector.setText(String.valueOf(stepDetect));
        circularProgressBar2.setProgress(stepDetect);
    }
}
```

[FIG 29] [sensor code screenshot]

Vault Activity

This is the method for adding items to the users saving vault. The idea is to be able to allow a user to add a certain amount of money to their vault and associate it with a specific item they are saving for. The onclick listener allows the user to save the change they have made. In the code I have done checks for ensuring that the user has entered an amount of money they wish to add to their vault as well as ensuring they have selected a valid item. If the user fails to meet these requirements, they will be presented with an error message as shown in the code.

```
private void additem() {
    AlertDialog.Builder myDialog = new AlertDialog.Builder(context, this);
    LayoutInflater inflater = LayoutInflater.from(this);
    View myView = inflater.inflate(R.layout.input_layout, root, null);
    myDialog.setView(myView);

    final AlertDialog dialog = myDialog.create();
    dialog.setCancelable(false);

    final Spinner itemspinner = myView.findViewById(R.id.itemspinner);
    final EditText amount = myView.findViewById(R.id.amount);
    final Button cancel = myView.findViewById(R.id.cancel);
    final Button save = myView.findViewById(R.id.save);

    save.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            String budgetAmount = amount.getText().toString();
            String budgetItem = itemspinner.getSelectedItem().toString();

            if(TextUtils.isEmpty(budgetAmount)){
                amount.setError("Amount is required"); //error handling
                return;
            }
            if(budgetItem.equals("Select Items")){
                Toast.makeText(context, VaultActivity.this, text: "Select a valid item", Toast.LENGTH_SHORT).show();
            }
            else{
                loader.setMessage("adding a budget item");
                loader.setCanceledOnTouchOutside(false);
                loader.show();

                String id = budgetRef.push().getKey();
                DateFormat dateFormat = new SimpleDateFormat(pattern: "dd-MM-yyyy");
                Calendar cal = Calendar.getInstance();
                String date = dateFormat.format(cal.getTime());
            }
        }
    });
}
```

[FIG 30] [vault code screenshot]

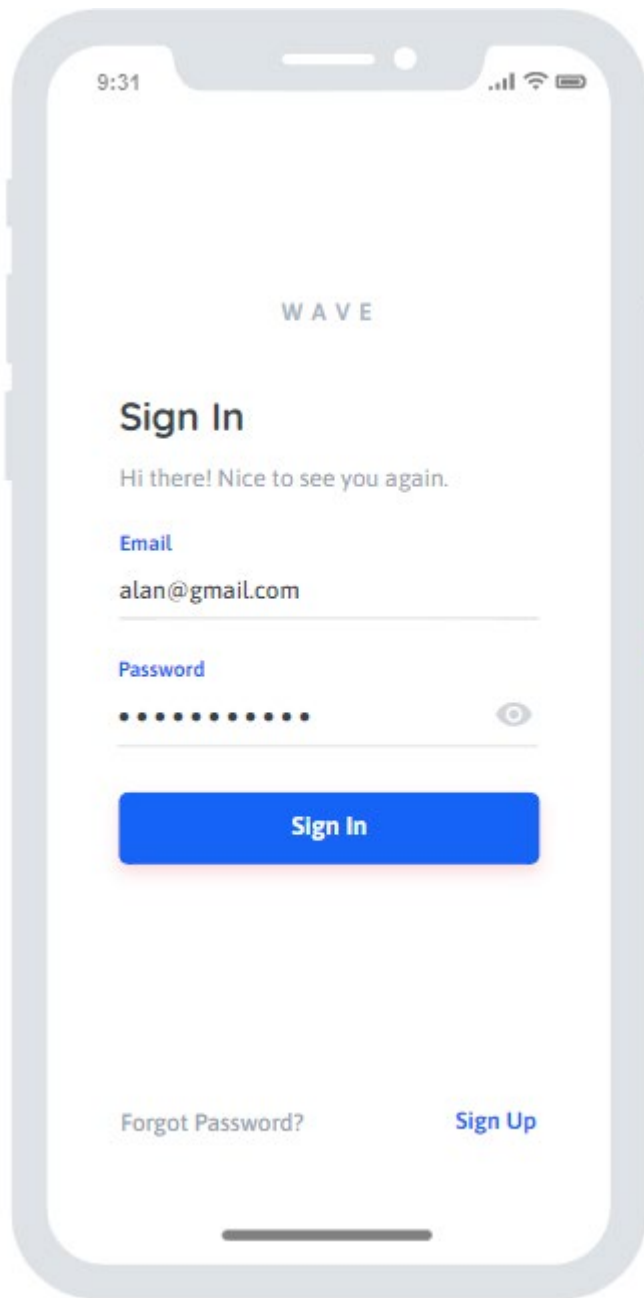
After this method was created, I made sure that the values were inputted to the firebase database by referencing the database and initializing the variables. I ensured that the items added were user specific and whenever an item was added I used "budgetRef.push()" to push the data to the firebase database.

```
String strURL = "https://wave-ccbfd-default-rtdb.europe-west1.firebaseio.com/";
budgetRef =
FirebaseDatabase.getInstance(strURL).getReference("budget").child(mAuth.getCurrentUser().getUid());
```


6.0 Graphical User Interface (GUI)

6.1 GUI – Wireframes

Login page:



[FIG 31] [Login page wireframe]

Registration page:

The wireframe shows a mobile application interface for a registration page. At the top, the status bar displays the time 9:31, signal strength, Wi-Fi, and battery icons. The page title 'Sign Up' is centered at the top. Below the title, there are two input fields: 'Email' with the placeholder text 'Your email address' and 'Password'. A 'Remember me' checkbox is checked. A blue 'Continue' button is positioned below the inputs. At the bottom, there is a link that says 'Have an Account? Sign In'.

9:31

Sign Up

Email

Your email address

Password

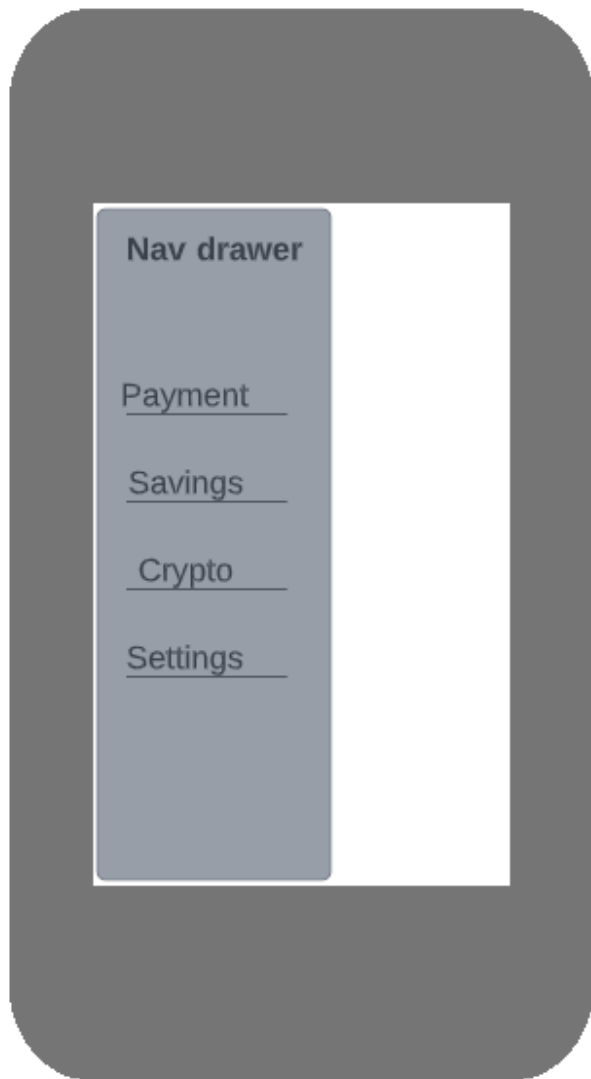
☒ Remember me

Continue

Have an Account? [Sign In](#)

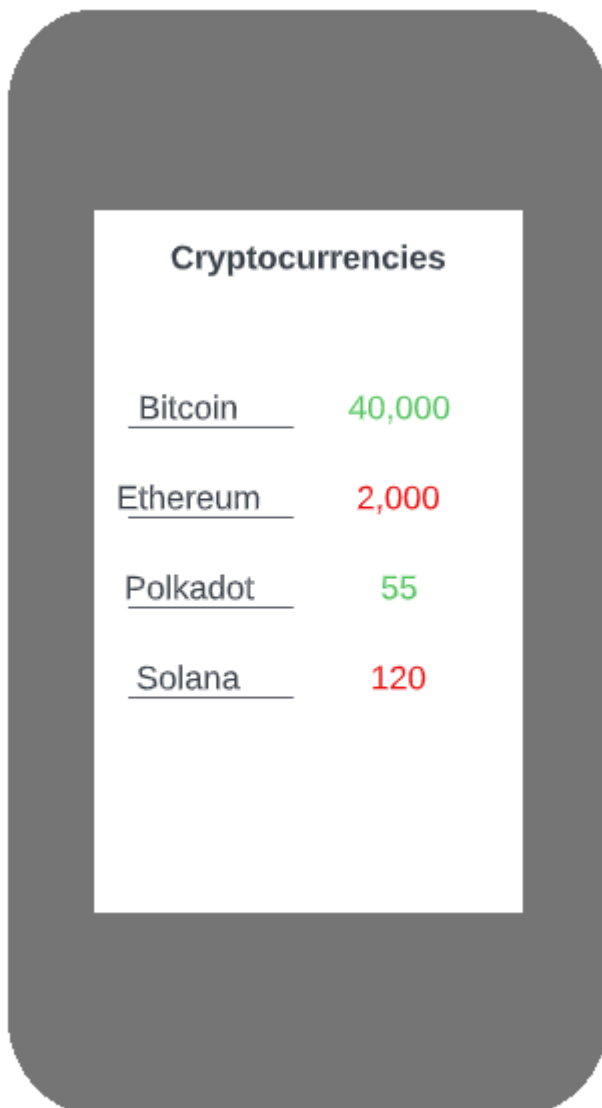
[FIG 32] [registration page wireframe]

Navigation



[FIG 33] [navigation page wireframe]

Cryptocurrency

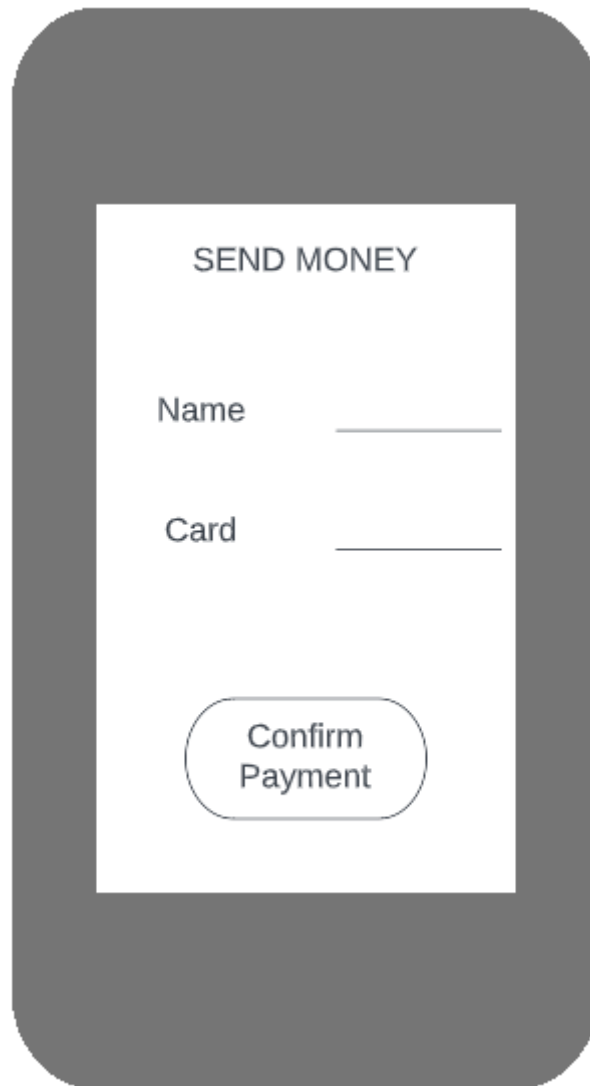


A wireframe of a mobile application page titled "Cryptocurrencies". The page features a list of four cryptocurrencies: Bitcoin, Ethereum, Polkadot, and Solana. Each entry consists of the name underlined, followed by its value. Bitcoin's value is 40,000 (green), Ethereum's is 2,000 (red), Polkadot's is 55 (green), and Solana's is 120 (red). The entire content is enclosed in a dark gray rounded rectangle.

Cryptocurrencies	
<u>Bitcoin</u>	40,000
<u>Ethereum</u>	2,000
<u>Polkadot</u>	55
<u>Solana</u>	120

[FIG 34] [cryptocurrency page wireframe]

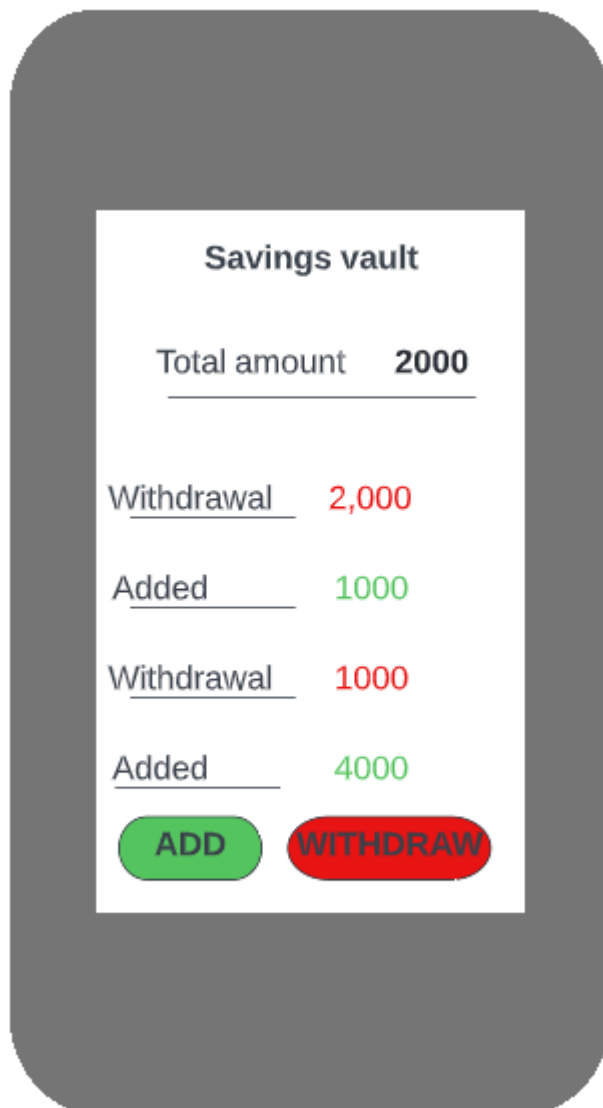
Send money



The wireframe shows a mobile app interface for sending money. It features a dark gray rounded rectangle representing the phone. Inside, a white rectangle contains the title "SEND MONEY" at the top. Below the title are two input fields: "Name" and "Card", each followed by a horizontal line for text entry. At the bottom of the white area is a rounded rectangular button with the text "Confirm Payment".

[FIG 35] [send money page wireframe]

Savings vault



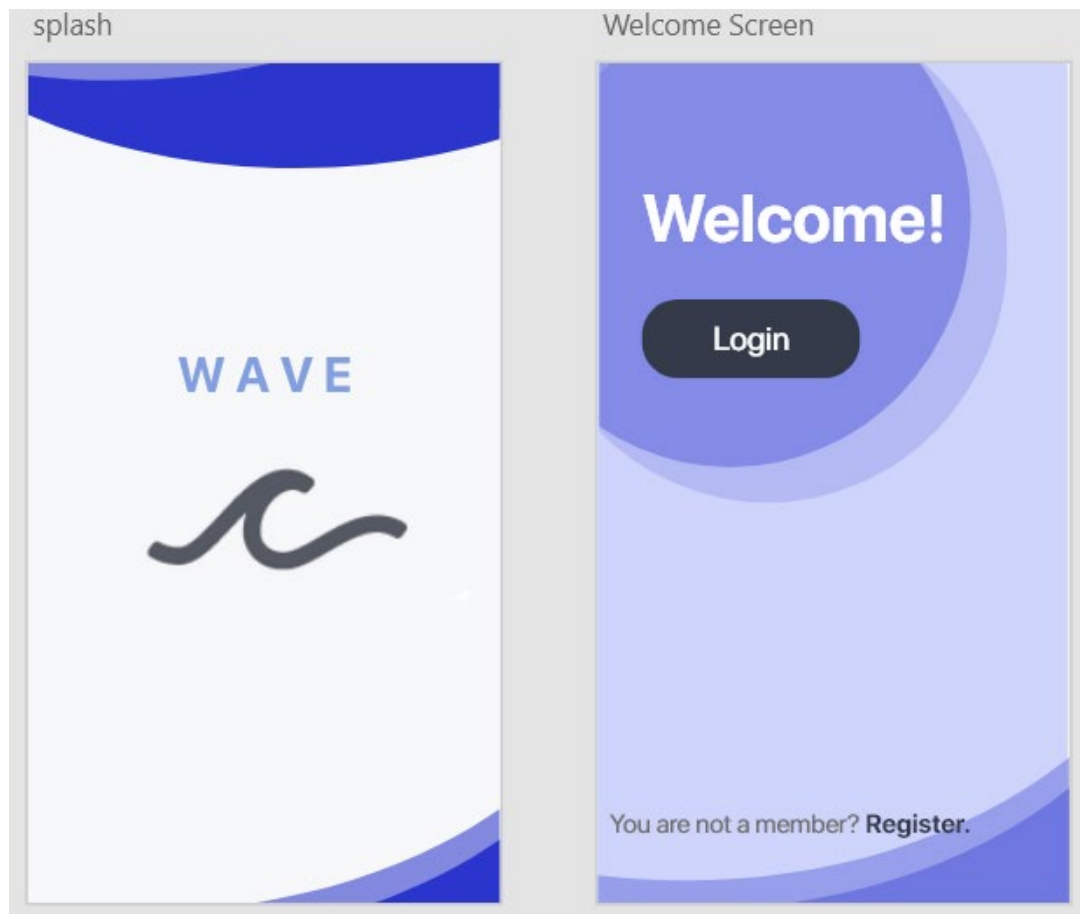
A wireframe of a mobile application screen titled "Savings vault". The screen has a dark gray background with rounded corners. In the center is a white rectangular area containing the following elements:

- Savings vault**: A title in bold black text.
- Total amount 2000**: A label and value in black text, underlined.
- Withdrawal 2,000**: A label and value in red text, underlined.
- Added 1000**: A label and value in green text, underlined.
- Withdrawal 1000**: A label and value in red text, underlined.
- Added 4000**: A label and value in green text, underlined.
- ADD**: A green rounded rectangular button.
- WITHDRAW**: A red rounded rectangular button.

[FIG 36] [savings vault page wireframe]

Home page

1. Opening the application
 - a. The first screen the user is presented with when opening the mobile application, I am currently developing, is a simplistic splash screen which shows the logo I created which is displayed for 1 second. The splash screen fades away, and the user is then presented with a welcome page, here the user has the option to either 'Login' or 'Register'. The user's decision will decide whether they are presented with a login or registration page.



[FIG 37] [splash and welcome screen UI]

2. Login and Registration

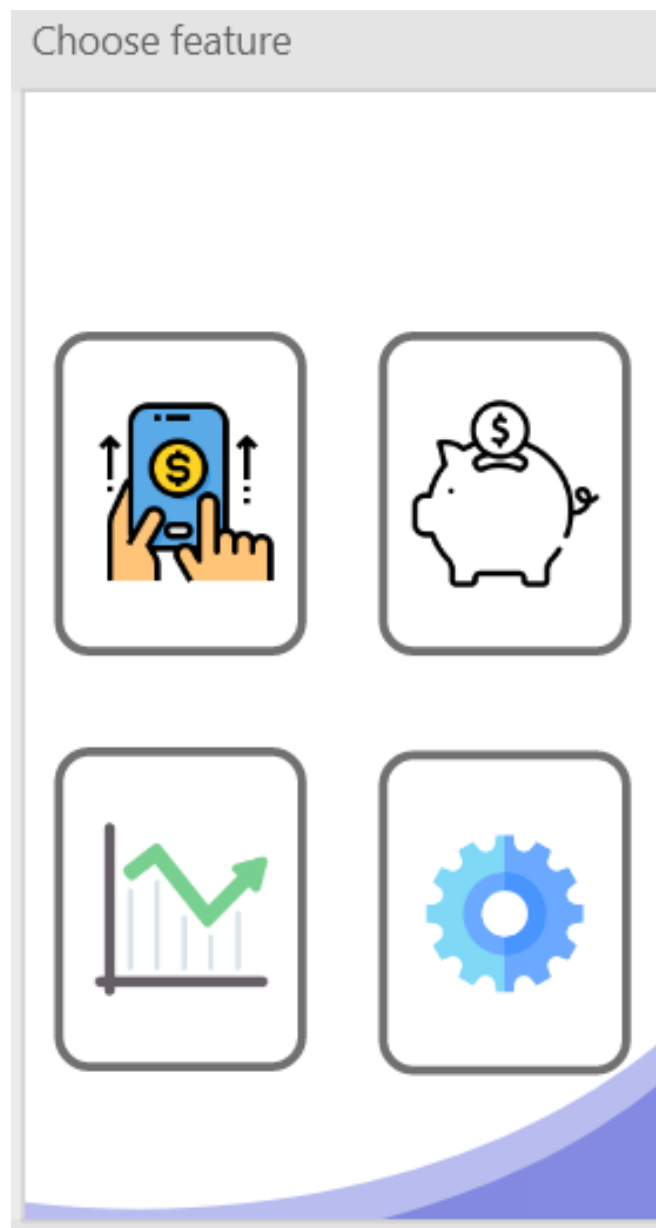
- a. If the user already has an account created, it is safe to say they most likely decided to press the login button shown in the UI above. Considering this was the case, the user will be presented with the login page shown below on the left. All they have to do is simply enter their email and password to login as their details have already been stored in the database.
- b. However, if the user is new and they do not have an account, I assume they would have clicked 'Register' in the UI above. Considering they did, the user will be presented with a registration page as shown on the bottom right. Here a user must enter an email followed by a 6-digit password in which they must confirm.

The image displays two mobile application screens side-by-side, both featuring a light gray header and a blue gradient footer. The left screen, titled 'Login', has a back arrow icon, the text 'Hello!', an email input field containing 'alan@gmail.com', a password input field with six dots, and a blue 'Login' button. The right screen, titled 'Register', also has a back arrow icon, the text 'Create an account', an email input field containing 'alan@gmail.com', a 'Password' input field with six dots, a 'Confirm Password' input field with six dots, and a blue 'Register' button.

[FIG 38] [login and registration UI]

3. Navigable grid view

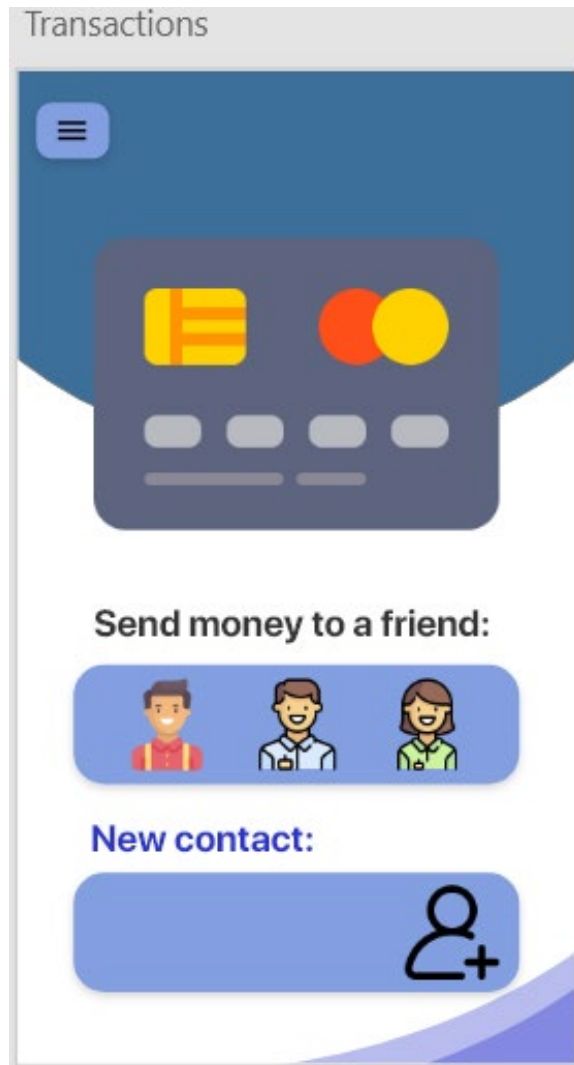
- a. Whether a user logged in or a new user registered an account, post login and registration the user will be presented with a grid view displaying four options. These four options are the core functionalities of the application. Top left option should be selected when a user wants to deal with transactions, top right option is for gaining access to the user's savings monitoring vault, bottom left option should be selected whenever the user would like to check the current state of the cryptocurrency market and the fourth and final option should be selected when the user would like to logout, change their email address or reset their password.



[FIG 39] [feature page UI]

4. Transactions

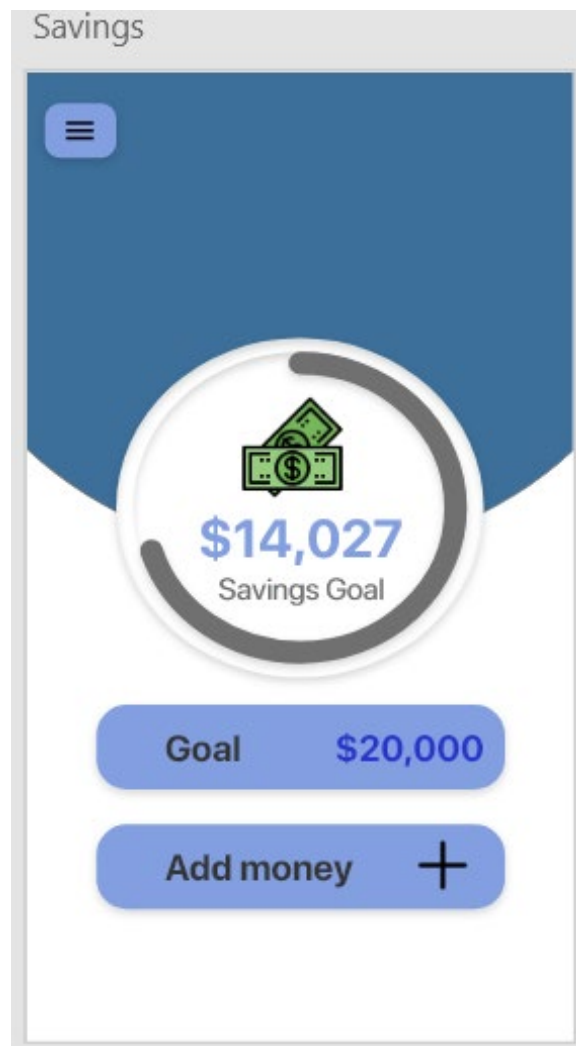
- a. In this section, considering the user selected the top right option in the card view above, the user will be able to deal with payments. Whether its sending money to a friend or paying for an item, this section will cover all payments with the aid of Stripes API.



[FIG 40] [transaction page UI]

5. Savings Monitoring

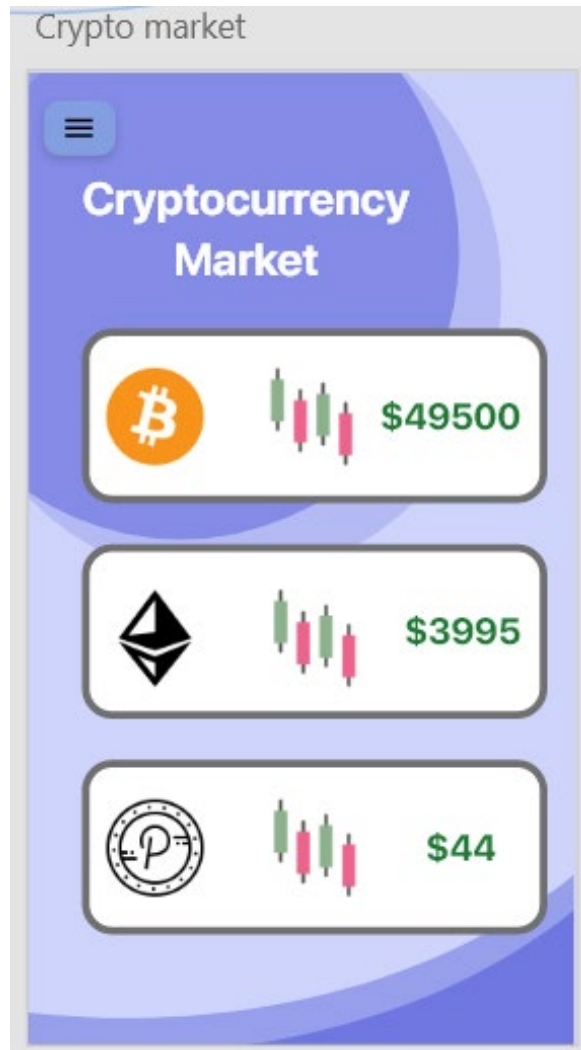
- a. Considering the user selected the top right option, they would then be presented with the savings monitoring screen. Here the user will see how much money they have saved up in comparison to what their goal is, this has been represented by a progress bar. The current amount in the users account is \$14,027 whilst the goal is just over \$20,000 which explains why the progress bar is positioned around 70% complete in regard to coming full circle. Here the user will have the option to add and withdraw money.



[FIG 41] [savings vaultUI]

6. Cryptocurrency Market

- a. In this screen we can see that there are currently three different listings being shown. The recycler view contains 100 cryptocurrencies in total which are pulled from coinmarketcaps API, in this screenshot there are three being shown. The three being shown are bitcoin, ethereum and Polkadot. The view will show us the cryptocurrency name and current price



[FIG 42] [crypto page UI]

7. Settings

- a. Currently my plans for this page are to allow the user to change their email address and reset their password. I have also provided the user with the option to logout of the mobile application.



[FIG 43] [settings page UI]

7.0 Testing

7.1 JUnit testing

Seeing as my application necessitates a great deal of interaction between the user and the user interface, I wanted to write UI unit tests to examine the various application components with which the user will interact. I used Espresso to help me test my user interface and ran tests for multiple test cases.

My first example is the login activity. The code I have provided as the Espresso flow is exactly the same as the user would intend on using this page. I have set two variables for my email and password. I then perform a `typeText` action the login and password text views whilst passing my login variables. Then before we click the login button, it is necessary to close the keyboard which is shown in the snapshot below. As the last step, a click is performed on the login button. Espresso makes these actions easy to understand the flow of the code. *“Espresso tests state expectations, interactions, and assertions clearly without the distraction of boilerplate content, custom infrastructure, or messy implementation details getting in the way.”* [Espresso, testing framework] The screenshots for the other tests will also be provide below.

```
* @see <a href="http://d.android.com/tools/testing">Testing documentation</a>
 */
@RunWith(AndroidJUnit4.class)
public class LoginTests {

    @Rule
    public ActivityTestRule<LogActivity> mLogActivityActivityTestRule =
        new ActivityTestRule<>>(LogActivity.class);

    @Test
    public void enterLoginDetails_logsUserIn() {
        String email = "alan14@gmail.com";
        String password = "123456";
        onView(withId(R.id.email)).perform(typeText(email));
        onView(withId(R.id.password)).perform(typeText(password));
        Espresso.closeSoftKeyboard();
        onView(withId(R.id.loginButton2)).perform(click());
    }
}
```

[FIG 44] [espresso login test code]

```

@RunWith(AndroidJUnit4.class)
public class CryptoTests {

    @Rule
    public ActivityTestRule<cryptoActivity> mCryptoActivityActivityTestRule =
        new ActivityTestRule<>>(cryptoActivity.class);

    @Test
    public void enterCryptoName_DisplaysCrypto() {
        String cryptocurrency = "DOT";
        onView(withId(R.id.EditSearch)).perform(typeText(cryptocurrency));
        Espresso.closeSoftKeyboard();
        onView(withText(cryptocurrency));
    }
}

```

[FIG 45] [espresso search crypto test code]

```

@RunWith(AndroidJUnit4.class)
public class MoneySpentTodayTests {

    @Rule
    public ActivityTestRule<MoneySpentTodayActivity> mVaultActivityActivityTestRule =
        new ActivityTestRule<>>(MoneySpentTodayActivity.class);

    @Test
    public void enterMoneySpentTodayAmount_addsEntryToMoneySpentToday() {
        String amount = "1000";
        String note = "Shopping";
        onView(withId(R.id.fab)).perform(click());
        onView(withId(R.id.amount)).perform(typeText(amount));
        onView(withId(R.id.note)).perform(typeText(note));
        onView(withId(R.id.save)).perform(click());
        Espresso.closeSoftKeyboard();
    }
}

```

[FIG 46] [espresso add money spent todayvault test code]

```

@RunWith(AndroidJUnit4.class)
public class RegisterTests {

    @Rule
    public ActivityTestRule<RegisterActivity> mRegisterActivityActivityTestRule =
        new ActivityTestRule<>(RegisterActivity.class);

    @Test
    public void enterRegistrationDetails_RegistersUser() {
        String email = "alan14@gmail.com";
        String password = "123456";
        onView(withId(R.id.email2)).perform(typeText(email));
        //.check();
        onView(withId(R.id.rpassword)).perform(typeText(password));
        onView(withId(R.id.rpassword2)).perform(typeText(password));
        Espresso.closeSoftKeyboard();
        onView(withId(R.id.registerBtn)).perform(click());
        //.check();
    }
}

```

[FIG 47] [espresso register user test code]

```

@RunWith(AndroidJUnit4.class)
public class VaultTests {

    @Rule
    public ActivityTestRule<VaultActivity> mVaultActivityActivityTestRule =
        new ActivityTestRule<>(VaultActivity.class);

    @Test
    public void enterVaultAmount_addsMoneyToVault() {
        String amount = "1000";
        onView(withId(R.id.fab)).perform(click());
        onView(withId(R.id.amount)).perform(typeText(amount));
        onView(withId(R.id.save)).perform(click());
        Espresso.closeSoftKeyboard();
    }
}

```

[FIG 48] [espresso add to vaulttest code]

7.2 Test plan – Manual testing

I created a testing template that I expanded on as new innovations were introduced during the development phase. This testing template provided a foundation for me to develop test strategies for each new feature added to the project. Whenever a merge request for a new feature was submitted, I would run through the testing template to ensure that the new changes did not cause any system bugs. These test events were recorded and are shown below. Espresso was created with the goal of allowing developers to design UI tests that are simple, dependable, and use a fluent API. [Espresso docs, 2022]. I didn't have to worry about view state transitions or implementation details because Espresso handles UI event synchronisation.

Test title	Home screen
Test description	Testing if the card view layout takes the user to the correct page
Type of test	Manual
Expected Outcome	<p>The home screen card view presents the user with four options.</p> <ul style="list-style-type: none">- Payments- Cryptocurrencies- Savings vault- Settings <p>It is expected that when the user selects one of these options in the card view, the system will take the user to the associated page.</p>
Actual Outcome	N/A

Test title	Register button
Test description	Testing if the registration page stores the users' credentials in the firebase database.
Type of test	Manual
Expected Outcome	<p>On the registration page the system accepts 2 different forms of input from the user.</p> <ul style="list-style-type: none">- Email address- Password <p>It is expected that these records should be stored safely in the firebase database.</p>
Actual Outcome	N/A

Test title	Login button
Test description	Testing if the login page only accepts users' credentials who exist in the firebase database.
Type of test	Manual
Expected Outcome	<p>On the login page the system accepts 2 different forms of input from the user.</p> <ul style="list-style-type: none"> - Email address - Password <p>It is expected that when these two results match records in the firebase database, the system will give the user access to the application.</p> <p>The user should not be able to gain access with credentials that don't exist.</p>
Actual Outcome	N/A

Test title	Search function for cryptocurrencies
Test description	Testing if a cryptocurrency can be retrieved by searching for its name.
Type of test	Manual
Expected Outcome	<p>As default, the user will be presented with the top 100 cryptocurrencies. I need to test if a user can apply a search filter based off the cryptocurrencies name. Search functionality will not be applied to the prefix of the cryptocurrency.</p> <p>E.g., to find Bitcoin, the user must search for matching letters in the String, the user will not be able to search for Bitcoins prefix which is BTC.</p>
Actual Outcome	N/A

Test title	Send money via PayPal
Test description	Testing to see if a user can transfer funds to another user using PayPal API.
Type of test	Manual
Expected Outcome	After the user has finished filling out the payment details within my application and they are ready to pay. Once the user clicks pay, the PayPal activity should load where it prompts the user to login to their PayPal account. The user can review the payment before sending it. Once the user confirms the payment the system should prompt the user saying, 'Payment was Successful'.
Actual Outcome	N/A

Test title	Send money via Google Pay
Test description	Testing to see if a user can transfer funds to another user using Google Pay API.
Type of test	Manual
Expected Outcome	After the user has finished filling out the payment details within my application and they are ready to pay. Once the user clicks pay, the Google Pay activity should load where it prompts the user to login to their Google Pay account. The user can review the payment before sending it. Once the user confirms the payment the system should prompt the user saying, 'Payment was Successful'.
Actual Outcome	N/A

Test title	Make a purchase using Stripe API
Test description	Testing to see if a user can make an in-app purchase using the Stripe API
Type of test	Manual
Expected Outcome	The user has the option to upgrade to a new card. When the user clicks on this feature they should be presented with the Stripe API where they are asked to enter their credit card details. Considering the user enters the correct card details, the system should prompt them saying that the payment was successful.
Actual Outcome	N/A

Test title	View daily spendings
Test description	Testing to see if a user can view the spendings they made today
Type of test	Manual
Expected Outcome	When the user goes to the daily spendings activity. They should be presented with any spendings they have made in the last 24 hours. The user should be able to apply CRUD functionalities to these items.
Actual Outcome	N/A

Test title	View weekly spendings
Test description	Testing to see if a user can view the spendings they made this week.
Type of test	Manual
Expected Outcome	When the user goes to the weekly spendings activity. They should be presented with any spendings they have made in the last 7 days.
Actual Outcome	N/A

Test title	View monthly spendings
Test description	Testing to see if a user can view the spendings they made this month.
Type of test	Manual
Expected Outcome	When the user goes to the monthly spendings activity. They should be presented with any spendings they have made in the last 31 days.
Actual Outcome	N/A

Test title	Use map activity
Test description	Testing to see if the user can use the map functionality to find the nearest ATM.
Type of test	Manual
Expected Outcome	Testing to see if a user can input a to and from destination in the application. When the user submits the application should redirect the user to the google maps application with both inputs filled in. If the user does not have google maps downloaded the application should redirect the user to the play store.
Actual Outcome	N/A

Test title	Logout button
Test description	Testing to see if the user can successfully log out.
Type of test	Manual
Expected Outcome	When the user is on the settings activity and presses the logout button, the application should redirect the user to the applications login page. When the user logs out, the next time they press login they should not be granted instant access to the application. The expected outcome is that the user should have to enter their correct credentials to regain access to the application.
Actual Outcome	N/A

Test title	View daily analytics
Test description	Testing to see if the user can see their daily spending analytics.
Type of test	Manual
Expected Outcome	When the user clicks on the daily analytics activity, it is expected that they will be presented with the daily spendings on a pie chart.
Actual Outcome	N/A

Test title	View weekly analytics
Test description	Testing to see if the user can see their weekly spending analytics.
Type of test	Manual
Expected Outcome	When the user clicks on the weekly analytics activity, it is expected that they will be presented with the weekly spendings on a pie chart.
Actual Outcome	N/A

Test title	View monthly analytics
Test description	Testing to see if the user can see their monthly spending analytics.
Type of test	Manual
Expected Outcome	When the user clicks on the monthly analytics activity, it is expected that they will be presented with the monthly spendings on a pie chart.
Actual Outcome	N/A

7.3 Test plan implementations

Merge Request (Login button, November 28th, 2021). Manual testing plan idea came from how to write test cases: sample template testing with examples [Thomas Hamilton] [2022].

Test title	Type of test	Outcome
Home card view	Manual	Pass

Test title	Type of test	Outcome
Register button	Manual	Pass

Test title	Type of test	Outcome
Login button	Manual	Pass

Merge Request (Search for cryptocurrencies, December 14th, 2021)

Test title	Type of test	Outcome
Home card view	Manual	Pass
Register button	Manual	Pass
Login button	Manual	Pass
Search function for cryptocurrencies	Manual	Pass

Merge Request (Send money via PayPal, February 14th, 2022)

Test title	Type of test	Outcome
Home card view	Manual	Pass
Register button	Manual	Pass
Login button	Manual	Pass
Search function for cryptocurrencies	Manual	Pass
Send money via PayPal	Manual	Partial Pass – fixed amount, user cannot enter customer amount of money.

Merge Request (Send money via Google Pay, March 1st, 2022)

Test title	Type of test	Outcome
Home card view	Manual	Pass
Register button	Manual	Pass
Login button	Manual	Pass
Search function for cryptocurrencies	Manual	Pass
Send money via PayPal	Manual	Pass
Send money via Google Pay	Manual	N/A – cannot test this feature as I do not have access to an Indian Google pay account

Merge Request (Make a payment using Stripe, March 12th, 2022)

Test title	Type of test	Outcome
Home card view	Manual	Pass
Register button	Manual	Pass
Login button	Manual	Pass
Search function for cryptocurrencies	Manual	Pass
Send money via PayPal	Manual	Pass
Send money via Google Pay	Manual	N/A
Make a payment using Stripe	Manual	pass

Merge Request (View daily spendings, March 18th, 2022)

Test title	Type of test	Outcome
Home card view	Manual	Pass
Register button	Manual	Pass
Login button	Manual	Pass
Search function for cryptocurrencies	Manual	Pass
Send money via PayPal	Manual	Pass
Send money via Google Pay	Manual	N/A
Make a payment using Stripe	Manual	Pass
View daily spendings	Manual	Pass

Merge Request (View weekly spendings, March 21st, 2022)

Test title	Type of test	Outcome
Home card view	Manual	Pass
Register button	Manual	Pass
Login button	Manual	Pass
Search function for cryptocurrencies	Manual	Pass
Send money via PayPal	Manual	Pass
Send money via Google Pay	Manual	N/A
Make a payment using Stripe	Manual	Pass
View daily spendings	Manual	Pass
View weekly spendings	Manual	Pass

Merge Request (View daily spendings, March 22nd, 2022)

Test title	Type of test	Outcome
Home card view	Manual	Pass
Register button	Manual	Pass
Login button	Manual	Pass
Search function for cryptocurrencies	Manual	Pass
Send money via PayPal	Manual	Pass
Send money via Google Pay	Manual	N/A
Make a payment using Stripe	Manual	Pass
View daily spendings	Manual	Pass
View weekly spendings	Manual	Pass
View monthly spendings	Manual	Partial pass – having some errors, weekly activity shows instead of monthly activity at times

Merge Request (View daily spendings, March 25th, 2022)

Test title	Type of test	Outcome
Home card view	Manual	Pass
Register button	Manual	Pass
Login button	Manual	Pass
Search function for cryptocurrencies	Manual	Pass
Send money via PayPal	Manual	Pass
Send money via Google Pay	Manual	N/A
Make a payment using Stripe	Manual	Pass
View daily spendings	Manual	Pass
View weekly spendings	Manual	Pass
View monthly spendings	Manual	Pass
View savings vault	Manual	Pass

Merge Request (View daily spendings, March 26th, 2022)

Test title	Type of test	Outcome
Home card view	Manual	Pass
Register button	Manual	Pass
Login button	Manual	Pass
Search function for cryptocurrencies	Manual	Pass
Send money via PayPal	Manual	Pass
Send money via Google Pay	Manual	N/A
Make a payment using Stripe	Manual	Pass
View daily spendings	Manual	Pass
View weekly spendings	Manual	Pass
View monthly spendings	Manual	Pass
View savings vault	Manual	Pass
Logout	Manual	Pass

Merge Request (View daily spendings, March 28th, 2022)

Test title	Type of test	Outcome
Home card view	Manual	Pass
Register button	Manual	Pass
Login button	Manual	Pass
Search function for cryptocurrencies	Manual	Pass
Send money via PayPal	Manual	Pass
Send money via Google Pay	Manual	N/A
Make a payment using Stripe	Manual	Pass
View daily spendings	Manual	Pass
View weekly spendings	Manual	Pass
View monthly spendings	Manual	Pass
View savings vault	Manual	Pass
Logout	Manual	Pass
Use map activity	Manual	Pass

Merge Request (View daily spendings, April 5th, 2022)

Test title	Type of test	Outcome
Home card view	Manual	Pass
Register button	Manual	Pass
Login button	Manual	Pass
Search function for cryptocurrencies	Manual	Pass
Send money via PayPal	Manual	Pass
Send money via Google Pay	Manual	N/A
Make a payment using Stripe	Manual	Pass
View daily spendings	Manual	Pass
View weekly spendings	Manual	Pass
View monthly spendings	Manual	Pass
View savings vault	Manual	Pass
Logout	Manual	Pass
Use map activity	Manual	Pass
Step activity	Manual	Pass

Merge Request (View daily spendings, April 5th, 2022)

Test title	Type of test	Outcome
Home card view	Manual	Pass
Register button	Manual	Pass
Login button	Manual	Pass
Search function for cryptocurrencies	Manual	Pass
Send money via PayPal	Manual	Pass
Send money via Google Pay	Manual	N/A
Make a payment using Stripe	Manual	Pass
View daily spendings	Manual	Pass
View weekly spendings	Manual	Pass
View monthly spendings	Manual	Pass
View savings vault	Manual	Pass
Logout	Manual	Pass
Use map activity	Manual	Pass
Step activity	Manual	Pass
Daily analytics	Manual	Fail

Merge Request (View daily spendings, May 8th, 2022)

Test title	Type of test	Outcome
Home card view	Manual	Pass
Register button	Manual	Pass
Login button	Manual	Pass
Search function for cryptocurrencies	Manual	Pass
Send money via PayPal	Manual	Pass
Send money via Google Pay	Manual	N/A
Make a payment using Stripe	Manual	Pass
View daily spendings	Manual	Pass
View weekly spendings	Manual	Pass
View monthly spendings	Manual	Pass
View savings vault	Manual	Pass
Logout	Manual	Pass
Use map activity	Manual	Pass
Step activity	Manual	Pass
Daily analytics	Manual	Pass

8.0 Evaluation

- As I stated at the outset of this report, I believe my application has market potential in today's market. My application successfully provides users with essential features such as:
- Making payments
- Sending money to other contacts
- Monitoring today's cryptocurrency market
- Managing their savings

The four core features that I mentioned were successfully implemented. Along the way I made time for implementing some additional activities. I implemented a map activity that directs users to the nearest ATM, the health of my users is at my greatest interest, to help keep them motivated I have provided a feature that helps monitor the total number of steps they have made today as well as the total number of steps they have made since they first opened my application.

Overall, I am impressed with the level of testing I was able to conduct. The Espresso framework was excellent for assisting me with automating tests in a simple yet effective manner. One method of testing was not deemed enough, I made sure to manually test the different components within my application to ensure that they are fit for the end user.

AdobeXD was a great tool for assisting me with designing my user interface. I stuck with a white/blue colour scheme and ensured consistency all throughout my application.

9.0 Conclusions

My application can allow users to successfully send money to other contacts, monitor cryptocurrencies, monitor their savings accounts whilst assisting with other consumer needs. My application offers a navigable user interface, appropriate usability design, smooth execution of functionality, and the steps for implementing windows server failover clustering if an investment was provided.

I would say that the main disadvantage of my application is that it is not compatible with IOS devices. If I had the chance to spend more time working with it, I would definitely work on ensuring that it is compatible in a future release.

Payment-driven programming challenges and numerous asynchronous requests to APIs such as Stripe, PayPal, and Google Pay have been shown to consume a significant amount of time. Displaying the users' daily spendings was an activity that took up a lot of time near the end of my project.

My strengths range between the different APIs I have utilized; the PayPal, Stripe, and Google Pay APIs ensure that users transactions will be handled securely. The firebase database may also be considered a strength as it handles the encryption of the users' passwords.

10.0 Further Development or Research

If I could do additional work with my project, I would not continue coding without conducting interviews and surveys to gather some user feedback. I feel as if this would be a wise move, I will gather useful information regarding the user interface and the overall functionality of my application.

An important feature which would be implemented in future releases would be the ability to receive money that has been sent from another user. I would like to move away from using PayPal and Google's API and develop my own stream where I can manage transactions in a similar manner to Revolut. There are some more features around user logins that I would like to add. Firstly, I would like to ensure that every user that is registered must verify their email address and in the case that a user forgets their password, I would like to present the ability of being able to reset their password via the email address they have verified when they first created their account.

Some more analytic features could present some useful information that could be requested by the user. As it stands, the user can analyse their daily spendings, but it would be great to implement weekly and monthly analytics.

With funding, I would be presented with the ability to implement windows server failover clustering. I would like to move away from storing data in the firebase database and instead, store it in a MySQL database. With the MySQL database I would like to utilize high availability by implementing windows server failover clustering.

It would have been great to demonstrate internationalisation so that users who have their devices set to a different language can still use the application.

11.0 References

prnewswire, 2019, Global Fintech Market Value is Expected to Reach \$309.98 Billion at a CAGR of 24.8% Through 2022

Available at: <https://www.prnewswire.com/news-releases/global-fintech-market-value-is-expected-to-reach-309-98-billion-at-a-cagr-of-24-8-through-2022--300926069.html>

Tuomo Sippola, 2017. Usability is a key element of user experience.

Available at: <https://eu.landisqyr.com/better-tech/usability-is-a-key-element-of-user-experience>

Raven Veal, 2021, How to define user persona

Available at: <https://careerfoundry.com/en/blog/ux-design/how-to-define-a-user-persona/>

doc.microsoft.com, 2022, Failover Clustering in Windows Server and Azure Stack HCI

Available at: <https://docs.microsoft.com/en-us/azure-stack/hci/manage/maintain-servers>

developer.android, 2021, Espresso documentation

Available at: <https://developer.android.com/training/testing/espresso>

developer.paypal, 2022, Native Checkout SDK

Available at: <https://developer.paypal.com/limited-release/native-checkout/>

stripe docs, 2020, Accept a payment

Available at: <https://stripe.com/docs/payments/accept-a-payment>

Google Pay, 2021, Support Google Pay for in-app payments

Available at: <https://developers.google.com/pay/india/api/android/in-app-payments>

CoinMarketCap, 2021, CoinMarketCap API Documentation

Available at: <https://coinmarketcap.com/api/documentation/v1/>

Vishwas Ng, 2019, Non-functional Requirement of the Mobile Development System

Available at: <https://medium.com/@vishwasng/non-functional-requirement-of-the-mobile-development-system-e0ed98f2a872>

Central bank, 2022, What is “fintech” and how is it changing financial products?

Available at: <https://www.centralbank.ie/consumer-hub/explainers/what-is-fintech-and-how-is-it-changing-financial-products#:~:text=The%20word%20%E2%80%9Cfintech%E2%80%9D%20is%20simply,anythin%20that%20relates%20to%20finance.>

Geeksforgeeks, 2020, How to add a pie chart into an android application

Available at: <https://www.geeksforgeeks.org/how-to-add-a-pie-chart-into-an-android-application/>

Developers.google.com, 2021, Google Maps Intents for Android

Available at: <https://developers.google.com/maps/documentation/urls/android-intents>

developer.android.com, 2022, Motion sensors

Available at: https://developer.android.com/guide/topics/sensors/sensors_motion

firebase.google.com, 2022, Get Started with Firebase Authentication on Android

Available at: <https://firebase.google.com/docs/auth/android/start>

Lucidchart, 2022, design tool

Available at: <https://www.lucidchart.com/pages/>

Xtensio, 2022, design user personas

Available at: <https://xtensio.com/>

Thomas Hamilton, 2022, How to write test cases: Sample Template with Examples

Available at: <https://www.guru99.com/test-case.html>

Project Proposal

Contents

1.0	Objectives	2
2.0	Background	2
3.0	State of the Art	3
4.0	Technical Approach	3
5.0	Technical Details	4
6.0	Special Resources Required	5
7.0	Project Plan	6
8.0	Testing	8
9.0	References	8

1.0 Objectives

The overall goal of this project is to provide a high availability fintech mobile application with a high-level standard of database security. High availability is self-explanatory, it is the ability of a system to continue functioning even when some of its components fail. Ensuring high availability, you help your organisation achieve maximum productivity and reliability. Service disruption may lead to negative impacts of downtime, with a high availability strategy in place it can help mitigate any financial losses.

Financial information stored in databases would be considered valuable assets, the database I plan on designing will hold confidential and sensitive information. Holding such valuable information would attract cyberattacks which in turn, would require a secure database to prevent these database breaches from happening. Implementing database security best practices will ensure a customer never feels vulnerable when storing their personal data when using my mobile application.

The end user will be able to navigate through my mobile applications UI and use it as a financial management application. The mobile application will prioritise transactions, investment management, savings monitoring as well as other financial details. The user will be able to keep track of their trading portfolio whilst also keeping track of their progress when saving capital.

2.0 Background

The fintech sector is an ever-progressive platform, fintech statistics show from case studies performed in Australia, the UK, and the US, have shown us that the fintech statistics have risen from 16% in 2015, rising to 31% in 2017 and reaching up to 60% in 2019. In a popular technology article, it was mentioned that *“The global **fintech market** was worth \$127.66 billion in 2018, and it is expected to reach \$309.98 billion at a CAGR Of 24.8% through 2022.” [prenewswire] [2019]*. It is also estimated that 75% of millennials in the US switched to digital banking, I expect that this will lead to massive growth opportunities for digital finance services.

To this day, companies still seem to be experiencing issues with providing high availability and highly secure systems. High availability, database security and developing have always been areas of interest to me. I believe providing these three topics to a good standard is necessary when creating a mobile application for an end user. When a system or database fails, organisations require high availability protection. They require high availability to keep their systems up and running and to mitigate loss of revenue, brand damage and unhappy customers. The high availability method I plan on implementing is known as ‘Failover clustering’. Failover clustering is a group of independent servers that work together to increase the availability of systems and applications. If a service fails, the services that were hosted on that node can be automatically transferred to another available node.

There are many reasons as to why database security is important seen as sensitive and personal data is on the line. I will meet my objectives by following some of Microsoft’s best practices as well as some practices I have learnt from working as a database administrator.

3.0 State of the Art

The banking as a fintech category is mainly kept back for conventional banks, who are scaling and progressing into all domains of fintech. Banking is the prime category by several applications, whilst payments are the biggest sub-vertical in the fintech apps category. The snapshot below is an excel spreadsheet I created to show the top 10 fintech applications of 2021.

Revolut is a top fintech app I would use on a daily basis. Revolut offers a blend of banking services and finance management services to its consumers. Revolut's top features consist of notifications for each payment, zero fees for account opening and modern fintech functionalities such as crypto trading. These features will also be found within my application however, I will also offer features such as offering savings, payments, and investments all in the one place whilst also seeking to improve quick notifications of transactions. I was looking through some reviews that were left on the IOS apple store, it seems to be a common trend that people weren't impressed with the overloading of ads and information being presented in front of them. One user mentioned that *"it's hard to find useful features and constantly having ads coming up on the home page"*. My goal is that my application will only present necessary information, as well as easy access to the most important pages within the application whilst providing a user-friendly user interface.

4.0 Technical Approach

Before I progress to fully developing my mobile application, there are some areas that I will need to do further research on to broaden my knowledge and ensure that this solution will be the best of my ability. Before I started coding in android studio or designing a mySQL database, I started designing the user interface for my mobile application and planning out my applications use cases. I waited until I was satisfied with the UI I designed; I then began creating an Entity Relationship Diagram. An ERD is a diagram that displays the relationship of entity sets stored in a database. From designing the ERD, I will have an idea of what database tables I will need to create when designing the SQL database. Post ERD design, I plan on moving onto designing the SQL database for my mobile application.

Developing the mobile application

I plan on developing my application in android studio. I researched a variety of different IDE's and concluded that Android Studio is most fitting for my requirements. Android studio is the official integrated development environment for Google's Android operating system, it is specifically designed for Android Development. Some tasks identified consist of: Linking the UI, adding functionality to core features such as login and registration pages, savings monitoring, transactions, investments, and a navigation drawer.

Designing the UI

Designing a navigable User Interface is key to my project, I need to ensure that the end user can seamlessly navigate through my application. From looking at other applications on the IOS apple

store I browsed through some of the app's reviews. It is a common occurrence that users prefer to keep the UI nice and simple, rather than being bombarded with tonnes of information at once. The UI should consist of access to its core features and the users' main attractions. Things can get messy, and the application can start to look unappealing when there isn't time spent designing the UI and gathering feedback, I will be looking at designing techniques such as strategically using colour schemes, keeping the UI purposeful and keeping the UI simple. I set up a monthly bill pay with Adobe XD. I purchased this application as it offers more features when designing UI's. Adobe XD is a vector-based user experience design tool, which in my case, is being used to for a mobile app. It is an excellent tool for user experience and interaction designers.

Creating the database

My plan is to create and design the database using 'SQL Server Management Studio (SSMS)'. SSMS is a software application that is renowned for allowing its users to be able to configure, manage and administer all components within Microsoft SQL Server. Here I will need to figure out what tables are needed and how I plan on dividing the information into these tables, this stage involves classifying data and identifying relationships, this will all be done with the aid of the ERD I created. When dealing with a financial application, storing user information will require a secure SQL database. I will look at obfuscating SQL data as well as potentially encrypting key information, whilst also ensuring all of Microsoft SQL Servers best practices are being followed.

Providing High Availability

Ideally the high availability method I would like to implement is failover clustering. Doing so, I will need 3 nodes. I plan on using Oracle VM VirtualBox to set up 2 nodes and the domain controller. The tasks will consist of configuring these nodes, installing SQL server and windows server, setting up the nodes for failover and configuring SQL server manager.

5.0 Technical Details

Developing the mobile application

Android Studio, being the official IDE for android development, its source code is in Java or Kotlin. For developing my application, I intend on using Java. Java is very suitable for my project; it is a highly regarded language for banking applications where security is amongst one of the main concerns. It is recommended that when developing a mobile application, security is a key aspect you should consider in order to say you have in fact built a good application. Hackers should never be underestimated that is why an approach I plan on taking when developing this application is that I will focus on improving my knowledge of providing better security services. Java is compatible with Stripe; I intend on using Stripe to assist me with processing transactions. Stripe is a company that primarily offers payment processing software and application programming interfaces for mobile applications. Stripe will consist of code for both, client and server side in order to process transactions.

Creating the database

There are a couple of key steps I must tick off when it comes to creating the SQL database. Going through these steps is the approach I plan on taking. When it comes to database design, I need to outline my business requirements, identify what tables are needed as well as identifying relationships. As mentioned before, I intend on creating an ERD to answer any database design and relationship issues I have. The main business requirements of my fintech application are that it needs to process transactions whilst providing users with the core features mentioned earlier on in

this report, for my business requirements to be fulfilled my application will also seek to provide high availability whilst ensuring customers feel comfortable and confident that their information is in safe hands. In terms of data inside my database I need to have an idea of how much data I intend on storing in certain tables as well as what populates the tables. A key business requirement I have mentioned is that I need my customers to feel like their data is in safe hands, this can be done by securing the database in the hopes of preventing any data leaks. Only I will have full access to the SQL database, I will also be the only person who is able to see user data however, user sensitive data such as passwords will be obfuscated. When addressing database security, we will need to find a perfect balance. Too much or too little security can result in problems. To perfectly keep track of any tables or datatypes created, I plan on creating an entity relationship diagram. This will help describe interrelated things of interest, essentially it will help specify relationships that can exist between entities.

Providing High Availability

A failover cluster is a group of independent computers that work together to increase the availability and scalability of clustered roles. If one or more cluster nodes, which is a clustered server, were to fail, the services that were hosted on that node will be automatically transferred to another available node in a process known as failover. Windows Server Failover Clustering provides high availability and disaster recovery scenarios of hosted server application such as Microsoft SQL Server. My environment will consist of 3 nodes, 1 which is the domain controller. This environment will be hosted in Oracle Virtual box which will assist me in creating the 3 virtual machines as well as installing windows server and SQL server.

6.0 Special Resources Required

UI Design – Adobe XD

I set up a monthly bill pay with Adobe XD. I purchased this application as it offers more features when designing UI's. Adobe XD is a vector-based user experience design tool, which in my case, is being used to for a mobile app. It is an excellent tool for user experience and interaction designers.

High Availability – Oracle VirtualBox

I will be using a Virtual Machine technology; in this instance I plan on using Oracle VirtualBox. The reason being it can be used to provide great flexibility in deploying servers and I will be able to practice a demonstration of high availability. Briefly, when a virtual machine is running in this highly available machine cluster, any failure of the physical hardware will not affect the running virtual machine as it will be automatically transferred to the other node I have configured in the cluster. Installation of windows server, this will be installed on each server that would be joined to the cluster.

7.0 Project Plan

To help me keep on track with the progress I plan on making over the next few months I designed a detailed Gantt chart in Microsoft excel. As it is a detailed Gantt chart, I couldn't manage to fit the entire thing in the one snapshot so attached below are multiple snapshots consisting of the objectives I plan on completing and in the relative timeframe.

As shown above, the first task I need to tackle is designing my mobile applications user interface. This task will be done using adobe XD. Not only will I need time to design the UI, but I will also need time to plan the page layout and design, I feel as if 8 days is fitting for this task. Next is designing the Entity relationship diagram. Thanks to the user interface for the mobile application this task shouldn't take long as I will have a good idea of what tables and values are necessary. As I mentioned above, I plan on designing the UI in adobe XD. This means I will need to export it from adobe XD and import it into android studio. Adobe XD will show an overall mock-up of what I want my application to look like however I will need to create buttons, text views etc in android studio. Following the ERD diagram, I will start to design the database, whilst researching methods suitable for my applications needs. I will assign relationships, possibly create indexes where needed to improve performance etc. App functionality is a broad task in which I plan on breaking down into microtasks over the next couple of days. There's a lot of work that needs to be done and things are bound to get intricate, I feel as if 28 days is a fitting estimate.

Hosting the mobile application and having the SQL database connected to a server will take some time to get set up. Providing high availability will consist of configuring the virtual machines, connecting the two nodes to the domain controller, and implementing the failover feature. Database security is a business requirement of mobile application and needs to be revised to ensure it is configured correctly. I have revising code down just to give myself some free time in case I didn't manage to get any of the features fully working. Report work will be done throughout and towards the end of my project.

Trello

Trello is a web-based list making application that I am using to organize my project into boards. Trello is useful as at first glance it tells me what is being worked on, what has been completed and what is due. I am using Trello as a digital whiteboard covered in sticky notes, each note is a task that I am working on, have previously worked on or have yet to work on. Trello will complement the Gantt chart I created in the sense that on Trello I can click into each note and get a full description of each task, while the Gantt chart gives me a rough estimate of how much time I have to complete each task.

8.0 Testing

Thorough testing is crucial to success of a mobile application. If it turns out that my mobile application is not working properly, it is very likely that most people will refrain from buying or using it.

Before moving my application into a production environment, I would like to perform user acceptance testing. This method of testing is performed by the end user to somewhat verify the mobile application I intend on developing before moving it into a production environment. User acceptance testing is done in the final phase of testing. It will be carried out after function, integration and system testing is done. I plan on steering away from saturated testing, I don't want to have to every value for an input. I would like to test from the user's perspective.

Testing to find defects or bugs can be time consuming, repetitive, and subject to human error. From doing some researching online, I have found that Quality Assurance teams use automated testing to run these detailed and repetitive tests automatically. There are test tools out there that teams test faster whilst improving test accuracy and testing a substantial amount of code.

To ensure that I get as much as possible from testing my mobile application I have put together a checklist:

- **Underline the test cases I wish to Automate.**
 - o It would be extremely inefficient to automate all testing, therefore we should determine what test cases we plan on automating.
- **Find the most fitting testing tool.**
 - o This step is essential for test automation. It's required that I find a testing tool that best suits the overall requirements I underline.
- **Manage the different tests based on your skill level.**
 - o It is important to identify my level of experience and skills when it comes to testing, not to overshoot it by setting absurd expectations.
- **Create solid, high standard test data.**
 - o Using external data makes your automated tests reusable and easier to maintain.
- **Changes to UI may affect test results.**
 - o Providing unique names for my configurations will makes my automated tests resistant to UI changes and ensure that my automated tests will work without having to make changes to the test itself.

9.0 References

[prenewswire][2019][Global Fintech Market Value is Expected to Reach \$309.98 Billion at a CAGR of 24.8% Through 2022]

2.2. Ethics Approval Application (only if required)

3.5 Reflective Journals

Supervision & Reflection Template
--

Student Name	Alan Mellowes
Student Number	18467362
Course	Bachelor of Science in Computing

Month: October-21

What?

Primarily this month my focus has been finding my strengths and trying to generate a project idea that demonstrates them well. For guidance, I reached out to Keith Maycock who gave me some great advice and helped me be more specific with what I would like my project to achieve. The main things I have been researching have been: How compatible the various technologies I plan on using are, what programming languages I could possibly use and coming up with a plan to stay on track, setting checkpoints on what I wish to achieve by a particular timeframe. I also uploaded my project pitch where I talked about what my project entails and why it would be used in today's world. A lot of research went into what high availability method would be most suitable for my application.

So What?

I must decide whether I am taking the approach of developing an app in android studio or a web application. Both would work but I need to overcome some challenges related to compatibility and how much time I can afford to spend learning how to use new technologies. I have experiences with SQL databases, developing web application and android applications however, it's all about finding a solution that demonstrates them to a good standard as well as compatibly working well. I feel like I'm on the right track and have succeeded in underlining a project idea as well as learning about application development, database design and high availability methods. I have read up on these 3 main focuses however it seems that the main challenge at the moment is clearly outlining what the best method is to combine the 3.

Now What?

Instead of researching my strong points separately I need to research a way to outline a feasible method to demonstrate them in an all-in-one compatible application. Finding a suitable way of dealing with transactions will be another challenge I hope to overcome and learn a lot from in the coming weeks. It has been tough finding a balance between work and college, as well as getting some time to myself but I feel like October has been successful so far. Keeping up the good habits that I researched prior to starting my final year should ensure I achieve my full potential and make some great progress this year.

Student Signature



Supervision & Reflection Template

Student Name	Alan Mellowes
Student Number	18467362
Course	Bachelor of Science in Computing

Month: November-21

What?

In November I received feedback regarding my soft copy submission for my project proposal. The main feedback I got was that I should consider narrowing the scope of my project which I completely agree with. I spent some time planning out the scope of my project and came up with a solution. My project primarily focuses on developing a fintech application, whilst features such as high availability and database security will simply compliment the fintech application instead of being included in the main scope. My project supervisor, Keith Maycock, organised a meeting which was very helpful. I found the call with Keith reassuring as he answered any questions I had as well as letting me know that I was on the right track. My main concern that I brought up in the meeting was that I am struggling with time management.

So What?

I had to come up with a solution or something to assist my time management concerns. I filled out a Gantt chart in details as well as creating a diagram using Trello. The Gantt chart will act as a reference as to where I should be, in terms of what I'm working on, to still be on track for finishing my application on time whilst also sparing myself some time for any unplanned issues. Since the last reflective journal, I have decided that I am going to build a mobile application instead of a web application. I confirmed that the features I intend on using are fully compatible with each other. My next task is to research the possibility of dealing with transactions within my mobile application.

Now What?

I am planning on not only designing the UI for my mobile application, but I would also like to start coding. Before I begin coding, I will find out if it is possible to process transactions. I have project proposal submission coming up in the first week of December which I plan on having ready. I will work alongside the Gantt chart I created, hoping to stay working at the right pace. I will aim to have some features ready for the mid-point presentation that is taking place at the end of December. After I design my UI I will decide on what pages of my mobile application I will have ready for the presentation.

Student Signature



Supervision & Reflection Template

Student Name	Alan Mellowes
Student Number	18467362
Course	Bachelor of Science in Computing

Month: December-21

What?

For this month I generally spent time preparing for my mid-point presentation and upload which consisted of a project proposal, technical document and partial functionality within my application working. My aim for this checkpoint was to complete navigability within my application by this I made it so the user was able to access all of my applications different pages. In my presentation I demonstrated that the user was in fact able to login or register if they didn't have an account yet, the registration process stored the newly created account in a firebase database. On top of this I added some functionality to the investment management page, I used coinmarketcap's crypto API to pull all current prices of the top 100 cryptocurrencies.

So What?

Creating a login and registration page was a success however for my demonstration the accounts were stored within a firebase database, for the final upload in May I would like for all the user account and any other information in my mobile application to be stored in my own SQL database. Getting an idea of the current state of the crypto market was a success by using an API provided by coinmarketcap. Some challenges with making the investment management page more appealing will still have to be dealt with. This month really put a lot into perspective for me, I realised that I may need to rethink my current time management plan as there is still a lot of work to be done.

Now What?

In the next month I hope to configure my SQL database and set it up with my application as well as starting to tackle payments. Some more research will have to go into this feature however I have already done a fair amount as it stands. I hope to make a lot of progress in January as it is the start of semester two and I am hoping I will be able to allocate more time to my project. I would also like to spend some more time going over the technical document in which I submitted for the mid-point upload, I feel as if I can do some cleaning up and potentially add some new information.

Student Signature



Supervision & Reflection Template

Student Name	Alan Mellowes
Student Number	18467362
Course	Bachelor of Science in Computing

Month: January-22

What?

This month I continued from where I left off after the mid-point presentation. I was happy with the amount of content I provided for the presentation. I mentioned in the previous report that I would like to move my information stored in my firebase database into a SQL database. I decided to not do this step as it is unnecessary. I have not been able to acquire an environment where I can configure failover clustering. Windows server standard edition is too expensive to buy so I have come up with the idea of walking through the implementation in my report.

So What?

This was the ideal scenario to be in as I am confident in my ability to configure a failover cluster, it would have been a nice addition to my application.

Now What?

I still think failover clustering should be a core factor revolving my application, so I have decided to start running through a details process on how to implement it.

Student Signature



ASupervision & Reflection Template

Student Name	Alan Mellowes
Student Number	X18467362
Course	Bachelor of Science in Computing

Month: February-22**What?**

I met up with my project supervisor this month to discuss my midpoint presentation results and how I feel I am progressing. I have also started coding a feature to process transactions. To aid me with the transactions I have found documentation on 3 useful APIs. The APIs I intend on using are

- Google Pay API
- Stripe API
- PayPal API

Google Pay and PayPal's APIs will aid my core functionality of being able to process payments. Whilst the Stripe API will be used to deal with an in-app purchase.

So What?

I am happy with how much work I got done on my report. I have successfully implemented Google Pay and PayPals API however, I am still working on implementing Stripe's API. The only issue with the Google API is that I cannot test it as I require an Indian Google Pay account.

Now What?

This month I will have all transactions working. I also hope to fully run through the implementation of windows server failover clustering. It would be a huge plus to continue working on my report as well as consistently meeting up with my project supervisor on a weekly basis.

Student Signature

Month: March-2022

What? I have successfully implemented all payment APIs. I have also finished running through the implementation for how to configure a windows server failover cluster. This month has consisted of setting up some functionality for adding money to a user's vault.

So What?

This meant that huge progress was made for my application. One of more core functionalities have been fully implemented. Stripe transactions proved to be difficult and time consuming however I feel as they have made a nice addition to my app.

Now What?

I need to work on the savings vault activity. This means coding CRUD functionality to allow the user to add and remove items from their vault. As an addition to the savings vault I would like to allow users to view their daily, weekly, and monthly spendings whilst also being able to gather daily analytics on what they have spent their money on.

Student Signature



Supervision & Reflection Template

Student Name	Alan Mellowes
Student Number	X18467362
Course	Bachelor of Science in Computing

Month: April-22

What?

I finished the core functionality of my application. Successfully implemented transactions, savings monitoring and investment monitoring. The daily analytics for what users have spent their money on proved to be time consuming. I ended up successfully displaying the user's spendings on a pie chart.

So What?

There is still some work I need to do with cleaning up my application and report. I will spend my last 2 weeks proofreading my report, fixing any errors. Some more testing needs to be done with my application. The daily analytic activity proved to be a challenge however it still ended up being a success later in the month along with allowing the user to perform CRUD functionality in their savings vault.

Now What?

At this stage, it is all about time management and being as efficient as possible. Over the next two weeks I am allocating as much time as possible to doing some final checks. I need to read over the android documentation for using the Espresso testing framework. I would like to add 2 additional activities, one is for showing users where the nearest ATM is whilst the other will be used for motivating my users. The second activity will help motivate users is a step detector that will display the user's total steps since they opened the application and since they first downloaded it. The results for the feature will be displayed on a progress bar.

Student Signature



Supervision & Reflection Template

Student Name	Alan Mellowes
Student Number	X18467362
Course	Bachelor of Science in Computing

Month: May-22

What?

I successfully ran traces using espressos testing framework. I executed multiple tests for activities such as:

- Login
- Register
- Search for cryptocurrency
- Add money to vault
- Add money to daily spendings

So What?

Learning how to use Espressos testing framework proved to be very interesting and quite successful.

Now What?

For the next two weeks I will proofread my report, and prepare my files that I will be uploading. I will work towards recording a demonstration of my application as well as creating PowerPoint slides.

Student Signature

