The Investigation and Development of a Personal Finance Tool to Improve Financial Capability

Kiran Sanganee
Department of Computer Science
University of Warwick

Supervised by Sara Kalvala
Year of Study: 3rd

7 April 2023

WARWICK
THE UNIVERSITY OF WARWICK

Abstract

Your abstract goes here. This should be about 2-3 paragraphs summarising the motivation for your project and the main outcomes (software, results, etc.) of your project.

Contents

Chapter 1

Introduction

The term personal finance tool encompasses a wide range of software, including budgeting tools, investment management software and credit score calculators. This project aims to investigate strategies which help build a user's financial capability and confidence; followed by the implementation of a web application to support these strategies. In particular, the research focused on strategies to help individuals to manage their, often several, bank accounts and expenses in one place.

Being financially capable is defined by the Financial Educators Council as having the "skills and knowledge of financial matters to confidently take effective action that best fulfills an individual's personal, family and global community goals" [9]. The website therefore aims to give user's the tools to view their financial circumstances, as part of the knowledge, and with it comes the confidence to make informed beneficial decisions. For the rest of this document, financial capability and financial literacy will be used interchangeably.

Part of the motivation for this kind of project comes from the recent open banking technology movement [10]. This is where thousands of major banks have opened a set of endpoints to allow third party applications to securely access their customers' financial data with authorisation. It allows developers to build personalised financial applications and services that are tailored to the needs of the user, based on this data; and is most appropriate for this project. Further detail and other motivations are discussed in the background chapter below (2).

Chapter 2

Background

The purpose of this section is to provide background information into the problem area, as well as introduce and explain concepts that will be used throughout the rest of the document.

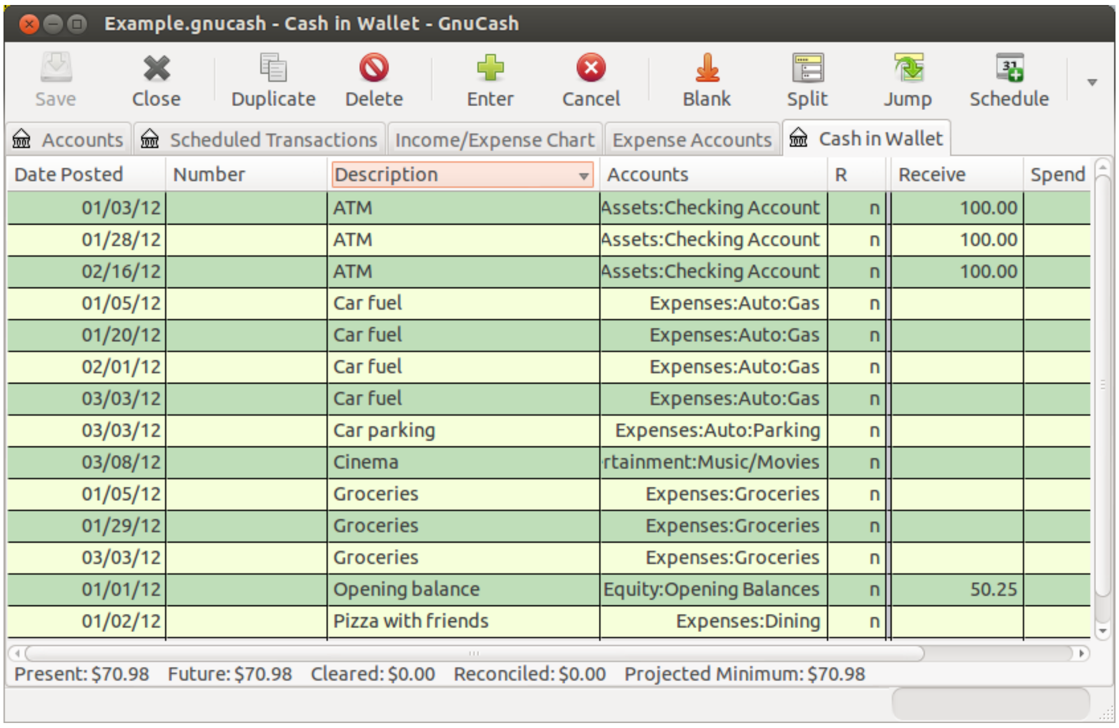
2.1 Current services

Understanding the current available applications and their limitations is important. It helps to narrow down the problem area, and identifies the requirements for the new system.

It is fair to say that there are a wide range of personal finance applications available, however, they are not all suitable for the same use case. For the purpose of this project, the focus is on those that help provide a user with an overview of their finances, as these are most applicable in improving a user's financial literacy. These existing tools can be divided into three main categories. Firstly, there are the applications that do not utilise open banking, and as such require manually importing the data. Secondly, there are the mobile banking applications that do use open banking, but instead often focus on single accounts, so not a big overview. Finally, there are applications which you often have to pay for, or a filled with advertisements and furthermore, lack analytics.

2.1.1 Manual Importing Applications

The first category of applications are those that do not use open banking. Often, these tools are older and lack updates which is why they haven't been improved to utilise the end points offered in open banking. The most prominent example of this GNU Cash [2]. This is a free and open source application that allows users to track their finances. It is a desktop application and very powerful tool, however, it is not very user friendly and requires a lot of experience using it before it is effectively used. Switch to Linux mentions that it is a "great FOSS tool [...], but it can be complicated to setup" [18], as part of their tutorial on how to use it. The fact that there are several tutorials and little documentation demonstrates that the UI is difficult to use, which is made worse from it looking very outdated and complex (see figure 2.1). The main weakness that software like this has is that it requires the user to manually import their data. Most users do not have all their accounts and transactions readily available in a structured format, and this even worse for when the user would want it to update live with their recent transactions. Not many aspects of these applications are useful for improving financial literacy, so few will be used in the final web application.



Date Posted	Number	Description	Accounts	R	Receive	Spend
01/03/12		ATM	Assets:Checking Account	n	100.00	
01/28/12		ATM	Assets:Checking Account	n	100.00	
02/16/12		ATM	Assets:Checking Account	n	100.00	
01/05/12		Car fuel	Expenses:Auto:Gas	n		
01/20/12		Car fuel	Expenses:Auto:Gas	n		
02/01/12		Car fuel	Expenses:Auto:Gas	n		
03/03/12		Car fuel	Expenses:Auto:Gas	n		
03/03/12		Car parking	Expenses:Auto:Parking	n		
03/08/12		Cinema	rtainment:Music/Movies	n		
01/05/12		Groceries	Expenses:Groceries	n		
01/29/12		Groceries	Expenses:Groceries	n		
03/03/12		Groceries	Expenses:Groceries	n		
01/01/12		Opening balance	Equity:Opening Balances	n	50.25	
01/02/12		Pizza with friends	Expenses:Dining	n		

Present: $70.98 Future: $70.98 Cleared: $0.00 Reconciled: $0.00 Projected Minimum: $70.98

Figure 2.1: Example GNU Cash UI [4]

2.1.2 Mobile Banking Applications

These applications are the default applications that come with each bank. Almost always, if the bank offers a website for online banking, they also offer a mobile application that perform the same functions. These include the major banks such as HSBC, NatWest, Lloyds etc., but also includes the online-only banks such as Monzo and Revolut. Sometimes, these applications do not use open banking as instead just display information about the accounts with that specific company, so do not need to use the services of other banks. There are some that do connect with other banks, but they often don't incorporate the information into the analytics, and instead just display the balances.

Taking Revolut [16] as a case study, we can find its strengths to incorporate, as well as its weaknesses to avoid in the web application. Firstly to even use the Revolut's app functionality, a user must open a bank account with the service and prove their identity. Although this does provide the user with confidence that their information is secure and personalised, it also means it is more difficult to use, slower to gain access to the information and overall just has a catch. It also means that the primary purpose of the app is not to provide a user with an overview of their finances, but instead to provide a banking service, so will not be aimed at improving financial capability. There are some features of Revolut which are worth acknowledging, such as the UI and ability to track expenditure. In having a slick and intuitive user interface, it enables users to understand all aspects of their finances and helps build confidence. The expenditure tracking is particularly useful, it allows users to customise a budget for a set time period and shows what and when they have spent money in during this period; overall aiming to help them stay within the budget. Revolut is a service that does utilise open banking as allows to you connect the app to other bank accounts, however it doesn't incorporate these into the budgets and only really includes them in the net worth section. This is a good example of a service that does use open banking, but doesn't utilise it to its full potential.

2.1.3 Paid Applications

The final category of applications is really just the 'other' section, however the majority of these do incur costs to use. These applications are often more powerful than the free alternatives, yet are often filled with advertisements and are not as user friendly. A good example of this is Quicken which won the awards for the best budgeting app in 2020 and 2021 [6], but costs up to £10/month. It utilises open banking effectively to work with many different bank accounts, yet it does not enable quick-toggling of bank accounts to incorporate/ignore during the analysis and overview. This is a feature which would be particularly useful in giving analysis and a better overview as a user would be able to segment e.g. all the savings accounts. Despite having some effective budgeting features, it lacks the ability to perform budgeting prediction from patterns in expenditure. This feature also would be useful in improving financial capability because it will help the user plan for future expenses and identify areas where they can save money. Overall, these paid applications have some aspects which would be useful in the web application, but they also are missing some basic ones which would be more focused on improving financial literacy.

2.2 Open Banking

Open banking is defined as "APIs [that] enables third-party developers to build applications and services around the financial institution", in this paper [14]. The open banking movement, is therefore the recent pressure on banks to open up their data to third-party developers. They do this by creating a set of endpoints which developers can query, and will respond with accurate and live data. For example, a user would first login to their online bank via a popup, this would then return an authentication token which the website can use to query e.g. to get their recent transactions; the information is then returned in a secure response. Each bank has their own set of endpoints such that the authentication process and available information differs across them all. This is where a lot of problems arise as reading the documentation is not straightforward and time consuming, however it is necessary to enable the web application to support all the major banks. This lack of standardisation is where Plaid [11] come in.

2.2.1 Plaid

Plaid is a platform that, effectively, wraps all the endpoints provided by each individual bank and provides a single standardised set of REST API endpoints as URLs. This means that the web application only needs to query Plaid, and in turn will be able to support all the banks that Plaid provide access to - over twelve-thousand institutions [12].

To use Plaid's endpoints, a strict authentication flow must be followed as it is handling very private data. Plaid has three different types of tokens as part of this flow. The first is the link token; Link is the name for the widget that is used to authenticate the user with their bank but to do so requires a link token. They can simply be requested from Plaid's API and are valid for 4 hours, but are not tied to any user and are not treated as passwords. Secondly, there are public tokens; these are what the link widget returns to the web application after the user has authenticated with their bank and so are unique to that user. They also are not treated as passwords as are valid for only 30 minutes and cannot be used to access a user's private information directly. They must be exchanged for an access token, which is the third type of token. This exchange is done via a Plaid endpoint but must be done via the web application's server, rather than client because the response access token must be treated like a password and be kept extremely securely. If it was done on the client, it risks being exposed and anyone with this token can access that user's information. Embedded within the access tokens can be a set of bank accounts, and a user may have several access tokens tied to them.

The flow is therefore: the client requests a link token on behalf of the user, the response link token is then passed to the link widget where the user signs in with their bank; the link widget returns a public token to the client which is passed to the application's backend; the backend exchanges the public token for an access token and then stores this in the database in such a way that only that user can access it. Whenever the client wants to gain access to that user's transactions, it queries the server acting as a proxy. The server identifies which user is asking for the information and attaches their access token to a new request to Plaid. The Plaid response is then forwarded back to the client and the client can display the information to the user. This flow is shown in Figure 2.2.

Further aspects of Plaid relevant to this project include the different modes of operation. When the web application makes queries to Plaid's endpoints, one of three modes can be specified and each produce different results. The first is sandbox mode, this is the default that everyone has access to. All the endpoints and request formats remain the same, but the responses are fake data generated by Plaid themselves. This is useful for testing the web application without having to connect to a bank so do not have to worry about leaking any private information, as well as getting fast and reliable responses. The second is development mode where the responses are real data. The full authentication flow must be followed and a user will login to their bank. To gain access to this mode, it must be requested by Plaid and they review the application to check

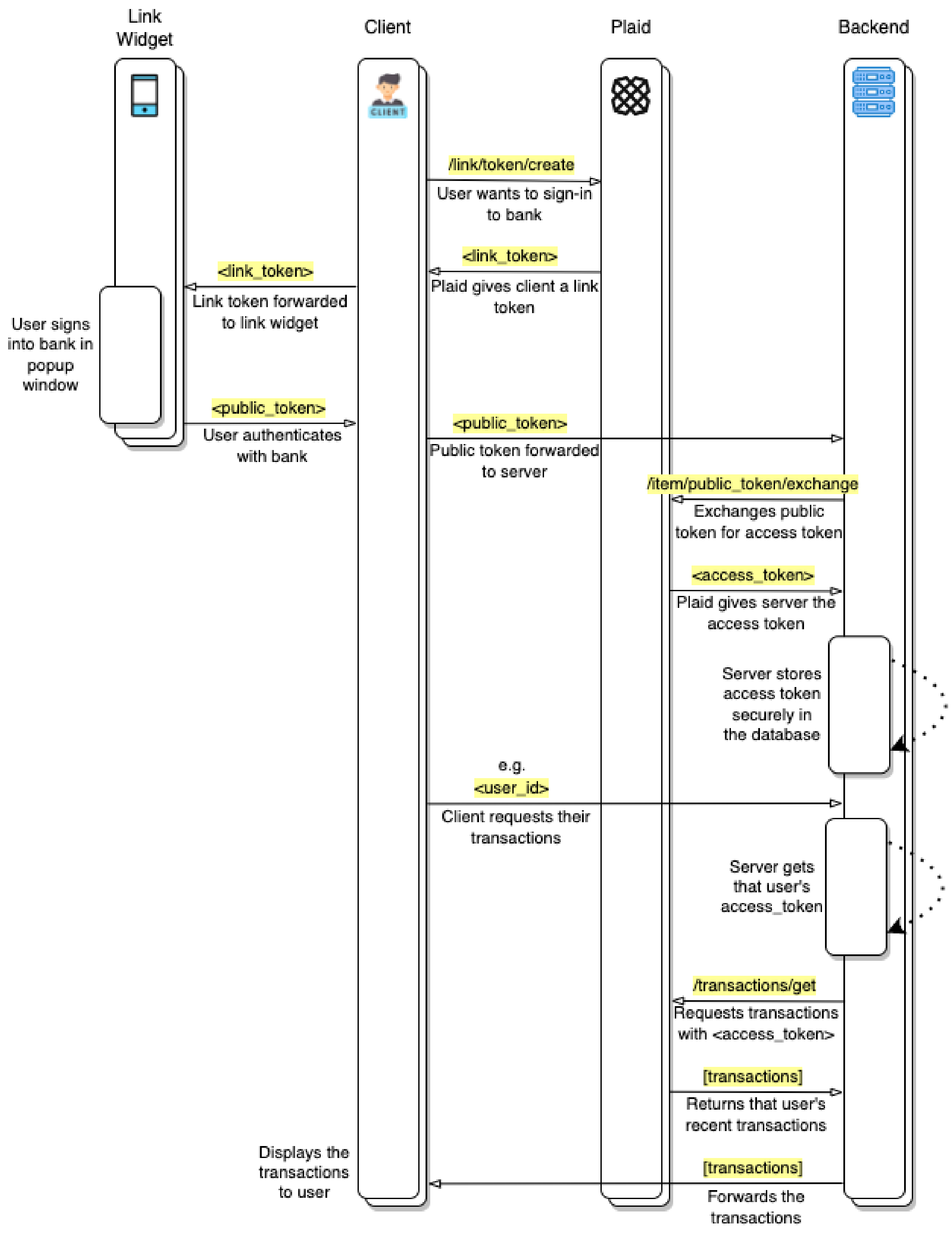

Link Widget
Client
Plaid
Backend
/link/token/create
User wants to sign-in to bank
<link_token>
Plaid gives client a link token
<link_token>
Link token forwarded to link widget
User signs into bank in popup window
<public_token>
User authenticates with bank
<public_token>
Public token forwarded to server
/item/public_token/exchange
Exchanges public token for access token
<access_token>
Plaid gives server the access token
Server stores access token securely in the database
e.g. <user_id>
Client requests their transactions
Server gets that user's access_token
/transactions/get
Requests transactions with <access_token>
[transactions]
Returns that user's recent transactions
[transactions]
Forwards the transactions
Displays the transactions to user

Figure 2.2: Plaid Authentication Flow

that it is secure and will not leak any information. Once given access, there is a limit of one-hundred user accounts connected to actual banks, so this had to be managed carefully. The final mode is production mode, this is the same as development mode but with no limits. To gain access to this, further vetting is required and the queries start to cost. For this project, only sandbox mode was initially used, then following the completion of the prototype, development mode was requested and given for further testing and demonstrations.

2.3 Further Motivation

As previously mentioned, the main motivation for this project came from the recent open banking movement, so using modern technology to help push the field. In addition to this, the limited current services available was also a factor as it was identified that there was a need for an application like this. Finally, the timing for this project is very appropriate as the UK government announced that there is a financial crisis currently going on, which they named the 'Cost of Living Crisis' [5]. Furthermore, in the Money and Pensions Service's recent report, they advised that people get a budget planner to help cope with the current times. This then helped concrete an objective for the web application to include budgeting features, as would also help improve financial capability.

Chapter 3

Design

This chapter will aim to describe the design of the web application and how it supports the overall goal of the project.

3.1 Technologies

As mentioned in the background section 2, the web application will be utilising Plaid for the open banking aspect. This requires a frontend and a backend to be developed to follow the authentication flow for the ensured security. The majority of the personal finance tool will be written in JavaScript and will be using the Next.js framework.

3.1.1 Next.js

Next.js is a full-stack React framework by vercel [19]. It incorporates the React library for the frontend, as well as supporting API routes for the backend, meaning it is an ideal framework for this project. Other benefits of using Next.js include the ability to perform server side rendering, which will improve the initial loading time as well as making the component development much less complicated. In addition, Next.js has in built automatic optimisations to improve building and serving times. The queries to get, for example, a user's transactions go through 3 stages. The first is the initial query from client to Next.js API route; then this accesses the database and then queries Plaid's endpoints; finally Plaid makes the query to the bank and response is propagates back. This means that any optimisations to improve the speed and help avoid making the loading times of data feel slow are valuable and what Next.js provides by default. Next.js is a popular web application framework with over one-hundred-thousand stars on GitHub [20], therefore has a large community and is well documented such that it is easy to find solutions to any common problems that may arise. Finally, Next.js is a framework that focuses on the ease of development to maximise productivity, and so means the whole experience will be more enjoyable and good software engineering practices can be followed.

3.1.2 TailwindCSS

As part of the Next.js environment, the website serves jsx components that can be styled as normal HTML elements with CSS. TailwindCSS is a framework that contains a large amount of utility CSS classes. The application incorporates these classes to completely customise the way the website looks. Another consideration for helping to build the UI was to use a component library. This may have cut down on development time, but often doesn't look as appealing and often look quite generic, as have limited ability to customise. TailwindCSS is more flexible and enabled the UI to be built to the exact specification of the design and the developer's vision. Similar to Next.js, TailwindCSS is also quite popular, so has a large amount of resources and cheat sheets to aid development. Often Next.js and TailwindCSS are used in combination, such as in the t3 web stack, as they synergise well together [17].

The UI is a big focus for this project, as is one of the limitations in the current systems that perform similar tasks; the background research found that their interfaces are often ugly and unintuitive. A lot of time was spent prototyping and designing the components before the functionality was actually implemented into them. By having an appealing UI, the user will have more confidence in the application and therefore their finances, which is a key part of financial capability. Furthermore, provided that the user's data is displayed in an informative and easy to understand way, they are more likely to absorb the information and therefore make better financial decisions.

3.1.3 PocketBase

Part of Plaid's authentication flow requires the access tokens are associated with only a single particular user, however that user may have several access tokens. To support this not only is a database needed for basic user authentication to signup and login, but also to store all the access tokens. PocketBase was determined to be most appropriate for these use cases due to its minimalism and fast setup.

PocketBase is an "open source backend consisting of embedded database (SQLite) with realtime subscriptions, built-in auth management, convenient dashboard UI and simple REST-ish API" [13]. It is run as a single executable file and the Next.js API routes can connect to it locally. In addition, the built-in authentication management is for the user to be able to login and persist their session across refreshes, this makes the user experience of the application much more seamless. Furthermore, means less time is spent on development for this basic feature. The dashboard UI is also useful for the developer to manage users and aid development as can help visualise the database structure as well as manage the tables inside. Finally, PocketBase has a JavaScript SDK that can be used to connect to the database from the API routes. This means less time is spent on ensuring the REST requests are all correct as instead is a simple function call.

3.1.4 Python and TensorFlow

Following the research into strategies which would help build financial capability, the incorporation of of budget prediction was proposed. The thought process and explanation as to why the project uses a recurrent neural network is explained later in this section, but it was decided that implementing it would be done with Python, and in particular, using the TensorFlow library. TensorFlow is a machine learning platform that is used to help build and optimise many machine learning models.

By using these technologies for the machine learning aspect of the project, it allows faster development. Python is a language that is often used for machine learning for a variety of reasons. Nazar Kvartalnyi [7] comments that some of theses reasons include the fact that it is simple, consistent and intuitive; that there are a variety of libraries and frameworks such as TensorFlow that support the process; and that there is a great community for giving support during development. Furthermore, the developer of the application already has experience with Python and TensorFlow so there would be little learning toll to implement the budget prediction.

Using Python does have some drawbacks such as the extra complexity and need for the application to communicate with the neural network. To overcome this, once the neural network had been trained, it was hosted with a python flask server such that it could be treated like an API. It would have a single route that would take the input data, and then response with the output of the neural network. This simple solution meant more time could be focused on ensuring the neural network is accurate and precise.

3.2 UI

As mentioned earlier, there was a big focus on making the user interface appealing and intuitive. By working with TailWindCSS, prototype designs could be made in software such as Figma, and then the developer could match the design exactly with appropriate CSS classes and JavaScript functionality. The design process that was opted for in the end, however was to first give the components their basic functionality; then perform adequate component-based styling; then go back to the functionality and modify if appropriate; and then finally finish of the styling with the whole page in mind and make sure it fits in well. This process was more suitable for the project as it meant that the websites functionality was not compromised by the styling, but also the on-the-fly styling is a added benefit from TailWindCSS classes as allows for quick viewing of designs and changes.

3.2.1 Authentication Pages

When a user first visits the site, they must first create an account. From research into the limitations of other services, it was noted that having a quick and seamless experience at this stage was important for the user. This is in contrast to applications like Monzo and Revolut, where the signup process is long as they are not signup to for the analytics, but also making a bank account. The focus hear was simplicity so the design for these pages were simple input boxes and a functionality button to login or create an account.

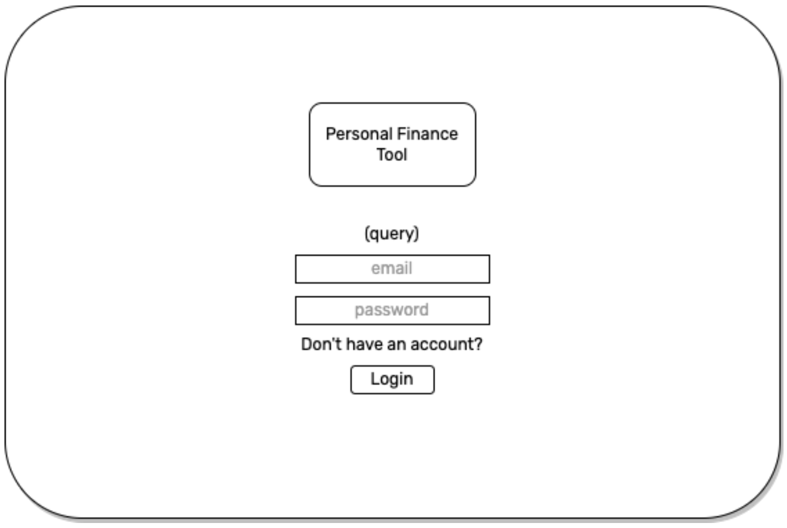


Figure 3.1: Login Page

Above is the basic design for these pages. Most of it is self explanatory, however where it says "(query)" is where any message or response can be displayed to the user. These include "you have successfully logged out", or "that user does not exist", or "incorrect password", and will be given the relevant colours . The "don't have an account?" and "login" text will be modified appropriately for each page that it is on - for example if instead the user is creating an account, it will say "already have an account?" and "signup" respectively.

This was the first and only page that was styled from a draw.io specification as the process of doing this was found quite slow and tedious. It didn't maximise the value of using TailwindCSS to quickly add the CSS classes and perform dynamic styling. For the rest of the pages, the more efficient process outlined in the introduction to this section was followed.

3.2.2 Dashboard

Once the minimum strategies for the web application had been determined - transactions, categories, budgets and investments. It was decided that this information was to be presented on a single central page, and the ability to switch between the strategies is done via a header bar. The was so quick access to all the information is possible, but also so that strategies not being viewed can be preloaded in the background to avoid long waits for the information to load, ultimately improving the user experience.

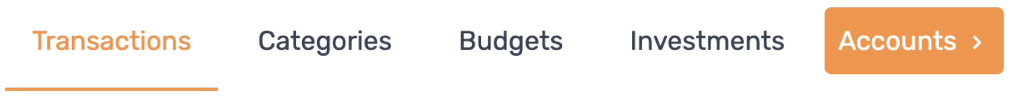


Figure 3.2: Dashboard Navigation Bar

The header bar shown above (3.2) was initial design for this dashboard (and minimally changed in the final product). By default, the transactions strategy is chosen. When hovering on the other strategies, a light grey bar underneath the strategy name appears; then being selected turns this bar orange and removes the bar from the old strategy. Underneath this bar is the information provided for that strategy, for example in the transactions strategy, below will be all the transactions.

A limitation in the current systems that was outlined earlier is the ability to include several accounts. Furthermore, of the few that do have this feature, none of them have a way to easily enable/disable the incorporation of these accounts in the analytics. Therefore, this aspect was key when thinking about the design of the header bar. On the right in a slightly different bar, there is an accounts dropdown. When clicking on this, all the bank accounts that the user as linked will be displayed hear, along with a toggle button to enable/disable the account. Only enabled accounts will be included in the analytics, and when switching between strategies, only the selected accounts will persist. An ideal use case for this feature is when a user has several accounts, but only wants to see their transactions on their savings accounts. They can disable all the other accounts to do so. Following this, they can quickly re-enable the other accounts when they want to see them all again by just clicking the toggle button.

3.3 Strategies

The only pages in the whole application are the two described above (authentication and dashboard). Each of the strategies will be implemented as separate components but will appear on the dashboard page. This means that each component ought to be roughly styled in the same way as to match the general theme.

3.3.1 Transactions

This is the default strategy that is shown on the dashboard because it contains the information that would be the most useful in helping the user can an overview of their finances. By viewing their recent transactions, they can see where money is being spent and where it is coming from, and then make adjustments accordingly. A lot of traditional banks to not allow quick access to this information, and viewing transactions for all the accounts that have been toggled on can be extremely useful in identifying expenses that are not necessary.

From Plaid, the endpoint returns a list of transactions in reverse chronological order, per account. Each transaction has account_id, amount, date, time, location, currency code, merchant name, category and more. The web application will group the transactions from all toggled accounts by their date, then display each transaction in reverse chronological order. The transaction will be displayed with the merchant name, category, amount and the bank logo of the account it is from. For each date their will be a separate label above all the transactions for that date. Finally a distinction between income and expenditure will be made by colouring the whole transaction background light-green for income and light-red for expenditure.

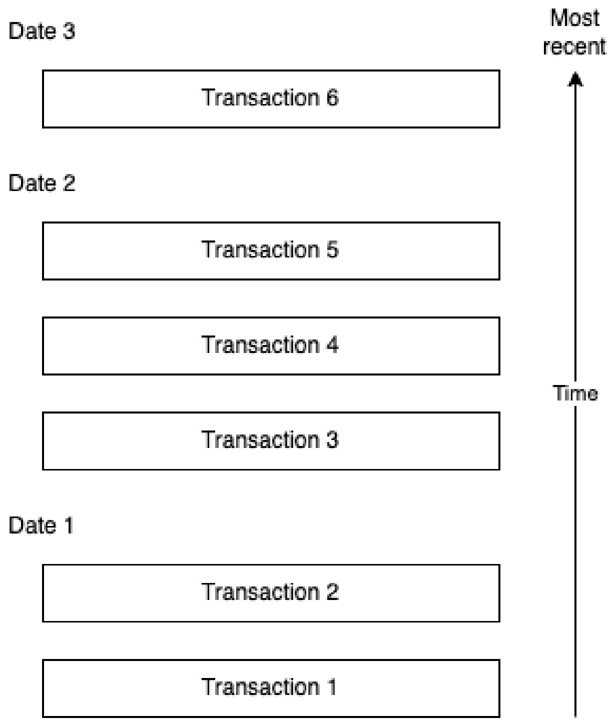


Figure 3.3: Reverse Chronological and Date Grouped of Transactions

3.3.2 Categories

The categories strategy would also be helpful at identifying unnecessary expenses. In this case, the application only takes the previous thirty days, and also only takes expenditure. It will list all the transactions in reverse chronological order again, but this time not grouped by anything. A pie chart will also be displayed with the values being the total amounts for each category. The user can then see which category they are spending the most in, and will be able to see the total amounts. In addition, they will be able to group each transaction by it's category and show only the transactions for that category by selecting it. All the transaction data will again be only for the toggled accounts.

The difference between this strategy and the transactions strategy is that in the transaction strategy, the emphasis will be on if the transactions were income or expenditure and to which back account; whereas in teh categories strategy, the emphasis will be on the category of the transaction and the total amount spent in that category. The input data will be the same, but it is a different way of presenting the data such that the user can get all the perspectives on their finances, and ultimately make better decisions.

3.3.3 Budgets

Two different methods of performing some analytics of budgets were considered for this strategy. The first method was giving the user a budget breakdown based on their income. Following some research, it was found that for the average person, utilising a 50:30:20 budget is extremely effective at helping people spend money on the necessities, still having some money for treats and also saving some money. The article [8] by N26, a well respected european bank, explains the strategy in more detail. In essence, it says to spend 50% of your income on needs, 30% of your income on wants, and 20% of your income into savings. Several other articles back up the effectiveness of this strategy and help give it credibility; [15] even talks about how this method carries over into the cost of living crisis.

The method does have some disadvantages however. The major one being that it is extremely difficult to narrow down which category each expense falls under. Often people will disagree what is considered essential, so is already subjective. A common example found from some of the articles that talk about it is a gym membership. In addition, altering your spending habit is quite difficult and often if your income is relatively low, it may not even be possible to save 20% of your income.

The design for this strategy would be to have an input element where the user can enter their income, however it would also help determine the income from previous transactions. The website would then breakdown their income into the 3 sections and give them values for what they can spend on each. Finally, for their previous thirty days of transactions (the usual time period between each pay day), the user will be able to drag and drop each transaction into the three categories to help them see how much they are currently spending. If they are spending too much on any, they can make adjustments accordingly.

The other method that was considered for a budgeting strategy was to perform budget prediction. This would involve viewing how much they have spent so far for the current month, and performing some pattern recognition to make predictions for the remaining days. They could then compare this to a budgeted amount they planned to spend that month and identify if they are on track. This would be useful for people who are trying to save money for a specific goal, such as a holiday or a new car. The user would be able to see if they are on track to reach their goal, and if not, they can make adjustments to their spending habits to reach it.

The method of performing the prediction can vary from manual pattern identification to machine learning, so at this stage it was more relevant to identify which strategy would be most effective as performing the original aim of the project. Overall, the budget prediction method was more appropriate because every use could utilise the information it provided, furthermore, it also provided analytics which would be extremely difficult for the user to manually, unlike the 50:30:20 method. Also in predicting future expenses, the user could make adjustments to what they spend money on sooner, before it becomes a problem so is more effective.

3.3.4 Investments

According to Finder.com, a reputable source for financial information, in Danny's article on investment statistics, he says that "Almost 1 in 5 Brits have owned stocks or shares" [1]. This is from a survery completed in the year 2023, so is recent, and is fact checked by other journalists. With over 13 million people in britain alone, a strategy to adhere to this demographic would be useful and popular.

A particularly useful tool, which continues the idea of getting an overview of the user's finances, would be a central portfolio manager where logged in users can view and track their investments. This would be the only strategy that doesn't utilise Plaid to access the data, so instead the user would have to add it themselves; however, this is not a problem as unlike the other strategies, once the asset had been purchased, they would only need to add it once. Following this, every time the user would come to check their investments, they could see the updated live price and view the amount of profit or loss they have made.

In order to perform this strategy, once the user has added their investments, they need to be stored. One option would be to store them in local storage within the browser. This would be the simplest to implement and as stored offline, would be somewhat secure, however it would have some disadvantages. To begin with, it would only be accessible from the same device and browser, so if the user wanted to access the tool from another device, they would have to add them again. In addition, if the user cleared their browser data, they would lose the investment entry so also have to manually add it again.

The other option would be to persist them using a database. As a PocketBase data store is already being used for the application, this wouldn't require too much extra work and complexity; furthermore, there are some major advantages. Unlike local storage, when using the database, the information is accessible and up to date from any device or browser, so the investment would only have to be added once. Also by linking the investments to only be accessed by the user who created them, they would be extremely secure and could be treated like the Plaid access_tokens.

In the end, the database option was chosen because synergises well with the current design of the system and has the same advantages as local storage. As well persisting the investments, the application would also have to get the current price of the investment to determine if is profitable. To do this, the application would have to make a request to a service that provides the current price of the investment. Financial modelling prep is an easy to use service that provides quick and live stock prices so is ideal for this task. Given the price the investment was bought for, and the current price, the application would be able to calculate the profit and for example could be displayed as green for positive, and red for negative. This means the user can identify which investments are profitable and should continue, as well as which ones are not and action should be taken.

3.4 Budget Prediction

One decided that budget prediction was the method for the budgeting strategy, the actual prediction method needed to be designed. For this there was 3 different considerations: a basic manual pattern identification method; utilising linear regression to find the relationship; and using a neural network to find the trends. For each of the method, the only accessible real training data was the author's past 3 years of transactions, so this factor has to also be considered when deciding which method to use.

3.4.1 Manual Pattern Identification

This method would be the simplest to implement and would involve manually identifying the patterns. Given the training data, set time periods would be suggested and then determining which one and by how much is the biggest of these. For example, given the three years transactions, split the data into the different seasons and identify in which season is the most spent. Using this knowledge, if a query is wanting a prediction in the most expensive season, then increase the normal prediction. This can be repeated on many different features, each of different time periods such as weeks, weekends, months, national holidays. Once many different factors have been analysed, the prediction can be made by combining all the factors together. This method would be very fast at making a prediction, but extremely complicated and inaccurate to initially implement. Furthermore, the predictions may not be very accurate as some features may not be factors, and others may not have been considered. Ultimately, this method would sacrifice accuracy for speed and complexity.

3.4.2 Linear Regression

Linear regression is a method for "modelling the relationship between a scalar response and one or more explanatory variables" [3]. To apply it to budget prediction, the output expenditure would be the scalar response, and the explanatory variables would include the budget set out by the user as well as the current date. The model would find the relationship based on the past 3 years of data, and then when a query comes in, it would quickly be able to use the learnt relationship to output an expected expenditure. The main advantage of this method is that it would not involve any manual pattern identification and the relationship would be computed. Despite this, it still wouldn't be that accurate as the relationship between expenditure and explanatory variables almost definitely will not be linear, which is the major downfall. One option to counter this would be to project the input data into a higher dimension. This would allow the model to find linear relationship in this projected space, but then a better non-linear relationship in the original space. This would be a good option, but would require a lot of computation to do so, may be considered over engineering for this problem, and would not even guarantee a better result.

3.4.3 Neural Network

The final option that was considered was to use a neural network. This would take the input data, manually identify the features and trends and then, with lots of training data, learn the patterns. Within choosing to do a neural network, there is still so much variation as each model can be very different. The neurons themselves can have different activation functions, but also the connections between them. It can be feed-forward with no loops or be a recurrent neural network with some feedback. Each layer can have various number of neurons and the number of layers can also vary. Acknowledging this, if the network was to be designed effectively, it could out perform the other two methods in terms of accuracy so would be most ideal for this problem. The drawbacks include the fact that it would involve a lot of testing and tuning to get the best results, and the fact that the input data is limited doesn't help.

3.4.4 Conclusion

In the end, out of the three options described above, the most applicable model for the problem of budget prediction was a neural network. This was because it didn't rely on manually finding the features to use, so would identify even the hidden and complex patterns, as well as, after being trained, generate quick predictions with high accuracy. The implementation of the budget prediction is described in much more detail in the implementation section 4, but a quick overall explanation is given below.

Deciding the architecture of the neural network would involve lots of trial and error to identify what is most appropriate. There is little research on the topic of predicting expenditure given past expenditure so not much even to help start. Through generalising the problem, the key idea which helped was to think of the problem as a time series problem. By modifying the input data from being a list of chronological transactions, to instead being the cumulative amount spent each day and resetting every month, the input data could be treated as a time series. This has much more research as a problem because time series prediction is a very important area for many industries, such as trading companies trying to predict the stock market. Applying lots of these ideas that have been applied to time series data found that using a recurrent neural network was the most effective.

3.5 Software Architecture

The figure above (3.4) shows the software architecture of the project and the interactions between each service. The four on the left have been developed as part of this project, with the two on the right pre-existing entities. The user interacts with the front end, which is the Next.js UI. Having this separation from the backend was required for the Plaid authentication. The Next.js API routes are how the UI communicates with Plaid and the database. An example API route would be "get_transactions", where the backend then returns the transactions received from Plaid, which itself returns the transactions from the bank itself. The access_tokens and other long-term information such as user's login

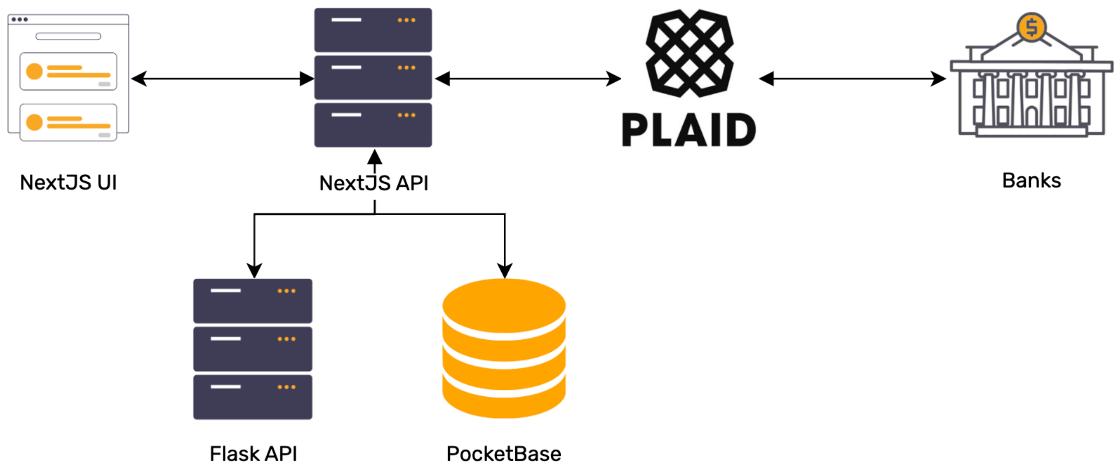


Figure 3.4: Software Architecture

details are stored in the PocketBase database. This can only be accessed by the backend, and for example the access_tokens can only be accessed by the user's who created in them in the first place for extra security. Finally, there is the Python Flask API which contains the budget prediction neural network. This API only has one route, which takes in the input data, propagates its through the neural network, and returns the output. It must be queried by the Next.js API routes as the input data requires the user's path month's transactions, therefore they must be requested from Plaid. This means when the Budget prediction page is accessed, the frontend queries Next.js, which itself queries the Python Flask API, after retrieving the transactions from Plaid. The Python Flask API then returns the prediction to the Next.js API, which then returns it to the frontend to be displayed.

3.6 Endpoints

Given the design for the strategies that the web application will exhibit, various Next.js API routes will be required. All of these routes will be prefixed with "/api" to be accessed by the client, and will be accessed using the HTTP POST method. This is because the requests often need to contain some contextual information for the server to decide what to do, for example, contain the user identifier to determine which user's transactions are being requested. The information passed between the client and server will be in JSON format as helps represent the data in a structured format as well as being easy to parse by both the client and server being written in JavaScript. In addition, python has a built in JSON library to work when communicating with the Python Flask API. Also each route will be written such that it can only perform its intended function or return an error. This is such that all the error checking can be done on the client side to improve security, as well as reduce the amount of code to be written and make the user experience better for graceful error handling.

Chapter 4

Implementation

4.1 Project Management

4.2 Strategies

4.2.1 Transactions

4.2.2 Categories

4.2.3 Budgets

4.2.4 Investments

4.3 Endpoints

4.4 Security

Chapter 5

Evaluation

Describe the approaches you have used to evaluate that the solution you have designed in Chapter 3 and executed in Chapter 4 actually solves the problem identified in Chapter 1.

While you can discuss unit testing etc. you have carried here a little bit, that is the minimum. You should present data here and discuss that. This might include *e.g.* performance data you have obtained from benchmarks, survey results, or application telemetry / analytics. Tables and graphs displaying this data are good.

Chapter 6

Conclusions

The project is a success. Summarise what you have done and accomplished.

6.1 Future work

Suggest what projects might follow up on this. The suggestions here should **not** be small improvements to what you have done, but more substantial work that can now be done thanks to the work you have done or research questions that have resulted from your work.

Bibliography

[1] Butler, Danny. Investment statistics: What percentage of the uk population invests in the stock market? *Finder.com*, March 2023. URL `https://www.finder.com/uk/investment-statistics`. Accessed: 7/04/2022.

[2] Clark, Robin. Gnucash, 1998. URL `https://www.gnucash.org/`. Accessed: 29/03/2022.

[3] Freedman, David Amiel. Linear regression. *Wikipedia*, March 2023. URL `https://en.wikipedia.org/wiki/Linear_regression`. Accessed: 5/04/2022.

[4] GNUCash, . Gnucash register page, April 2013. URL `https://www.facebook.com/GnuCash/`. Accessed: 30/03/2022.

[5] Hourston, Peter. Cost of living crisis. *Institue for Government*, Feb 2023. URL `https://www.instituteforgovernment.org.uk/explainer/cost-living-crisis`.

[6] Intuit, . Quicken, 2022. URL `https://www.quicken.com/`. Accessed: 30/03/2022.

[7] Kvartalnyi, Nazar. Why use python for machine learning? *inoxoft*, April 2022. URL `https://inoxoft.com/blog/why-use-python-for-machine-learning/`. Accessed: 3/04/2022.

[8] n26, . The 50/30/20 rule: how to budget your money more efficiently. *N26 Blog*, Aug 2022. URL `https://n26.com/en-eu/blog/50-30-20-rule`. Accessed: 22/11/2022.

[9] NFEC, . What is financial literacy?, 2022. URL https://www.financialeducatorscouncil.org/financial-capability-definition/. Accessed: 29/03/2022.

[10] OBIE, . What is open banking. *OpenBanking.org*, Sept 2020. URL https://www.openbanking.org.uk/what-is-open-banking/. Accessed: 29/03/2022.

[11] Plaid, . Plaid api documentation, 2022. URL https://plaid.com/docs/. Accessed: 30/03/2023.

[12] Plaid, . Plaid institutions, 2023. URL https://plaid.com/institutions/. Accessed: 30/03/2022.

[13] PocketBase, . Pocketbase documentation, 2022. URL https://pocketbase.io/docs. Accessed: 2/04/2022.

[14] Premchand, Anshu & Choudhry, Anurag. Open banking & apis for transformation in banking. In *2018 International Conference on Communication, Computing and Internet of Things (IC3IoT)*, pages 25–29, 2018. doi: 10.1109/IC3IoT.2018.8668107.

[15] Starling, Team. The 50/30/20 rule: Is it realistic in a cost of living crisis? *Starling Bank*, Sept 2022. URL https://www.starlingbank.com/blog/50-30-20-budgeting-rule-is-it-realistic-in-a-cost-of-living-crisis/. Accessed: 22/11/2022.

[16] Storonsky, Nikolay & Yatsenko, Vlad. Revolut, July 2015. URL https://www.revolut.com/. Accessed: 30/03/2022.

[17] (Theo), T3 Open Source. T3 stack. *GitHub*, 2022. URL https://create.t3.gg/. Accessed: 2/04/2022.

[18] to Linux, Switched. Introduction to gnucash - free accounting software, May 2019. URL https://www.youtube.com/watch?v=wBPg_AKdlG0. Accessed: 30/03/2022.

[19] Vercel, . Next.js, 2016. URL https://nextjs.org/. Accessed: 2/04/2022.

[20] Vercel, . Next.js github, 2016. URL `https://github.com/vercel/next.js/`. Accessed: 2/04/2022.