NANYANG TECHNOLOGICAL UNIVERSITY

CCDS24-0059

APPLICATIONS OF LARGE LANGUAGE MODELS IN WEALTH MANAGEMENT

Submitted in Partial Fulfillment of the Requirements
for the Degree of Bachelor of Computing in Computer Science
of the Nanyang Technological University

by

Julian Wong Wei Sheng

College of Computing and Data Science
2024

Abstract

While Large Language Models (LLMs) have been developing at a breakneck pace in recent years, there appears to be a lack of documented real-world use cases of LLMs in the domain of personal wealth management. In this final year project, a wealth management app called Fortuna is developed whose goal is to demonstrate how LLMs can be used to provide personalised financial advisory services to the masses. By utilising LLM solutions developed with OpenAI's state-of-the-art models and proprietary tools, on top of being built using a Flask architecture that is highly modular and flexible, Fortuna enables investors to embark on a holistic wealth management journey which starts with saving money and ends with investing money. Fortuna pioneers the integration of LLMs in personal finance, offering customized wealth management advice comparable to human advisors. Future improvements will focus on seamless banking integration, flexible data processing, efficient transaction classification, and specialized model fine-tuning.

Acknowledgements

I would like to express my sincere gratitude to everyone who supported me throughout the development of Fortuna.

First and foremost, I extend my deepest appreciation to my supervisor, Associate Professor Ng Wee Keong, whose expert guidance, constructive feedback and unwavering encouragement were instrumental to the success of this project. His insights into financial technology and portfolio management helped shape this work in invaluable ways.

I am profoundly grateful to the College of Computing and Data Science for providing the resources, facilities, and academic environment that made this research possible.

Finally, I owe my deepest gratitude to my family and friends for their unconditional love, patience, and encouragement throughout my academic journey. Their belief in me has been a constant source of motivation.

This project represents not just my work, but the collective support of all these individuals and institutions.

Table of Contents

Chapter 1: Introduction

1.1 Background

Wealth management can be defined as a service offered by institutions to high net worth individuals with the purpose of preserving and growing wealth for generations to come. In 2024, the worldwide amount of Assets Under Management (AUM) in the wealth management industry is projected to reach 129 trillion USD. In this multi-trillion dollar industry, it is imperative for wealth management institutions (WMI) to keep up with the latest trends as well as their clients' demands in order to appeal to a wider client base. One of the latest trends that has impacted the finance industry as a whole would be Large Language Models (LLMs). Based on the Transformer architecture that was first introduced by Vaswani et al., (2017), LLMs are a type of Artificial Intelligence (AI) which can 'understand' natural language by representing text data as sets of numbers and applying mathematical functions to process these numerical data. LLMs began to gain widespread media attention when OpenAI released ChatGPT in 2022, which is an AI chatbot that boasts remarkable conversational, reasoning and creative skills. With the arrival of such AI tools, many WMIs have shifted their focus to adopt these tools in their business operations, as shown in a survey among 386 management level respondents from the financial sector which reported that 68% of WMIs are deploying AI tools to support decision-making processes, with 32% of them reporting AI use for a significant number of cases (Martinez, 2022).

On the other hand, there is a growing paradigm shift among WMI clients towards user-centric, hyper-personalized advisory experiences that guarantee uniqueness (Bailey, 2022, as cited in Dzhaparov, 2022). According to a report by McKinsey & Company, in the next decade up to 80% of WMI clients want to access financial advice in a Netflix-style model that is data-driven and personalized (Baghai et al., 2020). Although wealth managers and financial advisors are traditionally used to fill the role of making personalised recommendations based on the needs of different client segments,

a 2014 study suggested that these professionals may not always provide financial advice based on their clients' financial situation, but rather be biased towards their own risk tolerance and personal portfolio allocation (Foerster et al., 2017). According to Lam (2016), robo-advisors perform a better job of offering objective and consistent guidance which is tailored to an individual's risk tolerance, investment timeline, personal characteristics and financial goals. Advancements in the field of Natural Language Processing (NLP) such as prompt engineering, retrieval augmented generation (RAG), and multi-agent systems have allowed LLMs to be deployed into wealth management applications as chatbots or virtual financial assistants (Du et al., 2025), which can provide personalised financial advice to clients at a fraction of the cost of wealth managers (Lam, 2016).

1.2 Problem statement

With the growing popularity of LLMs and robo-advisors, many WMIs have started to adopt these technologies into their day-to-day business operations. Below are some of the current shortcomings of LLM applications and robo-advisors in the wealth management industry:

1.2.1 Lack of applications in financial budgeting

Since the introduction of OpenAI's GPT models, financial institutions have been experimenting with various use cases to utilise LLMs in ways that benefit them most. For instance, JPMorgan Chase uses large language models to identify fraud by analyzing email patterns for indicators of compromise (Crosman, 2023a). Correspondingly, Goldman Sachs uses generative AI to support software engineers with coding tasks (Crosman, 2023b), while SouthState Bank has developed an enterprise-specific version of ChatGPT which enables employees to inquire about bank policies, compose emails and summarize meeting notes (Crosman, 2023c). However, there is a lack of literature and real-world applications of using LLMs in the domain of personal finance, specifically in the field of financial budgeting and planning. Considering the fact that saving money is the first step in investing, there exists an

opportunity to explore how LLMs can be used to assist users in their financial planning journey.

1.2.2 Lack of market and behavioral data used

In the financial advisory industry, robo-advisors use Investor Profile Analysis to determine a client's risk profile so that they can make personalized recommendations to their clients and align investment decisions with the client's financial goals. Typically, robo-advisors collect demographic information such as age, gender and marital status together with quantitative information such as financial goals and investment horizons. However, studies have shown that apart from demographic and quantitative info, an investor's behaviour can also be explained by other factors such as personality traits (Jiang et al., 2024) and market sentiment (Campisia & Muzzioli, 2023), both which are not considered by current robo-advisors in the market. Therefore, there exists another opportunity to explore how robo-advisors can utilise external market data and user behavioral data to provide personalised financial advice to investors.

1.2.3 Integrity of Large Language Models

One of the potential risks of using LLMs as financial advisors would be its tendency to output factually incorrect information. Also known as 'hallucination', this phenomenon occurs because LLMs rely heavily on the frequency of patterns seen during training which can lead the model to prefer more common or familiar statements over accurate but less frequent ones, or when LLMs get 'tricked' into recalling exact sentences from memory rather than reasoning about the user's prompt (McKenna et al., 2023). In the finance industry where performance robustness is paramount for maintaining public trust, it remains a challenge to ensure that LLM-based applications only output accurate and contextually correct information, especially in complex or sensitive matters (Shabsigh & Boukherouaa, 2023).

1.3 Objective of project

To address the above issues, this project aims to create a wealth management application called Fortuna which demonstrates several applications of LLMs in the domain of wealth management, with particular emphasis on personal finance and highly personalised financial advising. The application will contain the following LLM features:

1. **LLMs for Financial Transaction Categorisation**
    - Use GPT-4 to classify financial transactions uploaded by user
    - User can analyse their monthly expenditures and budget their expenses accordingly
2. **LLMs for Financial Report Generation**
    - Users can generate a summary report of their monthly expenditure
    - Users can gain additional insights into their monthly spending habits
    - Users can receive personalised tips on how to save more money
3. **LLMs for Portfolio Recommendation**
    - Users can receive customised advice on what portfolios to invest in
    - Advice is based on user risk profile and financial goals
4. **LLMs for Personal Financial Advisory**
    - Users can converse with chatbot to obtain general financial advice
    - Users can ask for any custom financial advice based on their risk profile, financial goals and spending behavior

Chapter 2: Literature Review

In this section, an in-depth review on some popular wealth management and personal finance applications in the market will be conducted, while paying attention to the features and design of each app.

2.1 Syfe

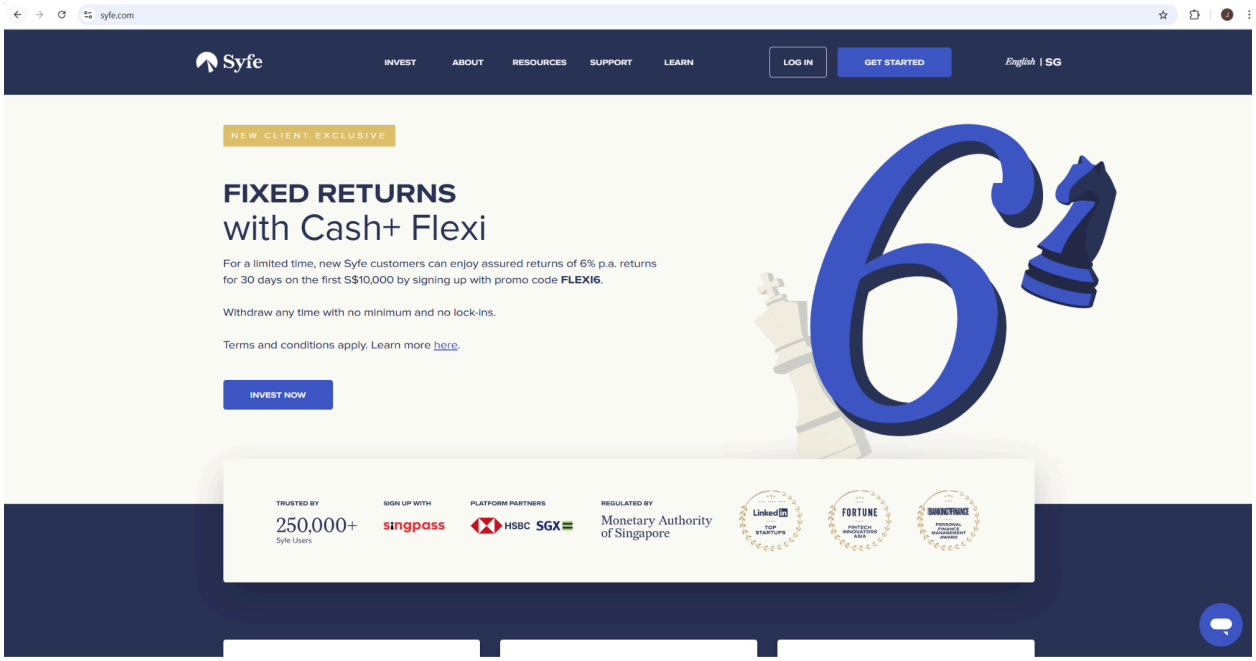

Figure 1.1: *Syfe Landing Page (Source: syfe.com)*

Established in 2019, Syfe is a Singapore-based wealth management company which provides automated investing services (robo-advisory) for individuals to invest efficiently without needing extensive financial knowledge. Syfe offers investors a diverse range of pre-made portfolios to invest in, as well as allowing investors to tailor make their own custom portfolio. Despite being relatively new to the market, Syfe has been gaining traction among local and overseas investors by achieving a total number of 250,000 registered users across Singapore, Australia, and Hong Kong as of October 2024.

2.1.1 Sign up

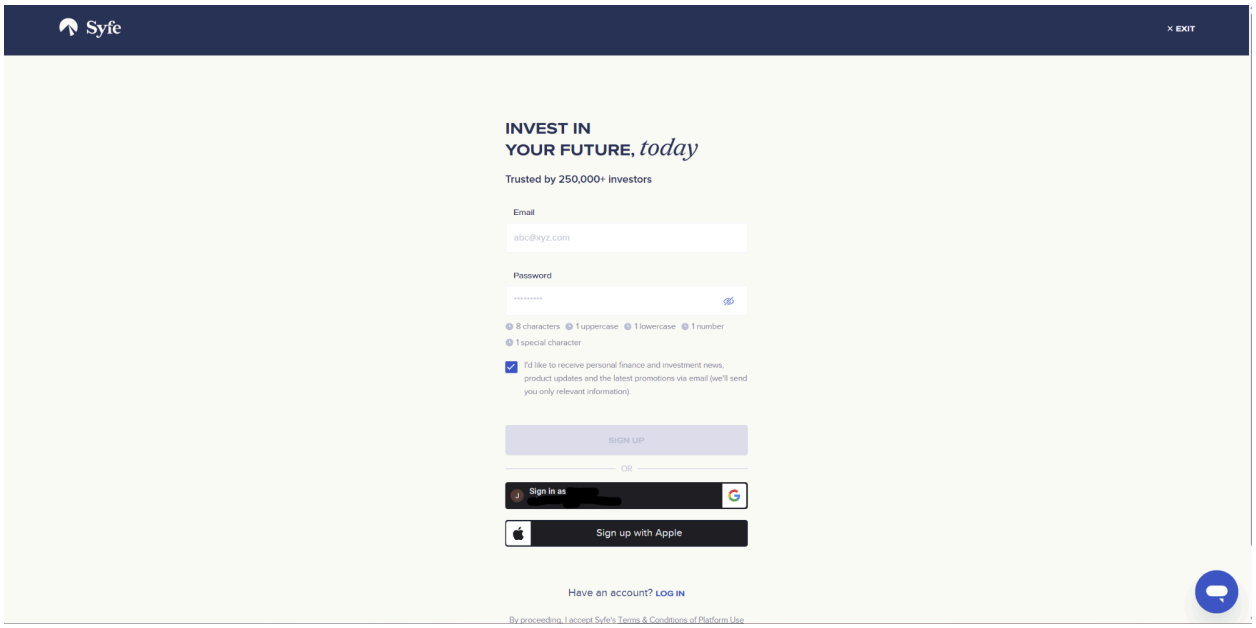

Figure 1.2.1: *Syfe Registration Page (Source: syfe.com)*

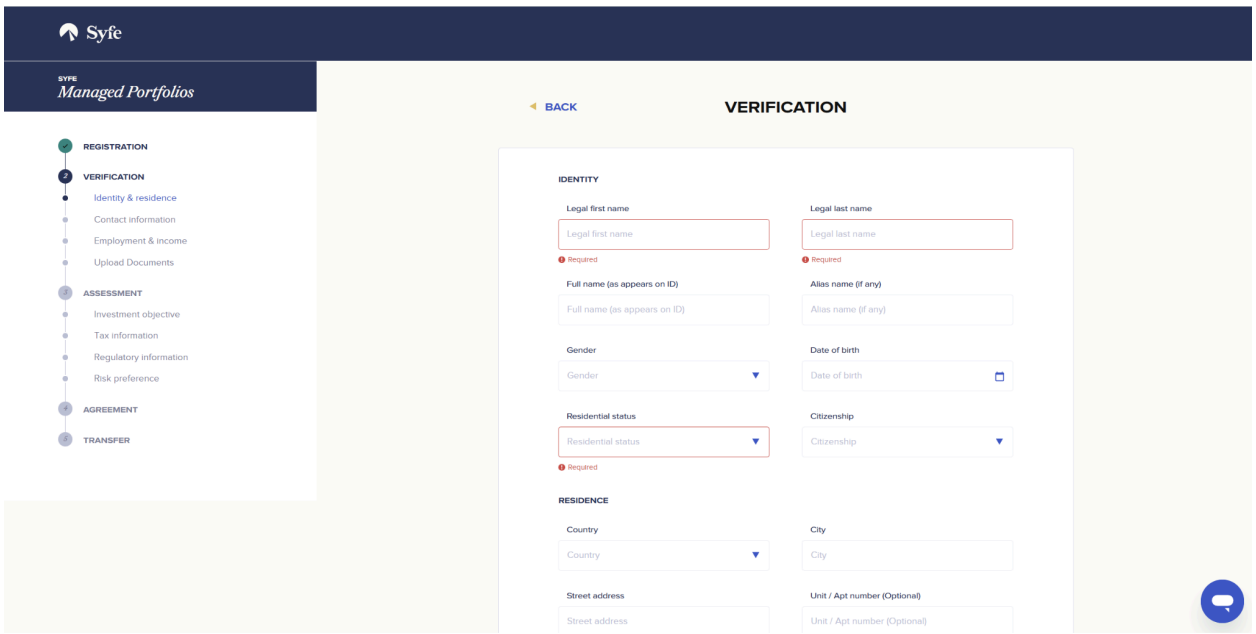

Figure 1.2.2: *Syfe Verification Page (Source: syfe.com)*

The sign up process is rather straightforward. After entering in a valid email address and a strong password, users will be brought to a menu to fill in their personal info for verification purposes.

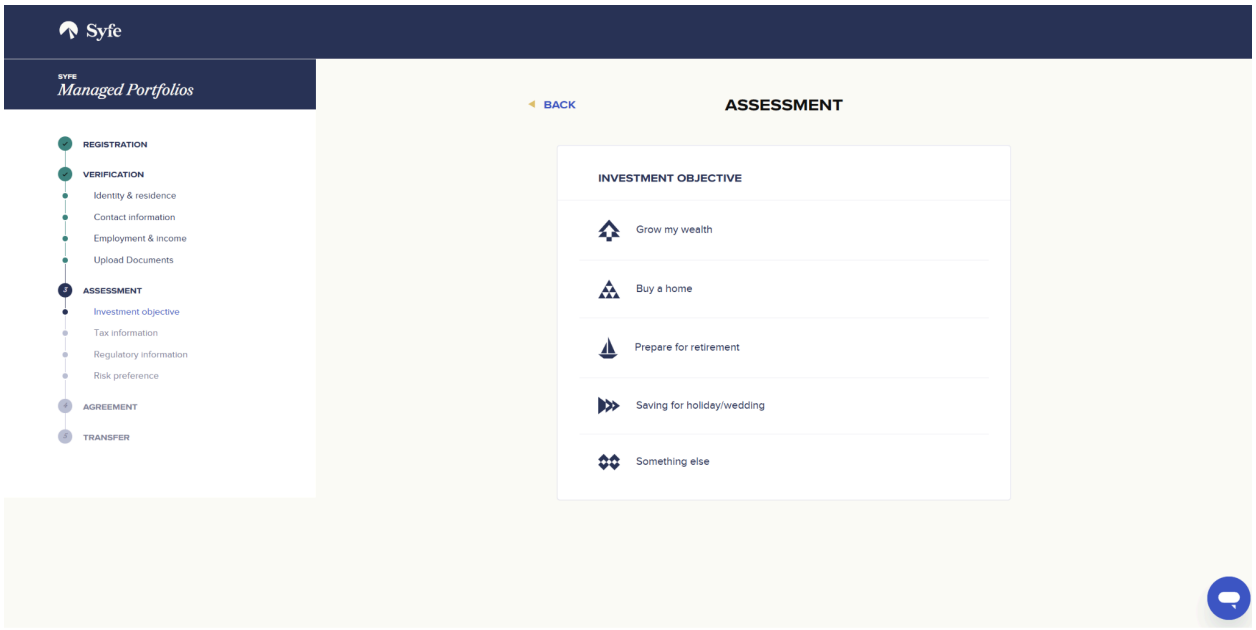

Figure 1.2.3: *Syfe Assessment Page (Source: syfe.com)*

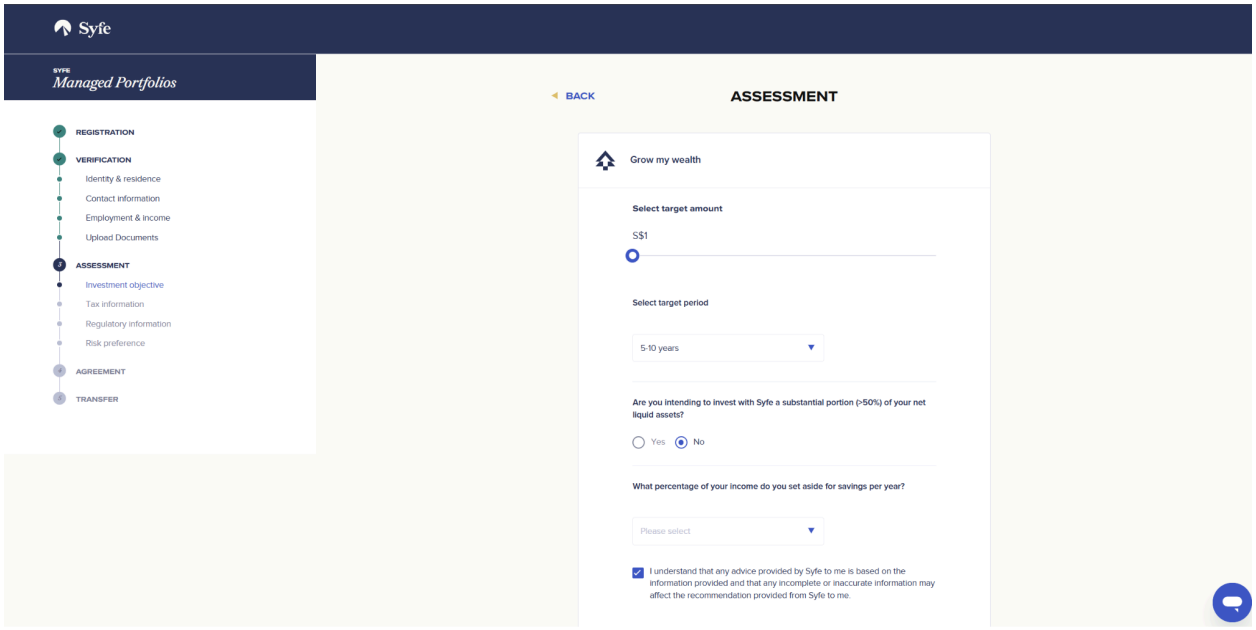

Figure 1.2.4: *Syfe Assessment Page (Source: syfe.com)*

Next, users will be asked to fill in their investment objectives and risk preferences so that Syfe can better understand the user for making personalized portfolio recommendations. Upon selecting an investment objective, users will be asked for their target amount, target period and savings information.

2.1.2 Portfolio Selection

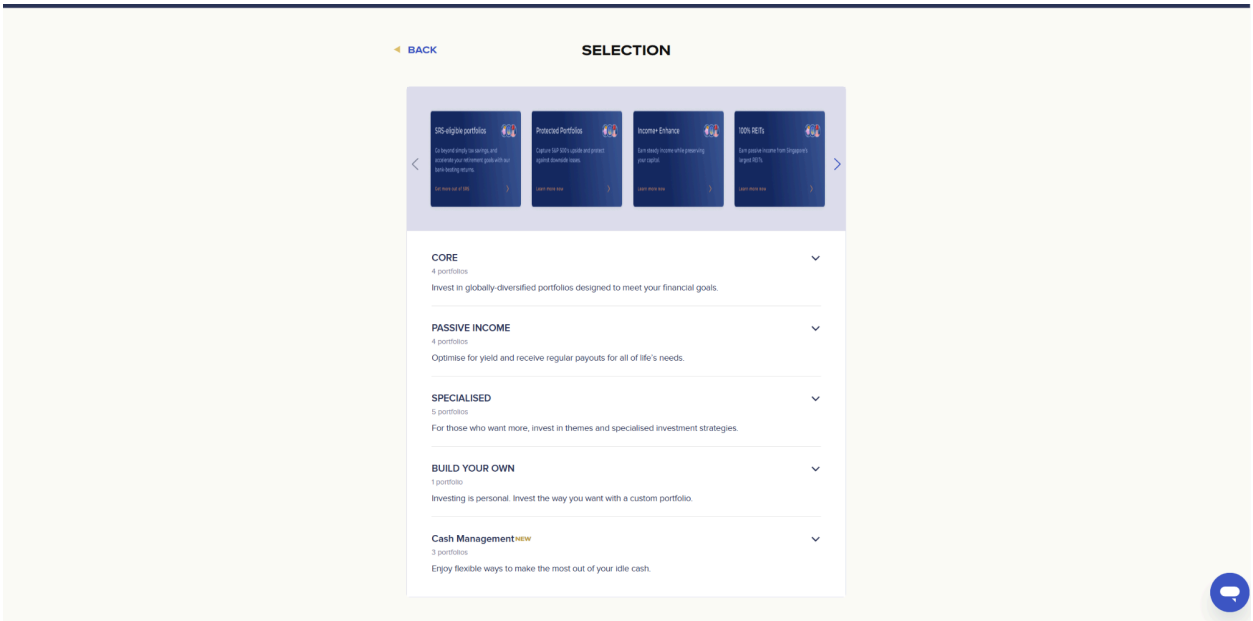

Figure 1.3: *Syfe Portfolio Selection Page (Source: syfe.com)*

After filling out their financial information, users will be asked to select the portfolio they wish to invest in. Syfe portfolios are grouped into 4 categories: Core, Passive Income, Specialised, and Cash Management. Core portfolios allow users to invest in globally diversified portfolios, whereas Passive Income provides users with regular monthly cash flows. On the other hand, Specialised portfolios lets users invest in various thematic portfolios, while Cash Management Portfolios provide users with low risk options to invest their spare cash. Each category also contains portfolios with different risk levels for users to choose.

The following table provides an overview of the available portfolios for Syfe users to invest in:

Category	**Name**	**Risk Level**	**Past Returns**
Core	Core Equity100	High	1Y: 16.38% 3Y: 8.02% 5Y: 10.94%
	Core Growth	Moderately high	1Y: 14.74% 3Y: 4.73% 5Y: 8.59%
	Core Balanced	Moderately Low	1Y: 13.38% 3Y: 1.52% 5Y: 4.46%
	Core Defensive	Low	1Y: 11.01% 3Y: 1.44% 5Y: 2.38%
Passive Income	Income+ Enhance	Moderately Low	YTM: 7% Monthly payout: 5.5 - 6% p.a.
	Income+ Preserve	Low	YTM: 6.5% Monthly payout: 5 - 5.5% p.a.
	100% REITs	High	1Y: -1.12% 3Y: -2.74% 5Y: -1.56%
	REITs with Risk Management	Moderately Low	1Y: 1.37% 3Y: -1.37% 5Y: -0.58%
Specialised	Downside Protected S&P 500	Moderately Low	Current upside cap: 10.8% Estimated max loss: 1.6%

	China Growth	Very High	N/A
	Disruptive Technology	Very High	1Y: 19.9% 3Y: 7.27% 5Y: 13.77%
	Healthcare Innovation	Very High	1Y: 0.79% 3Y: -0.69% 5Y: 5.78%
	ESG & Clean Energy	Very High	1Y: 4.37% 3Y: -1.14% 5Y: 6.77%
Cash management	Cash+ Guranteed (SGD)	Very Low	2.7% p.a. (fixed)
	Cash+ Flexi (SGD)	Very Low	3.2% p.a. (estimated)
	Cash+ Flexi (USD)	Very Low	4.3% p.a. (estimated)

Table 1: *Syfe Portfolios Overview (Source: syfe.com)*

Overall, Syfe provides users with an extensive list of portfolios to choose from.

2.1.3 Recommended Portfolios

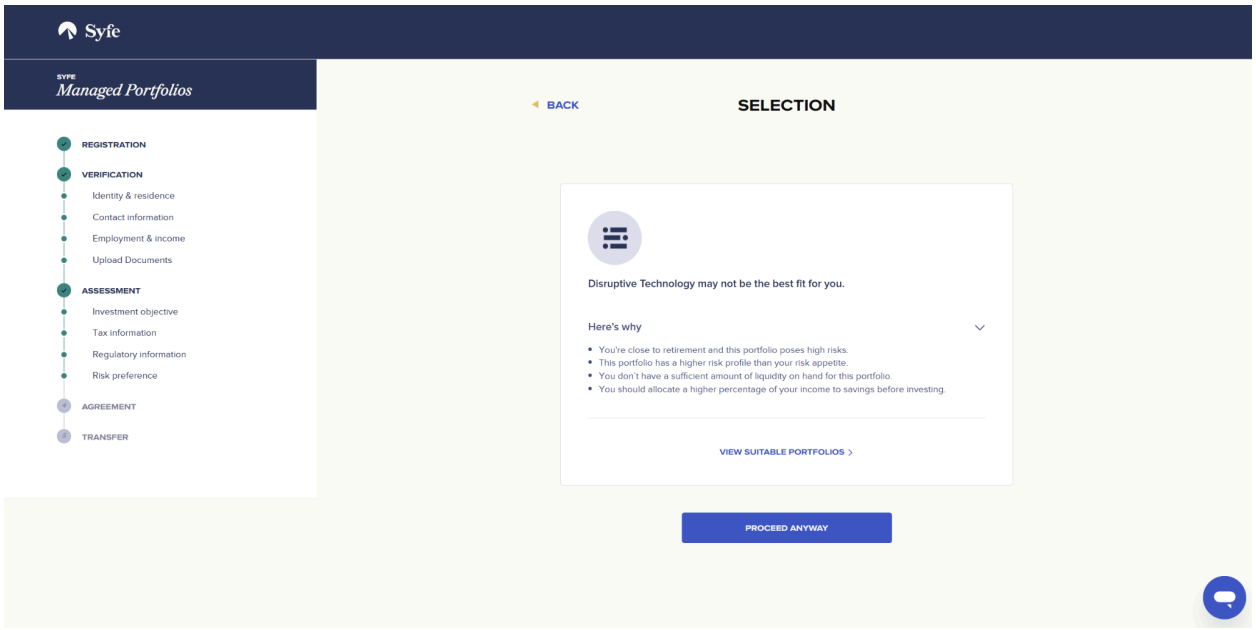

Figure 1.4.1 *Syfe Selection Warning Page (Source: syfe.com)*

When users select a suboptimal portfolio, Syfe will warn the user and give the user reasons on why the portfolio they selected may not be the best choice for them. Then, users will have the option to view a list of suitable portfolios for them.

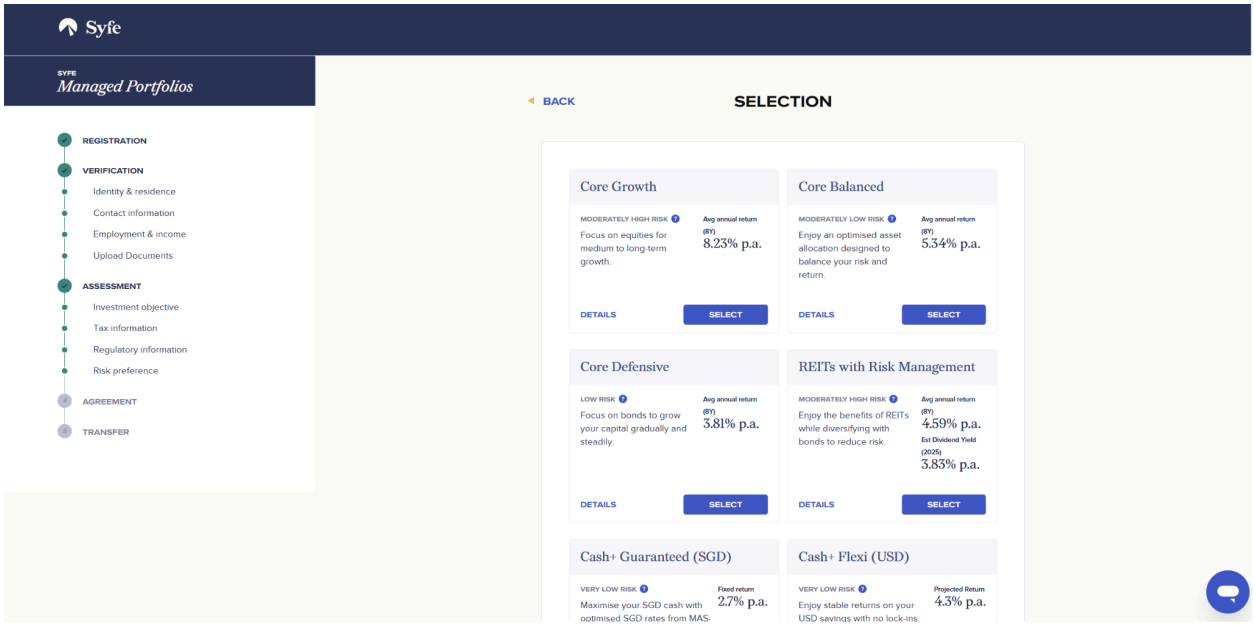

Figure 1.4.2: *Syfe Portfolio Recommendation Page (Source: syfe.com)*

It is interesting to note that this seems to be the only way for users to view the list of recommended portfolios. As the goal of wealth management is to provide personalized services to users, users should be able to view the recommended portfolio list right from the beginning, which is what was implemented in Fortuna.

2.1.4 Dashboard

Figure 1.5.1: *Syfe User Dashboard (Source: syfe.com)*

After funding their portfolio, users will be able to access the Dashboard to gain an overview of all their portfolios, which includes basic statistics about each portfolio such as current value and returns. Users can use the interactive chart of each portfolio to see how the portfolio has performed throughout different time intervals. Overall, the dashboard design is quite simplistic yet professional, due to the fact only the most important statistics are displayed for users to see. This prevents users from being overwhelmed with too much information, as perceived ease of use is one of the main drivers that can increase the willingness of investors to use robo-advisory services (Luo et al., 2024).

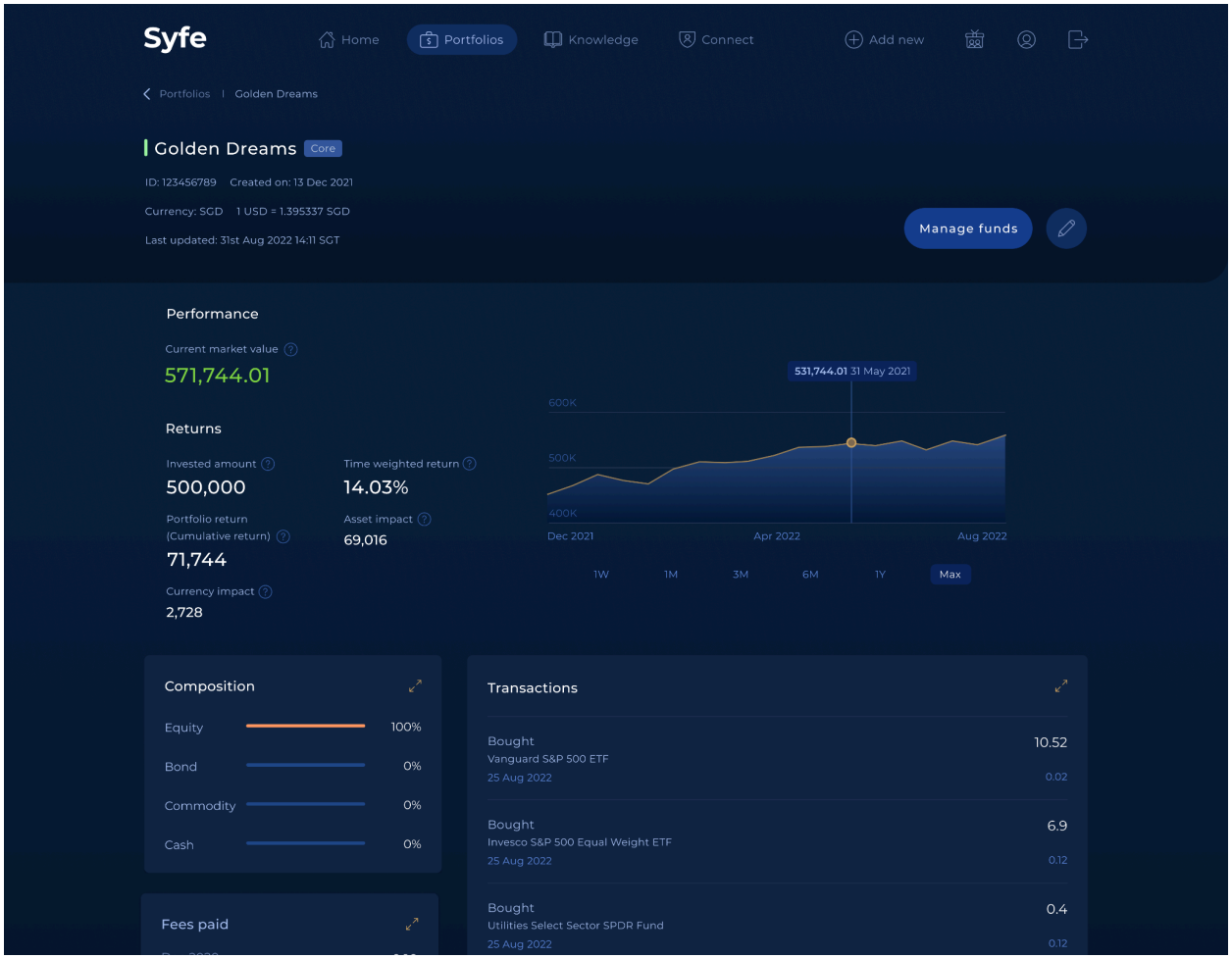

Figure 1.5.2: *Syfe Portfolio Details (Source: syfe.com)*

Users can also view details of each of their portfolios. This will show more information such as the different return metrics, return distribution, portfolio composition, and historical transactions. Again, only meaningful statistics are displayed to keep the dashboard design as clean and clutter-free as possible.

Inspired by this design choice, Fortuna also aims to keep the portfolio details page as simple as possible, all while maintaining only the most essential statistics and portfolio information for users to see.

2.1.5 Chatbot

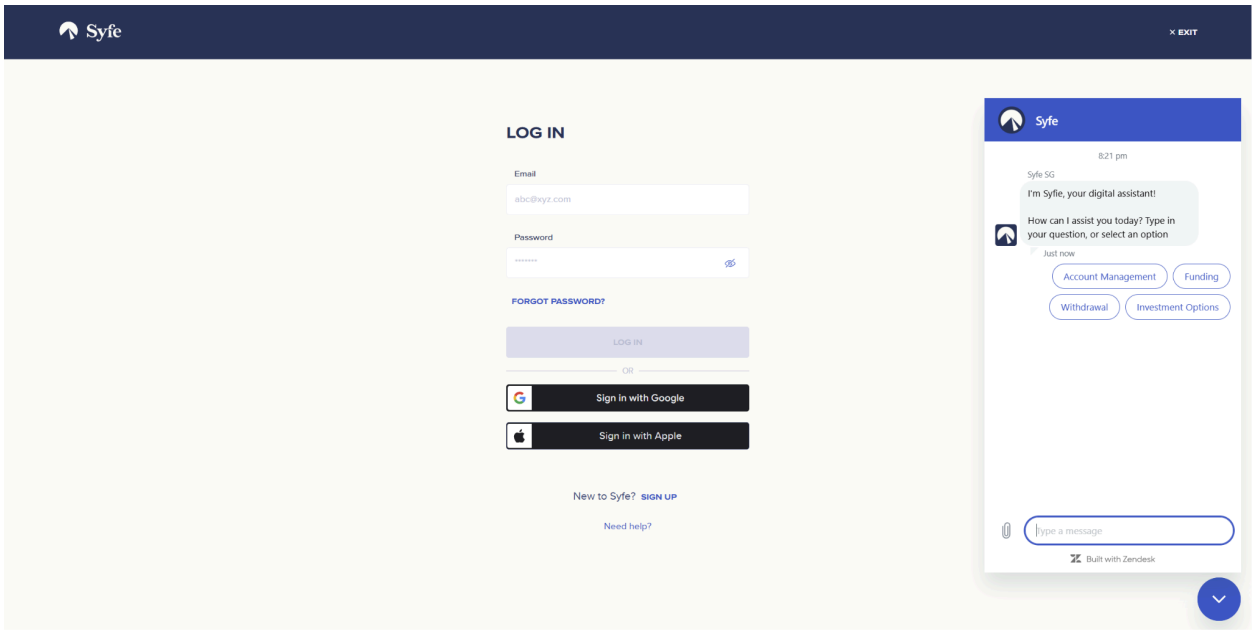

Figure 1.6: *Syfe Chatbot Interface (Source: syfe.com)*

Syfe boasts a chatbot feature which can only be accessed in the login page, registration page and the home page. Since the chatbot can be accessed without logging in, this implies that the chatbot does not have information about the user. The chatbot appears to be a rule-based chatbot, as it would constantly direct users to select one of the suggested queries about Syfe's products and services. Rule-based chatbots follow predefined scripts or decision trees to respond to specific user inputs, which allows them to only handle limited conversations. Overall, the chatbot in Syfe serves as a proxy for a 24/7 customer support agent, rather than being an intelligent financial advisor who can provide tailored financial advice for the user.

On the other hand, the chatbot deployed in Fortuna is developed using OpenAI's GPT-4 models, which has access to application and user data from the database. This allows Fortuna's chatbot to intelligently understand the user's intent to provide a smooth and natural conversational experience, all while referencing relevant financial data like how a real financial advisor would.

2.1.6 Other features

1. **Financial education**

   Under the ‘Knowledge’ tab, users can explore articles, advice, webinars and magazines which are curated by the Syfe team. These materials allow users to brush up on their financial knowledge and gain valuable insights into the latest financial news and trends.

   While Fortuna does not contain a separate module for financial education, users can prompt Fortuna’s chatbot for information about any financial topic they wish to learn about. Since the chatbot is developed with OpenAI’s GPT-4 models, it will have no problem answering most questions about the financial domain.

2. **Financial consultation**

   Syfe also allows users to book free telephone consultation sessions with their wealth advisors. According to Syfe, these ‘no strings attached’ consultations help users understand Syfe’s investment frameworks, investment options, and use Syfe’s risk analyser to review financial goals and investment objectives.

   As Fortuna’s chatbot has access to application and current user data, users can ask Fortuna about the investment products offered or any personalized financial advice, just like they would with a real-life financial advisor.

2.1.7 Summary

Overall, Syfe is a wealth management app whose main target audience are retail investors who have little to no financial knowledge. By providing users with a large number of custom made portfolios to choose from, not only does Syfe ensure that almost every investor will be able to find a portfolio that suits their needs, it also removes the mental and time burden required for investors to create their own portfolios from scratch. Furthermore, the portfolio dashboards are designed to only showcase the most important financial information to users, which further solidifies the fact that Syfe's main target audience are investors with little to no financial background.

While Fortuna lacks the diversity of portfolio options that Syfe provides, it makes up for in its superior chat bot capabilities which more closely resembles a real financial advisor. Furthermore, Fortuna provides users with the ability to immediately see all recommended portfolios right from the get go, which aligns with how a real financial advisor would work with their clients.

2.2 Wallet by BudgetBakers

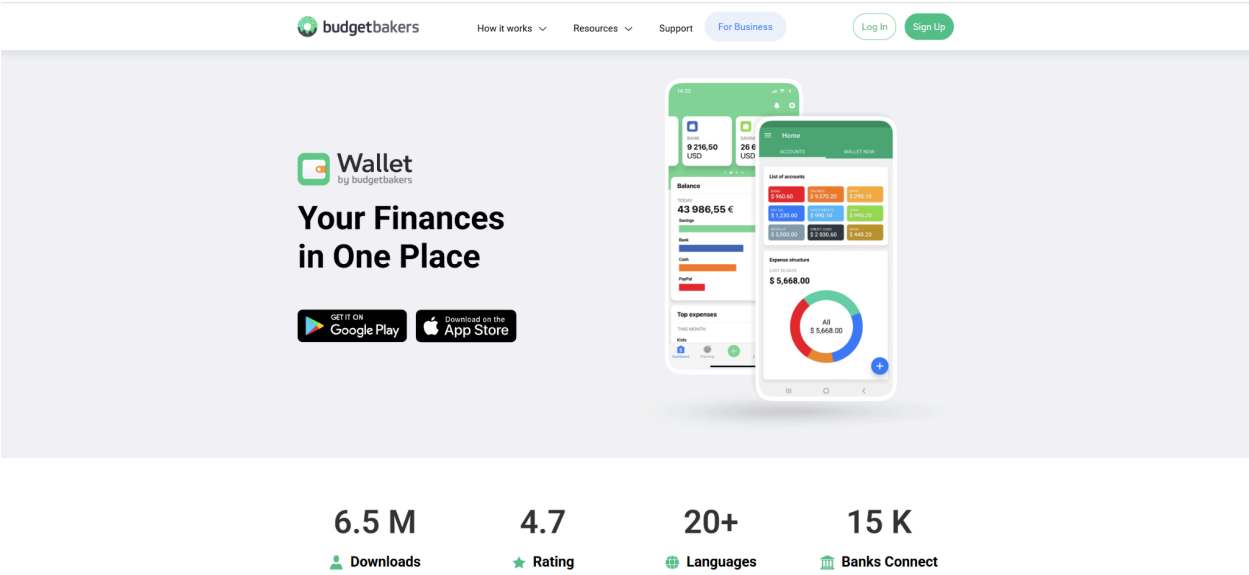

Figure 2.1: *Wallet Landing Page (Source: budgetbakers.com)*

BudgetBakers is a small fintech company founded in 2014 by Jan “Honza” Muller. Their personal finance app, Wallet, was designed to help users monitor their monthly expenditures and overall financial health. Some of the key features of this app include: monthly expenses tracking, syncing with banks for automatic transaction importing, and financial report generation. The app is available on Android, iOS and web platforms. For this section, we will be reviewing the web version of Wallet.

2.2.1 Sign up

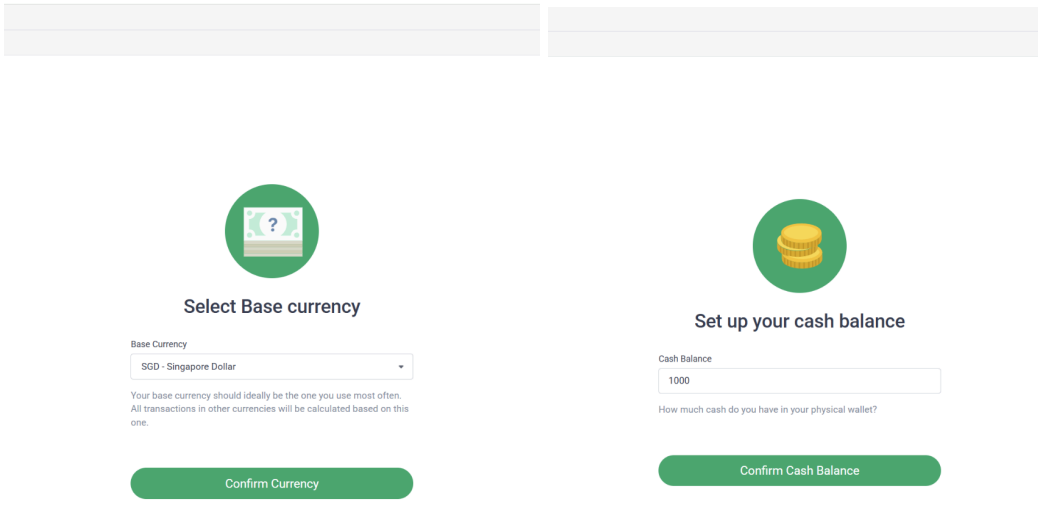

Figure 2.2: *Wallet Sign Up Page (Source: budgetbakers.com)*

The signup process on Wallet is relatively simple. Users will be prompted to select their currency choice, as well as input the amount of cash they have psychically.

In the current implementation of Fortuna, all transactions are assumed to be in SGD. Furthermore, the physical cash amount of the user is not taken into consideration, as it is believed that the majority of a user's financial transactions occur digitally in today's times.

2.2.2 Add Account

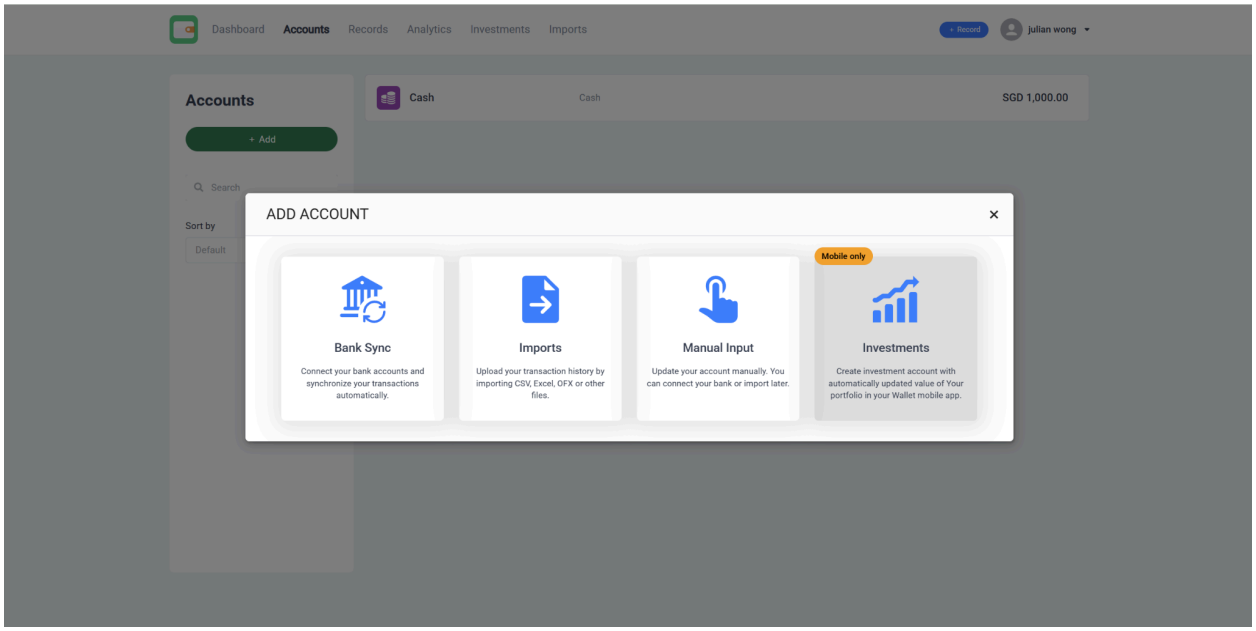

Figure 2.3: *Wallet Add Account Page (Source: budgetbakers.com)*

After signing up, the user will have to add an account for inputting their financial transactions into the system. Wallet provides 3 ways for users to add their transactions: syncing the app with their bank account, uploading their transactions via CSV files or Excel files, and manually keying in each transaction. It is worth noting that the Bank Sync feature can only be accessed by users who pay a monthly subscription fee, hence the Bank Sync feature will be excluded from this review. Since manually inputting records is not feasible for huge amounts of transactional data, this feature will also not be reviewed.

2.2.3 Import Transactions

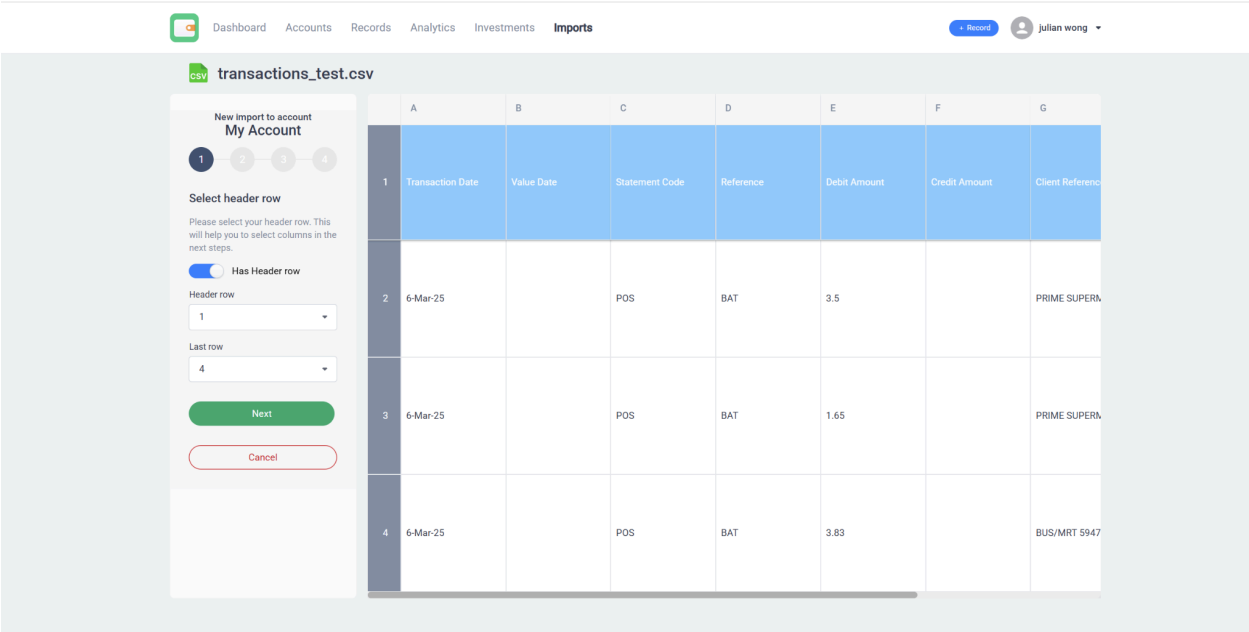

Figure 2.4: *Wallet Import Transactions Page (Source: budgetbakers.com)*

To import transactions via file upload, the first step will be to select the header row in the file. This is because transaction exports of different banks are formatted differently, so the header row is not always in the first row of the file. The second step would be to select the amount columns, i.e. the columns that correspond to the debit and credit amounts. There is also an option to specify if the debit and credit columns are in the same row or not. The third step would be to select the transaction date column, while the last step would be to select the column for transaction description and payee.

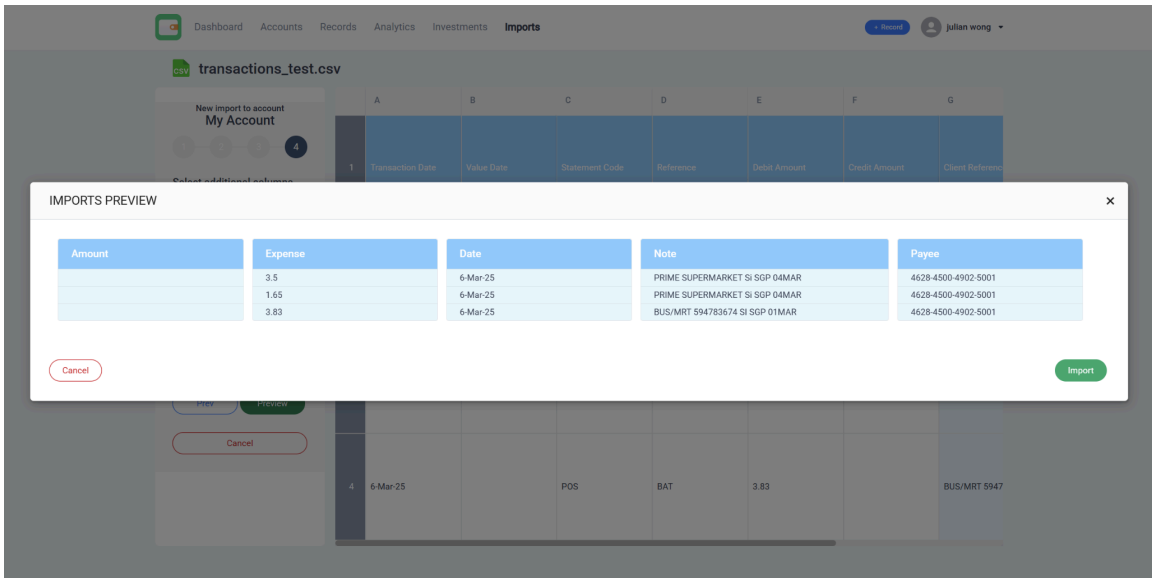

Figure 2.5: *Wallet Imports Preview Modal (Source: budgetbakers.com)*

After specifying all the required columns, the user can check if the columns are labelled correctly via the Imports Preview pop up.

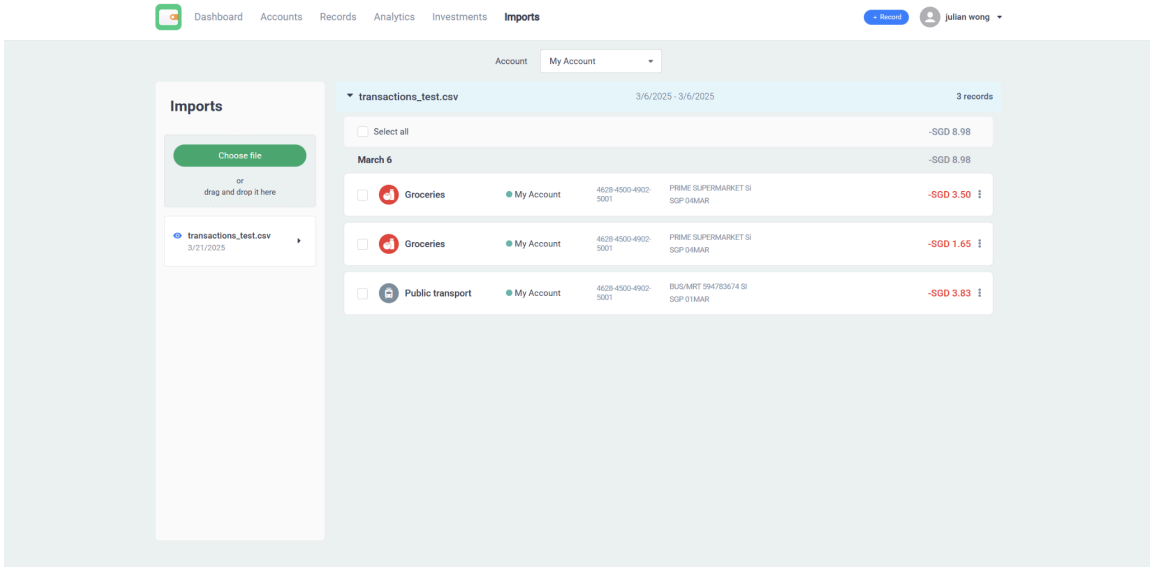

Figure 2.6: *Wallet Import Transactions Page (Source: budgetbakers.com)*

Upon successful import, the user will be able to see all their transactions displayed in the above menu.

2.2.4 Dashboard

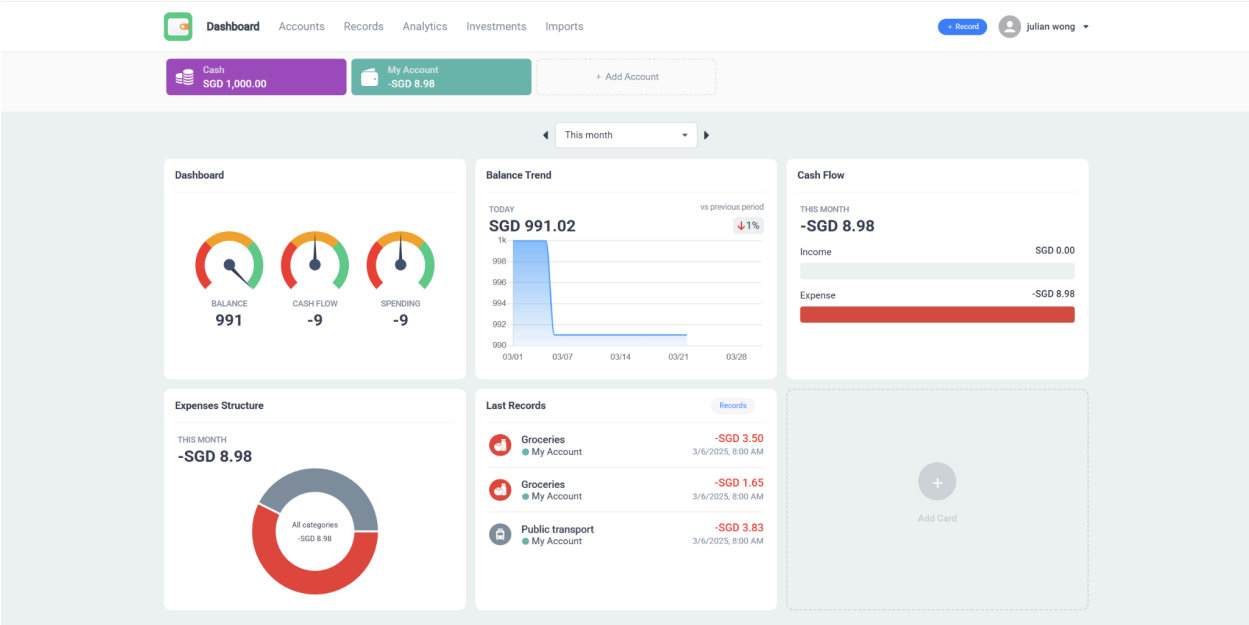

Figure 2.7: *Wallet Dashboard Page (Source: budgetbakers.com)*

With transaction data imported into the system, the user can now use the dashboard to gain a high-level view of their monthly expenditures. By default, the dashboard displays several data cards such as line graphs, bar charts, pie charts and tables which contain information about the user's monthly cash flows and expenditures. Each card can be removed by the user, and the user has the option to add additional cards to the dashboard such as cash flow trend lines and monthly expenditure breakdowns by category.

2.2.5 Analysis

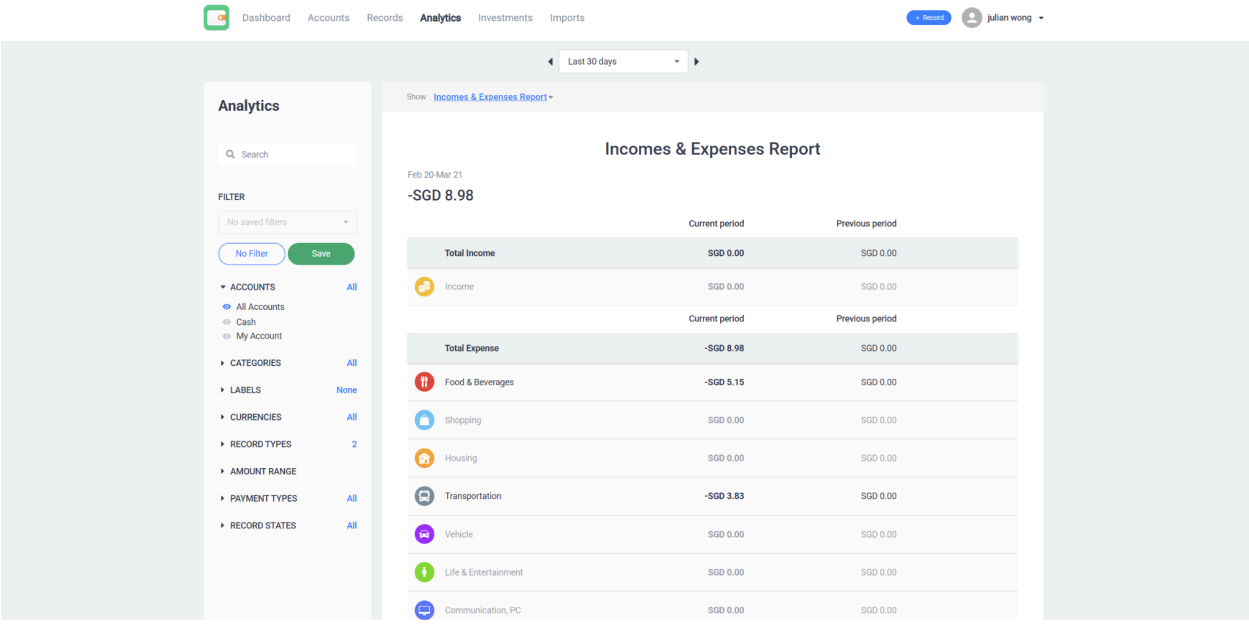

Figure 2.8 *Wallet Analysis Page (Source: budgetbakers.com)*

Under the Analysis tab, users can further analyse their monthly expenditures with different data visualisation options. In the ‘Income and Expenses Report’, users can obtain a detailed breakdown of how much they spent in each category. Furthermore, each category is further broken down into smaller subcategories, such as the ‘Food & Beverages’ category is split into ‘Food & Beverages’, ‘Bar, cafe’, ‘Groceries’ and ‘Restaurant, fast-food’ subcategories. According to the official Wallet webpage, it is mentioned that the categorisations are ‘AI empowered’, which suggests that a machine-learning framework is developed to classify the bank transactions. However, the exact model or tech stack used was never mentioned. In the case of Fortuna, users' bank transactions are classified using OpenAI’s GPT-4 models. Apart from the tabular breakdown of expenses, users can also view their expenditures in the form of bar charts, line charts and pie charts, as well as filter the data by different categories.

2.2.6 Summary

Overall, Wallet provides a great number of options for users to analyse and gain insights into their monthly spending habits, as well as provides a high degree of customisation and support for different data entry methods. When compared to Wallet, Fortuna's monthly expenditure module lacks the number of data visualisations and degree of personalisation that Wallet has. However, Fortuna makes up for its shortcomings in its report generation capabilities, as well as integration with other aspects of personal wealth management, all made possible with the power of LLMs.

Chapter 3: Design and Implementation

3.1 Design Principles

In this section, we will take a look at the design principles and choices behind Fortuna. Throughout the development cycle of Fortuna, the following 3 points were upheld as Fortuna's core design philosophies:

1. **Build application around LLM use cases**
   As the goal of this project is to demonstrate how LLMs can be applied in the field of wealth management, all of Fortuna's features are built with the sole purpose of giving LLMs as much data as possible to work with, and/or to exhibit a LLM use case (text summarisation, text classification, etc). Hence, the reason why Fortuna does not adopt sophisticated portfolio management features or optimization techniques is simply due to a design choice, as these features do not contribute to the main objective of the project.

2. **Pleasant User Interface (UI)**
   While the main goal of this project is to build a LLM-centric application, developing a product that is commercially viable is a good requirement to have for any application, especially in the highly competitive field of digital wealth management where user satisfaction is a key driver for success (Yin, 2024). To achieve this objective, it is important that Fortuna strives to deliver pleasant user experiences (UX), and the most direct way to achieve this goal is to ensure that the UI of each webpage is as attractive as possible. Furthermore, users must find it easy to navigate between different web pages.

3. **Application functionalities are interconnected**
   To allow Fortuna to stand out from other competitors, much effort has been put into ensuring that Fortuna's features are well connected with each other, leading Fortuna to behave as a thoughtfully-designed ecosystem which is conducive for personal financial planning and wealth management. From importing bank

transactions to monthly expenditure reports to customized portfolio suggestions, each feature in Fortuna is built upon the previous one to deliver a holistic wealth management experience for its users that starts with saving money and ends with investing the saved money.

3.2 Overview of Functionalities

The following are the core functionalities of Fortuna:

1. **User registration & authentication**
    a. Allows registered users to log in and out of their accounts
    b. For first time users, they will be able to register an account. After account registration, users will be prompted to complete a short investor profile questionnaire which helps Fortuna understand the user's risk tolerance, risk appetite and financial goals.

2. **Portfolio selection menu**
    a. Provides users with a list of premade portfolios to invest in.
    b. Displays details about each portfolio such as risk, annual returns, historical performance and portfolio composition.
    c. Users can get AI to recommend portfolios based on their risk profile and financial goals.
    d. Users can input how much money they want to invest in the portfolio.

3. **Dashboard**
    a. Provides users with a snapshot of their financial status.
    b. Users can manage and track their financial goals.
    c. Allows users to view their purchased portfolios.

4. **Monthly expenses tracker**
    a. Allows users to upload their monthly transaction data to Fortuna.
    b. Users can track how much money they spend per month, as well as how much they spend in each category.

c. Users can generate custom financial reports to gain more insights into their monthly spending habits, and get tips for saving more money.

5. **AI financial advisor chatbot**
    a. Users can interact with a financial advisor chatbot to obtain general financial advice.
    b. Users can also ask the chatbot questions related to their monthly expenditures.
    c. Users can ask the chatbot to recommend portfolios.
    d. Users can ask the chatbot about their financial goals.

3.3 Framework

Fortuna is developed using Flask, which is a lightweight and flexible web framework written in Python. As Flask adopts a minimalistic approach, it allows Fortuna to be built efficiently without adding unnecessary complexity to the codebase. Moreover, Flask's modular architecture promotes easy customisation and scalability, allowing new system functions to be easily added to Fortuna while keeping the codebase organised. Lastly, as Flask is built with Python, it allows easy integration with OpenAI's APIs which are also Pythonic in nature.

3.4 System Architecture

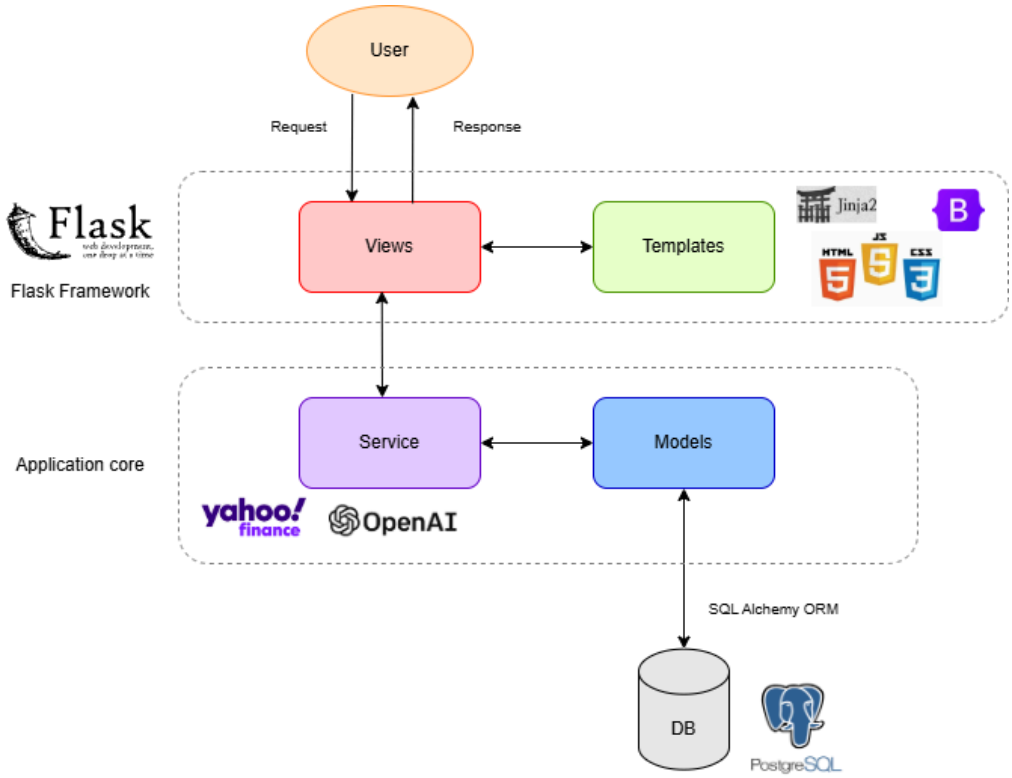

Figure 3: *Fortuna System Architecture Diagram*

As shown in the diagram, Fortuna adopts the Model-View-Controller (MVC) architecture. Below is a simple explanation of the MVC architecture:

1. **Modal**

   Manages data, logic and rules of the application. It represents the data structure and handles interactions with the database.

2. **View**

   Presents data to the user. It represents the UI component that displays information and captures user interactions.

3. **Controller**

   Processes user inputs from the View, interacts with the Model, and selects appropriate Views for responses. It acts as an intermediary between the Modal and View.

While Flask does not strictly enforce any architecture, it aligns well with MVC's concepts. The way MVC is implemented in Fortuna, in conjunction with Flask, is as follows:

1. **Model**
   The Model is handled by Flask's default Object Relational Mapping (ORM) which is SQLAlchemy for interaction with database objects.

2. **View**
   Flask's Jinja templates represent the View, which dynamically renders application data into a web page's HTML structure.

3. **Controller**
   The Controller is implemented through Flask routes that handle application requests, interact with external APIs, manipulate data from SQLAlchemy (Model), and render templates (Views)

3.5 Front end

3.5.1 Tech Stack

Fortuna's front end is built entirely with native HTML, CSS and JavaScript. While most developers rely on complex JavaScript and CSS frameworks such as React, Angular, or Tailwind to develop their front end, the decision to build Fortuna without using these tools serves a few benefits:

1. **Better integration with Flask**
   Vanilla Flask is designed to support native HTML, CSS and JavaScript right out of the box. By default, Flask apps serve static files such as HTML, CSS and JavaScript from a '/static' directory. Flask also uses Jinja2 templating to dynamically render HTML files, allowing web pages to have dynamic elements without relying on additional frameworks.

2. **Lightweight setup**
   By only using native HTML, CSS and JavaScript, the time taken to set up Fortuna as a new Flask project is greatly reduced, and more time could be allocated to developing application features and attractive UI. Since the goal of this project is to build a Minimum Viable Product (MVP) that can demonstrate LLM use cases, a simplistic set up is the best approach.

3. **Fast prototyping**
   As Fortuna does not rely on complex external frameworks, changes to the codebase to be made rather quickly. Hence, this allows for rapid testing of new ideas, as deploying new UI designs or application features does not require complicated build steps or dependencies.

3.5.2 Libraries

The following table contains a list of libraries used to develop Fortuna's front end, as well as a short description on how and why they were used.

Name	**Description**
Jinja2	Jinja2 is a template engine library written in Python. It is included by default in the Flask framework. Jinja2 is used to render HTML pages dynamically by inserting Python variables and data from the application backend into HTML, which lets Fortuna display application data such as user name, user portfolios, and financial goals correctly. Another useful feature of Jinja2 is template inheritance. Template inheritance allows the creation of a base template with common layouts and styles, which then can be inherited by child templates that fill in a specific context. In the case of Fortuna, Jinja2 templates were used to ensure that each web page uses the same header and side navigation bar, as well as consistent stylings such as text font and theme colors.
Bootstrap	Bootstrap is a popular front end framework that helps developers build responsive, mobile-first websites quickly and easily using prebuilt HTML, CSS, and JavaScript components. In Fortuna, most UI elements such as buttons, modals and forms were built using the default Bootstrap classes. Using Bootstrap helps ensure that UI elements in Fortuna have consistent styling. It also eliminates the need for designing UI elements from scratch.

jQuery	jQuery is a fast, small, and feature-rich JavaScript library that simplifies tasks like DOM manipulation, event handling, AJAX, and animations. jQuery is used in Fortuna as it simplifies the syntax of vanilla JavaScript, resulting in shorter code that is easier to read and maintain, all without the complicated setup required for complex JavaScript frameworks like React, Vue or Angular.
Chart.js	Chart.js is a simple yet flexible open-source JavaScript library for creating animated and responsive graphs using the HTML <canvas> element. Chart.js is used to generate interactive and responsive line charts, bar charts, and pie charts in Fortuna.
Marked.js	Marked.js is a lightweight JavaScript library that converts Markdown into HTML. This library is used to ensure that the output of Fortuna's chatbot is formatted neatly in the chatbot interface.
Lucide	Lucide is a collection of consistent, open-source Scalable Vector Graphics (SVG) icons that can be used in any website. In Fortuna, Lucide is incorporated into Jinja templates via a Content Delivery Network (CDN). Lucide icons are used in the navigation sidebar of Fortuna.

Table 2: *Description of front end libraries*

3.6 Back end

3.6.1 Code Organisation

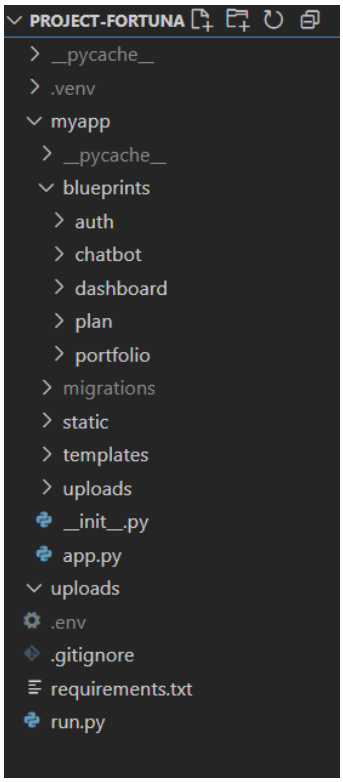

Figure 4.1: *Fortuna Top-Level Code Structure*

As Fortuna is a Flask-based application, the backend is written in Python.

In the top-most level module, **/myapp** is a folder which contains most of the application logic and data, **/uploads** is a folder for temporarily storing files that are uploaded by the user into Fortuna, requirements.txt is a text file for storing package version dependencies, and **run.py** is the entry point of the Flask application.

Inside **/myapp**, **/static** stores CSS files which contain common styles to be used throughout Fortuna (primary colors, font styles, background colors, etc), while **/templates** contain the main template HTML file to be inherited by all other templates in Fortuna. This main template file contains the code for the website's header and

navigation sidebar, which are all common components to every web page within Fortuna. **app.py** is the app factory used to register Blueprints and set up Flask configurations. Lastly, **__init__.py** is simply an empty file whose purpose is for Flask to recognise **/myapp** as an importable module.

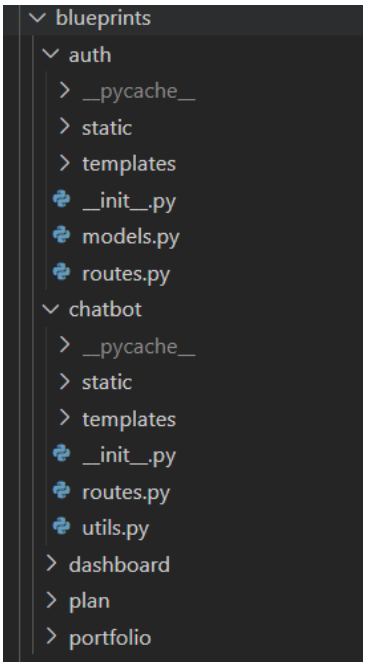

Figure 4.2: *Fortuna Auth Blueprint Code Structure*

To take full advantage of Flask's modularity, Fortuna is built using Flask's Blueprint system. Blueprints allows a Flask application to be organised into reusable modules. As seen in the image above, each blueprint has its own **/templates** and **/static** folders, code to handle application logic (**routes.py**), and code that defines database models (**models.py**). Thanks to the modular structure of Blueprints, Fortuna can be easily maintained as well as scaled up to include new features.

Fortuna is split into the following Blueprints:

1. **Auth**

   Module that contains code for user authentication features. This includes user sign in, sign out and setting up a new account, as well as the frontend code for displaying web pages related to user authentication.

2. **Chatbot**

   Module that contains code for chatbot features. This includes handling API requests to OpenAI's servers, as well as frontend code for the chatbot web page.

3. **Dashboard**

   Module that contains code for the main user dashboard. This module is connected to other modules which allows it to display user data such as user portfolios and user financial goals

4. **Plan**

   Module that contains code for financial planning features. This includes the backend logic for processing user's transaction data, classifying transactions with GPT-4, and front end code for displaying financial data.

5. **Portfolio**

   Module that contains code for portfolio related features. This includes fetching and displaying portfolios and user portfolios, as well as backend logic for adding portfolios to a user's account.

3.6.2 Application Logic

In this section, we will be taking a look at the application logic of each Blueprint, which is handled by the endpoints defined in **routes.py** of each Blueprint. Database and front end implementations will be described in sections 3.7 and 3.11 respectively.

3.6.2.1 Auth

This Blueprint handles user authentication functions of Fortuna. The url prefix for routes (endpoints) in this Blueprint is **/auth**. The following table lists down all the routes in **/auth**, as well as a description of each route's function:

Route	**Methods**	**Description**
/login	GET, POST	Handles user login. GET displays the login form, POST processes login credentials, validates them against the database, and redirects to the dashboard on success.
/create-account	GET, POST	Handles new user registration. GET displays the registration form, POST creates a new user in the database, logs them in, and redirects to the questionnaire. First step in the registration sequence.
/questionaire	GET, POST	Second step in registration. GET displays the questionnaire form, POST processes user answers about financial preferences (risk tolerance, investment knowledge, etc.) and stores them in session before redirecting to financial goals.
/financial-goals	GET, POST	Third step in registration. GET displays the financial goals form, POST processes user's

		financial goals (including name, amount, and target date) and stores them in session before redirecting to register endpoint.
/register	GET	Final step in registration. Processes all collected data from previous steps, saves questionnaire responses and financial goals to the database, and redirects to the dashboard.
/add-financial-goal	POST	Allows logged-in users to add new financial goals. Accepts JSON data with goal details and adds them to the database.
/logout	GET	Logs out the current user and displays the logout page.

Table 3.1: *Description of Flask routes in Auth Blueprint*

3.6.2.2 Chatbot

This Blueprint handles the chatbot functions of Fortuna. The url prefix for routes (endpoints) in this Blueprint is **/home/chatbot**. The following table lists down all the routes in **/home/chatbot**, as well as a description of each route's function:

Route	**Methods**	**Description**
/	GET, POST	Serves the main chatbot interface page. Requires user login and provides current time to the template.
/init_chat	GET, POST	Initializes a chat session with OpenAI Assistants API. Creates a new thread with a welcome message and stores assistant_id and thread_id in the session. Returns these IDs as JSON.

/get_model_response	GET, POST	Handles user messages to the chatbot. Sends user input to OpenAI Assistants API, processes the response, and handles any function calls required by the Assistant. Can call various utility functions defined in **utils.py** .

Table 3.2: *Description of Flask routes in Chatbot Blueprint*

utils.py contains helper functions that provide OpenAI's Assistant access to application data. Refer to section 3.8.2.2 for more information on the functions in **utils.py**.

3.6.2.3 Dashboard

This Blueprint handles the dashboard functions of Fortuna. The url prefix for routes (endpoints) in this Blueprint is **/home/dashboard**. The following table lists down all the routes in **/home/dashboard**, as well as a description of each route's function:

Route	**Methods**	**Description**
/	GET	Displays the main dashboard page. Retrieves the user's portfolios and financial goals from the database and passes them to the dashboard template. Requires user login.
/get_user_portfolios	POST	Retrieves portfolios for a specific user ID provided in the JSON request. Returns the portfolios as JSON data. Requires user login.

Table 3.3: *Description of Flask routes in Dashboard Blueprint*

3.6.2.4 Plan

This Blueprint handles the financial planner functions of Fortuna. The url prefix for routes (endpoints) in this Blueprint is **/home/plan**. The following table lists all the routes in **/home/plan**, as well as a description of each route's function:

Route	Methods	Description
/	GET	Displays the financial planning overview page with all transactions belonging to the current user.
/upload_csv	POST	Processes bank transaction CSV files uploaded by the user, validates the format, converts data to Transaction objects, and stores them in the database.
/submit_classification_batch	POST	Creates and submits a batch job via OpenAI's Batch API to classify uncategorised debit transactions into spending categories (housing, food, transportation, etc.).
/process_classification_batch	POST	Retrieves and processes the results from the OpenAI batch classification job, updating transaction categories in the database.
/generate_financial_report	POST	Get expense breakdown by category and cash flow summary of a user from the database, use data to prompt LLM to generate a financial report for the user via OpenAI's Chat Completions API.

Table 3.4: *Description of Flask routes in Plan Blueprint*

3.6.2.5 Portfolio

This Blueprint handles all portfolio related functions of Fortuna. The url prefix for routes (endpoints) in this Blueprint is **/home/portfolio**. The following table lists down all the routes in **/home/portfolio**, as well as a description of each route's function:

Route	Methods	Description
/	GET	Displays the portfolio selection menu with all available investment portfolios in the database.
/invest	POST	Displays a web page which shows users additional details about a portfolio. Users can purchase portfolios from this web page.
/get_stock_prices	POST	Retrieves real-time stock prices and percentage changes for a list of ticker symbols via yFinance API.
/get_portfolio_returns	POST	Calculates historical cumulative returns for a portfolio for a given time period via yFinance API.
/get_daily_performance	POST	Retrieves the daily return for a given portfolio from database
/add_portfolio	POST	Add a portfolio to the current user’s account
/get_recommended_portfolios	POST	Uses OpenAI’s Chat Completions API to generate personalized portfolio recommendations based on the user's risk profile and financial goals.

Table 3.5: *Description of Flask routes in Portoflio Blueprint*

3.7 Database

3.7.1 PostgreSQL

PostgreSQL is used as the underlying database for Fortuna. It is a powerful, open-source, object-relational database management system (ORDBMS) which is known for being robust, highly extensible, secure and compliant with SQL standards.

The following are the reasons why PostgreSQL is used in Fortuna:

1. PostgreSQL is open-source, which means it is free to use and modify under the PostgreSQL License.
2. Since it is a relational database, it supports tables and join operations, which allows complex queries involving multiple read tables.
3. PostgreSQL allows complex data types including JSON and JSON arrays, which is used by Fortuna to store data in a structured manner.

3.7.2 SQLAlchemy

SQLAlchemy is a powerful Object-Relational Mapping (ORM) library for Python, designed to simplify interaction with databases. It allows developers to manage database operations through Python objects instead of raw SQL statements.

Below are some key features of SQLAlchemy:

1. Converts Python classes into database tables.
2. Abstracts away differences between various database systems (e.g., SQLite, PostgreSQL, MySQL).
3. Provides two main layers: core and ORM. Core refers to the low-level API for executing raw SQL expressions, while ORM represents the high-level interface, mapping Python classes to database tables.

While Flask itself does not directly handle databases, it integrates seamlessly with SQLAlchemy through an extension called Flask-SQLAlchemy. Flask-SQLAlchemy is designed to provide simpler integration of SQLAlchemy with Flask applications. It offers convenient ways to manage sessions, database connections and models in a Flask app. It also adds useful features like automatic configuration from Flask's settings, and simpler management of database contexts.

3.7.3 Schema

As Fortuna uses Flask Blueprints, each Blueprint contains a **models.py** file which defines the database schema for objects related to the Blueprint. In this section, we will examine the schema of the objects defined in each Blueprint.

3.7.3.1 Auth

The **User** class is used to store information about a registered user. The following table contains details about the schema of **User**.

Attribute	**Type**	**Description**
id	Integer	Primary key for uniquely identifying users
username	String(80)	User's login name, must be unique and cannot be null
password_hash	String(128)	Hashed password for user authentication, cannot be null

Table 4.1: *Schema of User Class*

The **Questionaire** class is used to store the user's answers to the investor profiling questionnaire provided to the user during account creation. The following table contains details about the schema of **Questionaire**.

Attribute	**Type**	**Description**
id	Integer	Primary key for uniquely identifying questionnaires
user_id	Integer	Foreign key linking to users table, cannot be null
start_withdrawal	String(10)	When the user plans to start withdrawing funds
spend_funds	String(10)	How the user plans to spend funds
knowledge	String(50)	User's investment knowledge level
risk	String(50)	User's risk tolerance
investments	JSON	Array of user's current or past investment assets (optional)
decision	String(50)	Investment decision preference

Table 4.2: *Schema of Questionaire Class*

The **FinancialGoal** class is used to store the user's financial goals. The following table contains details about the schema of **FinancialGoal**.

Attribute	**Type**	**Description**
id	Integer	Primary key for uniquely identifying financial goals

user_id	Integer	Foreign key linking to users table, cannot be null
name	String(255)	Name of the financial goal, cannot be null
current_amount	Float	Current amount saved toward the goal (defaults to 0)
target_amount	Float	Target amount to reach, cannot be null
target_date	Date	Target completion date, cannot be null
added_on	DateTime	Timestamp when goal was created (defaults to current time)

Table 4.3: *Schema of FinancialGoal Class*

3.7.3.2 Plan

The **Transaction** class is used to store bank transactions uploaded by the user. The following table contains details about the schema of **Transaction**.

Attribute	**Type**	**Description**
id	Integer	Primary key for uniquely identifying transactions
user_id	Integer	Foreign key linking to users table, cannot be null
transaction_date	Date	Date when the transaction occurred, cannot be null
debit_amount	Float	Amount debited in the transaction (can be null)
credit_amount	Float	Amount credited in the transaction (can be

		null)
description	String(255)	Description of the transaction (can be null)
category	String(50)	Classification category for the transaction (can be null)
created_at	DateTime	Timestamp when the transaction record was created (defaults to current time)

Table 4.4: *Schema of Transaction Class*

3.7.3.3 Portfolio

The **Portfolio** class is used to store all available financial portfolios in Fortuna which users can invest in. The following table contains details about the schema of **Portfolio**.

Attribute	**Type**	**Description**
id	Integer	Primary key for uniquely identifying portfolios
name	String(100)	Name of the portfolio, cannot be null

labels	JSON	Array of JSON objects to store labels with structure:
		{
		"text": "label_text",
		"type": "risk" OR "label”
		}

short_description	Text	Brief description of the portfolio (can be null)
long_description	ARRAY(String)	Array of strings representing sentences that form the long description of the portfolio
assets	JSON	Array of JSON objects to store asset

		allocations with structure:
		{
		"name": "asset_category",
		"ticker": "ticker_symbol",
		"allocation": "percentage(%)"
		}

annual_returns	JSON	JSON object to store annual returns with structure:
		{
		"oneYear": "one_year_return",
		"threeYear": "three_year_return",
		"fiveYear": "five_year_return"
		}

daily_return	Float	Portfolio's daily return calculated as (current price - latest close)/latest close

Table 4.5: *Schema of Portfolio Class*

The **Portfolio** class also includes a method **update_daily_return(),** which is called by a background thread during startup to update the Portfolio's **daily_return** attribute using current asset prices obtained via yFinance API. This background thread is run once every 60 seconds.

The **UserPortfolio** class is used to store financial portfolios which users have invested in. The following table contains details about the schema of **UserPortfolio**.

Attribute	**Type**	**Description**
id	Integer	Primary key for uniquely identifying user portfolios
user_id	Integer	Foreign key linking to users table, cannot be null

portfolio_id	Integer	Foreign key linking to portfolios table, cannot be null
added_on	DateTime	Timestamp when the user portfolio was created (defaults to current time)
value	Float	Current value of the portfolio in USD (can be null)

assets	JSON	Array of JSON objects to store fractional units of each asset with structure:
		{
		"ticker": "ticker_symbol",
		"units": "fractional_value"
		}

portfolio	**Portfolio**	Refers to the **Portfolio** object associated with the **UserPortfolio**, created via Flask-SQLAlchemy. This relationship allows direct access to **Portfolio** attributes via **UserPortfolio**.

Table 4.6: *Schema of UserPortfolio Class*

The **UserPortfolio** class also includes a method **update_value()** which is called by a background thread during startup to update the UserPortfolio's **value** attribute using current asset prices obtained via yFinance API. This background thread is run once every 60 seconds.

3.8 External APIs

3.8.1 yFinance

The yFinance API is a Python library that provides access to historical market data, real-time financial data and other stock market-related information from Yahoo Finance. As yFinance is completely free, well-documented, and provides financial data granularity of up to 1 minute intervals, it has become one of the most popular financial libraries among non-commercial developers looking to access real-time financial data for virtually zero cost.

In the context of Fortuna, the yFinance API is used in the Portfolio Blueprint to fetch live financial data in the following routes:

1. **get_stock_prices**
   Given a list of asset ticker symbols, yFinance is used to get the latest closing price and the last minute's closing price of each asset in order to calculate how much the asset price has changed every minute.

2. **get_portfolio_returns**
   Given a portfolio, yFinance is used to fetch historical asset prices of the portfolio for the latest 6 months, 1 year and 5 years in order to calculate the portfolio's cumulative daily return throughout each time period.

3. **add_portfolio**
   When adding a new portfolio to the user's account, yFinance is used to fetch the current asset price for each asset in the portfolio in order to determine how many units of each asset does the investor own. This information will be used to determine the value of the user's portfolio.

3.8.2 OpenAI

The OpenAI API is a service provided by OpenAI that allows developers to access advanced AI models through simple API calls. It enables the integration of OpenAI's powerful language and image models such as GPT-4, GPT-3.5 and DALL-E into various applications or products.

Fortuna uses 3 different types of APIs from OpenAI, which is the Chat Completions API, Assistants API and Batch API. The Python version of these 3 APIs were used.

3.8.2.1 Chat Completions API

Chat Completions is OpenAI's legacy API which enables developers to create chat-based interactions using OpenAI's GPT models. Through this API, OpenAI provides support for various features such as image inputs, web search, file search, function calling, structured outputs, and many more.

In the case of Fortuna, the Chat Completions API was used in the **/generate_financial_report** and **/get_recommended_portfolios** routes. We will take a more detailed look on how the API was used for each route.

In **/generate_financial_report**, the goal was to get GPT to use the user's monthly cash flows and expenses breakdown by category as input, and generate a summary of the user's monthly expenses in the form of a JSON object as the output. During the API call, the **response_format** parameter is set to ensure that GPT generates a valid JSON object as the output. By ensuring that the API call always returns a valid JSON object, we increase the robustness of our application. Furthermore, setting the **temperature** parameter to be 0 ensures that the output is deterministic, which means inputting the same user information will always give the same output, further increasing the application's robustness.

The below table contains the configuration parameters used for the API. Note that the user prompt is not fixed as it depends on the user's actual financial statistics for the month.

route	/generate_financial_report
model	'gpt-4o-mini'
temperature	0
response_format	{"type": "json_object"}

system prompt	'''
	You are a financial assistant specializing in analyzing financial behavior. Your goal is to summarize a user's spending habits based on the financial data provided by the user,
	while providing helpful financial advice for the user to save money. Answer like you are talking to the user. Use the provided numbers and statistics in your answer.
	Return your answer according to the following JSON structure and definitions:
	{
	'summary': str, //Comment on the user's spending habits in 2-3 sentences
	'spending':[{'name': str, //name of category
	'percentage': int, //percentage spent on the category
	'description': str //analysis of spending behaviour
	}] //List of top 3 categories that the user spent their money on.
	'recommendations':[{'title': str, //short title for the recommendation
	'description': str, //recommendation for the user to save money, in about 2-3 sentences
	'icon': str //Bootstrap icon class that is relevant to the header }]
	//List of 3 objects

	} '''
Example user prompt	''' #Expenses breakdown [{'category': 'food', 'dollar_amount': 12.9, 'percentage': 7.65}, {'category': 'others', 'dollar_amount': 103.34, 'percentage': 61.31}, {'category': 'shopping', 'dollar_amount': 49.73, 'percentage': 29.5}, {'category': 'transportation', 'dollar_amount': 2.59, 'percentage': 1.54}] #Monthly Cashflow {'income': 0, 'expenses': 168.56} '''

Table 5.1.1: *Chat Completions Configuration for generate_financial_report*

Below is a sample of the JSON object to be returned by the API call

```
{
    "recommendations": [
        {
            "description": "Take some time to categorize your 'others' expenses more specifically. By identifying what these expenses are, you can find areas to cut back and save money.",
            "icon": "bi bi-eye",
            "title": "Track 'Others' Spending"
        },
        {
            "description": "Establish a monthly budget for shopping and stick to it. This will help you prioritize your needs over wants and reduce impulse purchases.",
            "icon": "bi bi-cart",
            "title": "Set a Shopping Budget"
        },
        {
```

```
            "description": "Consider meal prepping to save on food costs. Planning your meals can help you avoid last-minute takeout and reduce your overall food expenses.",
            "icon": "bi bi-knife-fork",
            "title": "Explore Meal Prep"
        }
    ],
    "spending": [
        {
            "description": "The majority of your spending is categorized as 'others', which could include miscellaneous expenses. It's important to track these items closely to see if there are unnecessary purchases that can be reduced.",
            "name": "Others",
            "percentage": 61
        },
        {
            "description": "You are spending nearly a third of your budget on shopping. Consider setting a monthly limit for shopping to help control this expense.",
            "name": "Shopping",
            "percentage": 29
        },
        {
            "description": "Your food expenses are relatively low compared to other categories. However, it's always good to look for deals or meal prep options to save even more.",
            "name": "Food",
            "percentage": 7
        }
    ],
    "summary": "Your spending habits indicate a significant portion of your expenses is allocated to 'others', which comprises over 61% of your total spending. This suggests that you may want to review what falls under this category to identify potential areas for savings."
}
```

For the **/get_recommended_portfolios** route, the API was used to prompt GPT to generate a list of recommended portfolios, given the user's risk profile and financial goals as input data. The user data was passed into the model as the initial user prompt, whereas portfolio data was formatted as a JSON string and inserted into the system prompt. A sample of the JSON output is also included in the system prompt. This technique is known as One-Shot Prompting, which has been shown to be able to improve the performance of LLMs (Brown et al., 2020).

For this API call, OpenAI's o3-mini model was used with **reasoning_effort** set to **'high'** as the task of portfolio recommendation requires more 'thinking' to be done by the LLM, compared to the previous task of summarizing monthly expenses. Again, the **response_format** parameter was set to **'json_object'** to ensure that the model's output is as deterministic as possible.

The below table contains the configuration parameters used for the API. Note that portfolios_str and the user prompt is not fixed as it depends on the actual application and user data in the database during the API call.

route	**/get_recommended_portfolios**
model	'o3-mini'
reasoning_effort	'high'
response_format	{"type": "json_object"}
Example portfolios_str	''' [{ "id": 1, "name": "Total Stock Market Portfolio", "labels": [{ "text": "High Risk",

```
      "type": "risk"
    },
    {
      "text": "Traditional",
      "type": "label"
    }
  ],
  "short_description": "The Total Stock Market Portfolio is the simplest and most cost-effective way for anyone to invest their money without overthinking it.",
  "long_description": [
    "Advocates simplicity by investing in a single total market stock fund and largely ignoring market fluctuations.",
    "Young investors are encouraged to focus on earning and saving rather than fine-tuning portfolios, embracing an aggressive 100% stock allocation.",
    "This approach is ideal for those seeking financial independence through a straightforward, high-risk, high-reward strategy, with the potential to add bonds decades later for stability."
  ],
  "assets": [
    {
      "name": "Total Stock Market",
      "allocation": 100,
      "ticker": "VTI"
    }
  ],
  "daily_return": 0.00039468993684085136,
  "annual_returns": {
    "oneYear": 20,
    "threeYear": 3,
    "fiveYear": 9
  }
```

```
  },
  {
    "id": 3,
    "name": "Core Four Portfolio",
    "labels": [
      {
        "text": "High Risk",
        "type": "risk"
      },
      {
        "text": "Traditional",
        "type": "label"
      }
    ],
    "short_description": "The Core Four Portfolio by Rick Ferri invests in four fundamentally unique asset types that are productive both on their own and as a group.",
    "long_description": [
      "Includes four assets that serve distinct economic purposes, generate regular cash flow, and provide diversification through low historic correlations.",
      "All asset classes are accessible via low-cost index funds or ETFs, ensuring simplicity and affordability.",
      "Balances individual asset performance with overall diversification, offering a solid and straightforward financial foundation."
    ],
    "assets": [
      {
        "name": "Domestic Stocks",
        "allocation": 48,
        "ticker": "VOO"
      },
      {
```

	"name": "International Stocks",
	"allocation": 24,
	"ticker": "VEU"
	},
	{
	"name": "Intermediate Bonds",
	"allocation": 20,
	"ticker": "BIV"
	},
	{
	"name": "REITs",
	"allocation": 8,
	"ticker": "VNQ"
	}
	],
	"daily_return": -0.0018925696242862787,
	"annual_returns": {
	"oneYear": 11,
	"threeYear": 0,
	"fiveYear": 5
	}
	}
	]
	'''

system prompt	''' Use the provided risk profile and financial goals to determine the most suitable financial portfolios for the user. Consider factors such as risk tolerance, financial goals, and time horizon. # Steps 1. **Analyze Risk Profile and Financial Goals**: Examine the user's risk tolerance, investment goals, and time horizon.

	2. **Portfolio Selection**: Match the user's profile with the given financial portfolios that align with their risk level and objectives. 3. **Justification**: Provide a rationale for the recommended portfolios, explaining how they align with the user's risk profile and financial goals. # Output instructions Provide the recommended financial portfolios in JSON format, including: - User's risk profile summary in 2-3 short sentences - Recommended portfolios - 2-3 short justifications for each recommendation, with reference to the user's risk profile and financial goals # Portfolios\n'" + **portfolios_str** + "'\n# Example Output Format { "summary": ["Based on your investment knowledge and willingness to take higher than average risks, you are comfortable with volatility over a medium-term horizon.", "Your near-term objective of saving $2000 by mid-2025 for a new PC, combined with the longer-term goal of gathering $50000 by 2030 for a business startup, suggests a need for portfolios that balance growth potential with some risk mitigation."], "portfolios": [{ "name": "Core Four Portfolio", "reasons": ["This portfolio offers a diversified mix with a heavy equity allocation (domestic and international stocks) balanced by a meaningful bond component, which can help cushion short-term downturns while pursuing growth.", "Its inclusion of a variety of asset classes aligns with your current holdings in stocks, bonds, and international securities, making it a

	well-rounded option for both near-term liquidity and longer-term appreciation.",
	"The mix of 72% equities and 28% fixed income/reits provides a balance that supports your high risk tolerance while addressing your need for funds in 3-5 years."
	]
	},
	{
	"name": "Total Stock Market Portfolio",
	"reasons": [
	"This portfolio is fully exposed to equities, maximizing the potential for high returns, which suits your willingness to take on higher than average risk.",
	"It is especially geared toward long-term growth, making it ideal for your goal of saving $50000 by 2030 to start a business.",
	"Given that you stated you would remain undistracted during market downturns, the aggressive 100% equity approach can work well for the portion of your funds designated for long-term investing."
	]
	}
	]
	}
	'''

Example user prompt	''' # User risk profile 1. The user plans to withdraw money from their investments in 3-5 years 2. Once the user begins withdrawing funds from their investments, they plan to spend all of the funds in 6-10 years 3. The user describes their knowledge of investments as Good 4. When investing, the user is willing to take higher than average risks expecting to earn higher than average returns

	5. The user owned or currently owns: ['bonds', 'stocks', 'international securities'] 6. In a hypothetical scenario where the overall stock market lost 25% of its value and an individual stock investment that the user owns also lost 25% of its value, the user will do nothing # User financial goals 1. The user wants to save $2000.0 by 1 Jun 2025 to Buy a PC 2. The user wants to save $50000.0 by 1 Jan 2030 to Start a business '''

Table 5.1.2: *Chat Completions Configuration for get_recommended_portfolios*

Below is a sample of the JSON object to be returned by the API call

```
{
    "portfolios": [
        {
            "name": "Total Stock Market Portfolio",
            "reasons": [
                "Its 100% equity allocation maximizes growth potential, which aligns with your preference for higher than average risks to achieve higher returns.",
                "The aggressive approach supports your long-term goal of saving $50,000 by 2030 for a business startup.",
                "Since you're comfortable with market volatility (as evidenced by your decision to do nothing in a 25% market drop), this portfolio's fluctuations should be manageable within your investment strategy."
            ]
        },
        {
            "name": "Core Four Portfolio",
            "reasons": [
                "This portfolio provides a diversified mix of domestic and international stocks along with bonds and REITs, offering potential growth while cushioning against short-term volatility.",
```

```
                "Its balanced yet aggressive allocation complements your current
holdings in bonds, stocks, and international securities.",
                "The diversified approach makes it a suitable option for your 3-5
year withdrawal timeline, helping mitigate risk while still targeting higher
returns."
            ]
        }
    ],
    "summary": [
        "You have a strong foundation of investment knowledge and are comfortable
with high risk, even in the face of market downturns.",
        "Although you plan to withdraw funds in 3-5 years for near-term goals and
then spend those funds over 6-10 years, your willingness to take higher risks
positions you for aggressive growth to meet both your short-term and long-term
financial objectives."
    ]
}
```

3.8.2.2 Assistants API

OpenAI's Assistants API was designed to help developers create AI assistants capable of maintaining context over extended interactions with the ability to use tools such as code interpreters, function calling and file search. The main difference between Assistants and Chat Completions API is that Assistants operate on Thread objects, which simplify AI application development by automatically storing message history and truncating it when the conversation gets too long for the model's context length. With Assistants API, the developer only has to create a Thread once and simply append Messages to it as the users reply to the LLM.

The below table provides explanations for Objects used with the Assistants API.

Object	**Definition**
Assistant	Purpose-built AI that uses OpenAI's models and calls tools

Thread	A conversation session between an Assistant and a user. Threads store Messages and automatically handle truncation to fit content into a model's context.
Message	A message created by an Assistant or a user. Messages can include text, images, and other files. Messages stored as a list on the Thread.
Run	An invocation of an Assistant on a Thread. The Assistant uses its configuration and the Thread's Messages to perform tasks by calling models and tools. As part of a Run, the Assistant appends Messages to the Thread.
Run Step	A detailed list of steps the Assistant took as part of a Run. An Assistant can call tools or create Messages during its run. Examining Run Steps allows you to introspect how the Assistant is getting to its final results.

Table 5.2.1: *Definition of Objects used with Assistants API*

Fortuna's chat bot is built using an Assistant. The table below describes the configurations used to set up Fortuna's Assistant.

Model	gpt-4o
System instructions	You are a **financial advisor assistant**, designed to provide users with **personalized financial advice** based on their **risk profile, past transactions, and financial goals**. Your responses should always be **clear, structured, and formatted neatly** to enhance readability. ## **Response Formatting Guidelines:** - Use **bold headings** for key sections. - Separate different ideas using **newlines**.

	- Use **bullet points (`-`)** for listing multiple recommendations. - Use **numbered lists (`1., 2., 3.`)** for step-by-step instructions. - Use **tables where necessary** to present financial data. - **Quotes (`> text`)** can be used for important insights or disclaimers. --- ## **How You Should Respond:** 1. **Assess the User's Financial Profile:** - Consider their **risk tolerance** (e.g., conservative, moderate, aggressive). - Review their **past transactions** to identify spending patterns. - Align advice with their **short-term and long-term financial goals**. 2. **Provide Personalized Financial Advice:** - Offer **investment recommendations** (e.g., stocks, bonds, ETFs, real estate). - Suggest **budgeting strategies** to improve financial health. - Identify **potential financial risks** and mitigation strategies. 3. **Explain Financial Concepts Clearly:** - Use **simple language** for beginners. - Provide **detailed insights** for experienced investors. - Include examples where necessary. 4. **Ensure Compliance and Risk Warnings:** - Clearly **state that you are an AI assistant and not a licensed

	financial advisor**. - Include appropriate **disclaimers** for investment risks. - Advise users to consult a **certified financial expert** for major decisions. --- ## **Example Outputs:** ### ** 💡 Investment Strategy for a Moderate Risk Profile** **Based on your risk tolerance and financial goals, here's a suggested portfolio allocation:** | Asset Class | Allocation (%) | |---|---| | Stocks (ETFs) | 50% | | Bonds | 30% | | Real Estate | 10% | | Cash & Others | 10% | 📌 **Key Insights:** - A mix of **stocks and bonds** balances growth and stability. - Consider **low-cost ETFs** for diversification. - Keep **10% in cash** for liquidity and emergencies. --- ### **📊 Budgeting Tips Based on Your Spending Pattern** Your past transactions indicate **high discretionary spending**. Here are some tailored recommendations:

	1. **Set a Monthly Budget:** Limit discretionary expenses to **20% of income**. 2. **Automate Savings:** Transfer **15% of income** to a savings or investment account. 3. **Cut Unnecessary Expenses:** Identify recurring subscriptions you don’t use. 4. **Use a Budgeting App:** Track spending habits in real time. --- ### **⚠️ Disclaimer:** > I am an AI assistant providing financial insights based on the information you provide. My recommendations should not be considered professional financial advice. Please consult a **certified financial planner** before making investment decisions.
Tools	Functions
Functions	get_portfolios get_user_portfolios get_risk_profile get_financial_goals get_transactions get_current_date
Response format	Text
Temperature	1.00
Top P	1.00

Table 5.2.2: *Assistant Configuration for Fortuna’s Chatbot*

Tool calling is one of the key features of Assistants. It allows Assistants to intelligently infer from the user prompt when to use custom tools that are created by the developer. There are 3 different types of tools that are supported by the Assistants API: Functions, File Search and Code Interpreter. Function calling lets developers describe custom application functions or external APIs to the Assistant. This allows the Assistant to intelligently call those functions by outputting a JSON object containing relevant arguments. Meanwhile, File Search enables Assistants with knowledge from files that are uploaded by the developer or the users to OpenAI's servers. Once a file is uploaded, the Assistant automatically decides when to retrieve content based on user requests. Lastly, Code Interpreter enables Assistants to write and run code. This tool can process files with diverse data and formatting, and generate files such as graphs.

In the case of Fortuna, function calling was enabled to allow Fortuna's Assistant to have access to portfolio data in the database, as well as user data such as user portfolios, financial goals, risk profile and bank transactions. These functions allow Fortuna to generate financial advice based on relevant and factually correct data, while also greatly reducing the chances of hallucinating. The functions that Fortuna's Assistant is able to access can be found in **utils.py** under the Chatbot blueprint folder. The following table contains the list of functions available in **utils.py**.

Function	**Description**
get_portfolios(p_name)	Retrieves portfolio information from the database. If p_name is 'All', returns all portfolios; otherwise, returns a specific portfolio matching the name. Returns data as a string representation of a dictionary or list of dictionaries.
get_user_portfolios(user_id)	Retrieves all portfolios belonging to a specific user. Returns the portfolios as a string representation of a list of dictionaries, or an error message if no portfolios are found.

get_risk_profile(user_id)	Retrieves a user's risk profile based on their questionnaire responses. Returns a formatted string containing information about withdrawal plans, investment knowledge, risk tolerance, and other preferences.
get_financial_goals(user_id)	Retrieves all financial goals belonging to a specific user. Returns the goals as a string representation of a list of dictionaries, or an error message if no goals are found.
get_current_date()	Returns the current date formatted as "DD Month YYYY" (e.g., "18 March 2025").
get_transactions(month, year, user_id)	Retrieves all transactions for a specific user within a given month and year. Returns the transactions as a string representation of a list of dictionaries, or an error message if no transactions are found.

Table 5.2.3: *Description of Functions in utils.py*

While Fortuna could have used the File Search tool for accessing application and user data, this approach was not taken due to cost concerns. Using the File Search tool requires data to be uploaded to OpenAI's vector database, and as of writing, files stored on their servers are charged at a rate of $0.10 per GB per day.

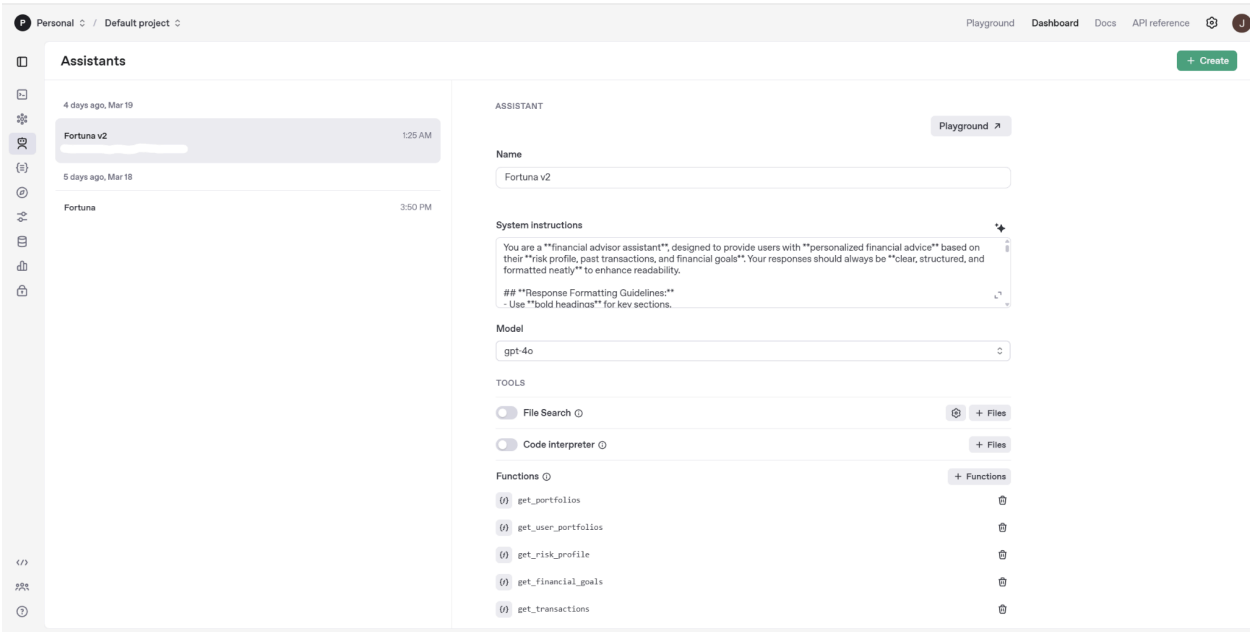

Figure 5: *OpenAI Assistants Dashboard (source: platform.openai.com/assistants)*

OpenAI provides a web interface for developers to create new Assistants and configure their Assistant settings. Once an Assistant has been set up, it can be reused as many times as needed by referencing the unique **assistant_id** of each Assistant.

To use an Assistant, a Thread object must first be created. A Thread essentially represents a chat session that the user has with the Assistant. In the case of Fortuna, a new Thread is created each time the user loads the chatbot interface. After creation of the Thread object, a new Message object is created with the message inputted by the user. The newly created Message is then appended to the Thread and is sent to OpenAI's servers for processing via a Run object. A Run is created by specifying the **thread_id** and **assistant_id** parameters so that the API knows which Assistant to use for processing the Thread. The API provides a function called **create_and_poll** for polling the status of the Run. If the Run is finished with a status of 'completed', the Assistant message can be directly displayed to the user. Else, if the Run finishes with a status of 'requires_action', this means the Assistant has decided to use a Tool (function calling). The application backend will have to process the Tool and provide the Tool outputs to the Assistant via a new Run object so that the Assistant can continue

processing the user prompt. This process is repeated iteratively until the last Run ends with a status of ‘completed’ or ‘failed’.

In summary, the Assistants API provides a convenient interface for implementing the chatbot features of Fortuna, all while allowing the chatbot to access custom application data from the database.

3.8.2.3 Batch API

The Batch API is provided by OpenAI to allow developers to send asynchronous groups of requests with 50% lower costs, a separate pool of significantly higher rate limits, and a clear 24-hour turnaround time. The service is ideal for processing jobs that don't require immediate responses. In the case of Fortuna, the Batch API was used to classify user bank transactions in the **/submit_classification_batch** route within the Plan blueprint.

To use the Batch API, the first step would be to prepare the batch file. Batches start with a .jsonl file where each line contains the details of an individual request to the API. In the case of Fortuna, the endpoint used was the Chat Completions API. The below table contains an example of a JSON object in one line of the .jsonl file.

Field	**Value**
custom_id	f"task-{**transaction.id**}"
method	“POST”
url	"/v1/chat/completions"

body	{
	"model": "gpt-4o",
	"temperature": 0,
	"response_format": {
	"type": "json_object"
	},

	"messages":[
	{
	"role": "system",
	"content": **categorize_system_prompt**
	},
	{
	"role": "user",
	"content": **description**
	}
	]
	}

Table 5.3: *Sample of JSON Object in a Batch File*

transaction.id refers to the id field of the Transaction object queried from the database. The **custom_id** field is used for mapping Batch responses back to the input requests, as each line in the Batch output may not be returned in the same order as the input request.

categorize_system_prompt is the system prompt to be sent to the model. Below is the system prompt used to categorize the model:

```
'''Classify the following bank transaction into one of these categories: [housing,
food, transportation, shopping, entertainment, utilities, others]
If you cannot determine the class, classify as: others
Output a json object containing the following information:
{
    category: string // Category based on the transaction description,
}
'''
```

Lastly, **description** refers to the transaction description of each Transaction object to be classified.

After the Batch file has been created, the next step is to upload the file to OpenAI's servers and create a Batch job. Once a Batch job has been created, the status of the Batch job can be continuously polled with **client.batches.retrieve()** until the Batch has reached one of the following terminal statuses: 'completed', 'failed', 'expired', 'cancelled'. Since the Batch job is running asynchronously in the backend, the web page remains responsive to other user events that may occur.

The following is an example of the JSON object to be received from the Batch API if the Batch job completed with a status of 'completed'.


```
{'id': 'batch_req_67d53dcac934819098bacd38ebeda1a7',
 'custom_id': 'task-0',
 'response': {'status_code': 200,
  'request_id': '93556d252ce0198999773cebaad4a7d9',
  'body': {'id': 'chatcmpl-BBHPOhItFvTiKvSynVq7mbUR7Mlol',
   'object': 'chat.completion',
   'created': 1742028226,
   'model': 'gpt-4o-mini-2024-07-18',
   'choices': [{'index': 0,
     'message': {'role': 'assistant',
      'content': '{\n    "categories": "others"\n}',
      'refusal': None,
      'annotations': []},
     'logprobs': None,
     'finish_reason': 'stop'}],
   'usage': {'prompt_tokens': 80,
    'completion_tokens': 10,
    'total_tokens': 90,
    'prompt_tokens_details': {'cached_tokens': 0, 'audio_tokens': 0},
    'completion_tokens_details': {'reasoning_tokens': 0,
     'audio_tokens': 0,
     'accepted_prediction_tokens': 0,
     'rejected_prediction_tokens': 0}},
   'service_tier': 'default',
```

```
  'system_fingerprint': 'fp_06737a9306'}},
 'error': None}
```

In summary, the Batch API serves as a lower cost option of accessing OpenAI's APIs when large amounts of data has to be sent and responses are not required immediately. This approach is used to classify bank transactions because the user may upload a large number of transactions at once, which may lead to rate limits and higher costs if directly calling the Chat Completions API for each transaction.

3.9 Portfolios

The following table contains information regarding all the investment portfolios available in Fortuna. These portfolios were taken from the PortfolioCharts website (Portfolio Charts, n.d.).

Name	**Risk level**	**Returns**	**Asset class (Ticker): Allocation**
Total Stock Market	High	1Y: 20% 3Y: 3% 5Y: 9%	Total Stock Market (VTI): 100%
Weird Portfolio	Low	1Y: 5% 3Y: -4% 5Y: 1%	Small Cap Value (VBR): 20% International Small Cap Blend (VSS): 20% Long Term Bonds (BLV): 20% REITS (VNQ): 20% Gold (IAUM): 20%
Core Four Portfolio	High	1Y: 11% 3Y: 0% 5Y: 5%	Domestic stocks (VOO): 48% International Stocks (VEU): 24% Intermediate Bonds (BIV): 20% REITs (VNQ): 8%
Golden Butterfly Portfolio	Low	1Y: 9% 3Y: -1% 5Y: 2%	Large Cap Blend (VOO): 20% Small Cap Value (VBR): 20% Long Term Bonds(BLV): 20% Short Term Bonds (BSV): 20% Gold (IAUM): 20%
Pin Wheel Portfolio	Medium	1Y: 8% 3Y: -1% 5Y: 2%	Large Cap Blend (VOO): 15% Small Cap Value (VBR): 10% International Stocks (VEU): 15% Emerging Markets (VWO): 10% Intermediate Bonds (BIV): 15%

			Cash (VGSH): 10% REITS (VNQ): 20% Gold (IAUM): 20%
No-Brainer	Medium	1Y: 9% 3Y: 0% 5Y: 4%	Large Cap Blend (VOO): 25% Small Cap Blend (VB): 25% International Stocks (VEU): 25% Short Term Bonds (BSV): 25%

Table 6: *Summary of Portfolios in Fortuna*

3.10 Investor Profiling Questionnaire

The investor profiling questionnaire used in Fortuna was adapted from Charles Schwab's Investor Profiling Questionnaire (Charles Schwab, n.d.).

The following table shows the mapping between each question to each attribute of the **Questionaire** object used to store the user's answers.

Question	**Answers**	**Questionaire attribute**
I plan to begin withdrawing money from my investments in	1. Less than 3 years 2. 3–5 years 3. 6–10 years 4. 11 years or more	start_withdrawal
Once I begin withdrawing funds from my investments, I plan to spend all of the funds in:	1. Less than 2 years 2. 2–5 years 3. 6–10 years 4. 11 years or more	spend_funds
I would describe my knowledge of investments as:	1. None 2. Limited 3. Good 4. Extensive	knowledge
What amount of financial risk are you willing to take when you invest?	1. Take lower than average risks expecting to earn lower than average returns 2. Take average risks expecting to earn average returns	risk

	3. Take above average risks expecting to earn above average returns	
Select the investments you currently own or have owned:	1. Money market fund or cash investments 2. Bonds and/or bond funds 3. Stocks and/or stock funds 4. International securities and/or international funds	investments
Consider this scenario: Imagine that in the past three months, the overall stock market lost 25% of its value. An individual stock investment you own also lost 25% of its value. What would you do?	1. Sell all of my shares 2. Sell some of my shares 3. Do nothing 4. Buy more shares	decision

Table 7: *Details of Fortuna's Investor Profiling Questionnaire*

3.11 UI/UX

In this section we will take a look at the UI of Fortuna.

3.11.1 Log in

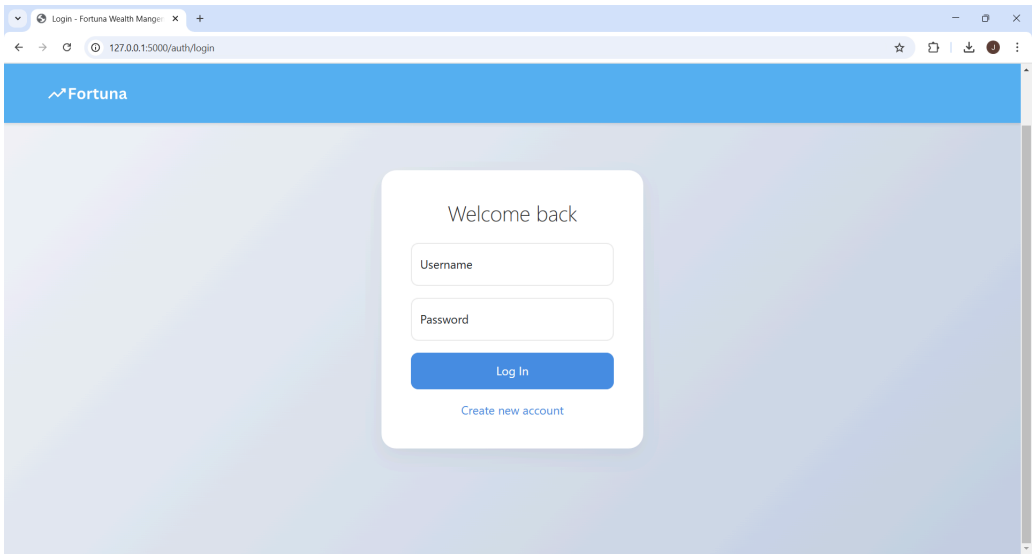

Figure 6.1: *Fortuna Log In Page*

When the user starts up the app, they will be redirected to the login page. The user has the option to log into the app with their registered username and password.

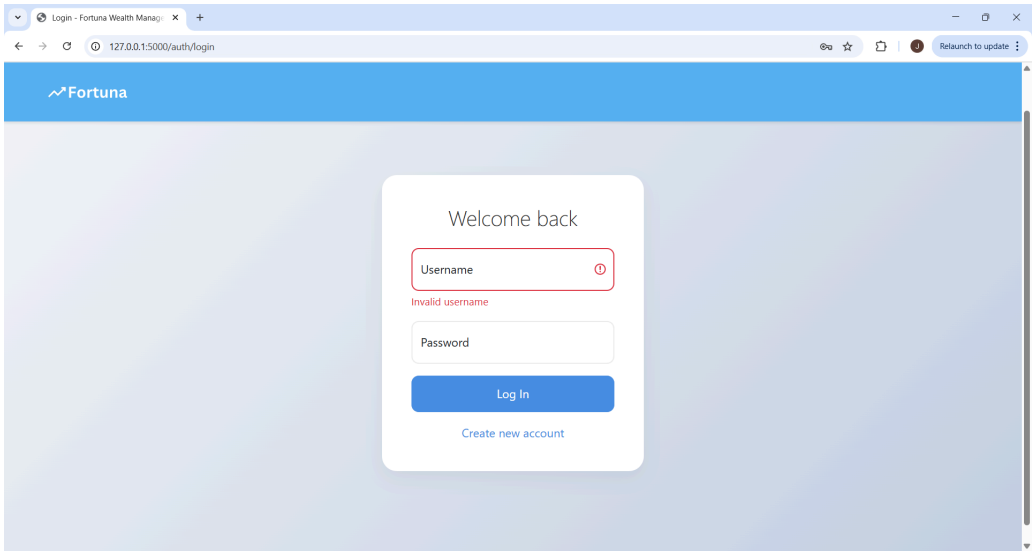

Figure 6.2.1: *Fortuna Invalid Username Error Message*

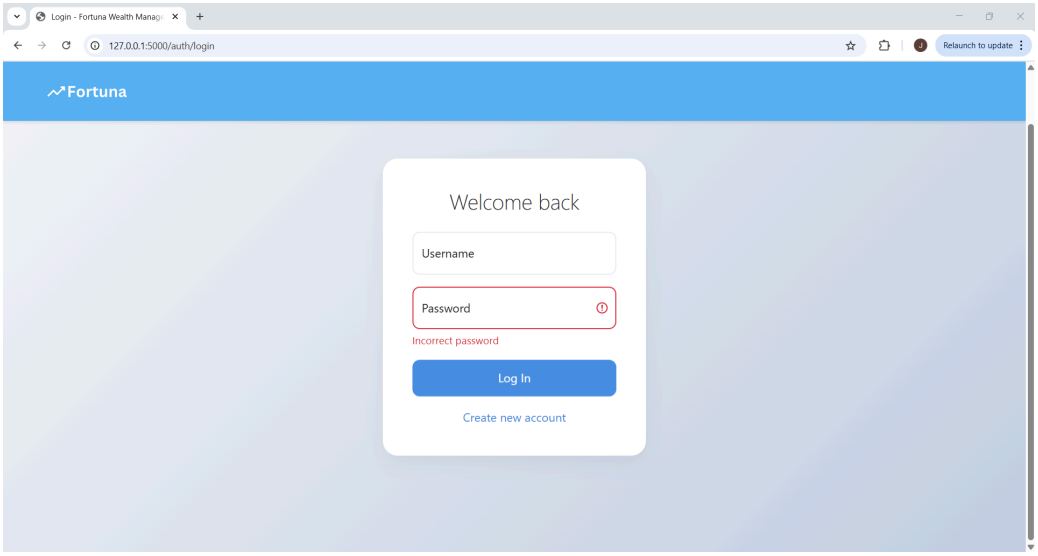

Figure 6.2.2: *Fortuna Incorrect Password Error Message*

If the user inputs the wrong username or password, the corresponding error message will be displayed to warn the user.

3.11.2 Create New Account

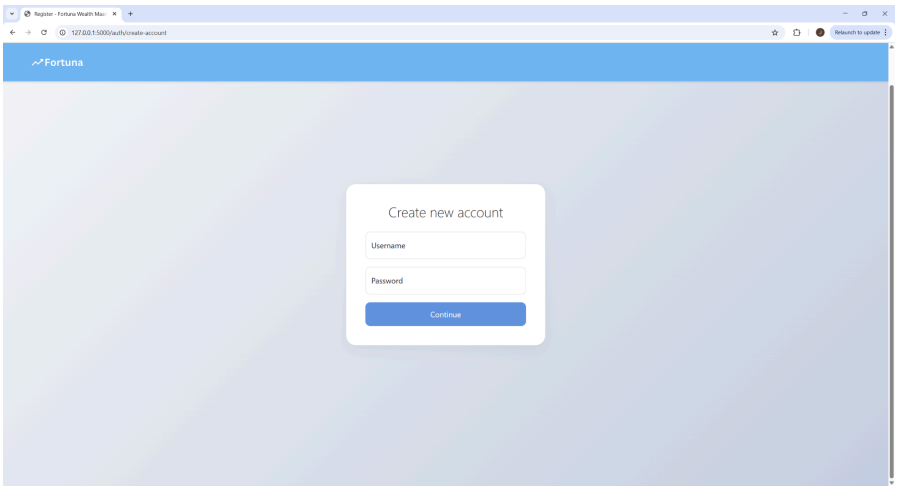

Figure 6.3.1: *Fortuna Create New Account Page*

If the user clicks on the ‘Create new account’ link in the login page, they will be brought to the web page for creating a new account.

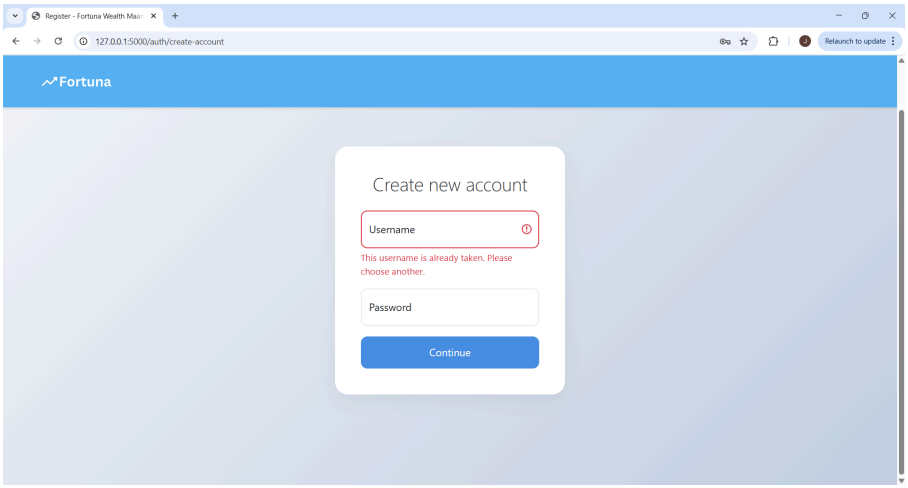

Figure 6.3.2: *Fortuna Username Taken Error Message*

If the user tries to create an account with a username that already exists in the database, the corresponding error message will be displayed to warn the user.

3.11.3 Questionnaire

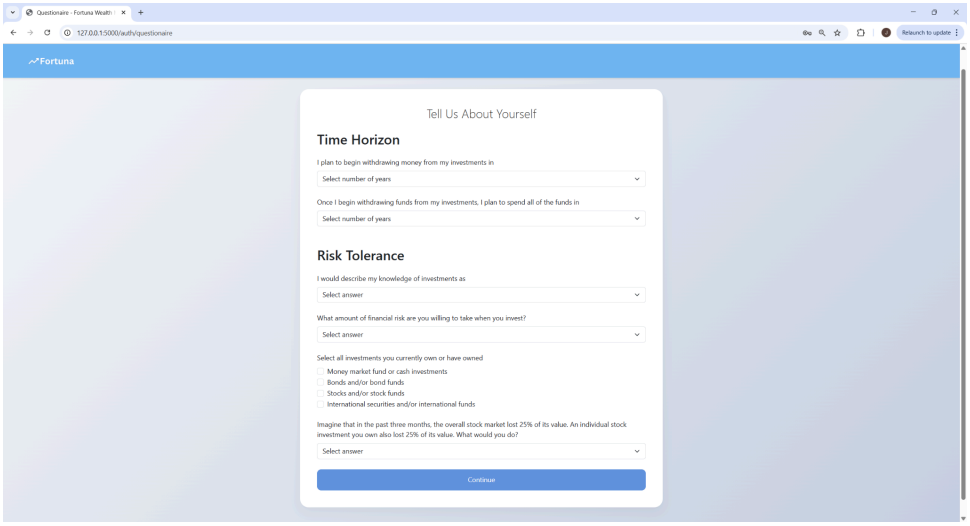

Figure 6.4.1: *Fortuna Questionnaire Page*

the user successfully created an account with a unique username, the user will be brought to the investor profiling questionnaire web page. The user will be required to select an option for every question (except the question on investments owned).

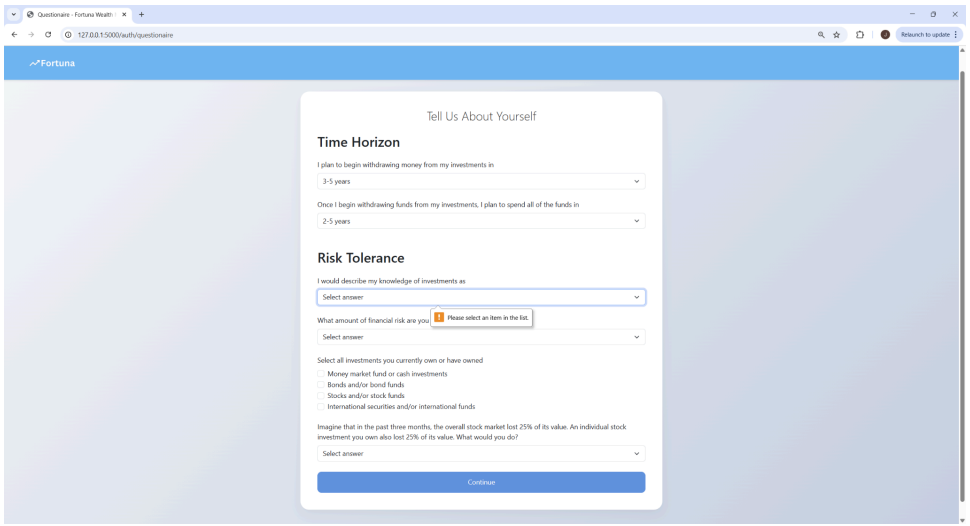

Figure 6.4.2: *Fortuna Questionnaire Incomplete Warning*

the user tries to continue without answering all the required fields, the system will display a warning message.

3.11.4 Financial Goals

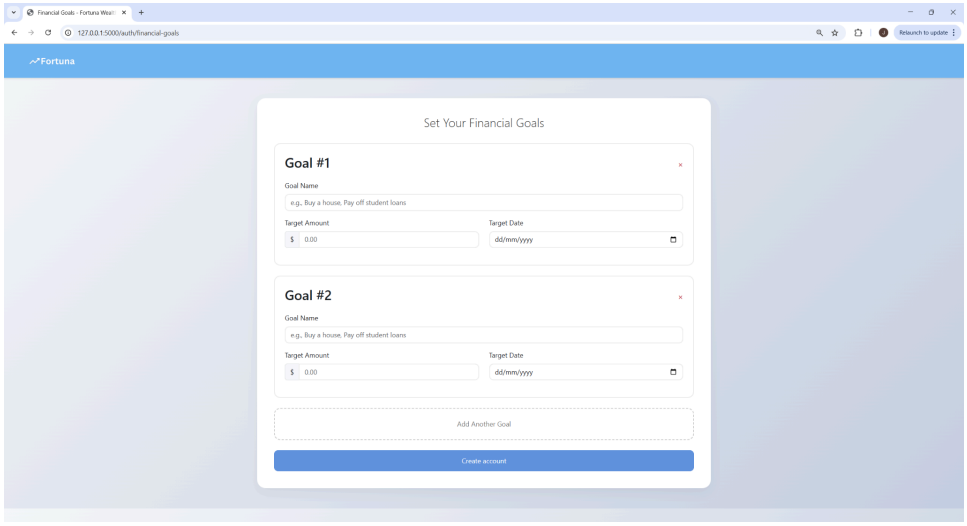

Figure 6.5.1: *Fortuna Financial Goals Page*

As the last step of the account registration process, the user will be asked to enter their financial goals. The user can add as many financial goals as they want. Upon adding a financial goal, the user must enter all required fields for the goal, which are goal name, target amount and target date.

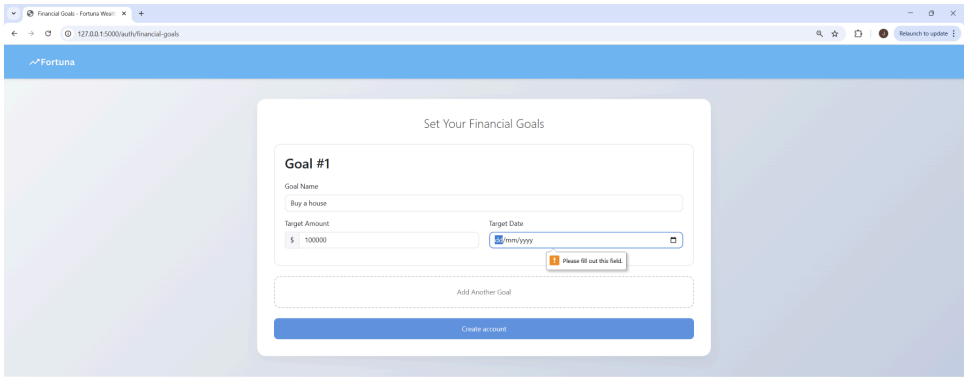

Figure 6.5.2: *Fortuna Financial Goals Incomplete Warning*

If the user tries to create an account without inputting data in all the required fields, the system will display a warning message.

3.11.5 Dashboard

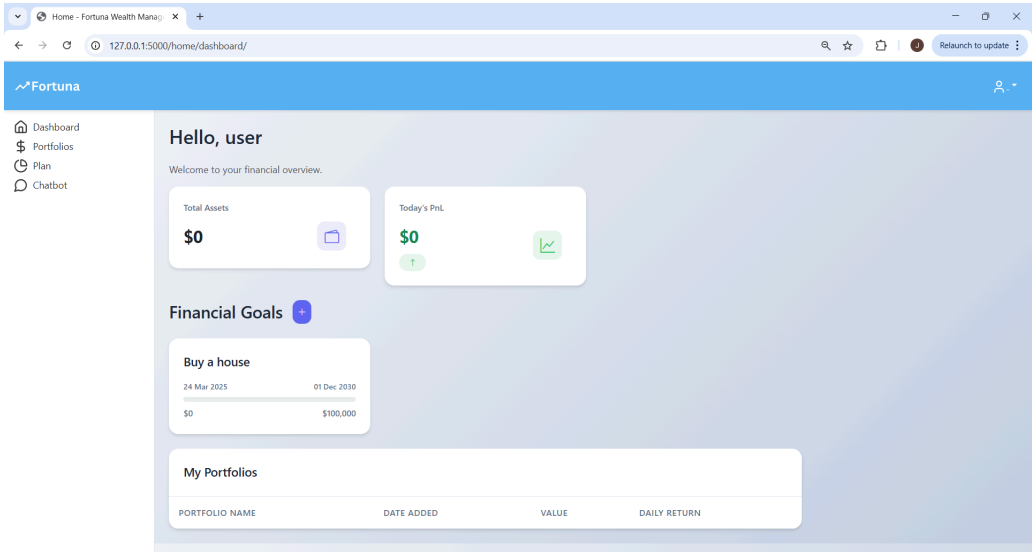

Figure 6.6.1: *Fortuna Dashboard, No Portfolios*

Upon completing the account registration process, the user will be brought to Fortuna's main dashboard. The dashboard is designed to give the user a quick overview of their financial status. On the left of the dashboard is a simple navigation menu that allows the user to navigate to different sections of Fortuna.

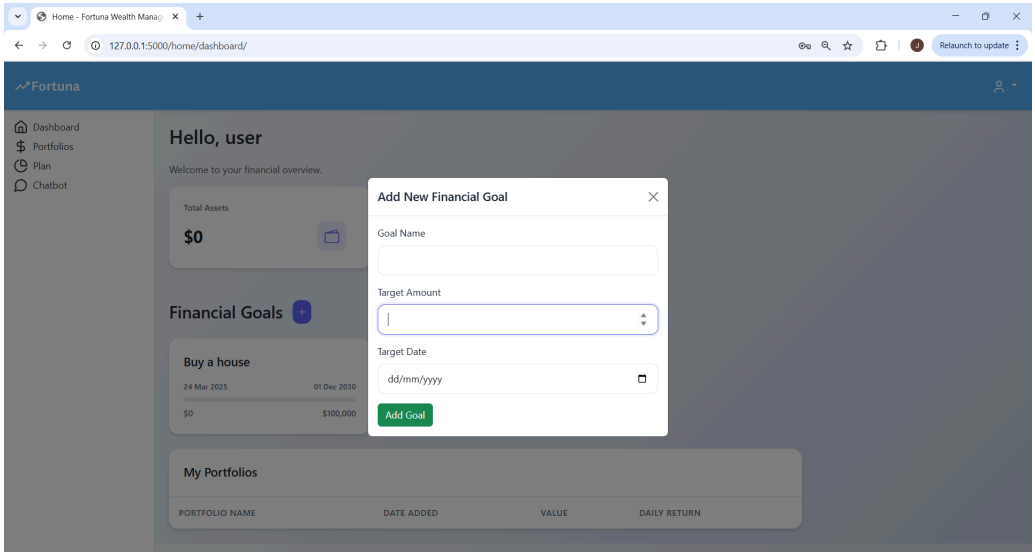

Figure 6.6.2: *Fortuna Add Financial Goal Modal*

The user is also able to add new financial goals from the dashboard.

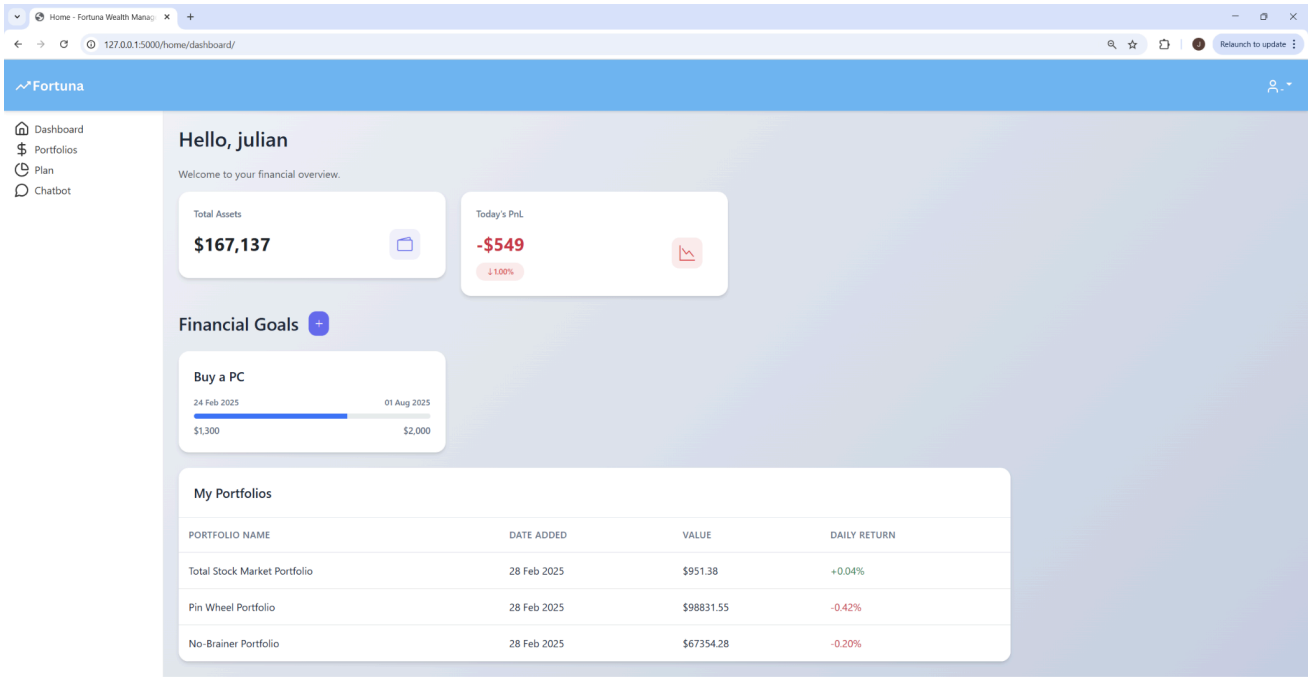

Figure 6.6.3: *Fortuna Dashboard, With Portfolios*

After the user has added portfolios to their account, the user will be able to take advantage of the full features of the dashboard. On the first row, the user can view their total assets which is the sum of current values of all their owned portfolios. The user can also view their Profit and Loss (PnL) for today, which represents how much their portfolios have changed in value since yesterday. The PnL is color coded to be green for positive values and red for negative values, which allows the user to quickly know if they made a net gain or loss.

At the bottom of the dashboard, there is a table which lists down all the portfolios currently owned by the user. The ‘date added’ column will help the user differentiate between 2 portfolios of the same name that were added to the user’s account on different dates, while the ‘value’ column lets the user know how much each portfolio is worth. Lastly, the ‘daily return’ column tells the user the percentage change in value of each portfolio since yesterday. The ‘daily return’ values are also color coded to help users easily identify which portfolio made a profit and which made a loss.

It is also worth mentioning that all the numeric values seen in this page are being updated in real-time. Using jQuery AJAX functions, the web page will fetch portfolio data from the backend server every 30 seconds to dynamically update the total assets card, total PnL card and the portfolio table. This enables the user to actively monitor the real-time status of their assets without refreshing the web page.

3.11.6 Log out

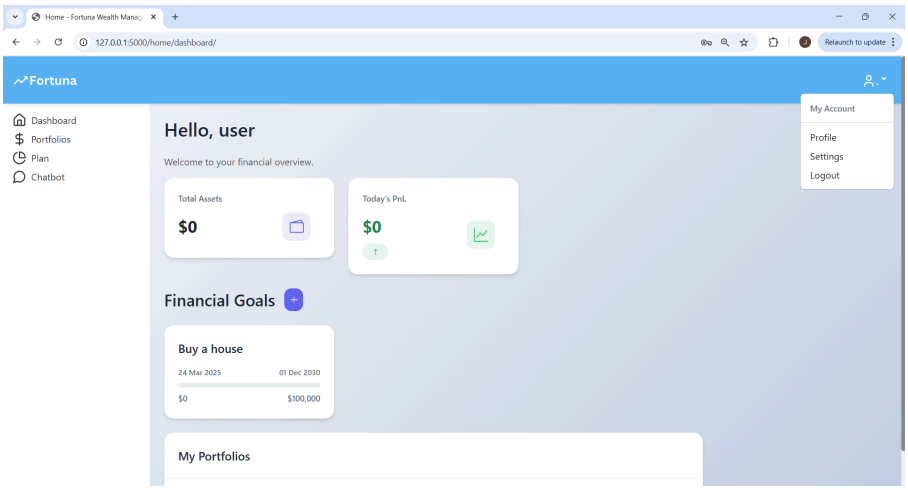

Figure 6.7.1: *Fortuna User Icon Dropdown Menu*

To log out from Fortuna, the user will have to click on the profile icon on the top right corner of the web page. On click, a dropdown menu will appear. Clicking on ‘Logout’ will log the user out of Fortuna.

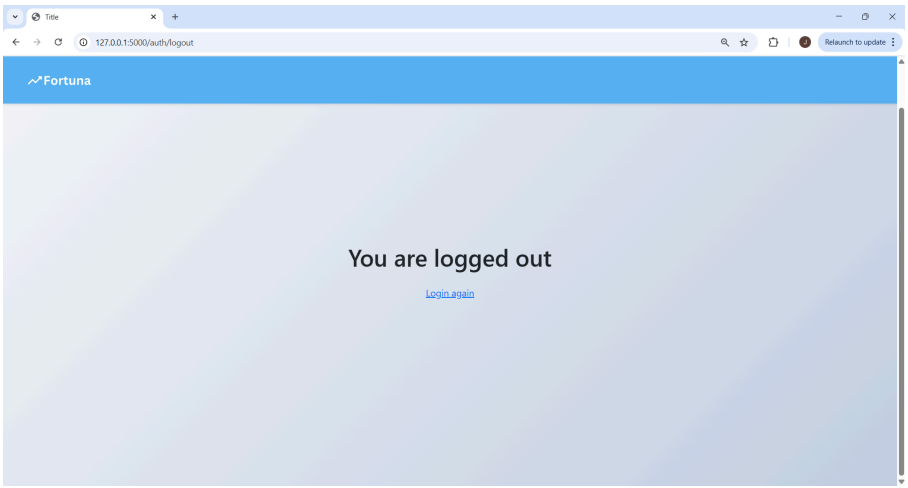

Figure 6.7.2: *Fortuna Logout Page*

Upon successfully logging out, Fortuna will display the above message. Users can click on the ‘Login again’ link to return to the login web page.

3.11.7 Portfolio selection menu

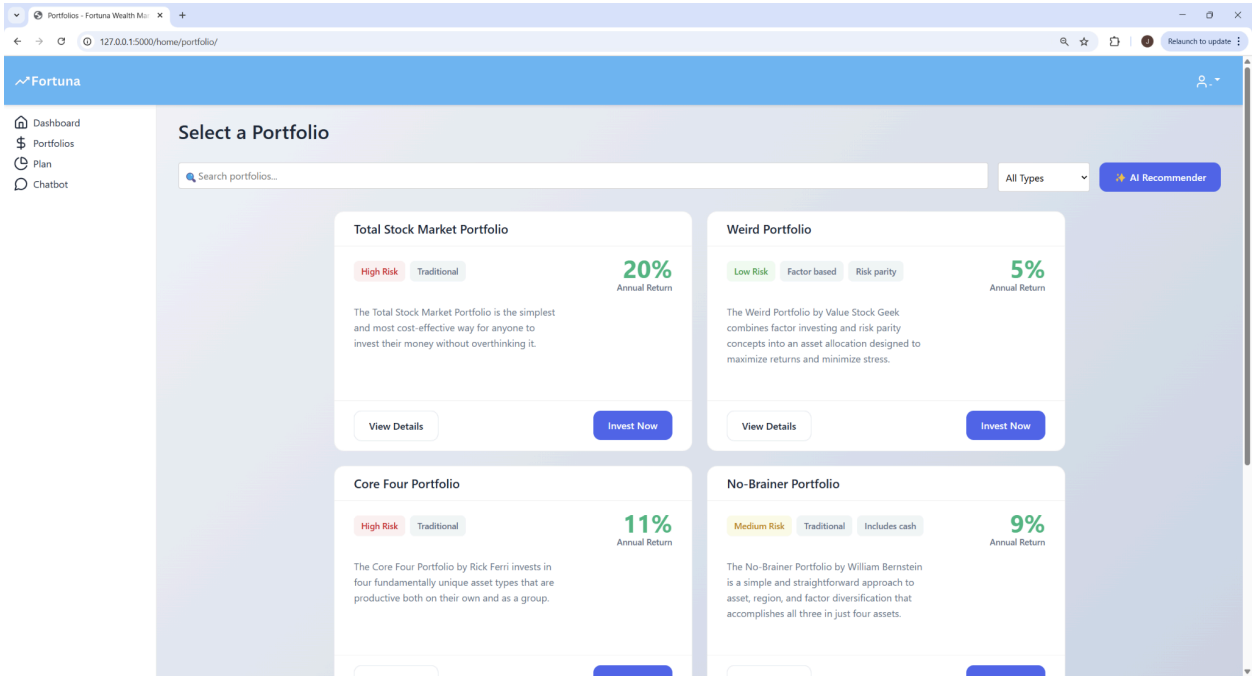

Figure 6.8: *Fortuna Portfolio Selection Page*

Users can navigate to the portfolio selection menu by clicking on ‘Portfolios’ in the left navigation menu. The portfolio menu will display the list of all available portfolios in Fortuna for the user to invest in. Each portfolio card contains essential information about the portfolio, such as risk level, annual return and a short description. The risk level label in each portfolio card is color coded so that users can easily identify which portfolios are more risky and which are less risky.

On the top of the menu, there is a search bar to search for specific portfolios. The moment users start to type in the search bar, the portfolio menu will immediately be updated to display portfolios whose name contains the typed characters, without any page refreshing. Beside the search bar is a filter dropdown which allows users to filter portfolios based on the portfolio tags. The filtering is also done immediately without page refreshes.

3.11.8 AI Recommended Portfolios

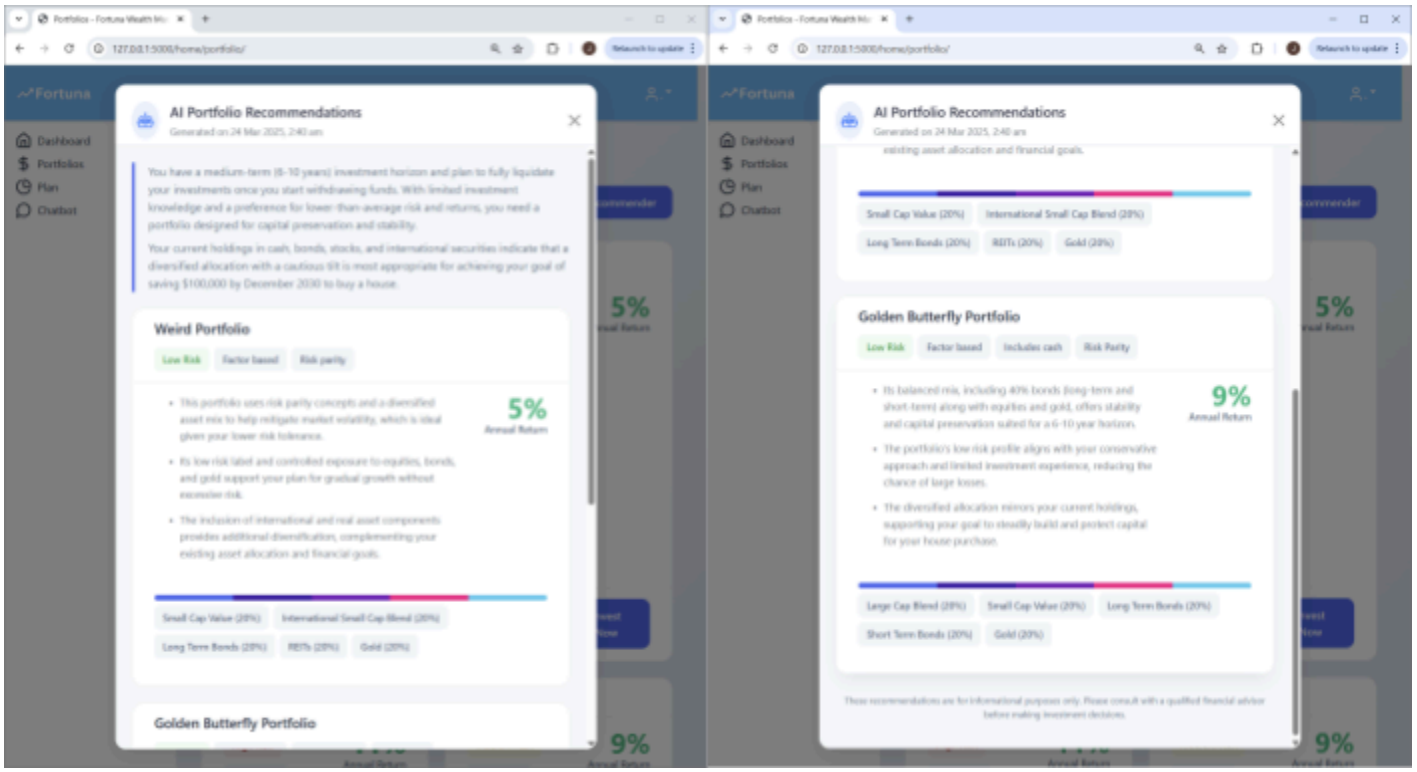

Figure 6.9: *Fortuna AI Recommended Portfolios Modal*

Upon clicking on the ‘AI Recommender’ button on the top right corner, Fortuna will display a pop up which shows a list of recommended portfolios for the user. Using OpenAI’s o3-mini model with high reasoning effort, Fortuna is able to generate a short summary of the user’s risk profile and financial goals, as well as explain what sort of portfolios the user should invest in.

For each of the portfolios recommended by Fortuna, there are 3 bullet points to explain the characteristics of the portfolio that makes it a suitable fit for the user, how the portfolio aligns with their risk profile, and how the portfolio can help them achieve their financial goals.

Lastly, there is a small disclaimer for the user at the bottom of the pop up.

3.11.9 View Details

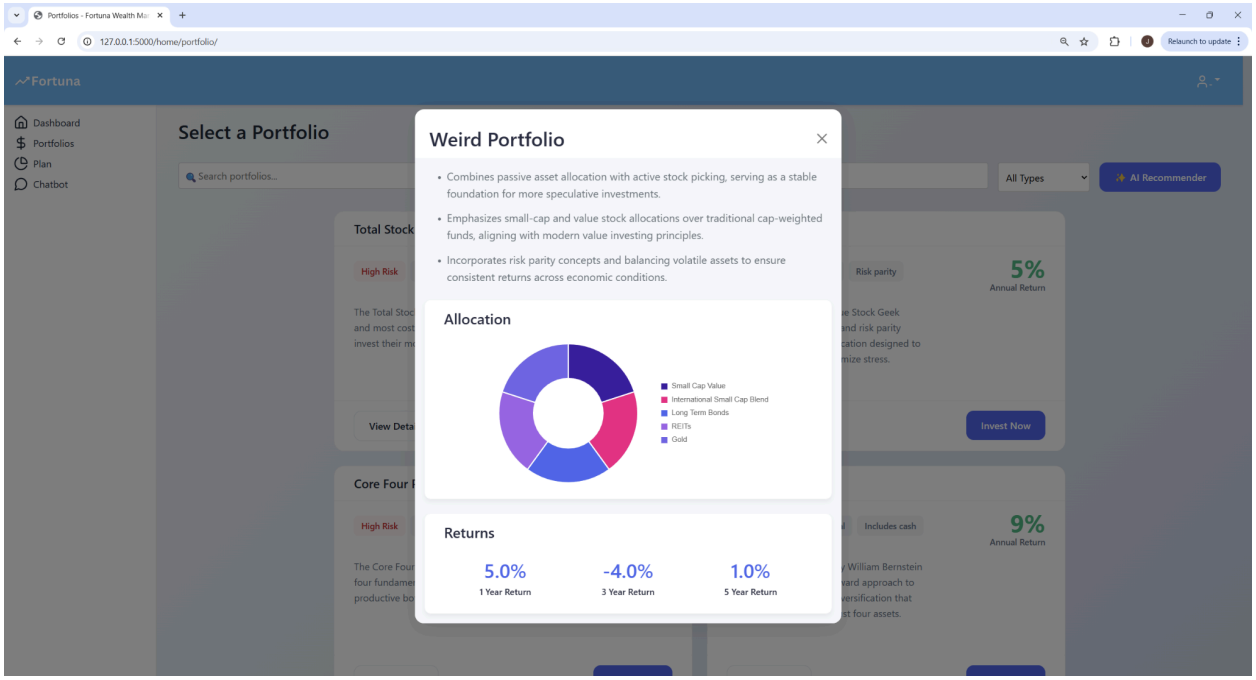

Figure 6.10: *Fortuna Portfolio Details Modal*

When the user clicks on the ‘View Details’ button on the bottom left of the portfolio card, a pop up will appear which contains additional information about the portfolio. Users can read a more detailed summary of the portfolio, inspect the portfolio allocation chart, and view the 1-year, 3-year and 5-year returns of the portfolio. As the allocation chart is created using the Chart.js library, it is a fully interactive chart which allows the user to view the allocation percentage of each asset class when the cursor is hovered above the respective chart segment. The chart is also animated which can be seen when the pop up is first opened.

3.11.10 Invest Now

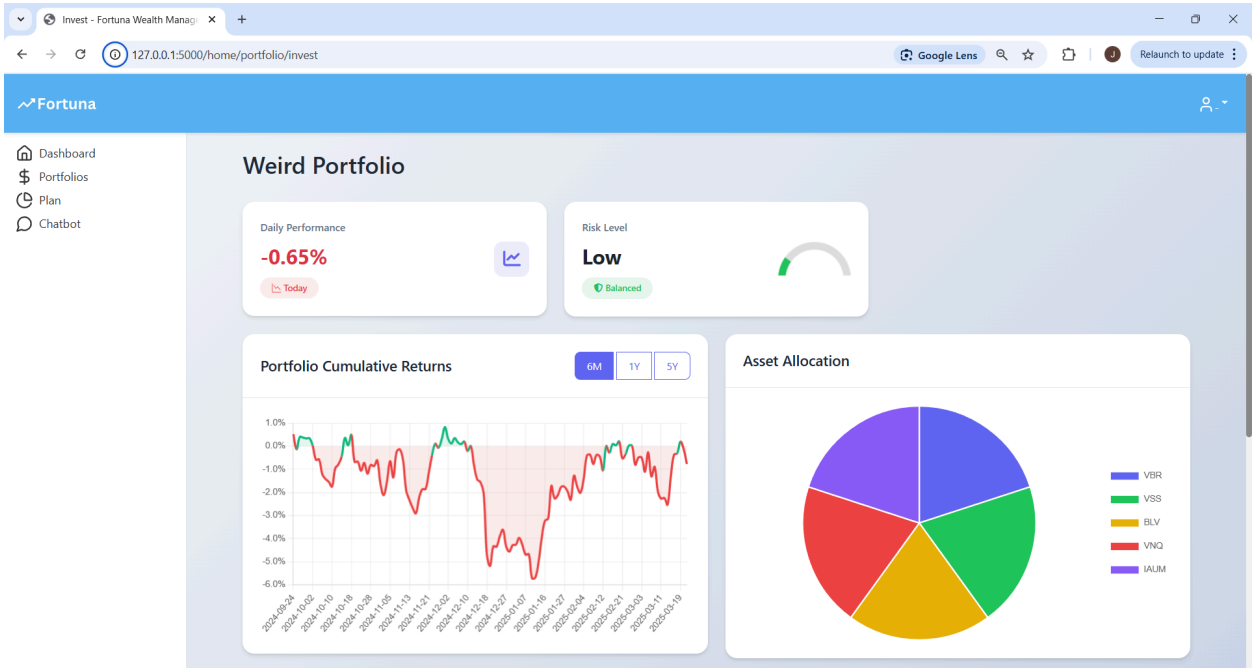

Figure 6.11.1: *Fortuna Invest Now Page (1)*

When the user clicks on the ‘Invest now’ button on the bottom right of the portfolio card, the user will be brought to this web page which displays additional information about the selected portfolio such as daily performance, risk level, historical cumulative returns and asset allocation.

The line chart in this web page is fully interactive as users can hover over each data point to see its exact value. Furthermore, there is a time period filter in the card header which allows users to see how the portfolio has performed across different time periods. When users click on the time period button, the chart is immediately updated without a page refresh.

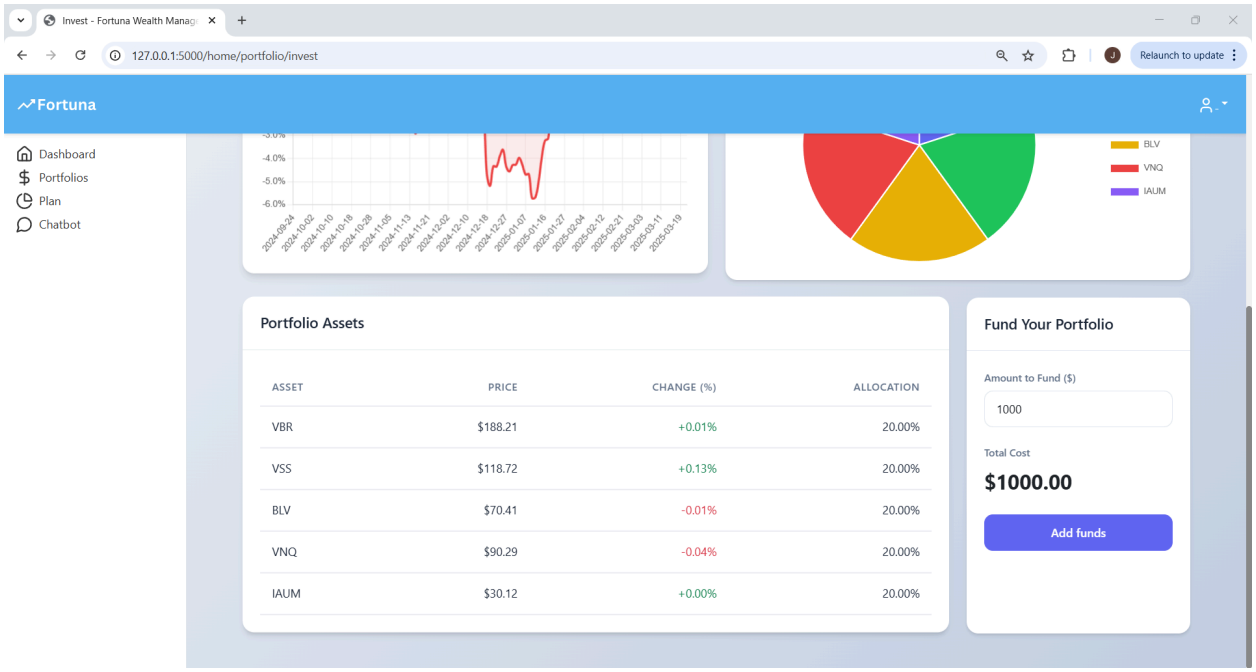

Figure 6.11.2: *Fortuna Invest Now Page (2)*

Upon scrolling down, the user will be able to see a table containing all the assets within the portfolio, which also includes information such as the current price, percentage change and allocation. Again, all price data in this web page are real-time prices which are constantly fetched and updated by the jQuery of this web page. The numbers and line graphs of this web page are also color coded so that users can quickly identify any potential gains or losses.

To purchase a portfolio, users will have to key in the amount of money which they wish to allocate into the portfolio before clicking on the ‘Add funds’ button. An alert will appear on the screen to notify the user if Fortuna has successfully added the portfolio to their account.

3.11.11 Financial Planner

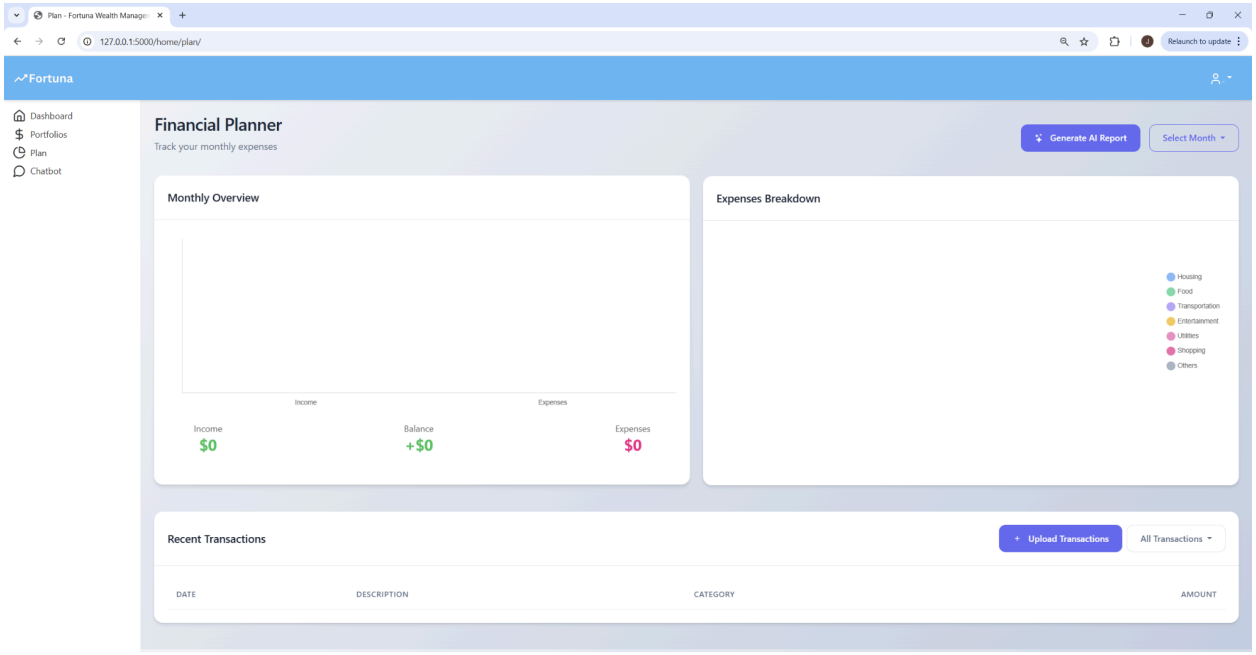

Figure 6.12.1: *Fortuna Financial Planner Page, No Transactions*

Users can access Fortuna's financial planner dashboard by clicking on the ‘Plan’ button in the left navigation sidebar. For users who have not uploaded any bank transactions, the dashboard will appear blank as there is no data to display to the user.

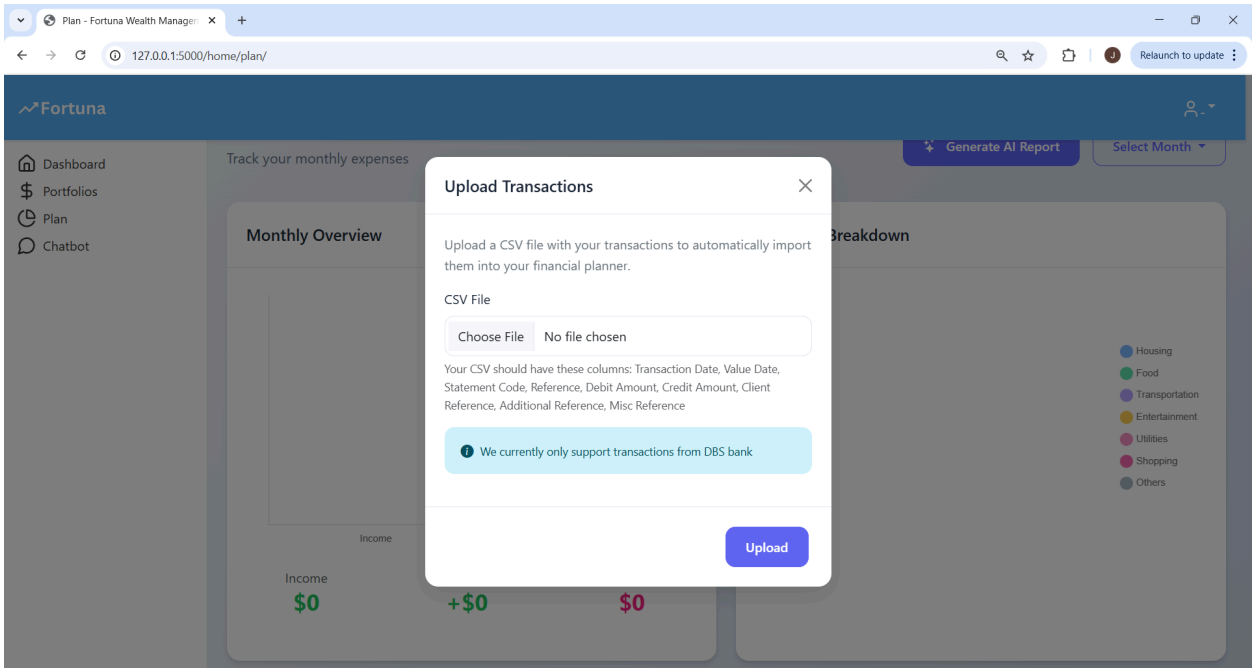

Figure 6.12.2: *Fortuna Upload Transactions Modal*

To upload bank transactions, users will have to click on the ‘Upload transaction’ button found in the header of the Past Transactions table. The user will then be prompted to upload a .csv file to Fortuna which contains a list of all their historical bank transactions. For technical and practicability reasons during development, Fortuna is only able to support .csv files downloaded from DBS bank. If the user tries to click the ‘upload’ button without attaching any file, Fortuna will display a warning message to alert the user. Upon successfully uploading bank transactions to Fortuna, an alert will notify the user about the success. Once Fortuna processes all the uploaded transactions and saves it into the database, Fortuna will categorise all transactions with OpenAI’s Batch API. This process will usually take a few minutes.

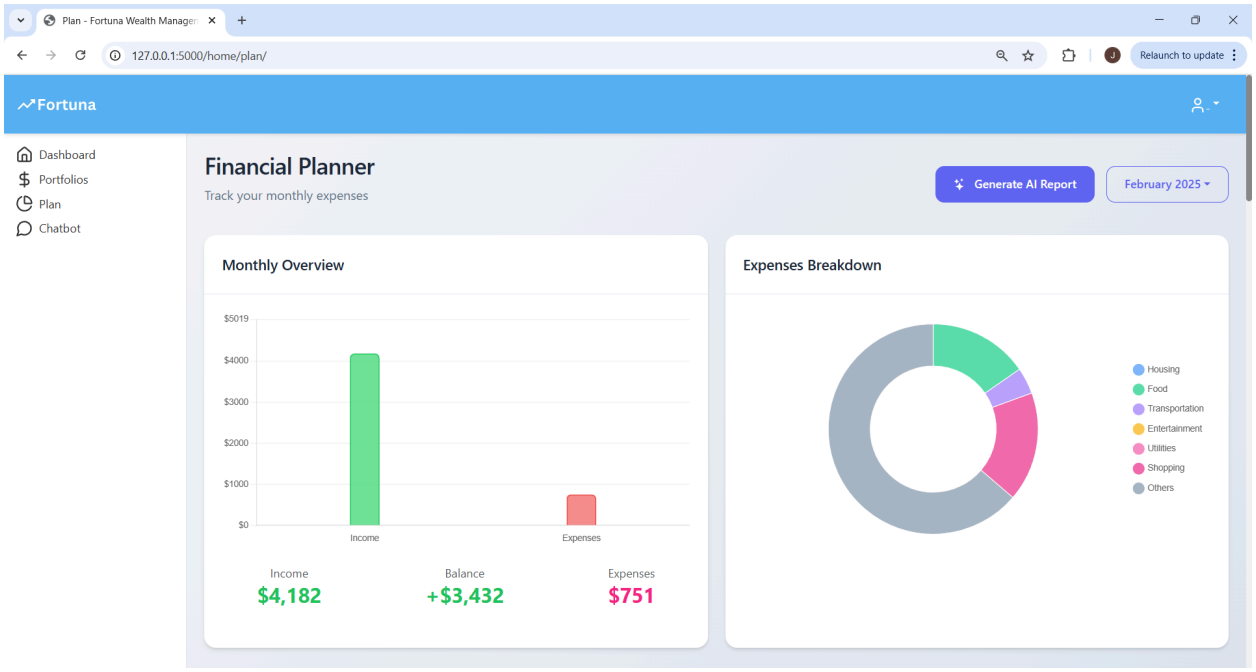

Figure 6.12.3: *Fortuna Financial Planner Page, With Transactions*

Once all the transactions have been categorised, the user can now start to use the financial planner dashboard. The Monthly Overview chart helps users to visualise and compare their income vs expenses for the month, whereas the Expenses Breakdown chart allows users to see how their expenses are distributed across a range of 7 categories, which are: Housing, Food, Transportation, Entertainment, Utilities, Shopping, Others.

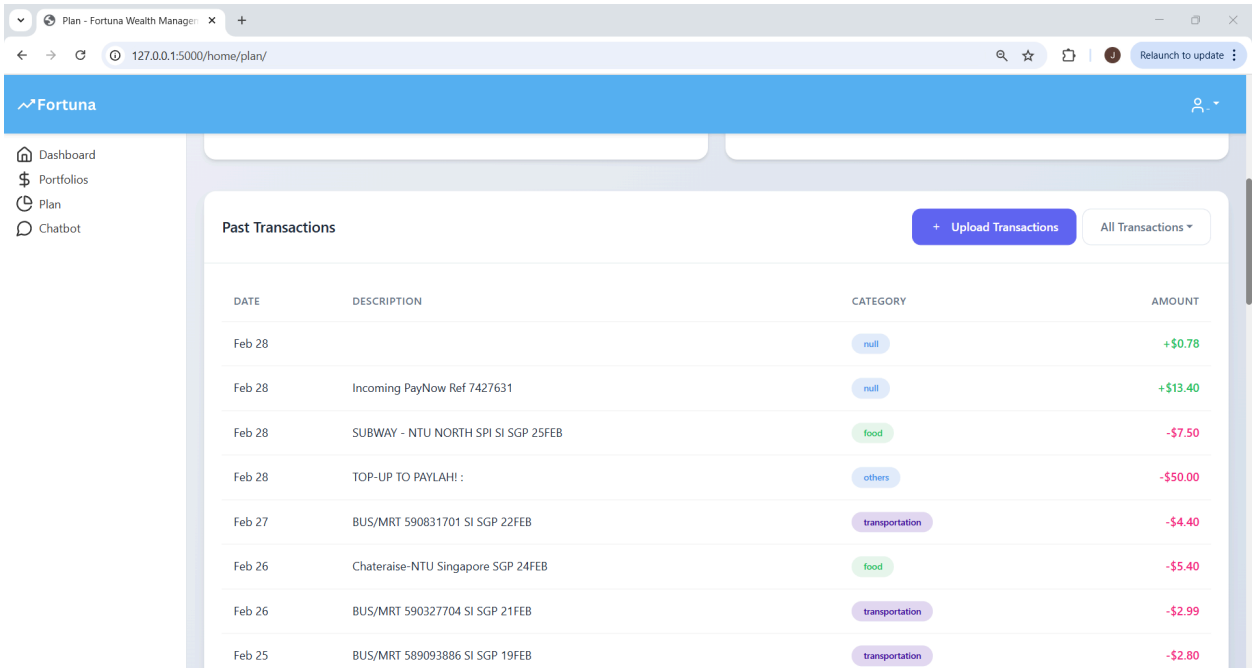

Figure 6.12.4: *Fortuna Past Transactions Table*

Moving down, the users will see a table which contains all their past transactions which have been uploaded to Fortuna. There is a filter dropdown menu in the table header which allows users to filter the transactions table by income, expenses or all transactions.

Going back to the top of the web page, there is another filter which controls the month and year to display transaction data for. Selecting a particular month and year will cause all the charts and tables in the dashboard to be updated accordingly.

3.11.12 AI Financial Report

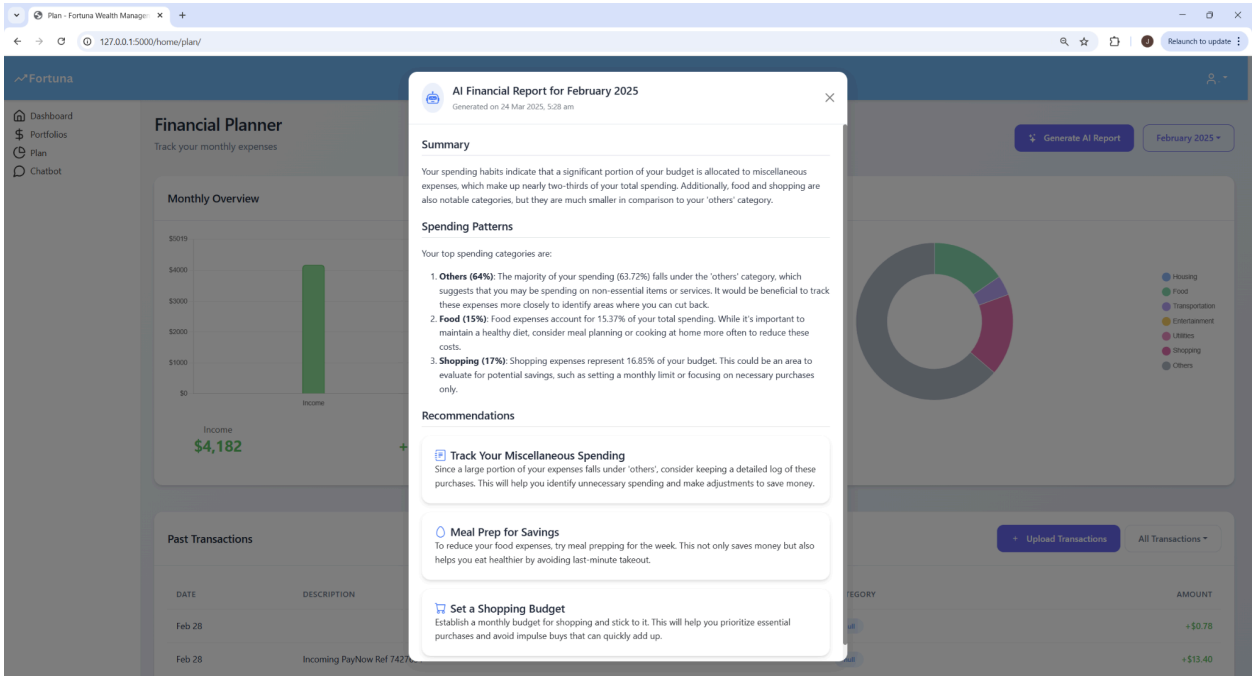

Figure 6.13: *Fortuna AI Financial Report Modal*

Upon clicking on the ‘Generate AI Report’ button at the top right corner of the web page, Fortuna will send the user’s transaction data for the currently selected month to OpenAI’s servers for analysis with GPT models via the Chat Completions API. Fortuna will then format GPT’s response into a financial report for users to gain insights into their spending habits and to receive budgeting advice. In the report, users will receive a summary on their spending habits, a detailed breakdown of their top 3 spending categories, as well as 3 tips from GPT on how to optimize their budget for the future.

3.11.13 Chatbot

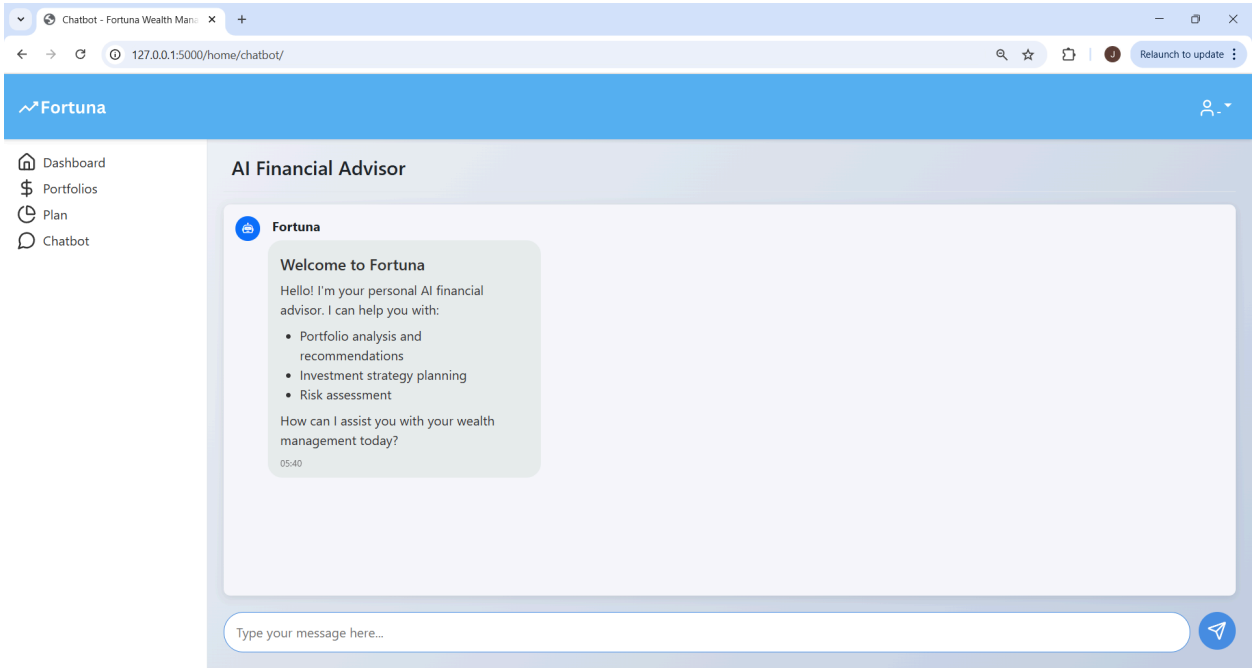

Figure 6.14.1: *Fortuna Chatbot Page*

Users can access Fortuna's chatbot feature by clicking on the ‘Chatbot’ button in the left navigation sidebar. Fortuna’s chatbot is developed using OpenAI’s Assistants. Users will be greeted with a welcome message by Fortuna. Each time the user loads this page, a new chat session will be created.

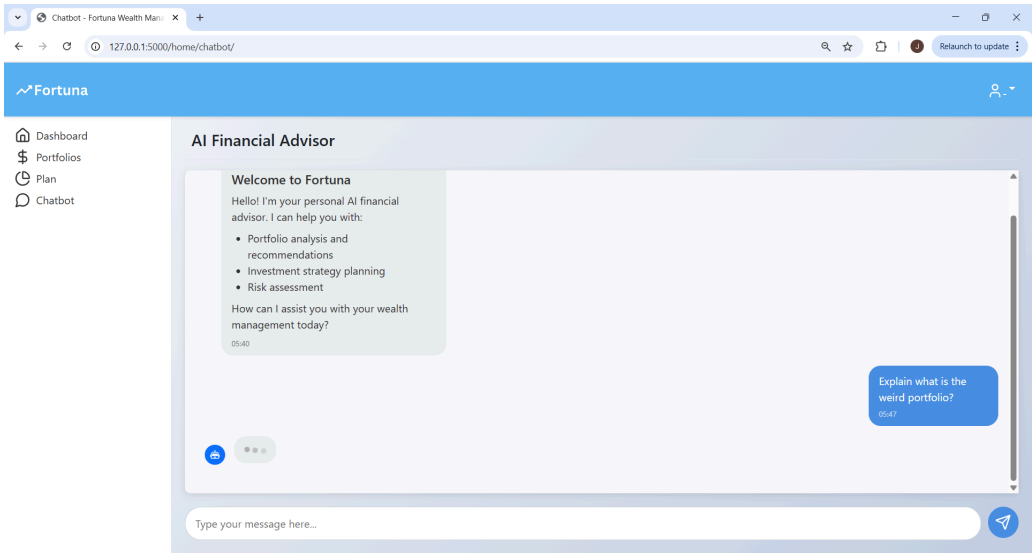

Figure 6.14.2: *Fortuna Chatbot Typing Indicator*

When Fortuna is processing a user message, an animated typing indicator will be displayed to indicate to the user that something is happening in the backend.

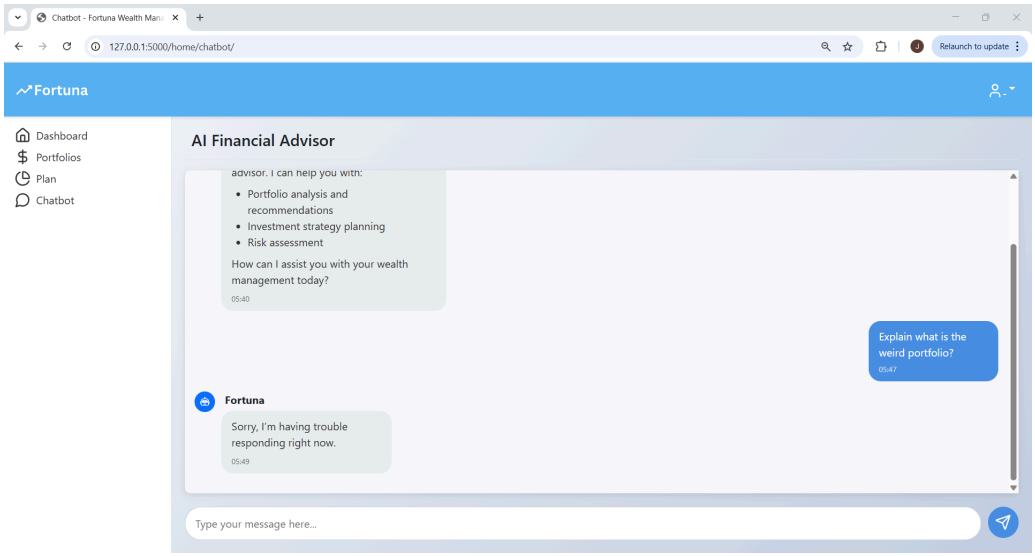

Figure 6.14.3: *Fortuna Chatbot Error Message*

In the case where there is an issue with the servers, Fortuna's chatbot will display a default error message to the user.

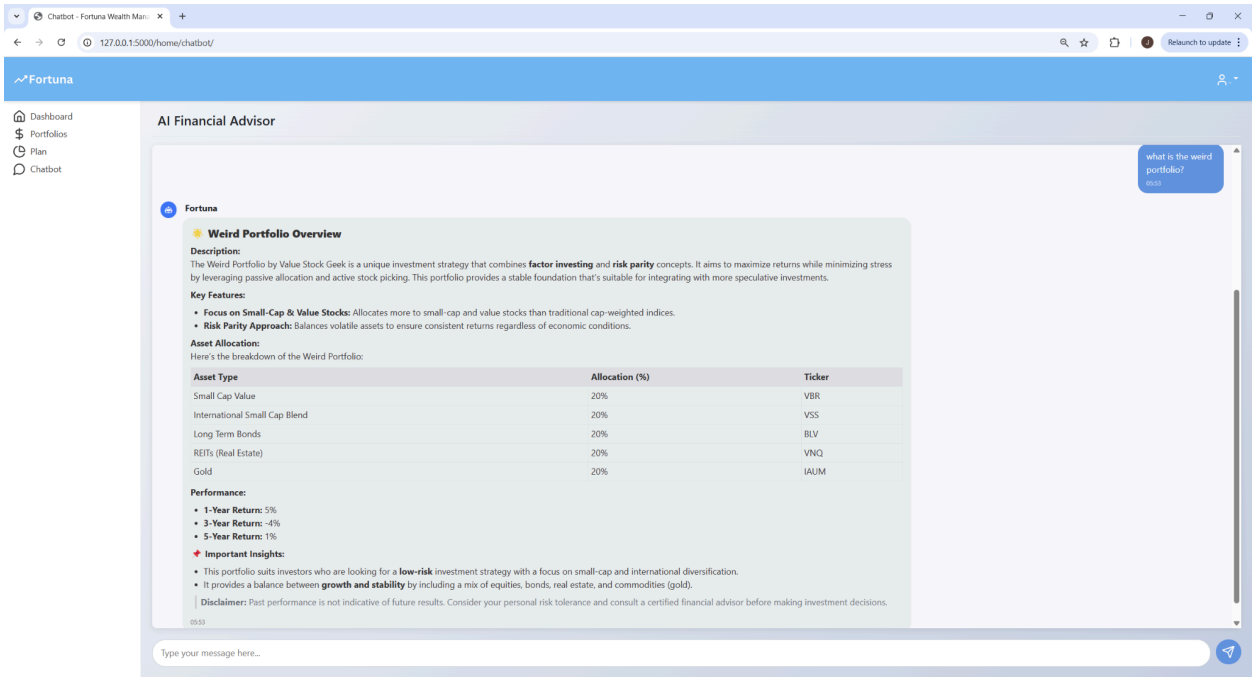

Figure 6.14.4: *Fortuna Chatbot Response*

Fortuna's chatbot is developed such that it will always generate its response using different levels of headers, unordered lists and tables in Markdown syntax. By using the Marked.js library, the chatbot's responses are displayed inside the chat message interface with Markdown format, allowing for a neat and organised look.

Since the chatbot is configured as an OpenAI Assistant with function calling enabled, the chatbot has access to all application data and current user's data, which allows users to prompt the chatbot for various financial advisory tasks.

The below screenshots will demonstrate some of the chatbot's capabilities.

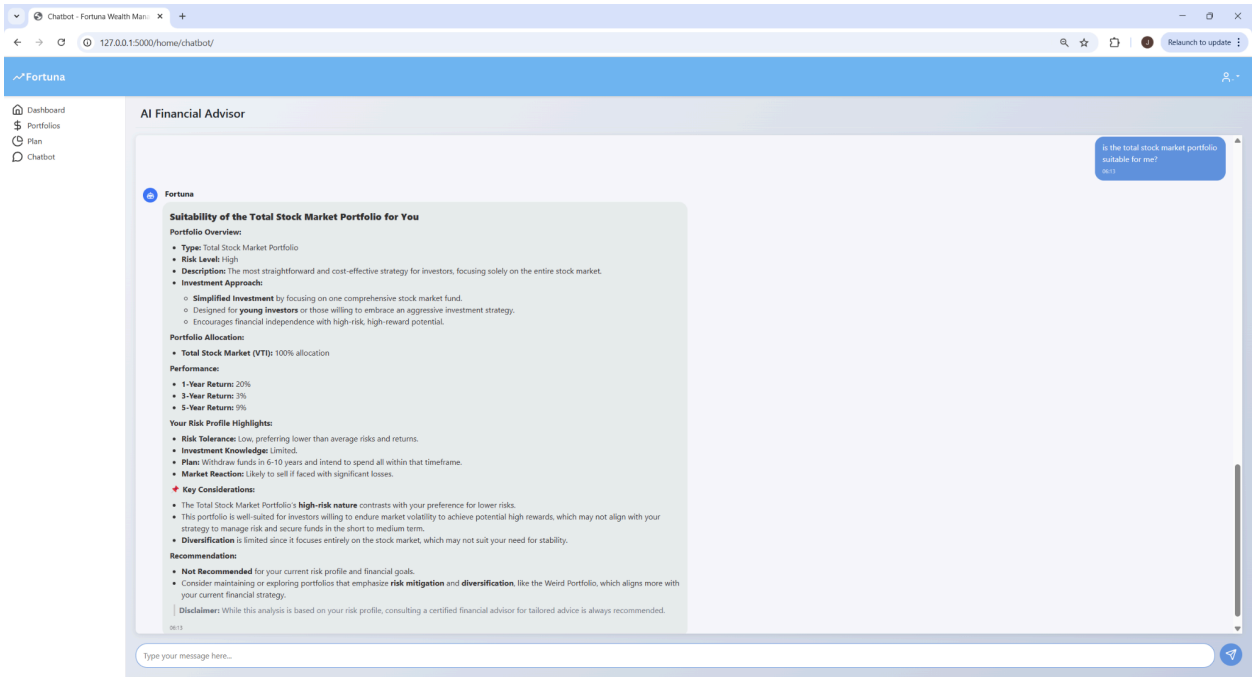

Figure 6.14.5: *Fortuna Chatbot Response (Query Portfolio Suitability)*

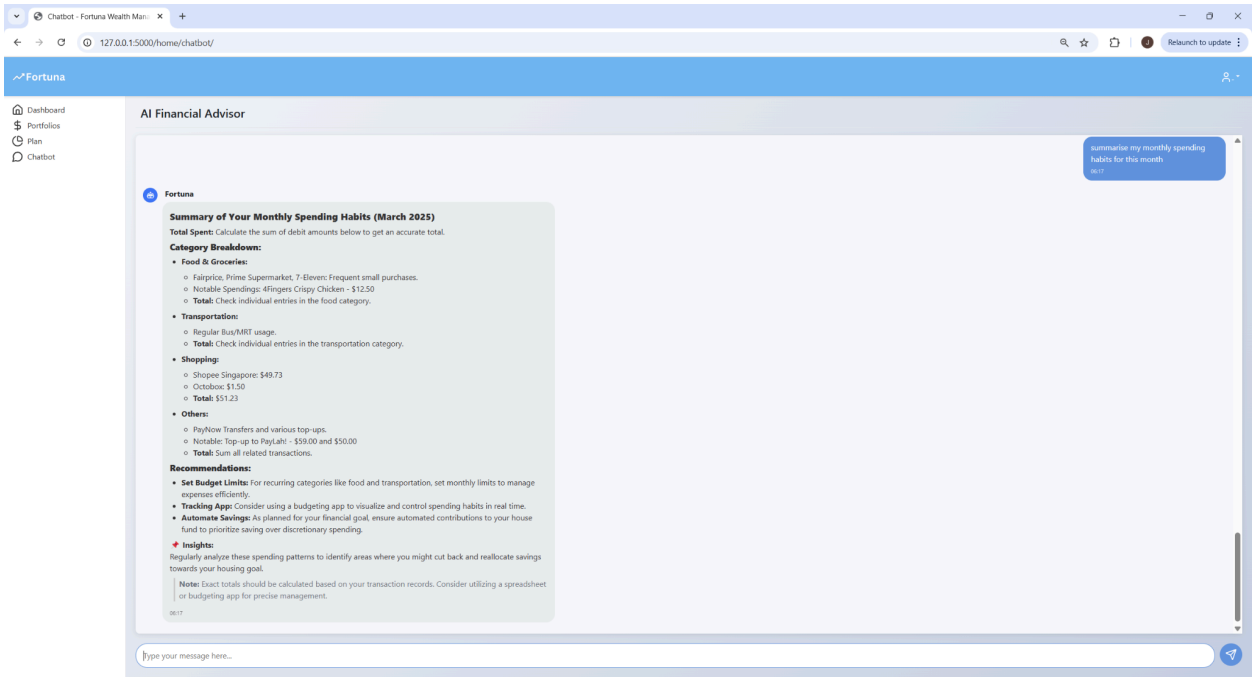

Figure 6.14.6: *Fortuna Chatbot Response (Summarise Monthly Spending Habits)*

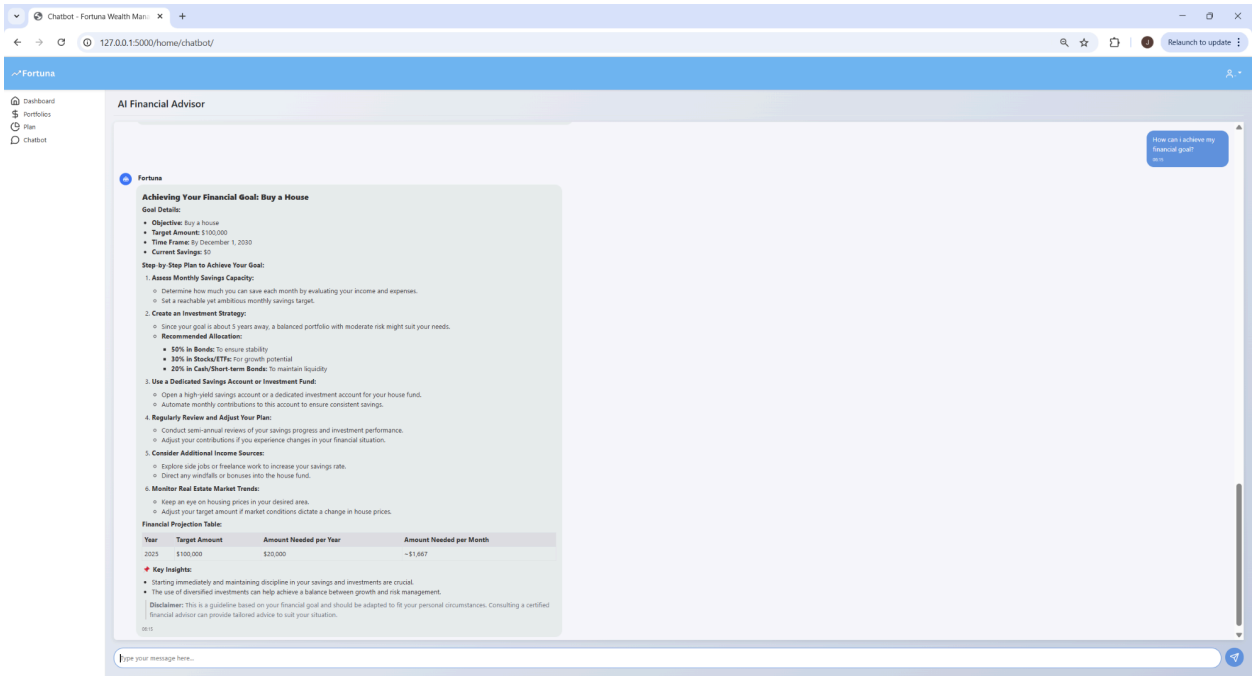

Figure 6.14.7: *Fortuna Chatbot Response (Financial Goals)*

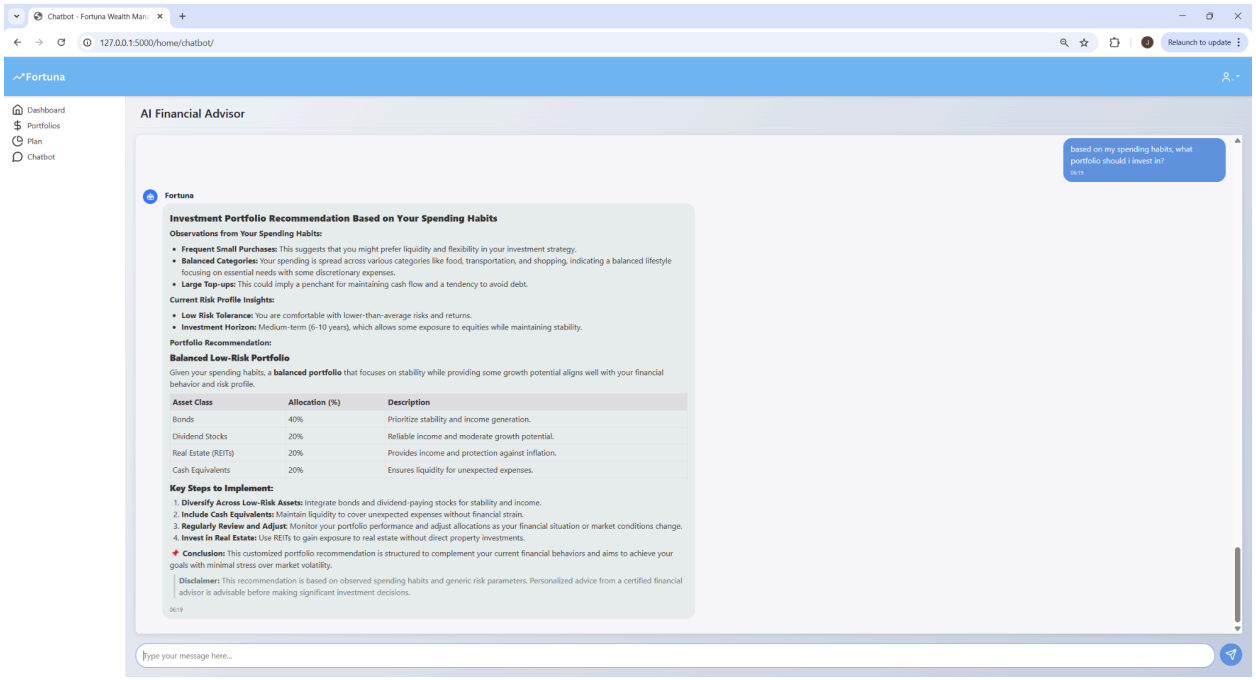

Figure 6.14.8: *Fortuna Chatbot Response (Portfolio Recommendation by Spending Habits)*

Chapter 4: Testing

4.1 Black Box Testing

Black box testing is a software testing technique in which the functionality of the software is evaluated without examining its internal structures or workings. Essentially, testers only know about the inputs (data provided to the software) and the outputs (results), but not how the software actually processes that input.

Below are the reasons why Black Box Testing was used in Fortuna:

1. **Simple and accessible**
   By keeping the testing process simple, this allows testers to not need deep programming knowledge or internal implementation details. This also allows non-technical users or stakeholders to participate in testing.

2. **User Perspective**
   Black box testing simulates end-user behavior by checking if the software meets user expectations. This can help identify usability, functionality and performance issues from a user's viewpoint.

3. **Catch Requirement Issues**
   Black box testing can help validate the software against stated requirements and specifications. This is useful in identifying missing or misunderstood requirements early on in the development process.

4.2 Test cases

The following table contains a comprehensive list of 38 test cases which are carefully selected to provide an exhaustive test of all of Fortuna's functional and error handling features.

Test ID	Description	Expected Results	Actual Results
1	To test the login of a registered user into Fortuna. **Pre-condition:** 1. User is a registered user. 2. User is currently at the login page of the application. **Steps:** 1. User types in a valid username 2. User types in the correct password 3. User clicks on 'Log In' button	User is authenticated and is directed to the dashboard.	Passed
2	To test user login with invalid username (username not in database). **Pre-condition:** 1. User is at the login page of	User is redirected back to the login page. A 'Invalid username' error message is displayed.	Passed

	the application. **Steps:** 1. User types in a invalid username 2. User types in any password 3. User clicks on ‘Log In’ button		
3	To test registered user login with incorrect password. **Pre-condition:** 1. User is a registered user. 2. User is currently at the login page of the application. **Steps:** 1. User types in a valid username 2. User types in an incorrect password 3. User clicks on ‘Log In’ button	User is redirected back to the login page. A ‘Incorrect password’ error message is displayed.	Passed
4	To test creation of new account with valid username. **Pre-condition:**	User is redirected to Questionnaire page.	Passed

	1. User clicked on ‘create new account’ link 2. User is redirected to create new account page. **Steps:** 1. User types in a valid username 2. User types in a password 3. User clicks on ‘continue’ button		
5	To test creation of new account with invalid username (username already in database). **Pre-condition:** 1. User clicked on ‘create new account’ link. 2. User is redirected to create new account page. **Steps:** 1. User types in a invalid username 2. User types in a password 3. User clicks on ‘continue’ button	User is redirected back to create new account page. A warning message is displayed to notify the user that this username is already taken.	Passed
6	To test submission of questionnaire form when all form	User is redirected to financial goals page.	Passed

	fields are answered. **Pre-condition:** 1. User is redirected from create account page. 2. User is in questionnaire page. 3. User has answered all the form fields. **Steps:** 1. User clicks on ‘continue’ button		
7	To test submission of questionnaire form when at least one form field is unanswered. **Pre-condition:** 1. User is redirected from create account page. 2. User is in questionnaire page. 3. At least one form field is unanswered. **Steps:** 1. User clicks on ‘continue’ button	User remains on questionnaire page. An alert is shown on the missing form field to ask the user to fill in the missing field.	Passed
8	To test if user can access	User is redirected to	Passed

	questionnaire page without creating a new account. **Pre-condition:** User is not redirected from create new account page. **Steps:** 1. User loads questionnaire page directly by using the /auth/questionaire URL	create account page	
9	To test submission of financial goals form when all form fields are answered. **Pre-condition:** User is redirected from questionnaire page, is in financial goals page and has answered all the form fields. **Steps:** 1. User clicks on ‘continue’ button	User is redirected to dashboard.	Passed
10	To test submission of financial goals form when at least one form field is unanswered. **Pre-condition:**	User remains on financial goals page. An alert is shown on the missing form field to ask the user to fill in the missing field.	Passed

	User is redirected from questionnaire page, is in financial goals page and at least one form field is unanswered. **Steps:** 1. User clicks on ‘continue’ button		
11	To test if user can access financial goals page without creating a new account. **Pre-condition:** User is not redirected from questionnaire page. **Steps:** 1. User loads financial goals page directly by using the /auth/financial-goals URL	User is redirected to create account page	Passed
12	To test dashboard cards’ and table’s display when the user does not own any portfolios. **Pre-condition:** User is in dashboard page and does not own any portfolios **Steps: -**	Dashboard total assets card displays 0. Dashboard PnL card displays 0. My Portfolios table is empty.	Passed

13	To test dashboard cards' and table's display when the user owns at least 1 portfolio. **Pre-condition:** User is in dashboard page and owns at least 1 portfolio. **Steps: -**	Dashboard total assets card displays total value of all owned portfolios. Dashboard PnL card displays total daily PnL for all owned portfolios. My Portfolios table displays all owned portfolios.	Passed
14	To test addition of new financial goal when all form fields are answered. **Pre-condition:** 1. User is in dashboard page. **Steps:** 1. User clicks on '+' button. 2. User fills out all fields in the form. 3. User clicks on 'Add Goal' button.	An alert with the message 'Financial goal added successfully' is displayed	Passed
15	To test addition of new financial goal when at least 1 form field is not answered. **Pre-condition:** 1. User is in dashboard page.	An alert is shown on the missing form field to ask the user to fill in the missing field.	Passed

	Steps: 1. User clicks on '+' button. 2. User does not fill out at least 1 form field. 3. User clicks on 'Add Goal' button.		
16	To test if portfolios table is being updated in real time. **Pre-condition:** 1. User is in dashboard. 2. User owns at least 1 portfolio. **Steps: -**	The values of the 'VALUE' column and 'DAILY RETURN' column of the portoflios table change every minute.	
17	To test user log out. **Pre-condition:** 1. User is logged into Fortuna. **Steps:** 1. User clicks on profile icon on top right corner of web page 2. User clicks on 'Logout' button of dropdown menu	The user is logged out and is redirected to log out page	Passed
18	To test portfolio selection menu search bar.	The portfolio menu page is immediately updated to	Passed

	Pre-condition: 1. User is in portfolio selection page. **Steps:** 1. User starts typing into search bar.	only display portfolio cards with names containing the string that is currently in the search bar.	
19	To test portfolio selection menu filter button. **Pre-condition:** 1. User is in portfolio selection page. **Steps:** 1. User clicks on filter button 2. User clicks on a portfolio tag in the dropdown menu	The portfolio menu page is immediately updated to only display portfolio cards that have a tag that matches the user selected tag.	Passed
20	To test AI recommended portfolios functionality. **Pre-condition:** 1. User is in portfolio selection page. **Steps:** 1. User clicks on 'AI Recommender' button	A modal appears on screen which contains a spinning indicator. After a short period of time, the spinning indicator disappears and the modal becomes populated with the AI's recommendations.	Passed

21	To test ‘View Details’ button of portfolio card. **Pre-condition:** 1. User is in portfolio selection page. **Steps:** 1. User clicks on ‘View Details' button in the footer of a portfolio card	A modal appears on screen which contains additional information about the related portfolio such as long description, asset allocation and 1Y, 3Y, and 5Y returns.	Passed
22	To test ‘Invest Now’ button of portfolio card. **Pre-condition:** 1. User is in portfolio selection page. **Steps:** 1. User clicks on ‘Invest Now' button in the footer of a portfolio card	User is redirected to invest page of the corresponding portfolio	Passed
23	To test time period filter buttons of portfolio returns line chart. **Pre-condition:** 1. User is in invest page. **Steps:**	The line chart is immediately updated to display data points within the selected time period.	Passed

	1. User clicks on one of the time period buttons.		
24	To test if assets table is being updated in real time. **Pre-condition:** 1. User is in invest page. **Steps: -**	The values of the 'PRICE' column and 'CHANGE (%)' column of the asset table change every minute.	Passed
25	To test user adding portfolios to account. **Pre-condition:** 1. User is in invest page. **Steps:** 1. User keys in a number in the 'Amount to Fund' form field. 2. User clicks on 'Add Funds' button.	An alert is displayed with the message 'Portfolio successfully added'.	Passed
26	To test display of financial planner dashboard when there are no user transactions **Pre-condition:** 1. User is in financial planner page. 2. User has not uploaded any	The monthly overview chart, expenses breakdown chart and past transactions chart are all empty.	Passed

	transactions. **Steps: -**		
27	To test upload of bank transactions when a csv file is attached. **Pre-condition:** 1. User is in financial planner page. **Steps:** 1. User clicks on 'Upload Transactions' button 2. User attaches a csv file with the correct format 3. User clicks on the 'Upload' button	A alert is displayed which notifies the user that all bank transactions have been uploaded.	Passed
28	To test upload of bank transactions when no file is attached. **Pre-condition:** 1. User is in financial planner page. **Steps:** 1. User clicks on 'Upload Transactions' button	An alert is displayed on the attach file form input to ask the user to attach a csv file.	Passed

	2. User clicks on the ‘Upload’ button		
29	To test display of financial planner dashboard when there are user transactions. **Pre-condition:** 1. User is in financial planner page. 2. User has uploaded transactions. 3. User transactions are categorised. **Steps: -**	The monthly overview chart and expenses breakdown chart is visible. The past transactions table contains all past user transactions of the latest month.	Passed
30	To test month year filter of financial planner dashboard. **Pre-condition:** 1. User is in financial planner page. 2. User has uploaded transactions. 3. User transactions are categorised. **Steps:** 1. User clicks on the month year filter	The monthly overview chart, expenses breakdown chart and past transactions table are immediately updated to only contain transactions of the selected month year.	Passed

	2. User selects a month year from the dropdown menu		
31	To test generate AI financial report feature. **Pre-condition:** 1. User is in financial planner page. 2. User has uploaded transactions. 3. User transactions are categorised. **Steps:** 1. User clicks on 'Generate AI Report' button	A modal is displayed which contains a spinning indicator. After a short period of time, the spinning indicator disappears and the modal is populated with the AI generated financial report.	Passed
32	To test error message of chatbot **Pre-condition:** 1. User is in chatbot page. 2. There is an error with the backend server. **Steps:** 1. User types some text in the message input. 2. User presses 'Enter' key or clicks on the send message button.	A typing indicator is displayed at the bottom of the initial bot message. After a short period of time, the typing indicator is replaced with the message: "Sorry. I'm having trouble responding right now."	Passed

33	To test if chatbot remembers the previous user messages. **Pre-condition:** 1. User is in chatbot page. **Steps:** 1. User sends some message to the chatbot. 2. User asks chatbot a question about the previous message sent.	Chatbot is able to answer question about the previous user message.	Passed
34	To test if chatbot has access to portfolio data in Fortuna. **Pre-condition:** 1. User is in chatbot page. **Steps:** 1. User asks the chatbot a question about the available portfolios in Fortuna.	Chatbot is able to answer the question about portfolios in Fortuna correctly.	Passed
35	To test if chatbot has access to user portfolio data. **Pre-condition:** 1. User is in chatbot page. 2. User has added portfolios	Chatbot is able to correctly identify which portfolio does the user own.	Passed

	to their account. **Steps:** 1. User asks the chatbot a question about the their owned portfolios.		
36	To test if chatbot has access to user risk profile data. **Pre-condition:** 1. User is in chatbot page. **Steps:** 1. User asks the chatbot to summarise their risk profile.	Chatbot is able to correctly summarise the user's risk profile.	Passed
37	To test if chatbot has access to user financial goals data. **Pre-condition:** 1. User is in chatbot page. **Steps:** 1. User asks the chatbot about their financial goals.	Chatbot is able to correctly identify the user's financial goals.	Passed
38	To test if chatbot has access to user transaction data. **Pre-condition:**	Chatbot is able to correctly summarise the user's monthly transactions.	Passed

	1. User is in chatbot page. 2. User has uploaded transactions. **Steps:** 1. User asks the chatbot to summarise their monthly transactions.		

***Table 8**: Description of Test Cases*

Chapter 5: Conclusion

5.1 Summary

In this final year project, we have taken a look at the wealth management and robo-advisory industry at large, together with how these industries are utilising disruptive technologies such as Large Language Models (LLMs) to stay ahead of the curve. While the state of LLMs and Natural Language Processing (NLP) as a whole has been advancing rapidly in recent years, there remains a lack of real-life applications of LLMs in the financial industry, especially in the domain of personal finance. Hence, Fortuna was created as a proof-of-concept to showcase some potential use cases of LLMs as a personal financial advisor. By harnessing the powerful textual analysis capabilities of LLMs, together with innovative tools that provide LLMs with access to big data, Fortuna is able to provide highly customised financial advice and invaluable insights to its users which can rival the performance of actual financial advisors. Thus, it is hoped that Fortuna can pave the way for more innovations that bridge the gap between state-of-the-art AI technology and practical financial advisory services, potentially revolutionizing how individuals can benefit from personalized wealth management solutions.

5.2 Future Work

The following are some avenues that could be improved on or explored further in the future.

1. **Integration with banks**
   To provide a more seamless user experience, it would be beneficial for Fortuna to receive support from major banks worldwide to integrate APIs. This would allow Fortuna to directly gain access to users' past transactions after authenticating users via their online banking accounts, removing the need for users to manually upload their bank transactions into Fortuna.

2. **Better logic for processing csv files**
   In the current iteration of Fortuna, it is required that all csv files to be uploaded into Fortuna follow a specific sequence of column headers in the first row. In reality, different banks will export csv with different formats, so to support as many users as possible Fortuna should consider letting users map their csv columns to the columns that Fortuna requires.

3. **More sophisticated classification algorithms**
   Currently, user transactions are classified by prompting GPT once for each transaction in the database via the Batch API. In the case where a user uploads thousands of transactions at once, this approach would be quite costly from both a time and monetary standpoint. To reduce the number of API calls, a rule based classifier could be implemented to classify transactions based on certain words that appear in the description. For instance, transactions which contain the word 'bus' or 'mrt' can immediately be classified into the transport category. Furthermore, once there is a sufficient number of labeled data in the database, an embedding-based nearest-neighbor search strategy could be implemented to classify new transactions, simply by retrieving the category of the transaction which is the closest to the new transaction in the embedding space.

4. **Fine-tune for financial advising**

   One potential way of improving the financial advising capabilities of Fortuna's chatbot would be to fine-tune a LLM on a dataset which consists of client questions together with their risk profiles as input and financial advisors' advice as output. However, whether it is even feasible to curate such a dataset is still left for debate.

Chapter 6: References

Baghai, P., D'Amico, A., Roche-Zhu, R. d. l., Erzan, O., Golyk, V., & Zucker, J. (2020, January 22). *On the cusp of change: North American wealth management in 2030*. McKinsey & Company. https://www.mckinsey.com/industries/financial-services/our-insights/on-the-cusp-of-change-north-american-wealth-management-in-2030

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., … Amodei, D. (2020). Language Models are Few-Shot Learners. *Advances in Neural Information Processing Systems*, *33*, 1877–1901. https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf

Campisia, G., & Muzzioli, S. (2023, September 1). Investor sentiment and trading behavior. *Chaos*, *30*(9). https://doi.org/10.1063/5.0011636

CAPCO. (2024, June 14). *TRANSFORMING WEALTH MANAGEMENT : BALANCING HIGH TOUCH & HIGH TECH*. https://www.capco.com/about-us/newsroom-and-media/singapore-wealth-management-survey

Charles Schwab. (n.d.). *Investor Profile Questionnaire*. https://www.schwab.com/resource/investment-questionnaire

Crosman, P. (2023, May 16). *AI will make workers 'superhuman': Goldman Sachs CIO Marco Argenti*. American Banker. https://www.americanbanker.com/news/ai-will-make-workers-become-superhuman-goldman-sachs-cio-marco-argenti

Crosman, P. (2023, June 20). *'It's worth all the hype': SouthState Bank deploys enterprise ChatGPT*. American Banker. https://www.americanbanker.com/news/its-worth-all-the-hype-southstate-bank-deploys-chatgpt-like-tech

Crosman, P. (2023, July 3). *JPMorgan Chase using advanced AI to detect fraud*. American Banker. https://www.americanbanker.com/news/jpmorgan-chase-using-chatgpt-like-large-language-models-to-detect-fraud

Du, K., Zhao, Y., Mao, R., Xing, F., & Cambria, E. (2025). Natural language processing in finance: A survey. *Information Fusion*, *115*(March 2025). https://doi.org/10.1016/j.inffus.2024.102755

Dzhaparov, P. (2022). Artificial Intelligence – a Key Success Factor for Wealth Management Industry. *Izesstia, Journal of the Union of Scientists - Varna, Economic Sciences Series*, *11*(2), 97–104.

Foerster, S., Linnainmaa, J. T., Melzer, B. T., & Previtero, A. (2017, April 12). Retail Financial Advice: Does One Size Fit All? *The Journal of Finance*, *72*(4), 1441-1482. https://doi.org/10.1111/jofi.12514

Jiang, Z., Peng, C., & Yan, H. (2024, March). Personality differences and investment decision-making. *Journal of Financial Economics*, *153*.

Katsuki, F., Khana, A., Kim, D. D., Kwok, T., Park, J., Suneja, S., & Thomas, R. (2023, February 2). *Digital and AI-enabled wealth management: The big potential in Asia*. McKinsey & Company. https://www.mckinsey.com/industries/financial-services/our-insights/digital-and-ai-enabled-wealth-management-the-big-potential-in-asia

Lam, J. W. (2016, April 4). *Robo-Advisors: A Portfolio Management Perspective*. [Senior Essay, Yale College]. Yale College. https://economics.yale.edu/sites/default/files/2023-01/Jonathan_Lam_Senior%20Essay%20Revised.pdf

Luo, H., Liu, X., Lv, X., Hu, Y., & Ahmad, A. J. (2024). Investors' willingness to use robo-advisors: Extrapolating influencing factors based on the fiduciary duty of investment advisors. *International Review of Economics & Finance*, *94*(July 2024). https://doi-org.remotexs.ntu.edu.sg/10.1016/j.iref.2024.103411

Martinez, A. (2022). *The future of the data-driven workplace*. Arizent. https://arizent.brightspotcdn.com/be/b2/9042045949d9a57d5da9b545b0b5/ai-driven-decision-making-researchreport-072022-v8.pdf

McKenna, N., Li, T., Cheng, L., Hosseini, M. J., Johnson, M., & Steedman, M. (2023). Sources of Hallucination by Large Language Models on Inference Tasks. *Conference on Empirical Methods in Natural Language Processing*. https://api.semanticscholar.org/CorpusID:258865517

Portfolio Charts. (n.d.). *Portfolios*. https://portfoliocharts.com/portfolios/

Shabsigh, G., & Boukherouaa, E. B. (2023). *Generative Artificial Intelligence in Finance: Risk Considerations.* (2023/006). IMF Fintech Note.

https://www.imf.org/en/Publications/fintech-notes/Issues/2023/08/18/Generative-Artificial-Intelligence-in-Finance-Risk-Considerations-537570

Statista. (n.d.). *Wealth Management - Worldwide*. https://www.statista.com/outlook/fmo/wealth-management/worldwide

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention Is All You Need. *NIPS'17: Proceedings of the 31st International Conference on Neural Information Processing Systems*, 6000 - 6010. https://doi.org/10.48550/arXiv.1706.03762

Yin, Y. (2024). *Does The Adoption Of Digital Wealth Management Platforms (DWPs) Result In Positive Net Benefits? From Retail Investors' Perspective On DWPs Success*. [Master's Theses, Dalhousie University]. Dalhousie University. https://hdl.handle.net/10222/84750