Student Number: 2107350

*Final Report*

# *Enhancing Personal Finance Management through Data Visualization*


Author: Tushar Vashista (2107350)

Supervisor: Jing Wu

*BSc. Computer Science School of Computer Science and Informatics*

17 May 2024

## ABSTRACT

This project, titled "Enhancing Personal Finance Management through Data Visualization," addresses the gap in personal finance management tools designed specifically for students. As the current financial management applications are often made for business use, making them overly complex and less suitable for individual users, however, this project utilizes data visualization techniques to transform financial data into insightful visual representations. While the initial plan included the use of Optical Character Recognition (OCR) technology to extract data from receipts, however, due to limitations in finding a capable OCR API led to a focus on manually adding expenses. Hence, now the primary goal is to create a user-friendly interface that helps students track and analyse their spending, providing actionable insights to improve their financial literacy and decision-making.

## ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my project supervisor, Jing Wu, for her invaluable guidance, support, and encouragement throughout this project. Their feedback and input have been instrumental in shaping the outcomes and results of this project.

I would also like to thank the participants who took part in the surveys and interviews, providing essential feedback and requirements that helped shape the development of the financial management tool.
Special thanks to my family and friends for their unwavering support and encouragement during this challenging yet rewarding journey.

Lastly, I appreciate the resources and support provided by the university and open-source community, which have been crucial in the successful completion of this project.

# Table of Contents

# 1. Introduction

## 1.1 Problem Background

As students step into adulthood, they often face the daunting task of managing their finances independently for the first time, and this challenge can persist well into their early adult years. This newfound financial freedom can be overwhelming, leading many students to exceed their budgets and struggle with financial management. Before university, most students' expenses are typically handled by their parents or guardians, leaving them unprepared for the responsibilities of budgeting and financial planning. This project was inspired by my personal experience and the difficulties I observed among fellow students in managing their finances. There is a clear need for a tool that helps students better understand their spending habits and provides insights into how they can improve their financial management skills.

## 1.2 Objectives

The primary objective of this project is to develop a personalized financial management tool specifically designed for students. Although some aspects have evolved from the initial plan, such as the use of OCR to accurately extract text, challenges in finding a reliable free OCR API led to necessary adjustments. However, the core aim remains unchanged: to help students understand their spending habits, identify which habits positively or negatively impact their budgets, and ultimately enhance their financial literacy.
The key objectives of the project are:

- To provide students with a method to self-learn financial management.
- To create a financial dashboard that visualizes spending habits and highlights areas for improvement.
- To ensure the tool is user-friendly and tailored to the unique needs of students.

## 1.3 Scope of the Project

This project is being developed without any financial investment, relying solely on freely available resources and my own time. As a result, there may be limitations in the quality of third-party services used, such as OCR APIs. Additionally, this is my first time using Dash to create a dashboard, which may impact the overall quality and functionality of the final product. Given the limited time available, the project aims to deliver a functional prototype rather than a polished, professional-level product.

## 1.4 Initial Hypothesis

At the outset of the project, I assumed that the OCR API chosen would effectively and accurately extract data from receipt images. This data would then be used to generate insightful visualizations of students' spending habits, providing them with a clear understanding of their financial behaviors. However, due to

limitations in the OCR APIs explored, the project pivoted to focus on manually adding expenses while maintaining the primary emphasis on data visualization.

### 1.5 Significance of the Project

This project holds significant value both for the market of personal finance management tools and for my personal learning and development. If successful, the tool could fill a critical gap in the market by providing a financial management solution tailored specifically for students. This would not only help students manage their finances more effectively but also contribute to their overall financial literacy. From a personal perspective, this project offers an opportunity to deepen my understanding of OCR technology, data visualization techniques, and the Dash framework. Regardless of the project's final outcome, the knowledge and skills gained throughout the development process will be invaluable for my future endeavours.

## 2. Background

### 2.1 Literature Review

#### 2.1.1 Personal Finance Management for Students

Managing personal finances is a critical skill that many students find challenging as they transition into adulthood. Lusardi, Mitchell, and Curto (2010) highlight that financial literacy among young adults is alarmingly low, contributing to poor financial decisions and increased debt. This issue is particularly pronounced among university students who must juggle student loans, living expenses, and part-time work income. Research shows that equipping students with the right tools and education can significantly boost their financial literacy and management skills (Kumar, 2017).

Most financial management tools like Moss, Pleo, and Concur are designed with businesses in mind. These tools offer features such as invoice tracking, payroll management, and detailed financial reporting, which can be overwhelming for individual users. Students need simpler interfaces focused on daily expenses, personal budgeting, and savings goals. The complexity of business-oriented tools often results in a steep learning curve and unnecessary complications in personal finance management (Heath, 2020).

#### 2.1.2 Optical Character Recognition (OCR) Technology

OCR technology plays a crucial role in converting various documents, like scanned papers, PDFs, or images, into editable and searchable text. It works by analysing shapes within images, detecting patterns that correspond to letters and numbers and matching these to a database of known characters (Smith, 2007). Despite advancements, challenges persist, especially in accurately recognizing text from receipts with different formats, fonts, and image qualities.

Several OCR APIs are available, from open-source options like Tesseract to paid services such as ABBYY FineReader and Google Cloud Vision OCR. While these tools can handle basic OCR functions, their effectiveness varies significantly depending on the quality and consistency of the input data. For example, Tesseract, despite being highly flexible, may struggle with low-quality images or unusual fonts (Ray Smith, 2007).

### 2.1.3 Data Visualization Techniques

Data visualization transforms complex data into accessible and understandable visual representations, making it easier to identify patterns, trends, and outliers. Python's data visualization libraries, such as Matplotlib, Seaborn, and Plotly, are widely used for creating static, animated, and interactive visualizations (Hunter, 2007; Waskom, 2021).

- **Matplotlib**: A comprehensive library for creating various types of plots, suitable for a wide range of applications (Hunter, 2007).
- **Seaborn**: Built on top of Matplotlib, Seaborn provides a high-level interface for drawing attractive and informative statistical graphics (Waskom, 2021).
- **Plotly**: Known for its interactive plots, Plotly is powerful for developing web-based visualizations and integrates well with web frameworks like Dash (Sievert, 2020).

## 2.2 Problem Specification

### 2.2.1 Gap in Existing Tools

There's a significant gap in personal finance management tools designed for students. Business-oriented tools often include features irrelevant or overly complex for individual use, such as multi-account handling and detailed tax implications. Students need tools that emphasize simplicity and usability, focusing on daily expense tracking, personal budgeting, and straightforward spending analytics (Heath, 2020).

A study by Tella (2018) highlighted that students often struggle with managing their finances due to inexperience and a lack of appropriate tools. Tools designed with students' unique needs in mind could significantly improve their financial literacy and management capabilities. This includes features like intuitive interfaces, visualizations of spending patterns, and personalized financial advice based on spending habits.

### 2.2.2 Challenges with OCR Implementation

A critical component of this project is using OCR technology to extract data from receipts. However, finding a capable OCR API that meets the project's requirements has been challenging. Many free OCR services lack the necessary data parsing and extraction capabilities, leading to inaccuracies in the extracted data. For instance, Tesseract, despite being one of the most popular open-source OCR engines, often struggles with

inconsistent receipt formats and varying image qualities (Smith, 2007). This led to the decision to manually input expenses while focusing on data visualization.

### 2.2.3 Focus on Data Visualization

Given the challenges with OCR implementation, the project has shifted its primary focus to data visualization. Effective data visualization can enhance the understanding of financial data, making it more accessible and actionable. Using Python's data visualization libraries, the project aims to create a user-friendly dashboard that provides clear and insightful visualizations of students' spending habits. Research by Few (2006) underscores the importance of effective data visualization in making data comprehensible and useful. Few's principles of data visualization, such as clarity, accuracy, and efficiency, are essential in designing visualizations that communicate information effectively. The project will employ these principles to create a dashboard that not only visualizes spending data but also provides actionable insights to help students manage their finances better.

### 2.2.4 User-Centric Design

To ensure the tool meets students' needs, the project will adopt a user-centric design approach, involving conducting surveys and interviews to gather user requirements and feedback. According to Norman (2013), user-centric design is crucial in developing products that are intuitive and meet users' needs. By involving students in the design process, the project aims to create a tool that is both functional and user-friendly.

### 2.2.5 Iterative Development and Feedback

The project will follow an iterative development process, incorporating user feedback at each stage. This approach is supported by research from Preece, Rogers, and Sharp (2015), who argue that iterative design and prototyping are essential in developing effective interactive systems. By continually refining the tool based on user feedback, the project aims to ensure the final product is well-aligned with user expectations and needs.

### 2.2.6 Ethical and Legal Considerations

Ethical and legal considerations are also critical in this project. Since the project involves collecting user feedback and survey data, ensuring the anonymity and privacy of participants is paramount. Ethical approval will be sought to ensure compliance with university guidelines and standards. Additionally, using open-source OCR and data visualization libraries will help avoid potential legal issues related to intellectual property rights (Ray Smith, 2007).

### 2.2.7 Conclusion

In conclusion, this project addresses a significant gap in personal finance management tools for students. By leveraging data visualization techniques and focusing on user-centric design, the project aims to create a tool that helps students better understand and manage their finances. Despite the challenges with OCR implementation, the project's emphasis on data visualization ensures it can still provide valuable insights to users. Through iterative development and user feedback, the project will refine the tool to meet the needs and expectations of students, ultimately contributing to their financial literacy and management skills.

# 3. Methodology & Design

## 3.1 Overall Methodology

The project adopted an iterative development approach, which is particularly effective for software development projects that require continuous feedback and refinement. The iterative approach involves developing the project in small sections, testing each section, and then refining it based on feedback before moving on to the next section. This method ensures that the final product aligns more closely with user needs and expectations.

### 3.1.1 Iterative Development

The iterative development process was structured around several key steps:

1. **Requirement Gathering**:
   I began by distributing questionnaires to gather detailed information about students' financial management habits. This was done with the participants' consent, ensuring ethical data collection. The questionnaires were designed to understand the specific needs and preferences of students regarding a financial management tool. This initial step was crucial in laying the foundation for the project's objectives and scope.
2. **Initial Prototype Development**:
   Based on the gathered requirements, I created initial wireframes and prototypes of the financial dashboard. These wireframes served as a blueprint, outlining the basic structure and layout of the dashboard. The prototypes were interactive, providing a tangible representation of the envisioned tool. This phase focused on translating the gathered requirements into a visual and functional format that could be tested and refined.
3. **User Testing and Feedback**:
   I conducted user testing sessions where participants interacted with the prototype and provided feedback. These sessions were essential for understanding the usability and functionality of the tool from the users' perspective. Participants' feedback was collected through follow-up questionnaires, allowing me to identify areas for improvement and prioritize changes.
4. **Feedback Analysis and Refinement**:

The collected feedback was thoroughly analysed to pinpoint the most critical and common issues. This analysis helped in making informed decisions about necessary adjustments and enhancements to the prototype. Prioritizing changes based on user feedback ensured that the tool evolved in a direction that closely matched user expectations.

5. **Prototype Refinement**:
   I made the necessary adjustments and enhancements to the prototype based on the feedback received. This step is iterative, meaning that I repeatedly refined the prototype and conducted further user testing if time allowed. However, since it is difficult to take user feedback for every iteration, I used my supervisor's knowledge and experience to fill this gap and took one final user feedback before the deadline. This cycle of development, testing, feedback, and refinement continued until the tool met the desired standards of functionality and usability.
6. **Final Review**:
   After several iterations, the refined prototype underwent a final review. This review included a comprehensive evaluation of the tool's features, usability, and overall performance. The final review ensured that all critical functionalities were correctly implemented and that the tool provided a user-friendly experience.

By following this iterative development approach, I was able to create a financial management tool that closely aligns with the needs and expectations of students. This methodology emphasized continuous improvement and user-centric design, which are crucial for developing effective software solutions.

## 3.2 Design

The design phase was a critical part of the project, involving the creation of detailed wireframes and prototypes to guide the development process. The goal was to ensure that all necessary features and functionalities were included in a user-friendly interface.

### 3.2.1 Functional and Non-Functional Requirements

To better understand and analyse the requirements, I identified and categorized them into must-have functional requirements, could-have functional requirements, and non-functional requirements based on the initial questionnaires:

**Must Have Functional Requirements:**

| Requirement Reference | Function | Criteria |
|---|---|---|

| | | |
|---|---|---|
| **MFR1** | OCR Data Extraction | Accurately extract text from receipt images using OCR technology, supporting multiple formats and qualities. |
| **MFR2** | Expense Tracking | Allow manual entry and categorization of expenses, or automatic extraction and categorization via OCR data. |
| **MFR3** | Budget Management | Enable setting, tracking, and alerts for weekly and monthly budgets. Alerts when spending approaches set limits. |
| **MFR4** | Data Visualization | Implement interactive bar graphs, pie charts, and line charts. Visualization should allow detailed drill-down capabilities. |
| **MFR5** | Customizable Dashboard | Dashboard should allow users to select and arrange financial metrics and visualizations according to their preference. |
| **MFR6** | Historical Analysis | Provide capabilities to compare spending over different periods (weekly, monthly, yearly) with visual trends. |

### Could Have Functional Requirements:

| Requirement Reference | Function | Criteria |
|---|---|---|
| **CFR1** | Reporting | Automatically generate reports on spending habits, budget adherence, and highlight financial insights. |
| **CFR2** | User Feedback System | Users can send their feedback on the tool's usability and features. |
| **CFR3** | Saving Goals | Users can set and track their savings goals, with progress visualizations and milestone alerts. |
| **CFR4** | Category Expense | Shows expenses by category. |

### Non-Functional Requirements:

| Requirement Reference | Function | Criteria |
|---|---|---|
| **NFR1** | Security | Implement encryption and secure authentication mechanisms to protect user data. |
| **NFR2** | Usability | Interface must be intuitive, easy to navigate, and designed for users with no financial expertise. |
| **NFR3** | Performance | Application must exhibit fast response times and efficient data processing, particularly for data visualization. |

| | | |
|---|---|---|
| **NFR4** | Accessibility | Ensure accessibility for users with disabilities. |
| **NFR5** | Mobile Optimization | Optimize for mobile use, ensuring full functionality and performance on smartphones and tablets. |
| **NFR6** | Compatibility | Ensure full compatibility across major operating systems (iOS, Android) and web browsers. |

### 3.2.2 Wireframes and Prototypes

Wireframes and prototypes played a crucial role in visualizing the design and layout of the financial dashboard. These preliminary designs served as blueprints, outlining the structure and placement of various elements within the dashboard.

1. **Wireframes**:

   I created wireframes to outline the basic structure and layout of the dashboard. These wireframes included placeholders for key elements such as charts, graphs, and input fields. The wireframes provided a clear visual representation of the dashboard's overall design, helping to identify the placement of different components and their relationships. [Refer to Appendix for wireframes]

2. **Prototypes**:

   Based on the wireframes, I developed interactive prototypes using Uizard, an online prototyping tool, building an interactive prototype. The prototypes were designed to be functional, allowing users to interact with different elements and get a feel of how the final tool would operate. Prototypes included interactive elements like buttons, dropdown menus, and nav-bar to change between different designed pages, providing a realistic preview of the dashboard's functionality. This prototype doesn't show any charts or any data it is mainly used to just demonstrate the dashboard will look and function. [Refer to Appendix for prototypes]

### 3.2.3 Design Decisions

Several key decisions were made during the design phase to ensure that the financial dashboard was both functional and user-friendly. These decisions were influenced by the requirements gathered from users and the goals of the project.

1. **Choice of Framework**:

   I chose Dash for two main reasons: it is based on Python, which is my strongest programming language, and it provides extensive capabilities for data visualization. Dash is well-suited for creating interactive and visually appealing web applications, making it an ideal choice for this project.

2. **User-Friendly Interface**:

The design aimed to create a user-friendly interface that allows students to easily track their expenses, set budgets, and visualize their financial data. Simplicity and intuitiveness were key considerations, ensuring that users could navigate the dashboard without any prior financial expertise.

3. **Key Features in the Design**:
    - **Overview Section**: Displays the total amount spent and the remaining budget. This section provides a quick summary of the user's financial status, making it easy to understand at a glance.
    - **Expenses Section**: Shows expense trends over time with line charts. This section helps users analyze their spending patterns and identify trends.
    - **Budget Section**: Enables users to set and track their budgets for different categories. This section includes visualizations that show budget adherence, helping users stay on track with their financial goals.

### 3.2.4 Visual Design and Layout

The visual design and layout of the dashboard were carefully planned to ensure a clean and organized presentation of financial data. The layout was designed to be intuitive, with a focus on ease of use and accessibility.

1. **Visual Hierarchy**:
   A clear visual hierarchy was established to guide users' attention to the most important elements first. Key metrics such as total amount spent and remaining budget were prominently displayed at the top of the dashboard. Supporting information, like detailed expense breakdowns and budget visualizations, were positioned below the key metrics.
2. **Color Scheme and Typography**:
   A consistent color scheme and typography were used throughout the dashboard to create a cohesive and visually appealing interface. Colors were chosen to enhance readability and highlight important information. Typography was selected for its clarity and legibility, ensuring that all text was easy to read.
3. **Interactive Elements**:
   The dashboard included several interactive elements, such as dropdown button to function as filter, navigation bar, buttons, and time pickers. These elements allowed users to filter and customize the displayed data according to their preferences. For example, users could select a specific time period or category to view detailed expense information.

### 3.2.5 Design Tool

Th design tool, Uizard, used to create the wireframes and prototypes. This tool facilitated the design process, allowing for efficient creation and iteration of the dashboard's visual elements.

1. **Wireframing Tools**:
   I used Uizard to create the initial wireframes. These tools provided an easy-to-use interface for designing the basic layout and structure of the dashboard. Wireframing tools allowed for quick adjustments and iterations based on feedback.
2. **Prototyping Tools**:
   Prototyping tool, Uizard, was used to develop interactive prototypes. These tools enabled the creation of functional prototypes with clickable elements and dynamic interactions. Prototyping tools allowed for a realistic preview of the dashboard's functionality, helping to identify potential usability issues early in the design process.

### 3.2.6 Key Design Features and Components

The design of the financial dashboard included several key features and components, each serving a specific purpose in helping users manage their finances effectively.

1. **Dashboard Overview**:
   The dashboard overview provided a summary of the user's financial status, including total amount spent, remaining budget, and a breakdown of expenses by category. This section was designed to give users a quick snapshot of their financial health.
2. **Detailed Expense Tracking**:
   The expense tracking section allowed users to view detailed information about their spending habits. This included visualizations of expense trends over time, expense breakdowns by category, and a heatmap of expenses. These visualizations helped users identify patterns and make informed financial decisions.
3. **Budget Management**:
   The budget management section enabled users to set and track their budgets for different categories. Users could specify budget limits, track their spending against these limits, and receive alerts when nearing or exceeding their budgets. This section included visual progress bars and alerts to help users stay on track with their financial goals.
4. **Customizable Dashboard**:
   The dashboard was designed to be customizable, allowing users to select and arrange financial metrics and visualizations according to their preferences. This flexibility ensured that users could tailor the dashboard to their specific needs and priorities.
5. **Historical Analysis**:
   The historical analysis feature provided capabilities to compare spending over different periods (weekly, monthly, yearly). This feature included visual trends that helped users understand how their spending habits changed over time.

## 3.3 Database Design

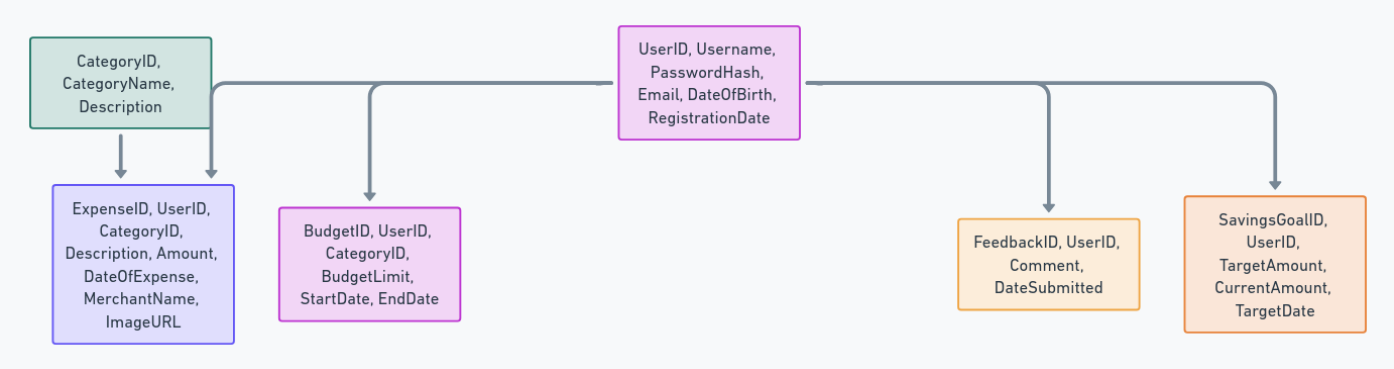
The database schema was a crucial part of the project's implementation, ensuring that all necessary data could be stored and retrieved efficiently. The design aimed to support various functionalities such as expense tracking, budgeting, and data visualization.

### 3.3.1 Schema Design

The database schema was designed to support the key functionalities of the financial management tool. MongoDB was chosen for its flexibility and scalability, making it suitable for handling diverse and complex data structures.

1. **Users Collection**:
   - Stores user information including username, email, password hash, and registration date.
   - Ensures secure storage of user credentials using encryption.
2. **Expenses Collection**:
   - Stores details of user expenses such as amount, category, description, and date of expense.
   - Supports manual entry and potential OCR data extraction.
3. **Categories Collection**:
   - Stores predefined and user-defined expense categories.
   - Helps in categorizing expenses for better data visualization.
4. **Budgets Collection**:
   - Stores user-defined budget limits for different categories and time periods.
   - Supports alerts when spending approaches set limits.
5. **Feedback Collection**:
   - Stores user feedback on the tool's usability and features.
   - Aids in iterative improvement based on user suggestions.
6. **Savings Goals Collection** (planned but not implemented):
   - Intended to store user-defined savings goals and track progress.
   - Would include visualizations for progress tracking and milestone alerts.

### 3.3.2 Database Schema Diagram



### 3.3.3 Implementation Details

The MongoDB database was used for its flexibility and scalability. Data was imported from **expenses.txt** and **categories.txt** files to populate the database initially. You can refer appendix 8.3.9 for Database import code snippet.

# 4. Implementation

The implementation phase of this project involved translating the design into a functional application. This process required setting up the development environment, coding the application, and integrating various components such as the database, data visualization tools, and user interface. Below is a detailed description of the implementation process, including the challenges faced and the solutions employed.

## 4.1 Setting Up the Environment

The first step in the implementation was setting up the development environment. This included installing the necessary libraries and frameworks, configuring the development environment, and preparing the initial codebase.

**Development Tools and Technologies**:

- **Programming Language**: Python
- **Framework**: Dash (for web application development)
- **Database**: MongoDB (for data storage)
- **Data Visualization**: Plotly (for creating interactive charts)
- **Other Libraries**: Flask (for backend services), Flask-Bcrypt (for password hashing), Pandas (for data manipulation), Google Cloud Vision (for OCR capabilities)

**Installing Required Packages**: To install the required packages, the following command was used:
pip install dash dash-bootstrap-components pymongo flask flask-bcrypt plotly pandas google-cloud-vision

*(Refer to Appendix 8.3.1 for a screenshot of the package installation commands)*

**Setting Up the Local MongoDB Database**: The local MongoDB database was set up with the name "financial_management" to store user data, expenses, categories, budgets, feedback, and savings goals. The database setup involved creating collections for each type of data to ensure organized and efficient data management. *(Refer to Appendix 8.3.3 for the database setup code)*

### 4.2 Implementing OCR (Optical Character Recognition)

Initially, the project aimed to implement OCR to automatically extract text from receipt images, streamlining the process of adding expenses by reducing manual data entry.

**Challenges**:

- **Precision**: Free OCR APIs like OCR.space were not precise enough for accurate data extraction.
- **Cost**: The Google Vision API provided more accurate data but required payment for each extraction.

**Solution**: Due to the limitations of free OCR services and the cost implications of using a paid service, the project pivoted to allow manual addition of expenses. This ensured data accuracy and kept the project within budget constraints. The Google Vision OCR code is included for reference to demonstrate its potential implementation but is not part of the working dashboard due to its reliance on a paid service. *(Refer to Appendix 8.3.2 for the OCR implementation code)*

### 4.3 Creating the Database

The database schema was designed to efficiently store and retrieve data related to users, expenses, categories, budgets, user feedback, and savings goals. MongoDB was chosen for its flexibility and scalability, allowing for efficient handling of the financial data.

**Database Collections**:

- **Users**: Stores user information including username, email, password hash, and registration date.
- **Expenses**: Stores details of user expenses such as amount, category, description, and date of expense.
- **Categories**: Stores predefined and user-defined expense categories.
- **Budgets**: Stores user-defined budget limits for different categories and time periods.
- **Feedback**: Planned but not implemented, intended to store user feedback on the tool's usability and features.
- **Savings Goals**: Intended to store user-defined savings goals and track progress (planned but not implemented).

The database was initially populated with data from **expenses.txt** and **categories.txt** files to simulate real-world use and test the application's functionality. *(Refer to Appendix 8.3.9 for the database import code snippet)*

```
# Importing expenses from Expenses.txt
# import_expenses('662947d472ecb2349f29cbe1') #Uncomment this to import expenses from data/expenses.txt

# Importing  categories
# import_categories() #Uncomment this to import expenses from data/categories.txt
```

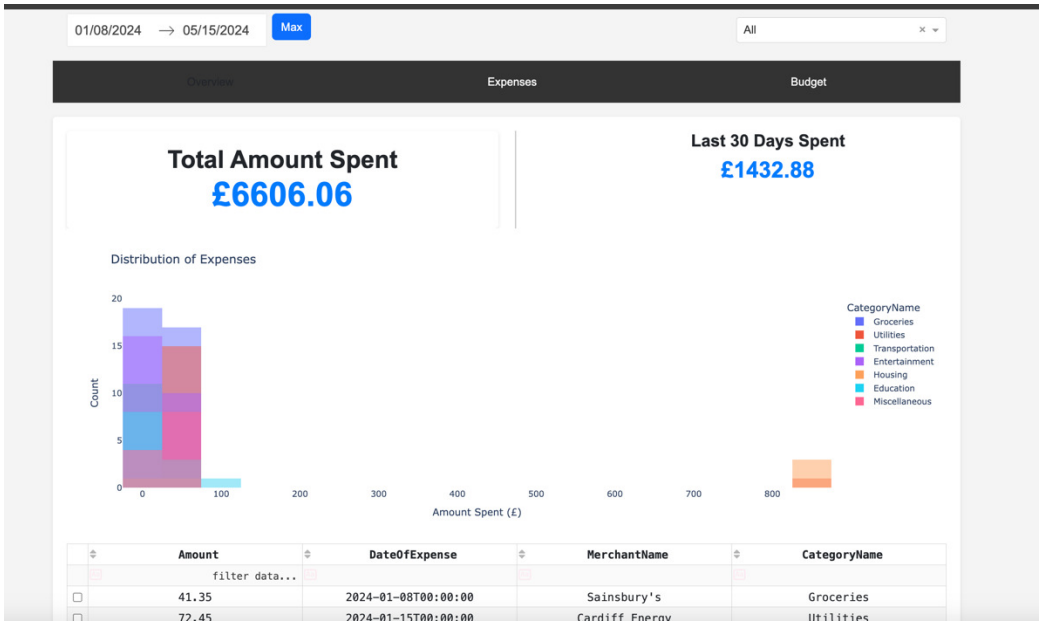
## 4.4 Developing the Front-End

The front-end of the application was developed using Dash, a Python framework for building analytical web applications. The front-end design focused on creating a user-friendly interface with interactive elements to enhance user engagement and data exploration.

**Key Components**:

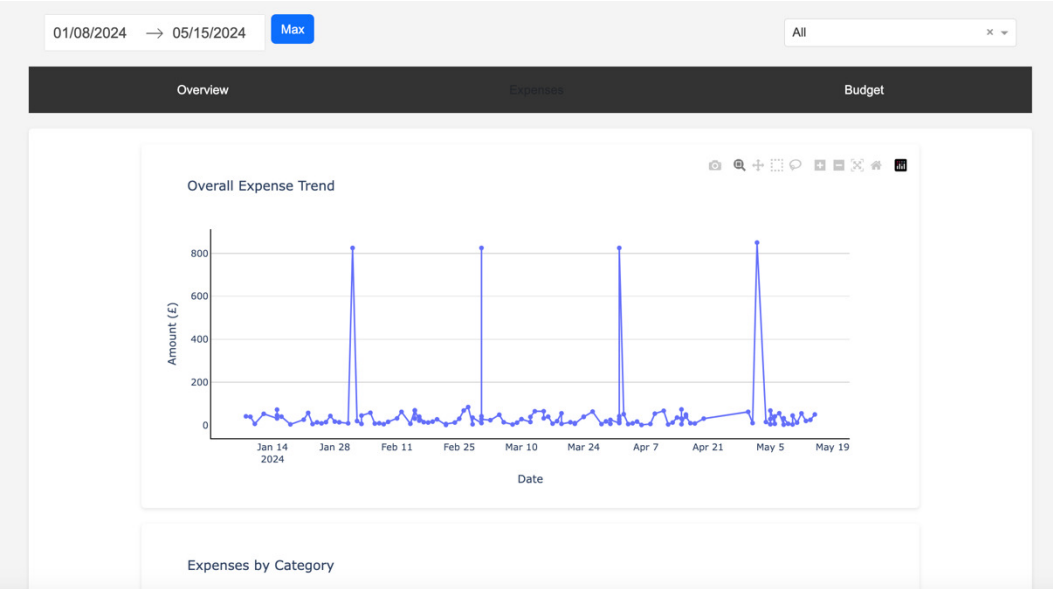
- **Overview Section**: Displays the total amount spent and the amount spent in the last 30 days. This provides users with a quick snapshot of their financial status.

```
def overview_layout(filtered_df, category_name):
    total_amount = filtered_df['Amount'].sum()
    total_amount_display = html.Div([
        html.H3('Total Amount Spent', style={'margin': '0', 'font-size'
        html.H1(f'£{total_amount:.2f}', style={'margin': '0', 'font-siz
    ], style={'text-align': 'center', 'padding': '20px', 'background-co
```



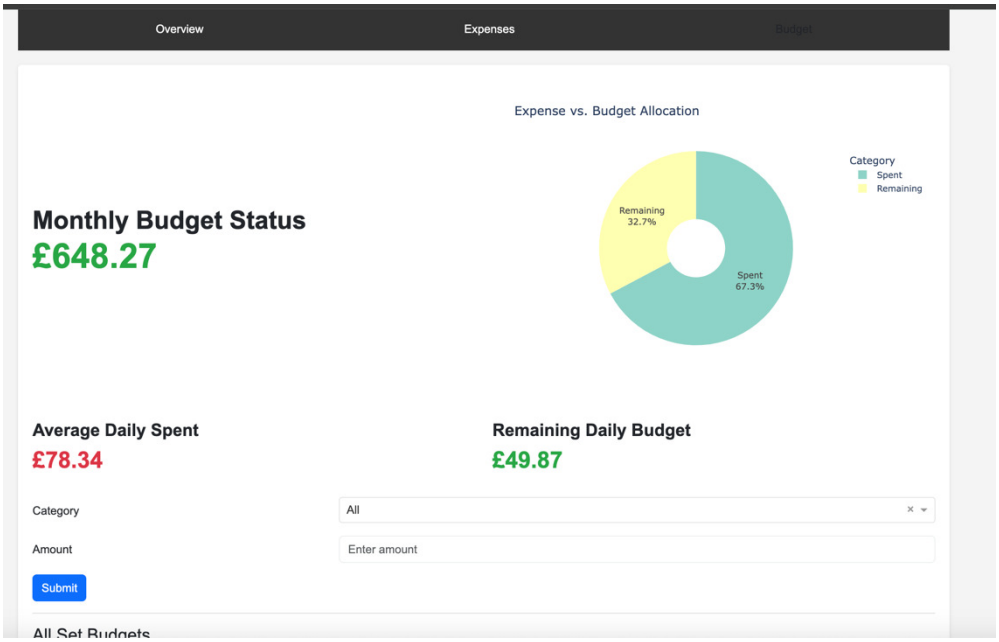
- **Expenses Section**: Shows expense trends over time and allows filtering by category and time period. Users can visualize their spending habits and identify areas for improvement.

```
def expenses_layout(sorted_df, category_name, start_date=None, end_date=None):
    # Filter data by the selected date range
    if start_date and end_date:
        start_datetime = pd.to_datetime(start_date)
        end_datetime = pd.to_datetime(end_date)
        sorted_df = sorted_df[(sorted_df['DateOfExpense'] >= start_datetime) & (sorte
```



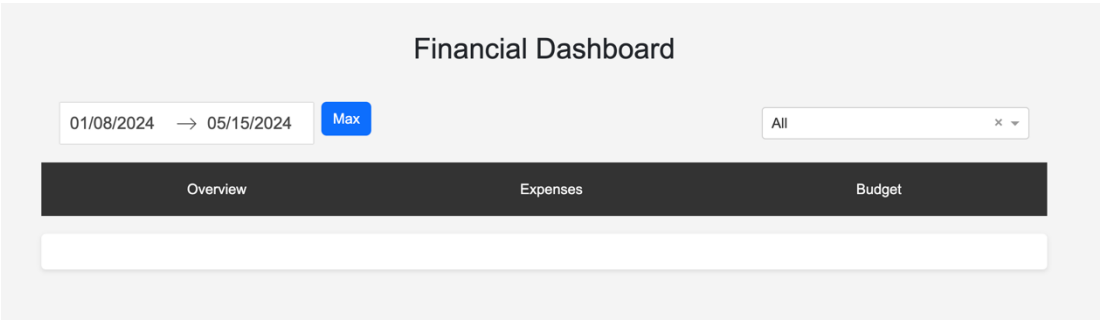

- **Budget Section**: Enables users to set and track budgets for different categories. This helps users manage their finances more effectively by setting spending limits.

```
def budget_layout(selected_category_id=None):
    categories = fetch_categories()
    category_options = [{'label': cat['CategoryName'], 'valu

    budgets = fetch_all_budgets()
```




**Design Considerations**:

- **Interactivity**: Used Dash to create interactive elements such as dropdown menus, buttons, and a time picker. These elements allow users to filter and explore their financial data dynamically.
- **Usability**: Ensured the interface was intuitive and easy to navigate, even for users with no financial expertise. This was achieved by organizing the layout logically and ensuring clear labeling and instructions. *(Refer to Appendix 8.3.4 for the front-end layout code snippet)*



## 4.5 Developing the Back-End

The back-end logic was implemented to handle data retrieval, processing, and storage. This involved creating APIs for various functionalities such as user management, expense tracking, and budget management.

**Initial Implementation**: Initially, all code was written in a single file, leading to a messy and unmanageable codebase.

**Modularization**: To improve organization, the code was modularized into separate files:

- **app.py**: Main application file that initializes the Dash app and sets up the server.
- **db.py**: Manages database interactions, including connecting to MongoDB and performing CRUD operations.
- **create_user.py**: Handles user management tasks such as creating new users. This file is run separately as an individual app.
- **add_expense_app.py**: Manages expense addition, allowing users to manually add expenses. This file is run separately as an individual app.
- **callbacks.py**: Contains callback functions for Dash interactivity, updating the user interface based on user inputs.
- **layouts.py**: Defines the layout and structure of the different sections of the dashboard.
- **style.css**: Applies styling to the application, ensuring a consistent and visually appealing interface.

This modular approach made the codebase more manageable and scalable, allowing for easier debugging and future development. *(Refer to Appendix 8.3.5 & 8.3.6 for modularization examples and back-end code snippets)*

## 4.6 Challenges and Solutions

Throughout the implementation phase, several challenges were encountered. The following outlines these challenges and the solutions employed to address them:

1. **OCR Data Extraction**:
   - **Challenge**: Free OCR APIs were imprecise, and paid services were costly.
   - **Solution**: Pivoted to manual data entry to ensure accuracy.
2. **Database Population**:
   - **Challenge**: Personal spending data was too uniform and did not represent diverse expenses.
   - **Solution**: Used an AI to generate a sample dataset of expenses over three months, providing more realistic data.
3. **Front-End and Back-End Integration**:
   - **Challenge**: Initially, the codebase was messy and unmanageable.

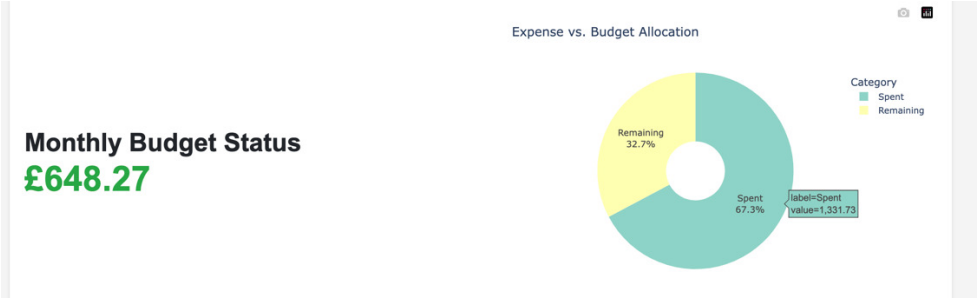
- **Solution**: Modularized the code into separate files based on functionality.

4. **Feature Implementation**:
   - **Challenge**: Limited time prevented the full implementation of certain features like creating new users and adding expenses.
   - **Solution**: Developed these features as separate apps to be integrated later.
5. **Login/Logout Feature**:
   - **Challenge**: Implementing the login/logout feature caused bugs in other parts of the dashboard.
   - **Solution**: Abandoned the feature to focus on core functionalities.
6. **Global Filters**:
   - **Challenge**: Implementing global filters for time and category was complex and caused recurring bugs.
   - **Solution**: Simplified the implementation by focusing on essential filter functionalities and refining the code to ensure stability. I used panda dataframes to manipulate the data and used it for global filter.

```
expenses_data, categories_data = fetch_data()  # Fetches data from the database
df = pd.DataFrame(expenses_data)
categories_df = pd.DataFrame(categories_data)

# Mapping dates and categories for ease of use
df['DateOfExpense'] = pd.to_datetime(df['DateOfExpense'])
category_mapping = dict(zip(categories_df['CategoryID'], categories_df['CategoryName']))
df['CategoryName'] = df['CategoryID'].map(category_mapping)

# Filtering data based on user-selected date range
start_datetime = pd.to_datetime(start_date)
end_datetime = pd.to_datetime(end_date)
filtered_df = df[(df['DateOfExpense'] >= start_datetime) & (df['DateOfExpense'] <= end_datetime)]

sorted_df = filtered_df.sort_values('DateOfExpense')
last_30_days_df = df[df['DateOfExpense'] >= (datetime.now() - timedelta(days=30))]  # Data for the last 30 days
```

7. **Budget Visualization**:
   - **Challenge**: Initially planned to implement both weekly and monthly budgets, but filtering data for both became too complex.
   - **Solution**: Focused on monthly budgets, providing a clear and straightforward visualization of budget adherence.

```
if selected_category_id is None or selected_category_id == 'all':
    selected_category_id = 1  # Default to 'All'

budget_status, total_spent, total_budget = get_budget_status("662947d472ecb2349f29cbe1", selected_category_id)
```



8. **User Feedback System**:
    - **Challenge**: Due to time constraints, the planned user feedback system was not implemented.
    - **Solution**: This feature remains a potential area for future development, where users can send their feedback on the tool's usability and features.

### 4.7 Implementation Highlights

The implementation phase resulted in a functional financial management tool with key features including expense tracking, budget management, and data visualization. The tool is designed to be user-friendly and tailored to the unique needs of students.

**Key Features Implemented**:

- Manual expense entry: Users can manually add expenses through a dedicated interface.
- Interactive data visualizations: Dynamic graphs and charts help users understand their spending habits.
- Budget setting and tracking: Users can set budgets for different categories and track their spending against these limits.
- Customizable dashboard: Users can personalize the dashboard to display the information most relevant to them.

The project code includes several key components, each responsible for different aspects of the application. By addressing the challenges and implementing the solutions outlined above, the project achieved its primary goals, resulting in a functional prototype that meets the needs of its users. Future work can build upon this foundation to further enhance the tool's capabilities.

*(Refer to Appendix 8.3 for all code snippets and screenshots discussed above)*

## 5. Result & Evaluation

### 5.1 Overview of Achievements

The project achieved several key objectives, resulting in a functional and user-friendly financial management tool tailored for students. The primary accomplishments include:

- **Dashboard Availability**: The financial dashboard is fully operational, providing users with a comprehensive tool to manage their finances.
- **Expense Tracking**: Users can manually enter and track their expenses with ease.
- **Budget Management**: The dashboard allows users to set, track, and manage their budgets effectively.

- **Customizable Dashboard**: Users have the flexibility to customize the dashboard to display the data they find most relevant.
- **Historical Analysis**: The tool offers robust historical analysis features, including expense tracking over time and heatmaps for different days of the week and month, helping users identify spending patterns.

## 5.2 Comparison with Initial Objectives

The initial objectives were ambitious, aiming to provide a comprehensive tool for financial management through OCR, data visualization, and customizable features. Here's a comparison of the objectives against the final outcomes:

### 5.2.1 Achieved Objectives

- Manual expense entry and tracking.
- Budget management with alerts for budget adherence.
- Interactive data visualizations (bar graphs, pie charts, line charts).
- Customizable dashboard features.
- Historical analysis through heatmaps.

### 5.2.2 Unmet Objectives

- **MFR1 (OCR Data Extraction)**: The OCR functionality was not implemented due to precision issues with free APIs and cost constraints with paid services.
- **CFR1 (Reporting)**: Automatic report generation was not implemented.
- **CFR2 (User Feedback System)**: A dedicated feedback system for user suggestions was not implemented.
- **CFR3 (Saving Goals)**: The feature to set and track savings goals was planned but not implemented.
- **NFR5 (Mobile Optimization)**: The dashboard was not fully optimized for mobile use.

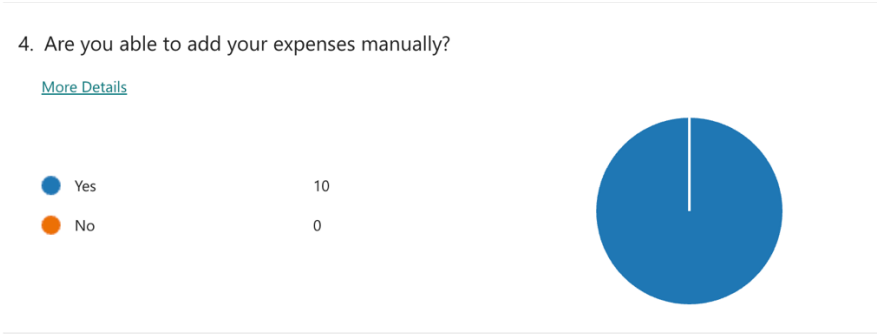
These unmet objectives will be discussed further in the future work section.

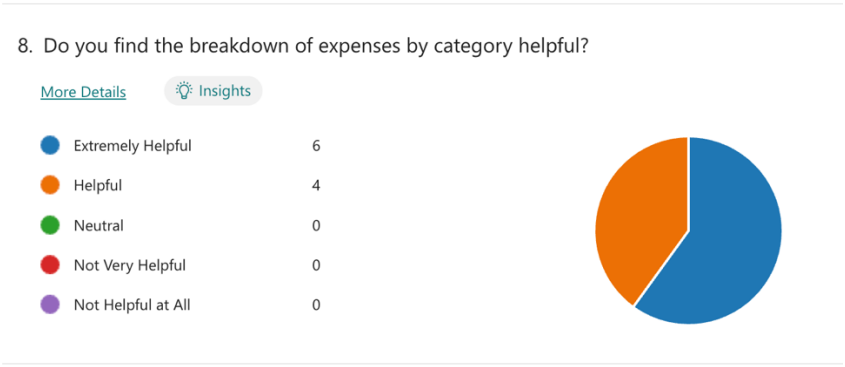
## 5.3 Testing and Validation

Testing and validation were conducted against the defined functional and non-functional requirements. The testing methodologies included manual testing, user testing, and validation against criteria defined in the requirements.

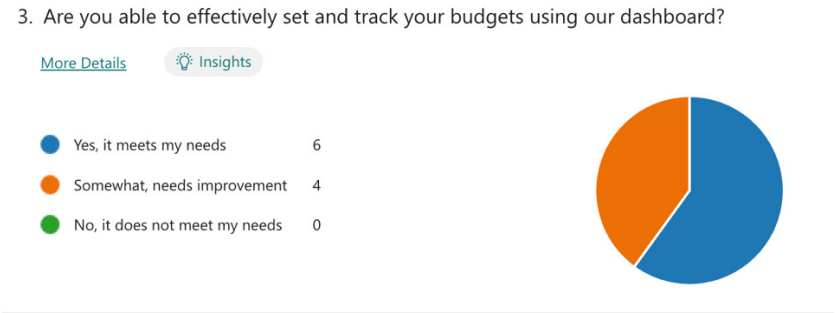
### 5.3.1 Functional Requirements Validation

- **MFR2 (Expense Tracking)**: Users confirmed the ability to manually enter and categorize expenses.

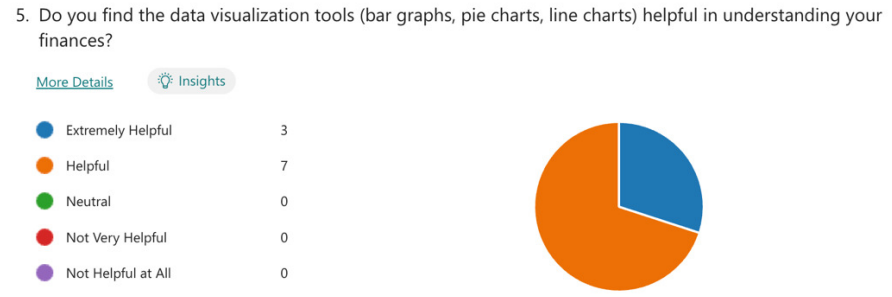


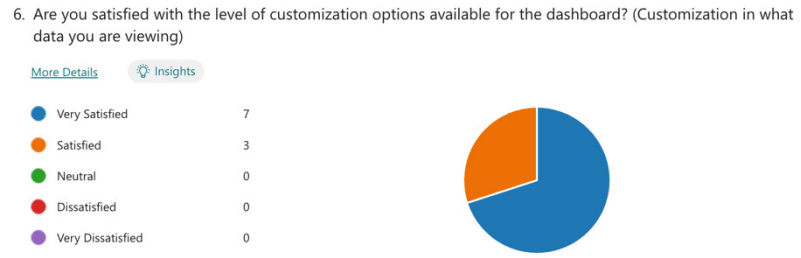

- **MFR3 (Budget Management)**: Users could set and track budgets, with alerts for approaching limits.



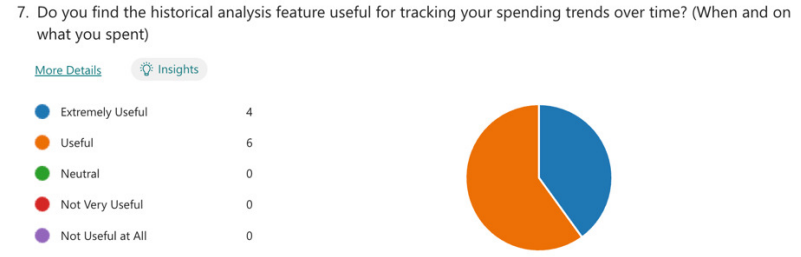

- **MFR4 (Data Visualization)**: The interactive visualizations were found to be helpful in understanding financial data.




- **MFR5 (Customizable Dashboard)**: Users were able to customize the dashboard to their preferences.



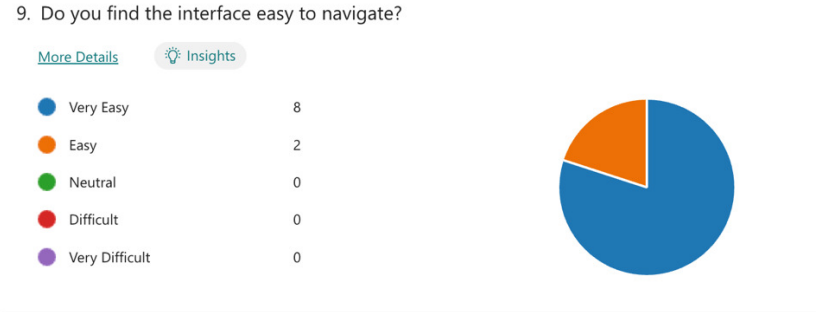

- **MFR6 (Historical Analysis)**: The heatmap and trend analysis features were validated by users.



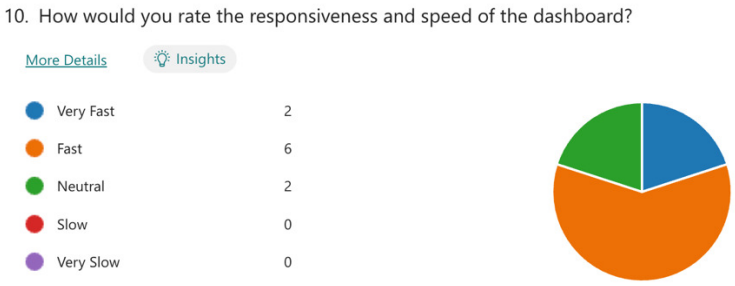

### 5.3.2 Non-Functional Requirements Validation

- **NFR1 (Security)**: Implemented secure authentication and encryption for user data. (e.g. using hash to encrypt the password before storing it in the database.)

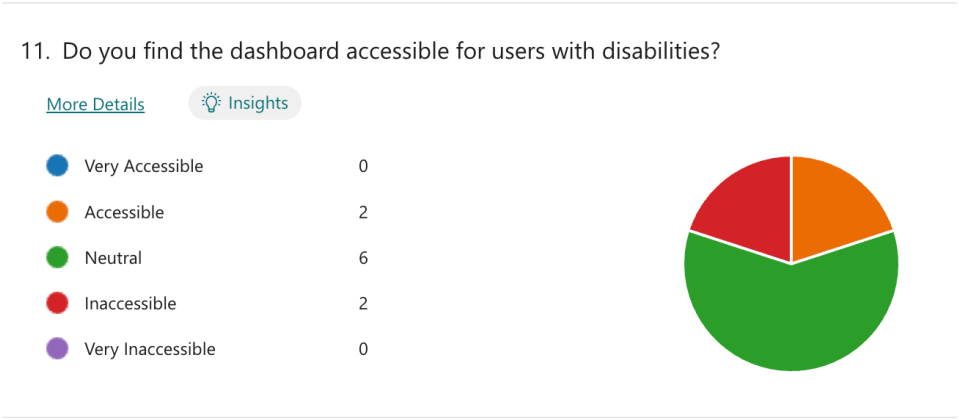

- **NFR2 (Usability)**: The interface was rated as intuitive and easy to navigate.



- **NFR3 (Performance)**: Users reported fast response times and efficient data processing.




- **NFR4 (Accessibility)**: Mixed feedback; some users found the dashboard accessible, while others noted areas for improvement.




## 5.4 User Feedback

The user feedback was collected through a questionnaire distributed to multiple participants, including a feedback session with my supervisor.

### 5.4.1 Summary of Feedback

- **Overall Experience**: Most users rated their experience as "Very good" or "Excellent."
- **Expense Tracking**: Rated as "Very Easy to Use" by most users.
- **Budget Management**: Some users found it somewhat needs improvement, while others found it met their needs.
- **Data Visualization**: Generally found helpful and useful for understanding finances.
- **Customization**: Users were satisfied with the level of customization but suggested more freedom in choosing what graphs to display in different sections.
- **Historical Analysis**: Highly valued by users for tracking spending trends.

### 5.4.2 Common Suggestions

- Adding the ability to create new categories.

- Enhancing customization options, such as changing the type of graphs displayed in the overview.
- Improving accessibility by reducing the need to reload the navigation bar after changing filters.
- Adding direct bank integration for expense data, though some users expressed privacy concerns.

### 5.4.3 Supervisor Feedback

- Suggested adding an average budget left per day and average budget spent per day of the month. (Implemented)
- Noted the need to fix the range of the graph in the overview section when a category is selected.
- Suggested that the heatmap function as historical analysis, showing expenses of the category on the particular day of the week in the month over the amount of expense.

### 5.4.4 Implemented Changes

Based on the feedback received, the following changes were implemented:

1. **Average Budget Metrics**: Added metrics to display the average budget left per day and the average budget spent per day of the month, as suggested by the supervisor.


2. **Improved Customization**: Users can now customize the dashboard to a greater extent, including selecting what data they want to see and arranging financial metrics and visualizations according to their preferences.
3. **Enhanced Historical Analysis**: The heatmap now shows the expenses of the selected category on particular days of the week over the month, providing a detailed historical analysis.
4. **UI Enhancements**: Improved the user interface based on feedback to make it more intuitive and user-friendly, such as ensuring that selected categories and time filters are retained without needing to reload the navigation bar.
5. **Accessibility Improvements**: Although some accessibility issues remain, efforts were made to improve the overall accessibility, such as adding sliders for better time and category selection.

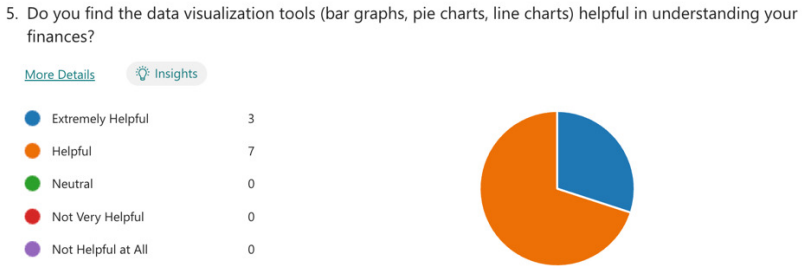
### 5.4.5 Planned Changes

- Fixing the graph range in the overview section when a category is selected.
- Further enhancing customization options, such as allowing users to select specific graphs to display in the overview.
- Implementing features for reporting, user feedback, and savings goals.
- Enhancing mobile optimization and overall accessibility.

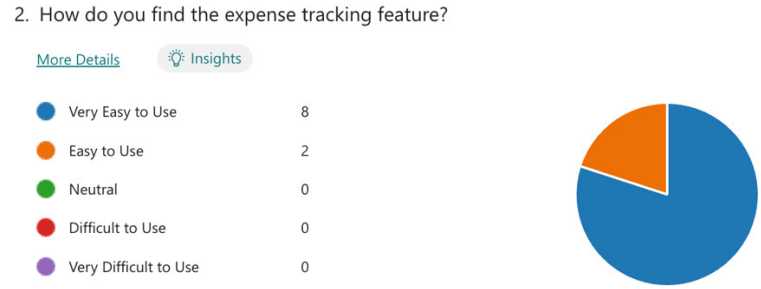
## 5.5 Critical Appraisal of Results

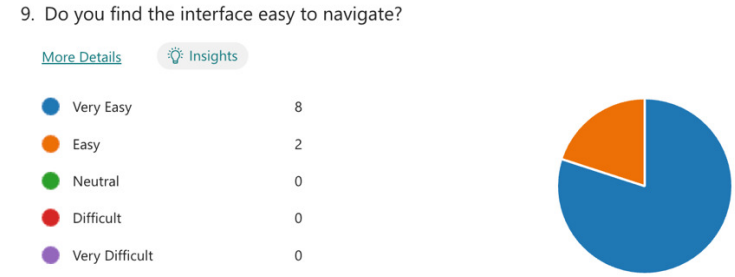
### 5.5.1 Strengths

- **Effective Data Visualization**: The dashboard provides clear and interactive visualizations, helping users understand their financial data.

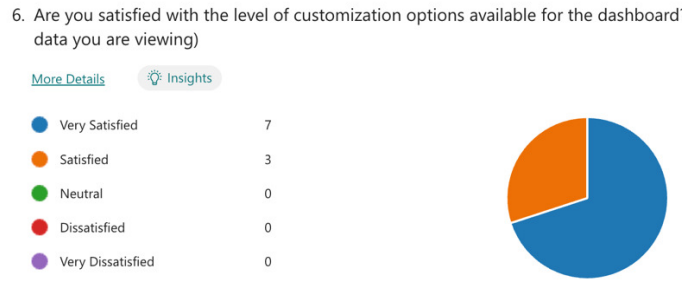



- **User-Friendly Interface**: Users found the interface intuitive and easy to navigate.



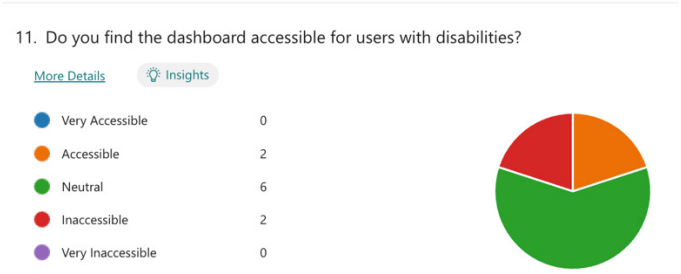

- **Customizability**: The ability to customize the dashboard was well-received.




### 5.5.2 Weaknesses

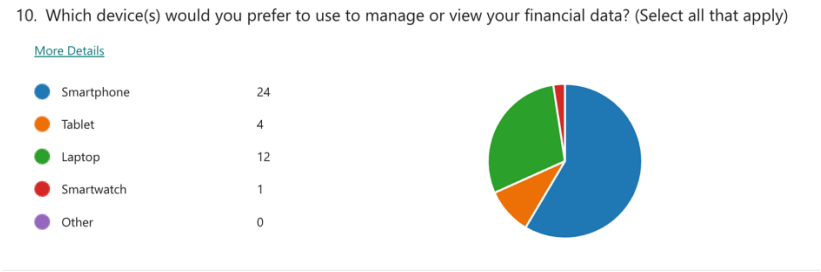
- **Unmet Objectives**: Some planned features were not implemented, impacting the overall functionality.

- **Accessibility**: Feedback indicated that the dashboard could be more accessible, particularly for users with disabilities.




### 5.5.3 Impact of Unmet Requirements

- The lack of OCR functionality limits the automation of expense tracking.
- Missing features like reporting, user feedback system, and savings goals reduce the tool's comprehensiveness.
- Mobile optimization needs to be addressed to enhance usability on smartphones and tablets. As according to the questionnaire for requirement analysis (first questionnaire), most of the users preferred to see/use the dashboard on mobile.




### 5.5.4 Future Work

- Implementing OCR for automatic data extraction.
- Adding features for reporting, user feedback, and savings goals.
- Enhancing mobile optimization and accessibility.
- Improving customization options and fixing existing usability issues.

# 6. Conclusion and Future Works

## 6.1 Summary of Findings

The financial management tool developed through this project successfully addressed the primary objective of helping students manage their finances more effectively. Several key features were implemented, including manual expense tracking, budget management, customizable dashboards, and historical analysis through data visualizations such as bar graphs, pie charts, line charts, and heatmaps. The project's iterative development approach, guided by user feedback and testing, ensured that the tool met the needs and expectations of its users.

### 6.1.1 Key Findings

- **Expense Tracking**: The manual entry system for tracking expenses proved to be user-friendly and efficient, allowing students to categorize and monitor their spending accurately.
- **Budget Management**: Users were able to set, track, and manage their budgets, receiving alerts when their spending approached set limits.
- **Data Visualization**: The interactive visualizations provided clear insights into financial data, making it easier for users to understand their spending habits and trends.
- **Customizable Dashboard**: The flexibility to customize the dashboard according to individual preferences was highly appreciated by users.
- **Historical Analysis**: The heatmap and trend analysis features offered valuable insights into spending patterns over time, helping users make informed financial decisions.

Despite the successes, some planned features, such as OCR data extraction, automatic report generation, a dedicated user feedback system, and savings goals tracking, were not implemented due to various challenges, including technical limitations and time constraints. Additionally, while the dashboard performed well on desktops, mobile optimization and certain accessibility features need further enhancement. You can refer to appendix 8.4 to see how the dashboard looks like.

## 6.2 Future Work

The future work section outlines the enhancements and additional features that could be implemented to improve the tool further. These enhancements aim to address the unmet objectives and incorporate user suggestions to create a more comprehensive and user-friendly financial management tool.

### 6.2.1 Planned Enhancements

1. **OCR Implementation**:
   - **Objective**: Integrate OCR functionality to automate the extraction of expense data from receipt images.
   - **Benefit**: Streamlining the expense entry process will reduce manual input and enhance data accuracy.
2. **Reporting**:
   - **Objective**: Develop features for automatic report generation, providing detailed insights into spending habits and budget adherence.
   - **Benefit**: Automated reports will offer users a deeper understanding of their financial health and highlight areas for improvement.

3. **User Feedback System**:
    - **Objective**: Implement a dedicated system for collecting and managing user feedback.
    - **Benefit**: A user feedback system will facilitate continuous improvement based on user suggestions and complaints, ensuring the tool evolves according to user needs.
4. **Savings Goals**:
    - **Objective**: Add features to set and track savings goals, including progress visualizations and milestone alerts.
    - **Benefit**: Helping users set and achieve savings goals will enhance the overall financial management capabilities of the tool.
5. **Mobile Optimization**:
    - **Objective**: Optimize the dashboard for use on mobile devices, ensuring full functionality and performance on smartphones and tablets.
    - **Benefit**: Mobile optimization will make the tool more accessible and convenient for users on the go.
6. **Enhanced Customization**:
    - **Objective**: Provide users with more freedom to customize the dashboard, including the ability to choose specific graphs and data visualizations.
    - **Benefit**: Increased customization options will allow users to tailor the tool to their specific needs and preferences.
7. **Accessibility Improvements**:
    - **Objective**: Enhance the accessibility features of the dashboard to better accommodate users with disabilities.
    - **Benefit**: Improving accessibility will ensure that the tool is usable by a wider range of individuals, promoting inclusivity.
8. **Fixing Graph Range Issues**:
    - **Objective**: Adjust the graph range in the overview section to display data properly when a category is selected.
    - **Benefit**: This improvement will enhance the clarity and usability of the data visualizations, ensuring users can view detailed insights without any issues.

By focusing on these enhancements, the financial management tool can be further developed into a more comprehensive, user-friendly, and inclusive application. Addressing the current limitations and incorporating user feedback will ensure that the tool continues to meet the evolving needs of students in managing their finances effectively.

# 7. Reflection on Learning

## 7.1 Technical Skills Development

Embarking on this project was both exciting and challenging. I had to dive into Dash from scratch, and although it's built on Python, which I was already comfortable with, learning Dash was a whole new ballgame. Getting the dashboard up and running in less than three months was a tough task.

Plotly was another tool I had to master to create vivid visualizations. Before this project, I underestimated how crucial data manipulation was. Working with Pandas and its DataFrames turned out to be the backbone of my visualizations. Manipulating data to get it into the right format was often more challenging than the visualization itself. It was a rewarding struggle, though. I can now confidently say that I'm much more proficient in Pandas and data manipulation.

Integrating MongoDB with Python was also new to me. I had used MongoDB in a previous module, but implementing it in my dashboard code was a fresh experience. Fortunately, this part wasn't too difficult, thanks to my prior knowledge.

## 7.2 Project Management and Process

Using an iterative development approach sounded great on paper, but in reality, it was a bit of a mixed bag. Scheduling user feedback sessions took more time than I anticipated, which delayed progress. I was so reliant on this feedback for making meaningful changes that it sometimes stalled the project.

Recognizing this bottleneck, I sought more frequent feedback from my supervisor, whose insights were invaluable. For instance, the idea of implementing global filters instead of separate filters for each tab came from her. This hierarchical structure significantly improved the user experience. Looking back, I could have managed my time better. Waiting for the AI to generate datasets took longer than expected, which also contributed to delays.

## 7.3 Problem-Solving and Adaptability

This project threw a lot of problems my way, from small bugs to more complex issues. One of the biggest challenges was working with Pandas DataFrames. Initially, the data fetched from the database was a mess. To make the data realistic, I rearranged the expenses in the imported list, which worked fine for simple graphs but caused issues for more complex visualizations like expenses over time.

Filtering data for both time and category simultaneously was another major hurdle. After much frustration, I stumbled upon a tutorial on data manipulation using Pandas, which was a game-changer. It helped me get the data in order and allowed me to create complex visualizations, like the heatmap that filters expenses by category, time, and days of the week.

Another significant challenge was filtering budget data by time. Initially, I faced difficulties in synchronizing the data from the expenses and budgets collections in MongoDB. The primary problem was ensuring that only the active budgets were displayed, which involved checking if the budget's start date was less than today and the end date was more than today. This was complicated by the fact that different categories could have varying start and end dates for their budgets.

To solve this, I decided to standardize the budget periods. Now, users can set only one budget per category for each month, regardless of when in the month the budget is set. This means that the budget will always be counted from the start of the month to the end of the month, ensuring consistency and simplifying the filtering process. This solution effectively resolved the synchronization issue and made the budget visualization much more straightforward.

*(Refer to Appendix 8.3.12 for code snippets and screenshots of Pandas DataFrame and filter budget data)*

## 7.4 User-Centric Design

User feedback was crucial in shaping this project. The responses from questionnaires and user testing sessions provided valuable insights into how the tool was being used and what improvements were needed.

For example, users suggested the need for global filters and better data visualization options. These suggestions led to significant changes in the design and functionality of the dashboard. My supervisor’s feedback was instrumental as well. Her constructive criticism and suggestions, such as implementing hierarchical global filters, greatly enhanced the usability of the tool.

## 7.5 Collaboration and Communication

Regular meetings with my supervisor were incredibly beneficial. At the start of the project, I was unsure about the scope and direction. My supervisor was extremely patient and provided much-needed guidance. Her constructive feedback helped me stay on track and continuously improve the project. Her regular check-ins kept me accountable and ensured steady progress.

Apart from my supervisor, my interactions with peers were limited to gathering responses for questionnaires and user feedback. These interactions were essential for understanding user needs and preferences. Despite time constraints, my supervisor's suggestions towards the end of the project were valuable, although I couldn’t implement all of them due to the lack of time.

## 7.6 Personal Growth

This project has been a significant contributor to my personal and professional growth. It not only enhanced my programming skills but also provided valuable experience in project management and problem-solving. I now have a deeper understanding of the importance of synthetic data in project development. Synthetic data offers numerous benefits, such as ensuring privacy, reducing bias, and providing comprehensive datasets for testing and validation.

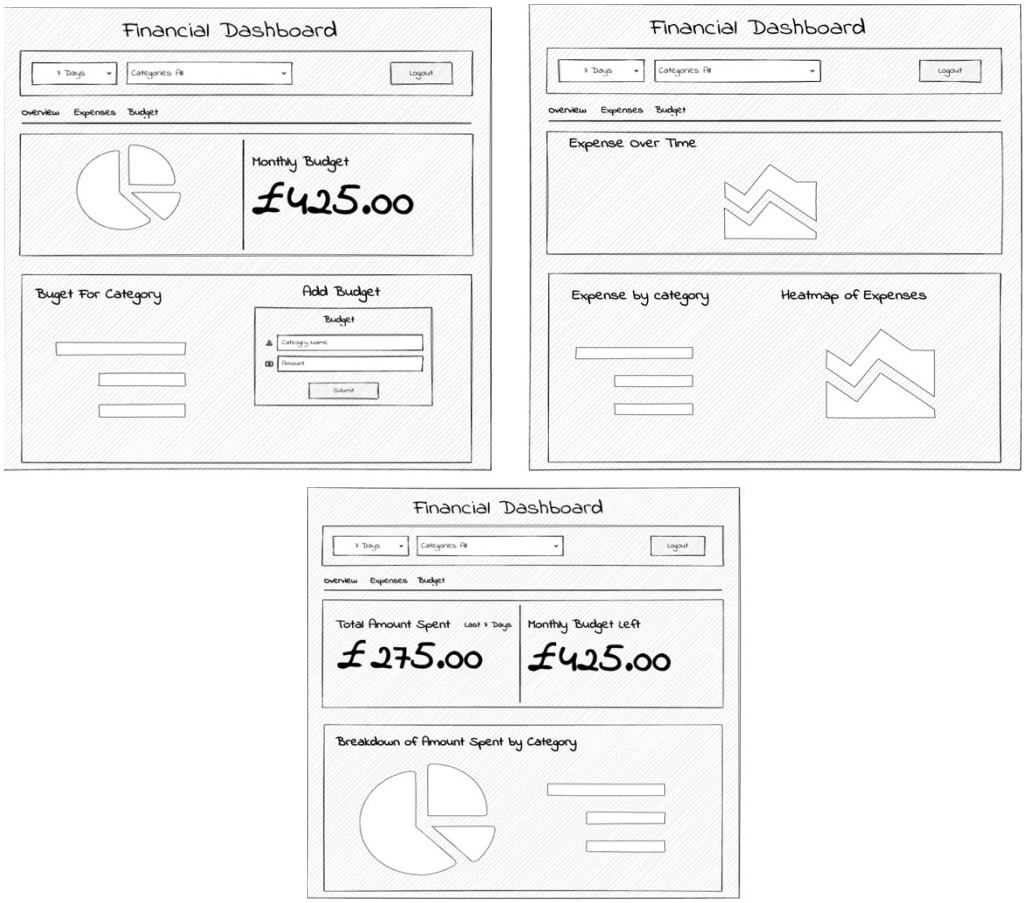
Before this project, I was unaware of the potential of synthetic data, but now I understand its importance and practical applications. This project also taught me the value of adaptability and learning new skills quickly. The experience of overcoming various challenges and finding solutions has boosted my confidence and prepared me for future projects.

Additionally, this project has improved my ability to document and present my work effectively. The process of writing detailed reports and reflecting on my learning has honed my communication skills. Moving forward, I feel better equipped to handle similar projects, manage time more effectively, and utilize user feedback to create more user-centric designs.

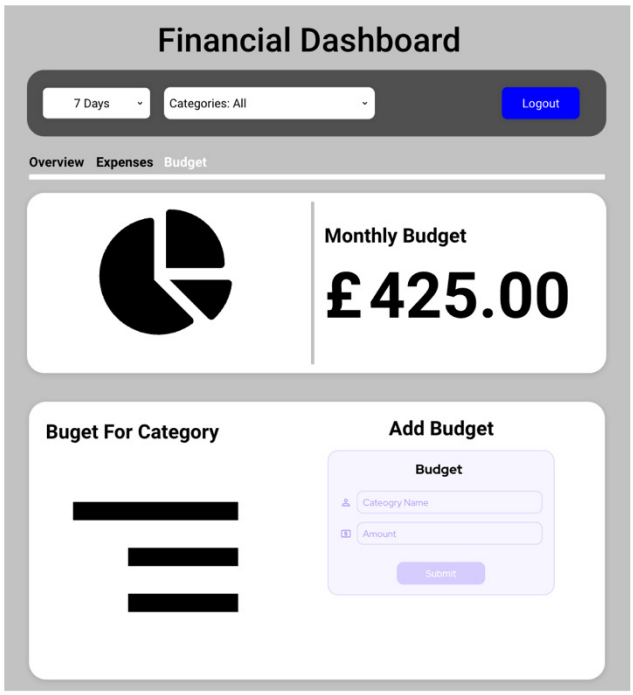
In conclusion, this project has been a transformative experience, providing me with valuable skills and insights that will be beneficial in my future career. The lessons learned and the challenges overcome have made me a more proficient and confident developer.

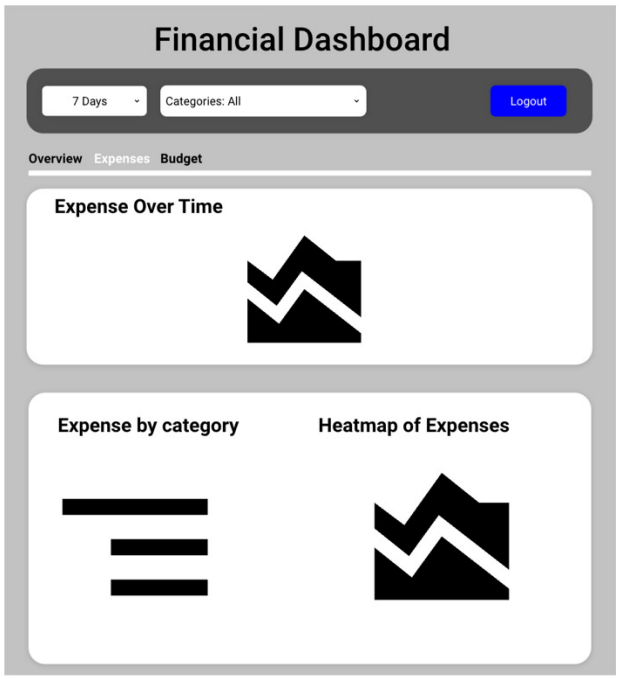
# 8. Appendix

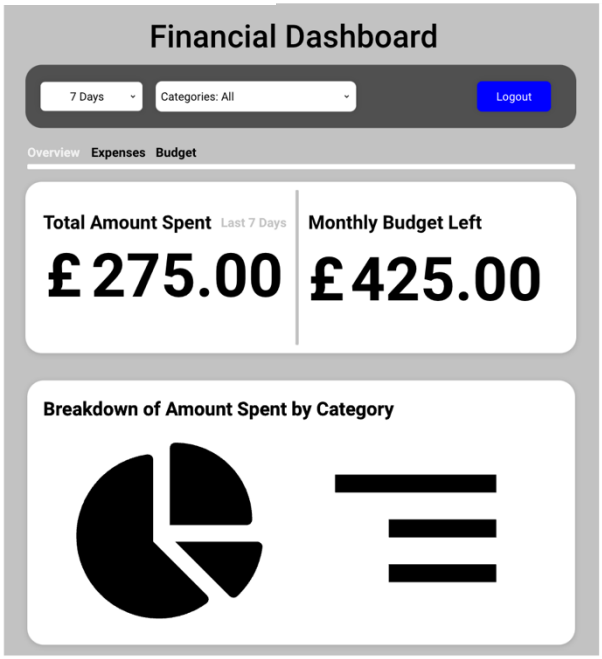
## 8.1 Dashboard Wireframe



## 8.2 Dashboard Prototype




## 8.3 Code Snippets and Screenshots

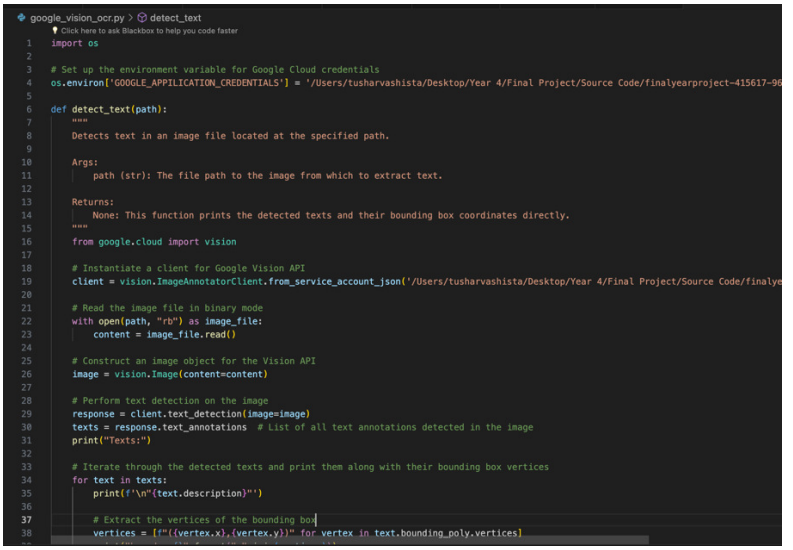
### 8.3.1 Setting Up the Environment

**Installing Required Packages**: Screenshot of the package installation commands.

```bash
pip install dash dash-bootstrap-components pymongo flask flask-bcrypt plotly pandas google-cloud-vision
```

### 8.3.2 Implementing OCR

Google Vision API Initialization and Text Extraction: Screenshot of the google_vision_ocr.py file showing the implementation of OCR.



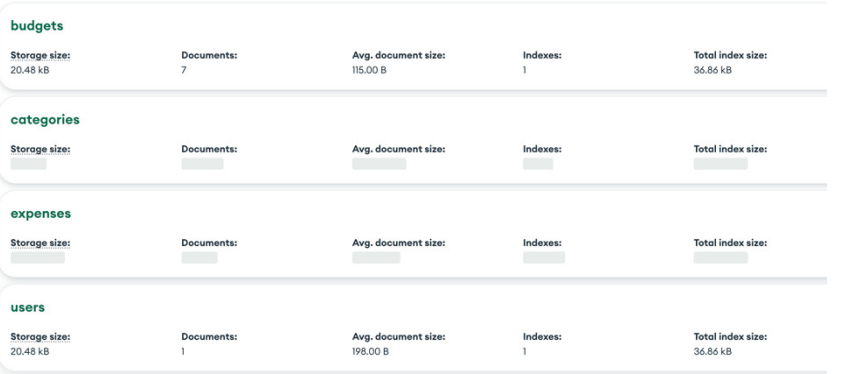
```python
import os

# Set up the environment variable for Google Cloud credentials
os.environ['GOOGLE_APPLICATION_CREDENTIALS'] = '/Users/tusharvashista/Desktop/Year 4/Final Project/Source Code/finalyearproject-415617-96

def detect_text(path):
    """
    Detects text in an image file located at the specified path.

    Args:
        path (str): The file path to the image from which to extract text.

    Returns:
        None: This function prints the detected texts and their bounding box coordinates directly.
    """
    from google.cloud import vision

    # Instantiate a client for Google Vision API
    client = vision.ImageAnnotatorClient.from_service_account_json('/Users/tusharvashista/Desktop/Year 4/Final Project/Source Code/finalye

    # Read the image file in binary mode
    with open(path, "rb") as image_file:
        content = image_file.read()

    # Construct an image object for the Vision API
    image = vision.Image(content=content)

    # Perform text detection on the image
    response = client.text_detection(image=image)
    texts = response.text_annotations  # List of all text annotations detected in the image
    print("Texts:")

    # Iterate through the detected texts and print them along with their bounding box vertices
    for text in texts:
        print(f'\n"{text.description}"')

        # Extract the vertices of the bounding box
        vertices = [f"({vertex.x},{vertex.y})" for vertex in text.bounding_poly.vertices]
```

### 8.3.3 Creating Database

MongoDB Connection Setup and Database Initialization: Screenshot of the db.py file showing the MongoDB connection and initialization.

```python
# Function to retrieve the database connection
def get_db():
    client = MongoClient('mongodb://localhost:27017/')
    return client['financial_management']

# Function to fetch expenses and category data from MongoDB and return as DataFrames
def fetch_data():
    db = get_db()
    expenses_data = list(db.expenses.find({}, {'_id': 0, 'Amount': 1, 'DateOfExpense': 1, 'CategoryID': 1, 'MerchantName': 1}))
    categories_data = list(db.categories.find({}, {'_id': 0}))

    expenses_df = pd.DataFrame(expenses_data)
    categories_df = pd.DataFrame(categories_data)

    # Convert date strings to datetime objects
    expenses_df['DateOfExpense'] = pd.to_datetime(expenses_df['DateOfExpense'])
    return expenses_df, categories_df

# Function to fetch all budgets from the database
def fetch_all_budgets():
    db = get_db()
    return list(db.budgets.find({}, {'_id': 0, 'CategoryID': 1, 'BudgetLimit': 1, 'StartDate': 1, 'EndDate': 1, 'Period': 1}))


# Function to fetch all categories from the database
def fetch_categories():
    db = get_db()
    categories = list(db.categories.find({}, {'_id': 0, 'CategoryID': 1, 'CategoryName': 1}))
    return categories
```

```python
# Collections
users = db['users']
expenses = db['expenses']
categories = db['categories']
budgets = db['budgets']
feedback = db['feedback']
savings_goals = db['savings_goals']
```



### 8.3.4 Developing the Front-End

Basic Structure of the Dash App and Layout Setup: Screenshot of the app.py and layouts.py file showing the Dash app structure and layout.

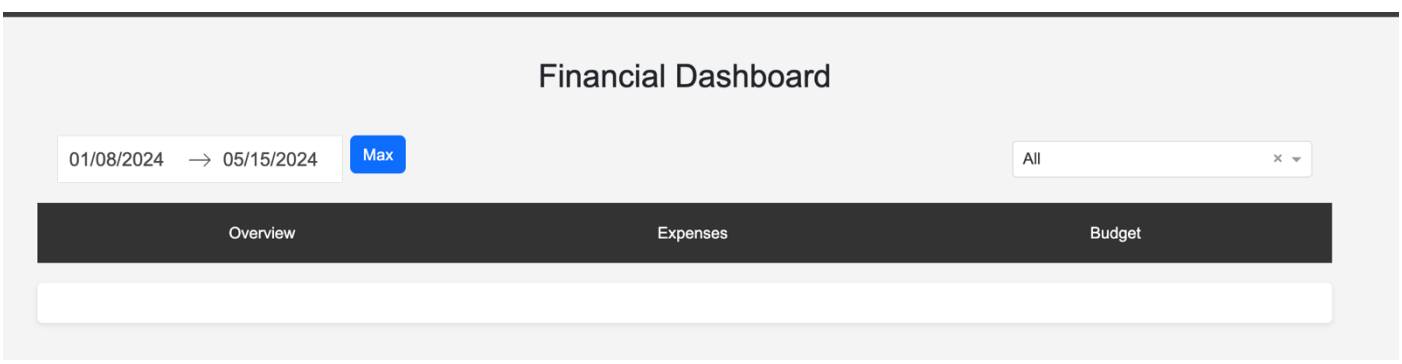
app.py > ...

Click here to ask Blackbox to help you code faster

```python
# Import necessary libraries
from dash import Dash  # Main Dash app
import dash_bootstrap_components as dbc  # Bootstrap components for styling
from layouts import create_layout  # Importing layout setup from layouts.py
from callbacks import register_callbacks  # Importing callback registration function from callbacks.py

# Create a Dash application instance
# Configure the application to use Bootstrap for styling
# suppress_callback_exceptions=True allows us to omit outputs in the callbacks during initialization
app = Dash(__name__, external_stylesheets=[dbc.themes.BOOTSTRAP], suppress_callback_exceptions=True)

# Set the layout of the app using a function from the layouts module
app.layout = create_layout()

# Register callbacks for interactive components using a function from the callbacks module
register_callbacks(app)

# Check if the script is the main program and not being imported elsewhere
if __name__ == '__main__':
    # Run the app with debug=True to enable auto-reloading and better error messages
    app.run_server(debug=True)

```

layouts.py > ...

```python
# Function to create the overall layout of the dashboard
def create_layout():
    expenses_data, categories_data = fetch_data()  # Fetch data from the database
    expenses_df = pd.DataFrame(expenses_data)  # Convert expenses data to a DataFrame
    categories_df = pd.DataFrame(categories_data)  # Convert category data to a DataFrame
    expenses_df['DateOfExpense'] = pd.to_datetime(expenses_df['DateOfExpense'])  # Convert expense dates from string to datetime objects
    category_mapping = dict(zip(categories_df['CategoryID'], categories_df['CategoryName']))  # Create a map of category IDs to names
    expenses_df['CategoryName'] = expenses_df['CategoryID'].map(category_mapping)  # Map category names to expenses DataFrame

    # Define the structure of HTML layout
    return html.Div(className='container', children=[
        html.Div(className='title-container', children=[
            html.H1("Financial Dashboard", className='title')  # Main title of the dashboard
        ]),
        html.Div(className='filters-container', style={'display': 'flex', 'justify-content': 'space-between', 'align-items': 'center'}, childr
            html.Div([
                dcc.DatePickerRange(
                    id='date-picker-range',
                    start_date=expenses_df['DateOfExpense'].min(),  # Set the minimum selectable date
                    end_date=expenses_df['DateOfExpense'].max()  # Set the maximum selectable date
                ),
                dbc.Button("Max", id='max-date-range', n_clicks=0, className='ms-2', style={'height': '38px'})
            ], style={'display': 'flex'}),
            dcc.Dropdown(
                id='category-dropdown',
                options=[{'label': 'All', 'value': 'all'}] + [{'label': cat, 'value': cat} for cat in expenses_df['CategoryName'].unique()],
                placeholder='Select a Category',
                value='all',
                style={'width': '300px'}
            )
        ]),
        html.Div([
            html.Button('Overview', id='overview-tab', n_clicks=0, className='nav-link'),
            html.Button('Expenses', id='expenses-tab', n_clicks=0, className='nav-link'),
            html.Button('Budget', id='budget-tab', n_clicks=0, className='nav-link'),
        ], className='navbar'),
        html.Div(id='content', className='content')  # Placeholder for dynamic content
    ])

```



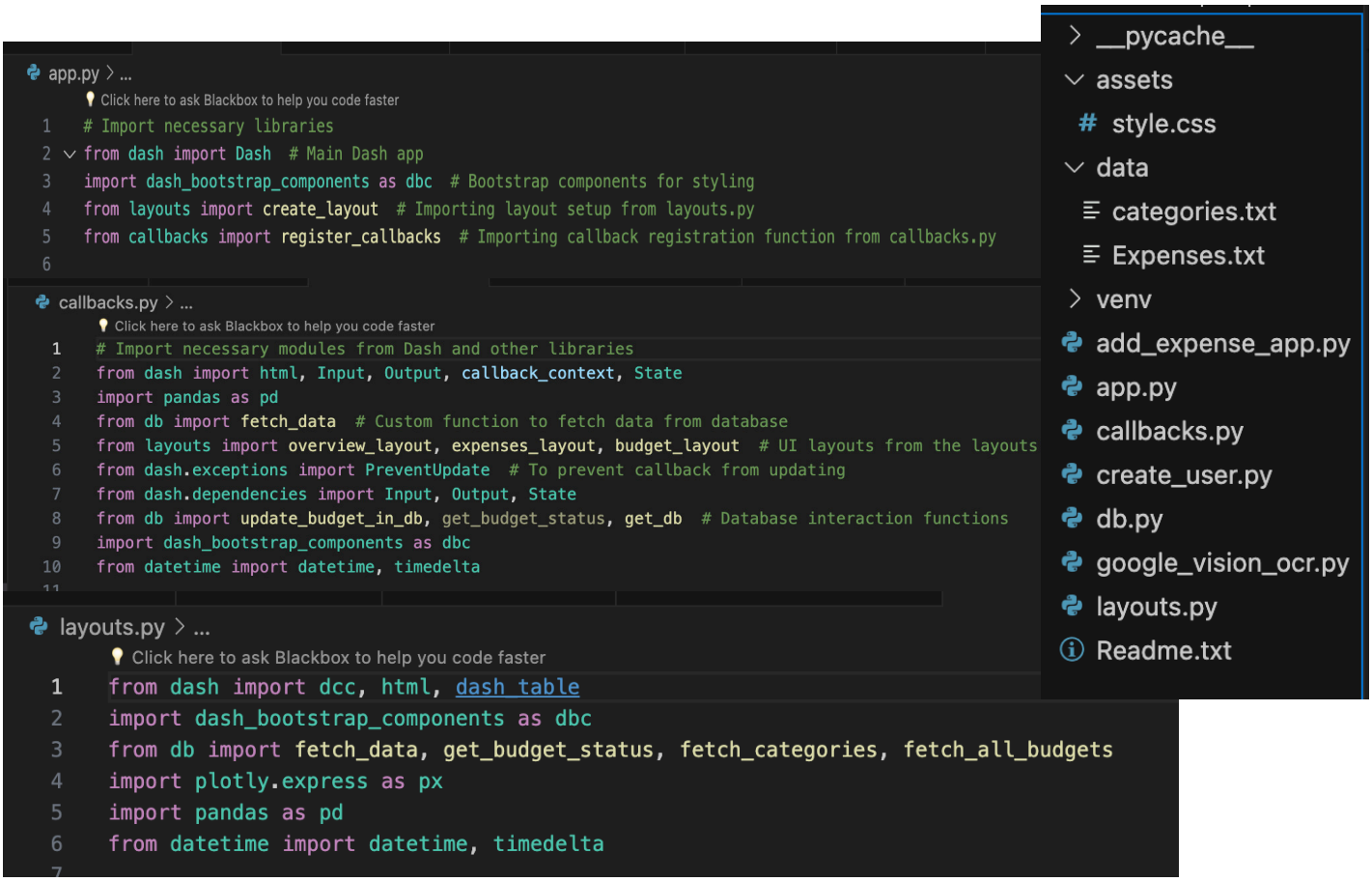
### 8.3.5 Developing the Back-End

**Callback Function**: Screenshot of the budget update callback function as an example from **callbacks.py** file.

```python
# Callback to update budget based on user inputs
@app.callback(
Output('budget-status-alert', 'children'),
[Input('submit-budget', 'n_clicks')],
[State('dropdown-category', 'value'),
 State('input-amount', 'value')]
)
def update_budget(n_clicks, category_id, amount):
    if n_clicks and n_clicks > 0 and amount is not None:  # Check if the button was clicked and amount is not None
        try:
            db = get_db()  # Retrieve the database connection
            update_budget_in_db("662947d472ecb2349f29cbe1", category_id, float(amount), db)
            return dbc.Alert("Budget updated successfully!", color="success")
        except Exception as e:
            return dbc.Alert(f"Failed to update budget: {str(e)}", color="danger")
    return ""

# Callback to update the budget display based on selected category
@app.callback(
    Output('budget-display', 'children'),
    [Input('category-dropdown', 'value')]
)
def update_budget_display(category_id):
    if not category_id:  # Handle None or invalid inputs
        category_id = 1  # Default to 'All'
    return budget_layout(category_id)
```


```python
    dbc.Form([
        dbc.Row([
            dbc.Label("Category", html_for="dropdown-category", width=4),
            dbc.Col(
                dcc.Dropdown(
                    id='dropdown-category',
                    options=category_options,
                    value=selected_category_id,
                    placeholder="Select a Category",
                    style={'width': '100%'}
                ),
                width=8
            )
        ], className='mb-3'),
        dbc.Row([
            dbc.Label("Amount", html_for="input-amount", width=4),
            dbc.Col(
                dbc.Input(type="number", id="input-amount", placeholder="Enter amount"),
                width=8
            )
        ], className='mb-3'),
        dbc.Button("Submit", id="submit-budget", color="primary", className='ml-3')
    ], className='mt-4'),
    html.Div(id='budget-status-alert'),
```

### 8.3.6 Modularizing the Code into Different Files



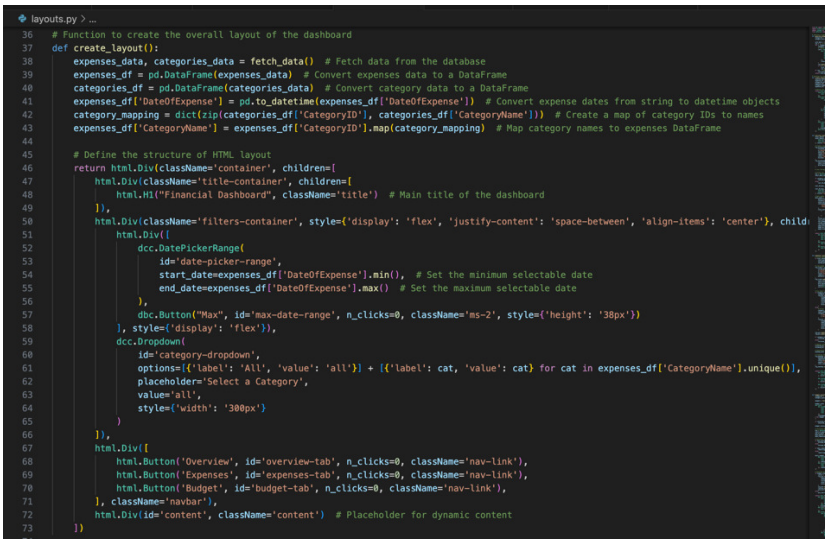
app.py
```python
# Import necessary libraries
from dash import Dash  # Main Dash app
import dash_bootstrap_components as dbc  # Bootstrap components for styling
from layouts import create_layout  # Importing layout setup from layouts.py
from callbacks import register_callbacks  # Importing callback registration function from callbacks.py
```

callbacks.py
```python
# Import necessary modules from Dash and other libraries
from dash import html, Input, Output, callback_context, State
import pandas as pd
from db import fetch_data  # Custom function to fetch data from database
from layouts import overview_layout, expenses_layout, budget_layout  # UI layouts from the layouts
from dash.exceptions import PreventUpdate  # To prevent callback from updating
from dash.dependencies import Input, Output, State
from db import update_budget_in_db, get_budget_status, get_db  # Database interaction functions
import dash_bootstrap_components as dbc
from datetime import datetime, timedelta
```

layouts.py
```python
from dash import dcc, html, dash_table
import dash_bootstrap_components as dbc
from db import fetch_data, get_budget_status, fetch_categories, fetch_all_budgets
import plotly.express as px
import pandas as pd
from datetime import datetime, timedelta
```

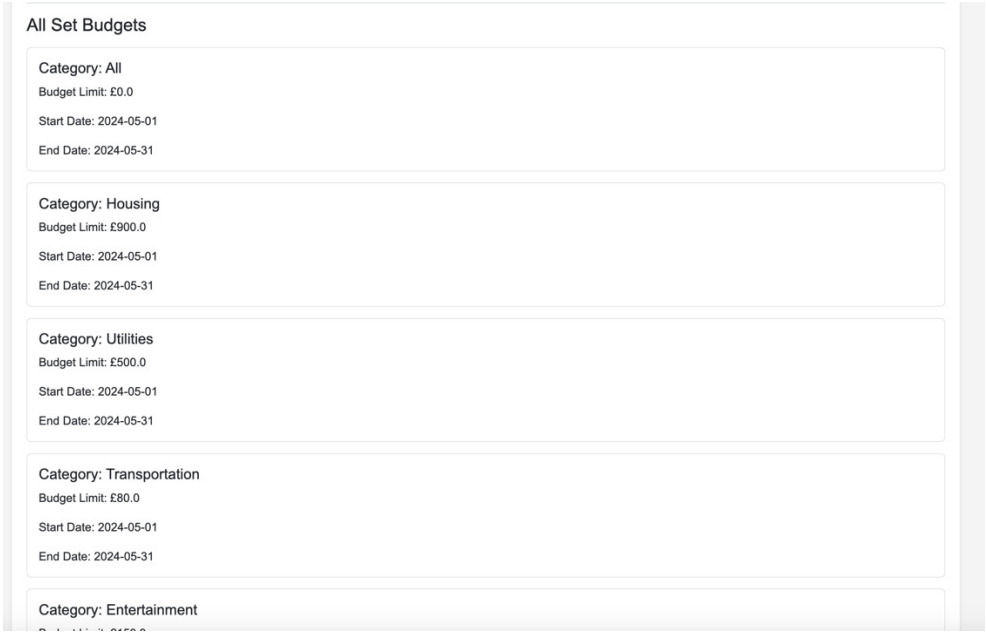
- __pycache__
- assets
  - style.css
- data
  - categories.txt
  - Expenses.txt
- venv
- add_expense_app.py
- app.py
- callbacks.py
- create_user.py
- db.py
- google_vision_ocr.py
- layouts.py
- Readme.txt

**8.3.7 Layout Definitions:** Screenshot of the layouts.py file showing the main layout definition, including the two global filer and nav-bar.



```python
# Function to create the overall layout of the dashboard
def create_layout():
    expenses_data, categories_data = fetch_data()  # Fetch data from the database
    expenses_df = pd.DataFrame(expenses_data)  # Convert expenses data to a DataFrame
    categories_df = pd.DataFrame(categories_data)  # Convert category data to a DataFrame
    expenses_df['DateOfExpense'] = pd.to_datetime(expenses_df['DateOfExpense'])  # Convert expense dates from string to datetime objects
    category_mapping = dict(zip(categories_df['CategoryID'], categories_df['CategoryName']))  # Create a map of category IDs to names
    expenses_df['CategoryName'] = expenses_df['CategoryID'].map(category_mapping)  # Map category names to expenses DataFrame

    # Define the structure of HTML layout
    return html.Div(className='container', children=[
        html.Div(className='title-container', children=[
            html.H1("Financial Dashboard", className='title')  # Main title of the dashboard
        ]),
        html.Div(className='filters-container', style={'display': 'flex', 'justify-content': 'space-between', 'align-items': 'center'}, child
            html.Div([
                dcc.DatePickerRange(
                    id='date-picker-range',
                    start_date=expenses_df['DateOfExpense'].min(),  # Set the minimum selectable date
                    end_date=expenses_df['DateOfExpense'].max()  # Set the maximum selectable date
                ),
                dbc.Button("Max", id='max-date-range', n_clicks=0, className='ms-2', style={'height': '38px'})
            ], style={'display': 'flex'}),
            dcc.Dropdown(
                id='category-dropdown',
                options=[{'label': 'All', 'value': 'all'}] + [{'label': cat, 'value': cat} for cat in expenses_df['CategoryName'].unique()],
                placeholder='Select a Category',
                value='all',
                style={'width': '300px'}
            )
        ]),
        html.Div([
            html.Button('Overview', id='overview-tab', n_clicks=0, className='nav-link'),
            html.Button('Expenses', id='expenses-tab', n_clicks=0, className='nav-link'),
            html.Button('Budget', id='budget-tab', n_clicks=0, className='nav-link'),
        ], className='navbar'),
        html.Div(id='content', className='content')  # Placeholder for dynamic content
    ])
```

**8.3.8 Database Interactions**: Screenshot of the **db.py** file showing an example database interaction. Grabbing all the budgets stored in the database and displaying them as cards




```
budget_cards = [
dbc.Card(
    dbc.CardBody([
        html.H5(f"Category: {next((cat['CategoryName'] for cat in categories if cat['CategoryID'] == budget['CategoryID']), 'Unknown')}",
        html.P(f"Budget Limit: £{budget['BudgetLimit']}", className="card-text"),
        html.P(f"Start Date: {budget['StartDate'].strftime('%Y-%m-%d')}", className="card-text"),
        html.P(f"End Date: {budget['EndDate'].strftime('%Y-%m-%d')}", className="card-text"),
    ]),
    className="mb-3"
) for budget in budgets
```

```
    ], className='mt-4'),
    html.Div(id='budget-status-alert'),
    html.Hr(),
    html.H4("All Set Budgets", className="mb-3"),
    html.Div(budget_cards)
])
```

layouts.py > budget_layout

Click here to ask Blackbox to help you code faster

```python
from dash import dcc, html, dash_table
import dash_bootstrap_components as dbc
from db import fetch_data, get_budget_status, fetch_categories, fetch_all_budgets
import plotly.express as px
import pandas as pd
from datetime import datetime, timedelta

# Function to generate visual cards for each budget category
def generate_budget_cards(budgets):
    categories = fetch_categories()  # Fetch category information from database
    category_dict = {cat['CategoryID']: cat['CategoryName'] for cat in categories}  # Map category IDs to names for easy lookup

    card_deck = []  # List to hold all the card rows
    row = []  # Temporary list to hold cards in a single row
    for i, budget in enumerate(budgets):
        category_name = category_dict.get(budget['CategoryID'], 'Unknown Category')  # Resolve category name using ID
        # Create a card for each budget entry
        card = dbc.Card(
            dbc.CardBody([
                html.H5(f"Category: {category_name}", className="card-title"),
                html.P(f"Budget: £{budget['BudgetLimit']}", className="card-text"),
                html.P(f"Period: {budget['Period']}", className="card-text"),
                html.P(f"Start Date: {budget['StartDate'].strftime('%Y-%m-%d')}", className="card-text"),
                html.P(f"End Date: {budget['EndDate'].strftime('%Y-%m-%d')}", className="card-text"),
            ]),
            className="mb-3"
        )
        row.append(dbc.Col(card, width=3))  # Add card to current row
        if (i + 1) % 4 == 0:  # Every four cards, start a new row
            card_deck.append(dbc.Row(row))
            row = []  # Reset the row list
    if row:  # Add the last row if it has less than four cards
        card_deck.append(dbc.Row(row))
    return card_deck  # Return the complete set of cards organized in rows
```

```python
# Function to fetch all budgets from the database
def fetch_all_budgets():
    db = get_db()
    return list(db.budgets.find({}, {'_id': 0, 'CategoryID': 1, 'BudgetLimit': 1, 'StartDate': 1, 'EndDate': 1, 'Period': 1}))
```

Fetching budgets from the database, generating a card for each budget with its information on it and placing them in rows, under budget layout.

**8.3.9 Database Import:** Snippet of code responsible for importing datasets from expenses.txt and categories.txt into the database.

```python
# Importing the dataset in the database

def import_expenses(user_id):
    # Connect to the local MongoDB server
    client = MongoClient('mongodb://localhost:27017/')
    db = client['financial_management']
    expenses_collection = db['expenses']

    # The ObjectId of the user "tushar"
    user_object_id = ObjectId(user_id)

    file_path = 'data/Expenses.txt'
    with open(file_path, 'r') as file:
        for line in file:
            expense_data = json.loads(line.strip())
            # Add the UserID reference before inserting
            expense_data['UserID'] = user_object_id
            expenses_collection.insert_one(expense_data)

    print("Data import complete.")
```

```python
#Importing catergories data into categories collection
def import_categories():
    # Connect to the local MongoDB server
    client = MongoClient('mongodb://localhost:27017/')
    db = client['financial_management']
    categories_collection = db['categories']

    # List of categories to be inserted
    categories_list = [
        {"CategoryID": 1, "CategoryName": "All"},
        {"CategoryID": 2, "CategoryName": "Housing"},
        {"CategoryID": 3, "CategoryName": "Utilities"},
        {"CategoryID": 4, "CategoryName": "Groceries"},
        {"CategoryID": 5, "CategoryName": "Transportation"},
        {"CategoryID": 6, "CategoryName": "Entertainment"},
        {"CategoryID": 7, "CategoryName": "Education"},
        {"CategoryID": 8, "CategoryName": "Miscellaneous"}

    ]

    # Insert all categories into the database
    categories_collection.insert_many(categories_list)

    print("Categories import complete.")
```

```python
# Calling the function created above to set up the database

# Importing expenses from Expenses.txt
# import_expenses('662947d472ecb2349f29cbe1') #Uncomment this to import expenses from data/expenses.txt

# Importing  categories
# import_categories() #Uncomment this to import expenses from data/categories.txt
```

**8.3.11 Add_expense.py:** Snippet of the that shows a form where user can enter info regarding the expense add it database

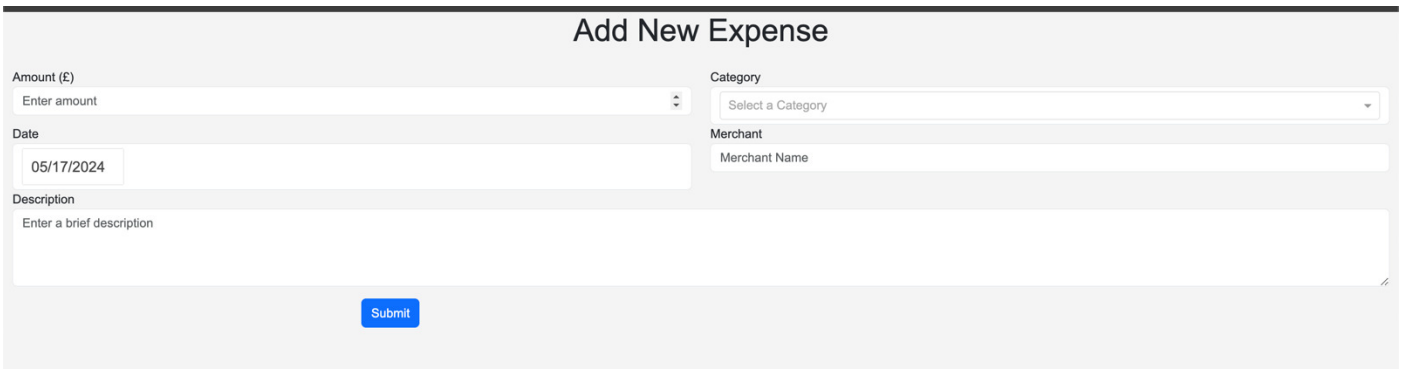
1. Creating the form for users to fill to add expenses. The form includes category dropdown, date picker, merchant field, amount field, and a submit button.

```
    return categories

# Define the layout of the Dash application, arranging form fields in a grid
app.layout = dbc.Container([
    html.H1("Add New Expense", className="text-center mb-4"),  # Title of the form
    dbc.Row([
        dbc.Col([
            html.Label("Amount (£)", className="label"),  # Label for the amount field
            dcc.Input(type="number", id="input-amount", placeholder="Enter amount", step="0.01", className="form-control"),
        ], width=6),
        dbc.Col([
            html.Label("Category", className="label"),  # Label for the category dropdown
            dcc.Dropdown(
                id='category-dropdown',
                options=[{'label': cat['CategoryName'], 'value': cat['CategoryID']} for cat in fetch_categories()],
                placeholder="Select a Category",
                className="form-control"
            ),
        ], width=6)
```

```
    ]),
    dbc.Row([
        dbc.Col([
            html.Label("Date", className="label"),  # Label for the date picker
            dcc.DatePickerSingle(
                id='date-picker-single',
                min_date_allowed=datetime(1995, 1, 1),  # Set minimum allowed date
                max_date_allowed=datetime.now(),  # Set maximum allowed date to today
                initial_visible_month=datetime.now(),  # Set the initially visible month in the picker
                date=datetime.now().date(),  # Default date is today
                className="form-control"
            ),
        ], width=6),
        dbc.Col([
            html.Label("Merchant", className="label"),  # Label for the merchant input field
            dcc.Input(type="text", id="input-merchant", placeholder="Merchant Name", className="form-control"),
        ], width=6)
    ]),
    dbc.Row([
        dbc.Col([
            html.Label("Description", className="label"),  # Label for the description textarea
            dcc.Textarea(id="textarea-description", placeholder="Enter a brief description", className="form-control", style={'height': '100p
        ], width=12)
    ]),
    dbc.Row([
        dbc.Col(dbc.Button("Submit", id="submit-new-expense", color="primary", className="mt-3", n_clicks=0), width={"size": 6, "offset": 3})
    ]),
    html.Div(id="submit-status", className="mt-3")  # Div to display the status of form submission
], fluid=True)
```

**2.** Adding the callback function that helps when user fills the value field and clicks submit then callback function stores the expense data in the expenses collection in the dataset.

```
@app.callback(
    Output('submit-status', 'children'),  # Output is the children property of the 'submit-status' div
    Input('submit-new-expense', 'n_clicks'),  # Trigger the callback on button click
    [State('input-amount', 'value'),  # Get the state of the amount input
     State('category-dropdown', 'value'),  # Get the selected category
     State('date-picker-single', 'date'),  # Get the chosen date
     State('input-merchant', 'value'),  # Get the merchant input
     State('textarea-description', 'value')],  # Get the description text
    prevent_initial_call=True  # Prevent callback from firing on initial load
)
def handle_form_submit(n_clicks, amount, category_id, date, merchant, description):
    if n_clicks > 0:  # Check if the button was clicked
        if not all([amount, category_id, date, merchant, description]):  # Validate that all fields are filled
            return dbc.Alert("All fields are required.", color="danger")
        try:
            expense = {  # Prepare the expense document
                "UserID": ObjectId("662947d472ecb2349f29cbe1"),  # Static user ID, replace as needed
                "CategoryID": category_id,
                "Description": description,
                "Amount": float(amount),
                "DateOfExpense": datetime.strptime(date, '%Y-%m-%d'),
                "MerchantName": merchant
            }
            expenses_collection.insert_one(expense)  # Insert the document into MongoDB
            return dbc.Alert("Expense added successfully!", color="success")
        except Exception as e:  # Handle any errors during insertion
            return dbc.Alert(f"Error adding expense: {str(e)}", color="danger")

# Run the server with debugging enabled
if __name__ == '__main__':
    app.run_server(debug=True)
```




**8.3.12 Pandas Data Frame:** Manipulating data for better data visualization and filter budget data

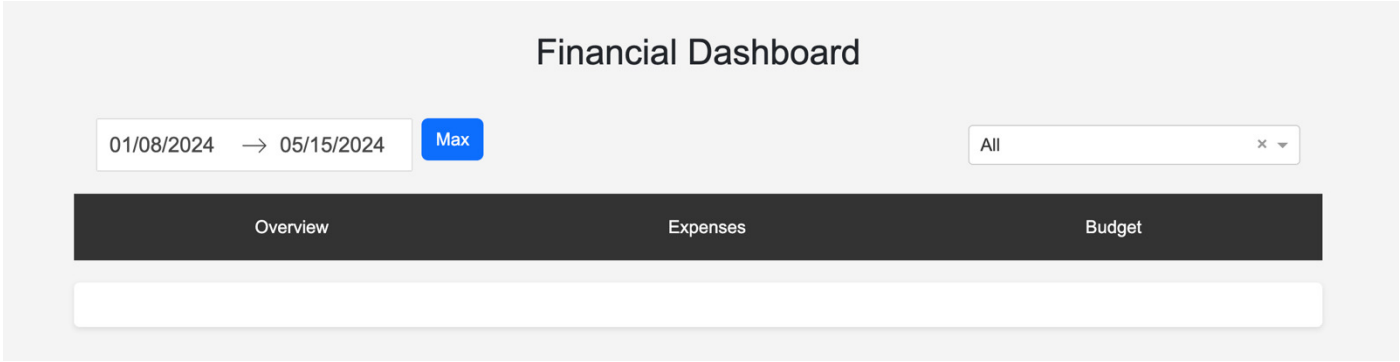
```python
def display_content(overview_clicks, expenses_clicks, budget_clicks, start_date, end_date, category_name):
    ctx = callback_context
    triggered_id = ctx.triggered[0]['prop_id'].split('.')[0]  # Identifies which input triggered the callback

    expenses_data, categories_data = fetch_data()  # Fetches data from the database
    df = pd.DataFrame(expenses_data)
    categories_df = pd.DataFrame(categories_data)

    # Mapping dates and categories for ease of use
    df['DateOfExpense'] = pd.to_datetime(df['DateOfExpense'])
    category_mapping = dict(zip(categories_df['CategoryID'], categories_df['CategoryName']))
    df['CategoryName'] = df['CategoryID'].map(category_mapping)

    # Filtering data based on user-selected date range
    start_datetime = pd.to_datetime(start_date)
    end_datetime = pd.to_datetime(end_date)
    filtered_df = df[(df['DateOfExpense'] >= start_datetime) & (df['DateOfExpense'] <= end_datetime)]

    sorted_df = filtered_df.sort_values('DateOfExpense')
    last_30_days_df = df[df['DateOfExpense'] >= (datetime.now() - timedelta(days=30))]  # Data for the last 30 days
```

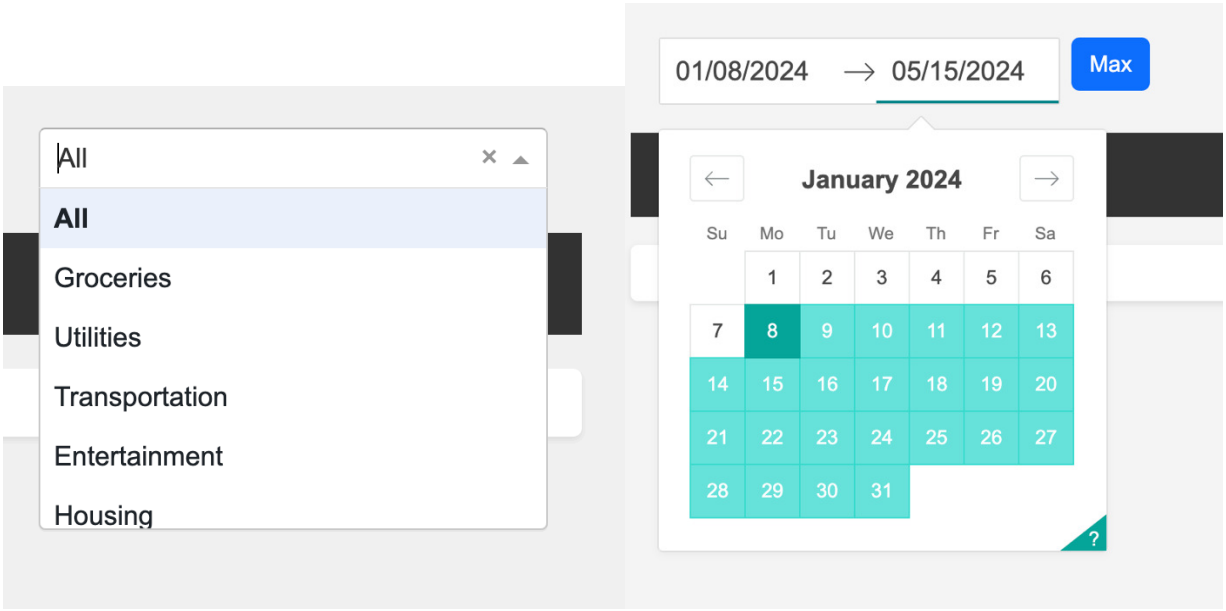
db.py > ...

```python
def get_budget_status(user_id, category_id):
    db = get_db()
    category_id = int(category_id)

    if category_id == 1:  # Assuming '1' is the ID for 'All'
        category_filter = {}
    else:
        category_filter = {"CategoryID": category_id}

    start_of_month, end_of_month = get_time_bounds()

    monthly_budgets = list(db.budgets.find({
        "UserID": ObjectId(user_id),
        **category_filter,
        "StartDate": {"$gte": start_of_month, "$lte": end_of_month},
    }))

    expenses = list(db.expenses.find({
        "UserID": ObjectId(user_id),
        **category_filter
    }))

    # Convert expenses to DataFrame and filter by date
    expenses_df = pd.DataFrame(expenses)
    if not expenses_df.empty:
        expenses_df['DateOfExpense'] = pd.to_datetime(expenses_df['DateOfExpense']).dt.tz_localize(timezone.utc)

    filtered_expenses_df = expenses_df[(expenses_df['DateOfExpense'] >= start_of_month) & (expenses_df['DateOfExpense'] <= end_of_month)]
    total_spent = filtered_expenses_df['Amount'].sum() if not expenses_df.empty else 0

    # Calculate total budget for the month
    total_budget = sum(b['BudgetLimit'] for b in monthly_budgets)

    # Calculate budget left
    monthly_budget_left = total_budget - total_spent
    budget_status = f"£{monthly_budget_left:.2f}" if monthly_budget_left >= 0 else "Budget exceeded"

    # Returning total spent and budget limit for pie chart visualization
    return budget_status, total_spent, total_budget
```

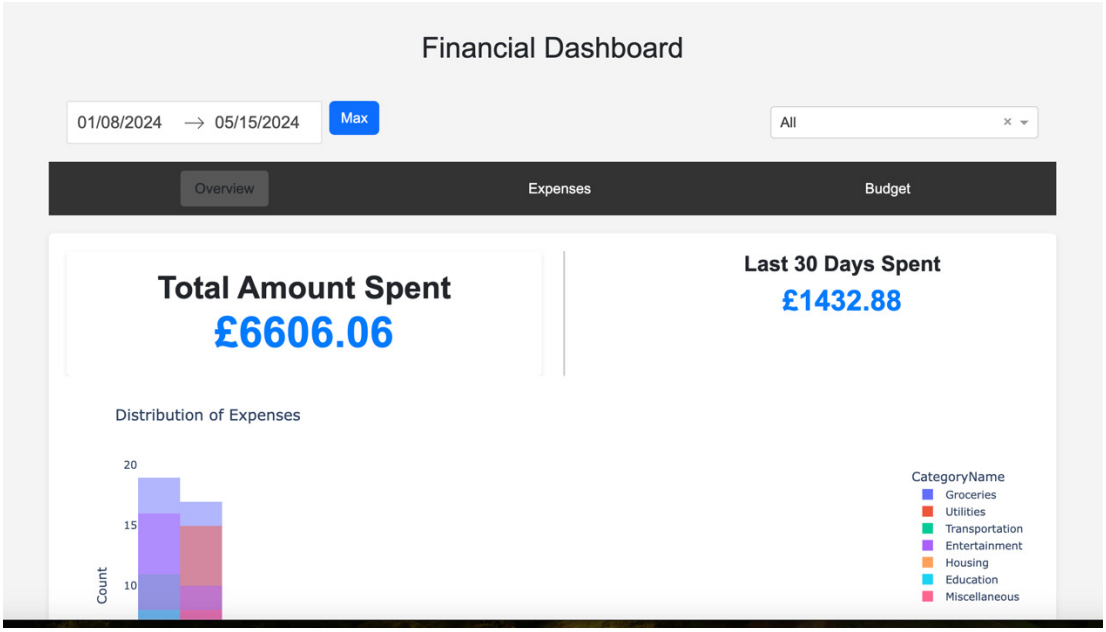
## 8.4 Financial Dashboard

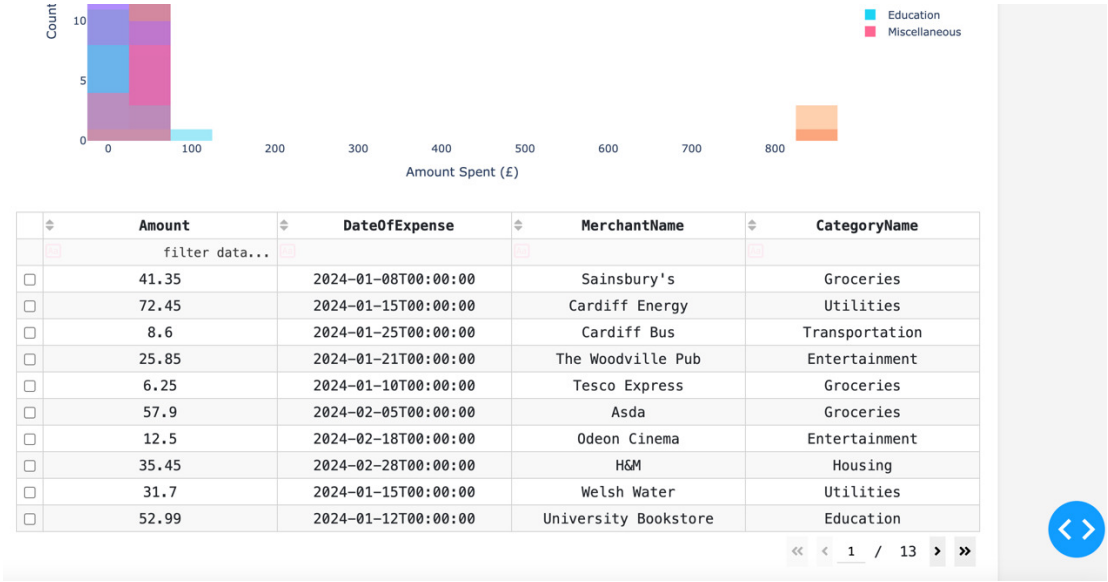
1. Global filters and Nav-bar




2. Overview Tab

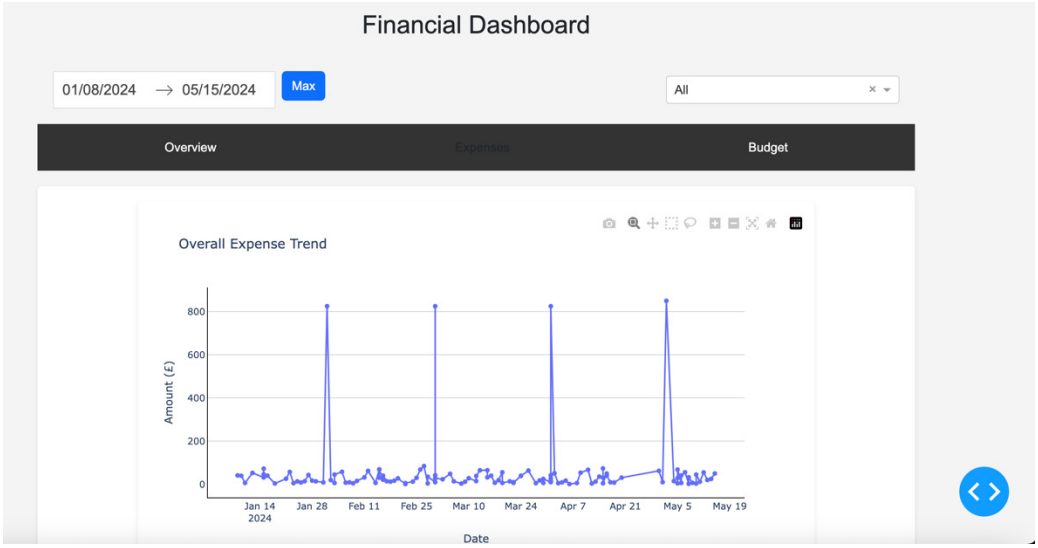


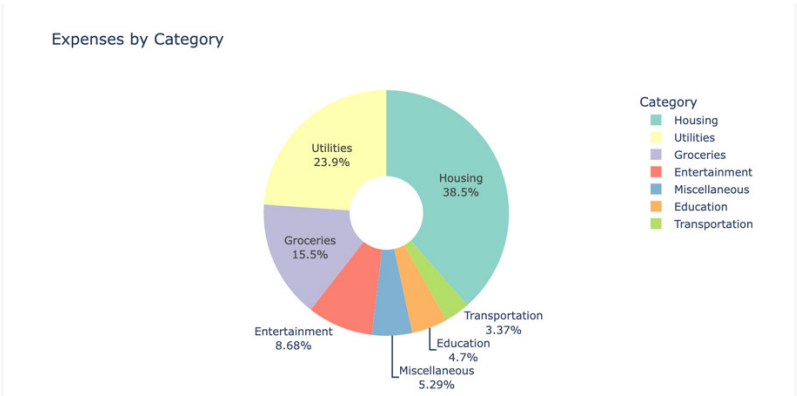




| | Amount | DateOfExpense | MerchantName | CategoryName |
|---|---|---|---|---|
| | filter data... | | | |
| ☐ | 41.35 | 2024-01-08T00:00:00 | Sainsbury's | Groceries |
| ☐ | 72.45 | 2024-01-15T00:00:00 | Cardiff Energy | Utilities |
| ☐ | 8.6 | 2024-01-25T00:00:00 | Cardiff Bus | Transportation |
| ☐ | 25.85 | 2024-01-21T00:00:00 | The Woodville Pub | Entertainment |
| ☐ | 6.25 | 2024-01-10T00:00:00 | Tesco Express | Groceries |
| ☐ | 57.9 | 2024-02-05T00:00:00 | Asda | Groceries |
| ☐ | 12.5 | 2024-02-18T00:00:00 | Odeon Cinema | Entertainment |
| ☐ | 35.45 | 2024-02-28T00:00:00 | H&M | Housing |
| ☐ | 31.7 | 2024-01-15T00:00:00 | Welsh Water | Utilities |
| ☐ | 52.99 | 2024-01-12T00:00:00 | University Bookstore | Education |

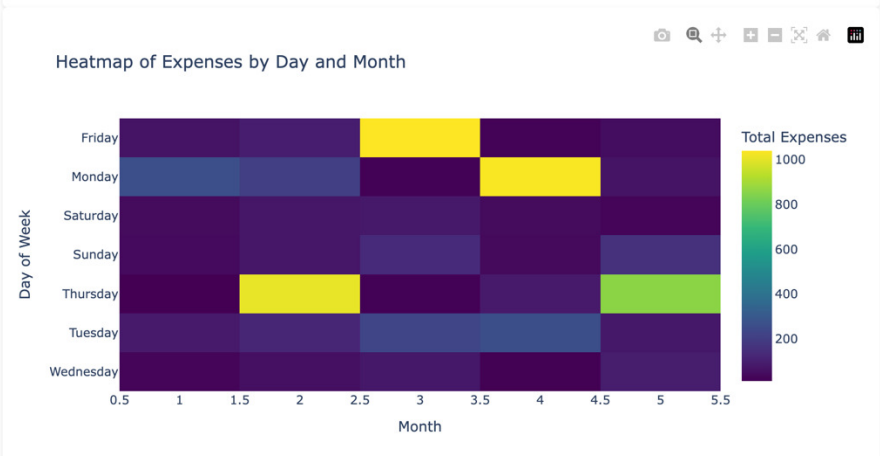
« ‹ 1 / 13 › »

3. Expense Tab

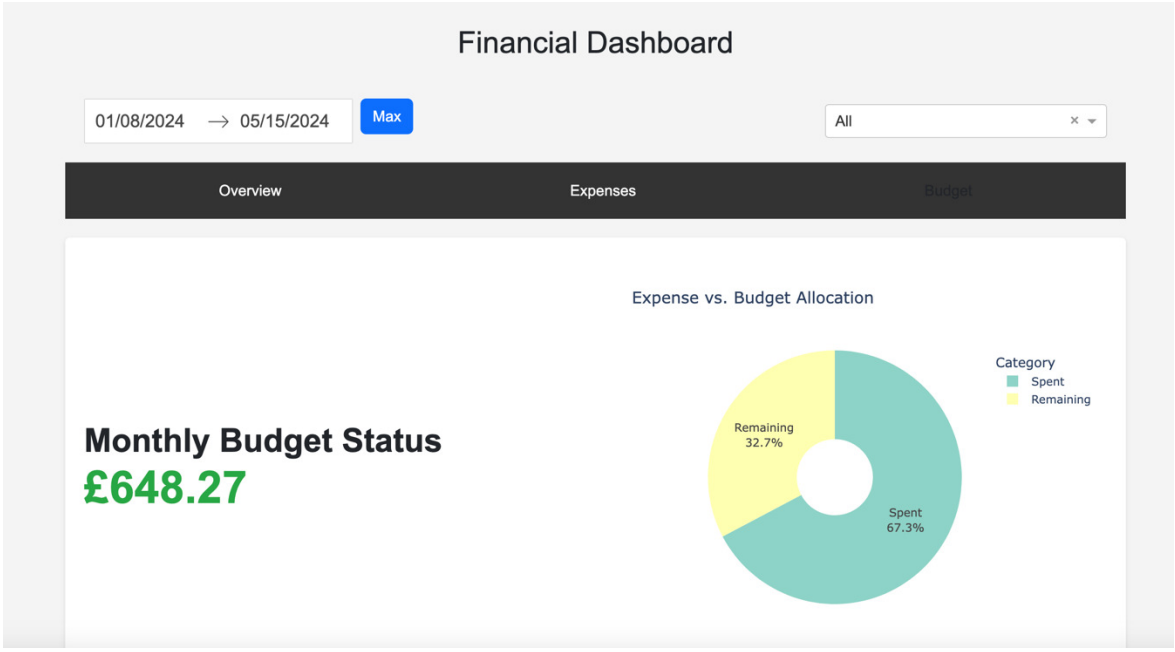

Financial Dashboard
01/08/2024 → 05/15/2024
Max
All
Overview
Expenses
Budget
Overall Expense Trend
Amount (£)
0
200
400
600
800
Jan 14 2024
Jan 28
Feb 11
Feb 25
Mar 10
Mar 24
Apr 7
Apr 21
May 5
May 19
Date

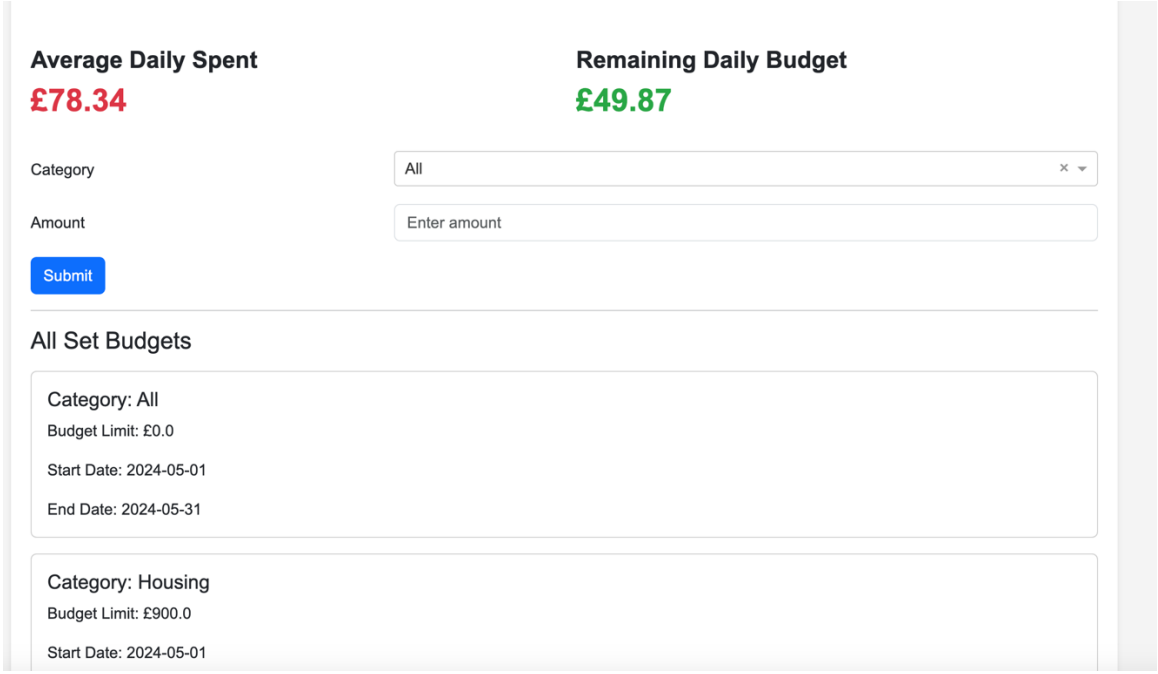

Expenses by Category
Utilities 23.9%
Housing 38.5%
Groceries 15.5%
Entertainment 8.68%
Transportation 3.37%
Education 4.7%
Miscellaneous 5.29%
Category
Housing
Utilities
Groceries
Entertainment
Miscellaneous
Education
Transportation


Heatmap of Expenses by Day and Month
Friday
Monday
Saturday
Total Expenses
1000
800

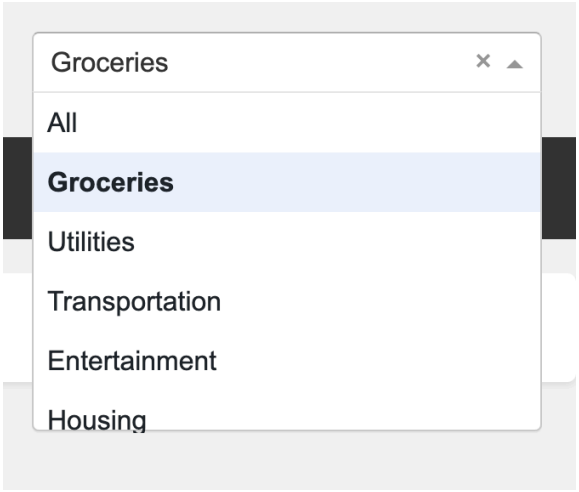

Heatmap of Expenses by Day and Month
Friday
Monday
Saturday
Sunday
Thursday
Tuesday
Wednesday
Day of Week
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
5.5
Month
Total Expenses
1000
800
600
400
200

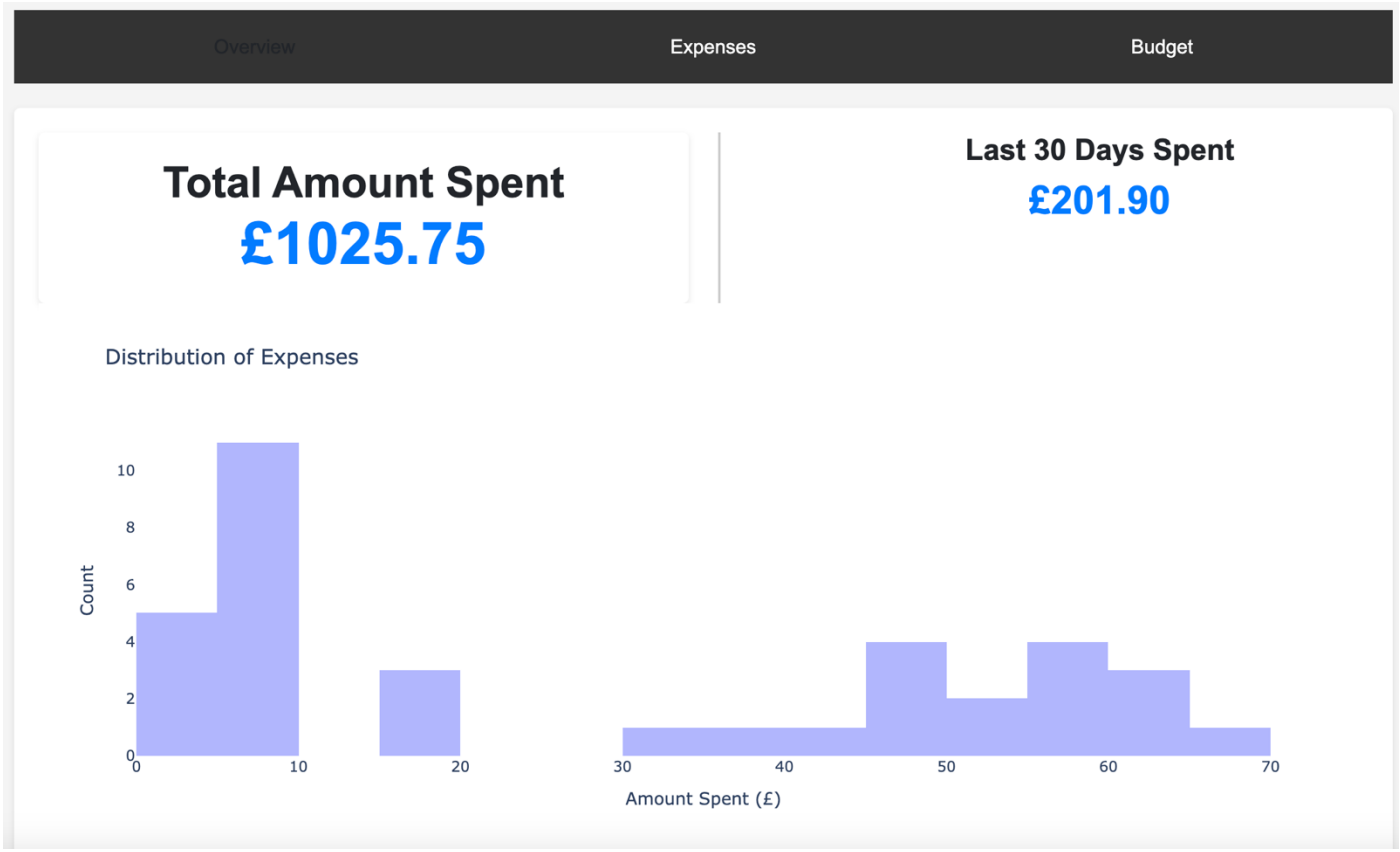
4. Budget Tab




5. When a specific category is selected

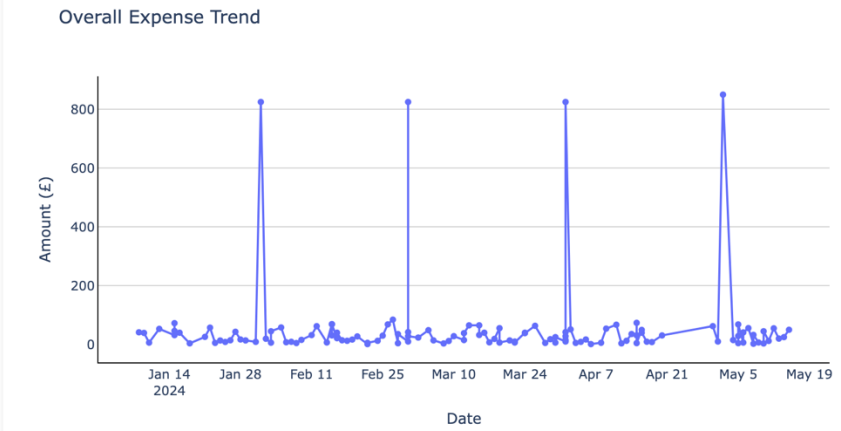

Groceries
All
Groceries
Utilities
Transportation
Entertainment
Housing


Overview
Expenses
Budget
Total Amount Spent
£1025.75
Last 30 Days Spent
£201.90
Distribution of Expenses
Count
0
2
4
6
8
10
0
10
20
30
40
50
60
70
Amount Spent (£)

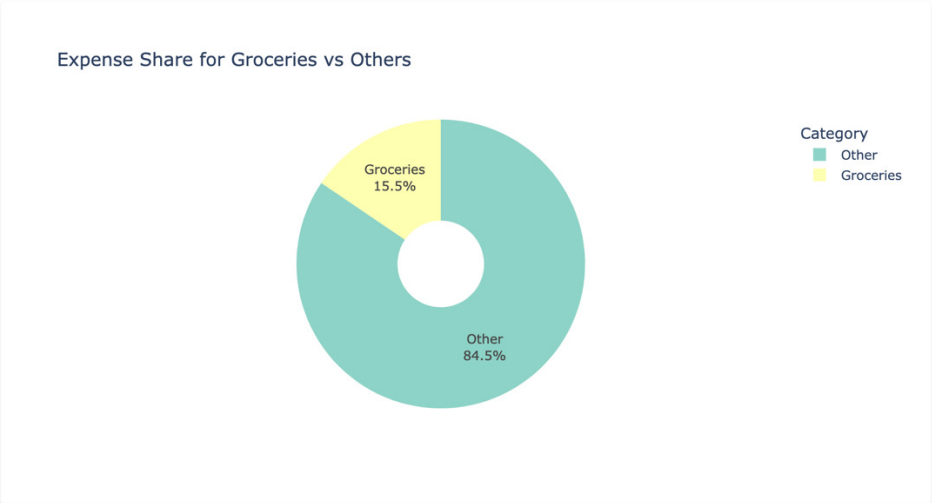
| | Amount | DateOfExpense | MerchantName | CategoryName |
|---|---|---|---|---|
| | filter data... | | | |
| ☐ | 41.35 | 2024-01-08T00:00:00 | Sainsbury's | Groceries |
| ☐ | 6.25 | 2024-01-10T00:00:00 | Tesco Express | Groceries |
| ☐ | 57.9 | 2024-02-05T00:00:00 | Asda | Groceries |
| ☐ | 5.35 | 2024-01-23T00:00:00 | Starbucks | Groceries |
| ☐ | 16.95 | 2024-01-28T00:00:00 | Domino's Pizza | Groceries |
| ☐ | 63.5 | 2024-03-26T00:00:00 | Coop | Groceries |
| ☐ | 46.2 | 2024-01-15T00:00:00 | Sainsbury's | Groceries |
| ☐ | 1.25 | 2024-02-22T00:00:00 | Coop | Groceries |
| ☐ | 19.5 | 2024-02-02T00:00:00 | Deliveroo | Groceries |
| ☐ | 6.4 | 2024-03-19T00:00:00 | Coffee #1 | Groceries |

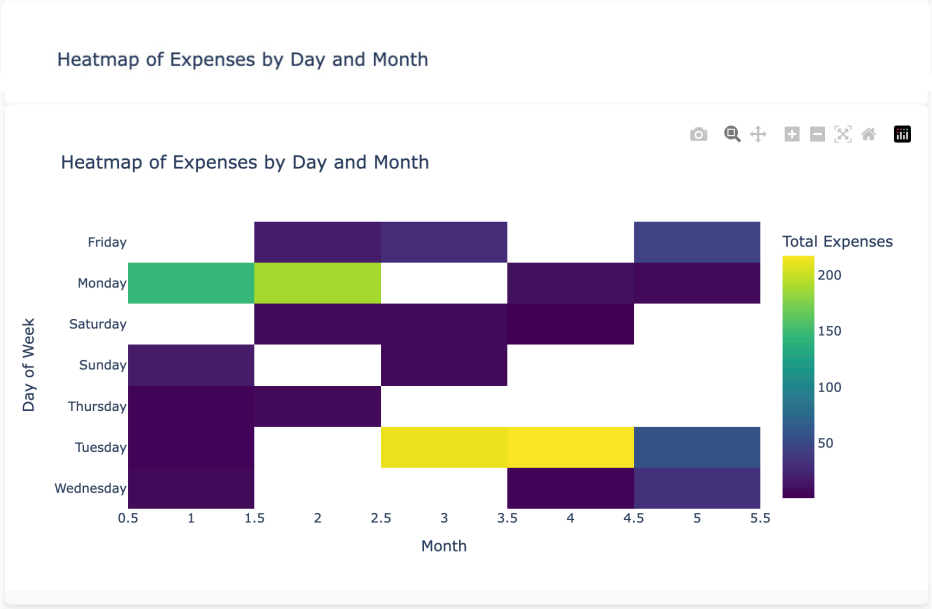
« ‹ 1 / 4 › »

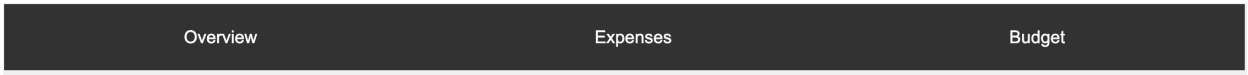
Overview | Expenses | Budget

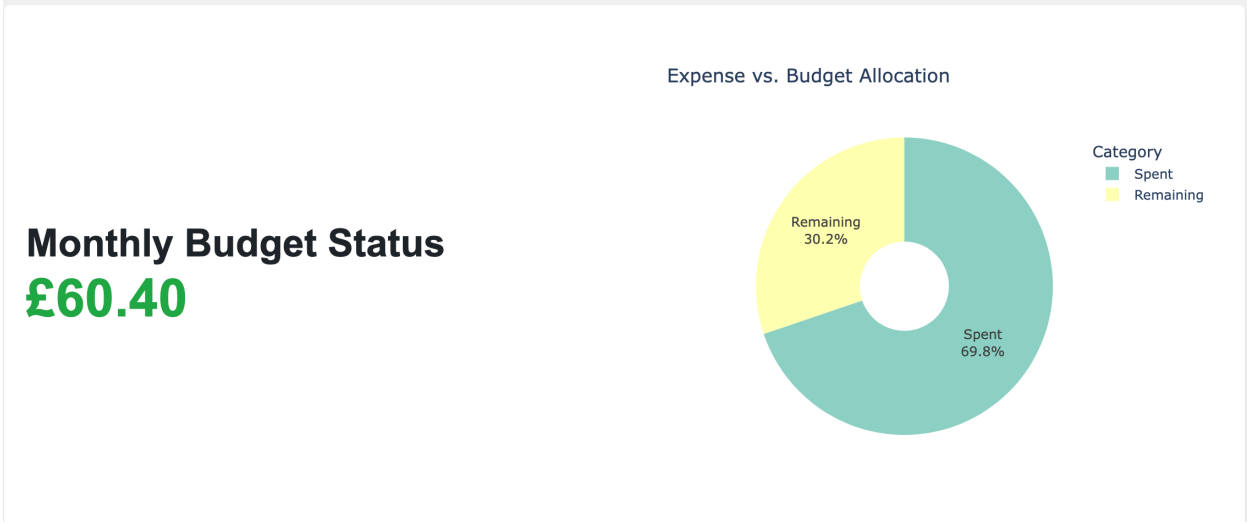


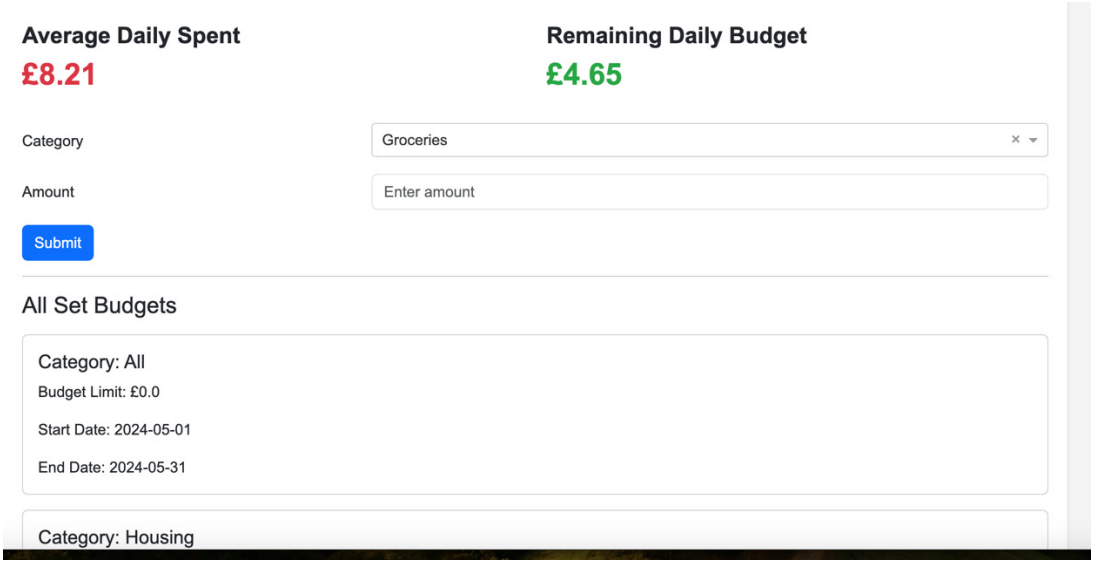

Expense Share for Groceries vs Others


Expense Share for Groceries vs Others
Groceries
15.5%
Other
84.5%
Category
Other
Groceries

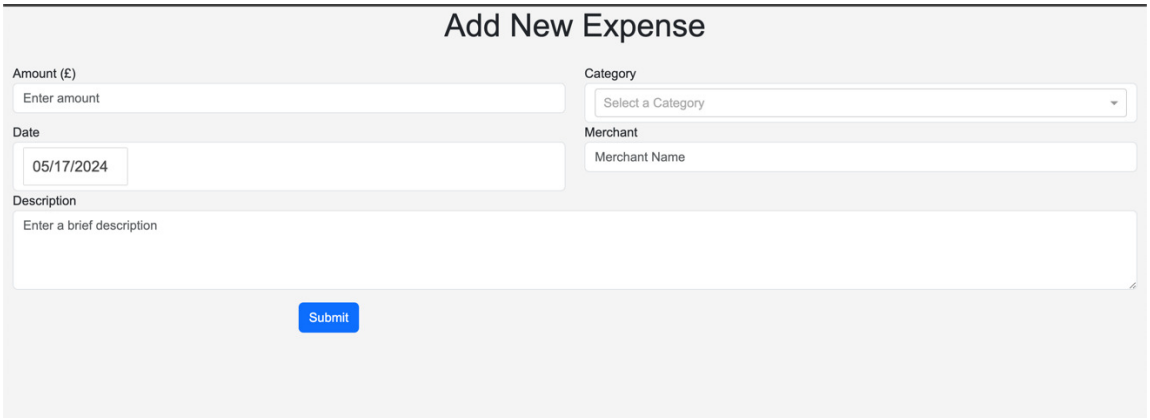

Heatmap of Expenses by Day and Month
Heatmap of Expenses by Day and Month
Friday
Monday
Saturday
Sunday
Thursday
Tuesday
Wednesday
Day of Week
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
5.5
Month
Total Expenses
200
150
100
50


Overview
Expenses
Budget

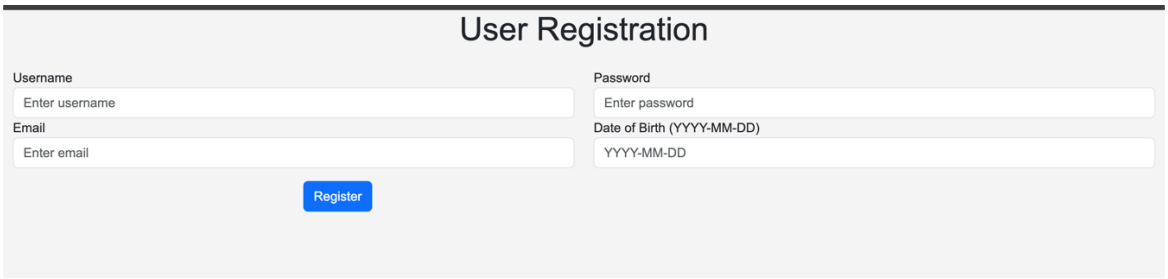

Monthly Budget Status
£60.40
Expense vs. Budget Allocation
Remaining
30.2%
Spent
69.8%
Category
Spent
Remaining




6. Add expense app




7. Create user app



# 9. References


1. Few, S., 2006. *Information Dashboard Design: The Effective Visual Communication of Data*. Sebastopol, CA: O'Reilly Media.
2. Heath, T., 2020. *Personal Finance for Students: A Comprehensive Guide*. New York: Financial Literacy Press.
3. Hunter, J.D., 2007. Matplotlib: A 2D Graphics Environment. *Computing in Science & Engineering*, 9(3), pp.90-95.
4. Kumar, V., 2017. Financial Literacy among University Students: A Case Study. *Journal of Financial Education*, 43(2), pp.33-47.
5. Lusardi, A., Mitchell, O.S. and Curto, V., 2010. Financial Literacy among the Young. *Journal of Consumer Affairs*, 44(2), pp.358-380.
6. Norman, D.A., 2013. *The Design of Everyday Things: Revised and Expanded Edition*. New York: Basic Books.
7. Preece, J., Rogers, Y. and Sharp, H., 2015. *Interaction Design: Beyond Human-Computer Interaction*. 4th ed. Chichester: Wiley.
8. Sievert, C., 2020. *Interactive Web-Based Data Visualization with R, plotly, and shiny*. Boca Raton, FL: CRC Press.
9. Smith, R., 2007. An Overview of the Tesseract OCR Engine. *Proceedings of the Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*, 2, pp.629-633.
10. Tella, A., 2018. *Financial Management Practices and Financial Literacy of Students in Higher Education*. Lagos: University Press.
11. Waskom, M.L., 2021. Seaborn: Statistical Data Visualization. *Journal of Open Source Software*, 6(60), p.3021.
12. Bostock, M., Ogievetsky, V. & Heer, J., 2011. D3: Data-Driven Documents. *IEEE Transactions on Visualization and Computer Graphics*, 17(12), pp. 2301-2309.
13. He, H. & Garcia, E. A., 2009. Learning from Imbalanced Data. *IEEE Transactions on Knowledge and Data Engineering*, 21(9), pp. 1263-1284.
14. MongoDB Inc., 2024. *MongoDB Documentation*. [online] Available at: https://docs.mongodb.com/ [Accessed 15 May 2024].
15. Plotly Technologies Inc., 2024. *Dash User Guide and Documentation*. [online] Available at: https://dash.plotly.com/ [Accessed 15 May 2024].
16. Chen, H., Chiang, R. H. L. & Storey, V. C., 2012. Business Intelligence and Analytics: From Big Data to Big Impact. *MIS Quarterly*, 36(4), pp. 1165-1188.
17. Agarwal, R. & Dhar, V., 2014. Big Data, Data Science, and Analytics: The Opportunity and Challenge for IS Research. *Information Systems Research*, 25(3), pp. 443-448.

18. Dean, J. & Ghemawat, S., 2008. MapReduce: Simplified Data Processing on Large Clusters. *Communications of the ACM*, 51(1), pp. 107-113.
19. Van Der Aalst, W. M. P., 2016. Process Mining: Data Science in Action. 2nd ed. Berlin: Springer.
20. Provost, F. & Fawcett, T., 2013. Data Science for Business: What You Need to Know about Data Mining and Data-Analytic Thinking. Sebastopol, CA: O'Reilly Media.
21. Russom, P., 2011. Big Data Analytics. [pdf] TDWI Best Practices Report, Fourth Quarter 2011. Available at: https://tdwi.org/research/2011/09/tdwi-best-practices-report-q4-2011-big-data-analytics.aspx [Accessed 15 May 2024].