East West University

CSE412: Software Engineering (Project Report)

PET : Personal Expense Tracker

Section: 2

Group: 08

Group Name: Team Noob

Name & ID of Students:

Abu Kahhar Mohammad Sakib	2021-1-60-040
Prioti Kar Tithy	2022-1-60-082
Budrun Nahar Bristy	2022-1-60-144
Fatema Tuz Zannat	2022-1-60-153

Course Instructor Information:

Yasin Sazid

Lecturer

Department of Computer Science and Engineering

East West University

Declaration Page:

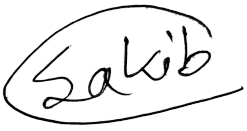
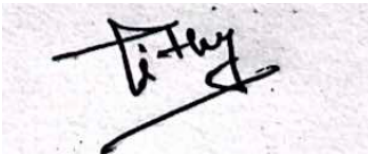
Member's Name	Signature
Abu Kahhar Mohammad Sakib	
Prioti Kar Tithy	
Budrun Nahar Bristy	
Fatema Tuz Zannat	

Table of Contents:

Chapter 1: Introduction

The Personal Expense Tracker is a software application that enables users to manage their daily financial transactions efficiently. It provides a structured platform where users can log income and expenses, categorize transactions, and generate insightful reports for better financial decision-making.

1.1 Project Overview

The Personal Expense Tracker is a web-based software application that allows users to efficiently monitor and manage their personal or family financial transactions. The platform enables the user to create new spending and income transactions by inputting fields like description, date, category, and amount. If they make a mistake, they can revise or remove transactions to maintain their records accurately.

To better control expenses, the user allocates transactions to predefined categories like Food, Rent, and Entertainment. They can define monthly budget limits for certain categories or all together to limit spending and get notified when they are near or have crossed their budget. The system also offers AI-driven forecasts for next month's spending based on historical spending patterns, enabling them to plan their finances.

The user can also invite family members to the platform, making it a family-shared expense tracker. Every member can monitor their income and expenses, which will create a combined financial summary. The system will update all transactions in real-time, so everyone will be made aware of the economic condition of the family. To check their spending, users visit the reports feature, where they get interactive graphs and charts displaying money trends. Users can filter reports based on date, category, or transaction type to receive precise information. The system should further provide a breakdown of total income, spending, and savings over a specified time and give a general financial overview for the user or the entire family.

1.2 Objectives

The core objective of the Personal Expense Tracker is to empower individuals and families to control over their financial lives by providing a comprehensive, user-friendly and intelligent

platform for tracking income and expenses. The system is designed not just for record keeping but for improving financial literacy,planning and decision-making through the use of smart technology.

- Financial Awareness and Discipline
- Effective Budgeting
- Data-Driven Decision Making
- Security and Accessibility
- Collaborative Financial Management

Scope

The scope of the Personal Expense Tracker includes all the functional and non-functional capabilities required to meet the diverse financial tracking needs of individual users and families. It focuses on building a modular and extensible system that offers transaction management, analytics, AI support, and multi-user collaboration.

- Manage expenses and income entries
- Expense and income tracking with categorization
- Monthly budget planning and alerts
- Data visualization through charts and graphs
- Secure authentication and data storage
- Report generation for better financial analysis
- AI-based predictive analytics
- Collaborative expense management

1.3 Stakeholder:

A stakeholder is any individual, group, or organization that has an interest in or is affected by a project, product, or system. Stakeholders can influence the project's success, and their needs and expectations must be considered during development.

- Users (Individuals Managing Personal Finances**)**
- Product Owner/Project Manager
- Development Team
- Financial Advisors
- Regulatory Authorities
- Technical Support & Maintenance Team

1.4 Technology Stack:

- **Frontend Framework**: React.js
- **Programming language :** JavaScript
- **Database:** Firebase Firestore
- **Authentication:** Firebase Authentication
- **Hosting:** Firebase Hosting
- **Version Control:** Git + GitHub
- **Build Tool:** React Script
- **Styling:** CSS + Tailwind CSS

Chapter 2: Software Requirements Specification & Analysis

2.1 Stakeholder Needs & Analysis

2.1.1 Primary Stakeholders

A primary stakeholder is an individual or group that is directly involved in or affected by the project's outcome. They usually interact with the system and have a significant interest in its success or failure**.**These are the main users who want to track their personal or household expenses. They require a simple, intuitive interface, accurate reports, and reliable performance. The Primary Stakeholders of our project are:

- **Users (Individuals Managing Personal Finances)**: People who will use the application to track income and expenses.Their feedback is crucial in designing a user-friendly experience.
- **Product Owner/Project Manager:** Responsible for ensuring the project meets its objectives and aligns with user needs.
- **Project Development Team:** Includes developers, UI/UX designers, and testers responsible for building and maintaining the application.

2.1.2 Secondary Stakeholders

A secondary stakeholder is an individual or group that is indirectly affected by the project. They may not use the system themselves but have an interest in its performance, outcomes, or impact.

Future Collaborators or Partners: Financial advisors or budgeting consultants who may integrate or recommend the system. The Secondary Stakeholders of our project are:

- **Financial Advisors**: May provide insights on financial management best practices to improve the application’s features.
- **Regulatory Authorities:** Entities ensure that the application complies with financial and data protection regulations.

- **Technical Support & Maintenance Team**: Responsible for maintaining and updating the application after deployment.

2.1.3 Methods for Requirement Elicitation

For requirement elicitation we did a Survey and Physical Interview both. Surveys were a useful method for gathering requirements because by doing so we collect information from a broad audience efficiently. Also did 6 physical interviews by asking their requirements for better interaction. For our project, both methods helped us understand the needs and preferences of potential users regarding personal finance management. The survey questionnaire and the interview drafts are attached in the appendix.

2.2 List of Requirements

We gather our system requirements through surveys and face to face interviews. Here are the the user requirements we've gathered:

- Add, edit and delete expenses
- Add, edit and delete sources of income
- Budgeting of expenditures (e.g., Food, Transport, Rent, Shopping)
- Monthly budget generation and tracking
- Graphical portrayal of financial figures (Graphs, Charts)
- Automatic generation of daily, weekly, and monthly reports
- The data must be secured
- Provide smart recommendations based on expenditure trends.
- Allow shared budgets for groups and families.
- Receipt scanning and integration with other apps and bank accounts. (Not Scalable)
- AI chatbot/ AI assistance (Not Scalable)
- Maintaining and updating the application after deployment.

2.2.1 Functional Requirements (FRs):

Functional requirements are the core capabilities and services that a system or software must provide to fulfill its intended purpose. These requirements define the behavior of the system under specific conditions, focusing on what the system should do. The Functional Requirements for you system are:

- A user should register/ sign up with proper authentication to enter the website
- Users should be able to add, edit, and delete expenses
- Users should be able to add, edit, and delete income sources
- Categorization of expenses (Food, Transport, Rent, Shopping) should be maintained by the website
- Users can set monthly budget and track expenses
- Users can get visual representation of financial data (Graphs, Charts)
- Report generation (Daily, Weekly, Monthly summaries) should be viewed by the website
- The users should get alert messages for nearing the set budget or exceeding it.

2.2.2 Non-Functional Requirements (NFRs) :

Non-functional requirements describe the quality attributes of a system. They define how the system should perform, rather than what it does. NFRs ensure the software is reliable, efficient, secure, scalable, and user-friendly. They are critical for determining the user experience and overall performance of the system. While not directly visible as features, NFRs play a vital role in ensuring the system meets user expectations and performs well under real-world conditionsFocuses on performance, security, usability, maintainability, etc. The Non-Functional Requirements are**:**

- User data should be protected and to login the user should need proper authentications.

- Application should load within 5 seconds.
- The interface should be simple and mobile-responsive.
- The system should support multiple users simultaneously.

2.2.3 Extra-Ordinary Requirements (Wow Factors):

It basically refers to needs or conditions that go beyond the uncommon case. Here users require advanced skills, special factors. The given requirements that we collected from our user:

- Predict categorical and total expense for the next month.
- Enable seamless synchronization of data across multiple devices.
- Provide smart recommendations based on spending habits using AI.
- The website should support shared budgeting for families and groups.

2.3 House of Quality (QFD Integration):

One of the main components of the Quality Function Deployment (QFD) process, which assists in converting customer requirements (CRs) into precise technical specifications for a good or service, is the House of Quality (HoQ). It is basically a matrix that shows the relationship between what customers want and how a business intends to fulfill that want.

2.3.1 Customer Requirements (CRs) List:

Customers' needs are reflected in Customer Requirements (CRs). They outline the characteristics, capabilities, performance, and quality that end customers anticipate from a good or service.

1. Secure Registration & Authentication
2. Expense & Income Management
3. Expense Categorization
4. Budget Setting & Tracking

5. Data Visualization
6. Reports & Summaries
7. Budget Alerts
8. Data Protection & Security
9. Fast & Responsive Interface
10. Multi-User Support
11. Smart AI-based Recommendations
12. Cross-Device Synchronization
13. Shared Budgeting

2.3.2 Engineering Requirements (TRs) List:

Technical requirements (TRs) outline the methodologies, technologies, and processes that will be used by the organization to meet the needs of its customers. These are created by designers, engineers, and developers.

1. Authentication & Authorization
2. Database Management and Expense & Income CRUD Operations
3. Expense Categorization Logic
4. Budget Tracking Algorithm
5. Data Visualization
6. Report Generation
7. Real-time Alerts
8. Mobile Responsiveness
9. Performance Optimization

10. Scalability & Multi-User Support

11. AI-based Spending Insights

12. Cloud Data Sync

13. Shared Budgeting Feature

2.3.3 QFD Matrix (House of Quality)

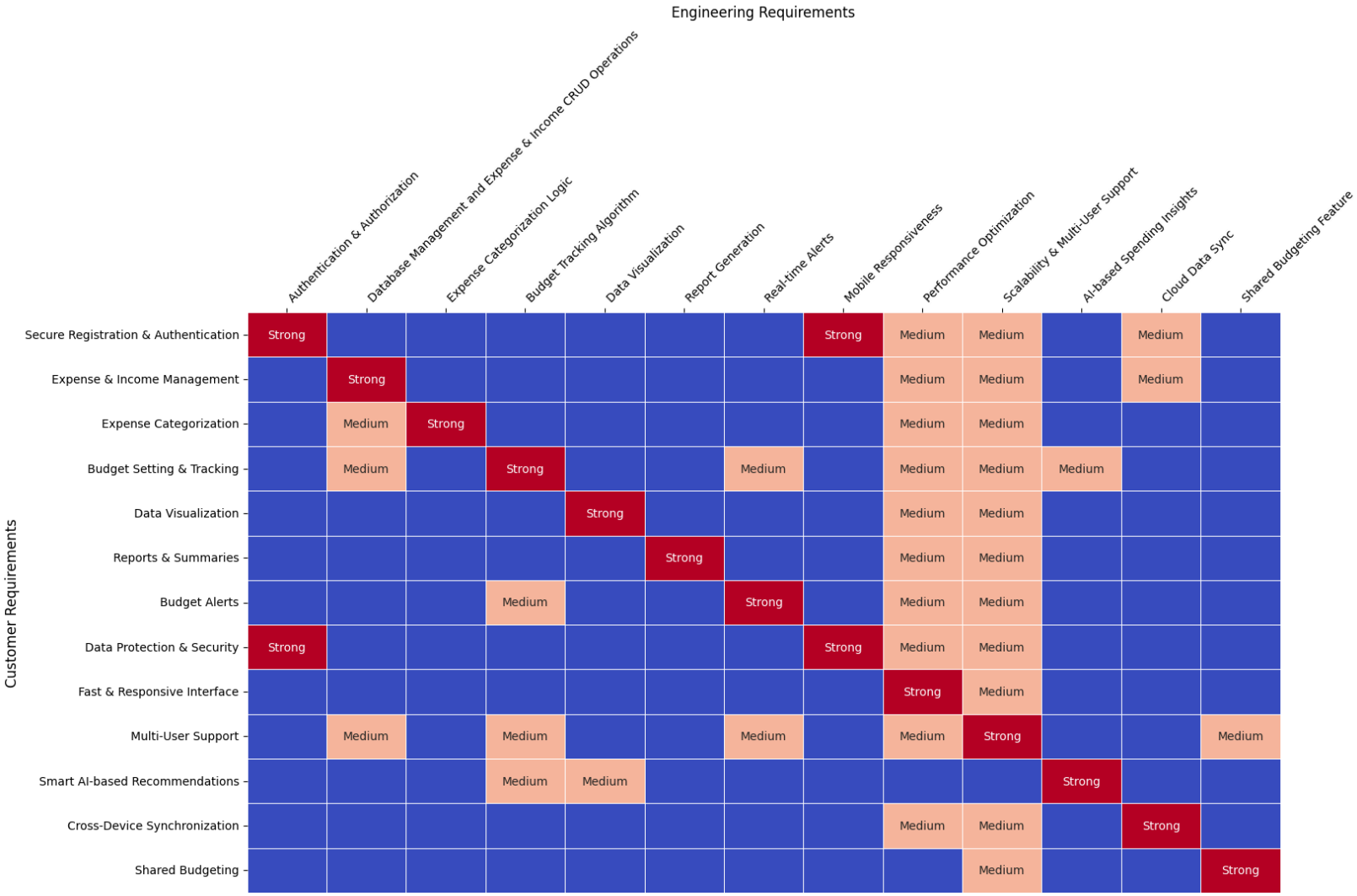


Figure 2.3.3 House of Quality

2.4 Requirements Modeling:

2.4.1 Use Case Diagrams

Level 0 :

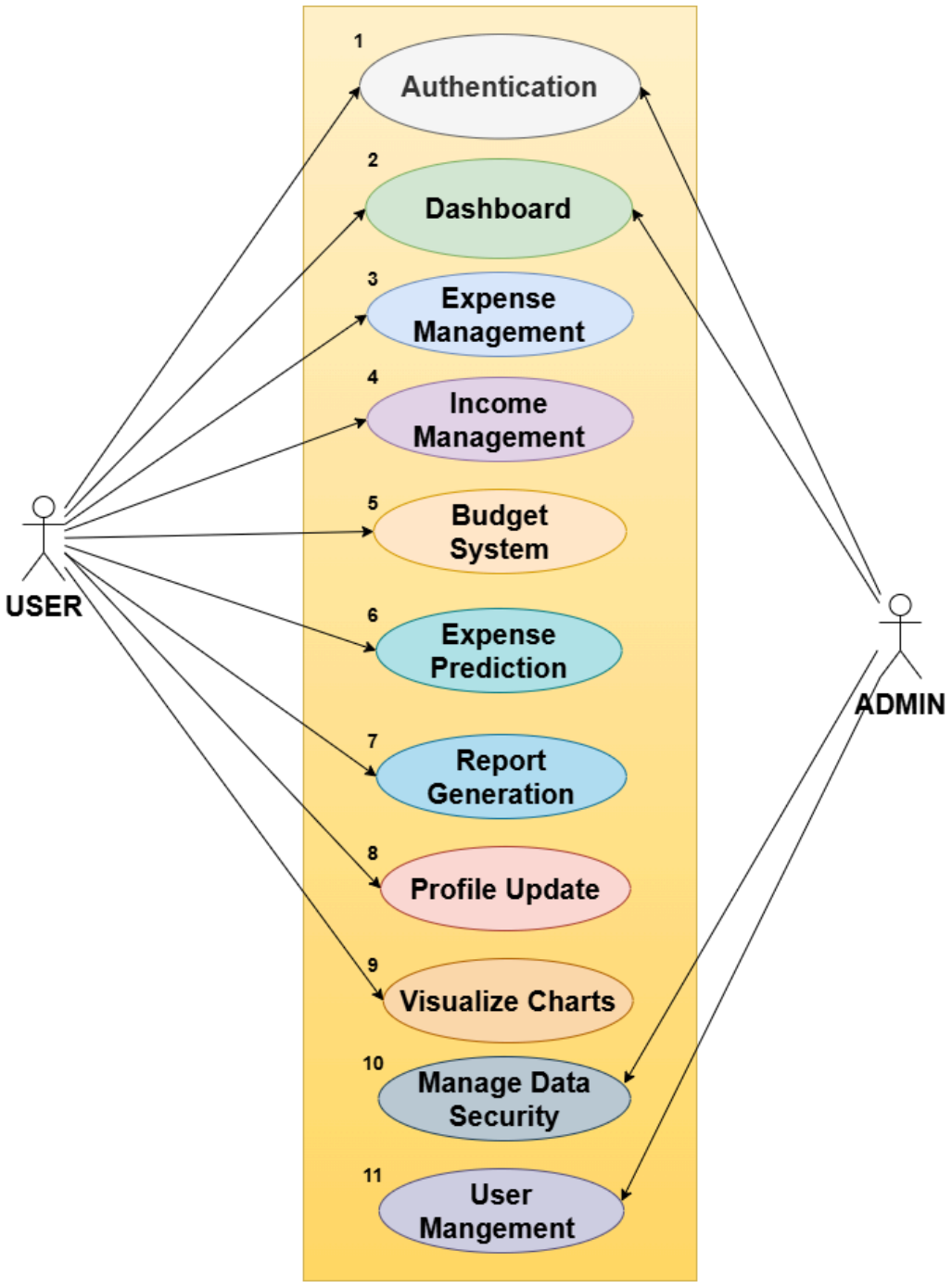


Figure 2.4.1 Use Case diagram level 0

Level 1:

Authentication:

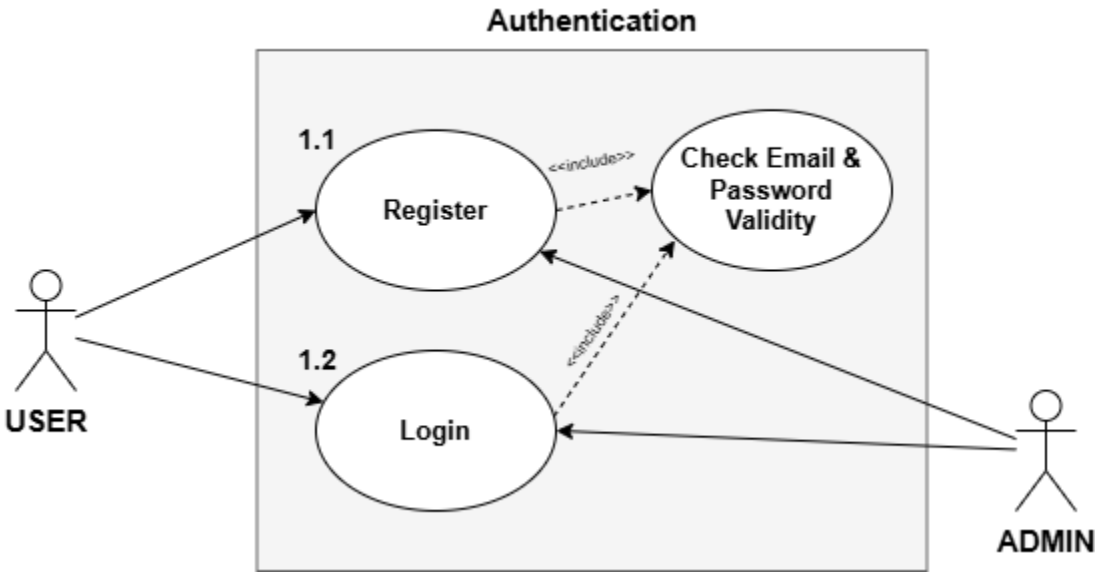


Figure 2.4.1 Use Case diagram level 1 for Authentication

Dashboard :

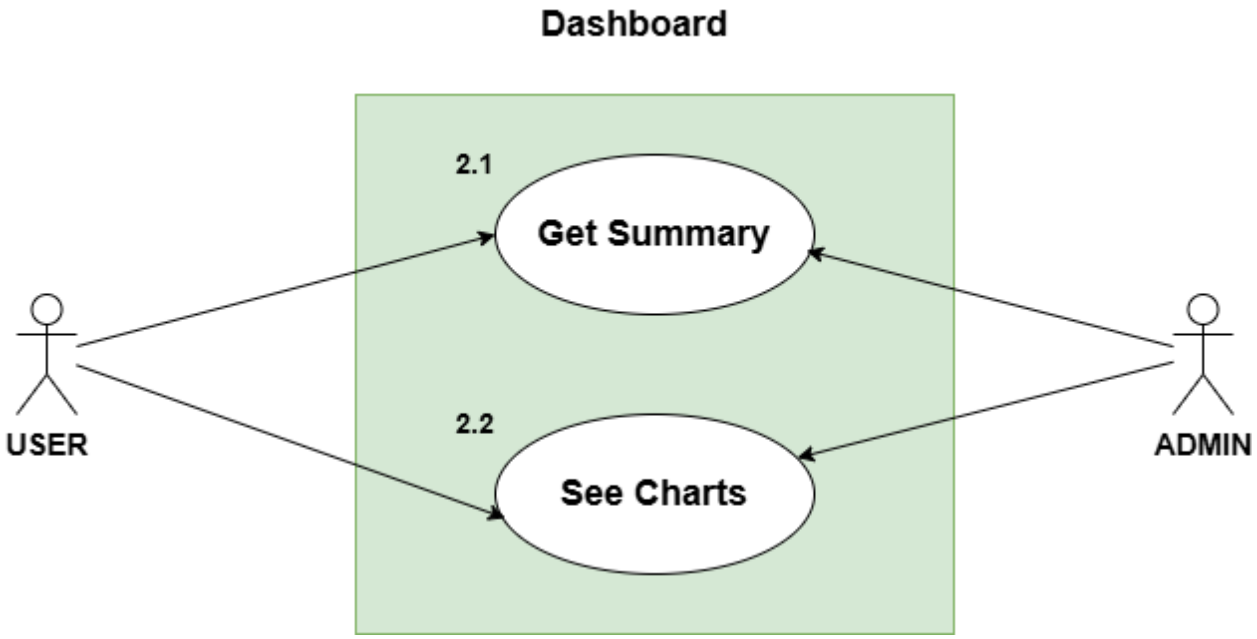


Figure 2.4.1 Use Case diagram level 1 for dashboard

Expense Management:

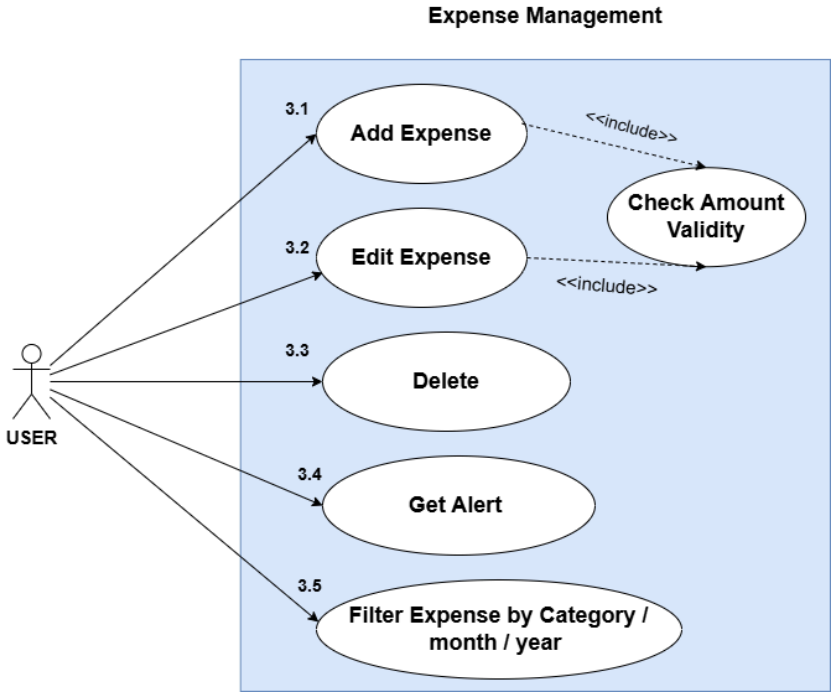


Figure 2.4.1 Use Case diagram level 1 for expense management

Income Management:

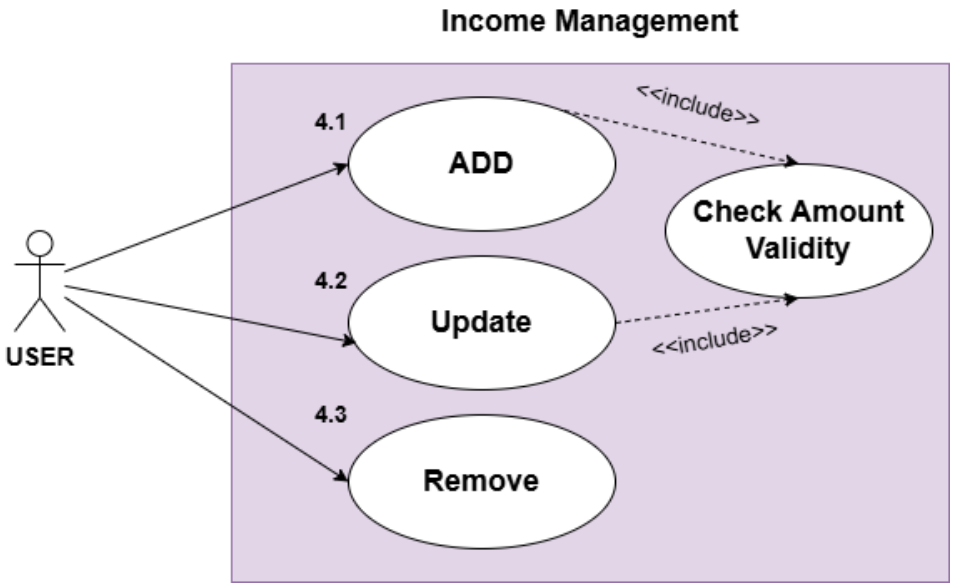


Figure 2.4.1 Use Case diagram level 1 for income mangement

Budget System:

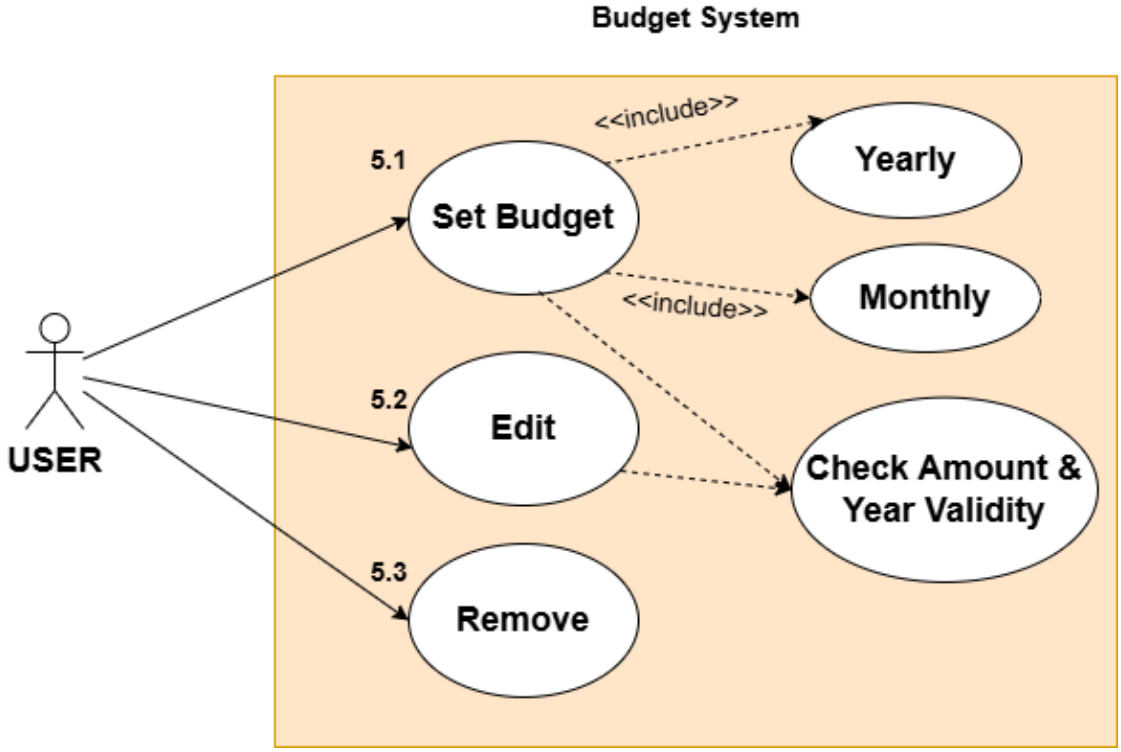


Figure 2.4.1 Use Case diagram level 1 for budget system

Report Generation:

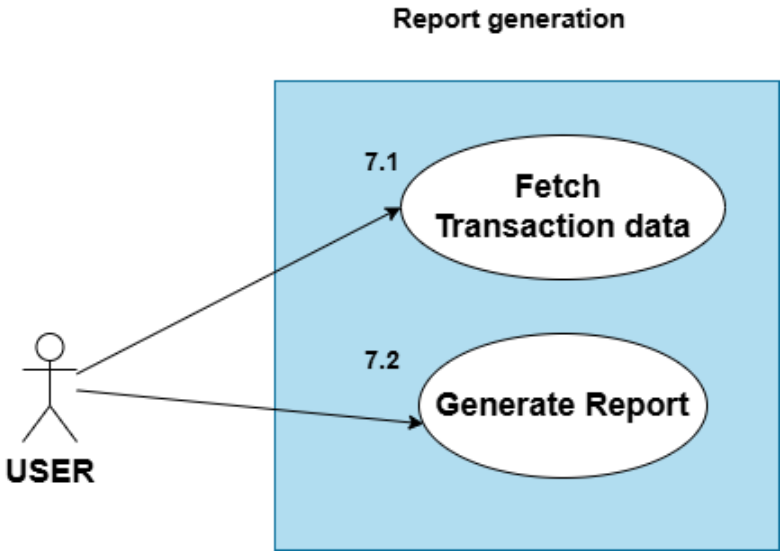


Figure 2.4.1 Use Case diagram level 1 for report generation

2.4.2 Activity Diagrams

Login:

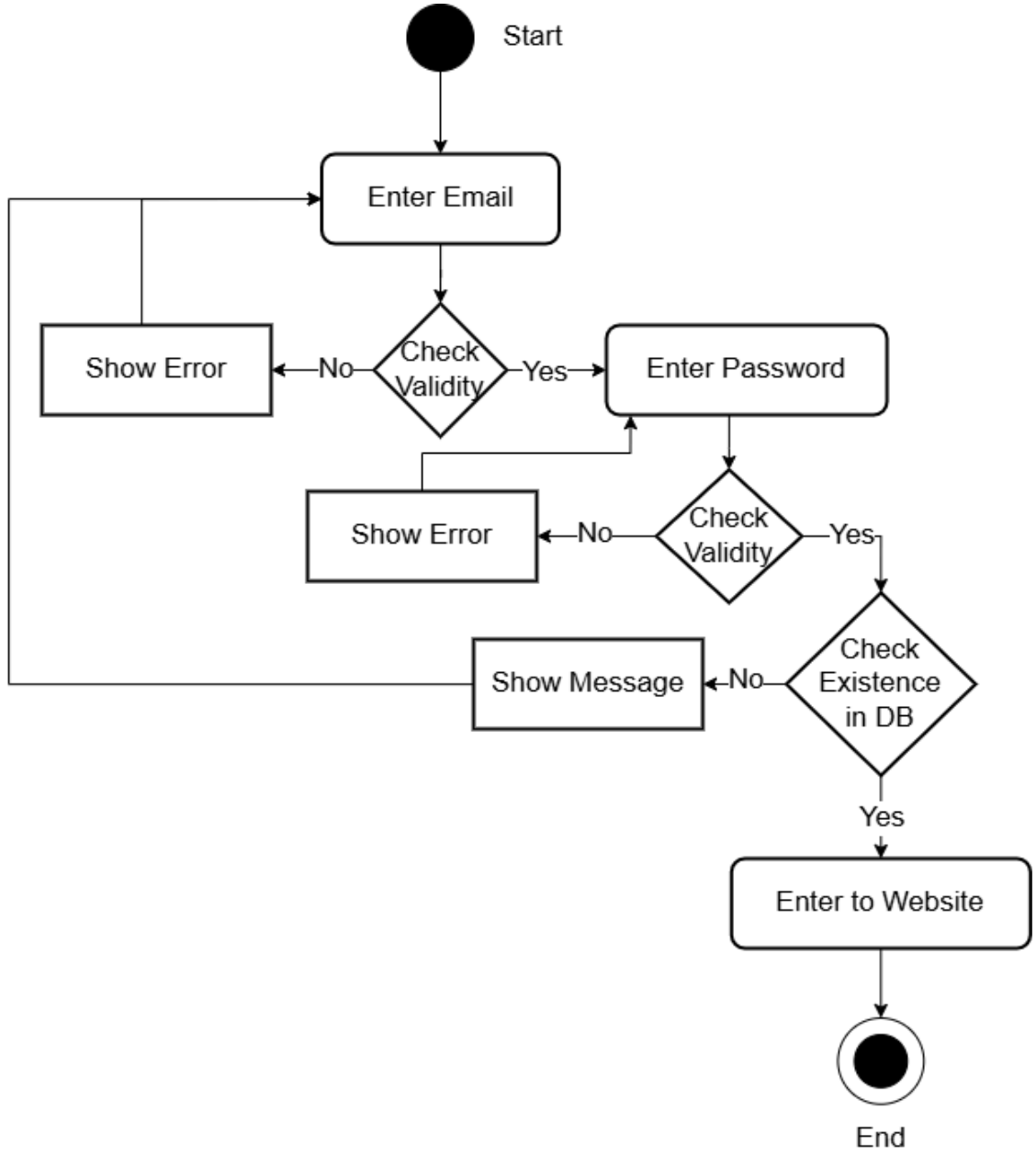


Figure 2.4.2 Activity diagram for login

Sign up

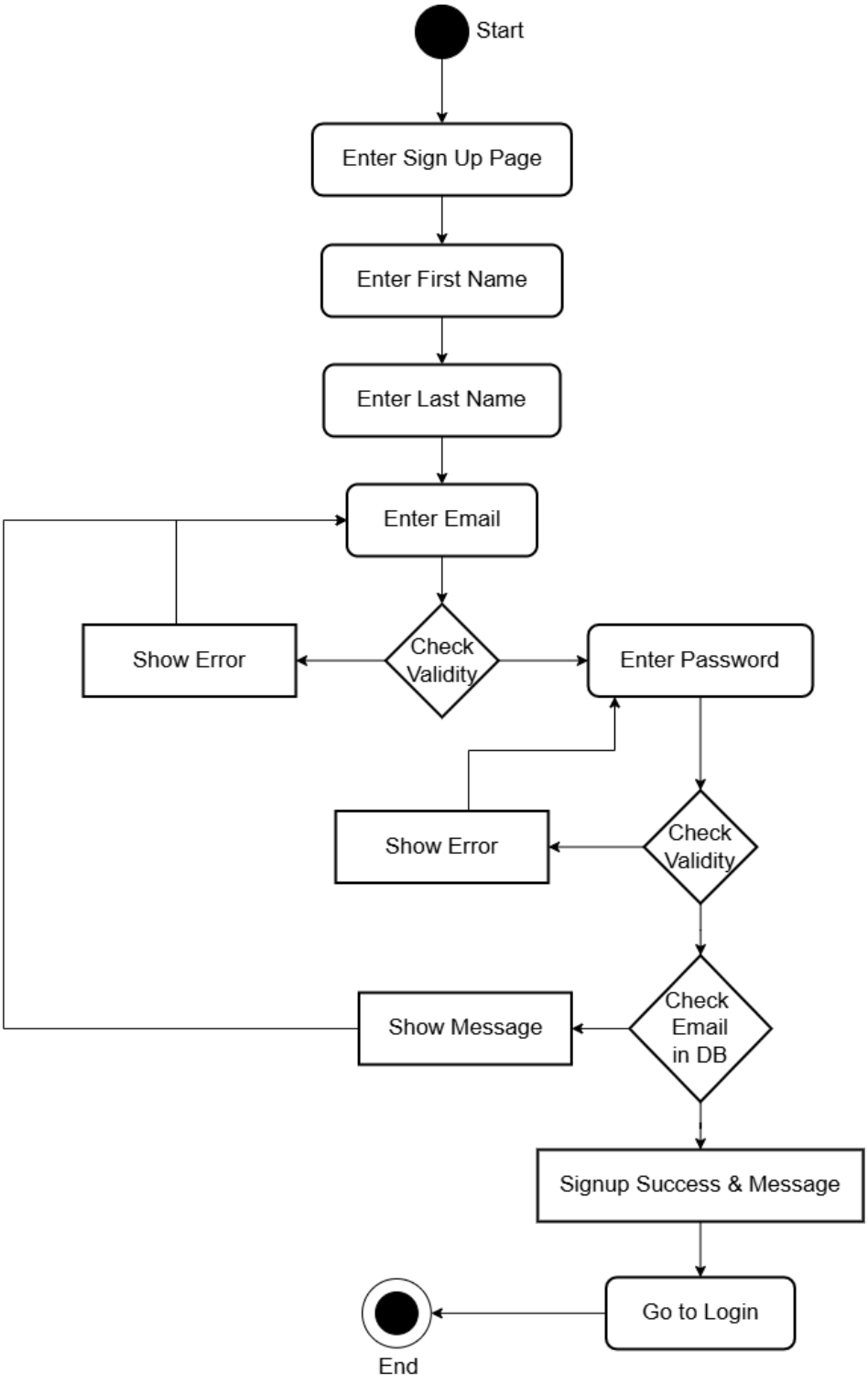


Figure 2.4.2 Activity diagram for sign up

Dashboard:

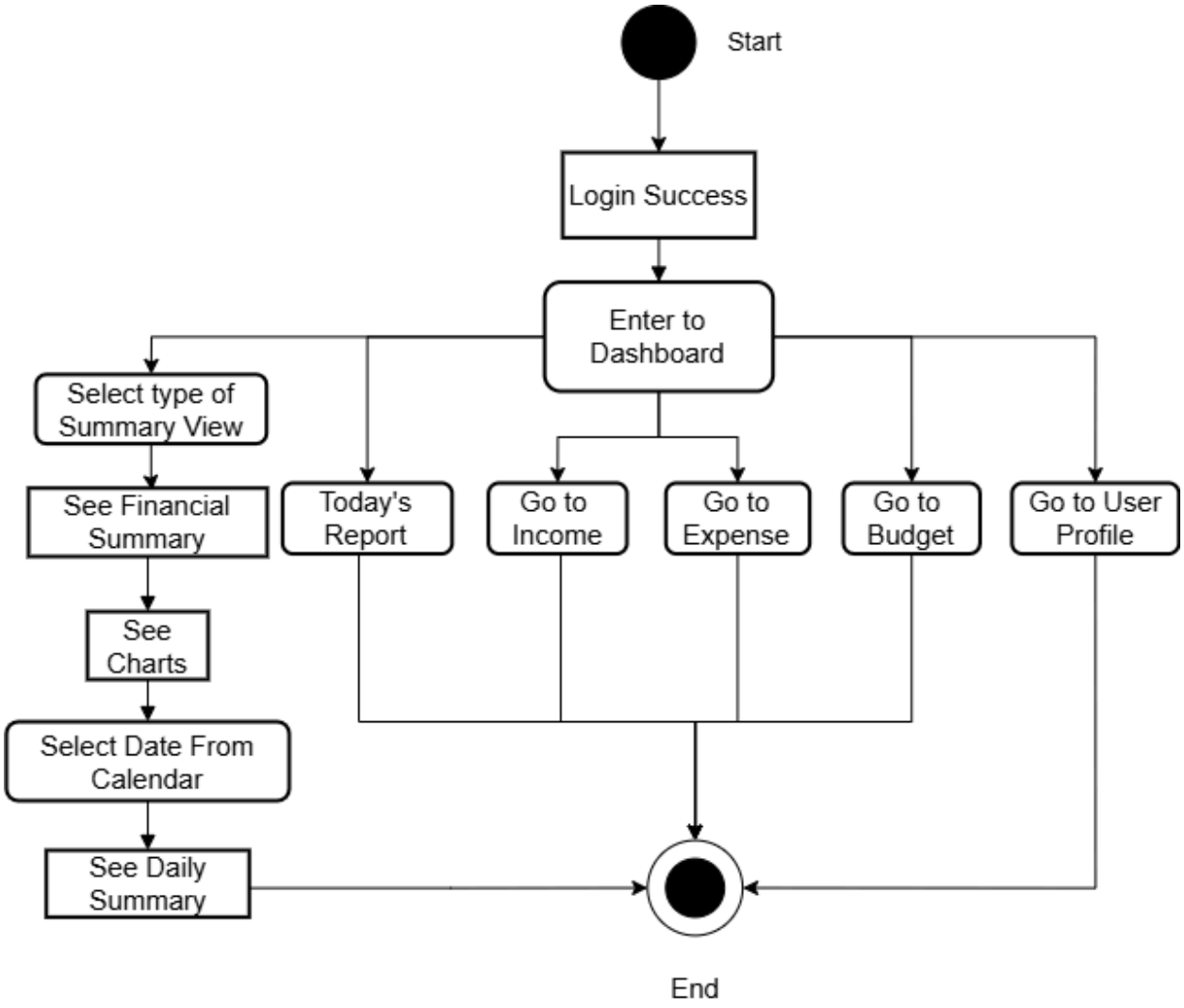


Figure 2.4.2 Activity diagram for dashboard

Income management:

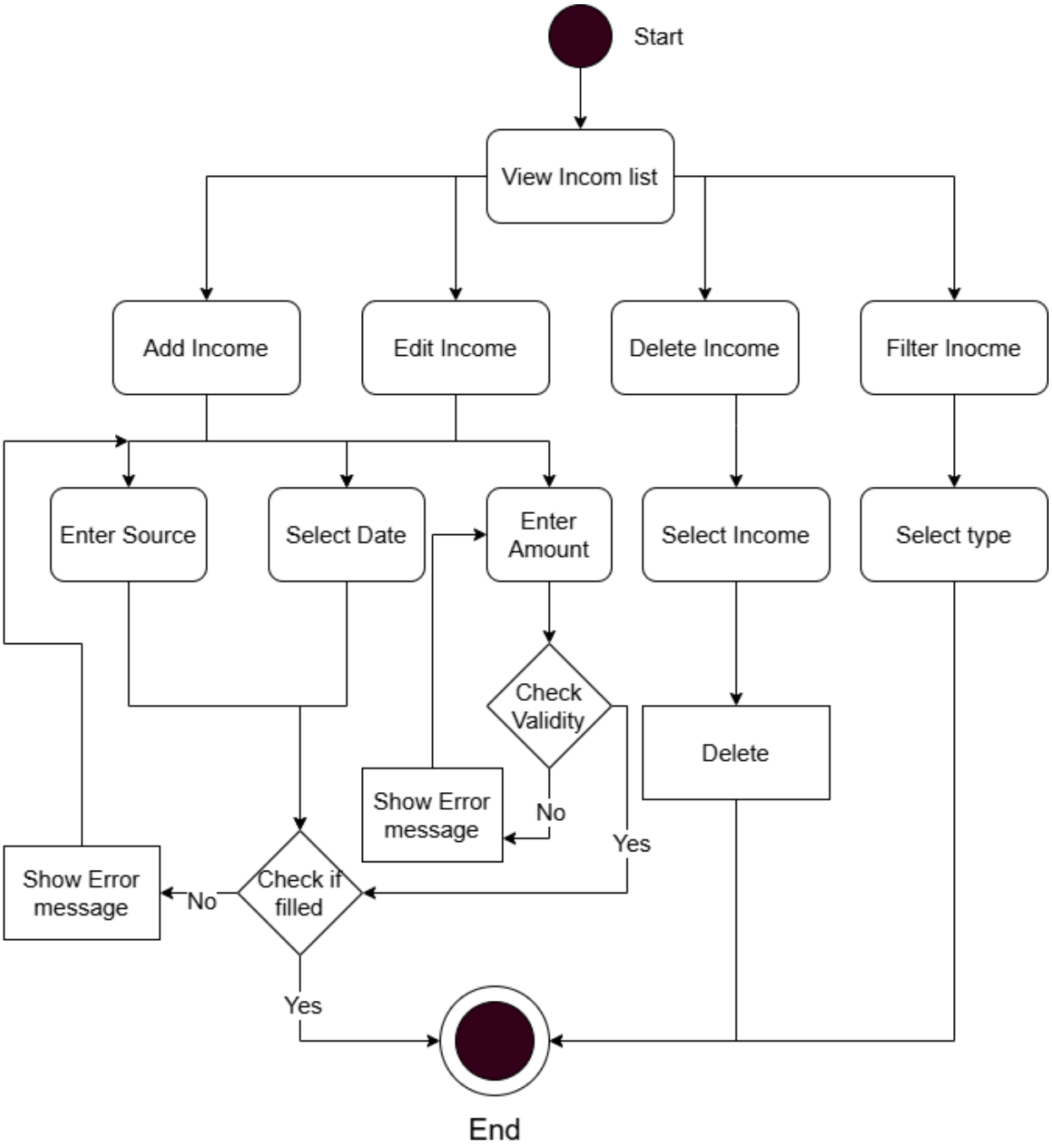


Figure 2.4.2 Activity diagram for income management

Expense System:

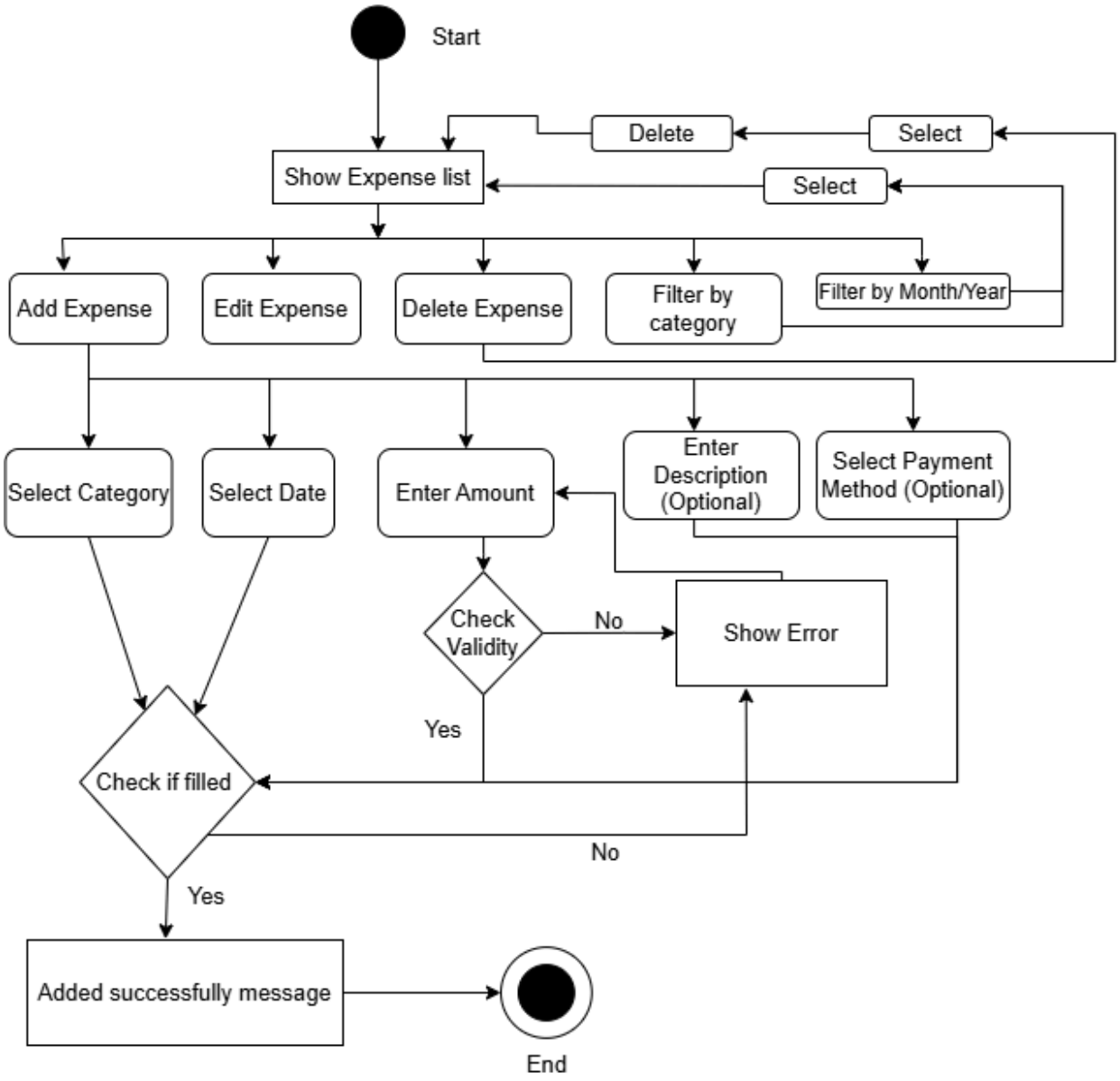


Figure 2.4.2 Activity diagram for expense system

Budget System:

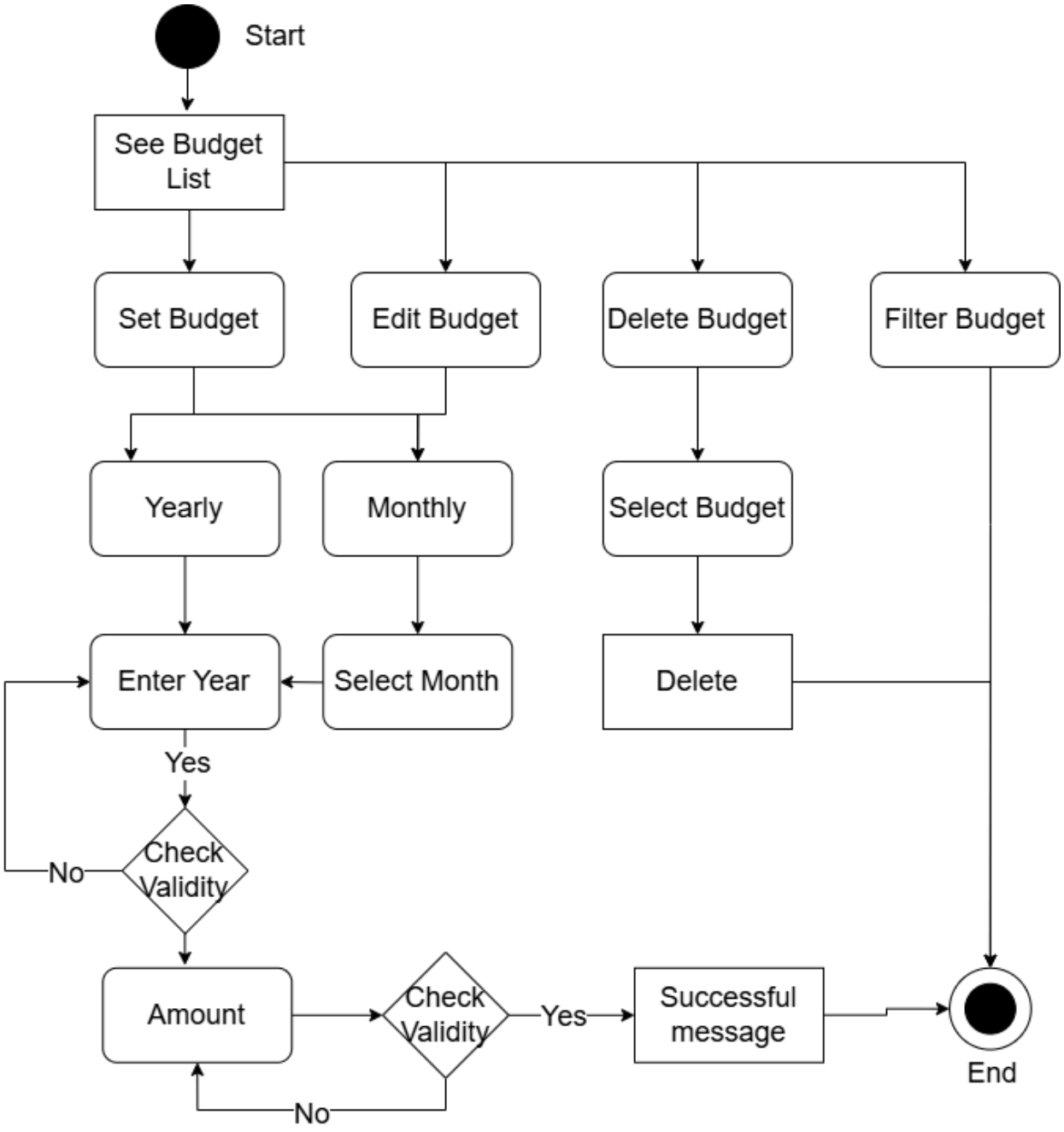


Figure 2.4.2 Activity diagram for budget system

User profile:

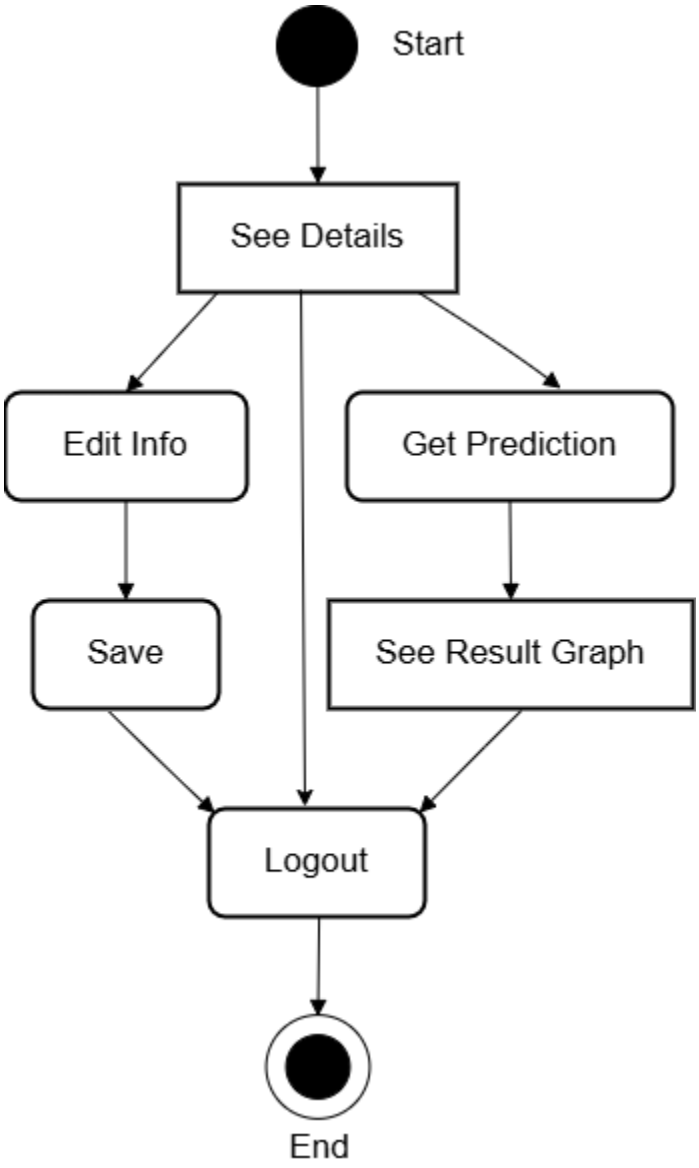


Figure 2.4.2 Activity diagram for user profile

2.4.3 UI Sketches

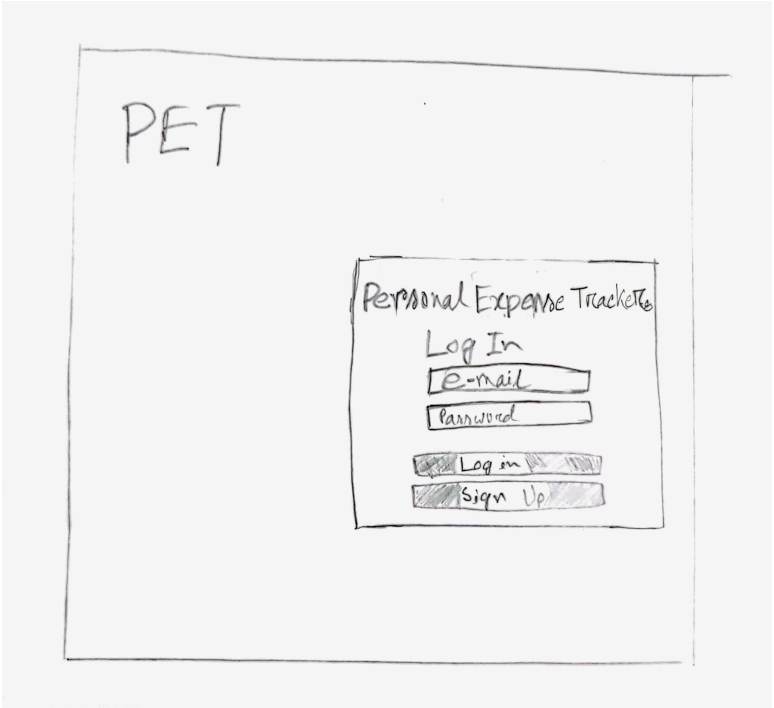


Figure 2.4.3 ui sketches for login

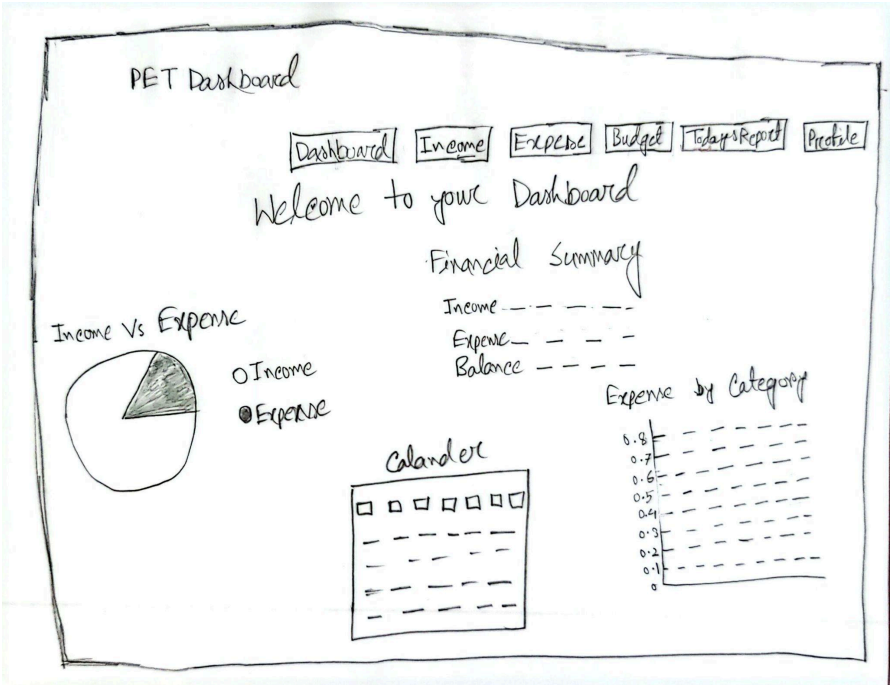


Figure 2.4.3 ui sketches for dashboard

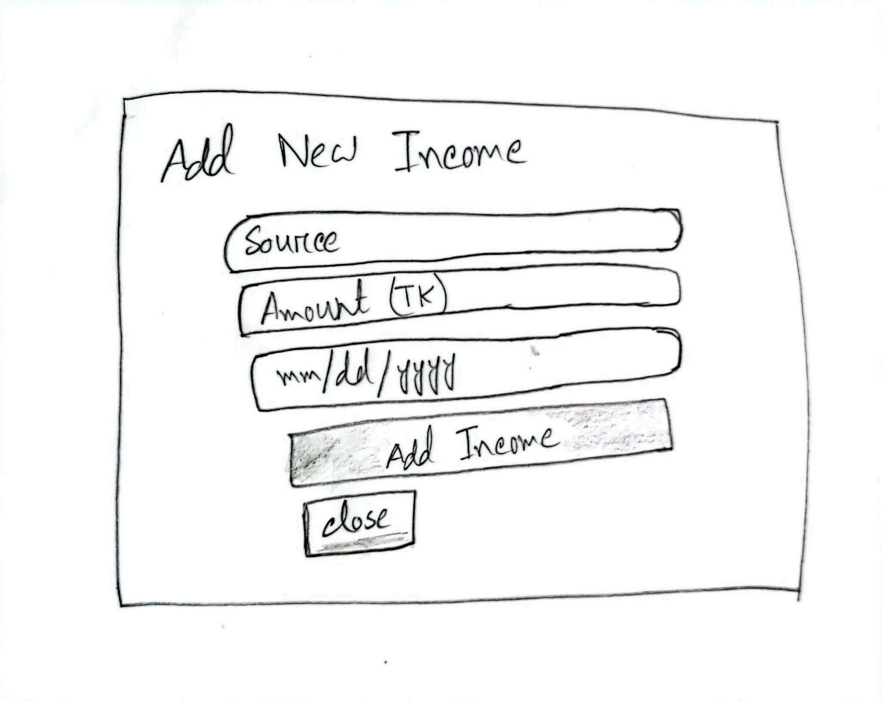


Figure 2.4.3 ui sketches for income form

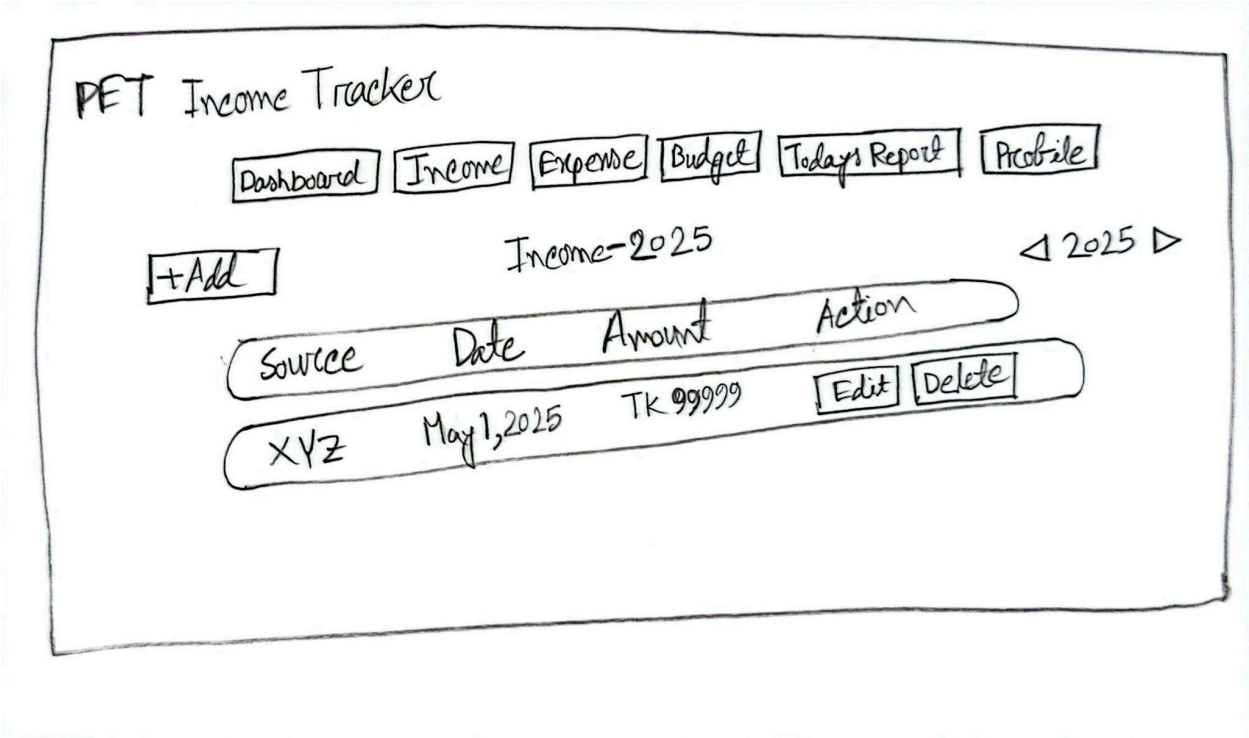


Figure 2.4.3 ui sketches for income tracker

Chapter 3: Software Design

3.1 Architectural Design:

Data-Centered Architectures:

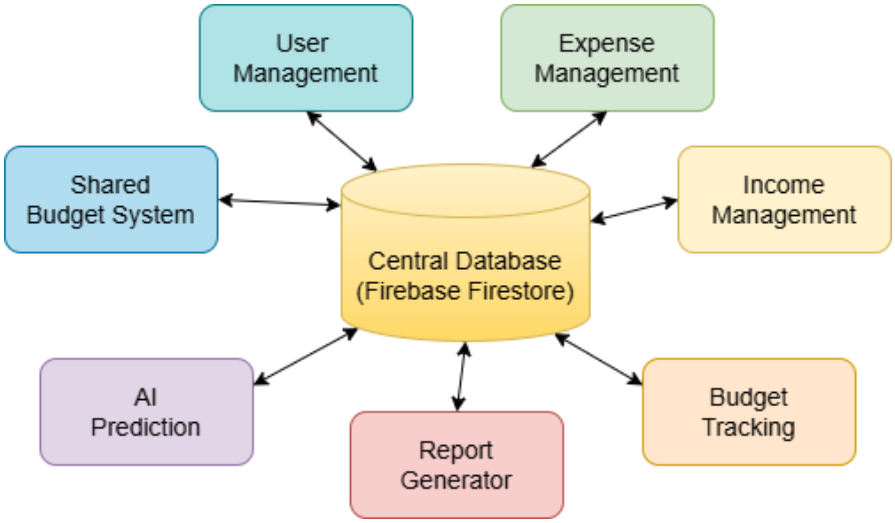


Figure 3.1 Data-Centered Architectures

Architectural Context Diagram: (Level-0)

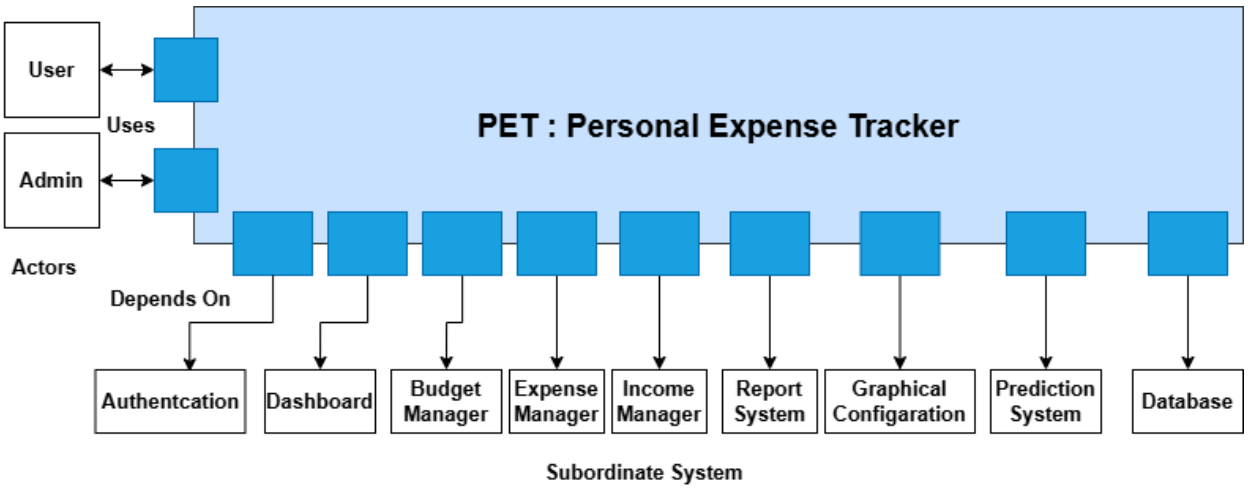


Figure 3.1 Architectural Context Diagram: (Level-0)

Architectural Context Diagram: (Level 1)

Authentication :

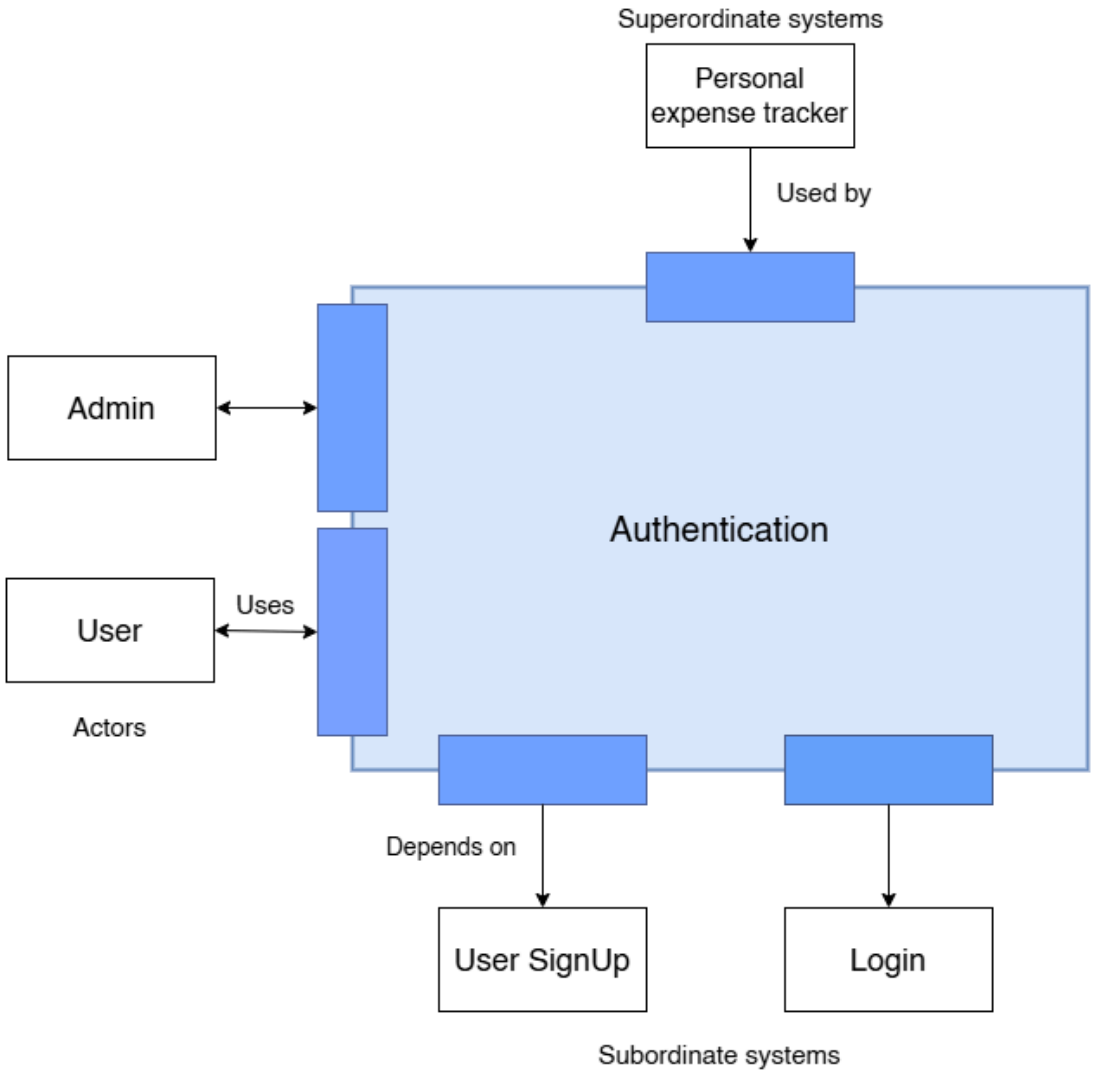


Figure 3.1 Architectural Context Diagram: (Level-1) for authentication

Dashboard :

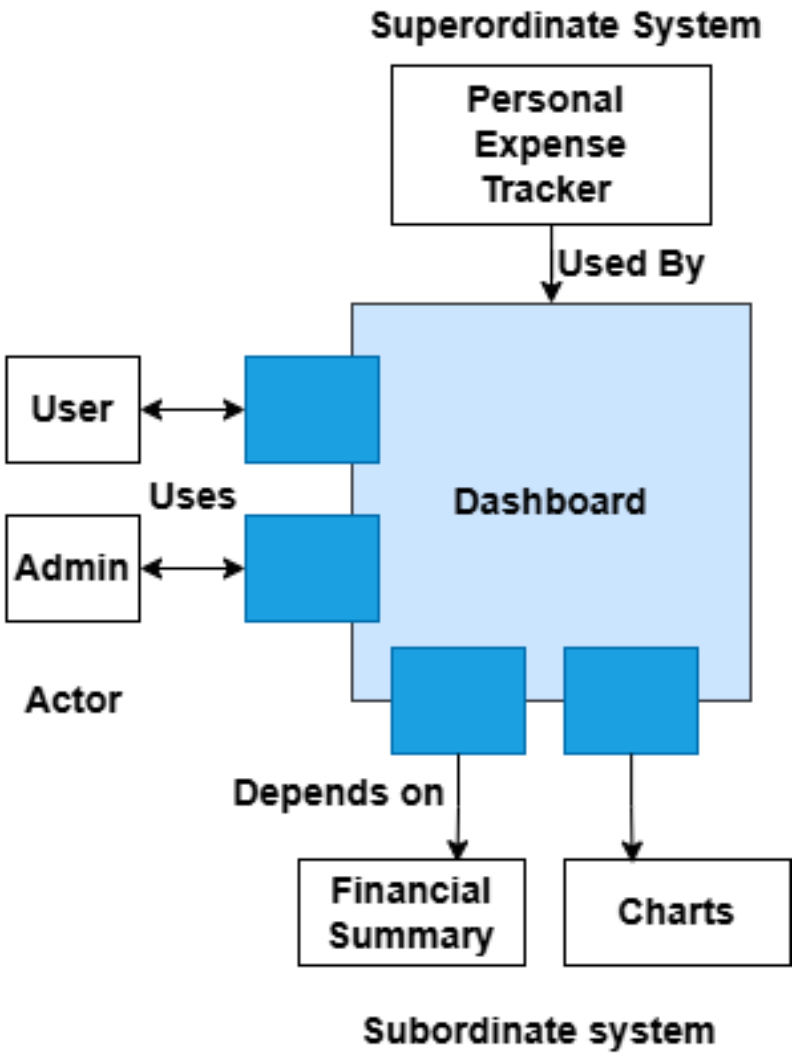


Figure 3.1 Architectural Context Diagram: (Level-1) dashboard

Budget :

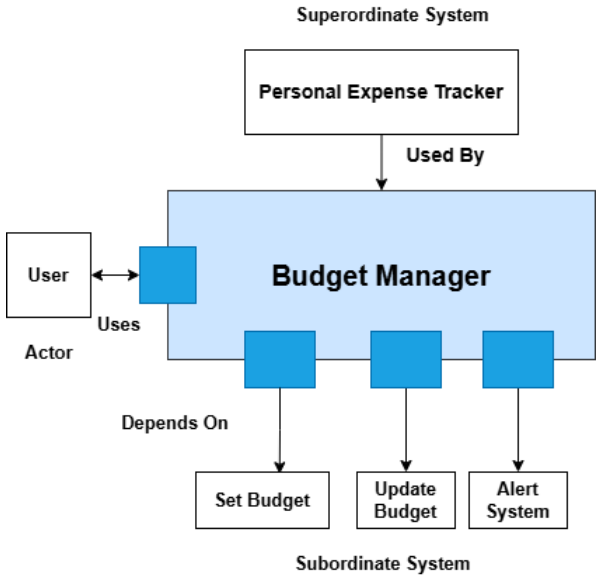


Figure 3.1 Architectural Context Diagram: (Level-1) for budget

Expense Manager:

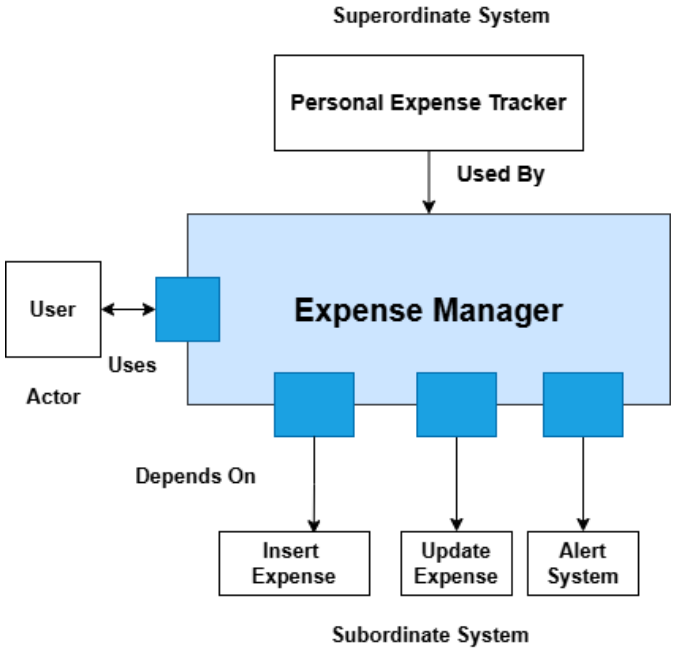


Figure 3.1 Architectural Context Diagram: (Level-1) for expense manager

Income Manager:

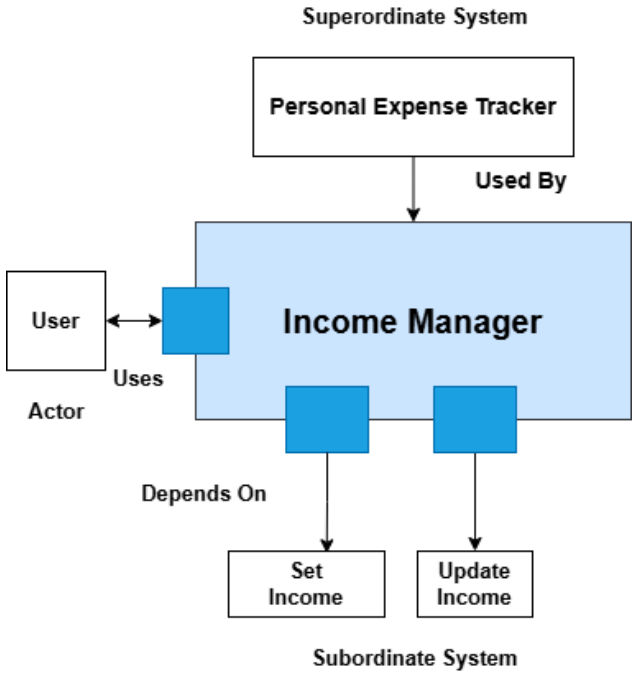


Figure 3.1 Architectural Context Diagram: (Level-1) for income manager

Report System:

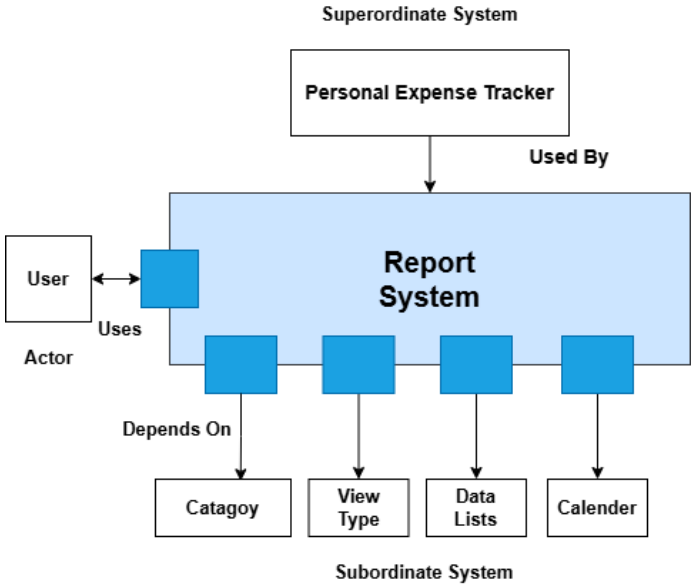


Figure 3.1 Architectural Context Diagram: (Level-1) report system

Graphical Configuration :

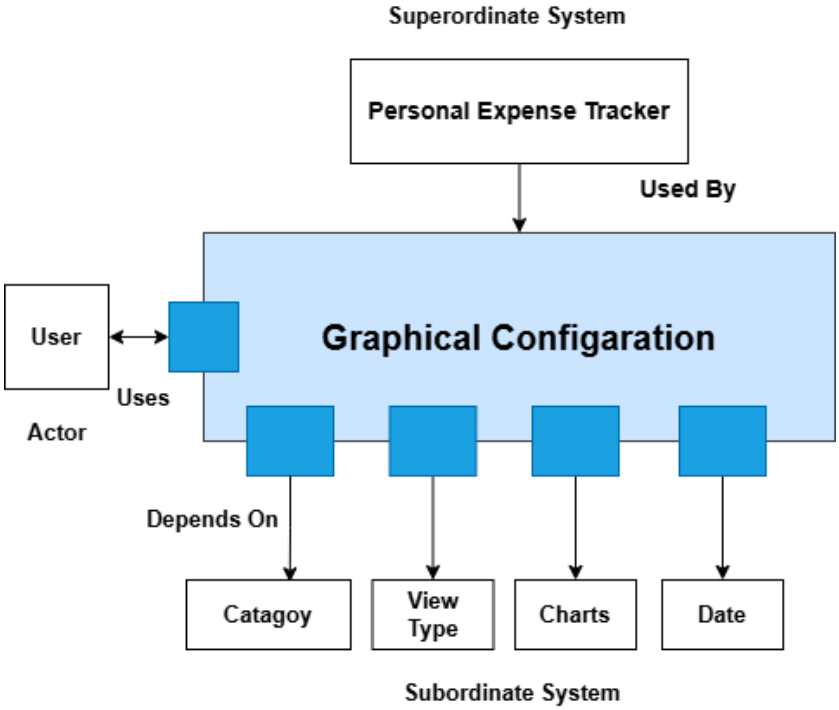


Figure 3.1 Architectural Context Diagram: (Level-1) graphical configuration

Prediction System:

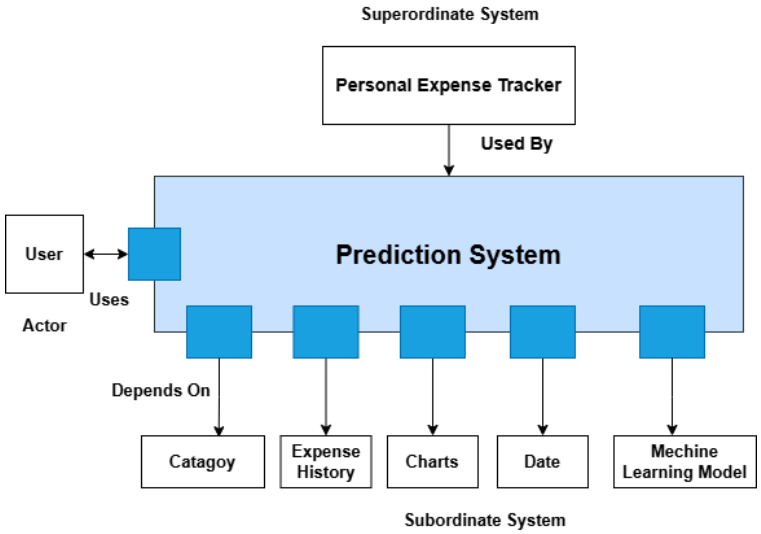


Figure 3.1 Architectural Context Diagram: (Level-1) prediction system

3.1.2 Top-Level Component Diagram:

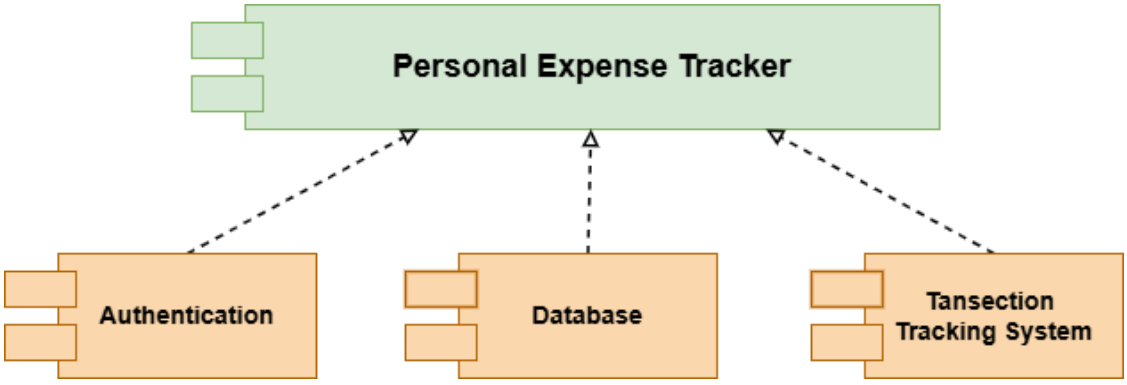


Figure 3.1.2: Top-Level Component Diagram

3.1.3 Instantiation of Each Component with Component Elaboration:

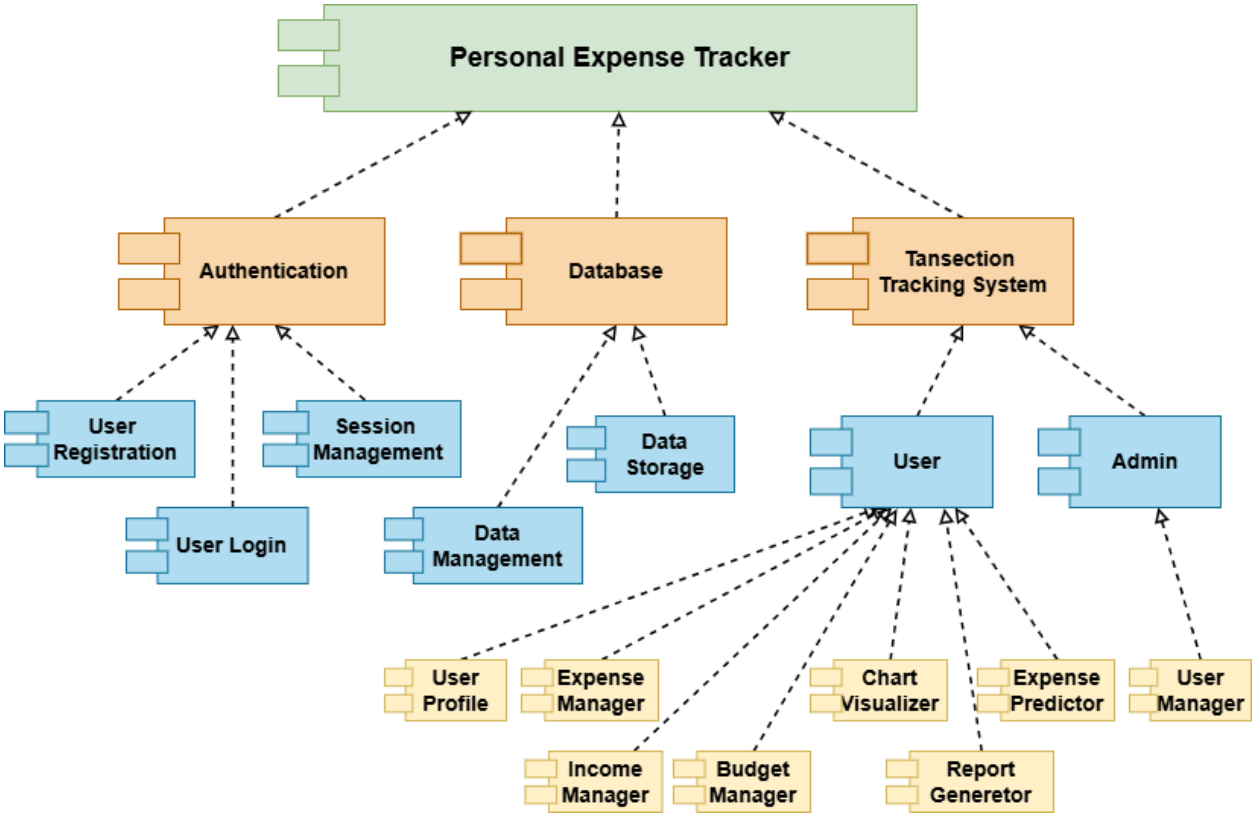


Figure 3.1.2: Instantiation of Each Component with Component Elaboration

3.2 Component-Level Design

3.2.1 Elaboration of Design Components

User Profile:

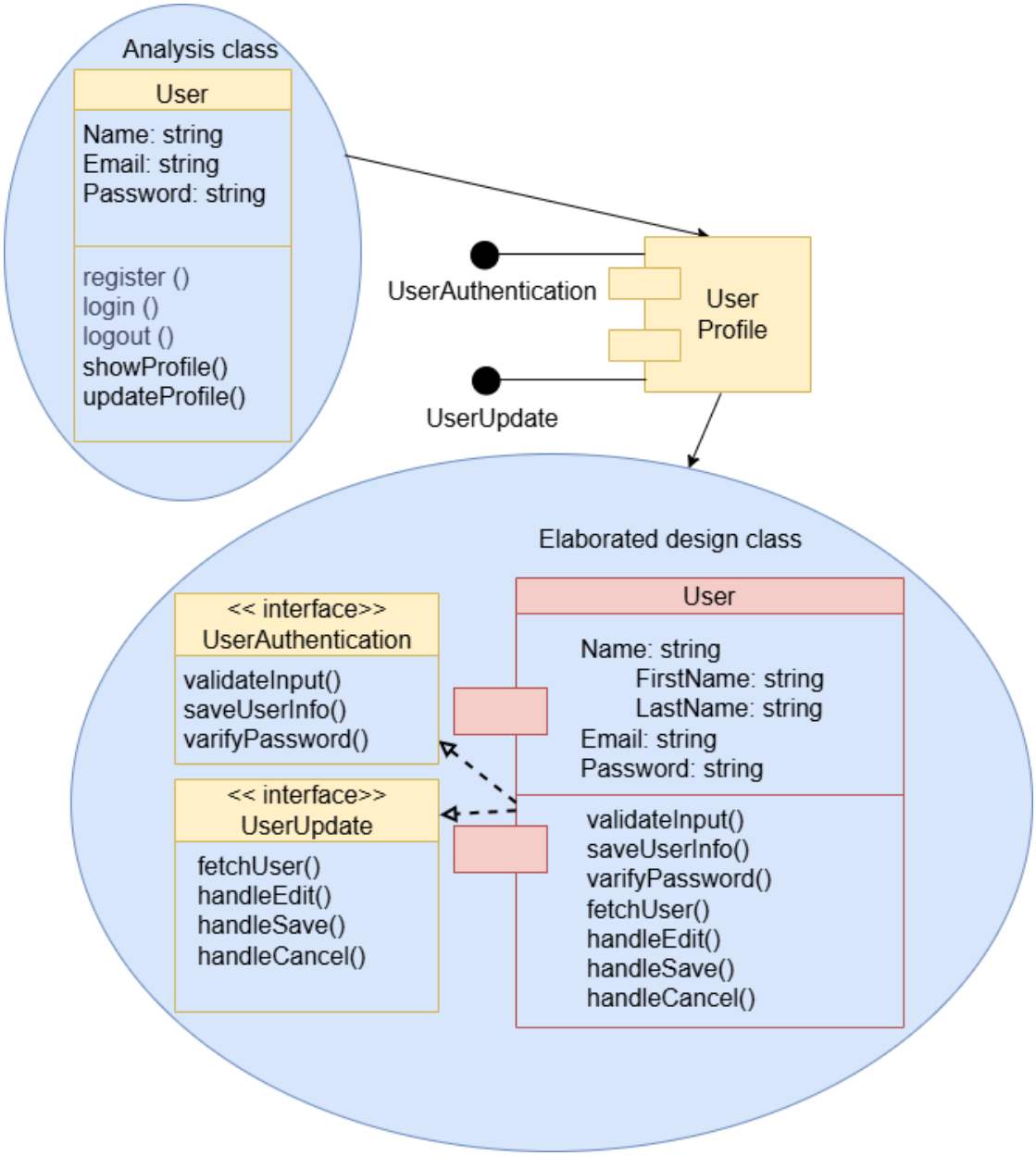


Figure 3.2.1 : Elaboration of Design Components for User profile

Expense Manager:

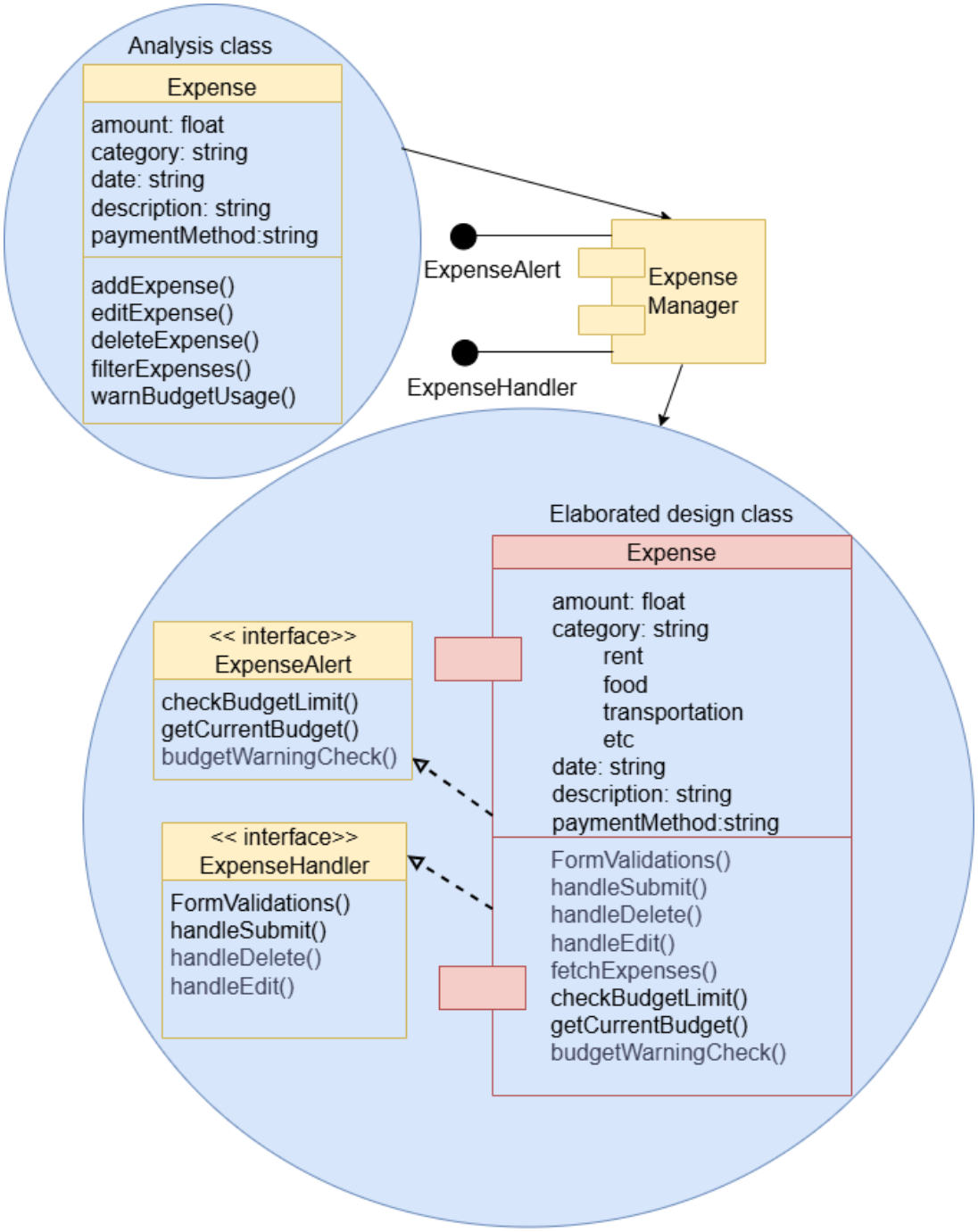


Figure 3.2.1 : Elaboration of Design Components for expense manager

Income Manager:

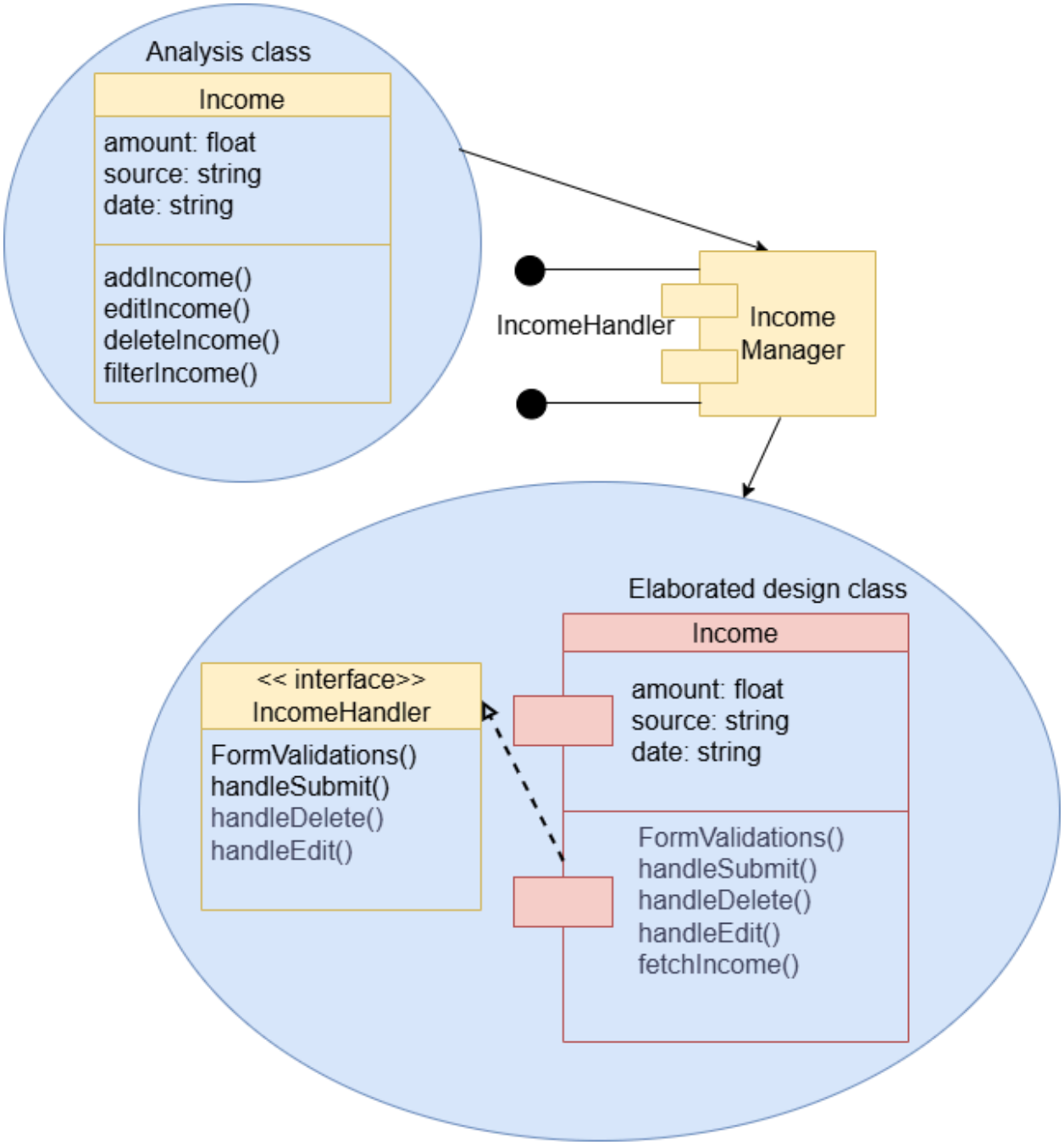


Figure 3.2.1 : Elaboration of Design Components for Income manager

Budget Manager:

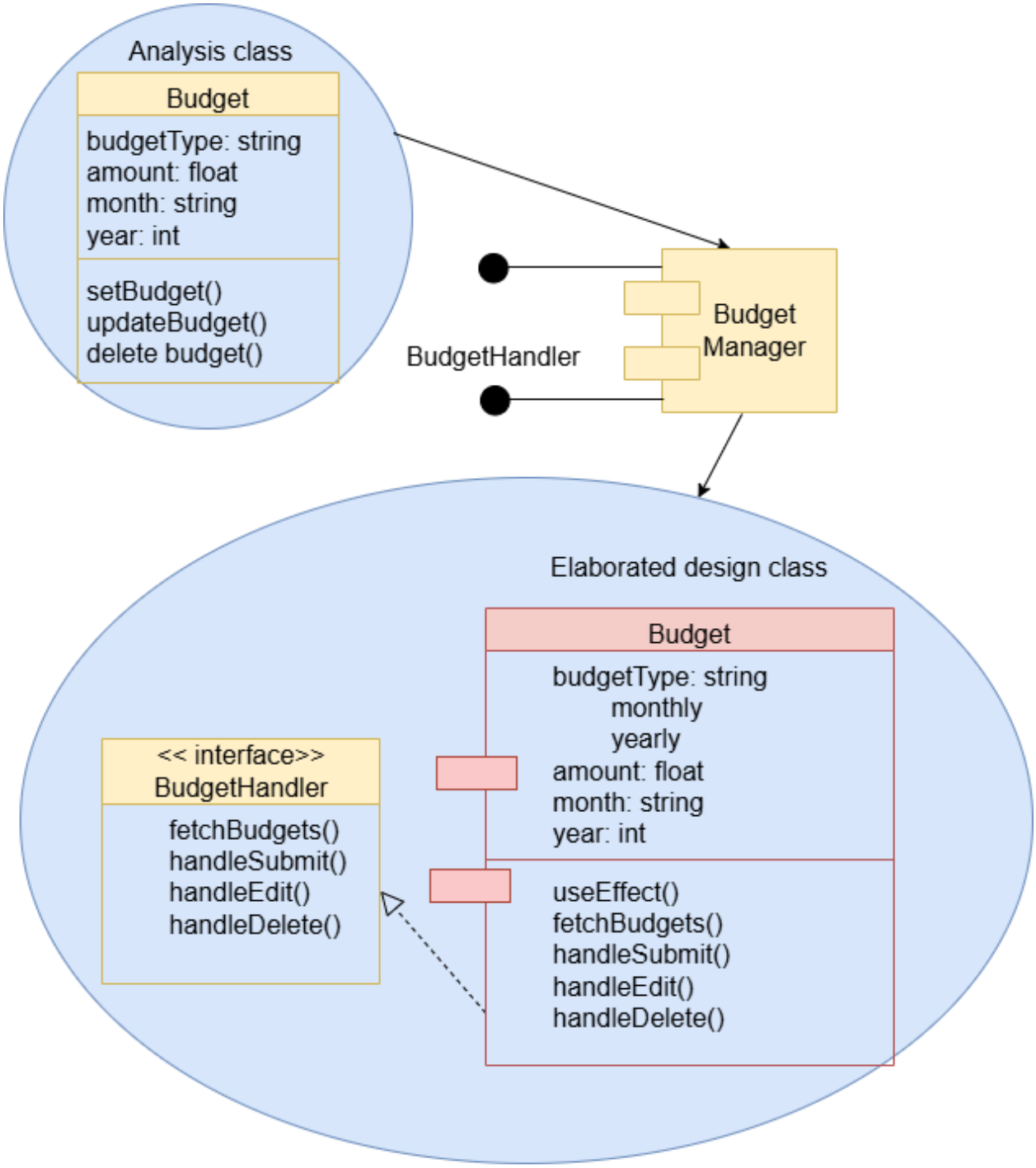


Figure 3.2.1 : Elaboration of Design Components for Budget manager

Report Generator:

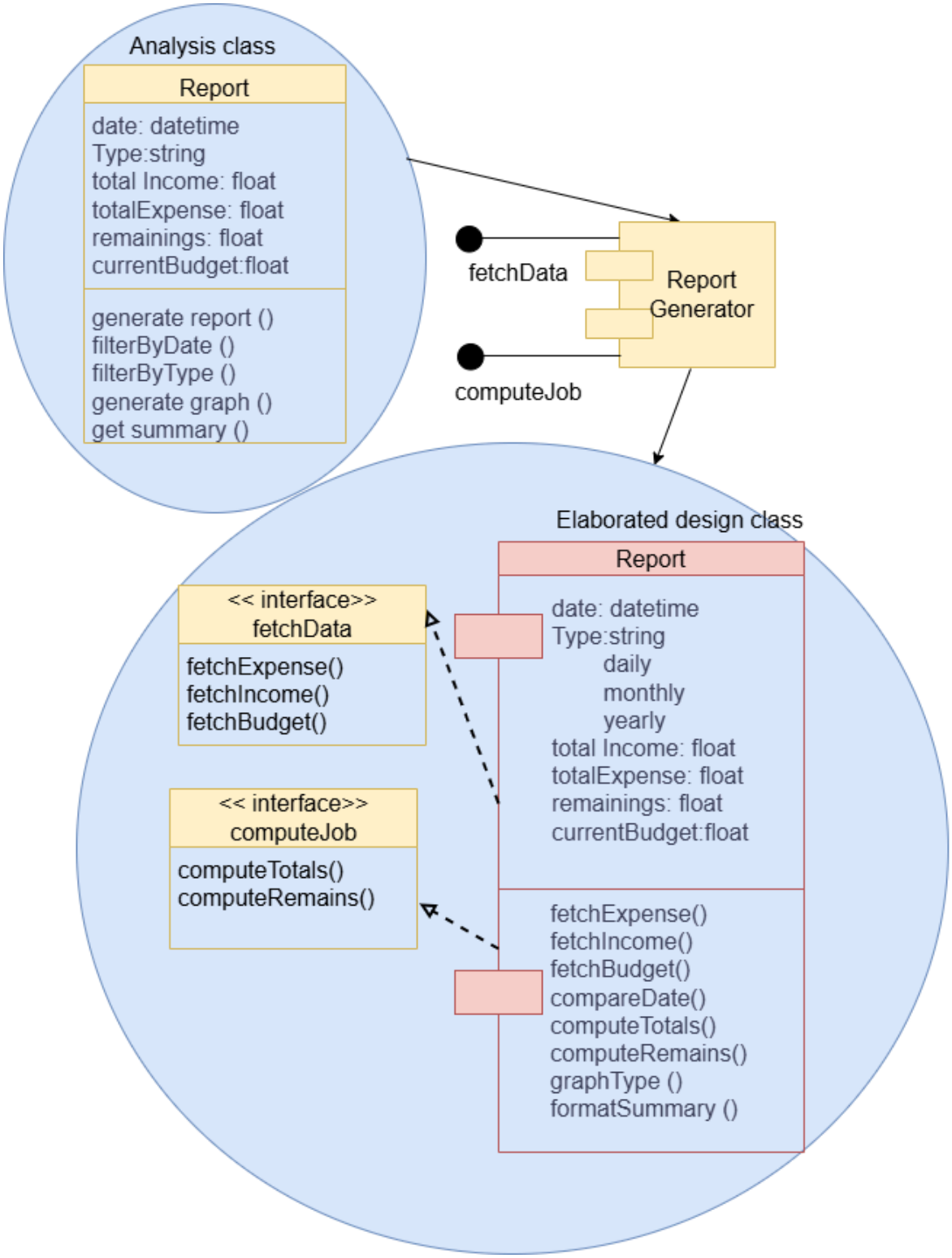


Figure 3.2.1 : Elaboration of Design Components for report generator

Expense Predictor:

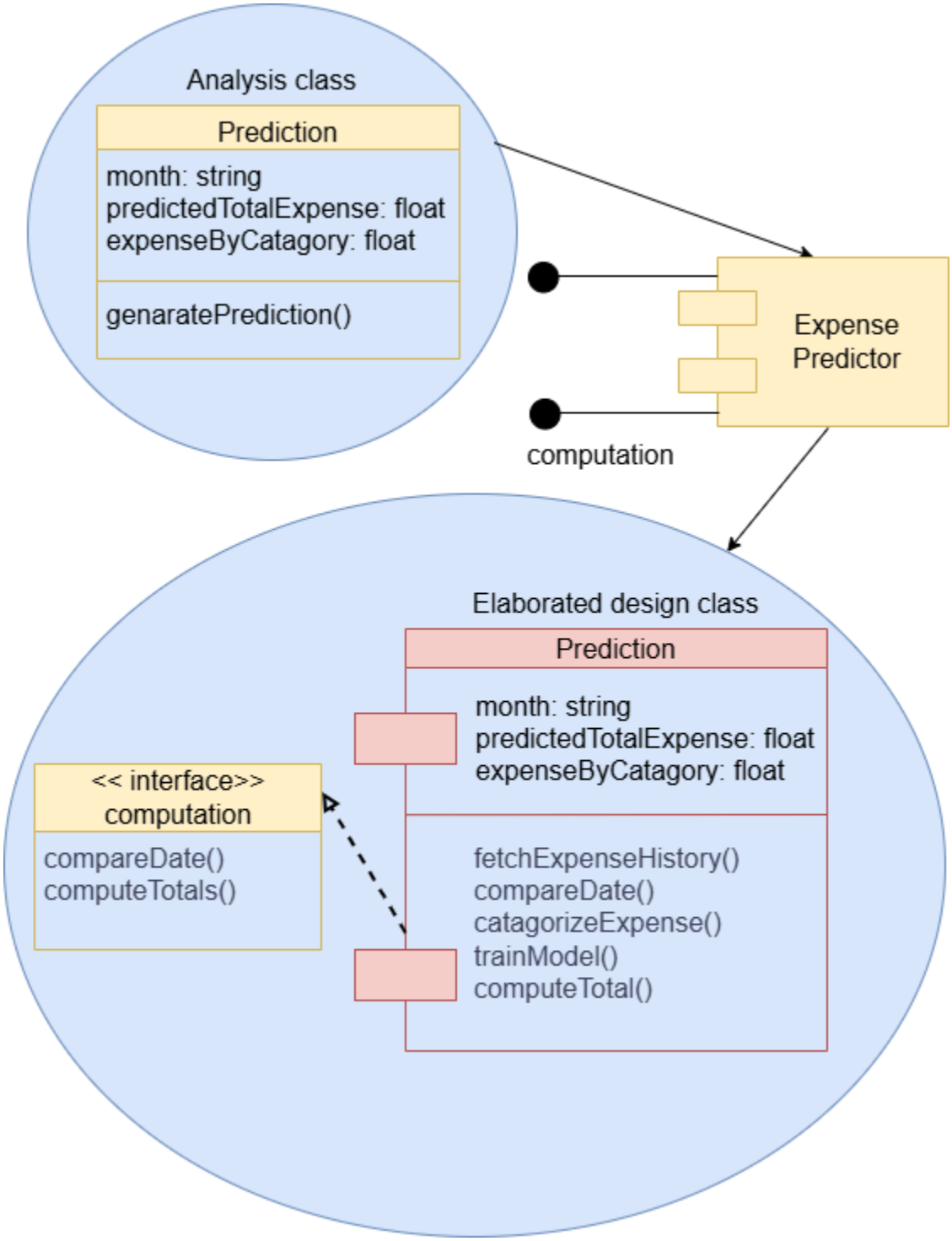


Figure 3.2.1 : Elaboration of Design Components for expense predictor

3.2.2 Class Diagram:

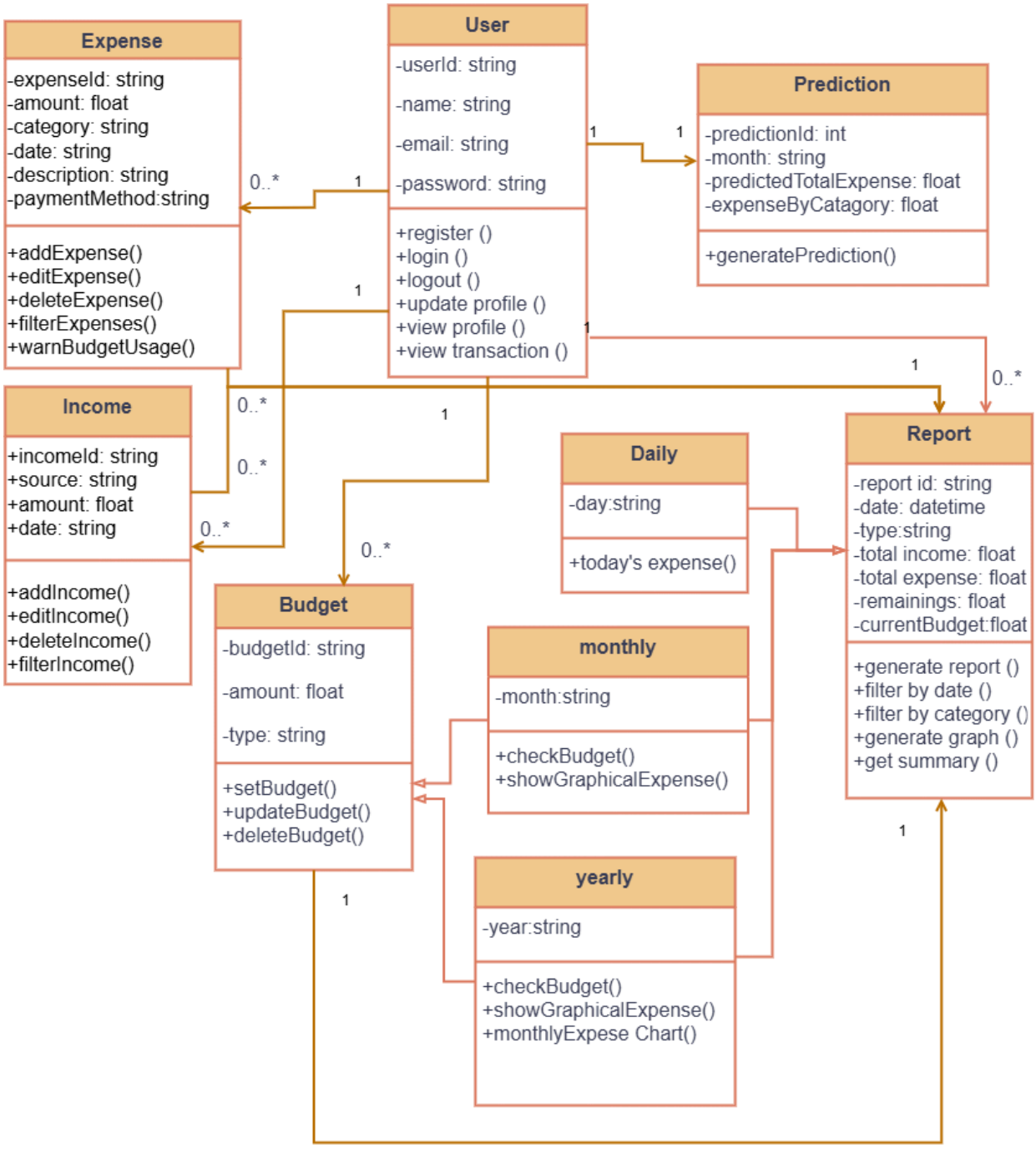


Figure 3.2.2: Class diagram

3.2.3 Database Design

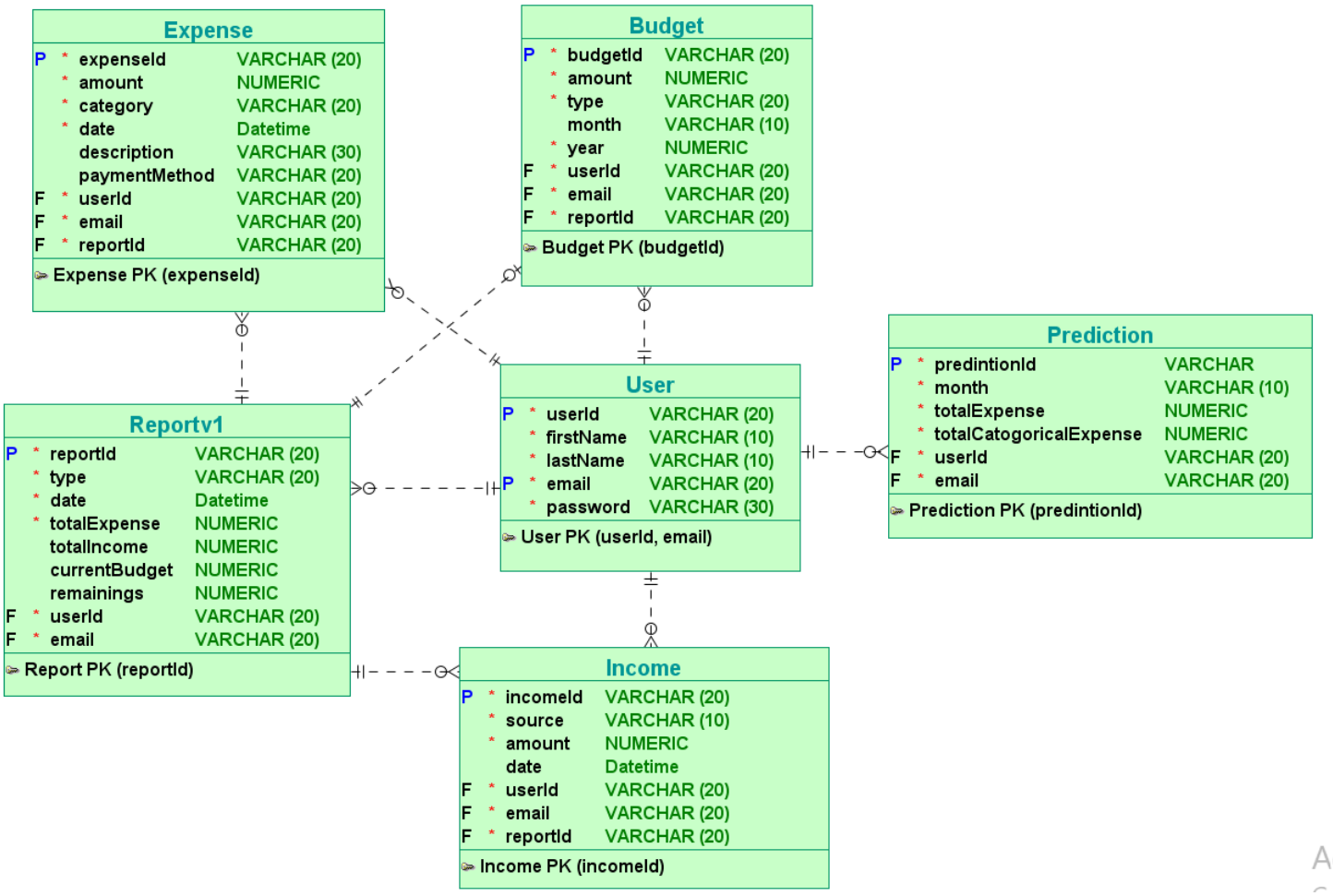


Figure 3.2.3 : ER Diagram

3.3 User Interface Design

UI Wireframes/Mockups/Prototypes

Figure 3.3 : UI Wireframes Prototypes 01

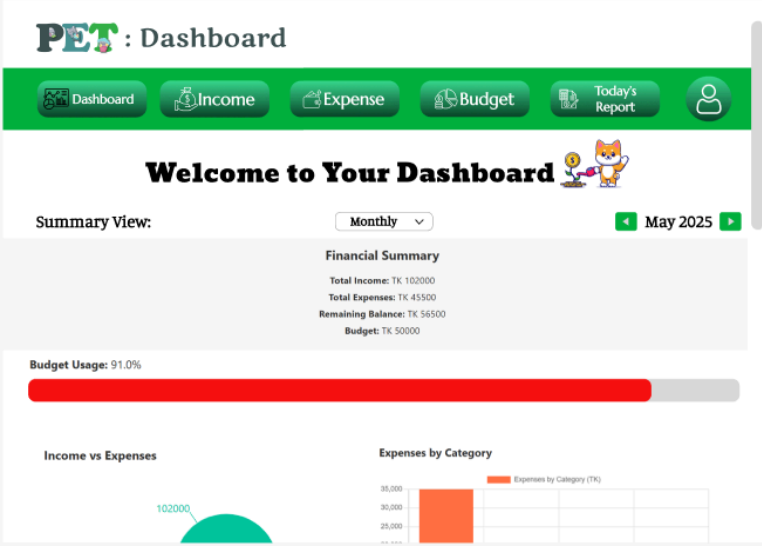


Figure 3.3 : UI Wireframes Prototypes 02

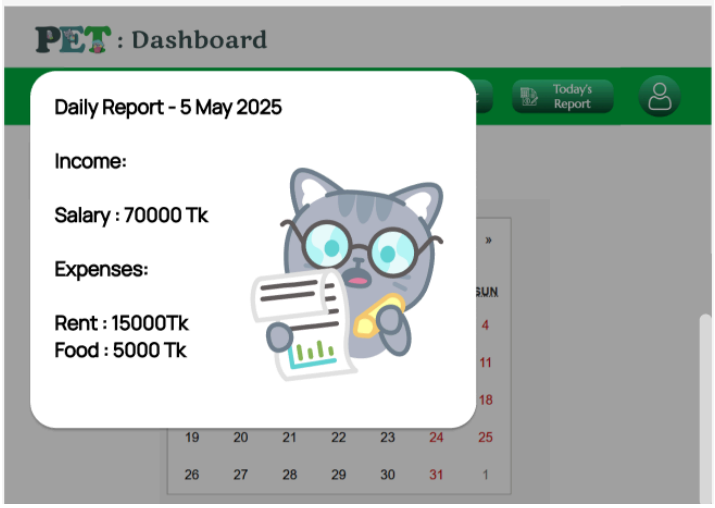


Figure 3.3 : UI Wireframes Prototypes 03

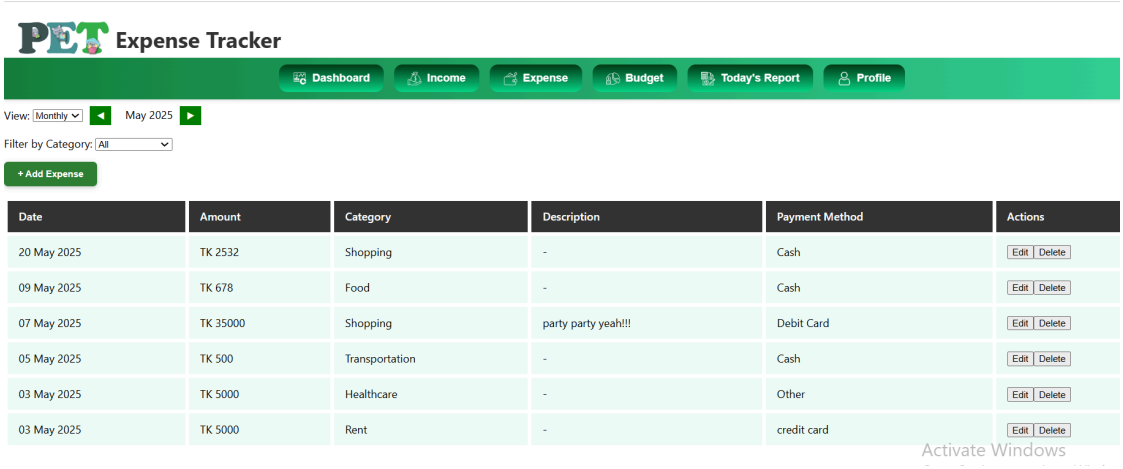


Figure 3.3 : UI Wireframes Prototypes 04

Navigation Flow:

To show the navigation flow we have developed a figma prototype. Below the link for Figma prototype is given:

Figma Prototype for PET: Personal Expense Tracker

Chapter 4: Implementation

4.1 Code Structure Overview

Folder Name	Contents
.firebase	Firebase hosting files
.github	Files to connect github with firebase
build	Application build files by ReactJS
Firebase import	C file code to generate dummy expense values and JS code to import the csv file to the firestore
node_modules	Firebase, react js, node js and other downloaded modules for the application
public	Default react app codes
src	Contains all the actual implemented project codes for the website
src → backgrounds	Contains implemented pictures in the website
src → components	Contains visual component codes for the website
others	Codes to connect firebase git and github and the whole deployment process

Table 4.1: Code Structure Overview

4.2 GitHub Repository URL

https://github.com/Fatema-Tuz-Zannat/Personal-Expense-Tracker

Chapter 5: Software Testing

5.1 White Box Testing:

White box testing is a software testing method in which testers examine the code to understand how it works. White box testing involves developers or testers inspecting a program's internal structure, logic, and code to ensure that everything works as expected. They examine the flow of data, how decisions are made (such as if-else expressions), and how various components of the code interact. Below unit testing on two classes was implemented:

For Expense Class:

handleSubmit( )

```
6   const AddExpenseForm = () => {
21    const handleSubmit = async (e) => {
22      e.preventDefault();
23      if (!category || !amount || !date) {
24        alert('Category, Amount, and Date are required.');
25        return;
26      }
27
28      const parsedAmount = parseFloat(amount);
29      if (parsedAmount <= 0) {
30        alert('Please enter a valid positive amount.');
31        return;
32      }
33
34      const expenseDate = new Date(date);
35      const year = expenseDate.getFullYear().toString();
36      const monthName = expenseDate.toLocaleString('default', { month: 'long' });
37
38      try {
39        await addDoc(collection(db, 'expenses'), {
40          userId: currentUser.uid,
41          category,
42          amount: parsedAmount,
43          date,
44          description: description || '',
```

```
      paymentMethod: paymentMethod || '',
    });

    const expenseQuery = query(
      collection(db, 'expenses'),
      where('userId', '==', currentUser.uid)
    );
    const expenseSnapshot = await getDocs(expenseQuery);
    const expenses = expenseSnapshot.docs
      .map(doc => doc.data())
      .filter(exp => {
        const expDate = new Date(exp.date);
        return (
          expDate.getMonth() === expenseDate.getMonth() &&
          expDate.getFullYear() === expenseDate.getFullYear()
        );
      });

    const totalExpenses = expenses.reduce((sum, exp) => sum + exp.amount, 0);

    const budgetQuery = query(
      collection(db, 'budgets'),
```

```
      where('userId', '==', currentUser.uid)
    );
    const budgetSnapshot = await getDocs(budgetQuery);
    const budgets = budgetSnapshot.docs.map(doc => ({ id: doc.id, ...doc.data() }));

    const monthlyBudget = budgets.find(b => b.type === 'monthly' && b.month === monthName && b.year.toS
    const yearlyBudget = budgets.find(b => b.type === 'yearly' && b.year?.toString() === year);

    const activeBudget = monthlyBudget || yearlyBudget;

    if (activeBudget && activeBudget.amount > 0) {
      const percentUsed = (totalExpenses / activeBudget.amount) * 100;
      if (percentUsed > 70 && percentUsed <= 100) {
        alert(
          `Warning: You've used ${percentUsed.toFixed(2)}% of your ${activeBudget.type} budget for ${
            activeBudget.type === 'monthly' ? monthName : year
          }.`
        );
      } else if (percentUsed > 100) {
        alert(
          `Warning: You've exceeded your ${activeBudget.type} budget for ${
```

```
            activeBudget.type === 'monthly' ? monthName : year
          } by ${(percentUsed - 100).toFixed(2)}%.`
        );
      }
    }

    setCategory('');
    setAmount('');
    setDate('');
    setDescription('');
    setPaymentMethod('');
    alert('Expense added successfully.');
  } catch (err) {
    console.error(err);
    alert('Failed to add expense.');
  }
};
```

DD Graph (Decision-to-Decision Graph) of handleSubmit( ):

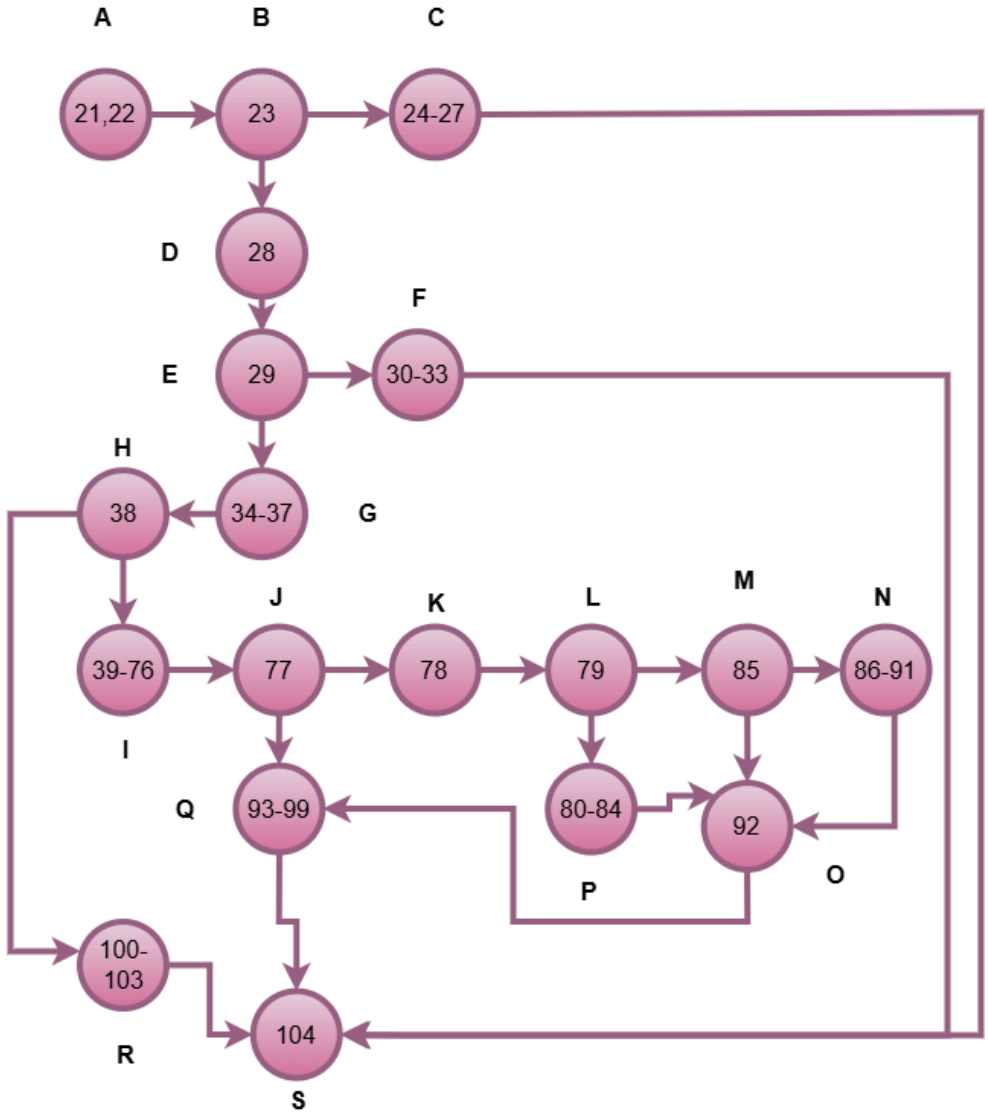


Figure 5.1 DD Graph (Decision-to-Decision Graph) of handleSubmit( )

Cyclomatic complexity of handleSubmit( ):

Using McCabe's formula: V(G) = E - N + 2P

Where:

E = Number of edges = 24

N = Number of nodes = 19

P = Number of connected components = 1

V(G) = E - N + 2P

= 24 – 19 + 2*1

= 7

So, there are 7 independent paths.

Independent Paths:

- A → B → C → S
- A → B → D → E → F → S
- A → B → D → E → G → H → R → S
- A → B → D → E → G → H → I → J → Q → S
- A → B → D → E → G → H → I → J → K → L → M → N → O → Q → S
- A → B → D → E → G → H → I → J → K → L → M → O → Q → S
- A → B → D → E → G → H → I → J → K → L → P → O → Q → S

Test Cases:

Test Case ID	Componen t	Class	Input	Expected Output'	Actual Output	Pass/Fail
WTC1	ExpenseM anager	Expense	category = "", amount = "", date = ""	Please fill out this Form	Please fill out this Form	Pass
WTC2	ExpenseM anager	Expense	category = "Food", amount = "-100", date = "2025-05-0 1"	Alert: "Please enter a valid positive amount."	Alert: "Please enter a valid positive amount."	Pass
WTC3	ExpenseM anager	Expense	Valid input, no budget in DB	Expense added successfull y, no warning	Expense added successfull y, no warning	Pass
WTC4	ExpenseM anager	Expense	Valid input, budget exists, >=70% used	Expense added successfull y, warning shown	Expense added successfull y, warning shown	Pass

WTC5	ExpenseManager	Expense	Valid input, budget exceeded	Expense added successfully, exceed warning	Expense added successfully, exceed warning	Pass
WTC6	ExpenseManager	Expense	Valid input, budget < 70% used	Expense added successfully, no warning	Expense added successfully, no warning	Pass
WTC7	ExpenseManager	Expense	DB write fails (mock error)	Alert: "Failed to add expense."	Alert: "Failed to add expense."	Pass

Table 5.1: white box testing01

Fig 1: WTC1

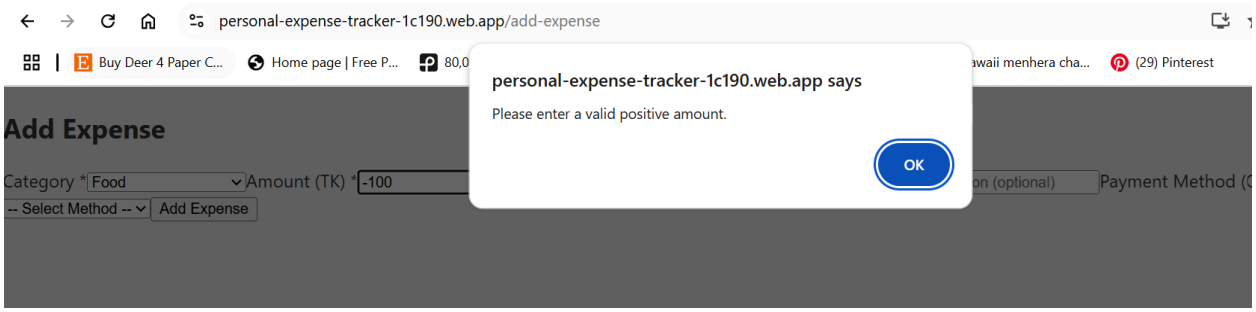


Fig 2: WTC2

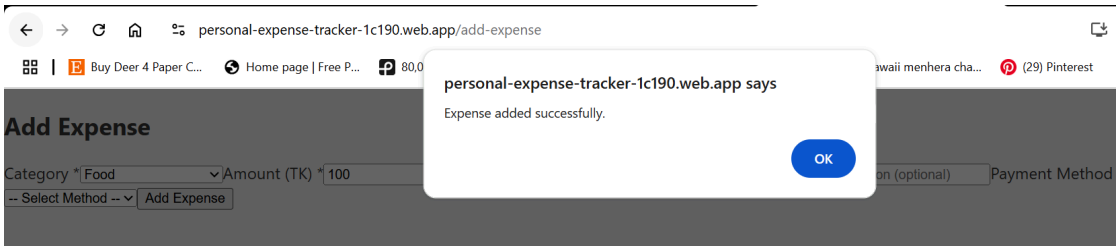
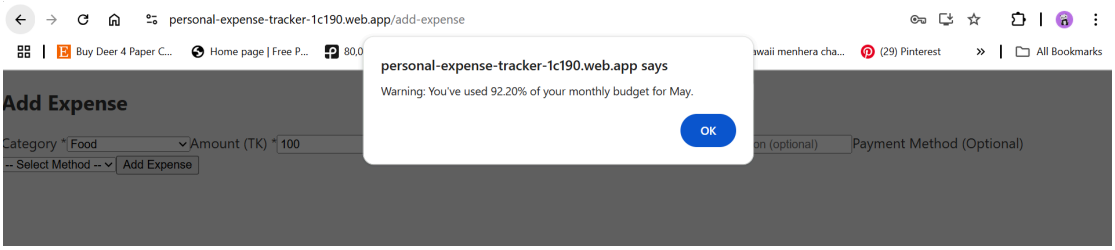


Fig 3: WTC3



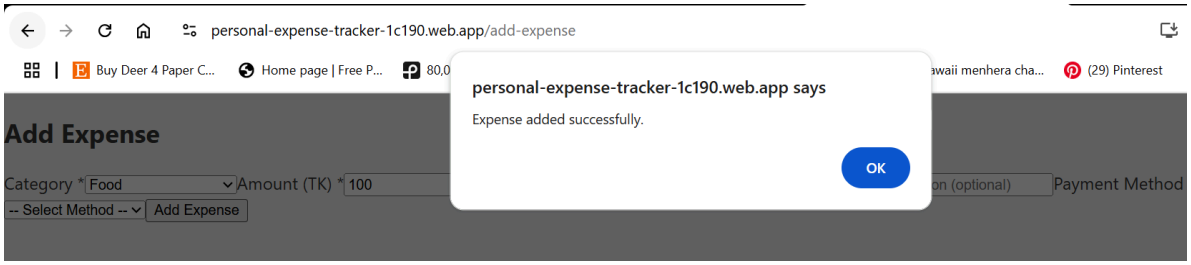
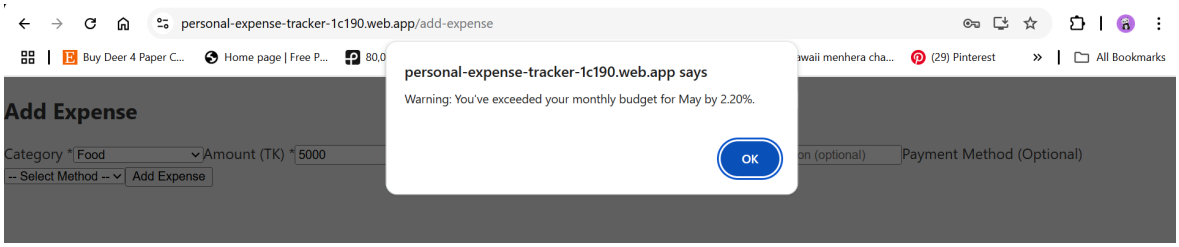


Fig 4: WTC4



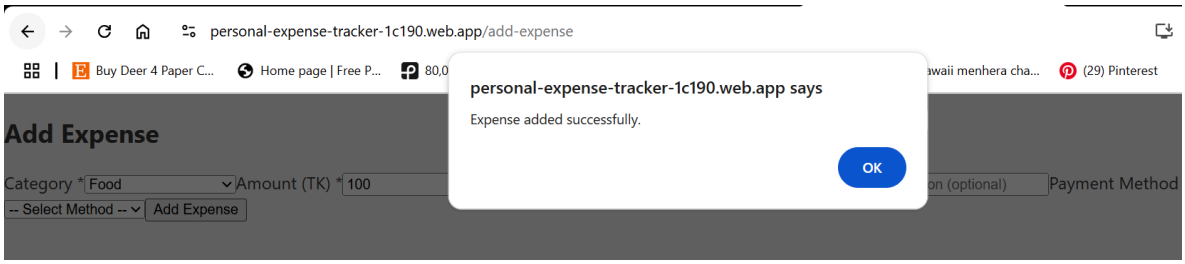


Fig 5: WTC5

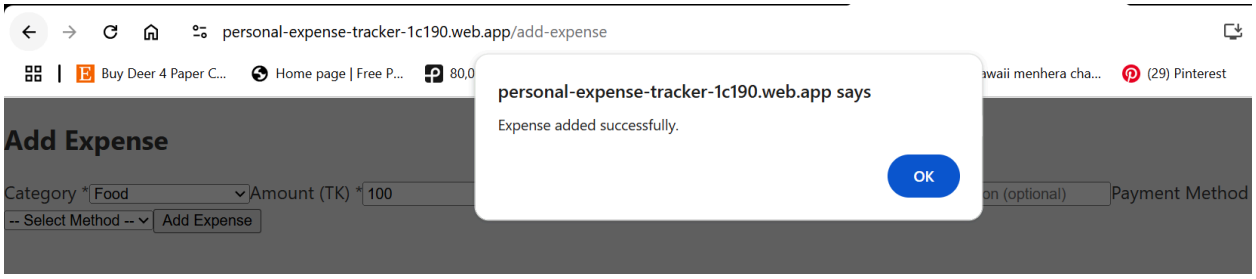


Fig 6: WTC6

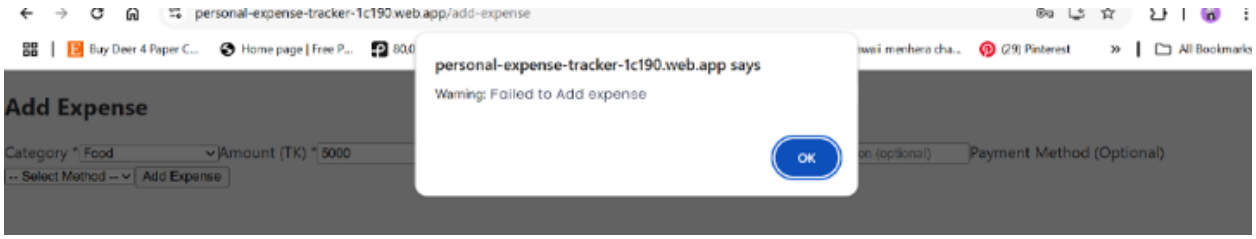
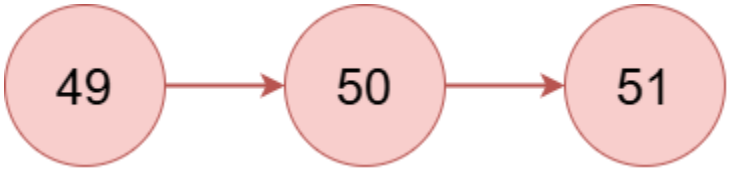


Fig 7: WTC7

handleDelete( )

```
49  const handleDelete = async (id) => {
50    await deleteDoc(doc(db, "expenses", id));
51  };
```

DD Graph (Decision-to-Decision Graph) of handleDelete( ):



Cyclomatic complexity of handleDelete( ):

E = Number of edges = 2

N = Number of nodes = 3

P = Number of connected components = 1

V(G) = E - N + 2P

= 2 – 3 + 2*1

= 1

So, there are 1 independent path.

Independent Paths:

- 49 → 50 → 51

Test Cases:

Test Case ID	Component	Class	Input	Expected Output'	Actual Output	Pass/Fail
WTC8	ExpenseManager	Expense	Select expense to delete	Deleted successfully	Deleted successfully	Pass

Table 5.2: white box testing02

Date	Amount	Category	Description	Payment Method	Actions
20 May 2025	TK 663	Entertainment	r grwgfcadgkcqmictlsaluoaebxk	cash	Edit Delete
19 May 2025	TK 1368	Entertainment	tqkmsihjhpbqhhtrjugrjk oviuu	bkash	Edit Delete
19 May 2025	TK 1776	Other	zvtnyniooqcwxubqtvs uziaumeal	bkash	Edit Delete

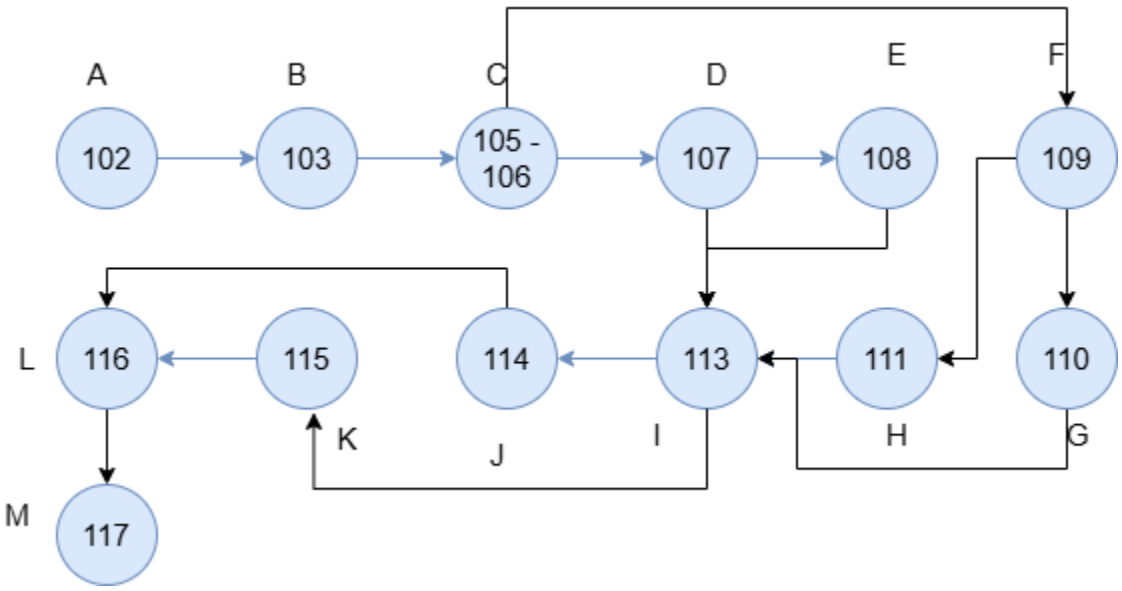
Date	Amount	Category	Description	Payment Method	Actions
20 May 2025	TK 663	Entertainment	r grwgfcadgkcqmictlsaluoaebxk	cash	Edit Delete
19 May 2025	TK 1776	Other	zvtnyniooqcwxubqtvs uziaumeal	bkash	Edit Delete

Fig 8: WTC8

Filter:

```
102    const filteredExpenses = expenses.filter((expense) => {
103      const expenseDate = new Date(expense.date);
104
105      const matchesDate =
106        filterType === "monthly"
107          ? expenseDate.getMonth() === selectedDate.getMonth() &&
108            expenseDate.getFullYear() === selectedDate.getFullYear()
109          : filterType === "yearly"
110          ? expenseDate.getFullYear() === selectedDate.getFullYear()
111          : true;
112
113      const matchesCategory = categoryFilter === "all" ?
114        true :
115        expense.category === categoryFilter;
116      return matchesDate && matchesCategory;
117    });
```

DD Graph (Decision-to-Decision Graph) of handleDelete( ):



Cyclomatic complexity of handleDelete( ):

E = Number of edges = 16

N = Number of nodes = 13

P = Number of connected components = 1

V(G) = E - N + 2P

= 16 – 3 + 2*1

= 6

So, there are 6 independent paths.

Independent Paths:

- A → B → C → D → E → I → J → L → M
- A → B → C → D → I → K → L → M
- A → B → C → F → H →I → K → L → M
- A → B → C → F → G →I → K → L → M
- A → B → C → D → E → I → k → L → M
- A → B → C → F → H →I → j → L → M

Test Cases:

Test Case ID	Component	Class	Input	Expected Output'	Actual Output	Pass/Fail
WTC9	ExpenseManager	Expense	category = "Food", type=”month	Show the list	Show the list	Pass

			ly”			
WTC10	ExpenseManager	Expense	category = "Food", type=”yearly”	Show the list	Show the list	Pass
WTC11	ExpenseManager	Expense	category = "Food", type=”all”	Show the list	Show the list	Pass
WTC12	ExpenseManager	Expense	category = "all", type=”all”	Show the list	Show the list	Pass
WTC13	ExpenseManager	Expense	category = "all", type=”monthly”	Show the list	Show the list	Pass
WTC14	ExpenseManager	Expense	category = "all", type=”yearly”	Show the list	Show the list	Pass

Table 5.1: white box testing03

View: Monthly ◀ May 2025 ▶

Filter by Category: Food

+ Add Expense

Date	Amount	Category	Description
11 May 2025	TK 678	Food	m ctgtntxtxowhkttvzdmr
07 May 2025	TK 147	Food	rgvvghaidu beadozxwpicv

Fig 9: WTC9

View: Yearly ◀ 2025 ▶

Filter by Category: Food

+ Add Expense

Date	Amount	Category	Description
11 May 2025	TK 678	Food	m ctgtntxtxow
07 May 2025	TK 147	Food	rgvvghaidu be
26 Apr 2025	TK 513	Food	icyyibockgghj
26 Apr 2025	TK 172	Food	viosbneuqbxa

Fig 10: WTC10

View: All

Filter by Category: Food

+ Add Expense

Date	Amount	Category	Description
11 May 2025	TK 678	Food	m ctgtntxtxowhkttvzdmr c
07 May 2025	TK 147	Food	rgvvghaidu beadozxwpicw
26 Apr 2025	TK 513	Food	icyyibockgghjrkncawfynkz

Fig 11: WTC11

View: All

Filter by Category: All

+ Add Expense

Date	Amount	Category	Description
20 May 2025	TK 663	Entertainment	r grwgfcadgkcq
19 May 2025	TK 1776	Other	zvtnyniooqcwxu
19 May 2025	TK 1086	Other	sxgwxvslsvbu ih

Fig 12: WTC12

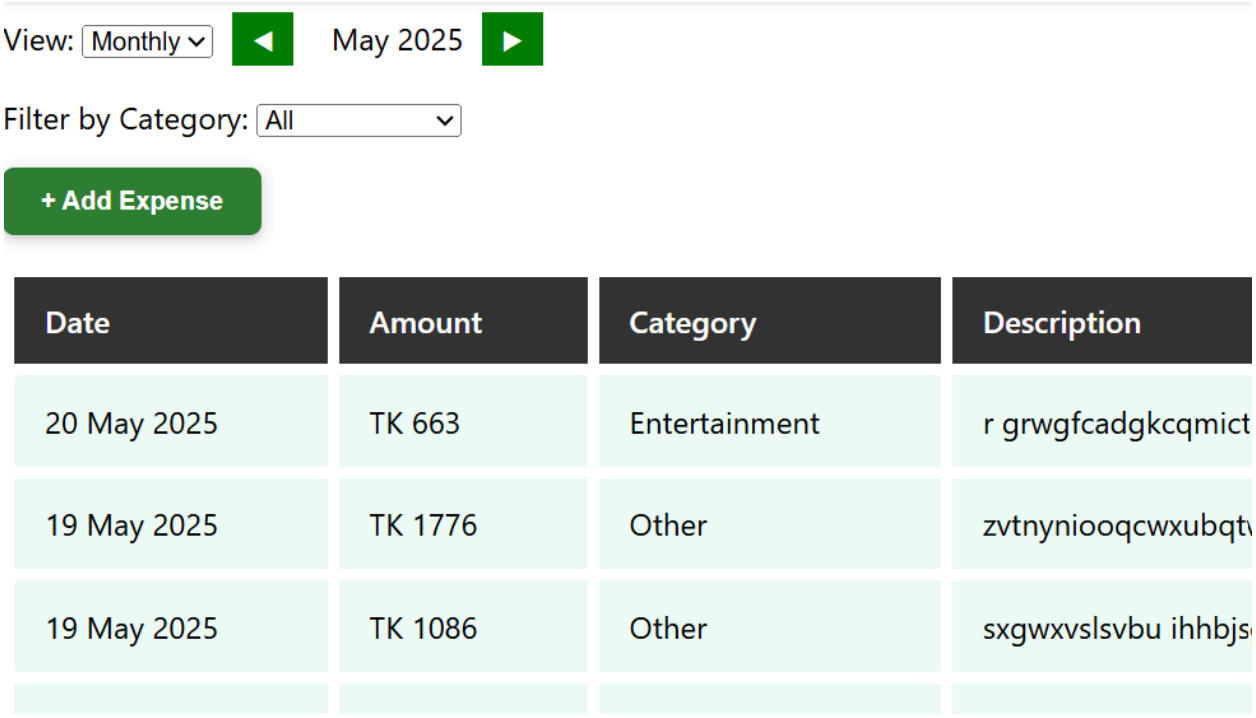


Fig 13: WTC13

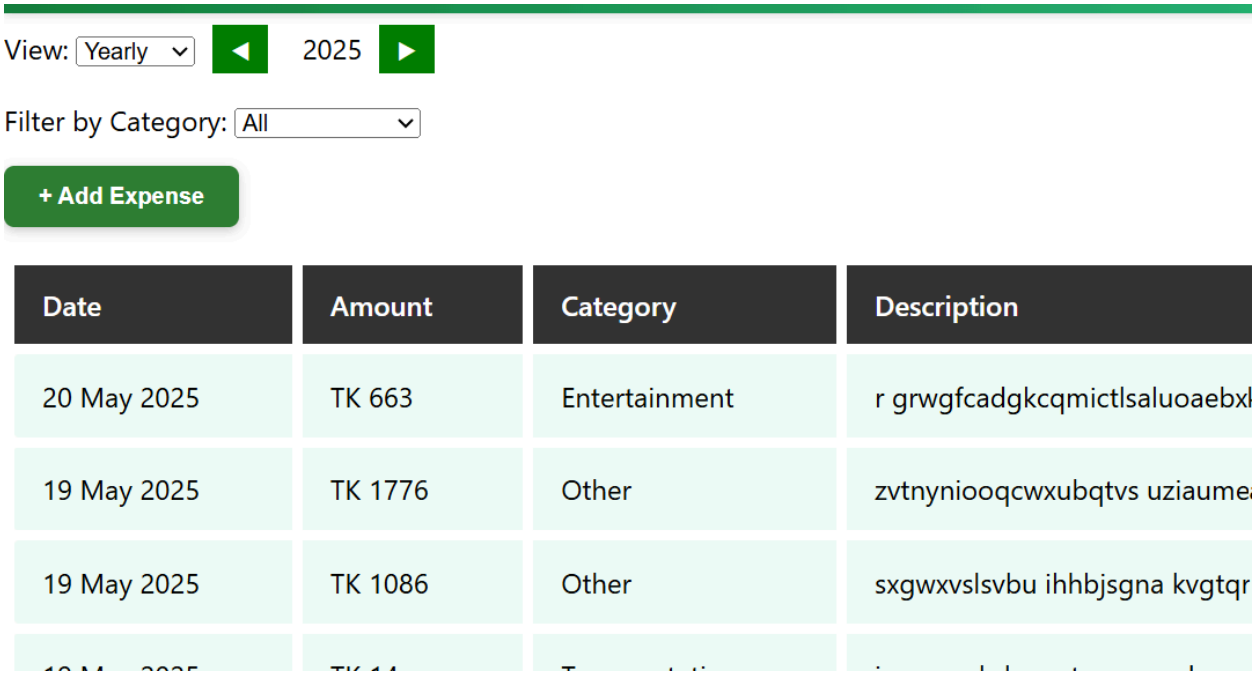


Fig 14: WTC14

For Budget Class:

handleSubmit( )

```
53  const handleSubmit = async (e) => {
54    e.preventDefault();
55    const parsedAmount = parseFloat(amount);
56    if (isNaN(parsedAmount) || parsedAmount <= 0) {
57      alert("Please enter a valid, positive amount.");
58      return;
59    }
60    if (
61      !budgetType ||
62      (budgetType === "monthly" && (!month || !year)) ||
63      (budgetType === "yearly" && !year)
64    ) {
65      alert("Please select valid budget type and date.");
```

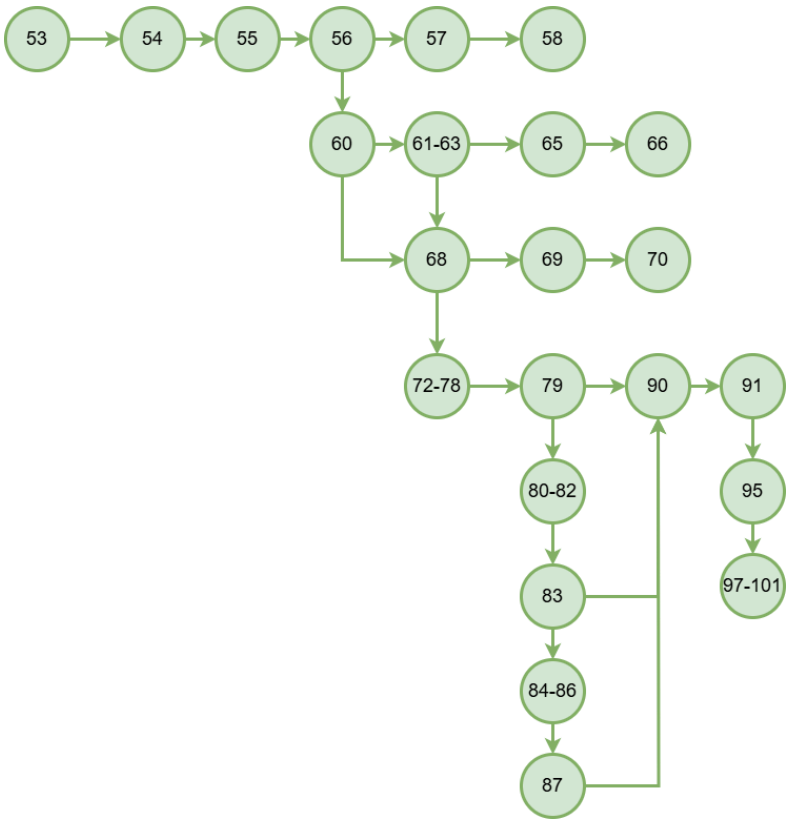
```
66      return;
67    }
68    if (parseInt(year) <= 1990) {
69      alert("Year must be greater than 1990.");
70      return;
71    }

72    const budgetData = {
73      userId: currentUser.uid,
74      type: budgetType,
75      amount: parsedAmount,
76      month: budgetType === "monthly" ? month : null,
77      year: parseInt(year),
```

```
78    };

79    if (budgetType === "yearly") {
80      const totalMonthly = budgets
81        .filter((b) => b.type === "monthly" && b.year === parseInt(year))
82        .reduce((sum, b) => sum + Number(b.amount), 0);

83      if (totalMonthly > parsedAmount) {
84        const confirm = window.confirm(
85          `You've already set monthly budgets totaling TK ${totalMonthly} for $
86        );
```

```
87        if (!confirm) return;
88      }
89    }

90    try {
91      if (editingId) {
92        await updateDoc(doc(db, "budgets", editingId), budgetData);
93        setEditingId(null);
94      } else {
95        await addDoc(collection(db, "budgets"), budgetData);
96      }
```

```
97        fetchBudgets(currentUser.uid);
98        setAmount("");
99        setMonth("");
100       setYear("");
101       setBudgetType("");
102     } catch (error) {
103       console.error("Error saving budget:", error);
104       alert("Failed to save budget.");
105     }
106 };
```

DD Graph (Decision-to-Decision Graph) of handleSubmit( ):



Cyclomatic complexity of handleDelete( ):

E = Number of edges = 26

N = Number of nodes = 23

P = Number of connected components = 1

V(G) = E - N + 2P

= 26 – 23 + 2*1

= 8

So, there are 8 independent paths.

Independent Paths:

- 53 → 54 → 55 → 56 → 57 → 58
- 53 → 54 → 55 → 56(F) → 60 → 61/62/63(T) → 65 → 66
- 53 → 54 → 55 → 56(F) → 60 → 61/62/63(F) → 68(T) → 69 →70
- 53 → 54 → 55 → 56(F) → 60(F) → 68(F) → 72–78 → 79(F) → 90 → 91(F) → 95 → 97–101
- 53 → 54 → 55 → 56(F) → 60(F) → 68(F) → 72–78 → 79(T) → 80–82 → 83(F) → 90 → 91(F) → 95 → 97–101
- 53 → 54 → 55 → 56(F) → 60(F) → 68(F) → 72–78 → 79(T) → 80–82 → 83(T) → 84–86 → 87(F) → 90 → 91(F) → 95 → 97–101
- 53 → 54 → 55 → 56(F) → 60(F) → 68(F) → 72–78 → 79(T) → 80–82 → 83(T) → 84–86 → 87(T)
- 53 → 54 → 55 → 56(F) → 60(F) → 68(F) → 72–78 → [79→87 optional] → 90 → 91(T) → 92 → 93 → 97–101

Test Cases:

Test Case ID	Component	Class	Input	Expected Output'	Actual Output	Pass/Fail
WTC15	Budget Manager	Budget	amount = "" or -100, budgetType = monthly, month = January, year = 2024	Alert: "Please enter a valid, positive amount."	Alert: "Please enter a valid, positive amount."	Pass
WTC16	Budget Manager	Budget	amount = 1000, budgetType = "" or monthly with no month/year	Alert: "Please select valid budget type and date."	Alert: "Please select valid budget type and date."	Pass
WTC17	Budget Manager	Budget	amount = 1000, budgetType = monthly, month = January, year = 1990	Alert: "Year must be greater than 1990."	Alert: "Year must be greater than 1990."	Pass
WTC18	Budget Manager	Budget	amount = 1000, budgetType = monthly, month = January, year =	Successfully Added	Successfully Added	Pass

			2024			
WTC19	Budget Manager	Budget	Select edit cancel amount = 1000, budgetType = monthly, month = January, year = 2024	Nothing changed	Nothing changed	Pass
WTC20	Budget Manager	Budget	amount = 500, budgetType = yearly, year = 2024, monthly total = 10000, confirm = true	Successfully Added	Successfully Added	Pass
WTC21	Budget Manager	Budget	amount = 500, budgetType = yearly, year = 2024, monthly total = 10000, confirm = false	Not added	Not added	Pass
WTC22	Budget Manager	Budget	Select to edit ,amount = 1500, budgetType = monthly, month = March, year = 2025	Successfully Edited	Successfully Edited	Pass

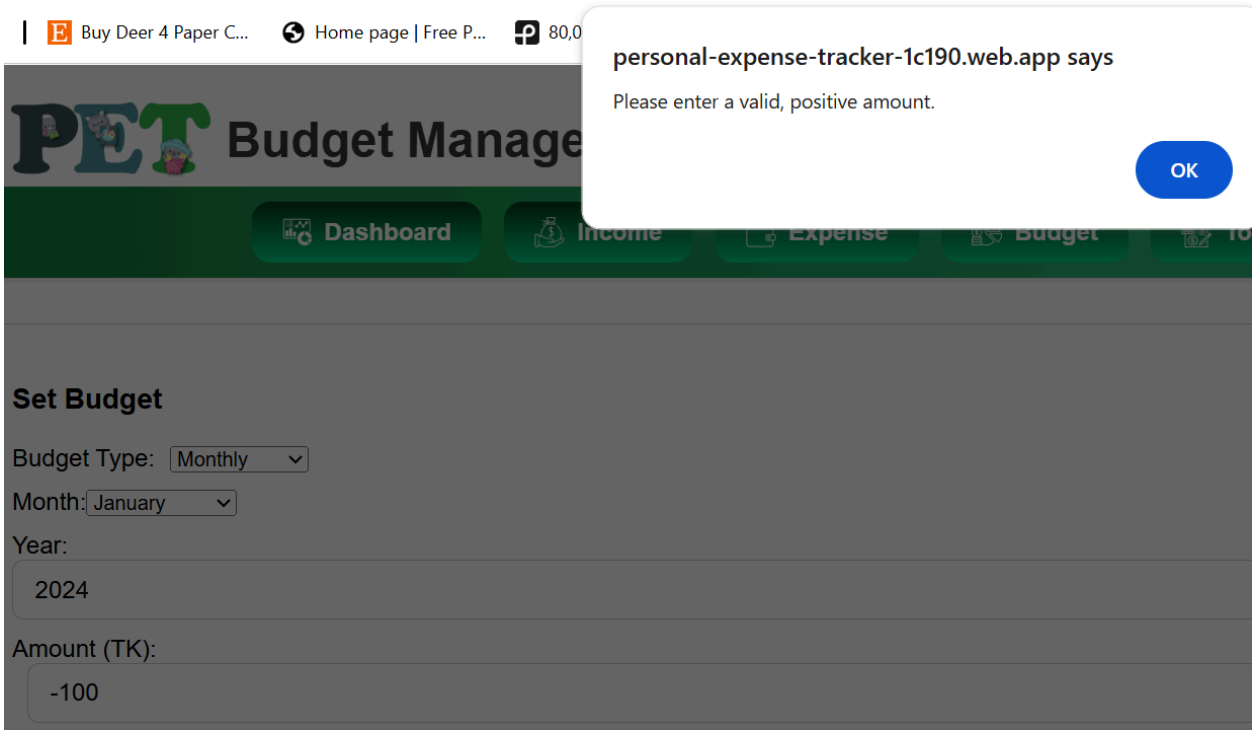


Fig 13: WTC15

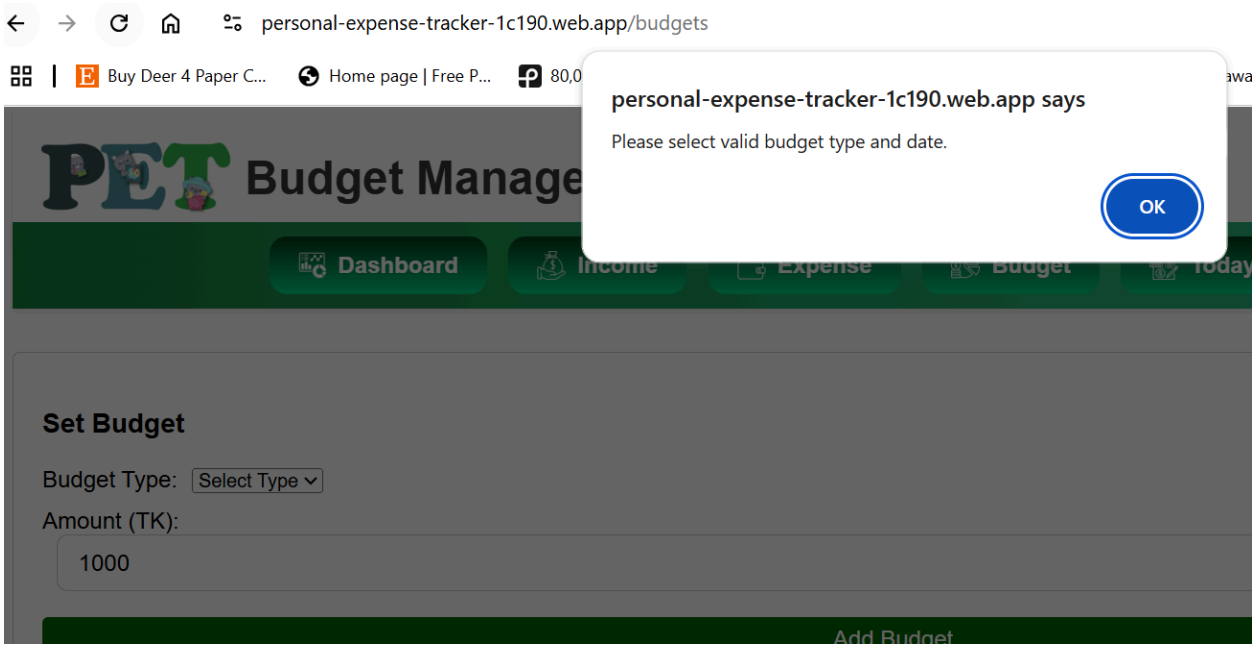


Fig 13: WTC16

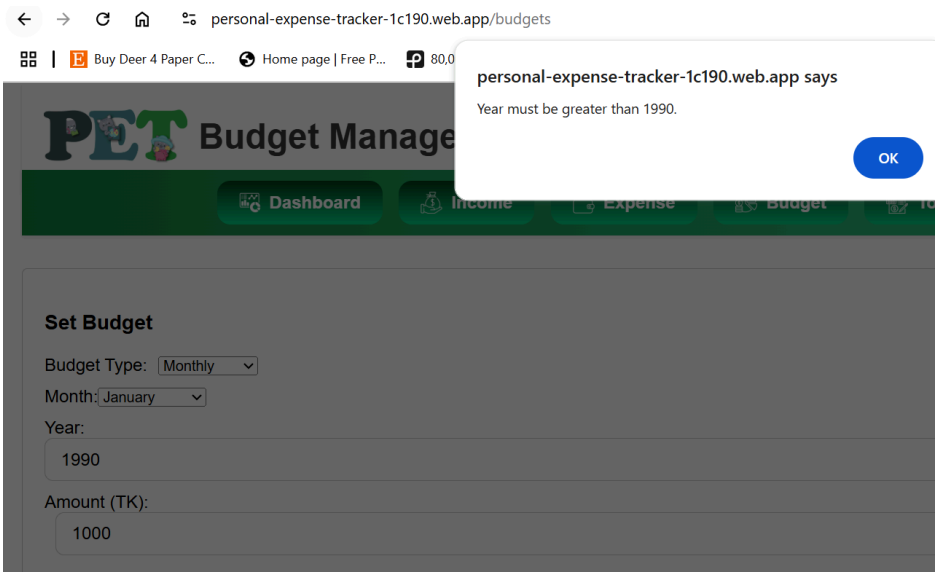


Fig 13: WTC17

< 2024 > Filter Type: All Filter Month: All

Type	Amount (TK)	Month	Year	Actions
monthly	TK 1000	January	2024	Edit Delete

Fig 13: WTC18

< 2024 > Filter Type: All Filter Month: All

Type	Amount (TK)	Month	Year	Actions
monthly	TK 1000	January	2024	Edit Delete

Fig 13: WTC19

< 2024 > Filter Type: All Filter Month: All

Type	Amount (TK)	Month	Year	Actions
yearly	TK 500	-	2024	Edit Delete
monthly	TK 1000	January	2024	Edit Delete

Fig 13: WTC20

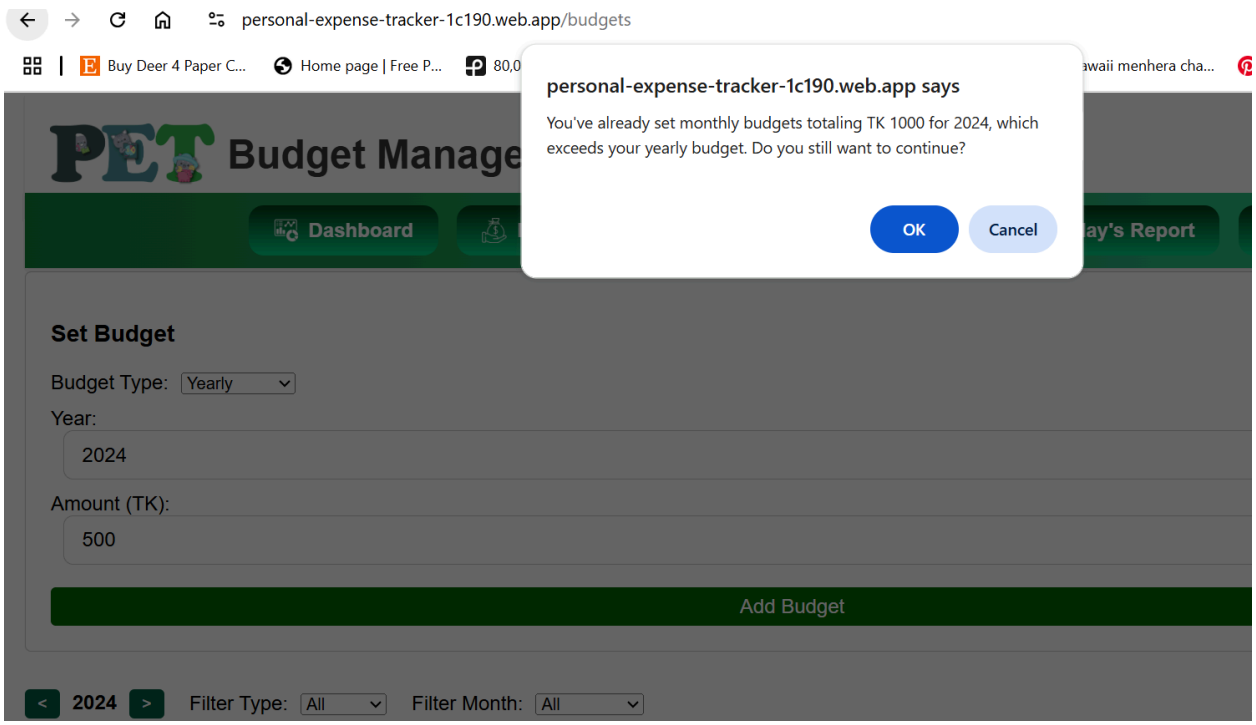


Fig 13: WTC21

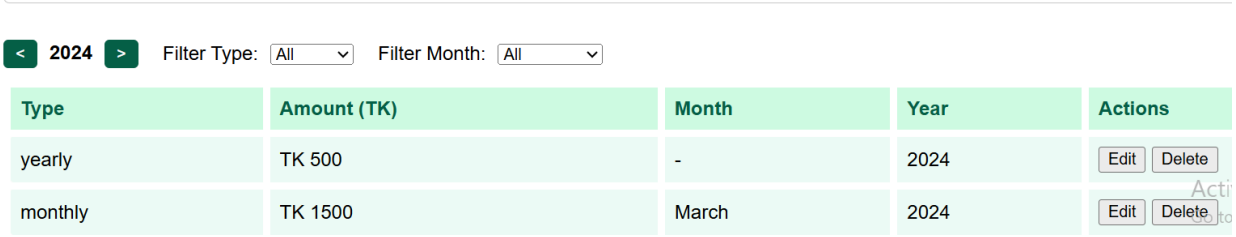


Fig 13: WTC22

5.2: Black Box Testing:

Black Box Testing is a software testing method in which the internal structure, design, or implementation of the item being tested is not known to the tester. The tester focuses solely on input and output, that is, verifying the functionality of the software by testing whether it behaves correctly for a given set of inputs. Below two of our core classes Expense and Budget's black box testing was implemented.

For Budget class:

Yearly Budget: User Input Field for BVA: Amount and Year

Total monthly budget is set 320000 TK

- ➢ Yearly Budget must be a valid positive amount
- ➢ Yearly Budget should not be less than Total monthly
- ➢ Year must be greater than 1990

So, A = [0.1, 999999] ; B= [1991, 4000]

minA = 0.1 minB = 1991

maxA = 999999 maxB = 4000

minA+ =1 minB+ =1992

maxA- = 999998 maxB- = 3999

nomA = 500000 nomB = 2025

Test Case ID	Component	Class	Input	Expected Output'	Actual Output	Pass/Fail
BTC1	BudgetManager	Budget	0.1, 2025	Budget added successfully to the list, with warning	Budget added successfully to the list, with warning	Pass
BTC2	BudgetManager	Budget	999999, 2025	Budget added	Budget added	Pass

				successfully to the list, no warning	successfully to the list, no warning	
BTC3	BudgetManager	Budget	1, 2025	Budget added successfully to the list, with warning	Budget added successfully to the list, with warning	Pass
BTC4	BudgetManager	Budget	999998, 2025	Budget added successfully to the list, no warning	Budget added successfully to the list, no warning	Pass
BTC5	BudgetManager	Budget	500000, 2025	Budget added successfully to the list, no warning	Budget added successfully to the list, no warning	Pass
BTC6	BudgetManager	Budget	500000, 1991	Budget added successfully to the	Budget added successfully to the	Pass

				list, no warning	list, no warning	
BTC7	BudgetManager	Budget	500000, 4000	Budget added successfully to the list, no warning	Budget added successfully to the list, no warning	Pass
BTC8	BudgetManager	Budget	500000, 1992	Budget added successfully to the list, no warning	Budget added successfully to the list, no warning	Pass
BTC9	BudgetManager	Budget	500000, 3999	Budget added successfully to the list, no warning	Budget added successfully to the list, no warning	Pass

Table 5.2: black box testing

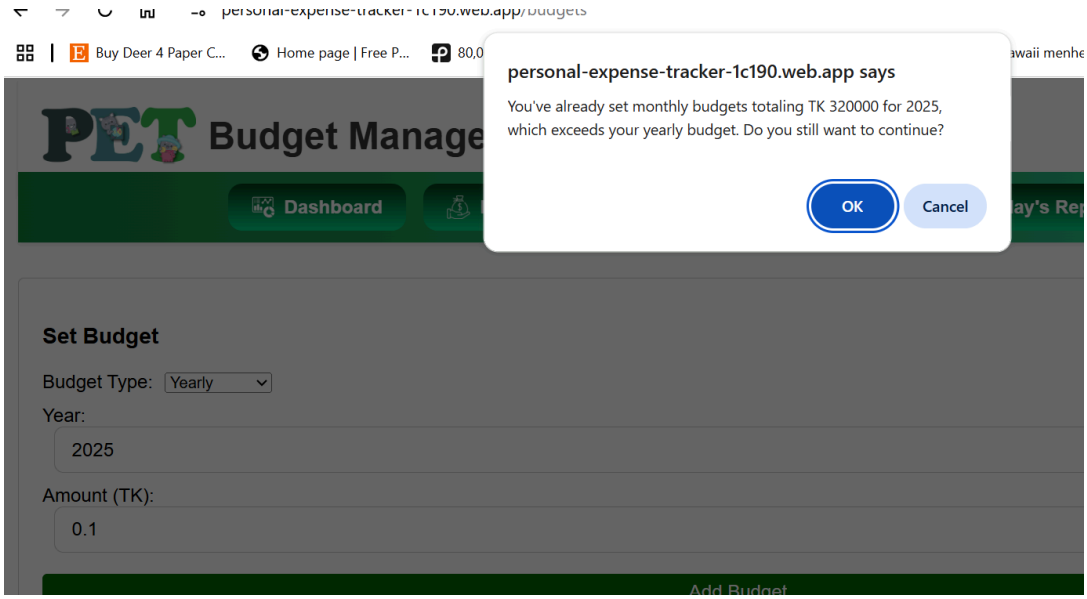


Fig 11: BTC1

Type	Amount (TK)	Month	Year	Actions
yearly	TK 999999	-	2025	Edit Delete

Fig 11: BTC2

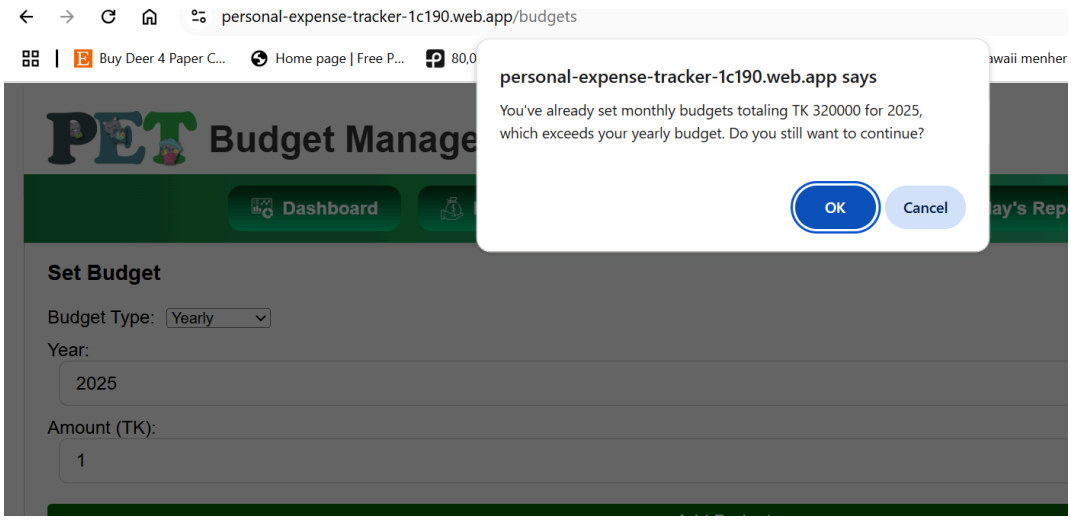
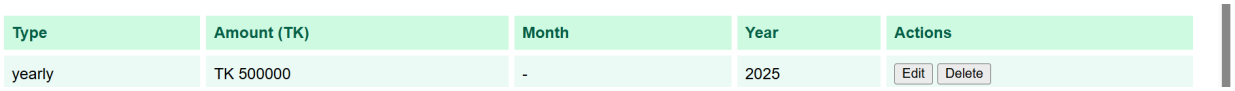


Fig 11: BTC3

Type	Amount (TK)	Month	Year	Actions
yearly	TK 999998	-	2025	Edit Delete

Fig 11: BTC4



Type	Amount (TK)	Month	Year	Actions
yearly	TK 500000	-	2025	Edit Delete

Fig 11: BTC5

Type	Amount (TK)	Month	Year	Actions
yearly	TK 500000	-	1991	Edit Delete

Fig 11: BTC6

Type	Amount (TK)	Month	Year	Actions
yearly	TK 500000	-	4000	Edit Delete

Fig 11: BTC7

Type	Amount (TK)	Month	Year	Actions
yearly	TK 500000	-	1992	Edit Delete

Fig 11: BTC8

Type	Amount (TK)	Month	Year	Actions
yearly	TK 500000	-	3999	Edit Delete

Fig 11: BTC9

For Expense Class:

AddExpense : User Input Field for BVA: Amount

- ➢ Expense must be a valid positive amount , So min = 0.01

Test Case ID	Component	Class	Input	Expected Output'	Actual Output	Pass/Fail

BTC10	ExpenseManager	Expense	0.01	Expense added successfully, no warning	Expense added successfully, no warning	Pass
BTC11	ExpenseManager	Expense	999999999	Expense added successfully, no warning	Expense added successfully, no warning	Pass

Table 5.2: black box testing02

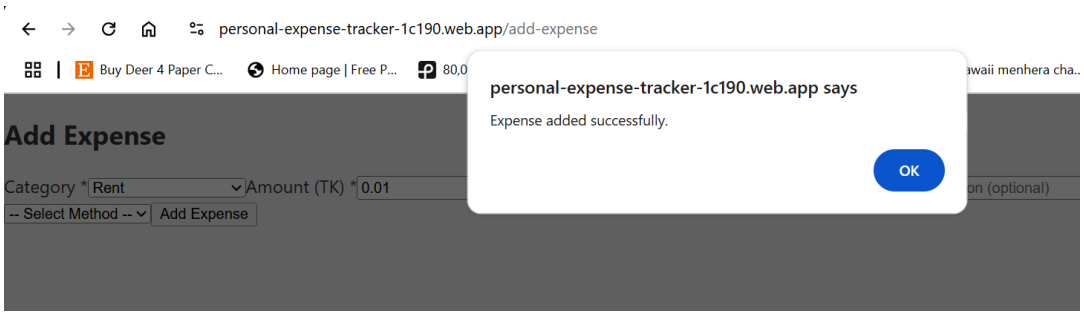


Fig 8: BTC10

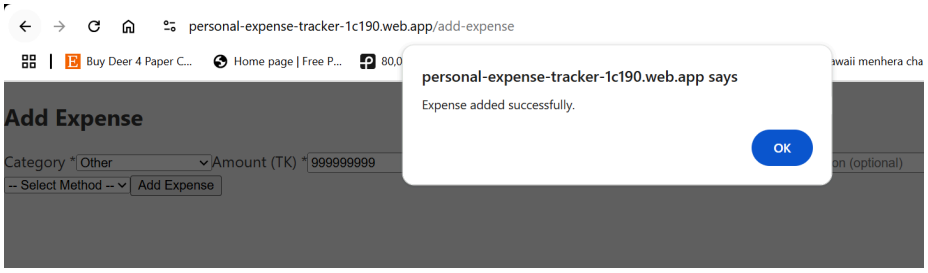


Fig 11: BTC11

5.3: Bug Detection and Solution

No bugs were detected while testing the above cases.

Chapter 6: Deployment

6.1 Deployment Platform

We used Firebase as a deployment tool since it has its own database and authentication system which were convenient for our website.

6. 2 Deployment Process

We first installed firebase configuration to our local git repository and then connected them. After that we collected it with gitHub so that when we push any code to the gitHub it automatically run build function and instantly updated the website.

6.3 Website URL

https://personal-expense-tracker-1c190.web.app/dashboard

Admin Email: adminemail@gmail.com

Password: @dmin123

User Email : zannat1212@gmail.com

Password: p@ssword1

Chapter 7: Conclusion

7.1 Learnings:

1.Understanding of Financial Management Concepts.

2.Scalability and Feature Limitation Awareness.

3.Secure Authentication Implementation.

4.AI and Predictive Analysis.

5.Limitations of Non-Scalable Features.

6.System Performance and Scalability Awareness.

7.2 Limitations:

1.Limited Integration with Financial Institutions

2.Scalability Constraints for AI Features

3.Some advanced features like receipt scanning.

4.No integration with external apps or bank accounts.

5.We cannot provide smart recommendations based on spending habits using AI.

6.We cannot add a family collaboration system.

7.3 Future Plan

1. We are planning to link with Banks APIs to ensure real time secure transactions.
2. Sync with google pay , apple wallet , PayPal ,bkash , upay, nagad and other mobile financing systems so that the user has a smooth user experience.
3. Support to add multiple bank accounts.
4. Auto fill expense forms from images use google lens .
5. Provide advance recommendations for future expenses .
6. Improve UI/UX with better visualizations.

7. Add a voice assistant or AI chatbot for a more user friendly experience.
8. Offline using and sync data automatically when reconnected.

Appendix

Google Form Responses

1.What is your name?

82 responses

Afnan

Sadika Afrin Ema

Md. Mariful Islam Zabir

Rafid

Sumaiya Akter Espa

My name is Muziba Mahmuda Mithy

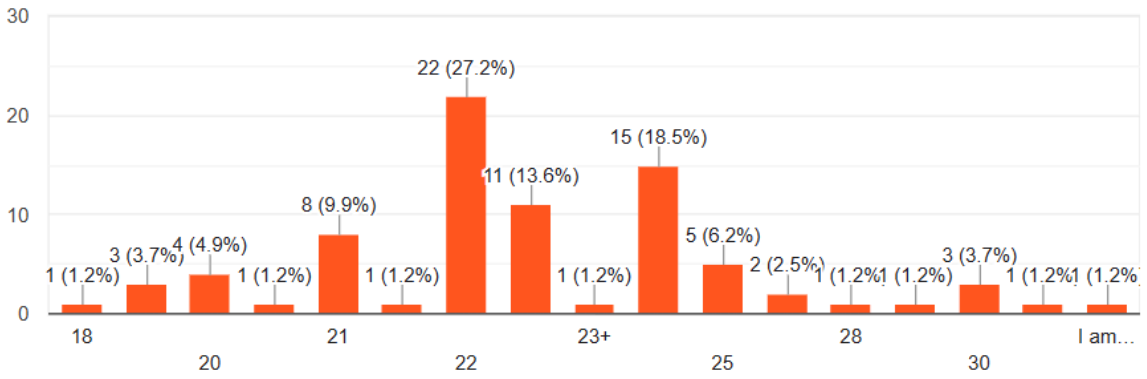
Mohammad Refat Shahriear

Fairuj Saima

2.How old are you?

Copy chart

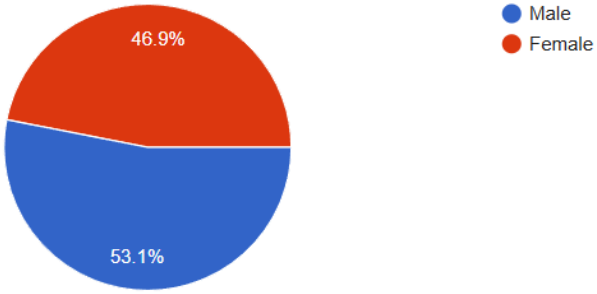
81 responses



3.What is your gender?

Copy chart

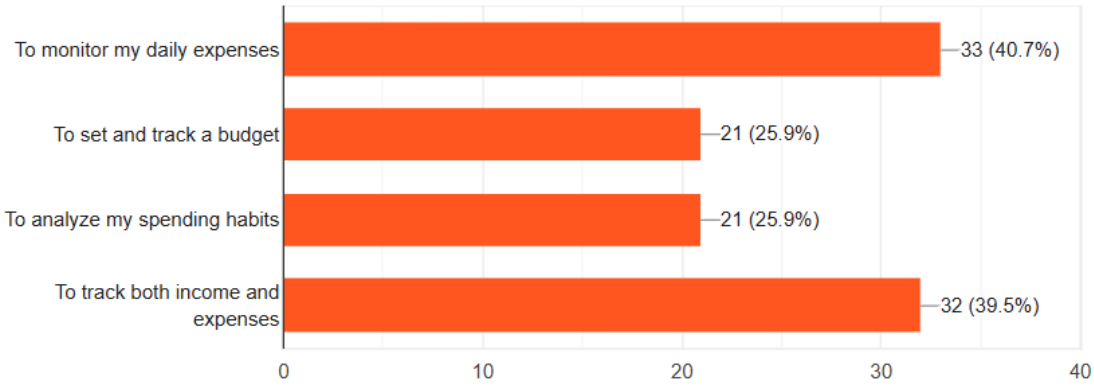
81 responses



4.What is the primary purpose for using a Personal Expense Tracker?

Copy chart

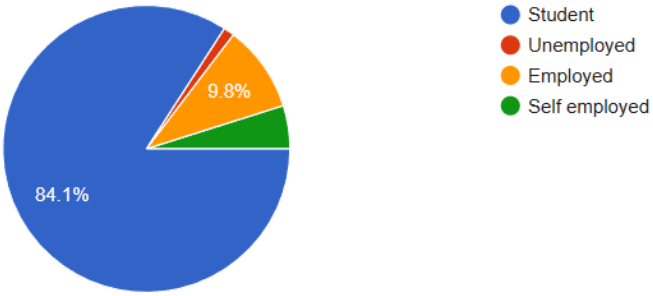
81 responses



5. What is your job status?

Copy chart

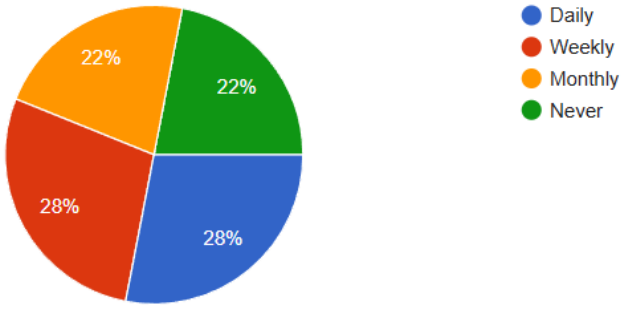
82 responses



6.How often do you track your expenses?

Copy chart

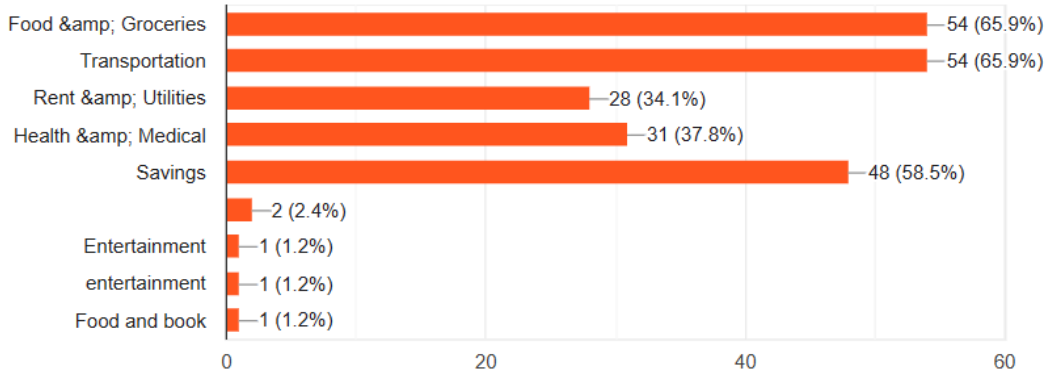
82 responses



7.How should expenses be categorized? (Select all that apply)

Copy chart

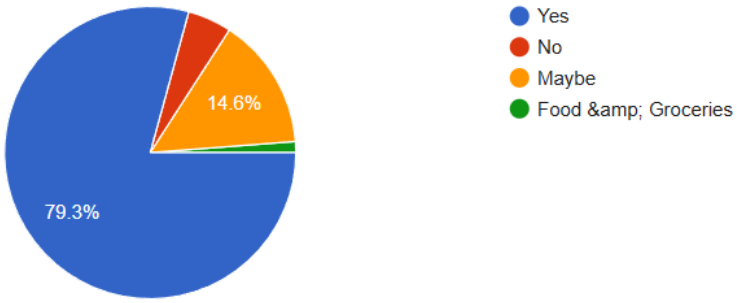
82 responses



8. Would you like to set a monthly budget and receive alerts when exceeding it?

Copy chart

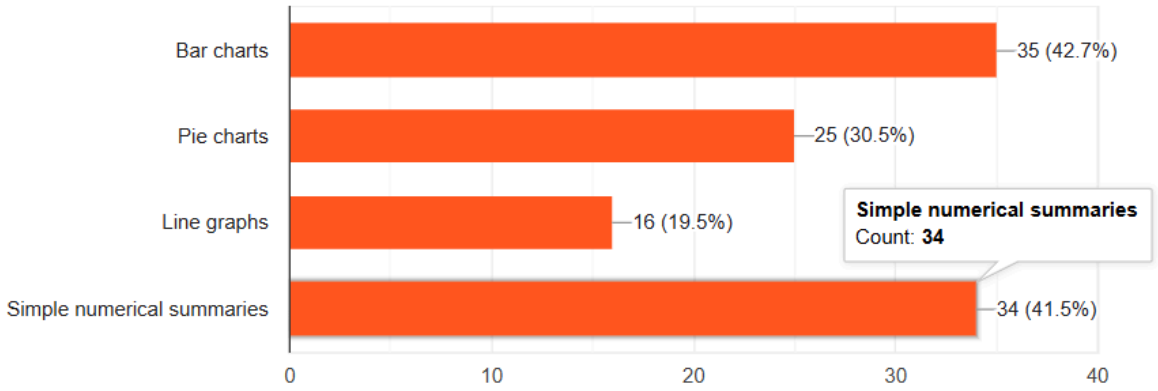
82 responses



9. What visual representation of data would you prefer? (Select all that apply)

Copy chart

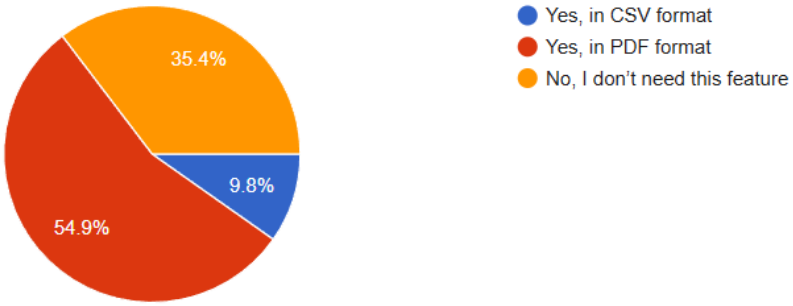
82 responses



10.Would you like to export your financial data?

Copy chart

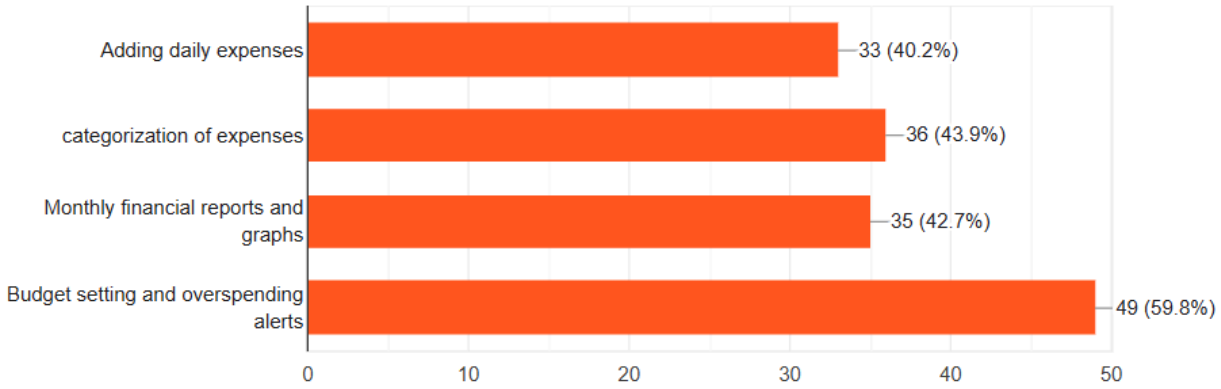
82 responses



11. Which features would you find most useful in an expense tracker? (Select all that apply)

Copy chart

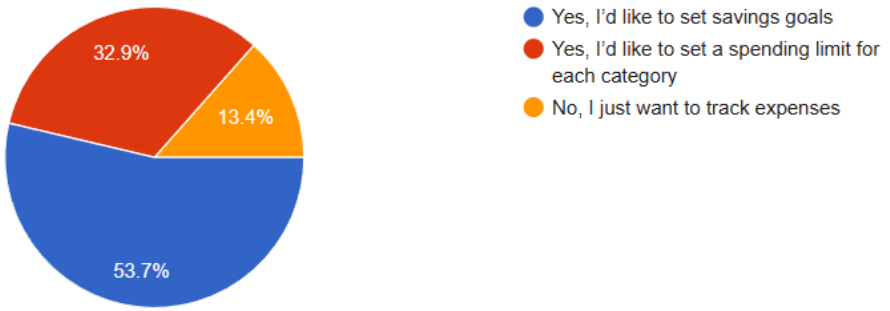
82 responses



12. Do you want to set financial goals in the app?

Copy chart

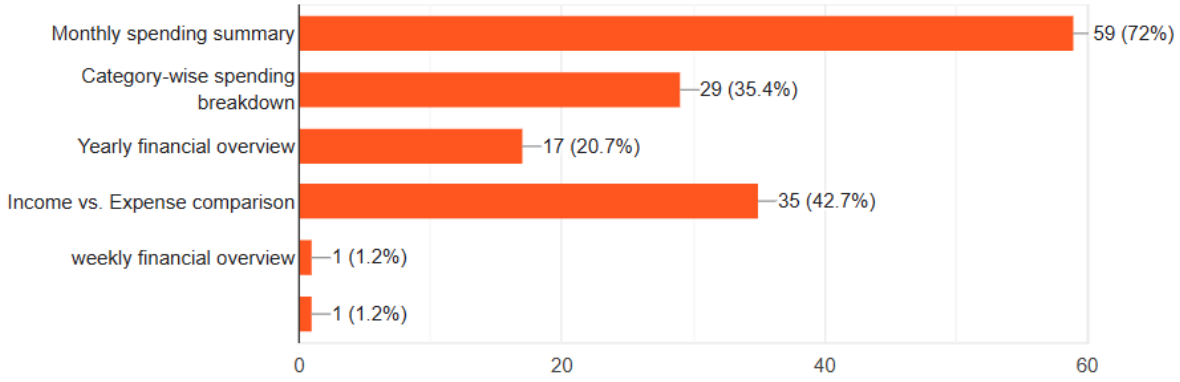
82 responses



13. What types of financial reports would be useful for you? (Select all that apply)

Copy chart

82 responses



14. What other features would you like to see in this expense tracker? (Open-ended question)

82 responses

Nothing

None

Nothing

NA

N/A

..

.

helps you manage your expenses by allowing you to record and track your spending habits with ease.

Invoice and Receipt Management

Expense tracker

Would like to learn budgeting.

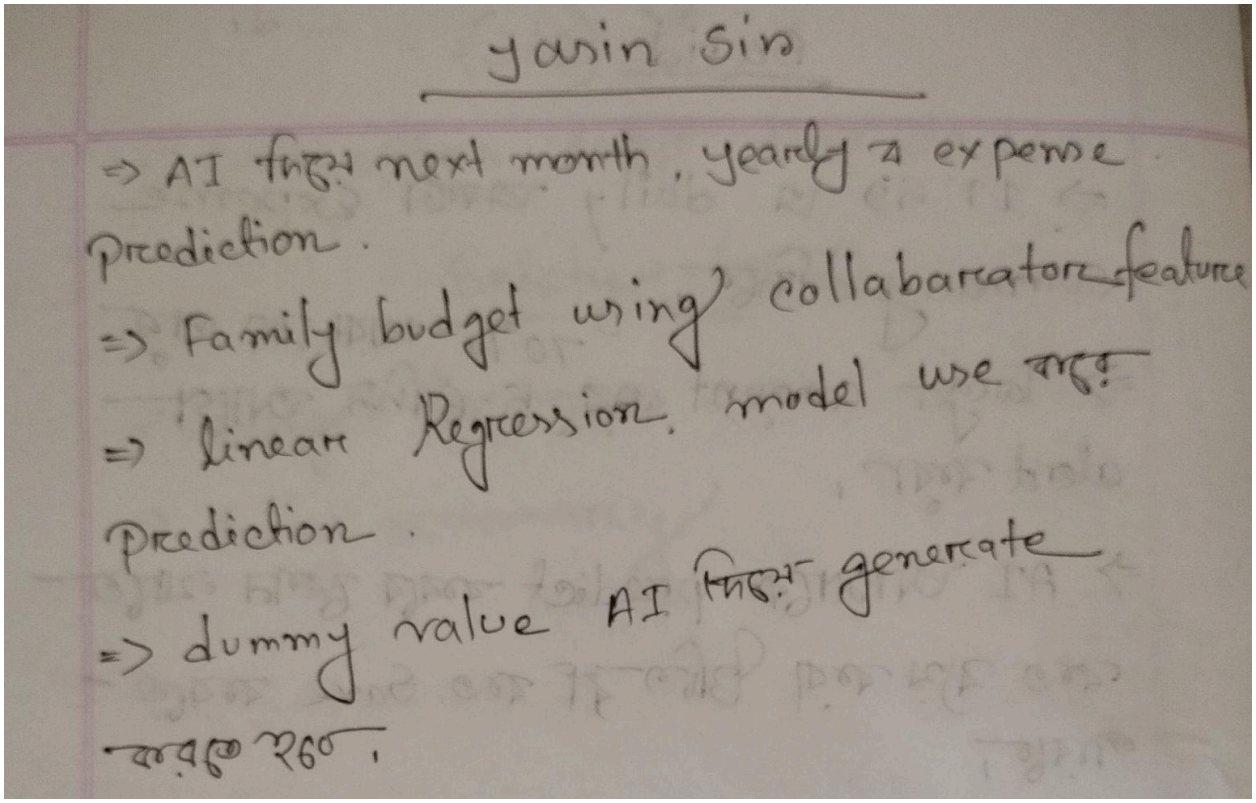
Bill Reminders & Notifications – Get alerts for upcoming due dates.

Expense & revenue

It's good enough.

Savings

Interview Drafts:



Sajid (CSE)

⇒ 11:59 এ daily একটা expense summary দিবে.

⇒ Target amount এর 70% খরচ হলে পর 5-6 দিন আগে alert করবে.

⇒ AI আমাকে predict করবে কোন খাতে কত ব্যয় করা উচিত বা কত save করতে পারব.

Niloy (civil engineering)

⇒ Date & বার mention করতে হবে.

⇒ weekly summary এর graph.
(skewed graph, leaf graph, bar graph, pie chart, line graph)

⇒ Mean, median, mode expense এর.

⇒ AI এর আমাকে daily একটা expense predict করে morning, noon, night.

=> Edit option থাকবে time or date

<u>Sristy</u> (CSE)

=> budget fix করতে হবে।

=> daily summary.

<u>~~Sreya~~ Ankon saha</u> (CSE)

=> Amount monthly divide করে daily expense দেখাবে।

=> daily expense ১৫০০০ এর বেশি খরচ হলে তাহলে alarm দিবে।

=>

Bithi (Genetic Engineering)

=> Different catogories (Food, Transport, donation etc).

=>

=> daily ~~ro~~ budget remain.

=>