SOFTWARE ENGINEERING I

Spring 2024

BUDGET BUDDY

-Assignment #3-

Team 5:
Andrew McNeill
Krishna Sruthi Velaga
Akash Reddy Karri
Ahmed Hamza
Tobechukwu Ejike

April 15th, 2024

TABLE OF CONTENTS

SECTION 1
PLANNING & SCHEDULING

Assignee Name/ Email (@students.towson.edu)	Task	Duration (Hours)	Dependency	Due Date	Evaluation
Andrew McNeill / amcnei8	Initial Meeting	30mins		03/24/2024	100%
	Task 3	30 mins	Project A2	04/01/2024	
	Implementation	5 hours		04/01/2024	
	Final changes to report	30mins		03/31/2024	
Krishna Sruthi Velaga/ kvelaga1	Initial Meeting	30mins		03/24/2024	100%
	Write Test Cases	1.5hr	Use Cases	04/01/2024	
	Format Report	2hrs		04/01/2024	
	Final changes to report	30mins		03/31/2024	
Akash Reddy Karri/ akarri2	Initial Meeting	30mins		03/24/2024	100%
	Write Test Cases	1.5hr	Use Cases	04/01/2024	
	Format Report	1.5hrs		04/01/2024	
Ahmed Hamza/ ahamza ***[Coordinator]***	Initial Meeting	30mins		03/24/2024	100%
	System Modeling	3 hours	Use Cases	03/29/2024	
	Final changes to report	30mins		03/31/2024	
Tobechukwu Ejike/ tejike1	Initial Meeting	30mins		03/24/2024	100%
	System Modeling	3 hours	Use Cases	03/29/2024	

	Implementation Final changes to report	5 hours 30mins	Project A2	03/31/2024 03/31/2024	

SECTION 2
PROBLEM STATEMENT

The product on a high level – Budget Buddy is a web-based personal finance tool which can help users to manage their spending and budgets. It features alerts for bills and budget limits, visual spending reports, transaction tracking, and secure user authentication. It simplifies financial management and promotes informed decision-making for better financial health.

Whom is it for? - It is for individuals looking to manage their personal finances, track their spending and stay within their budget. It is suitable for anyone, from students to families, who wish to understand and manage their finances better.

What problem does it solve?
Budget Buddy solves following problems:

- *Overspending:* It helps users stay within their budget limits by tracking spending and alerting them when they are close to exceeding their budget.
- *Untracked Expenses:* The application allows users to log all their transactions, providing a clear overview of where their money is going, which can help in identifying unnecessary expenses.
- *Missed Payments:* It alerts users to upcoming bills, reducing the risk of late payments and associated fees.
- *Complex Financial Tracking:* Budget Buddy simplifies the process of monitoring finances by providing visual reports, making it easier for users to understand their spending habits and financial trends.
- *Insecure Financial Data:* The application ensures that user data is secure through proper authentication measures, protecting sensitive financial information.

What alternatives are available?
Mint, YNAB, PocketGuard

- Why is this project compelling and worth developing?
The Budget Buddy project is compelling due to its innovative approach to budget planning, addressing a crucial but often neglected need for financial literacy. It introduces a feature that discerns spending patterns on a daily, monthly, quarterly, and yearly basis, enabling users to gain deeper financial insights. This analysis and personalized feedback empower individuals to effectively comprehend and control their finances. Financials being an important thing that school doesn't necessarily teach citizens, this application can help society.

- Describe the top-level objectives, differentiators, target customers, and scope of your product.
Top-level Objectives:

1. Present users with totals showing where their money is going such as food, entertainment, gas, bills, etc.
2. Provide users with an interface that's intuitive such that spending habits can be presented in graphs and diagrams.
3. Provide graphs that show past spending habits and prediction trends for future spending habits.

Differentiators:

1. This application is different then other financial planning applications because we identify and analyze spending trends over different timeframes

2. We provide authentication to ensure user information is secure

Target Customers:
For individuals looking to manage their personal finances. It is suitable for anyone, from students to families, who wish to understand and manage their finances better.

Scope:
The budget planner project aims to develop a user-friendly software application for individuals and organizations to manage their finances efficiently. Key features include customizable budgeting tools for expenses and income, tracking financial goals, generating reports, and ensuring data security. Users can set budgets, track spending, and monitor progress towards savings goals. The application will offer a seamless user interface across various devices and platforms, integrating with existing financial tools if necessary. Rigorous testing and ongoing maintenance will ensure reliability and performance, with user training and support provided for optimal use. Ultimately, the budget planner project seeks to empower users with the tools and insights they need to make informed financial decisions and achieve their financial objectives.

- What are the competitors and what is novel in your approach?
Mint, YNAB(You need a Budget), PocketGuard, Personal Capital are alternatives available in the market.

Buddy Budget differentiates itself through its trend analysis capabilities, where it offers insights about the spending habits of the user on a monthly, annually basis. It also provides user-friendly design.

- Make it clear that the system can be built, making good use of the available resources and technology.
Our current plan involves using Python and Java for the backend, along with HTML/CSS for the frontend

- What is interesting about this project from a technical point of view?
From a technical point of view our project is interesting because it includes machine learning to create metrics of users spending habits for past and future dates. The application will work on general devices making it openly available to a large population of users.

SECTION 3

REQUIREMENTS

3.1 User Requirements

01	**Entering Expense Data**
Actors:	Registered User, Budget Buddy Database
Description:	The registered user will enter the details of an expense made, which includes the amount, category (e.g., food, bills, entertainment, grocery), and the date on which the expense is made. This data will be recorded in the user data database. The system will then update the user spending data and therefore will get reflected in relevant reports and graphs.
Alternate Path:	If the registered user enters invalid date (e.g., a non-numeric value for the amount), the system prompts an alert asking the user to correct the given information
Pre-Condition:	The registered user is logged in to the system and navigated to the home screen of the application.

02	**User Registration**
Actors:	New User, Budget Buddy Database
Description:	The new user will be able to register into the application. The user will be asked to enter information like username, email, mobile number, password and a few others. After completion, upon successful registration the user will be navigated into the home screen of the application.
Alternate Path:	If the new user enters an email which is already in use. The application will ask to use a different email address.
Pre-Condition:	The new user doesn’t have an account with budget buddy application

03	**User Sign In**
Actors:	Registered User, Budget Buddy Database
Description:	The registered user will enter username and password to log into the application. The system will check if the credentials are stored in the user data database. If the given credentials are valid, the user will be able to gain access to their account. If not, the user will not be given access and may be asked to re-enter the credentials.
Alternate Path:	If the user has forgotten their password, and clicks on the “forgot password” option to reset their password after confirming user identity.
Pre-Condition:	The registered user has registered account and has valid credentials

04	**View Financial Reports**
Actors:	Registered User, Budget Buddy Database
Description:	The registered user will select the date range i.e., this month, last quarter and click the toggle button to view financial reports covering the selected period. The application displays the reports based on the expenses made by the user.
Alternate Path:	If there is insufficient data for generating a report, the system prompts an alert to the registered user to continue entering the expenses made and set budget.
Pre-Condition:	The registered user has logged into the system and has sufficient transaction history for the system to generate reports

05	**Edit Expense and Budget Amount**
Actors:	Registered User, Budget Buddy Database
Description:	The registered user will select an existing expense or budget amount to edit. They can update the amount, category, or date before saving the changes. The application updates the information in the registered user data database and reflects this change in all reports.
Alternate Path:	If a registered user cancels the edit before saving, no changes are made to the database.
Pre-Condition:	The registered user is logged into the application and has an existing expense or budget amount.

06	**Delete Expense**
Actors:	Registered User, Budget Buddy Database
Description:	The user selects a transaction to remove, confirms the deletion, and the application then removes the transaction from the user data database
Alternate Path:	The user may decide to keep the transaction, so cancels the deletion process. Also, if application sometimes cannot find the transaction (it may have been already deleted or never existed), and alerts the user same
Pre-Condition:	The registered user is logged into the application and has an existing expense or budget amount.

07	**Alerts and Notifications**
Actors:	Registered User, Budget Buddy Database
Description:	The application monitors the user's spending against the budget limit. If the user is nearing the limit (or) exceeds the limit, the application will send an alert or notification to the user. This will help the user adjust their spending habits.
Alternate Path:	The registered user can see budget status in the home screen of the application
Pre-Condition:	The registered user is logged in, has set a budget limit and also has entered expense data.

08	**Set Budget Limits**
Actors:	Registered User, Budget Buddy Database
Description:	The registered user can set a budget limit for different spending categories (e.g., groceries, entertainment, bills) within the application. The application will track them and send alerts as they near (or) exceed them.
Alternate Path:	The registered user enters an unrealistic budget limit (e.g. zero dollars). The application prompt an alert asking the user to give realistic budget limit
Pre-Condition:	The registered user must be logged in.

3.2 System Requirements

[01] Use Case 01	**Entering Expense Data**
Introduction:	The user enters details of an expense, including the amount, category (e.g., food, bills, entertainment), and the date of the expense. Budget Buddy records this data in the Stored User Data Database. The system then updates the user's current spending totals and reflects this in the relevant reports and graphs.
Inputs:	Decimals Values, allowing to show dollar and cent values;
Requirements Description:	The user needs to enter only decimal values so Budget Buddy can gather data accurately to process the data to become information through charts and spending predictions.
Outputs:	The entered expense value is stored in the Budget Buddy database.

[02] Use Case 02	**User Registration**
Introduction:	The user enters their username and password to log into Budget Buddy. The system verifies the credentials against the Stored User Data Database. If the credentials are correct, the user gains access to their account. Otherwise, the system denies access and may offer the user the chance to reset their password or try again.
Inputs:	Multiple textboxes where the user enters 1. First Name 2. Last Name 3. Username 4. Email 5. Phone Number
Requirements Description:	The textboxes can only have alphanumeric characters and symbols entered for the Budget Buddy database system can successfully store the information.
Outputs:	The user's computer screen outputs valid login credentials to enter the website to authenticate to user the Budget Buddy services and features.

[03] Use Case 03	**User Sign In**
Introduction:	The user enters details of an expense, including the amount, category (e.g., food, bills, entertainment), and the date of the expense. Budget Buddy records this data in the Stored User Data Database. The system then updates the user's current spending totals and reflects this in the relevant reports and graphs.
Inputs:	Decimals Values, allowing to show dollar and cent values;
Requirements Description:	The user needs to enter only decimal values so Budget Buddy can gather data accurately to process the data to become information through charts and spending predictions.
Outputs:	The entered expense value is stored in the Budget Buddy database.

[04] Use Case 04	**View Financial Reports**
Introduction:	The user selects a date range (e.g., this month, last quarter) and clicks the toggle button to view financial reports covering that period. Budget Buddy generates and displays the report, which includes graphs and summaries of expenses, budget adherence, and other relevant financial metrics.
Inputs:	Users select from a dropdown menu between daily, weekly, monthly, quarterly, and annually.
Requirements Description:	The user must select from the dropdown menu with the time according to the time frame.
Outputs:	Financial reports, graphs, and a summary of expenses are generated for display.

[05] Use Case 05	Edit Expense and Budget Amount
Introduction:	The user selects an existing expense or budget amount to edit. They can then update the amount, category, or date before saving the changes. Budget Buddy updates the information in the database and reflects these changes in all affected reports and analyses.
Inputs:	Inputs: 1. Enter a selected date. 2. Find the budget amount of the selected existing expense or budget amount to edit. 3. Select the budget expense and edit the information that needs to be edited. 4. Click the "update" button to save the changes made.
Requirements Description:	The user must select from the dropdown menu with the time according to the time frame. Once found, the input fields of the expense that's to be edited need textboxes to store the new data.
Outputs:	The entered expense value is stored in the Budget Buddy database for the selected date range.

[06] Use Case 06	Delete Expense
Introduction:	The user identifies an existing expense or budget amount to delete from the machine learning algorithm in the Budget Buddy software. Budget Buddy updates the information in the database to reflect the delete expense/budget amount.
Inputs:	Inputs: 1. Enter a selected date. 2. Find the budget amount of the selected existing expense or budget amount to delete 3. Select the budget expense and edit the information that needs to be delete. 4. Click the "delete" button to save the changes made.
Requirements Description:	The user must have at least one entered expense budget data.
Outputs:	The selected expense/budget value is deleted in the Budget Buddy database.

[07] Use Case 07	**Alerts and Notifications**
Introduction:	Budget Buddy monitors the user's spending against their set budget. If the user is nearing, has exceeded, or is significantly under their budget, the system sends an alert or notification to the user. This helps users adjust their spending habits accordingly.
Inputs:	The user needs to enter only decimal values so Budget Buddy can gather data accurately to process the data to become information through charts and spending predictions.
Requirements Description:	The user must have at least one entered expense budget data.
Outputs:	The user is notified on their cell phone whether he is under-spending, overspending, credit limit reached.

[08] Use Case 08	**Set Budget Limits**
Introduction:	The registered user can set a budget limit for different spending categories (e.g., groceries, entertainment, bills) within the application. The application will track them and send alerts as they near (or) exceed them.
Inputs:	Inputs 1. Select the spending category of interest. 2. Users must enter a decimal value in the textbox. 3. Click "save" to have the budget implemented and saved in the Budget Buddy database system.
Requirements Description:	The user must enter a decimal value.
Outputs:	The database system stores the budget and on the application within that category the budget will be shown to the user.

SECTION 4

SYSTEM MODELING

4.1 Class Diagram

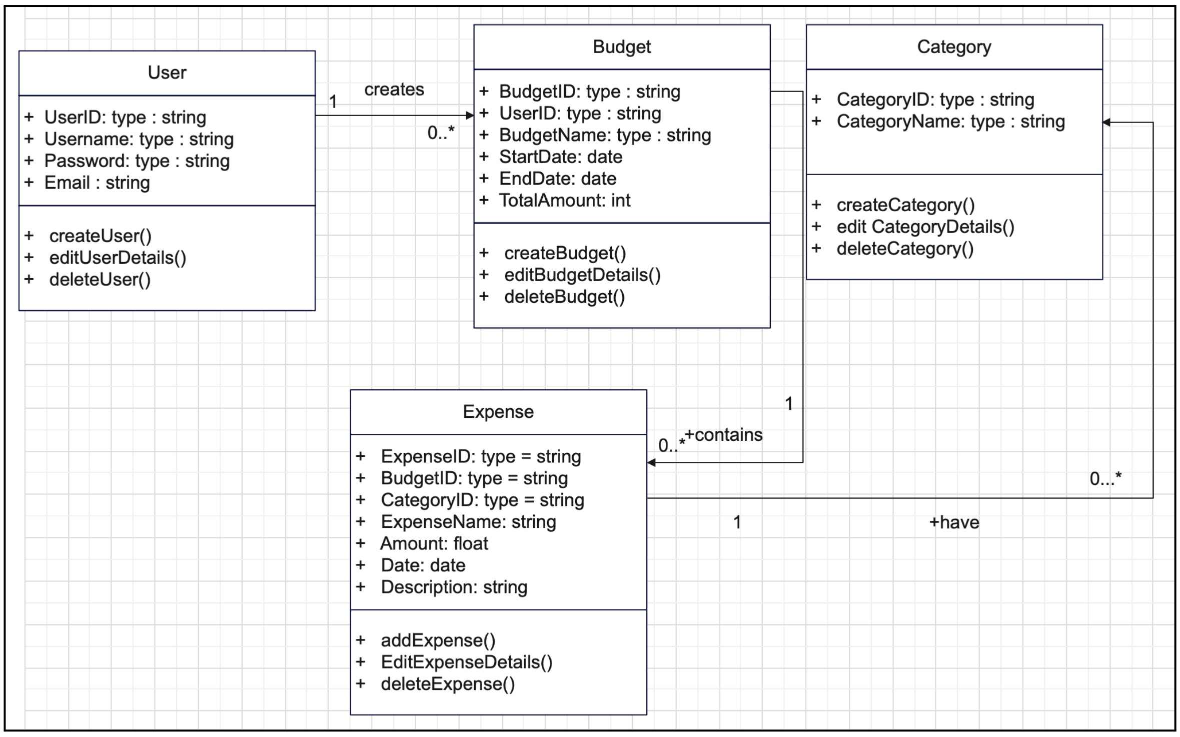


Fig 1: The class diagram represents the visual aspects of the budget buddy's system structure

1. *User:* Represents the individuals who use the budget planner.
   Attributes: UserID, Username, Password, Email.
2. *Budget:* Represents the budget plans created by users.
   Attributes: BudgetID, UserID, BudgetName, StartDate, EndDate, TotalAmount.
3. *Category:* Represents the categories or types of expenses.
   Attributes: CategoryID, CategoryName.
4. *Expense:* Represents individual expenses within a budget.
   Attributes: ExpenseID, BudgetID, CategoryID, ExpenseName, Amount, Date, Description.

Defining the associations between the objects

- User has a one-to-many association with Budget (one user can have multiple budgets).
- Budget has a one-to-many association with Expense (one budget can have multiple expenses).
- Category is associated with Expense in a many-to-one relationship (multiple expenses can belong to one category)

Defining the multiplicity of these associations between the objects

- User to Budget: 1 to many (one user can have many budgets).
- Budget to Expense: 1 to many (one budget can have many expenses).
- Category to Expense: 1 to many (one category can have many expenses).

Defining the operations that are used fo the objects

- User: createUser(), editUserDetails(), deleteUser().
- Budget: createBudget(), editBudgetDetails(), deleteBudget().
- Category: createCategory(), editCategoryDetails(), deleteCategory().
- Expense: addExpense(), editExpenseDetails(), deleteExpense().

4.2 Database Specifications and Analysis

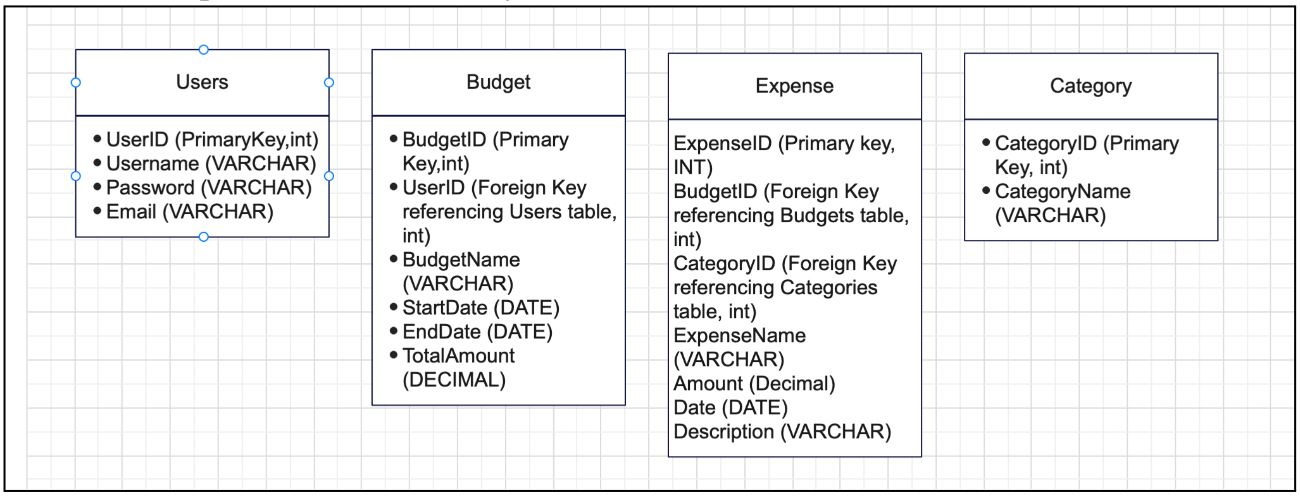

Users
• UserID (PrimaryKey,int)
• Username (VARCHAR)
• Password (VARCHAR)
• Email (VARCHAR)
Budget
• BudgetID (Primary Key,int)
• UserID (Foreign Key referencing Users table, int)
• BudgetName (VARCHAR)
• StartDate (DATE)
• EndDate (DATE)
• TotalAmount (DECIMAL)
Expense
ExpenseID (Primary key, INT)
BudgetID (Foreign Key referencing Budgets table, int)
CategoryID (Foreign Key referencing Categories table, int)
ExpenseName (VARCHAR)
Amount (Decimal)
Date (DATE)
Description (VARCHAR)
Category
• CategoryID (Primary Key, int)
• CategoryName (VARCHAR)

Fig 2: Tables and relationships of the database

This diagram shows the tables and relationships that will form the backbone of the budget buddy’s system database. We chose MySQL as the right database management system to choose from based on the team’s preferences and requirements.

4.3 Architectural Modeling

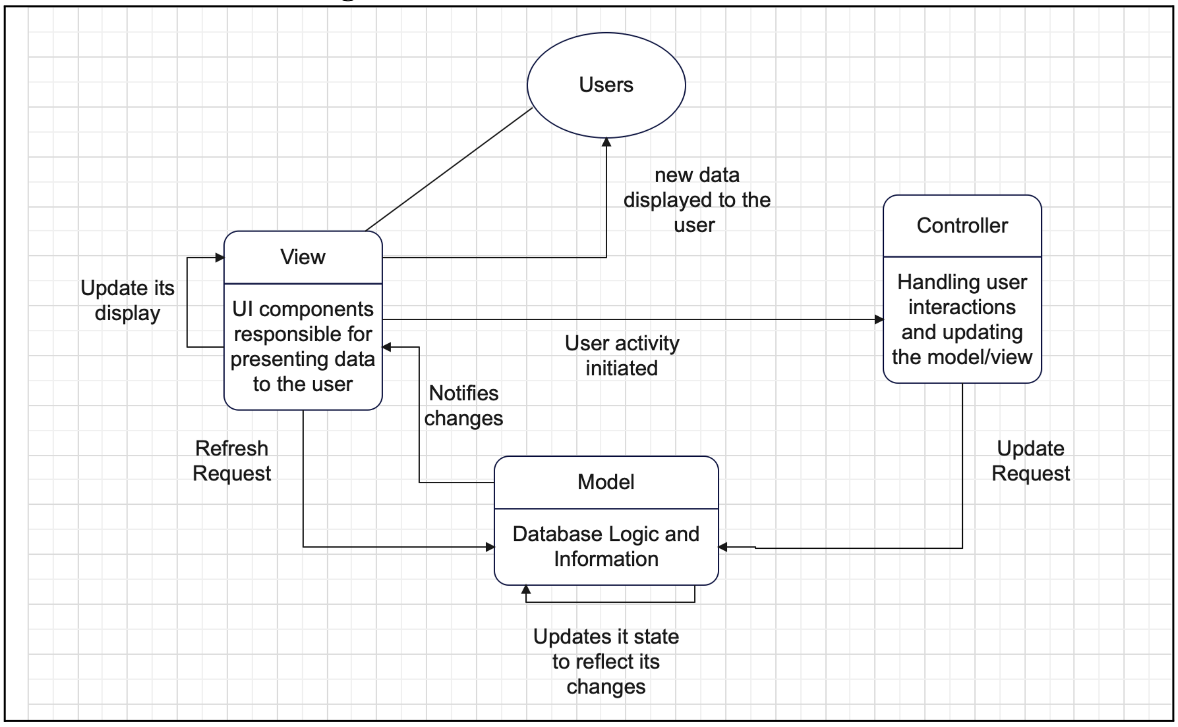

Users
new data displayed to the user
Controller
Handling user interactions and updating the model/view
View
UI components responsible for presenting data to the user
Update its display
User activity initiated
Notifies changes
Refresh Request
Update Request
Model
Database Logic and Information
Updates it state to reflect its changes

Fig 3: MVC Architectural pattern of the Budget Buddy

The architecture design pattern chosen is the MVC (Model,View,Controller) Design Pattern. MVC pattern aligns well with the requirements of the budget buddy project, it offers a structured approach to designing the class diagrams while promoting modularity, maintainability, and testability.

This is a simplified example of how these components of the MVC interact:

- User interacts with the View by, for example, adding a new expense through a form.
- The View captures this interaction and forwards it to the corresponding Controller.
- The Controller receives the user input, processes it (e.g., validates the input), and updates the Model accordingly.
- The Model updates its state to reflect the changes (e.g., adds a new expense to the list of expenses).
- The View observes the changes in the Model and updates its display to reflect the updated data (e.g., refreshes the expense list).

This cycle continues as users interact with our application, maintaining a clear separation of concerns between the data (Model), presentation (View), and application logic (Controller).

4.4 Behavioral Modeling

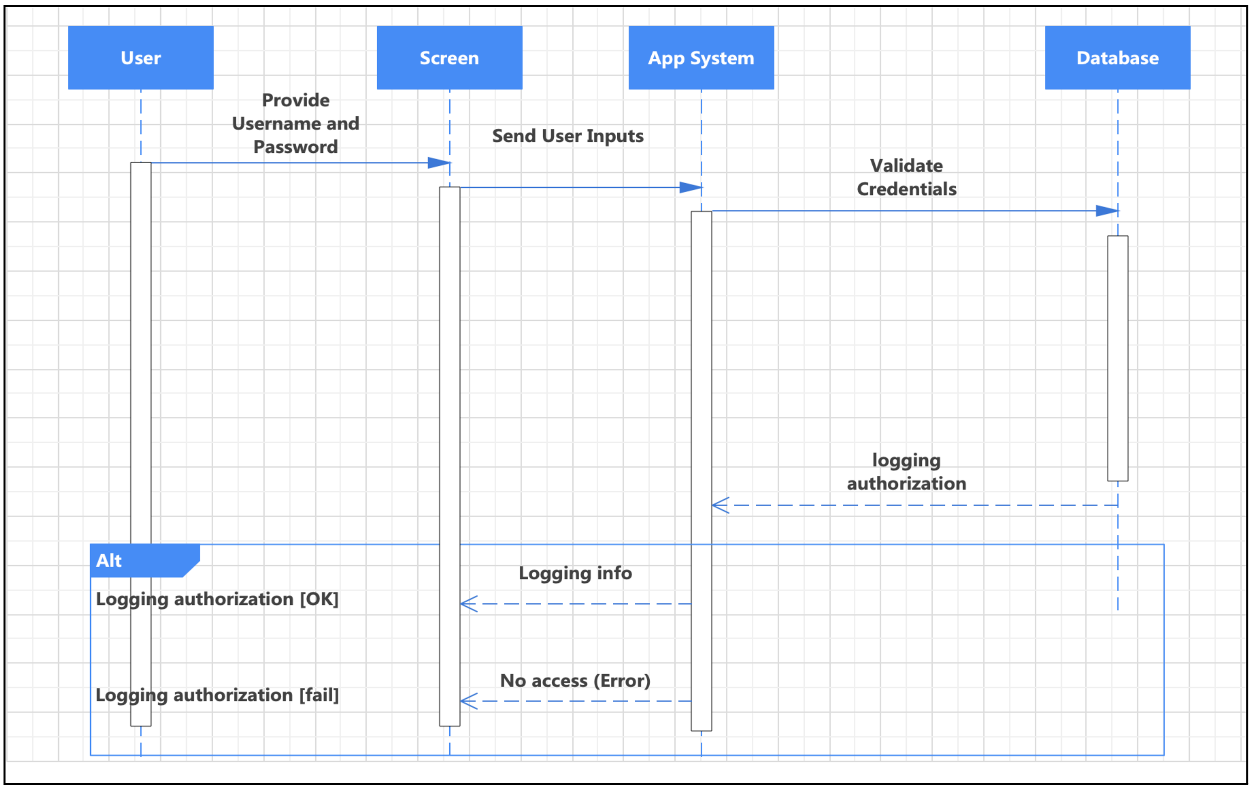


Fig 4: Sequence Diagram of use case 03: User Sign-in

The sequence diagram for the "User signing in" use case outlines the interactions between the user, screen(UI), App System and the database system. Here's the sequence of actions:

- The user enters the details of the information needed, including the username and password.
- The user submits the data.
- The UI sends the entered data to the app system.
- The app system validates the information by communicating with the database.
- The database sends a valid response to the app system
- The app system sends this valid response information to the UI.
- The updated information is displayed to the user.
- The user is signed in depending on the information received from the app system.

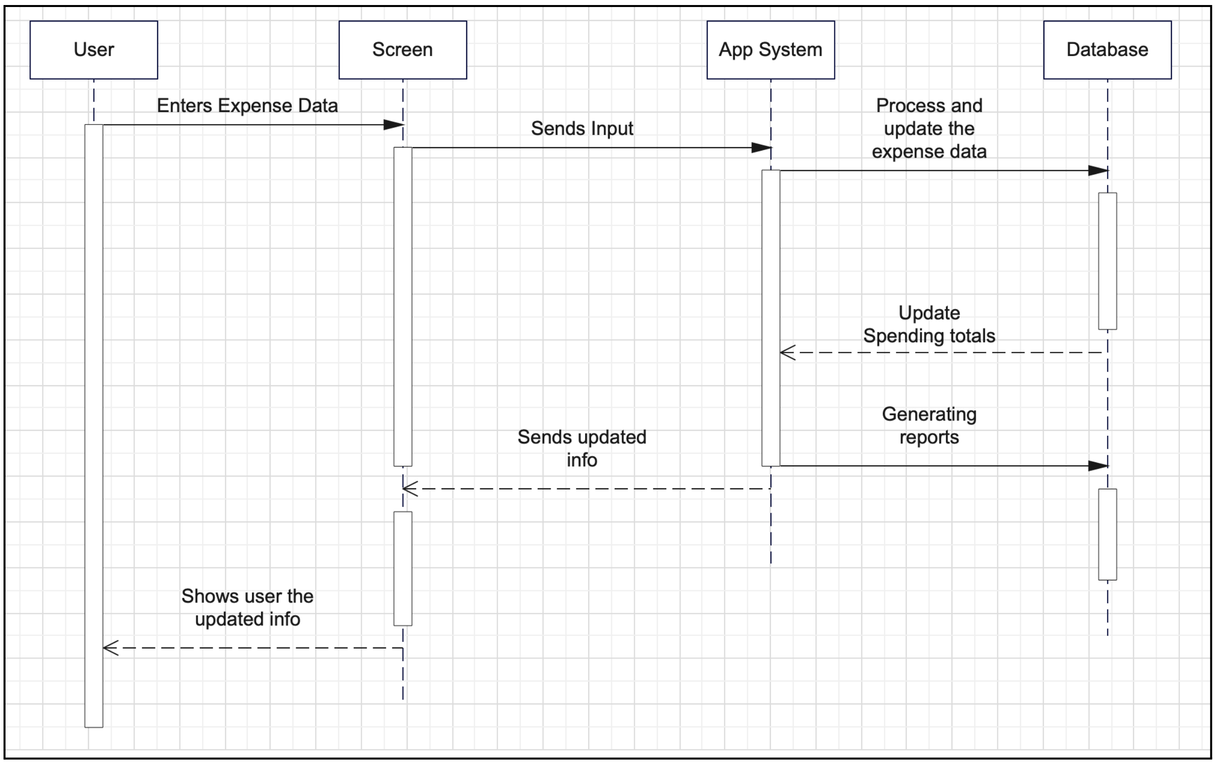


Fig 5: Sequential diagram of Use Case 1: Entering Expense Data

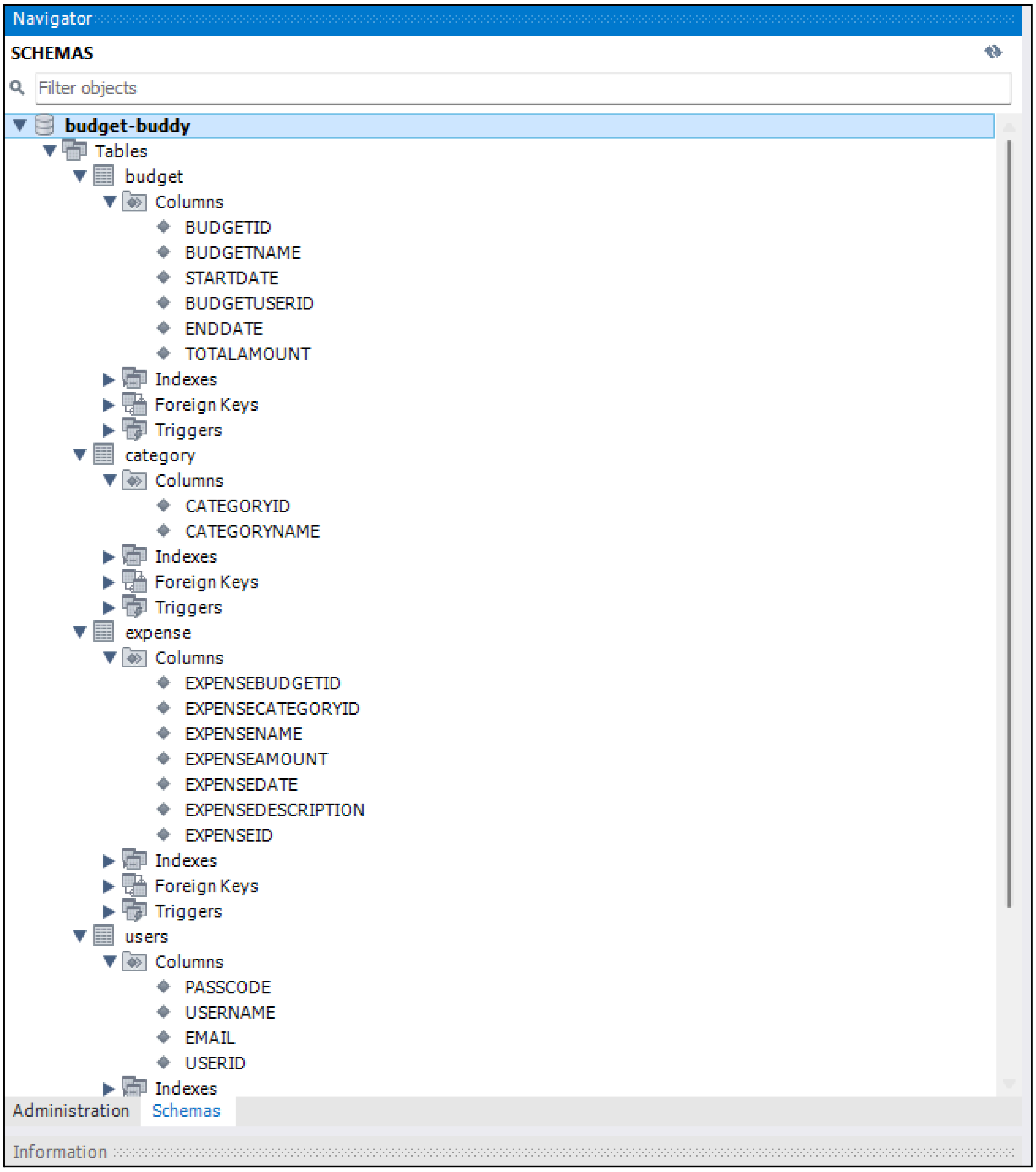
The sequence diagram for the "Entering Expense Data" use case outlines the interactions between the user, screen(UI), App System and the database system. Here's the sequence of actions:

- The user enters the details of the expense, including the amount, category, and date.
- The user submits the expense data.
- The UI sends the entered data to the app system.
- The app system processes the data and stores it in the database.
- The app system updates the user's spending totals.
- The app system generates relevant reports based on the updated data.
- The updated reports are displayed to the user.

SECTION 5

IMPLEMENTATION

5.1 Implement the Database design (Tables and Backend)



```
Navigator
SCHEMAS
Filter objects
▼ budget-buddy
  ▼ Tables
    ▼ budget
      ▼ Columns
          BUDGETID
          BUDGETNAME
          STARTDATE
          BUDGETUSERID
          ENDDATE
          TOTALAMOUNT
      ▶ Indexes
      ▶ Foreign Keys
      ▶ Triggers
    ▼ category
      ▼ Columns
          CATEGORYID
          CATEGORYNAME
      ▶ Indexes
      ▶ Foreign Keys
      ▶ Triggers
    ▼ expense
      ▼ Columns
          EXPENSEBUDGETID
          EXPENSECATEGORYID
          EXPENSENAME
          EXPENSEAMOUNT
          EXPENSEDATE
          EXPENSEDESCRIPTION
          EXPENSEID
      ▶ Indexes
      ▶ Foreign Keys
      ▶ Triggers
    ▼ users
      ▼ Columns
          PASSCODE
          USERNAME
          EMAIL
          USERID
      ▶ Indexes
Administration   Schemas
Information
```

Fig 6: Database Schema for the budget-buddy project

Fig 6 illustrates the database schema of our budget-buddy app, as designed using the MYSQL Workbench. It provides a visual representation of the structure and relationships between various entities within our system. The schema encompasses tables representing key entities such as users, expenses, incomes, categories, and budgets, along with their respective attributes and relationships. Additionally, it depicts the constraints, indexes, and keys defined to maintain data integrity and optimize query performance. This database schema serves as the foundation for storing and managing financial data, facilitating efficient budget tracking and analysis within our application.

5.2 Implement the Class Diagram design (Frontend and Logic)

Implementation Details

Software Frameworks and Languages
-Frontend: Java Swing Gui
-Backend Logic: Java application with business logic (JDBC for database communication)
-Database: MySQL

Software Installation
- Java Development Kit (JDK) is installed on our computer systems.
- MySQL database server (Workbench) and MySQL JDBC driver are also installed.

Main Parts Development
- Frontend: Developed user interfaces for User Registration and Sign-In functionalities using Java Swing components.
- Backend Logic: Implemented backend logic in Java, utilizing JDBC for database interaction to perform user registration and sign-in operations.

Code Structure
```java
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

public class App extends JFrame {

   private JTextField usernameField;
   private JPasswordField passwordField;

   public App() {
       setTitle("User Registration / Sign-In");
       setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
        setSize(300, 200);
        setLayout(new BorderLayout());

        JPanel inputPanel = new JPanel(new GridLayout(3, 2));
        inputPanel.add(new JLabel("Username:"));
        usernameField = new JTextField();
        inputPanel.add(usernameField);

        inputPanel.add(new JLabel("Password:"));
        passwordField = new JPasswordField();
        inputPanel.add(passwordField);

        JButton registerButton = new JButton("Register");
        registerButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                registerUser();
            }
        });

        JButton signInButton = new JButton("Sign In");
        signInButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                signInUser();
            }
        });

        inputPanel.add(registerButton);
        inputPanel.add(signInButton);

        add(inputPanel, BorderLayout.CENTER);
    }

    private void registerUser() {
        String username = usernameField.getText();
        String password = new String(passwordField.getPassword());

        // Load the MySQL JDBC driver
        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
        } catch (ClassNotFoundException e) {
            JOptionPane.showMessageDialog(this, "MySQL JDBC Driver not found!",
"Error", JOptionPane.ERROR_MESSAGE);
            return;
        }

        try (Connection conn =
DriverManager.getConnection("jdbc:mysql://localhost:3306/budgetbuddy","root",
"november281996")) {
            String sql = "INSERT INTO users(USERNAME, PASSCODE) VALUES(?, ?)";
            PreparedStatement pstmt = conn.prepareStatement(sql);
            pstmt.setString(1, username);
            pstmt.setString(2, password);
            pstmt.executeUpdate();
            JOptionPane.showMessageDialog(this, "User registered successfully!");
            clearFields();
        } catch (SQLException ex) {
            JOptionPane.showMessageDialog(this, "Error registering user: " +
ex.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
        }
    }

    private void signInUser() {
        String username = usernameField.getText();
        String password = new String(passwordField.getPassword());

        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
        } catch (ClassNotFoundException e) {
            JOptionPane.showMessageDialog(this, "MySQL JDBC Driver not found!",
"Error", JOptionPane.ERROR_MESSAGE);
            return;
        }

        try (Connection conn =
DriverManager.getConnection("jdbc:mysql://localhost:3306/budgetbuddy","root",
"november281996")) {
            String sql = "SELECT * FROM users WHERE USERNAME = ? AND PASSCODE = ?";
            PreparedStatement pstmt = conn.prepareStatement(sql);
            pstmt.setString(1, username);
            pstmt.setString(2, password);
            ResultSet rs = pstmt.executeQuery();

            if (rs.next()) {
                JOptionPane.showMessageDialog(this, "Sign in successful!");
            } else {
                JOptionPane.showMessageDialog(this, "Incorrect username or
password.", "Error", JOptionPane.ERROR_MESSAGE);
            }
            clearFields();
        } catch (SQLException ex) {
            JOptionPane.showMessageDialog(this, "Error signing in: " +
ex.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
        }
    }

    private void clearFields() {
        usernameField.setText("");
        passwordField.setText("");
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() {
                new App().setVisible(true);
            }
        });
    }
}
```

- The `App` class extends `JFrame` to create the GUI window for user registration and sign-in.
- User input fields (`JTextField` and `JPasswordField`) are used to capture username and password.
- Register and Sign In buttons are provided to trigger the respective actions.
- The `registerUser()` and `signInUser()` methods handle user registration and sign-in functionality, respectively.
- Database connection details are hardcoded in the `getConnection()` method.

Database Communication Testing
- Tested frontend-backend communication to ensure seamless data flow between GUI and database.
- Conducted tests for user registration and sign-in functionalities to verify database interaction.

Illustration of the test

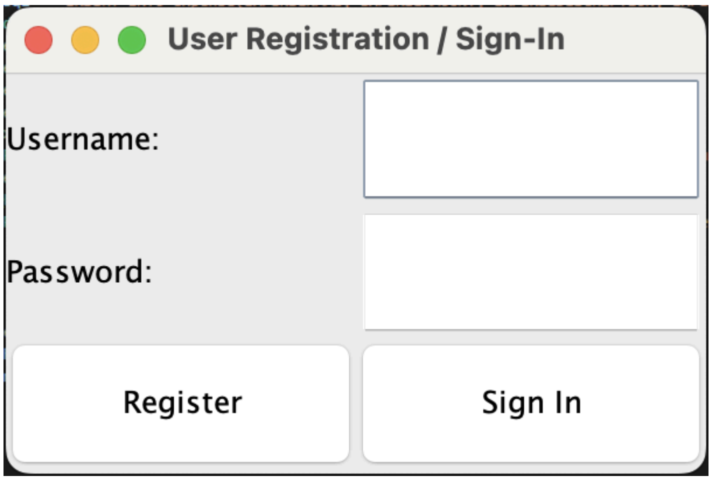


Fig 7 GUI representing the sign-in parameter for the user

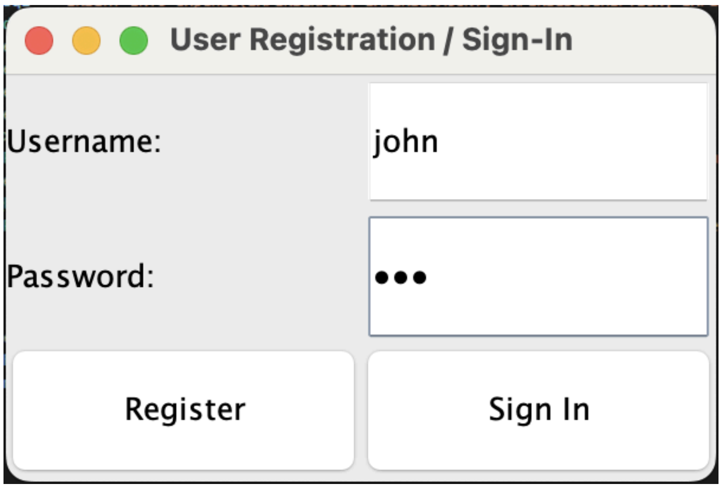
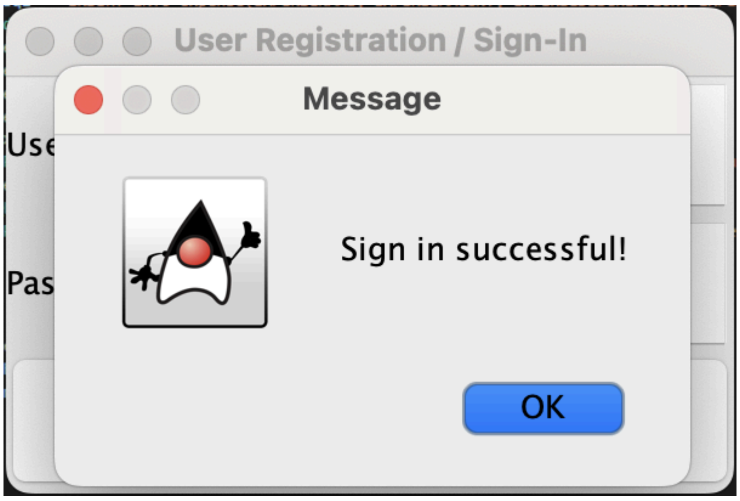
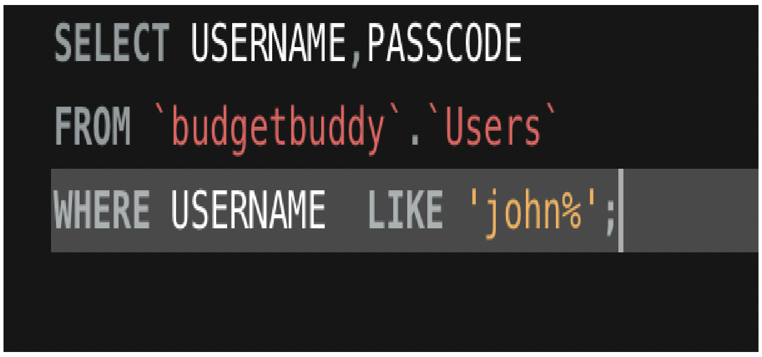


Fig 8 Represents the user entering the credentials for signing in



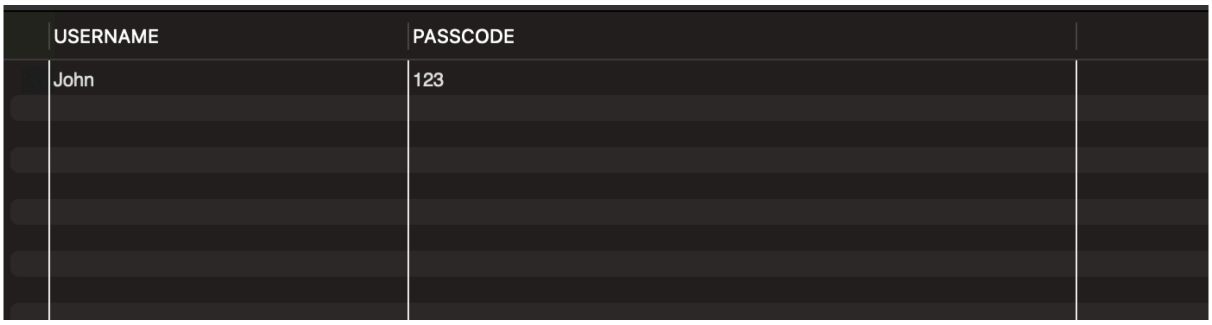
Sign in successful!

Fig 9 Signing in successful after user credentials have been validated



```
SELECT USERNAME,PASSCODE
FROM `budgetbuddy`.`Users`
WHERE USERNAME  LIKE 'john%';
```

Fig 10 Query that used to find username "john" in the budget-buddy database



USERNAME	PASSCODE
John	123

Fig 11 Show the user "john" credentials are successfully verified in the database

<u>Compiling and Running Instructions</u>

- To compile the application:
    - Ensure Java compiler is installed and available in the system's PATH.
    - Navigate to the directory containing the `App.java` file.
    - Run the command: `javac App.java`.

- To run the application:
    - Ensure MySQL database server is running.
    - Run the command: `java App` to launch the application.

SECTION 6

TESTING

For Use Case 03, "User Sign In":

1. Features

- User Authentication: Verify the credentials (username and password) against those stored in the Budget Buddy Database.
- Credential Validation: Check if the entered username and password match the required format and completeness.
- Access Control: Grant or deny access based on the validity of the entered credentials.

2. Partition Inputs into Equivalence Classes

For Username:

- Valid usernames: Correctly formatted, existing in the database.
- Invalid usernames: Incorrectly formatted, non-existing, or empty.

For Password:

- Valid passwords: Match the user's stored password in the database.
- Invalid passwords: Incorrect password, empty, or does not meet security requirements.

3. Test Specification

1. Sign in with valid credentials: signIn('registeredUser', 'ValidPassword')
2. Sign in with invalid username: signIn('nonExistingUser', 'AnyPassword')
3. Sign in with invalid password: signIn('registeredUser', 'WrongPassword')
4. Sign in with empty username: signIn('', 'AnyPassword')
5. Sign in with empty password: signIn('registeredUser', '')

4. Test Cases

Test Case 1: Sign in with valid credentials

- Input: ('registeredUser', 'ValidPassword')
- Expected Outcome: User is granted access to their account.

Test Case 2: Sign in with an invalid username

- Input: ('nonExistingUser', 'AnyPassword')
- Expected Outcome: User is not granted access, possibly with a message stating the username does not exist.

Test Case 3: Sign in with the wrong password

- Input: ('registeredUser', 'WrongPassword')
- Expected Outcome: User is not granted access, with a message indicating the password is incorrect.

Test Case 4: Attempt to sign in with an empty username

- Input: ('', 'AnyPassword')
- Expected Outcome: System alerts user that username cannot be empty.

Test Case 5: Attempt to sign in with an empty password
- Input: ('registeredUser', '')
- Expected Outcome: System alerts user that password cannot be empty.

For Use case 01, “Entering Expense Data”

1. Features

- Inputting expense details: amount, category, and date.
- Validation of input data for correctness and completeness.
- Recording and updating this data in the Budget Buddy Database.
- Reflecting the updated data in reports and graphs.

2. Partition Inputs into Equivalence Classes

For Amount:
- Negative amounts: invalid input.
- Positive amounts: valid input.
- Non-numeric values: invalid input.

For Category:
- Valid categories (e.g., food, bills, entertainment, grocery): valid input.
- Invalid categories (not listed or undefined categories): invalid input.

For Date:
- Future dates: invalid input.
- Past and present dates: valid input.
- Non-date formats (e.g., string that isn't a date, incorrect format): invalid input.

3. Test Specification

1. Enter valid expense data: enterExpenseData(100, 'food', '10-22-2023')
2. Enter expense with negative amount: enterExpenseData(-50, 'bills', '10-22-2023')
3. Enter expense with non-numeric amount: enterExpenseData('hundred', 'entertainment', '10-22-2023')
4. Enter expense with invalid category: enterExpenseData(50, 'unknownCategory', '10-22-2023')
5. Enter expense with future date (if considered invalid): enterExpenseData(50, 'grocery', 10-22-2024')
6. Enter expense with invalid date format: enterExpenseData(50, 'grocery', '10th January 2023')

4. Test Cases

Test Case 1: Entering a valid entry
- Input: (100, 'food', ‘10-22-2023’)
- Expected Outcome: Expense recorded successfully, data reflected in reports and graphs.

Test Case 2: Attempt to enter an expense with a negative amount.
- Input: (-50, 'bills', ‘10-22-2023’)
- Expected Outcome: System alerts user to correct the amount.

Test Case 3: Enter an expense with a non-numeric amount.

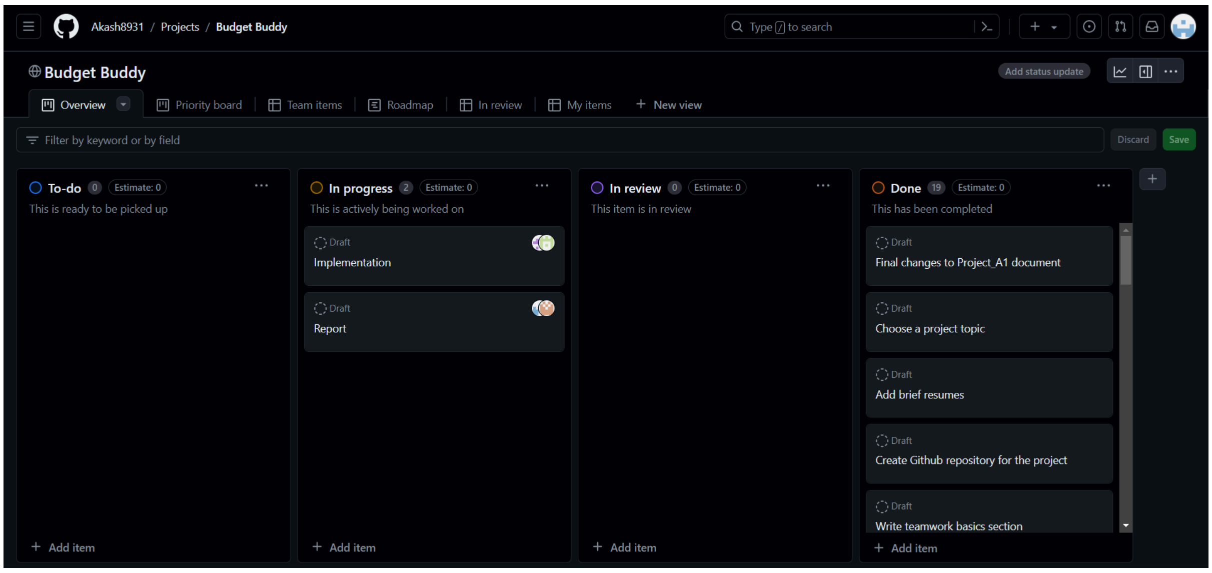
- Input: ('hundred', 'entertainment', ‘10-22-2023’)
- Expected Outcome: System prompts an alert asking the user to correct the amount.

Test Case 4: Enter an expense in an undefined category.
- Input: (50, 'unknownCategory',‘10-22-2023’)
- Expected Outcome: System alerts user to select a valid category.

Test Case 5: Enter an expense with a future date.
- Input: (50, 'grocery', '“10-01-2024”')
- Expected Outcome: System prompts an alert regarding date validity.

Test Case 6: Enter an expense with an invalid date format.
- Input: (50, 'grocery', '10th January 2023')
- Expected Outcome: System alerts user to enter the date in the correct format.

APPENDIX



To-do 0 Estimate: 0
This is ready to be picked up
Add item

In progress 2 Estimate: 0
This is actively being worked on
Draft
Implementation
Draft
Report
Add item

In review 0 Estimate: 0
This item is in review
Add item

Done 19 Estimate: 0
This has been completed
Draft
Final changes to Project_A1 document
Draft
Choose a project topic
Draft
Add brief resumes
Draft
Create Github repository for the project
Draft
Write teamwork basics section
Add item

Fig: Screenshot of Kanban Board for A3