
SOFTWARE ENGINEERING I

Spring 2024

BUDGET BUDDY

-Assignment #4-

Team 5:

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

April 15th, 2024

TABLE OF CONTENTS

S.No	Title	Page Number
1.	Section 1: Planning and Scheduling	3
2.	Section 2: Problem Statement	5
3.	Section 3: Requirements	
	3.1 User Requirements	7
	3.2 System Requirements	11
4.	Section 4: System Modeling	
	4.1 Class Diagrams	15
	4.2 Database specification and analysis	16
	4.3 Architecture modeling	16
	4.4 Behavioral modeling	19
5.	Section 5: Implementation	21
6.	Section 6: Testing	
	6.1 Testing environment and Test Cases	38
	6.2 Implementing the Test cases	43
	6.3 Test Documentation	48
7.	Appendix	49

SECTION 1 PLANNING & SCHEDULING

Assignee Name/ Email (@students.towson.edu)	Task	Duration (Hours)	Dependency	Due Date	Evaluation
Andrew McNeill / amcnei8 <i>[Coordinator]</i>	Initial Meeting	0.5hrs	Project A3	04/04/2024	100%
	Implementation	4hrs		04/15/2024	
Krishna Sruthi Velaga/ kvelaga1	Initial Meeting	0.5hrs	Project A3	04/04/2024	100%
	Testing	8hrs		04/15/2024	
	Format Report	1hr		04/15/2024	
Akash Reddy Karri/ akarri2	Initial Meeting	0.5hrs	Project A3	04/04/2024	100%
	Testing	4hrs		04/15/2024	
	Report	3hrs		04/15/2024	
Ahmed Hamza/ ahamza	Initial Meeting	0.5hrs	Project A3	04/04/2024	100%
	Communication and Collaboration	0.5 hrs		04/05/2024	
	System Modeling	7 hrs		04/13/2024	

Tobechukwu Ejike/ tejike1	Initial Meeting	0.5hrs	Project A3	04/04/2024	100%
	System Modeling	7hrs		04/15/2024	
	Implementation	5hrs		04/15/2024	

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 than 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

Use cases:

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 Resetting forgotten password	
Actors:	Registered User, Budget Buddy Database
Description:	When a registered user enters invalid password, they will be asked to enter the linked email address (or) mobile number to verify the user identity and then they will be able to change the password.
Alternate Path:	If the registered user remembers the password when they are trying to reset it, they can cancel the process and return to the sign-in page to access their account with the existing password
Pre-Condition:	The registered user has registered account and has valid credentials

05 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

06	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.

07	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.

08	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.

09**Set Budget Limits**

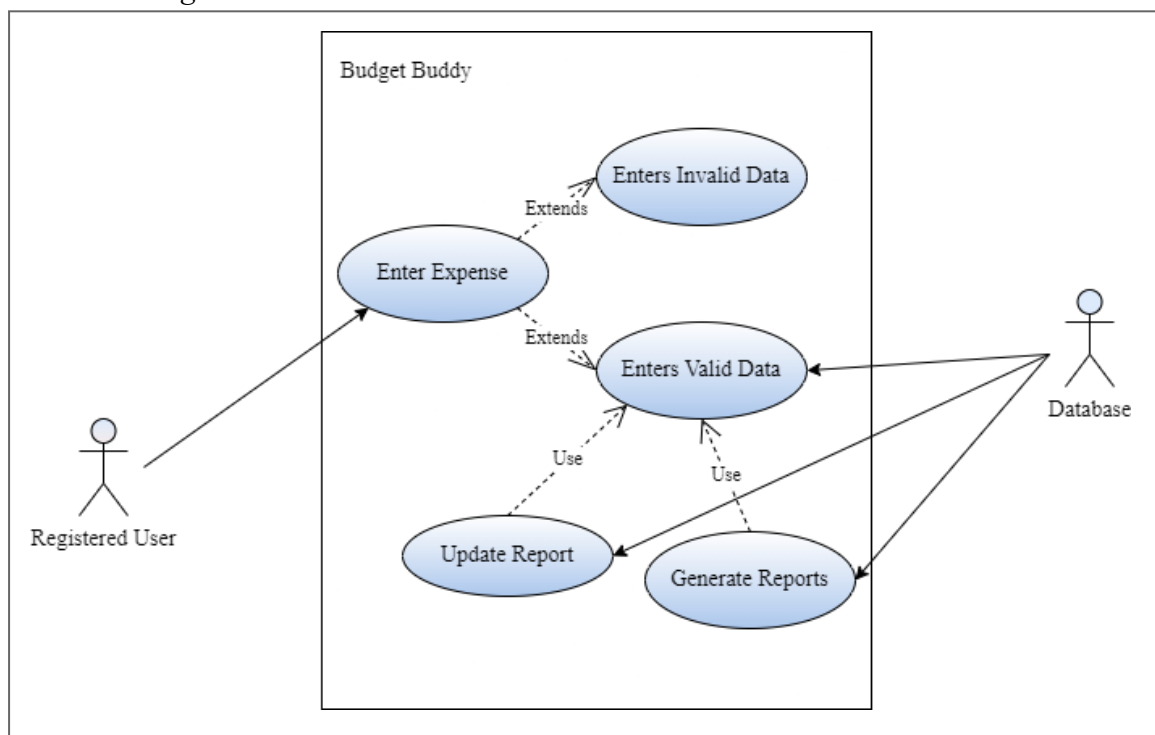
Actors: Registered User, Budget Buddy Database

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.

Use Case Diagram:



The use case “*Entering Expense Data*” is fundamental for the application. It is required that the user needs to give accurate and up-to-date information for proper management of expenses which enables the application to provide insightful reports and also send alerts to users.

3.2 System Requirements

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.

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 <ol style="list-style-type: none">1. First Name2. Last Name3. Username4. Email5. 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.

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.

Use Case 04 Resetting forgotten password	
Introduction:	When a registered user enters invalid password, they will be asked to enter the linked email address (or) mobile number to verify the user identity and then they will be able to change the password.
Inputs:	Selects the text label “Forgot Password”
Requirements Description:	The user needs to enter a valid email address.
Outputs:	The valid entered email address is sent an email with directions for the process of password changing.

Use Case 05 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.

Use Case 06	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.

Use Case 07	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.

Use Case 08 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.

Use Case 09 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 <ol style="list-style-type: none"> 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

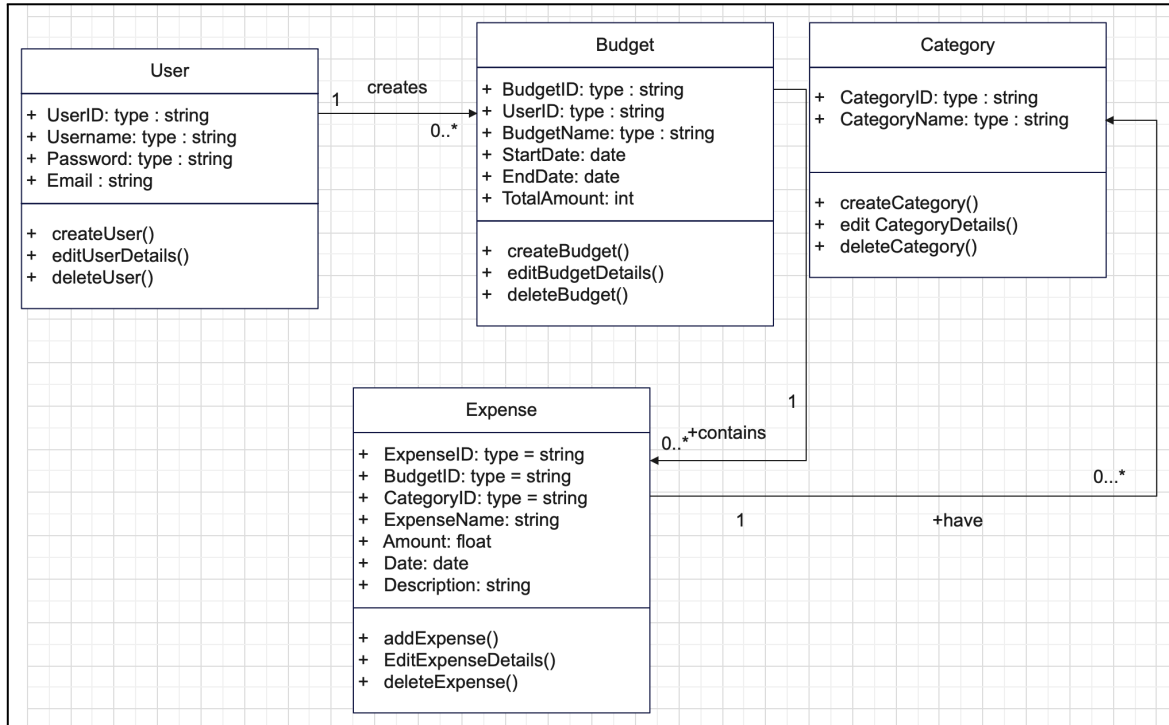


Fig 1: The class diagram represents the visual aspects of the budget buddy's system structure

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 for the objects

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

4.2 DATABASE SPECIFICATIONS AND ANALYSIS

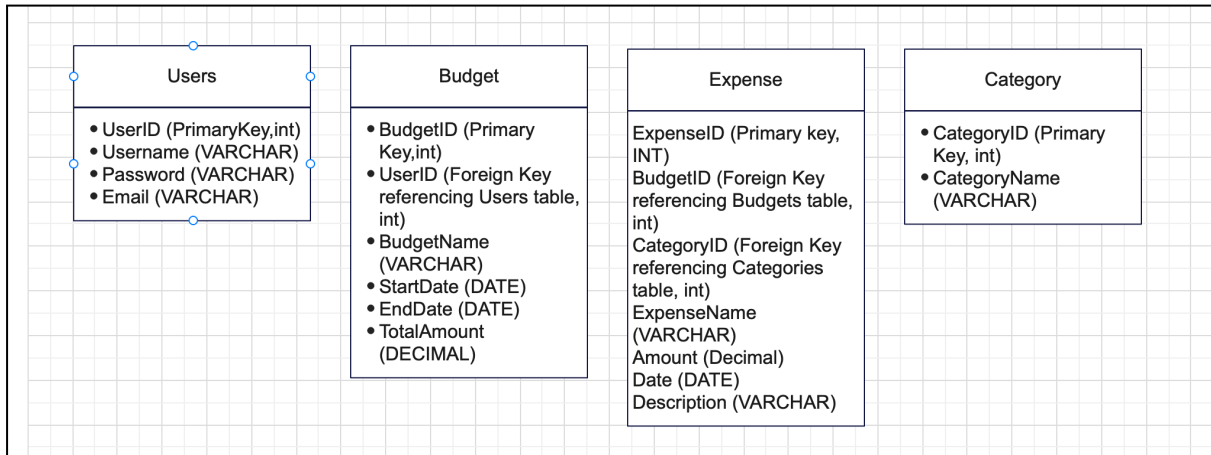


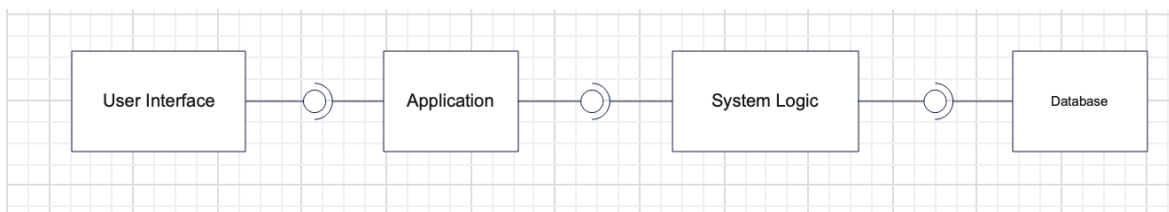
Fig 2: Tables and relationships of the database

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 ARCHITECTURE MODELING

4+1 ARCHITECTURE MODEL

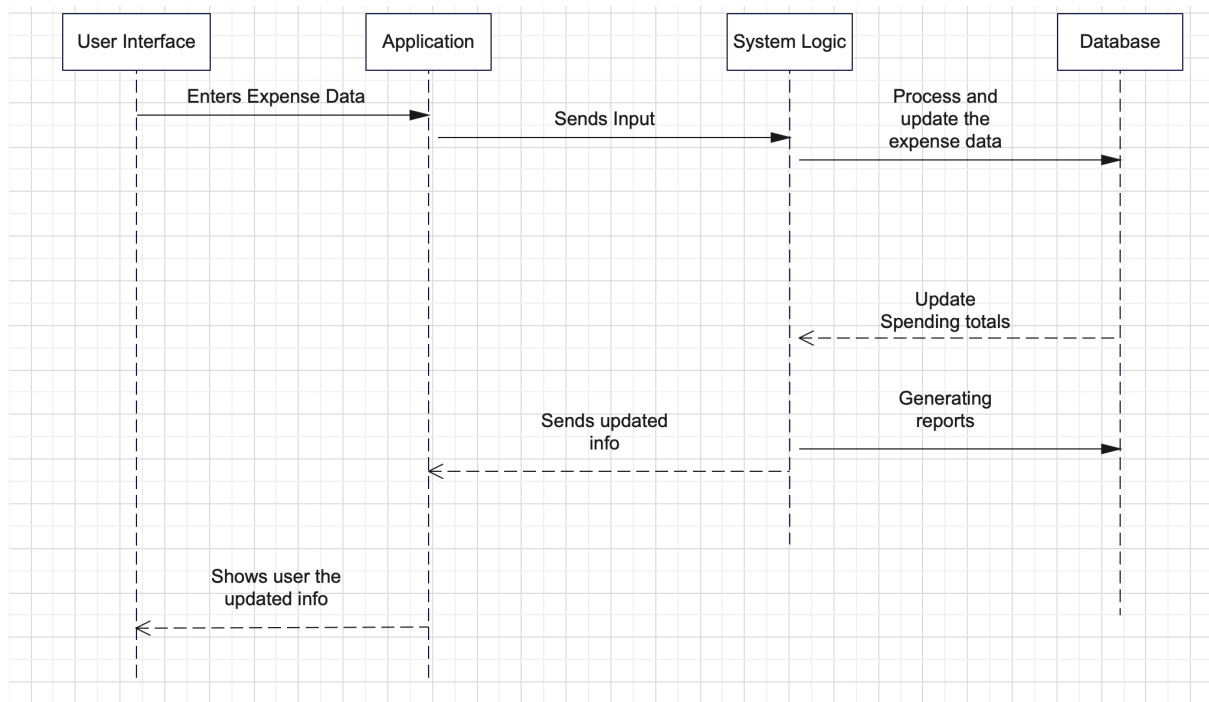
LOGICAL VIEW



The Logical View describes the functionalities of the system from a user's perspective. It focuses on what the system does rather than how it is implemented.

- User Interface (UI): This component represents the user-facing part of the system where users interact with the application. It includes elements such as forms, buttons, and navigation menus.
- Application and System Logic: These services handle the core business logic of the application, including user authentication, data processing, and interaction with the database.
- Database: This component manages the storage and retrieval of data. It includes functions such as data modeling, querying, and transaction management.

PROCESS VIEW

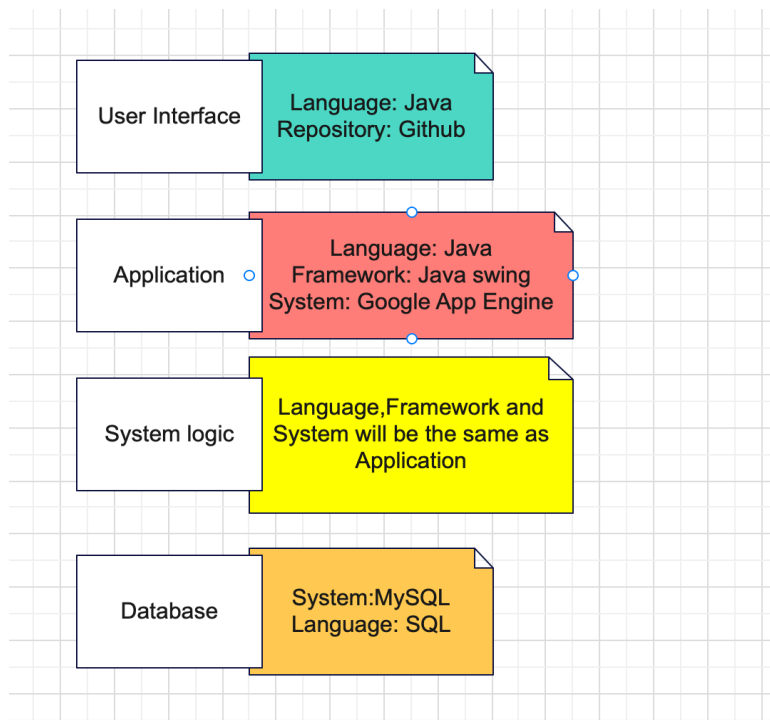


Use case: Enter Expense Data

The Process View focuses on the dynamic behavior of the system, including process flow and interactions between components.

- User Interaction Flows: This includes the sequence of steps users take to perform tasks within the system, such as logging expenses, setting budgets, and generating reports. The diagram represents an example of a user interaction.

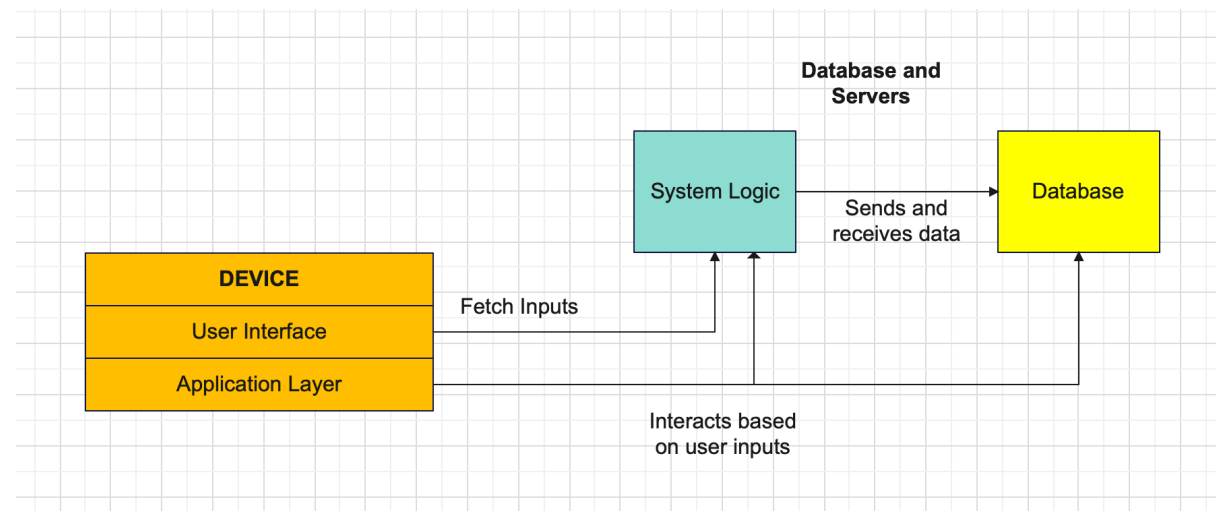
DEVELOPMENTAL VIEW



The Development View focuses on the software development aspect of the system, including modules, components, and technologies used.

- User Interface (Frontend): These are the components responsible for rendering the user interface elements and handling user interactions on the client side.
- Backend Components (Application & System Logic): These components handle requests from the client, process data, and execute business logic on the server side.
- Database: This includes the database management systems used for storing and retrieving data efficiently.

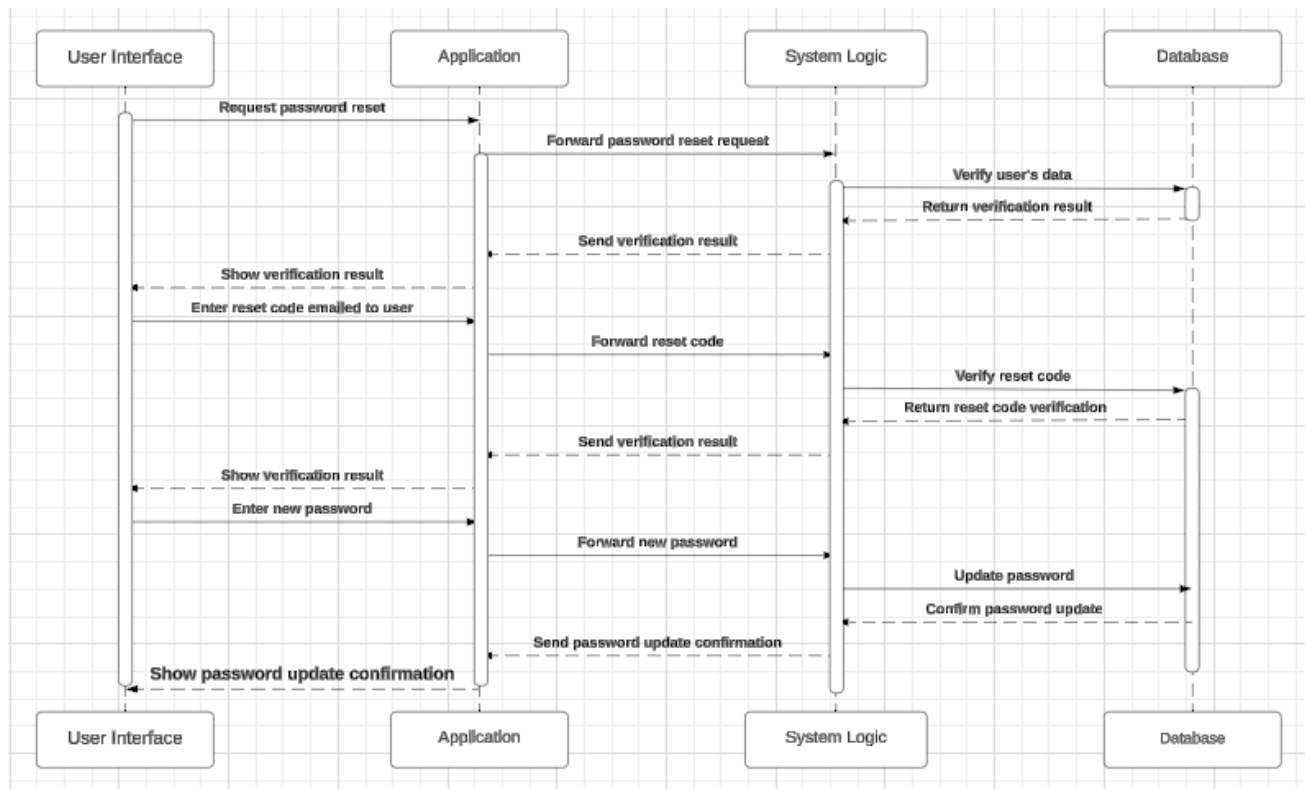
PHYSICAL VIEW



The Physical View illustrates the physical deployment of the system, including servers, networks, and infrastructure.

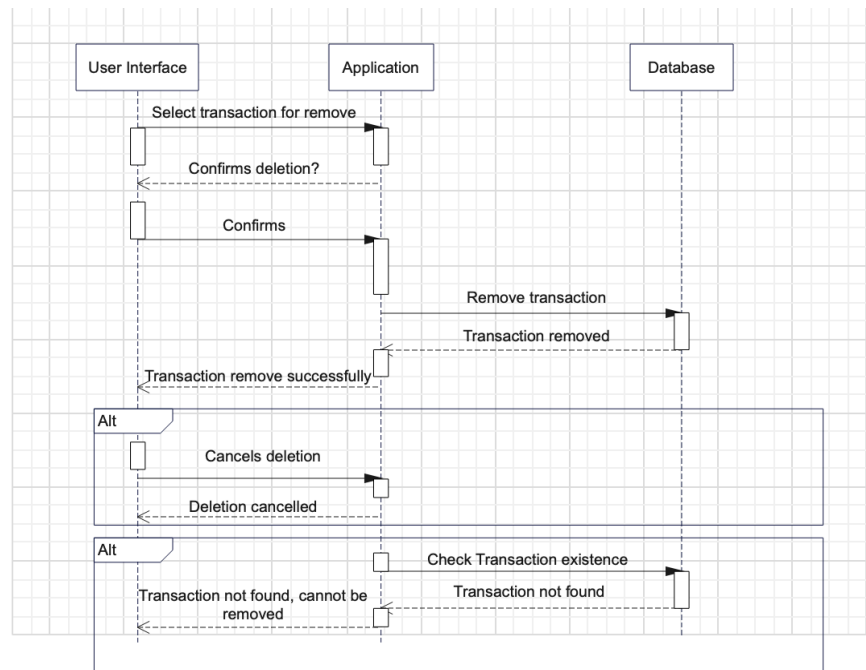
- Database and Cloud Platform (AWS, Azure, etc.): This represents the cloud infrastructure where the application is deployed. Cloud platforms will provide scalability, reliability, and accessibility.
- Load Balancer (Optimization): Load balancers which will help with the interactions between the Application, System logic and database, distribute incoming traffic across multiple instances of the application to ensure optimal performance and reliability.
- Data Replication (Redundancy): Data replication strategies are implemented between the UI and the system logic in order to ensure data availability and disaster recovery in case of failures.

BEHAVIORAL MODELING



Use Case 09 : Resetting Forgotten Password

1. User Requests Password Reset: The sequence starts with the user requesting to reset their password on the application. This could be done through a user interface where the user clicks on the forgotten password tab.
2. Forward Password Reset Request: The application then forwards the password reset request to the system logic which then moves it to the database for verification.
3. User Data Verification: The System logic forwards the user data to the database for verification.
4. Verification Results: The database verifies the user data and send the results back to the user, which then prompts the user to enter reset password code sent to their email.
5. User Enters Reset Code: The user enters a reset password code that has been sent to him to reset his password.
6. Reset Code Verification: The reset code is sent to the database and it is verified for correctness, if it is correct, the user will receive a return message to enter new password.
7. User Enters New Password: The user enters a new password and this is sent to the database and the database is updated.
8. Password reset or Update Confirmation: The user is sent a prompt to confirm the new password.



Use Case 07 : Delete Expense Data

1. User Selects Transaction: The sequence starts with the user selecting a transaction to remove from the application. This could be done through a user interface where the user interacts with a list of transactions.
2. Confirmation Request: The application then prompts the user for confirmation to proceed with the deletion. This step ensures that the user intends to remove the selected transaction.
3. User Confirmation: Upon receiving the confirmation from the user, the application proceeds with the deletion process.
4. Deletion Process: The application communicates with the database to remove the selected transaction from the user's data. This step involves sending a request to the database to delete the transaction.
5. Transaction Removed: If the deletion process is successful, the database acknowledges the removal by sending a response back to the application.
6. Success Notification: The application informs the user that the transaction has been successfully removed.

the alternate paths:

1. User Cancels Deletion: If the user decides to cancel the deletion process at any point before confirmation, the application cancels the deletion and informs the user accordingly.
2. Transaction Not Found: In case the application cannot find the transaction in the database (possibly because it has already been deleted or never existed), it communicates with the database to check the existence of the transaction. If the transaction is not found, the application notifies the user that the transaction cannot be removed.

The sequence diagram outlines the interactions between the user, the application, and the database when removing a transaction, considering both the main path and alternate paths based on user actions and database responses.

SECTION 5

IMPLEMENTATION

5.1 Implement the Database design (Tables and Backend)

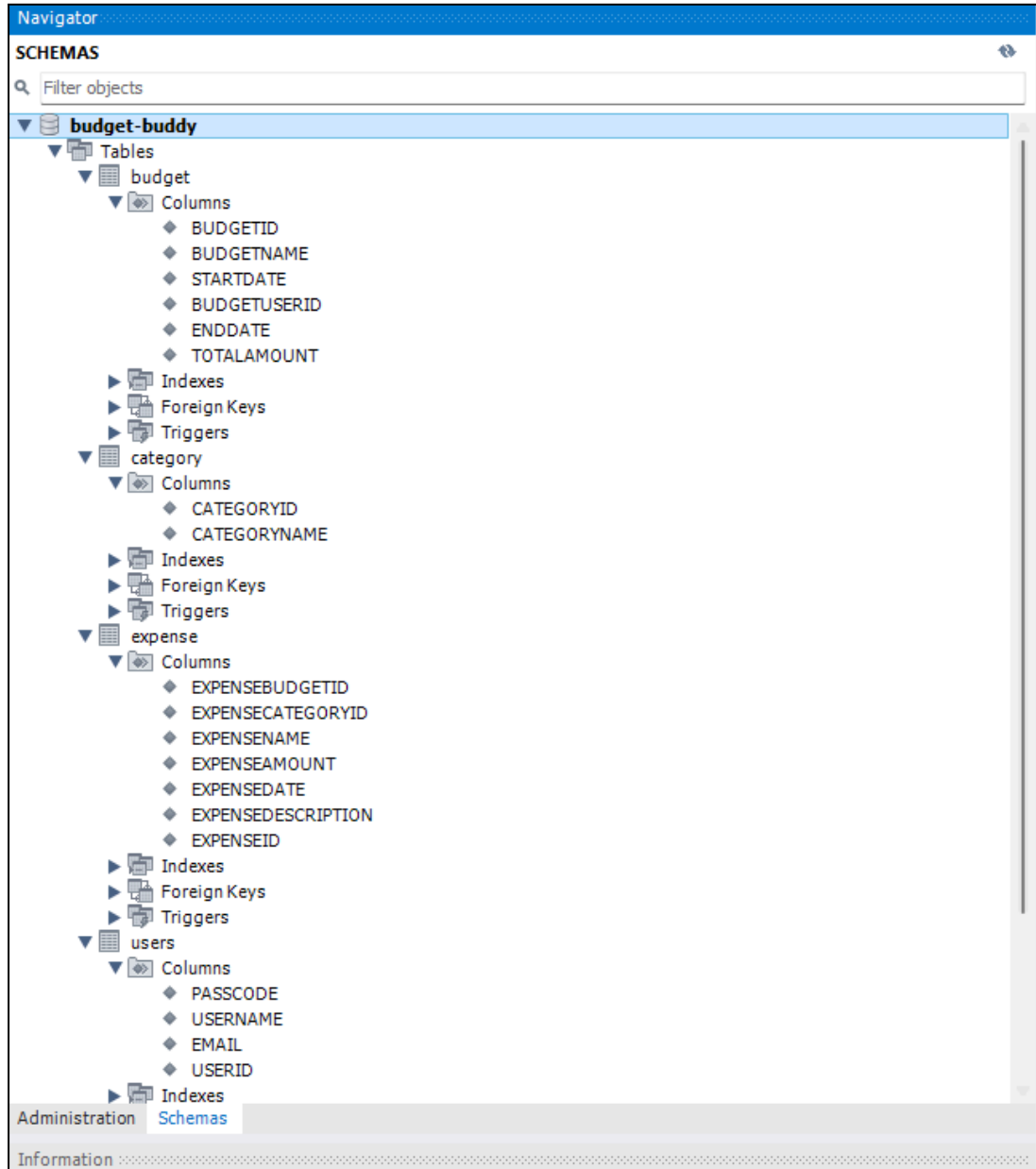


Fig 6: Database Schema for the budget-buddy project

The diagram 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 for Use Case : Forgotten Password

```
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 JTextField emailField;

    private JTextField mobileNumberField;

    private JPasswordField passwordField;
```

```
public App() {

    setTitle("User Registration / Sign-In / Forgot Password");

    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    setSize(400, 250);

    setLayout(new BorderLayout());

    JPanel inputPanel = new JPanel(new GridLayout(7, 2));

    inputPanel.add(new JLabel("Username:"));

    usernameField = new JTextField();

    inputPanel.add(usernameField);

    inputPanel.add(new JLabel("Email:"));

    emailField = new JTextField();

    inputPanel.add(emailField);

    inputPanel.add(new JLabel("Mobile Number:"));

    mobileNumberField = new JTextField();

    inputPanel.add(mobileNumberField);

    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 forgotPasswordButton = new JButton("Forgot Password");

forgotPasswordButton.addActionListener(new ActionListener() {

    @Override

    public void actionPerformed(ActionEvent e) {

        forgotPassword();

    }

});

JButton signInButton = new JButton("Sign In");

signInButton.addActionListener(new ActionListener() {

    @Override

    public void actionPerformed(ActionEvent e) {

        signInUser();

    }

});

inputPanel.add(registerButton);

inputPanel.add(forgotPasswordButton);

inputPanel.add(signInButton);

add(inputPanel, BorderLayout.CENTER);

}

private void registerUser() {

    String username = usernameField.getText();
```



```

String email = emailField.getText();

String mobileNumber = mobileNumberField.getText();

String password = new String(passwordField.getPassword());

// Check if email ends with ".com"

if (!email.endsWith(".com")) {

    JOptionPane.showMessageDialog(this, "Email must end with '.com'. Please
enter a valid email address.", "Error", JOptionPane.ERROR_MESSAGE);

    return;

}

// 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")) {

    // Check if email is already in use

    String emailCheckQuery = "SELECT * FROM users WHERE EMAIL = ?";

    PreparedStatement emailCheckStmt =
conn.prepareStatement(emailCheckQuery);

    emailCheckStmt.setString(1, email);

    ResultSet emailCheckResult = emailCheckStmt.executeQuery();

```

```

        if (emailCheckResult.next()) {

            JOptionPane.showMessageDialog(this, "Email already in use. Please
use a different email address.", "Error", JOptionPane.ERROR_MESSAGE);

            return;

        }

        // Register the user

        String registerQuery = "INSERT INTO users (USERNAME, EMAIL,
MOBILE_NUMBER, PASSCODE) VALUES(?, ?, ?, ?)";

        PreparedStatement registerStmt = conn.prepareStatement(registerQuery);

        registerStmt.setString(1, username);

        registerStmt.setString(2, email);

        registerStmt.setString(3, mobileNumber);

        registerStmt.setString(4, password);

        registerStmt.executeUpdate();

        JOptionPane.showMessageDialog(this, "User registered successfully!");

        // Navigate to home screen or perform any other action here

        clearFields();

    } catch (SQLException ex) {

        JOptionPane.showMessageDialog(this, "Error registering user: " +
ex.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);

    }

}

private void forgotPassword() {

    String username = usernameField.getText();

    String email = emailField.getText();

    String mobileNumber = mobileNumberField.getText();

```

```

        // 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")) {

            // Check if username exists

            String usernameCheckQuery = "SELECT * FROM users WHERE USERNAME = ?";

            PreparedStatement usernameCheckStmt =
conn.prepareStatement(usernameCheckQuery);

            usernameCheckStmt.setString(1, username);

            ResultSet usernameCheckResult = usernameCheckStmt.executeQuery();

            if (!usernameCheckResult.next()) {

                JOptionPane.showMessageDialog(this, "Username not found. Please
enter a valid username.", "Error", JOptionPane.ERROR_MESSAGE);

                return;

            }

            // Check if email or mobile number matches the user

            String getUserQuery = "SELECT * FROM users WHERE USERNAME = ? AND (EMAIL
= ? OR MOBILE_NUMBER = ?)";

            PreparedStatement getUserStmt = conn.prepareStatement(getUserQuery);

            getUserStmt.setString(1, username);

            getUserStmt.setString(2, email);

```

```

        getUserStmt.setString(3, mobileNumber);

        ResultSet getUserResult = getUserStmt.executeQuery();

        if (getUserResult.next()) {

            String newPassword = JOptionPane.showInputDialog(this, "Enter new
password:");

            if (newPassword != null && !newPassword.isEmpty()) {

                String resetPasswordQuery = "UPDATE users SET PASSCODE = ? WHERE
USERNAME = ?";

                PreparedStatement resetPasswordStmt =
conn.prepareStatement(resetPasswordQuery);

                resetPasswordStmt.setString(1, newPassword);

                resetPasswordStmt.setString(2, username);

                resetPasswordStmt.executeUpdate();

                JOptionPane.showMessageDialog(this, "Password reset
successful!");

            } else {

                JOptionPane.showMessageDialog(this, "Password cannot be empty!",
"Error", JOptionPane.ERROR_MESSAGE);

            }

        } else {

            JOptionPane.showMessageDialog(this, "Invalid email or mobile number
for the provided username.", "Error", JOptionPane.ERROR_MESSAGE);

        }

        clearFields();

    } catch (SQLException ex) {

        JOptionPane.showMessageDialog(this, "Error resetting password: " +
ex.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);

    }

}

```

```

private void signInUser() {

    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")) {

        // Check if username and password match

        String signInQuery = "SELECT * FROM users WHERE USERNAME = ? AND
PASSCODE = ?";

        PreparedStatement signInStmt = conn.prepareStatement(signInQuery);

        signInStmt.setString(1, username);

        signInStmt.setString(2, password);

        ResultSet signInResult = signInStmt.executeQuery();

        if (signInResult.next()) {

            JOptionPane.showMessageDialog(this, "Sign in successful!");

        } else {

            // If password is incorrect but username is correct, prompt user to
reset password

            String forgotPasswordPrompt = "Password is incorrect. Do you want to
reset your password?";

```

```
        int option = JOptionPane.showConfirmDialog(this,
forgotPasswordPrompt, "Forgot Password", JOptionPane.YES_NO_OPTION);

        if (option == JOptionPane.YES_OPTION) {

            forgotPassword();

        }

    }

    clearFields();

} catch (SQLException ex) {

    JOptionPane.showMessageDialog(this, "Error signing in: " +
ex.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);

}

}

private void clearFields() {

    usernameField.setText("");

    emailField.setText("");

    mobileNumberField.setText("");

    passwordField.setText("");

}

public static void main(String[] args) {

    SwingUtilities.invokeLater(new Runnable() {

        @Override

        public void run() {

            new App().setVisible(true);

        }

    });

}
```

```
}
```

1. GUI Setup:

- The application window is set up using `JFrame` with a title "User Registration / Sign-In / Forgot Password".
- The window size is set to 400 x 250 pixels, and the layout is organized using `BorderLayout`.
- Input fields for username, email, mobile number, and password are arranged in a `GridLayout` within a panel (`inputPanel`).

2. Input Fields and Buttons:

- Text fields (`usernameField`, `emailField`, `mobileNumberField`, `passwordField`) are provided for users to input their information.
- Buttons are created for registration, password reset, and sign-in functionalities (`registerButton`, `forgotPasswordButton`, `signInButton`).

3. Action Listeners:

- Action listeners are added to each button to handle user interactions:
- `registerButton` triggers the `registerUser()` method.
- `forgotPasswordButton` triggers the `forgotPassword()` method.
- `signInButton` triggers the `signInUser()` method.

4. User Registration (`registerUser()`):

- The method collects user input data (username, email, mobile number, password).
- It validates the email format to ensure it ends with ".com".
- If the email is invalid or already in use, an error message is displayed.
- If registration is successful, the user data is inserted into the database.

5. Forgot Password (`forgotPassword()`):

- The method allows users to reset their password if they provide the correct username, email, or mobile number.
- If the provided information matches a user's record, a new password is prompted and updated in the database.

6. Sign-In (`signInUser()`):

- The method verifies the provided username and password against the database records.
- If the credentials are correct, a success message is shown.
- If the password is incorrect but the username is correct, the user is prompted to reset their password.
- If the provided username or password is incorrect, an error message is displayed.

7. Database Interaction:

- The application interacts with the MySQL database named "budgetbuddy".
- It uses JDBC to establish a connection to the database and execute SQL queries for user registration, sign-in, and password reset.

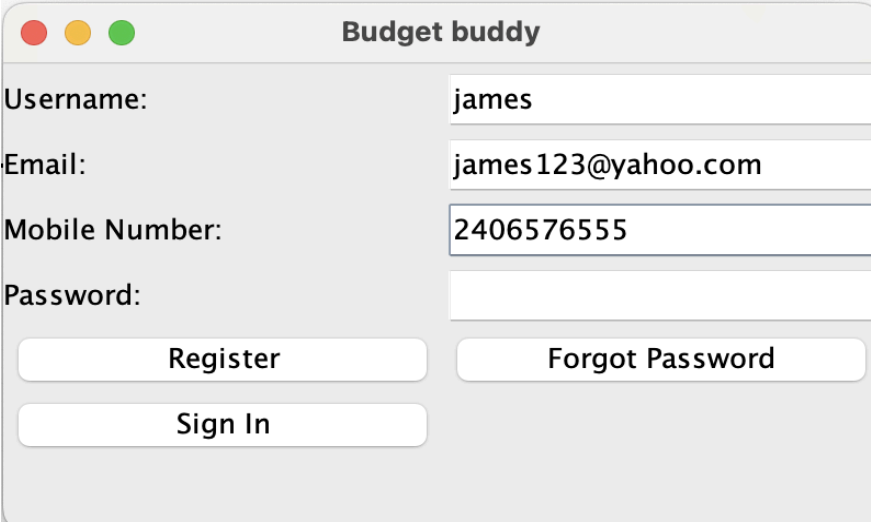
8. Error Handling:

- Error messages are displayed using `JOptionPane` for various scenarios such as invalid input, database connection issues, or SQL errors.

9. Main Method:

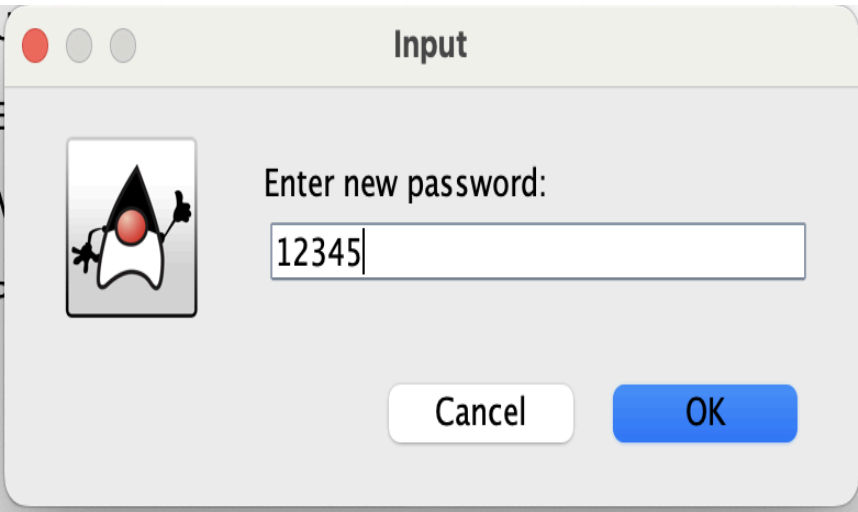
- The `main` method starts the Swing application by creating an instance of `App` and making it visible.

Illustration for test



The screenshot shows a Java Swing window titled "Budget buddy". It contains four text input fields for "Username:", "Email:", "Mobile Number:", and "Password:". The "Username" field contains the text "james", the "Email" field contains "james123@yahoo.com", and the "Mobile Number" field contains "2406576555". Below the input fields are three buttons: "Register", "Sign In", and "Forgot Password". The "Sign In" button is positioned below the "Register" button.

GUI representing the sign-in parameter for the user to reset password



The screenshot shows a Java Swing dialog box titled "Input". It features a small icon of a character with a red nose and a black hat on the left. To the right of the icon is the text "Enter new password:". Below this text is a text input field containing the text "12345". At the bottom of the dialog are two buttons: "Cancel" and "OK".

Image represents the user entering a new password

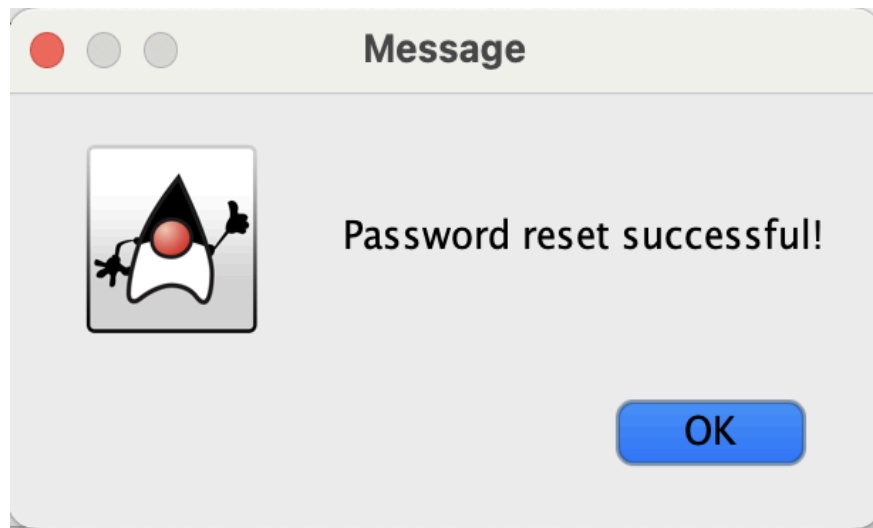


Image represents password being successfully reset

USERNAME	PASSCODE	EMAIL	USERID	MOBILE_NUMBER
james	1234	james123@yahoo.com	3	2406576555

Image represents user with previous password in the SQL database

USERNAME	PASSCODE	EMAIL	USERID	MOBILE_NUMBER
james	12345	james123@yahoo.com	3	2406576555

Image represents user with new password in the SQL database

Compiling and Running Instructions

- 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.

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 for Use Case :Deleting an Expense

```
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.SQLException;

public class Delete extends JFrame {
    private JTextField expenseNameField;
    private JTextField expenseDescriptionField;
    private JTextField expenseDateField;
    private JTextField expenseAmountField;
    private JButton deleteButton;

    public Delete() {
        setTitle("Expense Tracker");
        setSize(350, 350);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLocationRelativeTo(null); // Center the window

        JPanel panel = new JPanel();
        panel.setLayout(new GridLayout(5, 4));

        JLabel nameLabel = new JLabel("Expense Name To Delete:");
        expenseNameField = new JTextField();

        JLabel amountLabel = new JLabel("Expense Amount:");
        expenseAmountField = new JTextField();

        deleteButton = new JButton("Delete Expense");

        panel.add(nameLabel);
        panel.add(expenseNameField);

        panel.add(amountLabel);
        panel.add(expenseAmountField);

        panel.add(new JLabel()); // Placeholder for better alignment
        panel.add(deleteButton);
    }
}
```

```

deleteButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        // Add action to add expense here
        String name = expenseNameField.getText();
        String amount = expenseAmountField.getText();

        // Perform whatever operation you want with the expense
data
        // System.out.println("Expense Name: " + name);
        // System.out.println("Expense Amount: " + amount);
        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
        } catch (ClassNotFoundException error) {
            JOptionPane.showMessageDialog(null, "MySQL JDBC
Driver not found!", "Error",
                JOptionPane.ERROR_MESSAGE);
            return;
        }

        try (Connection conn =
DriverManager.getConnection("jdbc:mysql://localhost:3306/budget-buddy",
"root",
            "Mcneill16!")) {
            String sql = "DELETE FROM expense WHERE
(EXPENSENAME = ?) && (EXPENSEAMOUNT = ?)";
            PreparedStatement pstmt =
conn.prepareStatement(sql);
            pstmt.setString(1, name);
            pstmt.setString(2, amount);
            pstmt.executeUpdate();
            JOptionPane.showMessageDialog(null, "Expense
Successfully Deleted");
            clearFields();
        } catch (SQLException ex) {
            JOptionPane.showMessageDialog(null, "Error Try
Again: " + ex.getMessage(), "Error",
                JOptionPane.ERROR_MESSAGE);
        }
    }
}

```

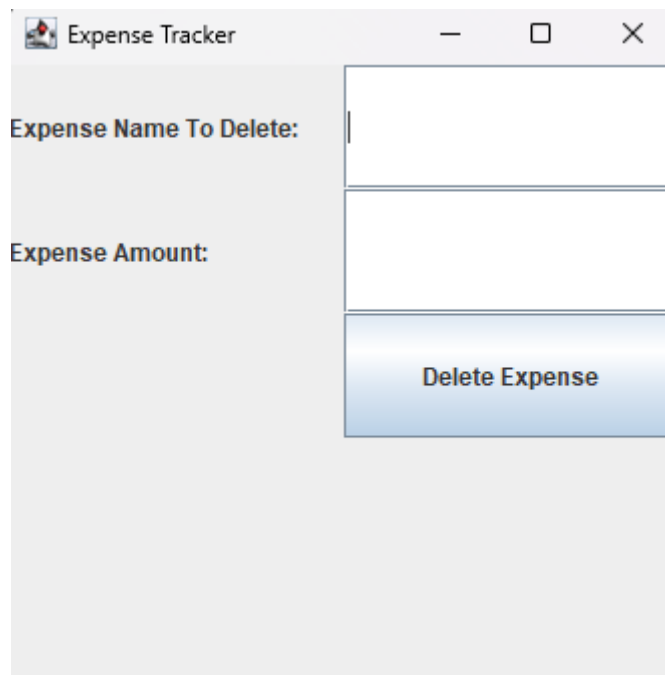
```

        private void clearFields() {
            expenseNameField.setText("");
            expenseAmountField.setText("");
        }
    });

    add(panel);
    setVisible(true);
}

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

```



The image shows a Java Swing window titled "Expense Tracker". Inside the window, there is a light gray panel. On the left side of the panel, there are two labels: "Expense Name To Delete:" and "Expense Amount:". To the right of these labels are two empty text input fields. Below these two fields is a blue button with the text "Delete Expense".

1. GUI Setup:

- The application window is set up using `JFrame` with a title "Budget Buddy".
- The window size is set to 350 x 350 pixels.
- Input fields for expense name and expense amount are arranged in a `GridLayout` within a panel (`inputPanel`).

2. Input Fields and Buttons:

- Text fields ("expenseNameField" and "expenseAmountField") are provided for users to delete an expense.
- Buttons are created for deleting the instance of the expense record in the SQL database ("deleteButton").

3. Action Listeners:

- Action listeners are added to each button to handle user interactions:
- The button labeled "Delete Expense" being clicked will trigger the deletion of the entered expense name and amount

7. Database Interaction:

- The application interacts with the MySQL database named "budgetbuddy".
- It uses JDBC to establish a connection to the database and execute SQL queries for deleting expense data

8. Error Handling:

- Error messages are displayed using `JOptionPane` for various scenarios such as invalid input, database connection issues, or SQL errors.

9. Main Method:

- The `main` method starts the application by creating an instance of `Delete`.

SECTION 6

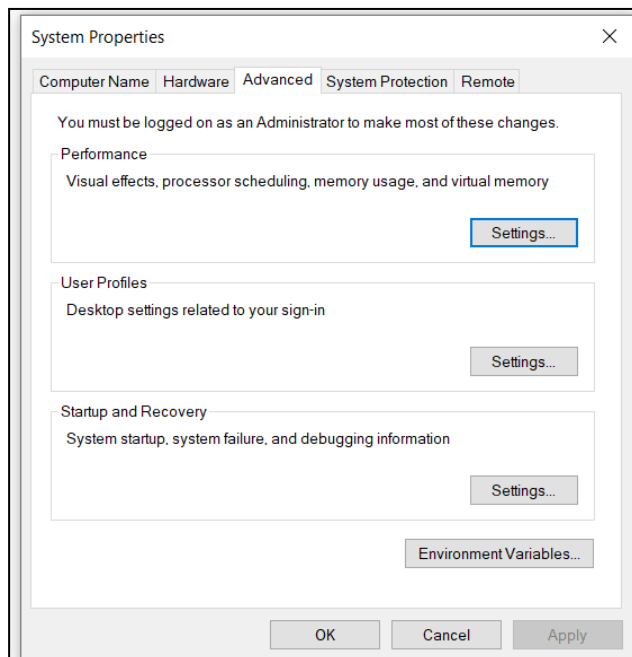
TESTING

6.1 Creating Testing environment and Test Cases

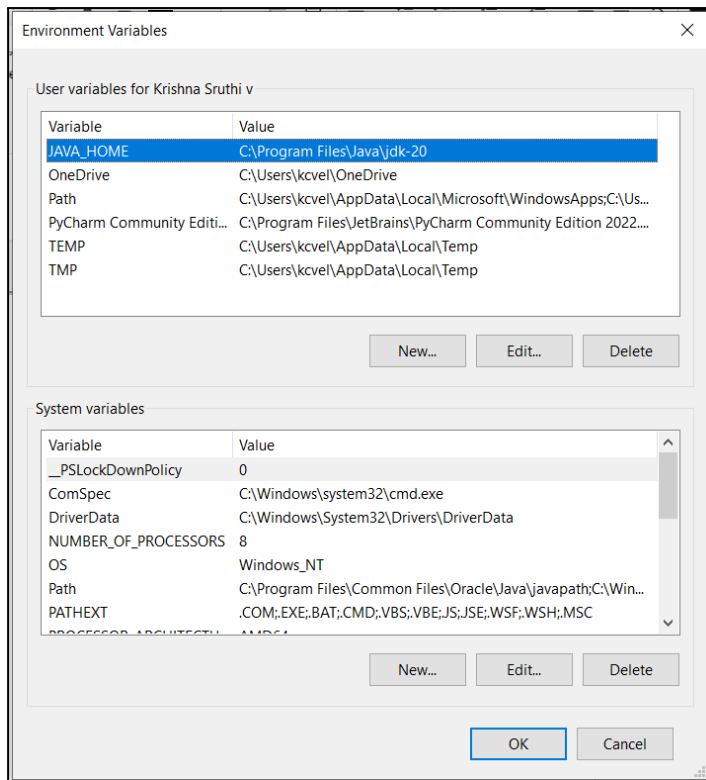
For our application, which is developed in Java, we have chosen to use JUnit.

Installation:

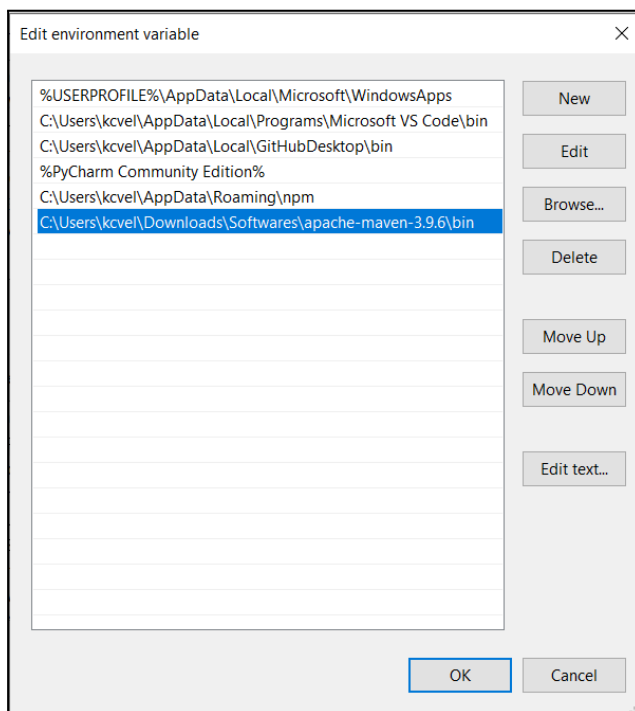
- Install the binary zip archive ('apache-maven-3.9.6-bin, i.e latest version) from Apache Maven download page (<https://maven.apache.org/download.cgi>)
- Extract the contents of the zip file to a directory in the system.
- Press 'Win+R', type 'sysdm.cpl' and hit Enter to open system properties
- Go to the "Advanced" tab located on the top



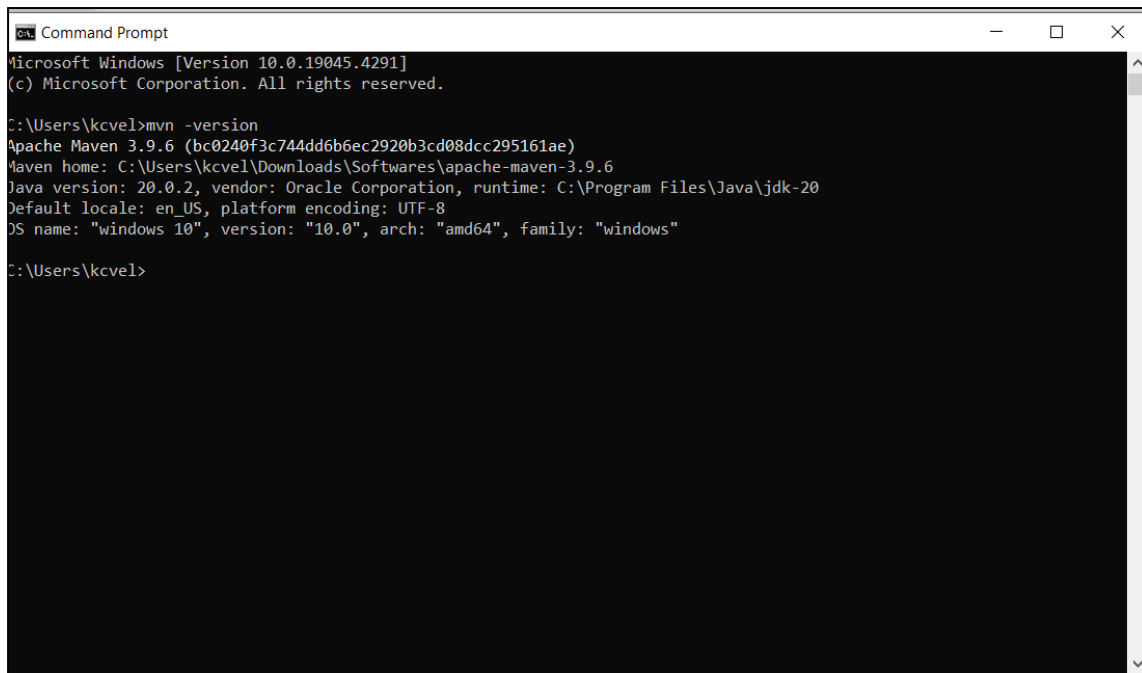
- Click on the "Environment Variables" button at the bottom
- If the 'JAVA_HOME' variable is not already set, Under system variables, click "New" to create a new variable. Then enter "Variable Name" as 'JAVA_HOME' and point "Variable Value" to path where the JDK is installed



- Under "System variables", select the "Path" variable. Click "Edit"
- In the "Edit environment variable", click "New" and add the path to Maven 'bin' directory.



- Click "OK" to close the "Edit environment variable" window.
- Click "OK" again to close the "Environment Variables" window.
- Click "OK" once more to close the "System Properties" window.
- To verify the installation, type 'mvn -version' in the command prompt and hit enter.



```
Command Prompt
Microsoft Windows [Version 10.0.19045.4291]
(c) Microsoft Corporation. All rights reserved.

C:\Users\kcvcl>mvn -version
Apache Maven 3.9.6 (bc0240f3c744dd6b6ec2920b3cd08dcc295161ae)
Maven home: C:\Users\kcvcl\Downloads\Softwares\apache-maven-3.9.6
Java version: 20.0.2, vendor: Oracle Corporation, runtime: C:\Program Files\Java\jdk-20
Default locale: en_US, platform encoding: UTF-8
OS name: "windows 10", version: "10.0", arch: "amd64", family: "windows"

C:\Users\kcvcl>
```

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 with a future date.

- Input: (50, 'grocery', '10-01-2024')
- Expected Outcome: System prompts an alert regarding date validity.

Test Case 5: 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.

6.2 Implementing the Test cases

For Use Case 03, "User Sign In":

Test ID	TC01
Purpose of Test	To verify that the 'signInUser' method correctly authenticates a user when provided with a valid username and password.
Test Environment	The test will be conducted using Junit as the testing framework within a Java Development Environment
Test Step	<ul style="list-style-type: none">- Configure the mockResultSet to return true when next() is called, simulating the presence of a user with valid credentials in the database.- Invoke the app.signInUser() method, which attempts to authenticate a user.- Check that the executeQuery() method on mockStatement is invoked exactly once to confirm that the database query is executed.- Ensure that next() on mockResultSet is called once, indicating that the method properly evaluates the query result.
Test Input	
Expected Result	The method should successfully authenticate the user
Likely Problems/Bugs Revealed	Inadequate handling of scenarios where next() returns false, indicating invalid credentials.

Test ID	TC02
Purpose of Test	It aims to verify the 'signInUser' method handles the case where a invalid username is provided.
Test Environment	The test will be conducted using Junit as the testing framework within a Java Development Environment
Test Step	<ul style="list-style-type: none">- Initialize mock objects for usernameField and passwordField, providing an existing username with an incorrect password.- Assign the mocked fields to the corresponding fields in the application.- Call the app.signInUser() method to attempt user authentication.- Verify that the appropriate error message is displayed if the username is invalid, as expected.
Test Input	The username field is set to "sarah1", representing an invalid username in the system.

	The password field is set to an incorrect password ("hello" as char[]).
Expected Result	The method should recognize the invalid username and display an error message indicating incorrect username or password.
Likely Problems/Bugs Revealed	If a valid username is given, the test case should be identify it is not invalid

Test ID	TC03
Purpose of Test	It aims to verify the 'signInUser' method handles the case where a valid username and incorrect password is entered
Test Environment	The test will be conducted using Junit as the testing framework within a Java Development Environment
Test Step	<ul style="list-style-type: none"> - Initialize mock objects for usernameField and passwordField, providing an existing username with a wrong password. - Assign the mocked fields to the corresponding fields in the application. - Call the app.signInUser() method to attempt user authentication. - Confirm that the method appropriately handles the scenario of an incorrect password.
Test Input	The username field is set to "sarah", representing an existing username in the system. The password field is set to an incorrect password ("towson" as char[]).
Expected Result	The method should recognize the wrong password and display an error message indicating incorrect username or password.
Likely Problems/Bugs Revealed	

Test ID	TC04
Purpose of Test	It aims to verify the 'signInUser' method handles the case where a valid username field is empty
Test Environment	The test will be conducted using Junit as the testing framework within a Java Development Environment
Test Step	<ul style="list-style-type: none"> - Initialize a mock object for usernameField, setting it to return an empty string. - Assign the mocked usernameField to the corresponding field in the application.

	<ul style="list-style-type: none"> - Call the <code>app.signInUser()</code> method to attempt user authentication. - Confirm that the method appropriately handles the scenario of an empty username field.
Test Input	The username field is set to "", representing an empty string
Expected Result	The method should detect the empty username field and display an error message indicating incorrect username or password.
Likely Problems/Bugs Revealed	

Test ID	TC05
Purpose of Test	It aims to verify the 'signInUser' method handles the case where a valid password field is empty
Test Environment	The test will be conducted using Junit as the testing framework within a Java Development Environment
Test Step	<ul style="list-style-type: none"> - Initialize a mock object for <code>passwordField</code>, setting it to return an empty string. - Assign the mocked <code>passwordField</code> to the corresponding field in the application. - Call the <code>app.signInUser()</code> method to attempt user authentication. - Confirm that the method appropriately handles the scenario of an empty username field.
Test Input	The password field is set to "", representing an empty string
Expected Result	The method should detect the empty password field and display an error message indicating incorrect username or password.
Likely Problems/Bugs Revealed	

For Use case 01, "Entering Expense Data"

Test ID	TC06
Purpose of Test	This test verifies that a valid expense entry is successfully added to the database when the user inputs correct details, as per the system's expense management requirements.
Test Environment	The test will be conducted using Junit as the testing framework within a Java Development Environment
Test Step	<ul style="list-style-type: none"> - Set the text fields for expense name, amount, and date with

	valid input values. <ul style="list-style-type: none"> - Simulate a button click on the "Add" button, which initiates the addition of the expense entry. - Verify that the application correctly interacts with the database by preparing and executing the SQL statement with the provided expense details.
Test Input	Expense name: "food" Expense amount: "10" Expense date: "2023-10-22"
Expected Result	The method should add the expense details into the database
Likely Problems/Bugs Revealed	

Test ID	TC07
Purpose of Test	The test ensures the method handles when a negative amount is entered for a transaction,
Test Environment	The test will be conducted using Junit as the testing framework within a Java Development Environment
Test Step	<ul style="list-style-type: none"> - Set the text fields for expense name, amount, and date with valid input values. - Simulate a button click on the "Add" button, which initiates the addition of the expense entry. - Confirm that no database operations were attempted, indicating that the application correctly detected and handled the invalid input.
Test Input	Expense name: "food" Expense amount: "-10" Expense date: "2023-10-22"
Expected Result	The method should detect it is an negative amount and should not add the expense details into the database
Likely Problems/Bugs Revealed	

Test ID	TC08
Purpose of Test	The test ensures the method handles when a non-numeric amount is given for a transaction,
Test Environment	The test will be conducted using Junit as the testing framework within a Java Development Environment

Test Step	<ul style="list-style-type: none"> - Set the text fields for expense name, amount, and date with valid input values. - Simulate a button click on the "Add" button, which initiates the addition of the expense entry. - Confirm that no database operations were attempted, indicating that the application correctly detected and handled the invalid input.
Test Input	Expense name: "food" Expense amount: "bills" Expense date: "2023-10-22"
Expected Result	The method should detect it is a non-numeric amount and should not add the expense details into the database
Likely Problems/Bugs Revealed	

Test ID	TC09
Purpose of Test	The test ensures the method handles when a future date is given for a transaction,
Test Environment	The test will be conducted using Junit as the testing framework within a Java Development Environment
Test Step	<ul style="list-style-type: none"> - Set the text fields for expense name, amount, and date with valid input values. - Simulate a button click on the "Add" button, which initiates the addition of the expense entry. - Confirm that no database operations were attempted, indicating that the application correctly detected and handled the invalid input.
Test Input	Expense name: "food" Expense amount: "bills" Expense date: "2025-10-22"
Expected Result	The method should detect it is a future date for the expense and should not add the expense details into the database
Likely Problems/Bugs Revealed	

Test ID	TC10
Purpose of Test	The test ensures the method handles when a invalid date is given

	for a transaction,
Test Environment	The test will be conducted using Junit as the testing framework within a Java Development Environment
Test Step	<ul style="list-style-type: none"> - Set the text fields for expense name, amount, and date with valid input values. - Simulate a button click on the "Add" button, which initiates the addition of the expense entry. - Confirm that no database operations were attempted, indicating that the application correctly detected and handled the invalid input.
Test Input	Expense name: "food" Expense amount: "10" Expense date: "10th January 2023"
Expected Result	The method should detect it is invalid date format for the expense and should not add the expense details into the database
Likely Problems/Bugs Revealed	

6.3 Test Documentation:

Bug	The test uncovered the bug	Description of the bug	Action was taken to fix the bug
Invalid Username Handling	TC002	The test case verifies the signInUser method's handling of an invalid username scenario. If a valid username is provided, the test incorrectly identifies it as invalid.	

APPENDIX

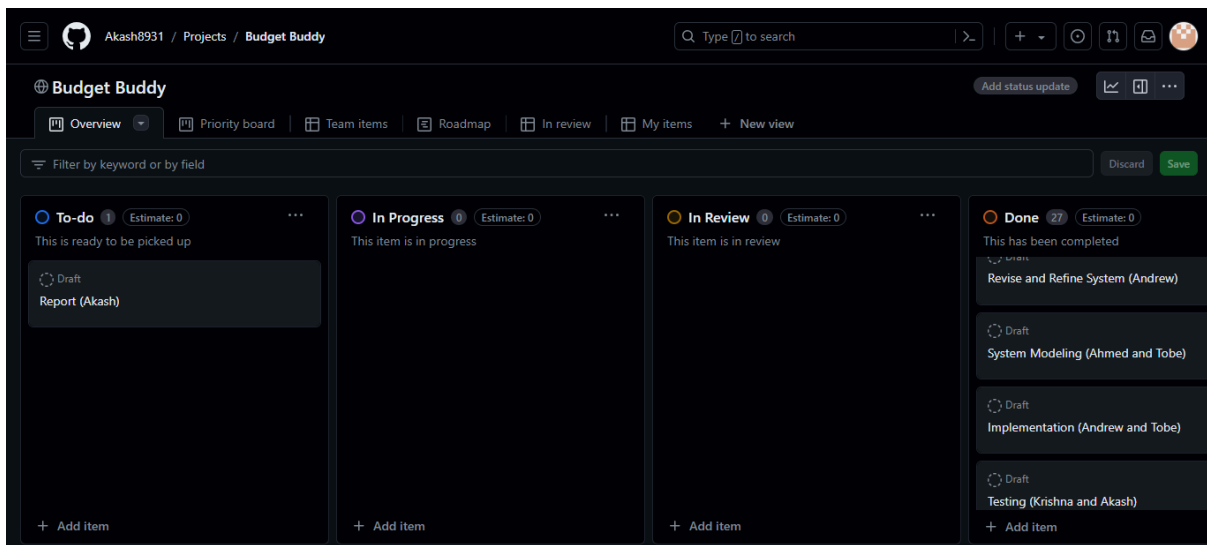


Fig: Screenshot of Kanban Board for A4