**FINANCIAL EXPENSE TRACKER**

**A MINI PROJECT REPORT**

**Submitted by**

**V** **HEMNATH**  **230701115**

**GOPIKRISHNAN L** **230701096**

In partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE

RAJALAKSHMI ENGINEERING COLLEGE (AUTONOMOUS)

THANDALAM

CHENNAI-602105

2023-2024

**BONAFIDE CERTIFICATE**

Certified that this project report “**FINANCIAL EXPENSE TRACKER**” is the bonafide work of **“V HEMNATH (230701115)**  **, GOPIKRISHNAN L**

**(230701096)”**

who carried out the project work under my supervision.

**Submitted for the Practical Examination held on \_\_\_\_\_\_\_\_\_\_**

**SIGNATURE** **SIGNATURE**

INTERNAL EXAMINER EXTERNAL EXAMIER

|  |
| --- |
| **ABSTRACT**:    The Financial Expense Tracker is a user-friendly application designed to help individuals or businesses manage their finances by tracking income and expenses efficiently. This system allows users to record, categorize, and analyze their spending patterns, providing a clear overview of their financial health. By storing transaction data in a MySQL database, the system ensures secure and organized record-keeping. Users can easily add, update, or delete expenses, categorize them into different spending types (e.g., food, utilities, transportation), and view detailed reports based on time periods or categories.  The application also features the ability to set budgets for various categories, helping users stay within their financial limits and track their progress over time. With its clean and intuitive user interface built in Java, the system makes it easy to navigate and access critical financial data. Detailed visualizations, such as pie charts or bar graphs, are provided to help users understand their spending habits and identify areas for improvement.  Additionally, the Financial Expense Tracker supports various filtering and search options, allowing users to quickly find specific transactions or view their spending summary. The system's ability to store historical data ensures that users can track their financial trends over time, making it easier to plan for future expenses and make informed financial decisions. Ultimately, this application serves as a practical tool for managing personal or business finances, improving budgeting accuracy, and fostering better financial planning. |

**TABLE OF CONTENTS**

**Chapter 1**

**1 INTRODUCTION**

1.1 INTRODUCTION ----------------------------------------------------6

1.2 OBJECTIVES ---------------------------------------------------------7

1.3 MODULES ------------------------------------------------------------7

**Chapter 2**

**2 SURVEY OF TECHNOLOGIES**

2.1 SOFTWARE DESCRIPTION ---------------------------------------9

2.2 LANGUAGES ---------------------------------------------------------9

2.2.1 SQL

2.2.2 JAVA

**Chapter 3**

**3 REQUIREMENTS AND ANALYSIS**

3.1 REQUIREMENT SPECIFICATION -----------------------------11

3.1.1 FUNCTIONAL REQUIREMENTS---------------------------11

3.1.2 NON FUNCTIONAL REQUIREMENTS-------------------12

3.2 HARDWARE AND SOFTWARE REQUIREMENTS ---------13

3.3 ARCHITECTURE DIAGRAM ------------------------------------14

3.4 ER DIAGRAM ------------------------------------------------------15

3.5 NORMALIZATION-------------------------------------------------16

**Chapter 4**

**4 PROGRAM CODE**

4.1 PROGRAM CODE--------------------------------------------------18 **Chapter 5**

**5 RESULTS AND DISCUSSION**

5.1 RESULTS AND DISCUSSION -------------------------------------27

|  |
| --- |
| **Chapter 6**   1. **CONCLUSION**     6.1 CONCLUSION----------------------------------------------------31  **Chapter 7**   1. **REFERENCES**                                               5 |

**Chapter 1 INTRODUCTION**

**1.1 INTRODUCTION**

Effective financial management is crucial in today’s fast-paced world, whether for personal budgeting, household expenses, or business operations. Tracking income and expenditures accurately helps individuals and organizations stay within budget, avoid unnecessary debt, and plan for the future. The *Financial Expense Tracker* is designed to streamline the process of managing finances by providing users with a comprehensive tool to record, categorize, and analyze their financial activities. This system simplifies the often complex task of budgeting and tracking expenses, ensuring that users have a clear and organized view of their financial health.

The Financial Expense Tracker allows users to efficiently log and manage various types of transactions, from daily expenditures to larger one-off payments. By storing all financial data in a MySQL database, the system ensures secure and reliable storage, while also offering the flexibility to update and retrieve information as needed. The application provides intuitive features such as budgeting tools, expense categorization, and customizable reports to help users track their spending patterns, set financial goals, and improve their budgeting accuracy.

Developed using Java for the application logic and MySQL for data storage, the Financial Expense Tracker combines robust backend functionality with a user-friendly interface. MySQL serves as the foundation for the system’s database, offering efficient storage and retrieval of financial records. Java provides the flexibility to implement advanced features such as transaction management, data analysis, and reporting. This combination of technologies results in a powerful and scalable tool that can be used for personal finance tracking as well as small business financial management.

This report will explore the development process of the Financial Expense Tracker, the technologies used, and the features implemented to meet the needs of modern financial management. The objective is to demonstrate how this system provides users with a comprehensive and efficient solution for tracking and controlling their finances, ultimately leading to improved financial decision-making and better fiscal responsibility.

**Chapter 3 REQUIREMENTS AND ANALYSIS**

**3.1 REQUIREMENT SPECIFICATION**

**3.1.1 Functional Requirements**

**3.1.1 FUNCTIONAL REQUIREMENTS**

The functional requirements describe the specific functions that the **Expense Tracker Application** must perform to fulfill user needs. These include:

1. **User Authentication**:
   * The system must allow users to log in with a username and password.
   * Upon successful login, the system should redirect the user to the **Dashboard**.
2. **Add Expense**:
   * The system must allow users to input a new expense by specifying the amount.
   * The system should store the expense data in a **MySQL** database.
3. **View Total Expenses**:
   * The system must calculate and display the **total expenses** entered by the user.
   * The system should fetch and display the sum of all stored expenses from the database.
4. **Calculate Monthly Savings**:
   * The system should allow users to input a **product price** and **number of months** to save.
   * The system must calculate how much the user needs to save each month to reach the product price, considering their current total expenses.
   * The system should display the result in a **dialog box**.
5. **Database Interaction**:
   * The system must be able to store and retrieve user data (expenses) from a **MySQL database** using **JDBC**.
6. **Error Handling**:
   * The system should provide **error messages** for invalid user inputs (e.g., entering non-numeric values for expenses or product prices).
   * The system should handle **database connection errors** gracefully.
7. **Data Persistence**:
   * Expenses and other relevant data should persist across application sessions by storing them in the **MySQL** database.

**3.1.2 Non-Functional Requirements**

Non-functional requirements define the performance, usability, and other quality attributes of the **Expense Tracker Application**:

1. **Performance**:
   * The application should handle the entry and retrieval of **thousands of expenses** without significant delays.
   * The monthly savings calculation should return results within **2-3 seconds**.
2. **Usability**:
   * The **user interface** should be **intuitive** and easy to use, with clear labels and buttons.
   * The application should allow users to enter data and view results with minimal effort.
3. **Reliability**:
   * The application should be stable and handle edge cases, such as invalid input or database connection failures, without crashing.
4. **Security**:
   * Since the current version does not include advanced security features, at the minimum, user authentication should be **secure**.
   * **Data privacy** should be ensured (even though the application is local).
5. **Maintainability**:
   * The code should be modular, allowing for easy updates and enhancements, such as adding more features (e.g., expense categories or reporting).
6. **Compatibility**:
   * The application should run on all platforms where **Java** is supported, including **Windows**, **macOS**, and **Linux**.
7. **Portability**:
   * The application should be portable, meaning that it can be easily transferred to different systems or environments (e.g., running it on different machines with MySQL).

**3.2 HARDWARE AND SOFTWARE REQUIREMENTS**

**Hardware Requirements:**

1. **Processor**: Any modern **Intel** or **AMD** processor (recommended **2 GHz** or higher).
2. **RAM**: At least **2 GB** of RAM for optimal performance.
3. **Hard Disk**: **50 MB** of available space for the project files and database.
4. **Display**: A minimum screen resolution of **1024x768** for a good UI experience.

**Software Requirements:**

1. **Operating System**: Compatible with **Windows**, **macOS**, or **Linux**.
2. **Java Development Kit (JDK)**: Version **8** or higher, required to run the Java application.
3. **IDE**: Recommended Integrated Development Environments (**IDE**) such as **Eclipse**, **IntelliJ IDEA**, or **NetBeans** for development.
4. **Database**: **MySQL** (with a version that supports **JDBC** connectivity).
5. **JDBC Connector**: **MySQL JDBC Driver** (for Java-MySQL interaction).
6. **Swing**: **Java Swing** (for building the graphical user interface).

3.3 ARCHITECTURE DIAGRAM

The architecture diagram represents the components of the system and how they interact with each other.

Here's the description of the components involved:

1. **Client (User Interface)**:
   * The user interacts with the system through the **Swing-based GUI**, where they input expenses, view totals, and calculate savings.
2. **Business Logic Layer**:
   * The **ExpenseTracker** class handles the core logic, such as adding expenses, calculating total expenses, and determining monthly savings.
   * It also validates user input and interacts with the **database** through **JDBC**.
3. **Database Layer**:
   * **MySQL Database**: Stores the expense data (amounts, timestamps).
   * The **ExpenseDatabaseHandler** manages the interaction between the business logic and the database.

+---------------------------------+

| User Interface |

| (Swing GUI for user input) |

+---------------------------------+

|

v

+---------------------------------+ +-------------------------------+

| Business Logic Layer |----->| MySQL Database (Expense Data) |

| (ExpenseTracker.java, Logic) | | (expenses table) |

+---------------------------------+ +-------------------------------+

|

v

+-----------------------------+

| Database Connection |

| (JDBC) |

+-----------------------------+

3.4 ER DIAGRAM

+-----------------------+

| Expense |

|-----------------------|

| id (PK) |

| amount |

| date |

+-----------------------+

**3.5 NORMALISATION**

Normalization is the process of organizing data in a database to reduce redundancy and improve data integrity. In this case, the database is already in **1NF (First Normal Form)**, as each table contains atomic (indivisible) values.

* **1NF**: The table has a primary key (id), and all columns contain atomic values. For example, the amount field only contains numeric values, and date contains a single date timestamp for each record.

Since this project is fairly simple, there's no need to go beyond **1NF** unless you add more features (e.g., categorizing expenses, tracking users, etc.). However, if you were to expand the database, you'd want to apply **2NF** and **3NF** to eliminate any transitive dependencies and ensure all non-key attributes depend only on the primary key.

**For example**:

* **2NF**: If we had multiple categories for expenses (e.g., food, travel), we might need a **Category** table and relate it to the **Expense** table via a foreign key.
* **3NF**: If the database had more complex relationships (e.g., tracking user details), we would eliminate transitive dependencies by separating data into multiple tables.

PROGRAM

**PROGRAM 1**

The code you've shared is for a **Java-based desktop application** that functions as an Expense Tracker. It allows users to manage their expenses and calculate savings for specific goals using a **Graphical User Interface (GUI)** built with **Java Swing**. Let me explain the components and functionality of your program step by step.

**Overview of the Program:**

Your program is essentially an **Expense Tracker** that:

1. Allows users to add expenses.
2. Displays the total of all expenses.
3. Helps the user calculate how much they need to save each month to reach a specific financial goal (product price) within a certain number of months.

The **GUI** is built using **Java Swing**, and the application includes basic **event handling** to respond to user actions like button clicks. The program interacts with an **ExpenseTracker class** (not shown here) that handles the business logic of adding expenses, retrieving total expenses, and calculating monthly savings.

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

public class Dashboard {

public Dashboard() {

JFrame dashboardFrame = new JFrame("Dashboard");

dashboardFrame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

dashboardFrame.setSize(400, 300);

dashboardFrame.setLayout(new FlowLayout());

JButton addExpenseButton = new JButton("Add Expense");

JButton viewExpensesButton = new JButton("View Total Expenses");

JButton calculateSavingsButton = new JButton("Calculate Savings");

// Adding buttons to the frame

dashboardFrame.add(addExpenseButton);

dashboardFrame.add(viewExpensesButton);

dashboardFrame.add(calculateSavingsButton);

// Action listeners for the buttons

addExpenseButton.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

String input = JOptionPane.showInputDialog("Enter Expense Amount:");

if (input != null) {

try {

double amount = Double.parseDouble(input);

ExpenseTracker.addExpense(amount);

} catch (NumberFormatException ex) {

JOptionPane.showMessageDialog(null, "Please enter a valid number.");

}

}

}

});

viewExpensesButton.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

double totalExpenses = ExpenseTracker.getTotalExpenses();

JOptionPane.showMessageDialog(null, "Total Expenses: " + totalExpenses);

}

});

calculateSavingsButton.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

try {

String productPriceInput = JOptionPane.showInputDialog("Enter Product Price:");

String monthsInput = JOptionPane.showInputDialog("Enter Months to Save:");

if (productPriceInput != null && monthsInput != null) {

double productPrice = Double.parseDouble(productPriceInput);

int months = Integer.parseInt(monthsInput);

double totalExpenses = ExpenseTracker.getTotalExpenses();

double monthlySavings = ExpenseTracker.calculateSavings(productPrice, totalExpenses, months);

JOptionPane.showMessageDialog(null, "You need to save " + monthlySavings + " each month.");

}

} catch (NumberFormatException ex) {

JOptionPane.showMessageDialog(null, "Please enter valid numbers.");

}

}

});

dashboardFrame.setVisible(true);

}

PROGRAM 2

The code you provided is part of the **Expense Tracker** program, specifically the **database interaction** part, where the program interacts with a **MySQL** database to store and retrieve expenses. The **ExpenseTracker** class manages the logic for connecting to the database, adding expenses, calculating total expenses, and calculating the savings needed for a user-defined goal.

import java.sql.\*;

import javax.swing.\*;

public class Expensetracker {

// MySQL database connection details

private static final String DB\_URL = "jdbc:mysql://127.0.0.1:33060/xpensedb";

private static final String USER = "root"; // Your MySQL username

private static final String PASSWORD = "Gopi@2005"; // Your MySQL password

// Method to connect to the database

private static Connection connect() throws SQLException {

return DriverManager.getConnection(DB\_URL, USER, PASSWORD);

}

// Method to add an expense to the database

public static void addExpense(double amount) {

String query = "INSERT INTO expenses (amount) VALUES (?)";

try (Connection conn = connect(); PreparedStatement stmt = conn.prepareStatement(query)) {

stmt.setDouble(1, amount);

stmt.executeUpdate();

JOptionPane.showMessageDialog(null, "Expense added successfully!");

} catch (SQLException e) {

JOptionPane.showMessageDialog(null, "Error adding expense: " + e.getMessage());

}

}

// Method to get total expenses from the database

public static double getTotalExpenses() {

double total = 0;

String query = "SELECT SUM(amount) FROM expenses";

try (Connection conn = connect(); Statement stmt = conn.createStatement(); ResultSet rs = stmt.executeQuery(query)) {

if (rs.next()) {

total = rs.getDouble(1);

}

} catch (SQLException e) {

JOptionPane.showMessageDialog(null, "Error fetching total expenses: " + e.getMessage());

}

return total;

}

// Method to calculate monthly savings based on product price and expenses

public static double calculateSavings(double productPrice, double totalExpenses, int months) {

if (months <= 0) {

return 0;

}

return (productPrice - totalExpenses) / months;

}

}

PROGRAM 3

The program you have provided is a **Login Page** for a Java application using **Swing** for the graphical user interface (GUI). The purpose of this program is to prompt users for a username and password, validate the input, and, based on the validity of the credentials, either allow access to the next page (a dashboard in this case) or show an error message.

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

public class LoginPage {

public LoginPage() {

JFrame loginFrame = new JFrame("Login");

loginFrame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

loginFrame.setSize(300, 150);

loginFrame.setLayout(new FlowLayout());

JLabel userLabel = new JLabel("Username:");

JTextField userField = new JTextField(10);

JLabel passLabel = new JLabel("Password:");

JPasswordField passField = new JPasswordField(10);

JButton loginButton = new JButton("Login");

loginFrame.add(userLabel);

loginFrame.add(userField);

loginFrame.add(passLabel);

loginFrame.add(passField);

loginFrame.add(loginButton);

// Action listener for the login loginButton.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

String username = userField.getText();

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

// Simple authentication check (username: user, password: password)

if (username.equals("user") && password.equals("password")) {

JOptionPane.showMessageDialog(loginFrame, "Login successful!");

loginFrame.dispose(); // Close login window

new Dashboard(); // Open dashboard

} else {

JOptionPane.showMessageDialog(loginFrame, "Invalid username or password.");

}

}

});

loginFrame.setVisible(true); }}

CHAPTER 5 CONCLUSION

The Expense Tracker project provides a simple yet effective way for users to manage and track their personal expenses. By combining a Java Swing user interface with backend logic for handling expenses and savings calculations, this application helps users stay informed about their spending habits and plan for their future savings goals.

Throughout the development process, key functionalities were implemented, including:

•User Authentication: A secure login page to ensure that only authorized users can access their data.

•Expense Management: Users can easily add new expenses, view total expenses, and calculate monthly savings needed to achieve their financial goals.

•Savings Calculation: The dashboard allows users to input their target product price and months to save, and it calculates how much they need to save each month, factoring in their current total expenses.

By using JDBC and MySQL (or an in-memory list for simplicity in this version), the application tracks the total expenses and helps users make informed decisions on how to save for future purchases.

Challenges and Learning Points:

During the development of this project, several key learning points and challenges were encountered:

1.Database Integration: Initially, expenses were stored in memory. This project can be extended by integrating a database (e.g., MySQL) to persist user data across sessions.

2.User Authentication: Simple username and password authentication were implemented. However, for real-world applications, password hashing (e.g., with bcrypt) and session management could be added to enhance security.

3.UI Design: While the user interface is functional, it is basic. Future improvements could include more polished designs and features like dropdowns for expense categories or charts to visualize spending patterns.

Future Enhancements:

There are several directions in which this project can be extended:

•Data Persistence: Integrating a database like MySQL for persistent storage of user credentials, expenses, and transaction history.

•Advanced Features: Implementing advanced features such as expense categorization, detailed reports, and monthly budgeting tools.

•User Interface Improvement: Enhancing the graphical user interface with more modern UI frameworks like JavaFX or even transitioning to a web-based interface using Spring Boot and React.

•Data Export: Adding functionality to export expenses to CSV or PDF for better reporting.

Final Thoughts:

Overall, this project demonstrates a basic but practical approach to personal finance management. It provides a solid foundation for further development and can be adapted into a full-featured financial planning tool with the addition of more advanced features. The skills gained from working on this project, such as Java programming, UI development with Swing, and basic database handling, are valuable and can be applied to more complex financial applications in the future.CHAPTER 6 REFERENCES

* 1. <https://youtube.com/playlist?list=PLLGI5phu9E4682ZewgdG9A52N8YRHdP-G&si=4DoB3XqLmGVPp2wB>
  2. <https://youtu.be/_pLLZE_jg3s?si=Tm4ghHfGrGh1BMLb>

**Mr Ragu,Assistant Professor, Computer Science and Engineering, Rajalakshmi Engineering College, Thandalam, Chennai 602105 Mrs Divya M,Assistant Professor, Computer Science and Engineering, Rajalakshmi Engineering College, Thandalam, Chennai 602105**