Personal Budget Tracker

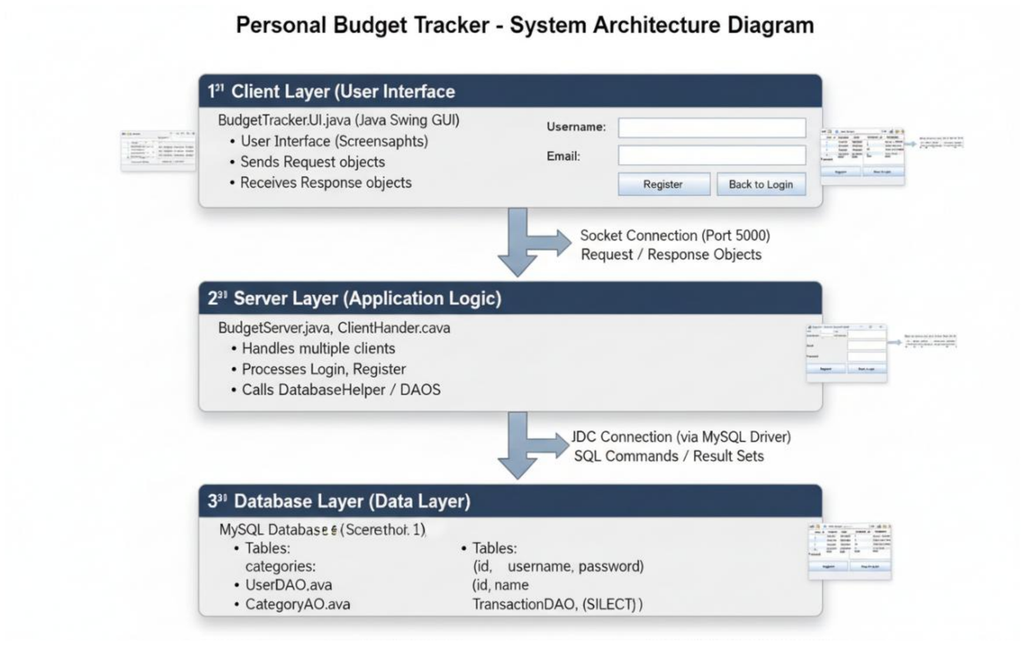
Introduction:

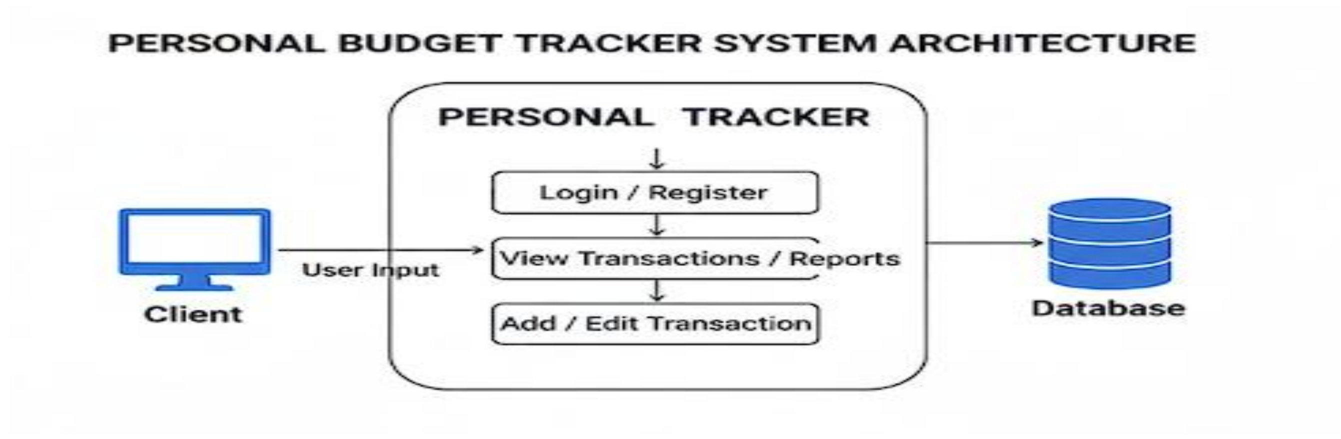
The Personal Budget Tracker is a Java-based desktop application designed to assist users in efficiently managing their daily financial activities. In today's fast-paced digital era, individuals often find it difficult to maintain a clear record of their income and expenses. This system provides a structured, secure, and user-friendly environment for tracking personal finances and ensuring better money management. Built using Java Swing for the user interface and MySQL for database management, the application offers real-time synchronization of data through a client-server model, ensuring reliability and consistency across sessions. Users can seamlessly log their income and expenditures, organize transactions by category, and generate clear financial summaries to understand their spending habits. The system enhances financial awareness and helps users plan budgets effectively. It promotes accountability and empowers individuals to make data-driven decisions regarding their expenses. In addition, the architecture is designed for scalability—future upgrades may include AI-powered analytics, predictive expense trends, automated reminders, and personalized financial recommendations, making it a powerful step toward smart financial management.

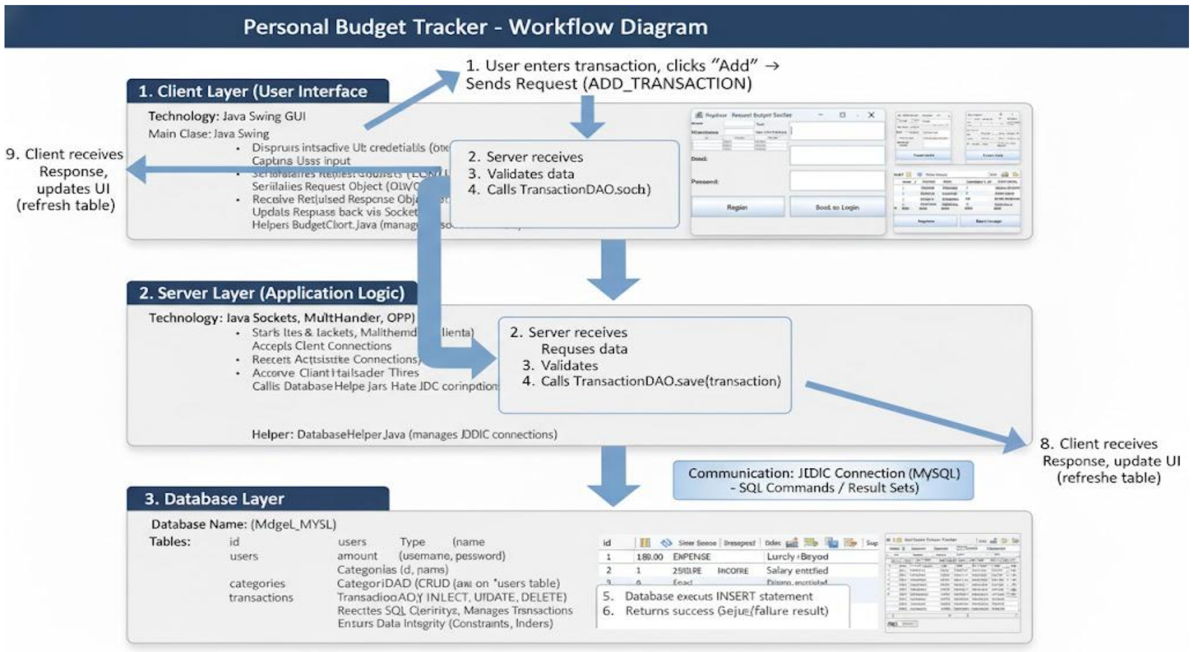
Project Description:

The **Personal Budget Tracker** allows users to register and log into a secure account to manage their financial transactions. The server handles database communication, while the client provides an interactive interface. Each transaction contains details such as amount, type (income/expense), category, date, and description. The user can add, view, and delete transactions. The system uses **MySQL** as its backend database and **JDBC** for connectivity. A **multithreaded server** ensures that multiple users can connect simultaneously without data conflict. This project demonstrates full-stack Java development, combining GUI, database management, and networking. Future enhancements include adding analytics, budget forecasting, and mobile integration.

System Architecture Diagram:


Personal Budget Tracker - System Architecture Diagram
1st Client Layer (User Interface
BudgetTrackerUI.java (Java Swing GUI)
• User Interface (Screensaphts)
• Sends Request objects
• Receives Response objects
Username:
Email:
Register
Back to Login
Socket Connection (Port 5000)
Request / Response Objects
2nd Server Layer (Application Logic)
BudgetServer.java, ClientHander.cava
• Handles multiple clients
• Processes Login, Register
• Calls DatabaseHelper / DAOS
JDC Connection (via MySQL Driver)
SQL Commands / Result Sets
3rd Database Layer (Data Layer)
MySQL Databases (Scerethot. 1)
• Tables:
categories:
• UserDAO.ava
• CategoryAO.ava
• Tables:
(id, username, password)
(id, name
TransactionDAO, (SILECT))




Personal Budget Tracker - Workflow Diagram
1. User enters transaction, clicks "Add" →
Sends Request (ADD_TRANSACTION)
1. Client Layer (User Interface
Technology: Java Swing GUI
Main Clase: Java Swing
9. Client receives Response, updates UI (refresh table)
2. Server receives
3. Validates data
4. Calls TransactionDAO.soch)
Register
Back to Login
2. Server Layer (Application Logic)
Technology: Iava Sockets, MultHandler, OPP)
Accepls Clent Connections
Calls DatabaseHelpe Jars Hate JDC corinption
2. Server receives
Requses data
3. Validates
4. Calls TransactionDAO.save(transaction)
Helper: DatabaseHelper.Java (manages JDDIC connections)
8. Client receives Response, update UI (refreshe table)
Communication: JEDIC Connection (MySQL) - SQL Commands / Result Sets)
3. Database Layer
Database Name: (Mdgel_MYSL)
Tables:
users
categories
transactions
5. Database execus INSERT statement
6. Returns success Gejue(failure result)

SOURCE CODE

1. Main Launcher

```
package client;

import client.ui.LoginUI;

import javax.swing.*;

public class MainLauncher {

    public static void main(String[] args) {

        try {

            // Initialize client and connect to the server

            BudgetClient client = new BudgetClient("127.0.0.1", 5000);

            client.connect();

            System.out.println("Connected to server successfully.");

            // Launch login window

            SwingUtilities.invokeLater(() -> {

                new LoginUI(client).setVisible(true);

            });

        } catch (Exception e) {

            e.printStackTrace();

            JOptionPane.showMessageDialog(null,

                    "Could not connect to server: " + e.getMessage());

        }

    }

}
```

2. Database Connection

```
package database;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

public class DBConnection {

        private static final String URL = "jdbc:mysql://localhost:3306/budget_tracker";

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

        private static final String PASSWORD = "";

    public static Connection getConnection() throws SQLException {

        return DriverManager.getConnection(URL, USER, PASSWORD);

    }

}
```

---

3. Transaction DAO

```
package database;

import model.Transaction;

import java.sql.*;

import java.util.ArrayList;

import java.util.List;

public class TransactionDAO {

    // Get all transactions for a user

    public List<Transaction> getAllByUser(int userId) {

        List<Transaction> list = new ArrayList<>();
        String sql = "SELECT id, user_id, amount, type, category_id, description, date FROM
transactions WHERE user_id=?";
        try (Connection conn = DBConnection.getConnection();
             PreparedStatement ps = conn.prepareStatement(sql)) {
            ps.setInt(1, userId);
            ResultSet rs = ps.executeQuery();
            while (rs.next()) {
                Transaction t = new Transaction(
                        rs.getInt("id"),
                        rs.getInt("user_id"),
                        rs.getDouble("amount"),
                        rs.getString("type"),
                        rs.getInt("category_id"),
                        rs.getString("description"),
                        rs.getDate("date")
                );
                list.add(t);
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return list;
    }
    // Insert a new transaction
    public boolean insert(Transaction t) {
        String sql = "INSERT INTO transactions (user_id, amount, type, category_id, description,
date) VALUES (?, ?, ?, ?, ?, ?)";
        try (Connection conn = DBConnection.getConnection();
             PreparedStatement ps = conn.prepareStatement(sql)) {
            ps.setInt(1, t.getUserId());
            ps.setDouble(2, t.getAmount());
            ps.setString(3, t.getType());
            ps.setInt(4, t.getCategoryId());
            ps.setString(5, t.getDescription());
            java.util.Date dt = t.getDate();
            if (dt != null) {
                ps.setDate(6, new java.sql.Date(dt.getTime()));
            } else {
                ps.setNull(6, Types.DATE);
            }
            return ps.executeUpdate() > 0;
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return false;
    }
    // Delete transaction by ID
    public boolean delete(int id) {
        String sql = "DELETE FROM transactions WHERE id=?";
        try (Connection conn = DBConnection.getConnection();
             PreparedStatement ps = conn.prepareStatement(sql)) {
            ps.setInt(1, id);
            return ps.executeUpdate() > 0;
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return false;
    }
}
```

---

4. BudgetServer

```
package server;
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
/**
 * Multi-threaded server that accepts client connections.
 */
public class BudgetServer {
    private static final int PORT = 5000;
    private ServerSocket serverSocket;
    private ExecutorService pool = Executors.newCachedThreadPool();
    public void start() {
        try {
            serverSocket = new ServerSocket(PORT);
            System.out.println("BudgetServer started on port " + PORT);
            while (true) {
                Socket clientSocket = serverSocket.accept();
                System.out.println("Accepted connection: " +
clientSocket.getRemoteSocketAddress());
                pool.submit(new ClientHandler(clientSocket));
            }
        } catch (IOException ex) {
            ex.printStackTrace();
        } finally {
            stop();
        }
    }
    public void stop() {
        try {
            if (serverSocket != null) serverSocket.close();
            pool.shutdown();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    public static void main(String[] args) {
        new BudgetServer().start();
    }
}
```

---

5. BudgetTrackerUI

```
package client;
import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import java.awt.*;
import java.awt.event.*;
import java.sql.*;
import java.time.LocalDate;
import java.util.Vector;
import client.BudgetClient;
import database.DBConnection;
public class BudgetTrackerUI extends JFrame {
    private final BudgetClient client;
    private JTable table;
    private JTextField amountField, descriptionField, dateField;
    private JComboBox<String> typeBox, categoryBox;
    public BudgetTrackerUI(BudgetClient client) {
        this.client = client;
        setTitle("Personal Budget Tracker");
        setSize(800, 500);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLocationRelativeTo(null);
        initUI();
        loadTransactions();
     }
    private void initUI() {
        JPanel panel = new JPanel(new BorderLayout());
        JPanel inputPanel = new JPanel(new GridLayout(2, 6, 10, 10));
        amountField = new JTextField();
        descriptionField = new JTextField();
        dateField = new JTextField(LocalDate.now().toString());
        typeBox = new JComboBox<>(new String[]{"Expense", "Income"});
        categoryBox = new JComboBox<>();
        JButton addBtn = new JButton("Add Transaction");
        // Load category names from DB
        loadCategories()
        inputPanel.add(new JLabel("Amount:"));
        inputPanel.add(amountField);
        inputPanel.add(new JLabel("Type:"));
        inputPanel.add(typeBox);
        inputPanel.add(new JLabel("Category:"));
        inputPanel.add(categoryBox);
        inputPanel.add(new JLabel("Description:"));
        inputPanel.add(descriptionField);
        inputPanel.add(new JLabel("Date (YYYY-MM-DD):"));
        inputPanel.add(dateField);
        inputPanel.add(new JLabel(""));
        inputPanel.add(addBtn);
        // Table setup
        table = new JTable(new DefaultTableModel(
            new String[]{"ID", "Amount", "Type", "Category", "Description", "Date"}, 0));
        JScrollPane scrollPane = new JScrollPane(table);
        panel.add(inputPanel, BorderLayout.NORTH);
        panel.add(scrollPane, BorderLayout.CENTER);
        add(panel);
        // Button Action
        addBtn.addActionListener(e -> addTransaction());
    }
    private void loadCategories() {
        try (Connection conn = DBConnection.getConnection();
             PreparedStatement ps = conn.prepareStatement("SELECT name FROM categories"))
  {
            ResultSet rs = ps.executeQuery();
            while (rs.next()) {
                categoryBox.addItem(rs.getString("name"));
            }
        } catch (Exception e) {
            e.printStackTrace();
            JOptionPane.showMessageDialog(this, "Error loading categories: " + e.getMessage());
    }
    }
    private void addTransaction() {
        try (Connection conn = DBConnection.getConnection()) {
            String sql = "INSERT INTO transactions (user_id, amount, type, category_id, description, date) " +  "VALUES (?, ?, ?, (SELECT id FROM categories WHERE name=?), ?, ?)";
            PreparedStatement ps = conn.prepareStatement(sql);
            ps.setInt(1, 1); // default user id
            ps.setDouble(2, Double.parseDouble(amountField.getText()));
            ps.setString(3, (String) typeBox.getSelectedItem());
            ps.setString(4, (String) categoryBox.getSelectedItem());
            ps.setString(5, descriptionField.getText());
            ps.setDate(6, java.sql.Date.valueOf(dateField.getText()));
            ps.executeUpdate();
            JOptionPane.showMessageDialog(this, "Transaction added successfully!");
            loadTransactions();
            amountField.setText("");
            descriptionField.setText("");
        } catch (Exception e) {
            e.printStackTrace();
            JOptionPane.showMessageDialog(this, "Error adding transaction: " + e.getMessage());
        }
    }
    private void loadTransactions() {
        DefaultTableModel model = (DefaultTableModel) table.getModel();
        model.setRowCount(0);
        try (Connection conn = DBConnection.getConnection();
             Statement st = conn.createStatement();
             ResultSet rs = st.executeQuery(
                     "SELECT t.id, t.amount, t.type, c.name AS category, t.description, t.date " +
                     "FROM transactions t LEFT JOIN categories c ON t.category_id = c.id")) {
            while (rs.next()) {
                Vector<Object> row = new Vector<>();
                row.add(rs.getInt("id"));
                row.add(rs.getDouble("amount"));
                row.add(rs.getString("type"));
                row.add(rs.getString("category"));
                row.add(rs.getString("description"));
                row.add(rs.getDate("date"));
                model.addRow(row);
            }
        } catch (Exception e) {
            e.printStackTrace();
            JOptionPane.showMessageDialog(this, "Error loading transactions: " +
e.getMessage());
        }
    }
}
```

MODULAR EXPLANATION: Personal Budget Tracker

1. Login Module

The login module serves as the secure entry point to the Personal Budget Tracker system. Users must enter their registered email or username along with a password to access their personal dashboard. The entered credentials are verified against records stored in the database. If the details match, the user is granted access; otherwise, an appropriate error message is displayed. This module ensures data privacy and prevents unauthorized users from accessing sensitive financial information.

2. Register (User Signup) Module

The register module allows new users to create an account in the system. During registration, the user provides basic details such as name, email, username, and password. The module validates the inputs (e.g., password strength, unique username, and valid email format) before storing them in the database. Once registration is successful, the user can log in using the provided credentials. This module ensures smooth onboarding and securely stores user credentials for future authentication.

3. Dashboard Module

The dashboard module is the central interface of the Personal Budget Tracker. After login, users are directed to the dashboard, which displays an overview of their financial activities such as total income, total expenses, savings summary, and monthly spending charts. The interface uses data visualization (charts, graphs, and tables) for easy understanding of financial health. Users can navigate to different sections like adding transactions or viewing reports directly from the dashboard. This module improves user engagement and helps track overall financial performance.

4. Add Transaction Module

The add transaction module enables users to record their financial activities by adding details of income or expenses. The user inputs fields such as transaction name, amount, category (e.g., food, rent, salary), date, and payment mode. The module validates that the entered amount is numeric and positive. Once submitted, the transaction is stored in the database and immediately reflected on the dashboard. This module ensures accurate recording of financial data and helps maintain a clear spending record.

---

5. View Transactions Module

The view transactions module allows users to review all their past transactions in a structured and organized table format. Each record displays the transaction ID, type (income or expense), category, amount, and date. Users can filter or sort transactions by category or time period for better analysis. This feature helps users track their spending patterns and identify unnecessary expenses. The table is dynamically updated from the database to ensure real-time accuracy.

---

6. Database Connection Module

The database connection module establishes a connection between the Java application and the MySQL database using JDBC. It loads the MySQL driver and connects using the proper credentials. This module handles all CRUD (Create, Read, Update, Delete) operations related to users, transactions, and categories. If the connection fails, an error message is displayed, ensuring that no data corruption occurs. It serves as the backbone for data storage and retrieval in the entire application.

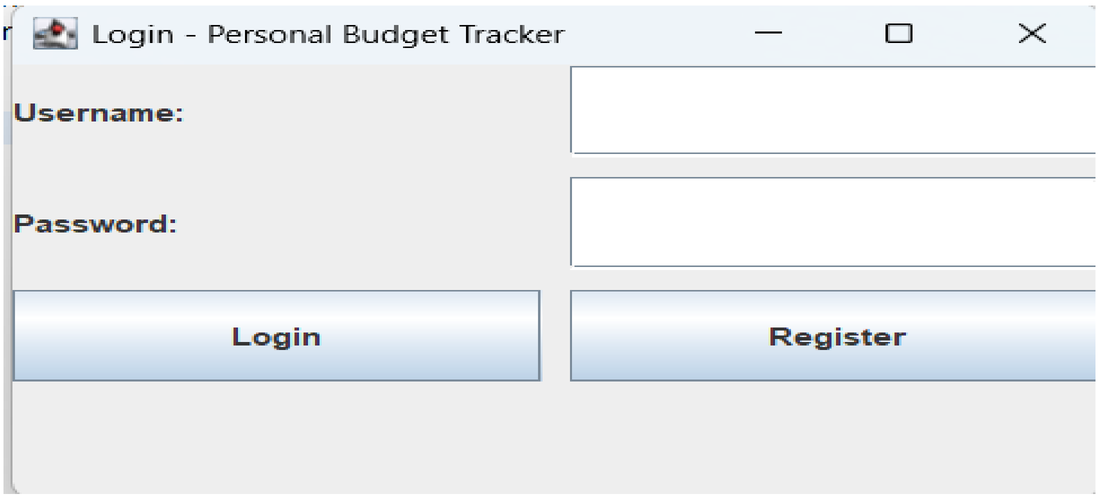
---

7. Database View Module

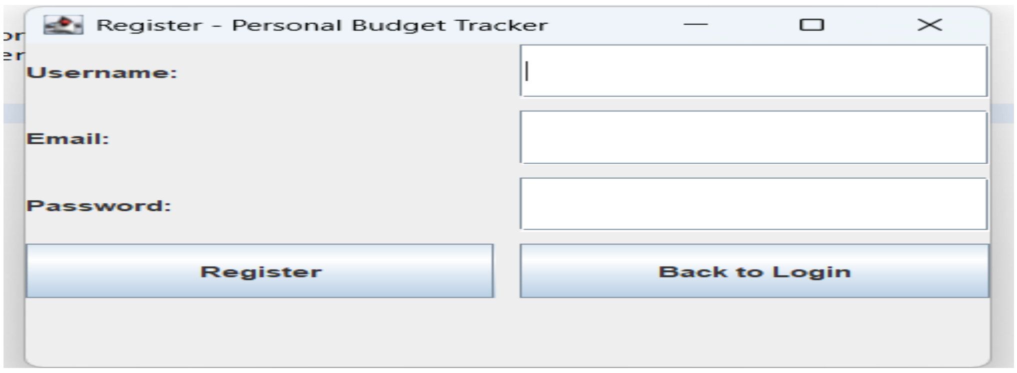
The database view module allows administrators or developers to directly view all stored records in the database for validation and debugging purposes. It includes tables for user information, transactions, income/expense categories, and savings data. This module ensures data transparency and helps verify that all operations—such as login, registration, and transactions—are correctly reflected in the database. It supports maintaining data integrity and consistency across the system.

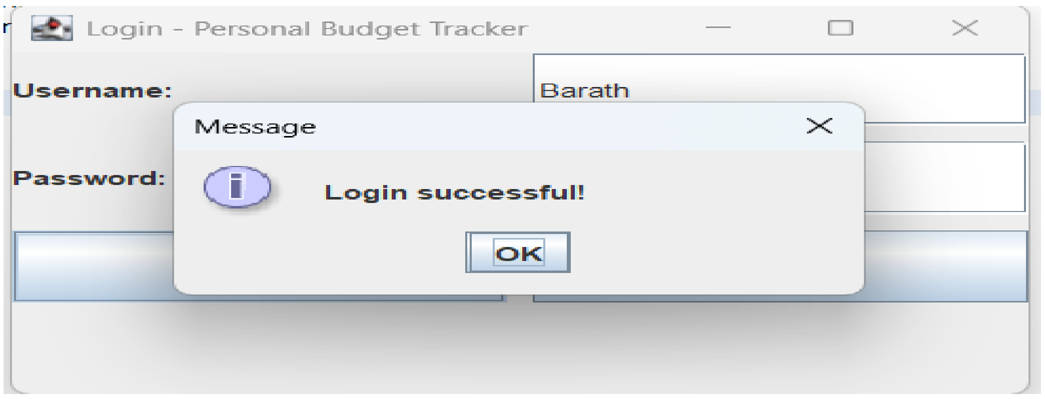
Screenshots :

1. **Login Page**



2. **Register Page:**





Login - Personal Budget Tracker

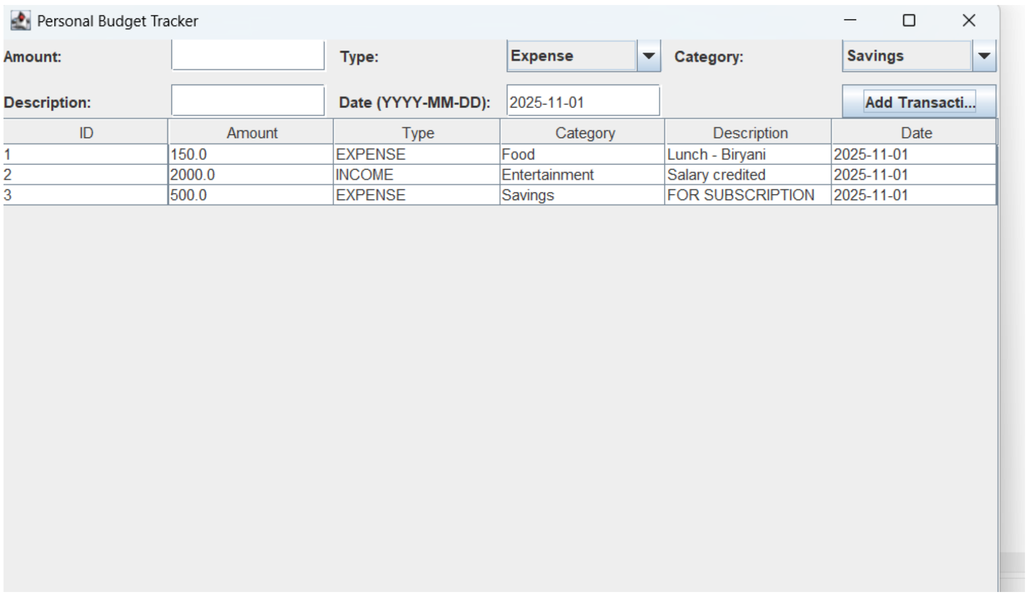
Username: Barath

Password:

Message

Login successful!

3. Dashboard Interface



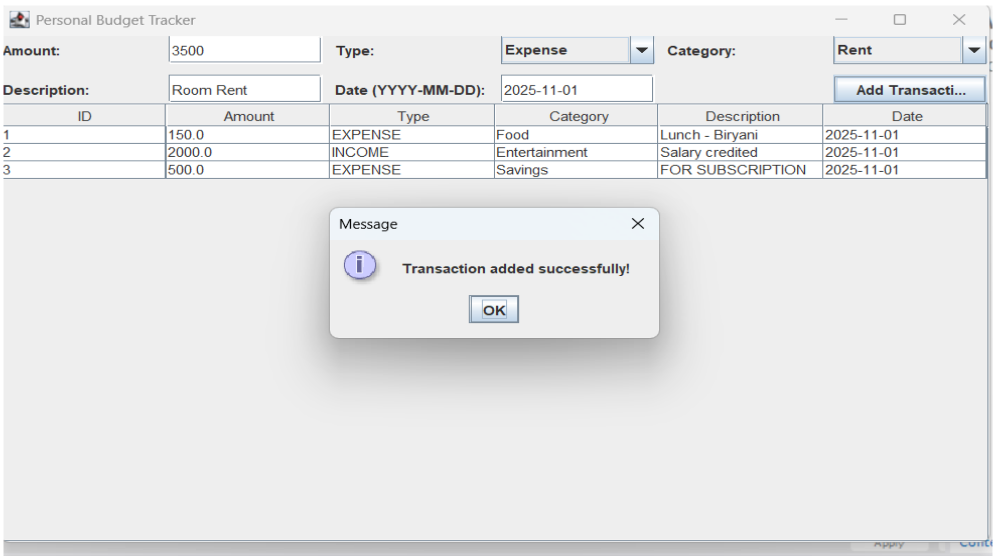
Personal Budget Tracker

Amount: Type: Expense Category: Savings

Description: Date (YYYY-MM-DD): 2025-11-01

ID	Amount	Type	Category	Description	Date
1	150.0	EXPENSE	Food	Lunch - Biryani	2025-11-01
2	2000.0	INCOME	Entertainment	Salary credited	2025-11-01
3	500.0	EXPENSE	Savings	FOR SUBSCRIPTION	2025-11-01

4. Adding Transactions



Personal Budget Tracker

Amount: 3500 Type: Expense Category: Rent

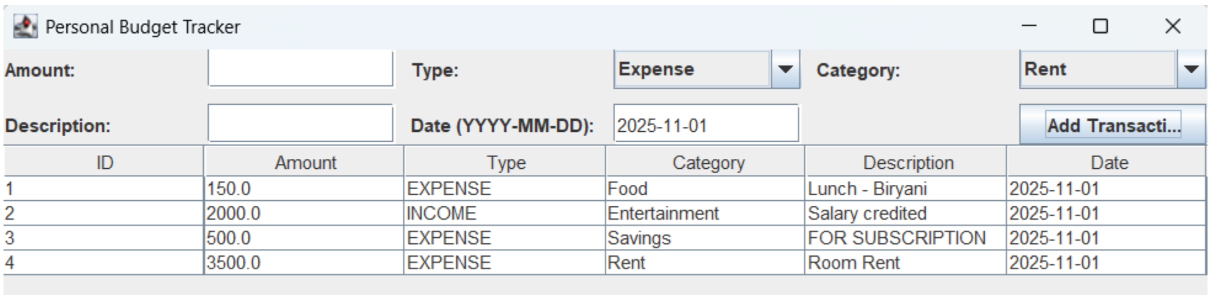
Description: Room Rent Date (YYYY-MM-DD): 2025-11-01

ID	Amount	Type	Category	Description	Date
1	150.0	EXPENSE	Food	Lunch - Biryani	2025-11-01
2	2000.0	INCOME	Entertainment	Salary credited	2025-11-01
3	500.0	EXPENSE	Savings	FOR SUBSCRIPTION	2025-11-01

Message

Transaction added successfully!

5. Viewing Transaction History

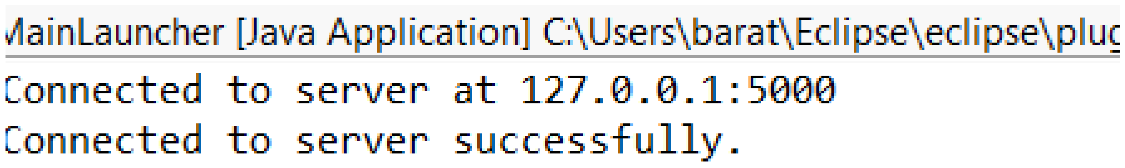


Personal Budget Tracker

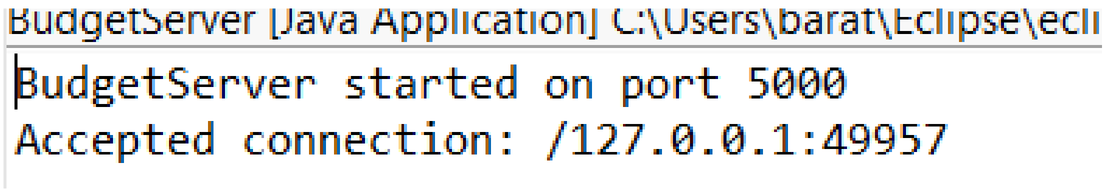
Amount:		Type:	Expense	Category:	Rent
Description:		Date (YYYY-MM-DD):	2025-11-01		Add Transacti...

ID	Amount	Type	Category	Description	Date
1	150.0	EXPENSE	Food	Lunch - Biryani	2025-11-01
2	2000.0	INCOME	Entertainment	Salary credited	2025-11-01
3	500.0	EXPENSE	Savings	FOR SUBSCRIPTION	2025-11-01
4	3500.0	EXPENSE	Rent	Room Rent	2025-11-01

6. Successful Connection Message

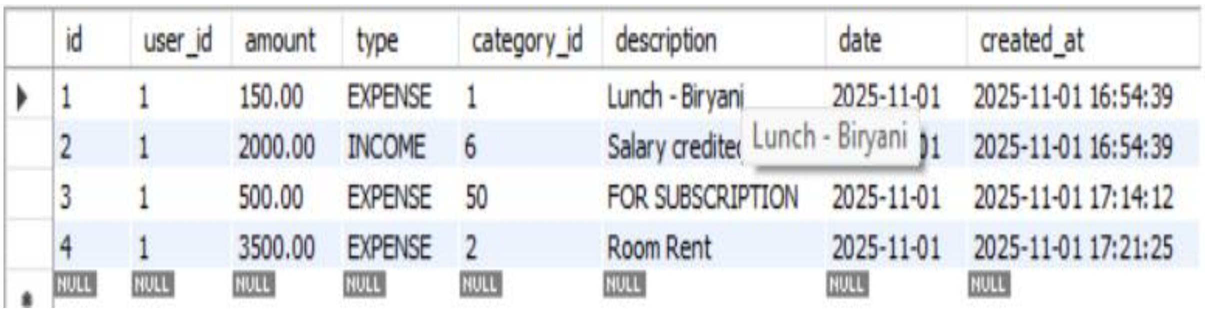


```
MainLauncher [Java Application] C:\Users\barat\Eclipse\eclipse\plug
Connected to server at 127.0.0.1:5000
Connected to server successfully.
```

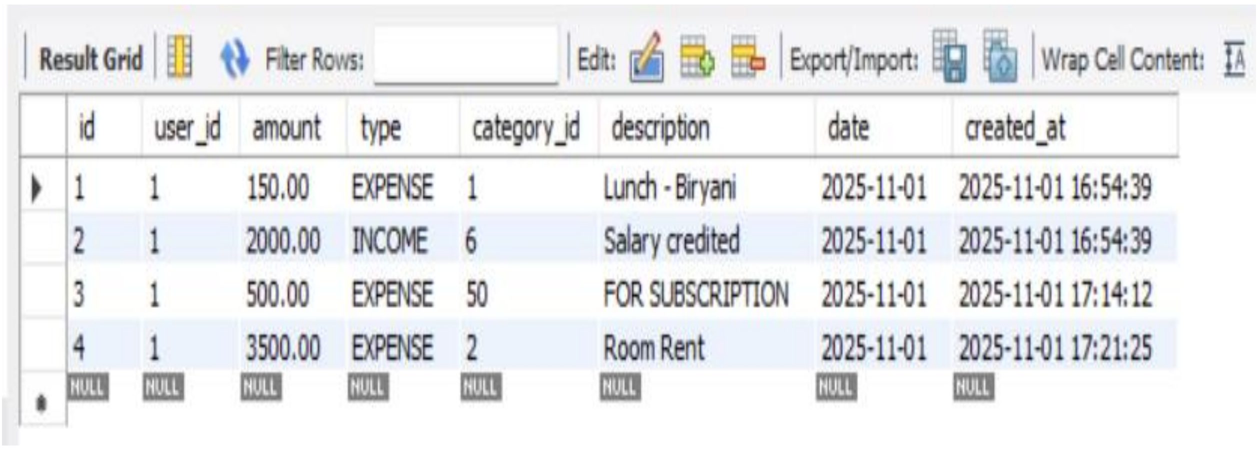


```
BudgetServer [Java Application] C:\Users\barat\Eclipse\ecli
BudgetServer started on port 5000
Accepted connection: /127.0.0.1:49957
```

7. Database Table View



id	user_id	amount	type	category_id	description	date	created_at
1	1	150.00	EXPENSE	1	Lunch - Biryani	2025-11-01	2025-11-01 16:54:39
2	1	2000.00	INCOME	6	Salary credite	2025-11-01	2025-11-01 16:54:39
3	1	500.00	EXPENSE	50	FOR SUBSCRIPTION	2025-11-01	2025-11-01 17:14:12
4	1	3500.00	EXPENSE	2	Room Rent	2025-11-01	2025-11-01 17:21:25
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL


Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

id	user_id	amount	type	category_id	description	date	created_at
1	1	150.00	EXPENSE	1	Lunch - Biryani	2025-11-01	2025-11-01 16:54:39
2	1	2000.00	INCOME	6	Salary credited	2025-11-01	2025-11-01 16:54:39
3	1	500.00	EXPENSE	50	FOR SUBSCRIPTION	2025-11-01	2025-11-01 17:14:12
4	1	3500.00	EXPENSE	2	Room Rent	2025-11-01	2025-11-01 17:21:25
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Conclusion:

The **Personal Budget Tracker** provides an efficient and secure method to manage personal finances using a Java-based client-server model. It enables users to monitor income, expenses, and categorize spending. With MySQL as the backend, it ensures data consistency and scalability. The project enhances practical skills in software design, socket programming, and database handling .In the future, AI and data analytics can be integrated to **predict expenses**, **generate savings advice**, and **visualize trends** using graphical insights. It can also be extended as a **mobile or web application** for greater accessibility.

Future Enhancement:

- Integrate biometric or OTP authentication for secure user access.
- Implement AI-based expense prediction and smart budgeting insights.
- Add automatic expense categorization using machine learning.
- Enable cloud backup and multi-device synchronization for data safety.
- Provide interactive dashboards and exportable financial reports.
- Develop a mobile or web version for anytime, anywhere access.

GITHUB LINK:	https://github.com/BARATH0202/Personal-Budget-Tracker.git

---

PO–PSO Mapping Table for "Personal Budget Tracker"

PO 1	**PO 2**	**PO 3**	**PO 4**	**PO 5**	**PO 6**	**PO 7**	**PO 8**	**PO 9**	**PO 10**	**PO 11**	**PO 12**	**PSO 1**	**PSO 2**	**PSO 3**
3	3	3	2	3	2	1	3	2	3	3	3	2	3	2

PSO Justification

PSO	**Relevance**	**Justification**
PSO1 – Use AI insights for effective decision-making	2	Integrated AI-based analytics to help users make better budgeting and financial planning decisions.
PSO2 – Manage and visualize analytical financial data	3	Designed dynamic visual dashboards to track income, expenses, and savings using graphical data visualization.
PSO3 – Research and develop innovative AI-based tools	2	Explored AI integration possibilities such as automated expense categorization and spending pattern prediction.

PO Justification

PO	Relevance	Justification
PO1 – Engineering Knowledge	3	Applied programming logic and database concepts using Java and MySQL to develop the Personal Budget Tracker.
PO2 – Problem Analysis	3	Analyzed user requirements to design a secure, reliable, and efficient budget management system.
PO3 – Design/Development of Solutions	3	Designed a modular application that records transactions, categorizes expenses, and displays financial summaries.
PO4 – Conduct Investigations of Complex Problems	2	Tested data integrity, verified transaction accuracy, and validated reports using SQL queries.
PO5 – Modern Tool Usage	3	Utilized Java, MySQL Workbench, and GUI frameworks for efficient application design and implementation.
PO6 – The Engineer and Society	2	Ensured user data privacy and promoted responsible financial behavior through secure budgeting features.
PO7 – Environment and Sustainability	1	Encouraged paperless money management by providing a fully digital personal finance solution.
PO8 – Ethics	3	Maintained high ethical standards by ensuring data confidentiality, transparency, and accuracy.
PO9 – Individual and Team Work	2	Collaborated in planning, coding, and testing modules while maintaining version control and communication.
PO10 – Communication	3	Documented the project workflow, system architecture, and user guide for clear understanding and usability.
PO11 – Project Management and Finance	3	Managed timelines, assigned roles, and monitored progress efficiently to complete project goals on time.
PO12 – Life-long Learning	3	Gained new knowledge on integrating UI/UX with backend logic and financial data analysis for continuous improvement.