A

Mini Project Report

On

Expense Tracker

Submitted in partial fulfillment of the requirements for the degree

Second Year Engineering – Computer Science Engineering (Data Science)

By

Krupa Devar : 24107065

Shravni Bhosale 24107025

Shravani Karpe 24107047

Under the guidance of

Ms. Aavani Nair

DEPARTMENT OF COMPUTER SCIENCE

ENGINEERING (DATA SCIENCE)

A.P. SHAH INSTITUTE OF TECHNOLOGY

G.B. Road, Kasarvadavali, Thane (W)-400615

UNIVERSITY OF MUMBAI

Academic year: 2025-26

CERTIFICATE

This to certify that the Mini Project report on Expense Tracker has been submitted by Krupa Devar : 24107065, Shravni Bhosale 24107025, Shravani Karpe 24107047 who are bonafide students of A. P. Shah Institute of Technology, Thane as a partial fulfillment of the requirement for the degree in **Computer Science Engineering (Data Science)**, during the academic year **2025-2026** in the satisfactory manner as per the curriculum laid down by University of Mumbai.

Dr. Pravin Adivarekar
HOD, CSE (Data Science)

Dr. Uttam D. Kolekar
Principal

External Examiner:

1.

Internal Examiner:

1.

Place: A. P. Shah Institute of Technology, Thane

Date:

ACKNOWLEDGEMENT

This project would not have come to fruition without the invaluable help of our guide Ms. Aavani Nair. Expressing gratitude towards our HoD, **Dr. Pravin Adivarekar**, and the Department of Computer Science Engineering (Data Science) for providing us with the opportunity as well as the support required to pursue this project. We would also like to thank our project coordinator **Ms. Aavani Nair** who gave us his/her valuable suggestions and ideas when we were in need of them. We would also like to thank our peers for their helpful suggestions.

TABLE OF CONTENTS

Chapter 1

Introduction

The rapid growth of digital technology has transformed how individuals manage their daily activities, including personal finance. In today's fast-paced world, tracking expenses manually has become increasingly impractical and inefficient. Students, working professionals, and individuals from various backgrounds face challenges in monitoring their spending patterns, leading to poor financial awareness and budget management issues. The lack of accessible, user-friendly expense tracking tools creates a gap between financial awareness and actual spending behavior.

Traditional methods of expense tracking, such as maintaining physical notebooks or spreadsheets, are time-consuming and prone to human error. These methods often lack real-time accessibility and fail to provide instant insights into spending patterns. Moreover, the absence of categorization and automatic calculation features makes it difficult for users to analyze their expenditure across different categories such as food, transportation, entertainment, and bills.

The Expense Tracker application addresses these challenges by providing a web-based solution that enables users to record, view, and analyze their expenses efficiently. Built using Java web technologies, this application demonstrates the practical implementation of full-stack development concepts including server-side programming with Java Servlets, dynamic content generation using JavaServer Pages (JSP), and persistent data storage using MySQL database. The application follows industry-standard design patterns such as Model-View-Controller (MVC) and Data Access Object (DAO) to ensure maintainability, scalability, and separation of concerns.

1.1. Purpose

The purpose of this document is to present a comprehensive overview of the Expense Tracker application developed as a mini project for the Second Year Computer Science Engineering (Data Science) program. This report serves multiple objectives:

Firstly, it documents the entire development lifecycle of the application, from problem identification and requirements analysis to design, implementation, and testing. The report provides detailed explanations of the technologies used, architectural decisions made, and implementation strategies employed during the project development.

Secondly, this document serves as technical documentation that can be referenced for understanding the system architecture, database schema, and code structure. It includes explanations of key components such as Java Servlets, JSP pages, DAO classes, and

database connectivity mechanisms. The report also highlights the learning outcomes achieved through hands-on implementation of these technologies.

Thirdly, the document demonstrates the practical application of theoretical concepts learned in coursework, including Object-Oriented Programming principles, database management, web application development, and client-server architecture. It showcases how these concepts integrate to create a functional, deployable web application.

The target audience for this report includes academic evaluators, project guides, peers interested in similar implementations, and future developers who may extend or maintain the application. The report is structured to provide both high-level overview and detailed technical insights, making it accessible to readers with varying levels of technical expertise.

1.2. Problem Statement

Individuals, particularly students and young professionals, lack an efficient and accessible system to record, view, and analyze their daily expenses. This absence of proper expense tracking leads to several challenges:

Financial Awareness Gap: Without systematic expense recording, individuals remain unaware of their actual spending patterns. This lack of awareness prevents them from making informed financial decisions and identifying areas where expenses can be reduced.

Manual Record-Keeping Challenges: Traditional methods of maintaining expense records in physical notebooks or spreadsheets are time-consuming and error-prone. Users often forget to record expenses immediately, leading to incomplete or inaccurate financial data.

Lack of Categorization: Manual tracking methods typically do not support easy categorization of expenses. This makes it difficult to analyze spending across different categories such as food, transportation, entertainment, and utilities, preventing users from understanding where their money goes.

No Real-Time Access: Physical records and local spreadsheets do not provide instant access to expense history. Users cannot quickly check their total expenditure or review past transactions when needed, especially when away from their primary recording medium.

Calculation Overhead: Manual calculation of total expenses across multiple entries is tedious and prone to arithmetic errors. Users need an automated system that can instantly compute totals and provide summary statistics.

The proposed Expense Tracker application addresses these challenges by providing a web-based solution that enables easy expense entry, automatic categorization, persistent storage in a database, real-time access through web browsers, and automatic calculation of total expenditure. The application eliminates the friction associated with expense tracking, encouraging users to maintain consistent financial records.

1.3. Objectives

The primary objectives of developing the Expense Tracker application are:

1. Develop a Web-Based Expense Management System: Create a functional web application that allows users to add expenses through an intuitive form interface and view all recorded expenses in a structured tabular format. The application should be accessible through standard web browsers without requiring installation of additional software.

2. Implement Database Integration: Establish persistent data storage using MySQL database to ensure that expense records are saved permanently and can be retrieved even after the application is closed. Implement proper database connectivity using JDBC (Java Database Connectivity) to enable seamless communication between the application and the database.

3. Apply MVC Architecture Pattern: Structure the application following the Model-View-Controller design pattern to achieve separation of concerns. This includes creating distinct layers for data representation (Model), user interface (View), and business logic (Controller), ensuring maintainability and scalability of the codebase.

4. Demonstrate Java Web Technologies: Showcase practical implementation of Java Servlets for handling HTTP requests, JavaServer Pages (JSP) for dynamic content generation, and Java classes for implementing business logic. This demonstrates understanding of server-side web development using Java.

5. Implement DAO Pattern: Utilize the Data Access Object pattern to encapsulate all database operations in dedicated classes, promoting code reusability and separation of data access logic from business logic.

6. Enable Real-Time Expense Tracking: Provide functionality for immediate expense entry and instant retrieval of expense history, allowing users to maintain up-to-date financial records and access them whenever needed.

7. Automatic Calculation Features: Implement automated calculation of total expenditure across all recorded expenses, eliminating manual calculation overhead and reducing the possibility of arithmetic errors.

1.4. Scope

The scope of the Expense Tracker application encompasses the following features and functionalities:

Functional Scope:

The application provides two primary functionalities: adding new expenses and viewing all recorded expenses. Users can add expenses by filling a form that captures four essential details: description of the expense, amount spent, category of expense (such as Food, Transport, Shopping, Bills, Entertainment, or Other), and the date of expense. The system stores this information in a MySQL database and provides confirmation to the user upon successful addition.

The view functionality displays all recorded expenses in a tabular format sorted by date in descending order, showing the most recent expenses first. The application automatically calculates and displays the total amount spent across all expenses. Users can navigate between the add and view pages using clearly labeled buttons.

Technical Scope:

The application is developed using Java as the primary programming language, leveraging Java Servlets for request handling and JSP for view rendering. The backend follows a layered architecture with distinct packages for models, data access objects, servlets, and utility classes. Database operations are performed using JDBC with prepared statements to prevent SQL injection attacks. The frontend utilizes HTML5 for structure, CSS3 for styling including gradient backgrounds and animations, and JavaScript for client-side form validation and user experience enhancements.

Deployment Scope:

The application is deployed on Apache Tomcat 9.0 server, which provides the servlet container for executing Java web applications. The application runs on localhost (development environment) and can be accessed through any modern web browser including Chrome, Firefox, and Edge. The MySQL database server runs independently and communicates with the application through JDBC driver.

Limitations and Future Scope:

The current implementation focuses on core functionality of adding and viewing expenses. Features not included in this version but identified for future enhancement include user authentication and authorization, update and delete operations for existing expenses, filtering and search capabilities based on date range or category, data visualization through charts and graphs, export functionality to PDF or Excel formats, and multi-user support with individual user accounts.

The application currently supports single-user operation in a local development environment. For production deployment, additional considerations such as security hardening, performance optimization, responsive design for mobile devices, and server hosting would be required.

Chapter 2

Proposed System

The proposed Expense Tracker system is a web-based application designed to simplify personal finance management through an intuitive interface and robust backend architecture. The system adopts a three-tier architecture comprising the presentation layer, application layer, and data layer, ensuring clear separation of concerns and maintainability.

The presentation layer consists of JavaServer Pages (JSP) that generate dynamic HTML content displayed in the user's browser. Two primary pages form the user interface: an expense entry form and an expense list view. The presentation layer also includes CSS stylesheets for visual design and JavaScript files for client-side interactivity and validation.

The application layer, implemented using Java Servlets, serves as the intermediary between the presentation layer and data layer. Servlets handle HTTP requests from the client, process form data, invoke business logic, interact with the data access layer, and determine the appropriate response to send back to the client. This layer implements the controller component of the MVC pattern.

The data layer comprises the MySQL database where expense records are persistently stored, and the Data Access Object (DAO) classes that encapsulate all database operations. The DAO pattern isolates the complexities of database interaction from the rest of the application, providing a clean interface for data operations such as inserting new expenses, retrieving all expenses, and calculating total expenditure.

The system employs several design patterns and best practices. The Model-View-Controller pattern separates data representation, user interface, and business logic. The Data Access Object pattern isolates database access logic. Prepared Statements are used for all SQL queries to prevent SQL injection attacks. Exception handling is implemented throughout the application to gracefully handle errors. The application follows Java naming conventions and coding standards for consistency and readability.

2.1. Features and Functionality

The Expense Tracker application provides the following features:

Add Expense Feature:

Users can add new expenses through a form-based interface that captures four essential pieces of information. The description field accepts text input describing what the expense was for, such as "Metro Card Recharge" or "Team Lunch". The amount field accepts numerical input with support for decimal values to accommodate cents or paise. The category field provides a dropdown menu with predefined categories including Food, Transport, Shopping, Bills, Entertainment, and Other. The date field uses an HTML5 date

picker that defaults to the current date but allows users to select past dates if they are recording historical expenses.

When the user submits the form, JavaScript validation ensures all fields are filled and the amount is a positive number. The form data is sent to the AddExpenseServlet via HTTP POST request. The servlet extracts the form parameters, creates an Expense object, and passes it to the ExpenseDAO for database insertion. Upon successful insertion, the servlet forwards the request back to the form page with a success message. If any error occurs during processing, an appropriate error message is displayed to the user.

View Expenses Feature:

Users can view all recorded expenses by clicking the "View All Expenses" button. This action sends a GET request to the ViewExpensesServlet, which retrieves all expense records from the database through the ExpenseDAO. The servlet also calculates the total expenditure across all expenses. Both the expense list and total amount are set as request attributes and forwarded to the expenses.jsp page for rendering.

The expenses page displays a prominently styled total expenditure banner at the top, followed by a table listing all expenses. The table columns show the date, description, category, and amount for each expense. Expenses are sorted by date in descending order, displaying the most recent expenses first. The table uses alternating row colors and hover effects for better readability. Each amount is formatted with the rupee symbol and two decimal places. At the bottom of the page, a button allows users to navigate back to the add expense form.

User Interface Features:

The application features a modern, gradient-based design with purple and blue color schemes. Form inputs have smooth focus animations and hover effects. Buttons use gradient backgrounds with shadow effects and transform animations. The layout is centered on the page with appropriate padding and spacing. Font sizes and weights are carefully chosen for readability. Icons from Unicode emojis add visual appeal to labels and headings. CSS animations provide smooth transitions between states. The design is consistent across both pages, creating a cohesive user experience.

Data Persistence:

All expense data is stored in a MySQL database, ensuring that records persist even when the application is closed or the server is restarted. The database uses auto-incrementing primary keys for expense IDs. Timestamps are automatically recorded for when each expense was created. The database schema enforces NOT NULL constraints on essential fields to maintain data integrity.

2.2. System Architecture

The Expense Tracker follows a layered architecture based on the Model-View-Controller pattern. The architecture consists of four primary layers:

Presentation Layer (View):

This layer comprises JSP files, HTML, CSS, and JavaScript that run in the client's web browser. The two main JSP files are index.jsp for the expense entry form and expenses.jsp for displaying the expense list. The JSP files contain embedded Java code (scriptlets) that dynamically generate HTML based on data received from servlets. CSS files provide styling for visual presentation, including layout, colors, fonts, and animations. JavaScript files handle client-side form validation, dynamic date setting, and user interaction enhancements. The presentation layer communicates with the application layer through HTTP requests triggered by form submissions and link clicks.

Application Layer (Controller):

This layer consists of Java Servlets that handle HTTP requests and responses. The AddExpenseServlet processes POST requests from the expense entry form. It extracts form parameters, validates the data, creates an Expense object, invokes the DAO to persist the data, sets success or error messages as request attributes, and forwards the request back to the form page. The ViewExpensesServlet handles GET requests for displaying expenses. It invokes DAO methods to retrieve all expenses and calculate the total, sets the data as request attributes, and forwards to the expenses.jsp page. Servlets act as controllers in the MVC pattern, coordinating between the view and model layers.

Business Logic Layer (Model):

This layer contains Java classes that represent the business entities and logic. The Expense class serves as the model, representing an expense entity with private fields for id, description, amount, category, and date. The class provides getters and setters for accessing these fields, following the JavaBeans convention. The Expense class uses appropriate data types: int for ID, String for description and category, BigDecimal for amount to handle currency values precisely, and Date for the expense date. This class encapsulates the data structure of an expense without containing any database or presentation logic.

Data Access Layer (DAO):

This layer encapsulates all database operations. The ExpenseDAO class provides methods for interacting with the database. The addExpense() method takes an Expense object, constructs an SQL INSERT statement, and executes it using a prepared statement. The getAllExpenses() method executes a SELECT query, iterates through the result set, creates Expense objects from each row, and returns a list of all expenses. The getTotalExpenses() method executes a SUM query and returns the total expenditure as a BigDecimal. The DatabaseUtil class manages database connections. It defines connection parameters such as URL, username, and password, loads the MySQL JDBC driver, and provides a static getConnection() method that returns a database connection. All DAO methods use try-with-resources to ensure proper resource cleanup and implement exception handling to manage database errors gracefully.

Data Flow:

When a user submits the expense form, the browser sends an HTTP POST request to the AddExpenseServlet. The servlet extracts form parameters and creates an Expense object. The servlet calls expenseDAO.addExpense() passing the Expense object. The DAO method obtains a database connection from DatabaseUtil, prepares an INSERT statement with the expense data, and executes the statement. The database stores the expense record and returns a success indicator. The DAO method returns the result to the servlet. The servlet sets an appropriate message attribute and forwards to index.jsp. The JSP page displays the form along with the success or error message.

When a user requests the expense list, the browser sends an HTTP GET request to ViewExpensesServlet. The servlet calls expenseDAO.getAllExpenses() and expenseDAO.getTotalExpenses(). The DAO methods execute SELECT queries and process the results. The expense list and total are returned to the servlet. The servlet sets these as request attributes and forwards to expenses.jsp. The JSP page iterates through the expense list, generates an HTML table, and displays the total amount.

This architecture promotes separation of concerns, where each layer has a specific responsibility. The presentation layer focuses on user interface, the controller layer handles request routing and coordination, the model layer represents business entities, and the DAO layer manages data persistence. This separation makes the application easier to understand, maintain, and extend.

Chapter 3

Software Requirements

The development and deployment of the Expense Tracker application require specific software components across different categories:

Development Tools:

Visual Studio Code (Version 1.80 or higher): Serves as the Integrated Development Environment for writing and editing code. Required extensions include Extension Pack for Java by Microsoft for Java language support, syntax highlighting, and IntelliSense. Community Server Connectors by Red Hat for server management and deployment. MySQL by Jun Han for database connectivity and query execution. The editor should be configured with proper Java classpath settings pointing to the lib folder containing JAR files.

MySQL Workbench (Version 8.0 or higher): Provides a graphical interface for database management, including creating databases and tables, executing SQL queries, viewing database schema and data, importing and exporting data, and managing database users and permissions. MySQL Workbench is used for initial database setup and ongoing database administration tasks.

Command Prompt / Terminal: Required for compiling Java files using javac compiler, starting and stopping Tomcat server, navigating file system, and executing various command-line operations during development and deployment.

Runtime Environment:

Java Development Kit (JDK 11): Provides the Java Runtime Environment and development tools. The JDK includes the Java compiler (javac) for compiling Java source files into bytecode, the Java Virtual Machine for executing compiled code, standard Java libraries and APIs, and development tools for debugging and profiling. The JAVA_HOME environment variable must be set to the JDK installation directory for proper functioning of development tools and application server.

Apache Tomcat (Version 9.0.109): Serves as the web server and servlet container for hosting the Java web application. Tomcat provides the implementation of Java Servlet and JavaServer Pages specifications, HTTP server functionality for handling web requests, management console for deploying and monitoring applications, and logging facilities for debugging and monitoring. The CATALINA_HOME environment variable should be set to the Tomcat installation directory.

MySQL Server (Version 8.0 or higher): Provides the relational database management system for persistent data storage. MySQL Server offers reliable data storage with ACID properties, support for multiple concurrent connections, SQL query processing and optimization, security features including user authentication and authorization, and backup and recovery capabilities. The MySQL service must be running on port 3306 (default) for the application to connect successfully.

Libraries and Dependencies:

MySQL Connector/J (Version 9.4.0): The JDBC driver for MySQL database connectivity. This JAR file must be placed in the application's WEB-INF/lib directory. The driver enables Java applications to connect to MySQL databases, execute SQL statements, process query results, and handle database transactions.

Java Servlet API (Version 4.0.1): Provides the servlet interface and related classes required for servlet development. The servlet-api.jar file contains interfaces such as HttpServlet, HttpServletRequest, and HttpServletResponse, servlet lifecycle methods, and session management capabilities. This library must be available in the classpath during compilation and is provided by Tomcat at runtime.

JSP API (Version 2.2): Contains interfaces and classes for JavaServer Pages development. The jsp-api.jar file includes JSP page lifecycle interfaces, implicit objects like request and response, and tag library support. This library is also provided by Tomcat at runtime but may be needed during compilation for JSP development.

Web Browser:

Google Chrome (Version 90 or higher), Mozilla Firefox (Version 88 or higher), or Microsoft Edge (Version 90 or higher): Modern web browser with support for HTML5, CSS3, and JavaScript ES6. The browser is required for accessing the web application, rendering the user interface, executing client-side JavaScript, and handling form submissions and navigation.

Operating System Requirements:

The application can be developed and deployed on Windows 10 or higher, macOS 10.14 or higher, or Linux distributions such as Ubuntu 20.04 or higher. The operating system should have at least 4GB of RAM for smooth operation of development tools and servers, 10GB of available disk space for software installations and project files, and network connectivity for downloading dependencies and accessing documentation.

Chapter 4

Project Design

The project design encompasses database schema design, class structure, and system workflow. Proper design ensures that the application is scalable, maintainable, and performs efficiently.

4.1. Database Design

The database schema consists of a single table named "expenses" within the "expense_tracker" database. The table structure is designed to efficiently store expense records while maintaining data integrity.

Table: expenses

Column Name	Data Type	Constraints	Description
id	INT	PRIMARY KEY, AUTO_INCREMENT	Unique identifier for each expense record
description	VARCHAR(255)	NOT NULL	Textual description of the expense
amount	DECIMAL(10,2)	NOT NULL	Monetary value with two decimal places
category	VARCHAR(100)	NOT NULL	Category classification (Food, Transport, etc.)
date	DATE	NOT NULL	Date when the expense occurred
created_at	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP	Timestamp of record creation

Figure 4.1: Expenses Table Schema

Design Rationale:

The id field uses auto-increment to automatically generate unique identifiers for each record, eliminating the need for manual ID assignment and ensuring uniqueness. The description field uses VARCHAR(255) which provides sufficient length for descriptive text while limiting excessive data storage. The amount field uses DECIMAL(10,2) data type instead of FLOAT or DOUBLE to ensure precise storage of monetary values without floating-point rounding errors. The DECIMAL(10,2) format allows for amounts up to 99,999,999.99 with exactly two decimal places.

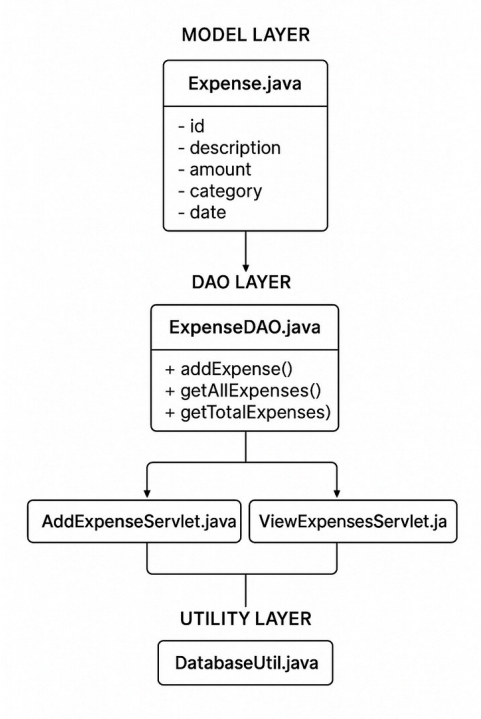
The category field stores predefined category values to enable expense classification and analysis. Although the current implementation does not enforce category values at the database level, the application layer restricts input to specific categories. The date field

uses the DATE data type which stores only the date portion without time information, as time is not relevant for expense tracking in this application. The created_at timestamp field automatically records when each expense was added to the database, providing an audit trail and enabling sorting by entry order if needed.

The table uses NOT NULL constraints on essential fields to prevent incomplete data entry. The PRIMARY KEY constraint on the id field ensures uniqueness and creates an index for efficient record retrieval. The table design follows normalization principles, currently in First Normal Form (1NF) with atomic values in each column. For future enhancements involving multiple users or additional attributes, the schema can be extended without modifying the core structure.

4.2. Class Diagram

The application follows object-oriented design principles with clear class responsibilities and relationships.



4.3. System Flow Diagram

Add Expense Flow:

1. User fills expense form in index.jsp with description, amount, category, and date
2. User clicks "Add Expense" button
3. JavaScript validates form data on client side
4. Browser sends HTTP POST request to /addExpense URL

5. AddExpenseServlet receives request and extracts parameters
6. Servlet creates Expense object with form data
7. Servlet calls expenseDAO.addExpense(expense)
8. ExpenseDAO obtains database connection from DatabaseUtil
9. DAO prepares SQL INSERT statement with expense data
10. DAO executes statement and database stores the record
11. DAO returns success/failure boolean to servlet
12. Servlet sets appropriate message attribute
13. Servlet forwards request to index.jsp
14. JSP displays form with success/error message
15. User sees confirmation of action

Figure 4.7: Add Expense Sequence Diagram

View Expenses Flow:

1. User clicks "View All Expenses" button
2. Browser sends HTTP GET request to /viewExpenses URL
3. ViewExpensesServlet receives request
4. Servlet calls expenseDAO.getAllExpenses()
5. DAO obtains database connection
6. DAO executes SELECT query
7. DAO iterates through ResultSet creating Expense objects
8. DAO returns List<Expense> to servlet
9. Servlet calls expenseDAO.getTotalExpenses()
10. DAO executes SUM query and returns BigDecimal total
11. Servlet sets expenses list and total as request attributes
12. Servlet forwards request to expenses.jsp
13. JSP iterates through expenses list
14. JSP generates HTML table with expense data
15. JSP displays total amount in header
16. Browser renders the complete page
17. User views expense list and total

Figure 4.8: View Expenses Sequence Diagram

Database Connection Management:

Every database operation follows this pattern: Obtain connection using DatabaseUtil.getConnection(), create PreparedStatement or Statement, execute query or update, process results if applicable, close resources in finally block or using try-with-resources, and handle SQLExceptions appropriately. This ensures proper resource management and prevents connection leaks.

Chapter 5

Implementation Details

This chapter provides detailed explanations of the code implementation across different layers of the application.

5.1. Model Layer

Expense.java

The Expense class serves as the data model representing an expense entity. It follows the JavaBeans convention with private fields and public accessor methods.

Field Declarations:

```
private int id;
private String description;
private BigDecimal amount;
private String category;
private Date date;
```

The fields use appropriate data types: int for numeric identifier, String for textual data, BigDecimal for precise decimal arithmetic required in financial calculations, and java.sql.Date for date representation compatible with SQL DATE type.

Constructors:

The class provides two constructors. The default no-argument constructor allows creating empty Expense objects that can be populated later, which is useful when retrieving data

from the database. The parameterized constructor accepts description, amount, category, and date, allowing creation of fully initialized Expense objects when adding new expenses. The id field is not included in the parameterized constructor as it is auto-generated by the database.

Accessor Methods:

All fields have corresponding getter and setter methods following JavaBeans naming conventions. Getters return the field values allowing read access. Setters accept parameters and assign them to fields allowing write access. This encapsulation protects the fields from direct external access while providing controlled access through methods. The methods enable frameworks and libraries to manipulate the object through reflection.

Design Considerations:

Using BigDecimal instead of double or float ensures precise representation of monetary values without floating-point rounding errors. The java.sql.Date type is chosen over java.util.Date for better integration with JDBC and SQL databases. Private fields with public accessors implement encapsulation principle of object-oriented programming. The class is a Plain Old Java Object (POJO) without dependencies on frameworks or libraries, making it lightweight and testable.

5.2. DAO Layer

ExpenseDAO.java

The Data Access Object encapsulates all database operations related to expenses.

addExpense() Method:

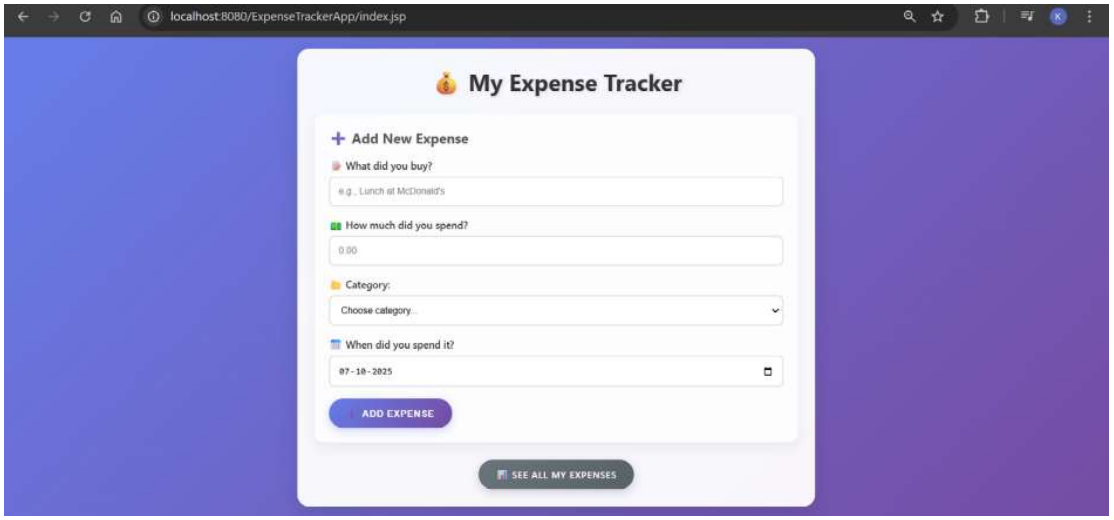
This method accepts an Expense object and inserts it into the database. The SQL INSERT statement uses placeholders (?) for values to create a prepared statement. Parameters are set using setter methods: setString() for description and category, setBigDecimal() for amount, and setDate() for date. The executeUpdate() method executes the INSERT statement and returns the number of rows affected. The method returns true if insertion is successful (rows affected > 0), false otherwise. All database operations are enclosed in try-with-resources ensuring automatic resource cleanup. Any SQLException is caught and logged, and the method returns false indicating failure.

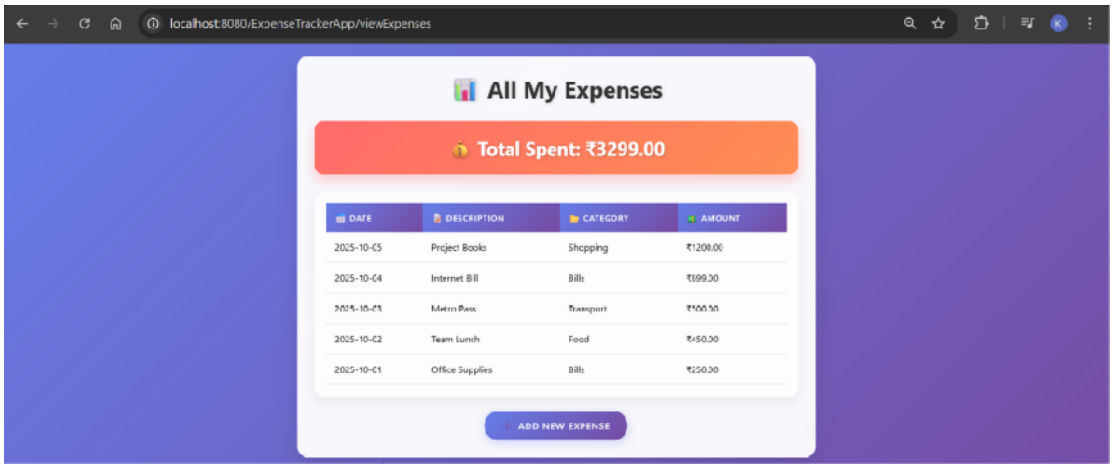
getAllExpenses() Method:

This method retrieves all expense records from the database and returns them as a List. An ArrayList is created to store

Chapter 6

Results





References

[1] Abror Abduvaliyev, Al-Sakib Khan Pathan, Jianying Zhou, Rodrigo Roman and WaiChoong Wong ,“On the vital Areas of Intrusion Detection Systems in Wireless Sensor networks”,IEEE Communications Surveys & Tutorials, Accepted For Publications, 2013-in press.

[2] H.H. Soliman, et al,“A comparative performance evaluation of intrusion detection techniques for hierarchical wireless sensor networks”, Egyptian Informatics Journal (2012) 13, 225238.

[3] Giannetsos Athanasios, “Intrusion Detection in Wireless Sensor Networks”, Master THESIS, Carnegie Mellon University, April 8, 2008.

[4] K.Fall and K.Varadhan,“The NS Manual”, http://www.isi.edu/nsnam/ns/doc/ns doc.pdf., 1 Feb 2014.

[5] Jae Chung and Mark Claypool,“NS by Example-Tutorial”, http://nile.wpi.edu/NS/overview.html , 1 Feb 2014.

[6] Network Simulator blog, http://Mohittahilani.blogspot.com , 1 Feb 2014.

[7] AWK Script for NS2, http://mohit.ueuo.com/AWK-Scripts.html , 1 Feb 2014.