A Project Report on

**BillBuddy**

*Submitted in partial fulfillment of the requirement for the award of the degree of*

**Master of Computer Application (MCA)**

Guru Gobind Singh Indraprastha University, Delhi

Guided By: Submitted by:

Mr. Munish Kumar (Int.) Drishti Arora

Mr. Mo. Masood Ansari (Ext.) Enroll No.:06211804423

MCA 4th Sem.



**Centre for Development of Advanced Computing**

2023-2025

**CERTIFICATE**

I, **Drishti Arora**, Enrollment No. **06211804423** certify that the Project Report/Dissertation (MCA-202) entitled “**BillBuddy**” is done by me and it is an authentic work carried out by me at **QSpiders, Noida**. The matter embodied in this project work has not been submitted earlier for the award of any degree or diploma to the best of my knowledge and belief.

Signature of the Student

Date:

Certified that the Project Report/Dissertation (MCA-202) entitled “**BillBuddy**” done by **Ms. Drishti Arora**, Enrollment No. **06211804423**, is completed under my guidance.

Signature of the Guide

Date:

Name of the Guide: Mr. Mo. Masood Ansari

Designation: Branch Manager

Address: QSpiders, Noida

ii

**CERTIFICATE**

I, **Drishti Arora,** Enrollment No. **06211804423** certify that the Project Report/Dissertation (MCA-202) entitled “**BillBuddy**” is done by me and it is an authentic work carried out by me at **QSpiders, Noida**. The matter embodied in this project work has not been submitted earlier for the award of any degree or diploma to the best of my knowledge and belief.

Signature of the Student

Date:

Certified that the Project Report/Dissertation (MCA-202) entitled “**BillBuddy**” done by **Ms. Drishti Arora**, Enrollment No. **06211804423**, is completed under my guidance.

Signature of the Guide

Date:

Name of the Guide: Mr. Munish Kumar

Designation:

Address: CDAC Noida

iii

**ACKNOWLEDGEMENT**

It is difficult to put into words how deeply grateful I am to the management team of the Centre for Development of Advanced Computing (CDAC) and the institute's faculty for granting me the opportunity to avail the excellent facilities and infrastructure. The education and values instilled in me have greatly assisted me at the beginning of my career.

I would like to express my sincere appreciation and thanks to Mr. Munish Kumar for his guidance and supervision, as well as to the institute's faculty members for always being available when I needed their help.

I am incredibly grateful to QSpiders for offering me the opportunity to undertake my Industrial Training, which is a crucial aspect of the MCA course. This experience exposes individuals to industry standards and assists them in adapting to the most recent trends and technologies while also providing them with project-based expertise. I am proud and fortunate to express my sincere gratitude to everyone who aided me in presenting this project.

Furthermore, I am indebted to my External Guide, Mr. Mo. Masood Ansari at QSpiders for his excellent guidance and cooperation throughout my project endeavor, which made my learning experience worthwhile. I would also like to express my appreciation for his valuable support and sincere efforts in helping me not only become familiar with but also well-versed in the fundamentals of this project. Under his guidance, completing this project has been an enlightening experience.

It is indeed a pleasure to thank my family and friends who persuaded and encouraged me to take up and complete this task. Last but not least, I express my gratitude and appreciation to all those who have helped me directly or indirectly towards the successful completion of this project.

Drishti Arora

06211804423

iv

**SYNOPSIS**

1. **Introduction**

Managing shared expenses is a common challenge, whether among roommates, friends, or travel groups. This project simplifies expense tracking, debt settlement, and group expense management. It ensures that everyone pays their fair share without needing to constantly exchange cash or track debts manually.

1. **Objectives**

* To develop a user-friendly application for managing shared expenses.
* To automate the process of splitting bills and calculating individual shares.
* To implement secure user authentication and authorization.
* To provide a detailed expense history and summary reports.
* To integrate payment options for easy debt settlement.

1. **Technologies Used:**
2. **Backend:** Spring Boot, Spring Data JPA
3. **Frontend:** React.js
4. **Database:** MySQL
5. **Authentication:** JWT/OAuth
6. **Version Control:** Git/Github
7. **Testing:** Postman API
8. **Modules:**

* **User Management** – Sign-up, login, profile management, authentication & authorization.
* **Expense Management** – Add, edit, delete expenses, attach receipts, and categorize expenses.
* **Group Management** – Create groups, add members, manage shared expenses.
* **Debt Calculation & Simplification** – Auto-calculate who owes whom, reduce transactions.
* **Notification System** – Send reminders, expense updates via email/SMS.

1. **System Architecture:**

The application follows a **3-tier architecture**:

v

* **Frontend** **(React.js):** Handles UI and API interactions.
* **Backend (Spring Boot & Hibernate):** Processes requests, manages business logic, and interacts with the database.
* **Database (MySQL):** Stores user, expense, and transaction data.

1. **Key Features:**

* User Registration & Login (JWT Authentication)
* Expense Splitting (Equal/Percentage-based split)
* Real-time Expense Tracking & History
* Group-based Expense Management
* Automated Debt Calculation & Simplification
* Secure Payments & Settlements (optional)
* Mobile-responsive UI with a Clean UX

vi

**Table of Contents**

|  |  |  |
| --- | --- | --- |
| **S No** | **Topic** | **Page No** |
| 1 | Certificate (s) | ii-iii |
| 2 | Acknowledgements | iv |
| 3 | Synopsis | v-vi |
| 4 | List of Figures | ix |
| 5 | List of Abbreviations | x |
| 6 | Chapter-1: Introduction | 1 |
|  | 1.1 Brief Description of the Organization | 1 |
|  | 1.2 Brief Description of the Project | 1 |
| 7 | Chapter-2: System Requirement Analysis | 3 |
|  | 2.1 Functional Requirements | 3 |
|  | 2.2 Non- Functional Requirements | 3 |
|  | 2.3 Hardware Requirements | 4 |
|  | 2.4 Software Requirements | 4 |
|  | 2.4.1 Frontend Technologies | 5 |
|  | 2.4.2 Backend Technologies | 5 |
|  | 2.4.3 Database | 6 |
|  | 2.4.4 Authentication and Security | 6 |
|  | 2.4.5 Email Service | 6 |
|  | 2.4.6 Development and Testing Tools | 6 |
| 8 | Chapter-3: System Design | 8 |
|  | 3.1 Architecture Design | 8 |
|  | 3.2 Database Design | 8 |

vii

|  |  |  |
| --- | --- | --- |
|  | 3.3 Frontend Design | 10 |
|  | 3.4 Email System Design | 11 |
|  | 3.4.1 Functionality | 11 |
|  | 3.4.2 Technical Setup | 11 |
|  | 3.5 Flowchart and Class Diagram for BillBuddy | 11 |
|  | 3.6 Snapshots of the Project | 13 |
| 9 | Chapter-4: Systems Development | 17 |
|  | 4.1 Frontend Development | 17 |
|  | 4.2 Backend Development | 17 |
|  | 4.3 Splitting Logic Implementation | 18 |
| 10 | Summary and Conclusions | 20 |
| 11 | References | 22 |

viii

**LIST OF FIGURES**

|  |  |  |
| --- | --- | --- |
| **Figure No.** | **Title** | **Page No** |
| 3.5.1 | Flowchart | 12 |
| 3.5.2 | Class Diagram | 13 |
| 3.6.1 | Image showing the Sign-up page | 13 |
| 3.6.2 | Image showing the Login page | 14 |
| 3.6.3 | Image showing the Dashboard | 14 |
| 3.6.4 | Image showing the created groups | 15 |
| 3.6.5 | Image showing the pop-up to create a group | 15 |
| 3.6.6 | Image showing the pop-up to add a member | 16 |
| 3.6.7 | Image showing the members in a created group | 16 |

ix

**List of Abbreviations**

|  |  |  |
| --- | --- | --- |
| **S. No.** | **Abbreviation** | **Full Form** |
| 1 | e.g. | For example, |
| 2 | i.e. | That is |
| 3 | IDE | Integrated Development Environment |
| 4 | JPA | Java Persistence API |
| 5 | SQL | Structured Query Language |
| 6 | HTTPS | Hypertext Transfer Protocol Secure |
| 7 | STS | Spring Tool Suite |
| 8 | API | Application Programming Interface |
| 9 | PK | Primary Key |
| 10 | FK | Foreign Key |
| 11 | SMTP | Simple Mail Transfer Protocol |

x

**Chapter – 1: Introduction**

* 1. **Brief Description of the Organisation:**

**QSpiders** (Unit of Test Yantra Software Solutions Pvt Ltd) is the world's ace software testing training organization with an aim to bridge the gap between the demands of the industry and the curriculum of educational institutions. With centers across the Globe, the institute is a platform where young minds are given the opportunity to build successful careers. "**QSpiders** is a place where businesses find talent.”

**QSpiders** is a best-in-class learning solutions organization headquartered in India's IT capital, Bangalore. We offer a wide range of courses in the area of software testing and are official partners of the ISTQB®. A "finishing school" in many ways, the institute provides young job aspirants the perfect launch-pad to build a rewarding career in the growing IT sector. From its humble beginnings, **QSpiders** has exponentially grown to be the world's largest software testing training organization spread across countries. At **QSpiders**, we ensure training is imparted by specialists with proven subject matter expertise and who have spent over a decade in their area of specialization. Our faculty are highly competent, skilled and dedicated to giving their best towards the professional development of our students. Besides training, we also provide placement assistance to our students and most of the big corporates in the corporate world hire our trained talent. It is indeed our pleasure to have placed over thousands of job-seekers in various IT firms

* 1. **Brief Description of the Project :**

**BillBuddy** is a full-stack web application designed to streamline the process of managing and tracking shared expenses among individuals or groups. Whether it's roommates sharing rent and groceries, friends splitting travel costs, or colleagues contributing to a team outing, **BillBuddy** offers an intuitive and automated solution to ensure fair distribution of costs and transparent tracking of all financial contributions.

The traditional method of settling shared expenses—through manual calculations, spreadsheet tracking, or constant messaging—can be cumbersome and error-prone. **BillBuddy** addresses this gap by providing a centralized platform where users can create expense groups, add individual or shared expenses, automatically split bills, and monitor pending balances with ease.

Built with **Core Java, Spring Boot, Spring Data JPA, and Lombok** on the backend, and using **React.js, HTML/CSS, JavaScript, and Axios** on the frontend, the application ensures smooth and secure communication between users and the server.

1

The backend is connected to a **MySQL** database for persistent and reliable data storage. The entire development process was supported using tools such as **Spring Tool Suite (STS), Visual Studio Code, and MySQL Workbench.**

**BillBuddy** not only facilitates real-time expense tracking but also introduces features such as debt simplification, group expense management, and detailed reporting, thereby enhancing financial clarity and collaboration within groups.

2

# Chapter – 2: System Requirement Analysis

**2.1 Functional Requirements**

This section outlines the core functionalities that the system must perform to meet user expectations and project objectives:

**User Registration/Login:**

The system provides a secure user authentication mechanism, allowing new users to register and existing users to log in. Proper validation and error handling are in place to ensure a seamless experience. User credentials are securely stored and managed.

**Email Notification:**

Email services are integrated to send automated notifications upon successful signup and login. These confirmation emails serve both as a security measure and user feedback, ensuring users are kept informed of account activities.

**Create/Join Group:**

Registered users can create new groups (e.g., "Trip to Goa") or join existing ones using group codes or invitations. Group data is persisted in the backend and supports multiple users per group.

**Add Expenses:**

Users can add shared expenses to a group. The system supports both equal and custom splitting of amounts among group members. Descriptive fields such as "paid by," "for what," and "split among" are available to keep transaction history clear.

**Track Balances:**

The system automatically calculates and displays individual balances. It visually shows who owes whom and how much, making it easier to manage shared finances without confusion.

**Settle Balances:**

Users can mark payments as settled. Once a settlement is recorded, the system updates the corresponding balances and transaction logs, maintaining a clear financial history.

**2.2 Non-Functional Requirements**

Non-functional requirements focus on the \*quality attributes\* of the system, ensuring a reliable and user-friendly experience:

3

**Usability:**

The application features a clean and responsive UI developed with modern web standards. It is intuitive, with minimal learning curve, enabling users to perform tasks effortlessly on both desktop and mobile devices.

**Security:**

The system employs secure practices such as encrypted communication via HTTPS, password hashing, and token-based authentication. Sensitive data is protected at every stage of the application lifecycle.

**Performance:**

Optimized frontend and backend ensure fast page loads and minimal latency. The use of asynchronous API calls and efficient database queries provides real-time updates and smooth user interactions.

**Maintainability:**

The codebase follows modular and component-based architecture, making it easy to maintain, extend, and debug. Clear separation of concerns between frontend and backend enhances scalability and future development.

**2.3 Hardware Requirements**

To develop and run the system efficiently, the following hardware specifications are recommended:

* A modern personal computer or laptop.
* Minimum 4GB RAM and 1.5 GHz processor for smooth performance during development and execution.
* A stable and reliable internet connection is essential, particularly for API communication and remote testing.

**2.4 Software Requirements**

The development and deployment of BillBuddy require a combination of frontend, backend, database, and development tools. These software components ensure the system is responsive, secure, scalable, and maintainable.

4

**2.4.1 Frontend Technologies:**

**HTML (HyperText Markup Language):**

Provides the structural framework for the application’s user interface, including forms, buttons, and display components.

**CSS (Cascading Style Sheets):**

Handles the styling and layout of pages. It ensures the interface is user-friendly, aesthetically pleasing, and responsive across devices.

**JavaScript:**

Adds interactivity and dynamic content to the UI. JavaScript drives client-side logic and form validations.

**React.js:**

A JavaScript library for building reusable, component-driven interfaces. React’s virtual DOM enhances rendering efficiency and provides a seamless user experience.

**Axios:**

A promise-based HTTP client used for making API calls between the frontend and backend. It simplifies the process of sending and receiving asynchronous data securely.

**2.4.2 Backend Technologies:**

**Core Java:**

Provides the base for developing the server-side logic and business operations.

**Spring Boot:**

A powerful framework built on top of Spring, used to develop stand-alone, production-ready backend applications. It supports rapid API development, dependency injection, and built-in server configuration.

**Spring Data JPA:**

A part of the Spring ecosystem used to simplify data persistence. It provides an abstraction over JPA (Java Persistence API) and reduces boilerplate for database access.

**Spring DevTools:**

Enhances the development experience with automatic restarts, live reloads, and configuration optimizations.

**Lombok:**

A Java library that reduces boilerplate code using annotations like @Getter, @Setter, @Data, and @NoArgsConstructor, making the code cleaner and easier to maintain.

5

**2.4.3 Database:**

**MySQL:**

A relational database management system used to store structured data including users, groups, expenses, and settlements. MySQL supports ACID compliance, foreign key relationships, and is well-suited for transactional data.

**2.4.4 Authentication and Security:**

**Spring Security:**

Provides secure user authentication and authorization mechanisms including session handling and password encryption.

**HTTPS Protocol:**

Ensures encrypted communication between the client and the server to protect sensitive user data.

**2.4.5 Email Service:**

**JavaMail with SMTP:**

Email functionality is integrated using the JavaMail API, configured with an SMTP server (e.g., Gmail). It is used to send:

\* Welcome emails after signup.

\* Login alerts for user awareness.

\* Reminders for unsettled balances.

**2.4.6 Development and Testing Tools:**

**Spring Tool Suite (STS):**

An IDE built on Eclipse, optimized for Spring development with built-in Spring Boot support and project scaffolding.

**Visual Studio Code:**

Used for writing and managing frontend code with support for JavaScript, React, and Git integration.

6

**MySQL Workbench:**

A graphical user interface for designing and managing MySQL databases, executing queries, and inspecting schema relationships.

**Git:**

Version control system used to manage and track changes to the source code across the project lifecycle.

7

# Chapter – 3: System Design

The system design of BillBuddy is structured to support secure user management, reliable expense tracking, flexible group collaboration, and responsive notifications. It adheres to modern software engineering principles, ensuring modularity, scalability, and maintainability.

**3.1 Architecture Design**

BillBuddy follows a **Three-Tier Client-Server Architecture**, with a clear separation between presentation, application logic, and data storage:

**Frontend (Client Layer):**

\* Developed using **React.js**, HTML, CSS, and JavaScript.

\* Communicates with backend RESTful APIs via secure **HTTPS** using **Axios**.

\* Handles all user interactions including authentication, viewing groups, adding expenses, and settling balances.

**Backend (Application Layer):**

Built using **Spring Boot**, which provides REST APIs and core business logic.

Manages:

\***Routing** of client requests.

\***Authentication** using Spring Security (optional).

\*Expense split calculations.

\*Group and user management\*.

\***Email dispatching** using JavaMail API.

\*Utilizes **Spring Data JPA** to interact with the database layer, minimizing boilerplate code and improving efficiency.

**Database (Data Layer):**

\***MySQL** is used to persist all structured data.

\*The backend interacts with the database using **JPA Entities and Repositories**, which map directly to tables and handle CRUD operations.

**3.2 Database Design**

The system follows a relational database schema modeled using **MySQL**. The design supports relationships between users, groups, expenses, and settlements.

8

**Users Table:**

\* user\_id (PK): Unique identifier.

\* name: User’s full name.

\* email: Registered email.

\* password: Encrypted user password.

\* created\_at: Registration timestamp.

**Groups Table**

\* group\_id (PK): Unique group identifier.

\* group\_name: Name of the group.

\* created\_by: FK referencing the user who created the group.

\* created\_at: Timestamp.

**Group\\_Members Table** (Many-to-Many relationship between users and groups)

\* id (PK)

\* user\_id: FK to Users.

\* group\_id: FK to Groups.

**Expenses Table**

\* expense\_id (PK): Unique expense identifier.

\* group\_id: FK to Groups.

\* title: Description of the expense.

\* amount: Total amount spent.

\* paid\_by: FK to Users (payer).

\* timestamp: When the expense was recorded.

**Expense\\_Splits Table**

\* id (PK)

\* expense\_id: FK to Expenses.

\* user\_id: FK to Users (each participant).

\* split\_amount: Amount owed by each participant.

**Settlements Table**

\* settlement\_id (PK)

\* group\_id: FK to Groups.

\* from\_user\_id: FK to Users (payer).

\* to\_user\_id: FK to Users (receiver).

\* amount: Amount settled.

\* timestamp: When the settlement occurred.

9

This relational structure ensures data integrity and supports efficient querying for balance tracking and settlement history.

**3.3 Frontend Design**

The frontend of BillBuddy is built using **React.js**, styled with CSS, and structured for intuitive navigation.

**Login/Signup Pages:**

\* Secure forms with input validations.

\* Password visibility toggles.

\* Feedback alerts on success/failure.

**User Dashboard:**

\* Displays all groups the user is part of.

\* Shows current balances (owed/owing).

\* Quick links to view group details.

**Group View Page:**

Shows:

\* Expense history.

\* Member list with balances.

\* Buttons for adding expenses and settling dues.

**Add Expense Modal:**

Pop-up form with:

\* Expense title.

\* Amount.

\* Payer selection.

\* Dynamic split logic (equal, custom, percentage).

\* Real-time calculation of who owes what.

Responsive Design:

\* Optimized for mobile, tablet, and desktop.

\* CSS Flexbox and Grid used for layout adaptability.

10

**3.4 Email System Design**

Email functionality is implemented using **JavaMail API** configured with **Gmail SMTP** for sending transactional emails.

**3.4.1 Functionality:**

\***Signup Email**:

Welcome message with account confirmation.

\***Login Email:**

Last login timestamp and alert for security.

\***Reminder Emails:**

\* Automatically sent to users who have pending dues in a group.

\* Option to customize reminder frequency (future scope).

**3.4.2 Technical Setup:**

\* SMTP host: smtp.gmail.com

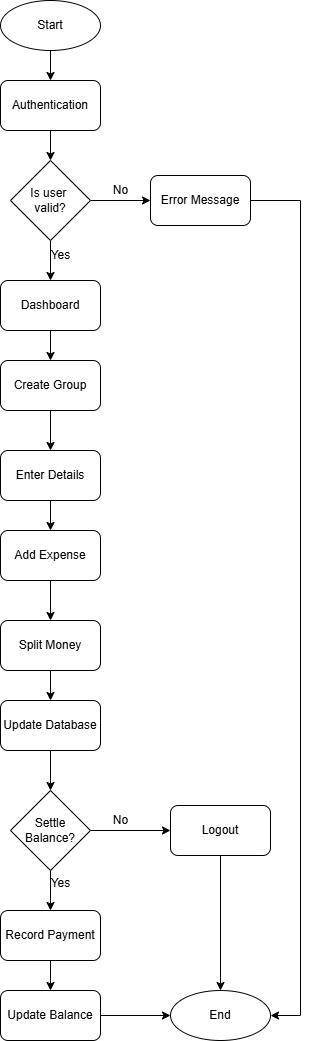
\* Port: 587 (TLS) or 465 (SSL)

\* JavaMail API integrated within Spring Boot.

\* Email content is templated using HTML for better readability.

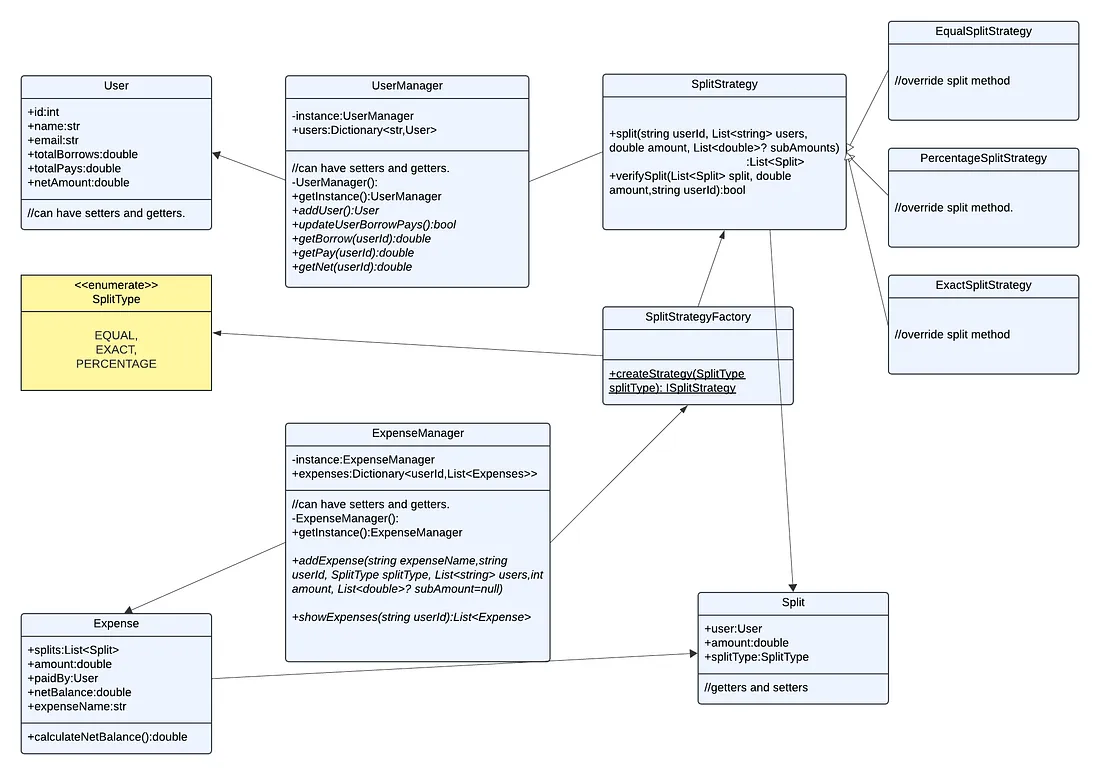
**3.5 Flowchart and Class Diagram For BillBuddy**

11



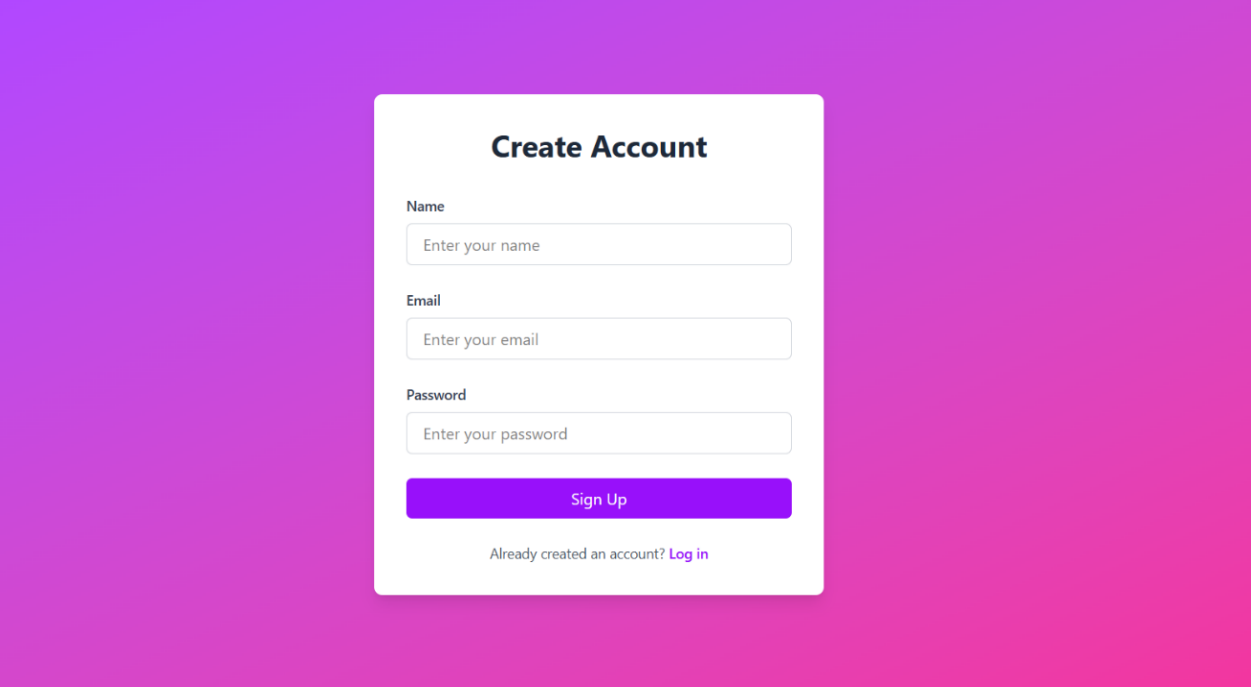
**3.5.1 Flowchart For BillBuddy**

12



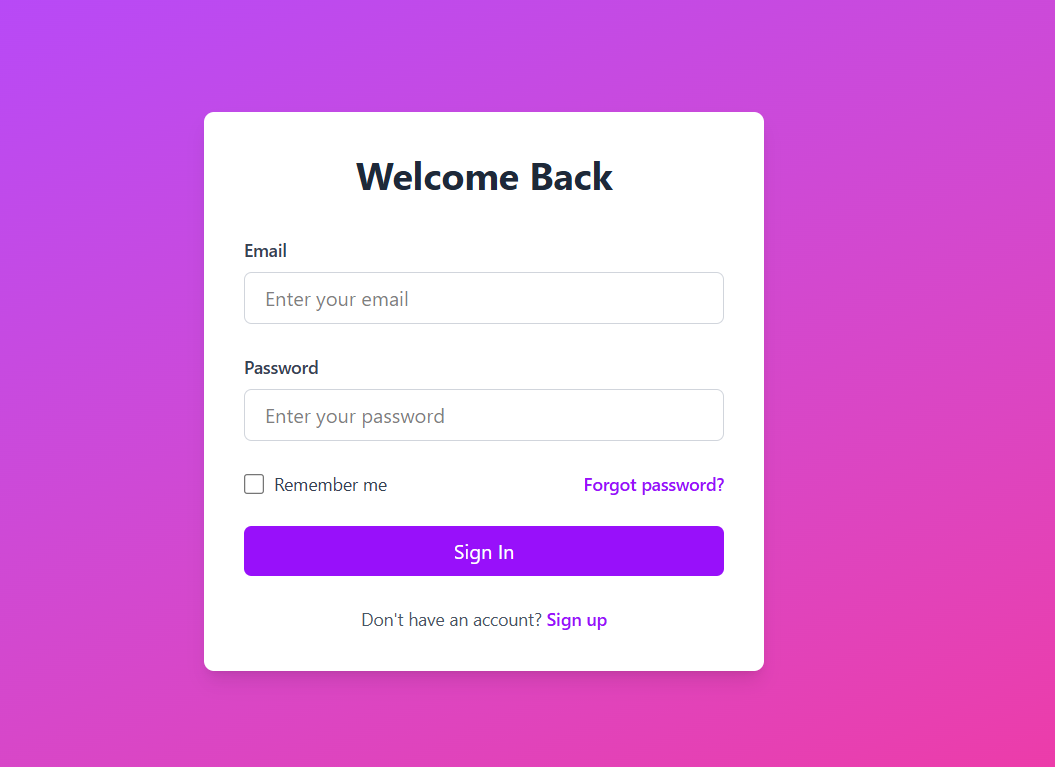
**3.5.2 Class Diagram For BillBuddy**

# 3.6 Snapshots of the project

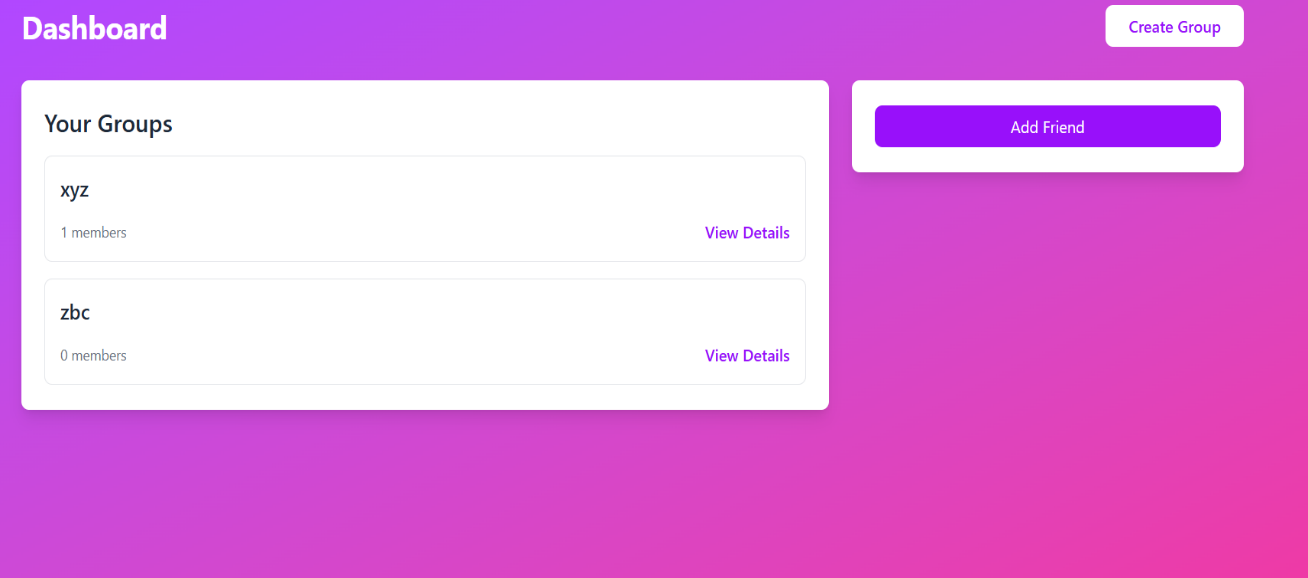


**3.6.1 Image showing the Sign-up page**

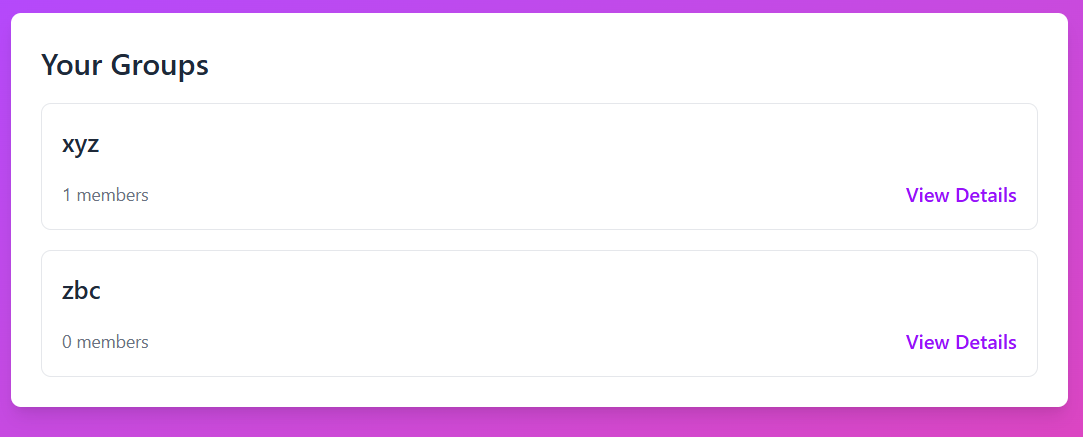
13



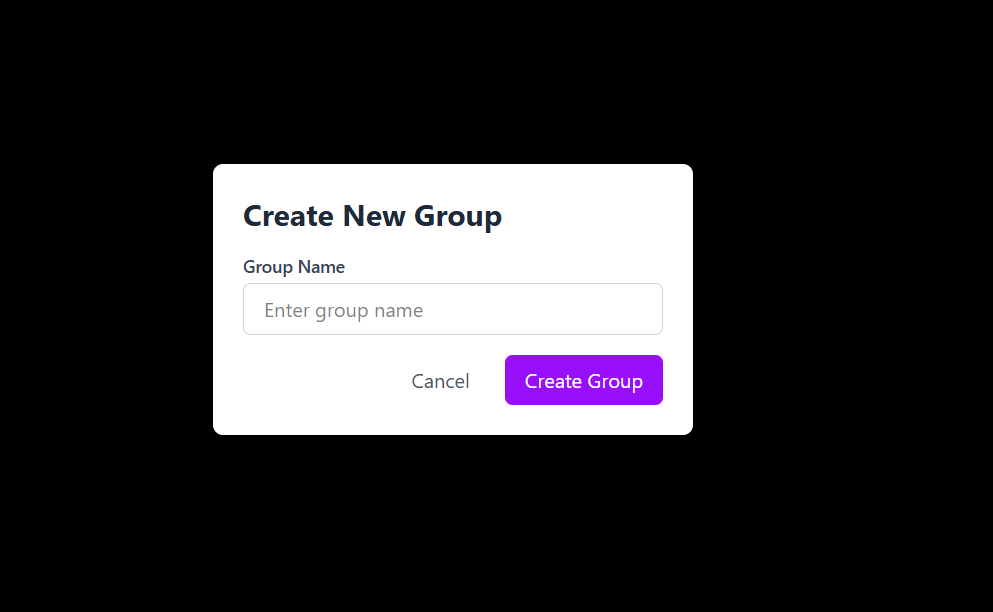
**3.6.2 Image showing the Login page**



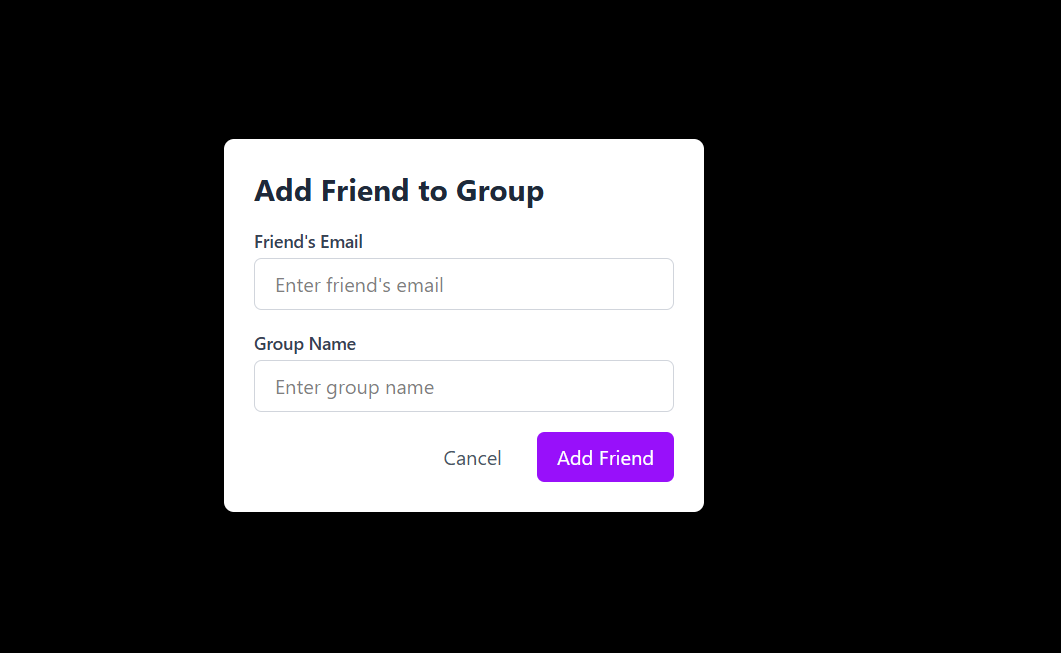
**3.6.3 Image showing the dashboard containing all the groups created and the option to add a friend**

14

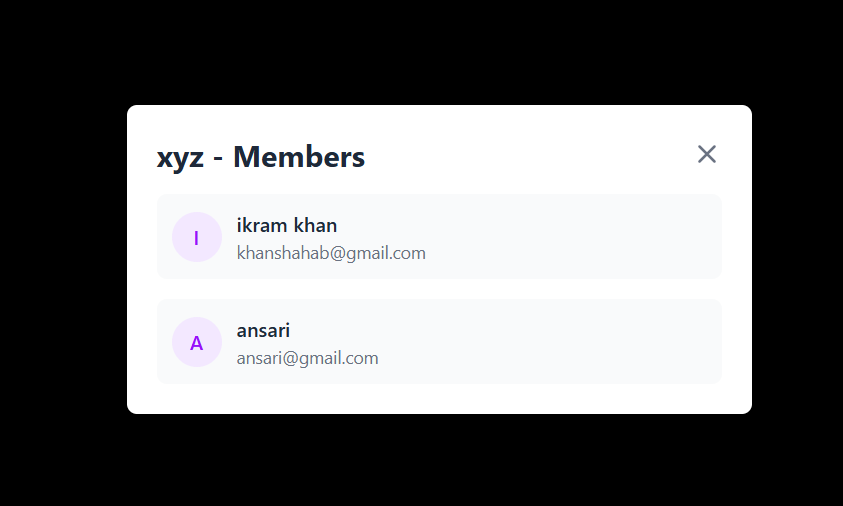
**3.6.4 Image showing all the Groups after they are created**



**3.6.5 Image showing the pop-up to create a group**

15

**3.6.6 Image showing the option to Add friends to a group**



**3.6.7 Image showing the members in a created group.**

16

# Chapter – 4: Systems Development

**4.1 Frontend Development**

The frontend of BillBuddy is developed using **React.js,** which provides a component-based structure and a reactive interface to improve user interaction.

**React.js with Functional Components and Hooks:**

The UI is built using functional components along with hooks such as useState, useEffect, and useContext to manage state and handle side effects. This enables dynamic rendering based on user interactions and data changes.

**Styling with Tailwind CSS:**

Tailwind CSS, a utility-first CSS framework, was used to ensure consistent styling and responsive design. It allows rapid UI development and helps maintain a clean and scalable CSS structure.

**Axios for API Integration**:

Axios is used for sending HTTP requests from the frontend to the backend. It handles all CRUD operations (GET, POST, PUT, DELETE) and ensures secure communication with backend services.

**User Authentication UI:**

A custom user interface was designed for login and signup, integrated with backend authentication services built using Spring Security. This ensures a seamless and secure authentication experience.

**4.2 Backend Development**

The backend was implemented using **Spring Boot**, which provides a robust framework for building RESTful APIs and managing business logic.

**Spring Boot and RESTful API Development:**

Spring Boot simplifies backend development by offering built-in configurations and dependencies. RESTful endpoints were created for user registration, login, group management, expense tracking, and settlement operations.

**Spring Data JPA with MySQL**:

**JPA (Java Persistence API)** is used for mapping Java objects to database tables. Spring Data JPA, in combination with **MySQL**, handles all database operations including querying, inserting, updating, and deleting records for users, groups, expenses, and balances.

17

**Authentication and Authorization**:

**Spring Security** is implemented to handle user authentication and authorization using secure login credentials stored in the database. Passwords are encrypted using hashing techniques to prevent data leaks.

**Middleware for Error Handling and Validation:**

Global exception handlers and middleware filters are used to manage errors such as invalid requests, unauthorized access, and data inconsistencies. This improves the stability and user-friendliness of the application.

**Email Notifications Using Nodemailer:**

Although the system does not use Firebase, email functionality is achieved via **Nodemailer**, configured with SMTP to send transactional emails for account verification, password reset, and activity alerts.

**Database Management with MySQL Workbench**:

The **MySQL Workbench** tool was used for designing the database schema, managing table relationships (using foreign keys), and running SQL scripts during testing. It helped visualize ER diagrams and optimize queries.

**4.3 Splitting Logic Implementation**

One of the core features of BillBuddy is its flexible expense splitting system, which supports different types of splits and ensures accurate and fair distribution of costs among users.

**Equal Split:**

This is the default mode where the total expense is evenly divided among all group members. The backend ensures each user's share is computed by dividing the total by the number of participants, with rounding logic applied if necessary.

**Custom Split:**

Users can manually enter how much each person owes for a particular expense. The system validates that the total entered shares match the overall expense amount. Errors are thrown if the totals don’t align.

**Percentage Split:**

Users specify what percentage of the total each participant pays. The backend calculates each user's share based on their entered percentage. All percentage inputs must sum up to 100%, which is enforced by validation logic.

18

**Validation Mechanisms:**

The backend includes rigorous validation to ensure data integrity. For example, in custom and percentage splits, the system checks that the sum of individual entries equals the total expense. If discrepancies arise, users are notified to correct their inputs.

**Rounding and Mathematical Accuracy**:

To avoid floating-point precision issues (e.g., due to currency rounding), the system uses rounding techniques to ensure the final computed shares are mathematically consistent. Any minor rounding errors are adjusted automatically to the smallest share to ensure exact totals.

19

# Summary and Conclusion

# The development of BillBuddy aimed to build a robust and user-friendly expense management system tailored for group settings. Through a modular and scalable architecture, the project integrates modern web technologies and best practices in software development.

# Project Highlights:

# Frontend Interface: A responsive and interactive UI built with React.js and Axios for seamless API communication.

# Backend Services: A RESTful API developed using Spring Boot, utilizing JPA for ORM and MySQL for data persistence.

# Authentication: JWT-based secure login system ensures user data privacy and controlled access.

# Core Functionalities:

# Expense addition and modification

# Bill splitting (equal or custom ratio)

# Group creation and management

# Debt calculation and simplification

# Expense history and downloadable reports

# Architecture: Follows a clean three-tier architecture separating concerns across the frontend, backend, and database layers.

# Developer Tools: STS and VS Code for coding, MySQL Workbench for database management, and Postman for API testing.

# By focusing on modularity, maintainability, and user experience, the project not only meets the initial objectives but also lays the foundation for further enhancements such as integrating payment gateways (e.g., UPI or PayPal), real-time notifications, and mobile responsiveness.

In conclusion, **BillBuddy** successfully demonstrates how modern full-stack development practices can be applied to solve a common and real-world problem—managing shared expenses. The project achieved its key objectives,

including user management, expense tracking, group handling, and debt simplification. Through the use of Java and Spring Boot for backend robustness, React.js for dynamic frontend behavior, and MySQL for reliable data handling, the application presents a complete and production-ready solution. The separation of concerns and adherence to best practices in both design and coding ensure that the system is scalable and maintainable.

20

This project has provided valuable hands-on experience in full-stack application development, API integration, database design, and secure authentication mechanisms. With further enhancements, such as real-time notifications and digital payment integration, BillBuddy can be transformed into a deployable product with real market potential.

21

**REFERENCES**

[1] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1994.

[2] M. Fowler, Patterns of Enterprise Application Architecture, Addison-Wesley, 2002.

[3] Mozilla Developer Network (MDN), “React – A JavaScript library for building user interfaces.” [Online]. Available: <https://developer.mozilla.org/en-US/docs/Web/JavaScript>. [Accessed: 7-May-2025].

[4] Firebase, “React Documentation.” [Online]. Available: https://reactjs.org/docs/gettingstarted.html. [Accessed: 3-May-2025].

[5] Splitwise, “**Splitwise Blog** – How expense splitting works.” [Online]. Available: https://blog.splitwise.com/. [Accessed: 3-May-2025].

22