**SRI KRISHNA COLLEGE OF TECHNOLOGY**

**An Autonomous Institution | Accredited by NAAC with 'A' Grade Affiliated to Anna University | Approved by AICTE**

**KOVAIPUDUR, COIMBATORE 641042**

**STARTUP VALIDATION**

**PORTAL**

**23CS503 – APP DEVELOPMENT**

**A PROJECT REPORT**

***Submitted by***

|  |  |  |
| --- | --- | --- |
| **KRITHICK BALA B** | **-** | **727823TUAD050** |

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

**BACHELOR OF TECHNOLOGY**

**IN**

**ARTIFICIAL INTELLIGENCE**

**AND DATA SCIENCE**

**OCTOBER 2025**

**SRI KRISHNA COLLEGE OF TECHNOLOGY**

**An Autonomous Institution | Accredited by NAAC with 'A' Grade Affiliated to Anna University | Approved by AICTE**

**KOVAIPUDUR, COIMBATORE 641042**

**BONAFIDE CERTIFICATE**

Certified that this project report “ **STARTUP VALIDATION PORTAL** **”** is the bonafide work of **“KRITHICK BALA B”** who carried out the project work under my supervision.

|  |  |
| --- | --- |
| **SIGNATURE** | **SIGNATURE** |
| **Dr. K. VIMALA** | **Dr. C. P. MAHESHWARAN** |
| **SUPERVISOR** | **HEAD OF THE DEPARTMENT** |
| Assistant Professor, | Professor, |
| Department of Artificial Intelligence and Data Science | Department of Artificial Intelligence and Data Science |
| Sri Krishna College of Technology, Coimbatore-641042. | Sri Krishna College of Technology, Coimbatore-641042. |

Certified that the candidates were examined by us in the Project Viva Voce examination held on at Sri Krishna College of Technology, Kovaipudur, Coimbatore -641042

|  |  |
| --- | --- |
| **INTERNAL EXAMINER** | **EXTERNAL EXAMINER** |

**ACKNOWLEDGEMENT**

First and foremost we thank the **Almighty** for being our light and for showering his gracious blessings throughout the course of this project.

We express our gratitude to our beloved Principal, **Dr. M.G. Sumithra,**

for providing all facilities.

With the grateful heart, our sincere thanks to our Head of the Department **Dr. C. P. Maheshwaran**, Department of Artificial Intelligence and Data Science for the motivation and all support to complete the project work.

We are greatly indebted to our Industry Mentor **Mr .K. Selvaraj** for his valuable guidance and suggestions in all aspects that aided us to ameliorate our skills.

We thank **Dr. K. Vimala,** Department of Artificial Intelligence and Data Science, for his motivation and support

We are thankful to all the **Teaching and Non-Teaching Staff** of Department of Artificial Intelligence and Data Science and to all those who have directly and indirectly extended their help to us in completing this project work successfully.

We extend our sincere thanks to our family members and our beloved friends, who had been strongly supporting us in all our endeavor.

**ABSTRACT**

The Expense Splitter Application for group expense management leverages modern web technologies such as React JS, Spring Boot, REST API, and MySQL to provide a seamless platform for tracking, splitting, and settling shared expenses. It delivers an intuitive, user-friendly interface that simplifies the process of recording expenses, calculating individual shares, and maintaining transparent financial records among group members. Key features include group creation and management, member addition, expense tracking, automated balance calculations, and settlement tracking. Administrators or group creators benefit from a feature-rich dashboard for viewing expense summaries, member balances, and payment histories, while members can easily log new expenses, view their share, and track settlement status in real-time.

The system integrates dynamic frontend components using React JS for responsive user experiences, robust RESTful APIs via Spring Boot for reliable data handling, and MySQL’s scalable database architecture for efficient storage and retrieval of expense records. Real-time updates ensure that members stay informed of any new expenses, balance changes, or settlement confirmations through intuitive UI elements and optional notifications. By automating calculations and streamlining expense management workflows, the application minimizes errors, eliminates manual reconciliation, and enhances transparency among participants.

Its scalability supports both small and large groups, adapting to growing data while maintaining performance and accuracy. Advanced reporting and analytics tools provide insights into spending patterns, enabling better financial planning within groups. In conclusion, the Expense Splitter Application is a modern, efficient, and user-focused solution designed to simplify group expense management, ensuring fairness, clarity, and convenience across diverse use cases such as trips, events, or shared living arrangements.

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **CHAPTER NO** | **TITLE** | **PAGE NO** |
| **1** | **INTRODUCTION** | **1** |
| **2** | **SYSTEM SPECIFICATION** | **2** |
| **3** | **PROPOSED SYSTEM** | **3** |
| **4** | **METHODOLOGIES** | **4** |
| **5** | **IMPLEMENTATION AND RESULTS** | **9** |
| **6** | **CONCLUSION AND FUTURE SCOPE** | **36** |
| **7** | **REFERENCES** | **37** |

**CHAPTER 1**

**INTRODUCTION**

This project aims to provide a seamless and efficient solution for managing shared expenses among groups through an online platform. In this chapter, we will discuss the problem statement, provide an overview, and outline the main objectives of the Expense Splitter Application.

* 1. **PROBLEM STATEMENT**

How can we develop an Expense Splitter Application that allows users to efficiently manage and track shared expenses among friends, family, or colleagues, ensuring accurate calculation of individual balances, handling multiple expense splits, and providing a simple and intuitive user interface?

**1.2 OVERVIEW**

Managing shared expenses manually can often be challenging, especially in groups with multiple members contributing to different expenses. Traditional methods such as paper records or manual calculations are prone to errors, time-consuming, and can lead to misunderstandings or disputes among group members.

To address these issues, we propose the development of an Expense Splitter Application. This system will leverage modern web technologies to provide a robust, user-friendly platform that automates expense tracking and calculation. Key features will include creating groups, adding members, recording expenses, splitting costs based on predefined rules, and generating balance summaries for each member. By incorporating these features, the system aims to enhance transparency, simplify financial management, and improve communication among group members.

**1.3 OBJECTIVE**

The primary objective of this project is to develop an Expense Splitter Application that provides users with a simple, efficient, and reliable platform for managing shared expenses. The system aims to:

* Automate the calculation of individual balances in shared expenses.
* Enable easy creation and management of groups and members.
* Minimize errors and conflicts in expense tracking.
* Provide an intuitive and interactive user interface for seamless experience.
* Enhance transparency and communication among group members regarding financial obligations.

**CHAPTER 2**

**SYSTEM SPECIFICATION**

In this chapter, we describe the software used to develop the **Expense Splitter Application**. This includes the tools and technologies that make the development, storage, and execution of the application efficient and user-friendly.

**2.1 VISUAL STUDIO CODE (VS CODE)**

**Visual Studio Code** is a source code editor developed by Microsoft for Windows, Linux, and macOS. It includes features such as debugging support, embedded Git control, syntax highlighting, intelligent code completion, code snippets, and code refactoring.

For the **Expense Splitter Application**, VS Code was used to write and manage both the **backend (Spring Boot)** and **frontend (React)** code. Its benefits include:

* **Lightweight and customizable:** Developers can change themes, keyboard shortcuts, and preferences.
* **Built-in support for JavaScript, JSX, and TypeScript:** Makes React development faster and easier.
* **Integrated terminal:** Allows executing commands without leaving the editor.
* **Extensions library:** Enables features like linting, debugging tools, and code snippets.

VS Code provides a robust development environment, improving productivity and reducing errors in building the Expense Splitter Application.

**2.2 LOCAL STORAGE**

**Local Storage** is a client-side storage mechanism that allows web applications to store data on a user’s browser. It is widely supported by all modern browsers such as Chrome, Firefox, Safari, and Edge.

In the **Expense Splitter Application**, local storage is used to:

* Store **user login sessions** so users remain logged in after refreshing the page.
* Save **temporary data** such as selected groups or partially entered expense details.
* Cache frequently accessed data to **improve performance**.

**Key features of Local Storage include:**

* **Persistent Storage:** Data remains available even after closing the browser.
* **Key-Value Storage:** Data is stored as key-value pairs, allowing quick retrieval.
* **Security:** Stored locally, making it inaccessible to external servers.
* **Efficiency:** Reduces server load by keeping data on the client-side.

Using local storage in the Expense Splitter Application ensures fast, reliable, and secure handling of user-specific data, such as member preferences and group details, without overloading the server.

**CHAPTER 3**

**PROPOSED SYSTEM**

This chapter gives a small description about the proposed idea behind the development of our application.

**3.1 PROPOSED SYSTEM**

The Expense Splitter system provides a practical solution for managing group expenses in an organized and transparent manner. The application enables users to create groups, add members, record expenses, and automatically calculate individual shares. This ensures fair distribution of costs, reducing confusion and disputes among members.

Once an expense is recorded, the system assigns the share to each group member and updates the balance accordingly. Users can view detailed summaries of expenses, outstanding amounts, and settlements in real time. The application also maintains a clear history of transactions, helping users keep track of contributions and pending balances.

This system eliminates the need for manual calculations, spreadsheets, or offline records, ensuring accuracy and saving time. Especially in group activities like trips, events, or shared living situations, the application simplifies the process of tracking expenses and payments.

Moreover, the system enhances financial transparency, allowing all group members to access and verify expenses and settlements. By leveraging automation for expense distribution and notifications, the Expense Splitter ensures efficient management of shared finances, thereby improving user convenience and satisfaction.

**3.2 ADVANTAGES**

* **Efficiency:** The Expense Splitter application automates the calculation of shared expenses, eliminating manual calculations and reducing errors. This improves accuracy and saves time for users.
* **Transparency:** Every group member has access to recorded expenses, settlements, and outstanding balances. This transparency minimizes misunderstandings and promotes trust among members.
* **Fairness:** The system ensures that expenses are divided fairly among group members according to their shares, avoiding disputes and ensuring equal contribution.
* **Accessibility:** The application can be accessed from any device with internet connectivity, making it convenient for users to manage expenses remotely.
* **Real-Time Updates:** Any new expense, settlement, or adjustment is instantly updated in the system, ensuring that all members are aware of their current balances and dues.

**CHAPTER 4**

**METHODOLOGIES**

**User Registration:**

Users register on the platform by providing essential details such as name, email address, and password. Optional information like phone number may also be included for better communication. Registration ensures that all expenses and groups are associated with authenticated users.

**Role-Based Access:**

Upon registration/login, users are assigned roles such as Group Owner or Member, which determine their access permissions within the system. Group owners can create and manage groups and expenses, while members can view and participate in group activities.

**Group Management:**  
Users can create groups for expense sharing and invite other members to join. Groups provide a central platform to manage shared expenses and track balances among participants.

**Expense Addition:**  
Users can add expenses specifying details such as amount, description, date, and the members involved. Each expense is linked to a specific group and automatically updates the balances of involved members.

**Expense Splitting:**  
The system calculates and splits expenses among group members based on predefined rules (e.g., equally, percentage-wise, or custom share). This ensures transparency and accurate tracking of individual balances.

**Expense Updates and Deletion:**  
Users can update or delete expenses to reflect changes. The system automatically recalculates balances to maintain consistency among group members.

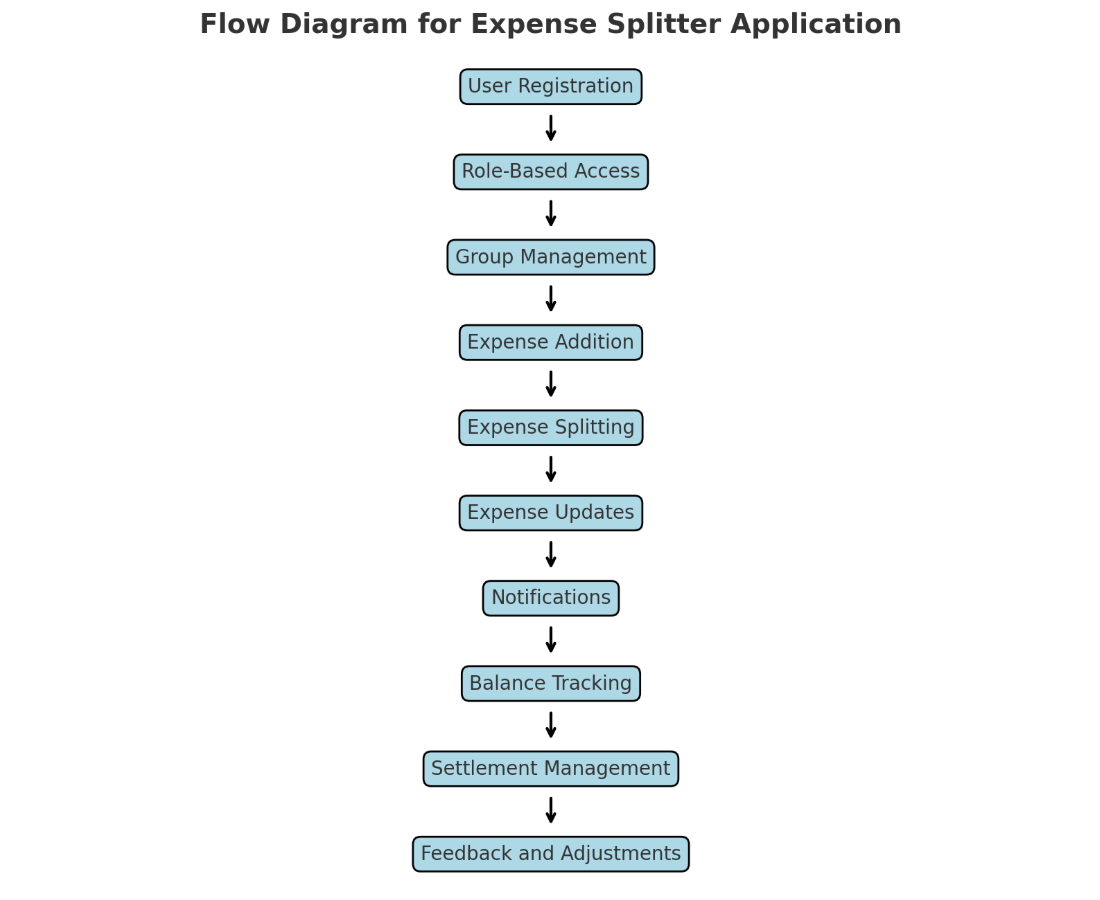
**Notifications:**  
After adding, updating, or settling expenses, the system generates notifications to inform members about the changes in group balances or newly added expenses.

**Balance Tracking:**  
Users can view real-time balances for each group and individual members. The platform provides summaries of amounts owed or to be received, helping users settle debts efficiently.

**Settlement Management:**Users can mark payments as settled, which updates the system and maintains an accurate record of debts and repayments.

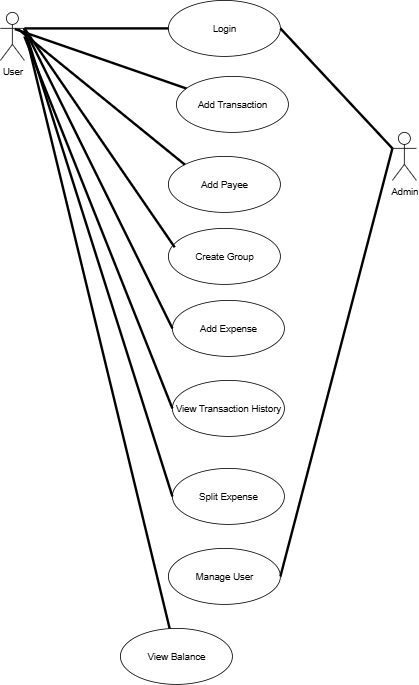
**Feedback and Adjustments:**  
After interacting with the application, users can provide feedback on group management or expense splitting. This helps in improving the usability and functionality of the platform.

**Flow Diagram:**

****

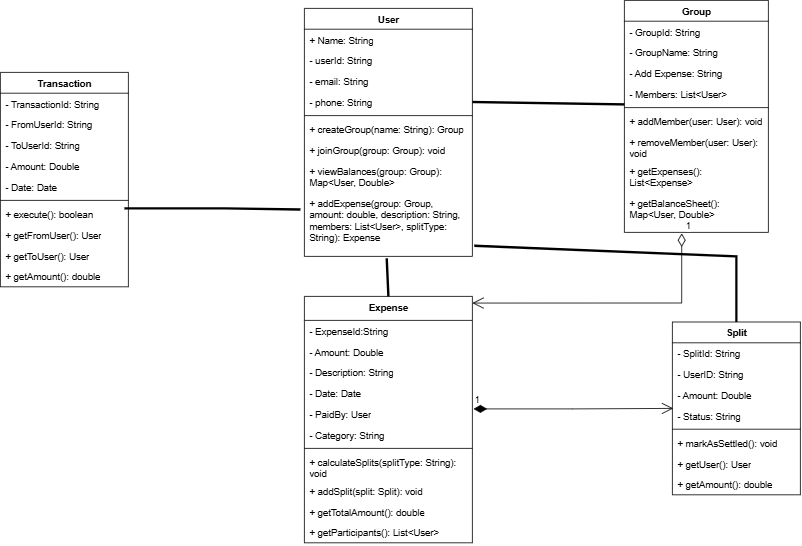
**Use Case Diagram:**

This diagram outlines the roles and responsibilities in an expense splitter application. Admin can manage users, monitor groups and expenses, handle disputes, and oversee settlements. User can create groups, add members, record and update expenses, view balances, and settle payments. Group Member can contribute expenses, check their share, track balances, and receive notifications about updates or settlements. The System automatically calculates shares, generates balance reports, and sends reminders to ensure smooth and transparent expense management**.**

****

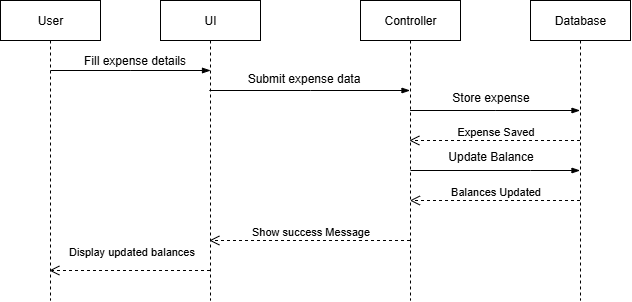
**Class Diagram:**

In the Expense Splitter Application, Users can register and create multiple groups, with each group consisting of several Members. Groups contain multiple Expenses, which are added by users and linked to both groups and members. Each expense includes details such as amount, payer, and date, and the system uses this to calculate Balances showing how much each member owes or is owed. Members can also settle payments, and the system maintains settlement history to ensure fair and transparent expense management.

****

**Sequence Diagram:**

This diagram shows an expense splitter application where a User can register or log in and create multiple Groups. Each Group includes several Members, and users can add Expenses with details like amount, payer, and description. The System automatically calculates each member’s share and updates their Balances, reflecting how much they owe or are owed. Members can then initiate Settlements, and the system records the settlement history. The interactions between users, groups, expenses, balances, and settlements ensure fair distribution of costs and transparent expense management.

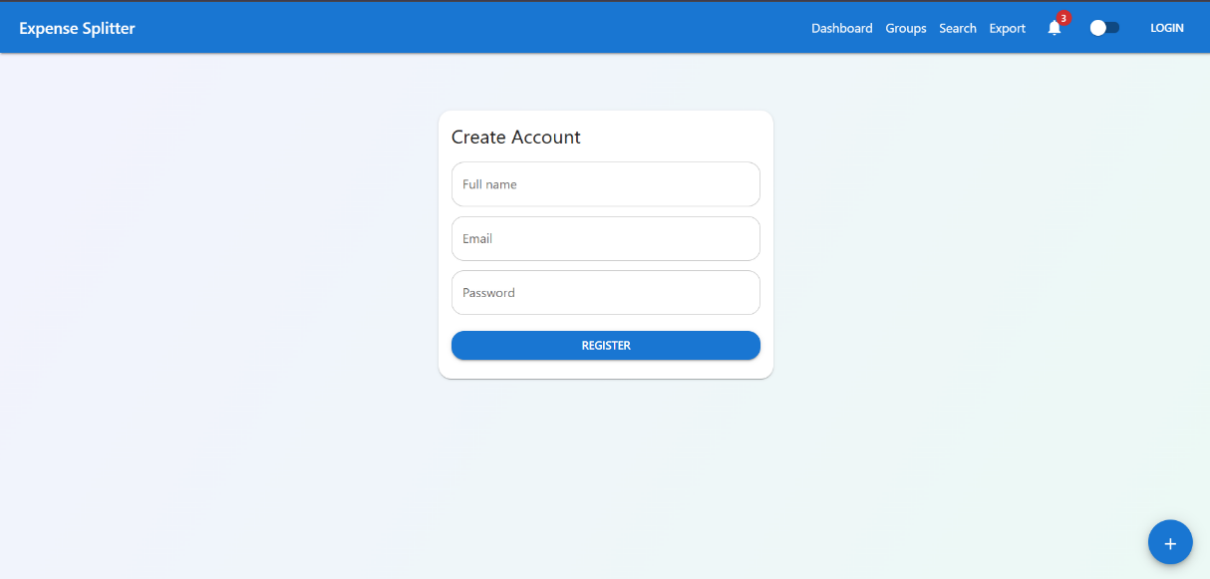
****

**CHAPTER 5**

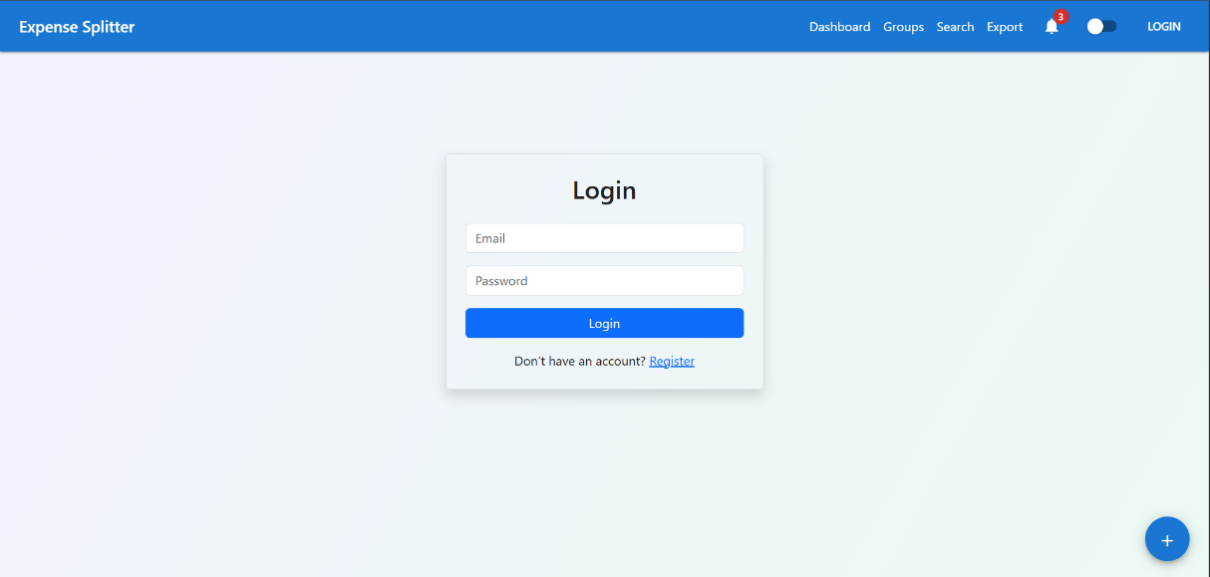
**IMPLEMENTATION AND RESULT**

Thus chapter gives a description about the output that we produced by developing the website of our idea**.**

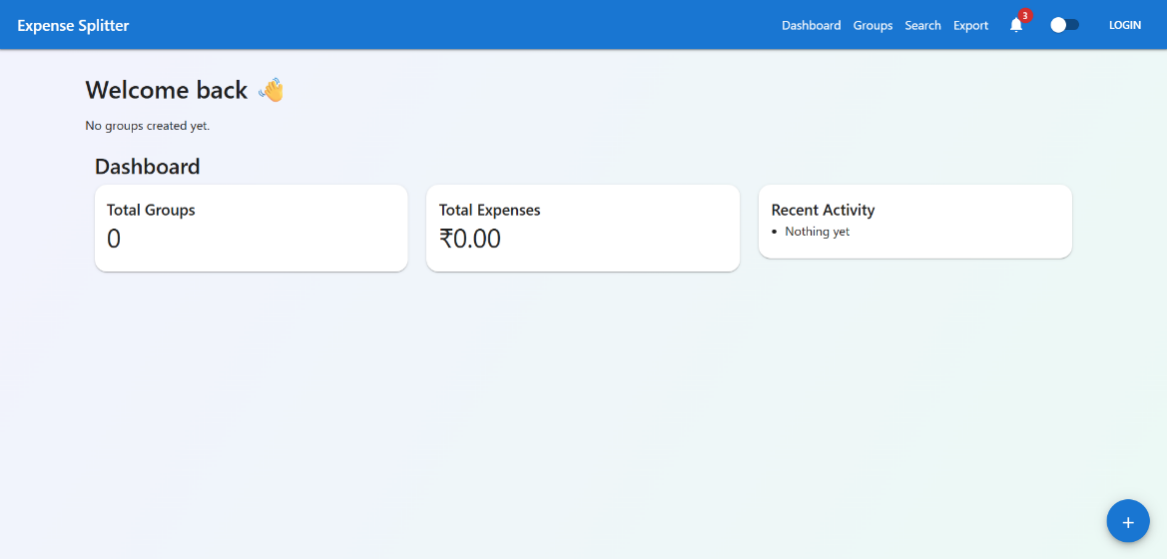
**5.1 REGISTER**

****

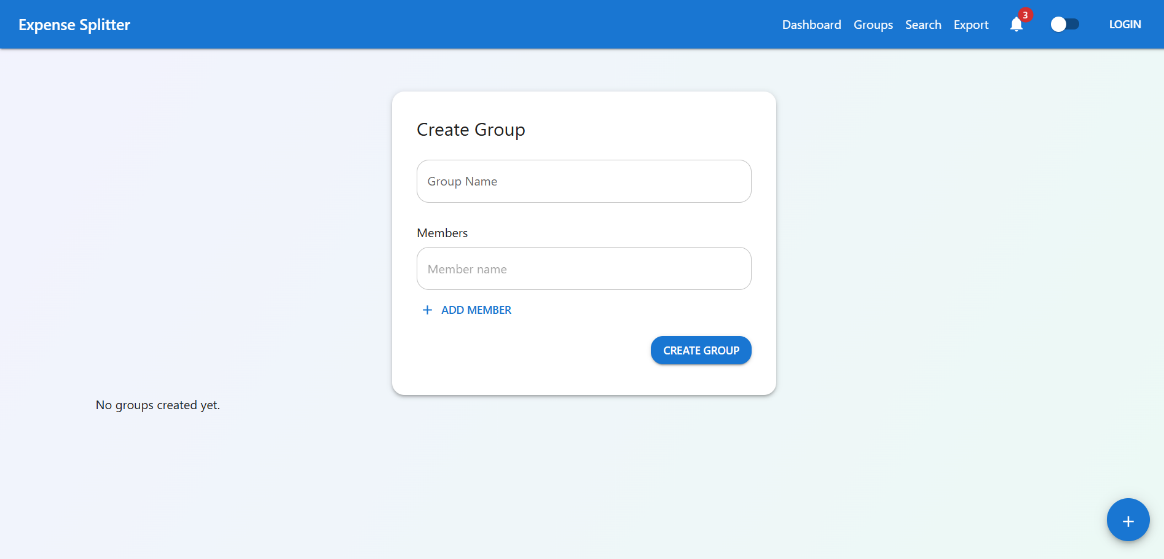
**5.2 LOGIN**

****

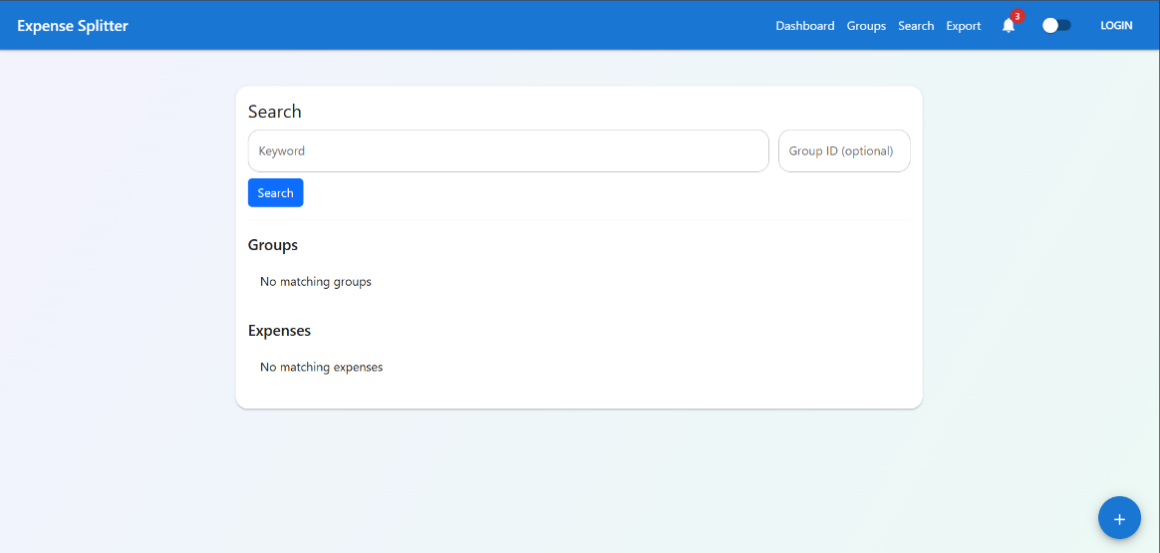
**5.3 DASHBOARD**

****

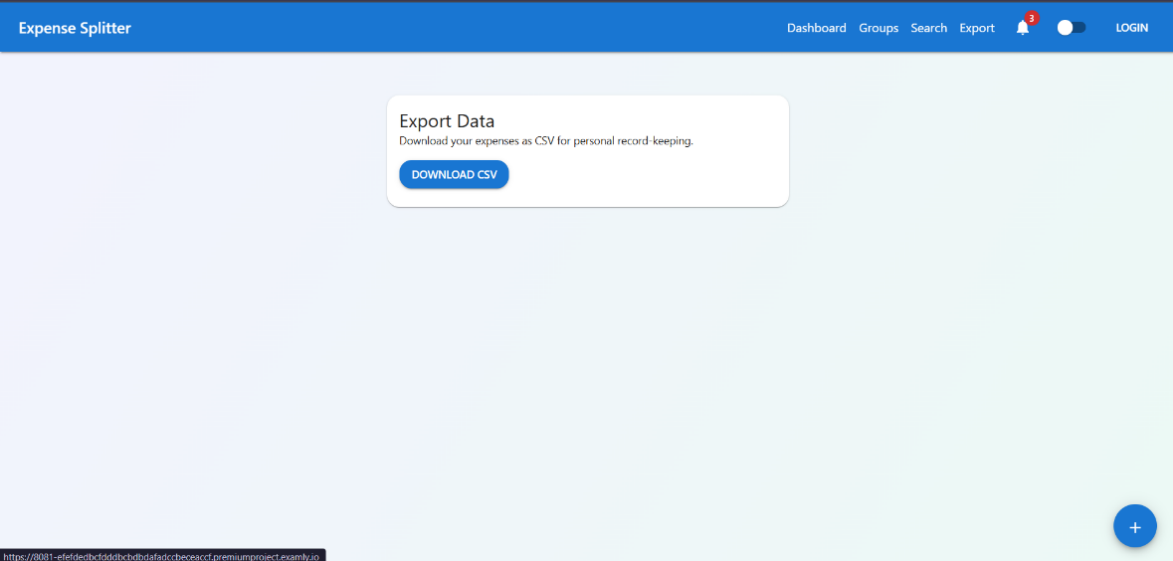
**5.4 CREATE GROUP**

****

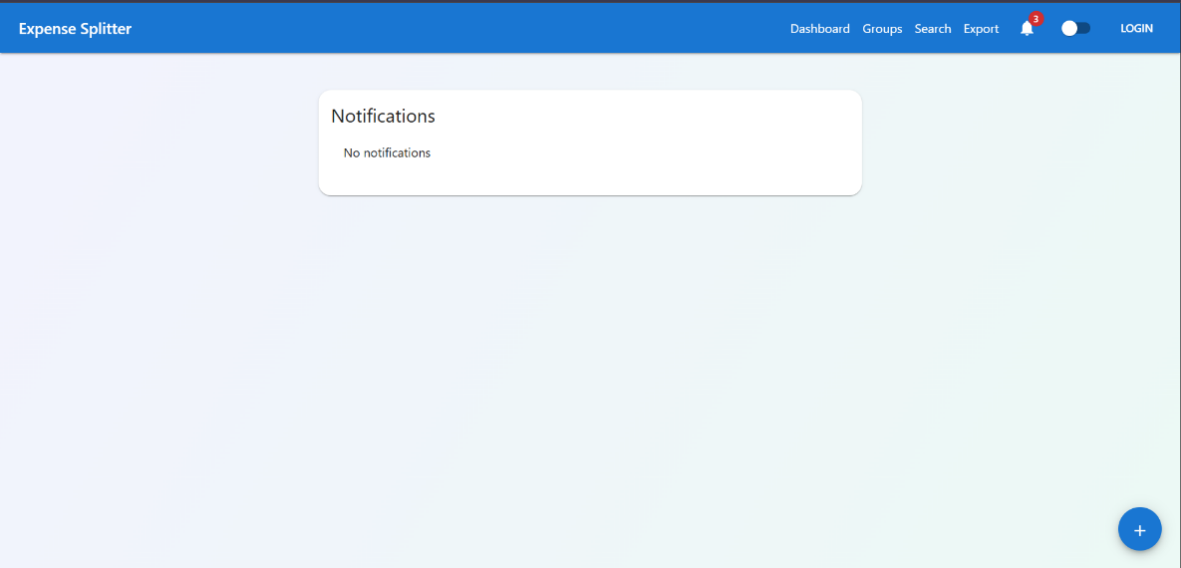
**5.5 SEARCH**

****

**5.6 EXPORT DATA**

****

**5.7 NOTIFICATION**

****

**5.8 CODING**

**FRONTEND CODING**

**AddExpenseForm.jsx**

import React, { useState } from "react";

export default function AddExpenseForm({ groupId, members, onAddExpense, postAdd }) {

    const [desc, setDesc] = useState("");

    const [amount, setAmount] = useState("");

    const [payer, setPayer] = useState("");

    const [date, setDate] = useState("");

    const [error, setError] = useState("");

    const handleSubmit = async () => {

        setError("");

        if (!desc || !amount || parseFloat(amount) <= 0 || !payer || !date) {

setError("Fill all fields correctly");

return;

}

try {

await onAddExpense(groupId, desc, parseFloat(amount), payer, date);

postAdd();

setDesc("");

setAmount("");

setPayer("");

setDate("");

} catch (err) {

setError(err.message);

}

};  
return (

<div data-testid="add-expense-form" className="p-4 border rounded bg-light shadow">

<h5>Add Expense</h5>

{error && <div className="alert alert-danger">{error}</div>}

<input

data-testid="desc-input"

type="text"

className="form-control mb-2"

placeholder="Description"

value={desc}

onChange={(e) => setDesc(e.target.value)}

/>

<input

data-testid="amount-input"

type="number"

className="form-control mb-2"

placeholder="Amount"

value={amount}

onChange={(e) => setAmount(e.target.value)}

/>

<select

data-testid="payer-select"

className="form-control mb-2"

value={payer}

onChange={(e) => setPayer(e.target.value)}

>

<option value="">Select Payer</option>

{members.map((m) => (

<option key={m} value={m}>

{m}

</option>

))}

</select>

<input

data-testid="date-input"

type="date"

className="form-control mb-2"

value={date}

onChange={(e) => setDate(e.target.value)}

/>

<button data-testid="add-expense-button" className="btn btn-primary mt-2" onClick={handleSubmit}>

Add Expense

</button>

</div>

);

}

**AddMemeberForm.jsx**

import React, { useState } from 'react';

export default function AddMemberForm({ groupName, onAddMember, postAdd }) {

const [name, setName] = useState('');

const [error, setError] = useState('');

const handleSubmit = async () => {

setError('');

if (!name.trim()) {

setError('Member name required');

return;

}

try {

await onAddMember(groupName, name.trim());

postAdd();

setName(''); // clear input after success

} catch (err) {

setError(err.message);

}

};

return (

<div

data-testid="add-member-form"

className="p-3 border rounded bg-light mt-3 shadow-sm"

>

<h5 className="mb-3 text-secondary">Add Member</h5>

{error && (

<div className="alert alert-danger py-1 my-2" data-testid="error-message">

{error}

</div>

)}

<div className="input-group mb-2">

<input

data-testid="member-input"

type="text"

className="form-control"

placeholder="Member Name"

value={name}

onChange={(e) => setName(e.target.value)}

/>

<button

data-testid="add-member-button"

className="btn btn-secondary"

type="button"

onClick={handleSubmit}

>

Add

</button>

</div>

</div>

);

}

**Balances.jsx**

import React from 'react';

export default function Balances({ balances }) {

if (!balances || balances.length === 0) {

return <p>No balances to show.</p>;

}

return (

<div data-testid="balances-section" className="mt-3">

<h5>Balances</h5>

<ul data-testid="balances-list" className="list-group">

{balances.map((b) => (

<li key={b.member} className="list-group-item">

{`${b.member}: ₹${b.balance >= 0 ? '+' : ''}${b.balance.toFixed(2)}`}

</li>

))}

</ul>

</div>

);

}

**GroupDetails.jsx**

import React from 'react';

const GroupDetail = ({ group, members, expenses, balances, onAddMember, onAddExpense }) => {

const [memberName, setMemberName] = React.useState('');

const [desc, setDesc] = React.useState('');

const [amount, setAmount] = React.useState('');

const [payer, setPayer] = React.useState('');

const [date, setDate] = React.useState('');

const handleAddMember = () => {

if (memberName.trim()) {

onAddMember(memberName.trim());

setMemberName('');

}

};

const handleAddExpense = () => {

if (desc && amount && payer && date) {

onAddExpense({ description: desc, amount: parseFloat(amount), payer, date });

setDesc('');

setAmount('');

setPayer('');

setDate('');

}

};

return (

<div className="p-3" data-testid="group-detail">

<h4>{group?.groupName}</h4>

<h5>Members</h5>

<div data-testid="group-members">

{members.length > 0 ? (

members.map((m) => (

<p key={m} data-testid={`member-item-${m}`}>{m}</p>

))

) : (

<p>No members.</p>

)}

</div>

<div className="p-3 border rounded bg-light" data-testid="add-member-form">

<h5>Add Member</h5>

{/\* Fix 2: change to member-input \*/}

<input

className="form-control mb-2"

data-testid="member-input"

placeholder="Member Name"

value={memberName}

onChange={(e) => setMemberName(e.target.value)}

/>

<button

className="btn btn-primary"

data-testid="add-member-button"

onClick={handleAddMember}

>

Add Member

</button>

</div>

<h5 className="mt-3">Expenses</h5>

<ul className="list-group" data-testid="expense-list">

{expenses.length > 0 ? (

expenses.map((exp) => (

<li key={exp.expenseId} className="list-group-item">

<span>{exp.description}</span>{' '}

<span>₹{exp.amount.toFixed(2)}</span>

</li>

))

) : (

<p>No expenses added yet.</p>

)}

</ul>

<div className="p-3 border rounded bg-light" data-testid="add-expense-form">

<h5>Add Expense</h5>

<input

className="form-control mb-2"

data-testid="desc-input"

placeholder="Description"

value={desc}

onChange={(e) => setDesc(e.target.value)}

/>

<input

className="form-control mb-2"

data-testid="amount-input"

placeholder="Amount"

type="number"

value={amount}

onChange={(e) => setAmount(e.target.value)}

/>

<select

className="form-control mb-2"

data-testid="payer-select"

value={payer}

onChange={(e) => setPayer(e.target.value)}

>

<option value="">Select Payer</option>

{members.map((m) => (

<option key={m} value={m}>{m}</option>

))}

</select>

<input

className="form-control mb-2"

data-testid="date-input"

type="date"

value={date}

onChange={(e) => setDate(e.target.value)}

/>

<button

className="btn btn-primary"

data-testid="add-expense-button"

onClick={handleAddExpense}

>

Add Expense

</button>

</div>

<div className="mt-3" data-testid="balances-section">

<h5>Balances</h5>

{balances.length > 0 ? (

balances.map((b) => (

<p key={b.member}>

{b.member}: ₹{b.balance.toFixed(2)}

</p>

))

) : (

<p>No balances to show.</p>

)}

</div>

</div>

);

};

export default GroupDetail;

**GroupList.jsx**

import React from 'react';

export default function GroupList({ groups, onSelect, selectedGroupId }) {

if (!groups || groups.length === 0) {

return (

<div data-testid="empty-groups">

<p>No groups created yet.</p>

</div>

);

}

return (

<ul data-testid="group-list" className="list-group">

{groups.map((g) => (

<li

key={g.groupId}

data-testid={`group-item-${g.groupId}`}

className={`list-group-item ${selectedGroupId === g.groupId ? 'active' : ''}`}

onClick={() => onSelect(g.groupId)}

style={{ cursor: 'pointer' }}

>

<div>{g.groupName}</div>

<small>({g.members.length} members)</small>

</li>

))}

</ul>

);

}

**Login.jsx**

import React, { useState } from "react";

import { useNavigate, Link } from "react-router-dom";

import api from "../utils/api"; // default import

function Login({ onLogin }) {

const [email, setEmail] = useState("");

const [password, setPassword] = useState("");

const navigate = useNavigate();

const handleLogin = async (e) => {

e.preventDefault();

try {

const res = await api.login({ email, password });

if (onLogin) onLogin(res);

navigate("/groups"); // redirect after login

} catch (err) {

console.error("Login failed", err);

alert("Invalid email or password");

}

};

return (

<div className="d-flex justify-content-center align-items-center vh-100">

<form

onSubmit={handleLogin}

className="p-4 border rounded shadow"

style={{ minWidth: "300px", maxWidth: "400px", width: "100%" }}

>

<h2 className="mb-4 text-center">Login</h2>

<div className="mb-3">

<input

type="email"

className="form-control"

placeholder="Email"

value={email}

onChange={(e) => setEmail(e.target.value)}

required

/>

</div>

<div className="mb-3">

<input

type="password"

className="form-control"

placeholder="Password"

value={password}

onChange={(e) => setPassword(e.target.value)}

required

/>

</div>

<button type="submit" className="btn btn-primary w-100">

Login

</button>

<div className="mt-3 text-center">

Don't have an account? <Link to="/register">Register</Link>

</div>

</form>

</div>

);

}

export default Login;

**Register.jsx**

import React, { useState } from "react";

import { Card, CardContent, TextField, Button, Typography } from "@mui/material";

import { useNavigate } from "react-router-dom";

import { useAuth } from "../context/AuthContext";

export default function Register() {

const { register } = useAuth();

const nav = useNavigate();

const [form, setForm] = useState({ name: "", email: "", password: "" });

const [msg, setMsg] = useState("");

const submit = async () => {

setMsg("");

try {

await register(form);

setMsg("✅ Registered! You can login now.");

setTimeout(() => nav("/login"), 1000);

} catch (e) {

setMsg(`❌ ${e.message}`);

}

};

return (

<Card sx={{ maxWidth: 420, mx: "auto", mt: 6 }}>

<CardContent>

<Typography variant="h5" gutterBottom>Create Account</Typography>

{msg && (

<Typography

sx={{ mb: 1 }}

color={msg.startsWith("✅") ? "success.main" : "error.main"}

>

{msg}

</Typography>

)}

<TextField

label="Full name"

name="name"

fullWidth

margin="dense"

value={form.name}

onChange={(e) => setForm({ ...form, name: e.target.value })}

/>

<TextField

label="Email"

name="email"

fullWidth

margin="dense"

value={form.email}

onChange={(e) => setForm({ ...form, email: e.target.value })}

/>

<TextField

label="Password"

name="password"

type="password"

fullWidth

margin="dense"

value={form.password}

onChange={(e) => setForm({ ...form, password: e.target.value })}

/>

<Button fullWidth variant="contained" sx={{ mt: 2 }} onClick={submit}>

Register

</Button>

</CardContent>

</Card>

);

}

**Api.jsx**

import axios from "axios";

const API\_BASE = "https://8080-efefdedbcfdddbcbdbdafadccbeceaccf.premiumproject.examly.io";

const api = axios.create({

baseURL: API\_BASE,

});

api.interceptors.request.use((config) => {

const token = localStorage.getItem("token") || localStorage.getItem("authToken"); // check both

if (token) {

config.headers["Authorization"] = `Bearer ${token}`;

}

return config;

},

(error)=>Promise.reject(error));

export const login = async (credentials) => {

const res = await api.post("/api/auth/login", credentials);

const token = res.data.token || res.data.jwt || res.data.authToken;

if (token) {

localStorage.setItem("token", token);

}

if (res.data.role) {

localStorage.setItem("role", res.data.role);

}

if (res.data.name) {

localStorage.setItem("name", res.data.name);

}

return res.data;

};

export const register = async (userData) => {

const res = await api.post("/api/auth/register", userData);

return res.data;

};

export const me = async () => {

const res = await api.get("/api/auth/me");

return res.data;

};

export const fetchGroups = async () => {

const res = await api.get("/groups");

return res.data;

};

export const createGroup = async (groupData) => {

const res = await api.post("/groups", groupData);

return res.data;

};

export const addExpense = async (groupId, expenseData) => {

const res = await api.post(`/groups/${groupId}/expenses`, expenseData);

return res.data;

};

export default {

login,

register,

me,

fetchGroups,

createGroup,

addExpense

};

**App.jsx**

import React, { useState, useEffect, useMemo } from "react";

import { BrowserRouter as Router, Routes, Route, Navigate } from "react-router-dom";

import "bootstrap/dist/css/bootstrap.min.css";

import { ThemeProvider, createTheme, CssBaseline } from "@mui/material";

import { AuthProvider } from "./context/AuthContext";

import Navbar from "./components/Navbar";

import Layout from "./components/Layout";

import QuickActions from "./components/QuickActions";

import Dashboard from "./components/Dashboard";

import Home from "./pages/Home";

import Groups from "./pages/Groups";

import GroupDetailPage from "./pages/GroupDetailPage";

import AddExpensePage from "./pages/AddExpensePage";

import CreateGroupPage from "./components/CreateGroupForm";

import NotificationsPage from "./pages/NotificationsPage";

import Search from "./pages/Search";

import Export from "./pages/Export";

import Profile from "./pages/Profile";

import Login from "./pages/Login";

import Register from "./pages/Register";

import { fetchGroups, createGroup, addExpense } from "./utils/api";

export default function App() {

const [mode, setMode] = useState("light");

const [groups, setGroups] = useState([]);

const theme = useMemo(

() =>

createTheme({

palette: { mode },

shape: { borderRadius: 16 },

typography: {

fontFamily: `"Inter", system-ui, -apple-system, Segoe UI, Roboto, Arial`,

},

}),

[mode]

);

const fetchAllGroups = async () => {

try {

const res = await fetchGroups();

setGroups(res || []);

} catch (err) {

console.error("Error fetching groups:", err);

}

};

const handleCreateGroup = async (groupName) => {

try {

const res = await createGroup({ groupName });

setGroups((prev) => [...prev, res]);

} catch (err) {

console.error("Error creating group:", err);

const backendError = err.response?.data?.error || err.message || "Unknown error";

alert(`Failed to create group: ${backendError}`);

}

};

const handleAddExpense = async (groupId, expense) => {

try {

await addExpense(groupId, expense);

fetchAllGroups();

} catch (err) {

console.error("Error adding expense:", err);

}

};

useEffect(() => {

fetchAllGroups();

}, []);

return (

<AuthProvider>

<ThemeProvider theme={theme}>

<CssBaseline />

<Router>

<Navbar

mode={mode}

onToggleMode={() => setMode((m) => (m === "light" ? "dark" : "light"))}

/>

<Layout>

<Routes>

<Route path="/" element={<Home />} />

<Route path="/login" element={<Login onLogin={fetchAllGroups} />} />

<Route path="/register" element={<Register />} />

<Route path="/dashboard" element={<Dashboard groups={groups} />} />

<Route

path="/groups"

element={

<Groups

groups={groups}

fetchGroups={fetchAllGroups}

onAddExpense={handleAddExpense}

onCreateGroup={handleCreateGroup}

/>

}

/>

<Route

path="/groups/create"

element={<CreateGroupPage onCreateGroup={handleCreateGroup} />}

/>

<Route path="/groups/:groupId" element={<GroupDetailPage />} />

<Route

path="/groups/:groupId/expenses"

element={<AddExpensePage groups={groups} onAddExpense={handleAddExpense} />}

/>

<Route path="/search" element={<Search />} />

<Route path="/notifications" element={<NotificationsPage />} />

<Route path="/export" element={<Export />} />

<Route path="/profile" element={<Profile />} />

<Route path="\*" element={<Navigate to="/" replace />} />

</Routes>

</Layout>

<QuickActions onCreateGroup={handleCreateGroup} onAddExpense={handleAddExpense} />

</Router>

</ThemeProvider>

</AuthProvider>

);

}

**BACKEND CODING**

**CorsConfig.java**

package com.examly.springapp.config;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.web.servlet.config.annotation.CorsRegistry;

import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

@Configuration

public class CorsConfig {

@Bean

public WebMvcConfigurer corsConfigurer() {

return new WebMvcConfigurer() {

@Override

public void addCorsMappings(CorsRegistry registry) {

registry.addMapping("/\*\*")

.allowedOrigins("\*") // You can restrict later to your frontend URL

.allowedMethods("GET", "POST", "PUT", "DELETE", "OPTIONS")

.allowedHeaders("\*")

.allowCredentials(false);

}

};

}

}

**GroupController.java**

package com.examly.springapp.controller;

import com.examly.springapp.model.Group;

import com.examly.springapp.model.Member;

import com.examly.springapp.service.GroupService;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.\*;

import org.springframework.web.bind.annotation.\*;

import java.util.\*;

@RestController

@RequestMapping("/groups")

public class GroupController {

@Autowired

private GroupService groupService;

@GetMapping("/{id}")

public ResponseEntity<?> getGroupById(@PathVariable Long id) {

Optional<Group> groupOpt = groupService.getGroupById(id);

if (groupOpt.isEmpty()) {

Map<String, String> error = new HashMap<>();

error.put("error", "Group not found");

return ResponseEntity.status(HttpStatus.NOT\_FOUND).body(error);

}

Group g = groupOpt.get();

Map<String, Object> groupMap = new LinkedHashMap<>();

groupMap.put("groupId", g.getGroupId());

groupMap.put("groupName", g.getGroupName());

groupMap.put("members", g.getMembers()); // ensures it’s an array in JSON

return ResponseEntity.ok(groupMap);

}

@PostMapping

public ResponseEntity<?> createGroup(@RequestBody Map<String, Object>

request) {

try {

String groupName = (String) request.get("groupName");

Object membersRaw = request.get("members");

List<String> members = new ArrayList<>();

if (membersRaw instanceof List<?>) {

for (Object obj : (List<?>) membersRaw) {

if (obj instanceof String) {

members.add((String) obj);

} else {

throw new IllegalArgumentException("Each member must be a string.");

}

}

} else {

throw new IllegalArgumentException("Members should be a list.");

}

Group group = groupService.createGroup(groupName, members);

Map<String, Object> response = new LinkedHashMap<>();

response.put("groupId", group.getGroupId());

response.put("groupName", group.getGroupName());

response.put("members", group.getMembers());

return new ResponseEntity<>(response, HttpStatus.CREATED);

} catch (IllegalArgumentException ex) {

return new ResponseEntity<>(Map.of("error", ex.getMessage()),

HttpStatus.BAD\_REQUEST);

}

}

@GetMapping

public ResponseEntity<List<Map<String, Object>>> getAllGroups() {

List<Group> groups = groupService.getAllGroups();

List<Map<String, Object>> groupList = new ArrayList<>();

for (Group g : groups) {

Map<String, Object> groupMap = new LinkedHashMap<>();

groupMap.put("groupId", g.getGroupId());

groupMap.put("groupName", g.getGroupName());

List<String> memberNames = g.getMembers().stream()

.map(Member::getName)

.toList();

groupMap.put("members", memberNames);

groupList.add(groupMap);

}

return ResponseEntity.ok(groupList);

}

@PostMapping("/add-member")

public ResponseEntity<?> addMember(@RequestBody Map<String, String> payload) {

try {

String groupName = payload.get("groupName");

String memberName = payload.get("memberName");

Group updatedGroup = groupService.addMemberToGroup(groupName, memberName);

return ResponseEntity.status(HttpStatus.CREATED).body(updatedGroup);

} catch (IllegalArgumentException e) {

return ResponseEntity.status(HttpStatus.BAD\_REQUEST)

.body(Map.of("error", e.getMessage()));

} catch (NoSuchElementException e) {

return ResponseEntity.status(HttpStatus.NOT\_FOUND)

.body(Map.of("error", e.getMessage()));

}

}

@GetMapping("/{groupId}/balances")

public Map<String, Object> calculateBalances(@PathVariable Long groupId) {

List<Map<String, Object>> balances = groupService.calculateGroupBalances(groupId);

return Map.of("balances", balances);

}

}

**ExpenseController.java**

package com.examly.springapp.controller;

import com.examly.springapp.model.Expense;

import com.examly.springapp.service.ExpenseService;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.\*;

import org.springframework.web.bind.annotation.\*;

import java.time.LocalDate;

import java.util.\*;

@RestController

@RequestMapping("/groups/{groupId}/expenses")

public class ExpenseController {

@Autowired

private ExpenseService expenseService;

@PostMapping

public ResponseEntity<?> addExpense(@PathVariable Long groupId, @RequestBody Map<String, Object> payload) {

try {

String description = (String) payload.get("description");

double amount = Double.parseDouble(payload.get("amount").toString());

String payer = (String) payload.get("payer");

LocalDate date = LocalDate.parse(payload.get("date").toString());

Expense expense = expenseService.addExpense(groupId, description, amount, payer, date);

return ResponseEntity.status(HttpStatus.CREATED).body(expense);

} catch (IllegalArgumentException e) {

return ResponseEntity.status(HttpStatus.BAD\_REQUEST)

.body(Map.of("error", e.getMessage()));

} catch (NoSuchElementException e) {

return ResponseEntity.status(HttpStatus.NOT\_FOUND)

.body(Map.of("error", e.getMessage()));

}

}

@GetMapping("/balances")

public ResponseEntity<Map<String, Object>> getGroupBalances(@PathVariable Long groupId) {

List<Map<String, Object>> balances = expenseService.calculateBalances(groupId);

Map<String, Object> response = new HashMap<>();

response.put("balances", balances);

return ResponseEntity.ok(response);

}

@GetMapping

public List<Expense> getExpenses(@PathVariable Long groupId) {

return expenseService.getExpensesByGroup(groupId);

}

}

**Expense.java**

package com.examly.springapp.model;

import jakarta.persistence.\*;

import java.time.LocalDate;

import com.fasterxml.jackson.annotation.JsonBackReference;

@Entity

public class Expense {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long expenseId;

private String description;

private double amount;

private String payer;

private LocalDate date;

@ManyToOne

@JoinColumn(name = "group\_id")

@JsonBackReference

private Group group;

public Long getExpenseId() {

return expenseId;

}

public void setExpenseId(Long expenseId) {

this.expenseId = expenseId;

}

public String getDescription() {

return description;

}

public void setDescription(String description) {

this.description = description;

}

public double getAmount() {

return amount;

}

public void setAmount(double amount) {

this.amount = amount;

}

public String getPayer() {

return payer;

}

public void setPayer(String payer) {

this.payer = payer;

}

public LocalDate getDate() {

return date;

}

public void setDate(LocalDate date) {

this.date = date;

}

public Group getGroup() {

return group;

}

public void setGroup(Group group) {

this.group = group;

}

}

**Group.java**

package com.examly.springapp.model;

import jakarta.persistence.\*;

import java.util.\*;

import com.fasterxml.jackson.annotation.JsonManaedReference;

@Entity

@Table(name = "groups")

public class Group {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long groupId;

@Column(unique = true, nullable = false)

private String groupName;

@OneToMany(mappedBy = "group", cascade = CascadeType.ALL, orphanRemoval = true, fetch = FetchType.EAGER)

@JsonManagedReference

private List<Member> members = new ArrayList<>();

@OneToMany(mappedBy = "group", cascade = CascadeType.ALL, orphanRemoval = true, fetch = FetchType.EAGER)

@JsonManagedReference

private List<Expense> expenses = new ArrayList<>();

public Long getGroupId() {

return groupId;

}

public void setGroupId(Long groupId) {

this.groupId = groupId;

}

public String getGroupName() {

return groupName;

}

public void setGroupName(String groupName) {

this.groupName = groupName;

}

public List<Member> getMembers() {

return members;

}

public void setMembers(List<Member> members) {

this.members = members;

}

public List<Expense> getExpenses() {

return expenses;

}

public void setExpenses(List<Expense> expenses) {

this.expenses = expenses;

}

}

**ExpenseRepository.java**

package com.examly.springapp.repository;

import com.examly.springapp.model.Expense;

import org.springframework.data.jpa.repository.JpaRepository;

import java.util.List;

public interface ExpenseRepository extends JpaRepository<Expense, Long> {

List<Expense> findByGroup\_GroupId(Long groupId);

}

**GroupRepository.java**

package com.examly.springapp.repository;

import com.examly.springapp.model.Group;

import org.springframework.data.jpa.repository.JpaRepository;

import java.util.Optional;

public interface GroupRepository extends JpaRepository<Group, Long> {

Optional<Group> findByGroupName(String groupName);

}

**ExpenseService.java**

package com.examly.springapp.service;

import com.examly.springapp.model.\*;

import com.examly.springapp.repository.\*;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import java.time.LocalDate;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

import java.util.NoSuchElementException;

@Service

public class ExpenseService {

@Autowired

private ExpenseRepository expenseRepository;

@Autowired

private GroupRepository groupRepository;

@Autowired

private MemberRepository memberRepository;

public Expense addExpense(Long groupId, String description, double amount, String payerName, LocalDate date) {

if (amount <= 0 || !String.format("%.2f", amount).matches("\\d+\\.\\d{1,2}")) {

throw new IllegalArgumentException("Amount must be a positive number up to two decimal places");

}

Group group = groupRepository.findById(groupId)

.orElseThrow(() -> new NoSuchElementException("Group not found"));

Member payer = group.getMembers().stream()

.filter(m -> m.getName().equalsIgnoreCase(payerName))

.findFirst()

.orElseThrow(() -> new NoSuchElementException("Payer not found in group"));

Expense expense = new Expense();

expense.setDescription(description);

expense.setAmount(amount);

expense.setPayer(payerName);

expense.setGroup(group);

expense.setDate(date);

return expenseRepository.save(expense);

}

public List<Map<String, Object>> calculateBalances(Long groupId) {

Group group = groupRepository.findById(groupId)

.orElseThrow(() -> new RuntimeException("Group not found"));

List<Member> members = group.getMembers();

List<Expense> expenses = expenseRepository.findByGroup\_GroupId(groupId);

if (members.isEmpty()) {

throw new RuntimeException("No members in the group");

}

Map<String, Double> paidMap = new HashMap<>();

for (Member m : members) {

paidMap.put(m.getName(), 0.0);

}

for (Expense expense : expenses) {

String payerName = expense.getPayer(); // already a String

if (!paidMap.containsKey(payerName)) {

throw new RuntimeException("Payer not found in group: " + payerName);

}

paidMap.put(payerName, paidMap.get(payerName) + expense.getAmount());

}

double totalExpense = expenses.stream()

.mapToDouble(Expense::getAmount)

.sum();

double share = totalExpense / members.size();

List<Map<String, Object>> balances = new ArrayList<>();

for (Member m : members) {

Map<String, Object> balanceEntry = new HashMap<>();

double balance = paidMap.get(m.getName()) - share;

balanceEntry.put("member", m.getName());

balanceEntry.put("balance", Math.round(balance \* 100.0) / 100.0);

balances.add(balanceEntry);

}

return balances;

}

public List<Expense> getExpensesByGroup(Long groupId) {

return expenseRepository.findByGroup\_GroupId(groupId);

}

**GroupService.java**

package com.examly.springapp.service;

import com.examly.springapp.model.\*;

import com.examly.springapp.repository.\*;

import jakarta.transaction.Transactional;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import java.time.LocalDate;

import java.util.\*;

@Service

public class GroupService {

@Autowired

private GroupRepository groupRepository;

@Autowired

private MemberRepository memberRepository;

@Autowired

private ExpenseRepository expenseRepository;

public Group createGroup(String groupName, List<String> members) {

if (groupRepository.findByGroupName(groupName).isPresent()) {

throw new IllegalArgumentException("Group name already exists"); // updated message

}

Group group = new Group();

group.setGroupName(groupName);

group = groupRepository.save(group);

List<Member> memberEntities = new ArrayList<>();

for (String memberName : members) {

Member member = new Member();

member.setName(memberName);

member.setGroup(group);

memberEntities.add(member);

}

memberRepository.saveAll(memberEntities);

group.setMembers(memberEntities);

return group;

}

public Group addMemberToGroup(String groupName, String memberName) {

Group group = groupRepository.findByGroupName(groupName)

.orElseThrow(() -> new NoSuchElementException("Group not found"));

boolean exists = group.getMembers().stream()

.anyMatch(m -> m.getName().equalsIgnoreCase(memberName));

if (exists) {

throw new IllegalArgumentException("Member already exists in the group");

}

Member newMember = new Member();

newMember.setName(memberName);

newMember.setGroup(group);

group.getMembers().add(newMember);

return groupRepository.save(group);

}

public Optional<Group> getGroupByName(String groupName) {

return groupRepository.findByGroupName(groupName);

}

public Group saveGroup(Group group) {

return groupRepository.save(group);

}

public Optional<Group> getGroupById(Long id) {

return groupRepository.findById(id);

}

public List<Map<String, Object>> calculateGroupBalances(Long groupId) {

Group group = groupRepository.findById(groupId)

.orElseThrow(() -> new IllegalArgumentException("Group not found"));

List<Member> members = group.getMembers();

List<Expense> expenses = group.getExpenses();

Map<String, Double> balanceMap = new HashMap<>();

for (Member member : members) {

balanceMap.put(member.getName(), 0.0);

}

for (Expense expense : expenses) {

double amountPerMember = expense.getAmount() / members.size();

for (Member member : members) {

balanceMap.put(member.getName(), balanceMap.get(member.getName()) - amountPerMember);

}

balanceMap.put(expense.getPayer(), balanceMap.get(expense.getPayer()) + expense.getAmount());

}

List<Map<String, Object>> result = new ArrayList<>();

for (Map.Entry<String, Double> entry : balanceMap.entrySet()) {

Map<String, Object> memberBalance = new HashMap<>();

memberBalance.put("member", entry.getKey());

memberBalance.put("balance", Math.round(entry.getValue() \* 100.0) / 100.0);

result.add(memberBalance);

}

return result;

}

public List<Group> getAllGroups() {

return groupRepository.findAll();

}

**CHAPTER 6**

**CONCLUSION**

This chapter presents the conclusions drawn from the project and the learning outcomes achieved through its development.

**6.1 CONCLUSION**

The Expense Splitter Application provides an efficient, reliable, and user-friendly solution for managing shared expenses within groups of friends, families, or organizations. By digitizing the traditional manual process of tracking and splitting expenses, the system minimizes errors, avoids disputes, and ensures fairness in cost distribution. It enables users to create groups, add members, log expenses, and automatically calculate balances, while also providing transparency in how much each member owes or is owed.

The system enhances collaboration among group members, improves financial management, and ensures that shared expenses are settled smoothly. With the integration of modern technologies like React JS, Spring Boot, REST APIs, and MySQL, the application is not only secure and scalable but also adaptable to different use cases such as travel groups, household expenses, or team projects. Ultimately, it reduces the manual effort required for expense tracking, increases accuracy, and contributes to better user satisfaction and financial coordination.

**6.2 FUTURE SCOPE**

The Expense Splitter Application has significant potential for future enhancement and expansion. In future developments, advanced features such as mobile application support, real-time notifications, and multi-currency handling can be integrated to make the system more intelligent and adaptive. Integration with payment gateways can enable seamless settlement of balances directly through online payments.

Additionally, incorporating data analytics and visualization dashboards can help users analyse spending patterns, identify frequent contributors, and make better financial decisions. Features such as AI-powered settlement recommendations, voice-based expense logging, and cloud deployment can further enhance accessibility, intelligence, and scalability, allowing the system to serve a larger user base across multiple locations.

With these improvements, the Expense Splitter Application can evolve into a comprehensive platform, capable of handling not only small friend or family groups but also large-scale financial collaborations. Thus, it can become a complete end-to-end solution for simplifying and managing shared expenses effectively.

**CHAPTER 7**

**REFERENCES**

1. Leonard Richardson & Mike Amundsen, RESTful Web APIs, O’Reilly Media, 2013.
2. Abraham Silberschatz, Henry F. Korth, and S. Sudarshan, Database System Concepts, McGraw-Hill, 7th Edition, 2020.
3. Official Spring Boot Documentation – [https://spring.io/projects/spring-boot](https://spring.io/projects/spring-boot" \t "_new)
4. React Official Documentation – https://reactjs.org/
5. MySQL Official Documentation – https://dev.mysql.com/doc/