DAYANANDA SAGAR COLLEGE OF ENGINEERING

(An Autonomous Institute, Affiliated to VTU, Approved by AICTE & UGC)

Shavige Malleshwara Hills, Kumaraswamy Layout, Bangalore 560 078

MINI PROJECT REPORT

ON

"SplitPay – Web Application for Expense Management and Payment"

(Course: Mini Project-2**, Code:** 22CB49)

Academic year 2023-24, Semester: 04

by

Prerana V	**USN: 1DS22CB038**
Priyanshu K S	**USN: 1DS22CB039**
Simha Harshith	**USN: 1DS23CB048**
Sneha R	**USN: 1DS22CB050**

Under the Guidance of

Dr. Dattatreya P Mankame
Professor and Head, Department of CSBS

Department of Computer Science and Business Systems

DAYANANDA SAGAR COLLEGE OF ENGINEERING,

BANGALORE

DAYANANDA SAGAR COLLEGE OF ENGINEERING

(An Autonomous Institute, Affiliated to VTU, Approved by AICTE & UGC)

Shavige Malleshwara Hills, Kumaraswamy Layout, Bangalore 560 078

Department of Computer Science and Business Systems

CERTIFICATE

Certified that the mini project work titled "**SplitPay- Web Application for Expense Management and Payment"** is carried out by **Ms. Prerana V, USN: 1DS22CB038**, **Mr. Priyanshu K S, USN: 1DS22CB039, Mr. Simha Harshith, USN: 1DS23CB048 and Ms. Sneha R, USN: 1DS22CB050** are bonafide students of Dayananda Sagar College of Engineering, Bangalore in partial fulfillment for requirement for IV semester **Mini Project-1** during the year 2023-2024. It is certified thatall the corrections/ suggestions indicated for Internal Assessment have beenincorporated in the report deposited in the departmental library. The report has been approved as it satisfies the academic requirements in respect of mini project work prescribed for the said course.

Guide and Head of Department

(Dr. Dattatreya P Mankame)

Semester End Examination

Name of the Examiners:	**Signature with Date**
1.	
2.	

Acknowledgement

We express our deepest gratitude to our guide **Dr. Dattatreya P. Mankame**, Head of Department, Computer Science and Business Systems, for his valuable guidance and encouragement while doing this mini project work. We also thank **Dr. B G Prasad,** Principal, for their support and facilities at various stages of work.

We also extend our thanks to the management of Dayananda Sagar College of Engineering, Bengaluru, for providing an excellent study environment, reference materials and laboratory facilities. We remain grateful to the co-operation and help rendered by the teaching and non-teaching staff of the Department.

Prerana V
Priyanshu K S
Simha Harshith
Sneha R

Abstract

SplitPay is a web-based application that simplifies expense management and payment processes for individuals and groups. The platform allows users to track, split, and settle expenses effortlessly, eliminating manual calculations and reminders. Key features include expense tracking, bill splitting, payment processing, and automated reminders, all within a secure and intuitive interface. It aims to reduce financial stress and promote a cashless economy.

Ideal for individuals, friends, and families, SplitPay offers a user-friendly dashboard where users can create groups, add members, and categorize expenses easily, making it an essential tool for modern personal finance management.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

SplitPay is a web-based application designed to facilitate effortless expense management and payment processes among individuals and groups. This user-friendly platform enables users to track, split, and settle expenses with ease, eliminating the need for manual calculations and reminders.

Key features include:
- Expense tracking and categorization
- Bill splitting and sharing
- Automated reminders and notifications
- Secure and intuitive interface

SplitPay aims to provide a comprehensive solution for managing shared expenses, reducing financial stress, and promoting a cashless economy. Its robust functionality and user-centric design make it an ideal choice for individuals, friends, and families seeking to streamline expense management.

SplitPay's intuitive dashboard provides a clear overview of expenses, pending payments, and settled transactions. Users can create groups, add members, and assign expenses to specific categories, making it easy to track and manage shared costs. The application also offers features like receipt uploading, expense categorization, and payment confirmation, ensuring a seamless and transparent experience. With SplitPay, users can effortlessly manage expenses, reduce financial disputes, and enjoy a hassle-free payment process, making it an indispensable tool for modern personal finance management.

Chapter 1

Literature Review

In paper [1], the authors Swaraj Mahindre, Stuti Gupta, Vaishnavi Rambhad, Shreyasi Kopisetti and Shruti Thakur (Computer Science and Engineering, G H Raisoni College of Engineering, Nagpur, India) has proposed a system for dividing group expenses and managing expenses that employs blockchain technology for data security. In this project, they integrated Blockchain technology with a web client so that users can easily access the application. This application will save all of the user's transaction data in a secure manner, ensuring that the data is not tampered by anyone. Also, options for dividing the bill equally and unequally are also available. This application will also keep a track of user's expenses, this feature will help them to manage their expenses accordingly.

In paper [2], the authors Muchsin Hisyam and Ida Bagus Kerthyayana Manuaba (Computer Science Department, Faculty of Computing and Media, Bina Nusantara University, Jakarta, Indonesia) has proposed the possibilities of integrating two different Payments Gateways which are Midtrans and Xendit and implementing them by using core API integration method for sample web app application prototype. For the result of the study, the architecture of integration model between these two payment gateways is successfully developed and tested, especially for splitting payment scenario between two e-money payment models from different payment gateways. Both payment gateways are integrated into a backend of marketplace scenarios where in this split payment scenarios might commonly occur.

Chapter 3

Problem Formulation

3.1 Problem Statement

The project aims to develop a web-based application that simplifies expense management and payment processes among individuals and groups. The focus lies on creating an intuitive platform that enables users to track, split, and settle expenses seamlessly, reducing the need for manual calculations and minimizing financial disputes. By leveraging modern web technologies and secure backend systems, this project seeks to provide a comprehensive solution for managing shared expenses, enhancing financial transparency, and promoting a cashless economy.

3.2 Objectives

1. To develop a module for efficient bill splitting and sharing among group members.
2. To create a secure system for user authentication and data protection.
3. To implement a dashboard providing clear graphical insights and analysis of group expenses.

Chapter 4

Requirement Specification

The requirement analysis for SplitPay involves identifying the essential features and functionalities needed to create an efficient and user-friendly expense management application. This process includes gathering and analyzing user needs, system specifications, and technological requirements to ensure the successful development and deployment of the platform.

Table 1.1 Hardware Requirements

Sl.No	Hardware / Equipment	Specification
1.	Processor	Intel i5 Core Processor
2.	Clock speed	2.20GHz
3.	Monitor	1024*768 Resolution, Color
4.	RAM	8GB
5.	Network	Broadband Internet connection

Table 1.2 Software Requirements

Sl.No	Software	Specification
1.	Operating System	Windows 11
2.	Web Browser	Chrome, Firefox
3.	NodeJS, npm	Latest Version
4.	MySQL, Prisma	Latest Version
5.	ReactJS, Redux	Latest Version
6.	Lodash, bcryptJS	Latest Version
7.	Deployment Platform	GitHub, Vercel
8.	JWT, Git, Github	Latest Version

System Architecture

The system architecture of SplitPay is designed to ensure efficient expense management, seamless user experience, and secure data handling. It follows a multi-tier architecture, comprising the presentation layer, business logic layer, and data layer, with deployment on cloud infrastructure for scalability and reliability.

5.1 Block Diagram

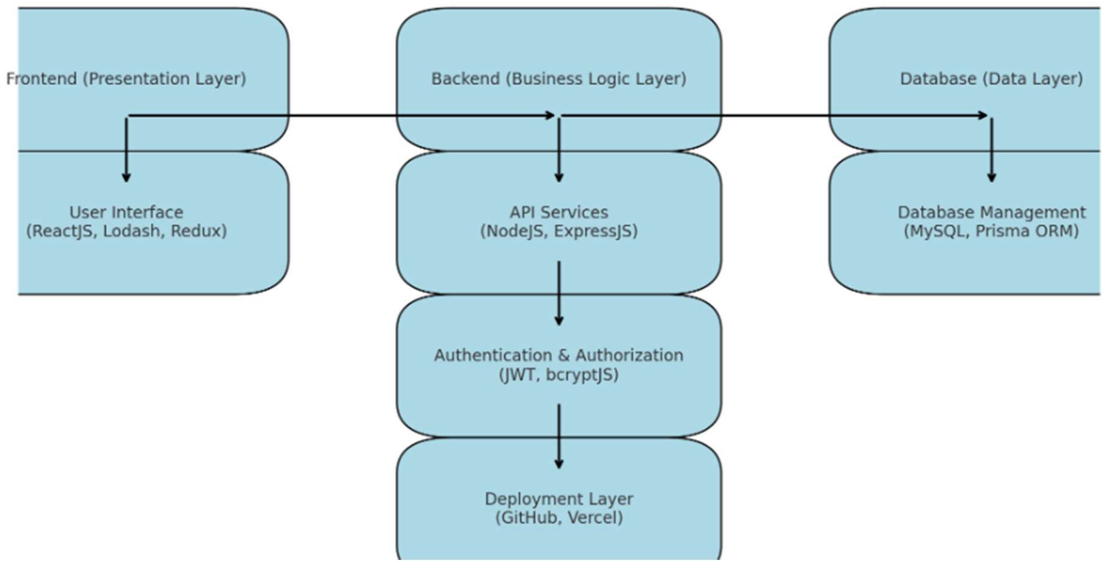

Frontend (Presentation Layer)
Backend (Business Logic Layer)
Database (Data Layer)
User Interface (ReactJS, Lodash, Redux)
API Services (NodeJS, ExpressJS)
Database Management (MySQL, Prisma ORM)
Authentication & Authorization (JWT, bcryptJS)
Deployment Layer (GitHub, Vercel)

Fig. 5.1: Block Diagram

1. Presentation Layer (Frontend)

- **Technologies Used:** HTML, CSS, ReactJS, Lodash, Redux
- **Description:**
  - **User Interface:** The frontend provides a user-friendly interface for users to interact with the application. It includes components for expense tracking, bill splitting, receipt uploading, and viewing notifications.
  - **State Management:** Redux is used for managing the application state, ensuring consistency and efficient data flow across components.
  - **Data Manipulation:** Lodash is utilized for efficient data handling and manipulation tasks.

2. Business Logic Layer (Backend)

- **Technologies Used:** NodeJS, ExpressJS, JWT, bcryptJS
- **Description:**
    - **API Services:** The backend comprises RESTful API services developed using NodeJS and ExpressJS. These services handle requests from the frontend, perform necessary operations, and send responses back.
    - **Authentication & Authorization:** JWT (JSON Web Tokens) is used for secure user authentication and session management. bcryptJS is employed for hashing and securely storing user passwords.
    - **Expense Management Logic:** The backend contains the core logic for tracking, categorizing, and splitting expenses, as well as managing notifications and reminders.

3. Data Layer

- **Technologies Used:** MySQL, Prisma ORM
- **Description:**
    - **Database Management:** MySQL is used as the relational database for storing user data, expense details, and other related information.
    - **ORM:** Prisma ORM is used for database interaction, providing an abstraction layer that simplifies data querying and manipulation.

4. Deployment Layer

- **Technologies Used:** GitHub, Vercel
- **Description:**
    - **Version Control:** GitHub is used for version control, code management, and collaboration.
    - **Continuous Deployment:** Vercel is used for continuous deployment of the application, ensuring that the latest code changes are automatically deployed to the live environment.

5.2 Flowchart

```
+--------------------------+
                 |     User Interaction     |
                 +--------------------------+
                           |
                           v
                 +--------------------------+
                 |  User Action (Add Expense)|
                 +--------------------------+
                           |
                           v
                 +--------------------------+
                 | Frontend (ReactJS, Lodash,|
                 |           Redux)          |
                 +--------------------------+
                           |
                           v
                 +--------------------------+
                 |  Send API Request to      |
                 |  Backend (NodeJS, Express)|
                 +--------------------------+
                           |
                           v
                 +--------------------------+
                 | Process Request:          |
                 | - Validate Input          |
                 | - Authenticate User       |
                 | - Execute Business Logic  |
                 +--------------------------+
                           |
                           v
                 +--------------------------+
                 |  Database Operations      |
                 |  (MySQL, Prisma ORM)      |
                 +--------------------------+
                           |
                           v
                 +--------------------------+
                 | Send Response to         |
                 | Frontend                 |
                 +--------------------------+
                           |
                           v
                 +--------------------------+
                 | Update User Interface    |
                 +--------------------------+
                           |
                           v
                 +--------------------------+
                 | Display Confirmation     |
                 | to User                  |
```

User Interaction: Users interact with the application through the user interface.

User Action (Add Expense): Users perform actions such as adding an expense.

Frontend (ReactJS, Lodash, Redux): The frontend handles the user action and sends an API request to the backend.

Send API Request to Backend (NodeJS, Express): The frontend sends the user request to the backend for processing.

Process Request:

Validate input data.

Authenticate the user using JWT.

Execute the business logic to handle the request.

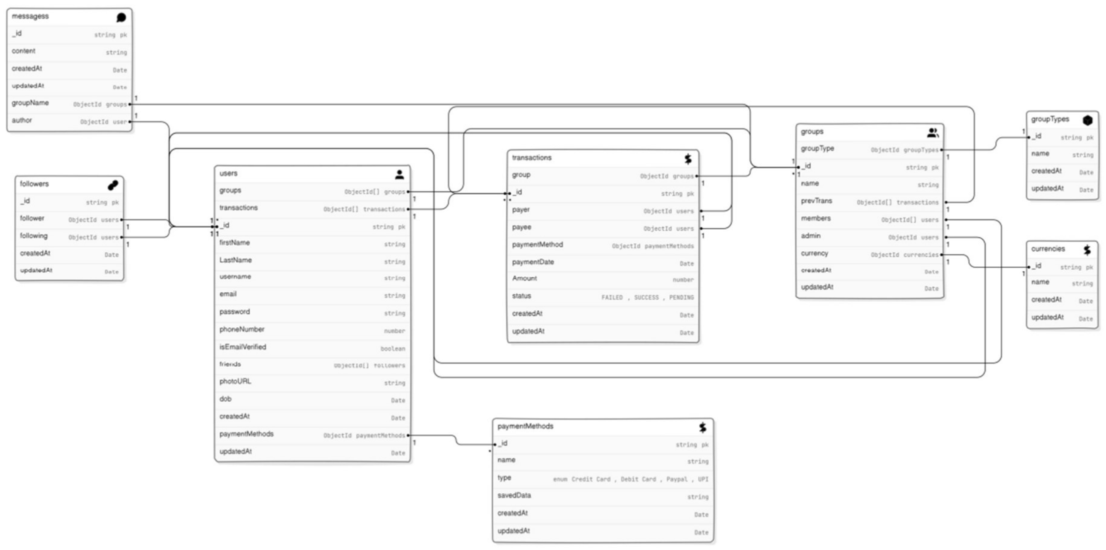
Database Operations (MySQL, Prisma ORM): The backend interacts with the database to perform necessary operations like storing the expense data.

Send Response to Frontend: The backend sends a response back to the frontend with the result of the operation.

Update User Interface: The frontend updates the user interface based on the backend response.

Display Confirmation to User: The application displays a confirmation message to the user, indicating that the expense has been added successfully.

5.3 schema diagram



Chapter 6

6.1

Implementing the SplitPay project involves developing both the frontend and backend, setting up the database, and deploying the application. Below are the detailed steps to implement this project:

### 1. Frontend Implementation

#### Setup
- **Initialize Project:** Create a React project using `create-react-app`.
- **Install Dependencies:** Install necessary packages like `react-redux`, `redux`, `react-router-dom`, `axios`, `lodash`, and `formik`.

#### Project Structure
```
/src
  /components
    /Auth
    /Dashboard
    /Expense
    /Group
    /User
  /redux
    /actions
    /reducers
    /store.js
  /utils
  /services
  App.js
  index.js
```

~~~
# Chapter 6

#### Key Components

- **Authentication Components:** `Login.js`, `Register.js`

- **Dashboard Components:** `Dashboard.js`, `ExpenseSummary.js`

- **Expense Components:** `AddExpense.js`, `ExpenseList.js`

- **Group Components:** `GroupList.js`, `GroupDetails.js`

- **User Components:** `Profile.js`, `Settings.js`

#### Redux Setup

- **Store Configuration:** Configure the Redux store in `store.js`.

- **Actions and Reducers:** Create actions and reducers for managing user authentication, groups, expenses, and notifications.

#### API Integration

- **Axios Setup:** Create an `axios` instance with base URL and interceptors for handling JWT.

- **Service Layer:** Create a service layer in `/services` for making API calls.

#### Example Code Snippets

**store.js**

```javascript
import { createStore, applyMiddleware } from 'redux';
import thunk from 'redux-thunk';
import rootReducer from './reducers';

const store = createStore(rootReducer, applyMiddleware(thunk));

export default store;
~~~

axios.js

```javascript
import axios from 'axios';
```

Chapter 6

```
const instance = axios.create({
  baseURL: 'http://localhost:5000/api'
});

instance.interceptors.request.use(config => {
  const token = localStorage.getItem('token');
  if (token) {
    config.headers.Authorization = `Bearer ${token}`;
  }
  return config;
});

export default instance;
```

### 2. Backend Implementation

#### Setup
- **Initialize Project:** Initialize a Node.js project using `npm init`.
- **Install Dependencies:** Install necessary packages like `express`, `mongoose`, `jsonwebtoken`, `bcryptjs`, `cors`, and `dotenv`.

#### Project Structure
```
/src
  /controllers
  /models
  /routes
  /middleware
  /config
  server.js
```

#### Key Components

Chapter 6

- **Authentication:** JWT-based authentication with routes for login, registration, and token verification.
- **Expense Management:** CRUD operations for managing expenses.
- **Group Management:** CRUD operations for managing groups and members.
- **Notifications:** Handling notifications for expense updates and reminders.

#### Database Setup

- **MongoDB with Mongoose:** Define schemas and models for users, groups, transactions, etc.

#### Example Code Snippets

server.js

```javascript
const express = require('express');
const mongoose = require('mongoose');
const cors = require('cors');
const dotenv = require('dotenv');

dotenv.config();

const app = express();
app.use(express.json());
app.use(cors());

mongoose.connect(process.env.MONGO_URI, { useNewUrlParser: true,
useUnifiedTopology: true })
  .then(() => console.log('MongoDB connected'))
  .catch(err => console.error(err));

// Routes
app.use('/api/auth', require('./routes/auth'));
app.use('/api/expenses', require('./routes/expenses'));
app.use('/api/groups', require('./routes/groups'));
```

```
const PORT = process.env.PORT || 5000;
app.listen(PORT, () => console.log(`Server running on port ${PORT}`));
```

models/User.js

```javascript
const mongoose = require('mongoose');
const bcrypt = require('bcryptjs');

const UserSchema = new mongoose.Schema({
  firstName: { type: String, required: true },
  lastName: { type: String, required: true },
  email: { type: String, required: true, unique: true },
  password: { type: String, required: true },
  createdAt: { type: Date, default: Date.now },
  updatedAt: { type: Date, default: Date.now }
});

UserSchema.pre('save', async function(next) {
  if (!this.isModified('password')) return next();
  const salt = await bcrypt.genSalt(10);
  this.password = await bcrypt.hash(this.password, salt);
  next();
});

module.exports = mongoose.model('User', UserSchema);
```

### 3. Database Implementation

#### Setup MySQL with Prisma ORM

- **Initialize Prisma:** Run `npx prisma init` to initialize Prisma.
- **Define Schema:** Define the schema in `prisma/schema.prisma`.

- **Run Migrations:** Run `npx prisma migrate dev` to create the database tables.

schema.prisma

```prisma
datasource db {
  provider = "mysql"
  url      = env("DATABASE_URL")
}

generator client {
  provider = "prisma-client-js"
}

model User {
  id            Int       @id @default(autoincrement())
  firstName     String
  lastName      String
  email         String    @unique
  password      String
  createdAt     DateTime  @default(now())
  updatedAt     DateTime  @updatedAt
  groups        Group[]
  transactions  Transaction[]
}

model Group {
  id            Int          @id @default(autoincrement())
  name          String
  members       User[]
  transactions  Transaction[]
  createdAt     DateTime     @default(now())
  updatedAt     DateTime     @updatedAt
}
```

Chapter 6

```
model Transaction {
  id          Int      @id @default(autoincrement())
  payerId     Int
  payeeId     Int
  amount      Float
  createdAt   DateTime  @default(now())
  updatedAt   DateTime  @updatedAt

  payer       User     @relation(fields: [payerId], references: [id])
  payee       User     @relation(fields: [payeeId], references: [id])
}
```

### 4. Deployment

#### Vercel Deployment
- **Connect Repository:** Connect your GitHub repository to Vercel.
- **Configure Environment Variables:** Add necessary environment variables in Vercel settings.
- **Deploy:** Deploy the frontend and backend using Vercel's CI/CD pipeline.

#### GitHub Actions
- **CI/CD Pipeline:** Set up GitHub Actions for continuous integration and deployment.
- **Testing:** Write and run tests for both frontend and backend before deploying.

### 5. Security Considerations

- **Data Protection:** Use HTTPS for secure data transmission.
- **Authentication & Authorization:** Implement role-based access control.
- **Input Validation:** Validate user inputs to prevent SQL injection and other attacks.
- **Environment Variables:** Store sensitive information like API keys and database credentials in environment variables.

By following these steps, you can successfully implement the SplitPay project, ensuring a robust and scalable solution for managing shared expenses.

Chapter 6

Chapter 6
IMPLEMENTATION

Chapter 7

Results & Snapshots

Chapter 8

Conclusion and Future work

Bibliography