Project Proposal: SplitEase

Date: 23 Mar 2025

1. Introduction

1.1 Project Overview

In social gatherings, shared expenses are inevitable, whether for group dinners, trips, or shared living arrangements. However, manually managing and tracking these expenses can be tedious, time-consuming, and prone to errors, leading to confusion and disputes among participants. Existing solutions often lack flexibility in handling different expense-sharing scenarios, such as unequal contributions or dynamic group compositions.

To address these challenges, our team is developing SplitEase, a modern web application designed to simplify bill splitting and expense tracking. With an intuitive user interface and smart automation, SplitEase ensures that payments are transparent, fair, and well-documented. Our platform will cater to a wide range of users, including travelers, roommates, and colleagues, providing them with a seamless experience for managing shared costs efficiently.

1.2 Motivation

During our recent group trip to the South Island of New Zealand, we frequently encountered situations where different members paid for various expenses such as accommodation, meals, and transportation. Sometimes, the bill needed to be split equally among all members, while other times it only involved specific individuals. Additionally, the expense-sharing ratios were often unequal—for instance, when dining out, some members ordered alcoholic drinks while others did not. Keeping track of these expenses manually proved to be a challenge, leading us to the idea of developing SplitEase as our project. This app will provide an efficient and transparent way to manage shared expenses, ensuring fair and hassle-free cost distribution among users.

2. Related Work

In the realm of bill-splitting applications, several platforms aim to simplify the process of managing shared expenses. Below is an analysis of three such platforms:

2.1 Existing Applications

Application	Strengths	Weaknesses
Spliito	Supports all foreign currencies, beneficial for international travelers	The created group is open to anyone with the URL, allowing unrestricted edits and lacking security
Billzer	No sign-up or downloads required, ensuring quick access	Bills can only be evenly split; users cannot choose custom proportions or fixed amounts. Additionally, it does not support multi-user collaboration.
ALL IN	Simple and intuitive interface that clearly displays bill amounts and who owes whom	Lacks multi-user collaboration and includes unnecessary design elements, such as requiring bill entries per person.

2.2 Improvement Areas

Based on our analysis, we identified several key improvement areas.

2.2.1 Security & Access Control

Spliito allows unrestricted editing by anyone with the group URL, which raises privacy and security concerns. Implementing authentication-based access control would prevent unauthorized changes.

2.2.2 Flexible Splitting Options

Billzer only supports equal splitting, lacking the ability to divide expenses by percentage or fixed amounts. Adding this feature would greatly enhance usability. Users can:

- Split expenses equally among all participants.
- Customize the split using percentages or fixed amounts.
- Apply different split rules based on the specific scenario.

2.2.3 Multi-User Collaboration

Both Billzer and ALL IN lack multi-user collaboration, making it difficult for groups to manage shared expenses collectively. Our solution will allow users with accounts to create groups and invite members. Users without an account can be assigned a virtual account within a group, but only registered users within the same bill group can add or edit bills.

2.2.4 Historical Data & Smart Expense Tracking

SplitEase will provide users with access to historical bill records and intelligent expense tracking. This includes:

- Expense Categorization – Bills are automatically categorized based on their nature.
- Personal Expense Overview – Users can view a summary of all their expenditures.
- Data Analytics & Visualization – A clear statistical representation of spending patterns, providing insights into expense trends.

2.2.5 Conclusion

By addressing these areas, a bill-splitting application can differentiate itself in the market, offering a more secure, feature-rich, and user-friendly solution for managing shared expenses.

3. Requirements

3.1 Must-haves:

a. Real-Time Collaborative Expense Tracking - Multiple users can record and manage finances in the same ledger with real-time synchronization.

b. Team Ledger Management - Supports categorizing expenses, recording amounts, and attaching images or notes for better clarity.

c. Smart Expense Splitting - Automatically categorizes and splits expenses based on:

- Equal sharing
- Custom percentage or fixed amount
- Flexible splitting for different scenarios

d. Personal Expense Overview - Users can track their financial details, including:

- Total spending and category breakdown
- Borrowing and lending records (debts & receivables)

e. User Authentication & Account Management - Secure login, registration, and personalized account management.

3.2 Should-Have:

a. Data Analytics & Visualization - Provides clear financial insights through:

- Total ledger expenses and spending trends
- Individual spending proportions and trends
- Final balance summary for team expense review

b. Currency Conversion - Multi-currency support with real-time exchange rate conversion.

c. Import & Export Bills - Supports different formats for easy data migration and sharing.

e. Multi-Language Support - Adapts to different language backgrounds.

f. Trip Ledger Summary - Generates a trip expense report with images and statistics.

g. Undo/Redo Expense Edits - Allows users to revert changes to maintain accuracy.

h. Smart Reminders & Notifications - Alerts for pending settlements or unpaid debts.

3.3 Could-haves:

a. Map-Based Expense Tracking - Logs spending locations for travel reviews.

b. Bank & Payment Integration - Connects to credit cards, PayPal, or local payment platforms for automatic expense tracking.

c. Customizable Themes - Allows users to personalize the UI.

d. Voice Input for Expenses - Converts spoken input into structured expense records.

e. Social Interaction - Enables comments and discussions under shared expenses

f. Basic Role-Based Permissions - Prevents accidental modifications:

- Admin: Manage categories, members, and edit expenses
- Member: Add expenses and view records

3.4 Nice-to-haves:

a. OCR Bill Recognition - Scans and extracts details from receipts automatically.

b. Smart Budget Recommendations - Suggests travel budgets based on past spending patterns.

c. Mobile App & Web App Support - Optimized for mobile devices.

d. Offline Mode - Enables expense tracking without an internet connection, syncing later.

e. AI-Powered Expense Recognition - Uses AI to auto-classify and process expenses.

4. Technologies

4.1 Frontend

4.1.1 React (Next.js)

Provides a fast and responsive user interface for managing expenses, transactions, and user accounts. Moreover, reusable components make the codebase modular and maintainable. UI elements like transaction lists, user dashboards, and expense charts can be built as independent components. The useState hooks also help manage UI states effectively. There is a large ecosystem of third-party libraries for charts, such as Chart.js, which is a better choice to show the data.

4.1.2 Tailwind CSS

A utility-first CSS framework that streamlines UI development with pre-designed classes. Enables quick styling adjustments without writing custom CSS, reducing development time. Ensures a responsive and modern UI with minimal effort. Furthermore, making it easy to toggle dark mode. Also, there are many official and third-party plugins for animations.

4.1.3 Socket.io

Enables real-time communication between users for a seamless collaborative experience. Updates shared ledgers in real-time when users add, edit, or delete transactions. Reduces manual page refreshes, ensuring a smoother user experience.

4.2 Backend

4.2.1 Express.js

A minimal and flexible Node.js framework for handling HTTP requests. Provides a RESTful API for managing user authentication, transactions, and shared ledgers. Middleware support for logging, authentication, and error handling. Express integrates well with MongoDB using Mongoose, making database operations simpler.

4.2.2 Mongoose

An Object Data Modeling library for MongoDB, providing a structured way to interact with the database. Simplifies database queries and data validation. Enables defining schemas for transactions, users, and shared ledgers.

4.2.3 jsonwebtoken (JWT)

Securely handle user authentication and authorization. Once a shared ledger is created, ensure that only authenticated users can access and modify the shared ledger. Enable session-based authentication without the need for server-side session storage.

4.3 Database- MongoDB

A NoSQL database optimized for document storage and high scalability. Stores user data, transaction records, and shared ledgers efficiently. Provides flexible schema support to accommodate various expense tracking scenarios.

4.4 Testing

4.4.1 Frontend Testing- Jest + React Testing Library

Ensures UI components render correctly and function as expected. Tests user interactions, such as adding or editing transactions in the same ledger. Verifies component states and responses to real-time updates.

4.4.2 Backend Testing- est + Supertest

Tests API endpoints for correctness and reliability. Validates user authentication logic, ensuring JWT tokens are generated and verified correctly. Confirms ledger and transaction operations work as expected, including expense splitting and calculations.

Manual Testing

4.4.3 Postman

Used for manually testing API endpoints before deployment. Validates response data and error handling. Helps debug and optimize backend logic by simulating various request scenarios.

5. Project Management

Our team will adopt an Agile development strategy, with the project divided into five 1-week sprints. We aim for incremental development, ensuring each sprint delivers tangible progress toward the final product.

5.1 Agile Strategy & Sprint Planning

We will conduct a Sprint Planning Meeting every Monday, with all team members attending in person.

During the meeting, we will:

- Define and prioritize user stories, each with an assigned story point (indicating effort required) and a priority level (determining implementation order).
- Assign story points based on complexity and priority.
- Track progress using a Kanban board.

Each sprint will contain 60 story points, with each team member taking approximately 10 points of work.

5.2 Daily Standup & Issue Resolution

A daily standup meeting (~15 minutes, online) will be held. Each team member will briefly update their progress yesterday and tasks for the day.

If any blockers arise, only the relevant team members will discuss them separately to avoid disrupting others.

5.3 Sprint Review & Retrospective

Every Friday, we will conduct a Sprint Demo to showcase completed work.

After the demo, we will hold a Sprint Retrospective:

- Team members will discuss what went well, what didn’t, and suggest improvements.
- Any agreed improvements will be implemented in the next sprint.

5.4 Version Control Strategy

We will use Git for version control, following a feature-branch workflow:

- Each user story will be developed in a separate branch (feature/story-xxx).
- Developers will submit a pull request (PR) to merge changes into main.
- Each PR requires approval from at least two team members before merging.
- Manual testing and peer code reviews will be conducted to ensure functionality and code quality.

6. Risks in Team Projects

Through our group discussion, we identified several potential risks that could impact both the progress and overall success of a group project. Our analysis focused on three key risk frameworks, Student Contribution, Team Self-Management, and Resources, as these areas are the most likely to present challenges throughout the project. To ensure smooth execution and minimize disruptions, it is crucial to implement well-structured risk mitigation strategies.

To systematically manage risks, we adopt a structured approach based on four key steps: Plan, Action, Monitor, and Escalate, ensuring that all strategies align with the Key Risk Mitigation Principles: Specific, Realistic, Verifiable, and Appropriate. Planning involves identifying potential risks early and developing proactive strategies, such as establishing clear communication protocols and defining roles and responsibilities. Action refers to implementing predefined response measures when risks arise, ensuring that team members can address issues promptly and effectively. Monitoring requires continuous assessment of project progress to detect early signs of risk, utilizing regular check-ins, progress tracking, and verification methods. If a risk escalates beyond the team's control, the Escalation process ensures that appropriate steps are taken, such as seeking external support from lecturer or tutors, to prevent the issue from compromising the project's success.

The specific risks and their corresponding mitigation strategies are summarized in the table below:

Student Contribution	**Potential Risk**	**Mitigation Strategy**
Engagement	Lack of participation in meetings or delayed tasks. Unclear project goals and task expectations. Poor communication and collaboration. Slow response to feedback.	**Plan:** Establish meeting schedules, roles, deadlines, and clear task expectations. Set up skill-sharing sessions and define coding standards. Give feedback, set priorities, and ensure flexibility for members with conflicting schedules or external pressures. **Action:** Follow up on absences, clarify tasks, have peer mentoring, and code reviews. Share time management tips, keep communicating for overwhelmed or sick members. Ensure members communicate about schedule conflicts early.
Expertise	Team members have varying levels of technical expertise. Team members are not familiar with GitHub in collaboration. The code quality is low, making it difficult to understand and maintain. Team members lack experience in testing and debugging. Lack of experience in team collaboration.	

Personality	Team members lack effective time management skills. Team members are unwilling to accept feedback or suggestions. Team members fail to effectively express their thoughts or needs.	**Monitor:** Track attendance, participation, and task completion. Assess code quality and communication. Regularly check team members' availability and well-being. **Escalate:** If issues persist, report to the lecturer or class representative. Seek additional support or deadline adjustments if necessary.
Availability	Team members have different or conflicting schedules. Some members have limited time due to family or work commitments. Unexpected situations (e.g., illness or family emergencies) cause members to be absent.	
Wellbeing	Team members feel overwhelmed due to project workload or external pressures. Team members being sick.	

Team Self-Management	**Potential Risk**	**Mitigation Strategy**
Communication	Missed deadlines due to ineffective communication. Team members don't inform others when they are falling behind on tasks. One team member dominates discussions during meetings. Remote work arrangements create communication challenges.	**Plan:** Set clear goals, deadlines, roles, and communication. **Action:** Have timely updates, address delays early, balance discussions, and prevent task overlaps. Clarify tasks and goals regularly. **Monitor:** Track progress, check task completion, and review code quality. Monitor communication and collaboration. **Escalate:** Escalate unresolved issues or conflicts to the lecturer or class representative.
Co-ordination	Issues arise when merging code from different members. Disagreements between team members lead to difficulty in decision-making. Team members inadvertently work on the same tasks, causing overlap.	
Process	The team decides to change or add features midway through the project. Struggles in meeting deadlines. Team members deviate from the project goals by adding extra features.	
Clarity	Team members have different understandings of the project goals. A team member misinterprets their assigned tasks. The team lacks clarity about expected features. Difficulty in understanding code written by other team members.	

Resources	Potential Risk	Mitigation Strategy
Hardware	Personal computer / laptop malfunction Limited access to necessary hardware for development and testing.	**Plan:** Ensure all members have access to necessary hardware, software, and reliable connectivity. Confirm cross-browser compatibility for the application. **Action:** Provide backup resources, set up software early, and test the application across devices and browsers. **Monitor:** Regularly check hardware functionality, software usage, and monitor application performance and connectivity. **Escalate:** If issues persist, escalate to the lecturer, IT services, or technical team for resolution.
Software	Required software is not available when needed, delaying progress. Software updates introduce compatibility issues with the existing codebase.	
Technical	The web application is not fully compatible with certain browsers or devices. Server downtime affects access to critical online services, such as Git repositories. Limited internet access or connectivity issues slow down collaboration and development.	

Final Notes

We believe this project is scoped appropriately to be completed within the available timeline and resources. It aligns well with our team's interests and technical strengths, and we look forward to building a useful, user-friendly web application.