HOSTED LINK ⟶ https://splitsync-umber.vercel.app/

VIDEO PITCH DRIVE LINK: https://drive.google.com/drive/folders/1rb2tlk_shQ66JCiw-q_HLCqFLOvJZg-u

GITHUB REPO : https://github.com/ZohaAnsari04/Splitsync

➢ Core Logic

The application follows a client-side state management approach using React's built-in hooks:

1. State Management:

- Uses `useState` for component-level state
- Uses `useContext` for global state (theme, sound settings)
- Persists data in localStorage for expenses, payment reminders, and groups
- Implements data deduplication for recent activity

2. Data Flow:

- User interactions trigger state updates
- State changes propagate through the component tree
- Components re-render with updated data
- Effects handle side effects like localStorage persistence

3. Key Features:

- Expense splitting with multiple methods (equal, percentage, custom)
- Payment reminders with status tracking
- Expense groups for organizing shared expenses
- Gamification with karma points and streak tracking
- Analytics and financial insights
- Digital receipts and enhanced UI features

➢ Design Tokens

The design system is built on a comprehensive set of design tokens defined in CSS custom properties:

➢ Feature Components

1. Dashboard - Main application interface with quick actions

2. AddExpenseModal - Modal for adding new expenses

3. Analytics - Spending insights with charts

4. Gamification - Karma points and achievements

5. FinancialInsights - Budget tracking and forecasting

6. EnhancedUI - Digital receipts and location features

7. PaymentReminders - Meme-based payment reminders

8. ExpenseGroups - Group expense management

9. ThemeToggle - Dark/light theme switcher

➢ **Specialized Components**

- DigitalReceipt - Detailed receipt viewer

- ParticipantSelector - Expense participant selection

- SplitOptions - Various expense splitting methods

- SplashScreen - Animated loading screen

- NavigationMenu - Feature category navigation

➢ **Responsive Design**

- Mobile-first approach

- Responsive grid layouts

- Adaptive component sizing

- Touch-friendly interactive elements

➢ **AI Features Implemented**

The SplitSync application includes several AI-powered features, though the actual AI processing appears to be simulated rather than connected to real AI services:

1. AI-Powered Categorization

- **Location**: AddExpenseModal.tsx
- **Implementation**: The application suggests categories for expenses using a predefined list
- **UI Component**: "AI Suggested Category" section with buttons for Food, Transport, Entertainment, and Shopping

2. Voice Command Processing

- **Location**: AddExpenseModal.tsx
- **Implementation**: Speech recognition that parses natural language commands

- **Supported Commands**: Users can say things like "Add ₹200 chai bill by Raj"

➢ **AI Prompt Processing and Refinements**

Voice Command Parsing Logic

The application uses regex pattern matching to extract information from voice commands:

1. **Amount Extraction**:
   - Matches various currency formats (₹, rs, rupees) followed by numbers
2. **Title Extraction**:
   - Extracts text between the amount and participant name
3. **Participant Extraction**:
   - Identifies participants mentioned after "by"

➢ **Refinements in Voice Processing**

1. **Language Support**:
   - Configured for US English (en-US) for better browser compatibility
   - Works best in Chrome, Edge, and Safari
2. **Error Handling**:
   - Comprehensive error handling for speech recognition failures
   - User-friendly error messages when processing fails
3. **Validation**:
   - Validates extracted amounts to ensure they're valid numbers
   - Ensures required fields (title, amount) are present
   - Checks for valid participant lists
4. **User Experience Improvements**:
   - Visual feedback during processing (spinner animation)
   - Success notifications with extracted information summary
   - Physical "Enter" button to confirm voice-processed expenses

➢ **Simulated AI Features**

The project README mentions "AI-Powered Categorization" as a core feature, but the implementation appears to be simulated with predefined categories rather than actual AI processing. The categories shown are:

- Food
- Transport
- Entertainment
- Shopping

Planned AI Integrations

There are plans for additional AI features:

1. **Bank Account Syncing** - Connect to bank accounts for automatic transaction import
2. **Email Parsing** - Parse expense details from email receipts
3. **Calendar Integration** - Sync recurring expenses with calendar

➢ **Code Quality and Refinements**

The AI-related code shows several refinements:

1. **Type Safety**: Full TypeScript typing for all AI-related functions
2. **State Management**: Proper React state management for voice recognition
3. **Memory Management**: Cleanup functions for speech recognition resources
4. **Accessibility**: Proper error handling and user feedback
5. **Performance**: Efficient regex patterns for command parsing

While the current implementation simulates AI functionality with pattern matching and predefined options, the architecture is designed to potentially integrate with real AI services in the future. The voice command processing is the most sophisticated AI feature currently implemented, with robust parsing logic and error handling.

Future Vision: How SplitSync Could Grow Into a Real App

1. Enhanced AI Capabilities

Real AI Integration

Currently, SplitSync simulates AI features with pattern matching. To become a real app, it would need to integrate with actual AI services:

- **Speech Recognition**: Connect to services like OpenAI Whisper or Google Speech-to-Text for more accurate voice processing
- **Natural Language Processing**: Use AI models to understand complex expense descriptions and automatically categorize them
- **Intelligent Suggestions**: Implement machine learning to suggest splitting methods based on historical patterns

Advanced Expense Categorization

The app could evolve to use AI for:

- Automatically detecting expense types from descriptions
- Learning user spending patterns to provide personalized categories
- Suggesting budget allocations based on historical data

2. Backend and Database Integration

User Authentication System

A real app would need:

- Secure user registration and login
- Password recovery mechanisms
- Social login options (Google, Facebook, Apple)
- Multi-factor authentication for security

Cloud Database Integration

Instead of localStorage, a production app would require:

- Cloud databases (like Supabase, Firebase, or MongoDB) to store user data
- Data synchronization across devices
- Backup and recovery systems
- GDPR-compliant data handling

3. Real-time Collaboration Features

Group Expense Management

To support collaborative features:

- Real-time updates when group members add expenses
- Notification systems for expense updates
- Permission management for group administrators
- Conflict resolution for disputed expenses

WebSocket Integration

For real-time functionality:

- Instant notifications when expenses are added
- Live updates to balance calculations
- Real-time chat features for groups

4. Advanced Analytics and Insights

Machine Learning-Powered Financial Insights

A mature app would offer:

- Spending pattern analysis using ML algorithms
- Predictive analytics for future expenses
- Personalized financial recommendations
- Trend identification across categories and time periods

Data Visualization

Enhanced analytics features:

- Interactive dashboards with customizable charts
- Export options for financial reports
- Comparison tools for different time periods
- Budget vs. actual spending analysis

5. Mobile App and Cross-Platform Support

Native Mobile Applications

To reach more users:

- Dedicated iOS and Android apps built with React Native or native technologies
- Mobile-specific features like camera receipt scanning
- Offline mode with sync capabilities
- Push notifications for payment reminders

Cross-Platform Consistency

Ensuring seamless experience across:

- Web application
- Mobile apps
- Desktop applications
- Browser extensions

6. Monetization and Business Model

Subscription Tiers

A sustainable business model would include:

- **Free Tier**: Basic features with usage limits
- **Pro Tier**: Advanced features like unlimited expenses, AI categorization, and analytics

- **Team Tier**: Group features for families or businesses

Additional Revenue Streams

- Premium themes and customization options
- API access for developers
- White-label solutions for businesses
- Partnership with financial institutions

7. Enterprise Features

API Integration Platform

To become a platform rather than just an app:

- REST API for third-party integrations
- Webhook support for real-time data exchange
- Developer portal with documentation
- SDKs for popular programming languages

Business Solutions

Enterprise-focused features:

- Team expense management
- Corporate card integration
- Approval workflows
- Reporting and compliance tools

8. Scalability and Performance Enhancements

Infrastructure Scaling

To handle growth:

- Cloud infrastructure that scales automatically
- Content delivery networks for global performance
- Database optimization for large datasets
- Caching strategies for improved response times

Performance Optimization

User experience improvements:

- Progressive web app capabilities
- Offline functionality with sync

- Performance monitoring and optimization
- Accessibility compliance for all users

9. Security and Compliance

Data Protection

Enterprise-grade security features:

- End-to-end encryption for sensitive data
- Regular security audits
- Compliance with financial regulations (PCI DSS, SOX)
- Privacy controls and data portability

Fraud Prevention

Protecting users from malicious activity:

- Anomaly detection for unusual spending patterns
- Two-factor authentication
- Transaction verification systems
- Suspicious activity alerts

10. Ecosystem Development

Third-Party Integrations

Connect with popular services:

- Bank account synchronization
- Payment platforms (PayPal)
- Accounting software (QuickBooks, Xero)
- Calendar integration for recurring expenses

Community Features

Building a user community:

- User forums for tips and support
- Feature request and voting systems
- User-generated expense templates
- Social sharing of financial achievements

This evolution would transform SplitSync from a prototype into a comprehensive financial management platform that could serve millions of users globally, with revenue streams, enterprise features, and advanced AI capabilities that provide real value beyond simple expense tracking.

Resources Used in the SplitSync Project

1. Development Technologies and Frameworks

Core Technologies

- **React 18**: Component-based UI library for building interactive interfaces
- **TypeScript**: Typed superset of JavaScript for enhanced code quality and developer experience
- **Vite**: Fast build tool and development server for modern web projects

UI Components and Libraries

- **shadcn/ui**: Reusable component library built on Radix UI with Tailwind CSS styling
- **Radix UI**: Unstyled, accessible UI primitives that provide the foundation for shadcn/ui
- **Tailwind CSS**: Utility-first CSS framework for rapid UI development and consistent styling
- **React Router**: Declarative routing for React applications to handle navigation

Icons and Visual Elements

- **Lucide React**: Beautiful and consistent icon set used throughout the application

2. State Management and Data Handling

React Built-in Solutions

- **React Hooks**: useState, useEffect, useContext for state management and side effects
- **React Context API**: For global state sharing (theme, sound settings)

Form Handling and Validation

- **React Hook Form**: Performant, flexible forms with easy validation
- **Zod**: TypeScript-first schema declaration and validation library

Data Visualization

- **Recharts**: Charting library built on D3 components for analytics and insights

Notifications

- **Sonner**: Minimal toast notification system for user feedback

Server State Management

- **TanStack Query**: Server state management for future API integration

3. Visual Effects

Animations and Styling

- **CSS Animations**: Custom animations for enhanced user experience
- **Glassmorphism Effects**: Frosted glass UI elements using CSS backdrop-filter
- **Neon Glow Effects**: Subtle glowing effects for interactive elements using CSS box-shadow

4. **Project Structure and Organization**

File Organization

- Component-based architecture with clear separation of concerns
- Dedicated directories for components, hooks, pages, and utilities
- TypeScript type definitions for improved code reliability

Configuration Files

- **tailwind.config.ts**: Tailwind CSS configuration with custom theme extensions
- **tsconfig.json**: TypeScript configuration files for app and node environments
- **vite.config.ts**: Vite build tool configuration
- **eslint.config.js**: ESLint configuration for code quality
- **postcss.config.js**: PostCSS configuration
- **components.json**: shadcn/ui component configuration

5. **Browser APIs and Features**

Web APIs

- **Speech Recognition API**: For voice input functionality
- **localStorage**: For client-side data persistence
- **CSS Custom Properties**: For theme management
- **CSS Animations and Transitions**: For micro-interactions

6. **Documentation and Assets**

Documentation

- **README.md**: Comprehensive project documentation
- **IMPLEMENTED_FEATURES.md**: Detailed feature implementation log

Assets

- Logo and hero background images
- CSS files for styling and custom utility classes