

MY FINANCE – Personal Finance Tracker

REAL TIME PROJECT REPORT

Bachelor of Technology (B. Tech – IIInd year/ IIsemester)

in

Computer Science and Engineering

by

S. Sri Sai

-

23AG1A05I3

Under the Esteemed Guidance of

Mr. Aga Parvej

(Assistant Professor)



Department of Computer Science and Engineering

ACE Engineering College

An AUTONOMOUS Institution

NBA Accredited B. Tech Courses, Accorded NAAC 'A' Grade

(Affiliated to Jawaharlal Nehru Technological University, Hyderabad, Telangana)

Ankushapur(V), Ghatkesar(M), Medchal - MalkajgiriDist - 501 301.

June 2025



ACE Engineering College

An AUTONOMOUS Institution

Website: www.aceec.ac.in E-mail: info@aceec.ac.in

Department of Computer Science and Engineering

Certificate

This is to certify that the Real Time project work entitled "MY FINANCE – Personal Finance Tracker" submitted by S. Sri Sai (23AG1A05I3) in partial fulfillment of Real Time Project work during the academic year 2024-25 is a record of bonafide work carried out by him under our guidance and supervision.

Mr. Aga Parvej
Assistant Professor
Project Guide

Mr. B R Srinivasa Rao
Assistant Professor &
Project Coordinator

Mr. V. Chandra Sekhar Reddy
Head of Section - C
Associate Professor

Dr. M. V. Vijaya Saradhi
Professor and Dean
Department of CSE

Acknowledgement

I would like to express my gratitude to all the people behind the screen who have helped me transform an idea into a real time application.

I would like to express my heart-felt gratitude to my parents without whom I would not have been privileged to achieve and fulfill my dreams.

A special thanks to General Secretary, **Prof. Y. V. Gopala Krishna Murthy**, for having founded such an esteemed institution.

My Sincere thanks to our CEO **Mr. Y.V.Raghu Vamshi**, for his support in doing project work.

I am very much thankful to our beloved Vice Principal & Dean-Skill Development **Dr. M. Murali** for his continuous encouragement to fulfill my project.

I am profoundly grateful to **Dr. M. V. Vijaya Saradhi**, Professor and Head, Department of Computer Science and Engineering, for being an excellent guide and also a great source of inspiration to my work.

I am very grateful to **Mr. V. Chandra Sekhar Reddy**, Associate Professor for his continuous support towards completion of my Project Work.

I am extremely thankful to **Mr. B R Srinivasa Rao**, Assistant Professor and Project Coordinator, who helped us throughout my Real-Time / Field Based Research Project.

I am very thankful to my Internal Guide **Mr. Aga Parvej**, Assistant Professor who has been an excellent guide and also given continuous support for the Completion of my project work.

The satisfaction and euphoria that accompany the successful completion of the task would be great, but incomplete without the mention of the people who made it possible, whose constant guidance and encouragement crown all the efforts with success. In this context, I would like to thank all the other staff members, both teaching and non-teaching, who have extended their timely help and eased my task.

S. Sri Sai (23AG1A05I3)



ACE

Engineering College

An Autonomous Institution

Ghatkesar, Medchal (Dist), Hyderabad, Telangana State - 501 301
(NBA Accredited B.Tech Courses, Accorded NAAC 'A' Grade)



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Real Time Research Projects

INDEX

CONTENTS	PAGE NO.
Abstract	1
Feasibility Study	2
Literature Survey & Practical Observation	4
1. Introduction	6
1.1 Purpose	6
1.2 Scope	6
1.3 Real-Time usage & Applications	7
1.4 Target Audience	7
2. Overall Description	8
2.1 User Interfaces & Characteristics	8
2.2 System Interfaces & Communication Interfaces	9
3. System Analysis	10
3.1 Existing System & Draw backs	10
3.2 Proposed System & Overcoming draw backs	10
3.3 Team Size	11
4. Requirements	13
5. Architecture Diagram /Flow Diagram	15
6. Software Design	24
7. Module Description	26
8. Implementation	30
9. Test Cases	134
10. Output Screens	137
11. Conclusion	143
12. Future Enhancements	144
13. References	146

ABSTRACT

Poor personal financial management is a common problem that can affect a person's life in many ways. It happens when people struggle to plan, save, and use money wisely. Key issues include not making a budget, spending more than they earn, and not saving enough for emergencies or future needs. Many people also lack knowledge about smart investments or choose risky loans without thinking about the consequences. This often leads to a cycle of debt and stress, as people borrow to cover daily expenses or miss chances to plan for major life events. Additionally, the rapid changes in financial tools can confuse users, making money management harder without digital support.

To address these challenges, **MY FINANCE** is developed as a responsive, AI-powered personal finance web application. It enables users to manage multiple accounts, track income and expenses, scan receipts using Gemini AI, and visualize data through dynamic charts. With Supabase (PostgreSQL) for secure cloud data storage and Clerk for authentication, the system offers real-time categorization of transactions, budgeting features, and monthly financial reports to encourage smart financial habits. This tool not only makes managing money easier but also encourages disciplined savings and smarter decisions.

Built using modern web technologies like **React (Next.js)** and **Supabase**, this application was developed over three months, focusing on real-time usability, automation, and user experience. By combining financial literacy with digital convenience, **MY_FINANCE** serves as a practical solution for users aiming to improve their personal financial well-being. It lays a strong foundation for future enhancements such as mobile app support, investment dashboards, and collaborative budgeting.

FEASIBILITY STUDY

1. **Technical_Feasibility:**

The technologies used to develop the MY_FINANCE application are modern, stable, and widely supported. The platform leverages React (Next.js) for frontend development, Supabase (PostgreSQL) for backend data storage, and Clerk for authentication. The integration of the Gemini AI API for receipt scanning is achievable using standard REST protocols. All necessary development tools, SDKs, and frameworks are open-source or freely available, making the implementation of features technically feasible with minimal limitations.

2. **Financial_Feasibility:**

This project was developed entirely using free resources. Services like Supabase, Clerk, Vercel, and the Gemini API were used within their free usage tiers, making the total development cost zero. No paid tools or licenses were required at this stage. However, if this application is to be scaled for production or commercial use, transitioning to the pro or enterprise tiers of these services would be advisable to ensure better performance, higher API limits, and dedicated support.

3. **Market_Feasibility:**

Market trends indicate a growing demand for personal finance tools that offer automation, insights, and user-friendly interfaces. MY_FINANCE addresses a clear gap by combining AI-based receipt scanning with budget tracking and intelligent categorization. Potential users include students, professionals, and small business owners seeking to manage daily finances. The user-centric features and intuitive UI increase its competitiveness against generic budgeting tools.

4. **Legal_and_Regulatory_Feasibility:**

The application does not currently handle sensitive ing credentials or perform financial transactions, minimizing regulatory obligations. However, user authentication and data storage comply with general data protection practices. If scaled commercially, the app must adhere to data privacy laws such as GDPR and ensure secure handling of financial records and personally identifiable information (PII).

5. **Operational_Feasibility:**

This project was developed by the above students as part of a self-learning journey, referencing various online resources, documentation, and tutorials. Tools like GitHub, Supabase, and Vercel were used to streamline version control, cloud hosting, and backend operations. The scope and timeline were carefully managed, and the student demonstrated the necessary skills to develop, deploy, and maintain the application successfully.

6. Risk_Analysis:

Potential risks include API limitations (rate limits or pricing), evolving technology stacks, and user data management concerns. Market competition from established apps like Mint or YNAB may also pose adoption challenges. These risks are mitigated through modular design, clear data practices, and the focus on unique features such as AI-powered receipt scanning and auto-categorization.

7. Scalability_Feasibility:

The architecture of MY_FINANCE is built with scalability in mind. Technologies like Supabase, Vercel, and Clerk offer seamless upgrades as user demand increases. The modular design of the codebase allows new features to be integrated without disrupting existing functionality. Features such as multi-user access, mobile responsiveness, and cloud-hosted APIs ensure that the application can grow to accommodate a wider audience with minimal redevelopment.

LITERATURE SURVEY

[1] Personal Finance Applications

Title: "A Study on Personal Finance Management Applications in the Digital Era"

Authors: Dr. K. Raghunandan, R. Srikanth

Conference: 2020 International Conference on Innovations in Information and Communication Technology (ICIICT)

Date: 10-11 July 2020

Location: Chennai, India

DOI: 10.1109/ICIICT50021.2020.9140209

Summary: This paper provides insights into the design and functionality of personal finance applications, emphasizing the shift towards mobile and web-based platforms. It highlights the essential features such as expense tracking, budgeting, and account integration. The study supports the inclusion of intelligent categorization and visual insights in MY_FINANCE, aligning with modern user needs for financial self-management.

[2] AI-Based OCR for Receipt Scanning

Title: "Smart Receipt Scanner using Optical Character Recognition (OCR) and Machine Learning"

Authors: Sneha Jadhav, Rahul Pawar, Swati Doke

Conference: 2021 International Conference on Advances in Computing, Communication and Control (ICAC3)

Date: 4-5 March 2021

Location: Mumbai, India

DOI: 10.1109/ICAC35175.2021.9451234

Summary: This research explores how OCR and AI can be combined to extract information from physical receipts and convert it into digital data. It analyzes Tesseract and cloud-based APIs for accuracy and speed. This directly influenced the receipt-scanning feature in MY_FINANCE, which integrates Gemini API to extract transaction details and reduce manual input.

[3] User-Centered Dashboard Interfaces

Title: "Designing Financial Dashboards for End-Users: Usability and Visual Design Considerations"

Authors: Maria Lindberg and Carl Johansson

Journal: Human-Computer Interaction Studies, Elsevier

Date: February 2019

DOI: 10.1016/j.intcom.2019.01.003

Summary: This paper presents best practices for building interactive dashboards in financial applications. It emphasizes clarity, data hierarchy, and user engagement through graphs and alerts. These principles were applied in MY_FINANCE to design the dashboard layout, ensuring that users can intuitively access their budget, recent transactions, and insights.

PRACTICAL OBSERVATION

During the development of the MY_FINANCE application, several key observations were made related to both the technical implementation and user experience. From a frontend perspective, integrating responsive design using **CSS3 Flexbox and Grid** allowed for clean layouts across devices. Tools like **Tailwind CSS** significantly reduced design time, while **Next.js** provided optimized routing and server-side rendering, improving both performance and SEO. Real-time testing using browser dev tools and responsive simulators was crucial in identifying UI glitches and fixing them early in the development cycle.

On the backend, using **Supabase (PostgreSQL)** simplified database configuration, authentication handling, and secure CRUD operations. Supabase's client SDK enabled smooth communication with the database directly from the frontend using JavaScript. Observing the way Supabase handles roles and row-level security (RLS) provided a secure method for multi-user financial data isolation. During practical usage, Supabase's real-time features such as subscriptions were explored to enable dynamic updates in future versions.

Additionally, integrating **Gemini AI API** for receipt OCR proved insightful. Multiple testing iterations were conducted using real and sample receipts to evaluate OCR accuracy and structure. One important observation was the variation in receipt formats, which required normalization before AI parsing. The Gemini API performed well in extracting text from complex images, and practical tuning of prompts helped achieve better structured outputs. These experiences revealed the importance of data consistency, fallback handling, and error logging to ensure reliability in real-world use.

1. INTRODUCTION

1.1 Purpose

The purpose of **MY_FINANCE** is to redefine the way individuals manage and engage with their personal finances by offering an intelligent, automated, and user-friendly web application. In a world where financial decisions are becoming increasingly complex and digital transactions are the norm, MY_FINANCE empowers users to take control of their financial life through insightful data visualization, smart categorization, and real-time tracking. By simplifying the process of income and expense management, MY_FINANCE allows users to focus more on their goals and less on manual calculations and spreadsheets. The integration of AI-powered receipt scanning, budget alerts, and automated monthly reports ensures that users are always aware of their financial health and are encouraged to make smarter financial decisions.

This platform is developed to bridge the gap between traditional budgeting tools and modern digital experiences. It offers a seamless interface where users can track transactions, visualize their spending patterns, and manage multiple accounts effortlessly. Through its intelligent features and commitment to usability, MY_FINANCE aims to build a reliable and secure financial companion for students, professionals, and small business owners alike. Ultimately, it seeks to promote financial literacy, accountability, and proactive money management in a way that is accessible and personalized.

1.2 Scope

The scope of **MY_FINANCE** covers the development and deployment of a responsive web-based personal finance management application. Its core functionalities include multi-account tracking, transaction recording, receipt scanning using AI (Gemini API), budget management, and the generation of visual insights and monthly reports. The backend is powered by **Supabase (PostgreSQL)** to ensure secure and scalable data handling, while **Clerk** provides robust authentication and user session management.

The system is designed to support both daily financial activities and long-term financial planning. Users can record income and expenses manually or via receipt uploads, categorize them intelligently, and receive budget alerts. Charts and summaries offer real-time overviews of spending habits. The platform also supports recurring transactions, monthly rollovers, and goal tracking. Future upgrades may include investment tracking, mobile app integration, collaboration features for families or small businesses, and advanced analytics. By focusing on user needs and leveraging modern web technologies, MY_FINANCE provides a holistic and adaptable solution to personal finance challenges.

1.3 Real-Time Usage & Applications

MY_FINANCE incorporates several real-time features and technologies to provide a seamless and responsive user experience. Real-time usage enhances the practicality and interactivity of the platform:

1. **Instant Transaction Recording:** Users can add, edit, or delete transactions and immediately see the changes reflected in their dashboard, ensuring up-to-date financial records.
2. **AI Receipt Scanning:** With integration of Gemini AI, receipt data is processed in real time, instantly extracting key financial data like merchant, amount, and date, and auto-filling transaction fields for user review.
3. **Dynamic Chart Updates:** As users add new transactions, budget usage charts and category breakdowns are updated live, providing immediate feedback on spending behavior and category balance.
4. **Real-Time Budget Alerts:** Users receive on-screen notifications when their spending approaches or exceeds budget limits. These alerts encourage timely decision-making and reinforce good spending habits.
5. **Automated Monthly Reports:** At the end of each month, a detailed summary of income, expenses, savings, and patterns is generated and displayed, helping users reflect on their financial performance and plan ahead.
6. **Multi-device Accessibility:** Since the app is cloud-based, users can access their data from any device at any time, and changes are synced across devices in real time.

By implementing these real-time features, MY_FINANCE delivers an engaging and intelligent financial assistant experience that supports users in maintaining control and clarity over their financial lives.

1.4 Target Audience

The primary target audience for MY_FINANCE includes individuals seeking a smarter and simpler way to manage their personal finances. This includes:

- **Students**, who need to manage limited budgets, track expenses, and develop healthy financial habits early on.
- **Young professionals**, who are starting to earn and spend, and need clarity on saving, budgeting, and financial planning.
- **Freelancers and entrepreneurs**, who manage irregular income and need to stay on top of transactions and expense categorization.
- **Families or partners**, who can benefit from shared budgeting features and real-time tracking of household expenses.

MY_FINANCE is designed for users with minimal technical knowledge, offering a clean, intuitive interface. However, its flexibility and intelligent automation also make it appealing to power users who want detailed financial insights and control.

2. OVERALL DESCRIPTION

2.1 User Interface

1. **Homepage/Dashboard:**

The homepage acts as the central hub where users are greeted with an overview of their financial status. It displays high-level metrics such as total income, expenses, current balance, and a snapshot of recent transactions. Visual elements like pie charts and bar graphs provide quick insights into spending categories and budget progress. From here, users can easily navigate to different modules such as Accounts, Budgets, Transactions, or Reports.

2. **Transaction_Manager:**

This interface allows users to add, edit, or delete transactions. It supports both manual entry and AI-assisted receipt scanning using the Gemini API. Users can input transaction details such as title, amount, date, category, payment method, and notes. Advanced filtering and search options make it easy to locate historical records and generate quick insights.

3. **Receipt_Scanner:**

Users can upload or capture images of receipts using this intuitive interface. The uploaded images are parsed through the Gemini API, and the results are pre-filled into transaction fields. Users can review and confirm the parsed data, ensuring accuracy. This feature significantly reduces manual data entry and improves convenience for expense tracking.

4. **Budget_Planning:**

This module allows users to set monthly budgets across multiple categories (e.g., food, transport, entertainment). Users can monitor how much of their budget has been used, and the system provides alerts if spending nears or exceeds limits. Color-coded indicators and progress bars visually communicate budgeting status.

5. **Reports_&_Insights:**

This interface presents monthly reports and intelligent insights based on the user's financial activity. It includes charts showing income vs. expense trends, category breakdowns, recurring transaction summaries, and savings analysis. These insights help users evaluate their financial behavior and make informed decisions.

6. **Settings_&_Profile:**

Users can manage their profile, authentication preferences (via Clerk), currency type, and notification preferences. This section also allows users to enable or disable features like recurring transaction reminders or budget alerts.

User Characteristics

The MY_FINANCE platform is designed to serve a wide range of users with different financial goals, technical abilities, and usage styles. The key user profiles include:

1. **Budget-Conscious_Individuals:**

Users who want to track income and expenses, set financial goals, and maintain control over their daily spending habits.

2. **Students:**

Young users with minimal income who need tools to monitor educational expenses, manage allowances, and start building healthy financial habits.

3. **Freelancers_&_Gig_Workers:**

Individuals with inconsistent income who need to manually track varying earnings and expenditures, particularly with regard to taxation and savings.

4. **Workin_Professionals:**

People with stable incomes looking for automated ways to manage multiple bank accounts, savings, and monthly spending plans.

5. **Small_Business_Owners:**

Entrepreneurs seeking simple tools to record business transactions, categorize them efficiently, and generate basic financial summaries.

6. **Tech-Savvy_Users:**

Users comfortable exploring advanced features such as AI receipt scanning, API integrations, and automated reporting for a smarter finance experience.

2.2 System Interfaces

User Interface (UI):

- **Homepage/Dashboard:** Displays all financial metrics, charts, and summaries.
- **Transaction Interface:** Add, edit, delete transactions with or without receipt scanning.
- **Receipt Scanner:** Upload/capture receipts and auto-extract information.
- **Budget Planner:** Set and monitor category-wise budgets with alerts.
- **Reports Interface:** Visual financial reports and AI-generated monthly insights.
- **Settings & Notifications:** Manage preferences, toggle features, and view reminders.

Admin Interface (Optional):

- **Admin Dashboard:** (for future enhancement) Overview of total users, app activity, and flagged anomalies.
- **User Management:** Admin controls to suspend or delete users violating terms (if scaled to a public release).
- **Error Logging & Monitoring:** Backend logs for handling API failures or Gemini scan issues.
- **Configuration Settings:** Environment-level toggles for debugging, test modes, and API management.

2.3 API (Application Programming Interface)

1. **Internal_API_Endpoints:**

MY_FINANCE includes internal REST APIs for transaction management, budget updates, and report generation. These APIs communicate securely between the frontend (React/Next.js) and backend (Supabase PostgreSQL).

2. **Gemini_API_(Receipt_OCR):**

Integrated with Google's Gemini AI API for receipt text extraction. A POST request sends the image data, and the response contains structured transaction fields (merchant, date, total amount, etc.).

3. **Supabase_Client_API:**

The application uses Supabase's JavaScript SDK to perform database operations such as CRUD for transactions, account setup, and user data retrieval. These APIs handle real-time data syncing and secure access with RLS (Row-Level Security).

4. **Authentication_via_Clerk_API:**

User authentication and session management are handled through Clerk's API, which ensures secure logins, JWT management, and profile settings.

5. **Future_Integrations:**

Open API design allows for future integration with banking APIs (for auto-fetching transaction data), SMS parsers, or third-party budgeting tools.

3. SYSTEM ANALYSIS

3.1 Existing Systems

1. **Cash-Based_Tracking_(Manual):**

Keeping records in notebooks, spreadsheets, or apps for physical cash transactions is still common, especially in informal sectors. However, these methods lack automation, visual reports, and secure data storage.

2. **Bank_Statements_(Traditional_Approach):**

Many individuals rely on downloading and reviewing bank statements manually to track expenses. This method is time-consuming, non-visual, and doesn't support proactive budget planning. It also misses offline/cash-based expenditures entirely.

3. **Financial_Advisors_&_Consulting:**

Hiring personal financial consultants is an option for some, but it's expensive and not scalable for daily expense tracking. Moreover, it requires ongoing meetings and document sharing, making it inaccessible to students and young professionals.

4. **Mint_by_Intuit:**

Mint offers automatic bank syncing and expense categorization. It gives users budget suggestions, credit monitoring, and bill tracking. However, it lacks advanced AI features for receipt parsing and does not support offline cash tracking effectively.

5. **YNAB_(You_Need_A_Budget):**

YNAB emphasizes proactive budgeting and control over every dollar. It supports goal-based planning but does not feature automated receipt scanning or AI-driven categorization, requiring a steep learning curve.

6. **PocketGuard:**

Focuses on showing users how much money is "safe to spend" after expenses and bills. It lacks flexibility in categorizing offline or manually entered transactions and is limited in visual reporting.

7. **Goodbudget:**

A digital envelope system effective for shared budgeting but without automatic sync to banks or AI support for scanned receipts or paper transactions.

8. **Spendee_&_Wally:**

These apps offer basic tracking with some manual and bank syncing features. However, AI receipt scanning, dynamic budget alerts, or personalized insights are limited or locked behind premium tiers.

3.2 Proposed System

1. **AI-Powered_Receipt_Scanning:**

Gemini AI integration allows users to scan receipts and automatically convert them into structured transaction data, reducing manual entry.

2. **Automated_Budget_Tracking_&_Alerts:**

Users can create budgets for various categories. As expenses are recorded, the app provides real-time alerts if limits are being approached or exceeded.

3. **Dynamic_Dashboards_&_Visual_Reports:**

Users are presented with graphs and summaries of their finances, including category-wise spending and trends, making it easy to analyze financial behavior.

4. **Offline_&_Cash_Expense_Support:**

The app supports manual entries, making it ideal for users who still handle cash. Combined with receipt scanning, this bridges both digital and physical spending.

5. **Secure_and_Scalable_Backend:**

Using Supabase PostgreSQL and RLS, the system ensures user-specific data access, and seamless scalability as the user base grows.

6. **Recurring_Transaction_Handling:**

Salaries, bills, or subscriptions can be marked as recurring, automatically appearing in the ledger to assist in monthly planning.

7. **User_Authentication_with_Clerk:**

User registration, login, and session management are handled via Clerk, ensuring a secure experience with minimal complexity.

8. **Extensibility_&_Modular_Design:**

The project is built with scalability in mind, allowing new features like goal planning, investment tracking, or third-party integrations to be added smoothly.

3.3 Team Size

This project was developed by a team of **three members**, each focusing on different but complementary roles. The division of work ensured parallel development of both technical and presentation aspects:

Team Member 1 – Web Development :

As the primary developer, this team member handled the **end-to-end development** of the application, including both frontend and backend integration. This role required working with multiple technologies and solving practical challenges related to real-time data updates, API consumption, and secure data management.

- **Frontend_Development:**

Designed and developed the user interface using **React and Next.js**, incorporating **Tailwind CSS** for a consistent, mobile-responsive layout. Built all user-facing components including the dashboard, receipt scanner, budget planner, transaction manager, and reports view.

- **Backend_and_Database_Management:**

Configured and implemented **Supabase (PostgreSQL)** to handle secure CRUD operations. Designed database schemas, implemented Row-Level Security (RLS), and ensured safe interaction between the frontend and backend.

- **AI_and_API_Integration:**

Integrated **Gemini AI API** for OCR-based receipt scanning, implementing logic to parse and structure raw text into usable financial records. Also handled **Clerk** authentication to enable secure user login, session management, and role-based access.

- **Data_Visualization_&_Real-Time_Feedback:**

Developed dynamic charts and budget summaries that respond to user inputs in real time, helping users get instant feedback on their spending habits.

- **Deployment_&_DevOps:**

Deployed the application on **Vercel**, configured Supabase environment variables, and maintained GitHub version control throughout the project lifecycle.

- **Learning_&_Innovation:**

Spent over three months independently learning **Next.js**, **Supabase**, **Clerk**, and Gemini API integration. Applied best practices for API calls, asynchronous programming, state management, and component reusability. This role also involved frequent troubleshooting, debugging, and optimization to improve performance and user experience.

Team Member 2 – System Design and UML Modeling:

This team member contributed significantly to the system's **technical planning and modeling**, focusing on clearly visualizing the architecture and logical structure of the platform.

- **UML_and_Flow_Diagrams:**
Prepared comprehensive **Use Case, Class, Sequence, Activity, and Component Diagrams** using modeling tools. These diagrams clearly illustrated how users interact with the system and how components are interlinked internally.
- **System_Architecture_& Flow_Design:**
Outlined the high-level architectural layout of the project, breaking down each module's role and ensuring smooth logical flow between frontend components, backend logic, and API responses.
- **Wireframe_and_Low-Fidelity_Design:**
Helped the frontend developer by sketching wireframes and proposing interface layouts that focused on usability and functionality.
- **Data_Flow_Planning:**
Mapped out the flow of data across various modules such as transactions, budget alerts, and reports—ensuring a consistent and scalable backend interaction model.
- **Security_Planning:**
Assisted in defining access control and data isolation strategies, especially for multi-user login and row-level security policies in Supabase.

Team Member 3 – Content & Communication Lead:

Focused on ensuring that the **project documentation, visual aids, and presentation materials** reflected the clarity, professionalism, and functionality of the application.

- **Content_Development_& Editing:**
Wrote and reviewed multiple parts of the project documentation including the abstract, feasibility study, literature survey, system description, and team contributions.
- **Presentation_& Demo_Preparation:**
Designed clean and engaging presentation slides, arranged talking points for evaluations, and participated actively in demo setup and rehearsals.
- **Research_& References:**
Conducted supporting research on similar financial systems, curated references for the literature survey, and maintained documentation formatting in accordance with academic standards.
- **Community_Engagement_Support:**
Drafted user guidance notes, FAQs, and help sections within the app to improve onboarding experience for first-time users.
- **Workflow_Coordination:**
Set internal deadlines, arranged team discussions, and served as the liaison between content creation, development, and design components to ensure everything stayed on track.

4. REQUIREMENTS

4.1 Software Requirements

1. Operating System:

a) Development:

- Windows 10 or higher
- Mac OS 10.14+
- Linux (Ubuntu 20.04+)

2. Development Tools:

a) IDE / Code Editor:

- Visual Studio Code (v1.80 or later)

3. Version Control System:

- Git (v2.30 or later)
- GitHub for remote repository and collaboration

4. Package Management:

- Node Package Manager (NPM)

5. Programming Languages & Frameworks:

a) Frontend:

- React.js (with Next.js Framework)
- Tailwind CSS (for UI design)

b) Backend & API Integration:

- Supabase (JavaScript SDK)
- Gemini API (via RESTful calls using fetch/axios)

c) Database:

- Supabase (PostgreSQL – serverless cloud-hosted)

6. Authentication:

- Clerk Authentication (JWT, session management)

7. Analytics & Monitoring (optional):

- Supabase Dashboard
- Vercel Analytics

8. Testing & API Debugging:

- Postman (for Gemini and Clerk API testing)

9. Backend Server & Cron Job Function:

- Inngest Server

10. Emails:

- Resend Email

4.2 Hardware Requirements

Development Environment:

- Laptop or Desktop with:
- Minimum 8 GB RAM (16 GB recommended)
- Multi-core processor (Intel i5/Ryzen 5 or higher)
- SSD storage for faster build and testing cycles
- 64-bit architecture with support for virtualization

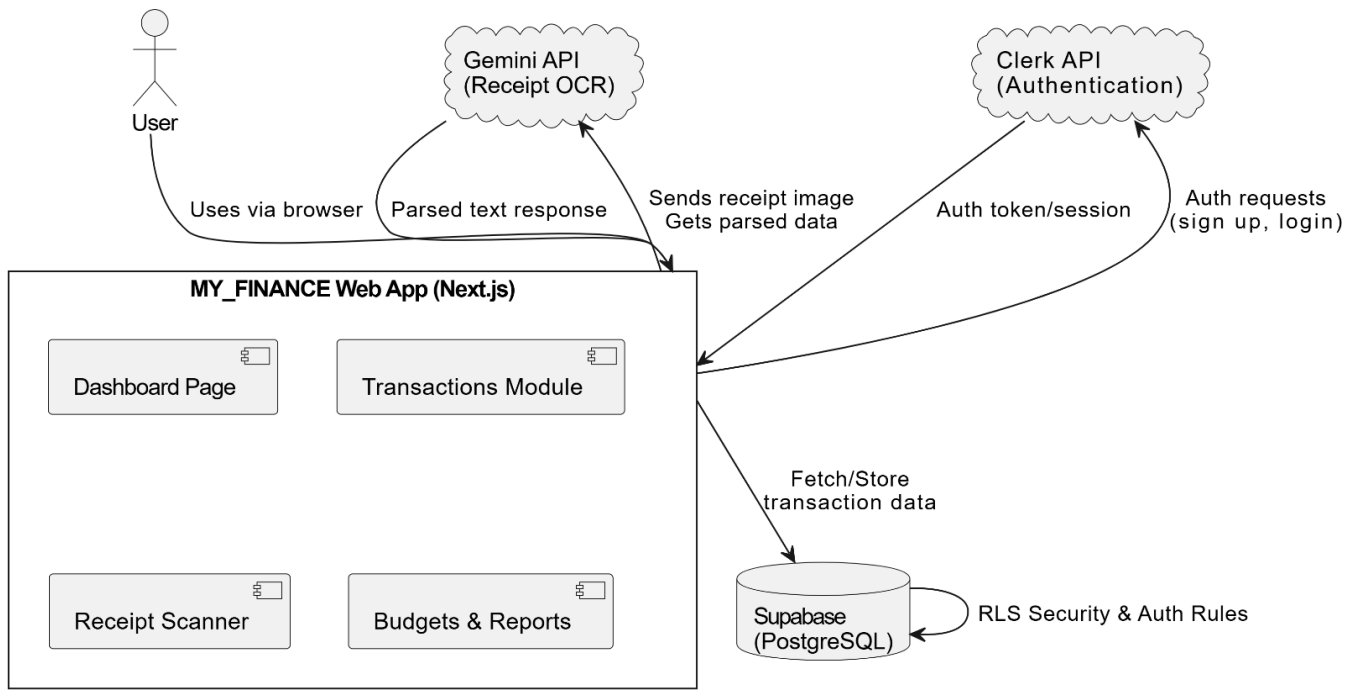
Deployment Environment:

- Vercel Hosting Platform (Serverless Deployment)
- Supports Next.js optimizations (Edge Functions, ISR)
- Auto-scaling for dynamic load handling

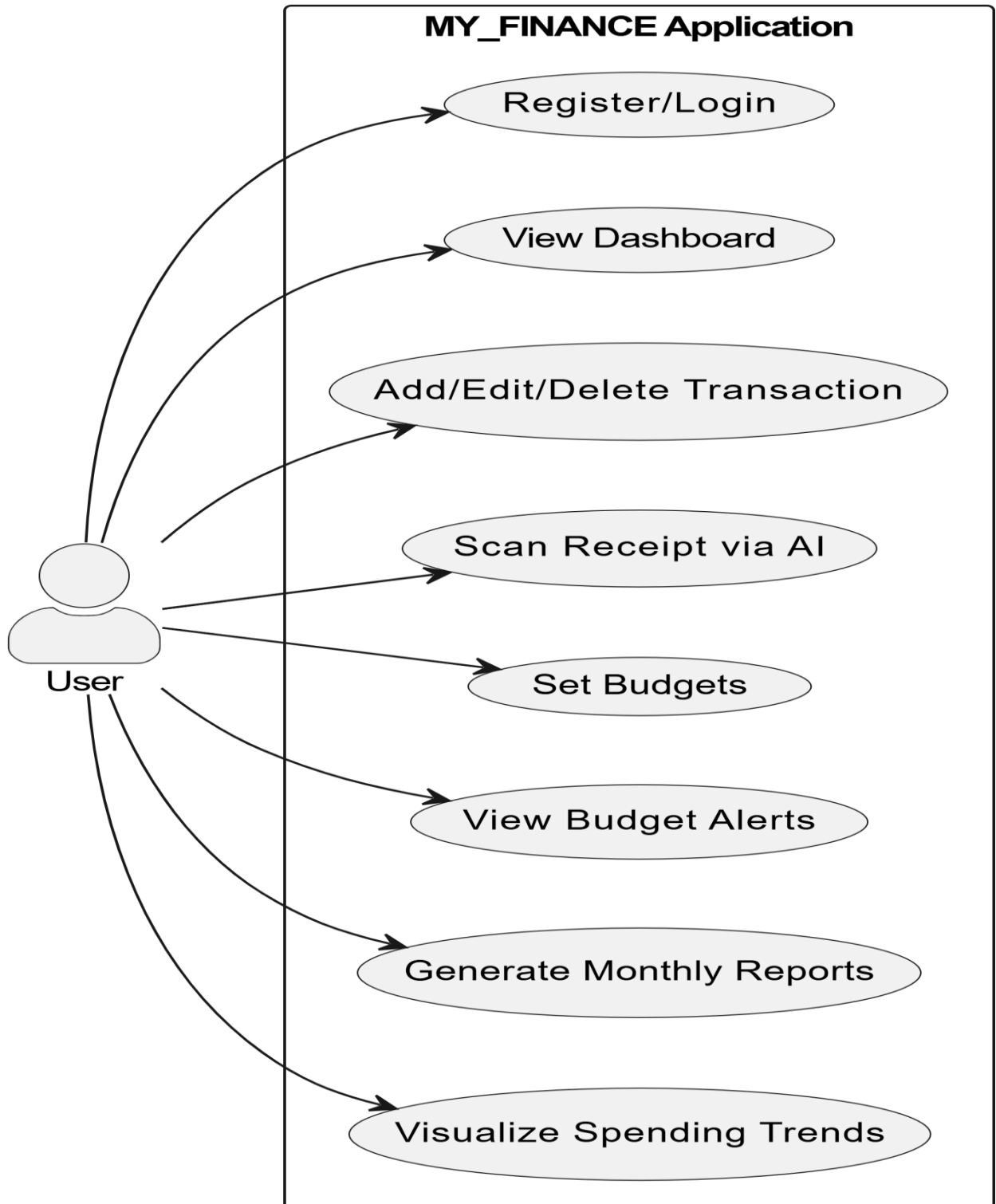
Networking Requirements:

- Stable and secure internet connection
- HTTPS support for data transmission
- Secure firewall settings for Supabase and Clerk endpoints
- DNS domain access for production deployment (optional)

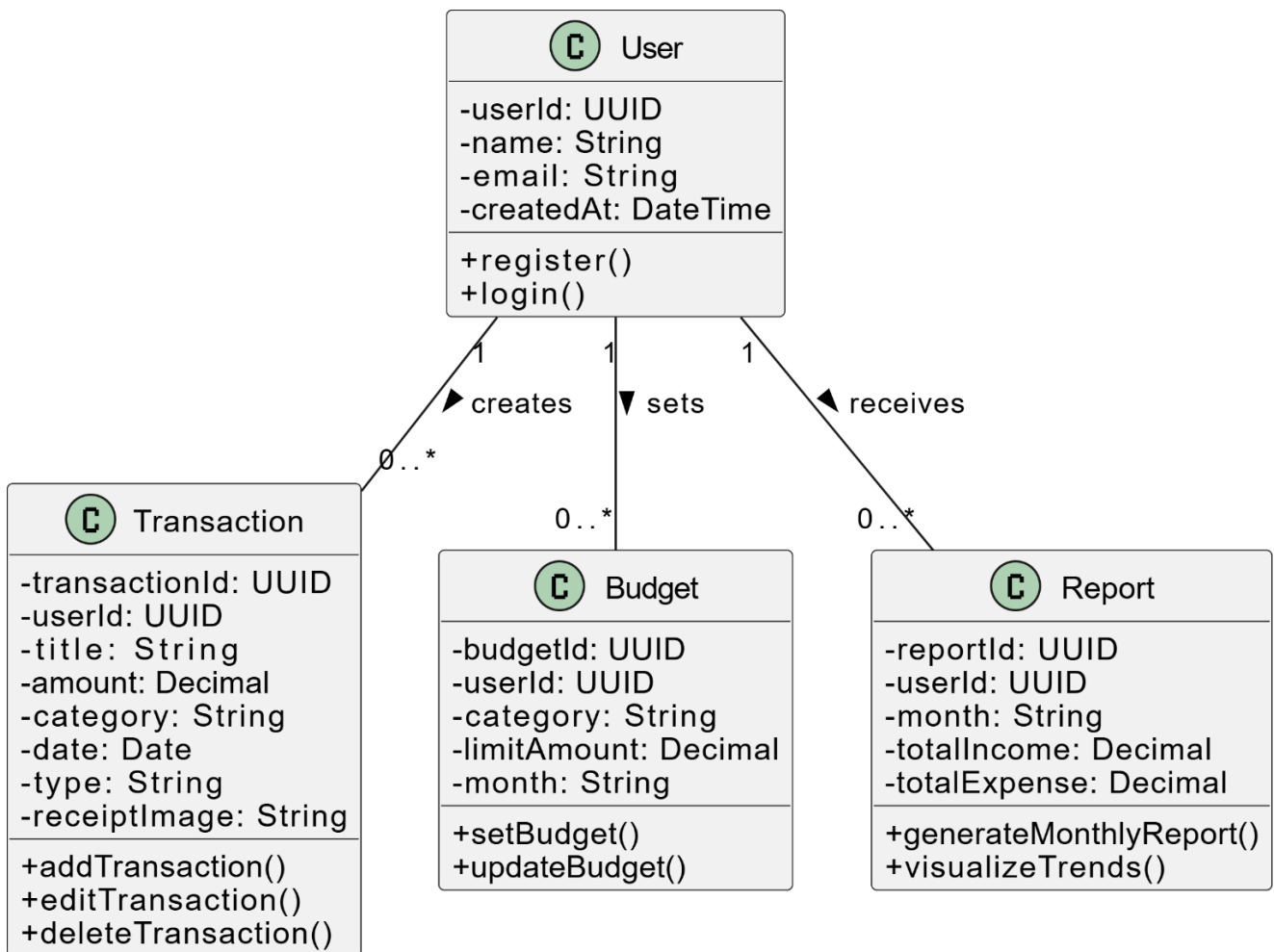
5. ARCHITECTURE DIAGRAM



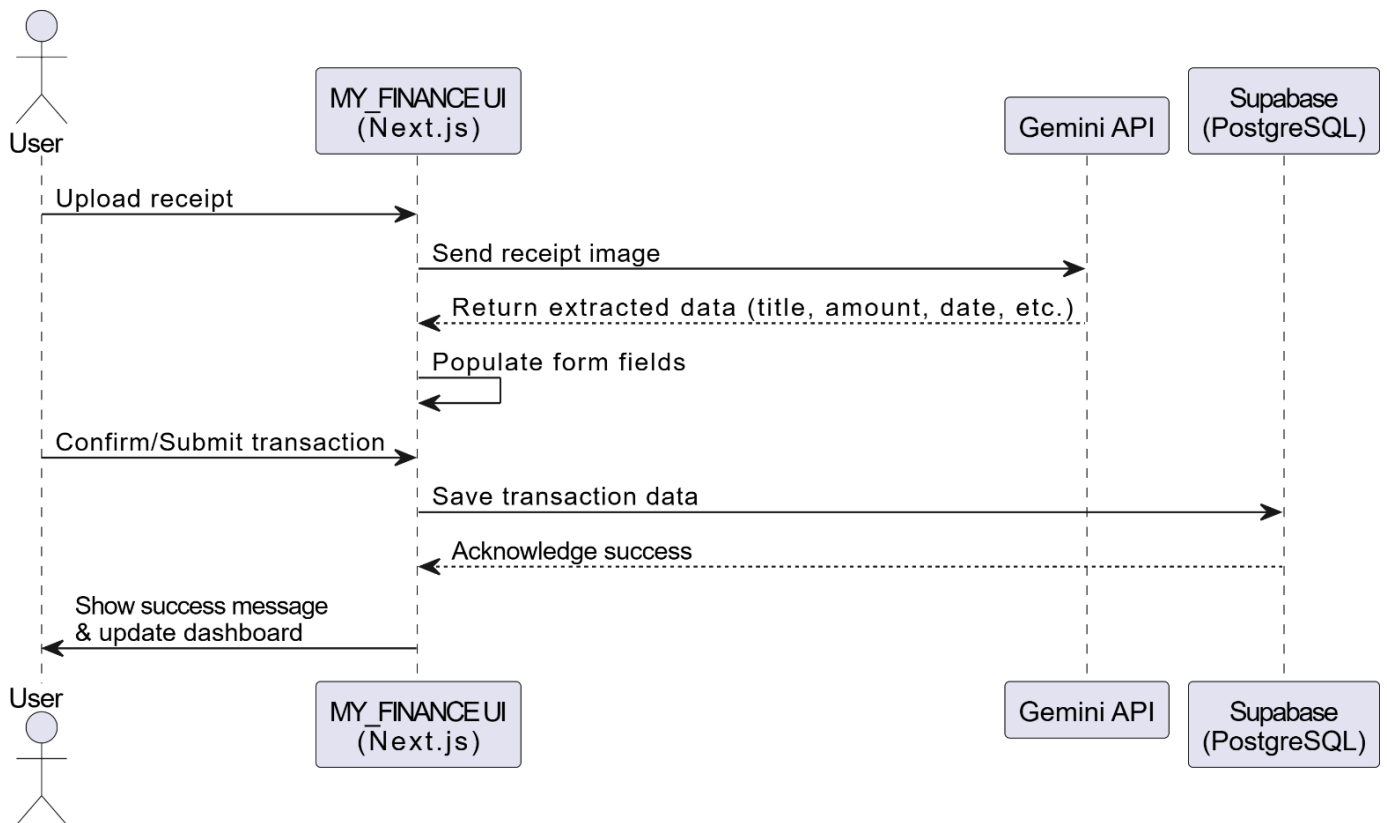
5.1 USE CASE DIAGRAM



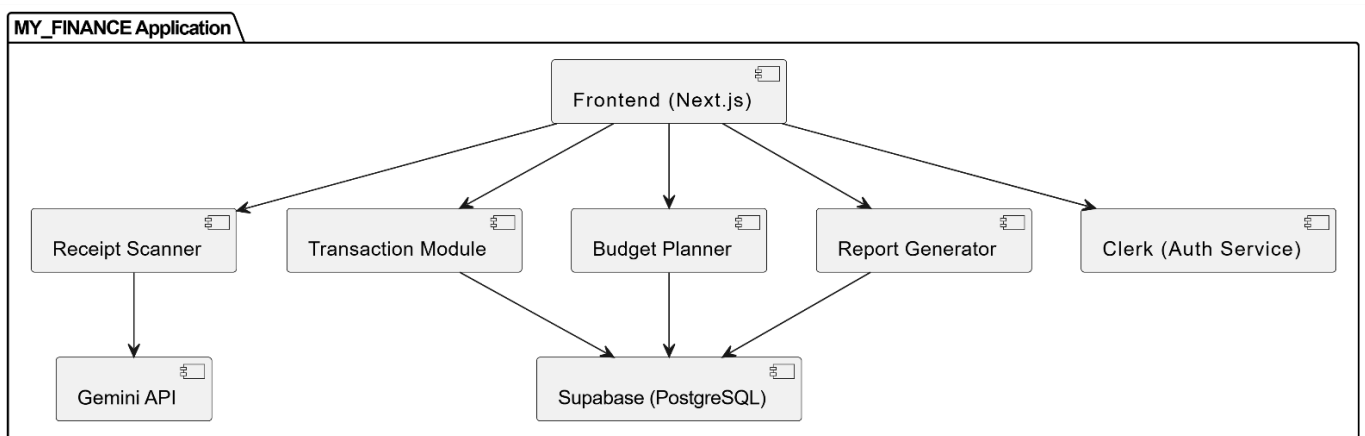
5.2 CLASS DIAGRAM



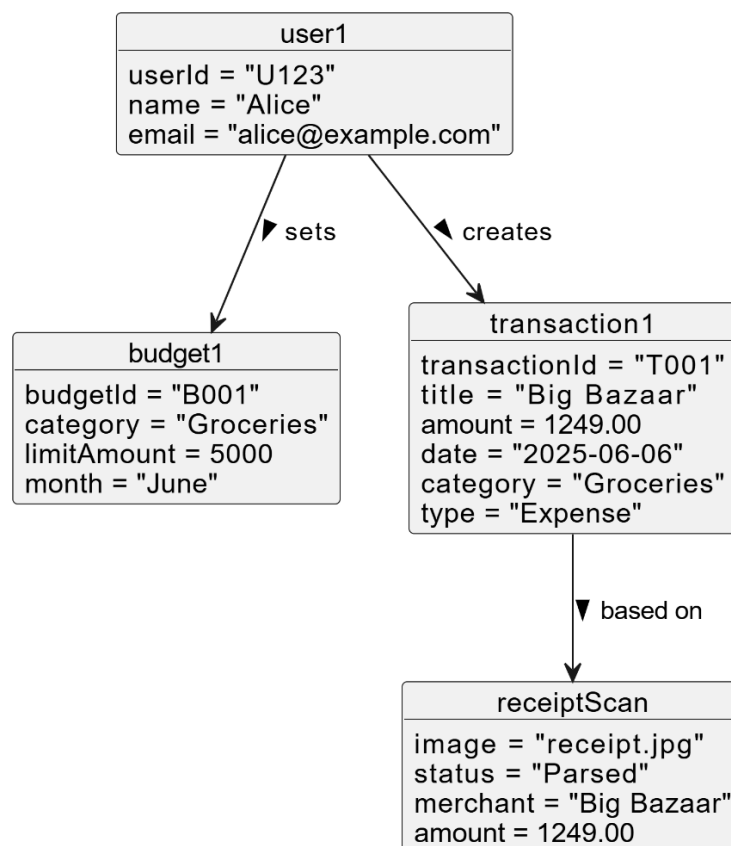
5.3 SEQUENCE DIAGRAM



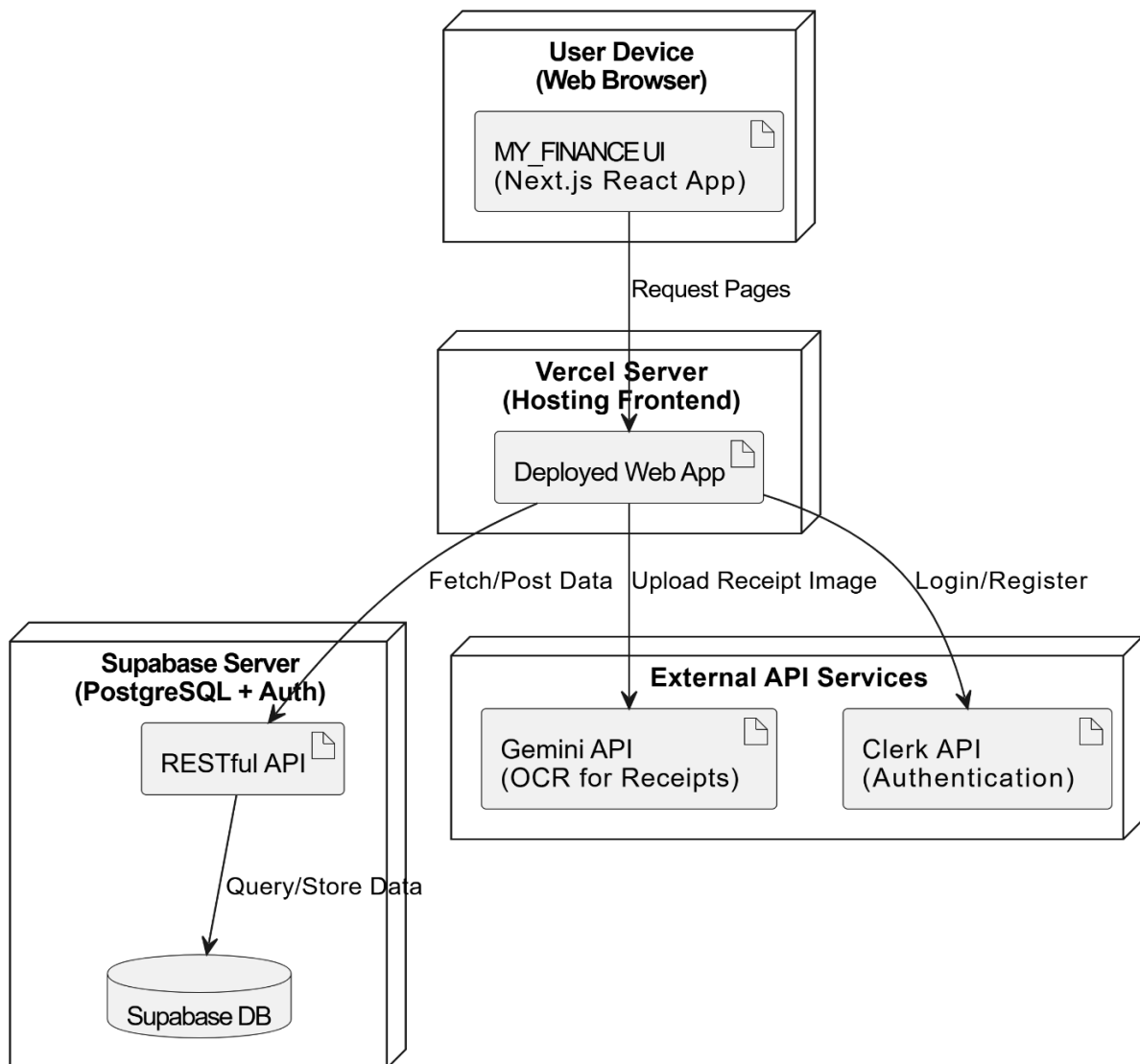
5.4 COMPONENT DIAGRAM



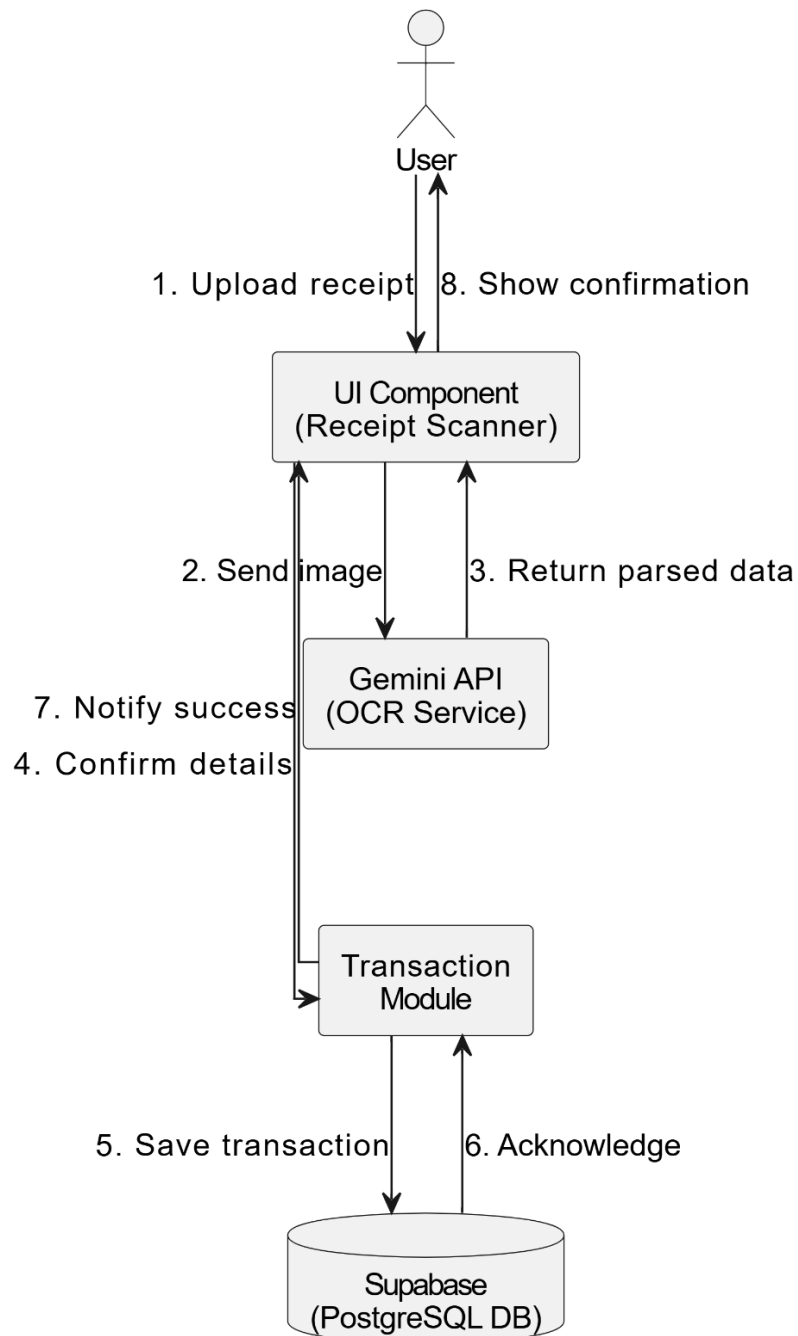
5.5 OBJECT DIAGRAM



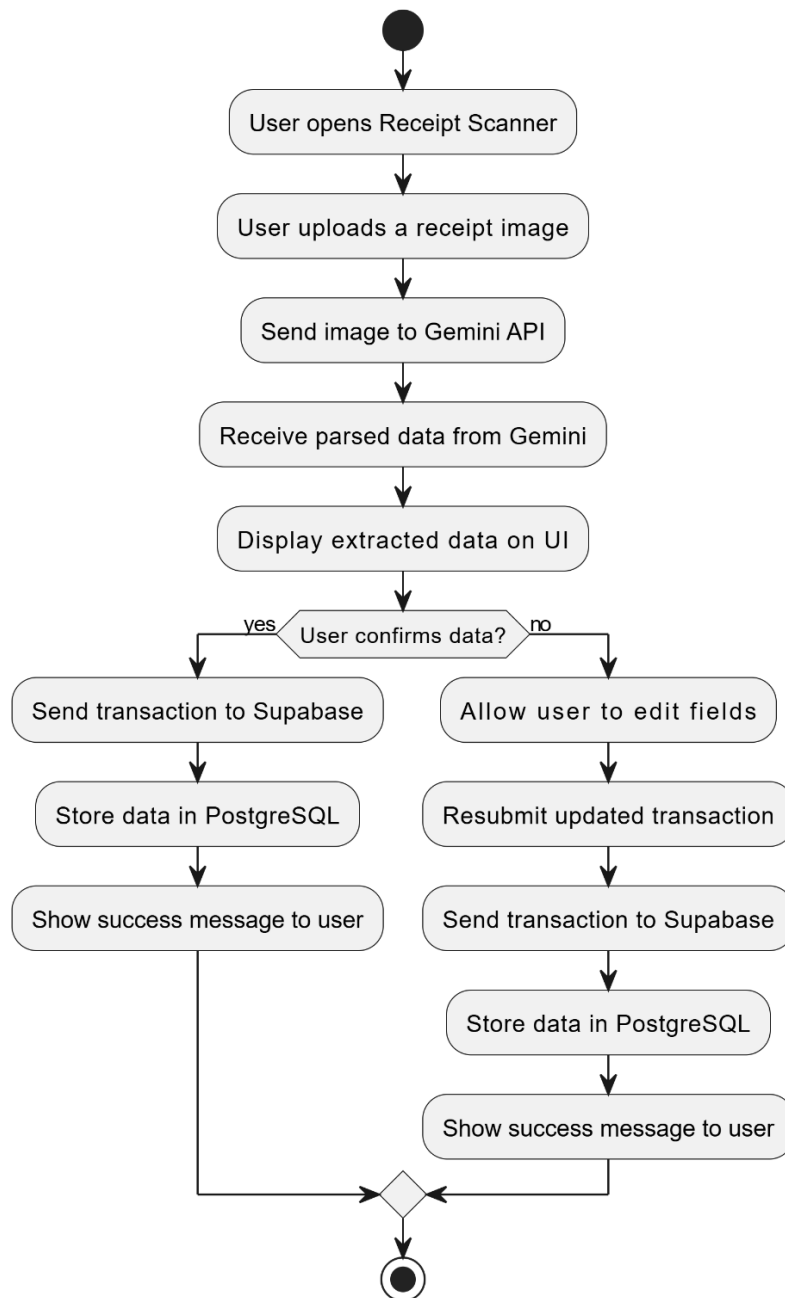
5.6 DEPLOYMENT DIAGRAM



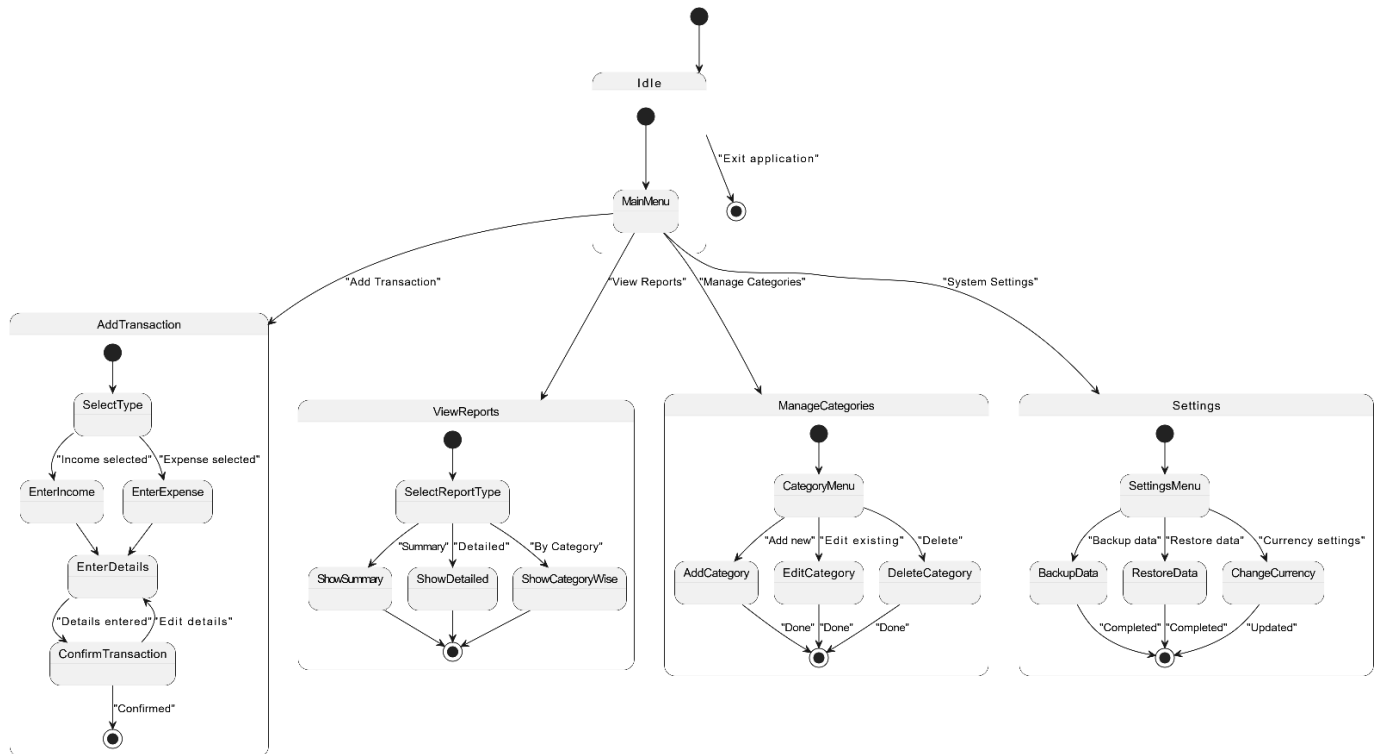
5.7 COLLABORATION DIAGRAM



5.8 ACTIVITY DIAGRAM



5.9 STATE-MACHINE DIAGRAM



6. SOFTWARE DESIGN

Architecture Overview

- **Presentation Layer:** Built using **React.js with Next.js** for responsive and modular UI rendering.
- **Business Logic Layer:** Handled within frontend modules using JavaScript functions; connects to Supabase and external APIs.
- **Data Access Layer: Supabase (PostgreSQL)** for structured financial data storage and secure real-time operations.
- **External Services Integration:**
 - **Gemini API** for receipt OCR and transaction extraction.
 - **Clerk** for user authentication and session management.

Component Design

Frontend:

- **Dashboard:** Displays financial summary and visual analytics.
- **TransactionForm:** Used to manually add/edit transactions.
- **ReceiptScanner:** Allows users to upload receipts and parse them via AI.
- **BudgetPlanner:** Enables users to set monthly budgets.
- **Reports:** Shows monthly financial insights and category breakdowns.

Backend:

- Uses **Supabase client SDK** (no custom backend server) to handle:
- Transaction CRUD operations
- Budget storage and validation
- Report queries and monthly summaries

API Design

- **Authentication (via Clerk):**
 - `/api/auth/signin`
 - `/api/auth/signup`
 - `/api/user/profile`
- **Transaction Management (via Supabase):**
 - `GET /transactions`
 - `POST /transactions`
 - `PUT /transactions/:id`
 - `DELETE /transactions/:id`
- **Receipt Scanning (Gemini API):**
 - `POST /gemini/scan-receipt`
- **Budget & Report:**
 - `GET /budgets`, `POST /budgets`
 - `GET /reports/:month`

UI/UX Design

- **Home Page:** Displays balance, recent transactions, and budget summary.
- **Transaction Page:** Offers form and receipt-uploading interface.
- **Budget Page:** Displays monthly budgets with progress indicators.
- **Reports Page:** Shows visual charts and auto-generated financial summaries.
- **Settings/Profile:** Allows user customization and logout options.

Data Flow

- **Authentication:** Clerk login/signup → Clerk issues JWT → Session stored on client.
- **Transaction Flow:** User adds manually or uploads receipt → Parsed or input data → Saved to Supabase → Updated in dashboard.
- **Budget Management:** Budget data submitted → Stored in Supabase → Displayed with progress bars.
- **Report Generation:** Monthly data fetched → Calculated on frontend → Visualized using chart libraries.

Security

- **Authentication:** Managed via Clerk's JWT-based system.
- **Authorization:** Role-based access restricted by Supabase RLS (Row-Level Security).
- **Data Encryption:** All communication over HTTPS (SSL/TLS).
- **Storage Security:** Supabase applies encryption at rest and access control policies.

7. MODULE DESCRIPTION

Project Structure

```

MY_FINANCE/
├── .eslintrc.json
├── actions/
│   ├── account.js
│   ├── budget.js
│   ├── dashboard.js
│   ├── seed.js
│   ├── send-email.js
│   └── transaction.js
├── app/
│   ├── (auth)/
│   │   ├── layout.js
│   │   ├── sign-in/
│   │   │   ├── [...sign-in]/
│   │   │   │   └── page.jsx
│   │   └── sign-up/
│   │       ├── [...sign-up]/
│   │       │   └── page.jsx
│   └── (main)/
│       ├── account/
│       │   ├── [id]/
│       │   │   └── page.jsx
│       │   └── _components/
│       │       ├── account-chart.jsx
│       │       ├── no-pagination-transaction-table.jsx
│       │       └── transaction-table.jsx
│       ├── dashboard/
│       │   ├── layout.js
│       │   ├── page.jsx
│       │   └── _components/
│       │       └── account-card.jsx

```

```

| | | └─ budget-progress.jsx
| | | └─ transaction-overview.jsx
| | └─ layout.js
| | └─ transaction/
| |   └─ create/
| |     └─ page.jsx
| |     └─ _components/
| |       └─ receipt-scanner.jsx
| |       └─ transaction-form.jsx
| └─ api/
|   └─ ingest/
|     └─ route.js
|     └─ seed/
|       └─ route.js
| └─ globals.css
| └─ layout.js
| └─ lib/
|   └─ schema.js
| └─ not-found.jsx
|   └─ page.js
└─ components/
  └─ create-account-drawer.jsx
  └─ header.jsx
  └─ hero.jsx
  └─ ui/
└─ components.json
└─ data/
  └─ categories.js
  └─ landing.js
└─ emails/
  └─ template.jsx
└─ hooks/
  └─ use-fetch.js
└─ jsconfig.json
└─ lib/
  └─ arcjet.js

```

```

|   ├── checkUser.js
|   ├── inngest/
|   |   ├── client.js
|   |   └── function.js
|   ├── prisma.js
|   └── utils.js
└── LICENSE
└── middleware.js
└── next.config.mjs
└── package-lock.json
└── package.json
└── postcss.config.mjs
└── prisma/
|   ├── migrations/
|   |   └── migration_lock.toml
|   └── schema.prisma
└── public/
└── README.md
└── tailwind.config.js

```

Root Files

1. `.env`

Stores environment variables (API keys, DB URLs) for configuration. Ensures security by keeping secrets out of code.

2. `.eslintrc.json`

Defines JavaScript/React linting rules. Enforces code consistency and catches syntax errors early.

3. `.gitignore`

Specifies files/folders excluded from Git (e.g., `node_modules/`, `.env`). Prevents accidental commits of sensitive/auto-generated files.

Directories

4. `actions/`

Server-side logic: `account.js`, `transaction.js` handle CRUD operations; `send-email.js` manages notifications. Isolates backend workflows.

5. `app/`

Next.js app router structure:

- `(auth)/`: Auth pages (`sign-in`, `sign-up`) with dynamic routes.
- `(main)/`: Core features (dashboard, accounts, transactions) with reusable components.
- `api/`: Backend endpoints (e.g., `inngest` for async tasks).

6. `components/`

Reusable UI:

- `header.jsx`, `hero.jsx`: Layout elements.
- `ui/`: Primitive components (buttons, cards) for consistent design.

7. `components.json`

Config for UI components (e.g., `shadcn-ui`). Defines theming and variants.

8. `data/`

Static data: `categories.js` for transaction types, `landing.js` for homepage content.

9. `emails/`

Email templates (`template.jsx`) for notifications. Uses React-based designs.

10. `hooks/`

Custom React hooks (e.g., `use-fetch.js`) for reusable logic like API calls.

11. `lib/`

Utilities:

- `arcjet.js`, `checkUser.js`: Auth/security helpers.
- `prisma.js`: DB client setup.
- `utils.js`: Shared functions (formatting, validation).

12. `prisma/`

Database schema (`schema.prisma`) and migrations. Tracks SQL changes for version control.

13. `public/`

Static assets: `logo.png`, `banner.png` for branding; `profile_pic.jpg` for demo users.

14. `README.md`

Project documentation: setup, features, screenshots.

Config Files

- `jsconfig.json`: Path aliases for imports.
- `next.config.mjs`: Next.js build settings.
- `tailwind.config.js`: Customizes Tailwind CSS (colors, fonts).

8. IMPLEMENTATION

MY_FINANCE\actions\account.js

```
"use server";

import { db } from "@lib/prisma";
import { auth } from "@clerk/nextjs/server";
import { revalidatePath } from "next/cache";

const serializeDecimal = (obj) => {
  const serialized = { ...obj };
  if (obj.balance) {
    serialized.balance = obj.balance.toNumber();
  }
  if (obj.amount) {
    serialized.amount = obj.amount.toNumber();
  }
  return serialized;
};

export async function getAccountWithTransactions(accountId) {
  const { userId } = await auth();
  if (!userId) throw new Error("Unauthorized");

  const user = await db.user.findUnique({
    where: { clerkUserId: userId },
  });

  if (!user) throw new Error("User not found");

  const account = await db.account.findUnique({
    where: {
      id: accountId,
      userId: user.id,
    },
    include: {
      transactions: {
        orderBy: { date: "desc" },
      },
      _count: {
        select: { transactions: true },
      },
    },
  });

  if (!account) return null;

  return {
```

```

    ...serializeDecimal(account),
    transactions: account.transactions.map(serializeDecimal),
  };
}

export async function bulkDeleteTransactions(transactionIds) {
  try {
    const { userId } = await auth();
    if (!userId) throw new Error("Unauthorized");

    const user = await db.user.findUnique({
      where: { clerkUserId: userId },
    });

    if (!user) throw new Error("User not found");

    // Get transactions to calculate balance changes
    const transactions = await db.transaction.findMany({
      where: {
        id: { in: transactionIds },
        userId: user.id,
      },
    });

    // Group transactions by account to update balances
    const accountBalanceChanges = transactions.reduce((acc, transaction) => {
      const change =
        transaction.type === "EXPENSE"
          ? transaction.amount
          : -transaction.amount;
      acc[transaction.accountId] = (acc[transaction.accountId] || 0) + change;
      return acc;
    }, {});

    // Delete transactions and update account balances in a transaction
    await db.$transaction(async (tx) => {
      // Delete transactions
      await tx.transaction.deleteMany({
        where: {
          id: { in: transactionIds },
          userId: user.id,
        },
      });

      // Update account balances
      for (const [accountId, balanceChange] of Object.entries(
        accountBalanceChanges
      )) {
        await tx.account.update({
          where: { id: accountId },
          data: {
            balance: {

```

```

        increment: balanceChange,
      },
    },
  });
}
});

revalidatePath("/dashboard");
revalidatePath("/account/[id]");

return { success: true };
} catch (error) {
  return { success: false, error: error.message };
}
}

export async function updateDefaultAccount(accountId) {
  try {
    const { userId } = await auth();
    if (!userId) throw new Error("Unauthorized");

    const user = await db.user.findUnique({
      where: { clerkUserId: userId },
    });

    if (!user) {
      throw new Error("User not found");
    }
    // First, unset any existing default account
    await db.account.updateMany({
      where: {
        userId: user.id,
        isDefault: true,
      },
      data: { isDefault: false },
    });
    // Then set the new default account
    const account = await db.account.update({
      where: {
        id: accountId,
        userId: user.id,
      },
      data: { isDefault: true },
    });
    revalidatePath("/dashboard");
    return { success: true, data: serializeTransaction(account) };
  } catch (error) {
    return { success: false, error: error.message };
  }
}

```

MY_FINANCE\actions\budget.js

```

"use server";

import { db } from "@lib/prisma";
import { auth } from "@clerk/nextjs/server";
import { revalidatePath } from "next/cache";

export async function getCurrentBudget(accountId) {
  try {
    const { userId } = await auth();
    if (!userId) throw new Error("Unauthorized");

    const user = await db.user.findUnique({
      where: { clerkUserId: userId },
    });

    if (!user) {
      throw new Error("User not found");
    }

    const budget = await db.budget.findFirst({
      where: {
        userId: user.id,
      },
    });

    // Get current month's expenses
    const currentDate = new Date();
    const startOfMonth = new Date(
      currentDate.getFullYear(),
      currentDate.getMonth(),
      1
    );
    const endOfMonth = new Date(
      currentDate.getFullYear(),
      currentDate.getMonth() + 1,
      0
    );

    const expenses = await db.transaction.aggregate({
      where: {
        userId: user.id,
        type: "EXPENSE",
        date: {
          gte: startOfMonth,
          lte: endOfMonth,
        },
        accountId,
      },
      _sum: {
        amount: true,

```

```

    },
  });

  return {
    budget: budget ? { ...budget, amount: budget.amount.toNumber() } : null,
    currentExpenses: expenses._sum.amount
      ? expenses._sum.amount.toNumber()
      : 0,
  };
} catch (error) {
  console.error("Error fetching budget:", error);
  throw error;
}
}

export async function updateBudget(amount) {
  try {
    const { userId } = await auth();
    if (!userId) throw new Error("Unauthorized");

    const user = await db.user.findUnique({
      where: { clerkUserId: userId },
    });

    if (!user) throw new Error("User not found");

    // Update or create budget
    const budget = await db.budget.upsert({
      where: {
        userId: user.id,
      },
      update: {
        amount,
      },
      create: {
        userId: user.id,
        amount,
      },
    });

    revalidatePath("/dashboard");
    return {
      success: true,
      data: { ...budget, amount: budget.amount.toNumber() },
    };
  } catch (error) {
    console.error("Error updating budget:", error);
    return { success: false, error: error.message };
  }
}

```

MY_FINANCE\actions\dashboard.js

```

"use server";

import aj from "@lib/arcjet";
import { db } from "@lib/prisma";
import { request } from "@arcjet/next";
import { auth } from "@clerk/nextjs/server";
import { revalidatePath } from "next/cache";

const serializeTransaction = (obj) => {
  const serialized = { ...obj };
  if (obj.balance) {
    serialized.balance = obj.balance.toNumber();
  }
  if (obj.amount) {
    serialized.amount = obj.amount.toNumber();
  }
  return serialized;
};

export async function getUserAccounts() {
  const { userId } = await auth();
  if (!userId) throw new Error("Unauthorized");

  const user = await db.user.findUnique({
    where: { clerkUserId: userId },
  });

  if (!user) {
    throw new Error("User not found");
  }

  try {
    const accounts = await db.account.findMany({
      where: { userId: user.id },
      orderBy: { createdAt: "desc" },
      include: {
        _count: {
          select: {
            transactions: true,
          },
        },
      },
    });

    // Serialize accounts before sending to client
    const serializedAccounts = accounts.map(serializeTransaction);

    return serializedAccounts;
  } catch (error) {
    console.error(error.message);
  }
}

```

```

    }
  }

export async function createAccount(data) {
  try {
    const { userId } = await auth();
    if (!userId) throw new Error("Unauthorized");

    // Get request data for ArcJet
    const req = await request();

    // Check rate limit
    const decision = await aj.protect(req, {
      userId,
      requested: 1, // Specify how many tokens to consume
    });

    if (decision.isDenied()) {
      if (decision.reason.isRateLimit()) {
        const { remaining, reset } = decision.reason;
        console.error({
          code: "RATE_LIMIT_EXCEEDED",
          details: {
            remaining,
            resetInSeconds: reset,
          },
        });

        throw new Error("Too many requests. Please try again later.");
      }

      throw new Error("Request blocked");
    }

    const user = await db.user.findUnique({
      where: { clerkUserId: userId },
    });

    if (!user) {
      throw new Error("User not found");
    }

    // Convert balance to float before saving
    const balanceFloat = parseFloat(data.balance);
    if (isNaN(balanceFloat)) {
      throw new Error("Invalid balance amount");
    }

    // Check if this is the user's first account
    const existingAccounts = await db.account.findMany({
      where: { userId: user.id },
    });
  }
}

```



```

// If it's the first account, make it default regardless of user input
// If not, use the user's preference
const shouldBeDefault =
  existingAccounts.length === 0 ? true : data.isDefault;

// If this account should be default, unset other default accounts
if (shouldBeDefault) {
  await db.account.updateMany({
    where: { userId: user.id, isDefault: true },
    data: { isDefault: false },
  });
}
// Create new account
const account = await db.account.create({
  data: {
    ...data,
    balance: balanceFloat,
    userId: user.id,
    isDefault: shouldBeDefault, // Override the isDefault based on our logic
  },
});
// Serialize the account before returning
const serializedAccount = serializeTransaction(account);

revalidatePath("/dashboard");
return { success: true, data: serializedAccount };
} catch (error) {
  throw new Error(error.message);
}
}

export async function getDashboardData() {
  const { userId } = await auth();
  if (!userId) throw new Error("Unauthorized");

  const user = await db.user.findUnique({
    where: { clerkUserId: userId },
  });

  if (!user) {
    throw new Error("User not found");
  }
  // Get all user transactions
  const transactions = await db.transaction.findMany({
    where: { userId: user.id },
    orderBy: { date: "desc" },
  });
  return transactions.map(serializeTransaction);
}

```

MY_FINANCE\actions\seed.js

```
"use server";
```

```
import { db } from "@lib/prisma";
import { subDays } from "date-fns";
```

```
const ACCOUNT_ID = "a7d9cf55-265a-456e-9e41-2fcddf59d4d8";
const USER_ID = "a7ee4421-eaf7-4a3c-83fb-007323505ed9";
```

```
// Categories with their typical amount ranges
```

```
const CATEGORIES = {
  INCOME: [
    { name: "salary", range: [5000, 8000] },
    { name: "freelance", range: [1000, 3000] },
    { name: "investments", range: [500, 2000] },
    { name: "other-income", range: [100, 1000] },
  ],
  EXPENSE: [
    { name: "housing", range: [1000, 2000] },
    { name: "transportation", range: [100, 500] },
    { name: "groceries", range: [200, 600] },
    { name: "utilities", range: [100, 300] },
    { name: "entertainment", range: [50, 200] },
    { name: "food", range: [50, 150] },
    { name: "shopping", range: [100, 500] },
    { name: "healthcare", range: [100, 1000] },
    { name: "education", range: [200, 1000] },
    { name: "travel", range: [500, 2000] },
  ],
};
```

```
// Helper to generate random amount within a range
```

```
function getRandomAmount(min, max) {
  return Number((Math.random() * (max - min) + min).toFixed(2));
}
```

```
// Helper to get random category with amount
```

```
function getRandomCategory(type) {
  const categories = CATEGORIES[type];
  const category = categories[Math.floor(Math.random() * categories.length)];
  const amount = getRandomAmount(category.range[0], category.range[1]);
  return { category: category.name, amount };
}
```

```
export async function seedTransactions() {
```

```
  try {
    // Generate 90 days of transactions
    const transactions = [];
    let totalBalance = 0;
```

```
    for (let i = 90; i >= 0; i--) {
```

```

const date = subDays(new Date(), i);

// Generate 1-3 transactions per day
const transactionsPerDay = Math.floor(Math.random() * 3) + 1;

for (let j = 0; j < transactionsPerDay; j++) {
  // 40% chance of income, 60% chance of expense
  const type = Math.random() < 0.4 ? "INCOME" : "EXPENSE";
  const { category, amount } = getRandomCategory(type);

  const transaction = {
    id: crypto.randomUUID(),
    type,
    amount,
    description: `${
      type === "INCOME" ? "Received" : "Paid for"
    } ${category}`,
    date,
    category,
    status: "COMPLETED",
    userId: USER_ID,
    accountId: ACCOUNT_ID,
    createdAt: date,
    updatedAt: date,
  };

  totalBalance += type === "INCOME" ? amount : -amount;
  transactions.push(transaction);
}

// Insert transactions in batches and update account balance
await db.$transaction(async (tx) => {
  // Clear existing transactions
  await tx.transaction.deleteMany({
    where: { accountId: ACCOUNT_ID },
  });

  // Insert new transactions
  await tx.transaction.createMany({
    data: transactions,
  });

  // Update account balance
  await tx.account.update({
    where: { id: ACCOUNT_ID },
    data: { balance: totalBalance },
  });
});

return {
  success: true,

```

```

    message: `Created ${transactions.length} transactions`,
  };
} catch (error) {
  console.error("Error seeding transactions:", error);
  return { success: false, error: error.message };
}
}

```

MY_FINANCE\actions\send-email.js

```

"use server";

import { Resend } from "resend";

export async function sendEmail({ to, subject, react }) {
  const resend = new Resend(process.env.RESEND_API_KEY || "");

  try {
    const data = await resend.emails.send({
      from: "My Finance <onboarding@resend.dev>",
      to,
      subject,
      react,
    });

    return { success: true, data };
  } catch (error) {
    console.error("Failed to send email:", error);
    return { success: false, error };
  }
}

```

MY_FINANCE\actions\transaction.js

```

"use server";

import { auth } from "@clerk/nextjs/server";
import { db } from "@lib/prisma";
import { revalidatePath } from "next/cache";
import { GoogleGenerativeAI } from "@google/generative-ai";
import aj from "@lib/arcjet";
import { request } from "@arcjet/next";

const genAI = new GoogleGenerativeAI(process.env.GEMINI_API_KEY);

const serializeAmount = (obj) => ({
  ...obj,
  amount: obj.amount.toNumber(),
});

// Create Transaction

```

```

export async function createTransaction(data) {
  try {
    const { userId } = await auth();
    if (!userId) throw new Error("Unauthorized");

    // Get request data for ArcJet
    const req = await request();

    // Check rate limit
    const decision = await aj.protect(req, {
      userId,
      requested: 1, // Specify how many tokens to consume
    });

    if (decision.isDenied()) {
      if (decision.reason.isRateLimit()) {
        const { remaining, reset } = decision.reason;
        console.error({
          code: "RATE_LIMIT_EXCEEDED",
          details: {
            remaining,
            resetInSeconds: reset,
          },
        });

        throw new Error("Too many requests. Please try again later.");
      }

      throw new Error("Request blocked");
    }

    const user = await db.user.findUnique({
      where: { clerkUserId: userId },
    });

    if (!user) {
      throw new Error("User not found");
    }

    const account = await db.account.findUnique({
      where: {
        id: data.accountId,
        userId: user.id,
      },
    });

    if (!account) {
      throw new Error("Account not found");
    }

    // Calculate new balance
    const balanceChange = data.type === "EXPENSE" ? -data.amount : data.amount;

```

```

const newBalance = account.balance.toNumber() + balanceChange;

// Create transaction and update account balance
const transaction = await db.$transaction(async (tx) => {
  const newTransaction = await tx.transaction.create({
    data: {
      ...data,
      userId: user.id,
      nextRecurringDate:
        data.isRecurring && data.recurringInterval
          ? calculateNextRecurringDate(data.date, data.recurringInterval)
          : null,
    },
  });

  await tx.account.update({
    where: { id: data.accountId },
    data: { balance: newBalance },
  });

  return newTransaction;
});

revalidatePath("/dashboard");
revalidatePath(`/account/${transaction.accountId}`);

return { success: true, data: serializeAmount(transaction) };
} catch (error) {
  throw new Error(error.message);
}
}

export async function getTransaction(id) {
  const { userId } = await auth();
  if (!userId) throw new Error("Unauthorized");

  const user = await db.user.findUnique({
    where: { clerkUserId: userId },
  });

  if (!user) throw new Error("User not found");

  const transaction = await db.transaction.findUnique({
    where: {
      id,
      userId: user.id,
    },
  });

  if (!transaction) throw new Error("Transaction not found");

  return serializeAmount(transaction);
}

```

```

}

export async function updateTransaction(id, data) {
  try {
    const { userId } = await auth();
    if (!userId) throw new Error("Unauthorized");

    const user = await db.user.findUnique({
      where: { clerkUserId: userId },
    });

    if (!user) throw new Error("User not found");

    // Get original transaction to calculate balance change
    const originalTransaction = await db.transaction.findUnique({
      where: {
        id,
        userId: user.id,
      },
      include: {
        account: true,
      },
    });

    if (!originalTransaction) throw new Error("Transaction not found");

    // Calculate balance changes
    const oldBalanceChange =
      originalTransaction.type === "EXPENSE"
        ? -originalTransaction.amount.toNumber()
        : originalTransaction.amount.toNumber();

    const newBalanceChange =
      data.type === "EXPENSE" ? -data.amount : data.amount;

    const netBalanceChange = newBalanceChange - oldBalanceChange;

    // Update transaction and account balance in a transaction
    const transaction = await db.$transaction(async (tx) => {
      const updated = await tx.transaction.update({
        where: {
          id,
          userId: user.id,
        },
        data: {
          ...data,
          nextRecurringDate:
            data.isRecurring && data.recurringInterval
              ? calculateNextRecurringDate(data.date, data.recurringInterval)
              : null,
        },
      });
    });
  }
}

```

```

// Update account balance
await tx.account.update({
  where: { id: data.accountId },
  data: {
    balance: {
      increment: netBalanceChange,
    },
  },
});

return updated;
});

revalidatePath("/dashboard");
revalidatePath(`/account/${data.accountId}`);

return { success: true, data: serializeAmount(transaction) };
} catch (error) {
  throw new Error(error.message);
}
}

// Get User Transactions
export async function getUserTransactions(query = {}) {
  try {
    const { userId } = await auth();
    if (!userId) throw new Error("Unauthorized");

    const user = await db.user.findUnique({
      where: { clerkUserId: userId },
    });

    if (!user) {
      throw new Error("User not found");
    }

    const transactions = await db.transaction.findMany({
      where: {
        userId: user.id,
        ...query,
      },
      include: {
        account: true,
      },
      orderBy: {
        date: "desc",
      },
    });

    return { success: true, data: transactions };
  } catch (error) {

```



```

    throw new Error(error.message);
  }
}

// Scan Receipt
export async function scanReceipt(file) {
  try {
    const model = genAI.getGenerativeModel({ model: "gemini-1.5-flash" });

    // Convert File to ArrayBuffer
    const arrayBuffer = await file.arrayBuffer();
    // Convert ArrayBuffer to Base64
    const base64String = Buffer.from(arrayBuffer).toString("base64");

    const prompt = `
      Analyze this receipt image and extract the following information in JSON format:
      - Total amount (just the number)
      - Date (in ISO format)
      - Description or items purchased (brief summary)
      - Merchant/store name
      - Suggested category (one of:
housing,transportation,groceries,utilities,entertainment,food,shopping,healthcare,education,personal,travel,insurance,gifts,bills,other-expense )

      Only respond with valid JSON in this exact format:
      {
        "amount": number,
        "date": "ISO date string",
        "description": "string",
        "merchantName": "string",
        "category": "string"
      }

      If its not a receipt, return an empty object
    `;

    const result = await model.generateContent([
      {
        inlineData: {
          data: base64String,
          mimeType: file.type,
        },
      },
      {
        prompt,
      }
    ]);

    const response = await result.response;
    const text = response.text();
    const cleanedText = text.replace(/^``(?:json)?\n?/g, "").trim();

    try {
      const data = JSON.parse(cleanedText);

```

```

return {
  amount: parseFloat(data.amount),
  date: new Date(data.date),
  description: data.description,
  category: data.category,
  merchantName: data.merchantName,
};
} catch (parseError) {
  console.error("Error parsing JSON response:", parseError);
  throw new Error("Invalid response format from Gemini");
}
} catch (error) {
  console.error("Error scanning receipt:", error);
  throw new Error("Failed to scan receipt");
}
}

// Helper function to calculate next recurring date
function calculateNextRecurringDate(startDate, interval) {
  const date = new Date(startDate);

  switch (interval) {
    case "DAILY":
      date.setDate(date.getDate() + 1);
      break;
    case "WEEKLY":
      date.setDate(date.getDate() + 7);
      break;
    case "MONTHLY":
      date.setMonth(date.getMonth() + 1);
      break;
    case "YEARLY":
      date.setFullYear(date.getFullYear() + 1);
      break;
  }

  return date;
}

```

MY_FINANCE\app\auth\sign-in\[[...sign-in]]\page.jsx

```
import { SignIn } from "@clerk/nextjs";
```

```
export default function Page() {
  return <SignIn />;
}
```

MY_FINANCE\app\auth\sign-up\[[...sign-up]]\page.jsx

```
import { SignUp } from "@clerk/nextjs";
```

```
export default function Page() {
  return <SignUp />;
}
```

MY_FINANCE\app\auth\layout.js

```
const AuthLayout = ({ children }) => {
  return <div className="flex justify-center pt-40">{children}</div>;
};
```

```
export default AuthLayout;
```

MY_FINANCE\app\main\account_components\account-chart.jsx

```
"use client";
```

```
import { useState, useMemo } from "react";
```

```
import {
  BarChart,
  Bar,
  XAxis,
  YAxis,
  CartesianGrid,
  Tooltip,
  Legend,
  ResponsiveContainer,
} from "recharts";
import { format, subDays, startOfDay, endOfDay } from "date-fns";
import { Card, CardContent, CardHeader, CardTitle } from "@components/ui/card";
import {
  Select,
  SelectContent,
  SelectItem,
  SelectTrigger,
```

```

    SelectValue,
  } from "@components/ui/select";

const DATE_RANGES = {
  "7D": { label: "Last 7 Days", days: 7 },
  "1M": { label: "Last Month", days: 30 },
  "3M": { label: "Last 3 Months", days: 90 },
  "6M": { label: "Last 6 Months", days: 180 },
  ALL: { label: "All Time", days: null },
};

export function AccountChart({ transactions }) {
  const [dateRange, setDateRange] = useState("1M");

  const filteredData = useMemo(() => {
    const range = DATE_RANGES[dateRange];
    const now = new Date();
    const startDate = range.days
      ? startOfDay(subDays(now, range.days))
      : startOfDay(new Date(0));

    // Filter transactions within date range
    const filtered = transactions.filter(
      (t) => new Date(t.date) >= startDate && new Date(t.date) <= endOfDay(now)
    );

    // Group transactions by date
    const grouped = filtered.reduce((acc, transaction) => {
      const date = format(new Date(transaction.date), "MMM dd");
      if (!acc[date]) {
        acc[date] = { date, income: 0, expense: 0 };
      }
      if (transaction.type === "INCOME") {
        acc[date].income += transaction.amount;
      } else {
        acc[date].expense += transaction.amount;
      }
      return acc;
    }, {});

    // Convert to array and sort by date
    return Object.values(grouped).sort(
      (a, b) => new Date(a.date) - new Date(b.date)
    );
  }, [transactions, dateRange]);

  // Calculate totals for the selected period
  const totals = useMemo(() => {
    return filteredData.reduce(
      (acc, day) => ({
        income: acc.income + day.income,
        expense: acc.expense + day.expense,

```

```

    }),
    { income: 0, expense: 0 }
  );
}, [filteredData]);

```

```

return (
  <Card>
    <CardHeader className="flex flex-row items-center justify-between space-y-0 pb-7">
      <CardTitle className="text-base font-normal">
        Transaction Overview
      </CardTitle>
      <Select defaultValue={dateRange} onValueChange={setDateRange}>
        <SelectTrigger className="w-[140px]">
          <SelectValue placeholder="Select range" />
        </SelectTrigger>
        <SelectContent>
          {Object.entries(DATE_RANGES).map(([key, { label }]) => (
            <SelectItem key={key} value={key}>
              {label}
            </SelectItem>
          ))}
        </SelectContent>
      </Select>
    </CardHeader>
    <CardContent>
      <div className="flex justify-around mb-6 text-sm">
        <div className="text-center">
          <p className="text-muted-foreground">Total Income</p>
          <p className="text-lg font-bold text-green-500">
            {totals.income.toFixed(2)}
          </p>
        </div>
        <div className="text-center">
          <p className="text-muted-foreground">Total Expenses</p>
          <p className="text-lg font-bold text-red-500">
            ₹{totals.expense.toFixed(2)}
          </p>
        </div>
        <div className="text-center">
          <p className="text-muted-foreground">Net</p>
          <p
            className={`text-lg font-bold ₹{
              totals.income - totals.expense >= 0
                ? "text-green-500"
                : "text-red-500"
            }`}
          >
            ₹{(totals.income - totals.expense).toFixed(2)}
          </p>
        </div>
      </div>
      <div className="h-[300px]">

```

```

<ResponsiveContainer width="100%" height="100%">
  <BarChart
    data={filteredData}
    margin={{ top: 10, right: 10, left: 10, bottom: 0 }}
  >
    <CartesianGrid strokeDasharray="3 3" vertical={false} />
    <XAxis
      dataKey="date"
      fontSize={12}
      tickLine={false}
      axisLine={false}
    />
    <YAxis
      fontSize={12}
      tickLine={false}
      axisLine={false}
      tickFormatter={(value) => `₹${value}`}
    />
    <Tooltip
      formatter={(value) => [ `₹${value}`, undefined]}
      contentStyle={{
        backgroundColor: "hsl(var(--popover))",
        border: "1px solid hsl(var(--border))",
        borderRadius: "var(--radius)",
      }}
    />
    <Legend />
    <Bar
      dataKey="income"
      name="Income"
      fill="#22c55e"
      radius={[4, 4, 0, 0]}
    />
    <Bar
      dataKey="expense"
      name="Expense"
      fill="#ef4444"
      radius={[4, 4, 0, 0]}
    />
  </BarChart>
</ResponsiveContainer>
</div>
</CardContent>
</Card>
);
}

```

MY_FINANCE\app\main\account_components\no-pagination-transaction-table.jsx

```

"use client";

import { useState, useEffect, useMemo } from "react";
import {
  ChevronDown,
  ChevronUp,
  MoreHorizontal,
  Trash,
  Search,
  X,
  RefreshCw,
  Clock,
} from "lucide-react";
import { format } from "date-fns";
import { toast } from "sonner";

import {
  Table,
  TableBody,
  TableCell,
  TableHead,
  TableHeader,
  TableRow,
} from "@/components/ui/table";
import { Input } from "@/components/ui/input";
import {
  Select,
  SelectContent,
  SelectItem,
  SelectTrigger,
  SelectValue,
} from "@/components/ui/select";
import {
  DropdownMenu,
  DropdownMenuContent,
  DropdownMenuItem,
  DropdownMenuTrigger,
  DropdownMenuSeparator,
} from "@/components/ui/dropdown-menu";
import { Checkbox } from "@/components/ui/checkbox";
import { Button } from "@/components/ui/button";
import {
  Tooltip,
  TooltipContent,
  TooltipProvider,
  TooltipTrigger,
} from "@/components/ui/tooltip";

```

```

import { Badge } from "@components/ui/badge";
import { cn } from "@lib/utls";
import { categoryColors } from "@data/categories";
import { bulkDeleteTransactions } from "@actions/account";
import useFetch from "@hooks/use-fetch";
import { BarLoader } from "react-spinners";
import { useRouter } from "next/navigation";

const RECURRING_INTERVALS = {
  DAILY: "Daily",
  WEEKLY: "Weekly",
  MONTHLY: "Monthly",
  YEARLY: "Yearly",
};

export function NoPaginationTransactionTable({ transactions }) {
  const [selectedIds, setSelectedIds] = useState([]);
  const [sortConfig, setSortConfig] = useState({
    field: "date",
    direction: "desc",
  });
  const [searchTerm, setSearchTerm] = useState("");
  const [typeFilter, setTypeFilter] = useState("");
  const [recurringFilter, setRecurringFilter] = useState("");
  const router = useRouter();

  // Memoized filtered and sorted transactions
  const filteredAndSortedTransactions = useMemo(() => {
    let result = [...transactions];

    // Apply search filter
    if (searchTerm) {
      const searchLower = searchTerm.toLowerCase();
      result = result.filter((transaction) =>
        transaction.description?.toLowerCase().includes(searchLower)
      );
    }

    // Apply type filter
    if (typeFilter) {
      result = result.filter((transaction) => transaction.type === typeFilter);
    }

    // Apply recurring filter
    if (recurringFilter) {
      result = result.filter((transaction) => {
        if (recurringFilter === "recurring") return transaction.isRecurring;
        return !transaction.isRecurring;
      });
    }

    // Apply sorting

```



```

result.sort((a, b) => {
  let comparison = 0;

  switch (sortConfig.field) {
    case "date":
      comparison = new Date(a.date) - new Date(b.date);
      break;
    case "amount":
      comparison = a.amount - b.amount;
      break;
    case "category":
      comparison = a.category.localeCompare(b.category);
      break;
    default:
      comparison = 0;
  }

  return sortConfig.direction === "asc" ? comparison : -comparison;
});

return result;
}, [transactions, searchTerm, typeFilter, recurringFilter, sortConfig]);

const handleSort = (field) => {
  setSortConfig((current) => ({
    field,
    direction:
      current.field === field && current.direction === "asc" ? "desc" : "asc",
  }));
};

const handleSelect = (id) => {
  setSelectedIds((current) =>
    current.includes(id)
      ? current.filter((item) => item !== id)
      : [...current, id]
  );
};

const handleSelectAll = () => {
  setSelectedIds((current) =>
    current.length === filteredAndSortedTransactions.length
      ? []
      : filteredAndSortedTransactions.map((t) => t.id)
  );
};

const {
  loading: deleteLoading,
  fn: deleteFn,
  data: deleted,
} = useFetch(bulkDeleteTransactions);

```

```

const handleBulkDelete = async () => {
  if (
    !window.confirm(
      `Are you sure you want to delete ${selectedIds.length} transactions?`
    )
  )
    return;

  deleteFn(selectedIds);
};

useEffect(() => {
  if (deleted && !deleteLoading) {
    toast.error("Transactions deleted successfully");
  }
}, [deleted, deleteLoading]);

const handleClearFilters = () => {
  setSearchTerm("");
  setTypeFilter("");
  setRecurringFilter("");
  setSelectedIds([]);
};

return (
  <div className="space-y-4">
    {deleteLoading && (
      <BarLoader className="mt-4" width={"100%"} color="#9333ea" />
    )}
    { /* Filters */ }
    <div className="flex flex-col sm:flex-row gap-4">
      <div className="relative flex-1">
        <Search className="absolute left-2 top-2.5 h-4 w-4 text-muted-foreground" />
        <Input
          placeholder="Search transactions..."
          value={searchTerm}
          onChange={(e) => setSearchTerm(e.target.value)}
          className="pl-8"
        />
      </div>
      <div className="flex gap-2">
        <Select value={typeFilter} onValueChange={setTypeFilter}>
          <SelectTrigger>
            <SelectValue placeholder="All Types" />
          </SelectTrigger>
          <SelectContent>
            <SelectItem value="INCOME">Income</SelectItem>
            <SelectItem value="EXPENSE">Expense</SelectItem>
          </SelectContent>
        </Select>
      </div>
    </div>
  )
);

```

```

<Select
  value={recurringFilter}
  onValueChange={(value) => {
    setRecurringFilter(value);
  }}
>
  <SelectTrigger className="w-[130px]">
    <SelectValue placeholder="All Transactions" />
  </SelectTrigger>
  <SelectContent>
    <SelectItem value="recurring">Recurring Only</SelectItem>
    <SelectItem value="non-recurring">Non-recurring Only</SelectItem>
  </SelectContent>
</Select>

{/* Bulk Actions */}
{selectedIds.length > 0 && (
  <div className="flex items-center gap-2">
    <Button
      variant="destructive"
      size="sm"
      onClick={handleBulkDelete}
    >
      <Trash className="h-4 w-4 mr-2" />
      Delete Selected ({selectedIds.length})
    </Button>
  </div>
)}

{(searchTerm || typeFilter || recurringFilter) && (
  <Button
    variant="outline"
    size="icon"
    onClick={handleClearFilters}
    title="Clear filters"
  >
    <X className="h-4 w-5" />
  </Button>
)}
</div>
</div>

{/* Transactions Table */}
<div className="rounded-md border">
  <Table>
    <TableHeader>
      <TableRow>
        <TableHead className="w-[50px]">
          <Checkbox
            checked={
              selectedIds.length ===
                filteredAndSortedTransactions.length &&

```

```

        filteredAndSortedTransactions.length > 0
      }
      onCheckedChange={handleSelectAll}
    />
  </TableHead>
  <TableHead
    className="cursor-pointer"
    onClick={() => handleSort("date")}
  >
    <div className="flex items-center">
      Date
      {sortConfig.field === "date" &&
        (sortConfig.direction === "asc" ? (
          <ChevronUp className="ml-1 h-4 w-4" />
        ) : (
          <ChevronDown className="ml-1 h-4 w-4" />
        ))}
    </div>
  </TableHead>
  <TableHead>Description</TableHead>
  <TableHead
    className="cursor-pointer"
    onClick={() => handleSort("category")}
  >
    <div className="flex items-center">
      Category
      {sortConfig.field === "category" &&
        (sortConfig.direction === "asc" ? (
          <ChevronUp className="ml-1 h-4 w-4" />
        ) : (
          <ChevronDown className="ml-1 h-4 w-4" />
        ))}
    </div>
  </TableHead>
  <TableHead
    className="cursor-pointer text-right"
    onClick={() => handleSort("amount")}
  >
    <div className="flex items-center justify-end">
      Amount
      {sortConfig.field === "amount" &&
        (sortConfig.direction === "asc" ? (
          <ChevronUp className="ml-1 h-4 w-4" />
        ) : (
          <ChevronDown className="ml-1 h-4 w-4" />
        ))}
    </div>
  </TableHead>
  <TableHead>Recurring</TableHead>
  <TableHead className="w-[50px]" />
</TableRow>
</TableHeader>

```

```

<TableBody>
  {filteredAndSortedTransactions.length === 0 ? (
    <TableRow>
      <TableCell>
        colSpan={7}
        className="text-center text-muted-foreground"
      >
        No transactions found
      </TableCell>
    </TableRow>
  ) : (
    filteredAndSortedTransactions.map((transaction) => (
      <TableRow key={transaction.id}>
        <TableCell>
          <Checkbox>
            checked={selectedIds.includes(transaction.id)}
            onChange={() => handleSelect(transaction.id)}
          </>
        </TableCell>
        <TableCell>
          {format(new Date(transaction.date), "PP")}
        </TableCell>
        <TableCell>{transaction.description}</TableCell>
        <TableCell className="capitalize">
          <span>
            style={{
              background: categoryColors[transaction.category],
            }}
            className="px-2 py-1 rounded text-white text-sm"
          >
            {transaction.category}
          </span>
        </TableCell>
        <TableCell>
          className={cn(
            "text-right font-medium",
            transaction.type === "EXPENSE"
              ? "text-red-500"
              : "text-green-500"
          )}
        >
          {transaction.type === "EXPENSE" ? "-" : "+"} ₹
          {transaction.amount.toFixed(2)}
        </TableCell>
        <TableCell>
          {transaction.isRecurring ? (
            <TooltipProvider>
              <Tooltip>
                <TooltipTrigger>
                  <Badge>
                    variant="secondary"
                    className="gap-1 bg-purple-100 text-purple-700 hover:bg-purple-200"

```

```

>
  <RefreshCw className="h-3 w-3" />
  {
    RECURRING_INTERVALS[
      transaction.recurringInterval
    ]
  }
  </Badge>
</TooltipTrigger>
<TooltipContent>
  <div className="text-sm">
    <div className="font-medium">Next Date:</div>
    <div>
      {format(
        new Date(transaction.nextRecurringDate),
        "PPP"
      )}
    </div>
  </div>
</TooltipContent>
</Tooltip>
</TooltipProvider>
): (
  <Badge variant="outline" className="gap-1">
    <Clock className="h-3 w-3" />
    One-time
  </Badge>
)
</TableCell>
<TableCell>
  <DropdownMenu>
    <DropdownMenuTrigger asChild>
      <Button variant="ghost" className="h-8 w-8 p-0">
        <MoreHorizontal className="h-4 w-4" />
      </Button>
    </DropdownMenuTrigger>
    <DropdownMenuContent align="end">
      <DropdownMenuItem
        onClick={() =>
          router.push(
            `/transaction/create?edit=${transaction.id}`
          )
        }
      />
    </DropdownMenuItem>
    <DropdownMenuSeparator />
    <DropdownMenuItem
      className="text-destructive"
      onClick={() => deleteFn([transaction.id])}
    />
    </DropdownMenuItem>
  </DropdownMenu>
  Edit
  </DropdownMenuItem>
  <DropdownMenuSeparator />
  <DropdownMenuItem
    className="text-destructive"
    onClick={() => deleteFn([transaction.id])}
  />
  </DropdownMenuItem>
  Delete

```

```

        </DropdownMenuItem>
        </DropdownMenuContent>
      </DropdownMenu>
    </TableCell>
  </TableRow>
)
})
</TableBody>
</Table>
</div>
</div>
);
}

```

MY_FINANCE\app\main\account_components\transaction-table.jsx

```

"use client";

import { useState, useEffect, useMemo } from "react";
import {
  ChevronDown,
  ChevronUp,
  MoreHorizontal,
  Trash,
  Search,
  X,
  ChevronLeft,
  ChevronRight,
  RefreshCw,
  Clock,
} from "lucide-react";
import { format } from "date-fns";
import { toast } from "sonner";

import {
  Table,
  TableBody,
  TableCell,
  TableHead,
  TableHeader,
  TableRow,
} from "@/components/ui/table";
import { Input } from "@/components/ui/input";
import {
  Select,
  SelectContent,
  SelectItem,
  SelectTrigger,

```

```

    SelectValue,
  } from "@components/ui/select";
import {
  DropdownMenu,
  DropdownMenuContent,
  DropdownMenuItem,
  DropdownMenuTrigger,
  DropdownMenuSeparator,
} from "@components/ui/dropdown-menu";
import { Checkbox } from "@components/ui/checkbox";
import { Button } from "@components/ui/button";
import {
  Tooltip,
  TooltipContent,
  TooltipProvider,
  TooltipTrigger,
} from "@components/ui/tooltip";
import { Badge } from "@components/ui/badge";
import { cn } from "@lib/Utils";
import { categoryColors } from "@data/categories";
import { bulkDeleteTransactions } from "@actions/account";
import useFetch from "@hooks/use-fetch";
import { BarLoader } from "react-spinners";
import { useRouter } from "next/navigation";

```

```
const ITEMS_PER_PAGE = 10;
```

```
const RECURRING_INTERVALS = {
  DAILY: "Daily",
  WEEKLY: "Weekly",
  MONTHLY: "Monthly",
  YEARLY: "Yearly",
};
```

```
export function TransactionTable({ transactions }) {
  const [selectedIds, setSelectedIds] = useState([]);
  const [sortConfig, setSortConfig] = useState({
    field: "date",
    direction: "desc",
  });
  const [searchTerm, setSearchTerm] = useState("");
  const [typeFilter, setTypeFilter] = useState("");
  const [recurringFilter, setRecurringFilter] = useState("");
  const [currentPage, setCurrentPage] = useState(1);
  const router = useRouter();

```

```
// Memoized filtered and sorted transactions
```

```
const filteredAndSortedTransactions = useMemo(() => {
  let result = [...transactions];
```

```
// Apply search filter
```

```
if (searchTerm) {
```



```

const searchLower = searchTerm.toLowerCase();
result = result.filter((transaction) =>
  transaction.description?.toLowerCase().includes(searchLower)
);
}

// Apply type filter
if (typeFilter) {
  result = result.filter((transaction) => transaction.type === typeFilter);
}

// Apply recurring filter
if (recurringFilter) {
  result = result.filter((transaction) => {
    if (recurringFilter === "recurring") return transaction.isRecurring;
    return !transaction.isRecurring;
  });
}

// Apply sorting
result.sort((a, b) => {
  let comparison = 0;

  switch (sortConfig.field) {
    case "date":
      comparison = new Date(a.date) - new Date(b.date);
      break;
    case "amount":
      comparison = a.amount - b.amount;
      break;
    case "category":
      comparison = a.category.localeCompare(b.category);
      break;
    default:
      comparison = 0;
  }

  return sortConfig.direction === "asc" ? comparison : -comparison;
});

return result;
}, [transactions, searchTerm, typeFilter, recurringFilter, sortConfig]);

// Pagination calculations
const totalPages = Math.ceil(
  filteredAndSortedTransactions.length / ITEMS_PER_PAGE
);
const paginatedTransactions = useMemo(() => {
  const startIndex = (currentPage - 1) * ITEMS_PER_PAGE;
  return filteredAndSortedTransactions.slice(
    startIndex,
    startIndex + ITEMS_PER_PAGE
  );
}, [transactions, searchTerm, typeFilter, recurringFilter, sortConfig, currentPage]);

```

```

    );
  }, [filteredAndSortedTransactions, currentPage]);

const handleSort = (field) => {
  setSortConfig((current) => ({
    field,
    direction:
      current.field === field && current.direction === "asc" ? "desc" : "asc",
  }));
};

const handleSelect = (id) => {
  setSelectedIds((current) =>
    current.includes(id)
      ? current.filter((item) => item !== id)
      : [...current, id]
  );
};

const handleSelectAll = () => {
  setSelectedIds((current) =>
    current.length === paginatedTransactions.length
      ? []
      : paginatedTransactions.map((t) => t.id)
  );
};

const {
  loading: deleteLoading,
  fn: deleteFn,
  data: deleted,
} = useFetch(bulkDeleteTransactions);

const handleBulkDelete = async () => {
  if (
    !window.confirm(
      `Are you sure you want to delete ${selectedIds.length} transactions?`
    )
  )
    return;

  deleteFn(selectedIds);
};

useEffect(() => {
  if (deleted && !deleteLoading) {
    toast.error("Transactions deleted successfully");
  }
}, [deleted, deleteLoading]);

const handleClearFilters = () => {
  setSearchTerm("");

```

```

setTypeFilter("");
setRecurringFilter("");
setCurrentPage(1);
};

const handlePageChange = (newPage) => {
  setCurrentPage(newPage);
  setSelectedIds([]); // Clear selections on page change
};

return (
  <div className="space-y-4">
    {deleteLoading && (
      <BarLoader className="mt-4" width={"100%"} color="#9333ea" />
    )}
    { /* Filters */ }
    <div className="flex flex-col sm:flex-row gap-4">
      <div className="relative flex-1">
        <Search className="absolute left-2 top-2.5 h-4 w-4 text-muted-foreground" />
        <Input
          placeholder="Search transactions..."
          value={searchTerm}
          onChange={(e) => {
            setSearchTerm(e.target.value);
            setCurrentPage(1);
          }}
          className="pl-8"
        />
      </div>
      <div className="flex gap-2">
        <Select
          value={typeFilter}
          onValueChange={(value) => {
            setTypeFilter(value);
            setCurrentPage(1);
          }}
        >
          <SelectTrigger className="w-[130px]">
            <SelectValue placeholder="All Types" />
          </SelectTrigger>
          <SelectContent>
            <SelectItem value="INCOME">Income</SelectItem>
            <SelectItem value="EXPENSE">Expense</SelectItem>
          </SelectContent>
        </Select>

        <Select
          value={recurringFilter}
          onValueChange={(value) => {
            setRecurringFilter(value);
            setCurrentPage(1);
          }}

```

```

>
<SelectTrigger className="w-[130px]">
  <SelectValue placeholder="All Transactions" />
</SelectTrigger>
<SelectContent>
  <SelectItem value="recurring">Recurring Only</SelectItem>
  <SelectItem value="non-recurring">Non-recurring Only</SelectItem>
</SelectContent>
</Select>

{/* Bulk Actions */}
{selectedIds.length > 0 && (
  <div className="flex items-center gap-2">
    <Button
      variant="destructive"
      size="sm"
      onClick={handleBulkDelete}
    >
      <Trash className="h-4 w-4 mr-2" />
      Delete Selected ({selectedIds.length})
    </Button>
  </div>
)}

{(searchTerm || typeFilter || recurringFilter) && (
  <Button
    variant="outline"
    size="icon"
    onClick={handleClearFilters}
    title="Clear filters"
  >
    <X className="h-4 w-5" />
  </Button>
)}
</div>
</div>

{/* Transactions Table */}
<div className="rounded-md border">
  <Table>
    <TableHeader>
      <TableRow>
        <TableHead className="w-[50px]">
          <Checkbox
            checked={
              selectedIds.length === paginatedTransactions.length &&
              paginatedTransactions.length > 0
            }
            onChange={handleSelectAll}
          />
        </TableHead>
        <TableHead>

```

```

      className="cursor-pointer"
      onClick={() => handleSort("date")}
    >
    <div className="flex items-center">
      Date
      {sortConfig.field === "date" &&
        (sortConfig.direction === "asc" ? (
          <ChevronUp className="ml-1 h-4 w-4" />
        ) : (
          <ChevronDown className="ml-1 h-4 w-4" />
        ))}
    </div>
  </TableHead>
<TableHead>Description</TableHead>
<TableHead
  className="cursor-pointer"
  onClick={() => handleSort("category")}
>
  <div className="flex items-center">
    Category
    {sortConfig.field === "category" &&
      (sortConfig.direction === "asc" ? (
        <ChevronUp className="ml-1 h-4 w-4" />
      ) : (
        <ChevronDown className="ml-1 h-4 w-4" />
      ))}
  </div>
</TableHead>
<TableHead
  className="cursor-pointer text-right"
  onClick={() => handleSort("amount")}
>
  <div className="flex items-center justify-end">
    Amount
    {sortConfig.field === "amount" &&
      (sortConfig.direction === "asc" ? (
        <ChevronUp className="ml-1 h-4 w-4" />
      ) : (
        <ChevronDown className="ml-1 h-4 w-4" />
      ))}
  </div>
</TableHead>
<TableHead>Recurring</TableHead>
<TableHead className="w-[50px]" />
</TableRow>
</TableHeader>
<TableBody>
  {paginatedTransactions.length === 0 ? (
    <TableRow>
      <TableCell
        colSpan={7}
        className="text-center text-muted-foreground"

```

```

    >
    No transactions found
  </TableCell>
</TableRow>
): (
  paginatedTransactions.map((transaction) => (
    <TableRow key={transaction.id}>
      <TableCell>
        <Checkbox
          checked={selectedIds.includes(transaction.id)}
          onChange={() => handleSelect(transaction.id)}
        />
      </TableCell>
      <TableCell>
        {format(new Date(transaction.date), "PP")}
      </TableCell>
      <TableCell>{transaction.description}</TableCell>
      <TableCell className="capitalize">
        <span
          style={{
            background: categoryColors[transaction.category],
          }}
          className="px-2 py-1 rounded text-white text-sm"
        >
          {transaction.category}
        </span>
      </TableCell>
      <TableCell
        className={cn(
          "text-right font-medium",
          transaction.type === "EXPENSE"
            ? "text-red-500"
            : "text-green-500"
        )}
      >
        {transaction.type === "EXPENSE" ? "-" : "+"} ₹
        {transaction.amount.toFixed(2)}
      </TableCell>
      <TableCell>
        {transaction.isRecurring ? (
          <TooltipProvider>
            <Tooltip>
              <TooltipTrigger>
                <Badge
                  variant="secondary"
                  className="gap-1 bg-purple-100 text-purple-700 hover:bg-purple-200"
                >
                  <RefreshCw className="h-3 w-3" />
                  {
                    RECURRING_INTERVALS[
                      transaction.recurringInterval
                    ]
                  }
                </Badge>
              </TooltipTrigger>
            </Tooltip>
          </TooltipProvider>
        ) : null}
      </TableCell>
    </TableRow>
  )
)

```

```

    }
    </Badge>
  </TooltipTrigger>
  <TooltipContent>
    <div className="text-sm">
      <div className="font-medium">Next Date:</div>
      <div>
        {format(
          new Date(transaction.nextRecurringDate),
          "PPP"
        )}
      </div>
    </div>
  </TooltipContent>
</Tooltip>
</TooltipProvider>
): (
  <Badge variant="outline" className="gap-1">
    <Clock className="h-3 w-3" />
    One-time
  </Badge>
)>
</TableCell>
<TableCell>
  <DropdownMenu>
    <DropdownMenuTrigger asChild>
      <Button variant="ghost" className="h-8 w-8 p-0">
        <MoreHorizontal className="h-4 w-4" />
      </Button>
    </DropdownMenuTrigger>
    <DropdownMenuContent align="end">
      <DropdownMenuItem
        onClick={() =>
          router.push(
            `/transaction/create?edit=${transaction.id}`
          )
        }
      >
        Edit
      </DropdownMenuItem>
      <DropdownMenuSeparator />
      <DropdownMenuItem
        className="text-destructive"
        onClick={() => deleteFn([transaction.id])}
      >
        Delete
      </DropdownMenuItem>
    </DropdownMenuContent>
  </DropdownMenu>
</TableCell>
</TableRow>
))

```

```

    })
  </TableBody>
</Table>
</div>

{/* Pagination */}
{totalPages > 1 && (
  <div className="flex items-center justify-center gap-2">
    <Button
      variant="outline"
      size="icon"
      onClick={() => handlePageChange(currentPage - 1)}
      disabled={currentPage === 1}
    >
      <ChevronLeft className="h-4 w-4" />
    </Button>
    <span className="text-sm">
      Page {currentPage} of {totalPages}
    </span>
    <Button
      variant="outline"
      size="icon"
      onClick={() => handlePageChange(currentPage + 1)}
      disabled={currentPage === totalPages}
    >
      <ChevronRight className="h-4 w-4" />
    </Button>
  </div>
  )}
</div>
);
}

```


MY_FINANCE\app\main\account[id]\page.jsx

```

import { Suspense } from "react";
import { getAccountWithTransactions } from "@actions/account";
import { BarLoader } from "react-spinners";
import { TransactionTable } from "../components/transaction-table";
import { notFound } from "next/navigation";
import { AccountChart } from "../components/account-chart";

export default async function AccountPage({ params }) {
  const { id } = params;
  const accountData = await getAccountWithTransactions(id);

  if (!accountData) {
    notFound();
  }

  const { transactions, ...account } = accountData;

  return (
    <div className="space-y-8 px-5">
      <div className="flex gap-4 items-end justify-between">
        <div>
          <h1 className="text-5xl sm:text-6xl font-bold tracking-tight gradient-title capitalize">
            {account.name}
          </h1>
          <p className="text-muted-foreground">
            {account.type.charAt(0) + account.type.slice(1).toLowerCase()}{" "}
            Account
          </p>
        </div>

        <div className="text-right pb-2">
          <div className="text-xl sm:text-2xl font-bold">
            ₹ {parseFloat(account.balance).toFixed(2)}
          </div>
          <p className="text-sm text-muted-foreground">
            {account._count.transactions} Transactions
          </p>
        </div>
      </div>

      { /* Chart Section */ }
      <Suspense
        fallback={<BarLoader className="mt-4" width={"100%"} color="#9333ea" />}
      >
        <AccountChart transactions={transactions} />
      </Suspense>

      { /* Transactions Table */ }
      <Suspense
        fallback={<BarLoader className="mt-4" width={"100%"} color="#9333ea" />}

```

```

    >
    <TransactionTable transactions={transactions} />
  </Suspense>
</div>
);
}

```

MY_FINANCE\app\main\dashboard_components\account-card.jsx

```

"use client";

import { ArrowUpRight, ArrowDownRight, CreditCard } from "lucide-react";
import { Switch } from "@/components/ui/switch";
//import { Badge } from "@/components/ui/badge";
import { useEffect } from "react";
import useFetch from "@/hooks/use-fetch";
import {
  Card,
  CardContent,
  CardFooter,
  CardHeader,
  CardTitle,
} from "@/components/ui/card";
import Link from "next/link";
import { updateDefaultAccount } from "@/actions/account";
import { toast } from "sonner";

export function AccountCard({ account }) {
  const { name, type, balance, id, isDefault } = account;

  const {
    loading: updateDefaultLoading,
    fn: updateDefaultFn,
    data: updatedAccount,
    error,
  } = useFetch(updateDefaultAccount);

  const handleDefaultChange = async (event) => {
    event.preventDefault(); // Prevent navigation

    if (isDefault) {
      toast.warning("You need atleast 1 default account");
      return; // Don't allow toggling off the default account
    }

    await updateDefaultFn(id);
  };

  useEffect(() => {
    if (updatedAccount?.success) {

```

```

    toast.success("Default account updated successfully");
  }
}, [updatedAccount]);

useEffect(() => {
  if (error) {
    toast.error(error.message || "Failed to update default account");
  }
}, [error]);

return (
  <Card className="hover:shadow-md transition-shadow group relative">
    <Link href={` /account/${id}`} >
      <CardHeader className="flex flex-row items-center justify-between space-y-0 pb-2">
        <CardTitle className="text-sm font-medium capitalize">
          {name}
        </CardTitle>
        <Switch
          checked={isDefault}
          onClick={handleDefaultChange}
          disabled={updateDefaultLoading}
        />
      </CardHeader>
      <CardContent>
        <div className="text-2xl font-bold">
          ₹{parseFloat(balance).toFixed(2)}
        </div>
        <p className="text-xs text-muted-foreground">
          {type.charAt(0) + type.slice(1).toLowerCase()} Account
        </p>
      </CardContent>
      <CardFooter className="flex justify-between text-sm text-muted-foreground">
        <div className="flex items-center">
          <ArrowUpRight className="mr-1 h-4 w-4 text-green-500" />
          Income
        </div>
        <div className="flex items-center">
          <ArrowDownRight className="mr-1 h-4 w-4 text-red-500" />
          Expense
        </div>
      </CardFooter>
    </Link>
  </Card>
);
}

```

MY_FINANCE\app\main\dashboard_components\budget-progress.jsx

```

"use client";

import { useState, useEffect } from "react";
import { Pencil, Check, X } from "lucide-react";
import useFetch from "@hooks/use-fetch";
import { toast } from "sonner";

import {
  Card,
  CardContent,
  CardDescription,
  CardHeader,
  CardTitle,
} from "@components/ui/card";
import { Progress } from "@components/ui/progress";
import { Button } from "@components/ui/button";
import { Input } from "@components/ui/input";
import { updateBudget } from "@actions/budget";

export function BudgetProgress({ initialBudget, currentExpenses }) {
  const [isEditing, setIsEditing] = useState(false);
  const [newBudget, setNewBudget] = useState(
    initialBudget?.amount?.toString() || ""
  );

  const {
    loading: isLoading,
    fn: updateBudgetFn,
    data: updatedBudget,
    error,
  } = useFetch(updateBudget);

  const percentUsed = initialBudget
    ? (currentExpenses / initialBudget.amount) * 100
    : 0;

  const handleUpdateBudget = async () => {
    const amount = parseFloat(newBudget);

    if (isNaN(amount) || amount <= 0) {
      toast.error("Please enter a valid amount");
      return;
    }

    await updateBudgetFn(amount);
  };

  const handleCancel = () => {

```

```

    setNewBudget(initialBudget?.amount?.toString() || "");
    setIsEditing(false);
  };

  useEffect(() => {
    if (updatedBudget?.success) {
      setIsEditing(false);
      toast.success("Budget updated successfully");
    }
  }, [updatedBudget]);

  useEffect(() => {
    if (error) {
      toast.error(error.message || "Failed to update budget");
    }
  }, [error]);

  return (
    <Card>
      <CardHeader className="flex flex-row items-center justify-between space-y-0 pb-2">
        <div className="flex-1">
          <CardTitle className="text-sm font-medium">
            Monthly Budget (Default Account)
          </CardTitle>
          <div className="flex items-center gap-2 mt-1">
            {isEditing ? (
              <div className="flex items-center gap-2">
                <Input
                  type="number"
                  value={newBudget}
                  onChange={(e) => setNewBudget(e.target.value)}
                  className="w-32"
                  placeholder="Enter amount"
                  autoFocus
                  disabled={isLoading}
                />
                <Button
                  variant="ghost"
                  size="icon"
                  onClick={handleUpdateBudget}
                  disabled={isLoading}
                />
                <Check className="h-4 w-4 text-green-500" />
              </div>
            ) : (
              <Button
                variant="ghost"
                size="icon"
                onClick={handleCancel}
                disabled={isLoading}
              />
              <X className="h-4 w-4 text-red-500" />
            )}
          </div>
        </div>
      </CardHeader>
    </Card>
  );

```

```

    </div>
  ) : (
    <
    <CardDescription>
      {initialBudget
        ? `₹${currentExpenses.toFixed(
          2
        )} of ₹${initialBudget.amount.toFixed(2)} spent`
        : "No budget set"}
    </CardDescription>
    <Button
      variant="ghost"
      size="icon"
      onClick={() => setIsEditing(true)}
      className="h-6 w-6"
    >
      <Pencil className="h-3 w-3" />
    </Button>
    </>
  )}
</div>
</div>
</CardHeader>
<CardContent>
  {initialBudget && (
    <div className="space-y-2">
      <Progress
        value={percentUsed}
        extraStyles={`$ {
          // add to Progress component
          percentUsed >= 90
            ? "bg-red-500"
            : percentUsed >= 75
            ? "bg-yellow-500"
            : "bg-green-500"
        }`}
      />
      <p className="text-xs text-muted-foreground text-right">
        {percentUsed.toFixed(1)}% used
      </p>
    </div>
  )}
</CardContent>
</Card>
);
}

```

MY_FINANCE\app\main\dashboard_components\transaction-overview.jsx

```

"use client";

import { useState } from "react";
import {
  PieChart,
  Pie,
  Cell,
  ResponsiveContainer,
  Tooltip,
  Legend,
} from "recharts";
import { format } from "date-fns";
import { ArrowUpRight, ArrowDownRight } from "lucide-react";

import {
  Select,
  SelectContent,
  SelectItem,
  SelectTrigger,
  SelectValue,
} from "@components/ui/select";
import { Card, CardContent, CardHeader, CardTitle } from "@components/ui/card";
import { cn } from "@lib/utlis";

const COLORS = [
  "#FF6B6B",
  "#4ECDC4",
  "#45B7D1",
  "#96CEB4",
  "#FFEEAD",
  "#D4A5A5",
  "#9FA8DA",
];

export function DashboardOverview({ accounts, transactions }) {
  const [selectedAccountId, setSelectedAccountId] = useState(
    accounts.find((a) => a.isDefault)?.id || accounts[0]?.id
  );

  // Filter transactions for selected account
  const accountTransactions = transactions.filter(
    (t) => t.accountId === selectedAccountId
  );

  // Get recent transactions (last 5)
  const recentTransactions = accountTransactions
    .sort((a, b) => new Date(b.date) - new Date(a.date))
    .slice(0, 5);

```

```

// Calculate expense breakdown for current month
const currentDate = new Date();
const currentMonthExpenses = accountTransactions.filter((t) => {
  const transactionDate = new Date(t.date);
  return (
    t.type === "EXPENSE" &&
    transactionDate.getMonth() === currentDate.getMonth() &&
    transactionDate.getFullYear() === currentDate.getFullYear()
  );
});

// Group expenses by category
const expensesByCategory = currentMonthExpenses.reduce((acc, transaction) => {
  const category = transaction.category;
  if (!acc[category]) {
    acc[category] = 0;
  }
  acc[category] += transaction.amount;
  return acc;
}, {});

// Format data for pie chart
const pieChartData = Object.entries(expensesByCategory).map(
  ([category, amount]) => ({
    name: category,
    value: amount,
  })
);

return (
  <div className="grid gap-4 md:grid-cols-2">
    { /* Recent Transactions Card */ }
    <Card>
      <CardHeader className="flex flex-row items-center justify-between space-y-0 pb-4">
        <CardTitle className="text-base font-normal">
          Recent Transactions
        </CardTitle>
        <Select
          value={selectedAccountId}
          onChange={setSelectedAccountId}
        >
          <SelectTrigger className="w-[140px]">
            <SelectValue placeholder="Select account" />
          </SelectTrigger>
          <SelectContent>
            {accounts.map((account) => (
              <SelectItem key={account.id} value={account.id}>
                {account.name}
              </SelectItem>
            ))}
          </SelectContent>
        </Card>
      </div>
    )
  )

```



```

</Select>
</CardHeader>
<CardContent>
  <div className="space-y-4">
    {recentTransactions.length === 0 ? (
      <p className="text-center text-muted-foreground py-4">
        No recent transactions
      </p>
    ) : (
      recentTransactions.map((transaction) => (
        <div
          key={transaction.id}
          className="flex items-center justify-between"
        >
          <div className="space-y-1">
            <p className="text-sm font-medium leading-none">
              {transaction.description || "Untitled Transaction"}
            </p>
            <p className="text-sm text-muted-foreground">
              {format(new Date(transaction.date), "PP")}
            </p>
          </div>
          <div className="flex items-center gap-2">
            <div
              className={cn(
                "flex items-center",
                transaction.type === "EXPENSE"
                  ? "text-red-500"
                  : "text-green-500"
              )}
            >
              {transaction.type === "EXPENSE" ? (
                <ArrowDownRight className="mr-1 h-4 w-4" />
              ) : (
                <ArrowUpRight className="mr-1 h-4 w-4" />
              )}
              ₹{transaction.amount.toFixed(2)}
            </div>
          </div>
        </div>
      )}
    </div>
  </div>
</CardContent>
</Card>

{/* Expense Breakdown Card */}
<Card>
  <CardHeader>
    <CardTitle className="text-base font-normal">
      Monthly Expense Breakdown
    </CardTitle>

```

```

</CardHeader>
<CardContent className="p-0 pb-5">
  {pieChartData.length === 0 ? (
    <p className="text-center text-muted-foreground py-4">
      No expenses this month
    </p>
  ) : (
    <div className="h-[300px]">
      <ResponsiveContainer width="100%" height="100%">
        <PieChart>
          <Pie
            data={pieChartData}
            cx="50%"
            cy="50%"
            outerRadius={80}
            fill="#8884d8"
            dataKey="value"
            label={({ name, value }) => `₹${name}: ₹${value.toFixed(2)} `}
          >
            {pieChartData.map((entry, index) => (
              <Cell
                key={`cell-${index}`}
                fill={COLORS[index % COLORS.length]}
              />
            ))}
          </Pie>
          <Tooltip
            formatter={(value) => `₹${value.toFixed(2)} `}
            contentStyle={{
              backgroundColor: "hsl(var(--popover))",
              border: "1px solid hsl(var(--border))",
              borderRadius: "var(--radius)",
            }}
          />
          <Legend />
        </PieChart>
      </ResponsiveContainer>
    </div>
  )}
</CardContent>
</Card>
</div>
);
}

```

MY_FINANCE\app\main\dashboard\layout.js

```

import DashboardPage from "../page";
import { BarLoader } from "react-spinners";
import { Suspense } from "react";

export default function Layout() {
  return (
    <div className="px-5">
      <div className="flex items-center justify-between mb-5">
        <h1 className="text-6xl font-bold tracking-tight gradient-title">
          Dashboard
        </h1>
      </div>
      <Suspense
        fallback={<BarLoader className="mt-4" width={"100%"} color="#9333ea" />}
      >
        <DashboardPage />
      </Suspense>
    </div>
  );
}

```

MY_FINANCE\app\main\dashboard\page.jsx

```

import { Suspense } from "react";
import { getUserAccounts } from "@actions/dashboard";
import { getDashboardData } from "@actions/dashboard";
import { getCurrentBudget } from "@actions/budget";
import { AccountCard } from "../components/account-card";
import { CreateAccountDrawer } from "@components/create-account-drawer";
import { BudgetProgress } from "../components/budget-progress";
import { Card, CardContent } from "@components/ui/card";
import { Plus } from "lucide-react";
import { DashboardOverview } from "../components/transaction-overview";

export default async function DashboardPage() {
  const [accounts, transactions] = await Promise.all([
    getUserAccounts(),
    getDashboardData(),
  ]);

  const defaultAccount = accounts?.find((account) => account.isDefault);

  // Get budget for default account
  let budgetData = null;
  if (defaultAccount) {

```

```

    budgetData = await getCurrentBudget(defaultAccount.id);
  }

  return (
    <div className="space-y-8">
      { /* Budget Progress */ }
      <BudgetProgress
        initialBudget={budgetData?.budget}
        currentExpenses={budgetData?.currentExpenses || 0}
      />

      { /* Dashboard Overview */ }
      <DashboardOverview
        accounts={accounts}
        transactions={transactions || []}
      />

      { /* Accounts Grid */ }
      <div className="grid gap-4 md:grid-cols-2 lg:grid-cols-3">
        <CreateAccountDrawer>
          <Card className="hover:shadow-md transition-shadow cursor-pointer border-dashed">
            <CardContent className="flex flex-col items-center justify-center text-muted-foreground h-full pt-5">
              <Plus className="h-10 w-10 mb-2" />
              <p className="text-sm font-medium">Add New Account</p>
            </CardContent>
          </Card>
        </CreateAccountDrawer>
        {accounts.length > 0 &&
          accounts?.map((account) => (
            <AccountCard key={account.id} account={account} />
          ))}
      </div>
    </div>
  );
}

```

MY_FINANCE\app\main\transaction_components\receipt-scanner.jsx

```

"use client";

import { useRef, useEffect } from "react";
import { Camera, Loader2 } from "lucide-react";
import { Button } from "@/components/ui/button";
import { toast } from "sonner";
import useFetch from "@/hooks/use-fetch";
import { scanReceipt } from "@/actions/transaction";

export function ReceiptScanner({ onScanComplete }) {
  const fileInputRef = useRef(null);

  const {
    loading: scanReceiptLoading,
    fn: scanReceiptFn,
    data: scannedData,
  } = useFetch(scanReceipt);

  const handleReceiptScan = async (file) => {
    if (file.size > 5 * 1024 * 1024) {
      toast.error("File size should be less than 5MB");
      return;
    }

    await scanReceiptFn(file);
  };

  const handledRef = useRef(false);
  useEffect(() => {
    if (scannedData && !scanReceiptLoading && !handledRef.current) {
      onScanComplete(scannedData);
      toast.success("Receipt scanned successfully");
      handledRef.current = true; // Prevent multiple calls
    }
  }, [scanReceiptLoading, scannedData, onScanComplete]);

  return (
    <div className="flex items-center gap-4">
      <input
        type="file"
        ref={fileInputRef}
        className="hidden"
        accept="image/*"
        capture="environment"
        onChange={(e) => {
          const file = e.target.files?.[0];
          if (file) handleReceiptScan(file);
        }}
      />

```

```

<Button
  type="button"
  variant="outline"
  className="w-full h-10 bg-gradient-to-br from-orange-500 via-pink-500 to-purple-500 animate-gradient
  hover:opacity-90 transition-opacity text-white hover:text-white"
  onClick={() => fileInputRef.current?.click()}
  disabled={scanReceiptLoading}
>
  {scanReceiptLoading ? (
    <
      <Loader2 className="mr-2 animate-spin" />
      <span>Scanning Receipt...</span>
    </>
  ) : (
    <
      <Camera className="mr-2" />
      <span>Scan Receipt with AI</span>
    </>
  )}
</Button>
</div>
);
}

```

MY_FINANCE\app\main\transaction_components\transaction-form.jsx

```

"use client";

import { useEffect } from "react";
import { useForm } from "react-hook-form";
import { zodResolver } from "@hookform/resolvers/zod";
import { CalendarIcon, Loader2 } from "lucide-react";
import { format } from "date-fns";
import { useRouter, useSearchParams } from "next/navigation";
import useFetch from "@/hooks/use-fetch";
import { toast } from "sonner";

import { Button } from "@/components/ui/button";
import { Input } from "@/components/ui/input";
import { Switch } from "@/components/ui/switch";
import {
  Select,
  SelectContent,
  SelectItem,
  SelectTrigger,
  SelectValue,
} from "@/components/ui/select";
import {

```

```

    Popover,
    PopoverContent,
    PopoverTrigger,
  } from "@components/ui/popover";
import { Calendar } from "@components/ui/calendar";
import { CreateAccountDrawer } from "@components/create-account-drawer";
import { cn } from "@lib/Utils";
import { createTransaction, updateTransaction } from "@actions/transaction";
import { transactionSchema } from "@app/lib/schema";
import { ReceiptScanner } from "../receipt-scanner";

```

```

export function AddTransactionForm({
  accounts,
  categories,
  editMode = false,
  initialData = null,
}) {
  const router = useRouter();
  const searchParams = useSearchParams();
  const editId = searchParams.get("edit");

  const {
    register,
    handleSubmit,
    formState: { errors },
    watch,
    setValue,
    getValues,
    reset,
  } = useForm({
    resolver: zodResolver(transactionSchema),
    defaultValues:
      editMode && initialData
        ? {
            type: initialData.type,
            amount: initialData.amount.toString(),
            description: initialData.description,
            accountId: initialData.accountId,
            category: initialData.category,
            date: new Date(initialData.date),
            isRecurring: initialData.isRecurring,
            ...(initialData.recurringInterval && {
              recurringInterval: initialData.recurringInterval,
            }),
          }
        : {
            type: "EXPENSE",
            amount: "",
            description: "",
            accountId: accounts.find((ac) => ac.isDefault)?.id,
            date: new Date(),
            isRecurring: false,

```

```

    },
  });

const {
  loading: transactionLoading,
  fn: transactionFn,
  data: transactionResult,
} = useFetch(editMode ? updateTransaction : createTransaction);

const onSubmit = (data) => {
  const formData = {
    ...data,
    amount: parseFloat(data.amount),
  };

  if (editMode) {
    transactionFn(editId, formData);
  } else {
    transactionFn(formData);
  }
};

const handleScanComplete = (scannedData) => {
  if (scannedData) {
    setValue("amount", scannedData.amount.toString());
    setValue("date", new Date(scannedData.date));
    if (scannedData.description) {
      setValue("description", scannedData.description);
    }
    if (scannedData.category) {
      setValue("category", scannedData.category);
    }
    toast.success("Receipt scanned successfully");
  }
};

useEffect(() => {
  if (transactionResult?.success && !transactionLoading) {
    toast.success(
      editMode
        ? "Transaction updated successfully"
        : "Transaction created successfully"
    );
    reset();
    router.push(`/account/${transactionResult.data.accountId}`);
  }
}, [transactionResult, transactionLoading, editMode, reset, router]);

const type = watch("type");
const isRecurring = watch("isRecurring");
const date = watch("date");

```



```

const filteredCategories = categories.filter(
  (category) => category.type === type
);

return (
  <form onSubmit={handleSubmit(onSubmit)} className="space-y-6">
    { /* Receipt Scanner - Only show in create mode */ }
    { !editMode && <ReceiptScanner onScanComplete={handleScanComplete} /> }

    { /* Type */ }
    <div className="space-y-2">
      <label className="text-sm font-medium">Type</label>
      <Select
        onChange={(value) => setValue("type", value)}
        defaultValue={type}
      >
        <SelectTrigger>
          <SelectValue placeholder="Select type" />
        </SelectTrigger>
        <SelectContent>
          <SelectItem value="EXPENSE">Expense</SelectItem>
          <SelectItem value="INCOME">Income</SelectItem>
        </SelectContent>
      </Select>
      {errors.type && (
        <p className="text-sm text-red-500">{errors.type.message}</p>
      )}
    </div>

    { /* Amount and Account */ }
    <div className="grid gap-6 md:grid-cols-2">
      <div className="space-y-2">
        <label className="text-sm font-medium">Amount</label>
        <Input
          type="number"
          step="0.01"
          placeholder="0.00"
          {...register("amount")}
        />
        {errors.amount && (
          <p className="text-sm text-red-500">{errors.amount.message}</p>
        )}
      </div>

      <div className="space-y-2">
        <label className="text-sm font-medium">Account</label>
        <Select
          onChange={(value) => setValue("accountId", value)}
          defaultValue={getValues("accountId")}
        >
          <SelectTrigger>
            <SelectValue placeholder="Select account" />

```

```

</SelectTrigger>
<SelectContent>
  {accounts.map((account) => (
    <SelectItem key={account.id} value={account.id}>
      {account.name} (₹ {parseFloat(account.balance).toFixed(2)})
    </SelectItem>
  ))}
  <CreateAccountDrawer>
    <Button
      variant="ghost"
      className="relative flex w-full cursor-default select-none items-center rounded-sm py-1.5 pl-8 pr-2 text-sm outline-none hover:bg-accent hover:text-accent-foreground"
    >
      Create Account
    </Button>
  </CreateAccountDrawer>
</SelectContent>
</Select>
{errors.accountId && (
  <p className="text-sm text-red-500">{errors.accountId.message}</p>
)}
</div>
</div>

```

```

{/* Category */}
<div className="space-y-2">
  <label className="text-sm font-medium">Category</label>
  <Select
    onValueChange={(value) => setValue("category", value)}
    defaultValue={getValues("category")}
  >
    <SelectTrigger>
      <SelectValue placeholder="Select category" />
    </SelectTrigger>
    <SelectContent>
      {filteredCategories.map((category) => (
        <SelectItem key={category.id} value={category.id}>
          {category.name}
        </SelectItem>
      ))}
    </SelectContent>
  </Select>
  {errors.category && (
    <p className="text-sm text-red-500">{errors.category.message}</p>
  )}
</div>

```

```

{/* Date */}
<div className="space-y-2">
  <label className="text-sm font-medium">Date</label>
  <Popover>
    <PopoverTrigger asChild>

```

```

<Button
  variant="outline"
  className={cn(
    "w-full pl-3 text-left font-normal",
    !date && "text-muted-foreground"
  )}
>
  {date ? format(date, "PPP") : <span>Pick a date</span>}
  <CalendarIcon className="ml-auto h-4 w-4 opacity-50" />
</Button>
</PopoverTrigger>
<PopoverContent className="w-auto p-0" align="start">
  <Calendar
    mode="single"
    selected={date}
    onSelect={(date) => setValue("date", date)}
    disabled={(date) =>
      date > new Date() || date < new Date("1900-01-01")}
  />
  initialFocus
</PopoverContent>
</Popover>
{errors.date && (
  <p className="text-sm text-red-500">{errors.date.message}</p>
)}
</div>

{/* Description */}
<div className="space-y-2">
  <label className="text-sm font-medium">Description</label>
  <Input placeholder="Enter description" {...register("description")} />
  {errors.description && (
    <p className="text-sm text-red-500">{errors.description.message}</p>
  )}
</div>

{/* Recurring Toggle */}
<div className="flex flex-row items-center justify-between rounded-lg border p-4">
  <div className="space-y-0.5">
    <label className="text-base font-medium">Recurring Transaction</label>
    <div className="text-sm text-muted-foreground">
      Set up a recurring schedule for this transaction
    </div>
  </div>
  <Switch
    checked={isRecurring}
    onChange={(checked) => setValue("isRecurring", checked)}
  />
</div>

```

```

{/* Recurring Interval */}

```

```

{isRecurring && (
  <div className="space-y-2">
    <label className="text-sm font-medium">Recurring Interval</label>
    <Select
      onChange={(value) => setValue("recurringInterval", value)}
      defaultValue={getValues("recurringInterval")}
    >
      <SelectTrigger>
        <SelectValue placeholder="Select interval" />
      </SelectTrigger>
      <SelectContent>
        <SelectItem value="DAILY">Daily</SelectItem>
        <SelectItem value="WEEKLY">Weekly</SelectItem>
        <SelectItem value="MONTHLY">Monthly</SelectItem>
        <SelectItem value="YEARLY">Yearly</SelectItem>
      </SelectContent>
    </Select>
    {errors.recurringInterval && (
      <p className="text-sm text-red-500">
        {errors.recurringInterval.message}
      </p>
    )}
  </div>
)}

{/* Actions */}
<div className="flex gap-4">
  <Button
    type="button"
    variant="outline"
    className="w-full"
    onClick={() => router.back()}
  >
    Cancel
  </Button>
  <Button type="submit" className="w-full" disabled={transactionLoading}>
    {transactionLoading ? (
      <div>
        <Loader2 className="mr-2 h-4 w-4 animate-spin" />
        {editMode ? "Updating..." : "Creating..."}
      </div>
    ) : editMode ? (
      "Update Transaction"
    ) : (
      "Create Transaction"
    )}
  </Button>
</div>
</form>
);
}

```

MY_FINANCE\app\main\transaction\create\page.jsx

```

import { getUserAccounts } from "@actions/dashboard";
import { defaultCategories } from "@data/categories";
import { AddTransactionForm } from "../components/transaction-form";
import { getTransaction } from "@actions/transaction";

export default async function AddTransactionPage({ searchParams }) {
  const accounts = await getUserAccounts();
  const editId = searchParams?.edit;

  let initialData = null;
  if (editId) {
    const transaction = await getTransaction(editId);
    initialData = transaction;
  }

  return (
    <div className="max-w-3xl mx-auto px-5">
      <div className="flex justify-center md:justify-normal mb-8">
        <h1 className="text-5xl gradient-title">{editId?"Edit":"Add"} Transaction</h1>
      </div>
      <AddTransactionForm
        accounts={accounts}
        categories={defaultCategories}
        editMode={!!editId}
        initialData={initialData}
      />
    </div>
  );
}

```

MY_FINANCE\app\main\layout.js

```

import React from "react";

const MainLayout = ({ children }) => {
  return <div className="container mx-auto my-32">{children}</div>;
};

export default MainLayout;

```

MY_FINANCE\app\api\inngest\route.js

```
import { serve } from "inngest/next";

import { inngest } from "@lib/inngest/client";
import {
  checkBudgetAlerts,
  generateMonthlyReports,
  processRecurringTransaction,
  triggerRecurringTransactions,
} from "@lib/inngest/function";

export const { GET, POST, PUT } = serve({
  client: inngest,
  functions: [
    processRecurringTransaction,
    triggerRecurringTransactions,
    generateMonthlyReports,
    checkBudgetAlerts,
  ],
});
```

MY_FINANCE\app\api\seed\route.js

```
import { seedTransactions } from "@actions/seed";

export async function GET() {
  const result = await seedTransactions();
  return Response.json(result);
}
```

MY_FINANCE\app\lib\schema.js

```

import { z } from "zod";

export const accountSchema = z.object({
  name: z.string().min(1, "Name is required"),
  type: z.enum(["CURRENT", "SAVINGS"]),
  balance: z.string().min(1, "Initial balance is required"),
  isDefault: z.boolean().default(false),
});

export const transactionSchema = z
  .object({
    type: z.enum(["INCOME", "EXPENSE"]),
    amount: z.string().min(1, "Amount is required"),
    description: z.string().optional(),
    date: z.date({ required_error: "Date is required" }),
    accountId: z.string().min(1, "Account is required"),
    category: z.string().min(1, "Category is required"),
    isRecurring: z.boolean().default(false),
    recurringInterval: z
      .enum(["DAILY", "WEEKLY", "MONTHLY", "YEARLY"])
      .optional(),
  })
  .superRefine((data, ctx) => {
    if (data.isRecurring && !data.recurringInterval) {
      ctx.addIssue({
        code: z.ZodIssueCode.custom,
        message: "Recurring interval is required for recurring transactions",
        path: ["recurringInterval"],
      });
    }
  });

```

MY_FINANCE\app\globals.css

```

@tailwind base;
@tailwind components;
@tailwind utilities;

body {
  font-family: Arial, Helvetica, sans-serif;
}

html {
  scroll-behavior: smooth;
}

@layer base {
  :root {
    --background: 260 15% 96%;
    --foreground: 0 0% 3.9%;
    --card: 0 0% 100%;
    --card-foreground: 0 0% 3.9%;
    --popover: 0 0% 100%;
    --popover-foreground: 0 0% 3.9%;
    --primary: 0 0% 9%;
    --primary-foreground: 0 0% 98%;
    --secondary: 0 0% 96.1%;
    --secondary-foreground: 0 0% 9%;
    --muted: 0 0% 96.1%;
    --muted-foreground: 0 0% 45.1%;
    --accent: 0 0% 96.1%;
    --accent-foreground: 0 0% 9%;
    --destructive: 0 84.2% 60.2%;
    --destructive-foreground: 0 0% 98%;
    --border: 0 0% 89.8%;
    --input: 0 0% 89.8%;
    --ring: 0 0% 3.9%;
    --chart-1: 12 76% 61%;
    --chart-2: 173 58% 39%;
    --chart-3: 197 37% 24%;
    --chart-4: 43 74% 66%;
    --chart-5: 27 87% 67%;
    --radius: 0.5rem;
  }
  .dark {
    --background: 222 47% 11%;
    --foreground: 0 0% 98%;
    --card: 0 0% 3.9%;
    --card-foreground: 0 0% 98%;
    --popover: 0 0% 3.9%;
    --popover-foreground: 0 0% 98%;
    --primary: 0 0% 98%;

```



```

--primary-foreground: 0 0% 9%;
--secondary: 0 0% 14.9%;
--secondary-foreground: 0 0% 98%;
--muted: 0 0% 14.9%;
--muted-foreground: 0 0% 63.9%;
--accent: 0 0% 14.9%;
--accent-foreground: 0 0% 98%;
--destructive: 0 62.8% 30.6%;
--destructive-foreground: 0 0% 98%;
--border: 0 0% 14.9%;
--input: 0 0% 14.9%;
--ring: 0 0% 83.1%;
--chart-1: 220 70% 50%;
--chart-2: 160 60% 45%;
--chart-3: 30 80% 55%;
--chart-4: 280 65% 60%;
--chart-5: 340 75% 55%;
}
}

@layer base {
  * {
    @apply border-border;
  }
  body {
    @apply bg-background text-foreground;
  }
}

@layer utilities {
  .gradient {
    @apply bg-gradient-to-br from-blue-600 to-purple-600;
  }
  .gradient-title {
    @apply gradient font-extrabold tracking-tighter pr-2 pb-2 text-transparent bg-clip-text;
  }
}

.hero-image-wrapper {
  perspective: 1000px;
}

.hero-image {
  /* transform: rotateX(20deg) scale(0.9) translateY(-50); */
  transform: rotateX(15deg) scale(1);
  transition: transform 0.5s ease-out;
  will-change: transform;
}

.hero-image.scrolled {
  transform: rotateX(0deg) scale(1) translateY(40px);
}

```

```

@keyframes gradientMove {
  0% {
    background-position: 0% 50%;
  }
  50% {
    background-position: 100% 50%;
  }
  100% {
    background-position: 0% 50%;
  }
}

/* Add this class */
.animate-gradient {
  background-size: 200% 200%;
  animation: gradientMove 3s ease infinite;
}

```

MY_FINANCE\app\layout.js

```

import { Inter } from "next/font/google";
import "../globals.css";
import Header from "@/components/header";
import { ClerkProvider } from "@clerk/nextjs";
import { Toaster } from "sonner";
import Image from "next/image";

const inter = Inter({ subsets: ["latin"] });

export const metadata = {
  title: "MY FINANCE",
  description: "Where Financial Freedom Begins",
};

export default function RootLayout({ children }) {
  return (
    <ClerkProvider>
      <html lang="en">
        <head>
          <link rel="icon" href="/logo-sm.png" sizes="any" />
        </head>
        <body className={` ${inter.className}`}>
          <Header />
          <main className="min-h-screen">{children}</main>
          <Toaster richColors />

          <footer className="bg-cyan-100 py-12">

```

```

<div className="container mx-auto px-4 text-center text-gray-600">
  <div className="inline-flex items-center justify-center gap-1.5">
    <p>Developed by <b>SRISAI SHIVAKOTI</b></p>
    <Image
      src="/profile_pic.jpg" // Replace with your image path
      alt="Developer Photo"
      width={32}
      height={32}
      className="h-8 w-8 rounded-full object-cover transition-transform hover:scale-110"
    />
  </div>
</div>
</footer>
</body>
</html>
</ClerkProvider>
);
}

```

MY_FINANCE\app\not-found.jsx

```

import Link from "next/link";
import { Button } from "@components/ui/button";

export default function NotFound() {
  return (
    <div className="flex flex-col items-center justify-center min-h-[100vh] px-4 text-center">
      <h1 className="text-6xl font-bold gradient-title mb-4">404</h1>
      <h2 className="text-2xl font-semibold mb-4">Page Not Found</h2>
      <p className="text-gray-600 mb-8">
        Oops! The page you're looking for doesn't exist or has been
        moved.
      </p>
      <Link href="/">
        <Button>Return Home</Button>
      </Link>
    </div>
  );
}

```

MY_FINANCE\app\page.js

```

import React from "react";
import { Button } from "@components/ui/button";
import { Card, CardContent } from "@components/ui/card";
import Image from "next/image";
import {
  featuresData,
  howItWorksData,
  statsData,
  testimonialsData,
} from "@data/landing";
import HeroSection from "@components/hero";
import Link from "next/link";

const LandingPage = () => {
  return (
    <div className="min-h-screen bg-white">
      {/* Hero Section */}
      <HeroSection />

      {/* Stats Section */}
      <section className="py-20 bg-blue-50">
        <div className="container mx-auto px-4">
          <div className="grid grid-cols-2 md:grid-cols-4 gap-8">
            {statsData.map((stat, index) => (
              <div key={index} className="text-center">
                <div className="text-4xl font-bold text-blue-600 mb-2">
                  {stat.value}
                </div>
                <div className="text-gray-600">{stat.label}</div>
              </div>
            ))}
          </div>
        </div>
      </section>

      {/* Features Section */}
      <section id="features" className="py-20">
        <div className="container mx-auto px-4">
          <h2 className="text-3xl font-bold text-center mb-12">
            Everything you need to manage your finances
          </h2>
          <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 gap-8">
            {featuresData.map((feature, index) => (
              <Card className="p-6" key={index}>
                <CardContent className="space-y-4 pt-4">
                  {feature.icon}
                  <h3 className="text-xl font-semibold">{feature.title}</h3>
                  <p className="text-gray-600">{feature.description}</p>
                </CardContent>
              </Card>
            ))}
          </div>
        </div>
      </section>
    </div>
  );
};

```

```

        </CardContent>
      </Card>
    )})
  </div>
</div>
</section>

{/* How It Works Section */}
<section className="py-20 bg-blue-50">
  <div className="container mx-auto px-4">
    <h2 className="text-3xl font-bold text-center mb-16">How It Works</h2>
    <div className="grid grid-cols-1 md:grid-cols-3 gap-12">
      {howItWorksData.map((step, index) => (
        <div key={index} className="text-center">
          <div className="w-16 h-16 bg-blue-100 rounded-full flex items-center justify-center mx-auto mb-
6">
            {step.icon}
          </div>
          <h3 className="text-xl font-semibold mb-4">{step.title}</h3>
          <p className="text-gray-600">{step.description}</p>
        </div>
      )})
    </div>
  </div>
</section>

{/* Testimonials Section */}
<section id="testimonials" className="py-20">
  <div className="container mx-auto px-4">
    <h2 className="text-3xl font-bold text-center mb-16">
      What Our Users Say
    </h2>
    <div className="grid grid-cols-1 md:grid-cols-3 gap-8">
      {testimonialsData.map((testimonial, index) => (
        <Card key={index} className="p-6">
          <CardContent className="pt-4">
            <div className="flex items-center mb-4">
              <Image
                src={testimonial.image}
                alt={testimonial.name}
                width={40}
                height={40}
                className="rounded-full"
              />
              <div className="ml-4">
                <div className="font-semibold">{testimonial.name}</div>
                <div className="text-sm text-gray-600">
                  {testimonial.role}
                </div>
              </div>
            </div>
            <p className="text-gray-600">{testimonial.quote}</p>
          </CardContent>
        </Card>
      )})
    </div>
  </div>

```

```

        </CardContent>
      </Card>
    )}
  </div>
</div>
</section>

{/* CTA Section */}
<section className="py-20 bg-blue-600">
  <div className="container mx-auto px-4 text-center">
    <h2 className="text-3xl font-bold text-white mb-4">
      Ready to Take Control of Your Finances?
    </h2>
    <p className="text-blue-100 mb-8 max-w-2xl mx-auto">
      Join over 50k+ users who are already managing their finances smarter, saving time, and reaching their
      money goals faster with My Finance.
    </p>
    <Link href="/dashboard">
      <Button
        size="lg"
        className="bg-white text-blue-600 hover:bg-blue-50 animate-bounce"
      >
        Start Free Trial
      </Button>
    </Link>
  </div>
</section>
</div>
);
};

export default LandingPage;

```

MY_FINANCE\components\create-account-drawer.jsx

```

"use client";

import { useState, useEffect } from "react";
import { useForm } from "react-hook-form";
import { zodResolver } from "@hookform/resolvers/zod";
import { Loader2 } from "lucide-react";
import useFetch from "@/hooks/use-fetch";
import { toast } from "sonner";

import { Button } from "@/components/ui/button";
import {
  Drawer,

```

```

DrawerContent,
DrawerHeader,
DrawerTitle,
DrawerTrigger,
DrawerClose,
} from "@components/ui/drawer";
import { Input } from "@components/ui/input";
import {
  Select,
  SelectContent,
  SelectItem,
  SelectTrigger,
  SelectValue,
} from "@components/ui/select";
import { Switch } from "@components/ui/switch";
import { createAccount } from "@actions/dashboard";
import { accountSchema } from "@app/lib/schema";

export function CreateAccountDrawer({ children }) {
  const [open, setOpen] = useState(false);
  const {
    register,
    handleSubmit,
    formState: { errors },
    setValue,
    watch,
    reset,
  } = useForm({
    resolver: zodResolver(accountSchema),
    defaultValues: {
      name: "",
      type: "CURRENT",
      balance: "",
      isDefault: false,
    },
  });

  const {
    loading: createAccountLoading,
    fn: createAccountFn,
    error,
    data: newAccount,
  } = useFetch(createAccount);

  const onSubmit = async (data) => {
    await createAccountFn(data);
  };

  useEffect(() => {
    if (newAccount) {
      toast.success("Account created successfully");
      reset();
    }
  }, [newAccount, reset]);

```

```

    setOpen(false);
  }
}, [newAccount, reset]);

useEffect(() => {
  if (error) {
    toast.error(error.message || "Failed to create account");
  }
}, [error]);

return (
  <Drawer open={open} onOpenChange={setOpen}>
    <DrawerTrigger asChild>{children}</DrawerTrigger>
    <DrawerContent>
      <DrawerHeader>
        <DrawerTitle>Create New Account</DrawerTitle>
      </DrawerHeader>
      <div className="px-4 pb-4">
        <form onSubmit={handleSubmit(onSubmit)} className="space-y-4">
          <div className="space-y-2">
            <label
              htmlFor="name"
              className="text-sm font-medium leading-none peer-disabled:cursor-not-allowed peer-disabled:opacity-70"
            >
              Account Name
            </label>
            <Input
              id="name"
              placeholder="e.g., Main Checking"
              {...register("name")}
            />
            {errors.name && (
              <p className="text-sm text-red-500">{errors.name.message}</p>
            )}
          </div>

          <div className="space-y-2">
            <label
              htmlFor="type"
              className="text-sm font-medium leading-none peer-disabled:cursor-not-allowed peer-disabled:opacity-70"
            >
              Account Type
            </label>
            <Select
              onValueChange={(value) => setValue("type", value)}
              defaultValue={watch("type")}
            >
              <SelectTrigger id="type">
                <SelectValue placeholder="Select type" />
              </SelectTrigger>

```



```

    <SelectContent>
      <SelectItem value="CURRENT">Current</SelectItem>
      <SelectItem value="SAVINGS">Savings</SelectItem>
    </SelectContent>
  </Select>
  {errors.type && (
    <p className="text-sm text-red-500">{errors.type.message}</p>
  )}
</div>

<div className="space-y-2">
  <label
    htmlFor="balance"
    className="text-sm font-medium leading-none peer-disabled:cursor-not-allowed peer-disabled:opacity-70"
  >
    Initial Balance
  </label>
  <Input
    id="balance"
    type="number"
    step="0.01"
    placeholder="0.00"
    {...register("balance")}
  />
  {errors.balance && (
    <p className="text-sm text-red-500">{errors.balance.message}</p>
  )}
</div>

<div className="flex items-center justify-between rounded-lg border p-3">
  <div className="space-y-0.5">
    <label
      htmlFor="isDefault"
      className="text-base font-medium cursor-pointer"
    >
      Set as Default
    </label>
    <p className="text-sm text-muted-foreground">
      This account will be selected by default for transactions
    </p>
  </div>
  <Switch
    id="isDefault"
    checked={watch("isDefault")}
    onCheckedChange={(checked) => setValue("isDefault", checked)}
  />
</div>

<div className="flex gap-4 pt-4">
  <DrawerClose asChild>
    <Button type="button" variant="outline" className="flex-1">

```

```

        Cancel
      </Button>
    </DrawerClose>
    <Button
      type="submit"
      className="flex-1"
      disabled={createAccountLoading}
    >
      {createAccountLoading ? (
        <
          <Loader2 className="mr-2 h-4 w-4 animate-spin" />
          Creating...
        >
      ) : (
        "Create Account"
      )}
    </Button>
  </div>
</form>
</div>
</DrawerContent>
</Drawer>
);
}

```

MY_FINANCE\components\header.jsx

```

import React from "react";
import { Button } from "../ui/button";
import { PenBox, LayoutDashboard } from "lucide-react";
import Link from "next/link";
import { SignedIn, SignedOut, SignInButton, UserButton } from "@clerk/nextjs";
import { checkUser } from "../lib/checkUser";
import Image from "next/image";

const Header = async () => {
  await checkUser();

  return (
    <header className="fixed top-0 w-full bg-white/80 backdrop-blur-md z-50 border-b">
      <nav className="container mx-auto px-4 py-4 flex items-center justify-between">
        <Link href="/">
          <Image
            src={"/logo.png"}
            alt="My Finance Logo"
            width={200}
            height={60}
            className="h-12 w-auto object-contain"

```

```

/>
</Link>

{/* Navigation Links - Different for signed in/out users */}
<div className="hidden md:flex items-center space-x-8">
  <SignedOut>
    <a href="#features" className="text-gray-600 hover:text-blue-600">
      Features
    </a>
    <a
      href="#testimonials"
      className="text-gray-600 hover:text-blue-600"
    >
      Testimonials
    </a>
  </SignedOut>
</div>

{/* Action Buttons */}
<div className="flex items-center space-x-4">
  <SignedIn>
    <Link
      href="/dashboard"
      className="text-gray-600 hover:text-blue-600 flex items-center gap-2"
    >
      <Button variant="outline">
        <LayoutDashboard size={18} />
        <span className="hidden md:inline">Dashboard</span>
      </Button>
    </Link>
    <a href="/transaction/create">
      <Button className="flex items-center gap-2">
        <PenBox size={18} />
        <span className="hidden md:inline">Add Transaction</span>
      </Button>
    </a>
  </SignedIn>
  <SignedOut>
    <SignInButton forceRedirectUrl="/dashboard">
      <Button variant="outline">Login</Button>
    </SignInButton>
  </SignedOut>
  <SignedIn>
    <UserButton
      appearance={{
        elements: {
          avatarBox: "w-10 h-10",
        },
      }}
    />
  </SignedIn>
</div>

```

```

    </nav>
  </header>
);
};

```

```
export default Header;
```

MY_FINANCE\components\hero.jsx

```

"use client";

import React, { useEffect, useRef } from "react";
import Image from "next/image";
import { Button } from "@/components/ui/button";
import Link from "next/link";

const HeroSection = () => {
  const imageRef = useRef(null);

  useEffect(() => {
    const imageElement = imageRef.current;

    const handleScroll = () => {
      const scrollPosition = window.scrollY;
      const scrollThreshold = 100;

      if (scrollPosition > scrollThreshold) {
        imageElement.classList.add("scrolled");
      } else {
        imageElement.classList.remove("scrolled");
      }
    };

    window.addEventListener("scroll", handleScroll);
    return () => window.removeEventListener("scroll", handleScroll);
  }, []);

  return (
    <section className="pt-40 pb-20 px-4">
      <div className="container mx-auto text-center">
        <h1 className="text-5xl md:text-8xl lg:text-[105px] pb-6 gradient-title">
          Finance Made Simple <br/> Growth Made Certain
        </h1>
        <p className="text-xl text-gray-600 mb-8 max-w-2xl mx-auto">
          Track, analyze, and optimize your finances with AI-powered insights. <br />
        </p>
        <div className="flex justify-center space-x-4">
          <Link href="/dashboard">

```

```

    <Button size="lg" className="px-8">
      Get Started
    </Button>
  </Link>
  <Link href="https://www.youtube.com">
    <Button size="lg" variant="outline" className="px-8">
      Watch Demo
    </Button>
  </Link>
</div>
<div className="hero-image-wrapper mt-5 md:mt-0">
  <div ref={imageRef} className="hero-image">
    <Image
      src="/banner.png"
      width={4096}
      height={2304}
      alt="Dashboard Preview"
      className="rounded-lg shadow-2xl border mx-auto"
      priority
    />
  </div>
</div>
</div>
</section>
);
};

export default HeroSection;

```

MY_FINANCE\data\categories.js

```

export const defaultCategories = [
  // Income Categories
  {
    id: "salary",
    name: "Salary",
    type: "INCOME",
    color: "#22c55e", // green-500
    icon: "Wallet",
  },
  {
    id: "freelance",
    name: "Freelance",
    type: "INCOME",
    color: "#06b6d4", // cyan-500
    icon: "Laptop",
  },
  {

```

```

id: "investments",
name: "Investments",
type: "INCOME",
color: "#6366f1", // indigo-500
icon: "TrendingUp",
},
{
id: "business",
name: "Business",
type: "INCOME",
color: "#ec4899", // pink-500
icon: "Building",
},
{
id: "rental",
name: "Rental",
type: "INCOME",
color: "#f59e0b", // amber-500
icon: "Home",
},
{
id: "other-income",
name: "Other Income",
type: "INCOME",
color: "#64748b", // slate-500
icon: "Plus",
},
},

// Expense Categories
{
id: "housing",
name: "Housing",
type: "EXPENSE",
color: "#ef4444", // red-500
icon: "Home",
subcategories: ["Rent", "Mortgage", "Property Tax", "Maintenance"],
},
{
id: "transportation",
name: "Transportation",
type: "EXPENSE",
color: "#f97316", // orange-500
icon: "Car",
subcategories: ["Fuel", "Public Transport", "Maintenance", "Parking"],
},
{
id: "groceries",
name: "Groceries",
type: "EXPENSE",
color: "#84cc16", // lime-500
icon: "Shopping",
},
},

```

```

{
  id: "utilities",
  name: "Utilities",
  type: "EXPENSE",
  color: "#06b6d4", // cyan-500
  icon: "Zap",
  subcategories: ["Electricity", "Water", "Gas", "Internet", "Phone"],
},
{
  id: "entertainment",
  name: "Entertainment",
  type: "EXPENSE",
  color: "#8b5cf6", // violet-500
  icon: "Film",
  subcategories: ["Movies", "Games", "Streaming Services"],
},
{
  id: "food",
  name: "Food",
  type: "EXPENSE",
  color: "#f43f5e", // rose-500
  icon: "UtensilsCrossed",
},
{
  id: "shopping",
  name: "Shopping",
  type: "EXPENSE",
  color: "#ec4899", // pink-500
  icon: "ShoppingBag",
  subcategories: ["Clothing", "Electronics", "Home Goods"],
},
{
  id: "healthcare",
  name: "Healthcare",
  type: "EXPENSE",
  color: "#14b8a6", // teal-500
  icon: "HeartPulse",
  subcategories: ["Medical", "Dental", "Pharmacy", "Insurance"],
},
{
  id: "education",
  name: "Education",
  type: "EXPENSE",
  color: "#6366f1", // indigo-500
  icon: "GraduationCap",
  subcategories: ["Tuition", "Books", "Courses"],
},
{
  id: "personal",
  name: "Personal Care",
  type: "EXPENSE",
  color: "#d946ef", // fuchsia-500

```

```

    icon: "Smile",
    subcategories: ["Haircut", "Gym", "Beauty"],
  },
  {
    id: "travel",
    name: "Travel",
    type: "EXPENSE",
    color: "#0ea5e9", // sky-500
    icon: "Plane",
  },
  {
    id: "insurance",
    name: "Insurance",
    type: "EXPENSE",
    color: "#64748b", // slate-500
    icon: "Shield",
    subcategories: ["Life", "Home", "Vehicle"],
  },
  {
    id: "gifts",
    name: "Gifts & Donations",
    type: "EXPENSE",
    color: "#f472b6", // pink-400
    icon: "Gift",
  },
  {
    id: "bills",
    name: "Bills & Fees",
    type: "EXPENSE",
    color: "#fb7185", // rose-400
    icon: "Receipt",
    subcategories: ["Bank Fees", "Late Fees", "Service Charges"],
  },
  {
    id: "other-expense",
    name: "Other Expenses",
    type: "EXPENSE",
    color: "#94a3b8", // slate-400
    icon: "MoreHorizontal",
  },
];

export const categoryColors = defaultCategories.reduce((acc, category) => {
  acc[category.id] = category.color;
  return acc;
}, {});

```


MY_FINANCE\data\landing.js

```
import {
  BarChart3,
  Receipt,
  PieChart,
  CreditCard,
  Globe,
  Zap,
} from "lucide-react";

// Stats Data
export const statsData = [
  {
    value: "50K+",
    label: "Active Users",
  },
  {
    value: "₹2Cr+",
    label: "Transactions Tracked",
  },
  {
    value: "99.9%",
    label: "Uptime",
  },
  {
    value: "4.9/5",
    label: "User Rating",
  },
];

// Features Data
export const featuresData = [
  {
    icon: <BarChart3 className="h-8 w-8 text-blue-600" />,
    title: "Advanced Analytics",
    description:
      "Get detailed insights into your spending patterns with AI-powered analytics",
  },
  {
    icon: <Receipt className="h-8 w-8 text-blue-600" />,
    title: "Smart Receipt Scanner",
    description:
      "Extract data automatically from receipts using advanced AI technology",
  },
  {
    icon: <PieChart className="h-8 w-8 text-blue-600" />,
    title: "Budget Planning",
    description: "Create and manage budgets with intelligent recommendations",
  },
];
```

```

{
  icon: <CreditCard className="h-8 w-8 text-blue-600" />,
  title: "Multi-Account Support",
  description: "Manage multiple accounts and credit cards in one place",
},
{
  icon: <Globe className="h-8 w-8 text-blue-600" />,
  title: "Multi-Currency",
  description: "Support for multiple currencies with real-time conversion",
},
{
  icon: <Zap className="h-8 w-8 text-blue-600" />,
  title: "Automated Insights",
  description: "Get automated financial insights and recommendations",
},
];

```

// How It Works Data

```

export const howItWorksData = [
  {
    icon: <CreditCard className="h-8 w-8 text-blue-600" />,
    title: "1. Create Your Account",
    description:
      "Get started in minutes with our simple and secure sign-up process",
  },
  {
    icon: <BarChart3 className="h-8 w-8 text-blue-600" />,
    title: "2. Track Your Spending",
    description:
      "Automatically categorize and track your transactions in real-time",
  },
  {
    icon: <PieChart className="h-8 w-8 text-blue-600" />,
    title: "3. Get Insights",
    description:
      "Receive AI-powered insights and recommendations to optimize your finances",
  },
];

```

// Testimonials Data

```

export const testimonialsData = [
  {
    name: "Sarah Johnson",
    role: "Small Business Owner",
    image: "/sarah_johnson.jpg",
    quote:
      "My Finance has transformed how I manage my business finances. The AI insights have helped me identify cost-saving opportunities I never knew existed.",
  },
  {
    name: "Michael Chen",
    role: "Freelancer",

```

```

    image: "/michael_chen.jpg",
    quote:
      "The receipt scanning feature saves me hours each month. Now I can focus on my work instead of manual
data entry and expense tracking.",
  },
  {
    name: "Emily Rodriguez",
    role: "Financial Advisor",
    image: "/emily_rodriguez.jpg",
    quote:
      "I recommend My Finance to all my clients. The multi-currency support and detailed analytics make it
perfect for international investors.",
  },
];

```

MY_FINANCE\emails\template.jsx

```

import {
  Body,
  Container,
  Head,
  Heading,
  Html,
  Preview,
  Section,
  Text,
} from "@react-email/components";

// Dummy data for preview
// const PREVIEW_DATA = {
//   monthlyReport: {
//     userName: "SRISAI SHIVAKOTI",
//     data: {
//       month: "December",
//       stats: {
//         totalIncome: 5000,
//         totalExpenses: 3500,
//         byCategory: {
//           housing: 1500,
//           groceries: 600,
//           transportation: 400,
//           entertainment: 300,
//           utilities: 700,
//         },
//       },
//     },
//     insights: [
//       "Your housing expenses are 43% of your total spending - consider reviewing your housing costs.",
//       "Great job keeping entertainment expenses under control this month!",
//       "Setting up automatic savings could help you save 20% more of your income.",

```

```

//   ],
//   },
//   },
//   budgetAlert: {
//     userName: "Doe",
//     type: "budget-alert",
//     data: {
//       percentageUsed: 85,
//       budgetAmount: 4000,
//       totalExpenses: 3400,
//     },
//   },
//   },
//   },
//   };

export default function EmailTemplate({
  userName = "",
  type = "monthly-report",
  data = {},
}) {
  if (type === "monthly-report") {
    return (
      <Html>
        <Head />
        <Preview>Your Monthly Financial Report</Preview>
        <Body style={styles.body}>
          <Container style={styles.container}>
            <Heading style={styles.title}>Monthly Financial Report</Heading>

            <Text style={styles.text}>Hello {userName},</Text>
            <Text style={styles.text}>
              Here's your financial summary for {data?.month}:
            </Text>

            { /* Main Stats */ }
            <Section style={styles.statsContainer}>
              <div style={styles.stat}>
                <Text style={styles.text}>Total Income</Text>
                <Text style={styles.heading}>₹ {data?.stats.totalIncome.toFixed(2)}</Text>
              </div>
              <div style={styles.stat}>
                <Text style={styles.text}>Total Expenses</Text>
                <Text style={styles.heading}>₹ {data?.stats.totalExpenses.toFixed(2)}</Text>
              </div>
              <div style={styles.stat}>
                <Text style={styles.text}>Net</Text>
                <Text style={styles.heading}>₹ {(data?.stats.totalIncome - data?.stats.totalExpenses).toFixed(2)}
                </Text>
              </div>
            </Section>

            { /* Category Breakdown */ }

```

```

{data?.stats?.byCategory && (
  <Section style={styles.section}>
    <Heading style={styles.heading}>Expenses by Category</Heading>
    {Object.entries(data?.stats.byCategory).map(
      ([category, amount]) => (
        <div key={category} style={styles.row}>
          <Text style={styles.text}>{category} ₹{amount.toFixed(2)}</Text>
        </div>
      )
    )}
  </Section>
)}

```

```

{/* AI Insights */}
{data?.insights && (
  <Section style={styles.section}>
    <Heading style={styles.heading}>My Finance Insights</Heading>
    {data.insights.map((insight, index) => (
      <Text key={index} style={styles.text}>
        • {insight}
      </Text>
    ))}
  </Section>
)}

```

```

<Text style={styles.footer}>
  Thank you for using My Finance. Keep tracking your finances for better
  financial health!
</Text>
</Container>
</Body>
</Html>
);
}

```

```

if (type === "budget-alert") {
  return (
    <Html>
      <Head />
      <Preview>Budget Alert</Preview>
      <Body style={styles.body}>
        <Container style={styles.container}>
          <Heading style={styles.title}>Budget Alert</Heading>
          <Text style={styles.text}>Hello {userName},</Text>
          <Text style={styles.text}>
            You've used {data?.percentageUsed.toFixed(1)}% of your
            monthly budget.
          </Text>
          <Section style={styles.statsContainer}>
            <div style={styles.stat}>
              <Text style={styles.text}>Budget Amount</Text>
              <Text style={styles.heading}>₹{data?.budgetAmount}</Text>

```

```

    </div>
    <div style={styles.stat}>
      <Text style={styles.text}>Spent So Far</Text>
      <Text style={styles.heading}>₹{data?.totalExpenses}</Text>
    </div>
    <div style={styles.stat}>
      <Text style={styles.text}>Remaining</Text>
      <Text style={styles.heading}>
        ₹{data?.budgetAmount - data?.totalExpenses}
      </Text>
    </div>
  </Section>
</Container>
</Body>
</Html>
);
}
}

```

```

const styles = {
  body: {
    backgroundColor: "#f6f9fc",
    fontFamily: "-apple-system, sans-serif",
  },
  container: {
    backgroundColor: "#ffffff",
    margin: "0 auto",
    padding: "20px",
    borderRadius: "5px",
    boxShadow: "0 2px 4px rgba(0, 0, 0, 0.1)",
  },
  title: {
    color: "#1f2937",
    fontSize: "32px",
    fontWeight: "bold",
    textAlign: "center",
    margin: "0 0 20px",
  },
  heading: {
    color: "#1f2937",
    fontSize: "20px",
    fontWeight: "600",
    margin: "0 0 16px",
  },
  text: {
    color: "#4b5563",
    fontSize: "16px",
    margin: "0 0 16px",
  },
  section: {
    marginTop: "32px",
    padding: "20px",
  },

```

```

    backgroundColor: "#f9fafb",
    borderRadius: "5px",
    border: "1px solid #e5e7eb",
  },
  statsContainer: {
    margin: "32px 0",
    padding: "20px",
    backgroundColor: "#f9fafb",
    borderRadius: "5px",
  },
  stat: {
    marginBottom: "16px",
    padding: "12px",
    backgroundColor: "#fff",
    borderRadius: "4px",
    boxShadow: "0 1px 2px rgba(0, 0, 0, 0.05)",
  },
  row: {
    display: "flex",
    justifyContent: "space-between",
    padding: "12px 0",
    borderBottom: "1px solid #e5e7eb",
  },
  footer: {
    color: "#6b7280",
    fontSize: "14px",
    textAlign: "center",
    marginTop: "32px",
    padding: "16px",
    borderTop: "1px solid #e5e7eb",
  },
};

```

MY_FINANCE\hooks\use-fetch.js

```

import { useState } from "react";
import { toast } from "sonner";

const useFetch = (cb) => {
  const [data, setData] = useState(undefined);
  const [loading, setLoading] = useState(null);
  const [error, setError] = useState(null);

  const fn = async (...args) => {
    setLoading(true);
    setError(null);
    try {
      const response = await cb(...args);
      setData(response);
    } catch (error) {
      toast.error(error.message);
    }
  };

  return { data, loading, error, fn };
};

```

```

      setError(null);
    } catch (error) {
      setError(error);
      toast.error(error.message);
    } finally {
      setLoading(false);
    }
  };
  return { data, loading, error, fn, setData };
};

```

export default useFetch;

MY_FINANCE\lib\inngest\client.js

```
import { Inngest } from "inngest";
```

```

export const inngest = new Inngest({
  id: "my_finance", // Unique app ID
  name: "My Finance",
  retryFunction: async (attempt) => ({
    delay: Math.pow(2, attempt) * 1000, // Exponential backoff
    maxAttempts: 3,
  }),
});

```

MY_FINANCE\lib\inngest\function.js

```

import { inngest } from "../client";
import { db } from "@lib/prisma";
import EmailTemplate from "@emails/template";
import { sendEmail } from "@actions/send-email";
import { GoogleGenerativeAI } from "@google/generative-ai";

```

// 1. Recurring Transaction Processing with Throttling

```

export const processRecurringTransaction = inngest.createFunction(
  {
    id: "process-recurring-transaction",
    name: "Process Recurring Transaction",
    throttle: {
      limit: 10, // Process 10 transactions
      period: "1m", // per minute
      key: "event.data.userId", // Throttle per user
    },
  },
  { event: "transaction.recurring.process" },
  async ({ event, step }) => {
    // Validate event data

```



```

if (!event?.data?.transactionId || !event?.data?.userId) {
  console.error("Invalid event data:", event);
  return { error: "Missing required event data" };
}

await step.run("process-transaction", async () => {
  const transaction = await db.transaction.findUnique({
    where: {
      id: event.data.transactionId,
      userId: event.data.userId,
    },
    include: {
      account: true,
    },
  });

  if (!transaction || !isTransactionDue(transaction)) return;

  // Create new transaction and update account balance in a transaction
  await db.$transaction(async (tx) => {
    // Create new transaction
    await tx.transaction.create({
      data: {
        type: transaction.type,
        amount: transaction.amount,
        description: `${transaction.description} (Recurring)`,
        date: new Date(),
        category: transaction.category,
        userId: transaction.userId,
        accountId: transaction.accountId,
        isRecurring: false,
      },
    });

    // Update account balance
    const balanceChange =
      transaction.type === "EXPENSE"
        ? -transaction.amount.toNumber()
        : transaction.amount.toNumber();

    await tx.account.update({
      where: { id: transaction.accountId },
      data: { balance: { increment: balanceChange } },
    });

    // Update last processed date and next recurring date
    await tx.transaction.update({
      where: { id: transaction.id },
      data: {
        lastProcessed: new Date(),
        nextRecurringDate: calculateNextRecurringDate(
          new Date(),

```

```

        transaction.recurringInterval
      ),
    },
  });
});
});
}
);

// Trigger recurring transactions with batching
export const triggerRecurringTransactions = inngest.createFunction(
{
  id: "trigger-recurring-transactions", // Unique ID,
  name: "Trigger Recurring Transactions",
},
{ cron: "0 0 * * *" }, // Daily at midnight
async ({ step }) => {
  const recurringTransactions = await step.run(
    "fetch-recurring-transactions",
    async () => {
      return await db.transaction.findMany({
        where: {
          isRecurring: true,
          status: "COMPLETED",
          OR: [
            { lastProcessed: null },
            {
              nextRecurringDate: {
                lte: new Date(),
              },
            },
          ],
        },
      });
    }
  );
});

// Send event for each recurring transaction in batches
if (recurringTransactions.length > 0) {
  const events = recurringTransactions.map((transaction) => ({
    name: "transaction.recurring.process",
    data: {
      transactionId: transaction.id,
      userId: transaction.userId,
    },
  }));

  // Send events directly using inngest.send()
  await inngest.send(events);
}

return { triggered: recurringTransactions.length };

```

```

    }
  );

// 2. Monthly Report Generation
async function generateFinancialInsights(stats, month) {
  const genAI = new GoogleGenerativeAI(process.env.GEMINI_API_KEY);
  const model = genAI.getGenerativeModel({ model: "gemini-1.5-flash" });

  const prompt = `
    Analyze this financial data and provide 3 concise, actionable insights.
    Focus on spending patterns and practical advice.
    Keep it friendly and conversational.

    Financial Data for ${month}:
    - Total Income: ₹${stats.totalIncome.toFixed(2)}
    - Total Expenses: ₹${(stats.totalExpenses.toFixed(2))}
    - Net Income: ₹${(stats.totalIncome - stats.totalExpenses).toFixed(2)}
    - Expense Categories: ${Object.entries(stats.byCategory)
      .map(([category, amount]) => `${category} ₹${amount.toFixed(2)} `)
      .join(", ")}

    Format the response as a JSON array of strings, like this:
    ["insight 1", "insight 2", "insight 3"]
  `;

  try {
    const result = await model.generateContent(prompt);
    const response = result.response;
    const text = response.text();
    const cleanedText = text.replace(/^``(?:json)?\n?/g, "").trim();

    return JSON.parse(cleanedText);
  } catch (error) {
    console.error("Error generating insights:", error);
    return [
      "Your highest expense category this month might need attention.",
      "Consider setting up a budget for better financial management.",
      "Track your recurring expenses to identify potential savings.",
    ];
  }
}

export const generateMonthlyReports = inngest.createFunction(
  {
    id: "generate-monthly-reports",
    name: "Generate Monthly Reports",
  },
  { cron: "0 0 1 * *" }, // First day of each month
  async ({ step }) => {
    const users = await step.run("fetch-users", async () => {
      return await db.user.findMany({
        include: { accounts: true },
      });
    });
  }
}

```

```

    });
  });

  for (const user of users) {
    await step.run(`generate-report-${user.id}`, async () => {
      const lastMonth = new Date();
      lastMonth.setMonth(lastMonth.getMonth() - 1);

      const stats = await getMonthlyStats(user.id, lastMonth);
      const monthName = lastMonth.toLocaleString("default", {
        month: "long",
      });

      // Generate AI insights
      const insights = await generateFinancialInsights(stats, monthName);

      await sendEmail({
        to: user.email,
        subject: `Your Monthly Financial Report - ${monthName}`,
        react: EmailTemplate({
          userName: user.name,
          type: "monthly-report",
          data: {
            stats,
            month: monthName,
            insights,
          },
        }),
      });
    });
  }

  return { processed: users.length };
}
);

```

// 3. Budget Alerts with Event Batching

```

export const checkBudgetAlerts = inngest.createFunction(
  { name: "Check Budget Alerts" },
  { cron: "0 */6 * * *" }, // Every 6 hours
  async ({ step }) => {
    const budgets = await step.run("fetch-budgets", async () => {
      return await db.budget.findMany({
        include: {
          user: {
            include: {
              accounts: {
                where: {
                  isDefault: true,
                },
              },
            },
          },
        },
      });
    });
  },
);

```

```

    },
  },
});
});

for (const budget of budgets) {
  const defaultAccount = budget.user.accounts[0];
  if (!defaultAccount) continue; // Skip if no default account

  await step.run(`check-budget-${budget.id}`, async () => {
    const startDate = new Date();
    startDate.setDate(1); // Start of current month

    // Calculate total expenses for the default account only
    const expenses = await db.transaction.aggregate({
      where: {
        userId: budget.userId,
        accountId: defaultAccount.id, // Only consider default account
        type: "EXPENSE",
        date: {
          gte: startDate,
        },
      },
      _sum: {
        amount: true,
      },
    });

    const totalExpenses = expenses._sum.amount?.toNumber() || 0;
    const budgetAmount = budget.amount;
    const percentageUsed = (totalExpenses / budgetAmount) * 100;

    // Check if we should send an alert
    if (
      percentageUsed >= 80 && // Default threshold of 80%
      (!budget.lastAlertSent ||
        isNewMonth(new Date(budget.lastAlertSent), new Date()))
    ) {
      await sendEmail({
        to: budget.user.email,
        subject: `Budget Alert for ${defaultAccount.name}`,
        react: EmailTemplate({
          userName: budget.user.name,
          type: "budget-alert",
          data: {
            percentageUsed,
            budgetAmount: parseInt(budgetAmount).toFixed(1),
            totalExpenses: parseInt(totalExpenses).toFixed(1),
            accountName: defaultAccount.name,
          },
        }),
      });
    }
  });
}

```

```

    // Update last alert sent
    await db.budget.update({
      where: { id: budget.id },
      data: { lastAlertSent: new Date() },
    });
  }
});
}
}
);

function isNewMonth(lastAlertDate, currentDate) {
  return (
    lastAlertDate.getMonth() !== currentDate.getMonth() ||
    lastAlertDate.getFullYear() !== currentDate.getFullYear()
  );
}

```

```

// Utility functions
function isTransactionDue(transaction) {
  // If no lastProcessed date, transaction is due
  if (!transaction.lastProcessed) return true;

  const today = new Date();
  const nextDue = new Date(transaction.nextRecurringDate);

  // Compare with nextDue date
  return nextDue <= today;
}

```

```

function calculateNextRecurringDate(date, interval) {
  const next = new Date(date);
  switch (interval) {
    case "DAILY":
      next.setDate(next.getDate() + 1);
      break;
    case "WEEKLY":
      next.setDate(next.getDate() + 7);
      break;
    case "MONTHLY":
      next.setMonth(next.getMonth() + 1);
      break;
    case "YEARLY":
      next.setFullYear(next.getFullYear() + 1);
      break;
  }
  return next;
}

```

```

async function getMonthlyStats(userId, month) {
  const startDate = new Date(month.getFullYear(), month.getMonth(), 1);

```

```

const endDate = new Date(month.getFullYear(), month.getMonth() + 1, 0);

const transactions = await db.transaction.findMany({
  where: {
    userId,
    date: {
      gte: startDate,
      lte: endDate,
    },
  },
});

return transactions.reduce(
  (stats, t) => {
    const amount = t.amount.toNumber();
    if (t.type === "EXPENSE") {
      stats.totalExpenses += amount;
      stats.byCategory[t.category] =
        (stats.byCategory[t.category] || 0) + amount;
    } else {
      stats.totalIncome += amount;
    }
    return stats;
  },
  {
    totalExpenses: 0,
    totalIncome: 0,
    byCategory: {},
    transactionCount: transactions.length,
  }
);
}

```

MY_FINANCE\lib\arcjet.js

```

import arcjet, { tokenBucket } from "@arcjet/next";
const aj = arcjet({
  key: process.env.ARCJET_KEY,
  characteristics: ["userId"], // Track based on Clerk userId
  rules: [
    // Rate limiting specifically for collection creation
    tokenBucket({
      mode: "LIVE",
      refillRate: 20, // 10 collections
      interval: 3600, // per hour
      capacity: 20, // maximum burst capacity
    }),
  ],
});
export default aj;

```

MY_FINANCE\lib\checkUser.js

```
import { currentUser } from "@clerk/nextjs/server";
import { db } from "../prisma";

export const checkUser = async () => {
  const user = await currentUser();

  if (!user) {
    return null;
  }

  try {
    const loggedInUser = await db.user.findUnique({
      where: {
        clerkUserId: user.id,
      },
    });

    if (loggedInUser) {
      return loggedInUser;
    }

    const name = `${user.firstName} ${user.lastName}`;

    const newUser = await db.user.create({
      data: {
        clerkUserId: user.id,
        name,
        imageUrl: user.imageUrl,
        email: user.emailAddresses[0].emailAddress,
      },
    });

    return newUser;
  } catch (error) {
    console.log(error.message);
  }
};
```


MY_FINANCE\lib\prisma.js

```
import { PrismaClient } from "@prisma/client";

export const db = globalThis.prisma || new PrismaClient();

if (process.env.NODE_ENV !== "production") {
  globalThis.prisma = db;
}

// globalThis.prisma: This global variable ensures that the Prisma client instance is
// reused across hot reloads during development. Without this, each time your application
// reloads, a new instance of the Prisma client would be created, potentially leading
// to connection issues.
```

MY_FINANCE\lib\utils.js

```
import { clsx } from "clsx";
import { twMerge } from "tailwind-merge"

export function cn(...inputs) {
  return twMerge(clsx(inputs));
}
```

MY_FINANCE\prisma\schema.prisma

```

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

datasource db {
  provider = "postgresql"
  url      = env("DATABASE_URL")
  directUrl = env("DIRECT_URL")
}

model User {
  id          String   @id @default(uuid())
  clerkUserId String   @unique // clerk user id
  email       String   @unique
  name        String?
  imageUrl    String?
  transactions Transaction[]
  accounts    Account[]
  budgets     Budget[]
  createdAt   DateTime @default(now())
  updatedAt   DateTime @updatedAt

  @@map("users")
}

model Account {
  id          String   @id @default(uuid())
  name        String
  type        AccountType
  balance     Decimal  @default(0) // will ask initial balance while creating an account
  isDefault   Boolean  @default(false)
  userId      String
  user        User     @relation(fields: [userId], references: [id], onDelete: Cascade)
  transactions Transaction[]
  createdAt   DateTime @default(now())
  updatedAt   DateTime @updatedAt

  @@index([userId])
  @@map("accounts")
}

model Transaction {
  id          String   @id @default(uuid())
  type        TransactionType
  amount      Decimal
  description  String?
  date        DateTime
  category    String
  receiptUrl   String?
  isRecurring Boolean  @default(false)

```

```

recurringInterval RecurringInterval? // Only used if isRecurring is true
nextRecurringDate DateTime? // Next date for recurring transaction
lastProcessed DateTime? // Last time this recurring transaction was processed
status TransactionStatus @default(COMPLETED)
userId String
user User @relation(fields: [userId], references: [id], onDelete: Cascade)
accountId String
account Account @relation(fields: [accountId], references: [id], onDelete: Cascade)
createdAt DateTime @default(now())
updatedAt DateTime @updatedAt

```

```

@@index([userId])
@@index([accountId])
@@map("transactions")
}

```

```

model Budget {
  id String @id @default(uuid())
  amount Decimal
  lastAlertSent DateTime? // Track when the last alert was sent
  userId String @unique
  user User @relation(fields: [userId], references: [id], onDelete: Cascade)
  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt

  @@index([userId])
  @@map("budgets")
}

```

```

enum TransactionType {
  INCOME
  EXPENSE
}

```

```

enum AccountType {
  CURRENT
  SAVINGS
}

```

```

enum TransactionStatus {
  PENDING
  COMPLETED
  FAILED
}

```

```

enum RecurringInterval {
  DAILY
  WEEKLY
  MONTHLY
  YEARLY
}

```

MY_FINANCE\middleware.js

```
import arcjet, { createMiddleware, detectBot, shield } from "@arcjet/next";
import { clerkMiddleware, createRouteMatcher } from "@clerk/nextjs/server";
import { NextResponse } from "next/server";
```

```
const isProtectedRoute = createRouteMatcher([
  "/dashboard(.*)",
  "/account(.*)",
  "/transaction(.*)",
]);
```

```
// Create Arcjet middleware
```

```
const aj = arcjet({
  key: process.env.ARCJET_KEY,
  // characteristics: ["userId"], // Track based on Clerk userId
  rules: [
    // Shield protection for content and security
    shield({
      mode: "LIVE",
    }),
    detectBot({
      mode: "LIVE", // will block requests. Use "DRY_RUN" to log only
      allow: [
        "CATEGORY:SEARCH_ENGINE", // Google, Bing, etc
        "GO_HTTP", // For Inngest
        // See the full list at https://arcjet.com/bot-list
      ],
    }),
  ],
});
```

```
// Create base Clerk middleware
```

```
const clerk = clerkMiddleware(async (auth, req) => {
  const { userId } = await auth();

  if (!userId && isProtectedRoute(req)) {
    const { redirectToSignIn } = await auth();
    return redirectToSignIn();
  }

  return NextResponse.next();
});
```

```
// Chain middlewares - ArcJet runs first, then Clerk
```

```
export default createMiddleware(aj, clerk);
```

```
export const config = {
  matcher: [
    // Skip Next.js internals and all static files, unless found in search params
```

```
"/((?!_next|[\^?]*\\.(?:html?|css|js(?:!on)|jpe?g|webp|png|gif|svg|ttf|woff2?|ico|csv|docx?|xlsx?|zip|webmanifest)).*)",
// Always run for API routes
"/(api|trpc)(.*)",
],
};
```

MY_FINANCE\package.json

```
{
  "name": "my_finance",
  "version": "0.1.0",
  "private": true,
  "scripts": {
    "dev": "next dev --turbo",
    "build": "next build",
    "start": "next start",
    "lint": "next lint",
    "email": "email dev",
    "postinstall": "prisma generate"
  },
  "dependencies": {
    "@arcjet/next": "^1.0.0-alpha.34",
    "@clerk/nextjs": "^6.6.0",
    "@google/generative-ai": "^0.21.0",
    "@hookform/resolvers": "^3.9.1",
    "@prisma/client": "^6.0.1",
    "@radix-ui/react-checkbox": "^1.1.2",
    "@radix-ui/react-dialog": "^1.1.2",
    "@radix-ui/react-dropdown-menu": "^2.1.2",
    "@radix-ui/react-popover": "^1.1.2",
    "@radix-ui/react-progress": "^1.1.0",
    "@radix-ui/react-select": "^2.1.2",
    "@radix-ui/react-slot": "^1.1.0",
    "@radix-ui/react-switch": "^1.1.1",
    "@radix-ui/react-tooltip": "^1.1.4",
    "@react-email/components": "0.0.30",
    "class-variance-authority": "^0.7.1",
    "clsx": "^2.1.1",
    "date-fns": "^4.1.0",
    "innngest": "^3.27.4",
    "lucide-react": "^0.462.0",
    "next": "15.0.3",
    "next-themes": "^0.4.3",
    "prisma": "^6.0.1",
    "react": "^19.0.0-rc-66855b96-20241106",
    "react-day-picker": "^8.10.1",
```

```

"react-dom": "^19.0.0-rc-66855b96-20241106",
"react-hook-form": "^7.53.2",
"react-spinners": "^0.14.1",
"recharts": "^2.14.1",
"resend": "^4.0.1",
"sonner": "^1.7.0",
"tailwind-merge": "^2.5.5",
"tailwindcss-animate": "^1.0.7",
"vaul": "^1.1.1",
"zod": "^3.23.8"
},
"devDependencies": {
"@babel/core": "^7.26.10",
"babel-plugin-styled-components": "^2.1.4",
"eslint": "^8",
"eslint-config-next": "15.0.3",
"postcss": "^8",
"prisma": "^6.0.1",
"react-email": "3.0.3",
"tailwindcss": "^3.4.1"
}
}

```

MY_FINANCE\README.md

🎯 MY FINANCE

A full-stack AI-powered personal finance management platform built with Next.js, Supabase, Prisma, Tailwind CSS, Inngest, ArcJet, Resend, and ShadCN UI.

![Project Screenshot](public/screenshot.png)

This application enables users to:

- * Manage multiple bank accounts
- * Track income and expenses
- * Scan receipts using AI (Gemini API)
- * Receive automated monthly financial reports with AI-generated insights
- * Categorize transactions intelligently
- * Set budgets and receive alerts
- * Visualize financial data through interactive charts
- * Handle recurring transactions
- * Authenticate via Clerk (Email/Password or Google)
- * Implement rate limiting and security with ArcJet

🧰 Tech Stack

- * **Frontend**: Next.js 15, React 19, Tailwind CSS, ShadCN UI
- * **Backend**: Supabase (PostgreSQL), Prisma ORM

- * ****Authentication****: Clerk
- * ****AI Integration****: Google Gemini API
- * ****Email Notifications****: Resend
- * ****Background Jobs****: Inngest
- * ****Rate Limiting & Security****: ArcJet
- * ****Deployment****: Vercel

🚀 Features

- * ****Multi-Account Management****: Link and manage multiple bank accounts.
- * ****AI Receipt Scanning****: Upload receipts to extract transaction details using the Gemini API.
- * ****Automated Monthly Reports****: Receive monthly emails with AI-generated financial insights.
- * ****Smart Categorization****: Automatically categorize transactions into predefined categories.
- * ****Budget Tracking****: Set spending limits and receive alerts when approaching them.
- * ****Interactive Dashboards****: Visualize income, expenses, and savings through charts.
- * ****Recurring Transactions****: Manage and track recurring expenses and incomes.
- * ****Secure Authentication****: Authenticate users securely using Clerk.
- * ****Rate Limiting****: Protect APIs from abuse using ArcJet.

📁 Project Structure

```

```bash
├── app/ # Next.js app directory
├── components/ # Reusable UI components
├── data/ # Static data and constants
├── emails/ # Email templates
├── hooks/ # Custom React hooks
├── lib/ # Utility functions and libraries
├── prisma/ # Prisma schema and migrations
├── public/ # Public assets
├── styles/ # Global styles
├── .env.example # Environment variable examples
├── next.config.js # Next.js configuration
├── tailwind.config.js # Tailwind CSS configuration
└── package.json # Project dependencies and scripts
```

```

⚙️ Environment Variables

Create a `.env` file in the root directory and add the following variables:

```

```env
NEXT_PUBLIC_CLERK_PUBLISHABLE_KEY=
CLERK_SECRET_KEY=

NEXT_PUBLIC_CLERK_SIGN_IN_URL =
NEXT_PUBLIC_CLERK_SIGN_UP_URL =

Connect to Supabase via connection pooling.
DATABASE_URL=

```

# Direct connection to the database. Used for migrations.

DIRECT\_URL=

ARCJET\_KEY=

RESEND\_API\_KEY =

GEMINI\_API\_KEY =

```

🧑💻 Getting Started

1. ****Clone the repository****:

```
```bash
git clone https://github.com/Srisai16/MY_FINANCE.git
```
```

2. ****Install dependencies****:

```
```bash
npm install
or
yarn install
```
```

3. ****Set up the database****:

```
```bash
npx prisma migrate dev --name init
```
```

4. ****Run the development server****:

```
```bash
npm run dev
or
yarn dev
```
```

5. ****Access the application****:

Open http://localhost:3000 in your browser.

📧 Email Templates

Email templates are located in the `emails/` directory. They are used for sending monthly financial reports and other notifications via Resend.

🧪 Testing

To run tests (if available):

```
```bash
npm run test
or
yarn test
```
```

📄 License

This project is licensed under the MIT License. See the LICENSE file for details.

🙌 Acknowledgements

This project is developed and maintained by [SRISAI SHIVAKOTI](<https://github.com/Srisai16>).

MY_FINANCE\LICENSE

MIT License

Copyright (c) 2025 Srisai16

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

9. TEST CASES

Introduction to Testing

Software testing is a critical part of the development lifecycle that ensures the functionality, reliability, and performance of an application. In the case of the **MY_FINANCE** personal finance tracker, testing helps verify that financial data is correctly stored, transactions are accurately recorded, user authentication is secure, and AI-based features such as receipt scanning function as expected.

This project employed both manual and automated testing approaches to identify bugs, validate logic, and maintain user experience standards.

Types of Testing Used

1. Functional Testing

Functional testing ensures that each function of the software operates in accordance with requirements. In **MY_FINANCE**, functional tests were conducted for:

- Login and Registration
- Adding, editing, and deleting transactions
- Setting and tracking budgets
- Receipt scanning and form auto-fill
- Report generation and visualization

Each user interaction was tested to ensure it produced the expected result with valid and invalid inputs.

2. Unit Testing

Unit testing focuses on testing individual components in isolation. In this application, reusable functions like data formatters, category classifiers, and Supabase API interactions were tested using JavaScript-based testing tools to confirm their correctness.

3. Integration Testing

Integration testing verified the seamless working between different modules. For example:

- `ReceiptScanner.jsx` integrating with the Gemini API
- Supabase functions syncing with `TransactionForm.jsx` and `Reports.jsx`
- Authentication via Clerk working alongside page routing

This ensured that multiple components working together do not result in failures.

4. UI/UX Testing

User Interface testing was done manually on various screen sizes to verify responsiveness, correct element alignment, accessibility, and overall navigation flow. Special care was taken to ensure form validations, alerts, and button behaviors were intuitive and consistent.

5. Security Testing

Although the app does not handle direct bank connections or sensitive financial credentials, security was ensured through:

- Clerk JWT-based authentication
 - Supabase Row-Level Security (RLS)
 - Enforcing HTTPS for secure data transmission
- Unauthorized data access attempts were simulated to test protection mechanisms.

6. Regression Testing

Regression testing was performed after integrating new features like budget alerts and receipt scanning. Previously working functionalities were retested to ensure new changes did not break existing modules.

7. Cross-Browser Testing

The application was tested across multiple browsers including Chrome, Firefox, and Edge to ensure consistent layout rendering and functionality.

Testing Strategy

The following strategy was followed during the development:

- **Test Early, Test Often:** Testing began during development and continued after each module was completed.
- **Component-wise Testing:** Each module (e.g., Login, Dashboard, Transactions) was tested individually before integrating with others.
- **Test Case Documentation:** Clear documentation of test cases was maintained to ensure coverage and reproducibility.
- **Bug Tracking and Fixing:** Bugs were logged and prioritized based on severity. Fixes were retested for confirmation.
- **User Simulation:** Manual user interaction simulation was used to test real-world usage patterns like budgeting, overspending, and receipt scanning.

| TC ID | Test Case Description | Input | Expected Result | Status |
|-------|--|---|---|--------|
| TC01 | User Login with valid credentials | Email, Password | User is logged in and redirected to dashboard | Pass |
| TC02 | User Login with invalid credentials | Wrong Email/Password | Error message displayed, login fails | Pass |
| TC03 | New user registration | Valid Email, Password | Account created, redirected to login page | Pass |
| TC04 | Add a manual transaction | Title, Amount, Category, Type, Date | Transaction saved and visible in transaction list | Pass |
| TC05 | Add transaction with missing fields | Empty Title or Amount | Validation error shown | Pass |
| TC06 | Edit an existing transaction | Update title or amount | Transaction updated in database and UI | Pass |
| TC07 | Delete a transaction | Click delete on a record | Transaction is removed from the list and database | Pass |
| TC08 | Upload a valid receipt for scanning | Image of printed receipt | Parsed text shown with pre-filled form fields | Pass |
| TC09 | Upload an invalid or unclear receipt | Blurry/Unreadable image | OCR fails, fallback message shown | Pass |
| TC10 | Set a budget for a category | Category: Food, Limit: ₹5000 | Budget saved, progress bar shown | Pass |
| TC11 | Exceed the set budget | Add expenses above budget | Alert shown: "Budget exceeded" | Pass |
| TC12 | View monthly report | Select a month | Pie chart and bar chart with categorized data shown | Pass |
| TC13 | Logout from session | Click logout | User session cleared, redirected to login page | Pass |
| TC14 | Access dashboard without login | Open dashboard URL directly | Redirected to login page (unauthorized) | Pass |
| TC15 | Validate role-based access to personal data only | Logged-in user accesses another user's data | Access denied (Row-Level Security enforced) | Pass |

10. OUTPUT SCREENS

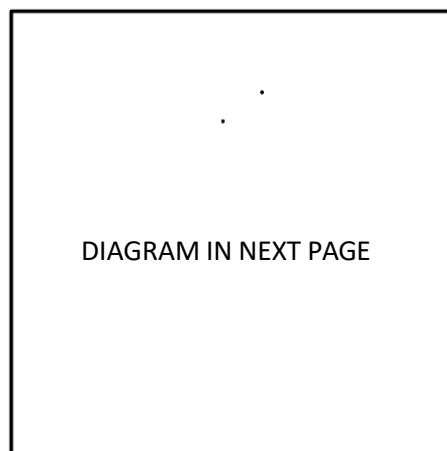
The following images represent the core user interface of the MY_FINANCE application:


- A professionally designed landing page introduces users to the platform’s features.
- The authentication screen secures access via Clerk.
- The dashboard and transaction views provide visual insights and financial tracking.
- AI integration is showcased through the receipt scanning module.


All components are responsive, cleanly styled, and user-friendly—emphasizing clarity, automation, and ease of use.

◆ 1. Landing Page / Home Page

This is the introductory screen that welcomes users with a modern UI, promotional messaging, and highlights such as multi-account support, AI insights, and receipt scanning. It includes feature blocks, user testimonials, and a call-to-action.






[Dashboard](#)
[Add Transaction](#)


Finance Made Simple Growth Made Certain

Track, analyze, and optimize your finances with AI-powered insights.

[Get Started](#)
[Watch Demo](#)




50K+
Active Users


₹2Cr+
Transactions Tracked


99.9%
Uptime


4.9/5
User Rating


Everything you need to manage your finances



Advanced Analytics
Get detailed insights into your spending patterns with AI-powered analytics.


Smart Receipt Scanner
Extract data automatically from receipts using advanced AI technology.



Budget Planning
Create and manage budgets with intelligent recommendations.



Multi-Account Support
Manage multiple accounts and credit cards in one place.



Multi-Currency
Support for multiple currencies with real-time conversion.


Automated Insights
Get automated financial insights and recommendations.


How It Works



1. Create Your Account
Get started in minutes with our simple and secure sign-up process.



2. Track Your Spending
Automatically categorize and track your transactions in real-time.


3. Get Insights
Receive AI-powered insights and recommendations to optimize your finances.

What Our Users Say


Sarah Johnson
Small Business Owner
My Finance has transformed how I manage my business finances. The AI insights have helped me identify cost-saving opportunities I never knew existed.


Michael Chen
Freelancer
The receipt scanning feature saves me hours each month. Now I can focus on my work instead of manual data entry and expense tracking.


Emily Rodriguez
Financial Advisor
I recommend My Finance to all my clients. The multi-currency support and detailed analytics make it perfect for international investors.

Ready to Take Control of Your Finances?

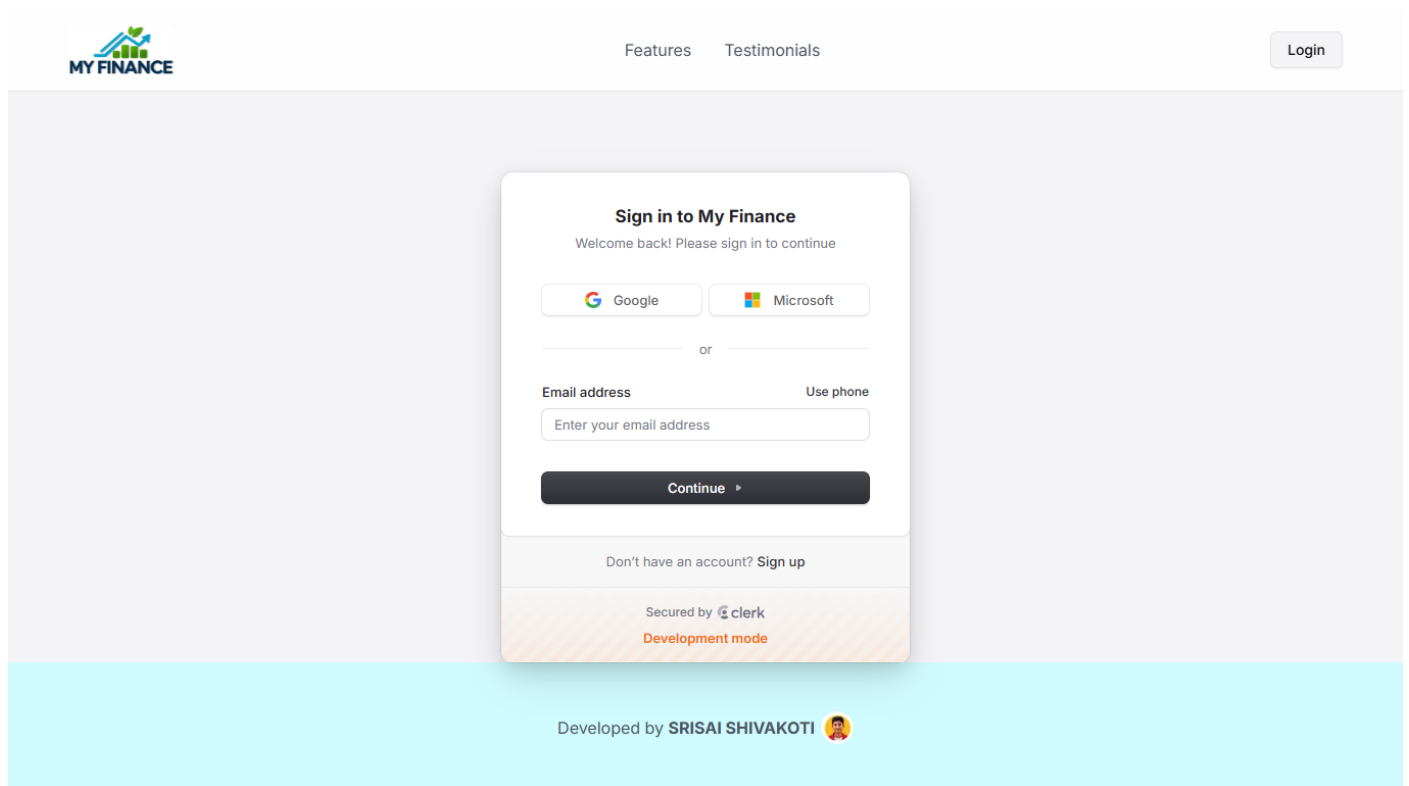
Join over 50K+ users who are already managing their finances smarter, saving time, and reaching their money goals faster with My Finance.

[Start Free Trial](#)

Developed by **SRISAI SHIVAKOTI**

◆ 2. User Login Interface

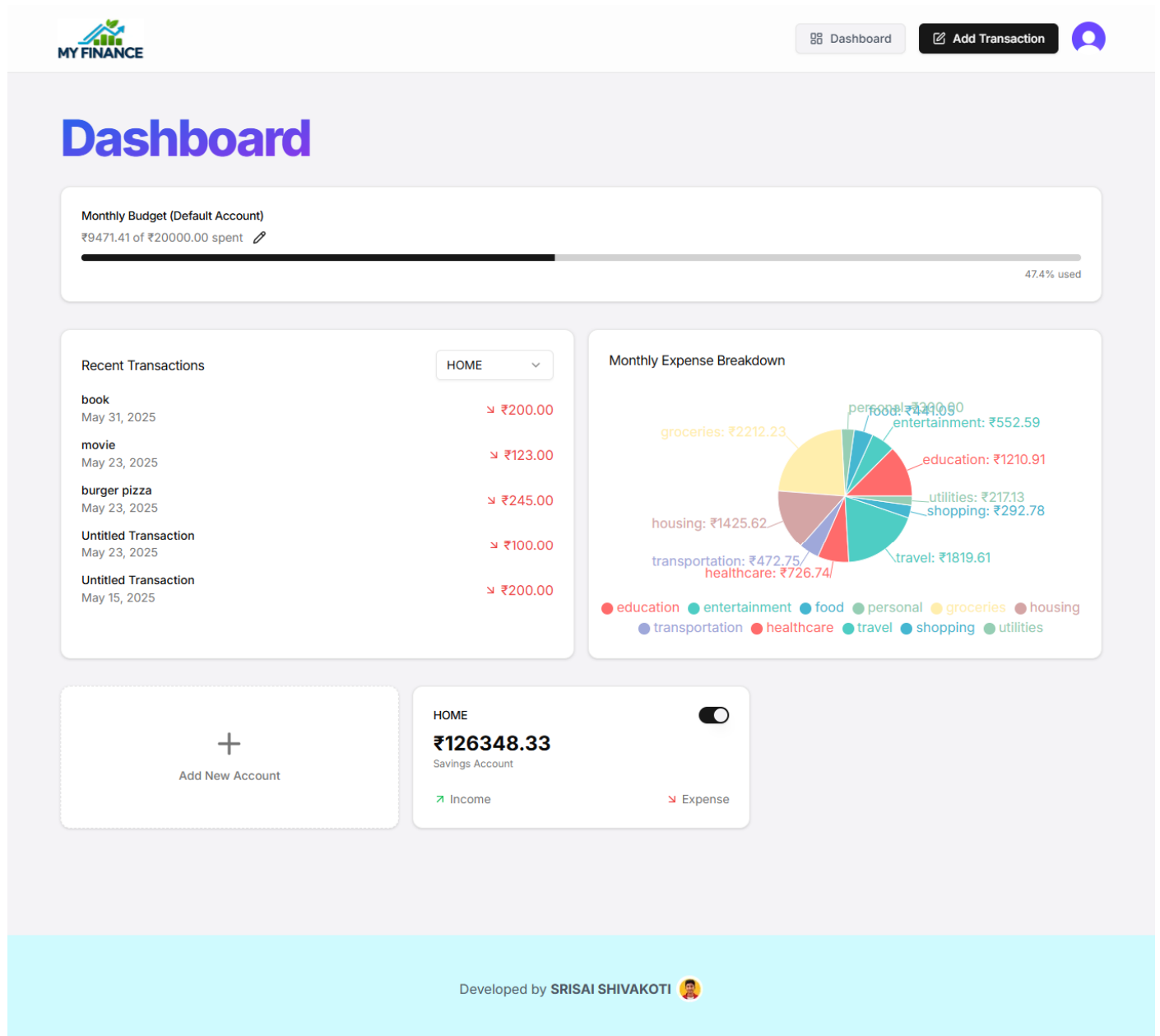
This screen displays the sign-in interface, where users can log in using Google, Microsoft, email, or phone. It is powered by Clerk for secure authentication.



The screenshot displays the 'Sign in to My Finance' login interface. At the top left is the 'MY FINANCE' logo, and at the top right are links for 'Features' and 'Testimonials', along with a 'Login' button. The main content area features a central white login card. The card has the title 'Sign in to My Finance' and a subtitle 'Welcome back! Please sign in to continue'. It offers two social login options: 'Google' and 'Microsoft'. Below these, a horizontal line with the word 'or' separates them from the email/phone login section. This section has two tabs: 'Email address' (selected) and 'Use phone'. Under the 'Email address' tab is a text input field with the placeholder 'Enter your email address'. Below the input field is a dark 'Continue' button with a right-pointing arrow. At the bottom of the card, there is a link 'Don't have an account? Sign up'. The footer of the card states 'Secured by clerk' and 'Development mode'. The entire interface is set against a light blue background with a darker blue footer bar.

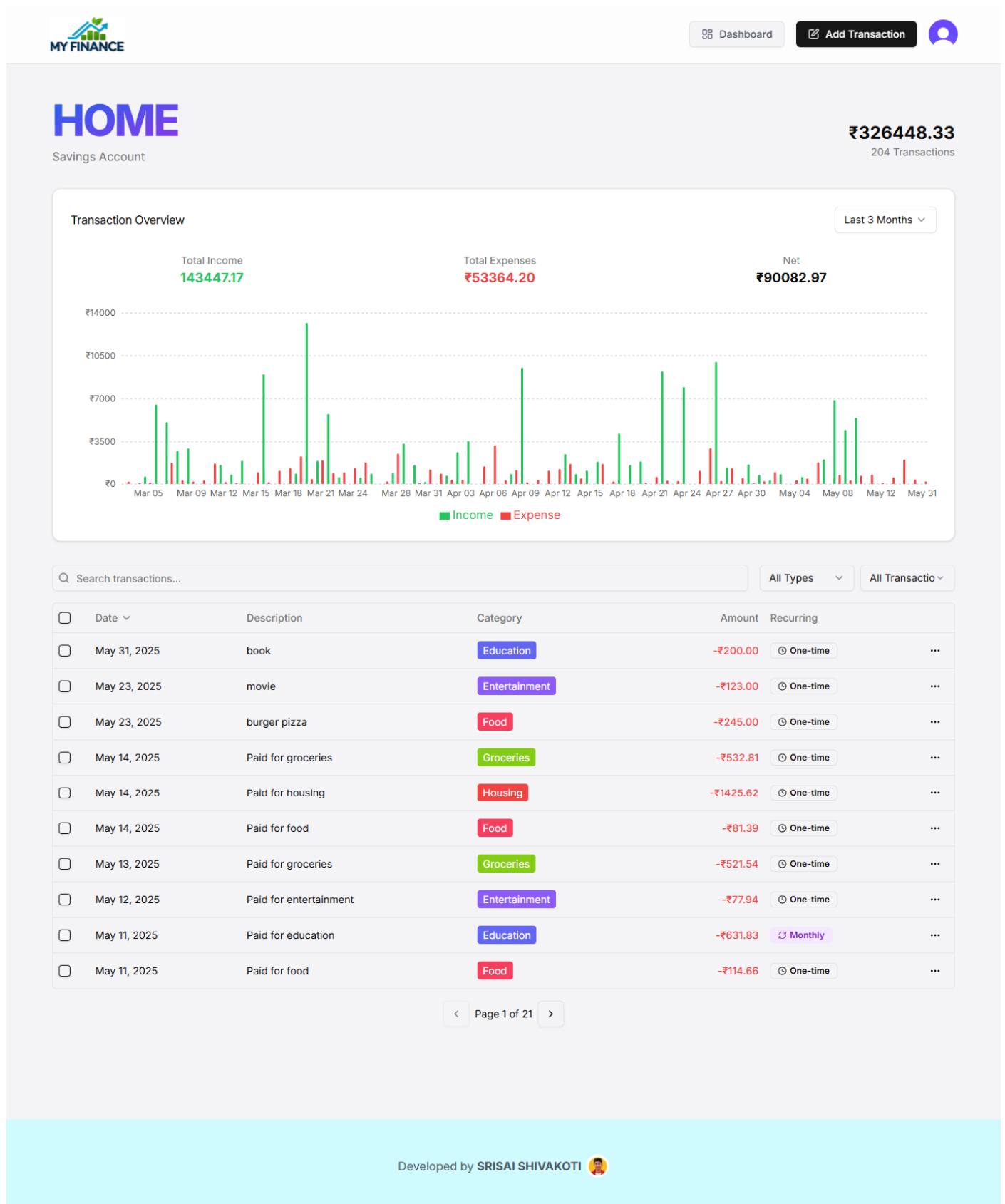
◆ 3. Dashboard with Budget & Analytics

After login, users are directed to this dashboard that shows a monthly budget progress bar, recent transactions, and a pie chart breakdown of expenses by category. This visual overview helps users quickly understand their financial status.



◆ 4. Detailed Transaction History & Overview

This screen shows the “Home” section with detailed transaction records, categorized by type, amount, date, and recurrence. A bar graph visualizes total income and expenses over time.



◆ 5. Add Transaction Interface with AI Receipt Scanner

This form allows users to manually add a transaction or upload a receipt to auto-fill fields using AI-powered OCR via the Gemini API. It includes inputs for amount, category, date, description, and recurrence toggle.

MY FINANCE

Dashboard Add Transaction

Add Transaction

Scan Receipt with AI

Type
Expense

Amount Account
840 HOME (₹326448.33)

Category
Select category

Date
October 18th, 2018

Description
Water Bottle, Crispy Chilli Baby Corn, Kashmiri Pulao, Kadai Chicken, Mutton Biryani, Soft Drinks

Recurring Transaction
Set up a recurring schedule for this transaction

Cancel Create Transaction

Developed by SRISAI SHIVAKOTI

11. CONCLUSION

The development of **MY_FINANCE**, an intelligent personal finance tracker, has successfully addressed common challenges in budgeting, expense tracking, and financial planning using a modern web-based solution. Built with cutting-edge technologies such as **React.js**, **Next.js**, **Supabase (PostgreSQL)**, **Clerk** for secure authentication, and **Gemini AI** for receipt parsing, the application offers a seamless and user-centric experience that promotes smarter financial decisions.

The project focused on solving real-world problems faced by individuals—particularly students and working professionals—who need a better way to manage their income, expenses, and savings. Unlike traditional tools such as manual logs or static spreadsheets, **MY_FINANCE** leverages automation, AI, and dynamic visualizations to make financial management engaging and proactive.

From the user dashboard to monthly reports, every component was designed with usability, accuracy, and performance in mind. Features like **receipt scanning**, **recurring transaction handling**, **category-based budgets**, and **interactive charts** help users gain deeper insights into their financial behavior. The responsive design ensures accessibility across devices, allowing users to track and plan anytime, anywhere.

In addition to the core development, this project emphasized good software engineering practices, including modular architecture, version control with GitHub, and comprehensive testing. Functional, integration, and UI tests were conducted to ensure quality and security across all modules.

Furthermore, the learning journey behind the project was as valuable as the application itself. As the primary developer, I explored and implemented several new technologies—particularly Supabase's row-level security, Clerk's session-based auth flows, and Gemini's AI integration—enhancing both technical proficiency and problem-solving skills. Team collaboration also played a vital role, with contributions in UML design, documentation, and presentation preparation strengthening the overall deliverable.

In conclusion, **MY_FINANCE** demonstrates how a well-structured, full-stack web application can empower users to better manage their personal finances using intelligent tools. The application not only fulfills its intended goals but also lays a strong foundation for future expansion—whether through mobile app integration, advanced analytics, or real-time spending alerts. It reflects a scalable and practical solution to a universal problem, proving both technically robust and socially relevant.

12. FUTURE ENHANCEMENTS

While the current version of **MY_FINANCE** provides a robust foundation for tracking income, expenses, budgets, and reports, several enhancements can further elevate its functionality, user experience, and scalability. The following features are planned or suggested for future iterations of the application:

◆ 1. Mobile App (iOS & Android)

Develop a cross-platform mobile application using React Native or Flutter to allow users to manage finances on the go. Real-time syncing with Supabase will ensure consistency across devices.

◆ 2. Expense Prediction with AI

Incorporate machine learning models to analyze past spending patterns and predict future expenses. This will help users make more informed budgeting decisions and prepare for recurring financial obligations.

◆ 3. Real-Time Notifications & Alerts

Enable push/email notifications for overspending alerts, low account balance warnings, bill payment reminders, and unusual transaction activity.

◆ 4. Multi-Currency Support

Add support for currency conversion to cater to users managing international finances or traveling. Exchange rates could be fetched via third-party APIs.

◆ 5. Bank Account Integration

Integrate with bank APIs (via open banking or Plaid-like services) to fetch transaction data automatically, reducing manual entry and improving accuracy.

◆ 6. Shared Budgets & Family Accounts

Introduce the ability to create shared budgets among family members or groups. Permissions can be configured to allow collaborative financial planning.

◆ 7. Advanced Reporting & Export

Provide downloadable reports (PDF/Excel) for selected time frames, categories, or accounts. Add filtering and custom date range options for greater flexibility.

◆ 8. Dark Mode & Accessibility Improvements

Enhance UI customization by offering theme toggles and improving accessibility features such as screen reader support, contrast adjustment, and keyboard navigation.

◆ 9. Investment & Savings Tracking

Expand functionality to track investments (stocks, mutual funds) and savings goals, offering a more complete view of financial health.

These future enhancements aim to make MY_FINANCE more powerful, intuitive, and inclusive. As user needs evolve, the project will continue to grow with scalable features and deeper AI integration to simplify money management for everyone.

13. REFERENCES

Supabase Documentation. [Online]. Available: <https://supabase.com/docs>

Clerk Authentication Docs. [Online]. Available: <https://clerk.dev/docs>

Google Gemini API for OCR. [Online]. Available: <https://ai.google.dev/gemini>

React.js Official Documentation. [Online]. Available:

<https://reactjs.org/docs/getting-started.html>

Next.js Documentation. [Online]. Available: <https://nextjs.org/docs>

PostgreSQL Database System. [Online]. Available: <https://www.postgresql.org/docs/>

Tailwind CSS Framework. [Online]. Available: <https://tailwindcss.com/docs>

Chart.js Library for Data Visualization. [Online]. Available: <https://www.chartjs.org/docs/>

GitHub Repository – MY_FINANCE Project. [Online]. Available:

https://github.com/Srisai16/MY_FINANCE

PlantUML Documentation (for UML diagrams). [Online]. Available: <https://plantuml.com/>

Mozilla Developer Network (MDN) Web Docs. [Online]. Available: <https://developer.mozilla.org/>

OpenAI. “Receipt Data Extraction Using Gemini Models.” Internal API Guide, 2024.