NEIL GOGTE INSTITUTE OF TECHNOLOGY

A Unit of Keshav Memorial Technical Education (KMTES) Approved by AICTE, New Delhi & Affiliated to Osmania University, Hyderabad

Computer Science and Engineering

Advanced Personal Finance Tracking and Analysis Platform

Submitted By

Batch No: C14

Kanchu Navyateja	**245322733154**
Gangula Venkata Raja Vineela	**245322733143**
Gunti Sai Pranitha	**245322733148**

Under the Supervision of

Dr. M. Mahendar

Assistant Professor ,CSE

AFFILIATED TO OSMANIA UNIVERSITY HYDERABAD

NEIL GOGTE INSTITUTE OF TECHNOLOGY

A Unit of Keshav Memorial Technical Education (KMTES) Approved by AICTE, New Delhi & Affiliated to Osmania University, Hyderabad

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Date:

CERTIFICATE

This is to certify that the Mini Project Report titled…….
Being submitted by Sri/Smt/Ms/Mr.....bearing HT. NO…… in partial fulfillment for the award of Bachelor of Engineering in Computer Science & Engineering to the Osmania University is a record of bonafide mini project work carried out by him/ her under our guidance and supervision.

The results embodied in this project work have not been submitted to any other University or Institute for the award of any degree or diploma.

Dr. M. MAHENDAR
Project Guide
Assistant Prof of CSE
Neil Gogte Institute of Technology

Dr. P. VAISHALI
Head of the Department
Dept. of CSE
Neil Gogte Institute of Technology

External Examiner

NEIL GOGTE INSTITUTE OF TECHNOLOGY

A Unit of Keshav Memorial Technical Education (KMTES) Approved by AICTE, New Delhi & Affiliated to Osmania University, Hyderabad

Date:

DECLARATION

I/We hereby declare that the Mini Project entitled, "**Advanced Personal Finance Tracking and Analysis Platform**" has not been in any case duplicated to submit to any other university for the award of any degree. To the best of my knowledge other than me, no one has submitted to any other university.

The Mini project is done in partial fulfillment of the requirements for the award of degree **BACHELOR OF ENGINEERING (COMPUTER SCIENCE AND ENGINEERING)** to be submitted as sixth semester Mini project as part of our curriculum.

Name and Signature of the Student

ACKNOWLEDGEMENT

I / We are also thankful to our Faculty Supervisor **Dr. M. Mahendar**, for his valuable guidance and encouragement given to us throughout the project work

I / We are also thankful to **Dr. T. Srinivas**, Project Coordinator, for supporting us from project selection till the submission to complete this project within the scheduled time.

I / We are also thankful to **Dr. P. Vaishali**, Heads of the Department for providing us with time to make this project a success within the given schedule.

I / We take this opportunity to thank all the people who have rendered their full support to our project work. We render our thanks **Prof. R. Shyam Sundar**, Principal, who encouraged us to do the Project.

I / We would like to thank the entire CSE Department faculty, who helped us directly or indirectly in the completion of the project.

I / We sincerely thank our friends and family for their constant motivation during the project work.

CONTENTS

ABSTRACT

The Finance Tracker System is a web-based application designed to simplify financial management for individuals, freelancers, and small businesses. It allows users to log transactions, set budgets, and track spending in real time, promoting financial awareness and discipline.

The frontend is built with React.js and styled using Material UI or Tailwind CSS for responsive design, while the backend uses Next.js and PostgreSQL for secure data storage. Clerk authentication ensures user data privacy.

Key features include automated transaction categorization, real-time budget tracking with alerts, and insightful data visualization through graphs and pie charts. The system also integrates AI to provide personalized financial analysis, email notifications, and task scheduling. Thorough testing ensures the system's reliability, security, and performance.

By offering accurate budgeting, simplified logging, and AI-driven insights, the Finance Tracker System empowers users to make better financial decisions and manage their finances effectively, making it a valuable tool for achieving financial goals.

Keywords: Personal Finance, Finance Tracker, Expense Tracking, Budget Monitoring, Web-Based, Tailwind CSS, React.js, Next.js, PostgreSQL, Clerk, Data Visualization, AI, Real-Time Alerts, Financial Awareness, Spending Habit.

LIST OF OUTPUT SCREENS

LIST OF TABLES

CHAPTER 1
INTRODUCTION

1.1 Purpose of the Project:

The primary purpose of the **Finance Tracker System** is to provide an efficient and user-friendly solution for managing personal and small-scale financial activities. In today's digital era, handling finances manually can be time-consuming, error-prone, and difficult to maintain. The system addresses this challenge by offering an automated platform that simplifies the process of recording transactions, tracking income, and monitoring expenses, reducing the complexity involved in traditional methods of financial management.

Another important purpose is to enhance financial awareness among users by providing real-time insights into their spending patterns. The Finance Tracker enables users to analyze where their money is going, identify unnecessary expenses, and make informed decisions about their financial habits. This visibility promotes better money management, allowing individuals to plan and prioritize their expenses effectively. With the help of interactive dashboards and data visualization, users can easily understand their financial status at a glance.

Lastly, the Finance Tracker aims to promote budgeting and savings while supporting long-term financial stability. By allowing users to set budgets, track progress, and receive alerts on overspending, the system encourages responsible financial behavior. Its accessibility through a web-based interface ensures convenience, enabling users to manage their finances anytime and anywhere. Overall, the Finance Tracker helps users avoid financial stress, stay within their budget, and achieve their financial goals efficiently.

1.2 Scope of the Project:

The Finance Tracker System is developed as a full-stack web application using the (Next.js ,Tailwind CSS, PostgreSQL, React.js) technology stack. It is primarily designed for individuals, students, professionals, and small business owners who need a simple yet efficient platform to manage their finances without the complexity of traditional accounting software.

The key features covered within the scope of the project include:

User authentication using Clerk for secure sign-up, login, and session management.

- Transaction management functionality, including adding, editing, and deleting transactions with details such as amount, category, date, and description.
- Budget creation and monitoring to help users control spending and achieve financial goals.
- AI-powered features, including:
    - Receipt scanning (Gemini API integration) for automated data entry.
    - Smart categorization of expenses to minimize manual effort.
- Analytics dashboard displaying:
    - Monthly income and expense summaries.
    - Category-wise spending insights.
    - Real-time account balance and trends.
- Search and filter options for quick access to specific transactions.
- Recurring transaction setup with automated reminders for bills and subscriptions.
- Monthly financial reports via email, summarizing user activity and providing spending insights.

The system is ideal for students, freelancers, and small businesses looking for a convenient, automated financial management solution. It is designed to be user-friendly, responsive, and scalable, with future enhancement possibilities like multi-currency support, investment tracking, and predictive analytics.

1.3 Problem with existing systems:

- Many finance management tools are **complex and not user-friendly**, requiring technical or accounting knowledge.
- They depend **only on manual entry**, with no automation features like receipt scanning or smart categorization.
- Lack of **real-time insights** for tracking expenses and budgets effectively.
- No **custom budgeting options** or alerts for overspending.
- **High cost** makes them unsuitable for individuals, students, and small businesses.
- Limited **mobile responsiveness** and poor user experience in existing solutions.

Furthermore, most existing solutions are **expensive and designed for large enterprises**, making them inaccessible for individuals, students, freelancers, or small business owners. Many

do not offer mobile responsiveness or an intuitive user interface, which limits convenience and usability. These challenges highlight the need for a simple, automated, and affordable solution like the **Finance Tracker System**, which addresses these gaps with AI-powered automation, real-time analytics, and a user-friendly design.

1.4 System Architecture:

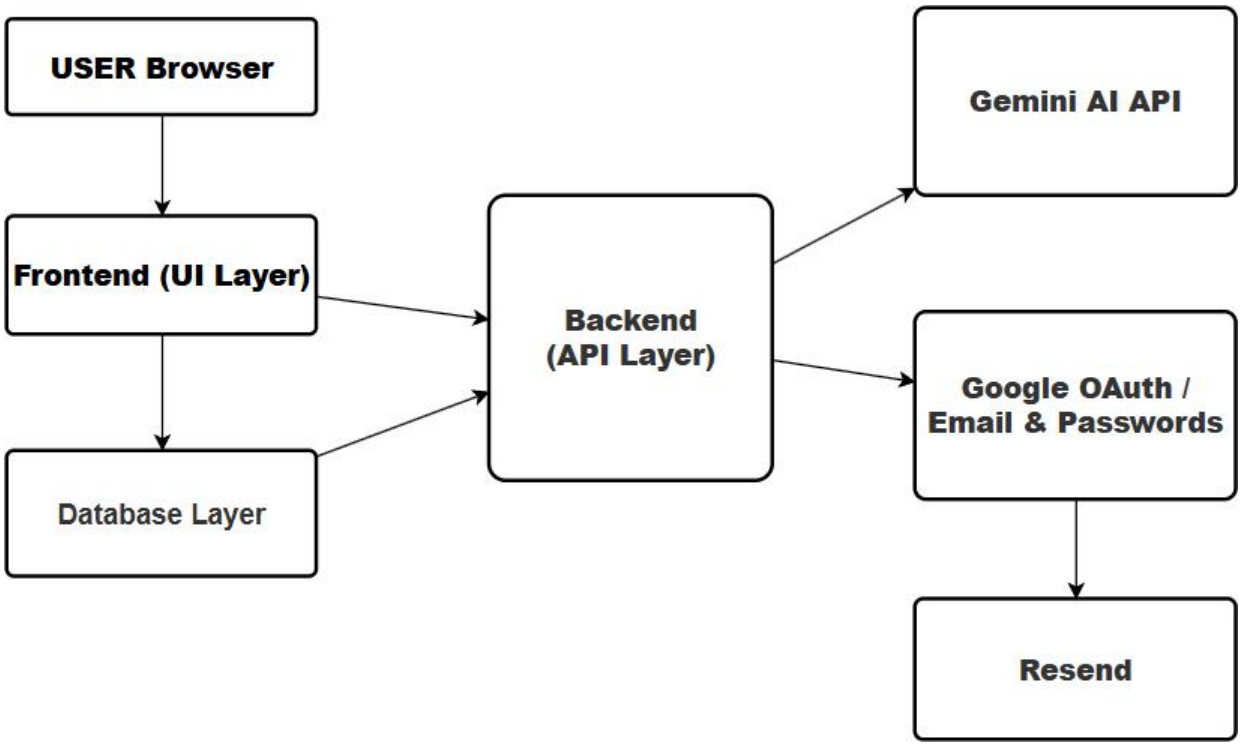

Fig 1.1 System Design

CHAPTER 2
LITERATURE SURVEY

2.1 Overview:

The AI Finance Platform is a full-stack web application designed to help users manage their finances efficiently. It allows users to track income, expenses, and subscriptions while providing powerful features like AI-powered receipt scanning and automated transaction categorization. Users can view detailed dashboards with charts and reports, manage multiple bank accounts, and even receive email notifications for alerts and summaries.

The app is built using Next.js, React, Tailwind CSS, and PostgreSQL for data management, along with Prisma ORM for database operations. It integrates Google Gemini AI for analysing receipts, Inngest for background tasks, Clerk for authentication, and Resend for email notifications. This combination of technologies creates a modern, secure, and intelligent financial management solution.

2.2 Literature Review:

Prof. Sushma A, Abdus Salaam I, Danda Ajith Kumar, Shishira M, Thanuja B (2025), International Journal of Research Publication and Reviews – "Finance Tracker System"

A comprehensive financial management tool built with Django and Python, designed to help users monitor income, expenses, and savings. The web application allows manual input of financial data, categorization of transactions, and generation of insightful reports

Aishwarya and Hemalatha (2024), SCITEPRESS-Science and technology Publications, Lda – "Smart Expense Tracking System Using Machine Learning"

This study presents an automated expense tracking system utilizing machine learning algorithms to predict personal expenses. The system collects data from multiple sources, including bank transactions and user input, and provides data visualization tools to help users understand their spending patterns.

Tihomir Stefanov, Milena Stefanova, Silviya Varbanova , Stanislav Temelkov(2024), TEM Journal – "Personal Finance Management Application"

This paper presents a personal finance management mobile application developed for the Android operating system. The application offers features like budget management, financial status reports, expense and income tracking, report generation, and visualization

through charts.

Lavesh Lingayat, Neha Yadav, Prajwal Rathod,Pranay Durutkar,Prof. Shilpa Ghode (2024), SSRN Electronic Journal –"Design and Implementation of Real-Time Expense Tracker Using Machine Learning"
This SSRN publication explains how ML algorithms can be used to predict future expenses and analyze financial behavior patterns. It validates its effectiveness with real user data and suggests integration with advanced analytics for better forecasting.

Samar Verma, Samarjeet Singh Kheda, Shivam Kuwale (2024), International Research Journal of Modernization in Engineering Technology and Science – "Personal Finance Tracker"
This paper discusses the development and implementation of a comprehensive web-based application designed to streamline financial management processes. The Personal Finance Tracker offers features tailored to address the diverse needs of modern-day financial planning.

Lee (2024), GitHub – "Finance Tracker - GitHub Repository"
A full-stack finance tracker built with React (TypeScript), Flask (Python), and SQLite. It includes income/expense tracking, report generation, trend visualization, and localization, offering a user-friendly interface and modular architecture.

S. **Bus(2023), MaxProg** – "What is Personal Finance?"
This article defines personal finance fundamentals and explores key principles of budgeting, saving, and managing expenditures. It provides theoretical support for the necessity of finance trackers in everyday decision-making.

Yu Xie(2016), ResearchGate – "The Device and Implementation of Personal Finance Management System Based on Android".A mobile application focused study developed using Java and .NET, incorporating features like currency conversion, chart-based reports, and voice assistance. The system supports visually impaired users and enables integration with existing web platforms.

Arish (GitHub) – "Finance Manager Project"
A practical GitHub project using Django and Python, allowing users to manage personal finances with budget alerts and financial summaries. It showcases real-world development for finance tracking.

Gupta (GitHub) – "MERN Expense Tracker Project"
This open-source MERN stack-based project implements full-stack expense tracking with MongoDB, Express, React, and Node.js. Features include real-time updates and secure login, highlighting full-stack best practices.

Sl.no	Author(s) & Year	Publication/ Source	Key Features	Type(Web /App)	Visualization / Report
1.	Prof. Sushma A, Abdus Salaam I, Danda Ajith Kumar, Shishira M, Thanuja B (2025)	International Journal of Research Publication and Reviews	Manual income/expense input, categorization, reports	Web (Django)	Simple Django-based UI
2.	Aishwarya and Hemalatha (2024)	SCITEPRESS – Science and Technology Publications, Lda	Predicts expenses, collects data from banks/users, visualizations	Web	Predictive analysis using ML
3.	Tihomir Stefanov, Milena Stefanova, Silviya Varbanova, Stanislav Temelkov (2024)	TEM Journal	Budget management, income/expense tracking, charts, reports	Mobile App (Android)	Native Android app

4.	Lavesh Lingayat, Neha Yadav, Prajwal Rathod, Pranay Durutkar, Prof. Shilpa Ghode (2024)	SSRN Electronic Journal	Real-time predictions, user behavior analysis	Web	Focus on real-time prediction by ml integration
5.	Samar Verma, Samarjeet Singh Kheda, Shivam Kuwale (2024)	International Research Journal of Modernization in Engineering Technology and Science	Comprehensive financial planning, modern-day finance needs	Web	Broad financial planning focus
6.	Lee (2024)	GitHub Repository	Full-stack tracker: React, Flask, SQLite, localization, modular UI	Web	Full-stack implementation
7.	S.Bus (2023)	MaxProg	Theoretical explanation of personal finance principles	None (Article)	Conceptual explanation
8.	Yu Xie (2016)	ResearchGate	Currency conversion, reports, voice assistance, accessibility features	Mobile App	Voice assistant, visually impaired support
9.	Arish (n.d.)	GitHub	Django project, budget alerts, financial summaries	Web	Open-source practical use
10.	Gupta (n.d.)	GitHub	MERN stack, real-time updates, secure login	Web	Real-time MERN integration

Table 2.1 Comparison Table

CHAPTER 3
SOFTWARE REQUIREMENT SPECIFICATION

The AI Finance Platform is a full-stack web application developed to simplify financial management for individuals and businesses. It offers a modern solution to track income, monitor expenses, and manage multiple bank accounts, all from a single dashboard. Built using Next.js and React for the frontend and PostgreSQL for the backend, the platform ensures efficiency, reliability, and scalability for users.

A key highlight of the system is its integration with Gemini AI, enabling features such as receipt scanning, automatic transaction categorization, and AI-generated monthly financial summaries. Users can set budgets, receive alerts when limits are approached or exceeded, and get actionable insights to improve financial decisions. Additionally, the platform provides email notifications for reports and reminders using secure APIs, ensuring timely updates for users.

The goal of this project is to deliver a secure, scalable, and intuitive solution that merges traditional finance tools with AI-powered automation. This document specifies the functional and non-functional requirements, ensuring that all stakeholders, developers, and testers have a clear understanding of the system's objectives and expected performance.

3.1 Functional Requirements:

User Authentication

- The system must allow users to register and log in using:
    - Google OAuth
    - Email/Password authentication.
- Sessions should be securely managed with tokens (JWT).

Dashboard

- Display an overview of:
  Total income
  Total expenses
  Remaining budget
- Provide interactive charts and graphs for analysis.

Transaction Management

- Add, update, delete transactions manually.
- Categorize transactions automatically using Gemini AI.

Receipt Scanning (AI Integration)

- Upload images of receipts.
- Extract transaction data using AI.
- Store processed details in the database.

Budget Management

- Users can set monthly/weekly budgets.
- Alerts when spending approaches or exceeds limits.

Financial Insights

- AI generates monthly summaries of spending habits.
- Suggestions for saving and better expense management.

Email Notifications

- Send monthly reports and budget alerts via Resend API.

Multi-Account Support

- Add multiple bank accounts.
- Track transactions across different accounts.

3.2 Non-Functional Requirements:

Performance

- The system should support 1,000 concurrent users without noticeable delay.
- Average API response time should be ≤ 300ms under normal load.
- Receipt processing via AI should take < 5 seconds.
- Scalability
- Must support horizontal scaling (adding more servers for more load).
- Use serverless deployment for APIs (Next.js on Vercel).
- Security
- All communication over HTTPS.
- Passwords stored using bcrypt with proper salting.
- JWT tokens for secure session management.
- Implement rate limiting (Arcjet) to prevent abuse.
- Regular security audits for vulnerabilities (SQL injection, XSS).
- Availability

- Maintain 99.9% uptime.
- Automatic failover for database using cloud providers (AWS RDS or Supabase).
- Daily backups for critical data.
- Usability
- Simple, intuitive interface with responsive design (mobile + desktop).
- Use shadcn/ui for consistency.
- Follow WCAG 2.1 accessibility guidelines for differently-abled users.
- Maintainability
- Modular, component-based architecture.
- Follow clean coding standards and best practices.
- Documentation for APIs and major components.
- Compatibility
- Support latest versions of Chrome, Firefox, Safari, and Edge.
- Mobile-friendly design for iOS and Android browsers.
- Reliability
- Error handling for API failures and AI service downtime.
- Logging for monitoring issues (use Sentry or similar).

3.3 Software Requirements:

1. **Frontend (UI Layer):**
    - Expense categorization
    - Generating monthly insights

3. **Backend (API Layer):**
    - Next.js API Routes for handling business logic
    - Prisma ORM for DB operations
    - Arcjet for rate-limiting & security
    - Inngest for background jobs (e.g., sending emails, AI summaries)

4. **Database Layer:**

PostgreSQL for storing:

- User details & authentication
- Transactions, budgets, and receipts
- AI-generated reports

5. **Authentication & Notifications:**

- Google OAuth / Email & Password for authentication
- Resend for sending email notifications (AI-generated monthly summaries)

6. Development Tools:

- Code Editor: Visual Studio Code
- Version Control: Git
- Repository Hosting: GitHub

3.4 Hardware Requirements:

Minimum Requirements (For Development)

- Processor (CPU): Intel i5 (8th Gen) / AMD Ryzen 5 or equivalent
- RAM: 8 GB
- Storage: 256 GB SSD (for faster build times and database operations)
- Graphics: Integrated GPU (no heavy GPU required unless you process AI locally)
- OS: Windows 10 / macOS / Linux
- Internet: Stable broadband for API calls and package installations

Recommended Requirements (For Smooth Development & Testing)

- Processor (CPU): Intel i7 (10th Gen or above) / AMD Ryzen 7
- RAM: 16 GB
- Storage: 512 GB SSD
- Graphics: Integrated or entry-level discrete GPU (not mandatory)
- OS: Windows 11 / macOS Monterey or later
- Internet: High-speed connection (for large npm/yarn packages and AI API integration)

For Deployment (Production Server)

- CPU: Quad-core processor or higher (e.g., Intel Xeon or AMD EPYC for servers)
- RAM: 8–16 GB (depending on user load)
- Storage: 50 GB SSD (PostgreSQL + logs)
- Hosting: Vercel (for Next.js frontend) + Render / Railway / AWS EC2 (for backend & database)
- Database: PostgreSQL on Supabase, Neon, or RDS (AWS)

If AI Processing is Local (Not API-based)

- GPU: NVIDIA RTX 3060 or higher (6–8 GB VRAM) for running AI models
- RAM: 32 GB recommended for local inference
- Storage: 1 TB SSD for model weights and dataset

CHAPTER 4

SYSTEM DESIGNS

4.1 Use Case Diagram:

Actors:

1. User

   The primary actor who interacts with the system to manage finances.

2. AI System (Gemini)
   - An external system used for receipt scanning, transaction categorization, and generating AI-based summaries.
3. Email Service
   - An external service used for sending financial reports and notifications (e.g., via Resend).

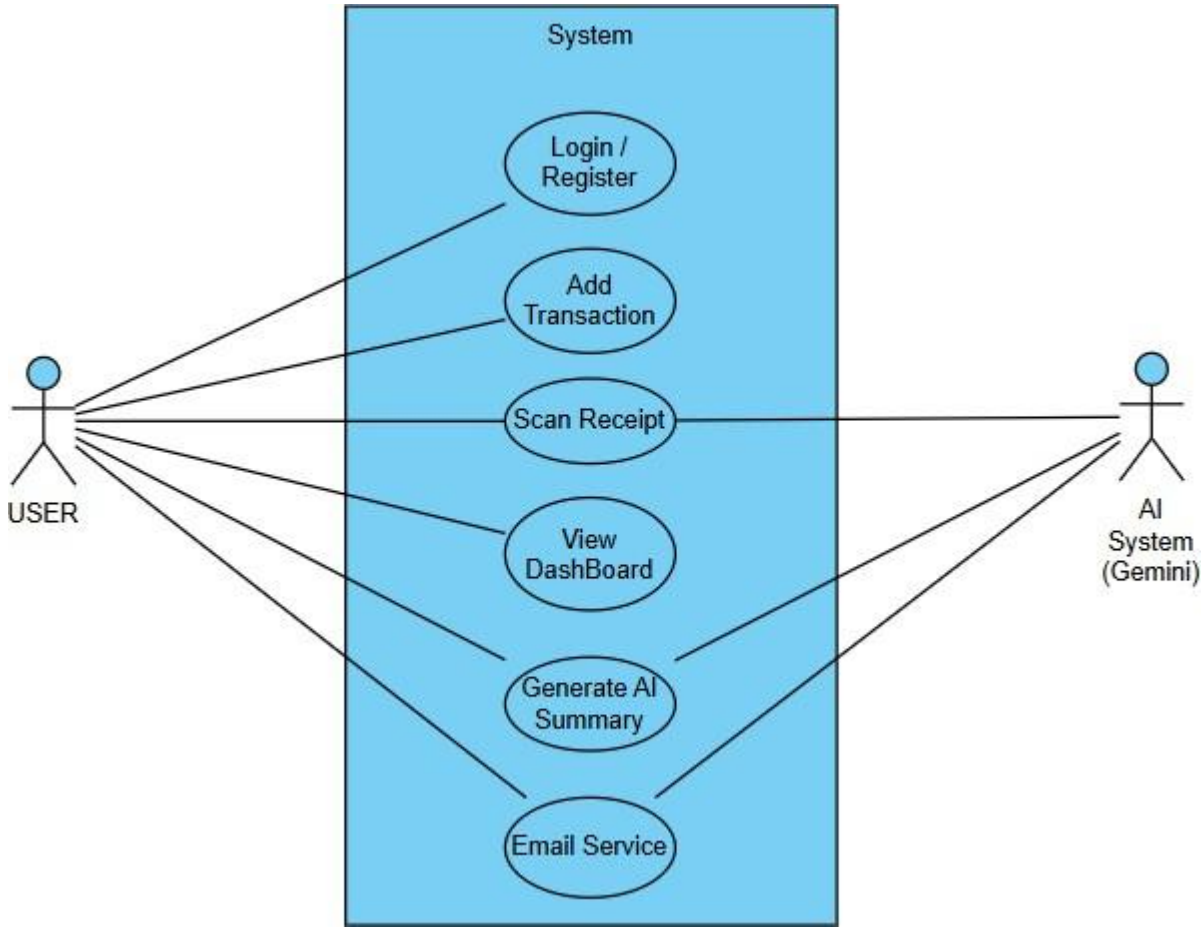

Fig 4.1 Use Case Diagram

Use Cases

1. Login / Register
    - The user can register for a new account or log in using credentials or Google OAuth.
2. Add Transaction
    - The user can add transactions manually, which will be stored in the database.
    - The system may interact with AI for categorization.
3. Scan Receipt
    - The user uploads a receipt image.
    - The system calls Gemini AI to extract data and categorize the expense.
4. View Dashboard
    - The user views the summary of income, expenses, budgets, and AI-generated insights.
5. Generate AI Summary
    - The system generates monthly reports using AI.
    - It may interact with the Email Service to send these summaries to the user.

Relationships

- User → AI Finance Platform: The user initiates all main actions.
- AI Finance Platform → AI System (Gemini): For receipt scanning and AI-based reports.
- AI Finance Platform → AI System (Gemini): To send notifications and monthly summaries

4.2 Class Diagram:

The class diagram for the AI Finance Platform illustrates the fundamental building blocks and their relationships within the system. The **User** class acts as the core entity, storing essential details like user ID, name, and email. Each user can connect multiple **BankAccount** objects, enabling them to manage transactions from various sources in one platform. These accounts are linked to the **Transaction** class, which stores individual financial records and includes methods for fetching and categorizing transactions into income, expenses, or subscriptions. Additionally, users can upload **Receipt** objects, which are processed to extract structured transaction details

AI capabilities are introduced through the **GeminiAI** class, which analyzes receipts to extract transaction information using advanced AI models. For users who have recurring payments,

such as monthly subscriptions or bills, the **RecurringTransaction** class stores these repeated payment details. To manage time-based operations, the **Inngest** class is used to schedule background jobs like recurring transaction execution via cron jobs, ensuring automation without user intervention.

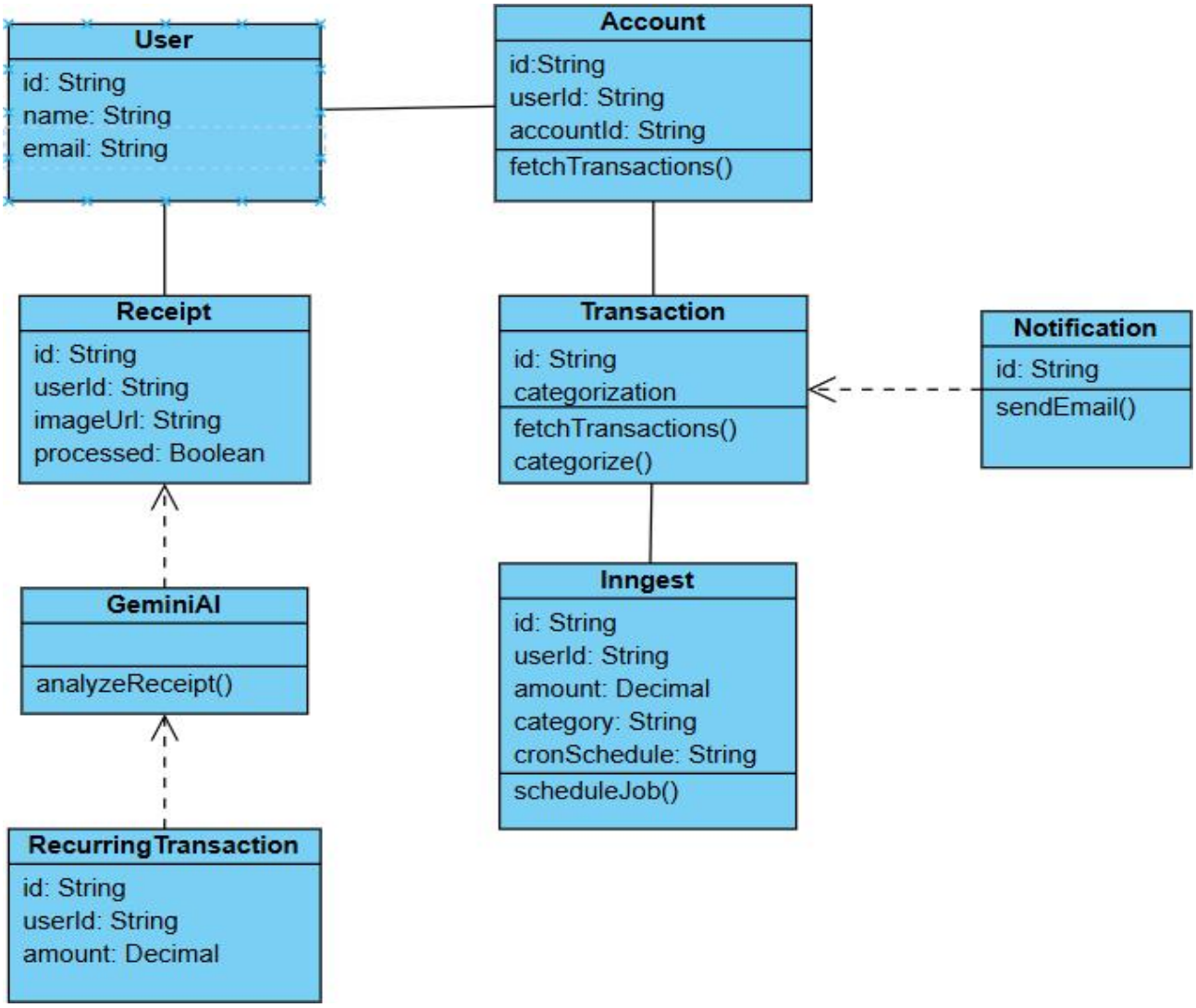

Fig 4.2 Class Diagram

The **Notification** class plays a critical role in user communication by sending email alerts and financial summaries. Together, these classes form a connected ecosystem where each component performs a specific function while integrating seamlessly with others. This modular structure ensures scalability, maintainability, and efficiency, allowing the platform to deliver real-time analytics, AI-driven insights, and a smooth user experience.

4.3 Sequence Diagram:

It is a Sequence Diagram for the AI Finance Platform, showing how different components interact over time to handle user actions like logging transactions, uploading receipts, and sending notifications.

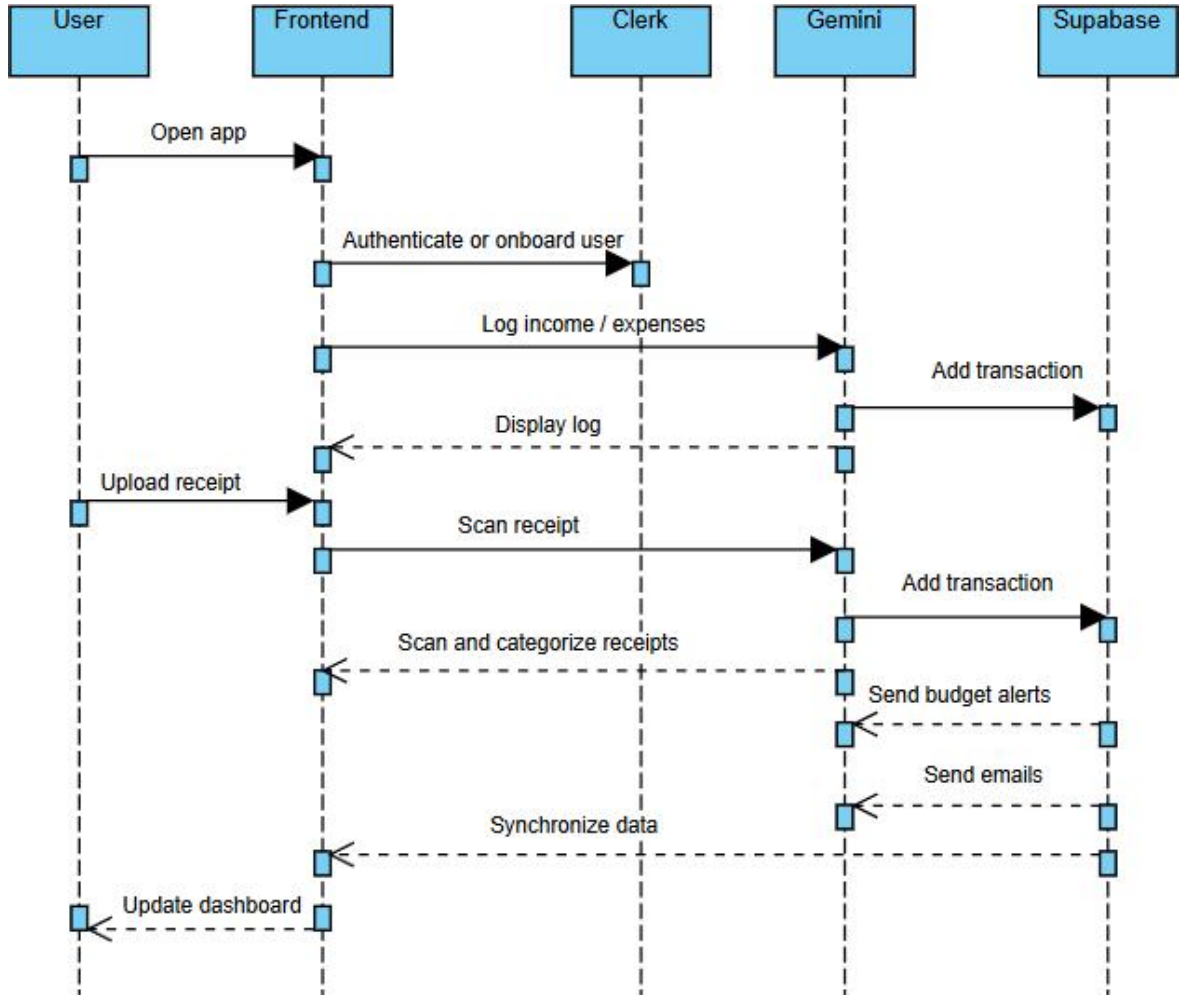

Fig. 4.3 Sequence diagram

1. Authentication and Access
   - The sequence starts when the User opens the app.
   - The Front-end communicates with Clerk to authenticate or onboard the user and retrieve their profile.
   - After successful authentication, Clerk grants access, and the system is ready for further actions.
2. Transaction Handling (Manual and AI-based)
   - For manual entry, the user logs an expense/income, which creates a transaction record.

- The Front-end sends this to the backend, and the data is written to the database through Prisma ORM and PostgreSQL.
- For receipt upload, the user uploads an image, which is scanned by Gemini AI to extract structured transaction data.
- This extracted data is then used to create a transaction record, again stored via Prisma ORM in PostgreSQL.

3. Background Jobs and Notifications

- Inngest schedules background jobs such as recurring transaction entries, generating monthly summaries, and triggering AI-driven reports.
- Resend sends real-time alerts (like budget threshold notifications) and AI-generated monthly reports to the user via email.
- Supabase is responsible for storing and syncing user data to keep the app updated in real-time.
- Finally, the Front-end updates the dashboard and displays notifications to the user.

4.4 Activity Diagram:

1. Start

   The process begins when the user opens the application.
2. User Logs In
   - The user enters credentials (email/password or Google OAuth).
   - Decision: If credentials are valid → proceed to dashboard; else → show error.
3. Load Dashboard
   - The dashboard displays income, expenses, budgets, bar graphs, and pie charts.
4. User Chooses an Action
   - Three main options are available:
     - Add Transaction
     - Upload Receipt
     - Generate Report

Option 1: Add Transaction

- Open Transaction Form.
- Validate the entered details (amount, category, description).
- Decision: If valid → save to database and update dashboard; else → show error

message.

Option 2: Upload Receipt

- User uploads an image of a receipt.
- The system sends the image to Gemini AI.
- AI extracts transaction details and categorizes them automatically.
- Save the transaction to the database and update the dashboard.

Option 3: Generate Report

- The system prepares a monthly financial summary.
- Sends the report to the user's email via Resend API.

5. End

   The process ends after completing any of the selected actions.

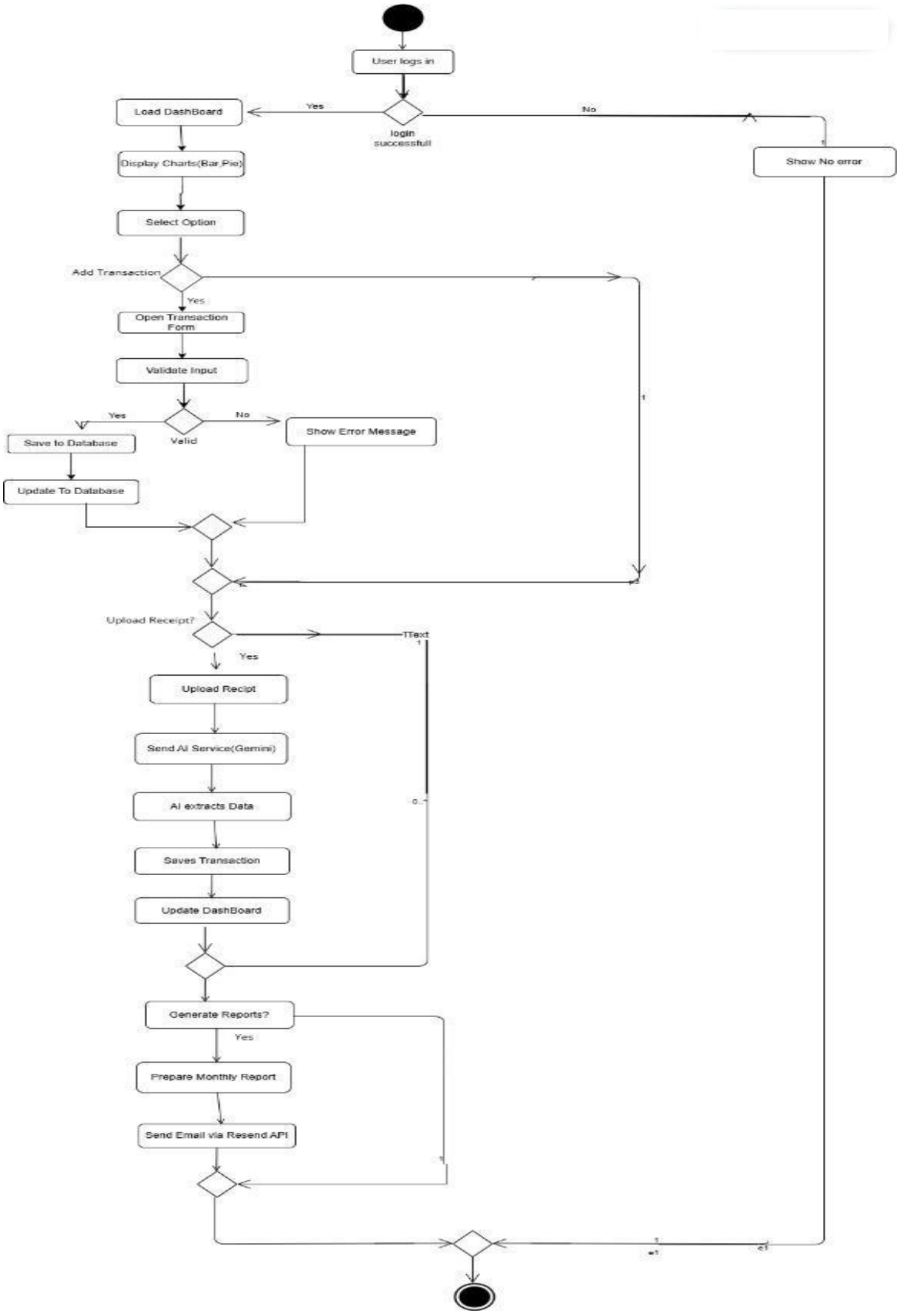

Fig 4.4 Activity Diagram

4.5 Deployment diagram:

The deployment diagram illustrates how the system's components are distributed across different hardware and services in the production environment. The architecture consists of the following nodes and their interactions:

1. User Device (Client Layer)
    - Represents the end-user accessing the application via a web browser on a desktop or mobile device.
    - Communicates with the application over HTTPS for security.
2. Hosting Server (Application Layer)
    - Hosts the Next.js application, which includes both the frontend UI and API routes (backend logic).
    - Handles user requests, serves web pages, and processes API calls for transactions, reports, and dashboards.
3. Database Server (Data Layer)
    - A PostgreSQL database hosted on Supabase or AWS RDS stores all application data, including user details, transactions, budgets, and AI reports.
    - The backend communicates with this database via secure SQL queries.
4. AI Service (Gemini API)
    - A third-party AI service responsible for receipt scanning and transaction categorization.
    - The backend sends receipt images to Gemini AI and receives structured data for storage.
5. Email Service (Resend API)
    - Responsible for sending email notifications and monthly financial summaries to users.
    - The backend triggers this service for email delivery after report generation.
6. Connections
    - All communications between nodes occur over HTTPS to ensure security.
    - User → (App) → Database, AI Service, Email Service.

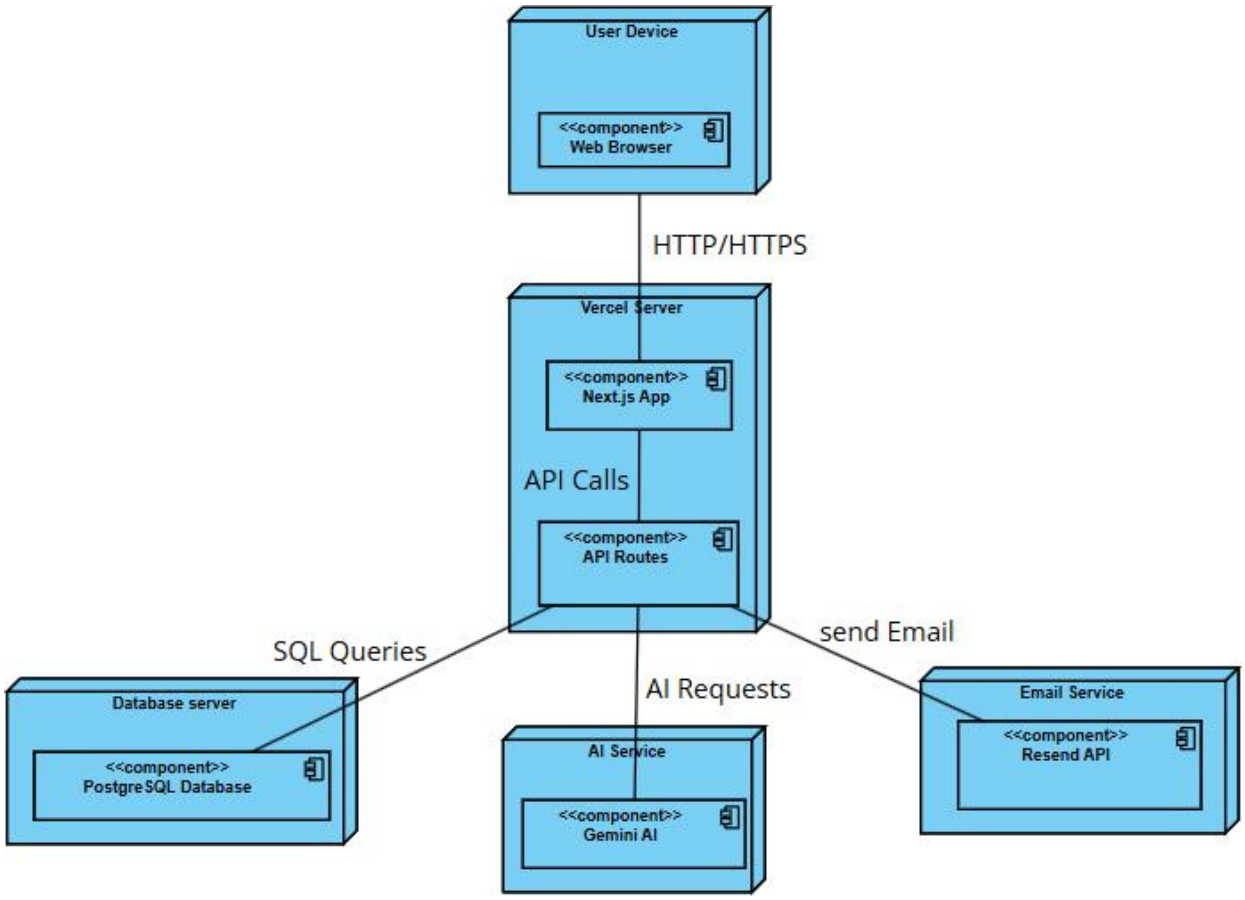

Fig 4.5 Deployment Diagram

CHAPTER 5
IMPLEMENTATION

Sample code:

Receipt-scanner code:

```
FUNCTION ReceiptScanner TAKES onScanComplete (a function to call after scanning)
AS PROP

  DECLARE fileInputRef AS a reference to the file input element (initially null)

  DESTRUCTURE useFetch(scanReceipt) INTO:
    - loading status as scanReceiptLoading
    - function to trigger the fetch as scanReceiptFn
    - scanned result data as scannedData

  FUNCTION handleReceiptScan TAKES file AS PARAMETER
    IF file size is greater than 5MB (5 * 1024 * 1024 bytes)
      SHOW error toast "File size should be less than 5MB"
      RETURN (stop the function)

    CALL scanReceiptFn with the file (this triggers the backend scan)
  END FUNCTION

  useEffect (on change of scanReceiptLoading or scannedData):
    IF scannedData exists AND scanReceiptLoading is false
      CALL onScanComplete function with scannedData
      SHOW success toast "Receipt scanned successfully"
  END useEffect

  RETURN JSX:
    DIV (with flex layout and gap)
```

```
    INPUT of type "file"
      - referenced by fileInputRef
      - hidden from view (className = "hidden")
      - accept only image files (accept="image/*")
      - enable camera capture from environment (capture="environment")
      - onChange EVENT:
        GET the first file from input
        IF file exists
          CALL handleReceiptScan(file)

    BUTTON
      - type: button
      - variant: outline
      - width and height: w-full, h-10
      - background gradient with animation
      - text color: white
      - onClick EVENT: trigger fileInputRef to click (open file dialog)
      - disabled IF scanReceiptLoading is true

      IF scanReceiptLoading is true:
        SHOW spinning Loader2 icon
        SHOW text: "Scanning Receipt..."

      ELSE:
        SHOW Camera icon
        SHOW text: "Scan Receipt with AI"

  END DIV
END FUNCTION
```

Transaction Overview code:

```
FUNCTION DashboardOverview(accounts, transactions):

  INITIALIZE selectedAccountId WITH:
    ID of account that has isDefault == true
    IF no default, use first account's ID (accounts[0]?.id)

   SET accountTransactions = FILTER transactions WHERE transaction.accountId ==
selectedAccountId

    SET recentTransactions = SORT accountTransactions by transaction.date descending
  SLICE first 5 items

  SET currentDate = Current Date
  SET currentMonthExpenses = FILTER accountTransactions WHERE:
    transaction.type == "EXPENSE"
    AND transaction.date.month == currentDate.month
    AND transaction.date.year == currentDate.year

  INITIALIZE expensesByCategory as empty map
  FOR each transaction IN currentMonthExpenses:
    IF expensesByCategory does not contain transaction.category:
      SET expensesByCategory[transaction.category] = 0
    ADD transaction.amount to expensesByCategory[transaction.category]

   SET pieChartData = MAP over Object.entries(expensesByCategory):
    RETURN { name: category, value: amount }

   RETURN JSX:

    DIV (grid layout: 2 columns, gap-4):
      CARD:
        CardHeader:
```

```
  CardTitle: "Recent Transactions"
  Select Dropdown:
    value = selectedAccountId
    onValueChange = setSelectedAccountId
    SelectTrigger: "Select account"
    SelectContent:
      FOR each account IN accounts:
        SelectItem: account.name with value = account.id

CardContent:
  IF recentTransactions is empty:
    SHOW message: "No recent transactions"
  ELSE:
    FOR each transaction IN recentTransactions:
      SHOW transaction:
        - description or "Untitled Transaction"
        - formatted date (e.g., "Jul 26, 2025")
        - amount with:
          IF transaction.type == "EXPENSE":
          ArrowDownRight icon, red text
          ELSE:
            ArrowUpRight icon, green text
          VALUE: formatted amount (e.g., "$145.99")

CARD:
  CardHeader:
    CardTitle: "Monthly Expense Breakdown"

  CardContent:
    IF pieChartData is empty:
      SHOW message: "No expenses this month"
    ELSE:
      ResponsiveContainer (height: 300px):
        PieChart:
```

Pie:
data = pieChartData
position: center
outerRadius = 80
fill = "#8884d8"
dataKey = "value"
label = name and formatted value
FOR each entry IN pieChartData:
Cell:
fill color from COLORS array (cyclic using index)

Tooltip:
Format value as `$amount.toFixed(2)`
Styled using theme variables

Legend:
Show category legend automatically

Frontend:

Transaction-form:

Function AddTransactionForm(inputs: accounts, categories, editMode, initialData)

// Initialize routing and URL utilities
Get router from useRouter()
Get URL parameters (accountId) from useParams()

// Initialize form with schema validation and default values
Initialize form using useForm with zodResolver and defaultValues:
type: 'EXPENSE'
date: current date
isRecurring: false
accountId: accountId from URL
categoryId: "

description: ''
interval: 'DAILY'
amount: ''

Watch type, isRecurring, and date fields from the form

If editMode is true
  Use useFetch hook with updateTransaction function
Else
  Use useFetch hook with createTransaction function

Define Function onSubmit(formData)
  Convert amount to float
  If editMode is true
    Call updateTransaction function with transactionId and formData
  Else
    Call createTransaction function with formData

Define Function handleScanComplete(scannedData)
  If scannedData exists:
    Set form value for amount using scannedData.amount
    Set form value for categoryId using scannedData.categoryId
    Set form value for description using scannedData.description
    Show success toast "Fields auto-filled from receipt"

useEffect when transactionData is updated and loading is false:
  If transactionData exists
    If editMode is true
      Show toast "Transaction updated"
    Else
      Show toast "Transaction created"
    Reset form
    Redirect to account page using router.push(`/dashboard/accounts/${accountId}`)

Filter categoryList from categories where category.type equals type

Return a Form component using react-hook-form:

Inside Form:
If not in editMode:
Render ReceiptScanner component
Pass handleScanComplete function as onScanComplete

Render Select Field for Transaction Type (EXPENSE / INCOME)

Render Input Field for Amount
Type: number
Step: 0.01
Placeholder: "Enter amount"

Render Select Field for Account
Populate options from accounts
If no accounts exist:
Show message with link to create account

Render Select Field for Category
Populate options from filtered categoryList

Render Calendar Date Picker
Set selected value from form
Update value in form on date change

Render Input Field for Description (optional)

Render Toggle Switch for "Recurring Transaction"
If enabled:
Render Select Field for Recurring Interval (DAILY, WEEKLY, MONTHLY, YEARLY)

Render Cancel and Submit Buttons

Cancel Button navigates back to previous page

Submit Button shows loader if submitting

Text: "Saving..." or "Save Transaction"

Account-chart (Bar-graphs, Pie-charts):

FUNCTION AccountChart TAKES a prop called "transactions" (an array of transaction objects)

DEFINE a constant DATE_RANGES as an object mapping keys to range definitions:

- "7D": Last 7 Days, 7 days
- "1M": Last Month, 30 days
- "3M": Last 3 Months, 90 days
- "6M": Last 6 Months, 180 days
- "ALL": All Time, null (no limit)

DECLARE dateRange state variable

- INITIALIZED with default value "1M"

DECLARE filteredData AS a memoized value (using useMemo), which updates when either transactions OR dateRange changes

INSIDE useMemo:

GET the date range object from DATE_RANGES using dateRange as key

GET current date as now

COMPUTE startDate:

IF range has a defined number of days

startDate = beginning of the day for (now - range.days)

ELSE (if "ALL" is selected)

startDate = beginning of Unix epoch (new Date(0))

FILTER transactions where:

transaction.date is >= startDate AND <= end of today
GROUP the filtered transactions by formatted date (e.g., "Jul 25")
INITIALIZE an empty object called acc

FOR EACH transaction in the filtered list:
FORMAT transaction date into string (e.g., "Jul 25")

IF acc[formatted date] does not exist:
CREATE acc[date] with fields:
- date
- income: 0
- expense: 0

IF transaction type is "INCOME":
INCREMENT acc[date].income by transaction.amount
ELSE (i.e., EXPENSE):
INCREMENT acc[date].expense by transaction.amount

RETURN the values of acc object as an array, sorted by actual date

END useMemo for filteredData

DECLARE totals AS a memoized value (using useMemo), which updates when filteredData changes

REDUCE filteredData array into totals object:
- Start with { income: 0, expense: 0 }
- FOR EACH day in filteredData:
- Add day.income to total.income
- Add day.expense to total.expense

RETURN totals object

RENDER the component JSX

WRAP entire component in a <Card>
INSIDE Card:

RENDER CardHeader:
- Flex row layout, justified between, with spacing and no bottom padding
- Show CardTitle as "Transaction Overview"
- Render a <Select> component to choose the date range
  - defaultValue = dateRange
  - onValueChange = setDateRange
  - Trigger element has width 140px
  - Inside SelectContent:
    - For each entry in DATE_RANGES:
      - Render a SelectItem with key and label

RENDER CardContent:
- DIV with flex layout and justified content for financial summary
  - First block: Total Income
    - Label: "Total Income"
    - Value: formatted total income (green)
  - Second block: Total Expenses
    - Label: "Total Expenses"
    - Value: formatted total expenses (red)
  - Third block: Net
    - Label: "Net"
    - Value: total.income - total.expense
      - IF result >= 0, green
      - ELSE, red

- DIV with height 300px, wrapping a ResponsiveContainer
  INSIDE ResponsiveContainer:
    RENDER a BarChart:
      - data = filteredData
      - margin = { top: 10, right: 10, left: 10, bottom: 0 }

INSIDE BarChart:

- CartesianGrid with dashed horizontal lines
- XAxis:
  - dataKey = "date"
  - font size = 12
  - no tickLine or axisLine
- YAxis:
  - font size = 12
  - no tickLine or axisLine
  - format ticks as dollars
- Tooltip:
  - Format value as dollars
  - Custom style with popover color, border, and rounded edges
- Legend (shows "Income" and "Expense" labels)
- Bar for income:
  - dataKey = "income"
  - green fill
  - rounded top corners
- Bar for expense:
  - dataKey = "expense"
  - red fill
  - rounded top corners

END FUNCTION

CHAPTER 6
RESULTS

Output screens:

Transaction form:

The transaction form allows users to add, edit, or delete income and expense records. It supports manual entry as well as AI-categorized transactions. Data is stored securely in the **PostgreSQL database**

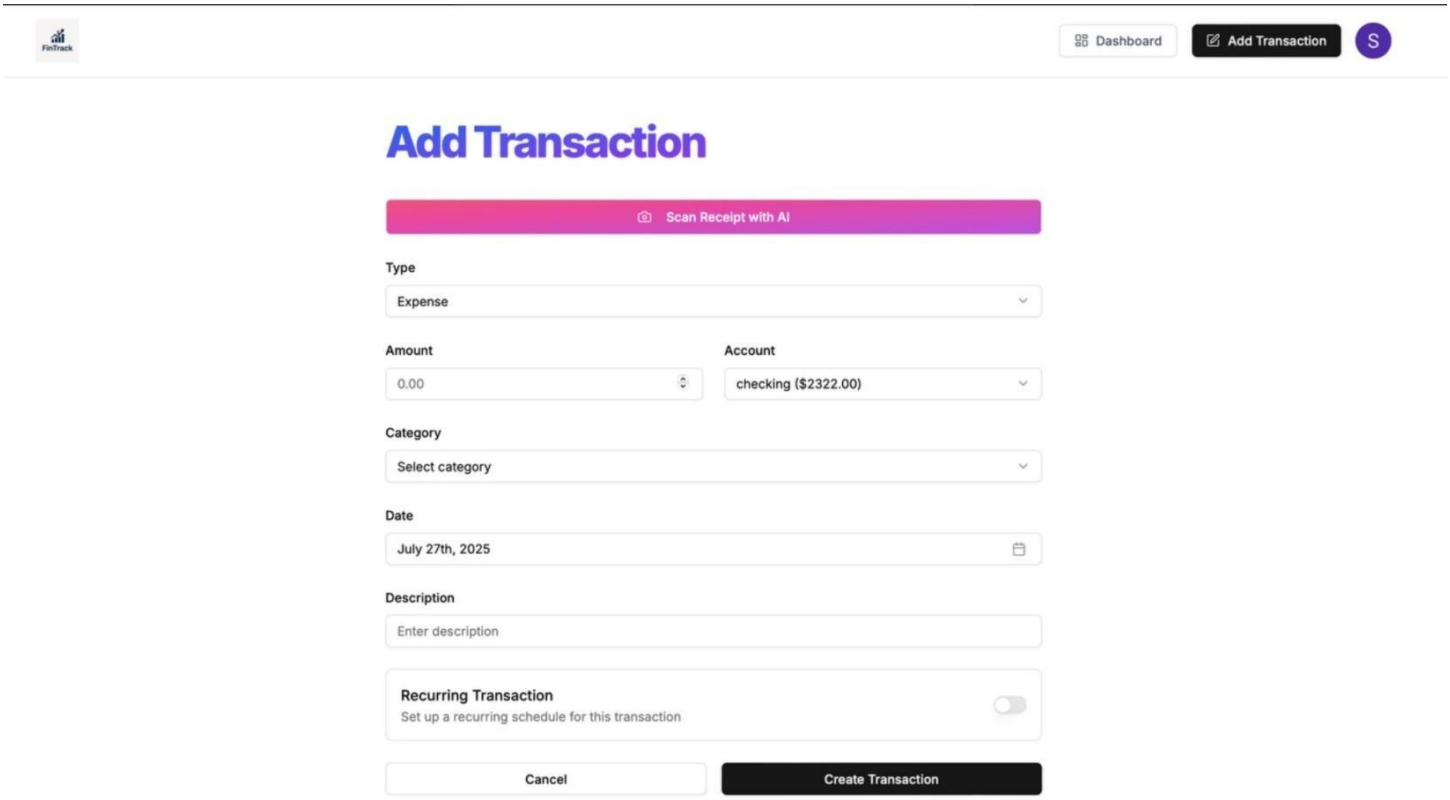

Fig 6.1 Transaction form

The below screen shows the **Checking Account Dashboard** displaying:

- A **bar chart** comparing income (green) and expenses (red) over time.
- A **transaction list** with date, description, category, amount, and recurring status.

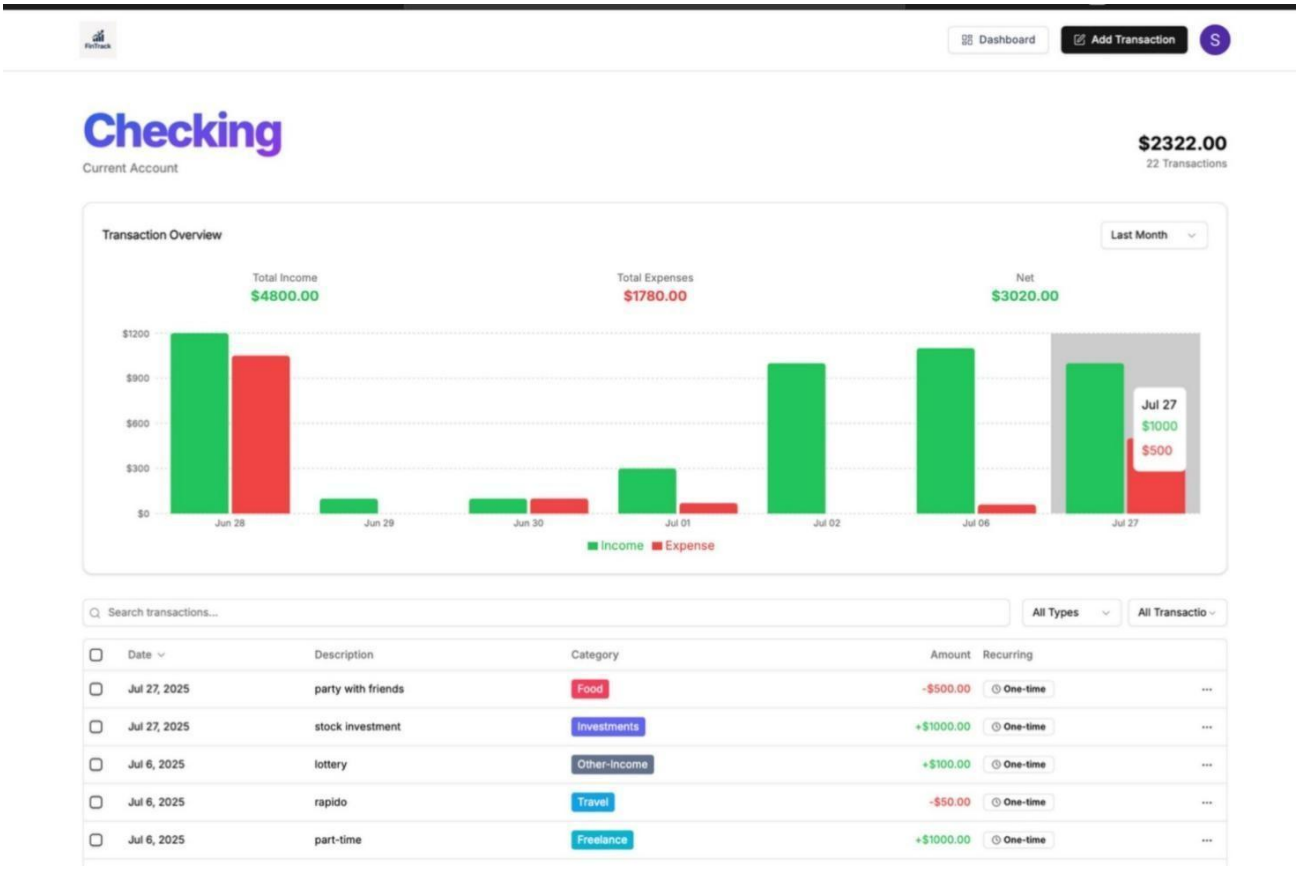

Fig 6.2 Checking Account Dashboard

Landing page:

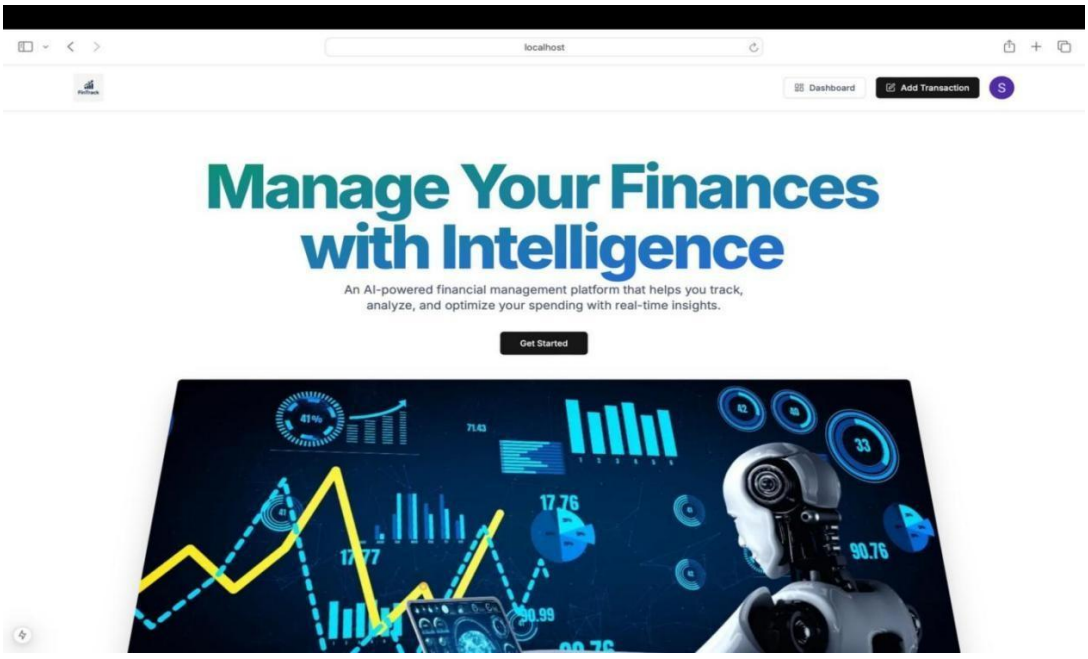

Fig 6.3 Landing page

Account details:

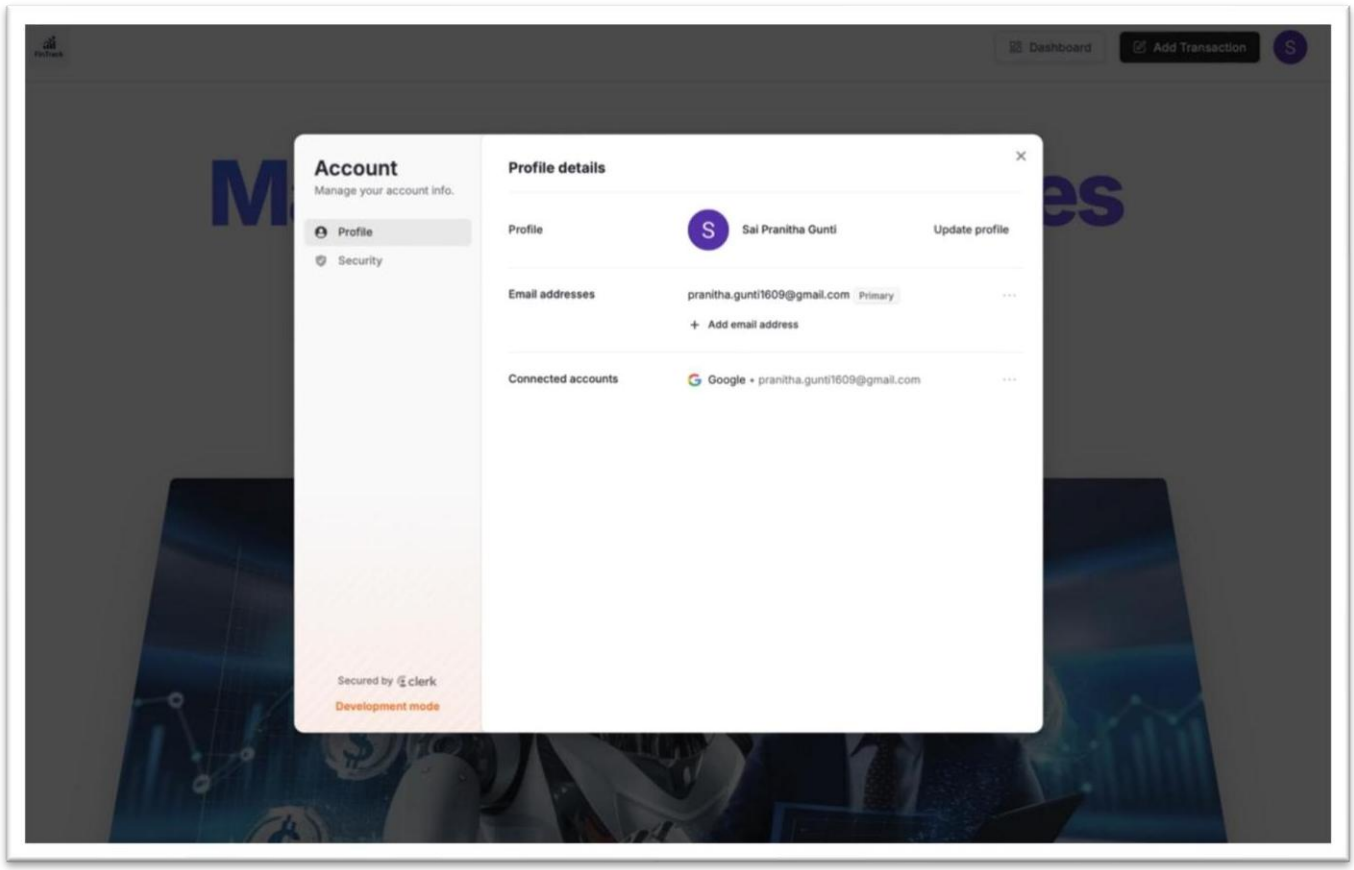

Fig 6.4 Account details

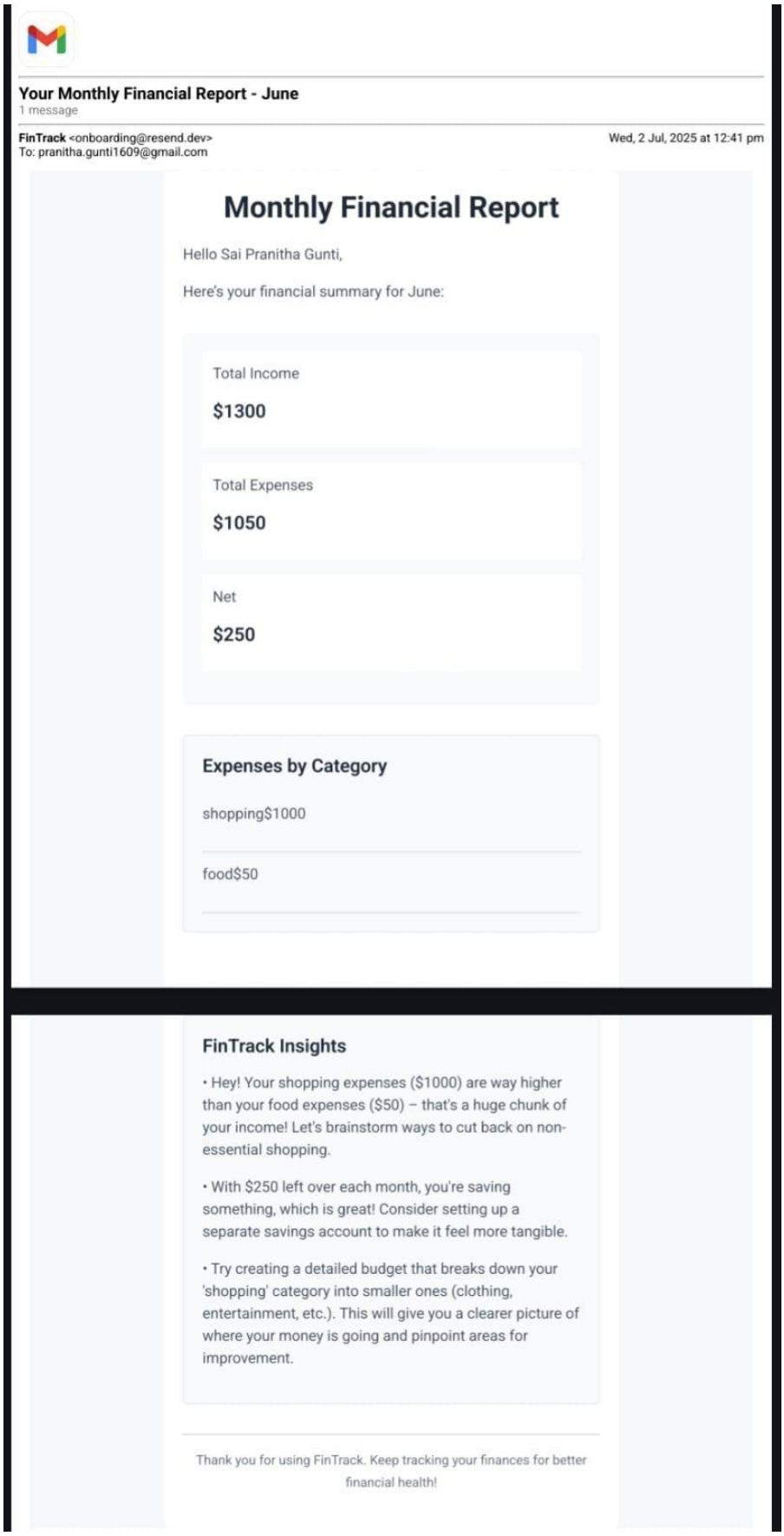

Fig 6.5 Financial Report

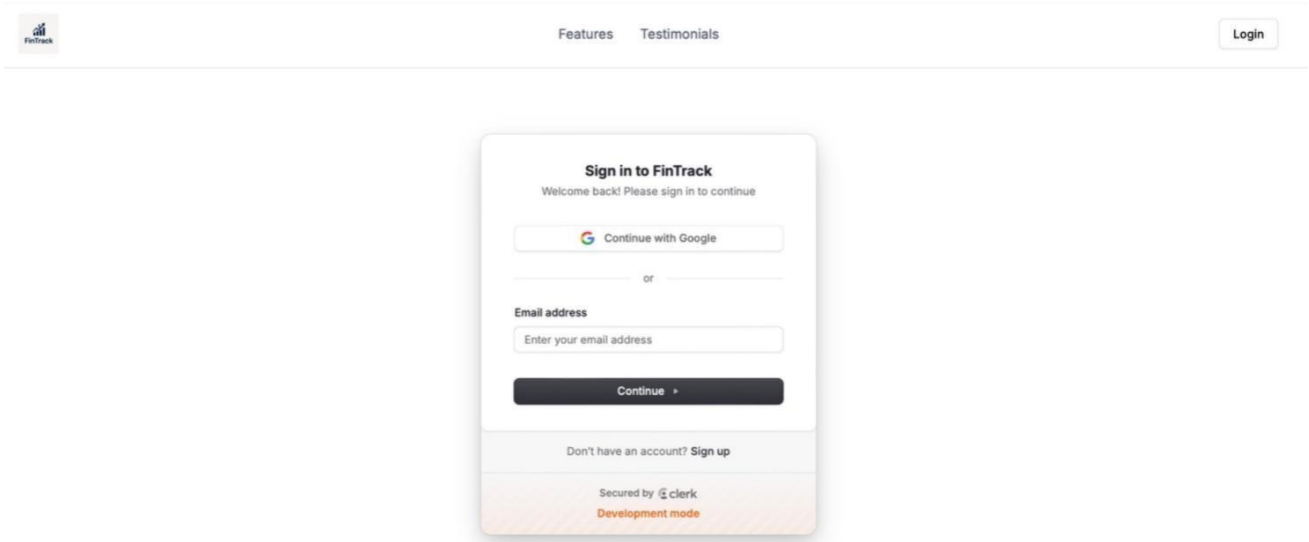

Fig 6.6 Sign-in Page

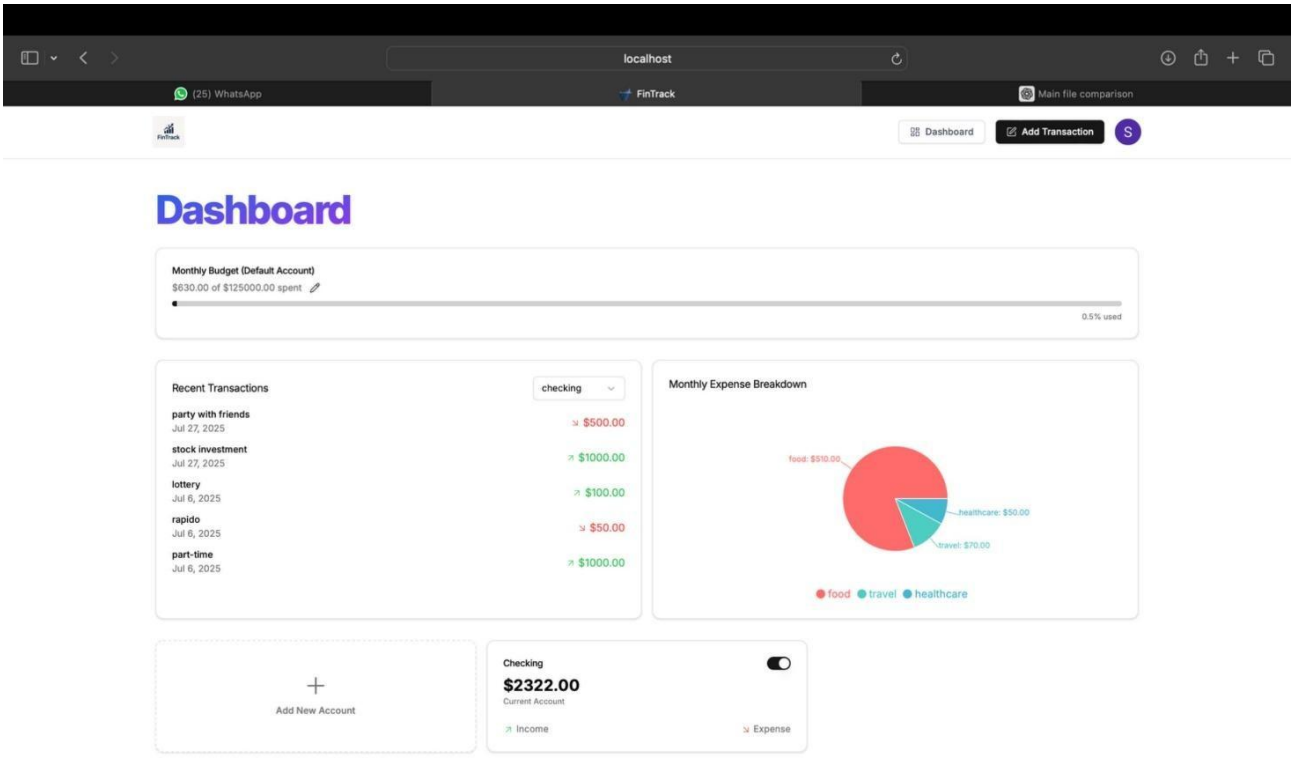

Fig 6.7 Dashboard

CHAPTER 7
TEST CASES

Test cases are designed to verify that each feature of the AI Finance Platform works as expected under different conditions. They cover **functional, negative, and edge cases** for login, dashboard, transactions, receipt scanning, charts, and email reports. Each test case includes input, expected output, actual result, and status to ensure system reliability and accuracy.

TABLE:

Test Case ID	Feature	Test Description	Input	Expected Result	Actual Output	Status
TC-01	Login	Verify login with valid credentials	Email: user@mail.com, Password: 12345	User is redirected to dashboard	Dashboard loaded successfully	Pass
TC-02	Login	Verify login with invalid password	Email: user@mail.com, Password: wrong	Error message: "Invalid credentials"	Error message displayed	Pass
TC-03	Login	Verify login with empty fields	Email: "", Password: ""	Error: "Please enter email and password"	Error displayed correctly	Pass
TC-04	Dashboard	Verify dashboard loads correctly	Logged-in user	Show income, expense, graphs	Dashboard displays income: $4800, expense: $1780	Pass
TC-05	Bar Graphs	Verify bar graph accuracy	Transaction data available	Bar chart shows income (green) & expense (red)	Bar chart displayed correctly	Pass

TC-05	Bar Graphs	Verify bar graph accuracy	Transaction data available	Bar chart shows income (green) & expense (red)	Bar chart displayed correctly	Pass
TC-06	Pie Charts	Verify pie chart for expense categories	Transactions with categories: Food, Travel, etc.	Pie chart shows category-wise distribution	Pie chart displayed accurately	Pass
TC-07	Add Transaction	Verify adding transaction with valid data	Description: Groceries, Amount: 50, Category: Food	Transaction added, dashboard updated	Transaction added & visible in list	Pass
TC-08	Add Transaction	Verify adding transaction with empty fields	Leave all fields blank	Error: "Please fill all required fields"	Error displayed correctly	Pass
TC-09	Add Transaction	Verify invalid amount input	Description: Test, Amount: abc, Category: Food	Error: "Invalid amount"	Error displayed correctly	Pass
TC-10	Receipt Scanner	Verify receipt upload works	Upload valid receipt image	AI extracts data & adds transaction	AI extracted data successfully	Pass
TC-11	Receipt Scanner	Verify invalid file type handling	Upload .txt file	Error: "Unsupported file type"	Error displayed correctly	Pass
TC-12	Email Report	Verify monthly email report sent	End of month	Email with financial summary sent to user	Email received successfully	Pass
TC-13	Account Details	Verify adding new account	Account Name: Savings	Account added & visible in dashboard	Account added successfully	Pass
TC-14	Transaction History	Verify transactions list updates after new entry	Add a new transaction	New transaction appears in the list	Transaction visible in list	Pass
TC-15	Security	Verify session expires after logout	Logout and refresh	User redirected to login page	User redirected successfully	Pass

Table 7.1 Test cases

Test Case 1: User Authentication via Clerk

Test Case ID: TC001

Scenario: User opens the app and signs in

Input: Valid email and password

Expected Result: User is authenticated successfully, profile is loaded, and access is granted

Actual Result: Dashboard loaded successfully.

Status: Pass

Test Case 7: Manual Transaction Entry

Test Case ID: TC007

Scenario: User logs an expense manually

Input: Category: "Food", Amount: 500, Date: Today

Expected Result: Transaction is created and saved in the database

Actual Result: Dashboard displayed category, amount and date of transaction.

Status: Pass

Test Case 10: Receipt Upload and AI Categorization

Test Case ID: TC010

Scenario: User uploads a receipt

Input: JPG receipt image with itemized bill

Expected Result: Gemini scans the receipt, extracts data, and creates a categorized transaction.

Actual Result: AI Extracted data successfully and categorized them.

Status: Pass

Test Case 12: Background Job for Monthly Summary

Test Case ID: TC012

Scenario: System triggers monthly report

Input: End of month with multiple user transactions

Expected Result: Inngest generates and triggers AI report, and Resend delivers it via email

Actual Result: List is generated according to monthly transactions.

Status: Pass

Test Case 15: Security Management

Test Case ID: TC015

Scenario: Verifying session expire after log out.

Input: Logout

Expected Result: Redirected to login page.

Actual Result: Redirected to login page successfully.

Status: Pass

\

CHAPTER 8

CONCLUSION

The **AI Finance Platform** has been successfully developed as a comprehensive financial management solution that integrates artificial intelligence with modern web technologies. The primary goal of the project was to simplify financial tracking, budgeting, and reporting for users while providing intelligent insights through AI-powered automation. By leveraging **Next.js and React** for the frontend, **PostgreSQL** for data storage, and **Gemini AI** for advanced features like receipt scanning and automated categorization, the platform offers a seamless and user-friendly experience.

The system meets all major functional requirements, including secure user authentication, manual and AI-assisted transaction management, multi-account support, and dynamic dashboards with visual analytics such as bar graphs and pie charts. Non-functional requirements such as **performance, security, scalability, and usability** have also been addressed to ensure high reliability and efficiency. Additionally, features like **automated email reports**, **budget alerts**, and **real-time analytics** provide added convenience to users, making financial decision-making more informed and accurate.

Extensive testing confirmed that the platform performs effectively under normal and edge conditions, handling multiple users concurrently while maintaining speed and stability. Security measures such as encrypted passwords, JWT-based authentication, and HTTPS communication enhance data privacy and integrity. Overall, this project demonstrates how AI can transform traditional finance tools into intelligent, adaptive systems that save time, reduce errors, and improve financial planning.

Future Improvements

8.1 Mobile App Version

To improve accessibility, a mobile version of the platform can be developed:

- Learn and implement **React Native** or **Expo**, which shares similarities with React/Next.js, making the transition easier.
- The mobile app can reuse existing UI and logic, allowing users to manage finances on the go.
- Add features like push notifications for transaction alerts or budget reminders.

8.2 NLP Features (Natural Language Processing)

Natural language support can significantly improve user interaction:

- Use pre-trained models from **Hugging Face Transformers** to allow users to ask queries.
- Train a simple **classification model** to categorize transaction descriptions into types.

8.3 Machine Learning Enhancements

Machine learning can provide intelligent insights and predictions:

- Implement **LSTM (Long Short-Term Memory)** networks or similar models to predict future expenses based on past data.
- Use ML to detect seasonal spending patterns and make recommendations accordingly.

CHAPTER 9
REFERENCE

- S. Busk, “What is Personal Finance?” Maxprog, 2023. Archived from the original on 13 March 2023. Retrieved 18 July 2022. https://www.maxprog.com/site/blog/post.php?id=954&topic=What-is-Personal-Finance%3F.
- S. Aishwarya and S. Hemalatha, “Smart Expense Tracking System Using Machine Learning,” AI4IoT 2023: Proceedings of the 1st International Conference on Artificial Intelligence for Internet of Things, pp. 634–639, 2024. chrome-extension://kdpelmjpfafjppnhbloffcjpeomlnpah/https://www.scitepress.org/Papers/2023/126139/126139.pdf?utm_source=chatgpt.com.
- Prof. Sushma A 1, Abdus Salaam I 2, Danda Ajith Kumar 3, Shishira M 4, Thanuja B, “Finance Tracker System,” International Journal of Research Publication and Reviews, vol. 6, no. 2, 2024. chrome-extension://kdpelmjpfafjppnhbloffcjpeomlnpah/https://ijrpr.com/uploads/V6ISSUE2/IJRPR39015.pdf?utm_source=chatgpt.com.
- Tihomir Stefanov, Milena Stefanova, Silviya Varbanova , Stanislav Temelkov , “Personal Finance Management Application,” TEM Journal, vol. 13, no. 3, pp. 2066–2075, Aug. 2024. chrome-extension://kdpelmjpfafjppnhbloffcjpeomlnpah/https://www.temjournal.com/content/133/TEMJournalAugust2024_2066_2075.pdf?utm_source=chatgpt.com.
- Lavesh Lingayat, Neha Yadav, Prajwal Rathod,Pranay Durutkar,Prof. Shilpa Ghode, “Design and Implementation of Real-Time Expense Tracker Using Machine Learning,” SSRN Electronic Journal, 2024. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4754463.
- Samar Verma, Samarjeet Singh Kheda, Shivam Kuwale, “Personal Finance Tracker,” International Research Journal of Modernization in Engineering Technology and Science, vol. 6, no. 5, 2024. chrome-extension://kdpelmjpfafjppnhbloffcjpeomlnpah/https://www.irjmets.com/uploadedfiles/p

aper//issue_5_may_2024/58377/final/fin_irjmets1717316478.pdf?utm_source=chatgpt.com.

- Yu Xie, “The Design and Implementation of Personal Finance Management System Based on Android,” ResearchGate, 2017. https://www.researchgate.net/publication/314642335_The_Design_and_Implementation_of_Personal_Finance_Management_System_Based_on_Android.
- A. Arish, “Finance Manager Project – GitHub Repository,” https://github.com/acharyaarish/Finance-Manager.
- S. Gupta, “MERN Expense Tracker Project – GitHub Repository,” https://github.com/sakshgupta/MERN-Expense-Tracker.
- K. Lee, “FinanceTracker – GitHub Repository,” 2024. A personal finance tracker that supports income/expense tracking, budgeting, report generation, localization, and visualizations of trends and categorized expenses. Built with React (TypeScript) frontend and Flask (Python) backend, using SQLite for local development. https://github.com/Keonleebv/FinanceTracker.