B.M.S. COLLEGE OF ENGINEERING BENGALURU
Autonomous Institute, Affiliated to VTU

OOMD Mini Project Report

SMART MOBILE BANKING AND PAYMENT SYSTEM

Submitted in partial fulfillment for the award of degree of

Bachelor of Engineering
in
Computer Science and Engineering

Submitted by:

GHANSHYAM SHARMA (1BM23CS100)
ABDUL AHAD (1BM23CS353)
SHREYASH SHAURYA (1BM23CS354)
UTKRISHT UMANG (1BM23CS355)

Department of Computer Science and Engineering
B.M.S. College of Engineering
Bull Temple Road, Basavanagudi, Bangalore 560 019
2025-26

B.M.S. COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DECLARATION

We, **Ghanshyam Sharma (1BM23CS100), Abdul Ahad (1BM23CS353), Shreyash Shaurya (1BM23CS354)** and **Utkrisht Umang (1BM23CS355)** students of 5th Semester, B.E, Department of Computer Science and Engineering, BMS College of Engineering, Bangalore, hereby declare that, this OOMD Mini Project entitled "Smart Mobile Banking and Payment System" has been carried out in Department of CSE, B.M.S. College of Engineering, Bangalore during the academic semester August 2025- December 2025. I also declare that to the best of our knowledge and belief, the OOMD mini Project report is not from part of any other report by any other students.

Signature of the Candidate

Ghanshyam Sharma (1BM23CS100)
Abdul Ahad (1BM23CS353)
Shreyash Shaurya (1BM23CS354)
Utkrisht Umang (1BM23CS355)

B.M.S. COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the OOMD Mini Project titled "Smart Mobile Banking and Payment System" has been carried out by Ghanshyam Sharma (1BM23CS100), Abdul Ahad (1BM23CS353), Shreyash Shaurya (1BM23CS354) and Utkrisht Umang (1BM23CS355) during the academic year 2025-2026.

Signature of the Faculty in Charge (Sonika Sharma D)

Table of Contents

Chapter 1: Problem Statement

Managing digital financial transactions through multiple standalone applications leads to inconvenience, security risks, and fragmented user experiences. Users face difficulties in handling wallet balance, payments, transfers, and bill payments across different platforms, resulting in inefficiency and lack of real-time visibility. Existing solutions often lack unified authentication, seamless QR/voice transactions, and consistent cross-platform accessibility. To address these issues, a centralized Smart Mobile Banking and Payment System is required to securely integrate wallet services, digital payments, utility bill processing, and transaction tracking in one unified platform. This system must ensure high security, fast performance, and ease of use for both technical and non-technical users.

Chapter 2: Software Requirement Specification

1. Introduction

1.1 Purpose of this Document

The purpose of this document is to define the requirements and specifications for the Smart Mobile Banking and Payment System. It provides clarity on the system's objectives, functional scope, constraints, and deliverables, serving as a reference for developers, stakeholders, and testers.

1.2 Scope of this Document

This document outlines the complete working and goals of the Smart Mobile Banking and Payment System. It includes a breakdown of features such as secure authentication, wallet services, digital payments, QR/voice-based transactions, and bill payments. It also highlights the expected development cost, duration, and associated constraints.

1.3 Overview

The Smart Mobile Banking and Payment System is a cross-platform fintech application designed to simplify digital transactions. It supports wallet management, peer-to-peer transfers, bill payments, QR-based payments, voice-activated transactions, and real-time account monitoring. Built with an emphasis on security, accessibility, and scalability, the system ensures seamless user experiences across mobile, web, and desktop platforms.

2. General Description

The Smart Mobile Banking and Payment System serves everyday users, merchants, and administrators by offering features such as secure login, wallet balance management, fund transfers, transaction history, bill payments, and voice-based operations. Non-technical users can comfortably navigate the system due to its intuitive interface. The backend provides real-time updates, encrypted storage, and secure API communication.

3. Functional Requirements

3.1 User Authentication

- Allow users to register and log in securely using email/phone and passwords.
- Implement token-based authentication to ensure authorized access.

3.2 Wallet & Account Management

- Allow users to view wallet balance, past transactions, and fund additions.
- Maintain accurate records of wallet top-ups, refunds, and spends.

3.3 Payment & Transfer Management

- Support sending money via QR codes, contacts, or mobile numbers.
- Facilitate voice-based transactions for enhanced accessibility.
- Process payments for merchants, peers, and services securely.

3.4 Bill & Utility Payments

- Enable users to pay utility bills (electricity, phone, DTH, broadband).
- Provide real-time form validation and confirm successful payment status.

3.5 Transaction History & Reporting

- Display detailed transaction logs including timestamps, IDs, and payment modes.
- Generate downloadable invoices for users and corporate clients.

4. Interface Requirements

4.1 User Interface

- Provide a responsive, intuitive interface using Material Design principles.
- Support Android, iOS, web, and desktop platforms with consistent UX.
- Include smooth animations, clear navigation, and accessible UI elements.

4.2 Integration Interfaces

- Integrate with payment gateways for secure online transactions.
- Integrate with QR libraries, voice recognition engines, and third-party bill APIs.
- Use backend APIs (Node.js, Express.js) for real-time data exchange.

5. Performance Requirements

5.1 Response Time

- The system should respond to all user requests within 2 seconds under normal load.

5.2 Scalability

- Handle a minimum of 10,000 concurrent users during peak usage.

5.3 Data Integrity

- Ensure consistent, reliable, and tamper-proof data across authentication, wallet, and payment modules.

6. Design Constraints

6.1 Hardware Limitations

- Support standard mobile devices, desktops, barcode scanners (for merchants), and printers where applicable.

6.2 Software Dependencies

- Flutter for the front-end (Android, iOS, Web, Desktop).
- Node.js + Express.js for backend services.
- Supabase (PostgreSQL) for secure database storage and real-time updates.
- Additional packages such as speech_to_text, QR code generators/scanners, flutter_secure_storage.

7. Non-Functional Attributes

7.1 Security

- Implement encrypted storage for sensitive data.
- Use robust authentication, authorization, and secure API communication with Bearer tokens.

7.2 Reliability

- Ensure high uptime and fault tolerance with redundant services and real-time sync.

7.3 Scalability

- Architect the system to support feature expansion, increased user load, and multi-region deployment.

7.4 Portability

- Ensure full functionality across Android, iOS, web browsers, and desktop platforms.

7.5 Usability

- Maintain a clean, user-friendly interface suitable for users of varying technical proficiency.

7.6 Reusability

- Use modular design and reusable components to simplify future enhancements and maintenance.

7.7 Compatibility

- Support major web browsers including Chrome, Safari, Firefox, and Edge.

7.8 Data Integrity

- Guarantee accurate, consistent, and secure data retrieval and storage across all modules.

8. Preliminary Schedule and Budget

The estimated development timeline for the Smart Mobile Banking and Payment System is **7–8 months**, covering planning, development, testing, and deployment.
The projected budget is approximately **$180,000**, covering backend integration, multi-platform development, payment gateway setup, and enhanced security requirements.

Chapter 3: Class Modeling

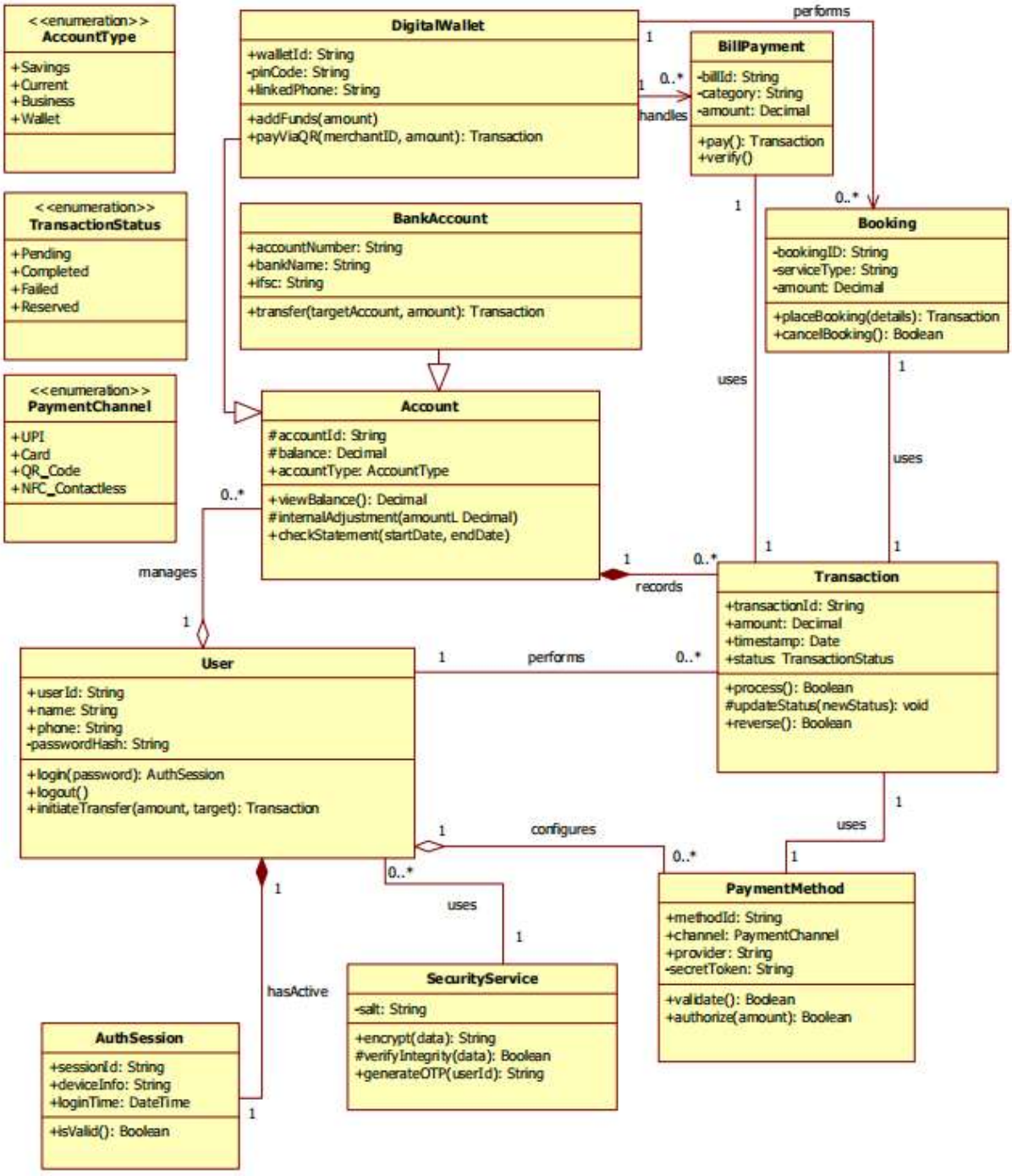

Fig 1: Class Diagram for the payment application

Explanation

The class diagram represents the structural design of the mobile banking and payment system by defining all essential entities involved in authentication, account handling, wallet operations, payments, and transaction management. It shows how users interact with their wallet, bank accounts, and payment methods while the system ensures security, recording, and verification of every financial activity. The relationships between these classes highlight how data flows through the system—such as a user performing a transaction, an account recording it, and a security service validating it.

The model uses several advanced UML features such as enumerations (AccountType, PaymentChannel, TransactionStatus), one-to-many associations (User–Account, Account–Transaction), composition (accounts tightly owning transactions), and service-based classes (SecurityService) to maintain modularity. The design ensures that sensitive operations like transfers, QR payments, bill payments, and bookings are handled securely, consistently, and traceably across the application.

Relevance of Each Class

User

Represents the primary system actor and stores basic profile and credential information. It initiates transactions, manages accounts, and authenticates through login and logout operations. This class acts as the entry point for almost all system functionalities and links the user to their financial assets.

AuthSession

Maintains the session state after a successful login. It records device information, timestamps, and session validity. This class ensures that all subsequent operations occur under a verified and active user identity.

SecurityService

Provides essential cryptographic operations such as hashing, encryption, integrity checks, and OTP generation. By centralizing security logic, it ensures consistent implementation of security protocols across login, payment authorization, and data transmission.

Account

Acts as the financial ledger associated with a user. It stores balance, account type, and exposes operations for viewing statements or performing adjustments. Every monetary transaction is linked to an account, making it the core of financial tracking.

BankAccount

Represents external bank accounts linked to the system. It stores bank-related details and enables money transfers between the digital system and the physical banking network. This class bridges internal wallet logic with real-world financial institutions.

DigitalWallet

Contains wallet-specific information and enables in-app payments, QR-based transactions, and fast transfers. It handles wallet balance updates through add-funds operations and supports merchant payments using linked identifiers.

PaymentMethod

Defines the channels available to users for performing transactions, such as UPI, cards, QR, or NFC. It includes provider details and authorization logic. This class allows the system to handle multiple payment channels with flexibility.

Transaction

Stores complete details of every financial action, including amount, timestamp, and status. It provides methods for processing, updating status, and reversing transactions when necessary. It forms the primary audit trail for the system.

BillPayment

Models utility or service-based payments. It stores bill details and performs the necessary verification before initiating a transaction. This class ensures that recurring and one-time bill payments are processed accurately.

Booking

Represents services or reservations that require a payment, such as ticket bookings or service appointments. It connects booking details to associated transactions and supports cancellation workflows.

Chapter 4: State Modeling

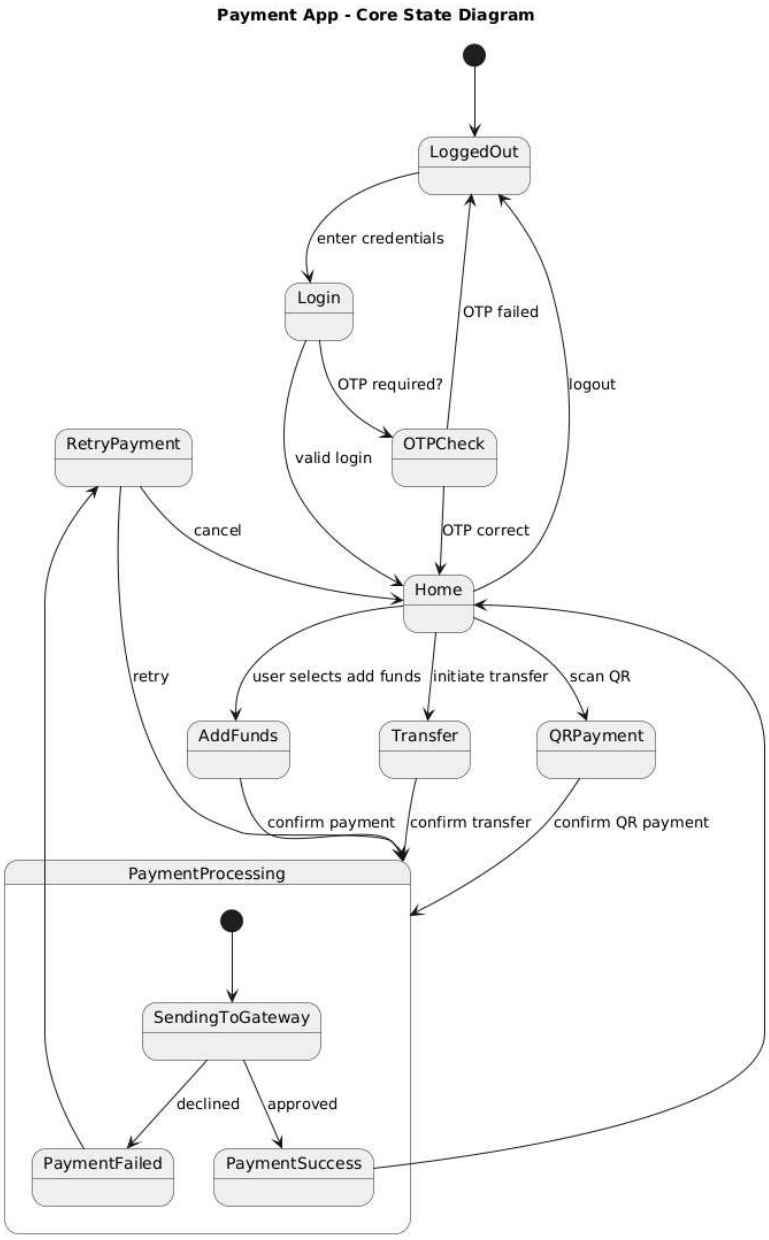

Fig 2.1: State Diagram for the payment application

Relevance of Each State

LoggedOut

Initial state where the user is not authenticated. Access to financial actions is blocked.

Login

User enters credentials, initiating authentication.

OTPCheck

System verifies OTP when additional security is required. Ensures high-security authentication.

Home

Represents the central application dashboard where the user may choose actions like add funds, transfer, view balance, or make QR payments.

AddFunds / Transfer / QRPayment

Service-specific operational states where the user prepares the intent for a payment. Each captures the user's chosen action.

PaymentProcessing

Core state that handles payment initiation and submission to the gateway.

SendingToGateway

Represents the moment payment details are transmitted for authorization.

PaymentSuccess

Indicates that the transaction has been approved and the operation succeeded.

PaymentFailed

Indicates gateway rejection or failure. No funds are moved.

RetryPayment

Allows the user to re-attempt the same transaction after a failure.

Relevance of Each Event

enter credentials

Triggers the transition from LoggedOut to Login. User begins authentication.

OTP required? / OTP correct / OTP failed

Security events used to validate identity. These determine whether the user proceeds to Home or returns to LoggedOut.

select payment action

User chooses Add Funds, Transfer, or QR Payment on the Home screen.

confirm payment / confirm transfer

User has reviewed and accepted the amount; the payment attempt starts.

send payment to gateway

System submits details for authorization.

approved

Indicates success from the gateway. Funds move or recharge completes.

declined

Indicates failure from the gateway. Transaction is aborted.

retry

User chooses to re-attempt the failed payment.

cancel

User abandons the flow and returns to Home.

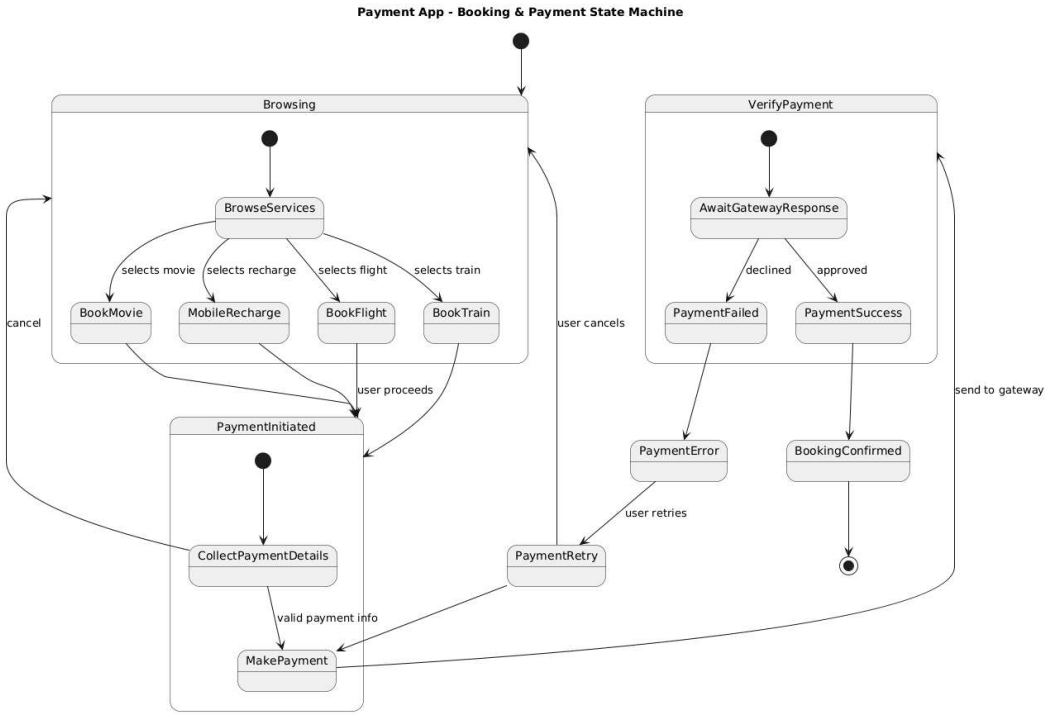

Fig 2.2: State Diagram for the booking module

Relevance of Each State

Browsing

Represents the starting point where the user views available services such as flights, trains, movies, or recharge options. It groups the discovery phase of the booking flow.

BrowseServices

Detailed state where the system displays available options and accepts user selection. This determines the path for the next booking activity.

BookFlight / BookTrain / BookMovie / MobileRecharge

Each represents a service-specific booking state where additional details (seat selection, passenger info, operator info) are collected. These states extend the general booking process.

PaymentInitiated

User has selected a service and is ready to proceed with payment. All booking details are finalized here.

CollectPaymentDetails

System collects payment method, amount, and verification inputs (PIN/OTP). Ensures valid user intent before initiating payment.

MakePayment

Represents the logic of sending payment details to the payment service. It triggers the actual payment process.

VerifyPayment

The system waits for confirmation from the Payment Gateway to determine if the transaction succeeded or failed.

PaymentSuccess

Represents successful transaction approval. The booking can now be finalized.

PaymentFailed

Represents a declined or failed transaction. No booking is confirmed in this state.

BookingConfirmed

The final booking details are generated and issued to the user. The transaction is complete.

PaymentError / PaymentRetry

Handles failure cases where the user is allowed to retry payment or cancel the booking.

Relevance of Each Event

selects flight/train/movie/recharge

User chooses a specific service, triggering a transition from browsing to the respective booking state.

proceed / confirm booking

Indicates the user is ready for payment, moving them into the payment phase.

cancel

User aborts the booking or payment process and returns to the home browsing state.

valid payment info

Indicates that the payment inputs (method, PIN, OTP) are validated successfully.

send to gateway

Triggers the payment attempt to the external Payment Gateway.

approved

Indicates that the gateway has authorized the payment; booking will be finalized.

declined

Indicates the gateway rejected the payment; user must retry or cancel.

retry

Allows user to attempt the payment again using the same or different payment method.

timeout / failure

Represents network or temporary gateway issues that occur during verification.

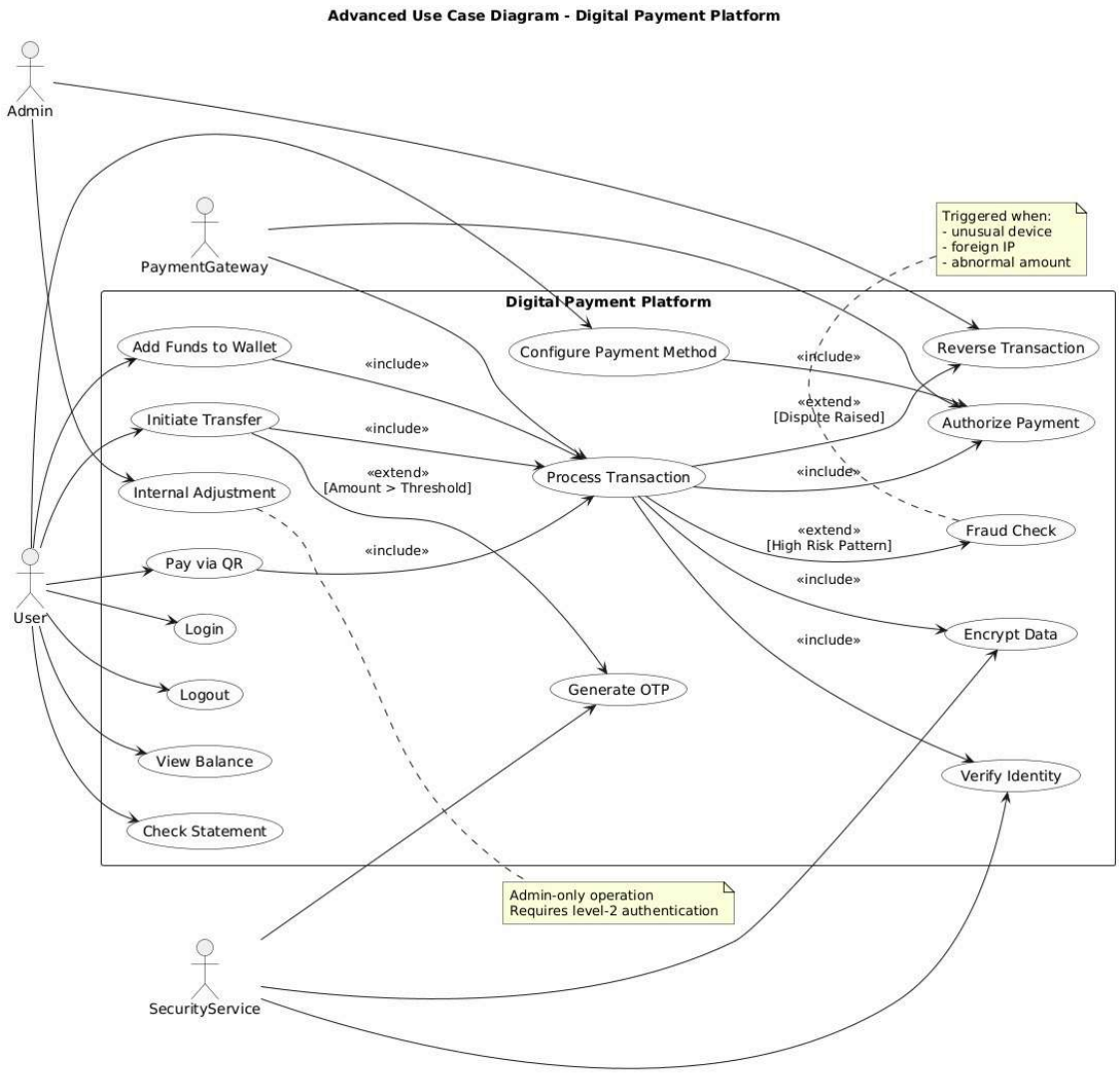

Fig 3.1: Use case Diagram for the payment application

Advanced Use Case Diagram

Overview

This advanced use case diagram represents the core operations of a digital payment platform, including authentication, wallet services, QR payments, transfers, security operations, and transaction processing. The system shows multiple actors—User, Admin, Payment Gateway, and SecurityService—each interacting with different parts of the platform. The diagram uses «include» and «extend» relationships to capture shared processes (like OTP generation, verification, encryption) and conditional tasks (such as fraud checks or dispute handling). This

advanced model reflects real-world workflow complexity and isolates high-risk or admin-only operations.

Relevance of Actors

User

The User is the primary actor who performs everyday actions such as logging in, viewing balance, adding funds, scanning QR codes, and initiating transfers. Most of the platform's functional requirements originate from this actor, making it central to the system boundary.

Admin

The Admin performs advanced or restricted operations such as reversing transactions, configuring payment methods, and performing internal adjustments. This actor represents system maintainers and compliance personnel who handle exceptional or privileged processes.

Payment Gateway

The Payment Gateway is an external entity responsible for authorizing and verifying payments. It interacts with the platform during the payment process and returns approval or failure responses. It is modeled as an external actor because it lies outside the system boundary.

SecurityService

SecurityService is a special actor representing internal security components such as encryption, OTP generation, identity verification, and high-risk checks. Modeling it as an actor highlights that security operations are triggered independently of user actions and interact with multiple use cases.

Relevance of Each Use Case

Login / Logout

Handles user authentication and session termination. This establishes secure access and protects sensitive operations from unauthorized users.

View Balance / Check Statement

Allows users to monitor wallet and account activity. These features are essential for transparency, financial awareness, and preventing fraudulent or unnoticed activity.

Add Funds to Wallet

Enables users to increase wallet balance using external payment methods. It connects with Payment Gateway and includes verification steps to prevent unauthorised top-ups.

Initiate Transfer

Used for sending money to another user or external account. It triggers the payment process and integrates validation, encryption, and optional fraud checks.

Internal Adjustment (Admin)

Allows privileged correction of account balances in exceptional circumstances (failed settlement, rollback scenarios). This is restricted to admin to prevent misuse.

Pay via QR

Supports QR-based quick payments. This use case includes internal payment validation and interacts directly with Process Transaction.

Configure Payment Method (Admin)

Enables administrators to add, modify, or validate payment channels like UPI, card, or NFC. This ensures that transaction routing remains functional and secure.

Process Transaction

The central use case of the platform. All monetary actions route through it. It includes authorization, fraud detection, encryption, gateway verification, and recording. It forms the backbone of the system.

Generate OTP

Provides one-time passwords for sensitive operations such as transfers, login, or payment authorization. It is included by multiple use cases for security.

Reverse Transaction

Allows refunding or rolling back a transaction. This is an admin-level security operation used when disputes or failures occur.

Authorize Payment

Ensures that the user has the necessary permissions and funds to complete a transaction. This step is included within the main Process Transaction flow.

Verify Payment

Triggered after each transaction attempt. It interacts with the Payment Gateway and confirms transaction success or failure.

Fraud Check (Extend)

Executes automatically when a transaction exceeds risk thresholds, unusual device/IP is detected, or suspicious behaviour occurs. It protects the platform from unauthorized and malicious activity.

Encrypt Data

Secures sensitive data including PINs, transaction details, and credentials before processing. Included across multiple use cases to maintain end-to-end security.

Verify Identity

Performs identity checks for high-risk actions or admin-restricted operations. Ensures the platform follows compliance and KYC rules.

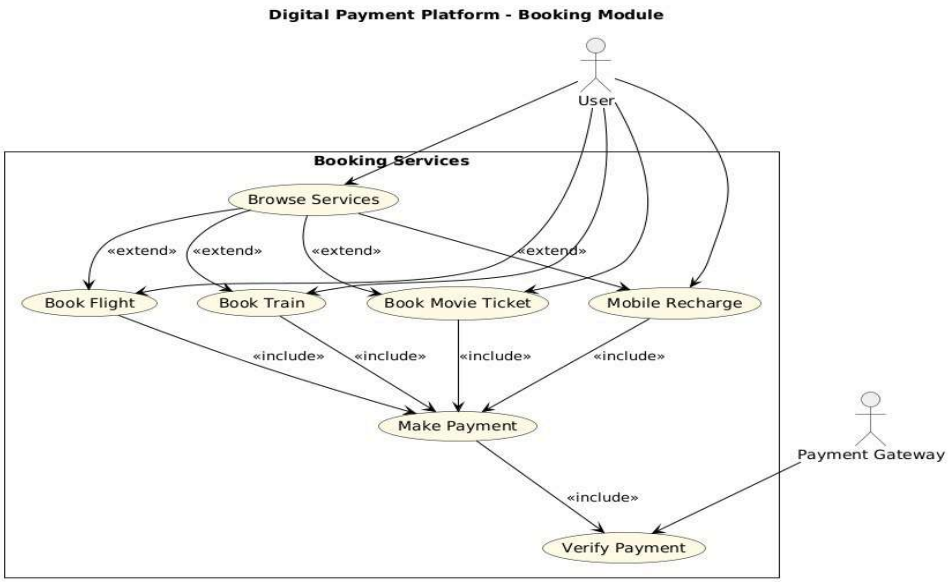

Fig 3.2: Use case Diagram for the booking module

Booking Module

Overview

This diagram models the booking services offered by the payment platform: users browse available services (flights, trains, movies, mobile recharge), select a specific service, and complete a booking using the platform's common payment flow. Common functionality (Make Payment → Verify Payment) is factored out using «include», and specialized booking behaviours

are shown with «extend» where optional steps or alternatives apply. The design clarifies the user journey from discovery to payment and isolates payment verification as an external reliance on the Payment Gateway.

Actors

User
Primary actor who discovers services, selects items, and completes bookings and payments. Represents all end-users (customers) and drives functional requirements for search, selection, and checkout flows.

Payment Gateway (external)
External system that performs payment verification and settlement. Modeled as an external actor because it resides outside the platform and provides the authoritative confirmation for money movement.

Use Cases — Relevance & Purpose

Browse Services
Entry point that lets users search, filter, and view available offerings (flights, trains, movies, recharge options). It's essential because it determines discoverability and the conversion funnel — a good browse experience increases bookings.

Book Flight (extend Browse Services)
Service-specific booking flow for airline reservations. It typically requires seat selection, passenger details, and may include additional validation (passport info, fare rules). Shown as an extension since it adds domain-specific steps beyond generic booking.

Book Train (extend Browse Services)
Rail booking flow including coach/seat selection, passenger PNR requirements, and reservation holds. It extends Browse Services to show the extra steps needed for train bookings.

Book Movie Ticket (extend Browse Services)
Cinema booking flow with seat selection, showtime selection, and optional concessions. It extends the generic process with UI and validation specific to seat map handling.

Mobile Recharge (extend Browse Services) Instant recharge flow for mobile top-ups. It's a lighter booking type with fewer inputs and faster payment needs; modeled as an extension because its flow is simpler but still follows the booking→payment pattern.

Make Payment (include by all Book… cases)
A mandatory, shared use case that handles collecting payment details, invoking the chosen PaymentMethod, and initiating the verification step. Extracting this common flow prevents duplication in service-specific use cases and centralizes payment rules (tokens, saved cards).

Verify Payment (included by Make Payment; external interaction)
Interacts with the Payment Gateway to confirm transaction success or failure. This use case is critical for finalizing bookings and triggering confirmations or rollbacks.

Advanced UML Features Used (brief)

- **«include»**: Make Payment and Verify Payment are included by all booking use cases to capture mandatory shared behavior.

- **«extend»**: Service-specific booking flows (flight/train/movie/recharge) extend the general Browse→Book path where additional steps are conditional or domain-specific.

- **System boundary**: Booking Services box defines what the platform provides vs external Payment Gateway.

- **Actor separation**: External actor (Payment Gateway) vs internal actor (User) clarifies trust and control boundaries.

- **Optional flows**: Use of extend models optional steps like seat-hold, extra validation or fare rules.

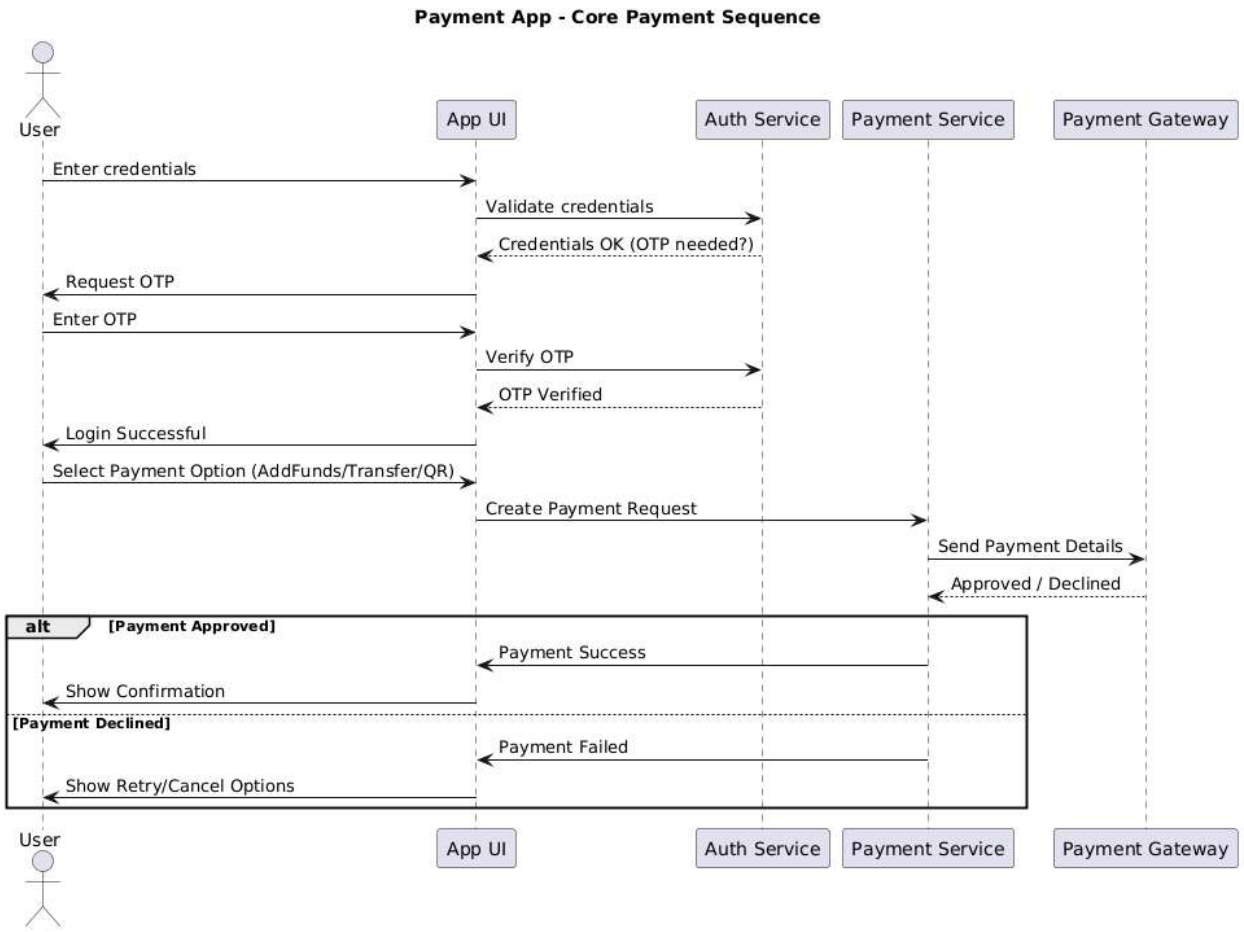

Fig 3.3: Sequence Diagram for the Payment Application

The sequence diagram illustrates the end-to-end flow of a payment in the mobile payment application, starting from user authentication and ending with the final transaction result. The process begins when the **User** submits login credentials through the **App UI**, which forwards them to the **Auth Service** for validation. If OTP verification is required, the Auth Service confirms it before the user is allowed into the home screen, ensuring secure access before financial actions.

After successful login, the user selects a payment action such as adding funds, transferring money, or paying via QR. The App UI passes the details to the **Payment Service**, which constructs the transaction request and sends it to the **Payment Gateway**, the external system responsible for authorizing payments. Based on the gateway's response—approved or declined—the Payment Service informs the UI.

If approved, the App UI shows a success message and the operation completes; if declined, the UI displays a failure message and offers retry or cancel options. The diagram clearly separates responsibilities: the UI handles user interactions, the Auth Service manages identity, the Payment Service executes transaction logic, and the Payment Gateway performs authorization.

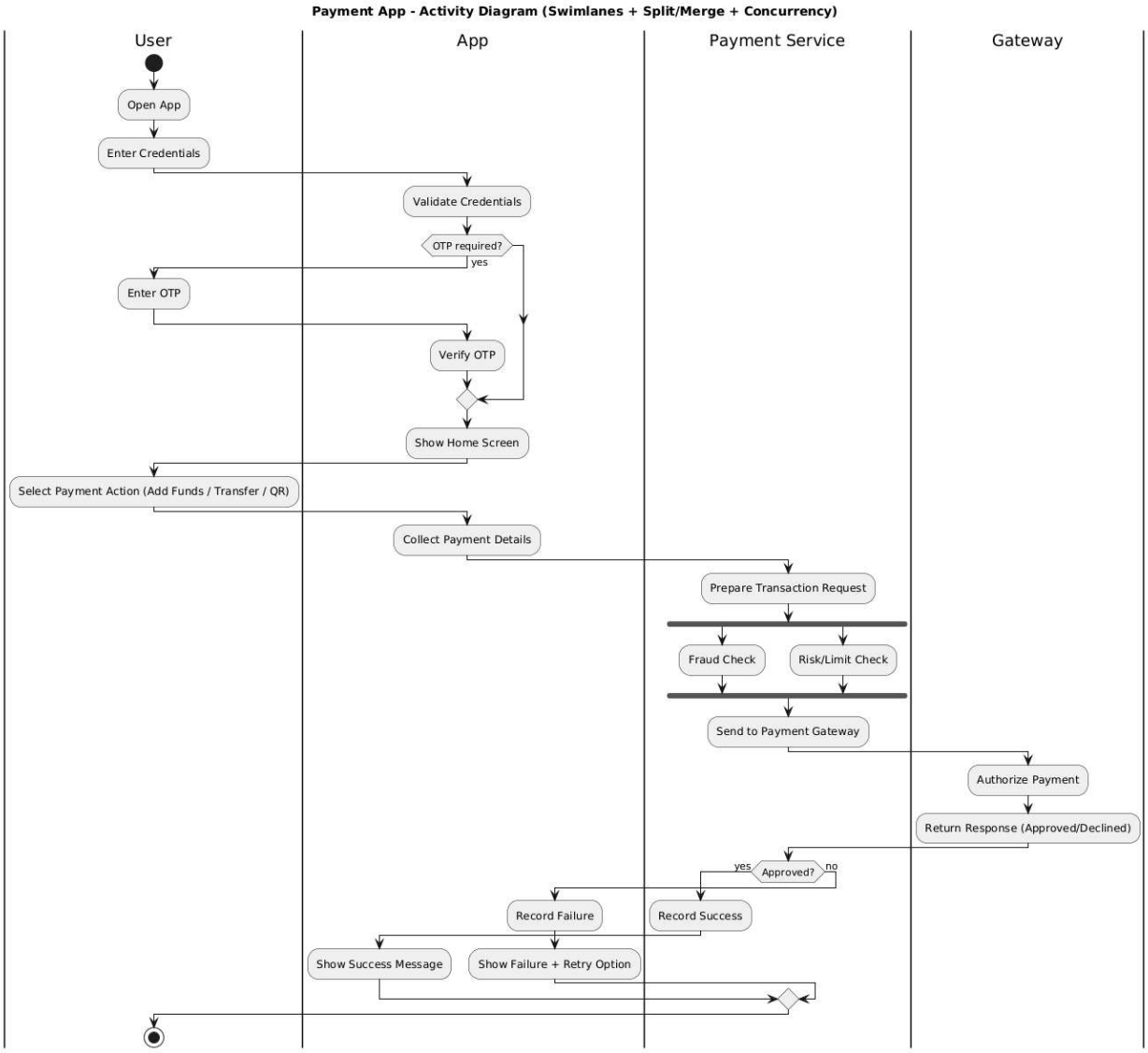

Fig 3.4: Activity Diagram for the Payment Application

The activity diagram illustrates the complete workflow of a payment operation within the mobile payment application, structured using swimlanes to clearly separate the responsibilities of the **User**, **App**, **Payment Service**, and **Payment Gateway**. The flow begins with the user opening the application, entering login credentials, and completing OTP verification if required. Once authenticated, the user selects an action such as adding funds, making a transfer, or performing a QR payment. The App then collects the necessary payment details and forwards them to the Payment Service.

A key feature of this diagram is the inclusion of **parallel processing (concurrency)**. After preparing the transaction request, the Payment Service performs a **Fraud Check** and a

Risk/Limit Check simultaneously using a fork node. These parallel branches later merge before the transaction is sent to the external Payment Gateway. This demonstrates how the system improves security and performance by evaluating risks in parallel rather than sequentially.

The Payment Gateway authorizes the transaction and returns an approval or decline response. Based on the outcome, the Payment Service records the result and the App displays either a success confirmation or a failure message with retry options. Decision nodes are used to represent branching logic (OTP required? Payment approved?), while merge nodes combine the flow back into a single path. Overall, the diagram effectively shows how the system handles authentication, parallel security checks, payment processing, and user interaction in a structured and coordinated sequence.

Chapter 6: UI Design with Screenshots

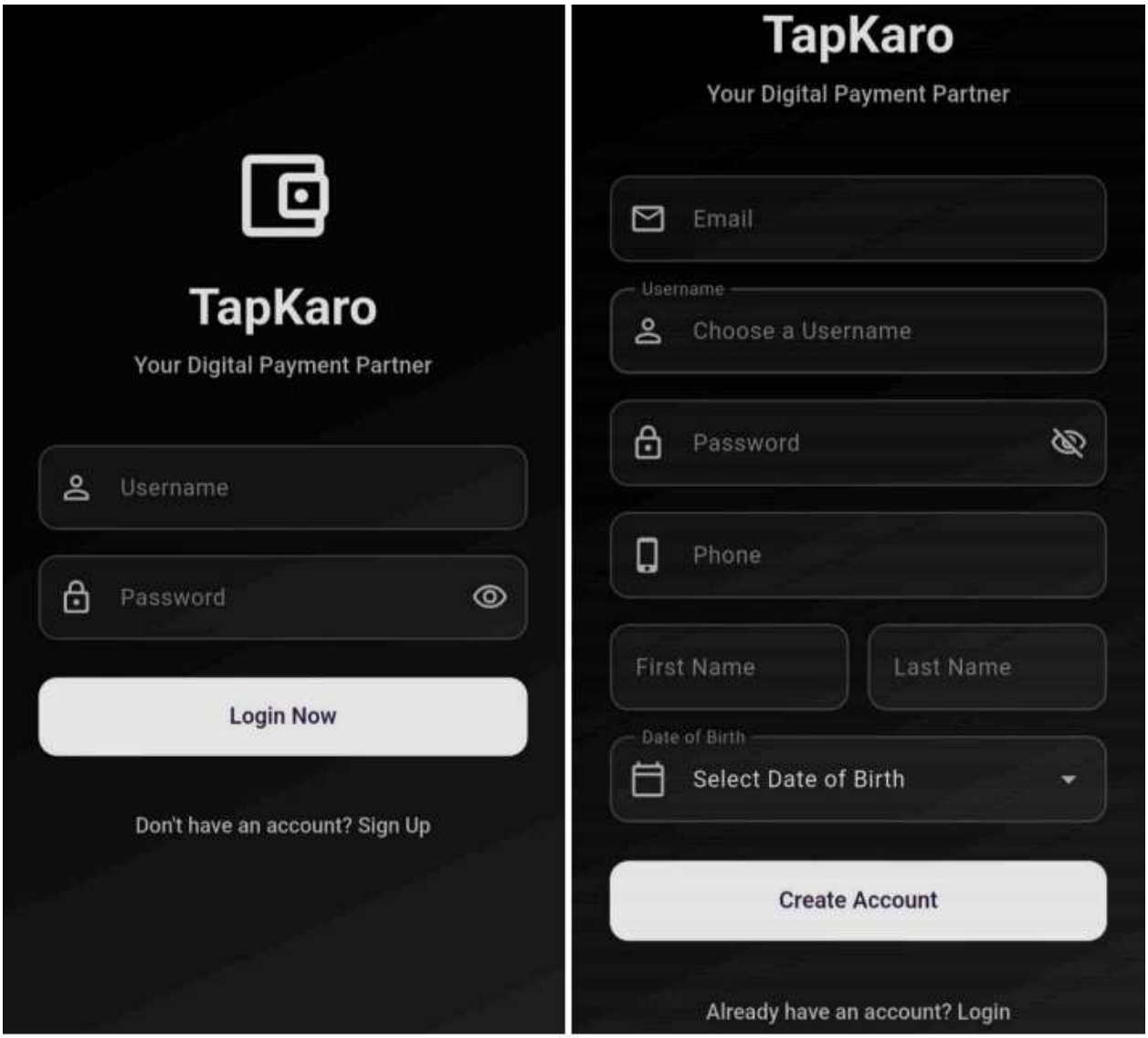

Fig. 4.1 Login and Signup page : The Login and Signup page allows users to securely register a new account or log in to an existing one.

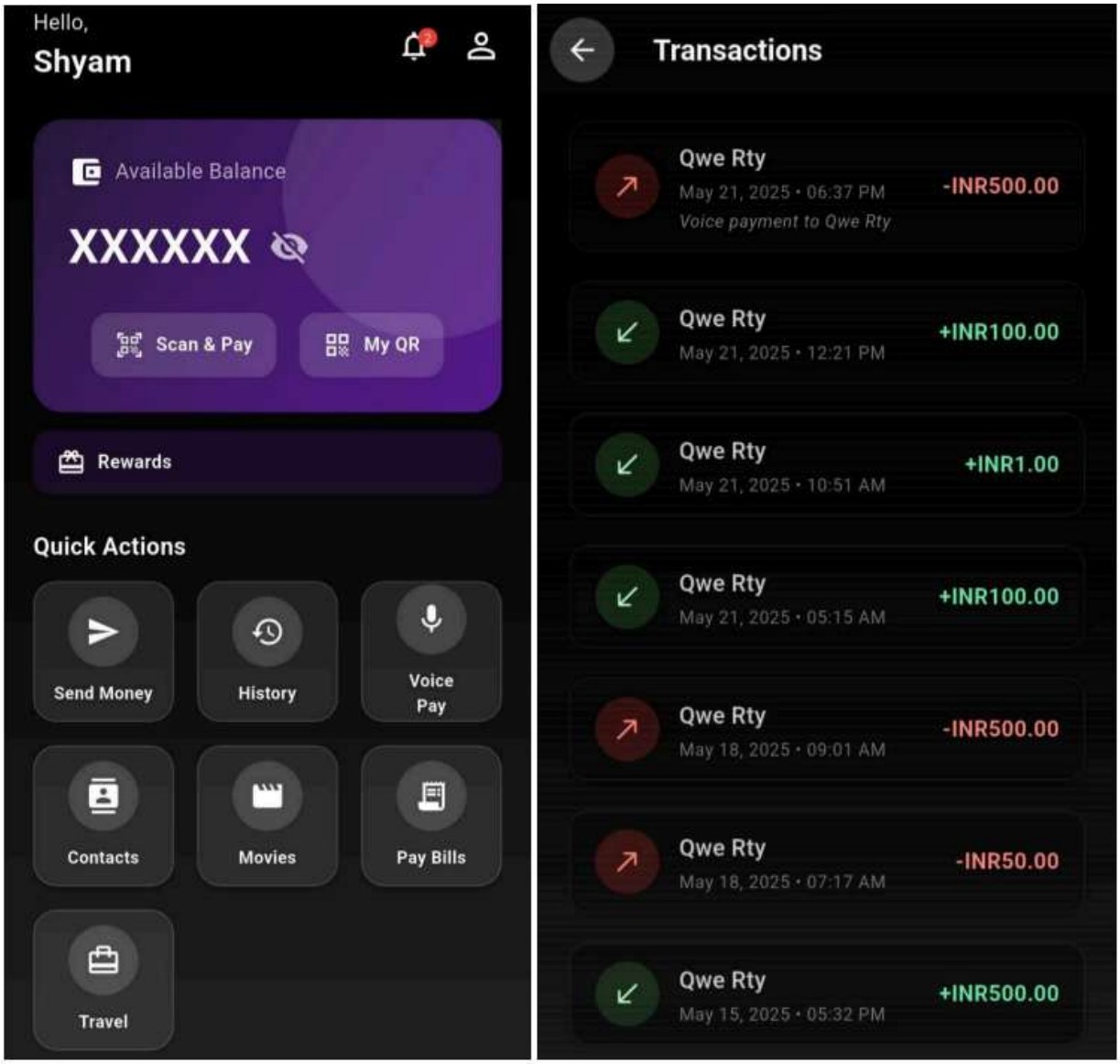

Fig. 4.2 Home page and transaction history page : The Home page offers quick access to main features, while the Transaction History page displays a detailed record of past transactions.

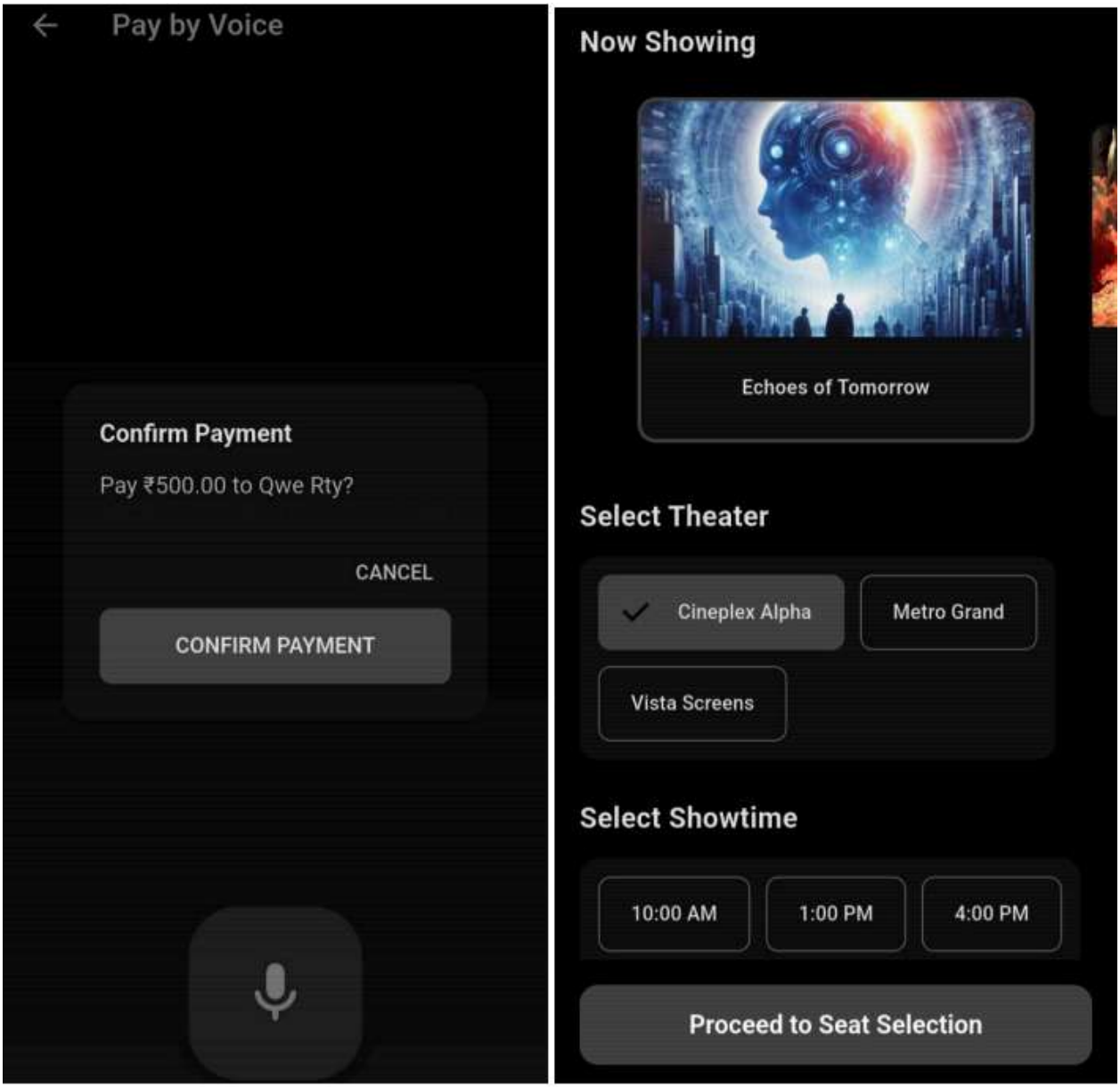

Fig 4.3: Voice pay and movie ticket booking page: The Voice Pay page enables users to make payments using voice commands, while the Movie Ticket Booking page allows seamless selection and purchase of movie tickets.

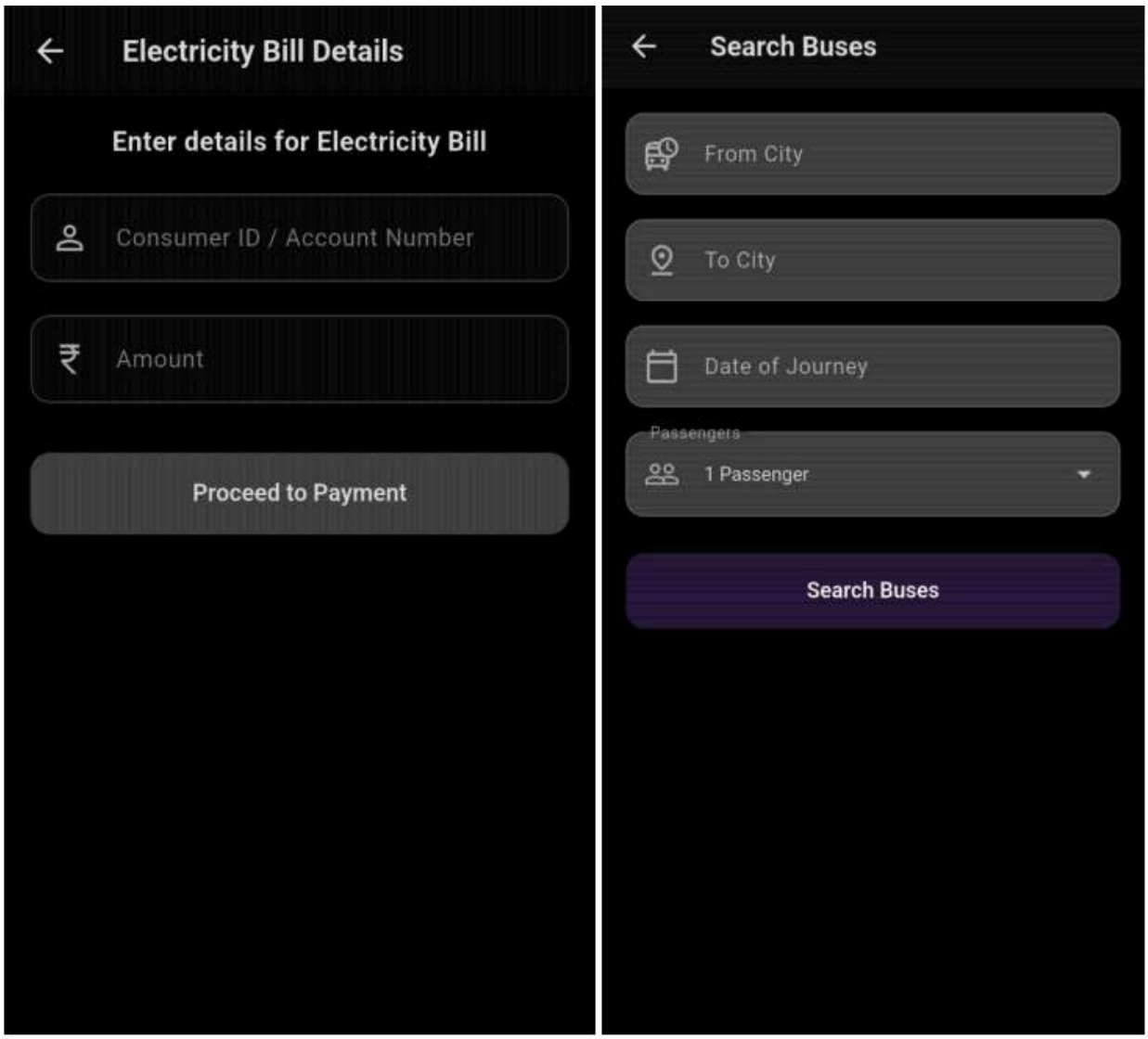

Fig 4.4: Bill payment and travel booking page: The Bill Payment page lets users pay utility bills with ease, while the Travel Booking page facilitates booking of flights, trains, or other travel services.