Title: Online Bank Management System

- **Subtitle:** Ex. 11 - Project Report
- **Name:** Karan Sehgal
- **Registration No:** 22BCE3939
- **Team No: 24**
- **Course/Subject:** Software Engineering Lab (BCSE301P)
- **Instructor’s Name:** Dr. Mehfooza M

Date of Submission: 03/04/25

INDEX

S.No	Topics	Page
1.	Analysis and Identification of Suitable Process Models	3
2.	Work Break-down Structure (Process Based, Product Based, Geographic Based and Role Based) and Estimations	17
3.	Requirement modelling using Entity Relationship Diagram	25
4.	Requirement modelling using Context flow diagram, DFD	29
5.	Requirement modelling using State Transition Diagram	40
6.	OO design – Use case Model, Class Model	44
7.	OO design – Interaction Models	52
8.	OO design – Package, Component and deployment models	58
9.	Design and demonstration of test cases. Functional Testing and Non- Functional Testing (using any open source tools)	68
10.	Story Boarding and User Interface design Modelling	75
11.	Challenges and Solutions	91
12.	Conclusion and Future Scopes	92

1. Analysis and Identification of Suitable Process Models

The Online Bank Management System is designed to centralize banking operations, optimize customer experiences, and ensure secure and efficient financial transactions. This system serves three key stakeholders: **customers**, **tellers**, and **administrators**, each with specific access levels and functionalities. The goal is to provide a user-friendly platform that aligns with modern technological standards and adheres to industry best practices.

System Requirements:

I have divided System Requirements into two sub categories:

Functional Requirements

These describe the service that the banking management system must offer, they are subdivided into three access levels: Admin Mode, Teller Mode, and Customer Mode:

Customer Mode

This mode allows customers to perform various self-service operations related to their accounts.

1. **Sign in with login and password:**
    - Customers authenticate themselves using a unique **username** and **password**.
    - Implement **Multi-Factor Authentication (MFA)** for enhanced security (e.g., SMS OTP, email verification).
2. **Update personal details:**
    - Modify profile information such as:
        - Name
        - Address
        - Contact information (phone, email)
    - Changes should require validation through a secure channel.
3. **Change password:**
    - Customers can reset their passwords through:
        - Old password verification.
        - Email or SMS-based confirmation for security.
        - Enforced strong password policies.
4. **View balance:**

- Real-time display of account balances.
- Separate views for **savings**, **current**, or other types of accounts.

5. **View personal history of transactions:**
    - Display a list of recent transactions, including:
        - Date and time.
        - Transaction type (debit/credit).
        - Amount and remaining balance.
        - Transaction remarks or references.
    - Enable filtering and searching (e.g., by date range or type).
6. **Transfer money:**
    - Internal and external transfers:
        - **Internal Transfers:** Between accounts in the same bank.
        - **External Transfers:** To accounts in other banks using NEFT, RTGS, or IMPS.
    - Include safeguards:
        - Daily transfer limits.
        - Beneficiary management with approval.
7. **Withdraw:**
    - Request withdrawals online for in-branch collection or linked ATM cards.
    - Option to generate a withdrawal slip or virtual token.
8. **Submit cash:**
    - Log cash deposits for in-branch processing or through automated cash deposit machines.
    - Generate receipts for records.

Manager Mode

Managers facilitate customer banking operations and manage accounts with specific permissions.

1. **Sign in with login and password:**
    - manager-specific credentials with role-based access controls.
    - Monitor logins and actions for audit purposes.
2. **Change password:**
    - Ability to reset their own passwords, adhering to strong password policies.

3. **Register new bank customers:**
    - Create new customer profiles by capturing:
        - Personal information (e.g., name, address, contact details).
        - Government-issued ID verification (e.g., KYC compliance).
        - Account preferences (e.g., savings, current).
    - Assign initial account credentials.
4. **View customer information:**
    - Retrieve customer profiles, including:
        - Personal and contact details.
        - Account types and balances.
        - Transaction history.
5. **Manage customer accounts:**
    - Perform actions such as:
        - Activate or deactivate accounts.
        - Update account details (e.g., account type changes).
        - Process requests for checkbooks, debit/credit cards.
        - Handle customer issues like account freezes.

Admin Mode

Admins manage the system's overall structure, including user roles, branches, and operational settings.

1. **Sign in with login and password:**
    - Admins authenticate with high-security credentials.
    - Two-factor authentication (TFA) is mandatory.
2. **View manager and customer details:**
    - Access detailed views of:
        - Teller accounts (e.g., roles, assigned branches, activity logs).
        - Customer accounts (e.g., balances, transaction histories).
        - Generate reports for managerial review.
3. **Add or update bank branch details:**
    - Manage branch information:
        - Create new branch records with location, contact details, and codes.
        - Modify existing branch data as needed.
4. **Add or update manager details:**

- Manage teller accounts:
    - Create or modify teller profiles.
    - Assign branches or roles.
    - Set permissions and access levels

Common Functionalities Across Modes

1. **Audit Logging:**
    - Record all activities performed by users for accountability.
    - Include timestamps, user IDs, and action descriptions.
2. **Session Management:**
    - Implement secure session handling (e.g., session timeout, token invalidation).
3. **Notifications:**
    - Real-time notifications for actions like password changes, balance updates, or account modifications.
4. **Dashboard View:**
    - Personalized dashboards showing relevant KPIs(Key performance indicator), alerts, or pending actions for each user type.

Non Functional Requirements

The **non-functional requirements** for an **Online Bank Management System (OBMS)** ensure that the system performs efficiently, reliably, and securely, meeting the broader quality attributes critical for banking applications. These requirements are just as vital as functional requirements and focus on system performance, usability, security, and maintainability.

1. Performance Requirements

- **Latency:**
    - System should process transactions (e.g., balance inquiry, fund transfer) in less than 2 seconds.
- **Scalability:**
    - Handle at least 10,000 concurrent users during peak load with minimal performance degradation.
- **Throughput:**
    - Process at least 500 transactions per second (TPS) for real-time operations

2. Security Requirements

- **Data Security:**
  - Encrypt all sensitive data (e.g., account details, passwords) using AES-256 encryption.
- **Authentication:**
  - Implement multi-factor authentication (MFA) for customer and admin logins.
- **Fraud Detection:**
  - Real-time anomaly detection to identify and block suspicious activities.
- **Compliance:**
  - Ensure compliance with industry standards like PCI-DSS and GDPR.

3. Availability and Reliability

- **Uptime:**
  - Guarantee 99.99% system availability to minimize downtime.
- **Disaster Recovery:**
  - Implement a disaster recovery system to restore operations within 15 minutes of failure.
- **Fault Tolerance:**
  - Ensure the system can handle hardware or software failures without impacting critical functions.

4. Usability Requirements

- **User Experience:**
  - Ensure an intuitive interface for customers, tellers, and admins.
  - Provide user assistance (e.g., FAQs, chatbots) for self-service operations.
- **Accessibility:**
  - Ensure compliance with WCAG 2.1 standards for accessibility to support users with disabilities.

5. Maintainability and Modularity

- **Code Maintainability:**
  - Ensure code is modular and adheres to coding standards for easy updates.
- **Version Upgrades:**
  - Allow seamless integration of new features or system updates with minimal disruption.
- **Error Logging and Monitoring:**

- Include centralized logging and monitoring to identify and resolve issues quickly.

6. Auditability

- **Transaction Logs:**
  - Maintain detailed logs for all transactions for at least 7 years.
- **Access Logs:**
  - Record all login attempts and actions performed by users for audit and compliance.
- **Audit Trail:**
  - Provide a tamper-proof audit trail for all system activities.

7. Legal and Regulatory Compliance

- **Regulations:**
  - Adhere to Anti-Money Laundering (AML), Know Your Customer (KYC), and other applicable banking regulations.
- **Data Privacy:**
  - Ensure customer data is stored and processed in compliance with local laws (e.g., GDPR, CCPA).

2.Justification of Process Model:

I feel the software development lifecycle model (SDLC) which best suits my project the most is AGILE. You may ask **why does Agile work for Banking systems?**

When creating a banking system, it's not just about writing code to make a software —it's about building something secure, flexible, and user-friendly. With all the moving parts in a bank system—customers, tellers, admins, regulations—things can change quickly. Agile is perfect for this because it's all about adapting, collaborating, and delivering value step by step. The literal sense of the word agile means flexible. It forms the basis of one of the most fast and responsive to change SDLCs which puts customer over how things are being done by taking constant feedback. It prioritizes close customer collaboration over contract negotiation, involving them throughout the development process to make sure that the final product aligns with their ever-changing needs. Contracts define boundaries but real value comes from ongoing dialogue and partnership.

Lets break it down to better understand Why Agile?

1. Adapting to Changes

Imagine you're halfway through building the system, and suddenly, new banking regulations require biometric login or a new government protocol. If you're following a rigid plan (i.e a iterative model or worse a waterfall model), this is a nightmare. But with Agile, you're working in short cycles (sprints), so you can adapt quickly. You prioritize the new feature, work on it in the next sprint, and keep going without throwing everything else off balance.

For Our Online Banking Management System, this means:

- Regulations? No problem, we adjust as needed.
- New customer demands? They're added to the next sprint.

2. Delivering Early and Often

Instead of waiting months (or years) to finish the entire system, Agile gets you started with the basics first. Picture this: in the first few sprints, you build login functionality and a simple dashboard where customers can check their balance. It's not the full system yet, but it's enough to start testing and getting feedback. If the user wants to launch the system as early as possible, it can be done with agile. Later, more advanced features like transaction histories or fraud detection are added incrementally. This way, the system evolves in real-time based on what's needed most urgently.

This gives user the freedom to access core features early so that they can start benefiting from the system sooner rather than later. This creates space for the development team to welcome some early feedback for improvements ,their input helps shape future iterations, ensuring the system meets their actual needs.

3. Putting the Customer First

What's the point of building a banking system if it doesn't work the way people expect? Agile brings customers (and other stakeholders) into the process. After every sprint, you show them what's been built:

- Customers might say, "The balance view is great, but can you make it easier to find recent transactions?"
- Tellers might add, "We need a faster way to search for a customer's account."

You will have to accommodate these grievances in the next sprint.

4. Teamwork Makes the Dream Work or should I say Work a Dream

Building a Bank Management System isn't a one-person job. Developers, testers, banking experts, and even compliance officers need to work together. Agile thrives on this collaboration. Everyone sits down during sprint planning to decide what's most important. If something isn't working, they regroup in retrospectives to figure out how to improve. At the end of each day

For example:

- A developer might say, "We're spending too much time fixing login bugs."
- A tester could respond, "Let's add more automated tests to catch these earlier."

This openness and teamwork ensure the system is high-quality and everyone's on the same page.

5. Reducing Risk

Banking systems handle sensitive data, so security and reliability are non-negotiable. Agile helps manage these risks by building and testing in small, manageable pieces.

For instance:

- After adding the login feature, it's immediately tested for security vulnerabilities.
- As you introduce transaction processing, you stress-test it to ensure it works under heavy loads.

By catching issues early, you avoid big, scary failures down the line.

Lets take a **case study** to better understand the justification:

Wells Fargo's Agile Journey

- **Background**: Wells Fargo embraced Agile to improve digital banking services and enhance customer experience.
- **Implementation**:
    - Introduced Agile processes in its product development lifecycle, including regular sprints and stakeholder reviews.
    - Focused on iterative improvements to its mobile app and online platforms.
- **Results**:

- Frequent updates to the mobile app, adding features like personalized alerts and streamlined account management.
- Enhanced fraud detection systems with real-time notifications and faster response mechanisms.

Wells Fargo’s success with Agile isn’t just a case study—it’s proof that banking systems thrive in an Agile environment. By embracing short iterations, customer feedback, and collaboration, Wells Fargo transformed its processes and delivered better results.

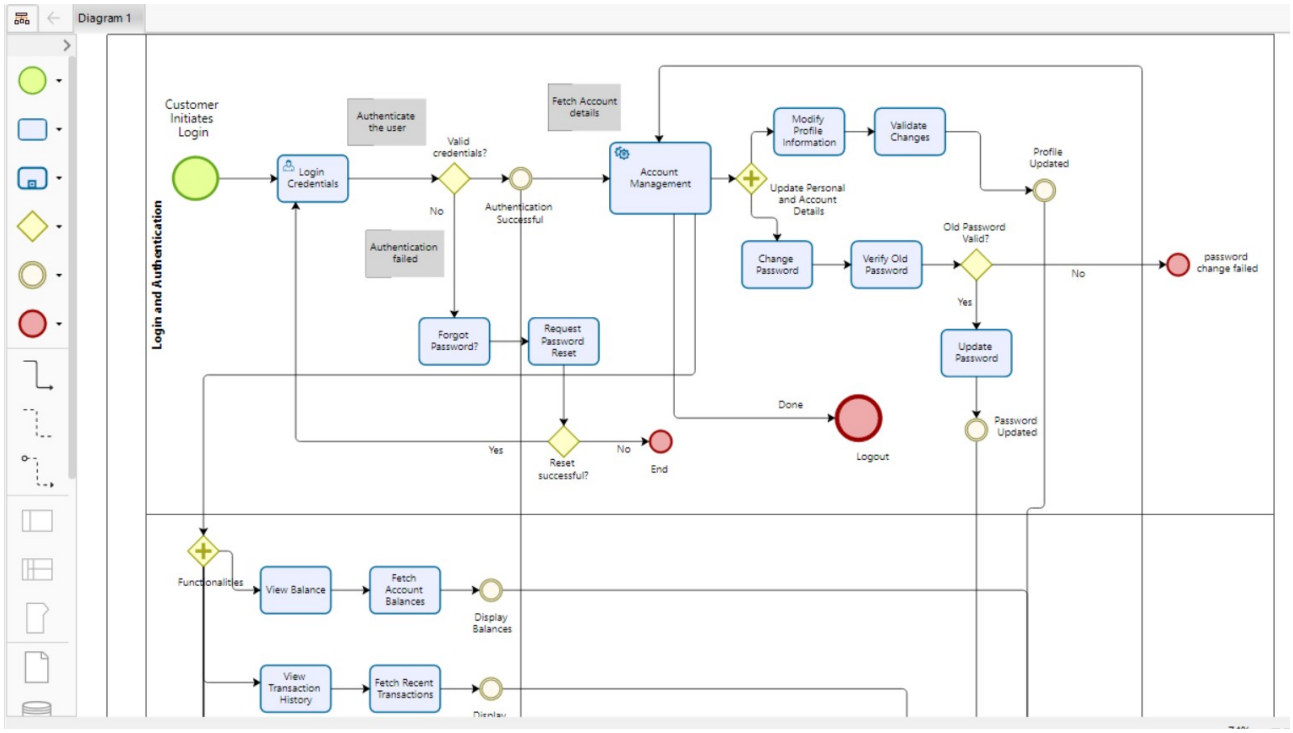
For my project, Agile ensures:

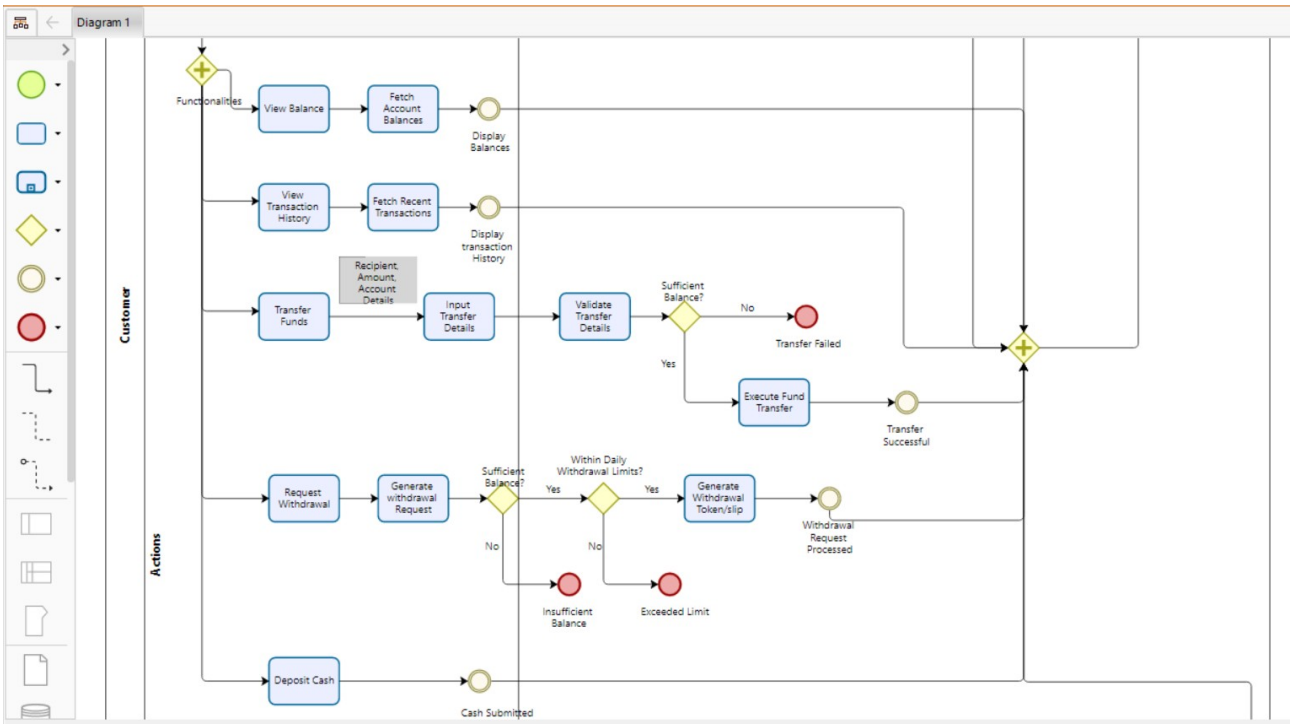
- Rapid adaptation to industry changes.
- Incremental delivery of features to build trust and value.
- A system that evolves with its users, just like Wells Fargo’s innovations.

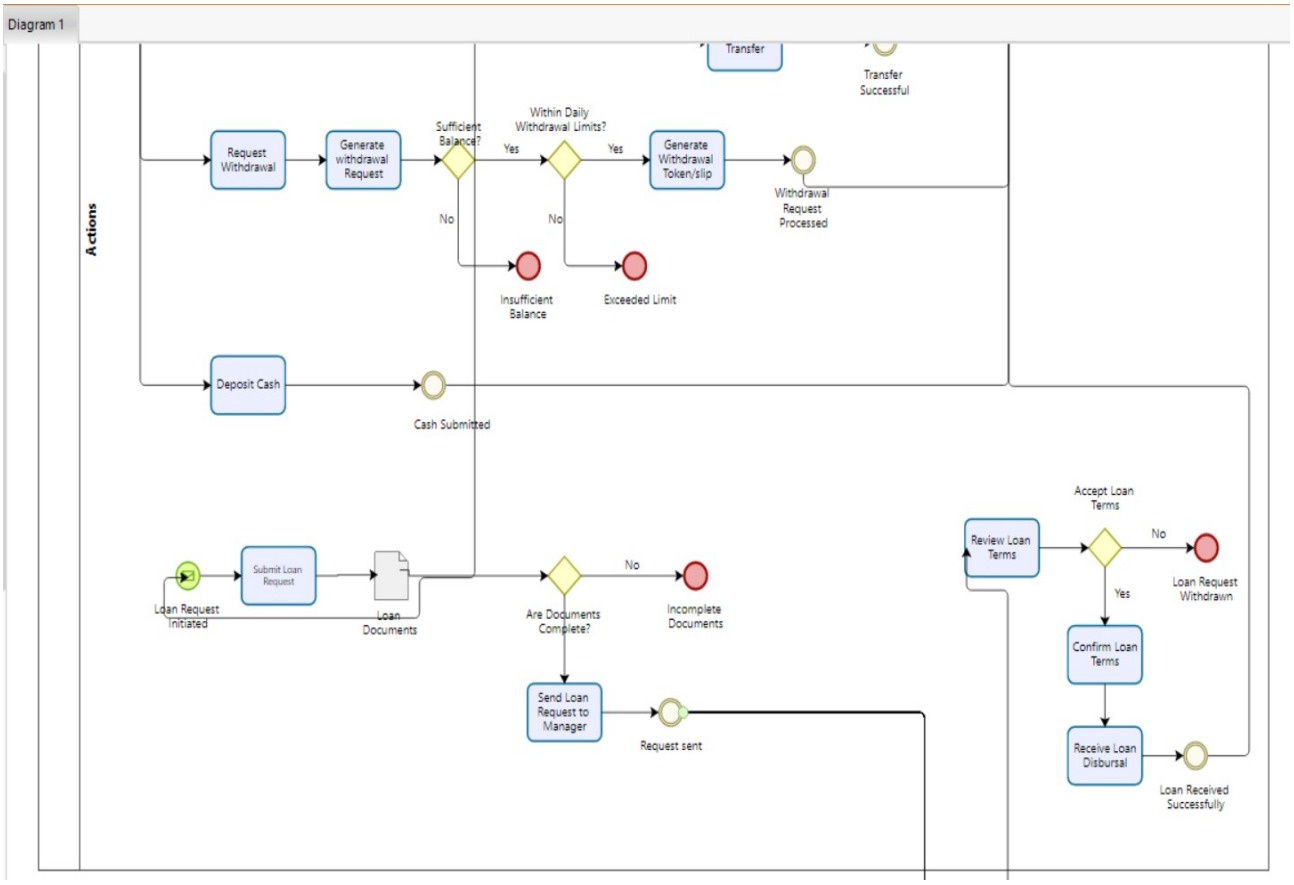
The lesson from Wells Fargo is clear: Agile doesn’t just build software—it builds better banking systems.

3. Process Diagram Output:

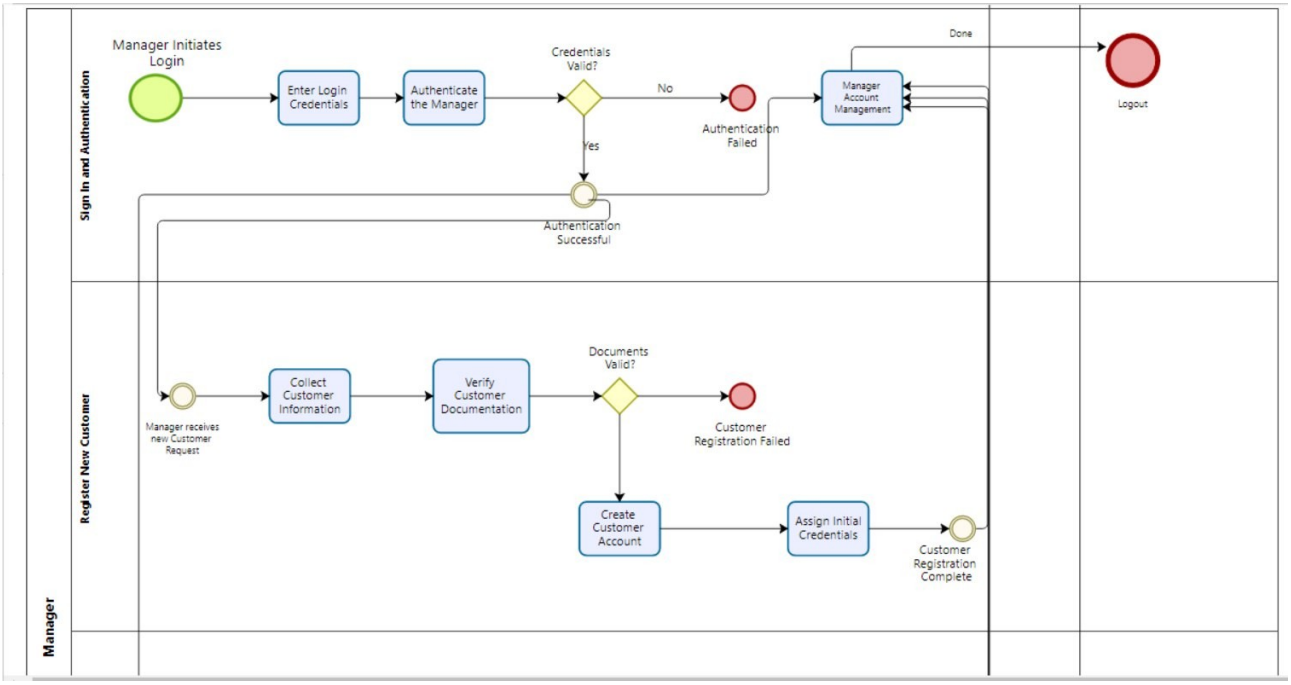
Customer View







Manager View


Manager
Sign In and Authentication
Register New Customer
Manager Initiates Login
Enter Login Credentials
Authenticate the Manager
Credentials Valid?
No
Yes
Authentication Failed
Authentication Successful
Manager Account Management
Done
Logout
Manager receives new Customer Request
Collect Customer Information
Verify Customer Documentation
Documents Valid?
Customer Registration Failed
Create Customer Account
Assign Initial Credentials
Customer Registration Complete

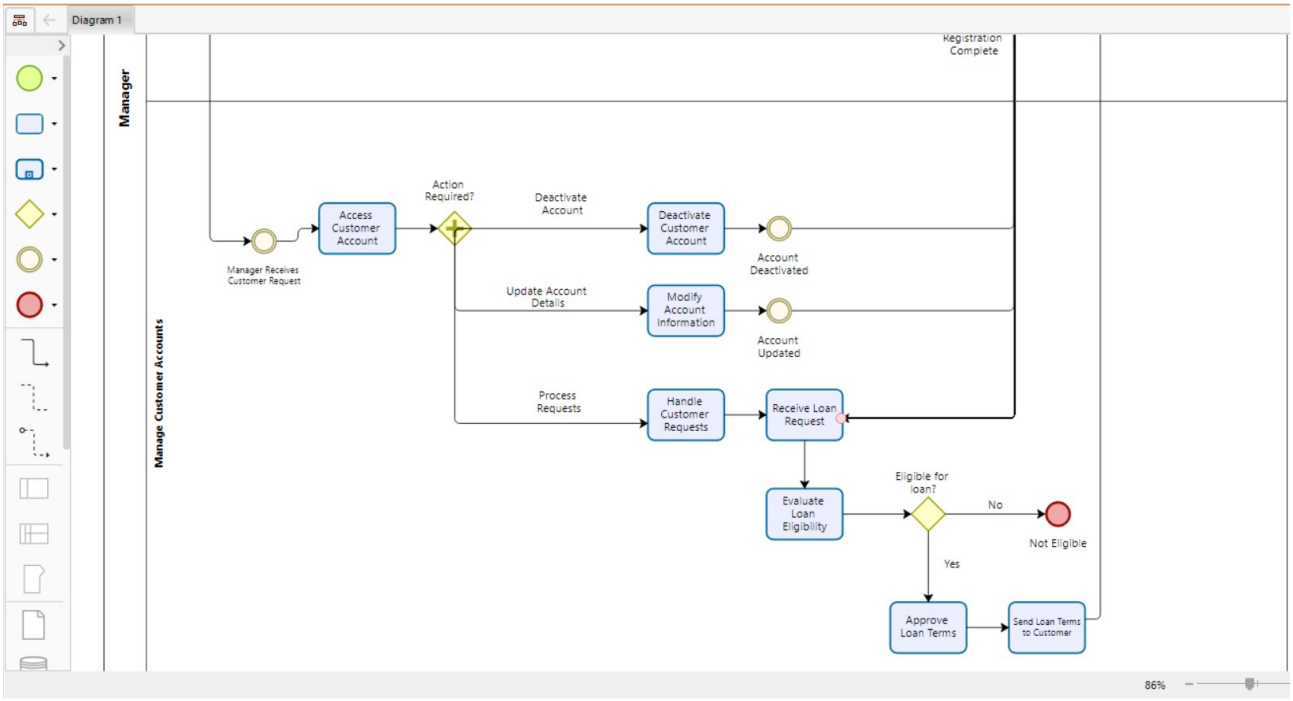
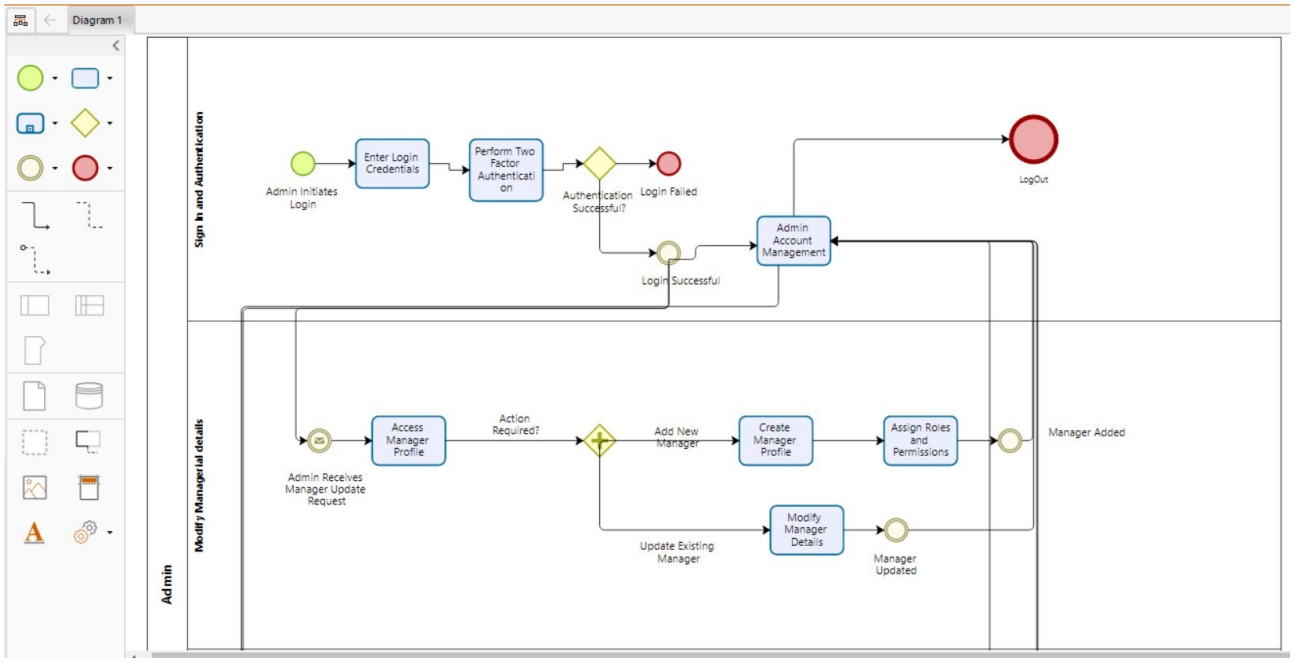
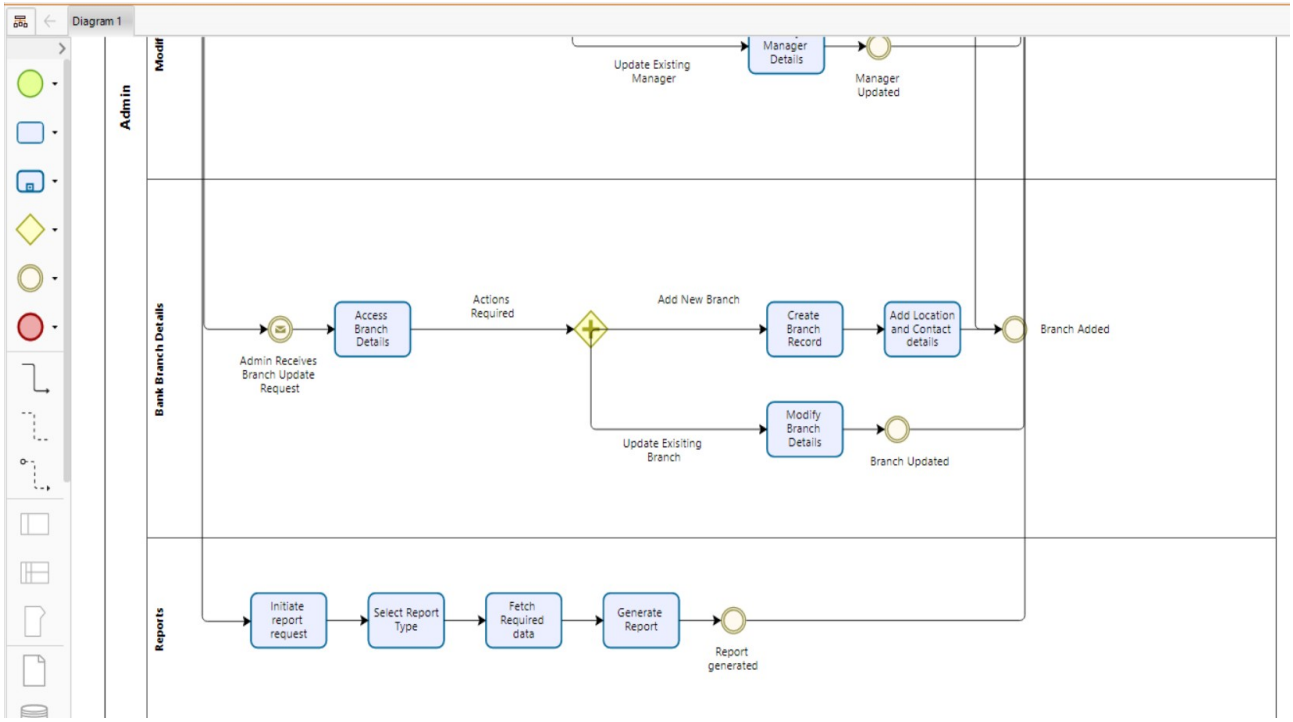


Diagram 1
Manager
Manage Customer Accounts
Registration Complete
Manager Receives Customer Request
Access Customer Account
Action Required?
Deactivate Account
Deactivate Customer Account
Account Deactivated
Update Account Details
Modify Account Information
Account Updated
Process Requests
Handle Customer Requests
Receive Loan Request
Evaluate Loan Eligibility
Eligible for loan?
No
Not Eligible
Yes
Approve Loan Terms
Send Loan Terms to Customer
86%

Admin View





4. Key Flow

Here's the **key flow** for each stakeholder:

Customer Flow

1. **Login/Authentication:**

- Input username and password.
- Optional: Multi-Factor Authentication (MFA).

2. **Perform Actions:**
    - View balance and transaction history.
    - Transfer funds to another account.
    - Update personal details or reset password.
3. **Logout:**
    - Securely terminate the session.

Manager Flow

1. **Login/Authentication:**
    - Input teller credentials.
    - Securely verify access levels.
2. **Perform Actions:**
    - Register a new customer.
    - View and update customer account details.
    - Handle deposits, withdrawals, or account closures.
3. **Logout:**
    - Securely log out to prevent unauthorized access.

Admin Flow

1. **Login/Authentication:**
    - Admin-specific credentials with high-security protocols.
2. **Perform Actions:**
    - Add or modify bank branch details.
    - View and manage teller and customer accounts.
    - Generate system reports or perform audits.
3. **Logout:**
    - Ensure complete session termination.

2. Work Break-down Structure (Process Based, Product Based, Geographic Based and Role Based) and Estimations

We have worked with four types of WBS tailored for OBMS, each serving a unique purpose:

- **Process-Based WBS:** Breaking down the project into sequential phases.
- **Product-Based WBS:** Organizing tasks by deliverables and system components.
- **Geographic-Based WBS:** Assigning tasks based on team location and regional deployment.
- **Role-Based WBS:** Dividing responsibilities according to team roles.

These WBS structures provide a clear roadmap for the project, ensuring all stakeholders have a well-defined understanding of their roles and tasks, leading to a more efficient and successful project delivery.

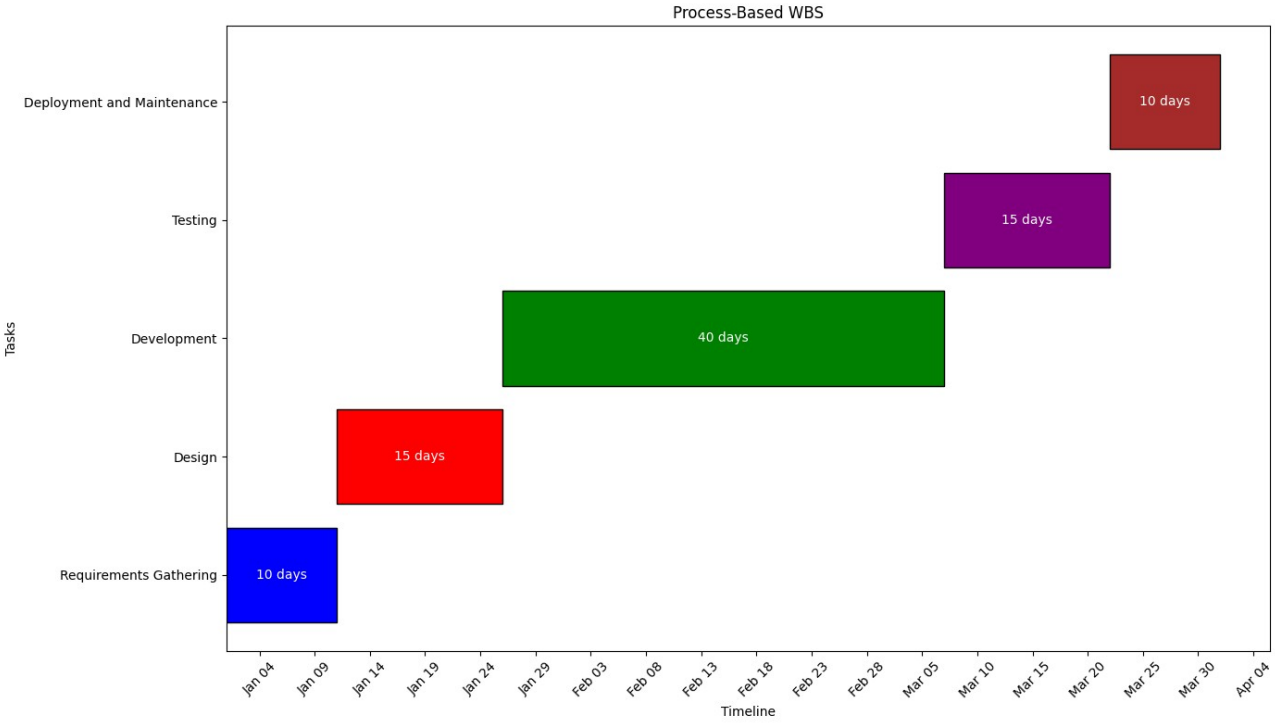
A) Process-Based WBS :

Phases and Tasks:

1. **Requirements Gathering (10 Days)**
    - Identify Stakeholders (2 Days): Admins, Managers, Customers.
    - Define System Features (3 Days): User roles and key functionalities.
    - Regulatory and Compliance Analysis (3 Days): Ensure compliance with AML, KYC, GDPR.
    - Create User Stories (2 Days): Define Agile user stories for sprint planning.
2. **Design (15 Days)**
    - High-Level Architecture (5 Days): Backend architecture, APIs.
    - UI/UX Design (5 Days): Wireframes for all portals.
    - Database Design (3 Days): Schema for users, transactions, and logs.
    - Integration Plan (2 Days): External system connections.
3. **Development (40 Days)**
    - Sprint 1: Authentication and Security (10 Days)

- Sprint 2: Customer Portal (15Days)
- Sprint 3: Manager Portal (15 Days)
- Sprint 4: Admin Portal (15 Days)

4. **Testing (15 Days)**
    - Unit Testing (5 Days): Validate modules like login and transactions.
    - Integration Testing (5 Days): Test interactions across workflows.
    - Security Testing (5 Days): Conduct penetration tests.
5. **Deployment and Maintenance (10 Days)**
    - Deployment (3 Days): Launch system on production servers.
    - Post-Deployment Testing (2 Days): Validate stability.
    - Bug Fixes and Updates (5 Days): Address post-launch issues



B) Product-Based WBS:

Organizes tasks based on deliverables or system components.

Components and Tasks Divided by Categories:

A. User Interfaces (30 Days)

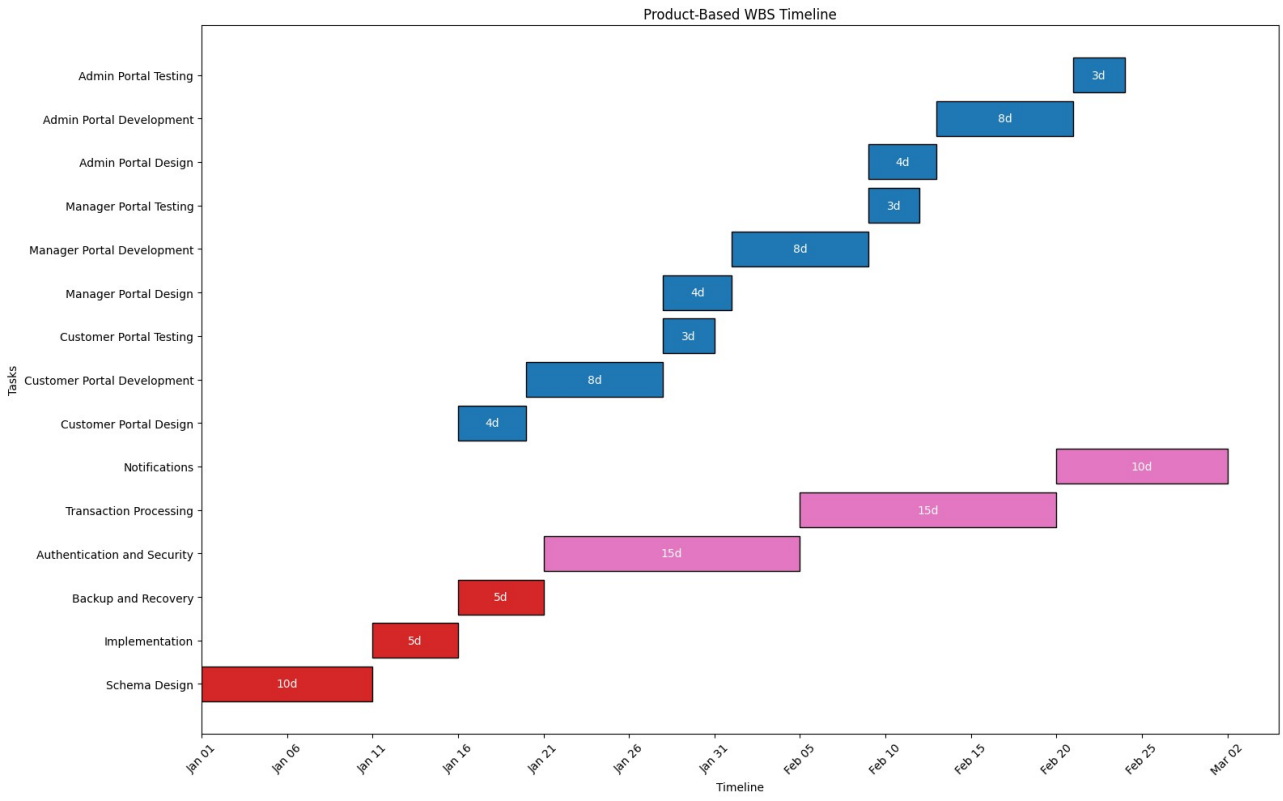
1. **Customer Portal (15 Days)**
    - Design (4 Days): Create wireframes for login, balance inquiry, and transactions.
    - Develop (8 Days): Build the frontend using appropriate frameworks.
    - Test (3 Days): Validate functionality and responsiveness.
2. **Manager Portal (15 Days)**
    - Design (4 Days): Create interfaces for account approvals and management.
    - Develop (8 Days): Build pages for registration and transaction reviews.
    - Test (3 Days): Conduct UI testing.
3. **Admin Portal (15 Days)**
    - Design (4 Days): Create dashboards for branch and role management.
    - Develop (8 Days): Build analytics and reporting interfaces.
    - Test (3 Days): Validate admin workflows.

B. Backend Modules (40 Days)

1. **Authentication and Security (15 Days)**
    - Role-Based Login System (5 Days): Implement authentication logic.
    - Multi-Factor Authentication (4 Days): Add OTP/email verification.
    - Encryption (3 Days): Secure sensitive data.
    - Penetration Testing (3 Days): Conduct security audits.
2. **Transaction Processing (15 Days)**
    - Develop Fund Transfers (7 Days): Internal and external workflows.
    - Withdrawals and Deposits (5 Days): Simulate operations.
    - Logging Mechanisms (3 Days): Ensure robust transaction records.
3. **Notifications (10 Days)**
    - Real-Time Alerts (6 Days): Implement SMS and email notifications.
    - System Alerts (4 Days): Add alerts for admin users.

C. Database (20 Days)

1. **Schema Design (10 Days)**
    - User Table (3 Days): Structure for user accounts.
    - Transaction Table (5 Days): Detailed schema for logging transactions.
    - Logs Table (2 Days): Security and activity logs.
2. **Implementation and Optimization (5 Days)**
    - Configure Databases (3 Days): Set up environments.
    - Optimize Indexing (2 Days): Ensure efficient queries.
3. **Backup and Recovery (5 Days)**
    - Backup Strategy (3 Days): Regular backups for disaster recovery.
    - **Recovery Protocols (2 Days): Test and validate recovery processes.**



C) Geographic-Based WBS

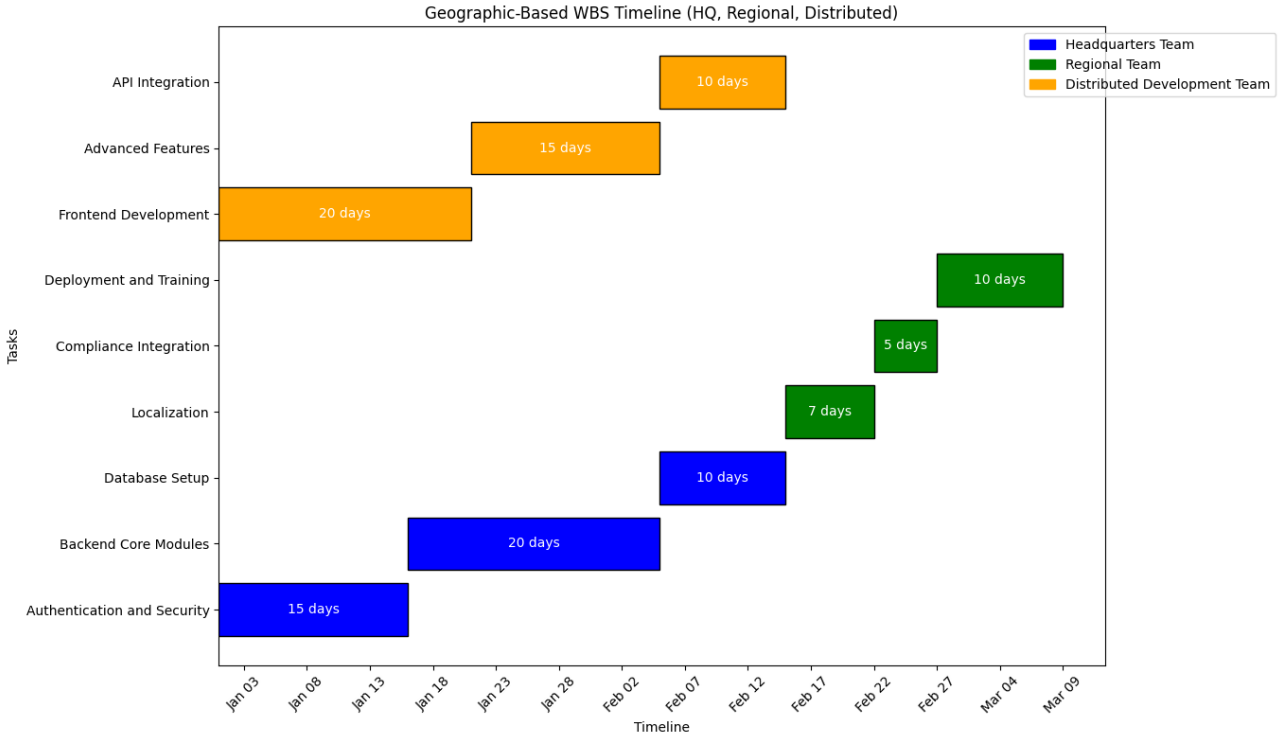
Organizes tasks by team location and regional implementation.

Team Locations and Tasks:

1. **Headquarters Team (45 Days)**

- Authentication and Security (15 Days): Develop secure login.
- Backend Core Modules (20 Days): Transactions and notifications.
- Database Setup (10 Days): Schema design and indexing.

2. **Regional Teams (22 Days per Region){Overlaps}**
    - Localization (7 Days): Translate UI and workflows.
    - Compliance Integration (5 Days): Align with regional regulations.
    - Deployment and Training (10 Days): Launch system and train staff.
3. **Distributed Development Teams (45 Days)**
    - Frontend Development (20 Days): Customer, Manager, and Admin portals.
    - Advanced Features (15 Days): Fraud detection and dashboards.
    - **API Integration (10 Days): Payment gateways and external systems**

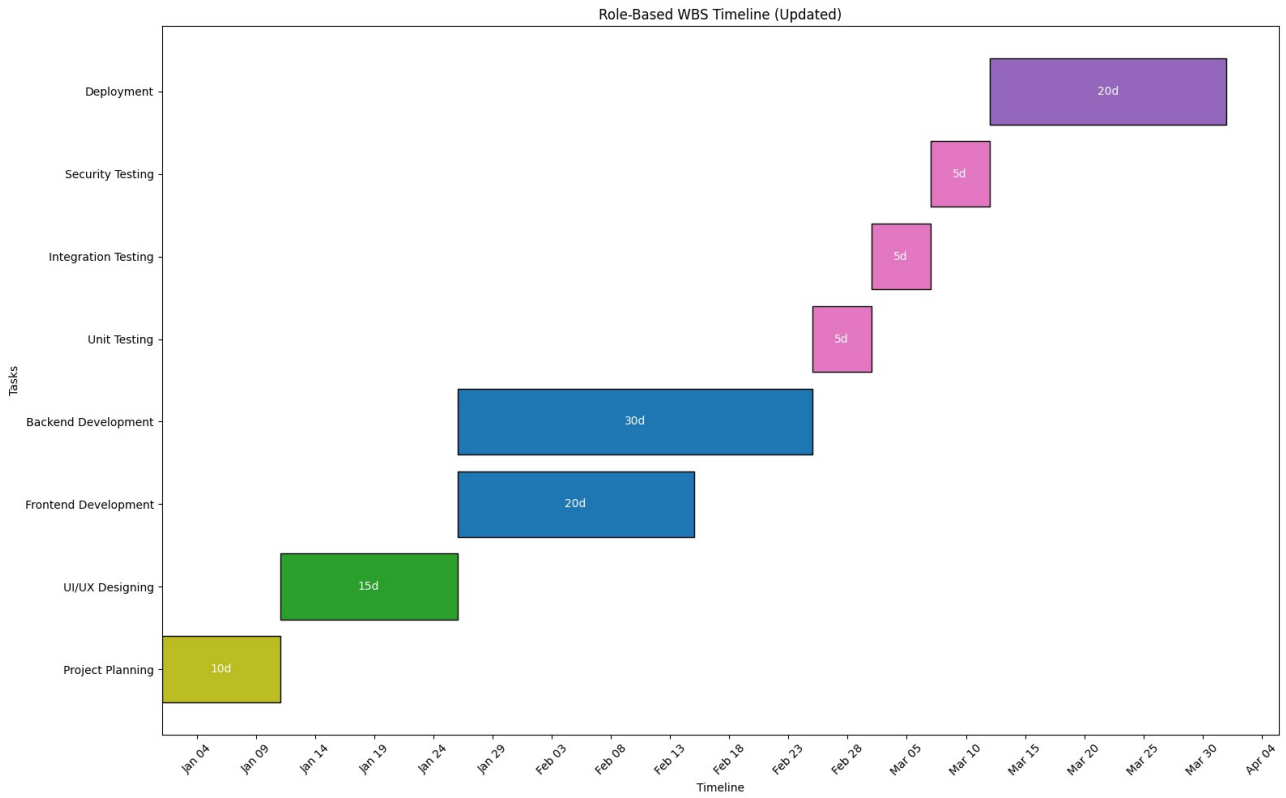


D) Role-Based WBS

Organizes tasks by roles in the project.

Roles and Tasks:

1. **Developers (30 Days)**
    - Frontend Development (20 Days): Implement portals.
    - Backend Development (30 Days): Build authentication, APIs, and notifications.
2. **Testers (15 Days)**
    - Unit Testing (5 Days): Validate individual modules.
    - Integration Testing (5 Days): Simulate workflows.
    - Security Testing (5 Days): Penetration tests and encryption validation.
3. **Project Managers (10 Days)**
    - Planning and Monitoring (8 Days): Develop timeline and track progress.
    - Stakeholder Communication (2 Days): Sprint reviews and feedback.
4. **UI/UX Designers (15 Days)**
    - Wireframing and Prototyping (10 Days): Create interactive wireframes.
    - Design Implementation (7 Days): Finalize themes and layouts.
5. **Administrators/DevOps (20Days)**
    - Deployment (10 Days): Set up servers.
    - Backup and Recovery (5 Days): Implement disaster recovery.
    - **Monitoring and Maintenance (5 Days): Post-deployment monitoring.**



Process-Based WBS Dependencies

1. **Design depends on Requirements Gathering**:
    - The design phase can only begin after stakeholders' needs and system features are fully defined.
2. **Development depends on Design**:
    - Development tasks rely on finalized architecture, UI/UX designs, and database schemas.
3. **Testing depends on Development**:
    - Testing can only start after core modules and workflows are implemented.
4. **Deployment depends on Testing**:
    - **Deployment occurs only after successful completion of all testing phases.**

Product-Based WBS Dependencies

1. **User Interfaces depend on Backend Modules**:
    - UI tasks like Customer Portal development require backend APIs to be functional.
2. **Notifications depend on Transaction Processing**:

    - Real-time alerts need data from transaction workflows.
3. **Database tasks influence Backend Modules**:
    - Backend logic depends on schema design and database readiness.
4. **Backend Modules influence User Interfaces and Notifications**:
    - **Core authentication and transaction features must be operational for UI and notification systems.**

Geographic-Based WBS Dependencies

1. **Regional Teams depend on Headquarters Team**:
    - Localization, compliance integration, and deployment tasks rely on HQ completing Authentication, Backend Core Modules, and Database Setup.
2. **Distributed Teams depend on Backend APIs from HQ**:
    - Frontend development and API integration tasks require backend readiness.
3. **Deployment depends on Localization and Compliance**:
    - **Regional deployment starts only after local compliance and translations are validated.**

Role-Based WBS Dependencies

1. **Testers depend on Developers**:
    - Testing tasks like unit and integration testing rely on modules developed by the developers.
2. **UI/UX Designers influence Developers**:
    - Developers require finalized designs for frontend implementation.
3. **Administrators/DevOps depend on Developers and Testers**:
    - Deployment and monitoring tasks require developed and tested modules.
4. **Project Managers oversee all phases**:
    - **Effective planning and monitoring ensure that dependencies across all roles are managed and resolved.**

3. Requirement modelling using Entity Relationship Diagram

A) Entities and Attributes

1. Customer

- Attributes: `CustomerID (PK)`, `Name`, `Email`, `PhoneNumber`, `Address`, `DateOfBirth`, `AccountType`, `RegistrationDate`, `Password`.

2. Account

- Attributes: `AccountID (PK)`, `CustomerID (FK)`, `BranchID (FK)`, `AccountType`, `Balance`, `OpeningDate`, `Status`.

3. Transaction

- Attributes: `TransactionID (PK)`, `AccountID (FK)`, `TransactionDate`, `Amount`, `TransactionType`, `BalanceAfterTransaction`, `TransactionMode`, `TransactionDescription`.

4. Branch

- Attributes: `BranchID (PK)`, `BranchName`, `BranchAddress`, `PhoneNumber`, `ManagerID (FK)`.

5. Manager

- Attributes: `ManagerID (PK)`, `Name`, `Email`, `PhoneNumber`, `AssignedBranchID (FK)`.

6. Loan

- Attributes: `LoanID (PK)`, `CustomerID (FK)`, `BranchID (FK)`, `LoanType`, `LoanAmount`, `InterestRate`, `ApprovalDate`, `Status`, `PaymentSchedule`.

7. Employee

- Attributes: `EmployeeID (PK)`, `Name`, `Email`, `PhoneNumber`, `BranchID (FK)`, `Role`, `JoiningDate`, `Salary`.

8. Service Request

- Attributes: `RequestID (PK)`, `CustomerID (FK)`, `RequestDate`, `RequestType`, `Status`, `AssignedEmployeeID (FK)`.

9. Card

- Attributes: `CardID (PK)`, `CustomerID (FK)`, `CardType`, `CardNumber`, `ExpiryDate`, `CVV`, `IssueDate`, `Status`.

10. Payment Gateway

- Attributes: `GatewayID (PK)`, `GatewayName`, `TransactionID (FK)`, `GatewayStatus`, `ProcessingFee`.

B) Relationships and Cardinality

1. Customer - Account

- **Relationship**: A customer can have one or more accounts. Each account belongs to one customer.
- **Cardinality**: 1:N.

2. Account - Transaction

- **Relationship**: An account can have multiple transactions. Each transaction is associated with one account.
- **Cardinality**: 1:N.

3. Customer - Loan

- **Relationship**: A customer can have one or more loans. Each loan belongs to one customer.
- **Cardinality**: 1:N.

4. Branch - Account

- **Relationship**: A branch can host multiple accounts. Each account is associated with one branch.
- **Cardinality**: 1:N.

5. Branch - Loan

- **Relationship**: A branch can issue multiple loans. Each loan is associated with one branch.
- **Cardinality**: 1:N.

6. Branch - Manager

- **Relationship**: A branch is managed by one manager. A manager can be assigned to one branch.
- **Cardinality**: 1:1.

7. Branch - Employee

- **Relationship**: A branch can employ multiple employees. Each employee is assigned to one branch.
- **Cardinality**: 1:N.

8. Customer - Service Request

- **Relationship**: A customer can raise one or more service requests. Each request is linked to one customer.
- **Cardinality**: 1:N.

9. Service Request - Employee

- **Relationship**: A service request can be assigned to one employee. An employee can handle multiple service requests.
- **Cardinality**: N:1.

10. Customer - Card

- **Relationship**: A customer can have multiple cards (credit, debit). Each card is issued to one customer.
- **Cardinality**: 1:N.

11. Transaction - Payment Gateway

- **Relationship**: A transaction can be processed through one payment gateway. A payment gateway handles multiple transactions.
- **Cardinality**: 1:N.

C) Final ER Diagram Details

Primary Keys (PK):

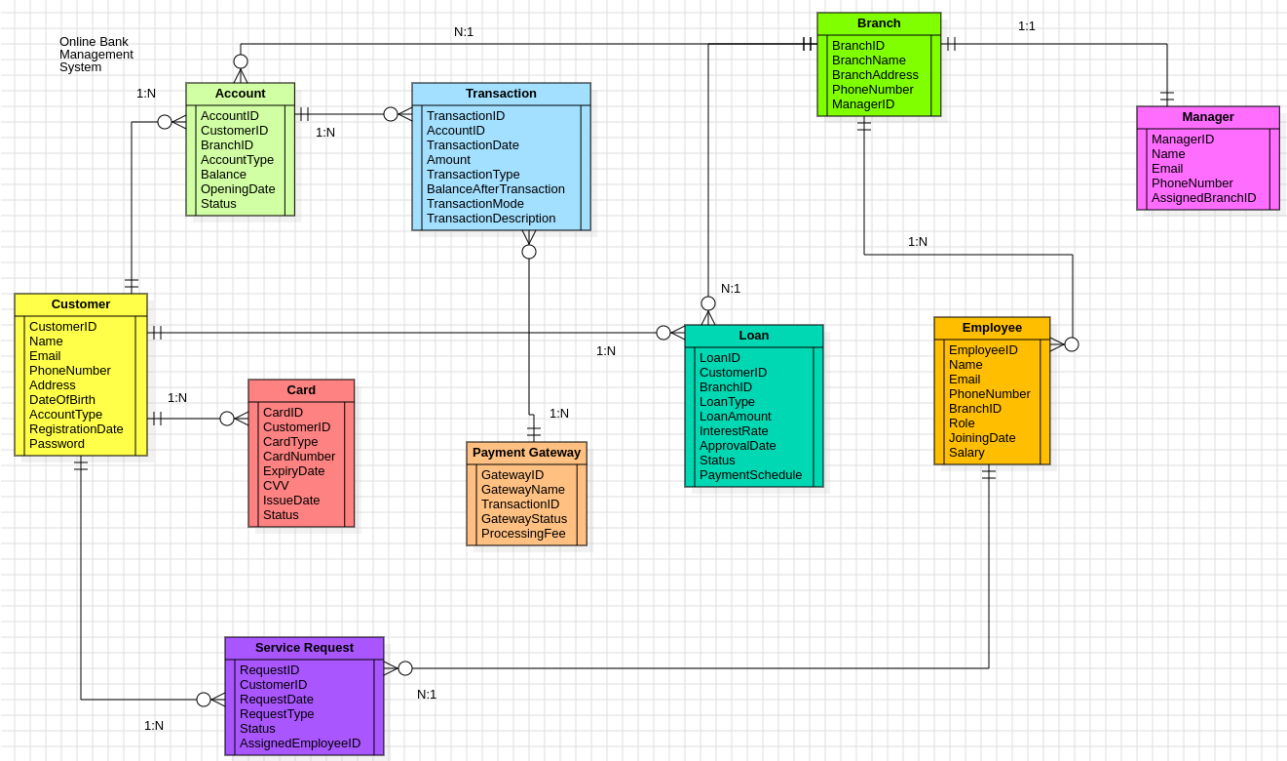
- `CustomerID` for **Customer**.
- `AccountID` for **Account**.
- `TransactionID` for **Transaction**.
- `BranchID` for **Branch**.
- `ManagerID` for **Manager**.

- `LoanID` for **Loan**.
- `EmployeeID` for **Employee**.
- `RequestID` for **Service Request**.
- `CardID` for **Card**.
- `GatewayID` for **Payment Gateway**.

Foreign Keys (FK):

- `CustomerID` in **Account**, **Loan**, **Service Request**, **Card**.
- `BranchID` in **Account**, **Loan**, **Employee**.
- `AccountID` in **Transaction**.
- `TransactionID` in **Payment Gateway**.
- `AssignedEmployeeID` in **Service Request**.
- `ManagerID` in **Branch**.

E-R Diagram



4. Requirement modelling using Context flow diagram, DFD (Functional Modeling)

This document outlines the entities, processes, and data flows involved in OBMS, focusing on structural modeling for clarity and scalability. Additional refinements include expanded entities, relationships, and granular detail in DFD levels for complete coverage.

Level 0 Data Flow Diagram (DFD)

1. Components

- **Processes**: These represent the core system functionalities.
    - **Online Bank Management System**: The main process managing all transactions, customer accounts, loans, and external communication.
- **External Entities**: Represent the sources and destinations of data interacting with the system.
    - **Manager**: Reviews flagged transactions and provides loan approval details.
    - **Customer**: Provides login credentials and transaction requests; receives transaction status, balance updates, and account details.
    - **Branch**: Supplies branch information and interacts with the system for configurations.
    - **Admin**: Configures branch settings and receives performance reports.
    - **Third-Party Services**: Handles payment gateway processing and SMS/email notifications.

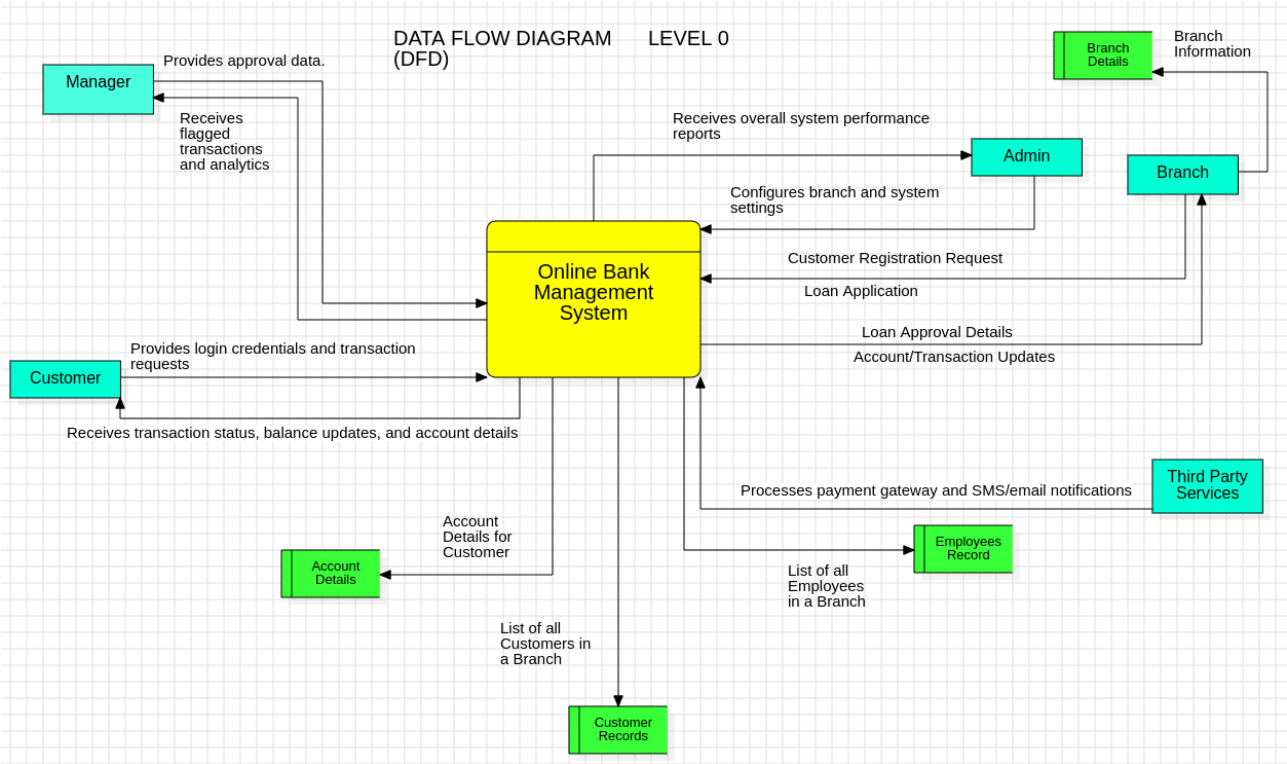
- **Data Stores**: Represent the repositories of data within the system.
    - **Account Details**: Stores detailed information about customer accounts.

- **Customer Records**: Maintains a list of all customers associated with a branch.
- **Employees Record**: Contains the details of all employees in a branch.
- **Branch Record** : Provides Information about the branch of the bank.

- **Data Flows**: The **data flows** between the entities, processes, and data stores are as follows:

1. **Manager ↔ Online Bank Management System**:
    - Sends **Approval Data** for loans.
    - Receives **Flagged Transactions and Analytics Reports**.
2. **Customer ↔ Online Bank Management System**:
    - Sends **Login Credentials** and **Transaction Requests**.
    - Receives **Transaction Status**, **Balance Updates**, and **Account Details**.
3. **Branch ↔ Online Bank Management System**:
    - Provides **Branch Details** and **Configuration Requests**.
    - Receives **System Configuration Updates** and **Reports**.
4. **Admin ↔ Online Bank Management System**:
    - Sends **Branch Settings Configurations** and **System Updates**.
    - Receives **System Performance Reports**.
5. **Third-Party Services ↔ Online Bank Management System**:
    - Processes **Payment Gateway Transactions**.
    - Sends **SMS/Email Notifications**.
6. **Online Bank Management System ↔ Account Details**:
    - Updates and retrieves **Account Details for Customers**.
7. **Online Bank Management System ↔ Customer Records**:
    - Accesses **Customer Registration Information** and updates customer-related data.

8. **Online Bank Management System ↔ Employees Record**:
    - Accesses **List of Employees in a Branch**.



Level 1 Data Flow Diagram (DFD)

The Level 1 DFD provides a deeper breakdown of the processes and interactions from the Level 0 DFD, showing the granularity of data management in a banking system.

1. Components

1. **Processes**:
    - **Customer Management**: Handles customer registration, profile updates, and other customer-related tasks.
    - **Account Management**: Manages customer accounts, including balance updates, account status, and transaction requests.
    - **Transaction Management**: Processes transaction requests and logs transaction details for reporting.
    - **Loan Management**: Handles loan applications, approvals/rejections, and updates loan details.

- **Analytics and Reporting**: Generates reports and performance analytics.
- **Notification System**: Sends notifications to customers and other entities based on events such as loan approvals, transactions, and account updates.
- **Online Bank Management System**: Acts as the central controller, integrating all processes and external entities.

2. **External Entities**:

- **Customer**: Interacts with the system for profile management, transactions, account updates, and loan applications.
- **Manager**: Provides loan approval/rejection instructions and reviews reports.
- **Branch**: Submits customer registration requests and branch details.
- **Admin**: Requests performance reports and configures system settings.
- **Third Party Services**: Facilitates notification delivery and integrates with external services for smooth operation.

3. **Data Stores**:

- **Account Details**: Stores account information for customers.
- **Customer Record**: Contains all customer-related data.
- **Loan Details**: Maintains details of loans including type, interest rate, and payment history.
- **Transaction Details**: Logs all transactions for reporting and auditing.
- **Employee Record**: Stores employee data for operational purposes.
- **Notification Logs**: Records details of sent notifications

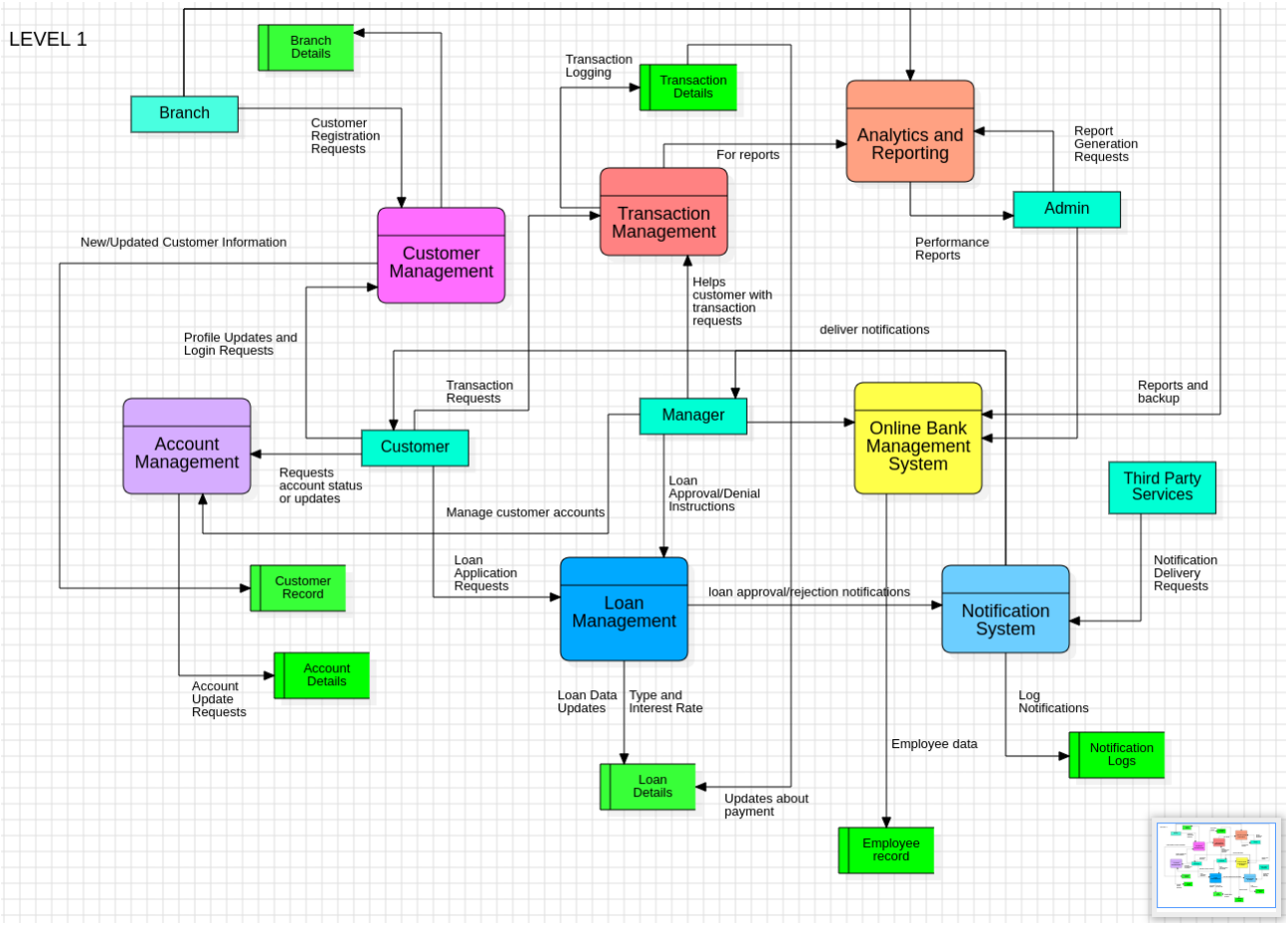
2. Data Flows

The **data flows** connecting the processes, entities, and data stores are as follows:

1. **Customer ↔ Customer Management**:
    - Sends **Customer Registration Requests**, **Profile Updates**, and **Login Requests**.
    - Receives **New/Updated Customer Information**.
2. **Customer ↔ Account Management**:
    - Sends **Transaction Requests** and **Account Status/Update Requests**.
    - Receives **Account Details and Updates**.
3. **Customer ↔ Loan Management**:
    - Sends **Loan Application Requests**.
    - Receives **Loan Approval/Denial Notifications**.
4. **Branch ↔ Customer Management**:
    - Sends **Customer Registration Requests** and **Branch Details**.
5. **Customer Management ↔ Customer Record**:
    - Stores and retrieves **Customer Information**.
6. **Account Management ↔ Account Details**:
    - Updates and retrieves **Account Information**.
7. **Transaction Management ↔ Transaction Details**:
    - Logs **Transaction Data** and retrieves it for reporting.
8. **Loan Management ↔ Loan Details**:
    - Updates **Loan Data** such as type, interest rate, and payment status.
9. **Loan Management ↔ Manager**:
    - Sends **Loan Approval/Denial Instructions**.
10. **Analytics and Reporting ↔ Admin**:
    - Sends **Performance Reports**.
    - Receives **Report Generation Requests**.
11. **Analytics and Reporting ↔ Transaction Management**:
    - Retrieves **Transaction Logs** for reporting.

12. **Notification System ↔ Customer**:
    - Sends **Notifications** such as loan status or account updates.
13. **Notification System ↔ Notification Logs**:
    - Logs **Notification History** for auditing.
14. **Notification System ↔ Third-Party Services**:
    - Sends **Notification Delivery Requests**.
15. **Online Bank Management System ↔ Analytics and Reporting**:
    - Receives **Reports and Analytics**.
16. **Online Bank Management System ↔ Notification System**:
    - Sends **Notification Requests** for delivery to customers or managers.
17. **Online Bank Management System ↔ Loan Management**:
    - Sends and receives **Loan Application Data** and **Loan Updates**.
18. **Online Bank Management System ↔ Employee Record**:
    - Accesses **Employee Data** for operational purposes.
19. **Manager ↔ Online Bank Management System**:
    - Receives **Reports and Analytics**.
    - Sends **Approval/Denial Instructions**.
20. **Branch ↔ Online Bank Management System**:
    - Provides **Branch Details** for configuration and updates.

Level 1 DFD



Level 2 Data Flow Diagram (DFD)

1. Components

1. **Processes**:

- **Customer Management Subsystem**:
  - Customer Registration
  - Profile Updates
  - Login Management
- **Account Management Subsystem**:
  - Account Information Updates
  - Balance Inquiry
  - Account Transaction Management
- **Transaction Management Subsystem**:
  - Transaction Logging
  - Fraud Monitoring (if applicable)

- **Loan Management Subsystem**:
    - Loan Applications
    - Loan Approval/Denial Processing
    - Loan Repayment Management
- **Analytics and Reporting Subsystem**:
    - Performance Reporting
    - Transaction Analysis
- **Notification System Subsystem**:
    - Notification Creation
    - Notification Delivery
    - Notification History Management

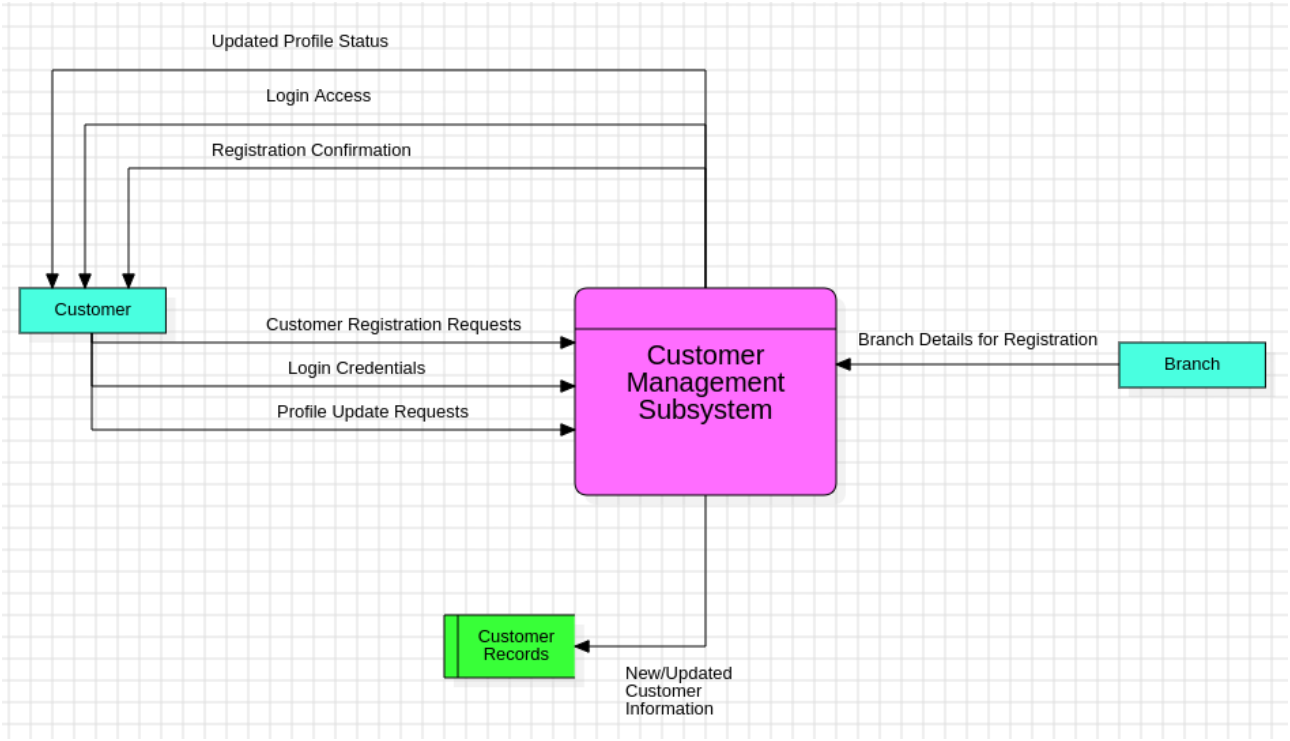
2. **External Entities**:

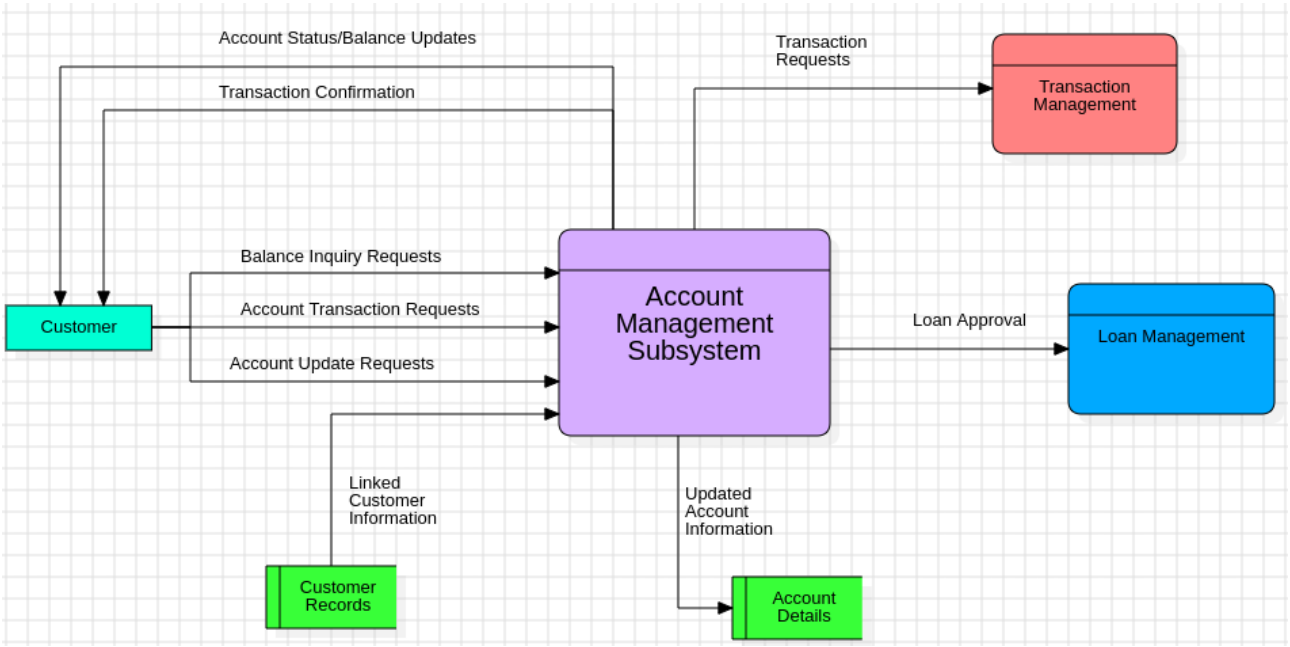
    - **Customer**: Initiates transactions, updates profiles, and applies for loans.
    - **Manager**: Reviews loan requests and provides approvals/rejections.
    - **Branch**: Provides branch-specific data, including customer registration requests.
    - **Admin**: Oversees reporting and system performance.
    - **Third-Party Services**: Delivers notifications (e.g., SMS, email).

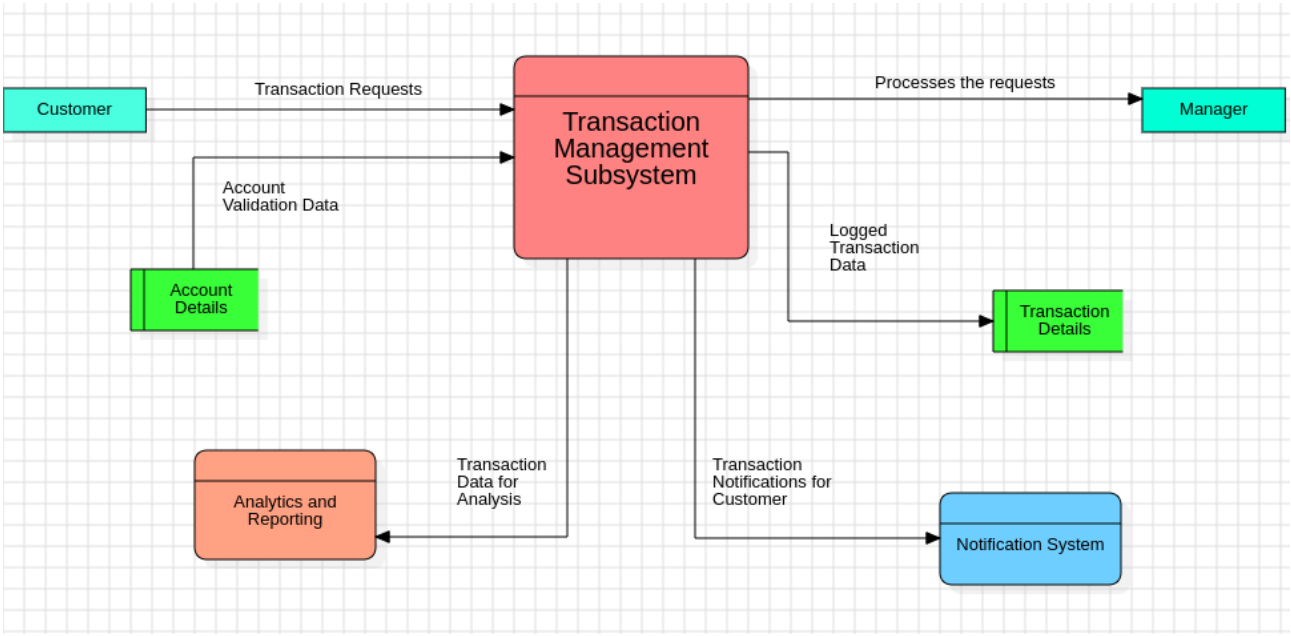
3. **Data Stores**:

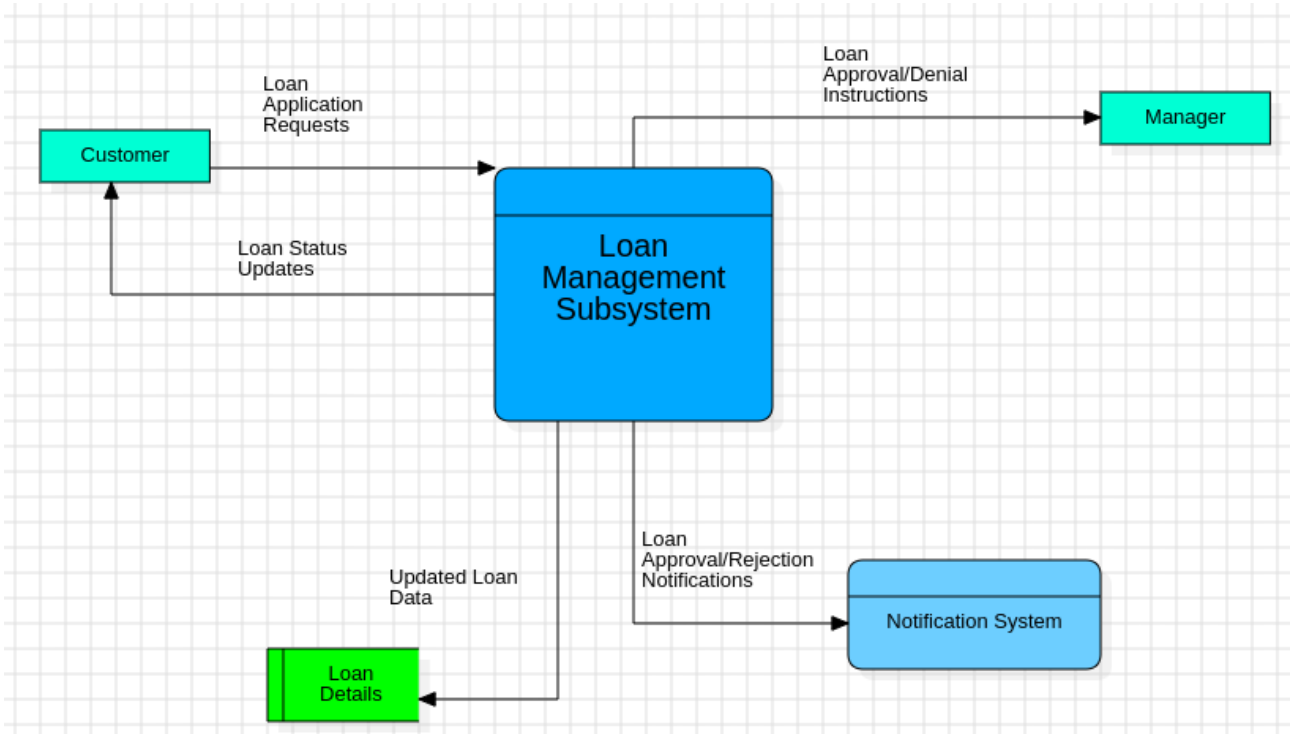
    - **Customer Records**: Stores customer details.
    - **Account Details**: Contains account-specific information.
    - **Loan Details**: Maintains loan-related data.
    - **Transaction Details**: Logs all transactions.
    - **Employee Records**: Tracks employee data.
    - **Notification Logs**: Stores notification history.

Level 2 DFD


Updated Profile Status
Login Access
Registration Confirmation
Customer
Customer Registration Requests
Login Credentials
Profile Update Requests
Customer Management Subsystem
Branch Details for Registration
Branch
Customer Records
New/Updated Customer Information


Account Status/Balance Updates
Transaction Confirmation
Transaction Requests
Transaction Management
Customer
Balance Inquiry Requests
Account Transaction Requests
Account Update Requests
Account Management Subsystem
Loan Approval
Loan Management
Linked Customer Information
Updated Account Information
Customer Records
Account Details


Customer
Transaction Requests
Transaction Management Subsystem
Processes the requests
Manager
Account Validation Data
Account Details
Logged Transaction Data
Transaction Details
Analytics and Reporting
Transaction Data for Analysis
Transaction Notifications for Customer
Notification System


Loan Application Requests
Loan Approval/Denial Instructions
Customer
Manager
Loan Status Updates
Loan Management Subsystem
Loan Approval/Rejection Notifications
Updated Loan Data
Notification System
Loan Details

The series of Data Flow Diagrams (DFDs) – Level 0, Level 1, and Level 2 – collectively provide a comprehensive representation of the **Online Bank Management System** and its processes. Each level of abstraction offers progressively deeper insights into the flow of data, interactions between entities, and system functionality.

- **Level 0** gives an overview of the system's main components, including external entities (customers, managers, branches, admin, and third-party services), core processes, and the exchange of high-level data flows. It highlights the system’s primary purpose: enabling transactions, managing customer accounts, and processing loans.
- **Level 1** breaks down the primary processes into subsystems such as Customer Management, Account Management, Loan Management, Transaction Management, Analytics, and Notification Systems. It demonstrates the intricate interaction between these subsystems, external entities, and data stores, ensuring system functionality and responsiveness.
- **Level 2 further decomposes the processes into their individual components and specific operations, showcasing the detailed data flows within subsystems. It provides clarity on how customer requests are handled, accounts are updated, loans are processed, and notifications are delivered.**

These diagrams together depict the **logical architecture** of the Online Bank Management System, emphasizing the efficient and secure flow of data between various components

This structured visualization not only facilitates seamless operations but also allows for scalability, adaptability, and troubleshooting within the system.

5. Requirement modelling using State Transition Diagram (Behavioral Modeling)

A) Introduction

The **State Transition Diagram** represents the dynamic behavior of the Online Bank Management System by illustrating different states, transitions, and events. It provides a clear understanding of how the system reacts to various user actions and system processes.

B) Identified States

The following are the states included in the system:

- **Idle State**: The system is in an idle state before any user interaction.
- **User Authentication**: The system verifies user credentials.
- **Register**: Users register for a new account.
- **Login**: Users successfully log in to the system.
- **Customer Dashboard**: The main interface where users can perform banking operations.
- **Transaction Processing**: Handles user transactions like withdrawal, deposit, and transfers.
- **Withdrawal**: Users withdraw money.
- **Deposit**: Users deposit money.
- **Transfers**: Users transfer funds between accounts.
- **Loan Application**: Users apply for a loan.
- **Managerial Review**: Loan applications are reviewed for approval or rejection.
- **Notification System**: Sends transaction or loan status notifications.

- **System Logout**: The user logs out or the session expires.
- **Final State**: The system process ends.

C) Events and Transitions

Each transition is triggered by an event, leading from one state to another:

Current State	Event (Triggering Action)	Next State
Idle State	Initiate Login	User Authentication
Idle State	Register New User	Register
Register	Registration Complete	Login
User Authentication	Authentication Failed	Idle State
User Authentication	Successful Login	Customer Dashboard
Customer Dashboard	Initiate Transaction	Transaction Processing
Customer Dashboard	Apply for Loan	Loan Application
Customer Dashboard	User Logs Out	System Logout
Transaction Processing	Choose Withdrawal	Withdrawal
Transaction Processing	Choose Deposit	Deposit
Transaction Processing	Choose Transfers	Transfers
Withdrawal, Deposit, Transfers	Transaction Successful / Failed	Notification System
Loan Application	Submit Loan Request	Managerial Review
Managerial Review	Loan Approved / Rejected	Notification System
Notification System	Notification Sent	Customer Dashboard
Notification System	Notification Failed	Customer Dashboard
System Logout	Session Expired	Final State

D) Explanation of State Transitions

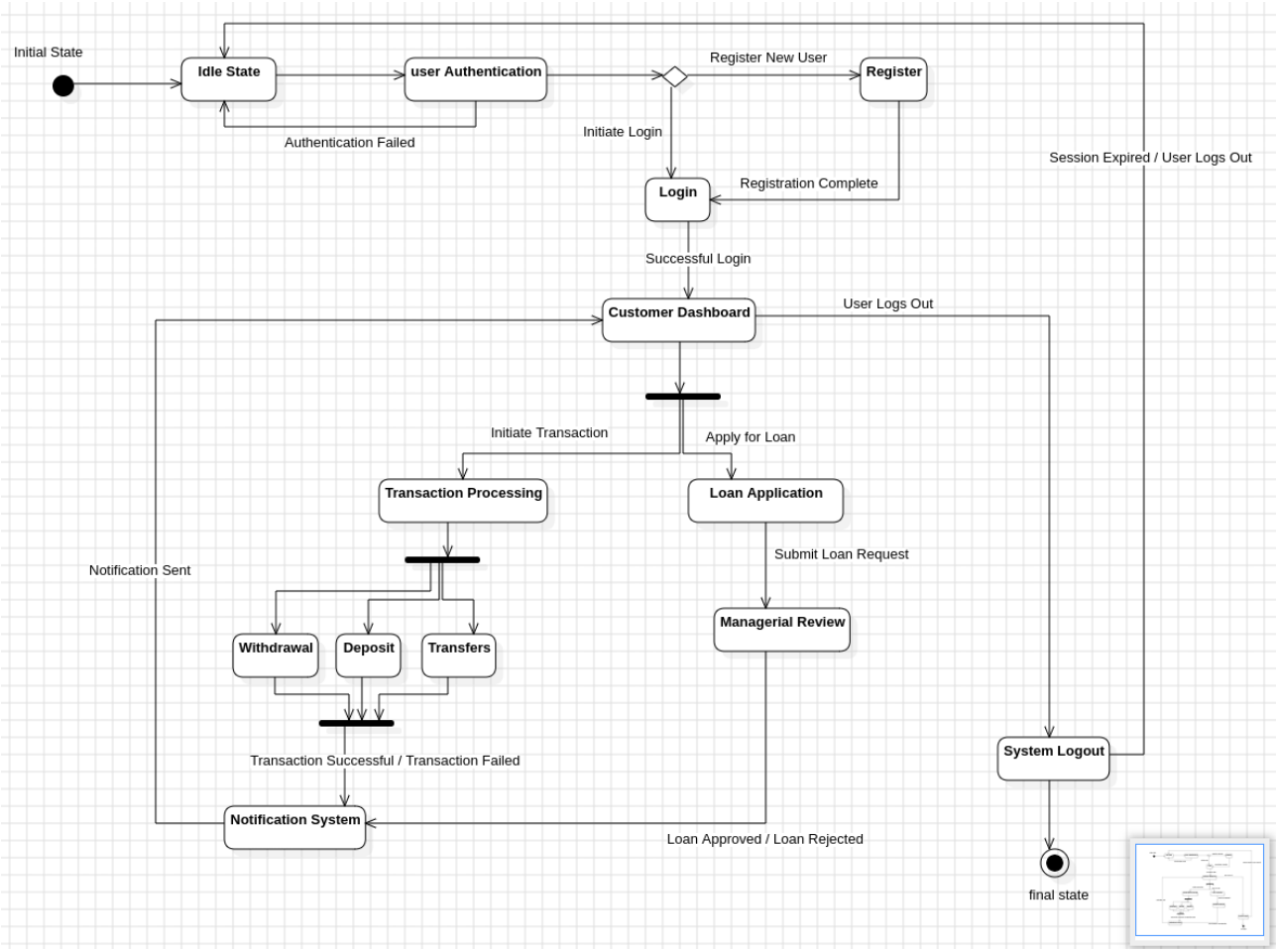
1. **User Authentication:** A user either logs in successfully and proceeds to the dashboard or authentication fails, redirecting back to the idle state.
2. **Transaction Processing:** Users initiate transactions, selecting withdrawal, deposit, or fund transfers, which lead to transaction success or failure, followed by a system notification.
3. **Loan Application:** Users apply for loans, which undergo managerial review. Based on approval or rejection, the notification system informs the user.
4. **System Logout:** Users can log out at any stage, leading to the **final state** where the session ends.

E) Conclusion

This **State Transition Diagram** provides a structured representation of how the **Online Bank Management System** functions dynamically. It highlights user authentication, transaction management, loan processing, and system notifications while ensuring smooth interactions between the states.

This diagram is essential for designing, implementing, and testing the system’s workflow efficiently.

State Transition Diagram



6. OO design – Use case Model, Class Model

Use Case Diagram

1. Introduction

The Online Bank Management System is designed to provide banking services to customers, manage employee operations, and handle transactions securely. This documentation outlines the use cases, actors, and their relationships within the system.

2. Actors

Primary Actors:

1. **Customer** - Performs banking activities like transactions, loan applications, and account management.
2. **Bank Employee** - Manages loans, approvals, and customer support.
3. **Admin** - Oversees employee management and system logs.
4. **Notification System** - Sends alerts and updates to customers.
5. **External Payment Gateway** - Facilitates secure online transactions.

3. Use Cases and Relationships

Authentication & Account Management

- **Login/Authentication** (Customer) - Customers authenticate before accessing services.
    - Includes: **Authenticate** (Validates credentials)
    - Extends: **Forgot Password** (Handles password recovery)
- **User Registration** (Customer) - Registers a new user.
- **View Account Details** (Customer) - Allows viewing of account information.

Transaction Management

- **Transaction** (Customer, External Payment Gateway) - Manages financial transactions.
    - Includes: **Credit** (Deposits money)
    - Includes: **Debit** (Withdraws money)
    - Includes: **Update Account Balance** (Reflects new balance after transactions)
- **Transfer Funds** (Customer) - Moves money between accounts.
    - Includes: **Credit** and **Debit**

    - Extends: **Send Notifications** (Alerts customers of the transfer)
- **Cancel Transaction** (Customer, Bank Employee) - Allows transaction reversals.
- **Dispute Transaction** (Customer, Bank Employee) - Enables customers to dispute erroneous transactions.

Loan Management

- **Loan** (Customer, Bank Employee) - Handles loan-related processes.
    - Includes: **Request Loan** (Customer applies for a loan)
    - Includes: **Credit Score Check** (Verifies eligibility)
    - Includes: **Term of the Loan** (Defines repayment terms)
    - Extends: **Approve Loan** (Handled by Bank Employee)
    - Extends: **Send Notifications** (Alerts customer about loan status)

System Management

- **Employee Management** (Admin) - Handles bank employee records.
- **System Logs** (Admin) - Maintains security logs.

Notifications

- **Send Notifications** (Notification System) - Sends transaction, loan, and security-related alerts.

4. Justifications

- **Stronger Actor-Use Case Links** - Explicitly connected Bank Employee and External Payment Gateway to relevant financial operations.
- **Detailed Loan Processing** - Enhanced with Credit Score Checks and Loan Terms.
- **Transaction Safety Measures** - Introduced "Cancel Transaction" and "Dispute Transaction" for error resolution.
- **Automated Notifications** - Ensures users receive updates on critical actions.

5. Conclusion

This use case model provides a structured approach to online banking, ensuring clarity in functionality and interactions between actors. The refinements help strengthen security, customer support, and system efficiency.

Diagram:


User Registration
System Logs
Admin
Login/Authentication
<<include>>
Authenticate
Customer
<<extend>>
Employee Management
View Account Details
Forgot Password
Bank Employee
Transfer Funds
Cancel / Failed Transaction
<<extend>>
Transaction
<<include>>
<<include>>
Debit
Credit
<<include>>
<<include>>
External Payment Gateway
Update Acc Balance
<<extend>>
Loan
<<extend>>
<<include>>
Approve Loan
Request Loan
Notification System
<<include>>
<<include>>
Term of the loan
Credit Score Check
Send Notifications

Class Diagram

Overview

This class diagram represents the structure and relationships between various entities in a Bank Management System. The main components include customers, employees, transactions, loans, accounts, notifications, and an external payment gateway. The system facilitates user interactions such as account management, fund transfers, loans, notifications, and transaction processing.

Classes and Their Descriptions

1. Customer

- **Attributes:**
  - `customerID: int` (Unique identifier for the customer)
  - `name: string` (Customer's full name)
  - `email: string` (Email ID for communication)
  - `phoneNumber: string` (Contact number of the customer)
  - `address: string` (Residential address)
  - `accountNumber: int` (Linked bank account number)
  - `balance: int` (Current account balance)
  - `username: string` (Login username)
  - `password: string` (Login password)
- **Methods:**
  - `login(): void` (Logs in the customer)
  - `viewAccountDetails(): void` (Displays account details)
  - `depositFunds(amount: double): void` (Adds money to account)
  - `withdrawFunds(amount: double): void` (Withdraws money from account)
  - `transferFunds(targetAccount: int, amount: double): void` (Transfers funds to another account)

- `applyForLoan(amount: double, tenure: int): void` (Requests a loan)
- `updateProfile(): void` (Updates customer information)

2. Account

- **Attributes:**
  - `accountNumber: int` (Unique account number)
  - `accountType: string` (Type of account - Savings, Current, etc.)
  - `balance: double` (Account balance)
  - `status: string` (Active or Inactive status)
- **Methods:**
  - `getBalance(): double` (Retrieves current balance)
  - `updateBalance(amount: double): void` (Modifies account balance)
  - `closeAccount(): void` (Closes the account)

3. Employee

- **Attributes:**
  - `employeeID: int` (Unique ID of the employee)
  - `name: string` (Employee’s name)
  - `role: string` (Job role)
  - `email: string` (Contact email)
  - `phoneNumber: string` (Contact number)
  - `department: string` (Department of employment)
- **Methods:**
  - `manageTransactions(): void` (Oversees transactions)
  - `approveLoans(): void` (Approves or rejects loan applications)
  - `viewCustomerDetails(): void` (Retrieves customer information)

4. Admin (Inherits Employee)

- **Methods:**
  - `manageEmployees(): void` (Handles employee operations)
  - `viewSystemLogs(): void` (Views system activity logs)
  - `configureSettings(): void` (Updates system configurations)

5. Loan

- **Attributes:**
  - `loanID: int` (Unique loan identifier)
  - `customerID: int` (Associated customer ID)
  - `amount: double` (Loan amount)
  - `interestRate: double` (Applicable interest rate)
  - `tenure: int` (Loan repayment duration in months)
  - `status: string` (Loan approval status)
  - `newAttr: Integer` (Placeholder attribute)
- **Methods:**
  - `applyForLoan(): void` (Initiates a loan application)
  - `approveLoan(): void` (Processes loan approval)
  - `rejectLoan(): void` (Declines the loan application)

6. Transaction

- **Attributes:**
  - `transactionID: int` (Unique identifier for transactions)
  - `transactionType: string` (Deposit, Withdrawal, Transfer, etc.)
  - `amount: double` (Transaction amount)
  - `date: DateTime` (Timestamp of the transaction)
  - `status: string` (Transaction status - Success, Pending, Failed)
- **Methods:**

- `processTransaction(): void` (Executes transaction)
- `validateTransaction(): boolean` (Checks transaction validity)
- `reverseTransaction(): boolean` (Cancels a transaction if needed)

7. Notification System

- **Attributes:**
  - `notificationID: int` (Unique notification ID)
  - `message: string` (Notification message content)
  - `timestamp: DateTime` (Notification time)
  - `recipient: string` (Recipient of notification)
- **Methods:**
  - `sendNotification(): void` (Sends notification to customers)

8. Payment Gateway

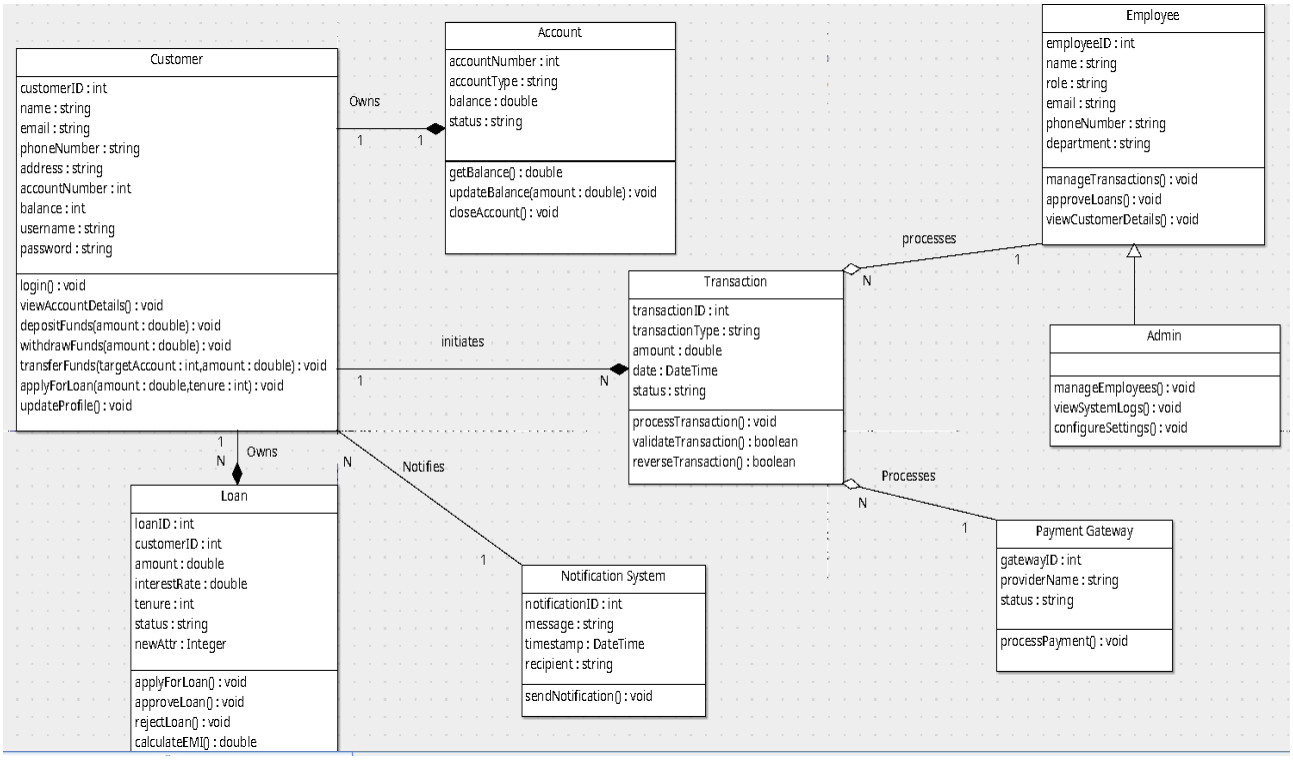
- **Attributes:**
  - `gatewayID: int` (Unique identifier for gateway)
  - `providerName: string` (Third-party payment processor name)
  - `status: string` (Operational status)
- **Methods:**
  - `processPayment(): void` (Handles external transactions)

Relationships and Associations

Relationship	Type	Description
Customer → Account	Composition	A customer owns an account; deletion of customer results in account removal.
Customer → Transaction	Aggregation	A customer can initiate multiple transactions, but transactions exist independently.
Customer → Loan	Composition	A loan is tied to a customer; deletion of customer results in loan removal.
Employee → Transaction	Aggregation	Employees manage transactions but do not own them.
Admin → Employee	Generalization	Admin is a specialized form of Employee.
Notification System → Customer	Association	The system notifies customers about updates and transactions.
External Payment Gateway → Transaction	Aggregation	Transactions are processed through external gateways but exist independently.

This class diagram effectively models the major functionalities of a banking system while maintaining scalability and modularity.

Diagram:



7. OO design – Interaction Models

Sequence and Collaboration Diagrams for Online Bank Management System

Part 1: Sequence Diagram

1. Use Case: User Login

Step 1: Identify a Use Case

- The user logs into the system.

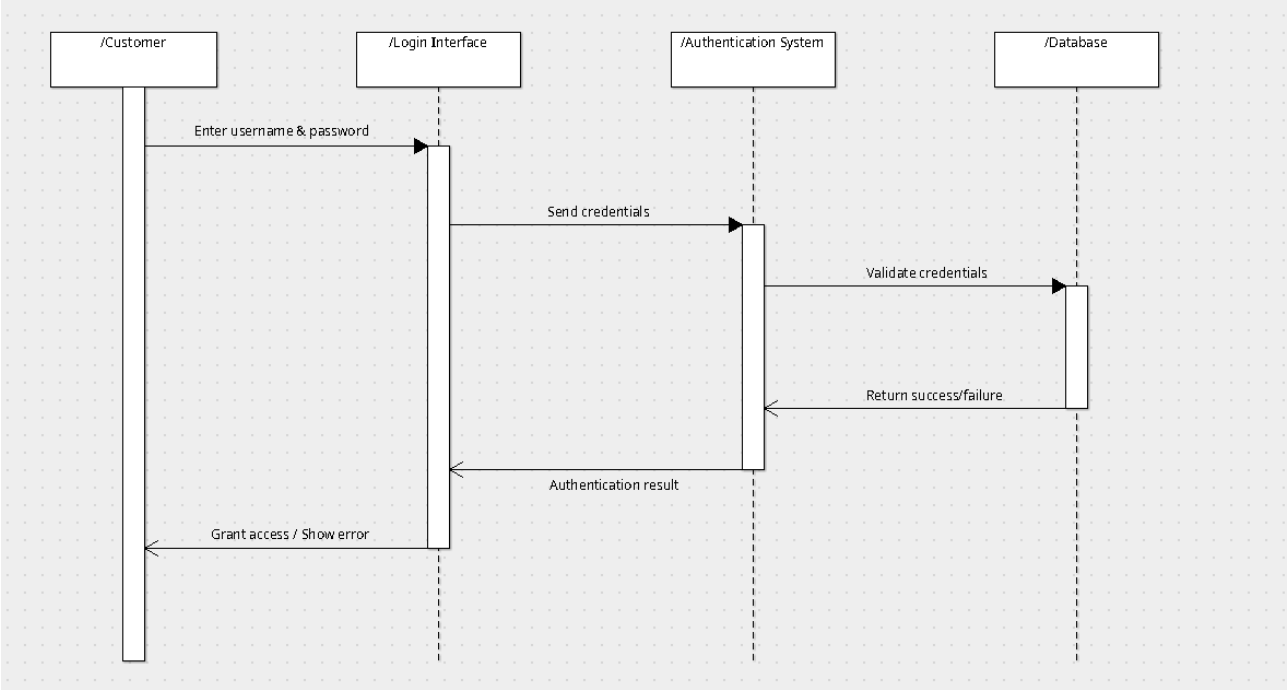
Step 2: Identify Objects

- User, Login Interface, Authentication System, Database

Step 3: Define the Sequence of Interactions

1. User enters credentials.
2. System validates credentials.
3. If valid, system grants access and displays dashboard.
4. If invalid, system shows an error message.

Sequence Diagram



Part 2: Collaboration Diagram

Step 1: Identify the Same Use Case

- The user logs into the system.

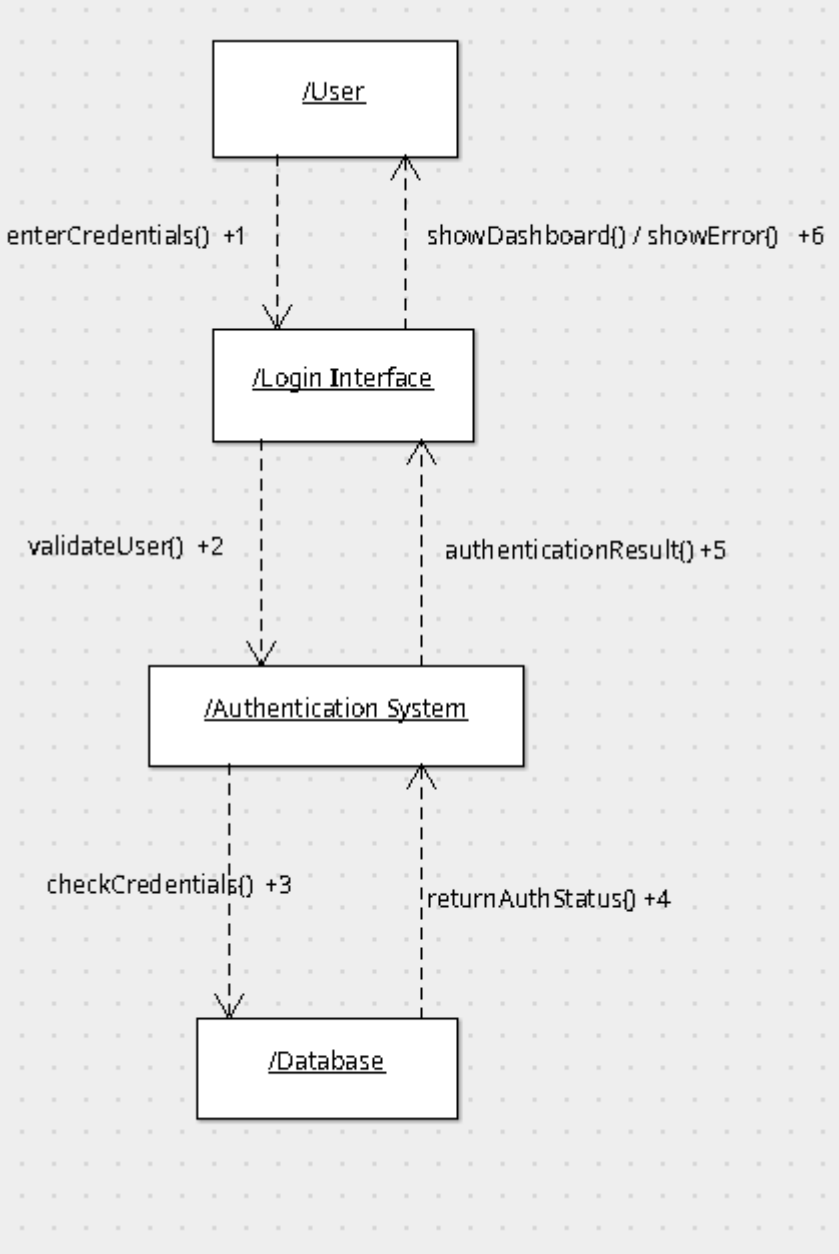
Step 2: Define Object Links

- User interacts with the Login Interface.
- Login Interface sends credentials to the Authentication System.
- Authentication System checks credentials against the Database.
- System responds with success or failure.

Step 3: Define Interactions

1. User requests login.
2. Login Interface forwards credentials to Authentication System.
3. Authentication System verifies credentials with Database.
4. System sends authentication result to User.

Collaboration Diagram



2. Use Case: Fund Transfer

Step 1: Identify a Use Case

- The user transfers funds to another account.

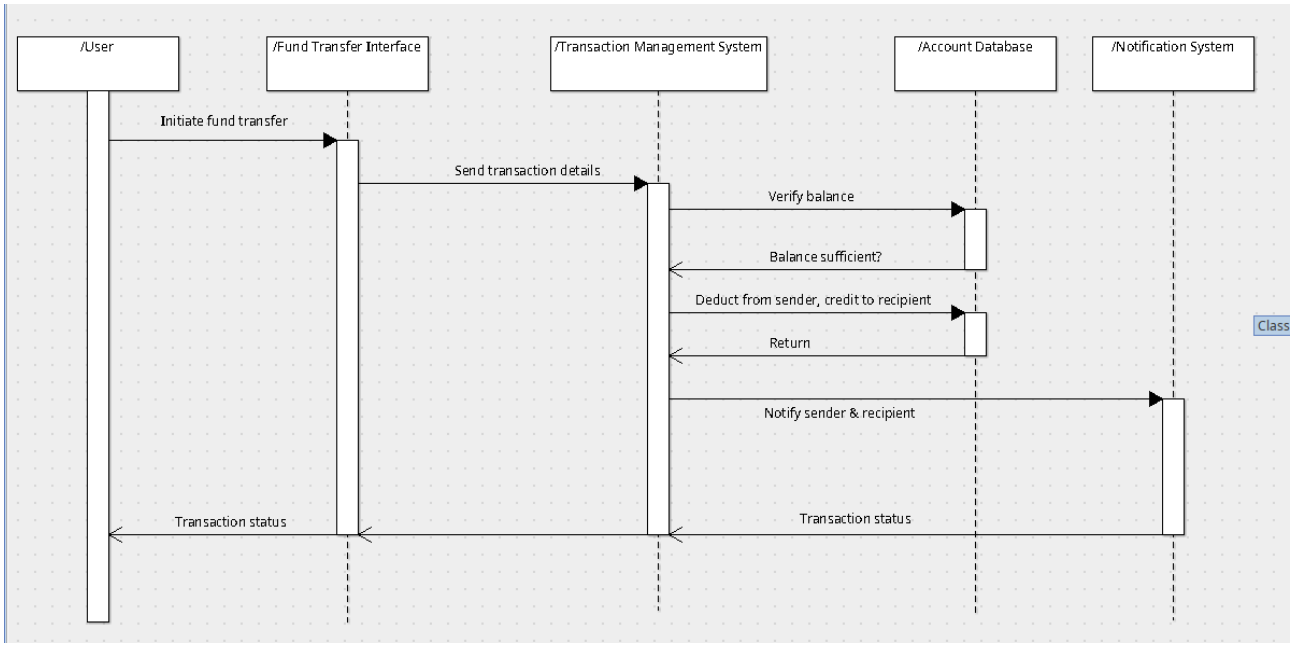
Step 2: Identify Objects

- User, Fund Transfer Interface, Transaction Processor, Account Database, Notification System

Step 3: Define the Sequence of Interactions

1. User selects fund transfer option.
2. User enters recipient details and amount.
3. System validates account balance.
4. If sufficient, system deducts amount from the user's account and credits to the recipient.
5. System confirms the transaction and notifies the user.

Sequence Diagram



Part 2: Collaboration Diagram

Step 1: Identify the Same Use Case

- The user transfers funds to another account.

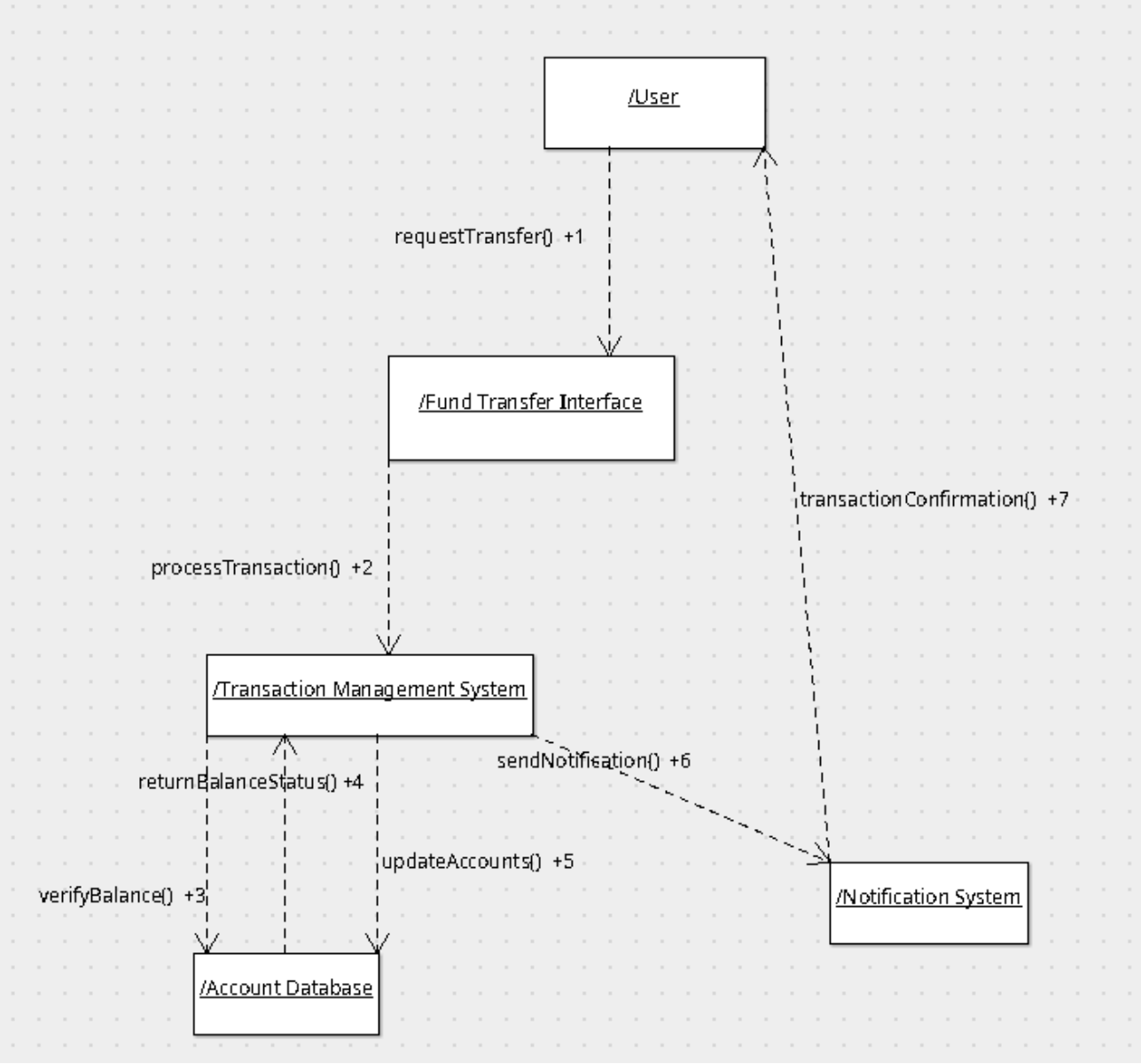
Step 2: Define Object Links

- User interacts with the Fund Transfer Interface.
- Fund Transfer Interface sends transaction details to Transaction Processor.
- Transaction Processor communicates with Account Database.
- Notification System is used to confirm the transaction.

Step 3: Define Interactions

1. User requests fund transfer.
2. Fund Transfer Interface forwards details to Transaction Processor.
3. Transaction Processor verifies balance with Account Database.
4. If sufficient, Transaction Processor updates both sender and recipient balances.
5. Transaction Processor sends confirmation to Notification System.
6. Notification System informs the User.

Collaboration Diagram



3. Use Case: Loan Application

Step 1: Identify a Use Case

- The user applies for a loan.

Step 2: Identify Objects

- User, Loan Application Interface, Loan Processing System, Loan Approval Officer, Notification System

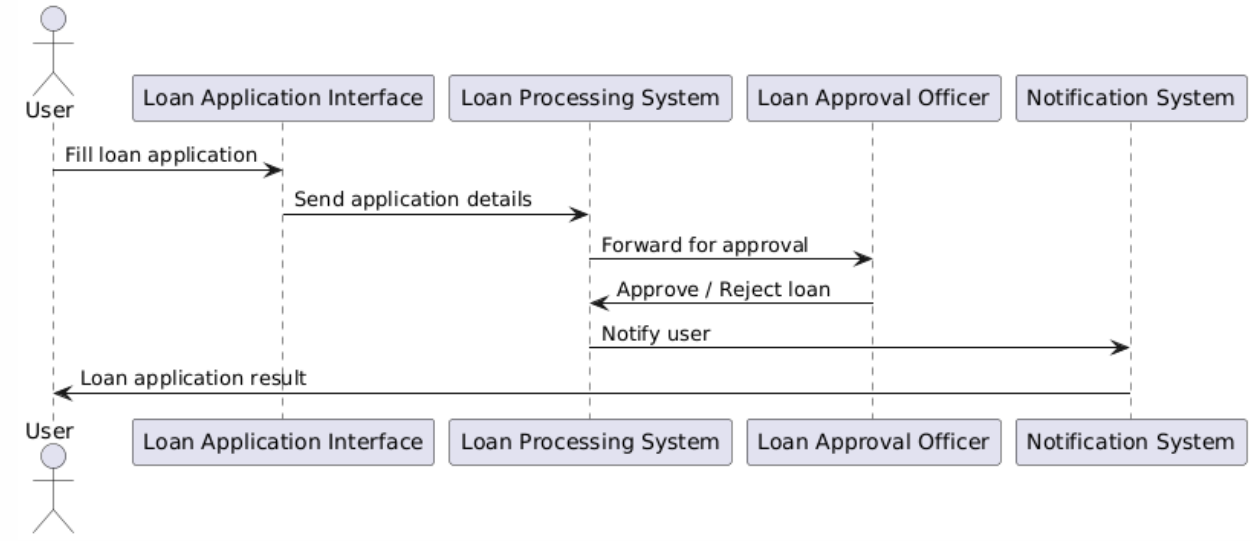
Step 3: Define the Sequence of Interactions

1. User selects loan application.
2. User enters loan amount and tenure.
3. System verifies eligibility and forwards request to the Loan Approval Officer.
4. Loan Approval Officer reviews and approves/rejects the loan.
5. System notifies the user of the decision.

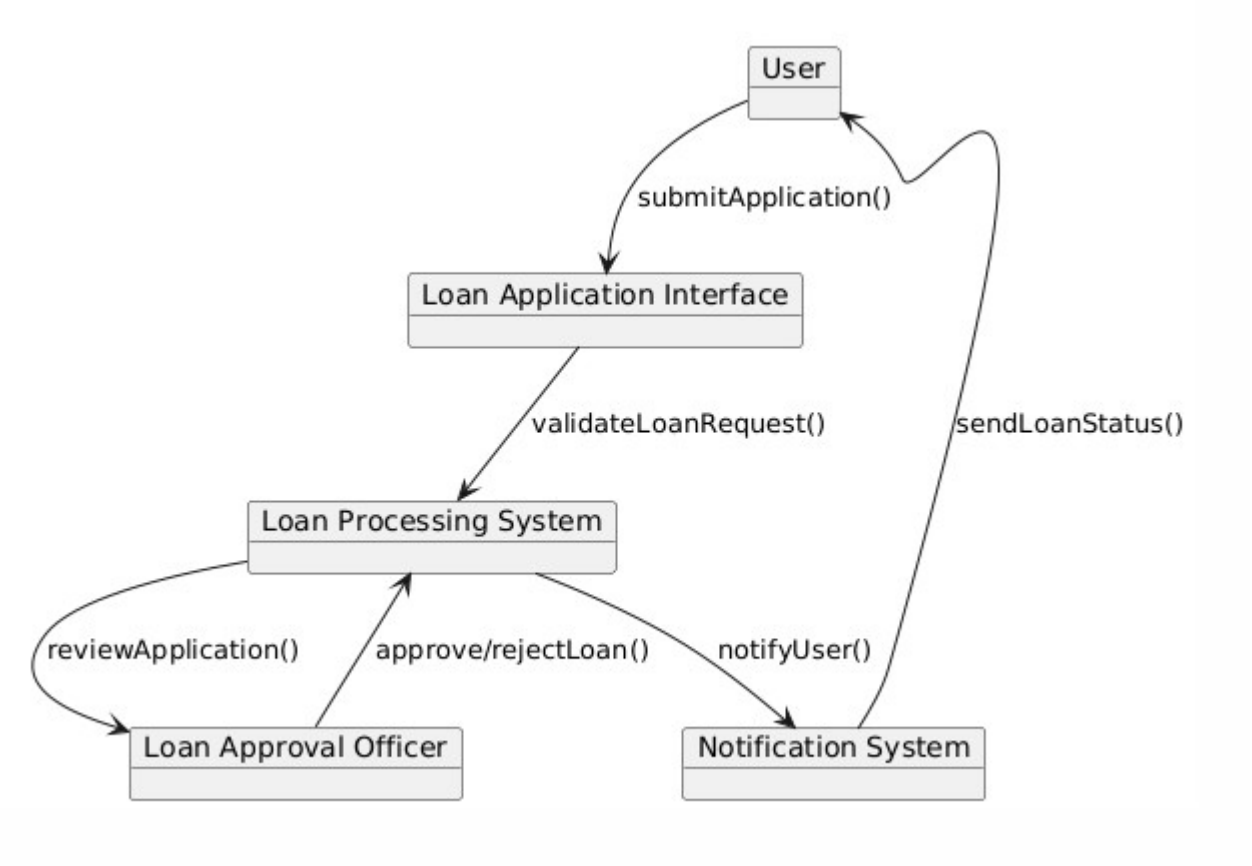
Sequence Diagram

```
@startuml
actor User
participant "Loan Application Interface" as LAI
participant "Loan Processing System" as LPS
participant "Loan Approval Officer" as LAO
participant "Notification System" as NS

User -> LAI : Fill loan application
LAI -> LPS : Send application details
LPS -> LAO : Forward for approval
LAO -> LPS : Approve / Reject loan
LPS -> NS : Notify user
NS -> User : Loan application result
@enduml
```



Collaboration Diagram



8. OO design – Package, Component and deployment models

A) Package Model

Overview

The Online Bank Management System is structured into distinct packages to modularize functionalities such as user authentication, transaction processing, loan management, and notifications. Each package contains relevant classes that handle specific banking operations.

Packages and Their Classes

1. **User Management Package**
    - `User` - Represents customers and employees.
    - `Customer` - Stores customer-specific details (e.g., name, email, account information).
    - `Employee` - Handles employee details and tasks.
    - `Admin` - Manages administrative actions (e.g., user account control).
    - `Authentication` - Manages user login, authentication, and access control.
2. **Account Management Package**
    - `Account` - Stores account details (e.g., account type, balance, status).
    - `SavingsAccount` - Inherits from `Account` for savings accounts.
    - `CurrentAccount` - Inherits from `Account` for current accounts.
    - `TransactionHistory` - Stores transaction logs for auditing.
3. **Transaction Processing Package**
    - `Transaction` - Manages all banking transactions.
    - `FundTransfer` - Handles online fund transfers between accounts.
    - `BillPayment` - Manages bill payments through the banking system.
    - `ExternalPaymentGateway` - Connects to third-party payment services.
4. **Loan Management Package**

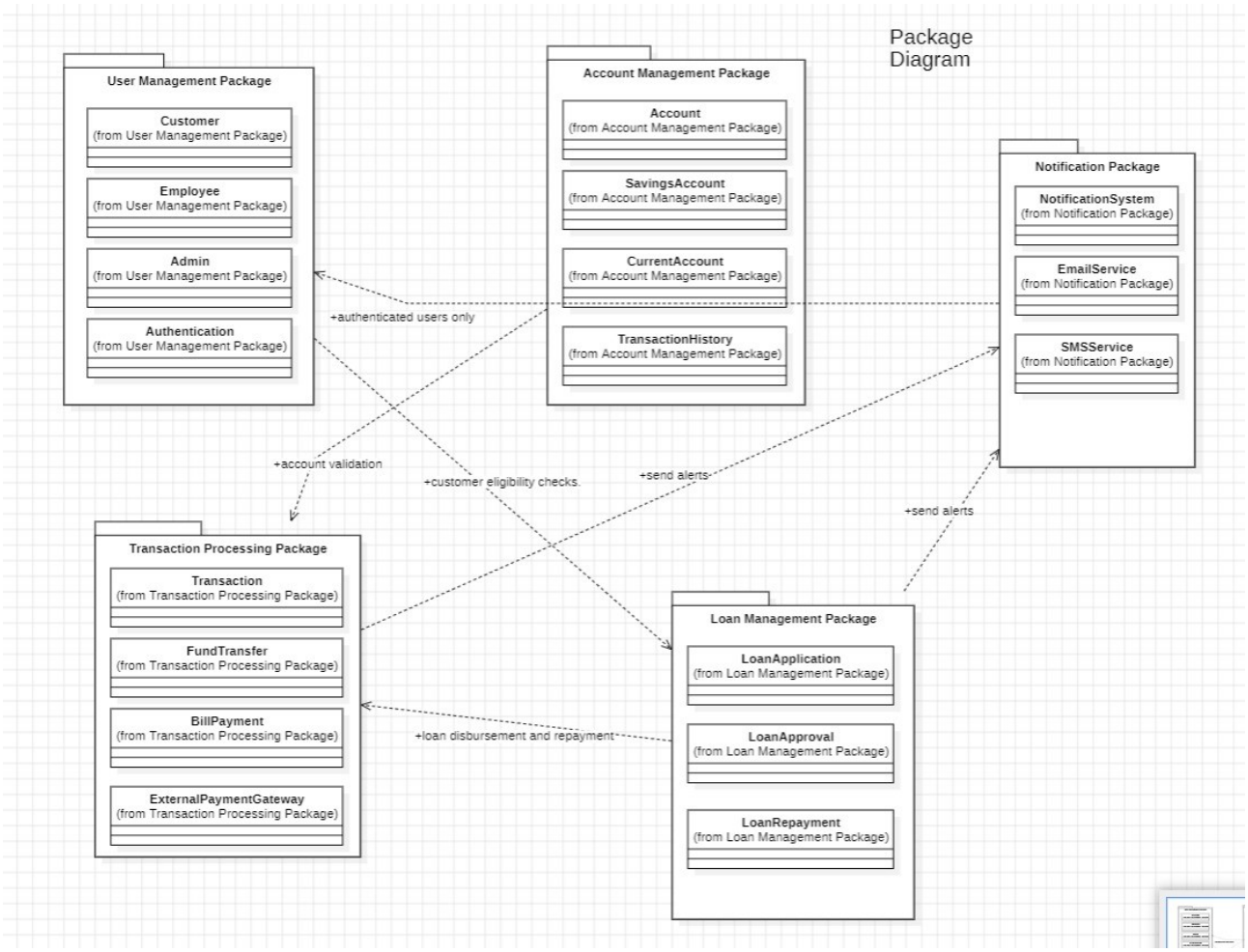
- `LoanApplication` - Handles customer loan applications.
- `LoanApproval` - Processes loan approvals and rejections.
- `LoanRepayment` - Manages EMI payments and schedules.
- `InterestCalculator` - Computes interest and repayment details.

5. **Notification Package**
    - `NotificationSystem` - Sends alerts and messages to customers.
    - `EmailService` - Sends email notifications.
    - `SMSService` - Sends SMS alerts.

Dependencies

- The `Transaction Processing` package depends on `Account Management` for account validation.
- The `Loan Management` package depends on `User Management` for customer eligibility checks.
- The `Notification` package interacts with `Transaction Processing` and `Loan Management` to send alerts.
- The `Authentication` package controls access to all other packages.
- The `Transaction Processing` package depends on the `ExternalPaymentGateway` for online transactions.
- The `Loan Management` package depends on `Transaction Processing` for loan disbursement and repayment.
- The `Admin` class in `User Management` depends on `Account Management` and `Loan Management` for monitoring transactions and loan approvals.
- The `Notification Package` depends on `Authentication` to ensure notifications are sent to authenticated users only.

Package Diagram:



B) Component Model

Overview

The Component Model represents the modular structure of the Online Bank Management System, showing interactions between different components through defined interfaces.

Components and Their Interfaces

1. **User Authentication Service**
    - Interfaces:
        - `registerUser()` - Registers a new customer.
        - `loginUser()` - Authenticates a customer.
        - `updateProfile()` - Allows users to modify account details.
        - `validateUser()` - Verifies login credentials.
    - **Motive:** Handles user authentication and profile updates.
2. **Transaction Processing System**
    - Interfaces:
        - `processTransaction()` - Handles fund transfers and payments.
        - `validateAccount()` - Ensures sufficient balance before transactions.
        - `transferFunds()` - Transfers money between accounts.
    - **Motive:** Ensures secure and accurate financial transactions.
3. **Loan Management System**
    - Interfaces:
        - `applyLoan()` - Submits a new loan request.
        - `approveLoan()` - Approves or rejects a loan.
        - `calculateEMI()` - Computes repayment details.
    - **Motive:** Manages loan applications, approvals, and repayments.
4. **Notification System**

- Interfaces:
    - `sendNotification()` - Sends general notifications.
    - `sendEmail()` - Sends email alerts.
    - `sendSMS()` - Sends SMS notifications.
- **Motive:** Keeps customers informed about transactions and account updates.

5. **Database Service**
    - Interfaces:
        - `storeData()` - Saves customer, account, and transaction details.
        - `fetchData()` - Retrieves stored data.
        - `updateData()` - Updates records in the database.
    - **Motive:** Ensures secure and efficient data storage.
6. **Admin Management System**
    - Interfaces:
        - `manageUsers()` - Blocks or unblocks users.
        - `manageAccounts()` - Freezes or deactivates accounts.
    - **Motive:** Allows administrative control over banking operations.

Dependencies

- `User Authentication` interacts with `Transaction Processing` and `Loan Management` for secure transactions.
- `Transaction Processing` and `Loan Management` rely on `Database Service` for data storage.
- `Notification System` works with `Transaction Processing` and `Loan Management` to send alerts.
- `Admin Management System` depends on `Transaction Processing` and `Loan Management` to monitor transactions and loan approvals.

Mappings Between Interfaces

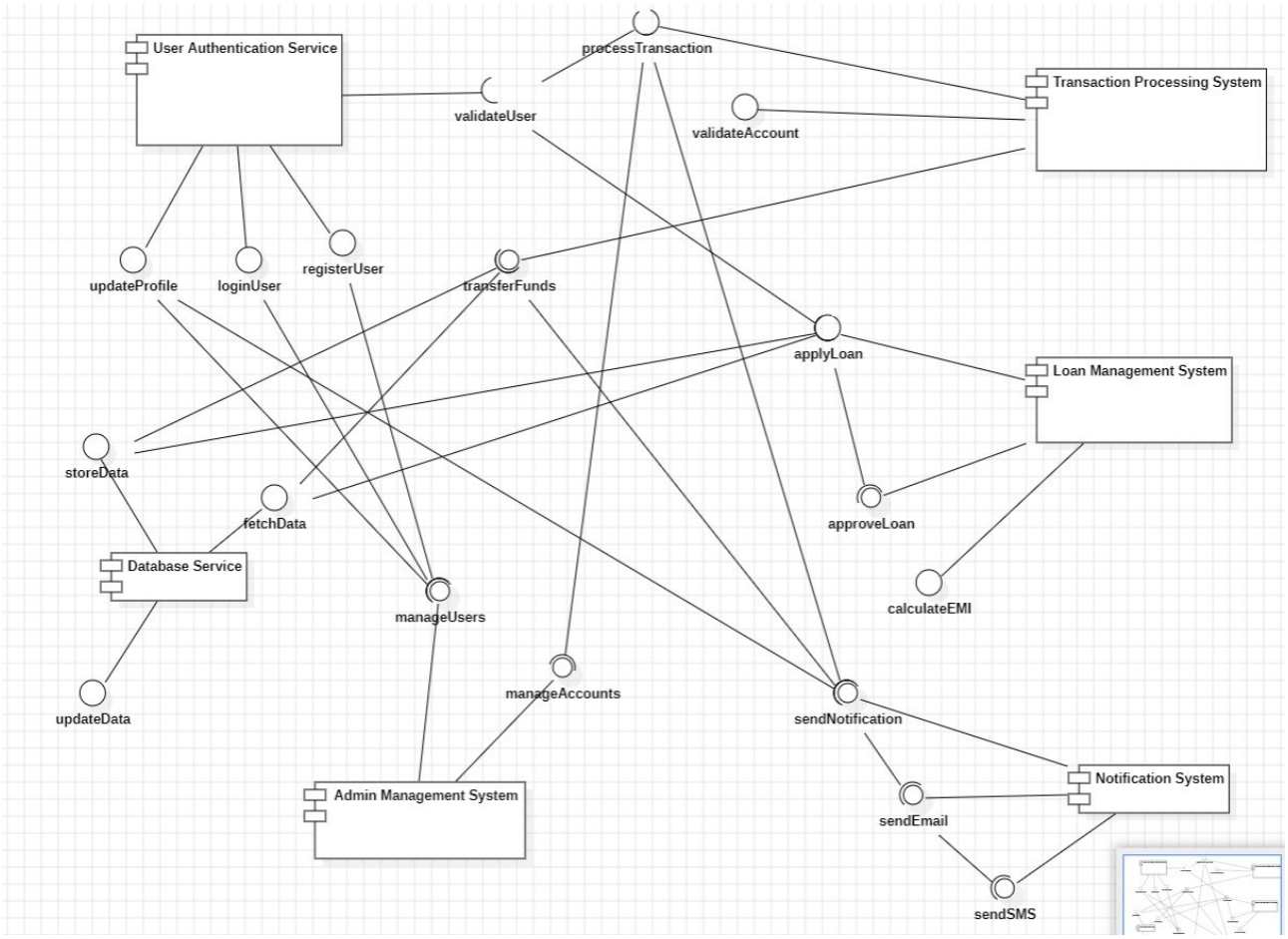
Component	Interface	Interacting Component(s)	Purpose of Interaction
User Authentication Service	`validateUser()`	Transaction Processing System, Loan Management System	Ensures only authenticated users can initiate transactions or apply for loans.
User Authentication Service	`registerUser()`	Database Service	Stores user registration details securely.
User Authentication Service	`updateProfile()`	Database Service	Updates user account details in the database.
Transaction Processing System	`processTransaction()`	Database Service, Notification System	Validates transactions, updates account balances, and notifies users.
Transaction Processing System	`validateAccount()`	User Authentication Service, Database Service	Ensures the user is authenticated and has sufficient balance for transactions.
Transaction Processing System	`transferFunds()`	Database Service, Notification System	Transfers money between accounts and alerts users upon successful transfer.
Loan Management System	`applyLoan()`	Database Service, User Authentication Service	Verifies user details and stores loan applications.
Loan Management System	`approveLoan()`	Admin Management System, Notification System	Sends loan approval/rejection updates to users and administrators.
Loan Management System	`calculateEMI()`	Database Service	Retrieves loan amount and computes EMI details.
Notification System	`sendNotification()`	Transaction Processing System, Loan Management System	Notifies users about transactions and loan updates.
Notification System	`sendEmail()`	Admin Management System, Transaction Processing System	Sends email alerts for system-related actions.
Notification	`sendSMS()`	Admin Management System, Transaction	Sends SMS alerts for transactions and critical

System		Processing System	updates.
Database Service	`storeData()`	User Authentication, Transactions, Loans	Stores various system-related data securely.
Database Service	`fetchData()`	All major components	Provides stored data for business logic processing.
Database Service	`updateData()`	All major components	Updates account, loan, or transaction details based on system actions.
Admin Management System	`manageUsers()`	Database Service, Notification System	Manages user access, blocking, and security.
Admin Management System	`manageAccounts()`	Database Service, Transaction Processing System	Freezes or deactivates accounts as required.

Interaction Flow Example

1. **User logs in** → `validateUser()` (User Authentication → Transaction Processing)
2. **User initiates a transaction** → `processTransaction()` (Transaction Processing → Database Service)
3. **Funds are transferred** → `transferFunds()` (Transaction Processing → Database & Notification)
4. **User receives a confirmation email & SMS** → `sendNotification()`, `sendEmail()`, `sendSMS()` (Notification System)
5. **Admin monitors user activities** → `manageUsers()`, `manageAccounts()` (Admin Management → Database & Notifications)

Component Diagram :



C) Deployment Model

Overview

The Deployment Model illustrates the physical architecture of the Online Bank Management System, showing how different components are deployed across multiple nodes.

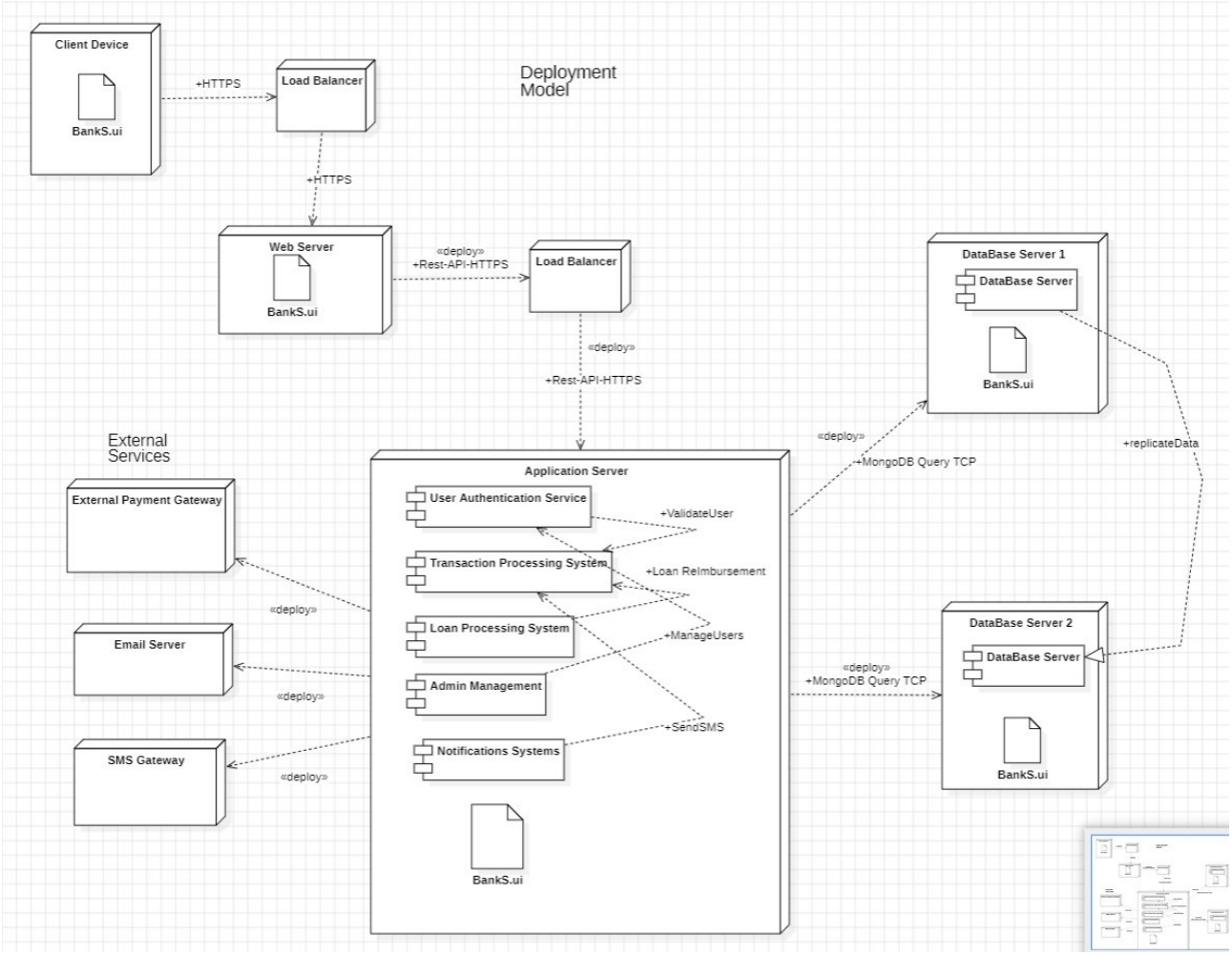
Deployment Components

1. **Client System (Web & Mobile Apps)**
    - Hosts: Web Browser, Mobile App
    - Interacts with: User Authentication Service, Transaction Processing System
2. **Load Balancer**
    - Distributes requests to web servers to enhance performance.
3. **Web Server**
    - Hosts: User interface components.
    - Communicates with: Application Server.
4. **Application Server**
    - Hosts: Business Logic (Authentication, Transactions, Loans, Admin Management, Notifications)
    - Communicates with: Database Server, External APIs (e.g., Payment Gateway, SMS Service)
5. **Database Server**
    - Hosts: Database Service
    - Stores: User, Account, Transaction, and Loan Data
    - Implements replication for high availability.
6. **External Services**
    - Includes: Payment Gateway, Email Server, SMS Gateway
    - Communicates with: Application Server for processing payments and sending notifications

Deployment Interactions

- **Clients** (Web/Mobile Apps) send requests to the **Application Server** via the **Load Balancer**.
- **Application Server** processes business logic and interacts with the **Database Server** for data storage and retrieval.
- **Application Server** connects to **External Services** for payments and notifications.
- **Admin System** is hosted on the **Application Server** and manages users and accounts.

Deployment Diagram :



9. Design and demonstration of test cases. Functional Testing and Non- Functional Testing (using any open source tools)

Test Table:

Functional Test Cases

Test Case ID	Test Case Description	Test Steps	Test Data	Expected Result	Actual Result	Pass/ Fail
TC_Func_01	Verify user registration	1. Navigate to registration page. 2. Enter valid details. 3. Submit form.	Username: user1 Email: user1@email.com Password: Pass@123	User should be registered and pending approval.	✅	Pass
TC_Func_02	Verify user login	1. Navigate to login page. 2. Enter correct credentials. 3. Click login.	Username: user1 Password: Pass@123	User should log in successfully and see dashboard.	✅	Pass
TC_Func_03	Verify fund transfer	1. Login as user. 2. Navigate to transaction page. 3. Enter recipient & amount. 4. Confirm transaction.	Sender: user1 Receiver: user2 Amount: $100	Transaction should complete successfully.	✅	Pass
TC_Func_04	Verify loan application	1. Login as user. 2. Navigate to loan management. 3. Apply for a loan.	Loan Amount: $5000 Duration: 12 months	Loan request should be submitted for approval.	✅	Pass
TC_Func_05	Verify viewing account statement	1. Login as user. 2. Navigate to profile page. 3. Click on "View Statement".	User: user1	Statement should be displayed with past transactions.	✅	Pass
TC_Func_06	Verify profile update	1. Login as user. 2. Go to profile. 3. Update personal details. 4. Save changes.	New Email: user1_new@email.com	Profile should be updated successfully.	✅	Pass
TC_Func_07	Verify logout functionality	1. Login as user. 2. Click on logout button.	User: user1	User should be logged out and redirected to homepage.	✅	Pass

TC_Func_08	Verify admin approval of new user	1. Login as admin. 2. Navigate to user management. 3. Approve a pending user.	User: user1	User should receive approval notification and be able to log in.	✅	Pass
TC_Func_09	Verify failed fund transfer due to insufficient balance	1. Login as user. 2. Navigate to transaction page. 3. Enter large transfer amount. 4. Confirm transaction.	Sender: user1 Amount: $10,000 (more than balance)	Transaction should fail with "Insufficient balance" error.	✅	Pass
TC_Func_10	Verify loan approval by admin	1. Login as admin. 2. Navigate to loan requests. 3. Approve/reject loan request.	Loan ID: 101	Loan should be approved or rejected with appropriate status update.	✅	Pass

Non-Functional Test Cases

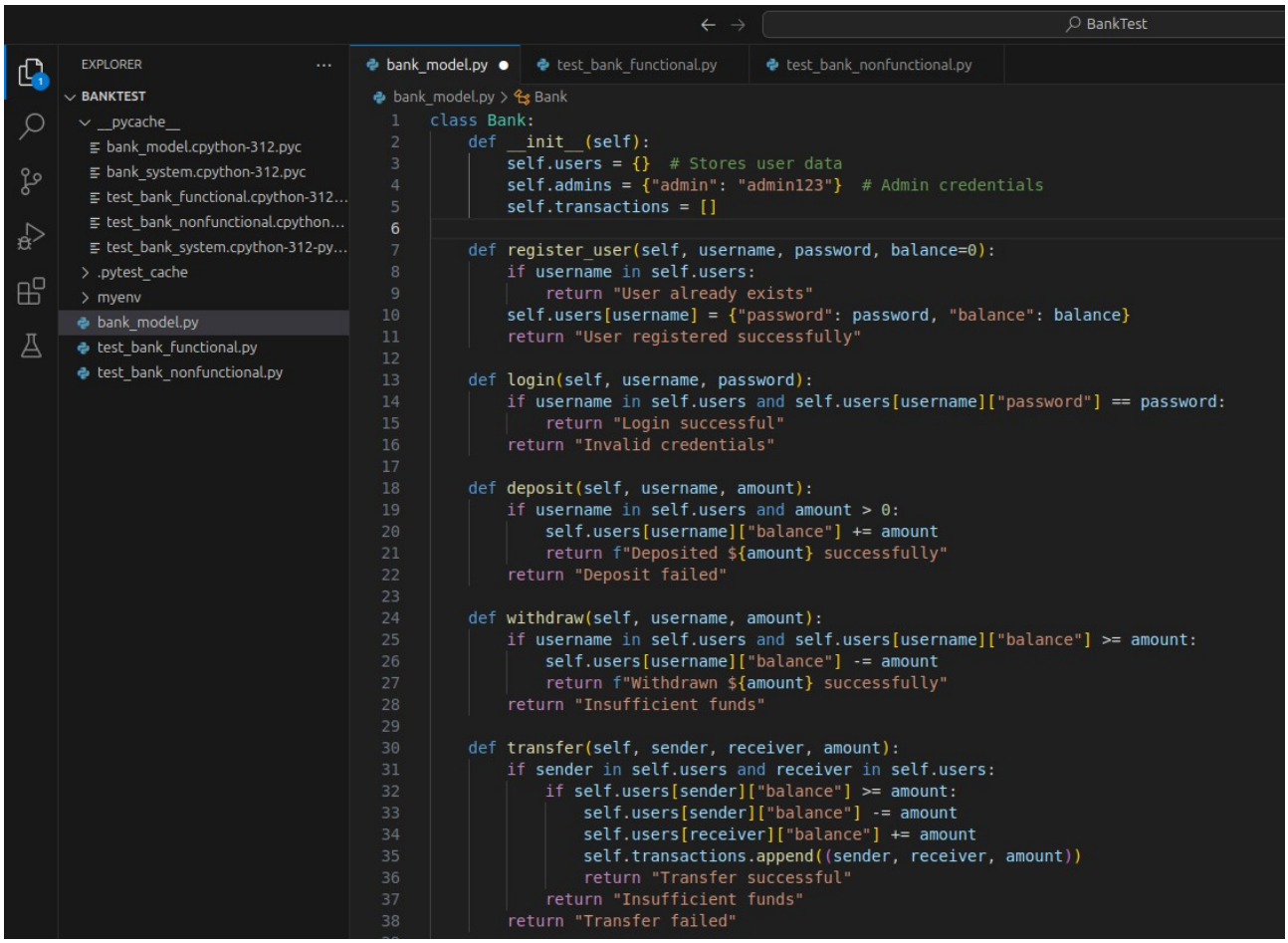
Test Case ID	**Test Case Description**	**Test Steps**	**Test Data**	**Expected Result**	**Actual Result**	**Pass/ Fail**
TC_NonFunc_01	Verify system response time for login	1. Attempt to log in. 2. Measure response time.	Username: user1	Login should complete in under 2 seconds.	✅	Pass
TC_NonFunc_02	Verify system handles 100 concurrent fund transfers	1. Simulate 100 users transferring funds. 2. Monitor system stability.	100 users	System should handle transactions without crashes.	✅	Pass
TC_NonFunc_03	Verify SQL injection protection	1. Enter malicious input in login fields. 2. Submit form.	Username: '" OR 1=1 --"	System should reject input and return an error message.	✅	Pass
TC_NonFunc_04	Verify cross-site scripting (XSS) protection	1. Enter malicious script in profile update. 2. Submit form.	Script: `<script>alert('XSS')</script>`	System should sanitize input and prevent execution.	✅	Pass
TC_NonFunc_05	Verify system stability after server restart	1. Restart server. 2. Attempt to log in and perform transactions.	Server Restart	System should function normally post-restart.	✅	Pass

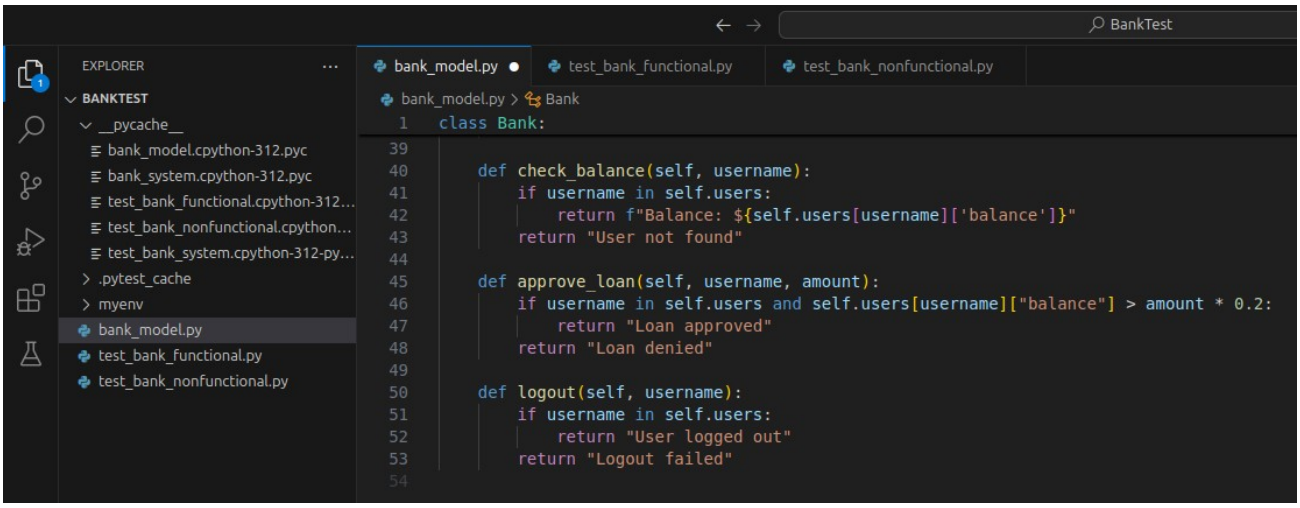
TC_NonFunc_06	Verify system recovery after database failure	1. Simulate database failure. 2. Attempt to retrieve past transactions.	Order ID: 201	Data should be recovered from backup.	✅	Pass
TC_NonFunc_07	Verify ease of use for fund transfers	1. Attempt fund transfer. 2. Measure time taken. 3. Collect user feedback.	User: user1	Transaction should be completed in under 5 seconds.	✅	Pass
TC_NonFunc_08	Verify system handles large transaction data	1. Attempt to transfer large amount of money. 2. Check system stability.	Amount: $1,000,000	System should handle transaction without crashing.	✅	Pass
TC_NonFunc_09	Verify invalid input handling	1. Pass invalid inputs to transfer funds. 2. Observe error messages.	Amount: -500	System should return an error without crashing.	✅	Pass
TC_NonFunc_10	Verify high load performance during transactions	1. Simulate 50 users making transactions. 2. Monitor system performance.	50 users	System should handle the load efficiently.		

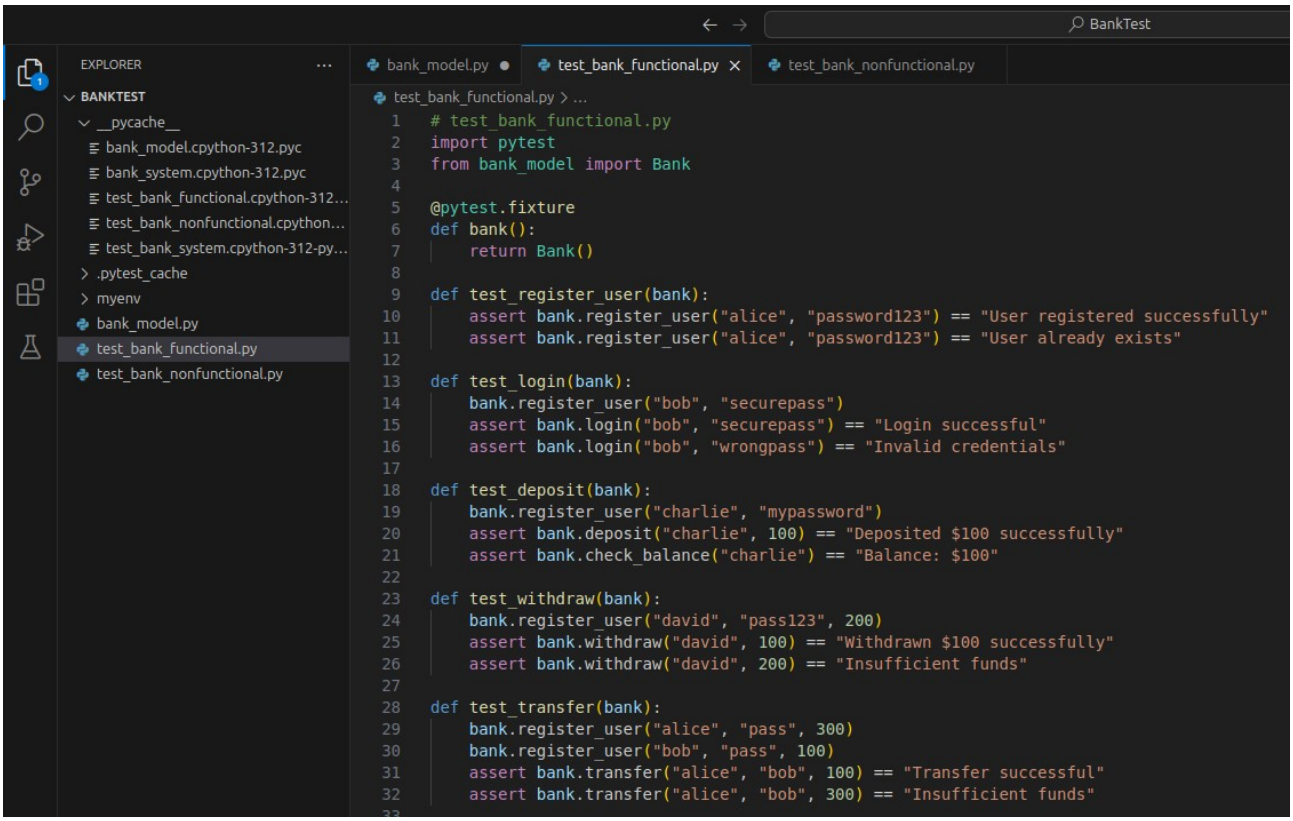
2. Functional Testing with PyTest

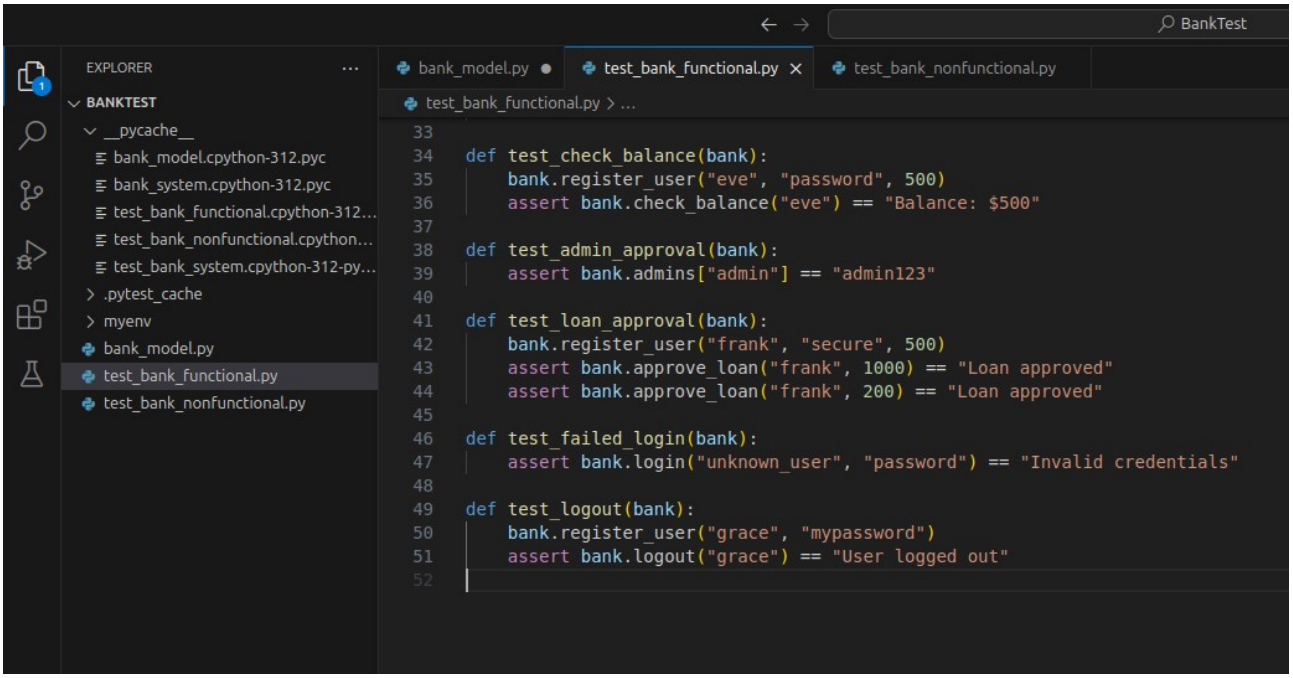
Functional testing ensures that the features of the Bank Management system work as expected. The following test cases cover key functionalities such as adding books to the cart, placing orders, processing payments, and admin tasks. These tests are written in PyTest format and operate on the Bank_model class.

PyTest Code for *Functional* Test Cases:

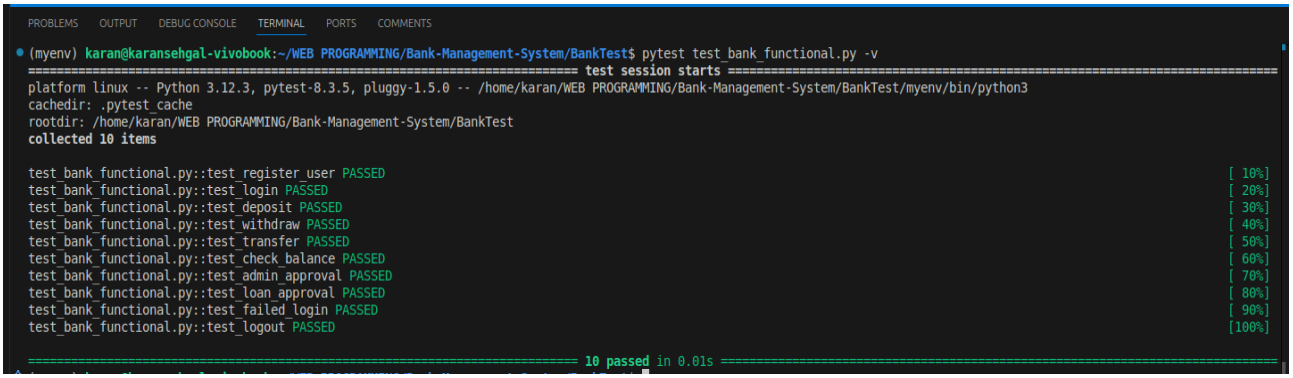




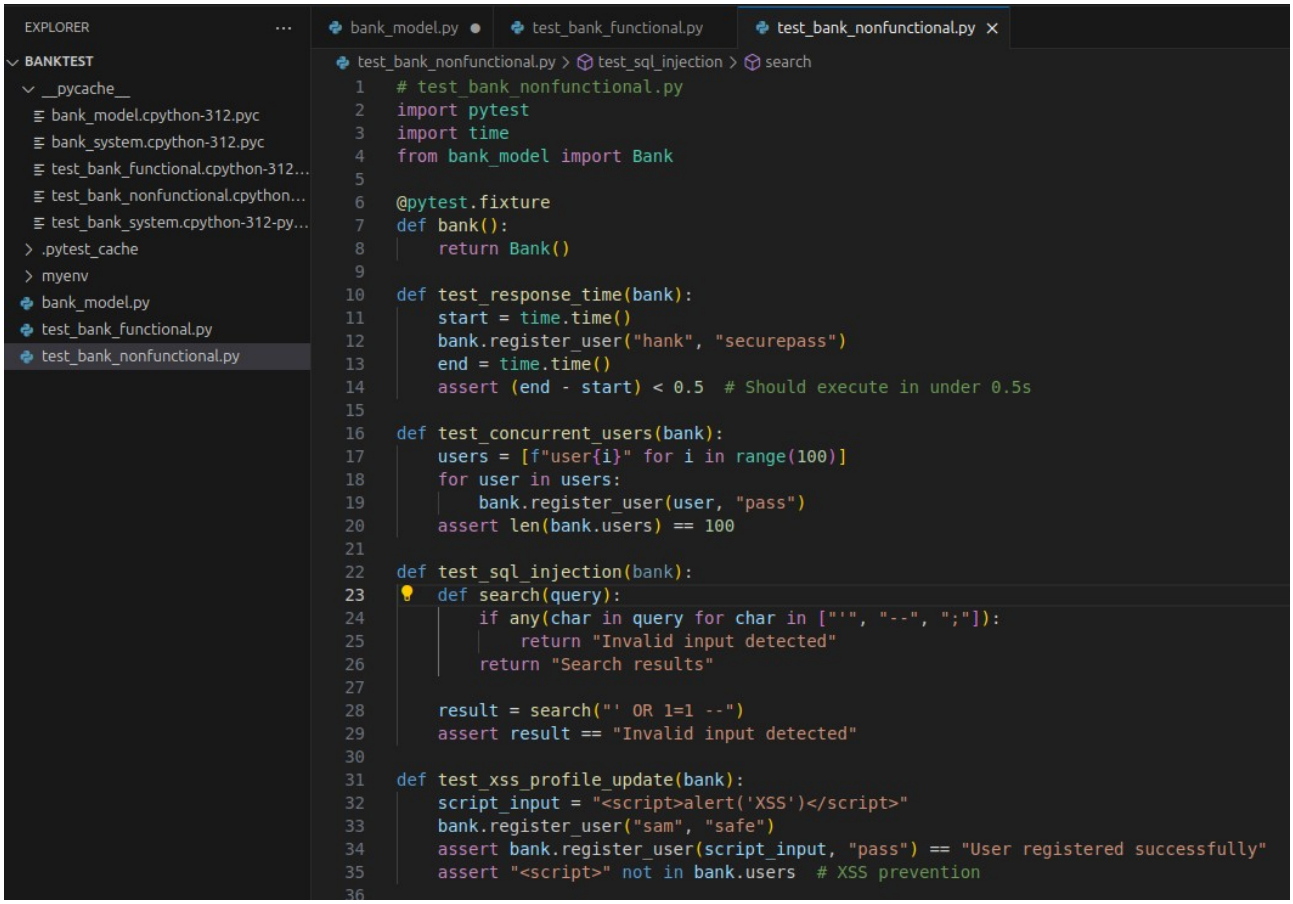

BankTest
EXPLORER
BANKTEST
__pycache__
bank_model.cpython-312.pyc
bank_system.cpython-312.pyc
test_bank_functional.cpython-312...
test_bank_nonfunctional.cpython...
test_bank_system.cpython-312-py...
.pytest_cache
myenv
bank_model.py
test_bank_functional.py
test_bank_nonfunctional.py
bank_model.py
test_bank_functional.py
test_bank_nonfunctional.py
test_bank_functional.py > ...
# test_bank_functional.py
import pytest
from bank_model import Bank
@pytest.fixture
def bank():
return Bank()
def test_register_user(bank):
assert bank.register_user("alice", "password123") == "User registered successfully"
assert bank.register_user("alice", "password123") == "User already exists"
def test_login(bank):
bank.register_user("bob", "securepass")
assert bank.login("bob", "securepass") == "Login successful"
assert bank.login("bob", "wrongpass") == "Invalid credentials"
def test_deposit(bank):
bank.register_user("charlie", "mypassword")
assert bank.deposit("charlie", 100) == "Deposited $100 successfully"
assert bank.check_balance("charlie") == "Balance: $100"
def test_withdraw(bank):
bank.register_user("david", "pass123", 200)
assert bank.withdraw("david", 100) == "Withdrawn $100 successfully"
assert bank.withdraw("david", 200) == "Insufficient funds"
def test_transfer(bank):
bank.register_user("alice", "pass", 300)
bank.register_user("bob", "pass", 100)
assert bank.transfer("alice", "bob", 100) == "Transfer successful"
assert bank.transfer("alice", "bob", 300) == "Insufficient funds"

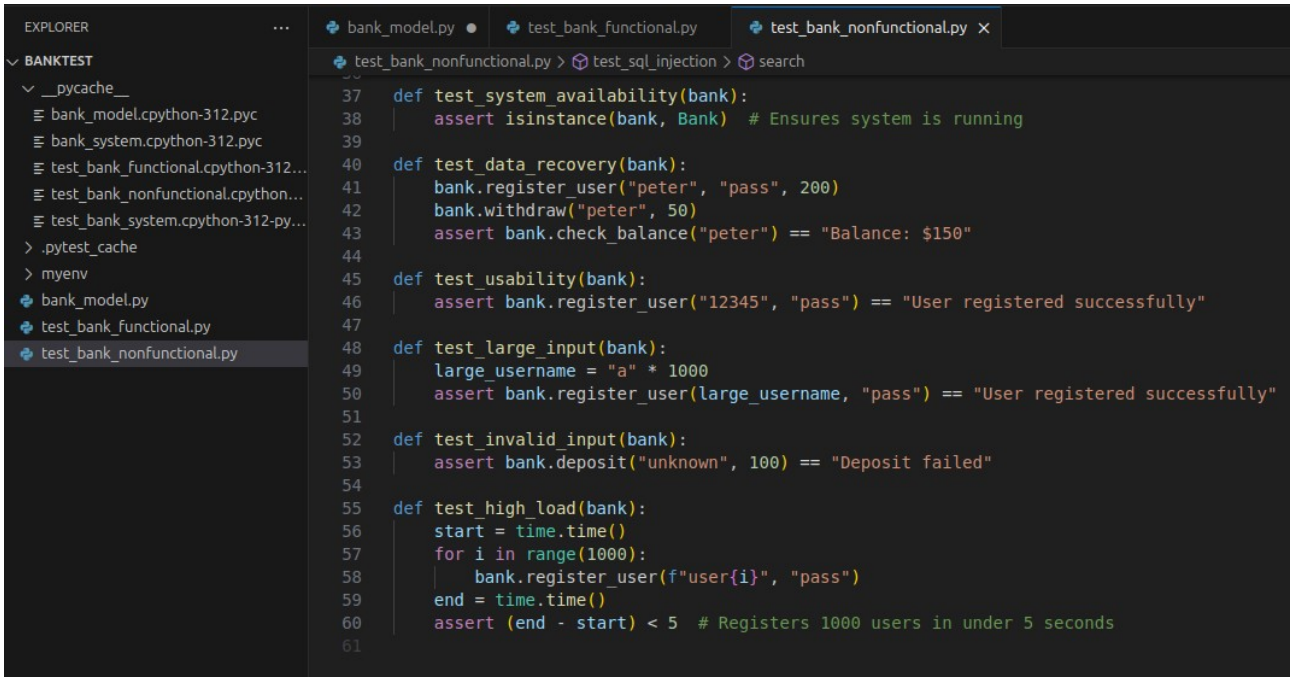

BankTest
EXPLORER
BANKTEST
__pycache__
bank_model.cpython-312.pyc
bank_system.cpython-312.pyc
test_bank_functional.cpython-312...
test_bank_nonfunctional.cpython...
test_bank_system.cpython-312-py...
.pytest_cache
myenv
bank_model.py
test_bank_functional.py
test_bank_nonfunctional.py
bank_model.py
test_bank_functional.py
test_bank_nonfunctional.py
test_bank_functional.py > ...
def test_check_balance(bank):
bank.register_user("eve", "password", 500)
assert bank.check_balance("eve") == "Balance: $500"
def test_admin_approval(bank):
assert bank.admins["admin"] == "admin123"
def test_loan_approval(bank):
bank.register_user("frank", "secure", 500)
assert bank.approve_loan("frank", 1000) == "Loan approved"
assert bank.approve_loan("frank", 200) == "Loan approved"
def test_failed_login(bank):
assert bank.login("unknown_user", "password") == "Invalid credentials"
def test_logout(bank):
bank.register_user("grace", "mypassword")
assert bank.logout("grace") == "User logged out"

Output for Functional Test Cases:

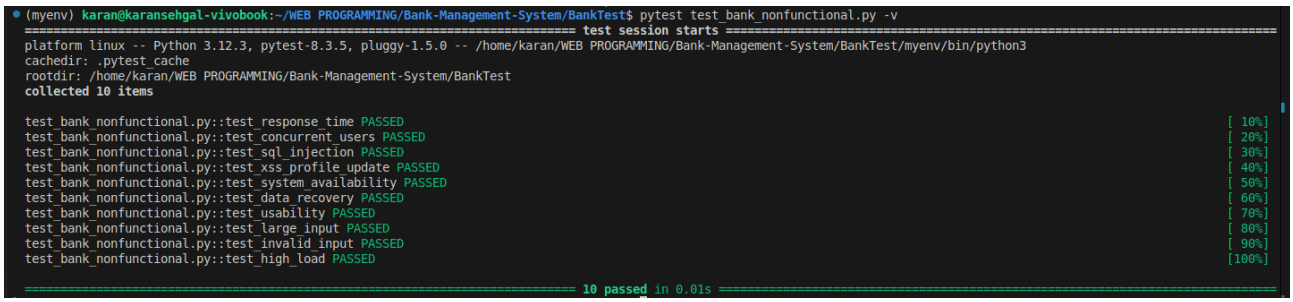


PyTest Code for *Non-Functional* Test Cases:





Output for Functional Test Cases:



10. Story Boarding and User Interface design Modelling

1. Storyboarding for Online Bank Management System

1.1 User Personas

The system is designed for three key users:

- **Customer**: Uses the system for transactions, account management, and loan applications.
- **Bank Employee (Manager/Admin)**: Approves loans, manages customers, and monitors bank activities.
- **System Administrator**: Manages user roles, security, and overall system performance.

1.2 Key Features & Screens

1. **Homepage**: Prompts users to log in or register as a first-time user.
2. **Registration Page**: Allows new users to sign up and wait for approval.
3. **Login Page**: Users enter credentials to access the system.
4. **Dashboard**: Displays account balance, transactions, and quick actions.
5. **Fund Transfer Page (Transaction Dashboard)**: Allows users to send money to another account.
6. **Loan Management Dashboard**: Customers apply for loans, employees approve/reject requests.
7. **User Profile & Statements**: Displays user details, past transactions, and account history.
8. **Support & Contact Us**: Provides assistance and customer service options.
9. **Logout**: Users can securely log out of the system.

1.3 Complete Customer Flow Storyboard

This storyboard illustrates the journey of a customer using the Online Bank Management System (banKS):

1. **Customer Visits Homepage**: The customer opens the banKS website and sees options to log in or register as a first-time user.

2. **Customer Registers as a New User**: The customer selects the "Register" option, enters personal details, and submits the application for approval.
3. **Customer Waits for Application Processing**: A confirmation message appears, informing the customer that the application is under review.
4. **Customer Logs into the System**: Once approved, the customer logs in using their registered credentials.
5. **Customer Accesses Dashboard**: After logging in, the customer is redirected to the dashboard, where account details, transaction history, and quick access options are available.
6. **Customer Initiates a Fund Transfer**: The customer selects the "Transaction Dashboard", enters recipient details, transaction amount, and submits the transfer.
7. **Customer Receives Transfer Confirmation**: The system confirms the transaction, and the balance updates accordingly.
8. **Customer Applies for a Loan**: The customer navigates to the "Loan Management Dashboard", fills in loan details, and submits the application.
9. **Customer Views Profile & Past Statements**: The customer accesses the profile section to check transaction history, download statements, or update account settings.
10. **Customer Seeks Support (If Needed)**: The customer visits the "Support & Contact Us" section for FAQs or assistance.
11. **Customer Logs Out**: After completing banking activities, the customer securely logs out.

2. UI Design Modeling

2.1 UI Requirements

- **Branding**: Blue and white theme for trust and professionalism.
- **Navigation**: Intuitive sidebar for quick access to key features.
- **Accessibility**: Large buttons, clear fonts, and proper contrast for readability.

2.2 Wireframes for Key Screens

1. **Homepage:**

- A welcoming screen with options to **Login** or **Register**.
- Clean and modern layout with bank branding and an informational footer.

2. **Registration Page:**
    - Fields for **name, email, password, and account type selection**.
    - A confirmation message stating that the **application is under review**.
3. **Login Page:**
    - Input fields for **username and password**.
    - A "Forgot Password" link for password recovery.
4. **Dashboard:**
    - Displays **account balance, quick actions, and recent transactions**.
    - Quick access buttons for **fund transfers and loan management**.
5. **Fund Transfer Page (Transaction Dashboard):**
    - A form where users select a recipient, enter the amount, and submit transactions.
    - A confirmation message upon successful transfer.
6. **Loan Management Dashboard:**
    - A user-friendly interface to apply for loans.
    - Fields for **loan amount, tenure, and interest rates**.
    - Displays **loan approval status**.
7. **User Profile & Statements:**
    - A profile section displaying **personal details, account settings, and past statements**.
    - Users can download **transaction statements**.
8. **Support & Contact Us:**
    - Users can access **FAQs, support tickets, and customer care contacts**.
    - A feedback form for user inquiries.

2.3 Descriptions of Website UI Designs

1. **Homepage**: Displays a login and registration option, along with bank branding and an informative footer.
2. **Registration Page**: Users enter their details to create an account. A confirmation message is displayed stating that the application is under review.
3. **Login Page**: Provides fields for username and password, along with a "Forgot Password" link.
4. **Dashboard**: Displays account details, recent transactions, and options to transfer funds or apply for loans.
5. **Fund Transfer Page**: Allows users to select a recipient, enter the transfer amount, and confirm the transaction.
6. **Loan Management Dashboard**: Users can apply for loans by entering the required amount and selecting repayment terms.
7. **User Profile Page**: Displays account details, transaction history, and an option to download statements.
8. **Support & Contact Us Page**: Provides users with access to FAQs, support tickets, and customer service contact details.
9. **Logout**: Users can securely log out of their account.

2.4 Interaction Design

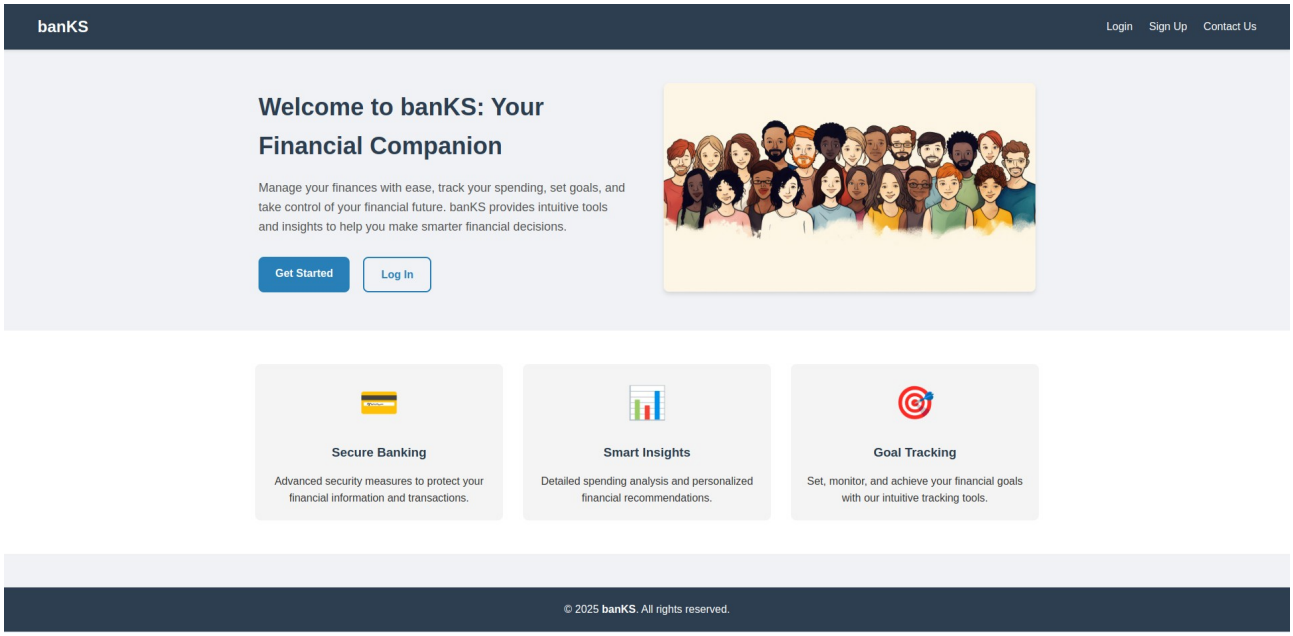
- **Buttons:** "Register", "Login", "Transfer", "Apply for Loan", "View Statements" with hover effects.
- **Form Validations:** Real-time error messages for invalid input.
- **Navigation Flow:** Smooth transitions between screens.

2.5 Prototyping & Testing

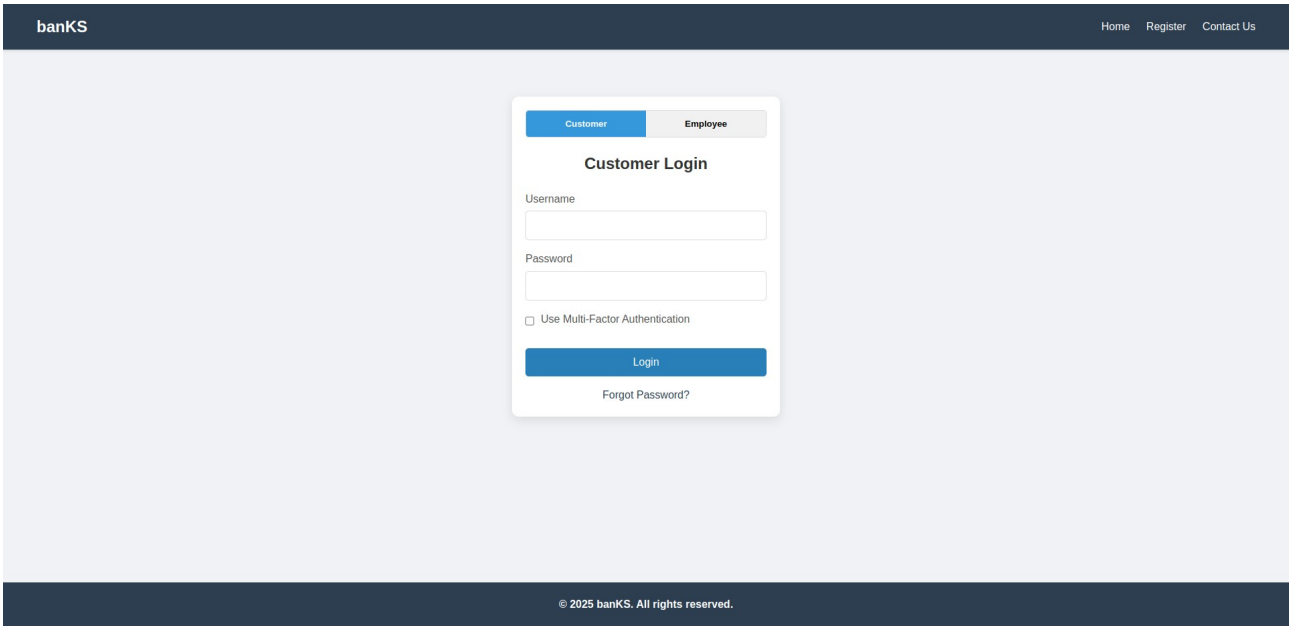
- **Interactive Prototype**: Designed in Figma and made with HTML,CSS, JS,REACT.
- **User Testing**: Test with 5 users for feedback on ease of navigation.
- **Final Adjustments**: Modify UI based on usability feedback.

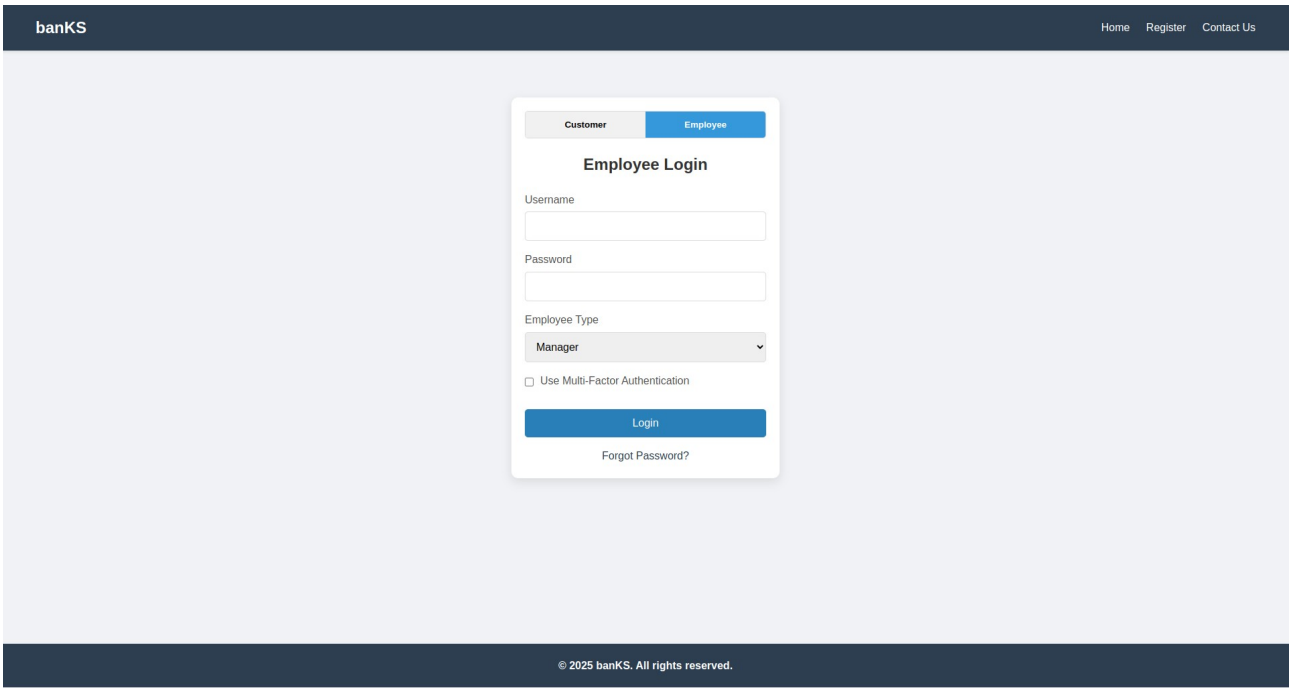
3. UI Design using website screenshots:

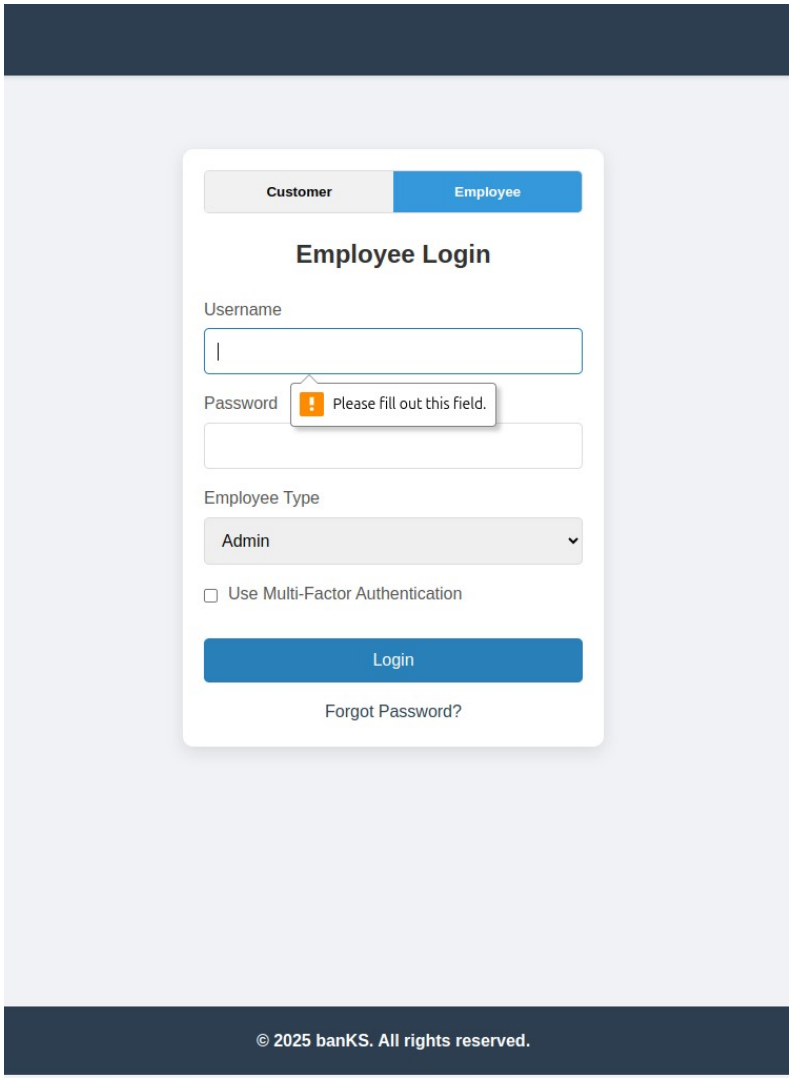
1) Hompage

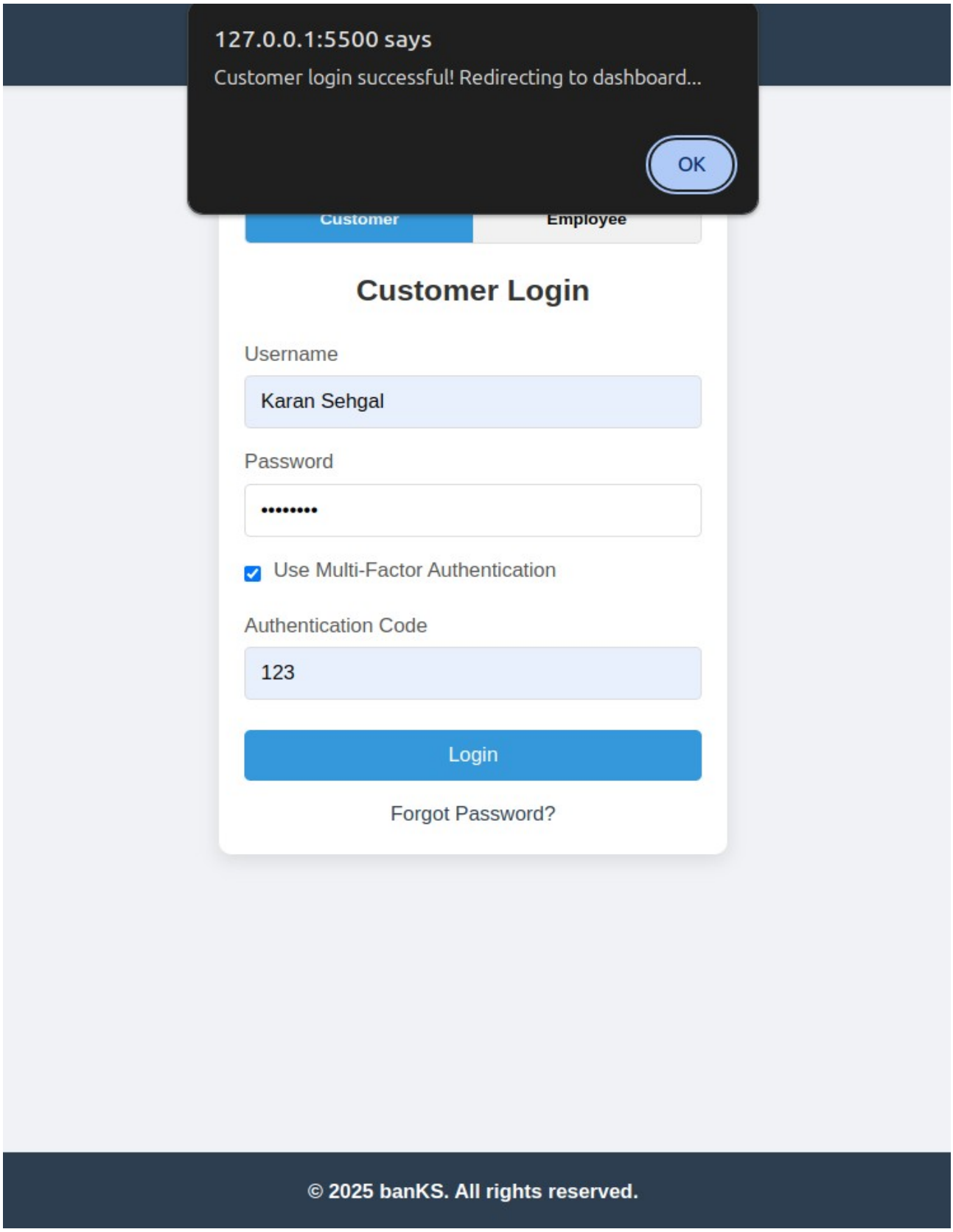


2) Login (provides two option : login as a customer or an employee)



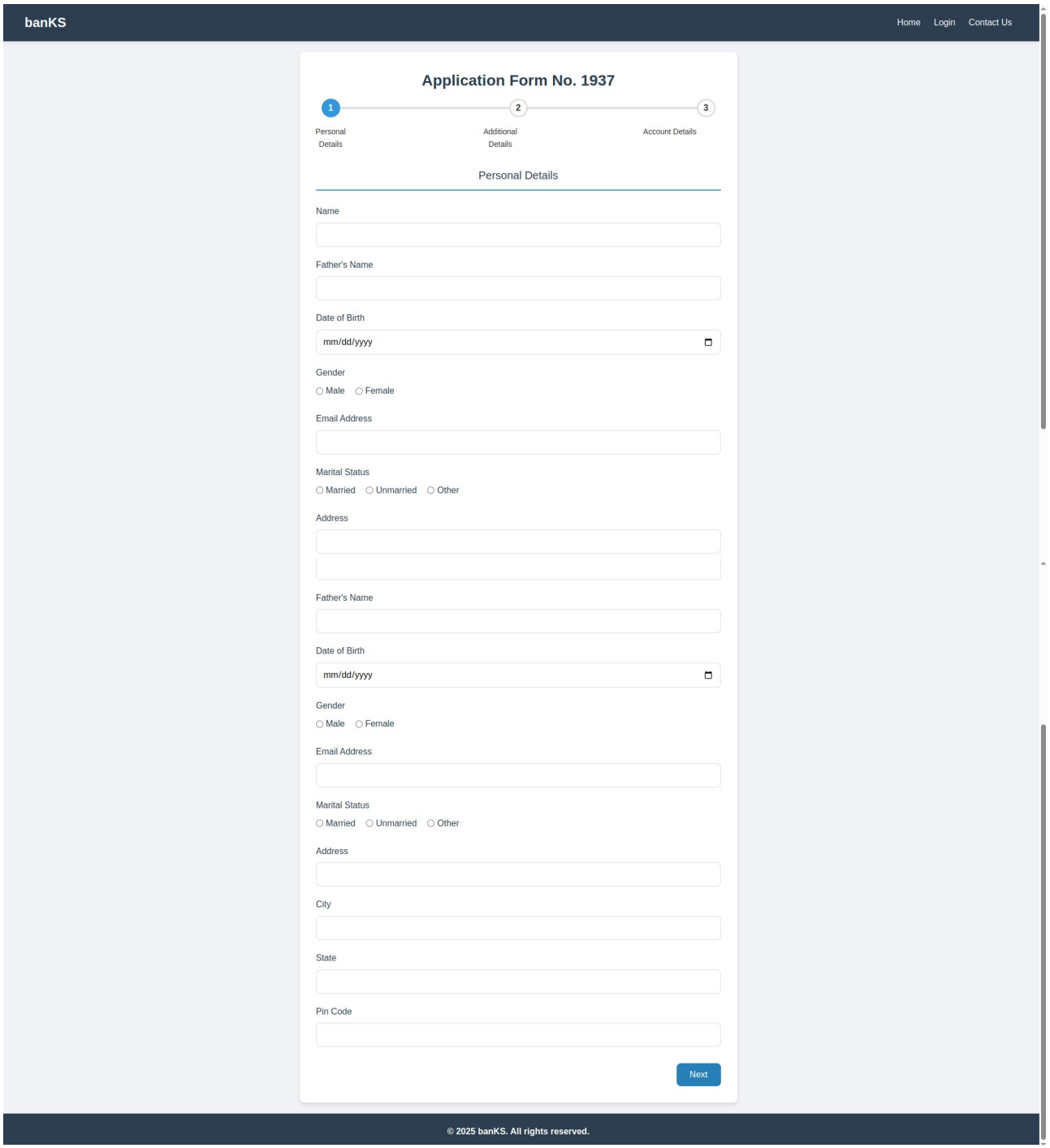

banKS
Home
Register
Contact Us
Customer
Employee
Employee Login
Username
Password
Employee Type
Manager
Use Multi-Factor Authentication
Login
Forgot Password?
© 2025 banKS. All rights reserved.


Customer
Employee
Employee Login
Username
Password
Please fill out this field.
Employee Type
Admin
Use Multi-Factor Authentication
Login
Forgot Password?
© 2025 banKS. All rights reserved.

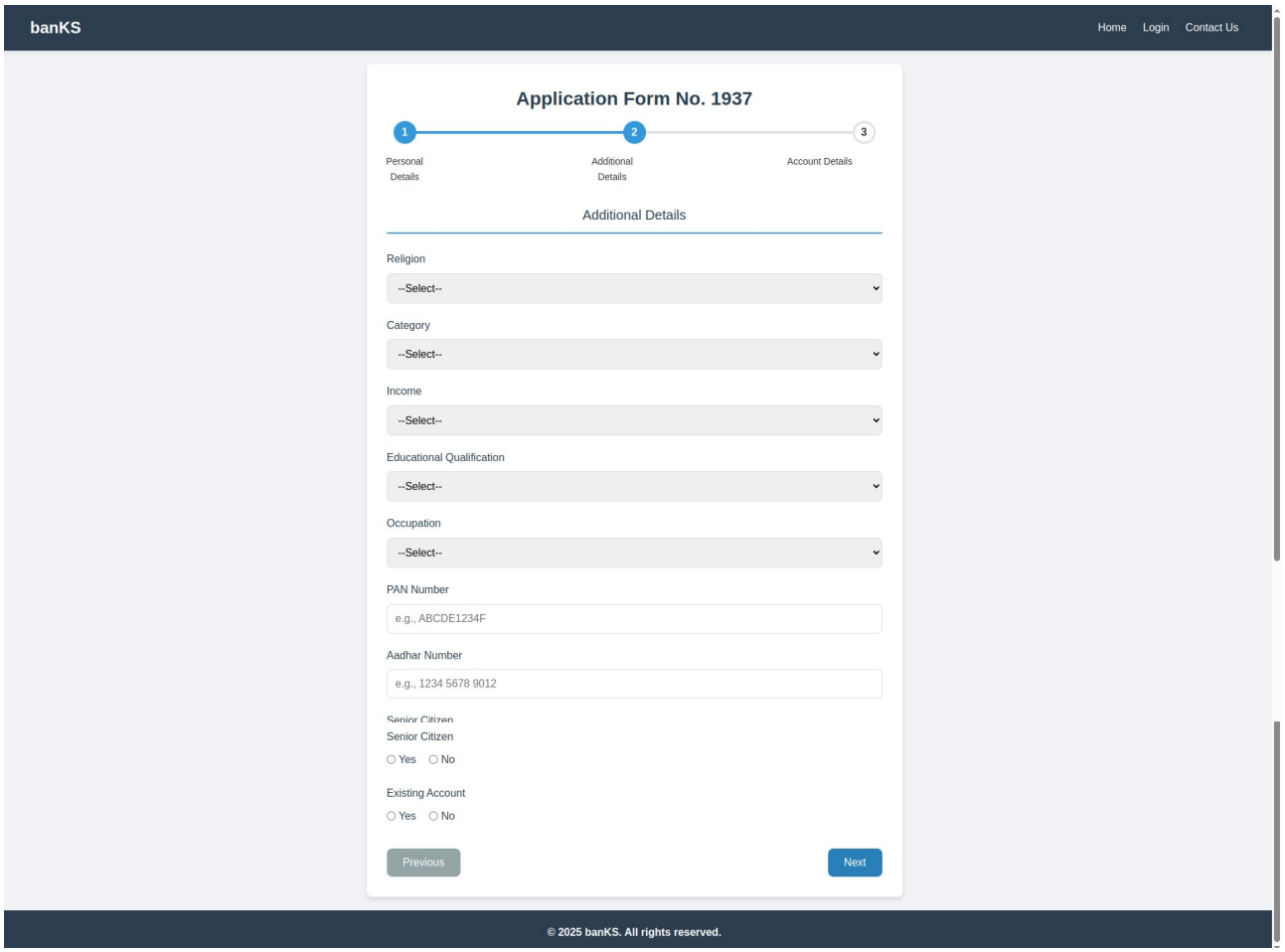

127.0.0.1:5500 says
Customer login successful! Redirecting to dashboard...
OK
Customer
Employee
Customer Login
Username
Karan Sehgal
Password
Use Multi-Factor Authentication
Authentication Code
123
Login
Forgot Password?
© 2025 banKS. All rights reserved.

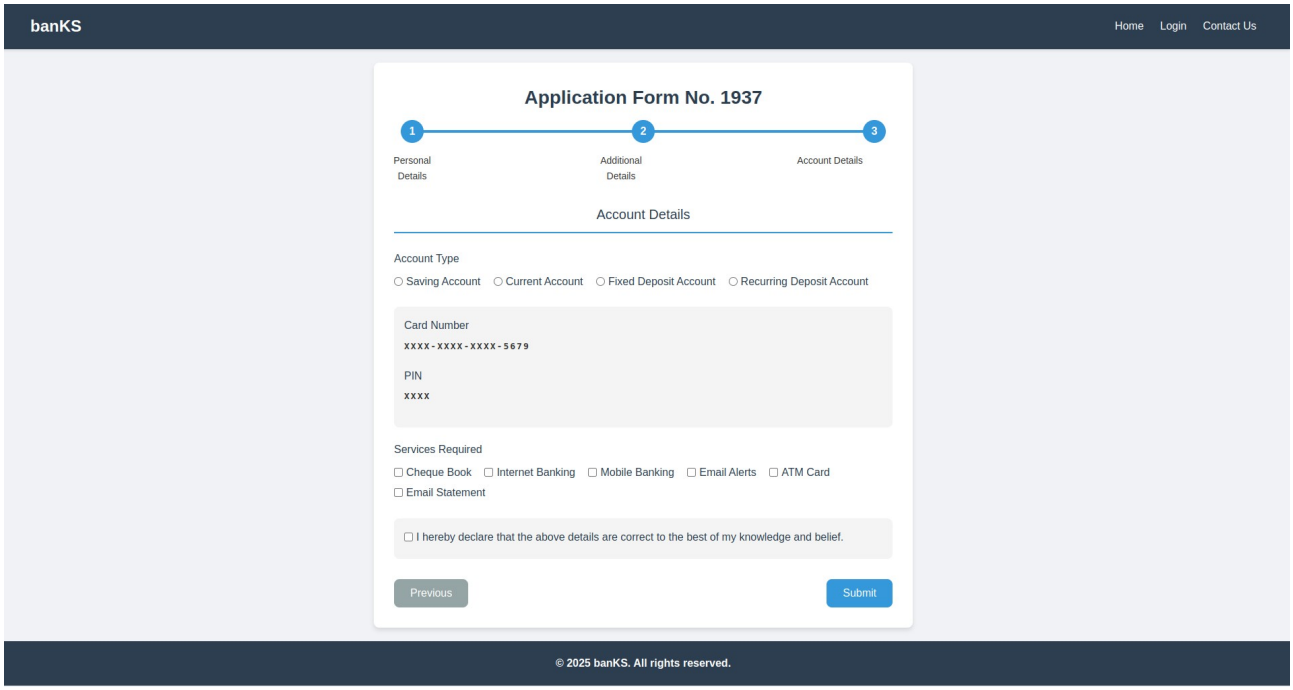
3) Register as a new user :

page1

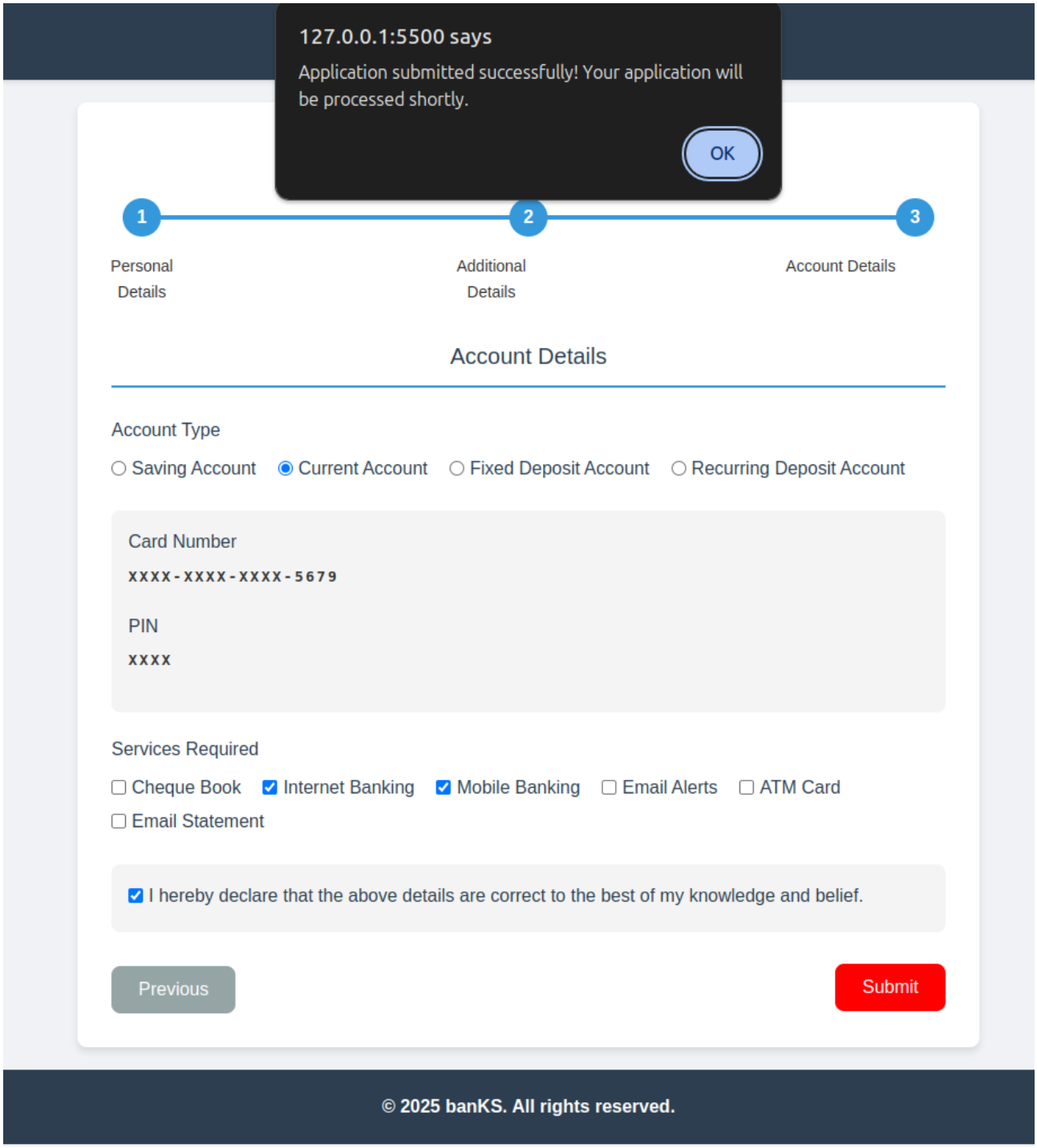


page 2

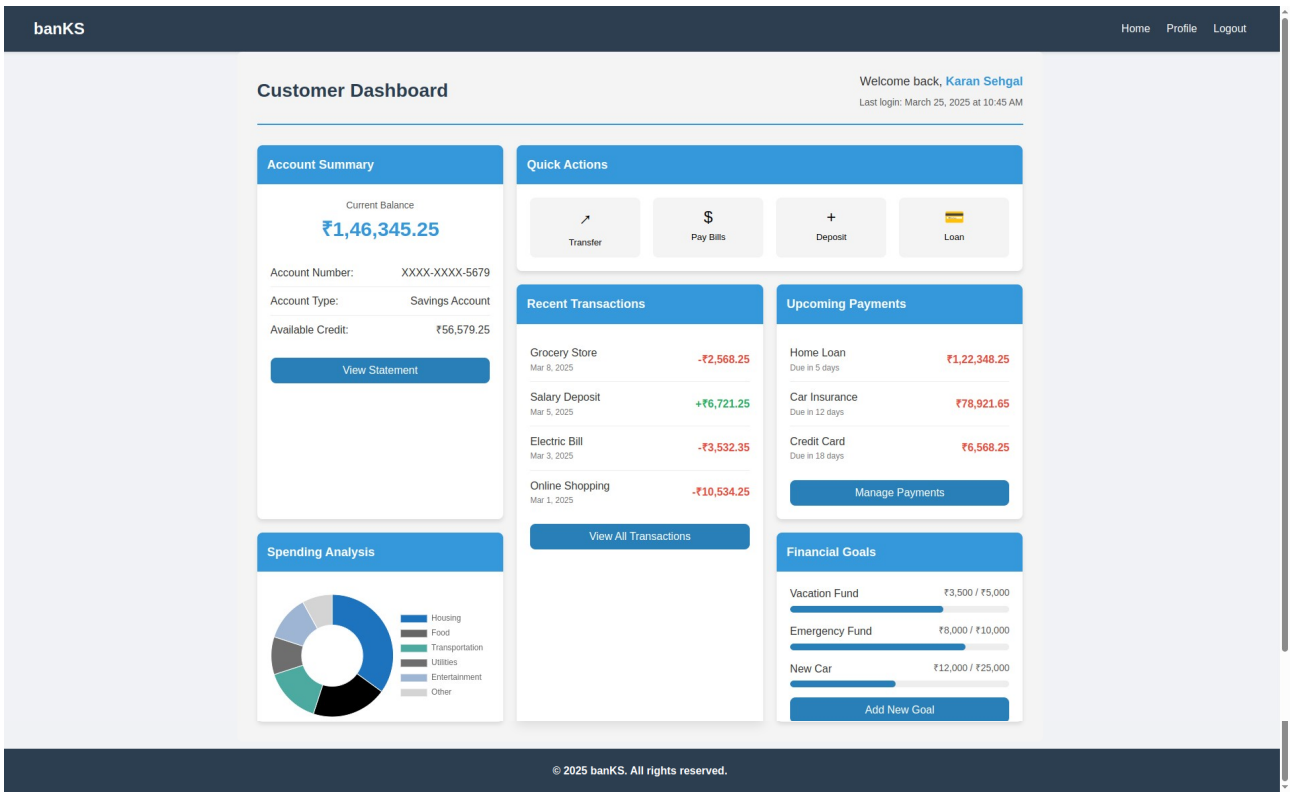

banKS
Home
Login
Contact Us
Application Form No. 1937
1
2
3
Personal Details
Additional Details
Account Details
Additional Details
Religion
--Select--
Category
--Select--
Income
--Select--
Educational Qualification
--Select--
Occupation
--Select--
PAN Number
e.g., ABCDE1234F
Aadhar Number
e.g., 1234 5678 9012
Senior Citizen
Senior Citizen
Yes
No
Existing Account
Yes
No
Previous
Next
© 2025 banKS. All rights reserved.


banKS
Home
Login
Contact Us
Application Form No. 1937
1
2
3
Personal Details
Additional Details
Account Details
Account Details
Account Type
Saving Account
Current Account
Fixed Deposit Account
Recurring Deposit Account
Card Number
xxxx-xxxx-xxxx-5679
PIN
xxxx
Services Required
Cheque Book
Internet Banking
Mobile Banking
Email Alerts
ATM Card
Email Statement
I hereby declare that the above details are correct to the best of my knowledge and belief.
Previous
Submit
© 2025 banKS. All rights reserved.

Waiting for application being processed, *user is directed to homepage*



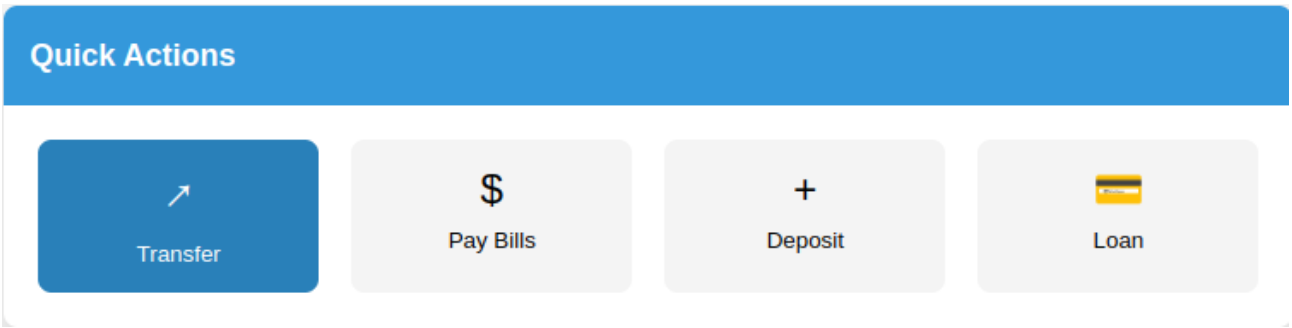
4) Once a user logins in successfully, He/she lands at the *customer dashboard*:



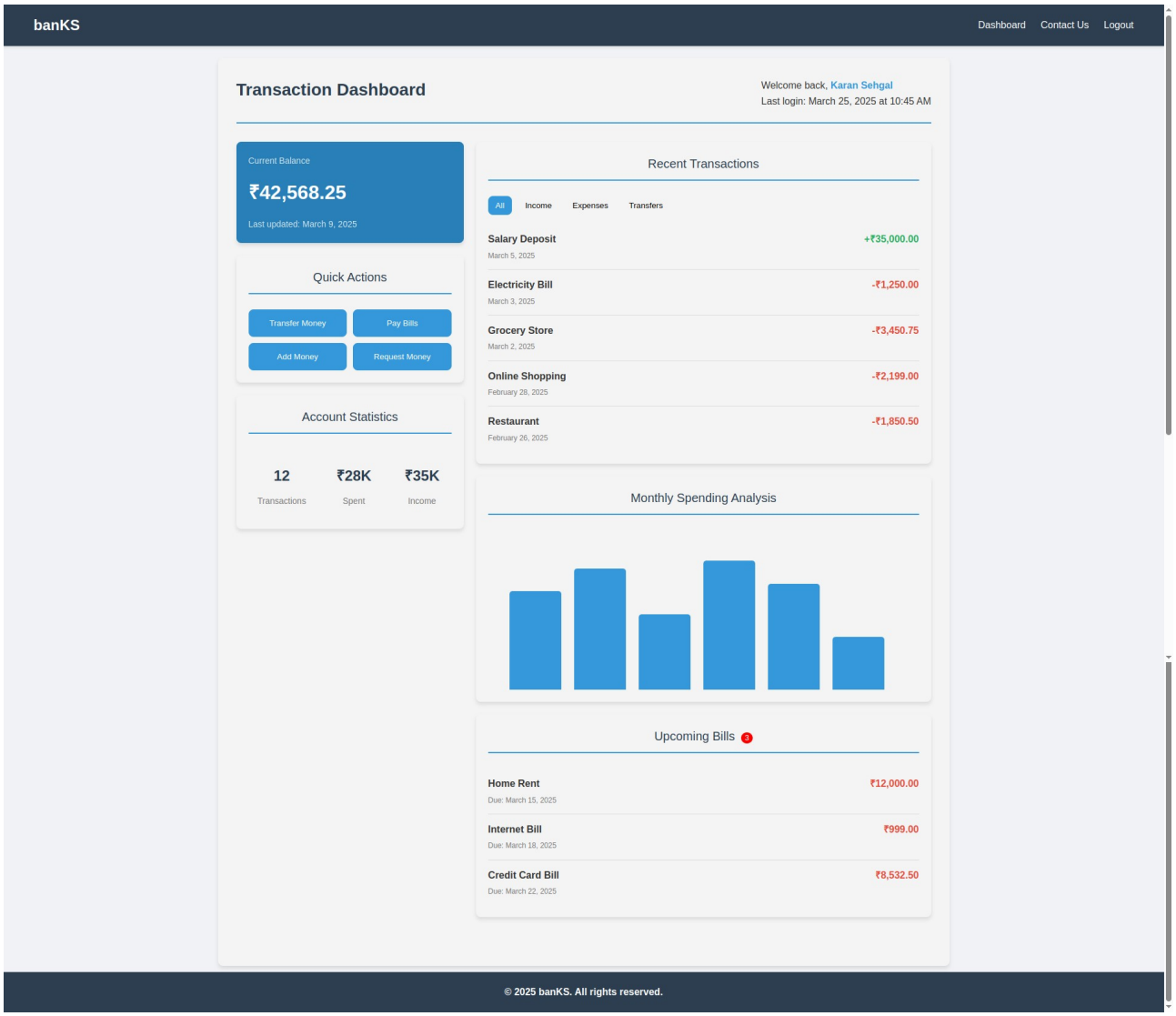
This pane allows user to access one of many functionalities :

transfer → *Transaction Management Dashboard*

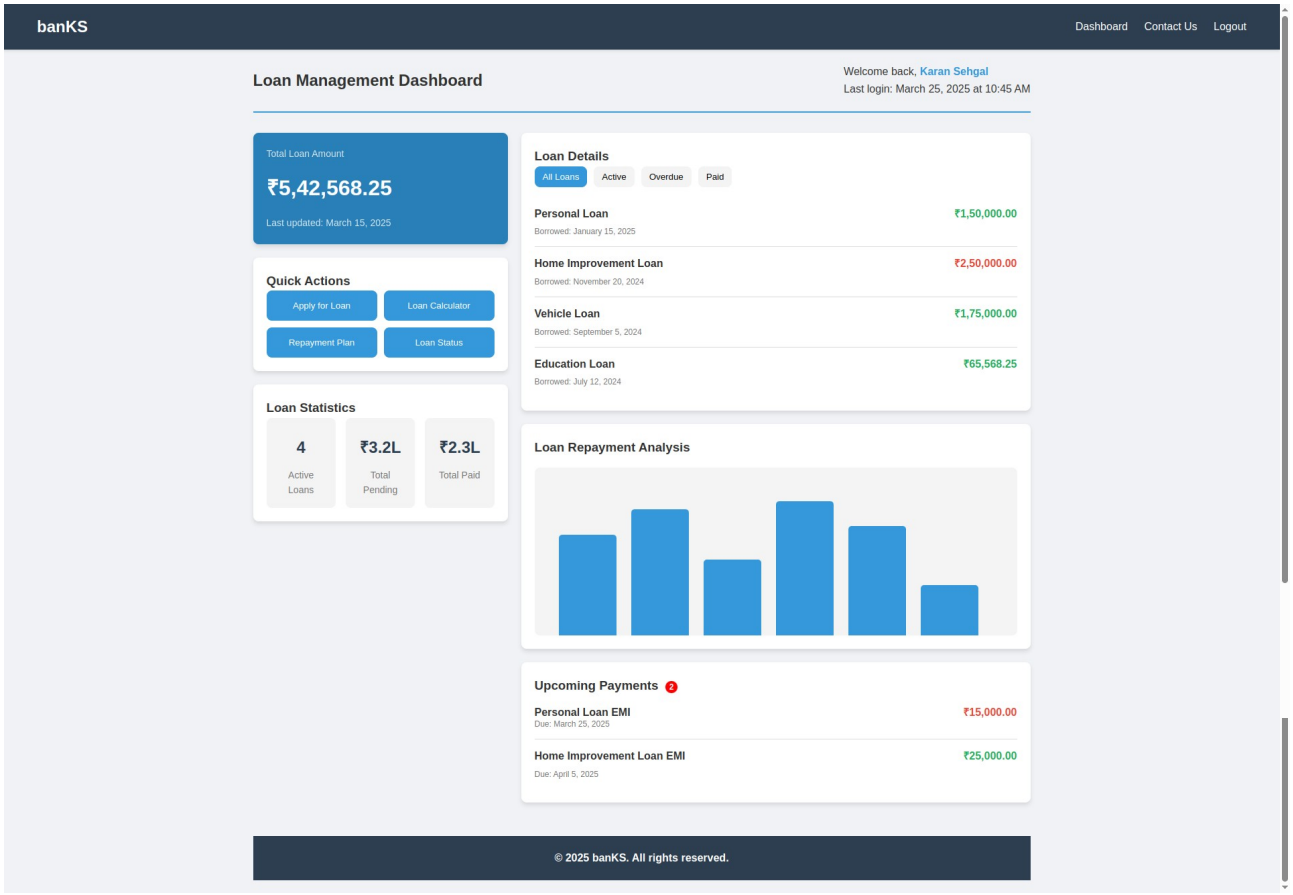
Loan → *Loan Management Dashboard*

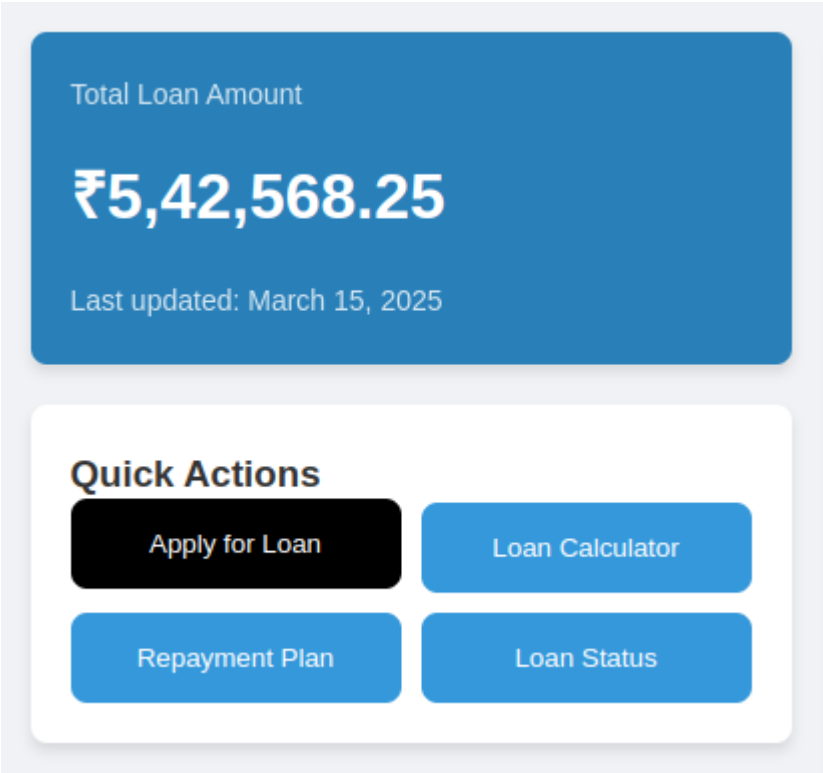


Transaction Management Dashboard:

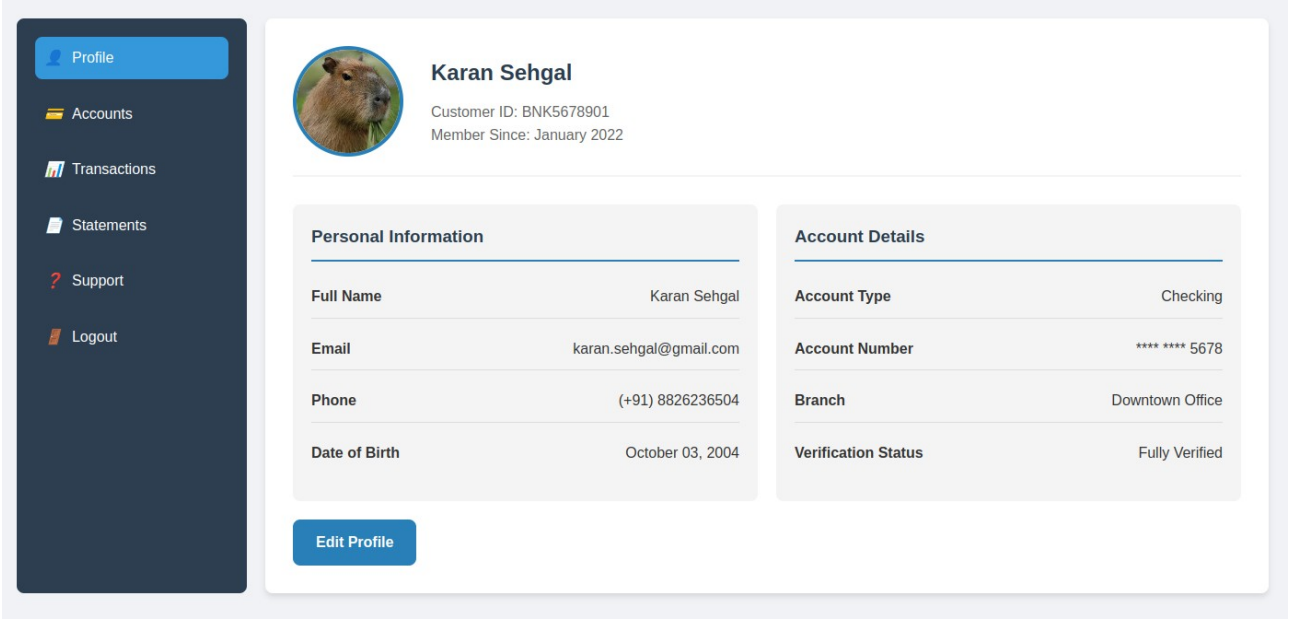


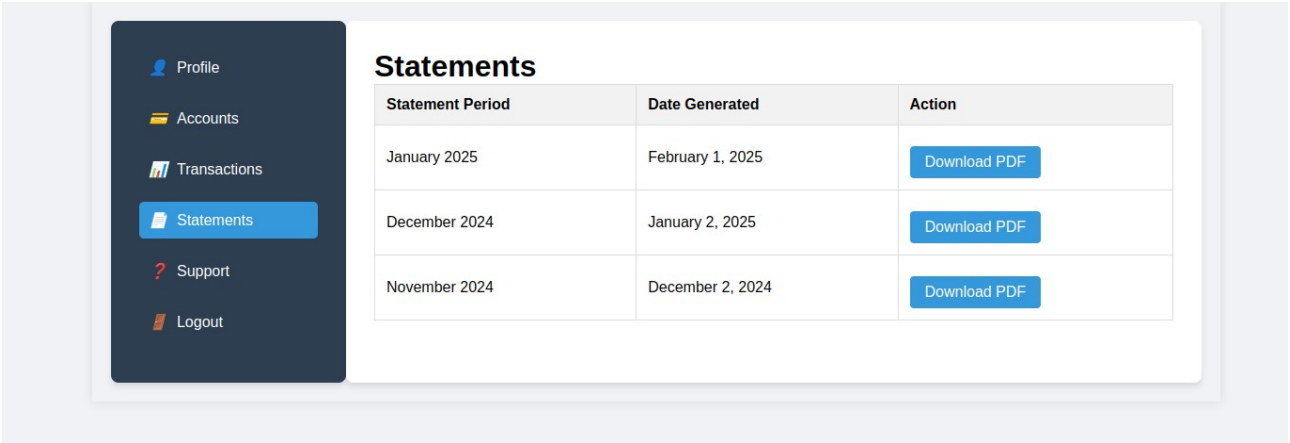
Loan Management Dashboard:

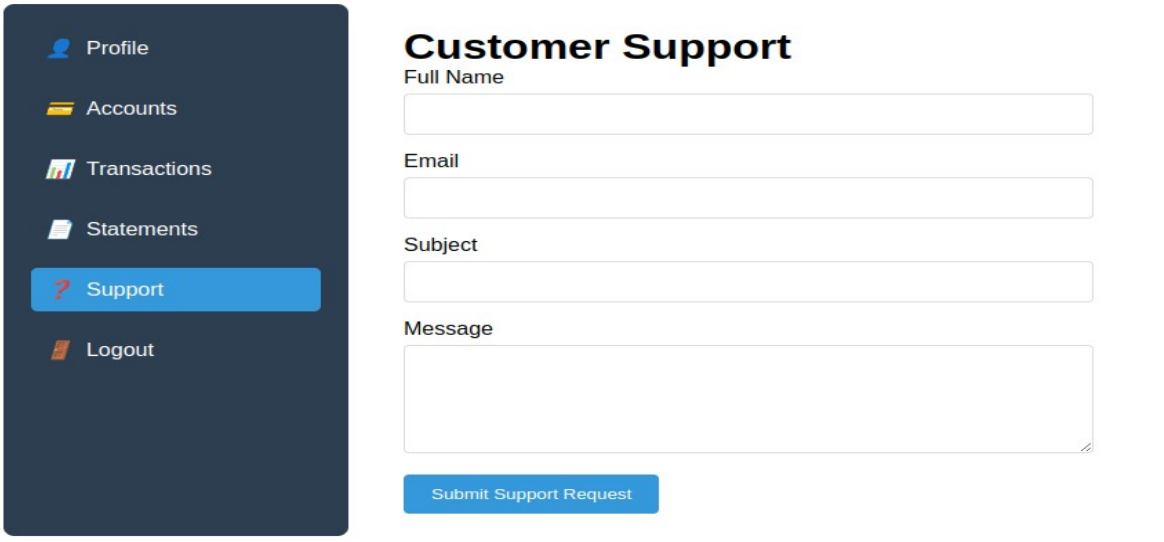




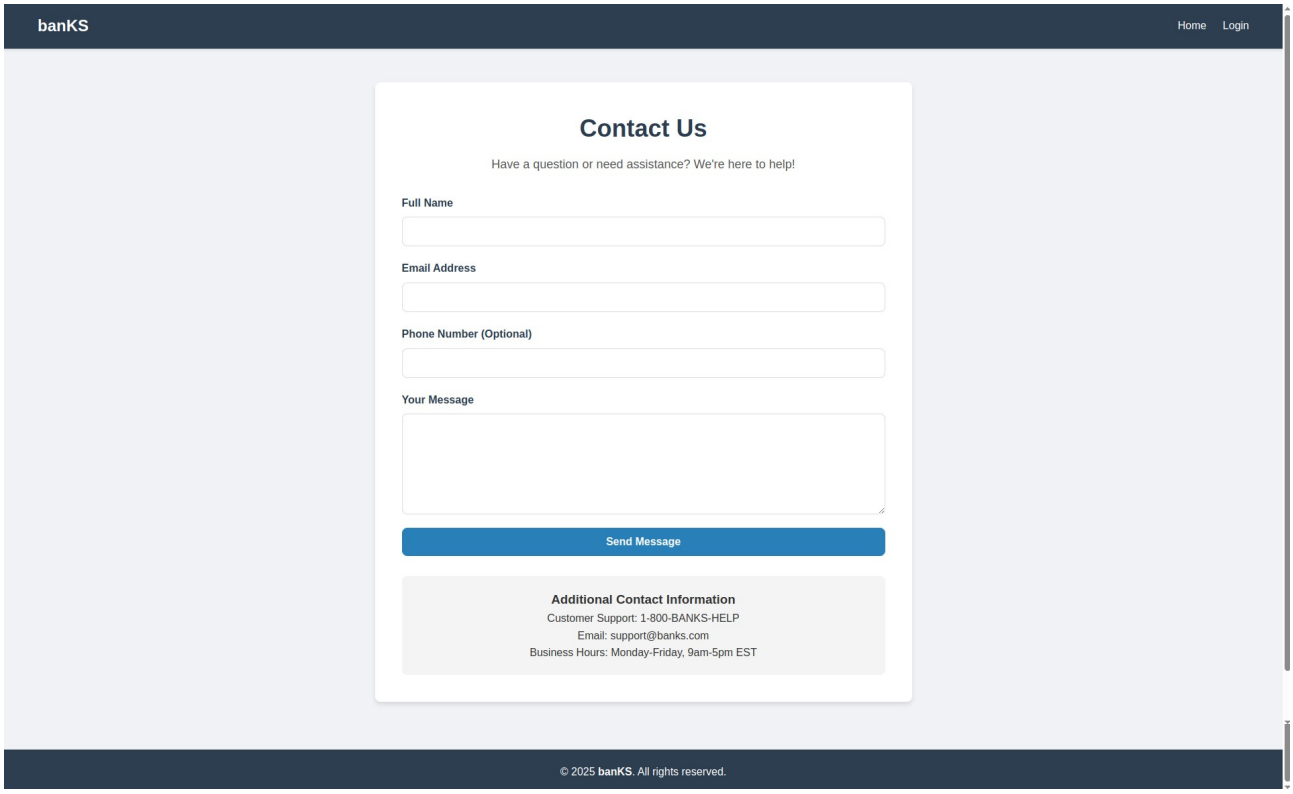
Profile


Profile
Accounts
Transactions
Statements
Support
Logout
Karan Sehgal
Customer ID: BNK5678901
Member Since: January 2022
Personal Information
Full Name
Karan Sehgal
Email
karan.sehgal@gmail.com
Phone
(+91) 8826236504
Date of Birth
October 03, 2004
Account Details
Account Type
Checking
Account Number
**** **** 5678
Branch
Downtown Office
Verification Status
Fully Verified
Edit Profile

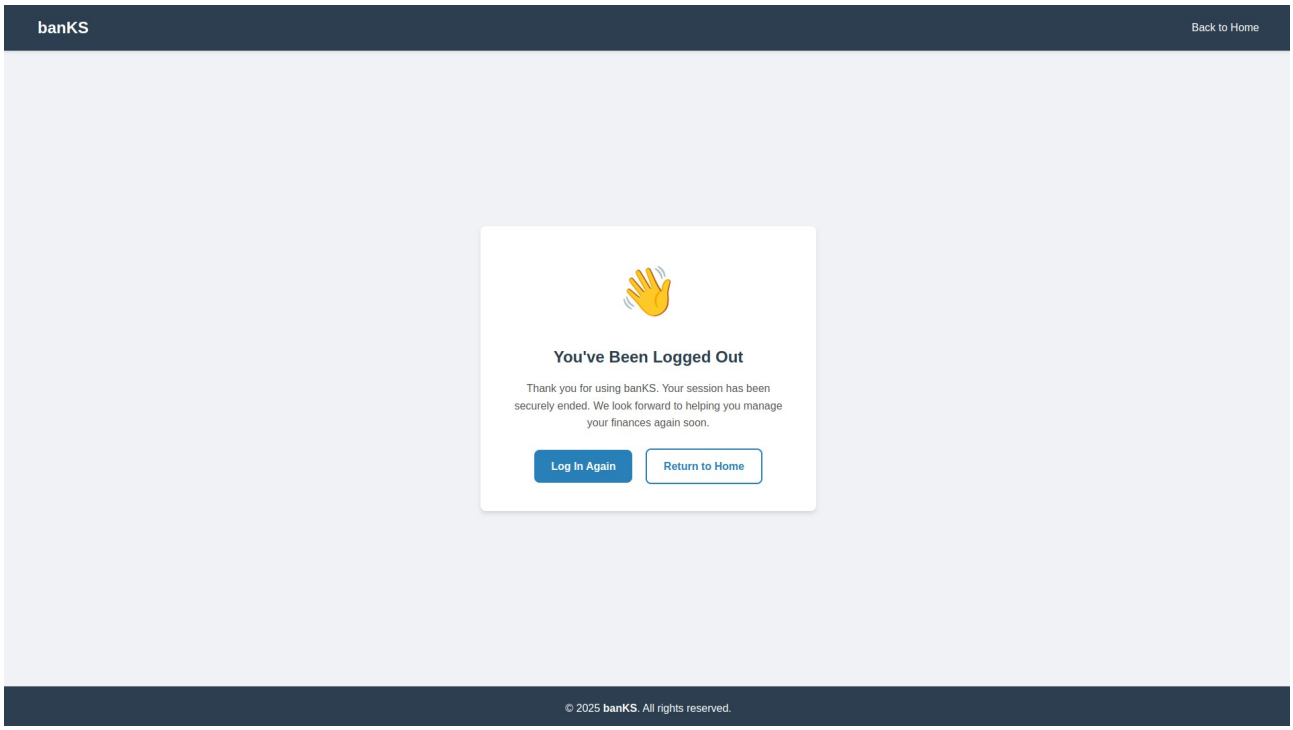

Profile
Accounts
Transactions
Statements
Support
Logout
Statements
Statement Period
Date Generated
Action
January 2025
February 1, 2025
Download PDF
December 2024
January 2, 2025
Download PDF
November 2024
December 2, 2024
Download PDF


Profile
Accounts
Transactions
Statements
Support
Logout
Customer Support
Full Name
Email
Subject
Message
Submit Support Request

ContactUs



LogOut:



Github Link : *https://github.com/Karansehgal0611/banKS*

11. Challenges and Solutions

During the development of the **Online Bank Management System (banKS)**, several challenges arose, requiring strategic problem-solving and adaptation. The major challenges and their solutions are as follows:

1. System Design Complexity

- **Challenge**: Ensuring a well-structured, scalable, and efficient design that meets banking standards.
- **Solution**: Used **StarUML** and **ArgoUML** for requirement modeling and OO design, which helped in defining clear system architecture and relationships between components.

2. Data Security & Integrity

- **Challenge**: Protecting sensitive customer data, including transactions and account details.
- **Solution**: Implemented encryption techniques, secure authentication (MFA), and access control mechanisms to restrict unauthorized access.

3. Functional & Performance Testing

- **Challenge**: Ensuring all functionalities work correctly and meet expected performance benchmarks.
- **Solution**: Conducted **functional and non-functional testing** using **pytest**, which helped in identifying and fixing critical bugs.

4. User Interface & Experience (UI/UX)

- **Challenge**: Designing an intuitive and user-friendly interface for customers, employees, and administrators.
- **Solution**: Followed **UI design principles**, created **wireframes and storyboarding**, and iterated on user feedback for improvements.

5. Handling Concurrent Transactions

- **Challenge**: Ensuring data consistency when multiple users access or update data simultaneously.
- **Solution**: Used **transaction locking mechanisms**, **ACID compliance**, and **proper error handling** to maintain data consistency.

12. Conclusion and Future enhancements

1. Conclusion

The **Online Bank Management System (banKS)** successfully demonstrates a comprehensive banking solution by integrating essential banking functionalities, including **customer account management, transactions, loan processing, and administrative controls**. Through **structured requirement analysis, UML-based design, functional modeling, and rigorous testing**, the system ensures reliability, security, and efficiency.

The project also provided valuable insights into **process modeling, object-oriented system design, UI/UX considerations, and security measures**. By leveraging modern web development technologies and industry-standard best practices, the system aligns with the evolving needs of digital banking.

2. Future Enhancements

To further improve the system, the following enhancements can be implemented:

1. **AI-Powered Fraud Detection**
    - Integrate **machine learning models** to detect fraudulent transactions in real time.
2. **Automated Loan Processing with Credit Scoring**
    - Implement AI-driven **credit risk analysis** for faster and smarter loan approvals.
3. **Mobile Banking Application**
    - Develop a **mobile app** to allow users to manage their accounts on the go.
4. **Multi-Currency & International Banking Support**
    - Extend support for multi-currency transactions and international banking services.
5. **Blockchain Integration for Enhanced Security**
    - Implement **blockchain-based identity verification** and **secure transaction logging**.
6. **Chatbot for Customer Support**
    - Deploy an **AI chatbot** for automated customer assistance and queries.

7. **Enhanced Data Analytics Dashboard**
    - Provide **visual reports** and **data insights** for admins and customers.

These future enhancements will improve the **usability, efficiency, and security** of the system, making it a **fully-fledged banking solution**.