UE22CS352B - Object Oriented Analysis & Design

Mini Project Report

Secure Bank- Bank Management System

Submitted by:

Shreya MP: PES1UG22CS574
Sinchana H: PES1UG22CS596
Spoorthi J: PES1UG22CS606
Spurti Bhat: PES1UG22CS607

6th Semester 'J' Section

Dr. Bhargavi Mokashi
January - May 2025

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
FACULTY OF ENGINEERING
PES UNIVERSITY
(Established under Karnataka Act No. 16 of 2013)
100ft Ring Road, Bengaluru – 560 085, Karnataka, India

Problem Statement:

The current banking environment demands a robust, scalable, and secure digital infrastructure to manage customer accounts, transactions, loans, and user authentication. Traditional systems often lack modularity, maintainability, and adaptability to new requirements. Moreover, poorly structured codebases lead to inefficient development, difficulty in testing, and potential security vulnerabilities.

The goal of this project is to design and implement a Bank Management System using Java Spring Boot, incorporating well-established design patterns to ensure maintainability, scalability, and adherence to software engineering best practices. The system must support multiple user roles, facilitate secure financial operations, and allow seamless interaction between components using layered architecture.

Key Features:

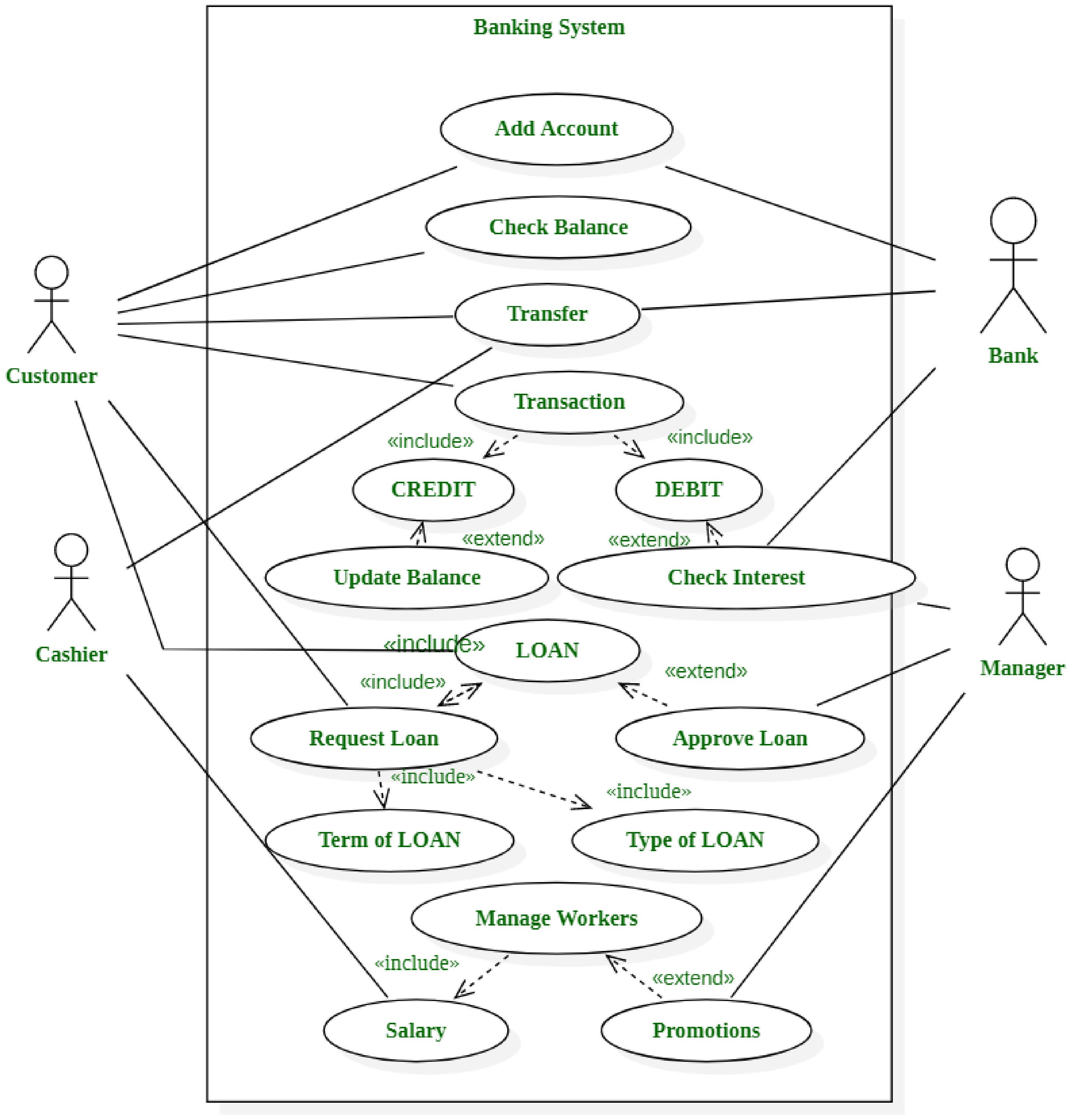
1. **User Management**
   - User registration with validation (DTOs and service layer logic)
   - Role-based access (Admin, Customer)
   - Secure login and logout using JWT authentication
2. **Account Management**
   - Create new bank accounts
   - View account details and current balance
   - Update account information
3. **Transaction Handling**
   - Deposit and withdraw functionality
   - View transaction history
   - Ensure transactional integrity using @Transactional
4. **Loan Module**
   - Apply for loans
   - View loan status
   - Admin approval and tracking
5. **Authentication & Security**
   - JWT-based login with Spring Security
   - Password encryption using BCryptPasswordEncoder
   - Global exception handling for better API responses
6. **RESTful APIs**
   - Clean, modular REST APIs following best practices
   - JSON-based request and response formats
   - Input validation using annotations (@Valid, @NotNull, etc.)
7. **Design Pattern Implementation**
   - Repository Pattern for data access abstraction

- Service Layer Pattern for business logic encapsulation
- DTO Pattern to separate internal models from exposed APIs
- Builder Pattern (via Lombok) for clean object creation
- Singleton, Factory, and Strategy patterns via Spring framework features
- Exception Handling Pattern using @ControllerAdvice

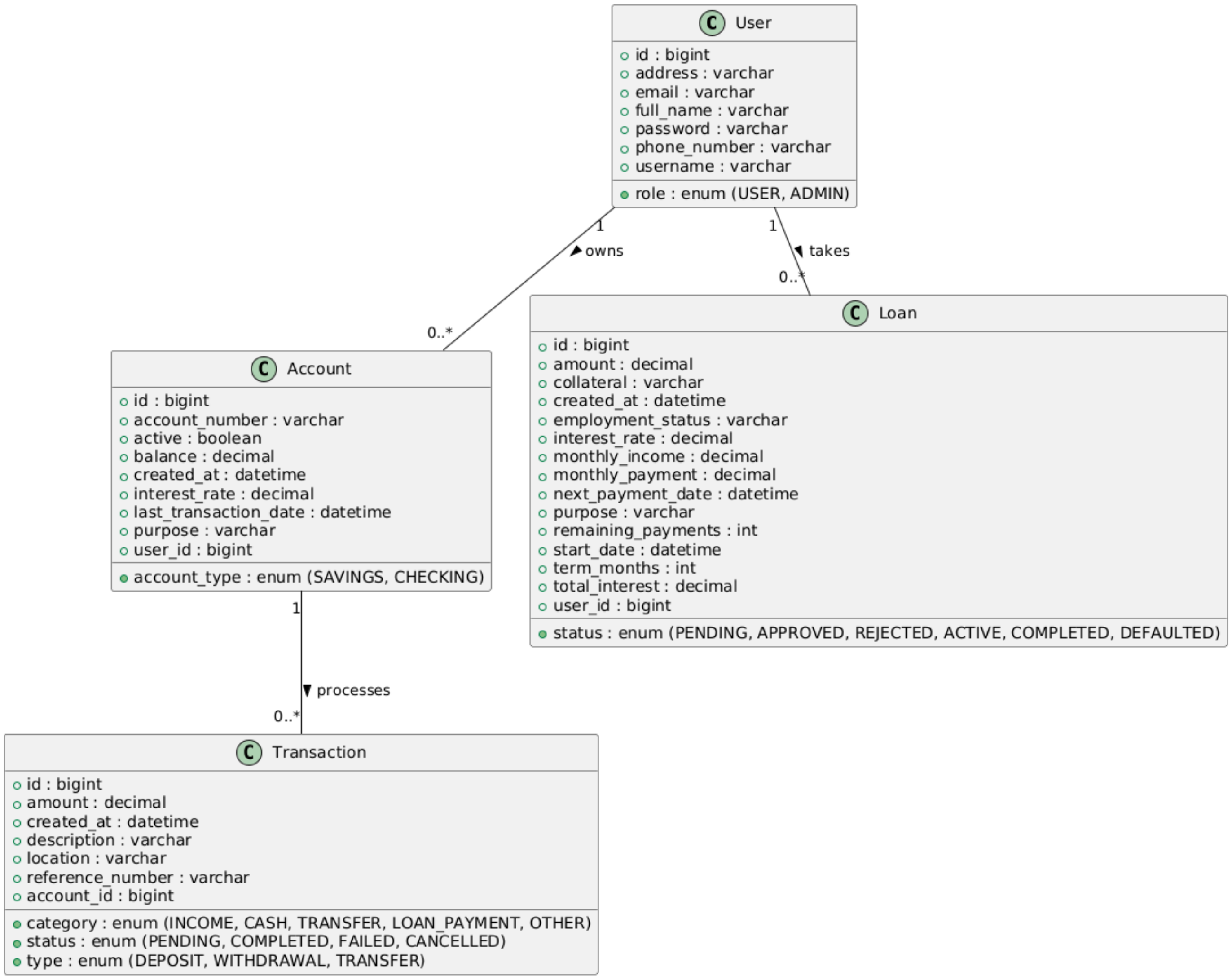
8. **Technology Stack**
   - Backend: Java, Spring Boot, Spring Data JPA, Spring Security
   - Database: MySQL
   - Build Tool: Maven
9. **Architecture**
   - MVC (Model-View-Controller) layered architecture
   - Clean separation between controller, service, repository, and model

Models:

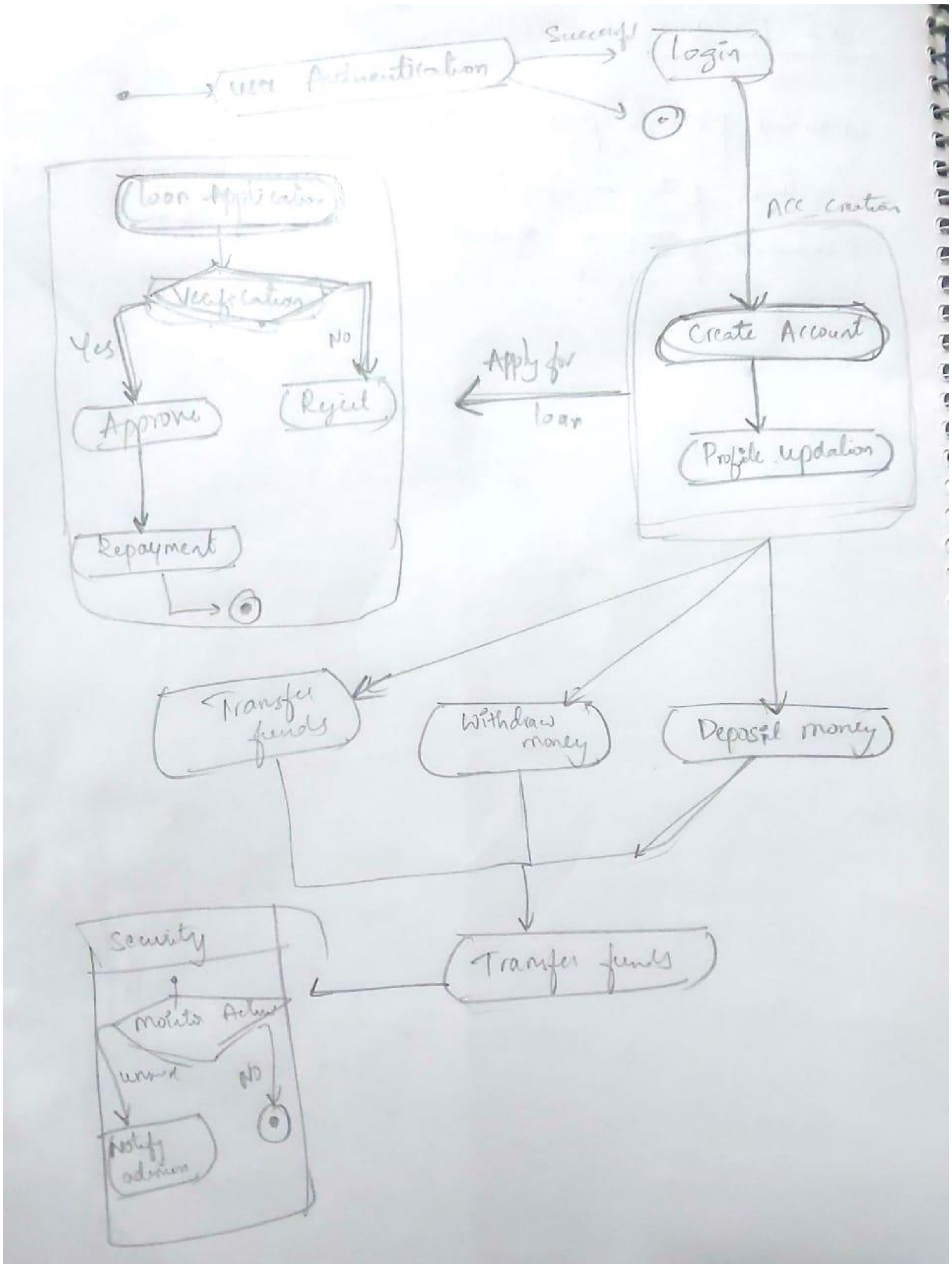
<u>Use Case Diagram:</u>



Class Diagram:

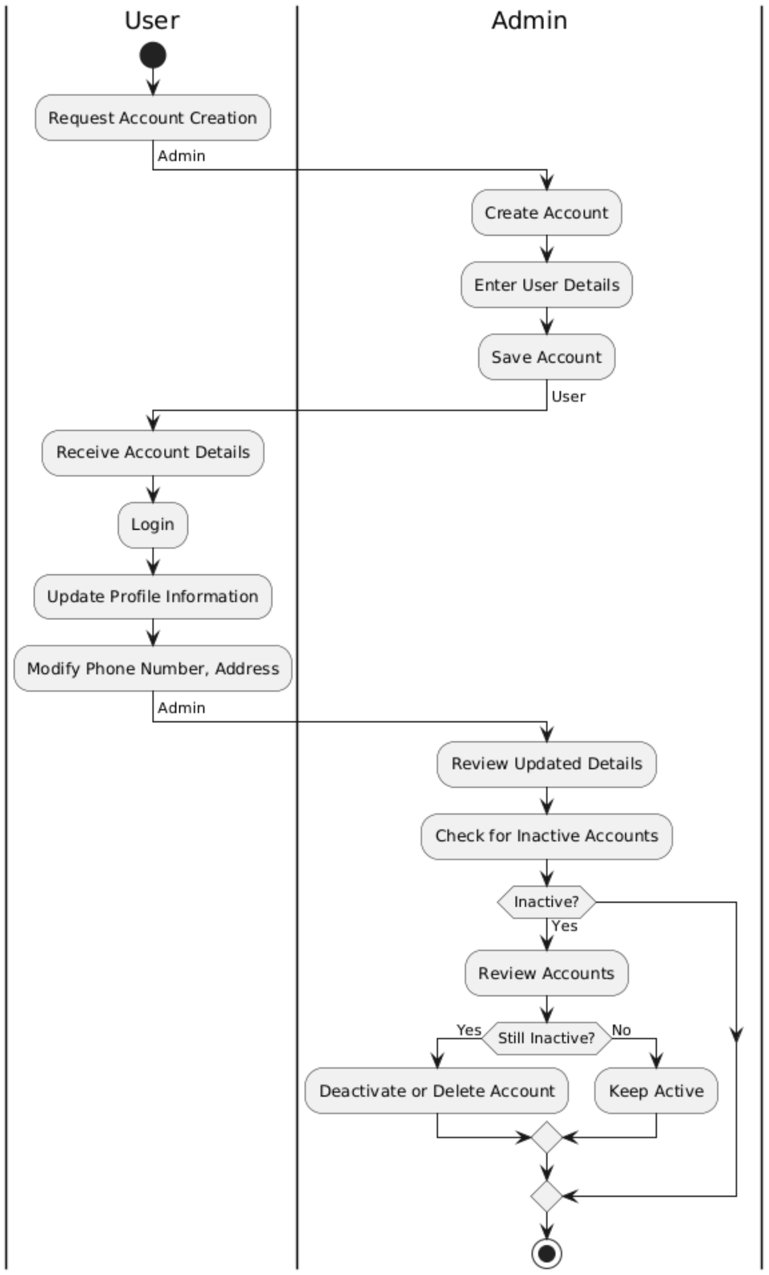


State Diagram:

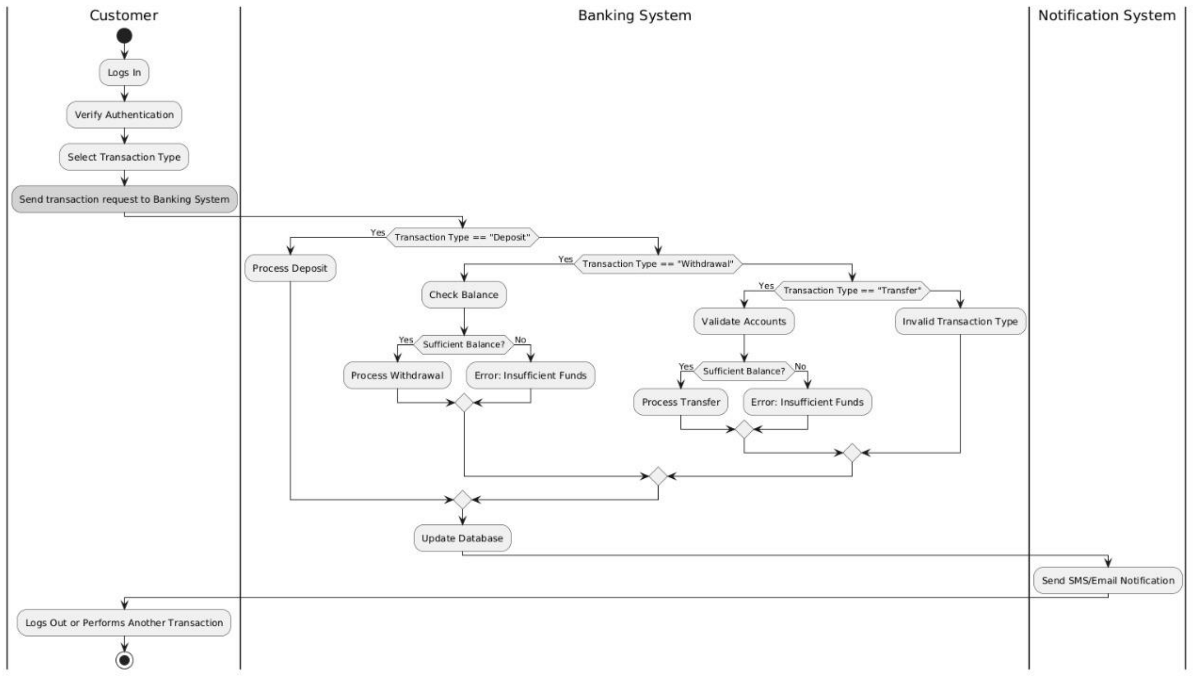


Activity Diagrams:

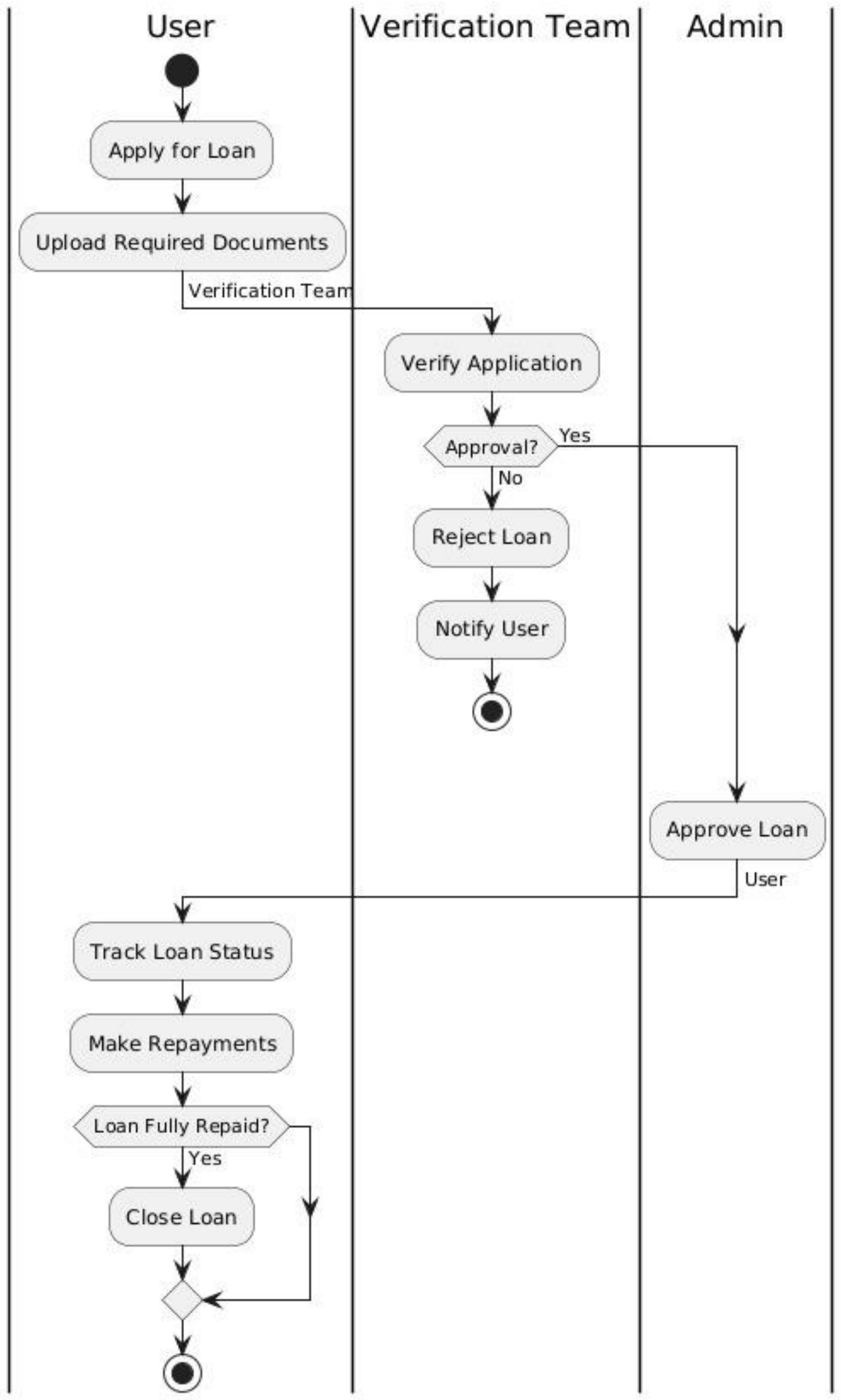
1. Major Usecase
   Account Management



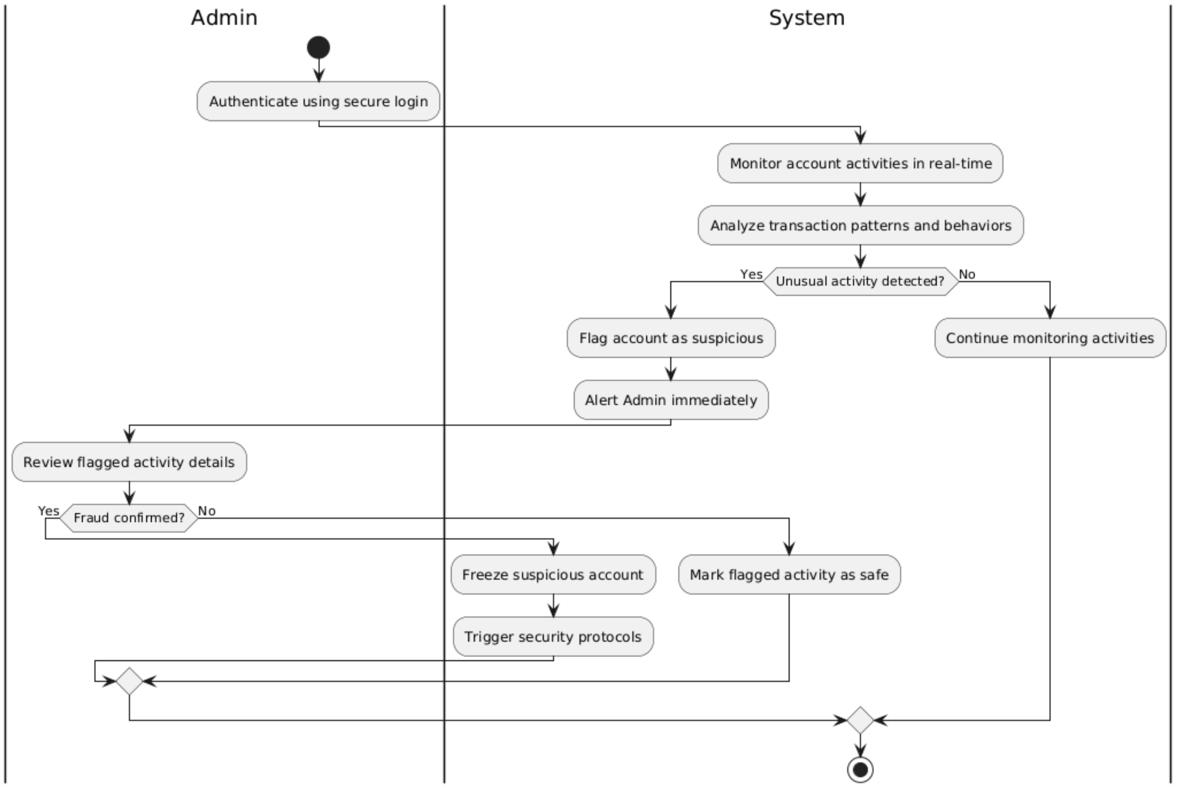
Transaction Management:



2. Minor Use case
   Loan Management



Security and Fraud detection


Admin
System
Authenticate using secure login
Monitor account activities in real-time
Analyze transaction patterns and behaviors
Unusual activity detected?
Yes
No
Flag account as suspicious
Continue monitoring activities
Alert Admin immediately
Review flagged activity details
Fraud confirmed?
Yes
No
Freeze suspicious account
Mark flagged activity as safe
Trigger security protocols

Architecture Patterns, Design Principles, and Design Patterns:

Architecture Patterns

Model – View – Controller Pattern (MVC)

1. **Model Layer**

   The Model represents the data and business logic of the application. In this project, it's implemented in the model package:

   Key characteristics of the Model layer

   - Represents the data structure
   - Contains business rules and logic
   - Handles data validation
   - Manages relationships between entities
   - Persists data to the database

2. **View Layer**

   In this Spring Boot + React application, the View layer is implemented in the frontend React components:

3. **Controller Layer**

   The Controller layer is implemented in the `controller` package

Benefits of MVC in this Project:

1. **Separation of Concerns:**
- Clear division between data, presentation, and control
- Each component has a specific responsibility
- Easier to maintain and modify

2. **Reusability:**
- Models can be reused across different views
- Controllers can handle multiple views
- Services can be used by different controllers

3. **Testability:**
- Each layer can be tested independently
- Clear interfaces between components
- Easy to mock dependencies

4. **Scalability:**
- Easy to add new features
- Simple to modify existing functionality

- Clear structure for expansion

5. **Maintainability:**

- Organized code structure
- Clear responsibilities
- Easy to debug

6. **Flexibility:**

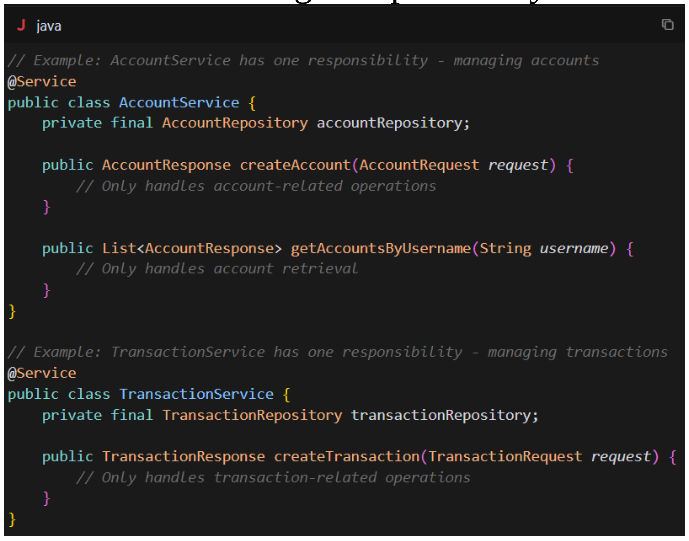
- Can change the view without affecting the model
- Can modify the controller without changing the view
- Can update the model without changing the controller

This MVC implementation provides a robust and maintainable architecture for the banking system, allowing for clear separation of concerns and easy expansion of functionality.

Design Principles

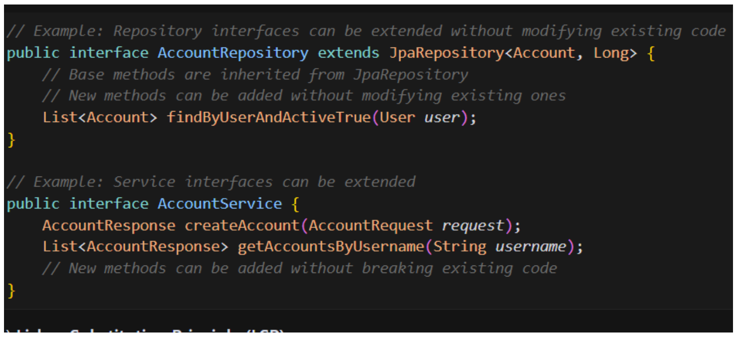
1.SOLID Principles

a) Single Responsibility Principle (SRP):
Each class has a single responsibility and reason to change.



```java
// Example: AccountService has one responsibility - managing accounts
@Service
public class AccountService {
    private final AccountRepository accountRepository;

    public AccountResponse createAccount(AccountRequest request) {
        // Only handles account-related operations
    }

    public List<AccountResponse> getAccountsByUsername(String username) {
        // Only handles account retrieval
    }
}

// Example: TransactionService has one responsibility - managing transactions
@Service
public class TransactionService {
    private final TransactionRepository transactionRepository;

    public TransactionResponse createTransaction(TransactionRequest request) {
        // Only handles transaction-related operations
    }
}
```

b) Open/Closed Principle (OCP):
Software entities should be open for extension but closed for modification.



```java
// Example: Repository interfaces can be extended without modifying existing code
public interface AccountRepository extends JpaRepository<Account, Long> {
    // Base methods are inherited from JpaRepository
    // New methods can be added without modifying existing ones
    List<Account> findByUserAndActiveTrue(User user);
}

// Example: Service interfaces can be extended
public interface AccountService {
    AccountResponse createAccount(AccountRequest request);
    List<AccountResponse> getAccountsByUsername(String username);
    // New methods can be added without breaking existing code
}
```

c) **Liskov Substitution Principle (LSP):**

Objects of a superclass should be replaceable with objects of its subclasses.

Java

d) **Interface Segregation Principle (ISP**): Clients should not be forced to depend on interfaces they don't use.

e) **Dependency Inversion Principle (DIP): High**-level modules should not depend on low-level modules. Both should depend on abstractions.

2.DRY (Don't Repeat Yourself) Principle:

Avoid code duplication by creating reusable components.

3. KISS (Keep It Simple, Stupid) Principle:

Keep the code simple and straightforward.

The **DRY (Don't Repeat Yourself)** and **KISS (Keep It Simple, Stupid)** principles are foundational concepts in software design aimed at improving code quality and maintainability. The DRY principle emphasizes avoiding code duplication by ensuring that a piece of logic is written only once and reused wherever needed. This reduces redundancy, makes the code easier to maintain, and minimizes the risk of inconsistencies during updates. On the other hand, the KISS principle encourages developers to keep their code as simple as possible, avoiding unnecessary complexity. By focusing on straightforward and clear implementations, KISS ensures that the code is easier to read, debug, and extend. Together, these principles promote clean, efficient, and scalable software development practices.

<u>Design Patterns</u>

1. Repository Pattern

- This pattern abstracts the data access layer
- Provides a collection-like interface for accessing domain objects
- Encapsulates all the logic needed to obtain data from a source
- Makes the application more maintainable by centralizing data access

2. Service Layer Pattern

- Separates business logic from presentation and data access layers

- Provides a clear API for the presentation layer
- Encapsulates complex business operations and transactions
- Acts as a facade for multiple repositories and business rules

3. Builder Pattern

- Provides a flexible solution to construct complex objects
- Separates the construction of an object from its representation
- Allows step-by-step construction of objects
- Makes the code more readable and maintainable

4. Factory Pattern

- Centralizes object creation logic
- Provides a way to create objects without specifying their exact classes
- Makes the code more maintainable and testable
- Allows for easy switching between different implementations

5. Singleton Pattern

- Ensures a class has only one instance
- Provides a global point of access to that instance
- Managed by Spring's dependency injection container
- Useful for services that need to maintain state or coordinate actions

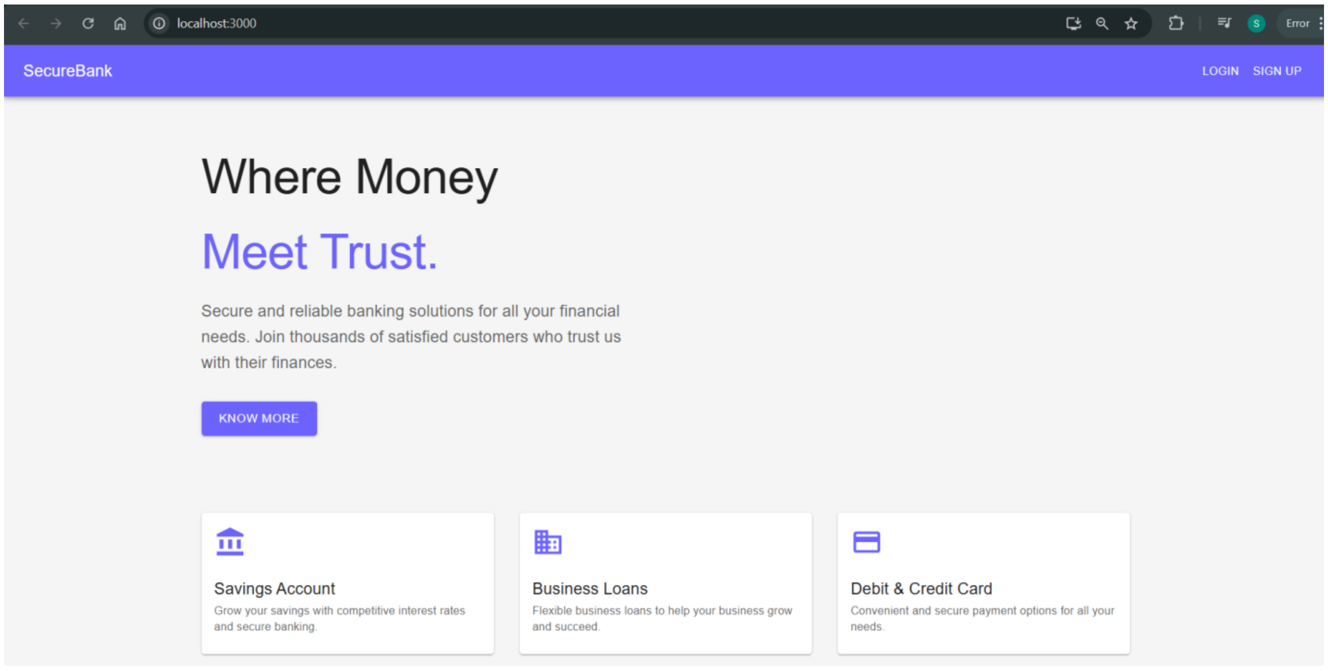
6. Strategy Pattern

- Defines a family of algorithms
- Encapsulates each algorithm
- Makes them interchangeable
- Lets the algorithm vary independently from clients that use it

Github link to the Codebase: https://github.com/shreyamp01/OOAD_Secure_Bank

Screenshots

UI:



localhost:3000

SecureBank LOGIN SIGN UP

Where Money

Meet Trust.

Secure and reliable banking solutions for all your financial needs. Join thousands of satisfied customers who trust us with their finances.

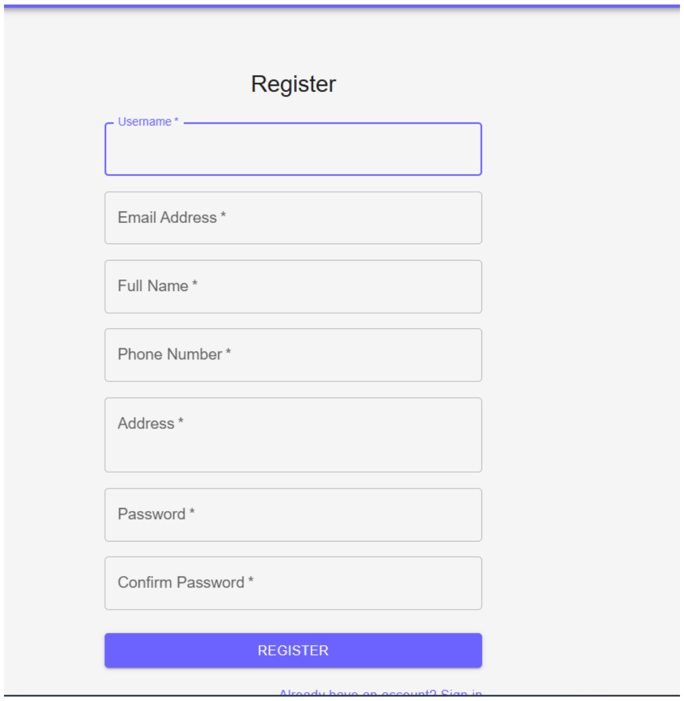
KNOW MORE

Savings Account
Grow your savings with competitive interest rates and secure banking.

Business Loans
Flexible business loans to help your business grow and succeed.

Debit & Credit Card
Convenient and secure payment options for all your needs.

User Signup/Register:



Register

Username *

Email Address *

Full Name *

Phone Number *

Address *

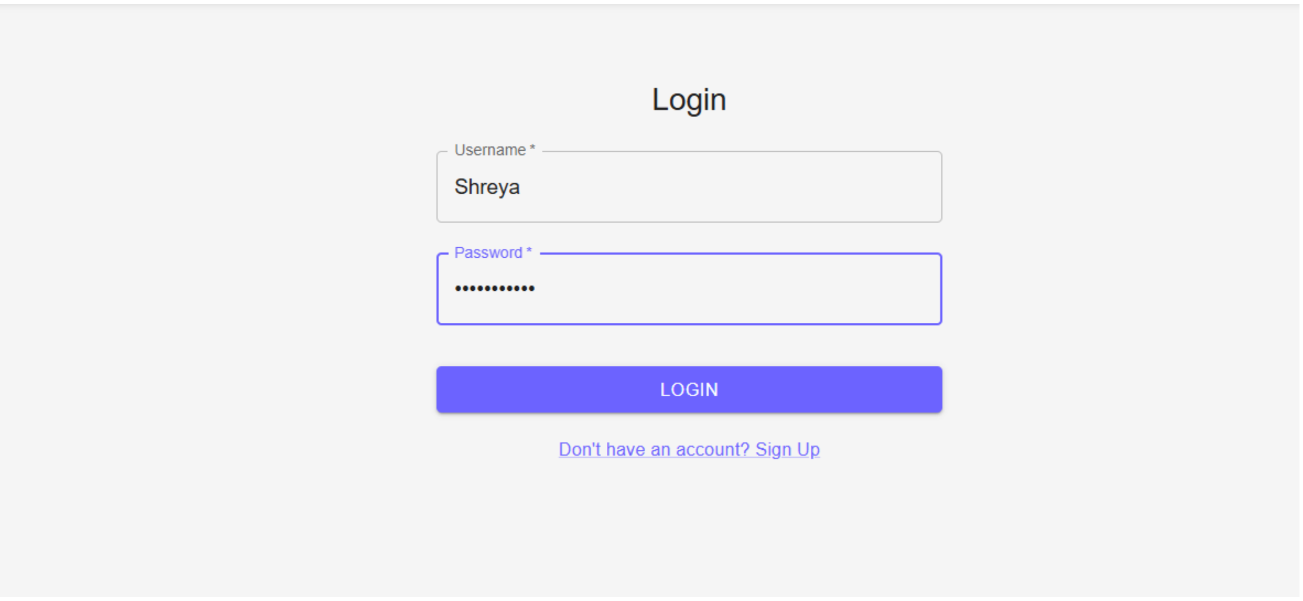
Password *

Confirm Password *

REGISTER

Already have an account? Sign in

User Login:


Login

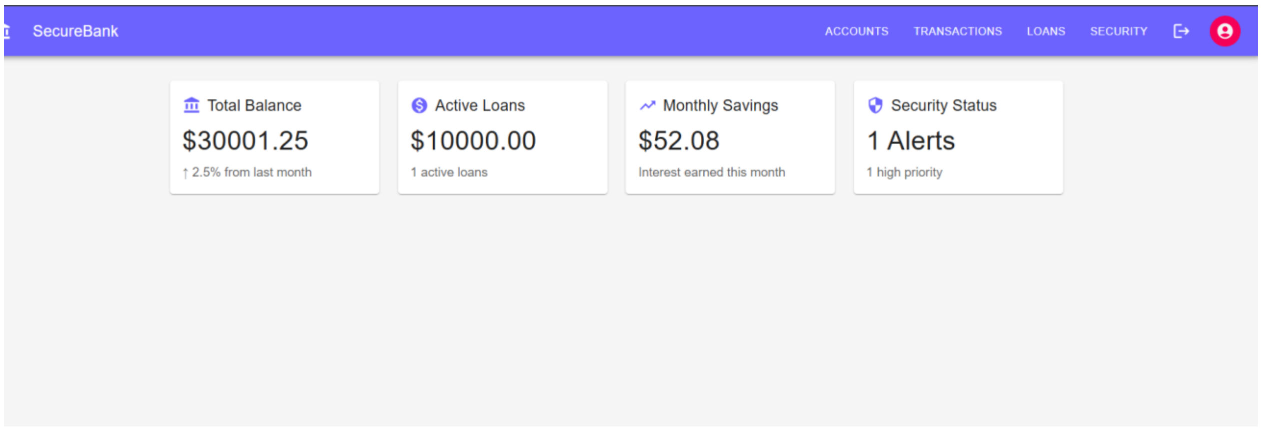
Username *
Shreya

Password *
•••••••••••

LOGIN

Don't have an account? Sign Up

Dashboard:



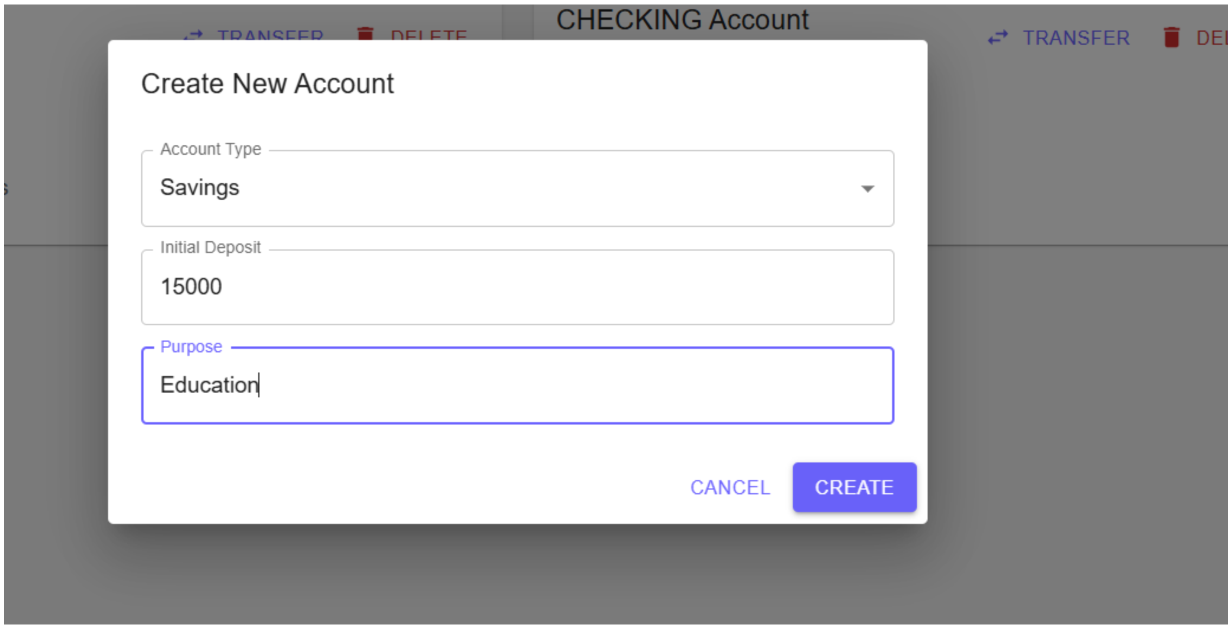
Total Balance
$30001.25
↑ 2.5% from last month

Active Loans
$10000.00
1 active loans

Monthly Savings
$52.08
Interest earned this month

Security Status
1 Alerts
1 high priority

Account creation with initial deposit:



TRANSFER DELETE

CHECKING Account

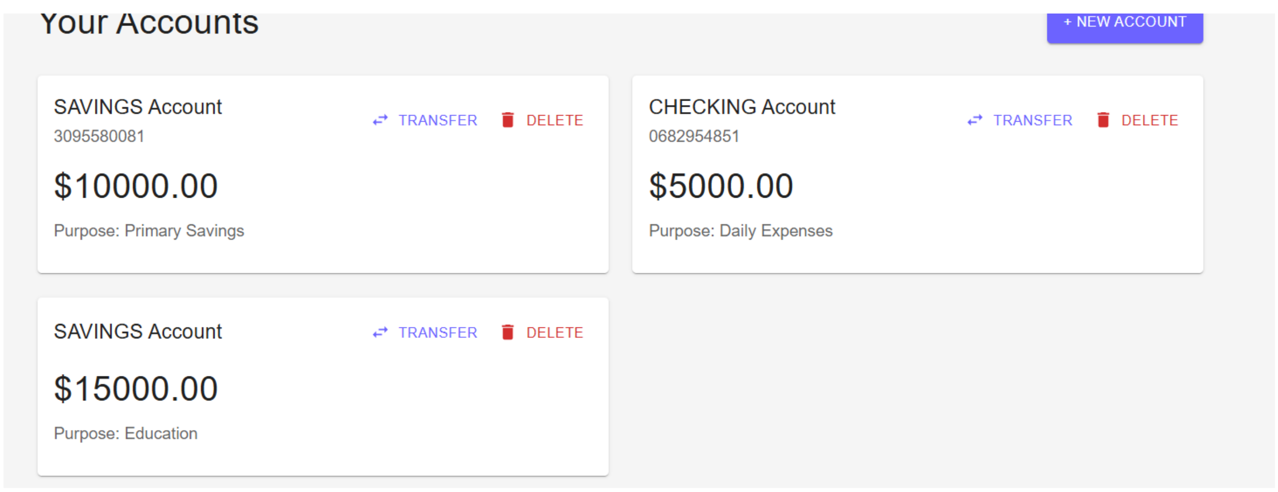
TRANSFER DEL

Create New Account

Account Type: Savings

Initial Deposit: 15000

Purpose: Education



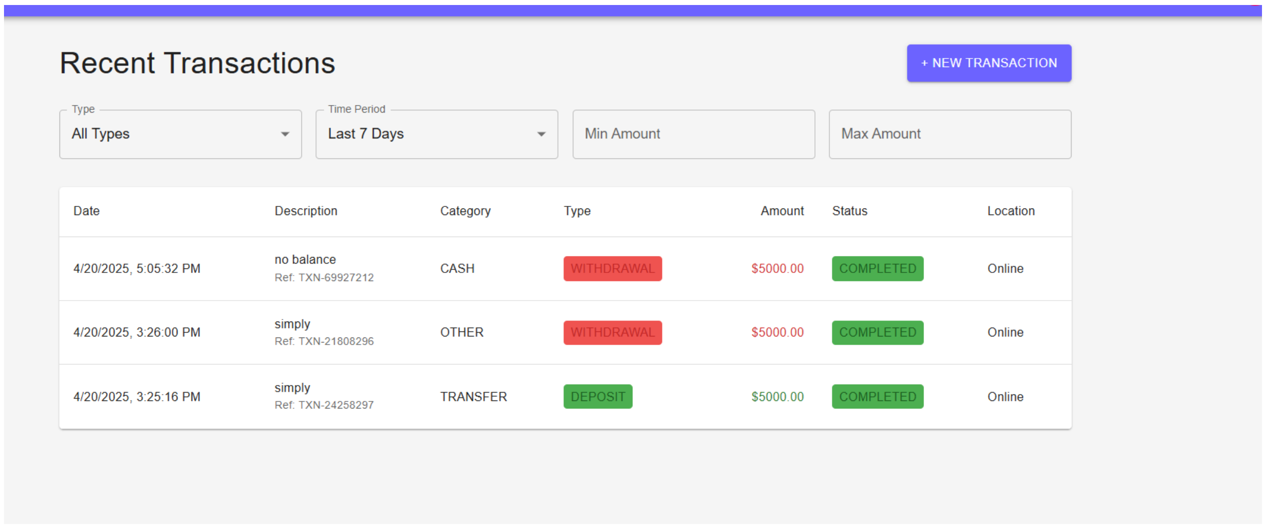
Your Accounts

SAVINGS Account
3095580081
$10000.00
Purpose: Primary Savings

CHECKING Account
0682954851
$5000.00
Purpose: Daily Expenses

SAVINGS Account
$15000.00
Purpose: Education

Transaction:



Recent Transactions

Type: All Types

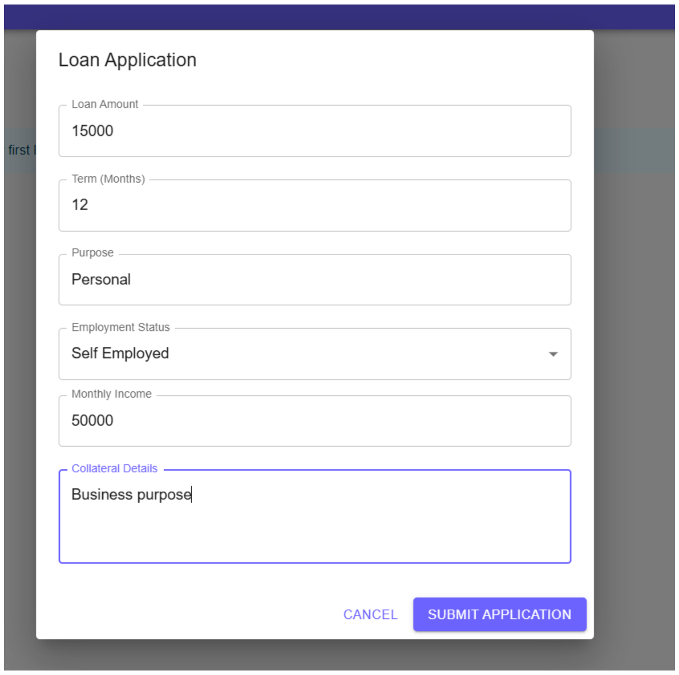
Time Period: Last 7 Days

Min Amount

Max Amount

Date	Description	Category	Type	Amount	Status	Location
4/20/2025, 5:05:32 PM	no balance Ref: TXN-69927212	CASH	WITHDRAWAL	$5000.00	COMPLETED	Online
4/20/2025, 3:26:00 PM	simply Ref: TXN-21808296	OTHER	WITHDRAWAL	$5000.00	COMPLETED	Online
4/20/2025, 3:25:16 PM	simply Ref: TXN-24258297	TRANSFER	DEPOSIT	$5000.00	COMPLETED	Online

Loan Application:


Loan Application

Loan Amount
15000

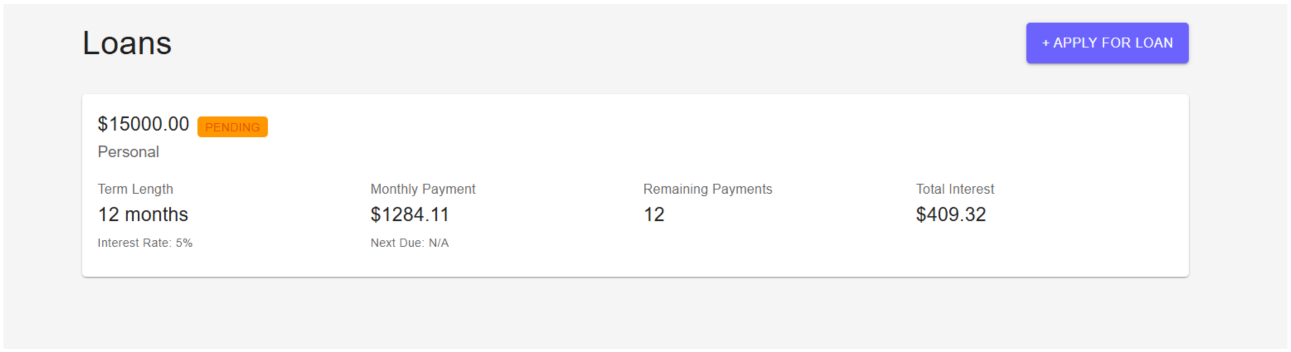
Term (Months)
12

Purpose
Personal

Employment Status
Self Employed

Monthly Income
50000

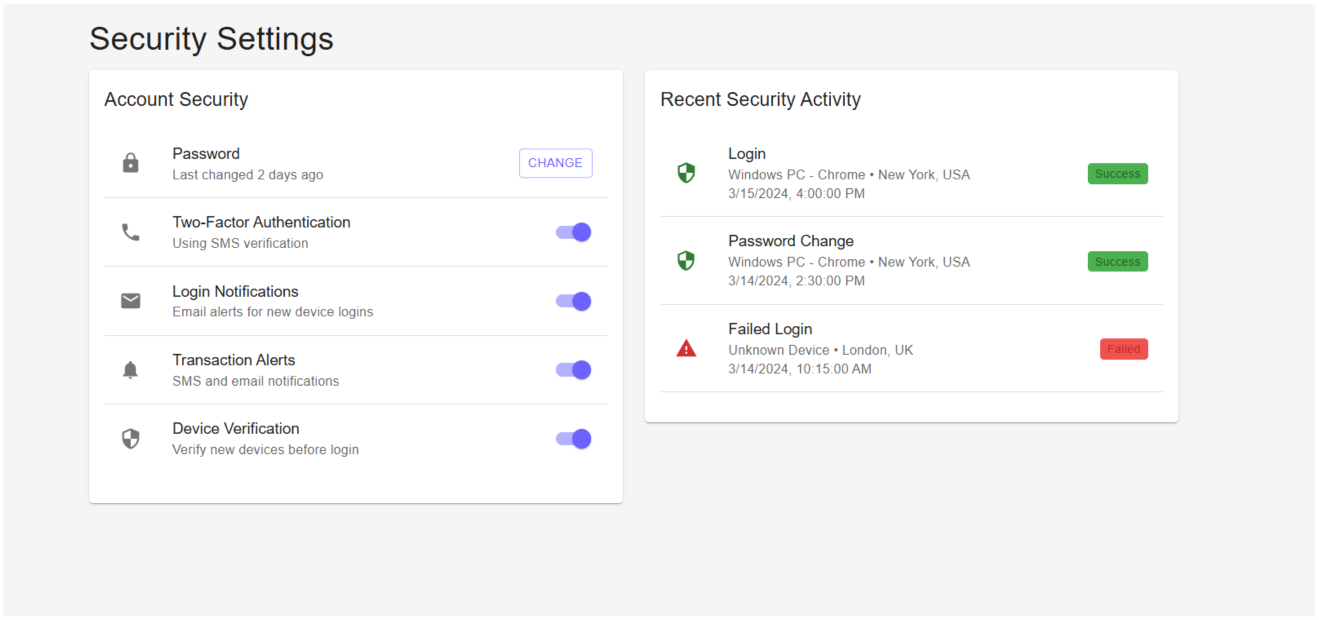
Collateral Details
Business purpose


Loans

$15000.00 PENDING
Personal

Term Length	Monthly Payment	Remaining Payments	Total Interest
12 months	$1284.11	12	$409.32
Interest Rate: 5%	Next Due: N/A		

Security and Fraud detection:


Security Settings

Account Security

Password
Last changed 2 days ago

Two-Factor Authentication
Using SMS verification

Login Notifications
Email alerts for new device logins

Transaction Alerts
SMS and email notifications

Device Verification
Verify new devices before login

Recent Security Activity

Login
Windows PC - Chrome • New York, USA
3/15/2024, 4:00:00 PM
Success

Password Change
Windows PC - Chrome • New York, USA
3/14/2024, 2:30:00 PM
Success

Failed Login
Unknown Device • London, UK
3/14/2024, 10:15:00 AM
Failed

Individual contributions of the team members:

Name	**Module worked on**
Shreya MP	Account creation
Sinchana H	Loan management
Spurti Bhat	Security
Spoorthi J	Transaction management