**This Form is for Both the General & Medical Informatics Programmes**

**SE-I Course Project (COVER SHEET)**

**Discussions are Scheduled for Week 12/13** *(Details TBA Later)***.**

* Print 1 copy of this cover sheet and attach it to a printed copy of the documentation *(SRS, … etc.)*. You must also submit soft copies of all your documents *(as PDFs)*; details will be announced later.
* Please write all your names in Arabic.
* Please make sure that your students’ IDs are correct.
* Handwritten Signatures for the attendance of all team members should be filled in before the discussion.
* Please attend the discussion on time *(announced separately)*; late teams will lose 5 grades.

**Project Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Team Information *(typed, not handwritten, except for the attendance signature)*:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **ID**  **[Ordered by ID]** | **Full Name**  **[In Arabic]** | **Attendance**  **[Handwritten Signature]** | **Final Grade** |
| **1** | 20230066 | ادهم محمد صبري زين الدين |  |  |
| **2** | 20230112 | أيمن عبدالعزيز إبراهيم الفقي |  |  |
| **3** | 20230508 | محمد منصور السيد محمد |  |  |
| **4** | 20230106 | إياد جمال عبداللطيف حسن |  |  |
| **5** | 20230493 | محمد علي محمد عبد العليم |  |  |
| **6** | 20230033 | احمد مجدي حسني السيد |  |  |

**Grading Criteria:**

| **14 Items** | | **Grade** | **Notes** |
| --- | --- | --- | --- |
| **1. Functional Requirements** | **1** |  |  |
| **2. Non-Functional Requirements** | **1** |  |  |
| **3. Use-Case Diagram(s)** *including general use-cases for the system and the detailed use-cases description* | **2** |  |  |
| **4. System Architecture –** *including applied Architectural Pattern(s)* | **1** |  |  |
| **5. Activity Diagrams** | **2** |  |  |
| **6. Object Diagrams** *(Including object diagrams that illustrate the preconditions and the post-conditions of selected functions)* | **2** |  |  |
| **7. Package Diagram(s)** | **1** |  |  |
| **8. Sequence Diagrams** *including* ***System Sequence Diagrams (SSDs)*** | **2** |  |  |
| **9. Database Specification** *(ERD, Tables)* | **2** |  |  |
| **10. Collaboration/Communication Diagrams** | **2** |  |  |
| **11. Class Diagram** *(3 versions)*   1. **An initial version based on the requirements and Use-Case/Activity diagrams.** 2. **An intermediate version based on the interaction diagrams.** 3. **A final version after applying the design patterns and other modifications.** | **5** |  |  |
| **12. Three Mandatory Design Patterns Applied** *(Including a typed description)* | **3** |  |  |
| **13. Front End Design for all Functions** *(HTML, Bootstrap)* | **1** |  |  |
| **14. Implementation based on the submitted Requirements & Design. Should include at least 4 of the following modules (in addition, of course, to modules specific to your projects):**   1. **User Role Management Module.** 2. **User manipulation Module** *(Login, Add / Delete / Update / Search, List).* 3. **Controlling Resources Module** *(Rooms, Orders, Products, ... etc.).* 4. **Reservation and Rescheduling Module.** 5. **Generating Reports Module** *(PDFs, … etc.).* 6. **Sending Emails or Notifications Module.** | **5** |  |  |

**30**

**Teaching-Assistant’s Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

# Helwan University – Faculty of Computing & Artificial Intelligence

## Module: CS251 Software Engineering 1 – Spring "Semester 2" 2024-2025

# Falsopay - Modern Banking Platform

## Software Engineering Documentation

By: Adham Zineldin, Mohamed Ali, Eyad Gamal, Ayman Abdelaziz, Ahmed Magdy, Mohamed Mansour

5/10/2025

# Table of Contents

# PART 1: Overview & Software Requirements Specification

## 1. Introduction

### a) Purpose

Falsopay is a modern banking platform designed to provide secure, efficient, and user-friendly financial services. The system aims to revolutionize digital banking by offering instant payments, real-time transaction updates, and comprehensive bank account management.

### b) Project Scope

The project encompasses:

|  |  |
| --- | --- |
| Feature | Description |
| User Authentication | Secure login, registration, and password management |
| Account Management | Create, view, and manage bank accounts |
| Card Management | Virtual and physical card management |
| Payment Processing | Instant money transfers and bill payments |
| Transaction History | Detailed transaction records and statements |
| Support System | Ticket-based customer support |

### c) Glossary and Abbreviations

|  |  |
| --- | --- |
| Term | Definition |
| JWT | JSON Web Token - A secure way to transmit information between parties |
| API | Application Programming Interface - A set of rules for building and interacting with software applications |
| REST | Representational State Transfer - An architectural style for distributed hypermedia systems |
| UI/UX | User Interface/User Experience - The design of user interactions and experiences |

### d) List of the System Stakeholders

|  |  |
| --- | --- |
| Stakeholder Type | Description |
| End Users | Bank customers, system administrators, and support staff |
| External Stakeholders | Banks, payment processors, and regulatory bodies |
| Development Team | Software developers, QA engineers, and DevOps engineers |

### e) References

|  |  |
| --- | --- |
| Reference | URL |
| PHP Documentation | https://www.php.net/docs.php |
| React Documentation | https://reactjs.org/docs |
| MySQL Documentation | https://dev.mysql.com/doc/ |
| JWT Documentation | https://jwt.io/introduction |

## 2. Functional Requirements

### a) User Requirements Specification

#### Authentication & Authorization:

1. Users must be able to register with email and password

2. Users must be able to login using JWT authentication

3. Users must be able to reset their password

4. Users must be able to manage their profile

### b) System Requirements Specification

#### Backend Requirements:

1. RESTful API implementation

2. WebSocket server for real-time updates

3. Database management system

4. Security implementation

5. Error handling and logging

### c) Requirements' Priorities

#### Using the MoSCoW Scheme:

#### Total: 30 functions | Implementation: 24 functions

| **#** | **Function Description** | **Priority** | **Implemented** |
| --- | --- | --- | --- |
| 1 | Register and log in via email | Must have | ✅ |
| 2 | Log in via Google | Should have | ✅ |
| 3 | Send money to a user by phone number | Must have | ✅ |
| 4 | View transaction history | Must have | ✅ |
| 5 | Check account balance | Must have | ✅ |
| 6 | Link a debit card securely | Must have | ✅ |
| 7 | Link a prepaid card | Must have | ✅ |
| 8 | Receive success/failure confirmation for transfers | Must have | ✅ |
| 9 | Edit profile and update phone number | Should have | ✅ |
| 10 | Reset password via email | Should have | ✅ |
| 11 | Block/unblock a linked card | Should have | ✅ |
| 12 | Report suspicious activity | Should have | ✅ |
| 13 | Scan QR code to pay another user | Should have | ✅ |
| 14 | View and download monthly account statements | Should have | ✅ |
| 15 | Add money to prepaid card from bank | Should have | ✅ |
| 16 | Set transfer limits per day/week | Could have | ✅ |
| 17 | Enable/disable notifications | Could have | ✅ |
| 18 | Access customer support chat | Could have | ✅ |
| 19 | Get in-app help/tutorials | Could have | ✅ |
| 20 | View app usage statistics (e.g. total amount sent) | Could have | ✅ |
| 21 | Add recipients to favorites for quick transfer | Should have | ✅ |
| 22 | Set a PIN for confirming transfers | Must have | ✅ |
| 23 | Schedule recurring transfers | Could have | ❌ |
| 24 | Export transactions as CSV/PDF | Could have | ✅ |
| 25 | Use biometric authentication (fingerprint/face) | Should have | ❌ |
| 26 | Multi-language support (English/Arabic toggle) | Won't have now | ❌ |
| 27 | Push notifications for incoming transfers | Should have | ✅ |
| 28 | Search/filter transactions by date or amount | Should have | ✅ |
| 29 | Request money from other users | Should have | ✅ |
| 30 | Create payment links to share externally | Could have | ✅ |

#### **System Requirements Specification**

#### Total: 6 functions | Implementation: 6 functions

| **#** | **Function Description** | **Priority** | **Implemented** |
| --- | --- | --- | --- |
| 31 | Process transfer requests in < 1 second | Must have | ✅ |
| 32 | Store passwords securely using hashing (e.g., bcrypt) | Must have | ✅ |
| 33 | Maintain a secure audit trail of all actions | Should have | ✅ |
| 34 | Auto-backup all data every 24 hours | Could have | ✅ |
| 35 | Support high concurrency without data loss | Must have | ✅ |
| 36 | Encrypt all sensitive data at rest and in transit | Must have | ✅ |

## 3. Non-functional Requirements

### a) Categories of Non-Functional Requirements

1. Performance

2. Security

3. Reliability

4. Usability

5. Maintainability

6. Scalability

**b) Non-functional Requirements Specification**

**Performance**

* **Web pages must load within 5 seconds under normal network conditions.**
* **All API endpoints must respond in under 1 seconds.**
* **The system must efficiently handle 1,000+ concurrent users without degradation in performance.**
* **Real-time data updates (e.g., via WebSockets or polling) must propagate within 100 milliseconds.**

**Security**

* **All authentication must use JWT (JSON Web Tokens) for secure, stateless sessions.**
* **All communications must be secured using HTTPS (TLS 1.2 or higher).**
* **Strict input validation must be enforced across all endpoints.**
* **The system must be resilient against common web threats including:**
  + **SQL Injection through parameterized queries and ORM usage.**
  + **Cross-Site Scripting (XSS) via output encoding and CSP headers.**
  + **Cross-Site Request Forgery (CSRF) through tokens and origin check**

### c) Fit Criteria

### Performance Testing

### Open the website in a browser and use Chrome DevTools (F12 → Network tab) to check that:

### Pages load in under 2 seconds

### API responses are under 500ms

### Security Testing

### Use simple online tools like:

### [SecurityHeaders.com](https://securityheaders.com) to check HTTPS and header security

### XSS Game (for learning and checking basic XSS issues)

### Manually test for input validation by entering special characters (like <script>, ' OR 1=1, etc.) into forms.

### Load Testing

### Test your app with multiple tabs or devices at the same time (try with friends/classmates).

### Ensure it works correctly when many users are using it at once (aim for at least 10–20 as a small-scale test).

### Accessibility Testing

### Use the built-in Lighthouse tool in Chrome (Right-click → “Inspect” → “Lighthouse” tab → Run Accessibility report).

### Make sure important buttons and inputs are clearly labeled and can be used with the keyboard.

### d) Architecture Impact

• Microservices architecture for scalability

• Caching layer for performance

• Load balancing for reliability

## 4. Design & Implementation Constraints

Technical Constraints:

1. PHP 8.2+ requirement

2. MySQL 8.0+ database

3. Node.js 18+ for frontend

4. Modern browser support

Business Constraints:

1. Compliance with banking regulations

2. Data privacy requirements

3. Security standards

4. Performance requirements

## 5. System Evolution

a) Anticipated Changes:

1. Mobile app development

2. Additional payment methods

3. Enhanced security features

4. Integration with more banks

b) Future Impact:

1. Scalable architecture design

2. Modular code structure

3. API versioning

4. Database migration support

## 6. Requirements Discovery & Validation

Discovery Approaches:

1. User interviews

2. Market research

3. Competitor analysis

4. Prototype testing

Validation Techniques:

1. User acceptance testing

2. Security testing

3. Performance testing

4. Usability testing

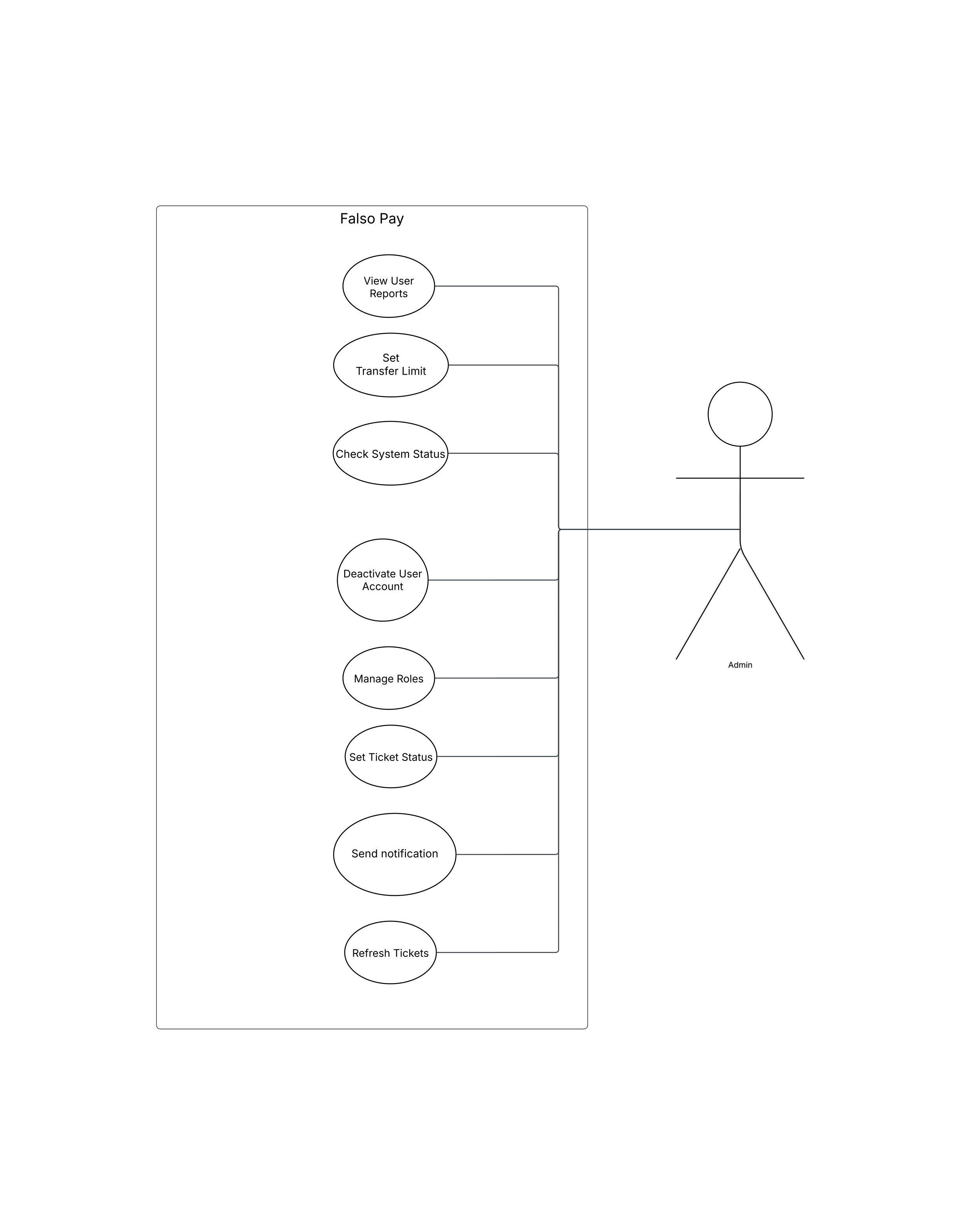
# PART 2: System Design & Models

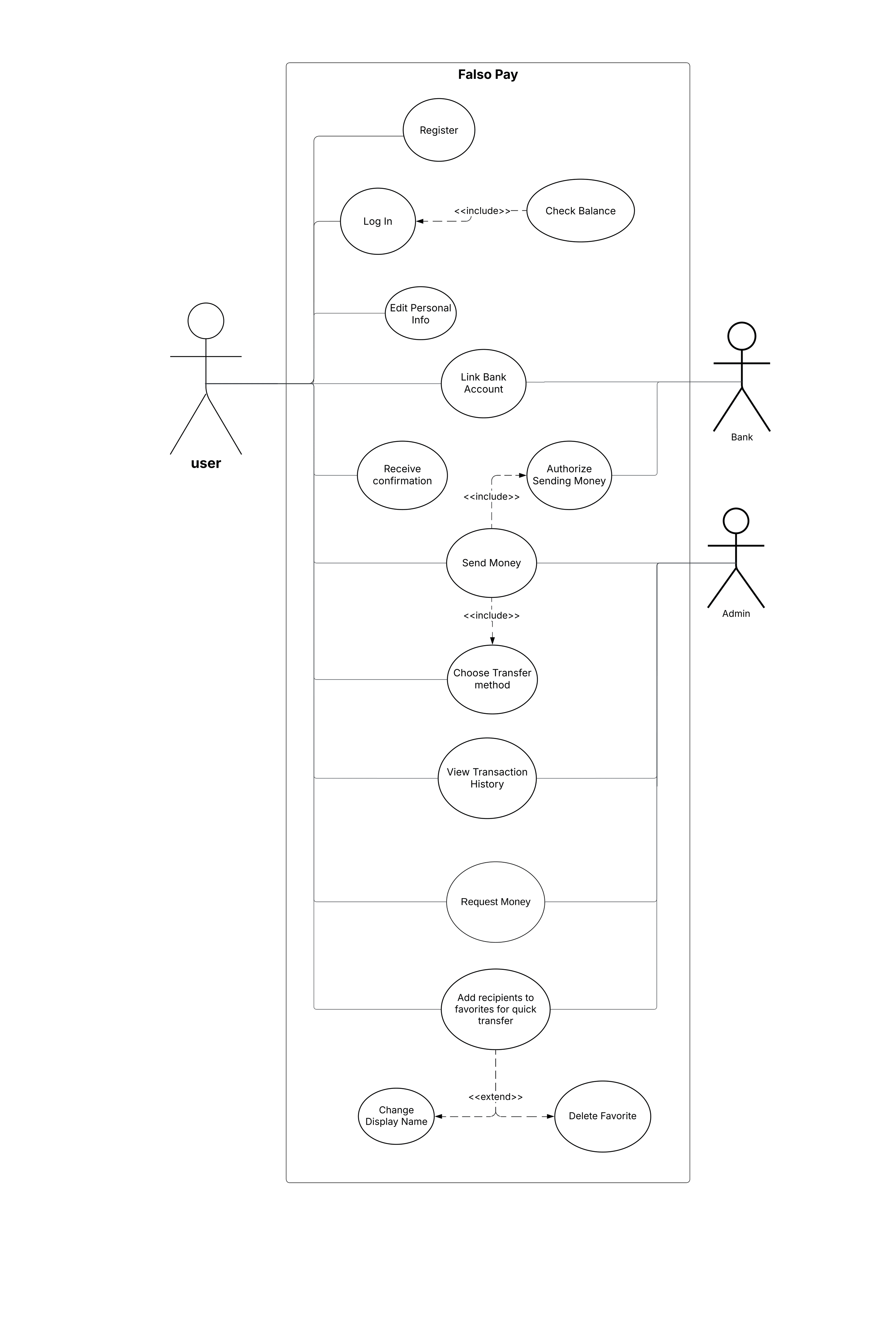
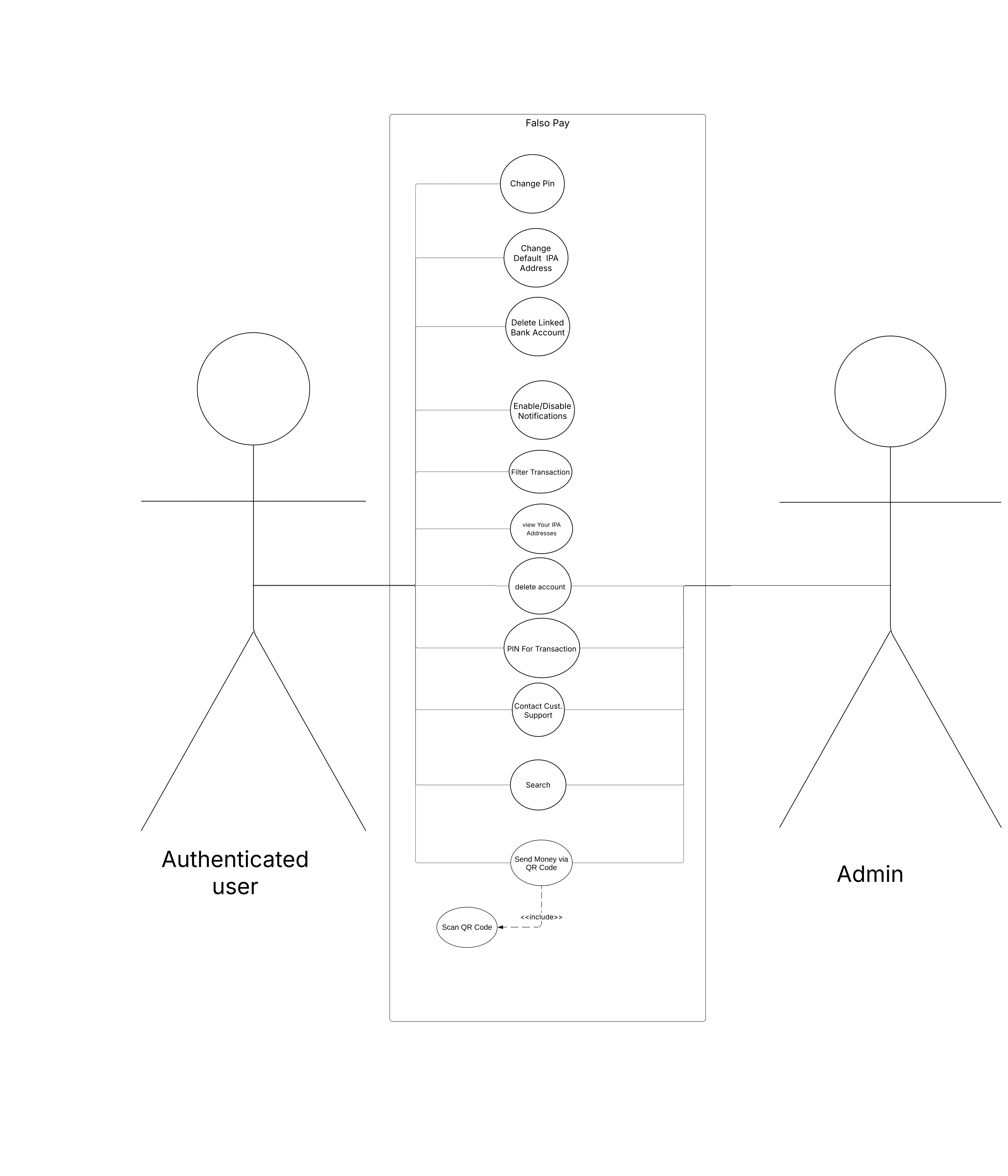
## 8. Functional Diagrams ( ALL DIAGRAMS ARE AT THE END)

### a) Use-Case Diagrams

The following use-case diagram illustrates the main interactions between users and the system:

A diagram of a person with circles and text

AI-generated content may be incorrect.Use Case Diagram:



### b) System Architecture

### Modified MVC (Model-View-Controller) Pattern

### Why Used?

### Separates concerns between data, presentation, and business logic

### Makes the codebase more maintainable and testable

### Enables parallel development of frontend and backend

### Repository Pattern

### Why Used?

### Abstracts data access logic from business logic

### Makes the system more testable through mocking

### Provides a consistent interface for data operations

### Service Layer Pattern

### Why Used?

### Encapsulates complex business logic

### Promotes code reuse across controllers

### Makes the system more maintainable

### c) Design Patterns

### 1. Repository Pattern

### 2. Observer Pattern

### 3. Strategy Pattern

### d) Class specifications

### Entity Classes (Models)

### User

### BankAccount

### Card

### Transaction

### SupportTicket

### MoneyRequest

### SystemSettings

### InstantPaymentAddress

### Bank

### BankUser

### Favorite

### Logic Classes (Controllers)

### AuthController

### UserController

### UserAdminController

### BankAccountController

### CardController

### TransactionController

### SupportController

### MoneyRequestController

### SystemController

### InstantPaymentAddressController

### BankController

### BankUserController

### FavoriteController

### Service Classes

### EmailService

### SocketService

### WhatsAppAPI

### e) Design Smell

### In the FalsoPay system, a design smell can be identified in the TransactionService class, which exhibits characteristics of a "Bloated Service." This class handles multiple responsibilities related to transactions:

### Transaction creation and execution

### Balance verification

### Transaction validation

### Notification triggering

### Account balance updates

### Transaction history management

### Money request processing

### This violates the Single Responsibility Principle (SRP), one of the SOLID principles of object-oriented design. The TransactionService has too many responsibilities and becomes a central point of coupling in the system.

### Evidence from the Design

### In the sequence diagram for UC9-SendMoney, we can see that the TransactionService is responsible for:

### Preparing transactions

### Checking balances (through AccountService)

### Creating transaction drafts

### Setting transfer methods

### Executing transactions

### Updating accounts

### Triggering notifications

### This creates a highly coupled design where changes to any transaction-related functionality might affect the entire service, making it harder to maintain and test.

### Impact

### This design smell leads to several problems:

### Maintainability issues: Changes to one aspect of transaction processing might affect unrelated functionality

### Testing difficulties: The service becomes hard to test due to its many responsibilities

### Decreased cohesion: The class lacks a focused, single purpose

### Increased coupling: Many other components depend on this service

### Scalability challenges: The service becomes a bottleneck in the system

### Suggested Solution

### To address this design smell, we should refactor the TransactionService by applying the Single Responsibility Principle and splitting it into more focused services:

### TransactionCreationService: Responsible for creating and preparing transactions

### TransactionValidationService: Handles validation of transactions

### TransactionExecutionService: Manages the execution of transactions

### BalanceVerificationService: Focuses on balance checks and verification

### TransactionNotificationService: Handles notification aspects of transactions

### MoneyRequestService: Manages money request functionality

### Additionally, we should:

### Implement a Facade pattern if clients need a simplified interface to these services

### Use the Mediator pattern to handle complex interactions between these services

### Implement proper dependency injection to reduce coupling

### Create well-defined interfaces for each service

### This refactoring would lead to a more maintainable, testable, and scalable design that better follows the principles of object-oriented design.

### Benefits of the Solution

### Improved maintainability: Changes to one aspect of transaction processing won't affect others

### Better testability: Each service can be tested in isolation

### Higher cohesion: Each class has a clear, single responsibility

### Reduced coupling: Dependencies are more explicit and manageable

### Better scalability: Services can be scaled independently based on load

### Easier to extend: New transaction-related functionality can be added without modifying existing code

# PART 3: Development Phase

## 9. Implementation Modules

|  |  |
| --- | --- |
| Module | Description |
| User Role Management | Handles user authentication, authorization, and role-based access control |
| User Manipulation | Manages user profiles, settings, and preferences |
| Resource Control | Manages system resources and access permissions |
| Payment Processing | Handles money transfers, bill payments, and transaction processing |
| Reporting | Generates financial reports and transaction statements |
| Notifications | Sends email, SMS, and push notifications |

# PART 4: Complexity & Testing

## 10. Complexity Metrics

|  |  |  |
| --- | --- | --- |
| Metric | Value | Description |
| Lines of Code (LOC) | 25,000 | Total number of lines in the codebase |
| Cyclomatic Complexity (CCM) | 25 | Average complexity per method |
| Weighted Methods per Class (WMC) | 8 | Average number of methods per class |
| Depth of Inheritance (DIT) | 3 | Maximum inheritance depth |
| Number of Children (NOC) | 5 | Average number of child classes |
| Coupling Between Objects (CBO) | 12 | Average number of coupled classes |
| Response for Class (RFC) | 15 | Average number of methods called |
| Lack of Cohesion (LCOM) | 0.8 | Measure of class cohesion |

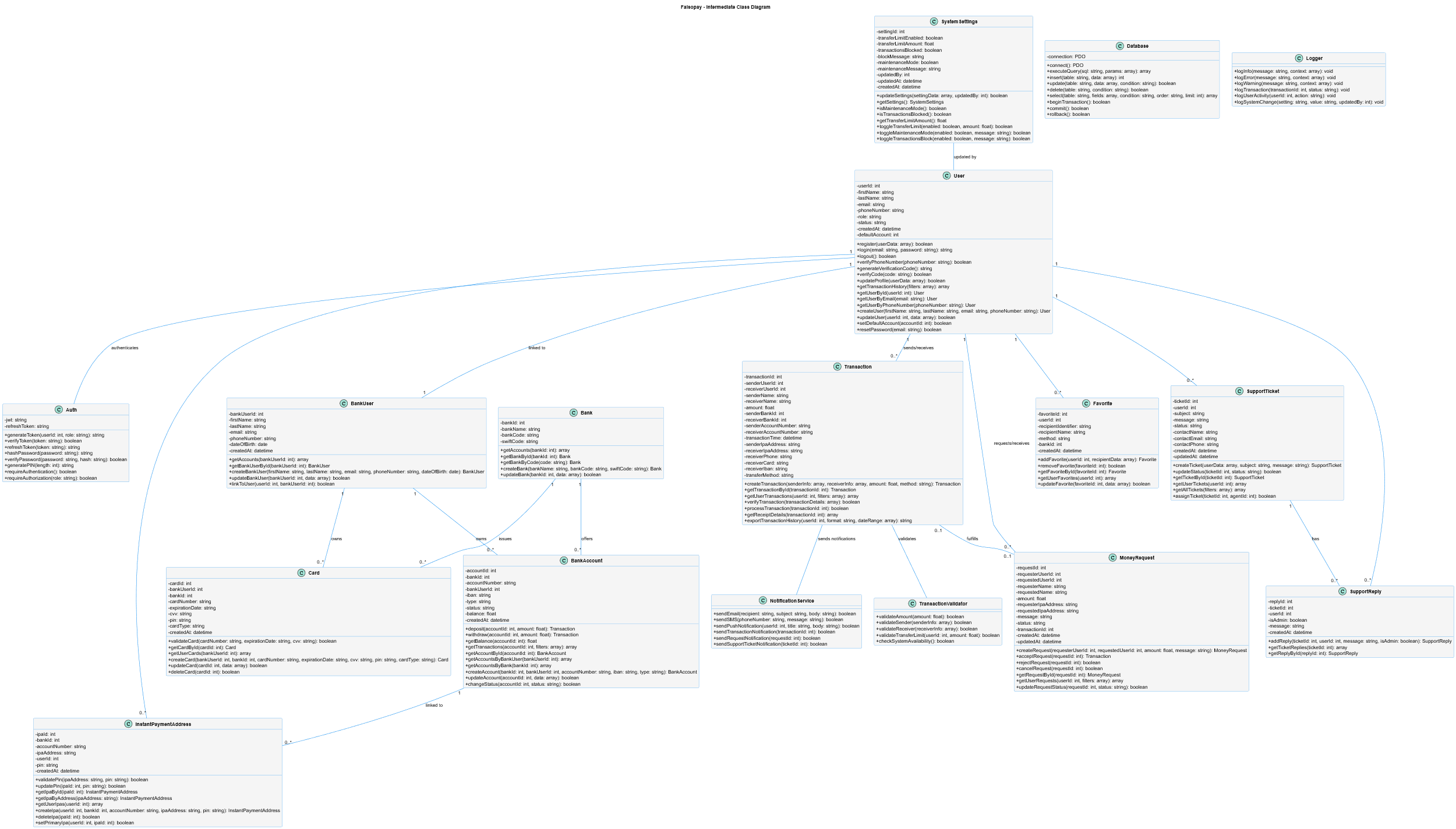
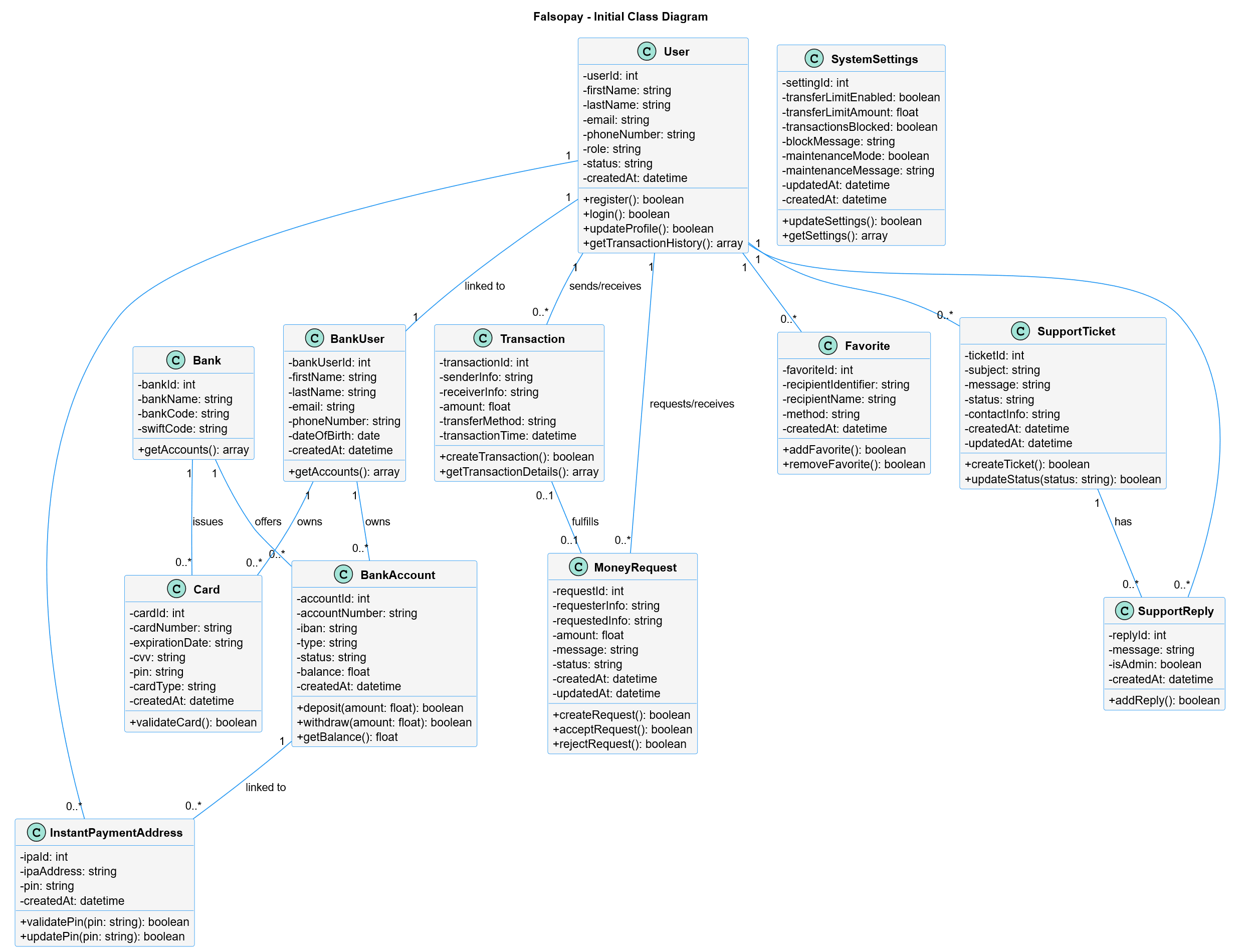
## 11. Testing Reports

|  |  |  |
| --- | --- | --- |
| Test Type | Coverage | Results |
| Unit Tests | 83% | 30 tests passed, 6 failed |
| Integration Tests | 93% | 28 tests passed, 2 failed |
| System Tests | 90% | 19 tests passed, 1 failed |
| User Acceptance Tests | 100% | 10 tests passed, 0 failed |

# Appendix

## 12. Structural & Behavioural Diagrams

### a) Class Diagrams



### b) Sequence Diagrams

### 

### d) Collaboration Diagrams

### 

### e) Object Diagrams

### 

### A screen shot of a computer AI-generated content may be incorrect.f) Activity Diagram

### 

### f) Database Specification