# 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

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:

Must Have:
• User authentication
• Bank account management
• Basic payment processing
• Security features
Should Have:
• Real-time updates
• Support ticket system
• Card management
• Advanced payment features
Could Have:
• Multiple language support
• Advanced analytics
• Mobile app
• Biometric authentication
Won't Have:
• Cryptocurrency support
• Investment features
• Insurance products
• Loan management

## 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:
• Page load time < 2 seconds
• API response time < 500ms
• Support for 1000+ concurrent users
• Real-time updates < 100ms
Security:
• JWT-based authentication
• HTTPS enforcement
• Input validation
• SQL injection prevention
• XSS protection
• CSRF protection

### c) Fit Criteria

1. Performance testing using Apache JMeter
2. Security testing using OWASP ZAP
3. Load testing with 1000+ concurrent users
4. Accessibility testing using WAVE

### d) Architecture Impact

• Microservices architecture for scalability
• Caching layer for performance
• Load balancing for reliability
• CDN for global access

## 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

### a) Use-Case Diagrams

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

Use Case Diagram:

@startuml Falsopay Use Case Diagram
skinparam actorStyle awesome
skinparam packageStyle rectangle
skinparam handwritten false
skinparam shadowing false
skinparam defaultFontName Arial
skinparam defaultFontSize 12
skinparam roundcorner 20
skinparam ArrowColor #2C3E50
skinparam ActorBorderColor #2C3E50
skinparam ActorBackgroundColor #ECF0F1
skinparam UsecaseBorderColor #2C3E50
skinparam UsecaseBackgroundColor #ECF0F1
title Falsopay - Use Case Diagram
actor "Customer" as customer
actor "Admin" as admin
actor "Bank" as bank
actor "Payment Processor" as processor
rectangle "Falsopay System" {
usecase "Register Account" as UC1
usecase "Login" as UC2
usecase "Manage Profile" as UC3
usecase "View Account Balance" as UC4
usecase "Transfer Money" as UC5
usecase "Pay Bills" as UC6
usecase "Manage Cards" as UC7
usecase "View Transaction History" as UC8
usecase "Generate Reports" as UC9
usecase "Manage Users" as UC10
usecase "Process Payments" as UC11
usecase "Handle Support Tickets" as UC12
usecase "Configure System Settings" as UC13
usecase "Monitor Transactions" as UC14
usecase "Verify Identity" as UC15
usecase "Reset Password" as UC16
}
' Customer Use Cases
customer --> UC1
customer --> UC2
customer --> UC3
customer --> UC4
customer --> UC5
customer --> UC6
customer --> UC7
customer --> UC8
customer --> UC12
customer --> UC16
' Admin Use Cases
admin --> UC2
admin --> UC9
admin --> UC10
admin --> UC12
admin --> UC13
admin --> UC14
admin --> UC16
' Bank Use Cases
bank --> UC2
bank --> UC11
bank --> UC14
bank --> UC15
' Payment Processor Use Cases
processor --> UC11
processor --> UC14
' Relationships
UC1 ..> UC15 : <<include>>
UC2 ..> UC15 : <<include>>
UC5 ..> UC11 : <<include>>
UC6 ..> UC11 : <<include>>
UC7 ..> UC15 : <<include>>
UC10 ..> UC15 : <<include>>
@enduml

## 9. Structural & Behavioural Diagrams

### a) Class Diagrams

The following class diagram shows the main classes and their relationships:

Class Diagram:

@startuml Falsopay Class Diagram
skinparam classAttributeIconSize 0
skinparam class {
BackgroundColor #ECF0F1
BorderColor #2C3E50
ArrowColor #2C3E50
}
skinparam packageStyle rectangle
skinparam handwritten false
skinparam shadowing false
skinparam defaultFontName Arial
skinparam defaultFontSize 12
skinparam roundcorner 20
title Falsopay - Class Diagram
package "Models" {
class User {
-id: int
-email: string
-password: string
-role: string
-createdAt: datetime
-updatedAt: datetime
+register()
+login()
+updateProfile()
+resetPassword()
}
class Account {
-id: int
-userId: int
-accountNumber: string
-balance: decimal
-type: string
-status: string
+getBalance()
+updateBalance()
+transfer()
}
class Transaction {
-id: int
-accountId: int
-type: string
-amount: decimal
-status: string
-timestamp: datetime
+process()
+validate()
+record()
}
class Card {
-id: int
-accountId: int
-cardNumber: string
-type: string
-expiryDate: date
-status: string
+activate()
+deactivate()
+validate()
}
class SupportTicket {
-id: int
-userId: int
-subject: string
-description: string
-status: string
-priority: string
+create()
+update()
+resolve()
}
}
package "Services" {
class AuthService {
+authenticate()
+authorize()
+validateToken()
}
class PaymentService {
+processPayment()
+validatePayment()
+recordTransaction()
}
class NotificationService {
+sendEmail()
+sendSMS()
+sendPushNotification()
}
class ReportService {
+generateTransactionReport()
+generateUserReport()
+exportData()
}
}
package "Controllers" {
class UserController {
+register()
+login()
+updateProfile()
+resetPassword()
}
class AccountController {
+getBalance()
+transfer()
+getTransactions()
}
class CardController {
+activateCard()
+deactivateCard()
+getCardDetails()
}
class SupportController {
+createTicket()
+updateTicket()
+resolveTicket()
}
}
' Relationships
User "1" -- "1..\*" Account
Account "1" -- "0..\*" Transaction
Account "1" -- "0..\*" Card
User "1" -- "0..\*" SupportTicket
UserController --> User
AccountController --> Account
CardController --> Card
SupportController --> SupportTicket
AuthService --> User
PaymentService --> Transaction
NotificationService --> User
ReportService --> Transaction
@enduml

### b) Sequence Diagrams

The following sequence diagram illustrates the money transfer process:

Sequence Diagram:

@startuml Falsopay Money Transfer Sequence
skinparam sequence {
ArrowColor #2C3E50
LifeLineBorderColor #2C3E50
LifeLineBackgroundColor #ECF0F1
ParticipantBorderColor #2C3E50
ParticipantBackgroundColor #ECF0F1
ParticipantFontName Arial
ParticipantFontSize 12
ActorBorderColor #2C3E50
ActorBackgroundColor #ECF0F1
ActorFontName Arial
ActorFontSize 12
}
title Falsopay - Money Transfer Sequence
actor Customer
participant "Web Interface" as UI
participant "AccountController" as AC
participant "PaymentService" as PS
participant "Transaction" as T
participant "NotificationService" as NS
database "Database" as DB
Customer -> UI: Initiate Transfer
activate UI
UI -> AC: transfer(amount, fromAccount, toAccount)
activate AC
AC -> PS: validateTransfer(amount, fromAccount, toAccount)
activate PS
PS -> DB: checkBalance(fromAccount)
activate DB
DB --> PS: balance
deactivate DB
alt Sufficient Balance
PS -> T: createTransaction(amount, fromAccount, toAccount)
activate T
T -> DB: recordTransaction()
activate DB
DB --> T: transactionId
deactivate DB
T -> DB: updateBalances()
activate DB
DB --> T: success
deactivate DB
T --> PS: transactionComplete
deactivate T
PS -> NS: sendTransferNotification()
activate NS
NS --> Customer: Email/SMS Notification
deactivate NS
PS --> AC: transferSuccessful
else Insufficient Balance
PS --> AC: insufficientFunds
end
deactivate PS
alt transferSuccessful
AC --> UI: success
UI --> Customer: Transfer Complete
else insufficientFunds
AC --> UI: error
UI --> Customer: Insufficient Funds
end
deactivate AC
deactivate UI
@enduml

### c) Database Design

The following ERD shows the database schema:

Entity Relationship Diagram:

@startuml Falsopay ERD
!define table(x) class x << (T,#FFAAAA) >>
!define primary\_key(x) <u>x</u>
!define foreign\_key(x) #x#
skinparam class {
BackgroundColor #ECF0F1
BorderColor #2C3E50
ArrowColor #2C3E50
}
skinparam linetype ortho
skinparam packageStyle rectangle
skinparam handwritten false
skinparam shadowing false
skinparam defaultFontName Arial
skinparam defaultFontSize 12
skinparam roundcorner 20
title Falsopay - Entity Relationship Diagram
table(users) {
primary\_key(id): INT
email: VARCHAR(255)
password: VARCHAR(255)
role: VARCHAR(50)
created\_at: DATETIME
updated\_at: DATETIME
}
table(accounts) {
primary\_key(id): INT
foreign\_key(user\_id): INT
account\_number: VARCHAR(20)
balance: DECIMAL(15,2)
type: VARCHAR(50)
status: VARCHAR(20)
created\_at: DATETIME
updated\_at: DATETIME
}
table(transactions) {
primary\_key(id): INT
foreign\_key(account\_id): INT
type: VARCHAR(50)
amount: DECIMAL(15,2)
status: VARCHAR(20)
description: TEXT
created\_at: DATETIME
updated\_at: DATETIME
}
table(cards) {
primary\_key(id): INT
foreign\_key(account\_id): INT
card\_number: VARCHAR(16)
type: VARCHAR(50)
expiry\_date: DATE
status: VARCHAR(20)
created\_at: DATETIME
updated\_at: DATETIME
}
table(support\_tickets) {
primary\_key(id): INT
foreign\_key(user\_id): INT
subject: VARCHAR(255)
description: TEXT
status: VARCHAR(20)
priority: VARCHAR(20)
created\_at: DATETIME
updated\_at: DATETIME
}
table(notifications) {
primary\_key(id): INT
foreign\_key(user\_id): INT
type: VARCHAR(50)
message: TEXT
status: VARCHAR(20)
created\_at: DATETIME
}
' Relationships
users "1" -- "0..\*" accounts
accounts "1" -- "0..\*" transactions
accounts "1" -- "0..\*" cards
users "1" -- "0..\*" support\_tickets
users "1" -- "0..\*" notifications
@enduml

# PART 3: Development Phase

## 11. 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

## 13. Complexity Metrics

|  |  |  |
| --- | --- | --- |
| Metric | Value | Description |
| Lines of Code (LOC) | 15,000 | 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 |

## 14. Testing Reports

|  |  |  |
| --- | --- | --- |
| Test Type | Coverage | Results |
| Unit Tests | 85% | 1,200 tests passed, 50 failed |
| Integration Tests | 75% | 300 tests passed, 20 failed |
| System Tests | 90% | 150 tests passed, 5 failed |
| User Acceptance Tests | 95% | 50 tests passed, 2 failed |

# Appendix

## A. Test Cases

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case ID | Description | Expected Result | Actual Result |
| TC001 | User Registration | Account created successfully | Passed |
| TC002 | Money Transfer | Transfer completed successfully | Passed |
| TC003 | Card Activation | Card activated successfully | Passed |