# PROJECT DOCUMENTATION TEMPLATE

Subject: Networking Issues in Mobile Systems

## Part 1 – Conceptual Design

Project Topic: Mobile Currency Exchange System

## 1. General Information

Project Name NCMAD Currency Exchange - Exchange Pro

Project Authors Ahmet Emin Yada

Field of Study Computer Science / Mobile Development

Year / Semester 7th Semester

Supervisor Marcin Kacprowicz

Submission Date Feb 5 / 10:40

## 2. Project Description

### 2.1. Project Objective

The aim of the project is to create a fully functional mobile application that enables virtual currency exchange in real-time. The system provides users with detailed insight into current market rates via the National Bank of Poland (NBP) API, allows for the management of a multi-currency wallet, and facilitates secure buy/sell transactions. The system's user value lies in its ability to simulate a real banking environment with instant feedback and secure data persistence.

### 2.2. Project Scope

The system consists of three main modules:

1. \*\*Mobile Application (iOS Client):\*\* A native SwiftUI interface for users to register, view rates, and perform trades.

2. \*\*Web Service (Backend API):\*\* A Node.js/Express server handling business logic, authentication, and data coordination.

3. \*\*Database:\*\* A SQLite database for persistent storage of user credentials, wallet balances, and transaction history.

## 3. System Requirements

### 3.1. Functional Requirements

A table presenting all system functions:

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Function Name | Description | Priority |
| F1 | User registration | The system allows new users to create an account with email and password. | High |
| F2 | User Login | Secure authentication using JWT tokens. | High |
| F3 | View Current Rates | Fetch and display live exchange rates from NBP API. | High |
| F4 | Fund Account | Simulate depositing PLN currency into the user's wallet. | High |
| F5 | Currency Exchange | Atomically exchange funds between two currency wallets (e.g., PLN -> USD). | High |
| F6 | View History | Display a chronological list of all past transactions. | Medium |
| F7 | Historical Rates | View exchange rate trends for the past 30 days. | Medium |

### 3.2. Non-Functional Requirements

Description of system quality requirements:

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Name | Description | Category |
| N1 | Performance | System API response time should be ≤ 2 seconds under normal load. | Performance |
| N2 | Security | User passwords must be hashed (bcrypt); API access secured via JWT. | Security |
| N3 | Data Integrity | Financial transactions must be atomic (ACID compliant). | Reliability |
| N4 | Usability | The UI must support standard iOS navigation patterns and dark mode. | Usability |

## 4. UML Diagrams

### 4.1. Use Case Duml-class-diagram-BACKEND.pnguml-class-diagram-IOS-APP.pngerd-database-diagrm.pngiagramuml-use-case-diagram.png

### 4.2. Class Diagram

## 5. Database Design

Description of Tables:

1. \*\*users:\*\* Stores user identity.

\* \*Primary Key:\* id (UUID)

\* \*Attributes:\* email, password\_hash, name.

2. \*\*currency\_wallets:\*\* Stores current balance for a specific currency.

\* \*Primary Key:\* id (UUID)

\* \*Foreign Key:\* user\_id (References users)

\* \*Compount Unique Key:\* (user\_id, currency\_code)

3. \*\*transactions:\*\* Immutable log of financial operations.

\* \*Primary Key:\* id (UUID)

\* \*Foreign Key:\* user\_id (References users)

\* \*Attributes:\* amount, rate, timestamp, type (BUY/SELL).

## 6. System Architecture

The project utilizes a \*\*Client-Server Architecture\*\*.

\* \*\*iOS App:\*\* Functions as the presentation layer. It communicates with the server via HTTP REST requests. It caches session data using SwiftData.

\* \*\*Web Server (Node.js):\*\* Acts as the logic layer. It processes requests, authorizes users via Middleware, and executes database commands.

\* \*\*External API (NBP):\*\* A third-party service providing raw financial data.

\*\*Logical Data Flow:\*\*

User Action (UI) -> ViewModel -> APIService -> REST Request -> Express Route -> Database Controller -> SQLite -> Response.

## 7. Project Implementation Plan

|  |  |  |  |
| --- | --- | --- | --- |
| Stage | Description | Deadline | Responsible Person |
| 1 | Requirements analysis and topic selection. | Week 1 | AHMET |
| 2 | ackend Setup (Node.js) and Database Schema Design. | Week 2 | AHMET |
| 3 | iOS UI Implementation (SwiftUI Views). | Week 3 | AHMET |
| 4 | API Integration (Connecting App to Backend & NBP) | Week 4 | AHMET |
| 5 | Testing and Bug Fixes (Auth, Transaction Atomic logic) | Week 5 | AHMET |
| 6 | Documentation and Final Polish. | Week 6 | AHMET |

## 8. Conclusions and Possible Extensions

The project successfully meets all defined functional requirements. The system is stable, secure, and user-friendly.

\*\*Possible Extensions:\*\*

\* \*\*Push Notifications:\*\* Alert users when a currency reaches a target rate.

\* \*\*Crypto Support:\*\* Add cryptocurrency rates via a different API (e.g., CoinGecko).

\* \*\*Payment Gateway:\*\* Integration with Stripe to allow real money deposits.

## 9. Sources

1. \*\*NBP Web API Documentation:\*\* `http://api.nbp.pl/`

2. \*\*SwiftUI Documentation (Apple Developer):\*\* `https://developer.apple.com/documentation/swiftui/`

3. \*\*Node.js & Express Guide:\*\* `https://expressjs.com/`

4. \*\*Better-SQLite3 Documentation:\*\* `https://github.com/WiseLibs/better-sqlite3`