Software Requirements Specification

File Name	Exchange Store
Creation Date	22.09.2025
Authored by	Kuzey Arda Bulut
Reviewed by	

REVISION HISTORY

VERSION	DATE	DESCRIPTION	AUTHOR
V1	15.09.2025	Choosed to do Exchange Store Application and did some reserach about the economic terms	**Kuzey Arda Bulut**
V2	16.09.2025	Created the program’s general structure	**Kuzey Arda Bulut**
V3	20.09.2025	Implemented the heap objects	**Kuzey Arda Bulut**
V4	21.09.2025	Finalized the Project	**Kuzey Arda Bulut**

CONTENTS

1 SW System Overview

Specify the purpose and the overview of the SRS.

1.1 Purpose

Describe the purpose of the system. What problem does it solve? Who are the intended users? Why is it being developed?

The system is designed to simulate the daily operations of a currency exchange office. Currently, there is no comprehensive exchange store simulation available that enables cashiers to check exchange rates, perform transactions, issue receipts, and generate end-of-day reports. The intended users of the system include exchange store staff and stakeholders, such as cashiers, customers, and management.

1.2 Scope

Define the scope of the system. What functionality is included? What is explicitly excluded? Mention benefits and key features.

The scope of the system covers the main functionalities required for running a small-scale exchange operation. It allows users to perform exchanges between currencies, display and update exchange rates, adjust reserves, and generate receipts and reports. In addition, the system offers denomination breakdowns, allowing cashiers to suggest appropriate note distributions when paying out customers. Certain features are intentionally excluded from the current implementation, such as real-time integration with market data, as the project's requirements explicitly prohibit internet access for applications. The system provides value to both users and exchange office owners by offering a realistic simulation that clarifies the logic of exchange operations and supports a better understanding of the workflow.

1.3 Use-Case Diagram

Provide a high-level UML use-case diagram showing main actors and their interactions with the system.

1.4 General Constraints

List technical and business constraints such as programming language, operating system, performance limitations, and standards.

1.5 Assumptions and Dependencies

State assumptions (e.g., availability of internet, supported devices) and dependencies (e.g., external APIs, hardware).

1.6 Acronyms and Abbreviations

List all acronyms and abbreviations used in the document along with their explanations.

Terms Used	Description of terms

2 SW Functional Requirements

2.1 Features / Functions to be Implemented

All functional requirements should be derived from User Stories or Use Cases.

This means that instead of listing abstract features, you first describe how users interact with the system and what goals they achieve.

User Stories – short, simple descriptions of a feature told from the perspective of the user (e.g., "As a registered user, I want to reset my password so that I can regain access to my account.").

Use Cases – structured scenarios that describe interactions between actors and the system, including preconditions, steps, and outcomes.

From these stories/cases, you can then identify:

- *User interactions (e.g., authentication, profile management).*
- *Business processes (e.g., order processing, reporting).*
- *Integrations (e.g., with external APIs or third-party systems).*
- *System logic (e.g., validation, workflows, automation).*
- *Algorithms (if required, e.g., recommendation or prediction).*

Each function must be traceable back to a User Story or Use Case, ensuring that the system is built strictly according to user and business needs.

The system provides several essential features that form its functional core. Users can perform full or partial currency exchanges, which automatically update the reserves of the respective currencies and calculate the profit made. Management functions allow for exchange rates to be displayed and updated, ensuring flexibility in adapting to new financial conditions. Cashiers can modify reserves by depositing or withdrawing funds and set critical minimum levels in order to ensure about operational continuity. Receipts are generated after each exchange, while end-of-day reports summarize all activity, showing profits and balances in an organized manner.

2.1 Acceptance Criteria

Define how each requirement will be validated: test cases, acceptance tests, or quality metrics.

2.2 Implementation Requirements

Provide details of specific implementation requirements if applicable. For example, integration with existing systems, supported platforms, or algorithms.

3 SW Non-Functional Requirements

3.1 Resource Consumption

Specify performance and resource limits (CPU, memory, storage, response time).

3.2 License Issues

State licensing requirements and constraints on third-party software or libraries.

3.3 Coding Standard

Define coding style and standards that must be followed.

3.4 Modular Design

Specify architectural requirements such as modularity, extensibility, and maintainability.

3.5 Reliability

Define requirements for reliability, error handling, and fault tolerance.

3.6 Portability

List target platforms and environments where the system should operate.

3.7 General Operational Guidelines

Provide guidelines for scalability, robustness, ease of use, and maintainability.

4 SW Design Artifacts

4.1 CRC Cards (Class–Responsibility–Collaboration)

List the main classes with their responsibilities (action verbs) and collaborators (related classes); keep items concise and implementation-agnostic.

4.2 Conceptual UML Diagram (entities & relationships)

Draw a conceptual class diagram with key entities and their relationships; focus on nouns from User Stories/Use Cases, omit methods and low-level details.