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.

1.1.1 Main Actors

- **Client**: requests a currency exchange and expects the converted result.
- **Cashier**: primary user in Release 1, inputs transactions and reads results.
- **System (Backend)**: validates inputs, performs conversion, and stores transaction details.
- **Administrator** *(future role)*: manages exchange rates, reserves, and generates automated reports (not part of Release 1).

1.1.2 Business Process

1. Cashier selects **FROM** and **TO** currencies from a fixed list (LOC, USD, EUR, GBP, JPY).
2. Cashier enters the amount to convert.
3. System validates input (amount > 0, valid indices).
4. If valid → system computes result using predefined demo rates and prints it.
5. Each transaction is recorded in memory with details (date, time, currencies, rate, profit).
6. At the end of the day, cashier can view a **manual profit summary**. *(Later releases: automated daily reports, receipt printing, CSV storage, reserve checks.)*

1.1.3 Problem Areas

- Invalid inputs: negative or zero amounts, out-of-range currency indices.
- No persistence in Release 1 → all transactions lost after program exits.
- Manual profit entry/reporting introduces risk of errors.
- Hardcoded exchange rates → not realistic for actual business use.
- *(Later problems: insufficient reserves, inconsistent rate updates, missing error logs.)*

1.1.4 Conclusion (Three Axes)

- **Right Product**: focus on a simple **console-based exchange tool**, not a financial calculator. Release 1 supports fixed conversions only.
- **Done Right**: input validation (amount > 0, valid currency codes) is enforced; clear error messages are shown. Transaction details are tracked in memory.
- **Managed Right**: Release 1 uses a simple structure (main.c + utils.c/h). Future releases will modularize (e.g., exchange.c, reports.c), add CSV logging, denomination breakdown, and automated reporting.

1.2 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.

1.3 Scope

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

Included (Scope):

- Perform exchanges between supported currencies.
- Input of amount and currency pair (FROM, TO).
- Validation of input (amount > 0, valid indices).
- Calculation using fixed, predefined exchange rates.
- Display converted result on screen.

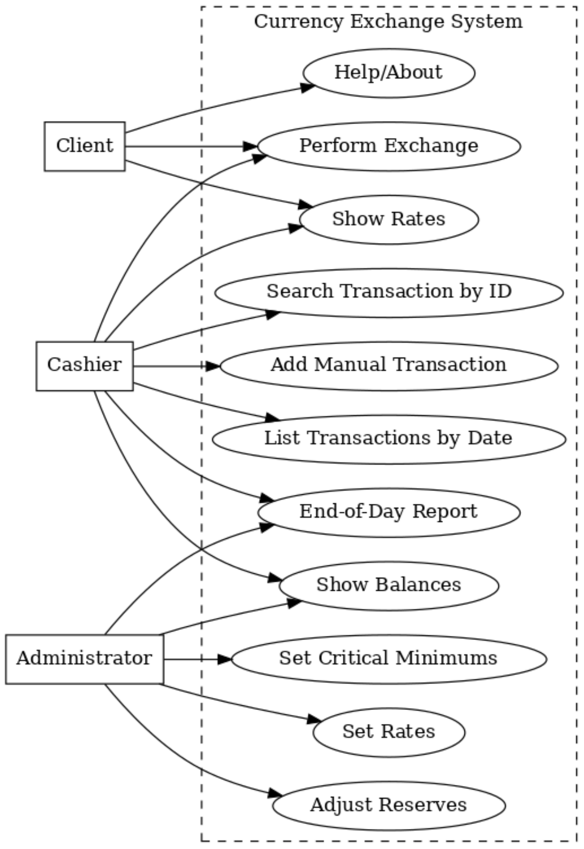
- Denomination breakdowns: suggest note distribution for payouts.
- View and update demo exchange rates (set manually by administrator in later releases).
- Generate receipts and simple daily reports (manual profit entry in Release 1).

Excluded (Not in Scope):

- Real-time integration with market/online data.
- Internet access or networking functionality.
- Graphical User Interface (console only).
- Automated reserve management (only manual adjustments in later releases).

1.4 Use-Case Diagram

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



1.5 General Constraints

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

- *Implementation language:* ***C++***
- *Platform:* ***Windows/Linux desktop***
- *Data storage:* ***file system*** *(text/CSV files: receipts_2025-09-26.txt, sales_2025-09-26.csv)*
- *Standards:* ***UML notation*** *for diagrams (use-case, activity, class)*
- *Performance:* ***exchange calculation and balance update must be completed within 1 second*** *after the cashier confirms the operation*
- *User interface:* ***simple console menu*** *or* ***display board*** *showing rates and availability*
- *Capacity constraint:* ***maximum number of supported currencies and denominations is fixed and configured*** *in the system*

1.6 Assumptions and Dependencies

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

- *The exchange office has stable access to electricity and a functioning local computer.*

- *The cashier or administrator manually sets and updates exchange rates within the system.*
- *The system depends on the local file system being accessible for storing rates, balances, transactions, and reports.*
- *No internet connection or integration with external APIs is required; all data is handled locally.*
- *Denomination lists and supported currencies are predefined and do not change dynamically.*
- *Users (cashier/administrator) are trained to operate a console-based interface.*

1.7 Acronyms and Abbreviations

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

Terms Used	Description of terms
CSV	Comma-Separated Values; file format used for storing transactions, balances, and rates.
LOC	Local currency (e.g., the home currency of the exchange office).
USD	United States Dollar.
EUR	Euro.
GBP	British Pound Sterling.
JPY	Japanese Yen.
UML	Unified Modeling Language; standard notation used for diagrams in this project.
SRS	Software Requirements Specification; this document describe the system requirements.
UI	User Interface; the console menu or display board through which users interact with the system.

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.

User Stories (Currency Exchange)

- *US-1 (Cashier): As a cashier, I want to enter/update exchange rates so that I can calculate exchanged amounts correctly.*
- *US-2 (Client): As a client, I want to receive a printed (or saved) receipt so that I have proof of the transaction.*
- *US-3 (Manager): As a manager, I want to generate a daily summary so that I can verify all transactions.*
- *US-4 (Cashier): As a cashier, I want to enter the transaction (from/to currency, amount) so that the system can compute and validate it.*
- *US-5 (System): As the system, I want to log each transaction to a file so that records are persisted for audits/reports.*

Derived Functions / Features

- *F-1: Rate Management — Create/Update/View fixed exchange rates. (From US-1)*
- *F-2: Transaction Input & Validation — Enter amount, select currencies, validate input and reserves. (From US-4)*
- *F-3: Exchange Calculation — Compute exchanged amount with configured rate, fees/rounding if applicable. (From US-1, US-4)*
- *F-4: Receipt Generation — Print/save receipt with transaction ID, timestamp, rates, amounts. (From US-2)*
- *F-5: File Logging — Append transaction records to CSV (ID, timestamp, from/to, rate, amounts). (From US-5)*

2.1 Acceptance Criteria

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

- *A receipt must be generated and either printed or saved in a file after each successful exchange.*
- *The system must calculate the exchanged amount with an accuracy of two decimal places.*
- *Invalid inputs (e.g., negative amounts, unsupported currency codes) must trigger an error message without processing the transaction.*
- *Each successful transaction must be logged into a CSV file including: transaction ID, timestamp,*

from/to currency, rate, amount, and exchanged result.

- *The daily summary report must include:*
  - *Total number of transactions,*
  - *Total exchanged amount per currency,*
  - *Overall balance and reserves status.*
- *The system must update and display reserve balances immediately after each transaction.*
- *Denomination breakdown must be provided whenever the cashier requests a payout in notes/coins.*

2.2 Implementation Requirements

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

- *All transactions must be stored in a CSV file with: transaction ID, timestamp, input currency, output currency, applied rate, input amount, and exchanged amount.*
- *The system must run as a console-based application (CLI) without GUI elements.*
- *The program must support both Windows and Linux platforms.*
- *Exchange rates must be stored in a configuration file, such as text and CSV, which can be updated by the administrator.*
- *The system must use floating-point arithmetic with precision up to two decimal places for currency conversions.*
- *Error handling must prevent invalid inputs from breaking program flow.*
- *UML diagrams must be delivered for use cases, classes and sequence flows, ensuring traceability to requirements.*

3 SW Non-Functional Requirements

3.1 Resource Consumption

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

- *Response time for an exchange operation: ≤ **1 second***
- *Maximum memory usage: ≤ **50 MB***
- *Maximum file size for daily logs (CSV): ≤ **2 MB***
- *Daily summary report must be generated within ≤ **2 seconds***

3.2 License Issues

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

- *Only **standard C libraries** are allowed.*
- *No proprietary third-party libraries are permitted.*
- *External libraries may only be used if they are **open-source with permissive licenses, as if** MIT, Apache 2.0 and BSD.*
- *Printing or file handling must rely on **native system functions** without external dependencies.*

3.3 Coding Standard

Define coding style and standards that must be followed.

- *Each function must include **clear descriptive comments** explaining inputs, outputs and error handling.*
- *Variable and function names must follow **lower_snake_case** convention.*
- *All constants must be declared using **#define or const**.*
- *Error handling must be implemented for **all user inputs and file operations**.*
- *Unit tests (where applicable) must cover critical components, including:*
  - *Exchange calculation*
  - *Input validation*
  - *File logging*

3.4 Modular Design

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

- *The system shall consist of separate modules for:*
  - ***Exchange calculation** (amount conversions, reserve checks)*
  - ***File logging** (transaction records in CSV format)*
  - ***Reporting** (daily summaries, reserve balances)*
  - ***User interaction** (menus, inputs, receipts)*
  - ***Configuration management** (exchange rates, thresholds)*
- *Modules must be designed for **low coupling and high cohesion**.*
- *Core logic (calculation, logging) must be separated from user interface code.*
- *The design must allow **future extensions** (e.g., support for new currencies or different file formats).*

3.5 Reliability

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

- *The system must reject invalid input without crashing.*
- *All file writes must be **atomic** to avoid corruption.*
- *In case of file system errors, the system must create a new valid file automatically.*
- *Clear error messages must be displayed to the cashier and logged into an **error log file** for troubleshooting.*
- *The system must ensure that partial transactions are either fully rolled back or re-executable without inconsistencies.*

3.6 Portability

List target platforms and environments where the system should operate.

- *The system must compile and run on **Windows 10+** and **Ubuntu Linux (20.04 or newer)**.*
- *Identical inputs must produce **identical outputs** across both platforms.*
- *No platform-specific features (e.g., Windows-only APIs) may be used.*

3.7 General Operational Guidelines

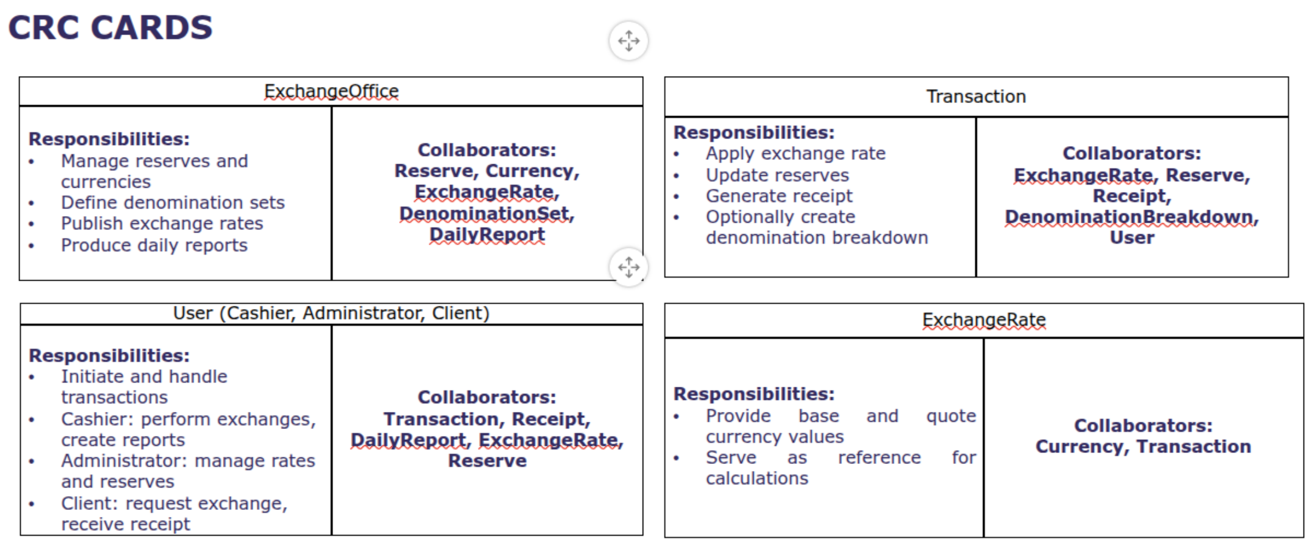
Provide guidelines for scalability, robustness, ease of use, and maintainability.

- *The system must be **robust, easy to maintain, and simple to use** for non-technical users (cashiers, managers).*
- *A **daily reset functionality** must be provided to start each new workday with a clean state while archiving previous logs.*
- *All operations must be logged for **accountability and auditing purposes**.*
- *The system must be designed to allow **future scalability**, e.g., adding more currencies or different reporting formats.*
- *The menu-driven interface must be intuitive, with **clear instructions and error messages**.*

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.



CRC CARDS

ExchangeOffice	
Responsibilities: • Manage reserves and currencies • Define denomination sets • Publish exchange rates • Produce daily reports	**Collaborators: Reserve, Currency, ExchangeRate, DenominationSet, DailyReport**

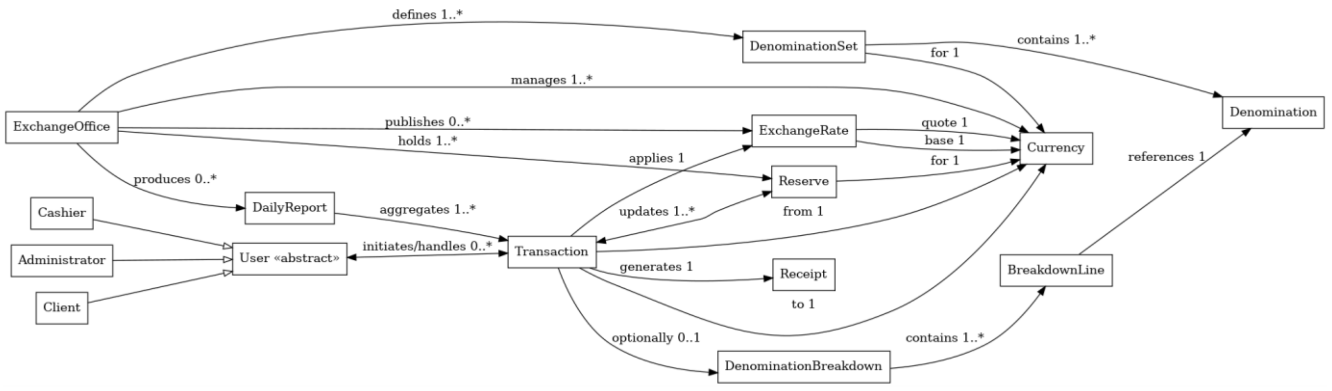
Transaction	
Responsibilities: • Apply exchange rate • Update reserves • Generate receipt • Optionally create denomination breakdown	**Collaborators: ExchangeRate, Reserve, Receipt, DenominationBreakdown, User**

User (Cashier, Administrator, Client)	
Responsibilities: • Initiate and handle transactions • Cashier: perform exchanges, create reports • Administrator: manage rates and reserves • Client: request exchange, receive receipt	**Collaborators: Transaction, Receipt, DailyReport, ExchangeRate, Reserve**

ExchangeRate	
Responsibilities: • Provide base and quote currency values • Serve as reference for calculations	**Collaborators: Currency, Transaction**

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.


defines 1..*
DenominationSet
contains 1..*
for 1
manages 1..*
Denomination
ExchangeOffice
publishes 0..*
ExchangeRate
quote 1
base 1
for 1
Currency
holds 1..*
applies 1
references 1
produces 0..*
Reserve
Cashier
DailyReport
aggregates 1..*
updates 1..*
from 1
Administrator
User «abstract»
initiates/handles 0..*
Transaction
BreakdownLine
generates 1
Receipt
Client
to 1
optionally 0..1
contains 1..*
DenominationBreakdown