Currency Exchange System

Software Requirements Specification

File Name	*Currency Exchange SRS*
Creation Date	*22.09.2025*
Authored by	*Vladislav Ignatjev*
Reviewed by	

REVISION HISTORY

VERSION	DATE	DESCRIPTION	AUTHOR
1.0	*22.09.2025*	*Initial Version*	***Vladislav Ignatjev***

CONTENTS

1 SW System Overview

This SRS describes the requirements for a software system designed to support currency exchange operations in a small exchange office. The system will help cashiers automate their tasks, such as: calculating exchange amounts, issuing receipts and recording daily operations in order to provide reports to the management. The software is aimed at reducing human errors and speeding up service.

1.1 Purpose

The purpose of the system is to complete a currency exchange transaction quickly, issue receipts and record operations in a file for daily reporting.

1.2 Scope

1. *Included: exchange rate management, calculation of exchanged amount, rounding, receipt generation, logging and summarizing transactions into a file, daily report generation.*
2. *Excluded: online conversion rate fetching, online connectivity, web pages, tax reporting*
3. *Benefits: increased service speed, reduced human errors, easier storage of records and reporting.*
4. *Key features: automatic calculating and rounding, printable receipt, summary reports generation.*

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
SRS	Software Requirements Specification

2 SW Functional Requirements

2.1 Features / Functions to be Implemented, User Stories

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

- *As a client I want to receive a receipt, so that I could have proof of the transaction.*
- *As a cashier, I want to check reserves for each currency so that I can let the client know whether a transaction can be fulfilled.*
- *As a cashier, I want a clear error message if my inputs are non numerical.*
- *As a cashier, I want to enter the source currency, target currency, and amount so that I can see the converted amount before I confirm the transaction.*
- *As a manager, I want to be able to load the current conversion rates into the system.*
- *As a manager, I want a session summary at the end of the day with the starting and ending balances of each currency, as well as the profit earned.*
- *As a cashier, I want the exchanged amount to be rounded to two decimal places using a consistent rule.*
- *As a client, I want to request money in specific denominations or partially exchanged (e.g., exchanging 100 units for 50 units in foreign currency and the remainder in local currency).*

From these stories you can identify functions:

- *User interactions: entering transaction data, printing or displaying receipts, viewing current reserves, receiving clear error messages, fulfilling requests for partial exchanges or specific denominations.*
- *Business processes: load the exchange rates at the beginning of the day, produce daily session summary with balance and profit*
- *System logic: automatic calculation and conversion using conversion rates, rounding to two decimal places, updating currency reserves after each transaction, and logging each confirmed transaction to a file, handling of partial or split transactions.*

2.1 Acceptance Criteria

- *A receipt must be generated and saved in a file after each successful exchange.*
- *The system must calculate the exchanged amount with an accuracy of two decimal places.*
- *The daily summary report must include the total number of transactions and the total exchanged amount.*
- *When reserves are insufficient, the system prevents the transaction and displays a clear message with the amount available versus required.*
- *Invalid inputs (non-numeric amounts or unsupported currency codes) are rejected, and the system provides a descriptive error message before re-prompting.*
- *Each confirmed transaction is appended to the log file as a new entry.*
- *If the client requests specific denominations or a partial exchange, the system supports splitting the transaction and records it correctly in the receipt.*

2.2 Implementation Requirements

- *All transactions must be stored in a CSV file with timestamp, input currency, output currency, and amount.*
- *The program must work in console mode (CLI) only*
- *The system must allow a single input transaction to be split into multiple outputs (partial exchange) or expressed in user-specified denominations, updating reserves and logs accordingly.*

3 SW Non-Functional Requirements

3.1 Resource Consumption

Resource Consumption
- Response time for exchange operation: ≤ 2 seconds
- Maximum memory usage: ≤ 100 MB
- Maximum file size for daily logs: ≤ 5 MB

3.2 License Issues

License Issues
- Only standard C++ STL libraries are allowed.
- No proprietary third-party libraries are permitted.
- External libraries may only be used if they have permissive open-source licenses (MIT, Apache2.0).

3.3 Coding Standard

Coding Standard
- Each function and class must include descriptive comments.
- Unit tests must cover all critical components (e.g., calculation of exchanged amount).

3.4 Modular Design

Modular Design
- The system shall consist of separate modules for:
• Exchange calculation
• File logging
• Reporting
• User interaction
- Modules must be designed for low coupling and high cohesion.

3.5 Reliability

Reliability
- The system must reject invalid input without crashing.
- File writes must be atomic to avoid corruption.
- Error messages must be logged in a text file for troubleshooting.

3.6 Portability

Portability
- The system must compile and run on Windows 10+ and Ubuntu Linux.
- Identical inputs must produce identical outputs on both platforms.

3.7 General Operational Guidelines

General Operational Guidelines
- The system must be robust, easy to maintain, and simple to use.
- Daily reset functionality must be provided to start each workday with a clean state.
- All operations must be logged for accountability and auditing purposes.

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.