Software Requirements Specification

File Name	Currency exchange system
Creation Date	22.09.2025
Authored by	Magnus Jaaska
Reviewed by	Magnus Jaaska

REVISION HISTORY

VERSION	DATE	DESCRIPTION	AUTHOR
1.0	22.09.2025	Initial Version	**Magnus Jaaska**

CONTENTS

1 SW System Overview

This document specifies the requirements for the **Currency Exchange System**, which is developed as part of the Fundamentals of C++ Programming course project. The overview introduces the purpose of the system, its scope, main use cases, constraints, and assumptions. It provides a high-level understanding of the system for all stakeholders, including students (developers), instructors (evaluators), and end-users (cashiers, clients, and managers in the modeled scenario).

1.1 Purpose

The purpose of the Currency Exchange System is to automate manual currency conversion at a local exchange office. The system reduces calculation errors, improves transaction speed, and ensures transparency through receipts and daily reports. The primary users are cashiers (executing exchanges), clients (receiving receipts), and managers (reviewing daily summaries).

1.2 Scope

In Scope:

- Exchange between currencies with user-entered rates
- Validation of available reserves before confirming a transaction
- Automatic calculation of converted amounts
- Receipt generation for each completed transaction
- Daily summary report showing balances and profit

Out of Scope:

- Internet-based rate feeds
- Multi-branch synchronization
- Integration with external tax/banking systems

The system benefits the exchange office by providing accurate, auditable, and faster services.

1.3 Use-Case Diagram

Actors: Client, Cashier, Manager
Use cases: Perform Exchange, Print Receipt, Manage Rates, Generate Report, Notify Low Reserve
(Insert simple UML diagram here — stick figures + 5 ovals)

1.4 General Constraints

- Language: C++17
- Platform: Windows 10+ and Ubuntu 22.04
- UI: Console-based (CLI)
- Storage: Local file system (CSV logs)
- Performance: ≤ 2 seconds per transaction

1.5 Assumptions and Dependencies

- Exchange rates are set manually by management before operations begin
- The office has stable power and a working PC
- Files can be stored locally without network dependencies
- No internet connection is required

1.6 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
CLI	Command Line Interface
UML	Unified Modeling Language

2 SW Functional Requirements

2.1 Features / Functions to be Implemented

- FR-1 Perform Exchange: Calculate converted amount and check reserves
- FR-2 Print Receipt: Provide client with receipt after each successful exchange
- FR-3 Record Transaction: Save all exchanges to CSV file
- FR-4 Daily Report: Summarize balances and profit at end of day
- FR-5 Manage Rates: Allow manager to update rates
- FR-6 Low Reserve Alert: Notify when reserve drops below threshold

2.2 Acceptance Criteria

- AC-1 (FR-1): Valid transaction executes in ≤2s and outputs correct converted amount
- AC-2 (FR-2): Receipt includes ID, timestamp, currencies, rate, amounts
- AC-3 (FR-3): Each transaction appended as one CSV line
- AC-4 (FR-4): Daily report includes total exchanges, balances, and profit
- AC-5 (FR-5): New rates take effect immediately for next transaction
- AC-6 (FR-6): If reserve insufficient, transaction rejected with clear error

2.3 Implementation Requirements

- Use C++ standard library only
- File format for logs: CSV with fields (timestamp, tx_id, source_currency, source_amount, target_currency, target_amount, rate)
- Separate modules for calculation, logging, reporting

3 SW Non-Functional Requirements

3.1 Resource Consumption

- Each transaction must complete in ≤ 2 seconds.
- The system must run on a standard PC with at least 2 GB RAM and 2 CPU cores.
- Daily transaction logs must not exceed 5 MB in size.
- Memory usage should remain below 100 MB during normal operation.

3.2 License Issues

- The system will use only C++ Standard Library features.
- If third-party libraries are added later, they must be under permissive licenses (MIT, Apache-2.0, or similar).

3.3 Coding Standard

- All code must compile under C++17.
- Classes, methods, and variables must use clear and consistent naming conventions (CamelCase for classes, lower_snake_case for variables).
- Each class and public method must include comments describing its purpose.

3.4 Modular Design

- The system must be divided into separate modules for:
    - Transaction handling
    - Reserve management
    - Receipt generation
    - Report generation
- Modules must be loosely coupled and highly cohesive to ease future maintenance.

3.5 Reliability

- Invalid inputs (negative amounts, unknown currencies) must be rejected gracefully.
- Transactions must only be confirmed if successfully logged.
- Error messages must be clear and guide the user to corrective action.

3.6 Portability

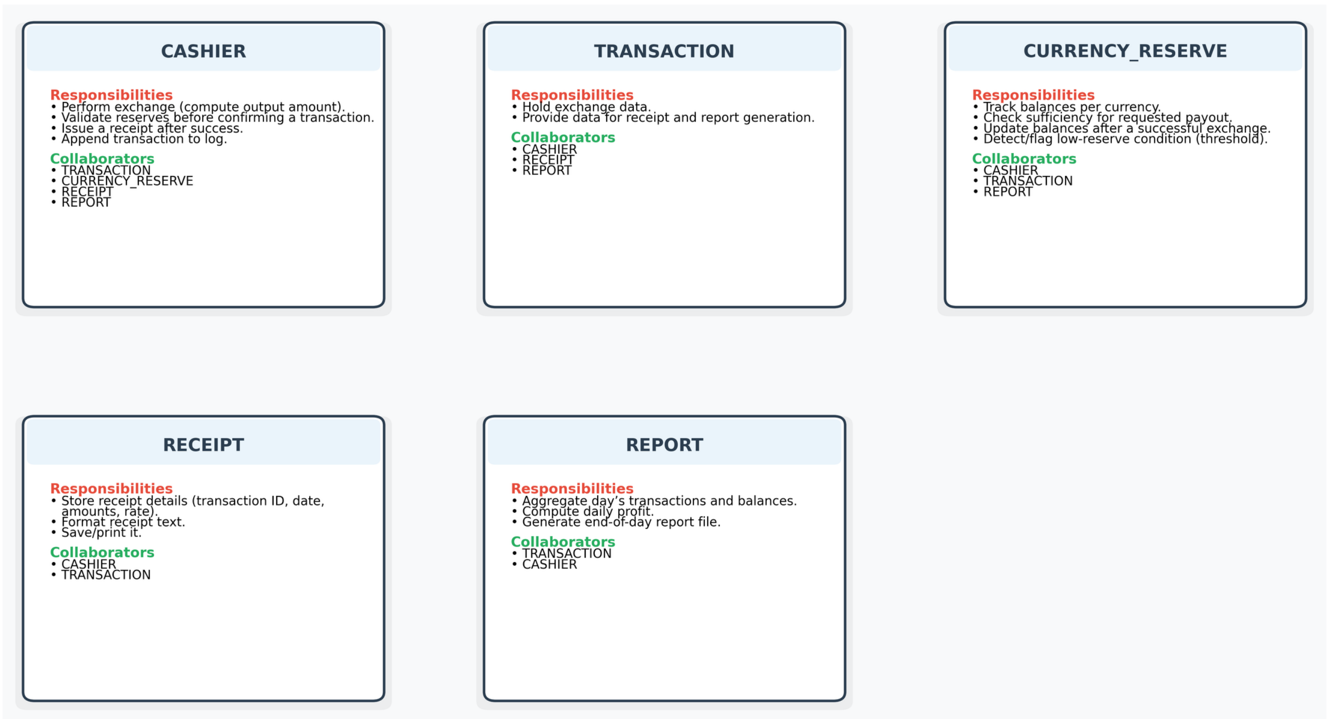
- The system must run without modification on Windows 10+ (MSVC) and Ubuntu 22.04 (g++/clang).
- The same input must produce identical output across platforms.

3.7 General Operational Guidelines

- The system must be robust against invalid user input and file I/O failures.
- Design must allow scalability, e.g., support for additional currencies without major redesign.
- The system must be easy to use, with simple console prompts.
- Maintainability must be supported through modular structure, comments, and clear file organization.

4 SW Design Artifacts

4.1 CRC Cards (Class–Responsibility–Collaboration)



CASHIER

Responsibilities
- Perform exchange (compute output amount).
- Validate reserves before confirming a transaction.
- Issue a receipt after success.
- Append transaction to log.

Collaborators
- TRANSACTION
- CURRENCY_RESERVE
- RECEIPT
- REPORT

TRANSACTION

Responsibilities
- Hold exchange data.
- Provide data for receipt and report generation.

Collaborators
- CASHIER
- RECEIPT
- REPORT

CURRENCY_RESERVE

Responsibilities
- Track balances per currency.
- Check sufficiency for requested payout.
- Update balances after a successful exchange.
- Detect/flag low-reserve condition (threshold).

Collaborators
- CASHIER
- TRANSACTION
- REPORT

RECEIPT

Responsibilities
- Store receipt details (transaction ID, date, amounts, rate).
- Format receipt text.
- Save/print it.

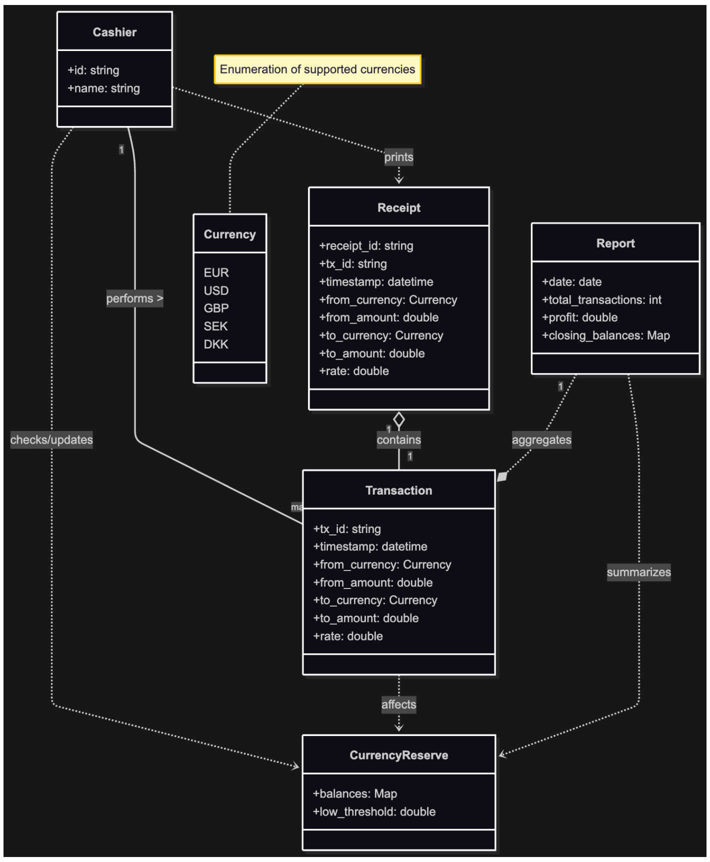
Collaborators
- CASHIER
- TRANSACTION

REPORT

Responsibilities
- Aggregate day's transactions and balances.
- Compute daily profit.
- Generate end-of-day report file.

Collaborators
- TRANSACTION
- CASHIER

4.2 Conceptual UML Diagram (entities & relationships)



4.3 User Stories

- **US-1:** As a **Cashier**, I want to enter the exchange details so that the system calculates the converted amount and validates reserves.
- **US-2:** As a **Client**, I want to receive a printed receipt so that I have proof of the transaction.
- **US-3:** As a **Manager**, I want a daily report so that I can review totals and profit.
- **US-4:** As a **Manager**, I want to set/update rates so that profitability stays controlled.
- **US-5:** As a **Cashier**, I want low-reserve warnings so that I don’t confirm impossible operations.