

SRS-DLD

S/W Detailed Level Design

Project Name	Currency Exchange System		
Block Name			
Author	Vladislav Ignatjev	Approver	
Team	13		



Vladislav Ignatjev

30.10.2025

S/W Detailed Level Design

Contents

1.	Overview.....	4
2.	System Overview / Architectural Context.....	5
3.	UML Class Diagram (Technical Design).....	6
4.	Class Specifications.....	6
5.	Interfaces and Abstractions.....	6
6.	Function Responsibilities.....	6
7.	Operation Flow.....	7
8.	Enumerations & Constants.....	7
9.	Validation Rules & Future Work.....	7
10.	Traceability Matrix.....	7
11.	Code Structure and File Mapping.....	7
12.	Revision History.....	8

TITLE S/W Detailed Level Design

■ Revision History

Version	Date	Revised contents	Author	Approver
1.0	2025-09-22	Initial version	Vladislav I.	
2.0	2025-10-30	Added Release 2 architecture	Vladislav I.	

■ Terms and Abbreviations

Term	Description

■ References

- ## 1. SW Requirements Specification

Vladislav Ignatjev

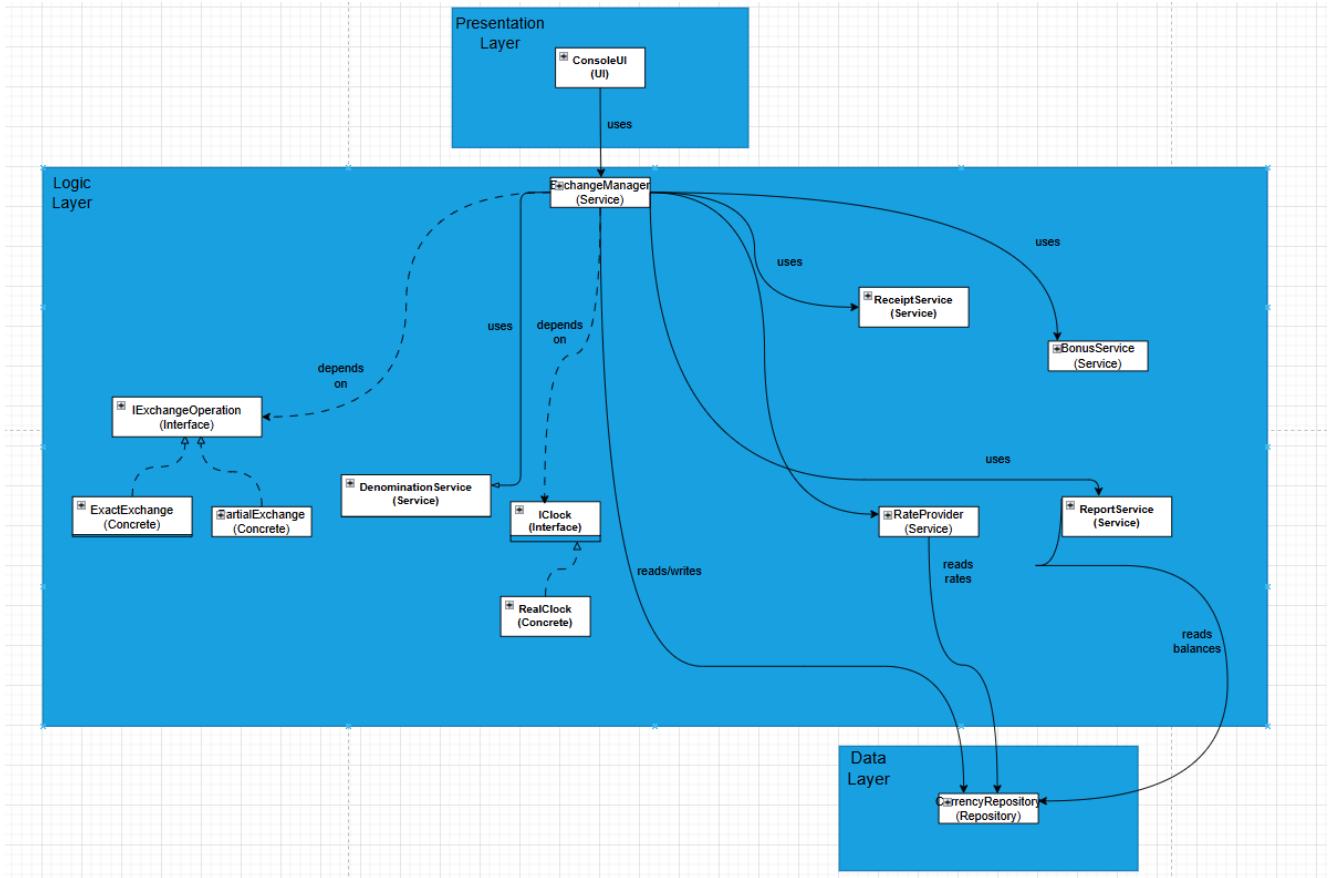
30.10.2025

S/W Detailed Level Design

1. Overview

This DLD describes the technical design of the Currency Exchange System for Release 2. The system is developed mainly for automating currency conversion. The system also checks reserves before exchanging, prints receipts after transactions, records essential transaction data in memory, produces a daily report with balances and profit and calculates the cashier bonus. The document defines architecture, classes, interfaces, method responsibilities, constants, and traceability.

2. System Overview / Architectural Context

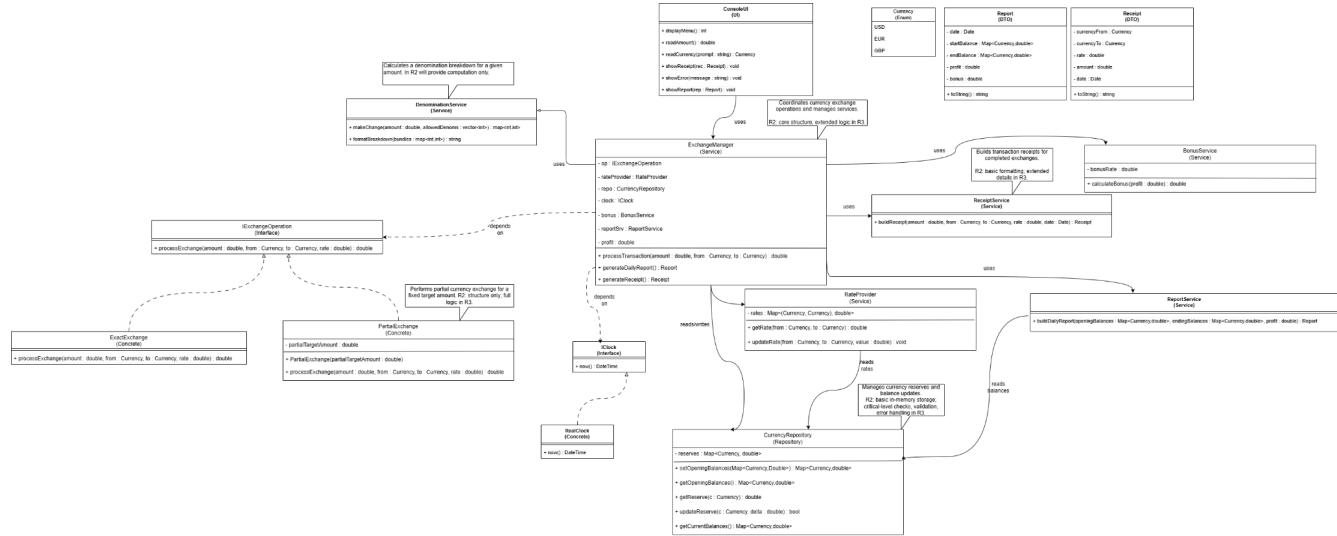


Architecture (3 layers):

- **Presentation / UI:** ConsoleUI reads input, shows results, and formats reports/receipts.
- **Logic / Services:** ExchangeManager coordinates operations; services encapsulate single responsibilities (RateProvider, ReceiptService, ReportService, BonusService, DenominationService). Polymorphism via IExchangeOperation (ExactExchange, PartialExchange).
- **Data:** CurrencyRepository stores balances (in memory for R2).

Dependency direction: ConsoleUI -> ExchangeManager -> Services -> Repository

Simplified: (UI -> Logic -> Data). UI contains no business rules; services contain no console formatting. Rates and balances are in-memory in R2 (file-based storage planned in R3).

S/W Detailed Level Design**3. UML Class Diagram (Technical Design)**[UML Class Diagram Link](#) (for better viewing experience)**4. Class Specifications**

Class	Type	Description	Attributes	Methods
ConsoleUI	UI	Handles all user interaction through the console; displays menus, reads user input, and shows results/reports.	-	<ul style="list-style-type: none"> displayMenu() : int readAmount() : double readCurrency(prompt : string) : Currency showReceipt(rec : Receipt) : void showError(message : string) : void showReport(rep : Report) : void
ExchangeManager	Service	Coordinates exchange operations and manages	op : IExchangeOperation	<ul style="list-style-type: none"> processTransaction(amount : double, from : Currency, to : Currency)

TITLE S/W Detailed Level Design

		<p>services.</p> <p>Retrieves rates, selects operation strategy, updates reserves, generates receipts and reports.</p>	<p>rateProvider : RateProvider</p> <p>repo : CurrencyRepository</p> <p>reportSrv : ReportService</p> <p>clock : IClock</p> <p>bonus : BonusService</p> <p>profit : double</p>	<p>Currency) : double</p> <p>generateDailyReport() : Report</p> <p>generateReceipt() : Receipt</p>
RateProvider	Service	<p>Provides and updates exchange rates between currencies.</p> <p>Stores rates in memory.</p>	<p>rates : Map<(Currency, Currency), double></p>	<p>getRate(from : Currency, to : Currency) : double</p> <p>updateRate(from : Currency, to : Currency, value : double) : void</p>
CurrencyRepository	Repository	<p>Manages currency reserves and handles add/remove operations for balances.</p>	<p>reserves : Map<Currency, double></p>	<p>setOpeningBalances(Map<Currency, double>) : Map<Currency, double></p> <p>getReserve(c : Currency) : double</p> <p>updateReserve(c : Currency, delta : double) : bool</p> <p>getCurrentBalances() : Map<Currency, double></p> <p>getOpeningBalances() :</p>

S/W Detailed Level Design

				Map<Currency, double>
DenominationService	Service	Calculates breakdown of an amount into available denominations; provides formatted text output for the breakdown.	-	makeChange(amount : double, allowedDenoms : vector<int>) : map<int, int> formatBreakdown(bundles : map<int,int>) : string
IExchangeOperation	Interface	Common interface for exchange operation strategies (exact or partial).	-	processExchange(amount : double, from : Currency, to : Currency, rate : double) : double
ExactExchange	Concrete	Performs exchange using allowed denominations for full payout; returns nearest-lower if exact not possible.	-	processExchange(amount : double, from : Currency, to : Currency, rate : double) : double
PartialExchange	Concrete	Performs partial exchange to a fixed target amount and returns remaining balance in local currency.	partialTargetAmount : double	PartialExchange(partialTargetAmount : double) processExchange(amount : double, from : Currency, to : Currency, rate : double) : double
IClock	Interface	Abstract time provider for generating timestamps in	-	now() : DateTime

TITLE S/W Detailed Level Design

		reports and receipts.		
RealClock	Concrete	Actual time provider using system time. Implements IClock.	-	now() : DateTime
Report	DTO	Data Transfer Object representing a summary of daily transactions.	date : Date startBalance : Map<Currency, double> endBalance : Map<Currency, double> profit : double bonus : double	toString() : string
Receipt	DTO	Data Transfer Object storing receipt data for a completed transaction.	currencyFrom : Currency currencyTo : Currency rate : double amount : double date : Date	toString() : string
ReceiptService	Service	Builds transaction receipts for completed exchanges and returns formatted string representation.	-	buildReceipt(amount : double, from : Currency, to : Currency, rate : double) : Receipt
ReportService	Service	Builds Report objects containing	-	buildDailyReport(openingBalances : Map<Currency, doubl

S/W Detailed Level Design

		summarized daily data. Used by ExchangeManager.		le>, endingBalances : Map<Currency,double>, profit : double) : Report
BonusService	Service	Calculates employee bonuses based on daily profit.	bonusRate : double	calculateBonus(profit : double) : double
Currency	Enum	Represents supported currencies in the system.	USD EUR GBP	-

5. Interfaces and Abstractions

Interface	Purpose	Key Methods	Planned For (Release)
IExchangeOperation	Strategy abstraction for computing payouts (exact denominations or partial split).	processExchange(amount, from, to, rate) : ExchangeResult	2
IClock	Decouple system time for testing/reporting.	now() : DateTime	2
IRateProvider (optional)	Abstraction for rates when switching from in-memory to file.	getRate(from,to) : double	3
IFileService (optional)	Abstraction for persistence of receipts/tx logs.	appendReceipt(receipt)	3
IFeePolicy (optional)	Support for different fee rules (no fee now, configurable later).	computeFee(amountIn, from, to) - double	3

TITLE S/W Detailed Level Design

6. Function Responsibilities

Class	Method	Purpose	Input	Output	Notes
ExchangeManager	processTransaction	Coordinate rate retrieval, execute exchange operation, update reserves and profit.	amount, from, to	double	Core control flow uses RateProvider, IExchangeOperation, CurrencyRepository, BonusService. Applies the constant fee to the converted amount and updates profit. Validation of amount > 0 and reserve sufficiency in later release.
RateProvider	getRate	Return current exchange rate for currency pair.	from, to	double	Rates stored in memory for R2.
CurrencyRepository	updateReserve	Modify currency reserve after transaction.	currency, delta	bool	Updates in-memory balances. Validation to prevent negative reserve later.
CurrencyRepository	getOpening Balances	Provide starting reserves for reporting.	-	map<Currency, double>	Used by ReportService in daily report generation.

S/W Detailed Level Design

IExchangeOperation/ExactExchange	processExchange	Compute the converted amount when using full exchange mode.	amount, from, to, rate	double	Implements IExchangeOperation; applies full conversion.
IExchangeOperation/PartialExchange	processExchange	Compute conversion for partial target amount.	amount, from, to, rate	double	Implements IExchangeOperation; used when user requests only partial amount to be exchanged.
DenominationService	makeChange	Produce denomination breakdown for converted amount.	amount, allowed Denoms	map<int ,int>	Breaks exchange result into allowed denominations.
BonusService	calculateBonus	Calculate daily bonus from total profit.	profit	double	Uses constant bonusRate; validation of profit ≥ 0 later.
ReportService	buildDailyReport	Create report object with balances, profit, and bonus.	opening Balances, sending Balances, profit	Report	For release 2, formatting delegated to UI.
ReceiptService	buildReceipt	Build receipt object from transaction data.	from, to, amount, rate, date	Receipt	Used to confirm operation. Formatting done in UI.
ConsoleUI	displayMenu	Show available operations to user and records mode choice	-	int	Presentation only; passes user's choice to ExchangeManager.

TITLE S/W Detailed Level Design

ConsoleUI	showReceipt	Display exchange results.	rec	-	Output only
IClock/Real Clock	now	Provide a timestamp	-	DateTime	Used for report and receipt date stamping

7. Operation Flow

Step 1 - User Interaction

The user starts the process in **ConsoleUI** by choosing:

- the mode (partial exchange, exact exchange, view report, exit)
- the total amount to exchange,
- the source and target currencies (in release 3 user could also choose to get only specific denominations)

ConsoleUI displays the main menu, reads inputs, and performs basic syntactic checks (e.g., empty input, non-numeric value).

If “View Report” is selected before any transactions, the system displays an empty report with opening balances

Responsibility: presentation only - no calculations or business logic. (formatting of receipts and reports in release 2 will be also done here)

Step 2 - Delegation to Logic Layer

ConsoleUI transfers control to ExchangeManager::processTransaction(amount, from, to).

It passes the entered data and the selected mode.

This marks the transition from the Presentation layer to the Logic layer.

S/W Detailed Level Design

Step 3 - Rate Retrieval

ExchangeManager requests the current rate from

RateProvider::getRate(from, to).

The provider returns the value from its in-memory rate map.

In Release 3, rates will be loaded from persistent storage.

Step 4 - Select and Execute Exchange Mode

ExchangeManager decides which implementation of IExchangeOperation to use based on the mode received from ConsoleUI:

- **ExactExchange** - will convert the full amount,
- **PartialExchange** - will convert only the requested partial amount; remainder stays in source currency.

ExchangeManager calls

op->processExchange(amount, from, to, rate)

and receives the converted value.

Step 5 - Update Reserves

After conversion, ExchangeManager updates in-memory balances using

CurrencyRepository::updateReserve(currency, delta).

This records the new amounts for both currencies.

Reserve validation will be done in Release 3.

Step 6 - Compute Profit and Bonus

ExchangeManager calculates the transaction profit from the applied fee (profit += convertedAmount × EXCHANGE_FEE) and calls BonusService::calculateBonus(profit) to determine the cashier's daily bonus.

Both profit and bonus are accumulated in the session memory for release 2.

TITLE S/W Detailed Level Design

Constants: BONUS_RATE = 0.05 (5 %) and EXCHANGE_FEE = 0.02 (2%)

Step 7 - Prepare Denomination Structure (limited in release 2)

ExchangeManager calls

DenominationService::makeChange(amount, allowedDenoms)

to compute the breakdown of the exchanged amount into available denominations.

In Release 2 this service performs computation only; the formatted output and user denomination selection will be implemented in Release 3.

The function formatBreakdown() is defined but not yet integrated into the UI.

Step 8 - Build Receipt and Report

ExchangeManager invokes

ReceiptService::buildReceipt(from, to, amount, rate, date)

and

ReportService::buildDailyReport(balances, profit, bonus).

These services create DTOs (Receipt, Report) that contain structured data only.

Formatting and presentation are deferred to ConsoleUI in release 2.

Step 9 - Display Results

ConsoleUI presents the receipt of the transaction to the user (denomination breakdown will be also shown in a later release)

All formatting (alignment, labels, monetary symbols) is handled in the UI layer.

Error or validation messages (e.g., invalid input, unsupported currency) are also displayed here via showError().

After each completed operation, control returns to ConsoleUI's main menu. The program remains active until the user explicitly selects "Exit".

S/W Detailed Level Design**8. Enumerations & Constants**

Name	Value / Type	Description
EXCHANGE_FEE	0.02 / double	2% fee applied to converted amount
BONUS_RATE	0.05 / double	5% of profit to compute cashier bonus
MIN_RESERVE (optional)	100.0 / double	Minimum per-currency reserve (validation in R3)
SUPPORTED_DENOMS (optional)	vector<int>	Allowed notes/coins per currency (used by DenominationService)
enum class Currency	{ EUR, USD, GBP }	Supported currencies
const int ROUNDING_SCALE	2	Places for money rounding
const char* REPORT_DATE_FMT	"YYYY-MM-DD"	Receipt and report date format

9. Validation Rules & Future Work

Rule / Planned Feature	Description	Target Release
Amount validation	amount > 0; numeric input checks moved from UI to logic.	3
Supported currency	Reject unsupported currency codes; enforce pairs with rates.	3
Positive rate	Ensure rate > 0 for any pair.	3
Reserve sufficiency	Prevent negative balances; enforce MIN_RESERVE.	3
Denomination UI	Let user request specific denominations; integrate formatBreakdown() formatting in UI.	3
Persistence	File I/O for rates, transactions, reports via IFileService.	3

TITLE S/W Detailed Level Design

Fee policy	Replace constant with IFeePolicy strategy.	3
Error handling	Structured exceptions and logging; try/catch around service calls.	3

10. Traceability Matrix

Requirement (SRS)	Class / Method (DLD)
Perform exact currency exchange	ExchangeManager::processTransaction, ExactExchange::processExchange
Perform partial exchange	ExchangeManager::processTransaction, PartialExchange::processExchange
Show daily report	ExchangeManager::generateDailyReport, ReportService::buildDailyReport, ConsoleUI::showReport
Print transaction receipt	ReceiptService::buildReceipt, ConsoleUI::showReceipt
Apply fee and compute bonus	ExchangeManager (fee/profit), BonusService::calculateBonus
Keep balances in memory	CurrencyRepository (get/update)

11. Code Structure and File Mapping

Class / Module	File Names
ConsoleUI	ConsoleUI.h, ConsoleUI.cpp
ExchangeManager	ExchangeManager.h, ExchangeManager.cpp
IExchangeOperation	IExchangeOperation.h
ExactExchange	ExactExchange.h, ExactExchange.cpp
PartialExchange	PartialExchange.h, PartialExchange.cpp

Vladislav Ignatjev

30.10.2025

S/W Detailed Level Design

RateProvider	RateProvider.h, RateProvider.cpp
CurrencyRepository	CurrencyRepository.h, CurrencyRepository.cpp
ReceiptService	ReceiptService.h, ReceiptService.cpp
ReportService	ReportService.h, ReportService.cpp
DenominationService	DenominationService.h, DenominationService.cpp
BonusService	BonusService.h, BonusService.cpp
IClock	IClock.h
RealClock	RealClock.h, RealClock.cpp
Receipt (DTO)	Receipt.h
Report (DTO)	Report.h
Types / constants	types.h, constants.h
main	main.cpp

12. Revision History

Date	Version	Change Summary	Author
2025-09-22	1.0	Initial version	Vladislav Ignatjev
2025-10-30	2.0	Added R2 architecture, UML, class specs, operation flow and mapping.	Vladislav Ignatjev