|  |
| --- |
| Railway Ticket Sales |

|  |
| --- |
| REVISION HISTORY |

|  |  |  |  |
| --- | --- | --- | --- |
| **VERSION** | **DATE** | **DESCRIPTION** | **AUTHOR** |
| 1.0 | 22.09.2025 | Initial Version | **Turks D.** |
| 2.0 | 28.10.2025 | Technical Design and Improved Code | **Turks D.** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |
| --- |
| CONTENTS |

[1 SW System Overview 4](#_Toc209507792)

[1.1 Purpose 4](#_Toc209507793)

[1.2 Scope 4](#_Toc209507794)

[1.3 Use-Case Diagram 4](#_Toc209507795)

[1.4 General Constraints 4](#_Toc209507796)

[1.5 Assumptions and Dependencies 4](#_Toc209507797)

[1.6 Acronyms and Abbreviations 4](#_Toc209507798)

[2 SW Functional Requirements 5](#_Toc209507799)

[2.1 Features / Functions to be Implemented 5](#_Toc209507800)

[2.1 Acceptance Criteria 5](#_Toc209507801)

[2.2 Implementation Requirements 6](#_Toc209507802)

[3 SW Non-Functional Requirements 7](#_Toc209507803)

[3.1 Resource Consumption 7](#_Toc209507804)

[3.2 License Issues 7](#_Toc209507805)

[3.3 Coding Standard 7](#_Toc209507806)

[3.4 Modular Design 7](#_Toc209507807)

[3.5 Reliability 7](#_Toc209507808)

[3.6 Portability 7](#_Toc209507809)

[3.7 General Operational Guidelines 7](#_Toc209507810)

[4 SW Design Artifacts 8](#_Toc209507811)

[4.1 CRC Cards (Class–Responsibility–Collaboration) 8](#_Toc209507812)

[4.2 Conceptual UML Diagram (entities & relationships) 9](#_Toc209507813)

# SW System Overview

## Purpose

The Railway Ticket Sales System aims to automate and streamline ticket purchasing, refund processing, and daily reporting for railway operations. It addresses issues such as double-bookings, manual errors, and inefficient reporting. Intended users include clients, cashiers, and central office managers.

## Scope

1. Included: Host-based, single-user C++ application with ticket search by date, destination, and *optionally* coach type, ticket reservation and release, refund processing with penalty calculation, daily report generation, persistent ticket history storage
2. Excluded: Online payment integration, real database implementation, real ticketing API, multi-user concurrency
3. Benefits: Eliminates double bookings, automates refund penalties, generates accurate reports, provides faster service
4. Key features: Real-time ticket status updates, CLI-based interface, file-based data persistence

## Use-Case Diagram

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

## General Constraints

1. Implementation language: C++
2. Platform: Windows/Linux desktop
3. Data storage: CSV or TXT files
4. Standards: UML notation for diagrams
5. Performance: Ticket operations ≤ 2 seconds
6. User interface: Command Line (CLI)

## Assumptions and Dependencies

* Stable electricity and local PC access
* No internet required
* Local file system available for data storage
* Fixed penalty rules provided by business for penalty calculations

## Acronyms and Abbreviations

|  |  |
| --- | --- |
| **Terms Used** | **Description of terms** |
| **CLI** | Command Line Interface |
| **CSV** | Comma-Separated Values |
| **UML** | Unified Modeling Language |
| **SRS** | Software Requirements Specification |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

# SW Functional Requirements

## 2.1 Features / Functions to be Implemented

***User stories***

1. As a Client, I want to purchase a ticket so that I can travel on my chosen date and destination.
2. As a Client, I want to be able to choose the type of coach I get so I can travel at the level of comfort I need.
3. As a Client, I want to be able to get a refund for a ticket if I cannot travel for any reason.
4. As a Cashier, I want to search for available tickets so that I can provide the client with valid options.
5. As a Cashier, I want to make as little mistakes as possible.
6. As a Central Office Manager, I want daily reports generated automatically so that I can control profit and efficiency without delays.

***Functions to be Implemented as derived from the user stories***

* Search for tickets by date, destination and optionally coach type [1, 2, 4]
* Temporarily reserve tickets during purchase [1, 5]
* Release the tickets if the purchase is cancelled/successful [5]
* Accept client passport information [1]
* Issue tickets to the client [1]
* Automatically calculate penalties on returned tickets and refund the client [3, 5]
* Mark refunded tickets as available right away [1, 5]
* Generate daily reports with profits and list of sold and refunded tickets (to .txt or .csv) [6, 5]
* Persist ticket data and history to files for retrieval on program startup [Supports all user stories indirectly: ensures ticket availability, purchase, returns, and daily reports are accurate after program restarts]

\*[N] - a list of user stories by their numbers that the function supports.

## Acceptance Criteria

* Ticket search must filter by date, destination, coach type and provide valid list of all available tickets according to filtering.
  + Input: Search criteria (date, destination, coach type)
  + Expected output: Only tickets matching all criteria AND with status “Available” are returned
* Reserved tickets are set the “Reserved” status during purchase process.
  + Input: Begin ticket purchase
  + Expected output: Ticket status changes from “Available” to “Reserved” immediately during purchase process
* The system must accept client passport information.
  + Input: Client provides passport data during purchase
  + Expected Output: Client information is correctly recorded and linked to the purchased ticket
* Reserved tickets are released if purchase is cancelled.
  + Input: Cancel purchase
  + Expected Output: Ticket status returns to “Available”
* The system must calculate the refunded amount with an accuracy of two decimal places.
  + Input: Ticket cost and return date (various days before departure)
  + Expected output: Refund amount correctly calculated using 1%, 5%, 10%, 30% penalty rules, rounded to two decimals
* Tickets must be issued to the client after purchase.
  + Input: Complete purchase
  + Expected Output: Ticket status changes to Sold, ticket is assigned to the correct client
* Refunds return money to client balance.
  + Input: Client requests ticket return
  + Expected Output: Client’s balance increases by the correct refunded amount
* Refunds mark the returned ticket as “Available” status.
  + Input: Ticket returned by client
  + Expected Output: Ticket status changes to “Available” in the system
* Refunds mark the correct ticket.
  + Input: Ticket returned by client
  + Expected Output: Only the specific returned ticket is marked Available and refunded; other tickets in the database remain unchanged
* Daily report includes sold/refunded tickets and profit.
  + Input: Generate daily report after several purchases and returns
  + Expected Output: Report accurately lists all sold and refunded tickets with totals and calculated profit
* All ticket operations are logged into a file.
  + Input: Perform ticket purchases and returns
  + Expected Output: Each operation is recorded in the file with timestamp, ticket ID, operation type, and relevant details. Check file exists and matches expected operations.
* Persist ticket data and history to files for retrieval on program startup.
  + Input: Restart program
  + Expected Output: All ticket statuses, client purchases, and operation history are correctly loaded from files

## Implementation Requirements

* All ticket related transactions must be stored in a CSV or TXT file with timestamp, operation type and cost.
* The program must work in console mode (CLI) only.
* UML diagrams must be delivered for use cases, classes, and sequence flows.
* Daily logs must be automatically generated and saved.

# SW Non-Functional Requirements

## Resource Consumption

* Response time for ticket operations (search, purchase, return) must be equal to or less than 3 seconds.
* Maximum memory usage must be equal to or less than 100 MB under normal workload.
* Maximum file size for daily reports must be equal to or less than 5 MB.
* Persistent ticket database (CSV/TXT files) must remain less than or equal to 50 MB to ensure efficient loading at startup.

## License Issues

* Only standard C++ STL libraries are allowed.
* External libraries may only be used if they have permissive open-source licenses (MIT, Apache-2.0).
* No proprietary or closed-source libraries are permitted.

## Coding Standard

* Each class and function must include descriptive comments.
* Unit tests must cover all critical components, including:
  + Ticket status changes (Available 🡪 Reserved 🡪 Sold 🡪 Available)
  + Refund penalty calculation according to the following scheme:
    - 1% if returned one month before the day of travel
    - 5% if returned 15 days before the day of travel
    - 10% if returned 3 days before the day of travel
    - 30% if returned on the day of travel
  + Report generation

## Modular Design

The system shall consist of separate modules for:

* Ticket management
* Client management
* Database persistence
* Reporting

Modules must be designed for low coupling and high cohesion. The design must allow for future extension, e.g., adding new coach types or different refund rules without breaking existing functionality.

## Reliability

* The system must reject invalid input (e.g., invalid passport format, negative balance) without crashing.
* File writes must be atomic to avoid database corruption.
* If a ticket purchase is interrupted, the ticket must revert to Available status.
* All errors must be logged to a dedicated error log file (error\_log\_[timestamp].txt).

## Portability

The system must compile and run on:

* Windows 10+
* Ubuntu Linux

Identical inputs must produce identical outputs across supported platforms. No platform-specific dependencies are allowed.

## General Operational Guidelines

* The system must be robust, easy to use, and simple to maintain.
* A daily reset function must be provided to start each workday with a clean state.
* All ticket operations (purchase, refund) must be logged with timestamps for auditing.
* Data persistence must be ensured: after restart, the system must reload tickets, purchases, and refunds correctly.

# SW Design Artifacts

## CRC Cards (Class–Responsibility–Collaboration)

|  |  |  |
| --- | --- | --- |
| **Class** | **Responsibilities** | **Collaborations** |
| Ticket | Store and get ticket details  Change ticket status | Database  Cashier |
| Database | Store tickets  Find tickets  Reserve tickets  Release tickets  Mark tickets as sold  Make tickets available again after refund | Ticket  Cashier |
| Cashier | Keep operations history  Process ticket purchase  Process ticket return  Process ticket search  Submit end-of-day report | Database  Client |
| Client | Keep list of purchased tickets  Initiate ticket purchase  Initiate ticket return  Initiate ticket search  See list of purchased tickets | Cashier  Ticket |

## Conceptual UML Diagram (entities & relationships)

*A diagram of a company

AI-generated content may be incorrect.*