|  |
| --- |
| Railway Tickets Sales  Software Requirements Specification |

|  |
| --- |
| REVISION HISTORY |

|  |  |  |  |
| --- | --- | --- | --- |
| **VERSION** | **DATE** | **DESCRIPTION** | **AUTHOR** |
| 1.0 | 22.09.2025 | Initial version | **Kamil Błasiak** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |
| --- |
| CONTENTS |

[1 SW System Overview 4](#__RefHeading___Toc800_1679131888)

[1.1 Purpose 4](#__RefHeading___Toc802_1679131888)

[1.2 Scope 4](#__RefHeading___Toc804_1679131888)

[1.3 Use-Case Diagram 4](#__RefHeading___Toc806_1679131888)

[1.4 General Constraints 4](#__RefHeading___Toc808_1679131888)

[1.5 Assumptions and Dependencies 4](#__RefHeading___Toc810_1679131888)

[1.6 Acronyms and Abbreviations 4](#__RefHeading___Toc812_1679131888)

[2 SW Functional Requirements 5](#__RefHeading___Toc814_1679131888)

[2.1 Features / Functions to be Implemented 5](#__RefHeading___Toc816_1679131888)

[2.1 Acceptance Criteria 5](#__RefHeading___Toc818_1679131888)

[2.2 Implementation Requirements 6](#__RefHeading___Toc820_1679131888)

[3 SW Non-Functional Requirements 7](#__RefHeading___Toc822_1679131888)

[3.1 Resource Consumption 7](#__RefHeading___Toc824_1679131888)

[3.2 License Issues 7](#__RefHeading___Toc826_1679131888)

[3.3 Coding Standard 7](#__RefHeading___Toc828_1679131888)

[3.4 Modular Design 7](#__RefHeading___Toc830_1679131888)

[3.5 Reliability 7](#__RefHeading___Toc832_1679131888)

[3.6 Portability 7](#__RefHeading___Toc834_1679131888)

[3.7 General Operational Guidelines 7](#__RefHeading___Toc836_1679131888)

[4 SW Design Artifacts 8](#__RefHeading___Toc838_1679131888)

[4.1 CRC Cards (Class–Responsibility–Collaboration) 8](#__RefHeading___Toc840_1679131888)

[4.2 Conceptual UML Diagram (entities & relationships) 8](#__RefHeading___Toc842_1679131888)

[4.3 Tracability table 9](#__RefHeading___Toc842_1679131888_kopia_1)

# SW System Overview

This SRS describes the requirements for a software system that supports railway tickets sales operations

in a local railway office. The system will automate cashier tasks, including searching for possible routes, blocking the tickets, and allowing to purchase them after entering passenger details. The primary goal is to reduce errors, speed up customer service, prevent double sales, and simplify daily reports to the management.

## Purpose

The purpose of this system is to automate railway ticket purchasing.

## Scope

1. Included: checking available connections, ticket sales, prevention of double booking, generating cashier daily reports, ticket returns

2. Excluded: online banking, multi-branch synchronization, tax reporting.

3. Benefits: reduced cashier errors, faster service, easier reporting.

4. Key features: automatic calculation, printable ticket, ticket returns

## Use-Case Diagram

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

## General Constraints

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

## Assumptions and Dependencies

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

## Acronyms and Abbreviations

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

|  |  |
| --- | --- |
| **Terms Used** | **Description of terms** |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

# SW Functional Requirements

## 2.1 Features / Functions to be Implemented

User stories:

* As a **cashier**, I want to search for tickets by destination and date so that I can offer them to clients.
* As a **cashier**, I want to block a ticket temporarily so that it is not double-sold.
* As a **cashier**, I want to sell a blocked ticket after entering passenger passport data so that the passenger gets a valid ticket.
* As a **cashier**, I want penalties to be automatically calculated based on days before departure so that refunds are accurate.
* As a cashier, I want a returned ticket to be marked as available again.
* As a **cashier**, I want to generate a daily report of sold and returned tickets so that I can submit it to the central railway office.
* As a passenger, I want to choose the type of couch (sleeper/seater/compartment).
* As a **passenger**, I want to return a purchased ticket so that the I get a refund minus a penalty.

Functional requirements:

* The system shall allow searching for available tickets.
* The system shall allow a choice of the type of coach.
* The system shall allow blocking a ticket for a cashier.
* The system shall allow selling a blocked ticket by entering passenger details.
* The system shall allow returning a sold ticket with automatic penalty calculation.
* The system shall mark returned tickets as available again.
* The system shall generate a daily report of sold and returned tickets.

## Acceptance Criteria

Examples:

- Every ticket must be searchable

- A customer can request a type of coach

- After the transaction begins the ticket appears a blocked.

- Blocked ticket can be either sold or released by a cashier; a ticket can’t be sold without passenger details.

- Penalty is automatically calculated when a passenger returns a ticket.

- Returned tickets are marked as available.

- A report is possible to be generated by a cashier.

## Implementation Requirements

- All transactions must be stored in a CSV file with timestamp, customer details, cashier details, and

the ticket cost.

- The program must work in console mode (CLI) only.

- UML diagrams must be delivered for use cases, classes, and sequence flows.

# SW Non-Functional Requirements

## Resource Consumption

Resource Consumption (Sample)

- Response time for any operation: ≤ 2 seconds

- Maximum memory usage: ≤ 100 MB

- Maximum file size for daily logs: ≤ 5 MB

## License Issues

License Issues (Sample)

- 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, Apache-

2.0).

## Coding Standard

Coding Standard (Sample)

- Each function and class must include descriptive comments.

- Unit tests must cover all critical components (e.g., calculation of exchanged amount).

## Modular Design

Modular Design:

- The system shall consist of separate modules for:

• ticket sales

• File logging

• Reporting

• User interaction

- Modules must be designed for low coupling and high cohesion.

## 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.

## Portability

Portability:

- The system must compile and run on Windows 10+ and Ubuntu Linux.

- Identical inputs must produce identical outputs on both platforms.

## 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.

# SW Design Artifacts

## CRC Cards (Class–Responsibility–Collaboration)

|  |  |
| --- | --- |
| **Cashier** | |
| Responsibilities:   * search for available tickets * block tickets * sell blocked tickets * refund tickets * collect passenger details * submit daily reports | Collaborators:   * Passenger * Ticket * Railway\_connection\_database |

|  |  |
| --- | --- |
| **Passenger** | |
| Responsibilities:   * buy tickets * return tickets * choose a couch type * provide personal data to the cashier | Collaborators:   * Cashier |

|  |  |
| --- | --- |
| **Ticket** | |
| Responsibilities:   * be sold * be returned | Collaborators:   * Cashier * Railway\_connection\_database |

|  |  |
| --- | --- |
| **Railway\_connection\_database** | |
| Responsibilities:   * host information about available connections and the amount and type of available tickets | Collaborators:   * Cashier * Ticket |

## Conceptual UML Diagram (entities & relationships)

## Tracability table

|  |  |  |  |
| --- | --- | --- | --- |
| **User story** | **CRC responsibility** | **Simple class diagram element** | **C++ element** |
| Cashier searches for a ticket | Cashier: search for available tickets | Cashier: search\_tickets() | Cashier: search\_tickets() |
| Cashier: block a ticker | Cashier: block ticket | Cashier: block\_ticket() | Cashier: block\_ticket() |
| Cashier: sell a blocked ticket | Cashier: sell blocked ticket | Cashier: sell\_ticket() | Cashier: sell\_ticket() |
| Cashier: penalty amount calculated | Cashier: refund tickets | Cashier: return\_ticket() | Cashier: return\_ticket() |
| Cashier: returned ticket available again | Cashier: refund tickets | Railway\_connection\_database: update\_status() | Railway\_connection\_database: updateStatus() |
| Cashier: generate a report | Cashier: submit daily reports | Cashier: generate\_report() | Cashier: generate\_report() |
| Passenger: choose a coach | Passenger: choose a coach type | Ticket: coach\_type | Ticket: couchType |
| Passenger: get a refund on returned tickets | Cashier: refund tickets | Cashier: return\_ticket() | Cashier: return\_ticket |