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

|  |
| --- |
| REVISION HISTORY |

|  |  |  |  |
| --- | --- | --- | --- |
| **VERSION** | **DATE** | **DESCRIPTION** | **AUTHOR** |
| 1.0 | 22.09.2025 | Initial Version | **Turks D.** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |
| --- |
| CONTENTS |

[1 SW System Overview 4](#_Toc208787546)

[1.1 Purpose 4](#_Toc208787547)

[1.2 Scope 4](#_Toc208787548)

[1.3 Use-Case Diagram 4](#_Toc208787549)

[1.4 General Constraints 4](#_Toc208787550)

[1.5 Assumptions and Dependencies 4](#_Toc208787551)

[1.6 Acronyms and Abbreviations 4](#_Toc208787552)

[2 SW Functional Requirements 5](#_Toc208787553)

[2.1 Features / Functions to be Implemented 5](#_Toc208787554)

[2.1 Acceptance Criteria 5](#_Toc208787555)

[2.2 Implementation Requirements 5](#_Toc208787556)

[3 SW Non-Functional Requirements 6](#_Toc208787557)

[3.1 Resource Consumption 6](#_Toc208787558)

[3.2 License Issues 6](#_Toc208787559)

[3.3 Coding Standard 6](#_Toc208787560)

[3.4 Modular Design 6](#_Toc208787561)

[3.5 Reliability 6](#_Toc208787562)

[3.6 Portability 6](#_Toc208787563)

[3.7 General Operational Guidelines 6](#_Toc208787564)

[4 SW Design Artifacts 7](#_Toc208787565)

[4.1 CRC Cards (Class–Responsibility–Collaboration) 7](#_Toc208787566)

[4.2 Conceptual UML Diagram (entities & relationships) 7](#_Toc208787567)

# SW System Overview

*Specify the purpose and the overview of the SRS.*

## Purpose

*Describe the purpose of the system. What problem does it solve? Who are the intended users? Why is it being developed?*

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

*Define the scope of the system. What functionality is included? What is explicitly excluded? Mention benefits and key features.*

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, 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

*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

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

*This means that instead of listing abstract features, you first describe how users interact with the system and what goals they achieve.*

*User Stories – short, simple descriptions of a feature told from the perspective of the user (e.g., “As a registered user, I want to reset my password so that I can regain access to my account.”).*

*Use Cases – structured scenarios that describe interactions between actors and the system, including preconditions, steps, and outcomes.*

*From these stories/cases, you can then identify:*

* *User interactions (e.g., authentication, profile management).*
* *Business processes (e.g., order processing, reporting).*
* *Integrations (e.g., with external APIs or third-party systems).*
* *System logic (e.g., validation, workflows, automation).*
* *Algorithms (if required, e.g., recommendation or prediction).*

*Each function must be traceable back to a User Story or Use Case, ensuring that the system is built strictly according to user and business needs.*

## Acceptance Criteria

*Define how each requirement will be validated: test cases, acceptance tests, or quality metrics.*

## Implementation Requirements

*Provide details of specific implementation requirements if applicable. For example, integration with existing systems, supported platforms, or algorithms.*

# SW Non-Functional Requirements

## Resource Consumption

*Specify performance and resource limits (CPU, memory, storage, response time).*

## License Issues

*State licensing requirements and constraints on third-party software or libraries.*

## Coding Standard

*Define coding style and standards that must be followed.*

## Modular Design

*Specify architectural requirements such as modularity, extensibility, and maintainability.*

## Reliability

*Define requirements for reliability, error handling, and fault tolerance.*

## Portability

*List target platforms and environments where the system should operate.*

## General Operational Guidelines

*Provide guidelines for scalability, robustness, ease of use, and maintainability.*

# SW Design Artifacts

## CRC Cards (Class–Responsibility–Collaboration)

*List the main classes with their responsibilities (action verbs) and collaborators (related classes); keep items concise and implementation-agnostic.*

## 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.*