Grad Laurentiu-Calin
30435

Bookshop Management System
Supplementary Specification

Version 1.0

Revision History

Date	Version	Description	Author
25/05/2025	1.0	This document ensures that all non-functional requirements and external influences on the system are clearly defined.	Grad Laurentiu-Calin

Table of Contents

Supplementary Specification

1. Introduction

The Supplementary Specification document provides a comprehensive overview of non-functional requirements that are essential for the development and successful deployment of the **Bookshop Management System**. While the primary functional requirements are captured through use cases, this document ensures that critical aspects such as legal standards, quality attributes, system constraints, and environmental dependencies are explicitly defined.

These supplementary requirements address the usability, reliability, performance, and supportability of the system, ensuring that it meets industry standards and fulfills both technical and regulatory obligations. Furthermore, system compatibility, design constraints, and environmental factors play a crucial role in shaping the architecture and overall functionality of the application.

This document serves as a guideline for developers, designers, and stakeholders to ensure that the system aligns with best practices and user expectations while remaining scalable and secure. The following sections elaborate on key supplementary requirements that influence the system's development and operation.

2. Non-functional Requirements

2.1 Availability

Ensure system availability and fault tolerance.

- Source of stimulus: A system component (database or backend) experiences failure.
- Stimulus: A failure occurs while processing a request.
- Environment: The system is running under normal operational conditions.
- Artifact: The affected component tries to recover.
- Response: The system automatically switches to a backup service or retries the operation.
- Response measure: The system should recover within 5 seconds of failure.
- Tactics: Implement redundant servers, auto-recovery mechanisms, and logging for debugging.

2.2 Performance

The system should provide fast response times and handle multiple concurrent users efficiently.

- Source of stimulus: A user performs a search for books.
- Stimulus: The system needs to fetch book details from the database quickly.
- Environment: The system is running under normal load conditions.
- Artifact: The database and backend services process the request.
- Response: The system retrieves the book list and displays it to the user.
- Response measure: The search results should be retrieved in less than 2 seconds.
- Tactics: Use database indexing, caching mechanisms, and optimized queries to enhance performance.

2.3 Security

Protect user data and ensure secure access control.

- Source of stimulus: An unauthorized user attempts to access the admin panel.
- Stimulus: The authentication mechanism identifies an unauthorized access attempt.

- Environment: The system is actively running with multiple users.
- Artifact: The authentication service detects the request.
- Response: The system denies access and logs the event for security monitoring.
- Response measure: Unauthorized access is blocked 100% of the time.
- Tactics: Implement JWT authentication, encrypted password storage, and role-based access control.

2.4 Maintainability

The system should be easy to update and modify.

- Source of stimulus: A developer updates a feature in the codebase.
- Stimulus: A new feature is added to the bookstore system.
- Environment: The system is under active development.
- Artifact: The codebase and development tools process the changes.
- Response: The update is seamlessly deployed without breaking existing functionality.
- Response measure: Updates should be integrated within 1 hour and tested efficiently.
- Tactics: Use modular coding, version control (Git), and automated testing.

2.5 Usability

Users should find the system easy to navigate and understand.

- Source of stimulus: A new user interacts with the system for the first time.
- Stimulus: The user attempts to register and search for books.
- Environment: The system is running under normal conditions.
- Artifact: The user interface and frontend services process the request.
- Response: The system provides clear navigation and intuitive actions.
- Response measure: New users should be able to complete basic tasks within 5 minutes without external help.
- Tactics: Use a clean UI design, tooltips, and guided onboarding features.

3. Design Constraints

The **Bookshop Management System** is subject to several design constraints that dictate the selection of technologies, architectural choices, and development practices. These constraints must be strictly adhered to for consistency, maintainability, and compatibility.

3.1. Software Languages & Frameworks

- The backend must be developed using **Java** and **SpringBoot** for handling business logic and API services.
- The frontend must use **React** with **Vite** to ensure efficient rendering and modern UI design.
- The database management system must be **PostgreSQL**, ensuring structured data storage and integrity.

3.2. Software Process Requirements

- The system development must follow **Agile methodology**, allowing iterative improvements and continuous integration.
- Source code versioning must be maintained using **Git** and hosted on **GitHub**.

- Automated testing should be integrated using **JUnit** for backend logic

3.3. Developmental Tools & Prescribed Use

- **IDE:** Development must be conducted using **IntelliJ IDEA** and **WebStorm** (preferred)
- **Dependency Management: Maven** (for Java) and **NPM** (for JavaScript) must be used to handle libraries and dependencies.
- **Containerization:** The system should be Dockerized for scalable deployment and portability.

3.4. Architectural & Design Constraints

- The system must follow a **layered architecture**, ensuring separation of concerns between presentation, business logic, and data access layers.
- **Dependency Injection (DI)** must be implemented using Spring for modular and maintainable code.
- The application must expose **RESTful APIs**, ensuring compatibility with future integrations.
- The system should support **JWT authentication** to secure user sessions.

3.5. Third-Party Libraries

- The system must integrate with **SMTP email services** for password recovery functionality.