Horseland Resort Web Application

Version <1.0>

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| 26/05/25 | 1.0 | Develop Supplementary Specification for the project | Sofia Dobra |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Contents

1. Introduction 4

2. Non-functional Requirements 4

2.1 Availability 4

2.2 Performance 4

2.3 Security 4

2.4 Testability 4

2.5 Usability 4

3. Design Constraints 4

# Introduction

[The introduction of the **Supplementary Specification** provides an overview of the entire document.

The **Supplementary Specification** captures the system requirements that are not readily captured in the use cases of the use-case model. Such requirements include:

Legal and regulatory requirements, including application standards.

Quality attributes of the system to be built, including usability, reliability, performance, and supportability requirements.

Other requirements such as operating systems and environments, compatibility requirements, and design constraints.]

This Supplementary Specification provides a detailed outline of non-functional requirements and system constraints that are not easily expressed in the use-case model of the Horseland Stud Web Application. It includes quality attributes such as performance, security, and usability, as well as legal, environmental, and technical constraints.

The Horseland Stud Web Application is a multi-role, full-stack system for managing users, horses, and activities within a stud or training facility. This document ensures that all regulatory, technical, and architectural boundaries are clearly defined to guide implementation and maintain consistency across development.

# Non-functional Requirements

*[Define system quality attributes in terms of scenarios according to the following template:*

* *Quality attribute definition*
* *Source of stimulus: the entity (human or another system) that generated the stimulus or event*
* *Stimulus: a condition that determines a reaction of the system*
* *Environment: the current condition of the system when the stimulus arrives*
* *Artifact: is a component that reacts to the stimulus. It may be the whole system or some pieces of it*
* *Response: the activity determined by the arrival of the stimulus*
* *Response measure: the quantifiable indication of the response*
* *Tactics*

*]*

Each quality attribute is expressed as a scenario using the following format:

* **Source of stimulus**: Origin of the trigger (user or system)
* **Stimulus**: What happens to initiate the system's behavior
* **Environment**: Conditions under which the event occurs
* **Artifact**: Part of the system involved
* **Response**: What the system does
* **Response measure**: How the result is evaluated
* **Tactics**: Strategies used to meet the goal

## Availability

* **Definition**: The system must remain accessible to authorized users with minimal downtime.
* **Source of stimulus**: Administrator or user
* **Stimulus**: A request is made to access the system
* **Environment**: Normal operation
* **Artifact**: Backend services and database
* **Response**: System responds to requests and maintains uptime
* **Response measure**: ≥99.5% uptime per month
* **Tactics**: Robust error handling, fallback routes, monitoring, and scheduled backups

## Performance

* **Definition**: The system must provide fast, responsive interactions
* **Source of stimulus**: End user
* **Stimulus**: Navigation through dashboard or entity lists
* **Environment**: Real-time usage
* **Artifact**: REST APIs, database queries
* **Response**: Load and display data quickly
* **Response measure**: ≥90% of responses <500ms
* **Tactics**: Indexed DB fields, DTO usage, efficient filtering and pagination

## Security

* **Definition**: Prevent unauthorized access and ensure secure data handling
* **Source of stimulus**: Any user (authenticated or malicious)
* **Stimulus**: Attempt to access or modify data
* **Environment**: Any operational state
* **Artifact**: Authentication filters, database, API endpoints
* **Response**: Block access if not authorized
* **Response measure**: 100% access control enforcement; encrypted sensitive data
* **Tactics**: JWTs, role-based access control, HTTPS, input validation, encryption (BCrypt, TLS)

## Testability

* **Definition**: Ensure the system can be efficiently tested and verified
* **Source of stimulus**: Developer or CI system
* **Stimulus**: Test cases are run
* **Environment**: Development or staging
* **Artifact**: Application services and database
* **Response**: All tests pass or fail deterministically
* **Response measure**: ≥85% unit test coverage; automated build feedback
* **Tactics**: Unit tests for services, integration tests for controllers, database stubs

## Usability

* **Definition**: End users can navigate and use the system intuitively
* **Source of stimulus**: Instructor, student, admin
* **Stimulus**: User attempts to perform a task (register horse, update activity)
* **Environment**: Web interface (desktop or mobile)
* **Artifact**: UI components and forms
* **Response**: Task is completed with minimal guidance
* **Response measure**: 90% task success rate without help documentation
* **Tactics**: Responsive design, contextual modals, intuitive navigation, consistent theming

# Design Constraints

[This section needs to indicate any design constraints on the system being built. Design constraints represent design decisions that have been mandated and must be adhered to. Examples include software languages, software process requirements, prescribed use of developmental tools, architectural and design constraints, purchased components, class libraries, and so on.]

* The backend must be developed using **Java 21** with **Spring Boot** and **Spring Security**.
* The frontend must be implemented using **React 19** and **TypeScript**, built with **Vite**.
* All backend services must expose **RESTful APIs**.
* **JWT** must be used for authentication, including role-specific token validation and expiration.
* The database must be **MySQL**, with ORM mappings managed via **Spring Data JPA**.
* Unit and integration testing is required across service and controller layers.
* Use of **BCrypt** for password hashing and **TLS (HTTPS)** for transport encryption is mandatory.
* The application must follow a **three-tier architecture**: presentation, logic, and data layers.
* Configuration and secrets must be handled via environment variables or external config files.