



**TechnoReady In-Mexico**

**Challenge 5 – Spring and Spring Boot in Java for Web Applications**

**Iván Kaleb Ramírez Torres**

**Nao ID: 3357**

October 16th, 2025

# Tracking Tables

## Table 1 – Requirements list

|  |  |
| --- | --- |
| Sprint | Requirements |
| Sprint 1:  **Creation of a web project using Spring Boot 3.0.** | 1. Create Spring Boot 3.0 web project (Java ≥17). 2. Implement Order resource connected to DB (H2 dev / PostgreSQL prod). 3. Clear structure, basic comments. 4. README.md with run/config. 5. Upload to GitHub (Digital NAO access). 6. Document key decisions. • CRUD controller & entity. • Postman CRUD tests, exported JSON. • Startup script with env vars. • JavaDoc for public classes/methods. |
| Sprint 2:  **Configuring the project by introducing environment profiles and system variables** | 1. Add profiles: dev, test, prod (YAML configs). 2. Use environment/system variables securely. 3. Include configs in repo. 4. Partial peer reviews & logs. 5. README section for profiles/env vars. 6. GitHub with correct configs (Digital NAO access). |
| Sprint 3:  **Testing API** | 1. OpenAPI/Swagger documentation. 2. Unit & integration tests (success, edge, fail cases). 3. Swagger config integrated. 4. JUnit tests and scripts. 5. Quality checklist before completion. 6. Upload Swagger + code to GitHub. |
| Final Project:  **Document Analysis & Results for the whole project** | * Make a video presentation explaning Analysis & Result of the Challenge 5. |

## Table 2: Prioritize list – Challenge 2

|  |  |  |  |
| --- | --- | --- | --- |
| Requirements | Stages (Steps) | Time Estimation | Deliverables |
| Spring Boot Project Setup | Create project using Spring Boot 3.0 and Java 17+, define structure and add README with usage and repo link. | 2h | GitHub repo with initialized Spring Boot project. |
| Order Entity & Database Configuration | Design Order entity, configure H2 for dev and PostgreSQL for prod. | 4h | Connected database and Order entity mapped. |
| CRUD REST Controller | Implement controller with Create, Read, Update, Delete endpoints. | 3h | Functional CRUD API for Orders. |
| Startup Script & Execution | Develop scripts to start application and configure env variables. | 1h | start-dev.bat / start-dev.sh and usage notes. |
| Postman Collection | Create Postman requests for CRUD operations with examples and comments. | 2h | Postman collection JSON uploaded. |
| Code Documentation (JavaDoc) | Add JavaDoc comments and ensure naming conventions. | 2h | Documented source code. |
| Decision Log | Document key decisions and justifications. | 1h | decision-log.md file in docs folder. |
| Environment Profiles | Create application-dev.yml, application-test.yml, and application-prod.yml files with environment configs. | 3h | Three environment-specific config files. |
| System Variables | Implement and test environment variable loading for sensitive data. | 2h | Secure env variable setup. |
| Partial Peer Reviews | Perform code reviews and document detected issues and fixes. | 2h | peer-reviews.md with findings. |
| Documentation Update | Expand README with profile usage and environment setup instructions. | 1h | Updated README.md. |
| Version Control & Upload | Push updated code and ensure Digital NAO access. | 0.5h | Updated GitHub repo accessible. |
| Environment Profiles | Create application-dev.yml, application-test.yml, and application-prod.yml files with environment configs. | 3h | Three environment-specific config files. |
| OpenAPI / Swagger Integration | Integrate Swagger for API documentation and validation. | 2h | Swagger UI + OpenAPI YAML available. |
| Unit & Integration Testing | Write tests for main API features, including success, edge, and failure cases. | 4h | JUnit test suite uploaded. |
| Quality Checklist | Use a final quality checklist to ensure project completeness. | 1h | quality-checklist.md file. |
| Final Swagger Export | Export Swagger YAML/JSON and document how to access Swagger UI. | 1h | openapi.yaml + Swagger UI URL in README. |
| Analysis & Results Presentation | Prepare slides describing project, results, and architecture. | 2h | Presentation.pdf file in final folder. |
| Video Recording | Record demo of the system showing setup, API usage, and documentation. | 2h | Recording.mp4 in final folder. |
| Final ZIP Packaging | Bundle presentation and video into final ZIP. | 1h | Final\_Submission.zip uploaded. |

As the User Stories was an exercise already made in Challenge 1, All this backlog was made according to Challenge 5 requirements for All 3 Sprints and Final Project.

Spring & Spring Boot Java learning experience for web applications.

Developing and optimizing a web system using spring & spring boot in Java

To achieve this you will

* Create & connect a web project using Spring boot 3.0
* Implement a resource for order creation connected to a database
* Configure profiles and system variables for environment management on the server
* Thoroughly document the develop resource using “Swagger”

Sprint 1:

* Create a basic web application with Spring Boot
* Configure the database connection and develop a core resource for the online store order creation
* Review of the Spring Mood and Java 17 documentation to ensure best practices
* Create and organizer request in Postman from the start to ensure clear documentation
* Automate service deployment using Startup Scripts that configure all necessary components to avoid manual errors

Spint 2:

* Configure application by introducing environment profiles and system variables. This is crucial for deploying and managing the application across different environments.
* Clearly define spring profiles for each environment, development, QA and production.
* Maintain specific and isolated configurations to avoid conflicts.
* Use property files to manage environment variables (This allows you to make quick and safe changes to the configuration without affecting the codebase.

Sprint 3:

* Document the API using API Swagger.
* Perform testing to ensure functionality
* Documentation and testing are essential for proper API validation and use.
* Integrate Swagger from the beginning to keep the API documentation up to date and facilitate automated testing.
* Develop a comprehensive set of tests that cover all API functionalities using tools like Junit to ensure code quality.