Phase 1: Core Spring Boot Foundations – Project 1 SRS

Author: Benjamin Soto-Roberts

Date Created: 12/30/2025

Version: 1.0

Summary/Overview

Purpose: Build proficiency with Spring Boot by creating a simple CRUD application with REST endpoints and database integration. No security, privacy, or compliance requirements in Phase 1.

Outcome: A working Product management app exposing REST APIs and a basic Thymeleaf UI for updating product descriptions, backed by an H2 database.

Scope

In scope:

- Spring Boot REST API for Products (CRUD).
- Spring MVC (Thymeleaf) page for editing product descriptions.
- Persistence using Spring Data JPA with H2.
- Basic DTOs and mapping.
- Basic validation for inputs (e.g., name not blank, price non-negative).

Out of scope (deferred to Phase 2+):

- Authentication/authorization, input sanitization beyond basic validation, audit logging, privacy/compliance controls, deployment hardening.

Objectives/Goals

- Learn controllers, endpoints, and JSON responses.
- Practice calling APIs with RestTemplate or WebClient.

- Understand templating (Thymeleaf) and MVC basics.
- Learn persistence with Spring Data JPA.
- Essentials for secure input handling(to be expanded in Phase 2).

Stakeholders

- Developer: Solo developer (author, primary user and maintainer)
- Reviewers: Any collaborators reviewing code on GitHub
- End users: Demo users interacting with the REST API and the Thymeleaf view

Functional Requirements

FR-1 Product entity

- Create, read, update, delete Product records.
- Fields (minimum): id (Long, auto-generated), name (String), description (String), price (BigDecimal, scale 2).

FR-2 REST API

- Create Product: POST /api/products
- Get Product by id: GET /api/products/{id}
- List Products: GET /api/products
- Update Product: PUT /api/products/{id}
- Delete Product: DELETE /api/products/{id}
- Return JSON responses with appropriate HTTP status codes.

FR-3 Web (Thymeleaf) UI

- List Products: GET /admin/products -> products/list.html
- Edit description form: GET /admin/products/{id}/edit -> products/form.html

- Update submission: POST /admin/products/{id} (update description only in Phase 1)
- Show basic validation errors on the form.

FR-4 DTOs and Validation

- Request DTO for create/update with:
    - name: not blank, max length (e.g., 100)
    - description: optional or max length (e.g., 1000)
    - price: not null, >= 0.00 (BigDecimal)
- Response DTO for returning minimal fields.
- Map DTOs to entities via a mapper (manual or MapStruct).

FR-5 Data Initialization

- Seed sample Products on startup via CommandLineRunner for demo purposes.

Technical Requirements

TR-1 Platform and Frameworks

- Java 21+ (or current LTS)
- Spring Boot 4.x
- Spring Web, Spring Data JPA, Thymeleaf, Validation (jakarta.validation)
- H2 in-memory or file-based database for development/demo

TR-2 Database

- Product table with columns:
    - id BIGINT (PK, auto-increment)
    - name VARCHAR(100) NOT NULL
    - description VARCHAR(1000) NULL

    - price DECIMAL(12,2) NOT NULL
- Use Hibernate DDL auto for schema generation (create-drop or update) for demo.

TR-3 Persistence Layer

- ProductRepository extends JpaRepository<Product, Long>.
- Entity annotated with @Entity/@Table. BigDecimal for price with @Column(precision=12, scale=2).

TR-4 Controllers

- ProductRestController under /api/products produces/consumes application/json.
- ProductWebController under /admin/products returns view names; uses Model for data.

TR-5 DTO Binding and Conversion

- DTO price field is BigDecimal to leverage automatic conversion from JSON numbers or strings.
- Normalize scale server-side: price.setScale(2, HALF_UP).

TR-6 Error Handling

- REST: Return 400 with validation messages on invalid input; 404 when entity not found.
- Web: Re-render form with BindingResult errors.

TR-7 Testing

- Unit tests for service and mapper.
- Slice tests for controllers (@WebMvcTest) and repository (@DataJpaTest).
- Basic integration test using @SpringBootTest with H2.

Non-Functional Requirements (Phase 1)

NFR-1 Usability

- Simple HTML forms and lists with Thymeleaf; no styling requirements.

NFR-2 Performance

- Suitable for small datasets; H2 suffices.

NFR-3 Reliability

- Basic happy-path tests; restart-safe if using file-based H2.

NFR-4 Maintainability

- Clear layering (Controller -> Service -> Repository), DTOs separate from entities.

Assumptions and Constraints

- Single developer; local development environment only.
- No external systems or third-party APIs required.
- No authentication in Phase 1; /admin paths are open during demo.
- Build tool: Maven

Acceptance Criteria

AC-1 Create Product

- Creating a product via POST /api/products returns 201 and a JSON body with id, name, description, price.

AC-2 Get Product by id

- Retrieving a product by id returns 200 with correct data; non-existent id returns 404.

AC-3 Update Product

- Updating a product via PUT persists changes, including price scale normalization to two decimals.

AC-4 Delete Product

- Deleting a product returns 204 and the record no longer appears in listings.

AC-5 Web (Thymeleaf) UI

- Thymeleaf list page displays seeded products; edit page updates description and redirects back to list showing the change.

AC-6 Validation errors

- Blank name → reported as error in both REST (400 with field messages) and UI (form error).
- Negative price → rejected with appropriate message.

Milestones

M-1 Project skeleton

- dependencies, basic app runs with H2.

M-2 Domain + repository

- Product entity, JpaRepository, schema generation.

M-3 DTOs

- mapper + service layer implemented.

M-4 REST

- controller endpoints complete with validation and error handling.

M-5 Thymeleaf

- pages and MVC controller for list/edit description.

M-6 Data

- seeding via CommandLineRunner and smoke tests.

M-7 Tests

- (unit + basic integration) and documentation.

Definitions

REST endpoint - a specific URL that acts as the entry point for an application or service to communicate with a server to access or manipulate resources via HTTP.

Entity - A persistence model class mapped to a database table with JPA annotations.

DTO (Data Transfer Object) - A class used to carry data across layers or over the wire (API), without persistence annotations.

Repository - A Spring Data interface providing CRUD access to entities.

CRUD - Create, Read, Update, Delete operations on resources.

JPA - Java Persistence API, specification for ORM mapping between Java objects and relational databases.

H2 - An in-memory/file-based relational database suited for development/testing.

Thymeleaf - A server-side Java template engine for rendering HTML views in Spring MVC.

CommandLineRunner - A Spring callback interface used to run code at application startup.

Resources

Spring. (n.d.-a). ResponseEntity.

ResponseEntity :: Spring Framework

Spring. (n.d.-b). Building A RESTful Web Service.

Getting Started | Building a RESTful Web Service

Spring. (n.d.-c). Consuming A RESTful Web Service.

Getting Started | Consuming a RESTful Web Service

Spring. (n.d.-d). Serving Web Content with Spring MVC.

Getting Started | Serving Web Content with Spring MVC

Spring. (n.d.-e). Accessing Data with JPA.

Getting Started | Accessing Data with JPA

Spring. (n.d.-f). Validating Form Input.

Getting Started | Validating Form Input