

# Building REST APIs with Spring Boot

## 3.1 Spring Boot Fundamentals

# Learning objectives

---

- ▶ What is Spring Boot and why it's popular
- ▶ How Spring Boot simplifies Java development
- ▶ Understanding the project structure and auto-configuration
- ▶ Exploring the Spring Boot lifecycle
- ▶ Creating and running your first application

# What Is Spring Boot?

- ▶ Framework for building stand-alone, production-ready Spring applications
- ▶ Opinionated: sensible defaults to reduce configuration
- ▶ Built on top of the Spring Framework
- ▶ Embedded servers (Tomcat/Jetty) – no need to deploy WAR files
- ▶ Simplifies dependency management and application setup

# Why Use Spring Boot?

- ▶ Fast project creation with Spring Initializr
- ▶ Auto-configuration – minimal boilerplate
- ▶ Built-in production features (logging, metrics, profiles)
- ▶ Easy integration with Spring modules (Web, Data JPA, Security)
- ▶ REST API-ready out of the box

# Core Features at a Glance

---

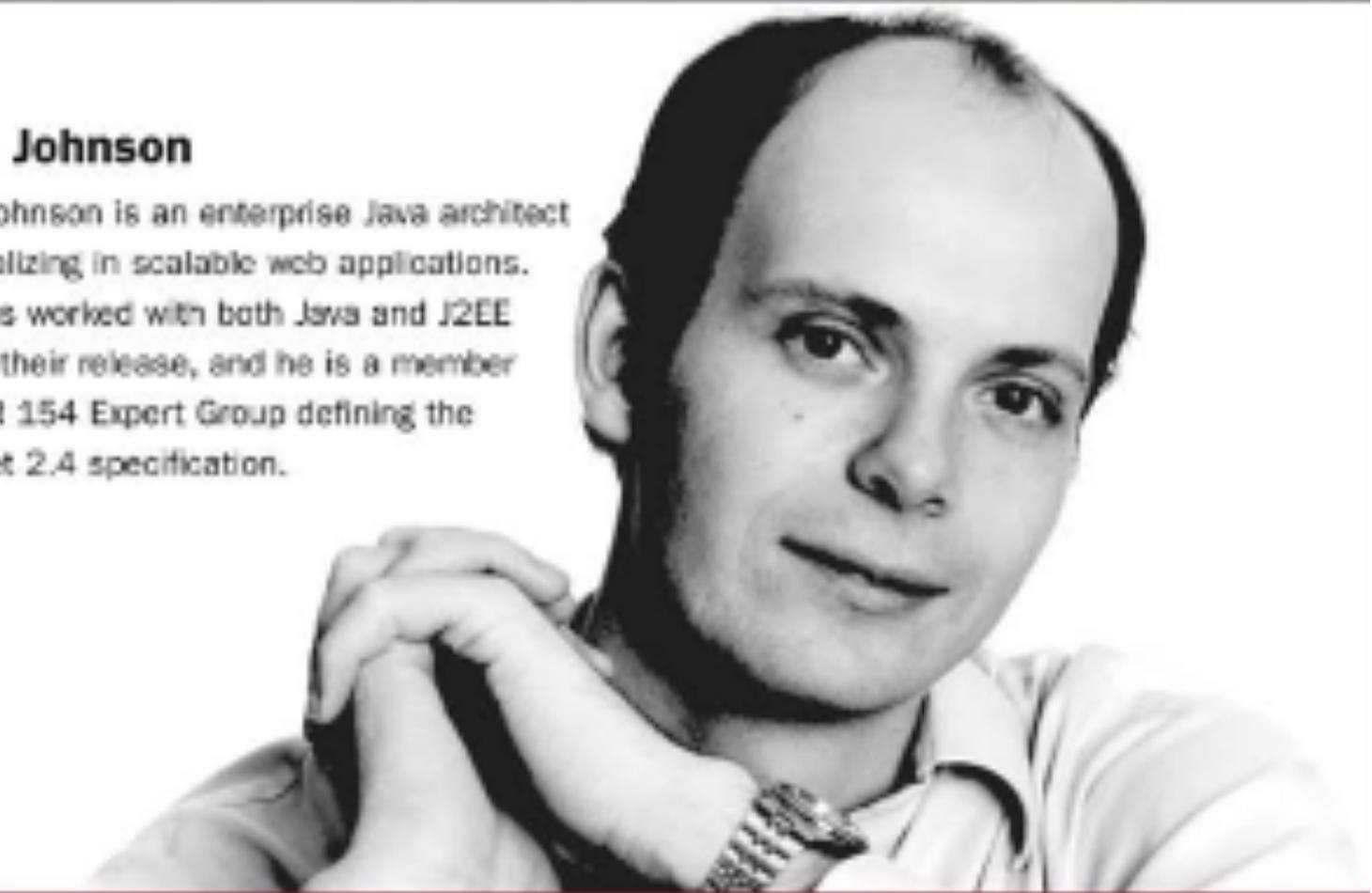
- ▶ Spring Initializr: generates project structure
- ▶ Spring Boot Starter Dependencies: opinionated dependency bundles
- ▶ Auto-Configuration: configures beans automatically
- ▶ Embedded Web Server: runs via `java -jar`
- ▶ Actuator: monitoring and health endpoints
- ▶ Application Properties: central configuration file

# Spring Framework

- ▶ Started out as code in a book by Rod Johnson
- ▶ A light weight alternative to Java EE and EJB
- ▶ Builds on Dependency Injection
- ▶ Spring Framework 1.0 released in 2003

## Rod Johnson

Rod Johnson is an enterprise Java architect specializing in scalable web applications. He has worked with both Java and J2EE since their release, and he is a member of JSR 154 Expert Group defining the Servlet 2.4 specification.



expert one-on-one  
**J2EE™ Design and Development**



Updates, source code, and Wrox technical support at [www.wrox.com](http://www.wrox.com)

# The beginning of Spring Boot



**Rob Winch**  
@rob\_winch

```
@Controller
class ThisWillActuallyRun {
    @RequestMapping("/")
    @ResponseBody
    String home() {
        "Hello World!"
    }
}
```

# Spring Boot vs Spring Framework

- ▶ Conventions
- ▶ Look at the ingredients



Följer

For those on [reddit.com/r/java/](https://www.reddit.com/r/java/) saying it's Spring Boot is “the framework for a framework” here's a diagram:

⌚ Oversatt tweet



# Understanding Spring Initializr

- ▶ Web-based tool for bootstrapping projects

- ▶ Available at: <https://start.spring.io>

- ▶ Choose:

- ▶ Project: Maven

- ▶ Language: Java

- ▶ Spring Boot version

- ▶ Dependencies (e.g., Spring Web)

- ▶ Downloads a ready-to-run zip project

The screenshot shows the Spring Initializr interface with the following configuration:

- Project:** Maven (selected)
- Language:** Java (selected)
- Spring Boot:** 3.5.7 (selected)
- Project Metadata:**
  - Group: com.example
  - Artifact: demo
  - Name: demo
  - Description: Demo project for Spring Boot
  - Package name: com.example.demo
  - Packaging: Jar (selected)
  - Configuration: Properties (selected)
- Java:** 21 (selected)
- Dependencies:** Spring Web (selected)
- Buttons:** GENERATE (highlighted), EXPLORE (CTRL + SPACE), and a three-dot menu button.

# Demo: Creating Your First Spring Boot App

- ▶ 1. Go to <https://start.spring.io>
- ▶ 2. Generate project:
  - ▶ Dependencies: Spring Web
- ▶ 3. Open in IntelliJ.
- ▶ 4. Run the main class `DemoApplication.java`.
- ▶ 5. Confirm startup log:
  - ▶ Tomcat started on port(s): 8080
  - ▶ Started `DemoApplication` in 2.5 seconds
- ▶ 6. Open browser → `http://localhost:8080` → observe empty response (no endpoints yet).
- ▶ "This is a working web server, built in just seconds. Now we'll add our first REST endpoint next."

# Demo: Creating Your First Spring Boot App

---

- ▶ 7. Create a Controller
  - ▶ Create a class named DemoController
  - ▶ Annotate the class with @RestController
  - ▶ Create a method that returns "Hello World!"
  - ▶ Annotate the method with: @GetMapping
- ▶ 8. Open browser → <http://localhost:8080> → observe "Hello World!" Being returned

# Exploring Project Structure

- ▶ `DemoApplication.java` → entry point (`@SpringBootApplication`)
- ▶ `application.properties` → configuration file
- ▶ `static/` → optional web assets (not used yet)
- ▶ `test/` → unit and integration tests

# How Spring Boot Starts

---

- ▶ The main() method calls SpringApplication.run()
- ▶ Bootstraps Spring context and embedded server
- ▶ Scans for beans and configurations automatically
- ▶ Starts HTTP server on default port 8080
- ▶ Loads application.properties

# Understanding Auto-Configuration

- ▶ Spring Boot scans dependencies and configures components automatically
- ▶ Example: adding `spring-boot-starter-web` automatically sets up:
  - ▶ Tomcat web server
  - ▶ JSON converter (Jackson)
  - ▶ Spring MVC dispatcher servlet
- ▶ You can override configurations using `application.properties`

# Configuration with application.properties

- ▶ Each property customizes runtime behavior
- ▶ Changes applied automatically on restart
- ▶ Common examples:
  - ▶ `server.port=8081`
  - ▶ `spring.application.name=demo-app`
  - ▶ `logging.level.org.springframework=INFO`

# Lab: Creating Your First Spring Boot Project

---

- ▶ 1. Go to <https://start.spring.io>
- ▶ Generate a project according to demo instructions and choose:
  - ▶ Project: Maven
  - ▶ Language: Java
  - ▶ Dependencies (e.g., Spring Web)
- ▶ Open the project in IntelliJ and create the Controller
- ▶ Run and test it in a browser → <http://localhost:8080>

# Key Takeaways

---

- ▶ Spring Boot simplifies Spring application setup
- ▶ Projects are created quickly using Spring Initializr
- ▶ Built-in web server lets you run apps as standalone JARs
- ▶ Auto-configuration reduces boilerplate code
- ▶ Properties file allows quick customization
- ▶ Ready to build your first REST controller next