Spring Boot 3 Quick Start

Spring Boot Overview

- **Spring** is a popular **framework** for building **Java applications** as it contains numerous **helper classes** and **annotations**
- **Spring Boot improves** upon **Spring**
  - Makes it **easier to start Spring** development
  - **Minimizes** the amount of **manual configuration**
  - Helps **resolve dependency conflicts** (Maven or Gradle)
  - Provides an **embedded HTTP server** to start quickly
- Relationship to **Spring**
  - Uses **Spring behind the scenes**
  - Makes it **easier to use Spring**
- **Spring Initializr**
  - A website to quickly create a starting Spring Boot project
    - http://start.spring.io
  - You may **select your dependencies**, **create a Maven/Gradle project**, and **import it into your IDE**
- **Spring Boot Embedded Server** → No need to install a separate server
- **Running Spring Boot** Apps
  - Can be run **standalone** (includes embedded server)
  - Can be run from the **IDE** or **command-line**
    - Example: **java -jar sample-app.jar**
- **Deploying Spring Boot Apps**
  - Can also be deployed traditionally
  - Deploy **WAR file** to an external server: Tomcat, JBoss, WebSphere, etc
- **Common Spring Boot Questions**
  - Does **Spring Boot** replace **Spring MVC**, **Spring REST**, etc?
    - No, it instead **uses all** of those technologies
    - **Spring Boot** is all about minimizing configuration
  - Does **Spring Boot** run faster than regular **Spring** code?

        - **No** → We care about minimized configuration, not speed
        - **Spring Boot** uses the **same code** of **Spring Framework**
    - Do I need a special **IDE** for **Spring Boot**?
        - **No** → Any IDE will suffice
        - The **Spring** team provides free **Spring Tool Suite (STS)** plugins

Spring Boot Initialzr Demo

- Quick Word on **Maven**
    - Tell **Maven** the projects you're working with **(dependencies)**
    - It will **download the JAR files for those projects automatically**, making them **available** during **compile-/run-time**
- **Development Process**
    - Configure the project at the **Spring Initializr** website
    - **Download** the **zip file**
    - **Unzip** the file
    - **Import** the project into the **IDE**

Spring Boot - Create a REST Controller

- **Create REST Controller**
    - We set up the REST controller with the **@RestController annotation**
    - We then use a **mapping annotation** with an **endpoint** to **handle HTTP requests**

Spring Framework Overview

- **Goals** of **Spring**
    - **Lightweight development** with **Java POJOs** (**P**lain-**O**ld-**J**ava-**O**bjects)
    - **Dependency injection** to promote **loose coupling**
    - **Minimize boilerplate** Java code
- **Core** Container
    - **Components**
        - Beans
        - Core
        - SpEL

        - Context
    - **Factory** for **creating beans**
    - **Manages bean dependencies**
- **Infrastructure**
    - **Components**
        - AOP → **Aspect Oriented Programming**
        - Aspects
        - Instrumentation
        - Messaging
    - **Add functionality** to **objects declaratively**
        - Logging, security, transactions
    - Java agents to remotely monitor your app with **JMX (Java Management Extension)**
- **Data Access Layer**
    - **Components**
        - JDBC → Reduces JDBC code by 50%
        - ORM
            - **Object to Relational Mapping**
            - Integration with **Hibernate** and **JPA**
        - Transactions
            - Add **transaction support**
            - Makes heavy use of AOP behind the scenes
        - OXM
        - JMS
            - **Java Message Service**
            - For **sending async message**s to a **Message Broker**
            - **Spring** provides helper classes for **JMS**
- **Web Layer**
    - **Components**
        - Servlet
        - WebSocket
        - Web

    - All web-related classes
    - Home of the Spring MVC Framework
- **Test Layer**
    - **Components**
        - Unit
        - Integration
        - Mock
    - Supports TTD → **Test-Driven Development**
    - Mock objects and out-of-container testing

Spring Projects

- **Projects** are additional **Spring modules** built on-top of the core Spring Framework
- You only **use** what you **need**
    - Spring Cloud, Spring Data
    - Spring Batch, Spring Security
    - Spring Web Services, Spring LDAP

What is Maven?

- **Spring Boot** and **Maven relationship**
    - Generating projects via **Spring Initializr** also generates **Maven projects**
- What is **Maven**?
    - A project management tool popular for **build management** and **dependencies**
- **Maven Solution**
    - Tell **Maven** what projects you're working with **(dependencies)**
        - Spring, Hibernate, etc.
    - Maven **automatically downloads** the **JAR files** for those projects for you
    - Maven will make those **JAR files** available during **compile-/run-time** in the build path
- How **Maven** Works

    - You will populate a **Project Config** file with all of your dependencies
    - **Maven** will check a **local repository** for these files
        - If these **files aren't present**, Maven will **pull them** from the **Maven** central repository to the **repository**
    - **Maven** will use these files to build the project
- Handling **JAR Dependencies**
    - When **Maven** retrieves **project dependencies**, it also **downloads supporting dependencies**
- **Building** and **Running**
    - **Maven** handles creating **class/build paths**
    - **Maven** adds **JAR** files according to the **config** file

Maven Project Structure

Directory	Description
pom.xml	Maven **configuration file** containing **dependencies**
src/main/java	Your **Java source code**
src/main/resources	**Properties** / **config files** used by your app
src/main/webapp	**JSP files** and **web config files**; other web assets (images, CSS, JS, etc)
src/test	**Unit testing** code and properties
target	Destination directory for **compiled code**; automatically created by Maven

- Standard Directory Structure **Benefits**
    - Can **easily find** code, properties files, unit tests, web files, etc
    - Most major **IDEs** have **built-in support** for **Maven**
        - IDEs can **easily read/import** Maven projects
    - **Maven** projects are **portable**

        - Developers can **easily share projects between IDEs**
- **Advantages** of **Maven**
    - **Dependency** Management
        - **Maven** will find **JAR** files for you
        - No more missing **JARs**
    - **Building** and **running** your project
        - No more **build/class path** issues
    - **Standard directory structure**

Maven Key Concepts

- **POM** File
    - **P**roject **O**bject **M**odel File
    - **Configuration file** for the project
    - Located in the **root** of your **Maven** project
- **POM** File **Structure**
    - Project **Meta Data**
        - **Project** name, version, etc.
        - Output file type: JAR, WAR, etc.
    - **Dependencies**
        - **List** of **projects** we **depend on**
        - Spring, Hibernate, etc.
    - **Plug-ins**
        - Additional **custom tasks** to run
        - Generate JUnit test reports, etc.
- **Project Coordinates**
    - Uniquely identify a project
    - Elements

Name	Description
Group ID	Name of company, group, or organization Convention is to use reverse domain name: **com.luv2code**

Artifact ID	Name for this project: **mycoolapp**
Version	A specific release version, like: **1.0**, **1.6**, **2.0**, etc. If the project is under active development, then: **1.0-SNAPSHOT**

- **Dependency Coordinates**
  - Adding dependencies requires
    - **Group/Artifact ID**
    - **Version** is **optional** → Best practice to still include versions
  - May be referred to as **GAV** → **G**roup ID, **A**rtifact ID, **V**ersion
- **How to Find Dependency Coordinates**
  - Visit the **project page** (**spring.io**, **hibernate.org**, etc.)
  - Visit **http://search.maven.org** (easiest approach)

Exploring Spring Boot Project Files

- **Maven Standard Directory Structure**

Directory	Description
src/main/java	Your Java source code
src/main/resources	Properties / config files used by your app
src/test/java	Unit testing source code

- **Maven Wrapper** Files
  - **mvnw** allows you to run a **Maven** project
    - **No need** to have **Maven installed** or present on your path
    - If the **correct version** of **Maven** is **not found** on your computer
      - **Automatically downloads** the correct version and **runs Maven**
  - Two files are provided

        - **mvnw.cmd** for MS Windows
        - **mvnw.sh** for Linux/Mac
    - If you already have **Maven installed properly**
        - You can **ignore/delete** the **mvnw** files
    - Just use **Maven** as you normally would
- **Maven POM File**
    - **pom.xml** includes info that you entered at the **Spring Initializr** website
    - Includes **Spring Boot Starters**
        - A collection of compatible **Maven dependencies**
        - **Saves the developer from having to list all** of the individual dependencies
        - Example: **spring-boot-starter-web**
    - **Spring Boot Maven plugin** → To package executable JAR or WAR archive
- Java **Source Code**
    - Main and controllers
- **Application Properties**
    - By default, **Spring Boot** will **load properties** from java/resources/**application.properties**
    - Created by **Spring Initializr** as an empty file
    - Can add **Spring Boot** or **custom properties** here
        - Such as **port** or **attribute-value**
    - **Read data** from **application.properties** via **injection**
        - Example: **@Value("${server.port}")**
- **Static Content**
    - src/main/resources/**static**
    - By default, **Spring Boot** will **load static resources** from **"/static"** directory
    - Examples of static resources include **HTML files, CSS, JavaScript, images, etc.**
    - **Warning**

        - Don't use the **src/main/webapp** directory if the **application** is **packaged** as a **JAR**
        - Only works with **WAR packaging**
- **Templates**
    - **Spring Boot** includes **auto-configuration** for the following **template engines**
        - FreeMarker
        - Thymeleaf
        - Mustache
    - **Spring Boot loads templates** from src/main/resources/**templates**
- **Unit Tests**
    - **Spring Boot unit test class**
        - Created by **Spring Initializr**
        - src/test/java/com/luv2code/springboot/demo/mycoolapp/**MycoolappApplicationTests.java**

Spring Boot Starters

- The **Problem**
    - Building a **Spring** application is difficult
    - **Which Maven dependencies do I need?**
- **Why** is it so **hard**?
    - It would be great if there was a **simple list of Maven dependencies**
    - Collected as a group of dependencies → **one-stop shop**
    - So we don't have to **search for every dependency**
- The **Solution - Spring Boot Starters**
    - A **curated list** of **Maven dependencies**
    - A **collection** of **dependencies** grouped together
    - **Tested** and **verified** by the **Spring Development team**
    - Makes it much **easier** for the developer to get **started** with **Spring**
    - **Reduces** the amount of **Maven configuration**
- **Spring Initializr**
    - If we're building a **Spring** app that needs: **Web, Security, etc.**

    - Simply select the **dependencies** in the **Spring Initializr**
    - It will add the appropriate **Spring Boot** starters to **pom.xml**
- **Spring Boot Starters**

Name	Description
spring-boot-starter-web	**Building web apps**, includes **validation**, **REST**; uses **Tomcat** as default embedded server
spring-boot-starter-security	Adding **Spring security** support
spring-boot-starter-data-jpa	**Spring database** support with **JPA** and **Hibernate**
...	...

- What's in the **Starter**?
    - View the starter's **POM file**
        - Normally **references** other **starters** → will need to further dig
        - Cumbersome
    - Most **IDEs** have a **Dependency Management / View** feature
        - Easier to navigate
        - In **IntelliJ** → **View > Tool Windows > Maven Projects > Dependencies**

Spring Boot Parents for Starters

- **Spring Boot Starter Parent**
    - **Spring Boot** provides a **Starter Parent**
    - This is a special starter that provides **Maven defaults**
        - Default compiler level, UTF-8 source encoding, etc.
    - To **override** a default, **set as a property**
        - Example:

            <properties>

<java.version>17</java.version>

</properties>

- No need to list versions for the **spring-boot-starter-***
- **Default configuration** of **Spring Boot plugin**

- **Benefits** of the **Spring Boot Starter Parent**
  - **Default Maven configuration**: Java version, UTF-encoding, etc
  - **Dependency management**
    - Use version on parent only
    - **spring-boot-starter-*** dependencies inherit version from the parent
  - **Default configuration** of **Spring Boot plugin**

Spring Boot Dev Tools

- The **Problem**
  - When running **Spring Boot** applications
  - Making **changes** to **source code** makes you **manually restart** your **application**
- The **Solution: Spring Boot Dev Tools**
  - **Automatically restarts** the **application** when **code is updated**
  - Simply **add** the **dependency** to your **POM file**
- **Development Process**
  - Apply **IntelliJ configurations**
  - Edit **pom.xml** and add **spring-boot-devtools**
  - Add new **REST endpoint** to our app
  - Verify the app is **automatically reloaded**

Spring Boot Actuator

- The **Problem**
  - How can I **monitor** and **manage** my application?
  - How can I **check** the application's **health**?
  - How can I **access** the application's **metrics**?
- The **Solution: Spring Boot Actuator**
  - **Exposes endpoints** to **monitor** and **manage** your application

    - You easily get **DevOps functionality out-of-the-box**
    - Simply **add** the **dependency** to your **POM** file
    - **REST endpoints** are **automatically added** to your application
- **Endpoints**
    - **Endpoints** are **prefixed** with "**/actuator**"
    - **/health** is **exposed**; **others** must be **exposed manually** in **application.properties**
        - **management.endpoints.web.exposure.include=[comma-delimited list]**
        - **management.info.enb.enabled=true** for **info**

Name	Description
/health	**Health information** about your application; normally used by monitoring apps to see if the **app is up or down**
/info	**Information** about the application; **empty** by **default**; **define** what's given in **application.properties**
/auditevents	**Audit events** for your application
/beans	List of all **beans registered** in the Spring application context
/mappings	List of all **@RequestMapping paths**
/threaddump	List of **all threads running** in your application
...	...

- **Development Process**
    - Edit **pom.xml** and add **spring-boot-starter-actuator**

    - View **actuator endpoints** for **/health**
    - Edit **application.properties** to customize **/info**

Spring Boot Actuator - Securing Endpoints

- What about **security**?
    - You may **not** want to **expose** all of this **information**
    - **Add Spring Security** to the project to **secure endpoints**
- **Secured Endpoints**
    - When used, **Spring Security** will prompt users for **login**
        - Default username: **user**
        - Password: located in **logs**
- **Spring Security configuration**
    - Can override default **username** and **password** in **application.properties**
- **Exclude Endpoints**
    - **management.endpoints.web.exposure.exclude=[comma-delimited list]**
- **Development Process**
    - Edit **pom.xml** and add **spring-boot-starter-security**
    - **Verify security** on actuator endpoints for **/beans**
    - **Disable endpoints** for **/health** and **/info**

Run Spring Boot Apps from the Command Line

- **Running** from the **Command-Line**
    - Since we're using **Spring Boot**, the **server** is **embedded** in our **JAR** file
        - **No need** to have a **separate server installed/running**
        - **JAR file** contains **application code** and the **server**
    - **Spring Boot apps** are **self-contained**
    - **Options**
        - **java -jar [file-name]**
        - **Spring Boot Maven plugin**
            - Package → **mvnw package**

        - **mvnw spring-boot:run [file-name]**
- **Development Process**
    - **Exit** the **IDE**
    - **Package** the app using **mvnw package**
    - **Run** app using **java -jar**
    - **Run** app using **Spring Boot Maven plugin**, **mvnw spring-boot:run**

Injecting Custom Application Properties

- **Problem**
    - You need your app to be **configurable** → **no hard-coding** values
    - You need to read app configuration from a **properties file**
- **Solution: Application Properties file**
    - By default, **Spring Boot** reads information from a **standard properties file**
        - Located at: **src/main/resources/application.properties**
    - You can define any custom properties in this file
    - Your **Spring Boot** app can **access properties** using **@Value**
- **Development Process**
    - **Define** custom properties in **application.properties**
    - **Inject** properties into **Spring Boot application** using **@Value**

Configuring the Spring Boot Server

- **Spring Boot Properties**
    - **Spring Boot** can be configured in the **application.properties** file
    - **Server port**, **context path**, **actuator**, **security**, etc.
- **Property Categories**
    - **Core, Web, Security, Data, Actuator, Integration, DevTools, Testing**
- **Development Process**
    - **Configure** the **server port**
    - **Configure** the application **context path**