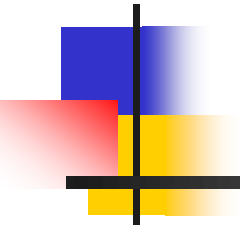
Advanced Application Development (CSE-214)
week-6 : Introduction to SpringBoot



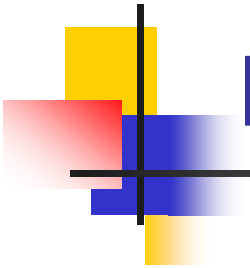
Dr. Alper ÖZCAN
Akdeniz University
alper.ozcan@gmail.com

Main Objective

Main Objective

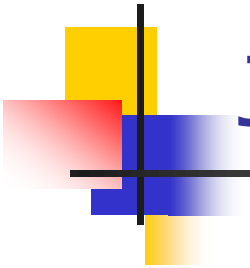
Learn how to build Java applications with

Spring Boot and Hibernate



Main Objective

- Develop Spring Boot applications
- Leverage Hibernate/JPA for database access
- Develop a REST API using Spring Boot
- Create a Spring MVC app with Spring Boot
- Connect Spring Boot apps to a Database for CRUD development



Java Development Environment

You should have the following items already installed

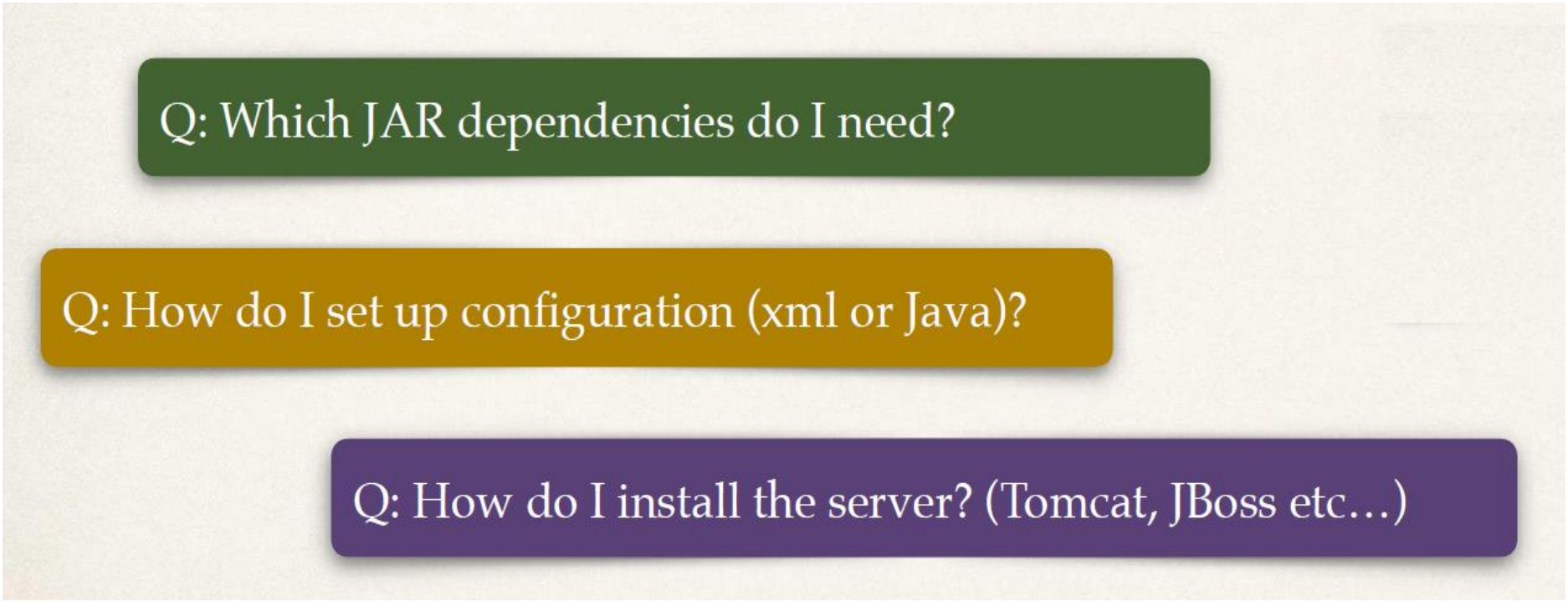
- Java Development Kit (JDK) - Spring Boot 3 requires JDK 17 or higher
- Java IDE (we'll use IntelliJ)
- Download from: **https://www.jetbrains.com/idea/download**

Spring Boot Introduction

- Very popular framework for building Java applications
- Provides a large number of helper classes and annotations

The Problem

Building a traditional Spring application is really HARD

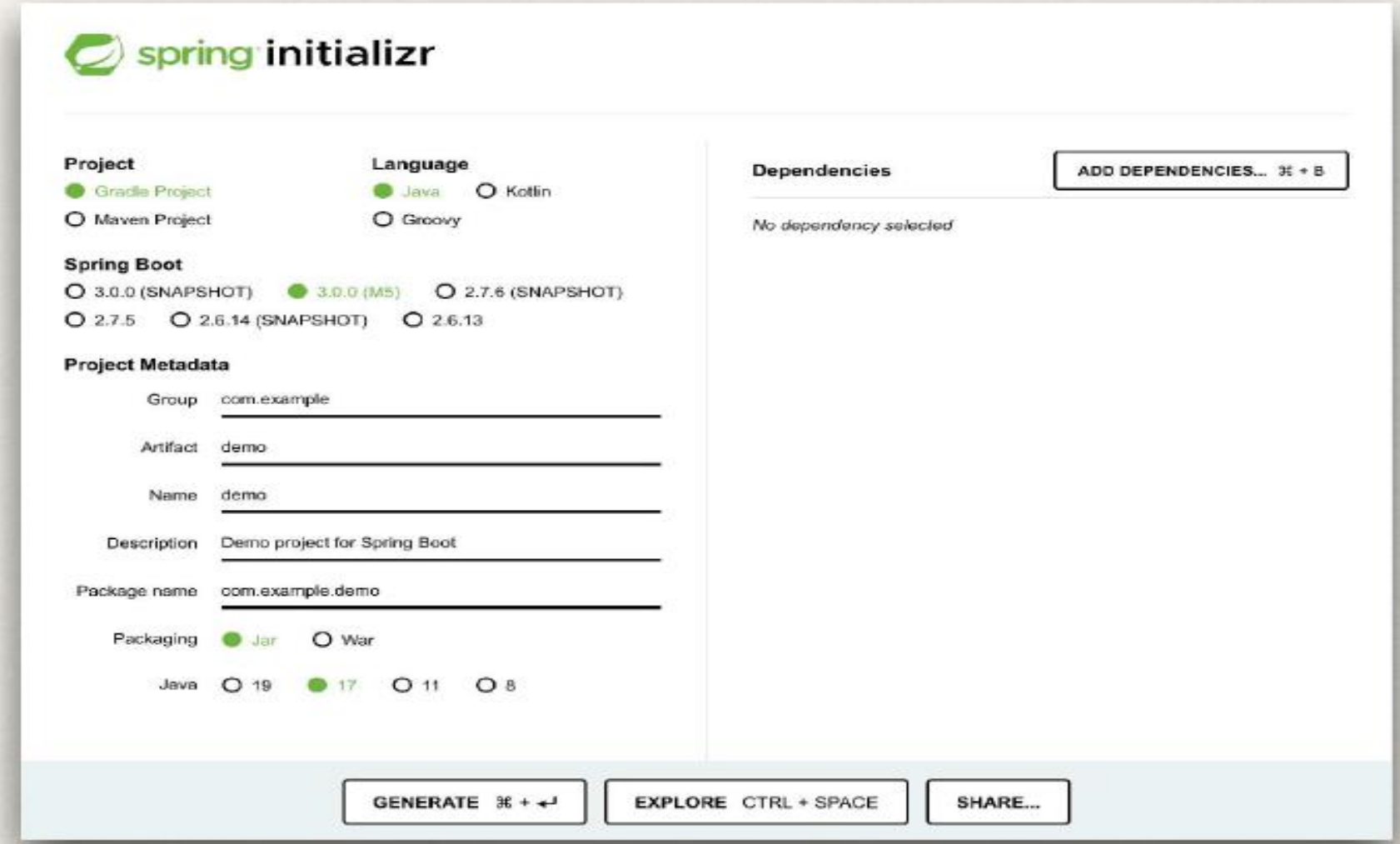


Spring Boot Solution

- Make it easier to get started with Spring development
- Minimize the amount of manual configuration
  - Perform auto-configuration based on props files and JAR classpath
- Help to resolve dependency conflicts (Maven or Gradle)
- Provide an embedded HTTP server so you can get started quickly
  - Tomcat, Jetty, Undertow, …

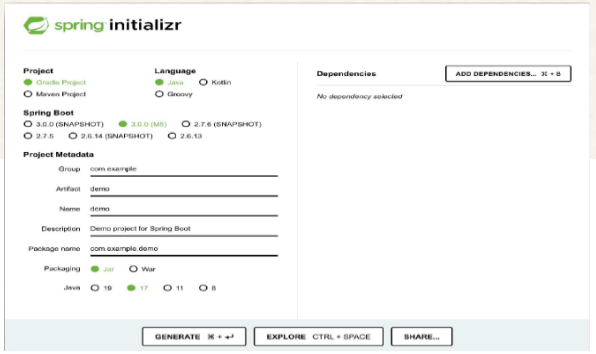
Spring Initializr

http://start.spring.io



Spring Initializr

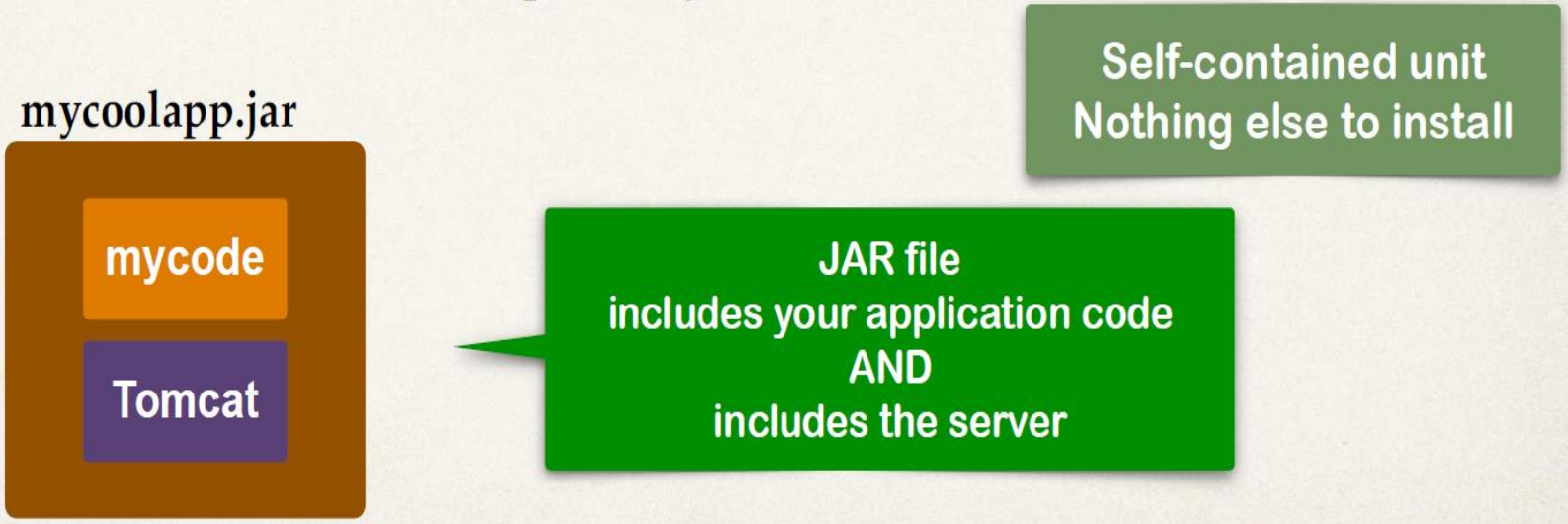
- Quickly create a starter Spring Boot project
- Select your dependencies
- Creates a Maven/Gradle project
- Import the project into your IDE
  - Eclipse, IntelliJ, NetBeans



http://start.spring.io

Spring Boot Embedded Server

- Provide an embedded HTTP server so you can get started quickly
  - Tomcat, Jetty, Undertow, …
- No need to install a server separately

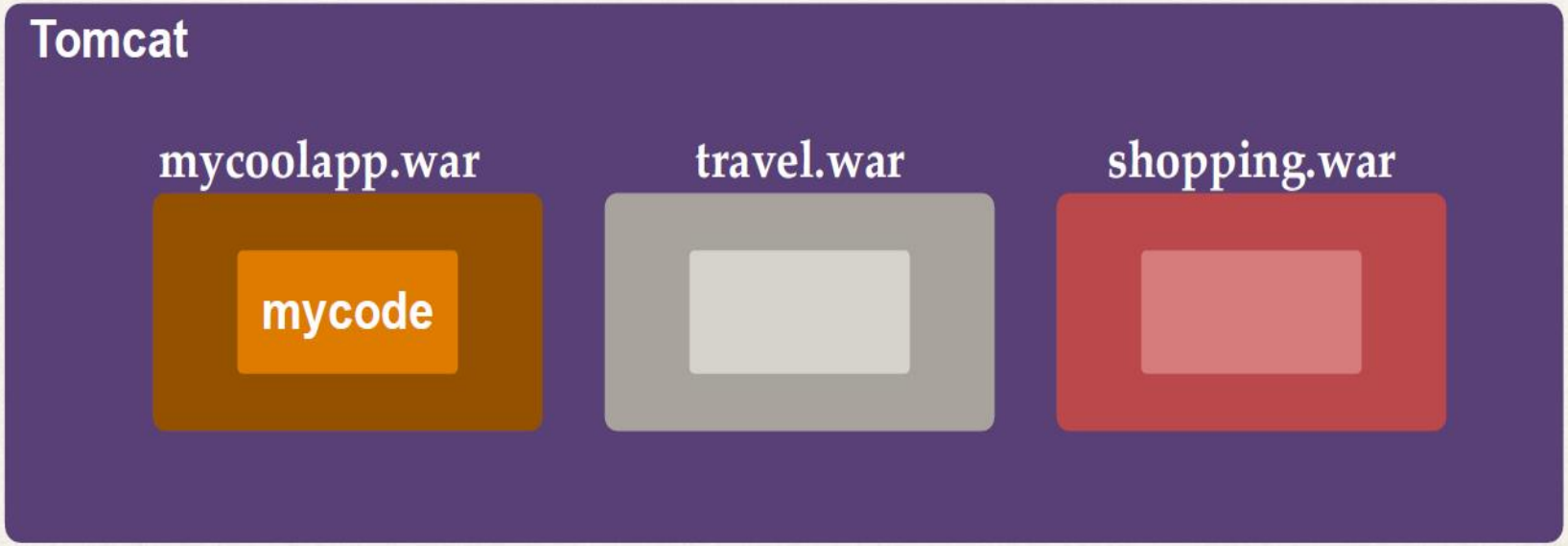


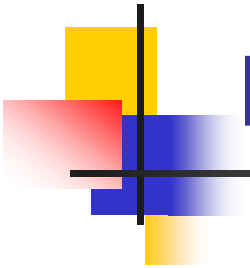
Running Spring Boot Apps

- Spring Boot apps can be run standalone (includes embedded server)
- Run the Spring Boot app from the IDE or command-line

Deploying Spring Boot Apps

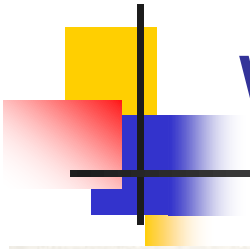
- Spring Boot apps can also be deployed in the traditional way
- Deploy **Web Application Archive (WAR) file** to an external server:
  - Tomcat, JBoss, WebSphere etc …





Maven

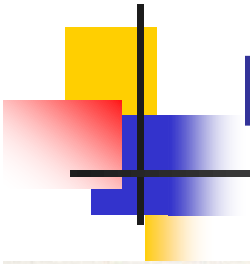
- When building your Java project, you may need additional JAR files
  - For example: Spring, Hibernate, Commons Logging, JSON etc…
- One approach is to download the JAR files from each project web site
- Manually add the JAR files to your build path / classpath



What is Maven?

- Maven is a Project Management tool
- Most popular use of Maven is for build management and dependencies

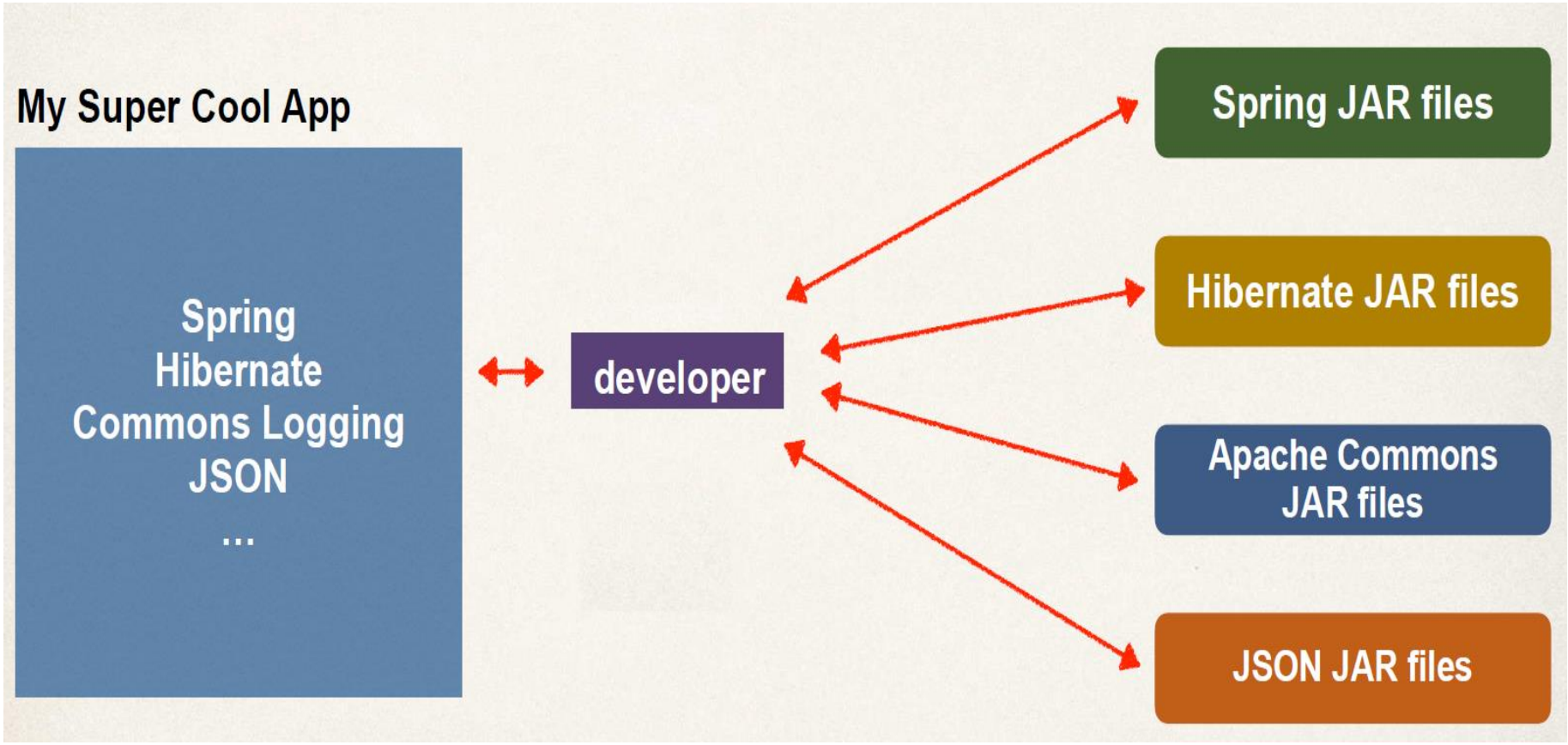
- When you generate projects using Spring Initializr: start.spring.io
  - It can generate a Maven project for you
  - Viewing dependencies in the Maven pom.xml file
  - Spring Boot Starters for Maven



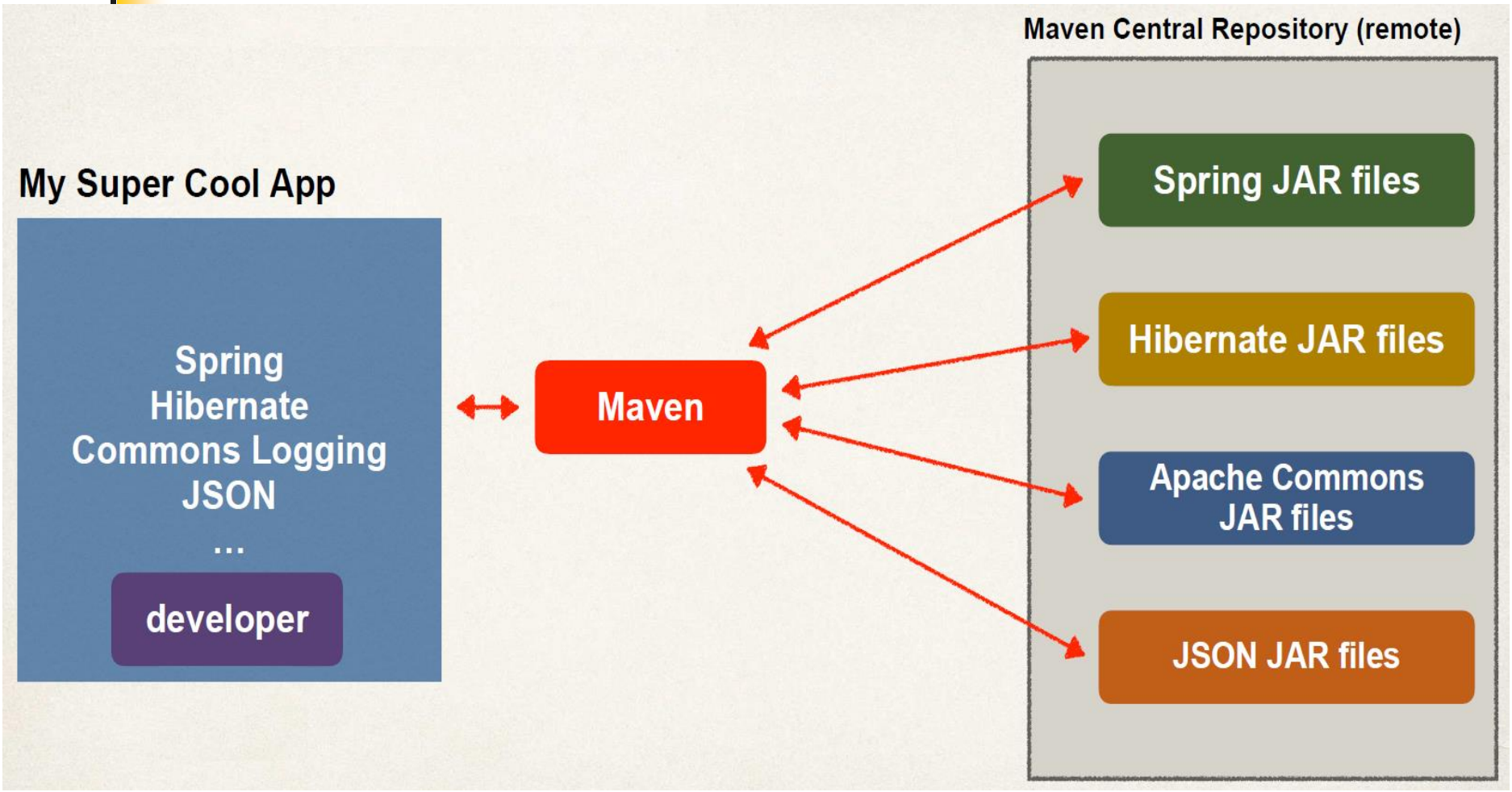
Maven Solution

- Tell Maven the projects you are working with (dependencies)
  - Spring, Hibernate etc ….
- Maven will go out and download the JAR files for those projects for you
- And Maven will make those JAR files available during compile/run
- Think of Maven as your friendly helper

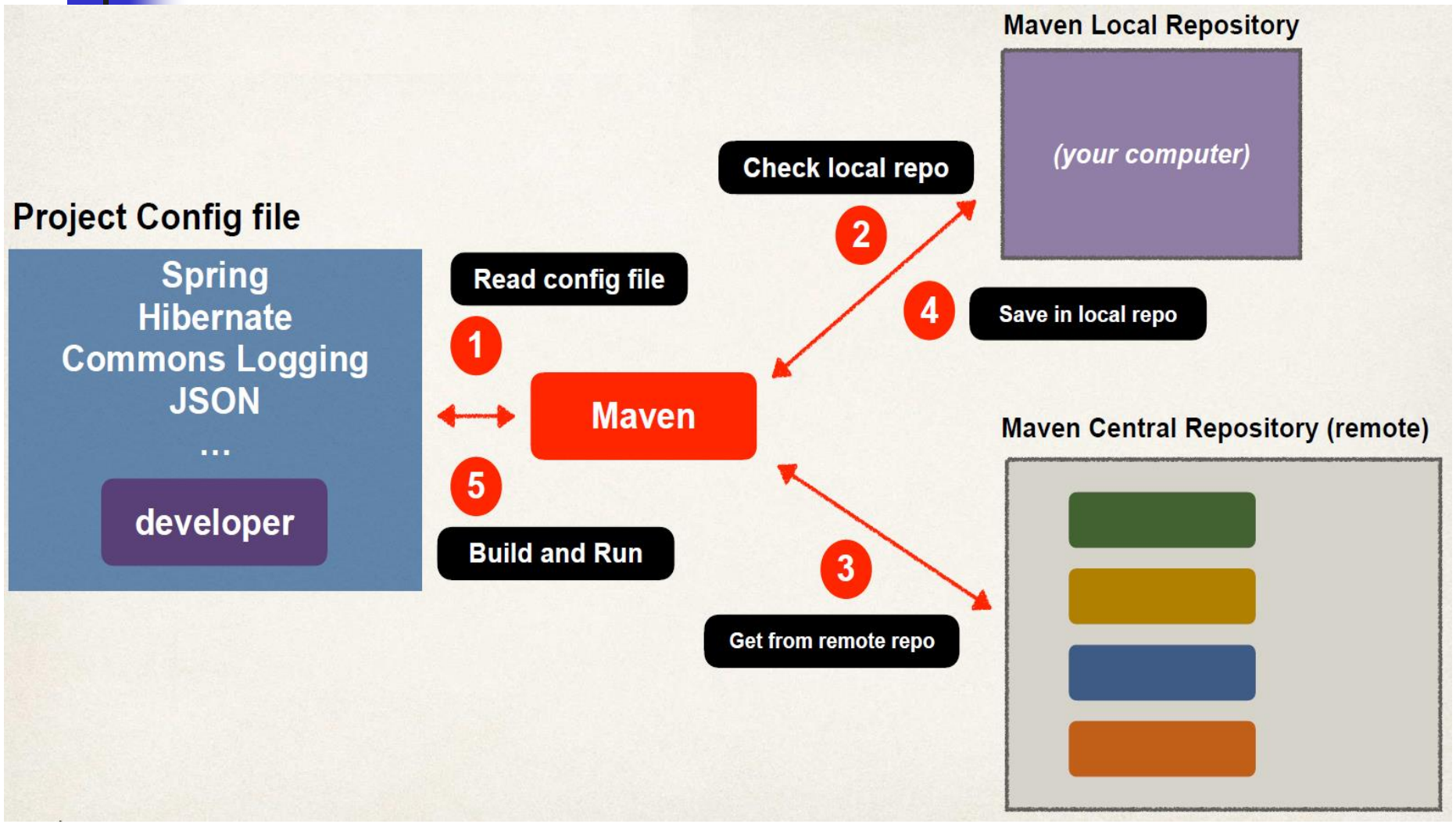
Project without Maven



Project with Maven



Maven - How It Works



Development Process

1. Configure our project at Spring Initializr website

   http://start.spring.io

2. Download the zip file

3. Unzip the file

4. Import the project into our IDE

Spring Boot - Create REST Controller

- Let's create a very simple REST Controller

Create REST Controller

Set up rest controller

```
@RestController
public class FunRestController {

  // expose "/" that returns "Hello World"

  @GetMapping("/")
  public String sayHello() {
    return "Hello World!";
  }

}
```

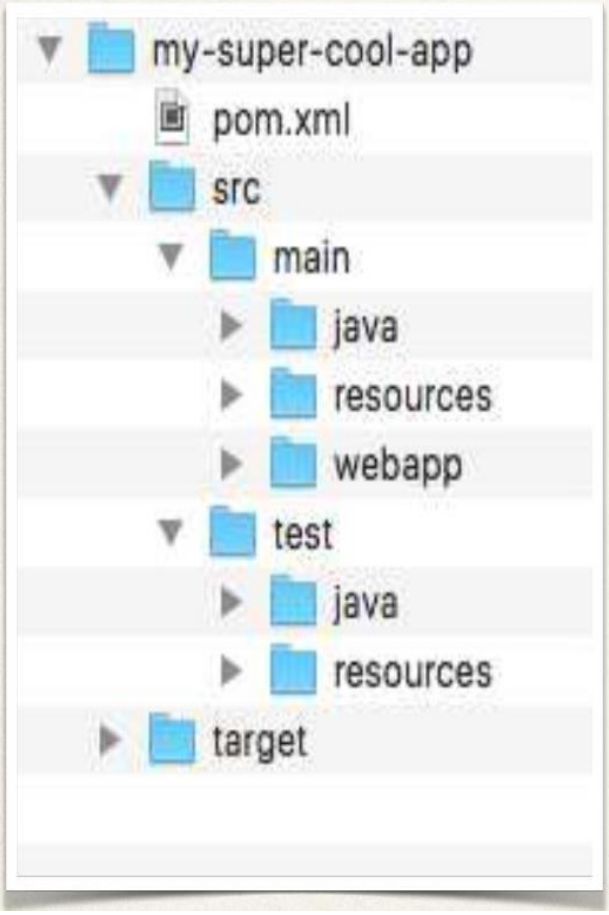
Handle HTTP GET requests

Handling JAR Dependencies

- When Maven retrieves a project dependency
  - It will also download supporting dependencies
  - For example: Spring depends on commons-logging …

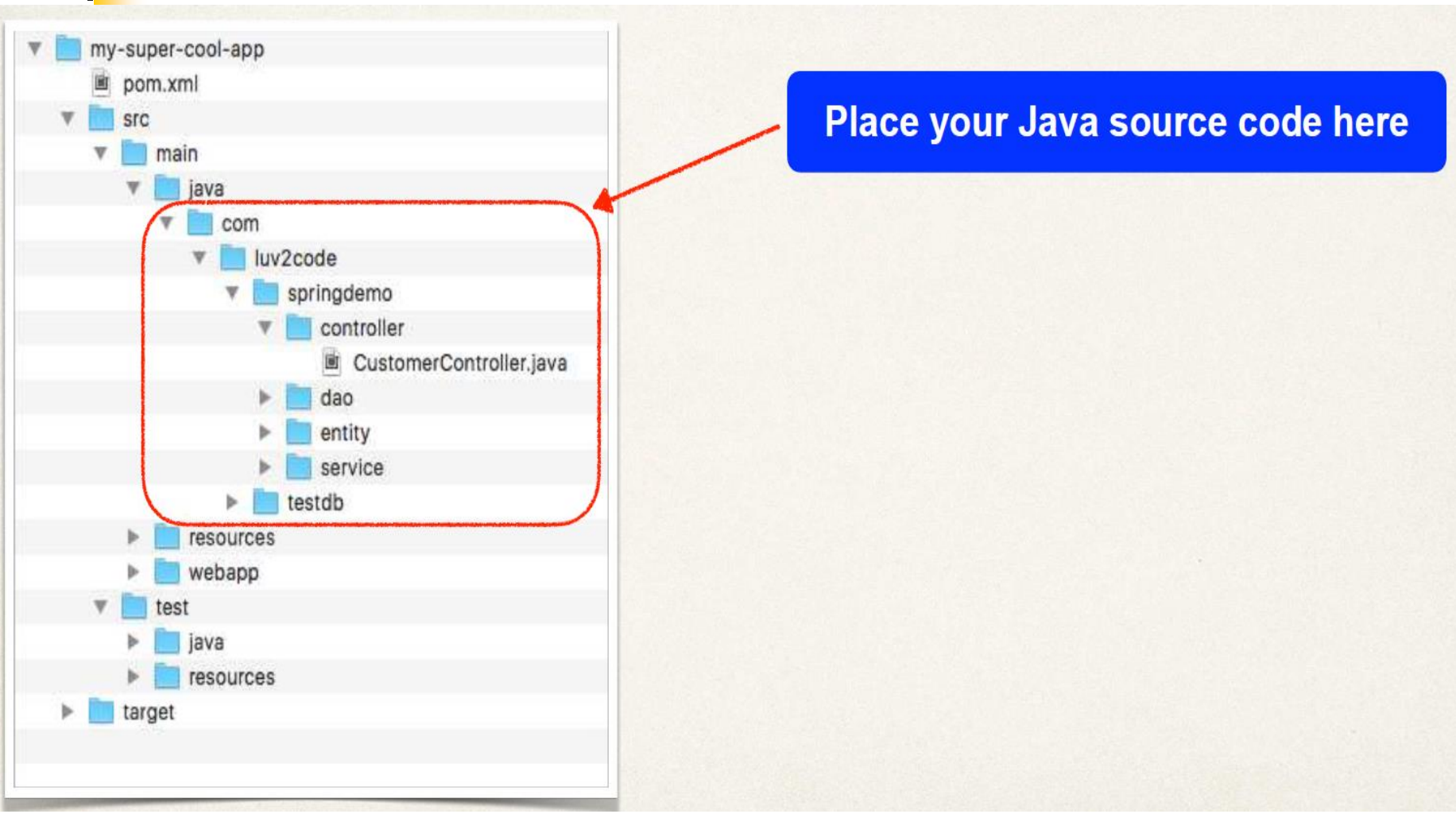
- When you build and run your app …
- Maven will handle class / build path for you
- Based on config file, Maven will add JAR files accordingly

Standard Directory Structure

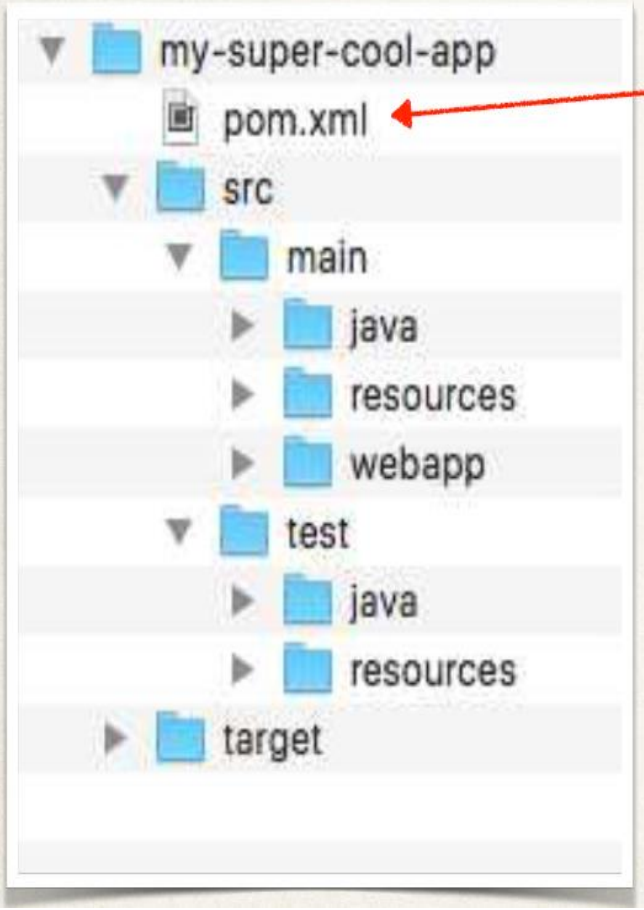


Directory	Description
src/main/java	Your Java source code
src/main/resources	Properties / config files used by your app
src/main/webapp	JSP files and web config files other web assets (images, css, js, etc)
src/test	Unit testing code and properties
target	Destination directory for compiled code. Automatically created by Maven

Standard Directory Structure



Maven Key Concepts-POM File-pom.xml



- Project Object Model file: POM file
- Configuration file for your project
- Located in the root of your Maven project

Simple POM File

```
<project ...>

  <modelVersion>4.0.0</modelVersion>

  <groupId>com.luv2code</groupId>
  <artifactId>mycoolapp</artifactId>
  <version>1.0.FINAL</version>
  <packaging>jar</packaging>

  <name>mycoolapp</name>

  <dependencies>
    <dependency>
      <groupId>org.junit.jupiter</groupId>
      <artifactId>junit-jupiter</artifactId>
      <version>5.9.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>

    <!-- add plugins for customization -->

</project>
```

Project name, version etc
Output file type: JAR, WAR, …

List of projects we depend on
Spring, Hibernate, etc…

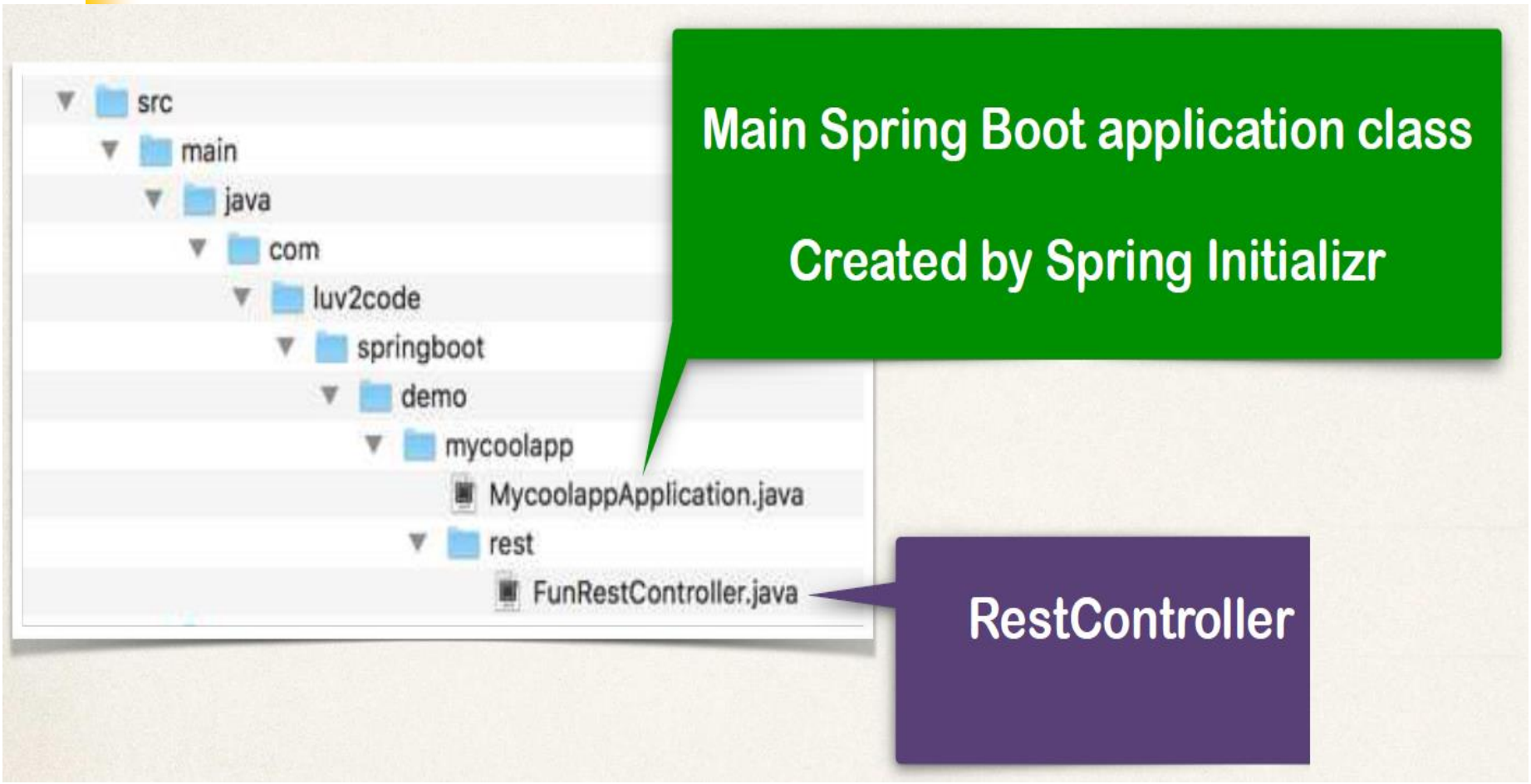
Additional custom tasks to run:
generate JUnit test reports etc…

Simple POM File

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** … If project is under active development then: **1.0-SNAPSHOT**

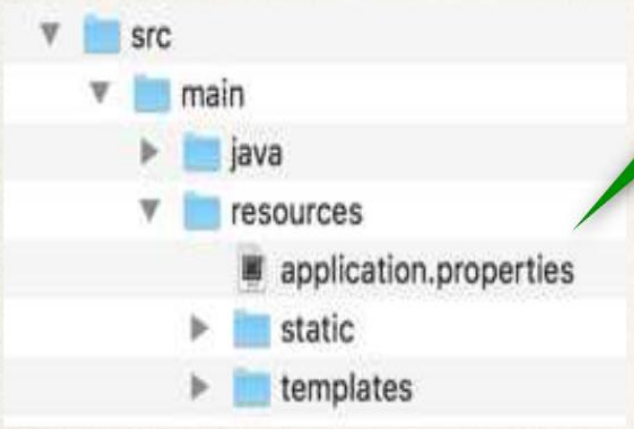
```
<groupId>com.luv2code</groupId>
<artifactId>mycoolapp</artifactId>
<version>1.0.FINAL</version>
```

Java Source Code



Application Properties

- By default, Spring Boot will load properties from: `application.properties`



Created by Spring Initializr

Empty at the beginning

Can add Spring Boot properties

`server.port=8585`

Also add your own custom properties

Application Properties

- Read data from: **application.properties**

```
# configure server port
server.port=8484

# configure my props
coach.name=Mickey Mouse
team.name=The Mouse Crew
```

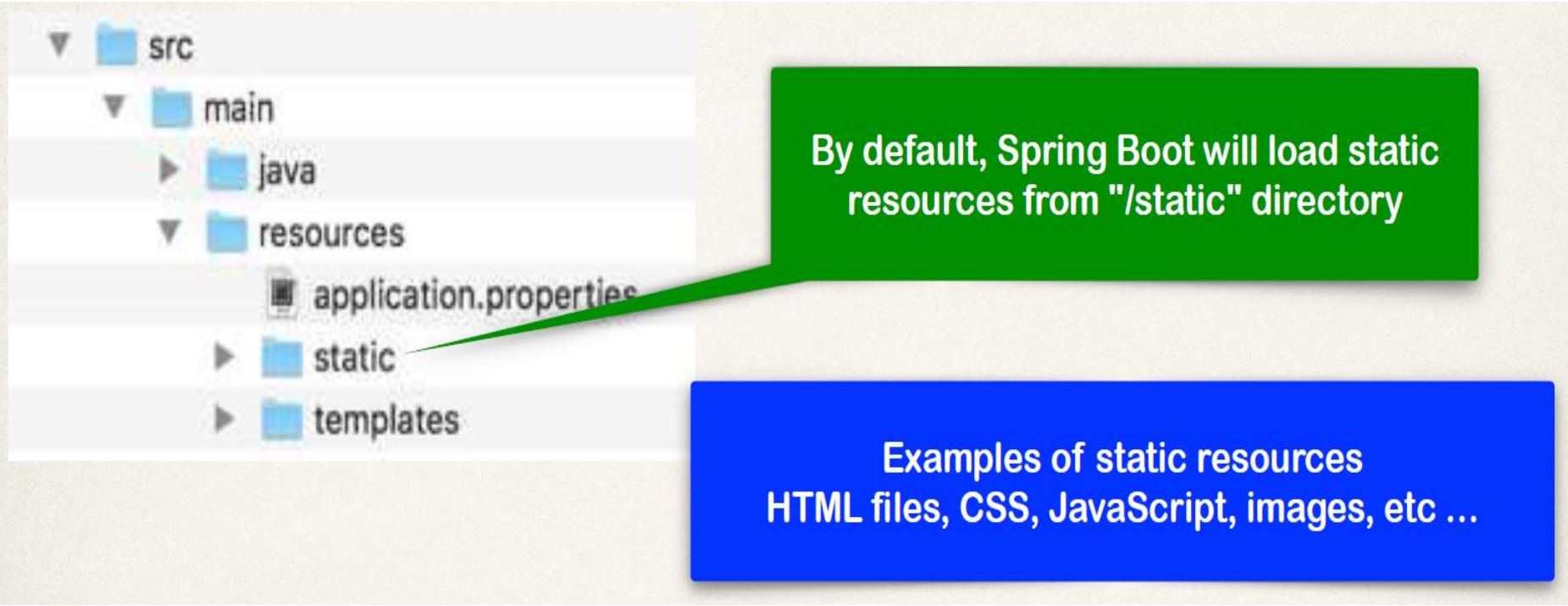
```
@RestController
public class FunRestController {

  @Value("${coach.name}")
  private String coachName;

  @Value("${team.name}")
  private String teamName;

  ...
}
```

Static Content



Spring Boot Starter - Web

- Spring Boot provides: **spring-boot-starter-web**

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

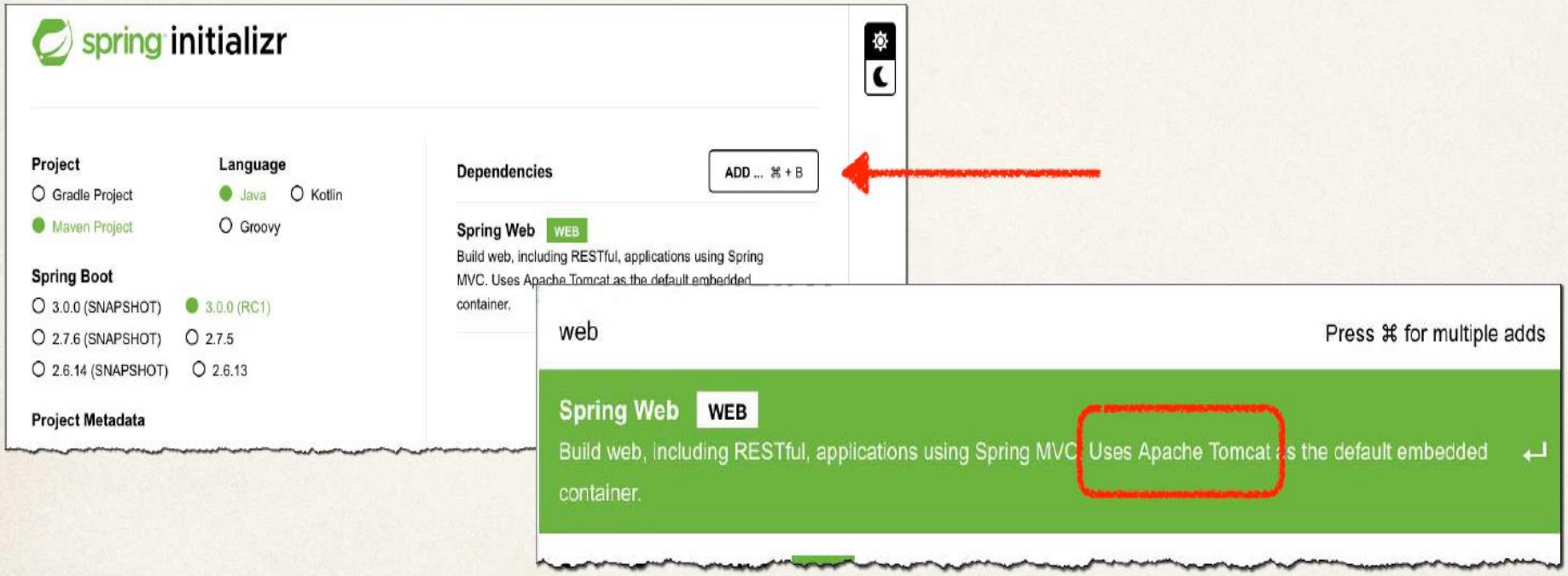
Spring Boot Starters

A collection of Maven dependencies (Compatible versions)

CONTAINS
spring-web
spring-webmvc
hibernate-validator
json
tomcat
...

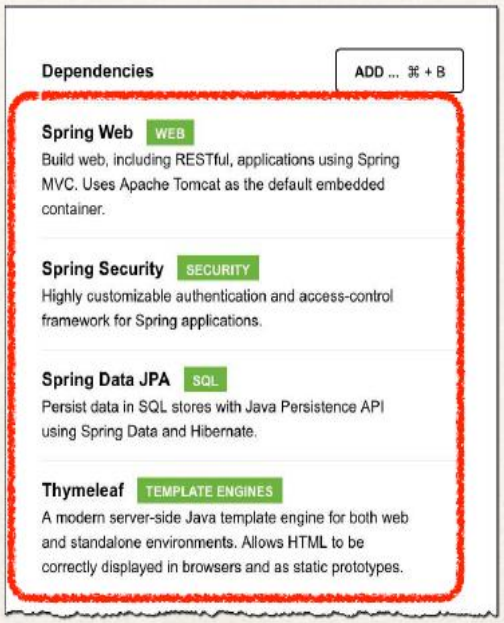
Spring Initializr

- In Spring Initializr, simply select **Web** dependency
- You automatically get **`spring-boot-starter-web`** in **`pom.xml`**

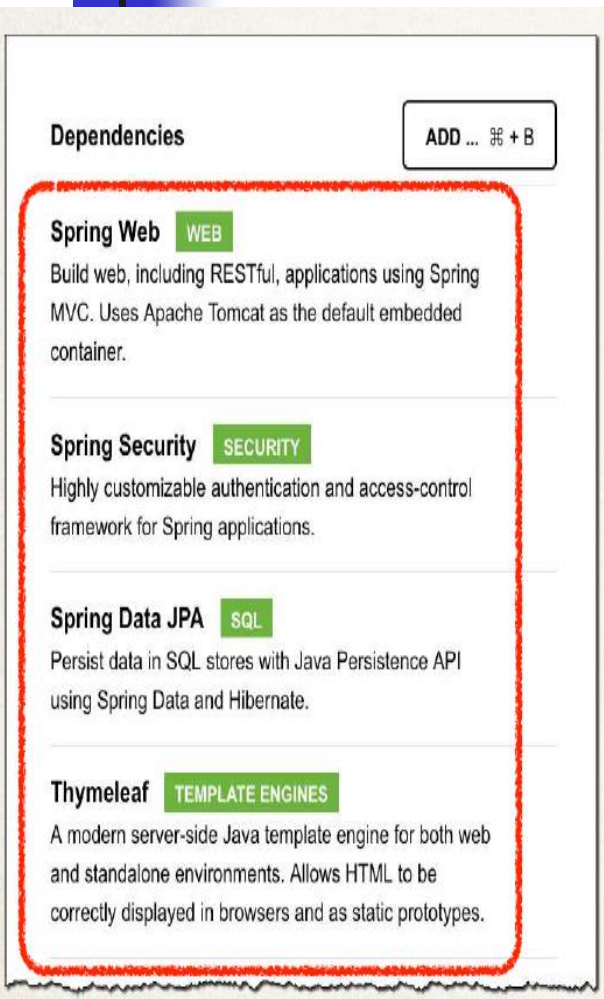


Spring Initializr

- If we are building a Spring app that needs: Web, Security, …
- Simply select the dependencies in the Spring Initializr
- It will add the appropriate Spring Boot starters to your **`pom.xml`**



Spring Initializr



File: pom.xml

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
```

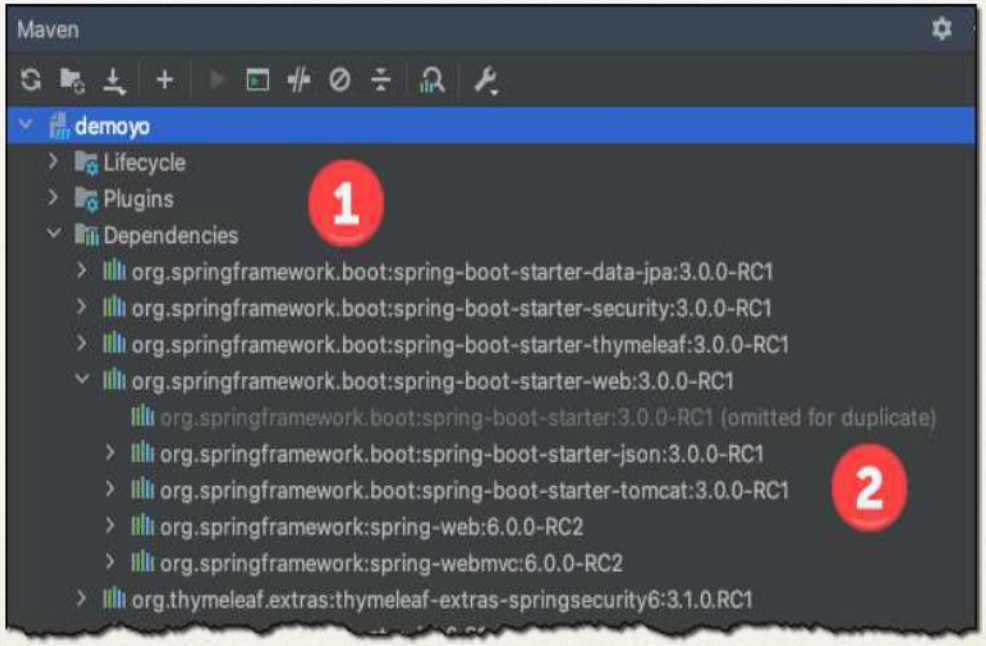
Spring Boot Starters

- There are 30+ Spring Boot Starters from the Spring Development team

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 Is In the Starter?

- For IntelliJ users
- Select **View > Tool Windows > Maven Projects > Dependencies**



Spring Boot Starter Parent

- Spring Boot provides a "Starter Parent"
- This is a special starter that provides Maven defaults

```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>3.0.0-RC1</version>
  <relativePath/> <!-- lookup parent from repository -->
</parent>
```

Included in pom.xml when using Spring Initializr

Spring Boot Starter Parent

- For the **`spring-boot-starter-*`** dependencies, no need to list version

```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>3.0.0-RC1</version>
  <relativePath/> <!-- lookup parent from repository -->
</parent>
```

Specify version of Spring Boot

```
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
```

Inherit version from Starter Parent

No need to list individual versions
Great for maintenance!

Spring Boot Starter Parent

- Default configuration of Spring Boot plugin

```
<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
  </plugins>
</build>
```

```
> mvn spring-boot:run
```

Spring Boot Dev Tools

The Problem

- When running Spring Boot applications
  - If you make changes to your source code
  - Then you have to manually restart your application

Spring Boot Dev Tools

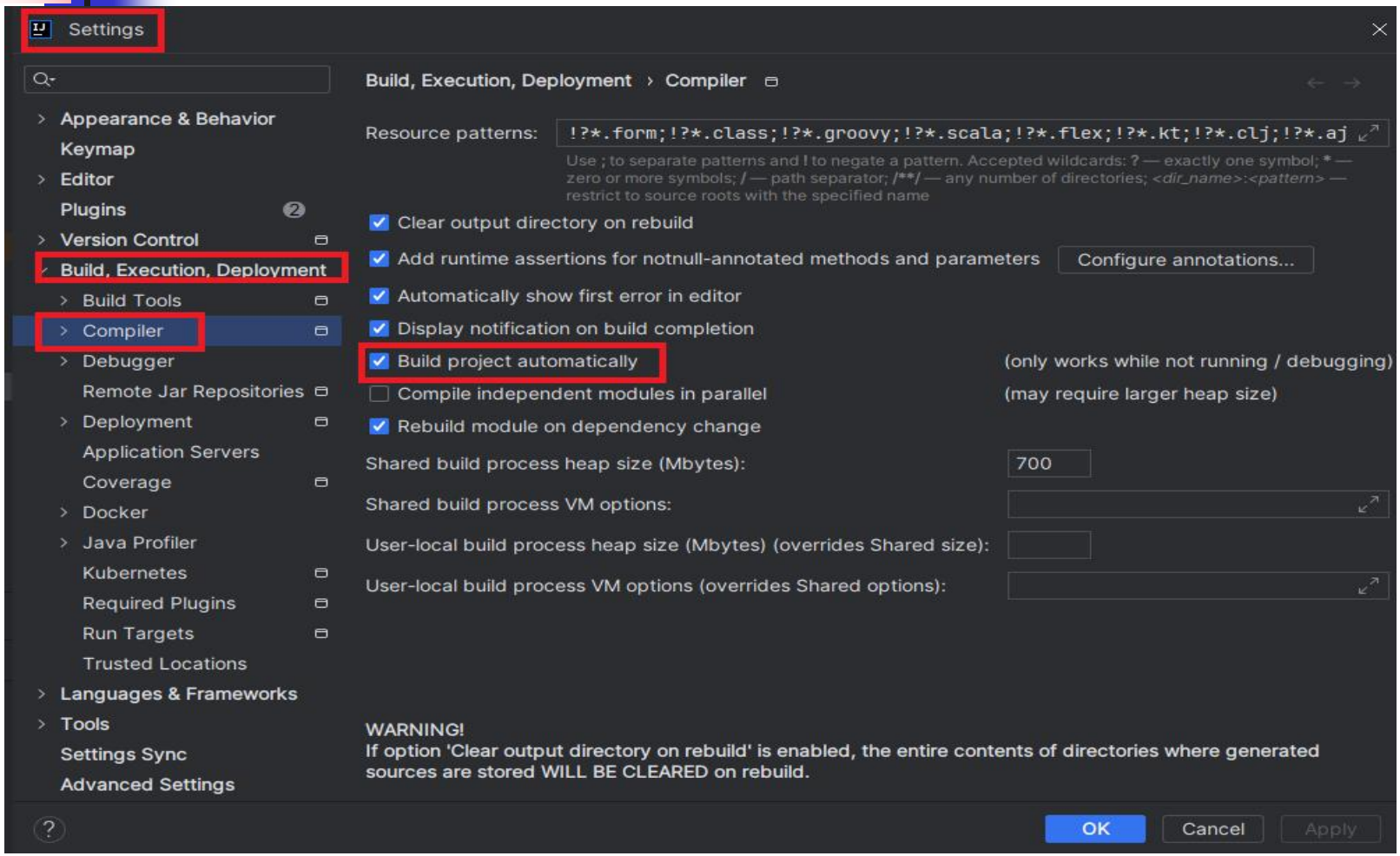
- Automatically restarts your application when code is updated
- Simply add the dependency to your POM file
- No need to write additional code
- For IntelliJ, need to set additional configurations

- Adding the dependency to your POM file

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
</dependency>
```

Automatically restarts your application when code is updated

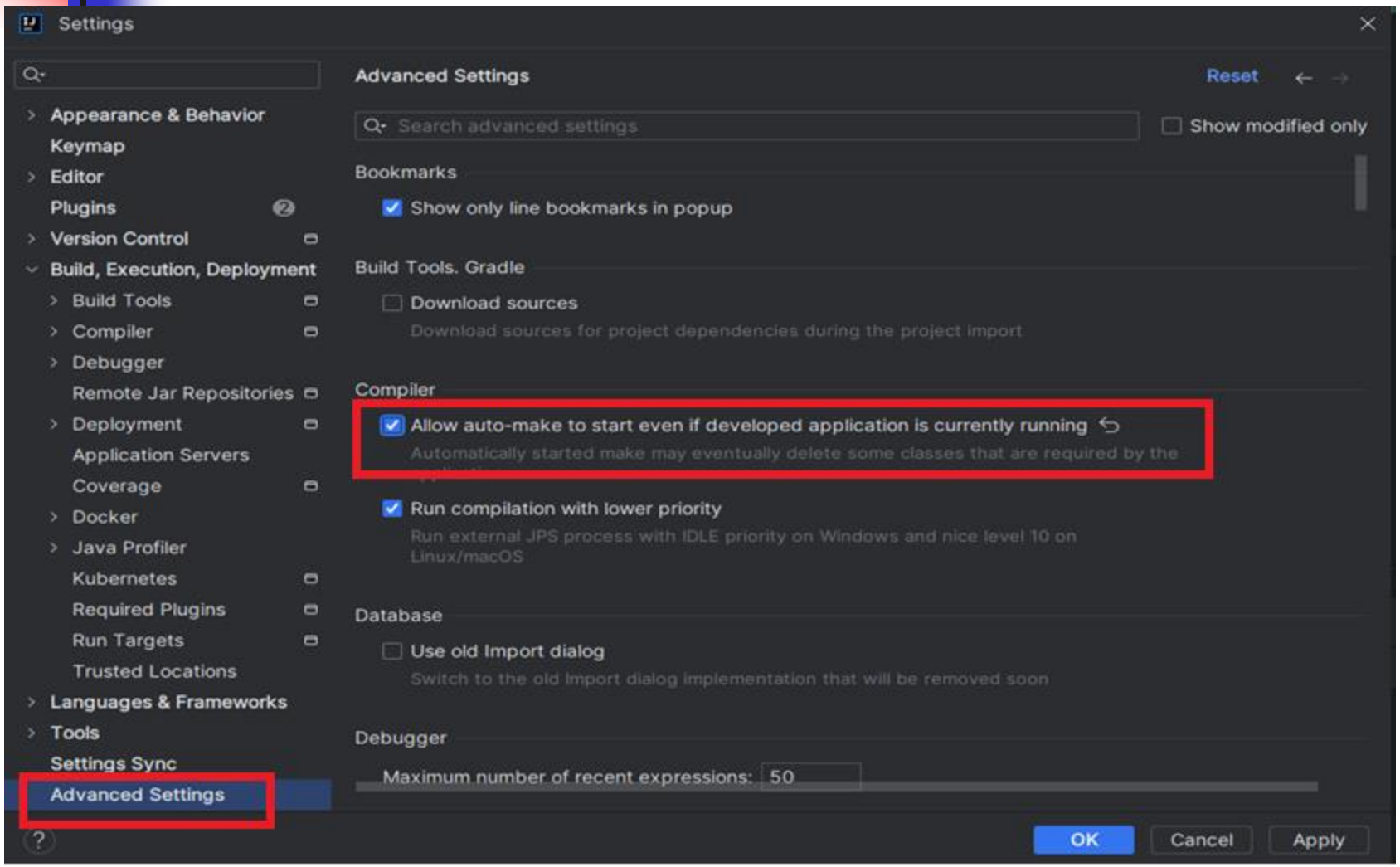
IntelliJ Community Edition - DevTools



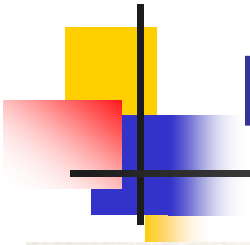
IntelliJ Community Edition does not support DevTools by default

Check box: **Build project automatically**

IntelliJ Community Edition - DevTools



Check box: **Allow auto-make to …**



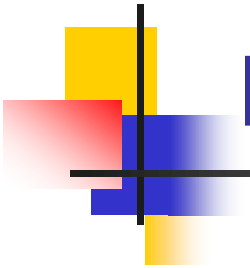
Development Process

1. Apply IntelliJ configurations

2. Edit **`pom.xml`** and add **`spring-boot-devtools`**

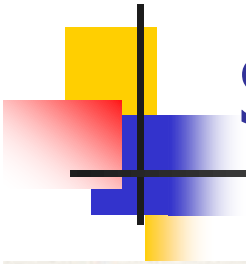
3. Add new REST endpoint to our app

4. Verify the app is automatically reloaded



Problem

- How can I monitor and manage my application?
- How can I check the application health?
- How can I access application metrics?



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

No need to write additional code!

You get new REST endpoints for FREE!

Spring Boot Actuator

- Adding the dependency to your POM file

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

- Automatically exposes endpoints for metrics out-of-the-box
- Endpoints are prefixed with: **`/actuator`**

Name	Description
`/health`	**Health information about your application**

Spring Boot Actuator - Monitor & Manage Your App

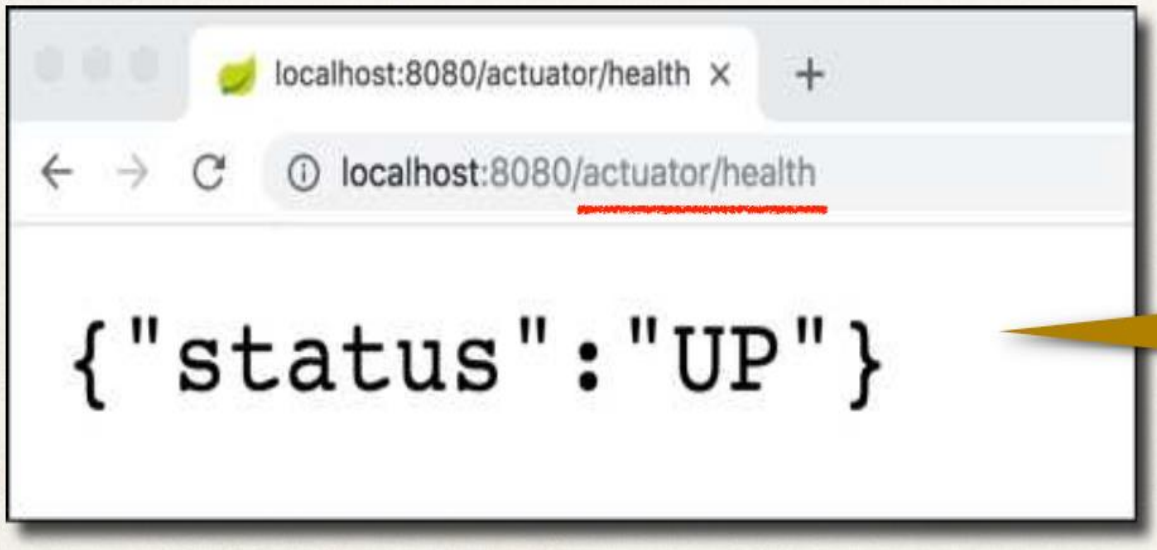
Spring Boot **Actuator** is a built-in tool that helps you:

- **Monitor** application health.
- **Access application metrics (CPU, memory, DB connections, requests/sec, etc.)**
- **Expose REST endpoints for DevOps & debugging**

Endpoint	Description
`/actuator/health`	Shows if the app is running & its status.
`/actuator/info`	Displays app metadata (version, name, etc.).
`/actuator/metrics`	Shows memory, CPU, request stats, etc.
`/actuator/env`	Displays environment properties & config.
`/actuator/loggers`	Manage logging levels at runtime.

Health Endpoint

- **`/health`** checks the status of your application
- Normally used by monitoring apps to see if your app is up or down



Health status is customizable based on your own business logic

Exposing Endpoints

- By default, only **`/health`** is exposed
- The **`/info`** endpoint can provide information about your application
- To expose **`/info`**

File: src/main/resources/application.properties

```
management.endpoints.web.exposure.include=health,info
management.info.env.enabled=true
```

Spring Boot Actuator Endpoints

- There are 10+ Spring Boot Actuator endpoints

Name	Description
`/auditevents`	Audit events for your application
`/beans`	List of all beans registered in the Spring application context
`/mappings`	List of all @RequestMapping paths
...	

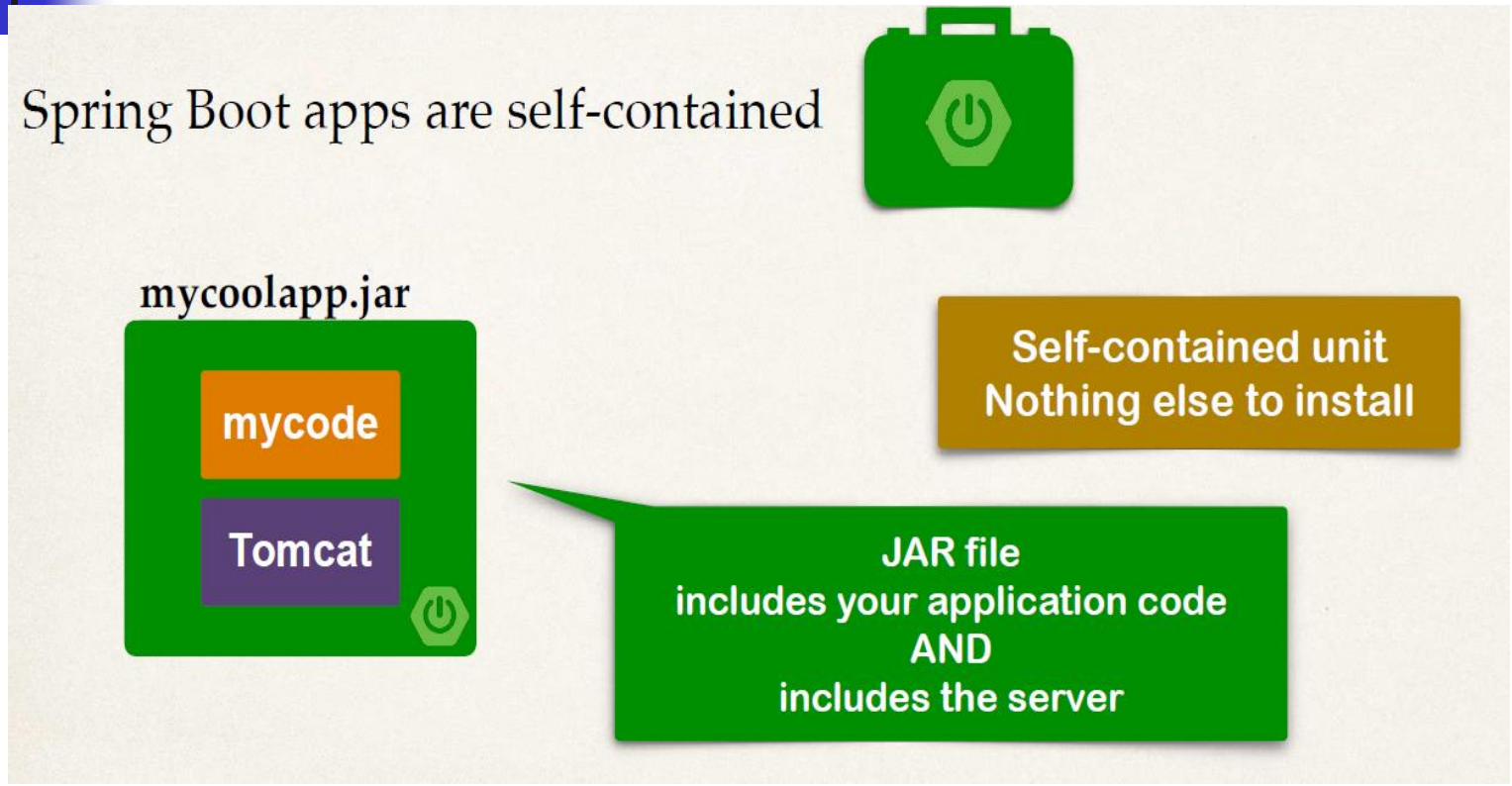
Running from the Command-Line

- When running from the command-line
  - No need to have IDE open/running

- Since we using Spring Boot, the server is embedded in our JAR file
  - No need to have separate server installed/running

- Spring Boot apps are self-contained

Running from the Command-Line



Spring Boot applications are **self-contained** because they package everything needed to run the app, including:

Application code
All required dependencies (JARs)
Embedded web server (like Tomcat, Jetty)

Running from the Command-Line

- Two options for running the app
- Option 1: Use `java -jar`
- Option 2: Use Spring Boot Maven plugin
  - `mvnw spring-boot:run`

Option 2: Use Spring Boot Maven plugin

```
<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
  </plugins>
</build>
```

To package executable jar or war archive

Can also easily run the app

```
$ ./mvnw package

$ ./mvnw spring-boot:run
```

Can also just use:

mvn package
mvn spring-boot:run

Spring Boot - Custom Application Properties

- You need for your app to be configurable … no hard-coding of values
- You need to read app configuration from a 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`**

Standard Spring Boot file name

No additional coding or configuration required

Spring Boot - Custom Application Properties

1. Define custom properties in `application.properties`

2. Inject properties into Spring Boot application using `@Value`

Spring Boot Properties

- Spring Boot can be configured in the **`application.properties`** file
- Server port, context path, actuator, security etc …

- The properties are roughly grouped into the following categories

Core Properties

File: src/main/resources/application.properties

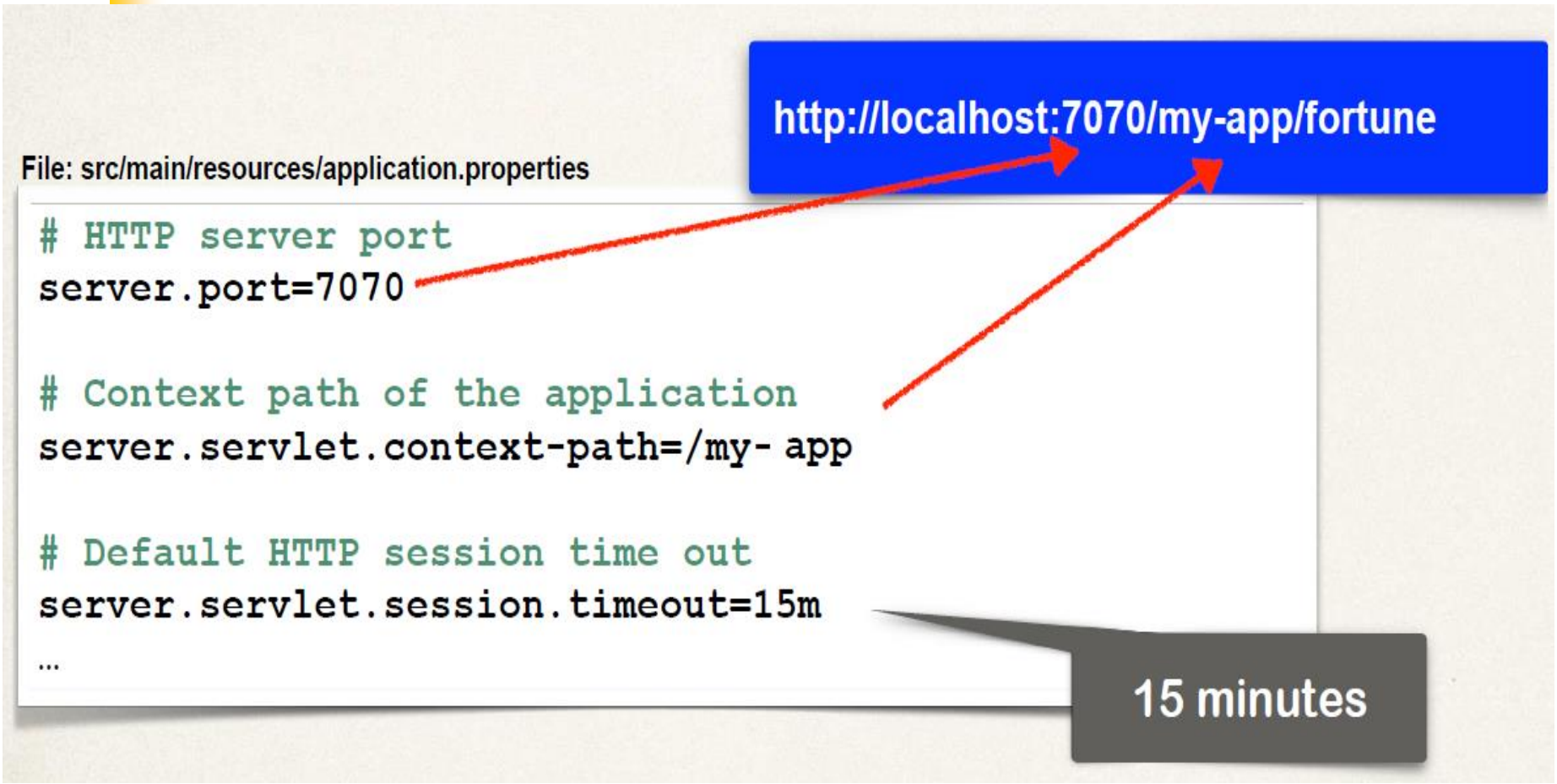
```
# Log levels severity mapping
logging.level.org.springframework=DEBUG
logging.level.org.hibernate=TRACE
logging.level.com.luv2code=INFO

# Log file name
logging.file.name=my-crazy-stuff.log
logging.file.path=c:/myapps/demo
...
```

Logging Levels

TRACE
DEBUG
INFO
WARN
ERROR
FATAL
OFF

Web Properties



Actuator Properties

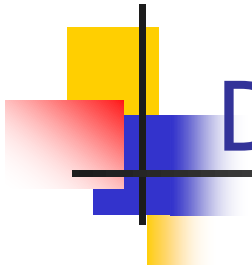
File: src/main/resources/application.properties

```
# Endpoints to include by name or wildcard
management.endpoints.web.exposure.include=*

# Endpoints to exclude by name or wildcard
management.endpoints.web.exposure.exclude=beans,mapping

# Base path for actuator endpoints
management.endpoints.web.base-path=/actuator
...
```

http://localhost:7070/actuator/health



Data Properties

File: src/main/resources/application.properties

```
# JDBC URL of the database
spring.datasource.url=jdbc:mysql://localhost:3306/ecommerce

# Login username of the database
spring.datasource.username=

# Login password of the database
spring.datasource.password=
...
```