

# Creating Your First Spring Boot App

*Introducing Spring Boot 2*



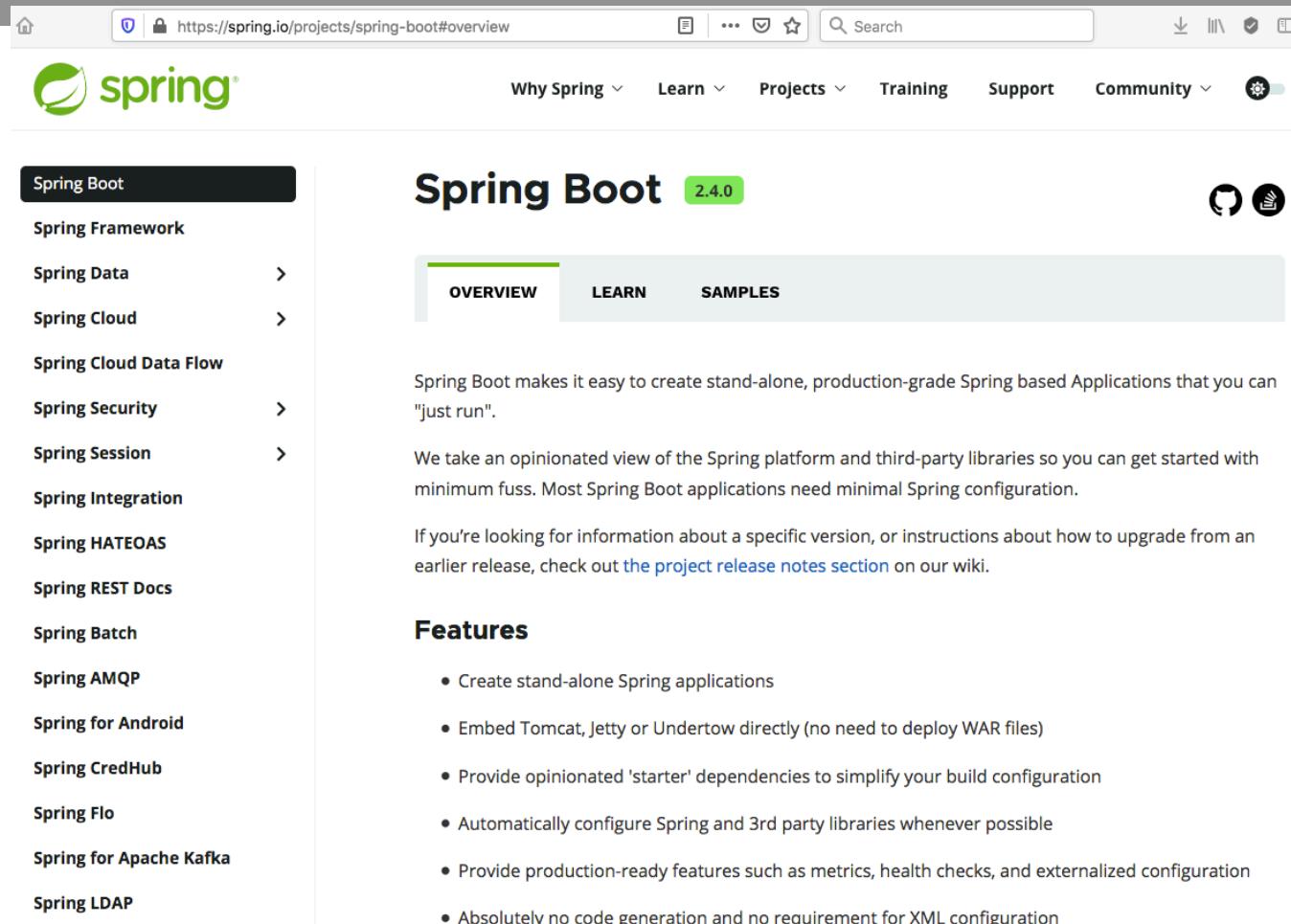
- what Spring can do – since years, by projects
- web app Spring Initializr – dependencies like Web incl. Tomcat
- IntelliJ CE vs Ultimate
- best practices for a simple project structure
- Spring Boot Starters supporting dependence injections
- starting Spring Boot Applications
- maven, eclipse, ..

# What Spring can do: Ecosystem beside Oracle

The screenshot shows the Spring.io website's ecosystem section. It features seven cards, each representing a different architectural style or technology stack:

- Microservices**: Quickly deliver production-grade features with independently evolvable microservices.
- Reactive**: Spring's asynchronous, nonblocking architecture means you can get more from your computing resources.
- Cloud**: Your code, any cloud—we've got you covered. Connect and scale your services, whatever your platform.
- Web apps**: Frameworks for fast, secure, and responsive web applications connected to any data store.
- Serverless**: The ultimate flexibility. Scale up on demand and scale to zero when there's no demand.
- Event Driven**: Integrate with your enterprise. React to business events. Act on your streaming data in realtime.
- Batch**: Automated tasks. Offline processing of data at a time to suit you.

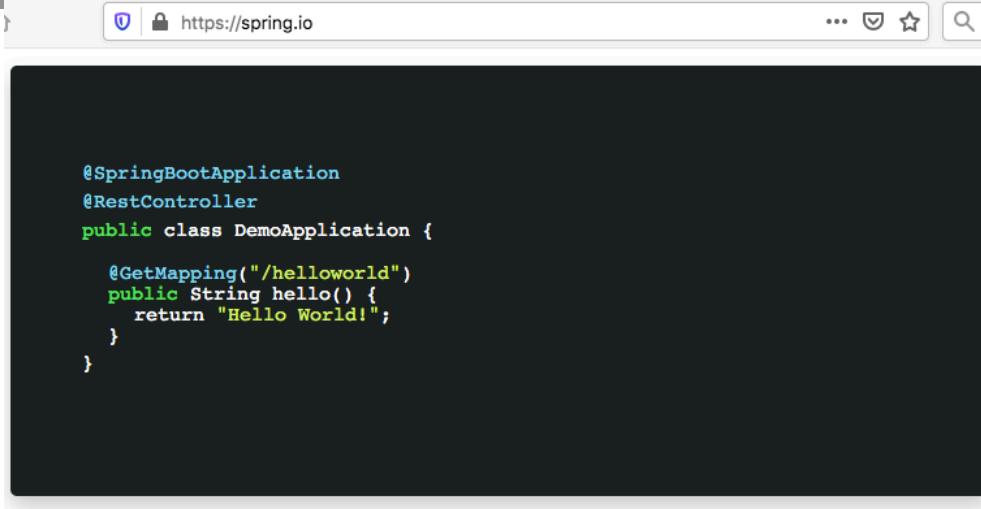
# Spring Projects



The screenshot shows the Spring Boot project page on the official Spring website. The URL in the browser is <https://spring.io/projects/spring-boot#overview>. The page features a navigation bar with links for Why Spring, Learn, Projects, Training, Support, and Community. The main content area is titled "Spring Boot 2.4.0". It includes tabs for OVERVIEW, LEARN, and SAMPLES. The OVERVIEW section contains text about Spring Boot's ease of use and its opinionated view of the Spring platform. The LEARN section provides upgrade instructions and links to release notes. The SAMPLES section is currently inactive. Below the tabs, there is a "Features" section with a bulleted list of benefits:

- Create stand-alone Spring applications
- Embed Tomcat, Jetty or Undertow directly (no need to deploy WAR files)
- Provide opinionated 'starter' dependencies to simplify your build configuration
- Automatically configure Spring and 3rd party libraries whenever possible
- Provide production-ready features such as metrics, health checks, and externalized configuration
- Absolutely no code generation and no requirement for XML configuration

# It's not really new, but ..



The screenshot shows a web browser window with the URL <https://spring.io>. The page displays a Java code snippet for a Spring Boot application:

```
@SpringBootApplication
@RestController
public class DemoApplication {

    @GetMapping("/helloworld")
    public String hello() {
        return "Hello World!";
    }
}
```

To the right of the code, there is a promotional section with the heading "Level up your Java™ code" and a brief description: "With [Spring Boot](#) in your app, just a few lines of code is all you need to start building services like a boss." Below this, a link reads "New to Spring? Try our simple [quickstart guide](#)".



Originally [Netflix's Java] libraries and frameworks were built in-house. I'm very proud to say, as of early 2019, we've moved our platform almost entirely over to Spring Boot."



TAYLOR WICKSELL, SENIOR SOFTWARE ENGINEER, NETFLIX  
[Watch now](#)

# Use Spring Initializr with Java, Maven

The screenshot shows the Spring Initializer web application interface. At the top, there are tabs for 'Spring Initializr' and a search bar. Below the header, the 'spring initializr' logo is displayed. On the left side, there are sections for 'Project' (Maven Project selected), 'Language' (Java selected), 'Spring Boot' (2.4.0 selected), and 'Project Metadata' (Group: com.che.example, Artifact: demo, Name: Hello SpringBoot World, Description: Demo project for Spring Boot, Package name: com.che.example.demo, Packaging: Jar selected). On the right side, there is a 'Dependencies' section with a button to 'ADD DEPENDENCIES...' and a note that says 'No dependency selected'.

# Add dependencies ..

spring:ir

Web, Security, JPA, Actuator, Devtools...

Press ⌘ for multiple adds

**DEVELOPER TOOLS**

**Spring Boot DevTools**  
Provides fast application restarts, LiveReload, and configurations for enhanced development experience.

**Lombok**  
Java annotation library which helps to reduce boilerplate code.

**Spring Configuration Processor**  
Generate metadata for developers to offer contextual help and "code completion" when working with custom configuration keys (ex.application.properties/.yml files).

**WEB**

**Spring Web**  
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

**Spring Reactive Web**  
Build reactive web applications with Spring WebFlux and Netty.

**Rest Repositories**  
Exposing Spring Data repositories over REST via Spring Data REST.

**Spring Session**  
Provides an API and implementations for managing user session information.

# Like Web incl. Tomcat



**Project**  
 Maven Project  
 Gradle Project

**Language**  
 Java  
 Kotlin  
 Groovy

**Spring Boot**  
 2.4.1 (SNAPSHOT)  
 2.4.0  
 2.3.7 (SNAPSHOT)  
 2.3.6  
 2.2.12 (SNAPSHOT)  
 2.2.11

**Project Metadata**

Group com.che.example

Artifact demo

Name Hello SpringBoot World

Description Demo project for Spring Boot

Package name com.che.example.demo

Packaging  Jar  
 War

**Dependencies** [ADD DEPENDENCIES... ⌘ + B](#)

**Spring Web WEB**  
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

# Press Generate button



**Project**

Maven Project  
 Gradle Project

**Language**

Java  
 Kotlin  
 Groovy

**Spring Boot**

2.4.1(SNAPSHOT)     2.4.0     2.3.7 (SNAPSHOT)     2.3.6  
 2.2.12 (SNAPSHOT)     2.2.11

**Project Metadata**

Group com.che.example

Artifact demo

Name Hello SpringBoot World

Description Demo project for Spring Boot

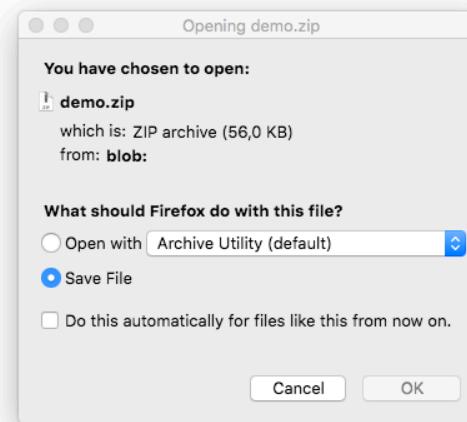
Package name com.che.example.demo

Packaging  Jar     War

**Dependencies** ADD DEPENDENCIES... ⌘ + B

**Spring Web** WEB

Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

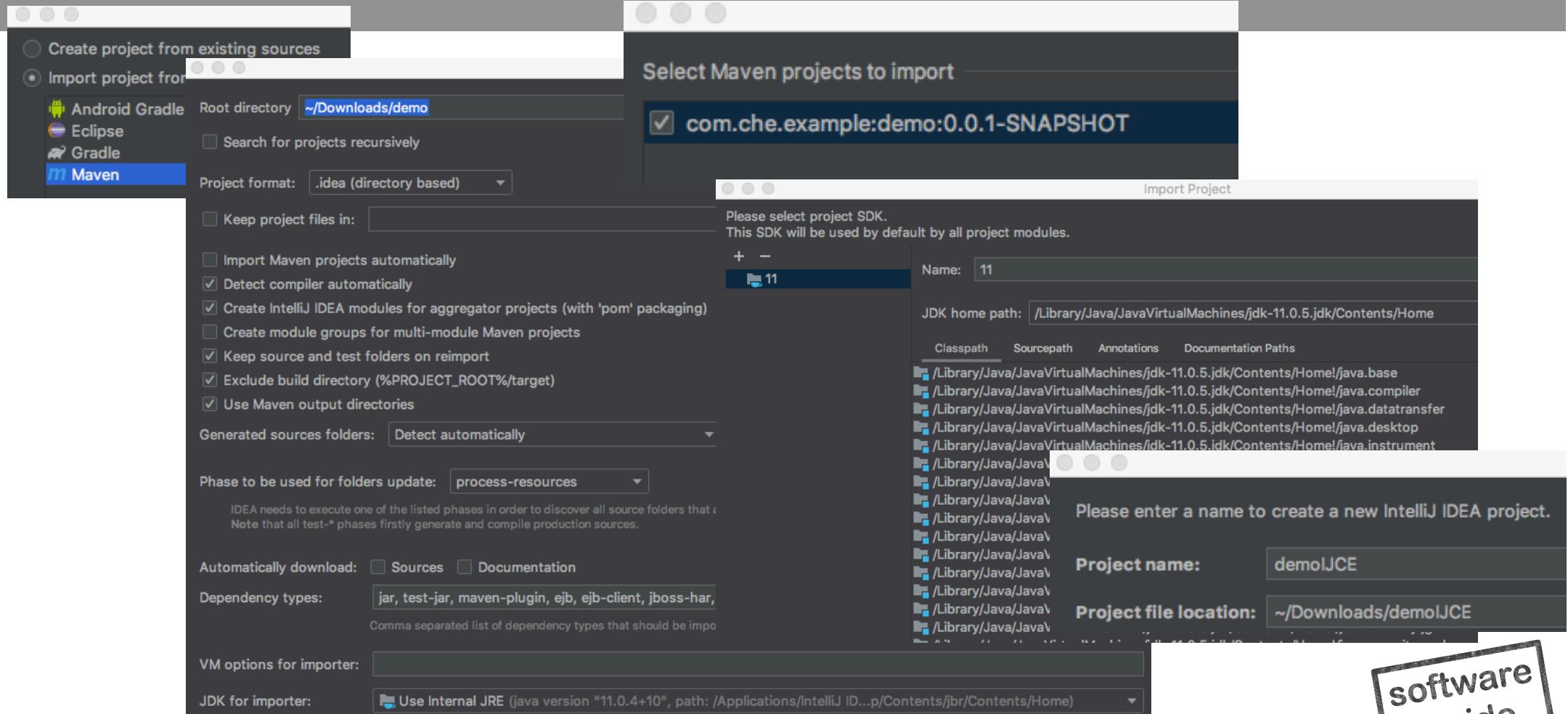


GENERATE ⌘ + ↩

EXPLORE CTRL + SPACE

SHARE...

# Import extracted zip by Community Edition

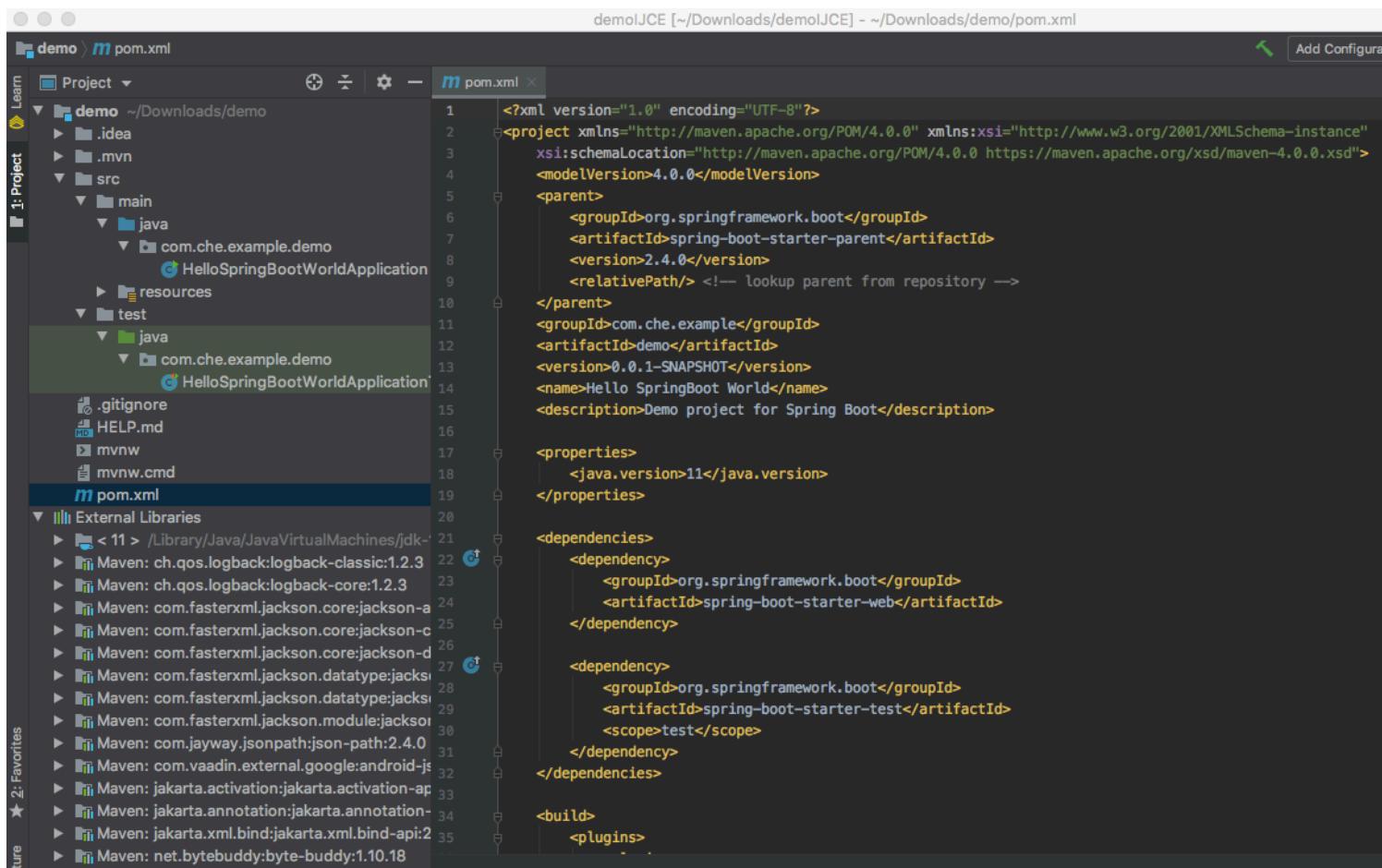


The screenshot shows the IntelliJ IDEA Community Edition interface with the following steps:

- Left Panel (Import Project):**
  - Radio button selected:  Import project from file...
  - Root directory: `~/Downloads/demo`
  - Project format: `.idea (directory based)`
  - Keep project files in: [empty field]
  - Import Maven projects automatically
  - Detect compiler automatically
  - Create IntelliJ IDEA modules for aggregator projects (with 'pom' packaging)
  - Create module groups for multi-module Maven projects
  - Keep source and test folders on reimport
  - Exclude build directory (%PROJECT\_ROOT%/.target)
  - Use Maven output directories
  - Generated sources folders: `Detect automatically`
  - Phase to be used for folders update: `process-resources`
  - IDEA needs to execute one of the listed phases in order to discover all source folders that are part of the project.
  - Note that all test-\* phases firstly generate and compile production sources.
  - Automatically download:  Sources  Documentation
  - Dependency types: `jar, test-jar, maven-plugin, ejb, ejb-client, jboss-har,`
  - Comma separated list of dependency types that should be imported.
  - VM options for importer: [empty field]
  - JDK for importer: `Use Internal JRE (java version "11.0.4+10", path: /Applications/IntelliJ IDEA.app/Contents/jbr/Contents/Home)`
- Middle Panel (Select Maven projects to import):**
  - Checkmark selected:  com.che.example:demo:0.0.1-SNAPSHOT
  - Please select project SDK. This SDK will be used by default by all project modules.
  - + - [button] 11 [button]
  - Name: `11`
  - JDK home path: `/Library/Java/JavaVirtualMachines/jdk-11.0.5.jdk/Contents/Home`
  - Classpath Sourcepath Annotations Documentation Paths
  - Classpath entries: `/Library/Java/JavaVirtualMachines/jdk-11.0.5.jdk/Contents/Home!/java.base`, `/Library/Java/JavaVirtualMachines/jdk-11.0.5.jdk/Contents/Home!/java.compiler`, `/Library/Java/JavaVirtualMachines/jdk-11.0.5.jdk/Contents/Home!/java.datatransfer`, `/Library/Java/JavaVirtualMachines/jdk-11.0.5.jdk/Contents/Home!/java.desktop`, `/Library/Java/JavaVirtualMachines/jdk-11.0.5.jdk/Contents/Home!/java.instrument`, `/Library/Java/JavaVirtualMachines/jdk-11.0.5.jdk/Contents/Home!/java.logging`, `/Library/Java/JavaVirtualMachines/jdk-11.0.5.jdk/Contents/Home!/java.management`, `/Library/Java/JavaVirtualMachines/jdk-11.0.5.jdk/Contents/Home!/java.net`, `/Library/Java/JavaVirtualMachines/jdk-11.0.5.jdk/Contents/Home!/java.nio`, `/Library/Java/JavaVirtualMachines/jdk-11.0.5.jdk/Contents/Home!/java.sql`, `/Library/Java/JavaVirtualMachines/jdk-11.0.5.jdk/Contents/Home!/java.util`, `/Library/Java/JavaVirtualMachines/jdk-11.0.5.jdk/Contents/Home!/java.util.concurrent`, `/Library/Java/JavaVirtualMachines/jdk-11.0.5.jdk/Contents/Home!/java.util.logging`, `/Library/Java/JavaVirtualMachines/jdk-11.0.5.jdk/Contents/Home!/java.util.prefs`, `/Library/Java/JavaVirtualMachines/jdk-11.0.5.jdk/Contents/Home!/java.xml`, `/Library/Java/JavaVirtualMachines/jdk-11.0.5.jdk/Contents/Home!/java.xml.bind`, `/Library/Java/JavaVirtualMachines/jdk-11.0.5.jdk/Contents/Home!/java.xml.ws`, `/Library/Java/JavaVirtualMachines/jdk-11.0.5.jdk/Contents/Home!/java.xml.stream`.
- Right Panel (Create New Project):**
  - Please enter a name to create a new IntelliJ IDEA project.
  - Project name: `demoJCE`
  - Project file location: `~/Downloads/demoJCE`



# The result is a POM file..



A screenshot of an IDE (IntelliJ IDEA) showing a Maven project named "demo". The project structure on the left includes ".idea", ".mvn", "src" (containing "main" and "test" packages), and "External Libraries" (listing various dependencies like logback, jackson, and json-path). The right panel displays the "pom.xml" file content:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.4.0</version>
    <relativePath/> 
  </parent>
  <groupId>com.che.example</groupId>
  <artifactId>demo</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>Hello SpringBoot World</name>
  <description>Demo project for Spring Boot</description>

  <properties>
    <java.version>11</java.version>
  </properties>

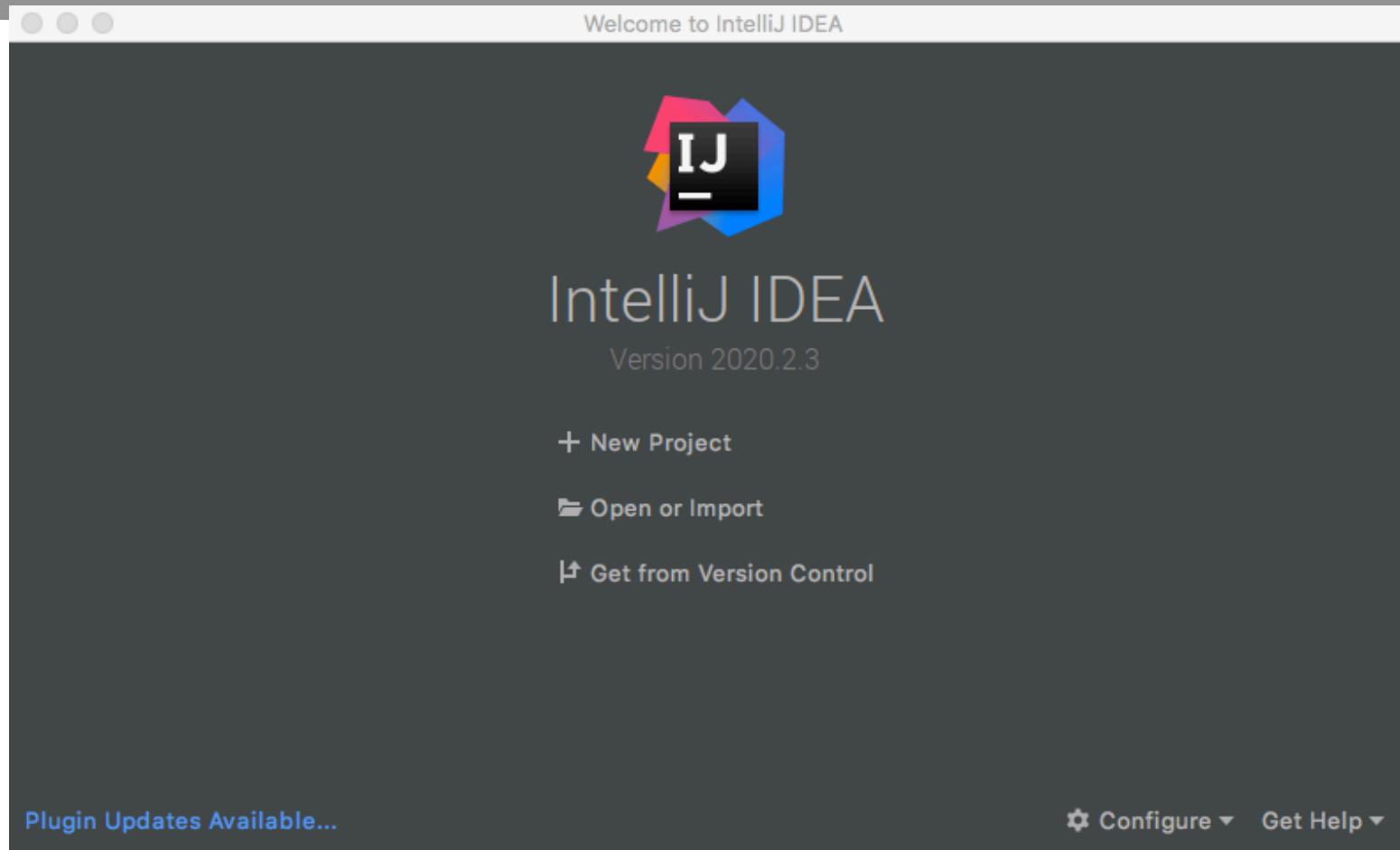
  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-test</artifactId>
      <scope>test</scope>
    </dependency>
  </dependencies>

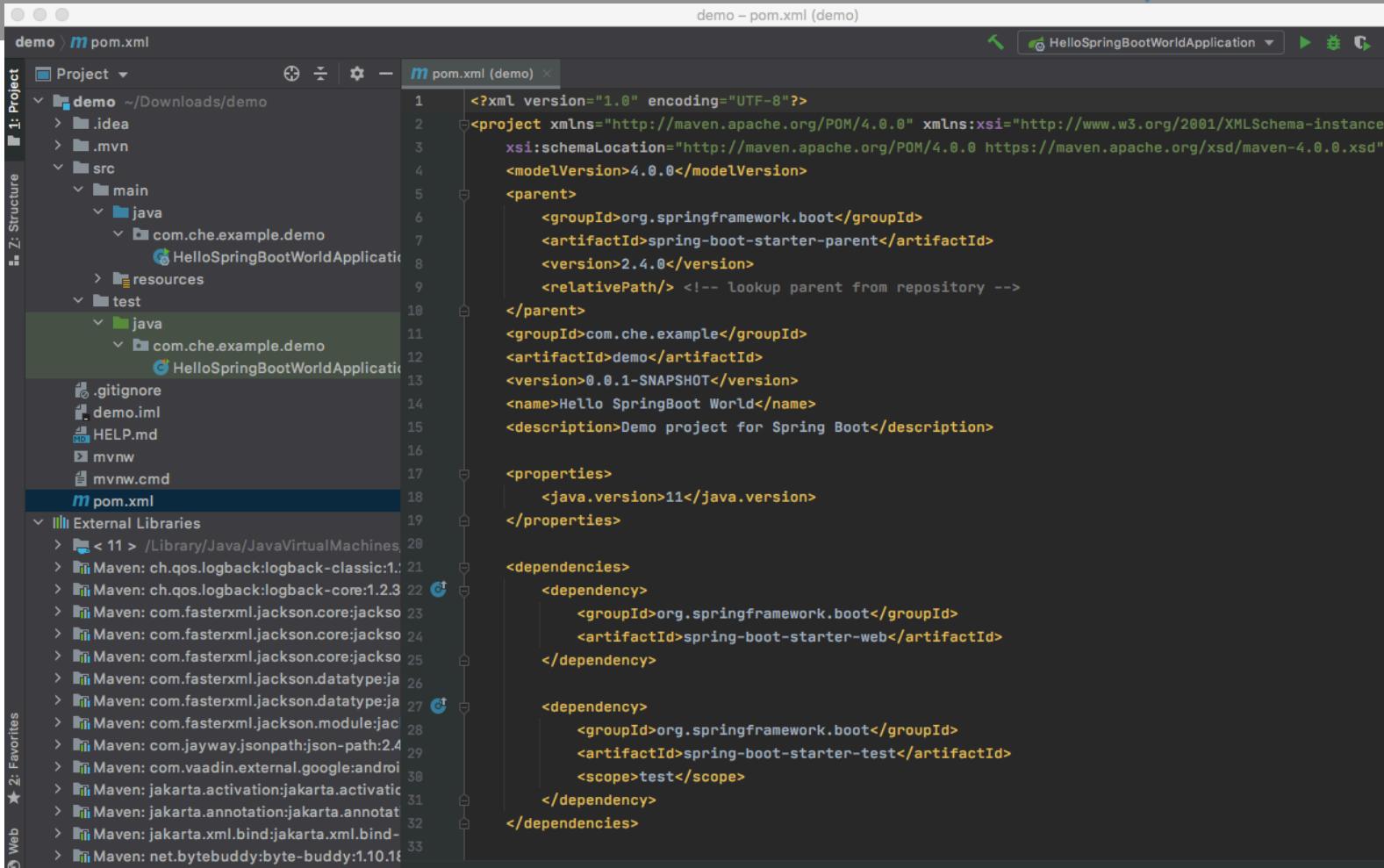
  <build>
    <plugins>
```

software  
inside

or re-extracted zip by Ultimate



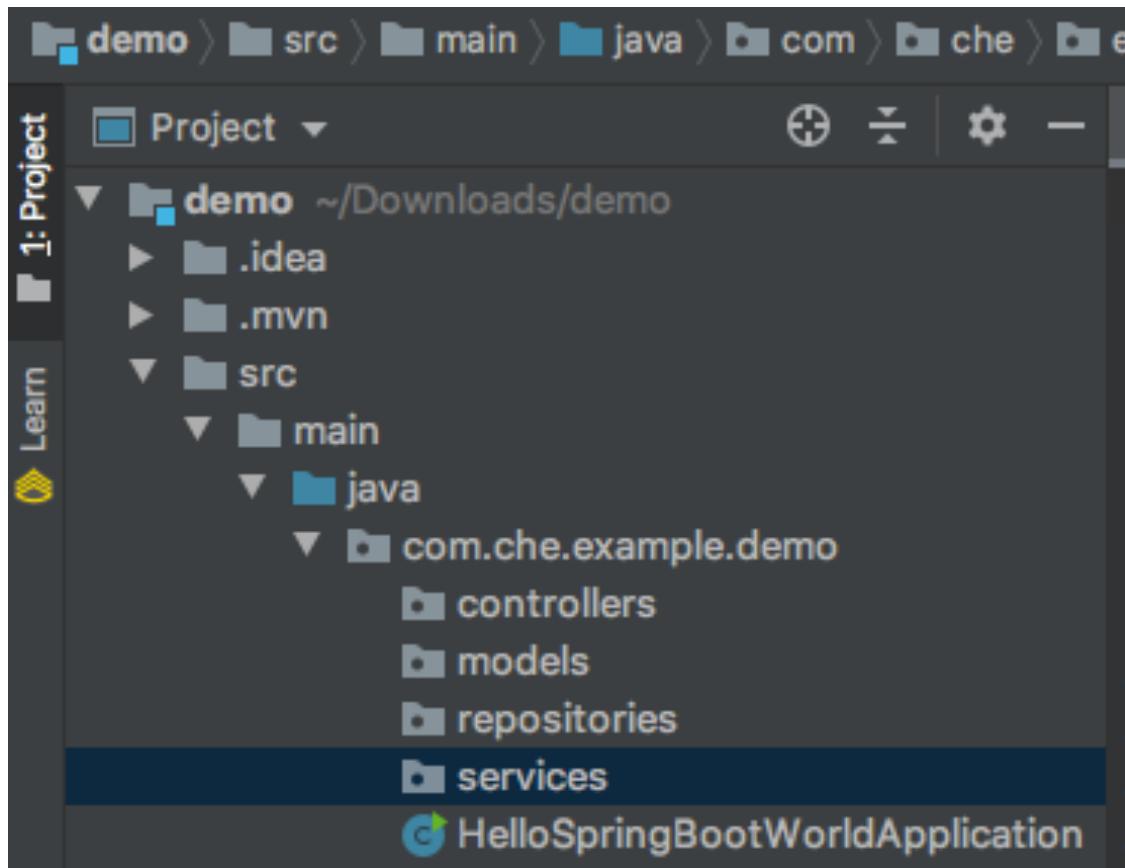
# result is the same



The screenshot shows an IDE interface with the following details:

- Title Bar:** demo – pom.xml (demo)
- Project Tree:** The project structure is displayed under "Project". It includes a .idea folder, .mvn, src (containing main and test), and resources. The test directory contains a java folder with a com.che.example.demo package containing a HelloSpringBootWorldApplication class.
- Code Editor:** The main content is the pom.xml file, which defines the project's metadata and dependencies. Key parts of the XML include:
  - <parent> section pointing to org.springframework.boot and spring-boot-starter-parent.
  - <groupId> com.che.example </groupId>
  - <version>0.0.1-SNAPSHOT</version>
  - <name>Hello SpringBoot World</name>
  - <description>Demo project for Spring Boot</description>
  - <java.version>11</java.version>
  - <dependencies> section with two dependency blocks for org.springframework.boot:spring-boot-starter-web and org.springframework.boot:spring-boot-starter-test.
- Bottom Bar:** Shows tabs for pom.xml, pom.xml (demo), and other files like .gitignore, .iml, and mvnw.

# Create best-practices packages on

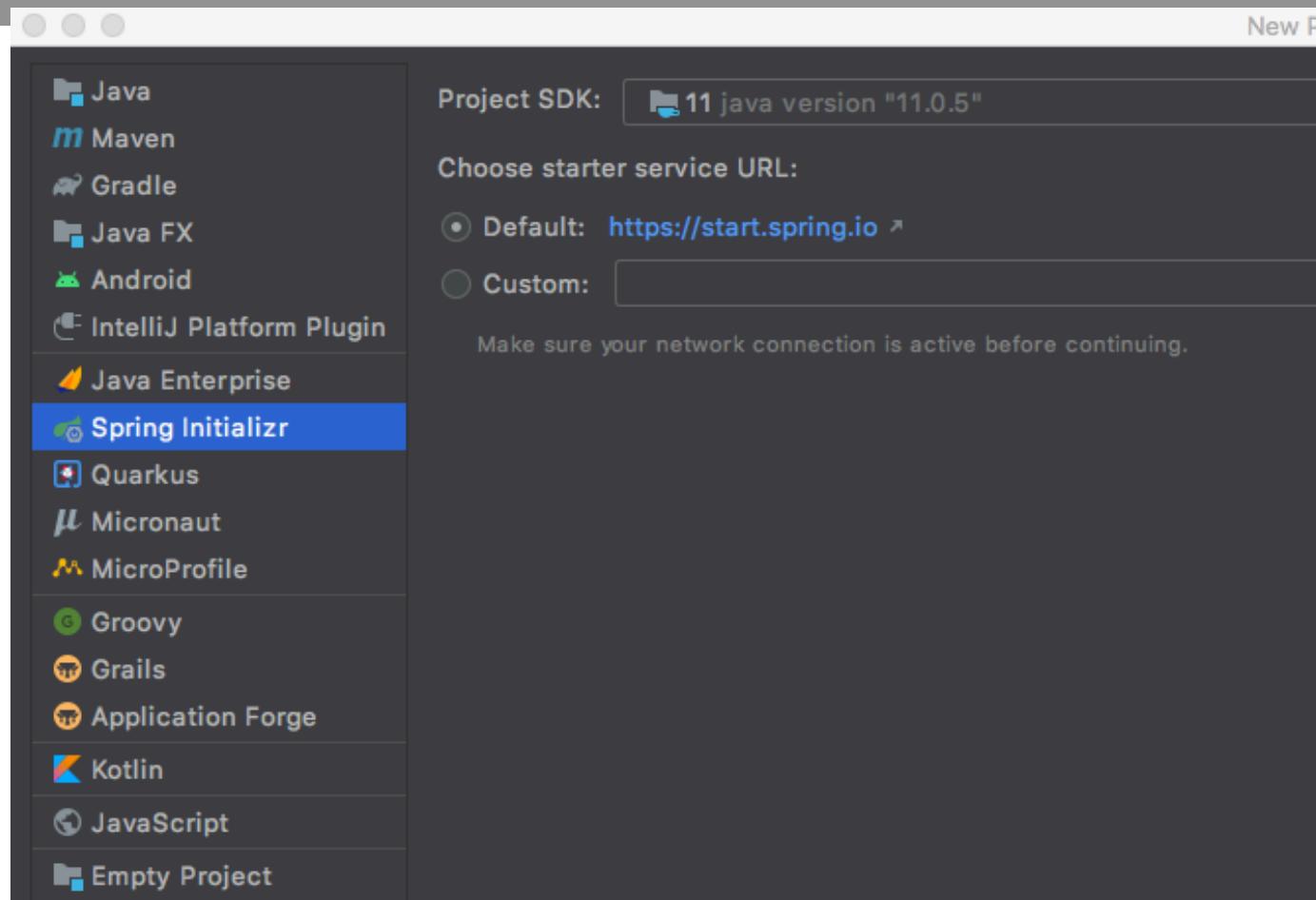


- controllers for the API
- models for JPA Entities or persistence information
- repositories for JPA access
- services for logic based code

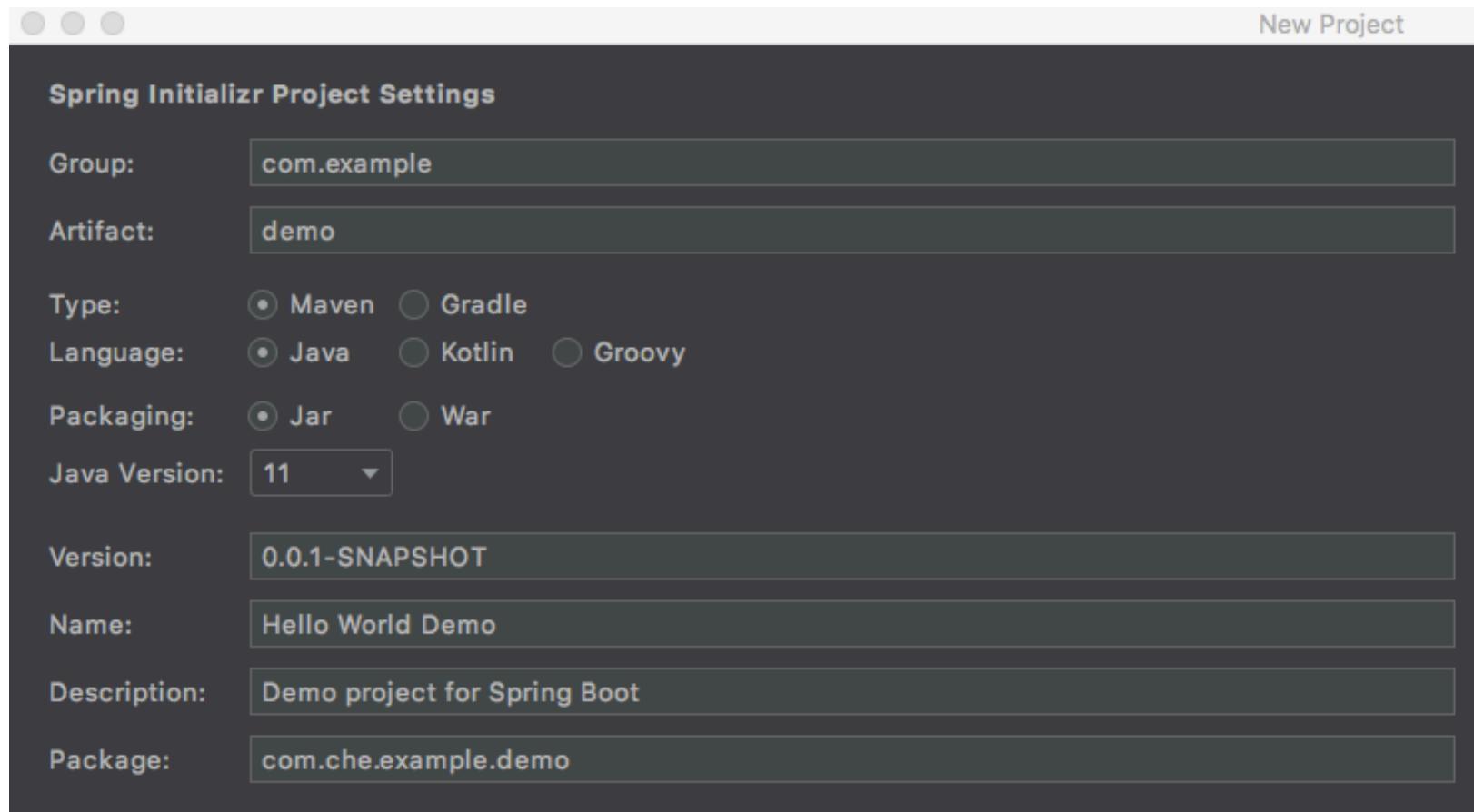
# In IntelliJ Ultimate create a new project



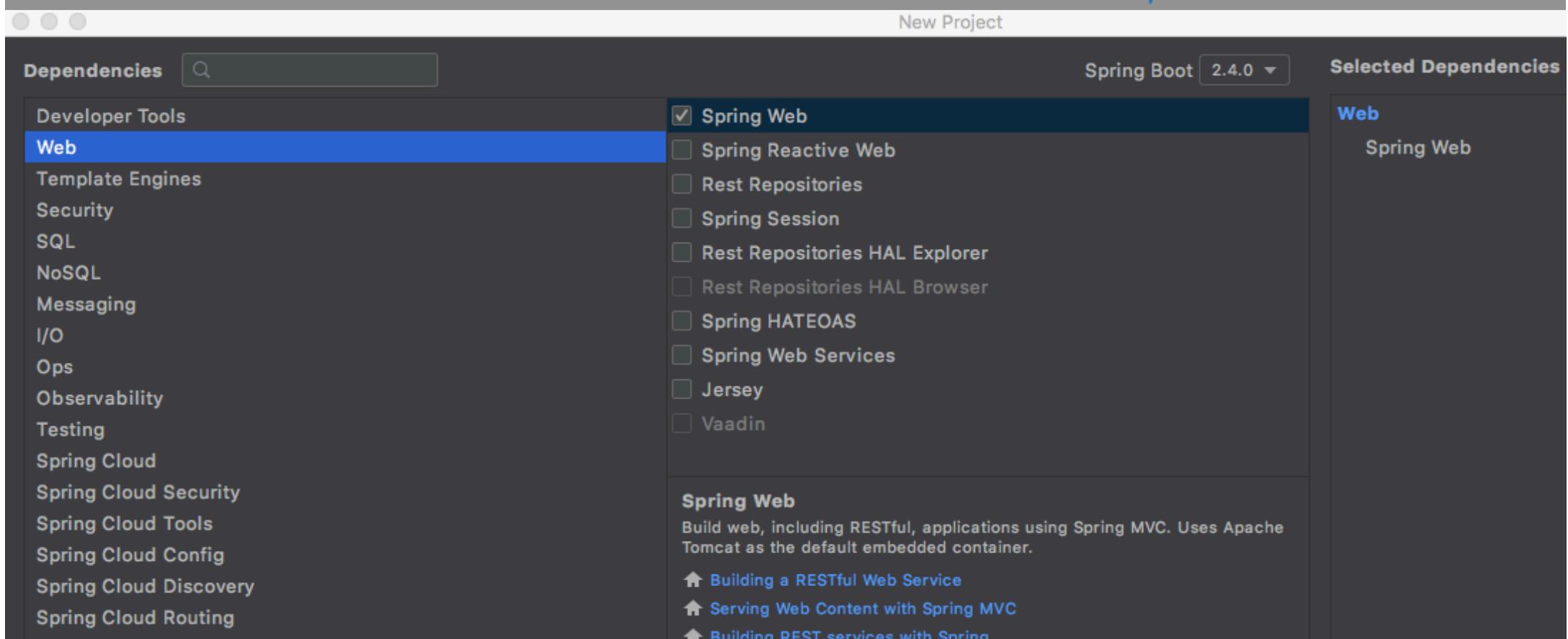
# Choose the Initializr wizard



# configure your project



# Select needed dependencies



The screenshot shows a dependency selection interface for a Spring Boot project. The left sidebar lists categories like Developer Tools, Web, Template Engines, Security, SQL, NoSQL, Messaging, I/O, Ops, Observability, Testing, Spring Cloud, Spring Cloud Security, Spring Cloud Tools, Spring Cloud Config, Spring Cloud Discovery, and Spring Cloud Routing. The 'Web' category is selected and highlighted in blue. The main panel displays a list of dependencies under the 'Web' category, with 'Spring Web' checked. A detailed description of 'Spring Web' is shown below, mentioning it's for building RESTful applications using Spring MVC and Apache Tomcat. The right panel shows the 'Selected Dependencies' list, which currently contains 'Spring Web'.

Dependencies

New Project Spring Boot 2.4.0 ▾

Selected Dependencies

Developer Tools

**Web**

Template Engines

Security

SQL

NoSQL

Messaging

I/O

Ops

Observability

Testing

Spring Cloud

Spring Cloud Security

Spring Cloud Tools

Spring Cloud Config

Spring Cloud Discovery

Spring Cloud Routing

Spring Web

Spring Reactive Web

Rest Repositories

Spring Session

Rest Repositories HAL Explorer

Rest Repositories HAL Browser

Spring HATEOAS

Spring Web Services

Jersey

Vaadin

**Spring Web**  
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

[Building a RESTful Web Service](#)

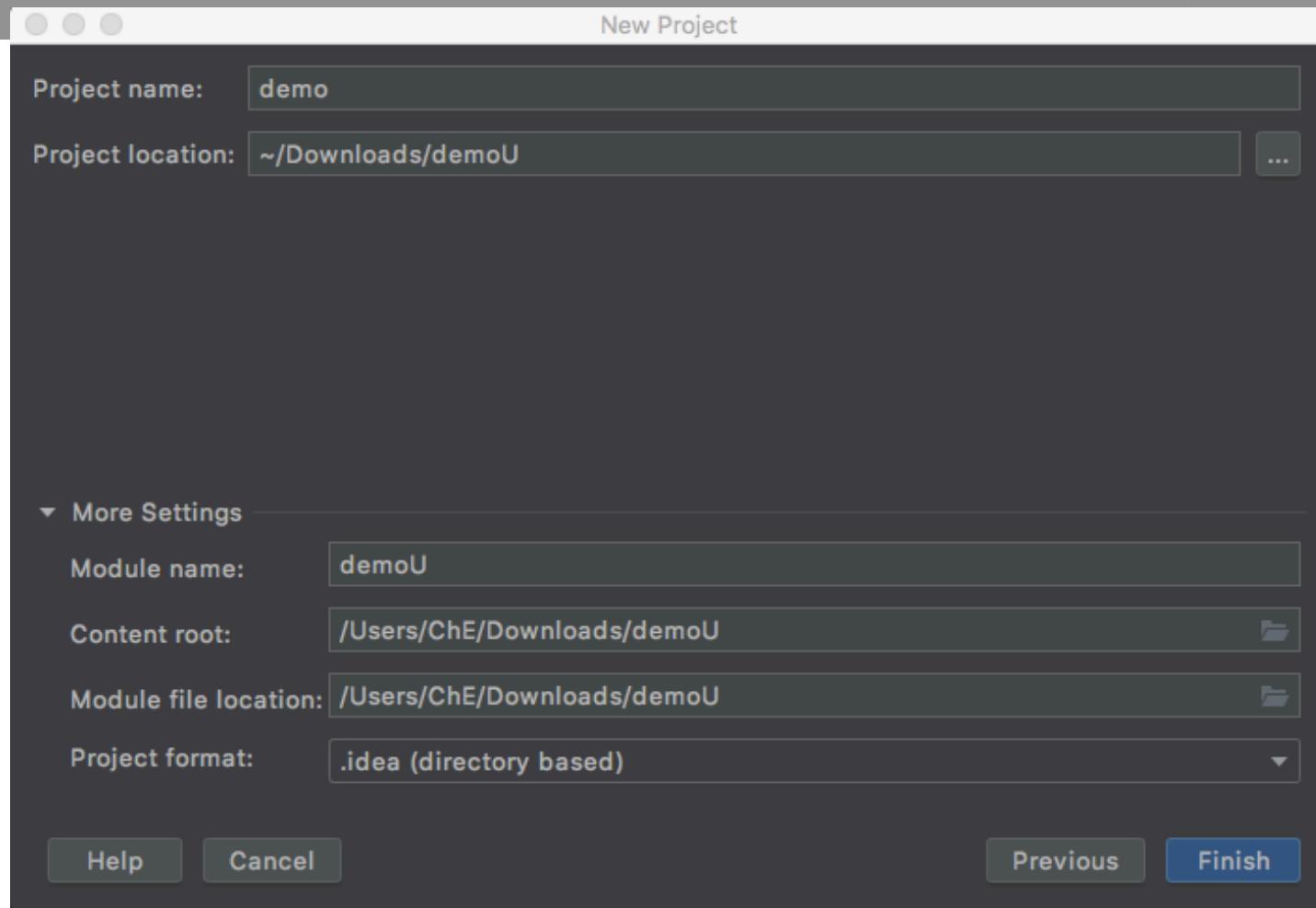
[Serving Web Content with Spring MVC](#)

[Building REST services with Spring](#)

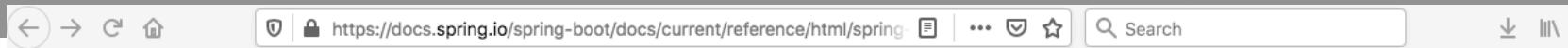
**Web**

Spring Web

# And finish to get the same result



# Or use the CLI



[Back to index](#)

## 1. Installing the CLI

- 2. Using the CLI
- 2.1. Running Applications with the CLI
- 2.2. Applications with Multiple Source Files
- 2.3. Packaging Your Application
- 2.4. Initialize a New Project
- 2.5. Using the Embedded Shell
- 2.6. Adding Extensions to the CLI
- 3. Developing Applications with the Groovy Beans DSL
- 4. Configuring the CLI with settings.xml
- 5. What to Read Next

## 1. Installing the CLI

The Spring Boot CLI (Command-Line Interface) can be installed manually by using SDKMAN! (the SDK Manager), Homebrew or MacPorts if you are an OSX user. See [getting-started.html](#) in the "Getting started" section for installation instructions.

## 2. Using the CLI

Once you have installed the CLI, you can run it by typing `spring` and pressing Enter at the command line. If no arguments are provided, a help screen is displayed, as follows:

```
$ spring
usage: spring [--help] [--version]
               <command> [<args>]
```

Available commands are:

```
run [options] <files> [--] [args]
  Run a spring groovy script
```

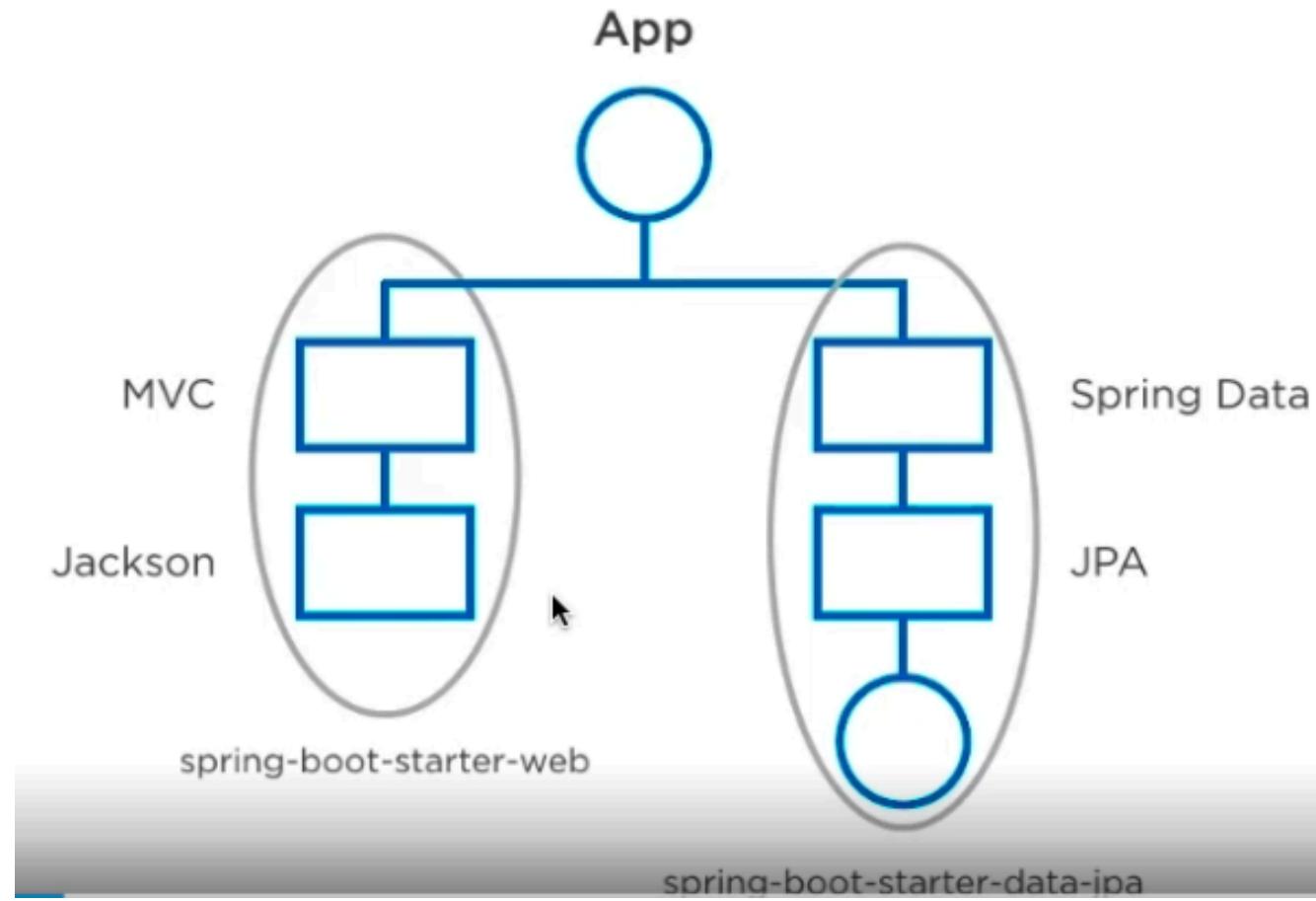
*... more command help is shown here*

You can type `spring help` to get more details about any of the supported commands, as shown in the following example:

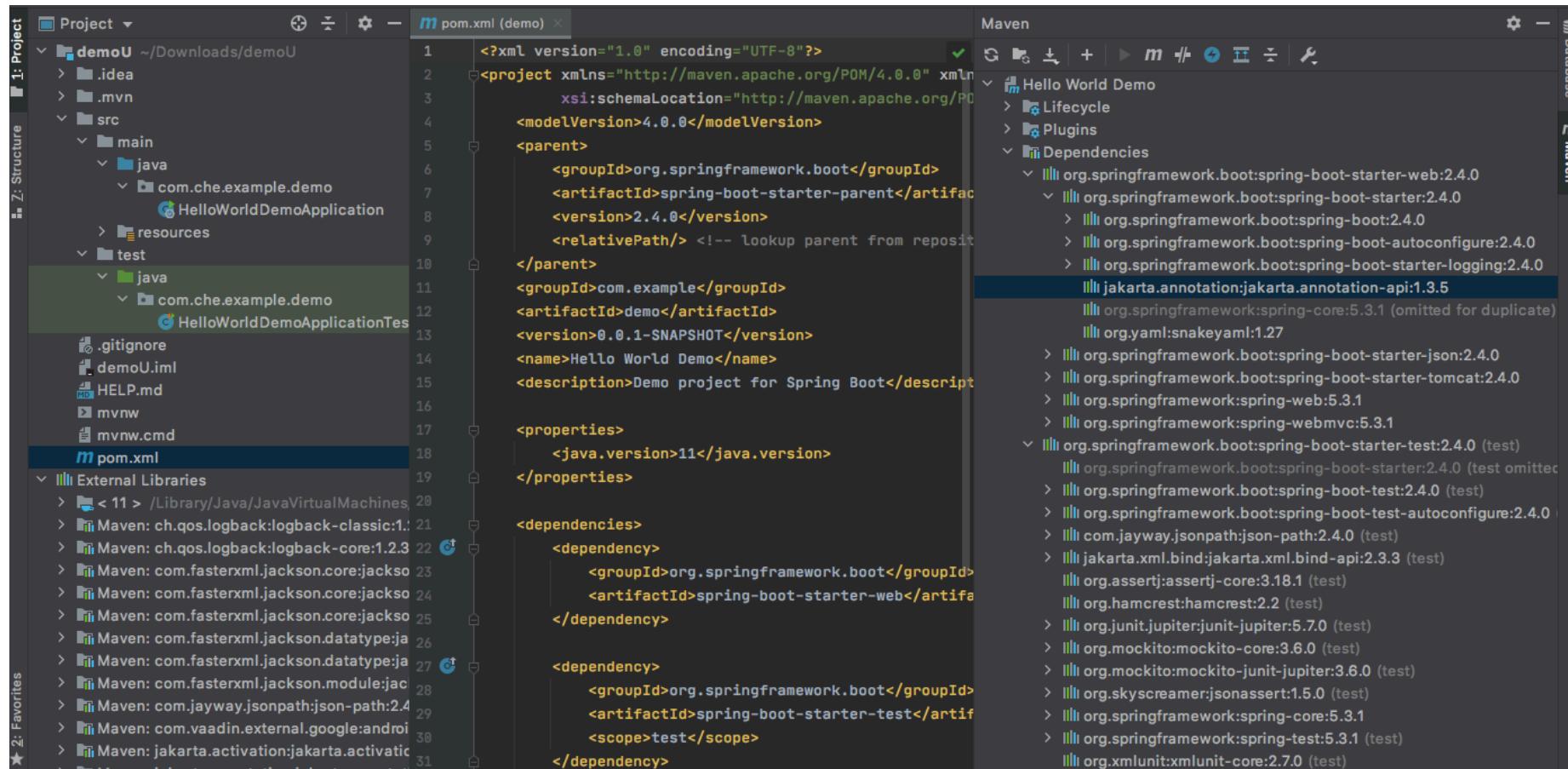
```
$ spring help run
spring run - Run a spring groovy script

usage: spring run [options] <files> [--] [args]
```

# Spring Boot Starters



# Understanding Spring Boot Starters



The screenshot shows a Java project named "demoU" in an IDE. The project structure includes a src directory with main and test packages, and a resources directory. The pom.xml file is open in the center, showing the XML configuration for a Spring Boot application. The Maven dependencies tab on the right lists the transitive dependencies for the project, including various versions of org.springframework.boot:spring-boot-starter-\* and other framework components like logback and json-path.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://maven.apache.org/POM/4.0.0"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/pom-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.4.0</version>
        <relativePath/> 
    </parent>
    <groupId>com.example</groupId>
    <artifactId>demo</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>Hello World Demo</name>
    <description>Demo project for Spring Boot</description>
    <properties>
        <java.version>11</java.version>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>

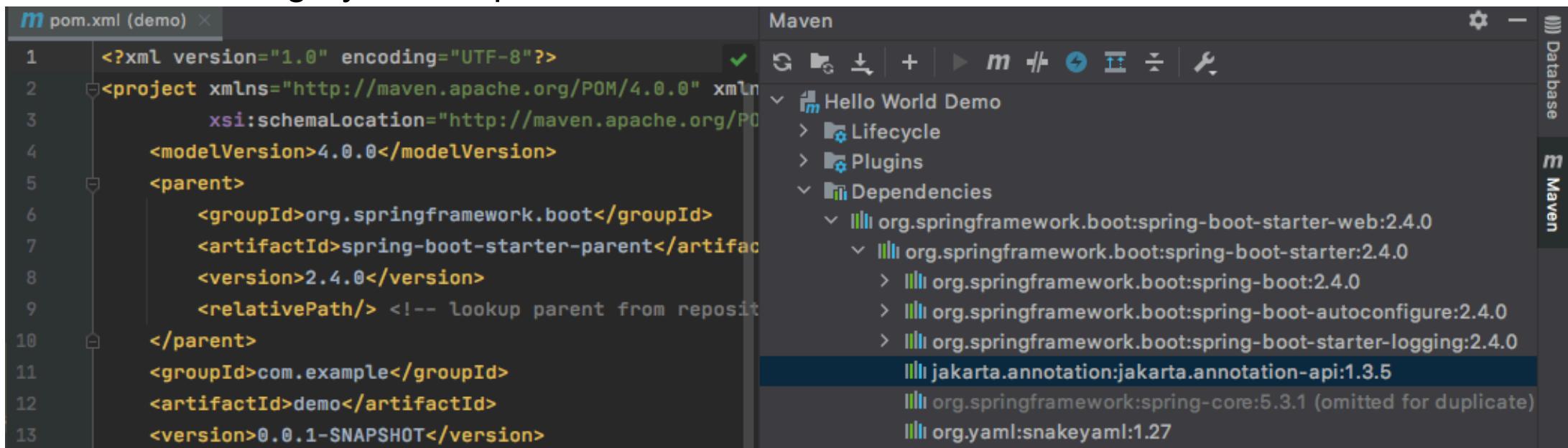
```

Maven

- Dependencies
  - org.springframework.boot:spring-boot-starter-web:2.4.0
    - org.springframework.boot:spring-boot-starter:2.4.0
    - org.springframework.boot:spring-boot:2.4.0
    - org.springframework.boot:spring-boot-autoconfigure:2.4.0
    - org.springframework.boot:spring-boot-starter-logging:2.4.0
    - jakarta.annotation:jakarta.annotation-api:1.3.5
    - org.springframework:spring-core:5.3.1 (omitted for duplicate)
    - org.yaml:snakeyaml:1.27
  - org.springframework.boot:spring-boot-starter-json:2.4.0
  - org.springframework.boot:spring-boot-starter-tomcat:2.4.0
  - org.springframework:spring-web:5.3.1
  - org.springframework:spring-webmvc:5.3.1
- org.springframework.boot:spring-boot-starter-test:2.4.0 (test)
  - org.springframework.boot:spring-boot-starter:2.4.0 (test omitted)
  - org.springframework.boot:spring-boot-test:2.4.0 (test)
  - org.springframework.boot:spring-boot-test-autoconfigure:2.4.0
  - com.jayway.jsonpath:json-path:2.4.0 (test)
  - jakarta.xml.bind:jakarta.xml.bind-api:2.3.3 (test)
  - org.assertj:assertj-core:3.18.1 (test)
  - org.hamcrest:hamcrest:2.2 (test)
  - org.junit.jupiter:junit-jupiter:5.7.0 (test)
  - org.mockito:mockito-core:3.6.0 (test)
  - org.mockito:mockito-junit-jupiter:3.6.0 (test)
  - org.skyscreamer:jsonassert:1.5.0 (test)
  - org.springframework:spring-core:5.3.1
  - org.springframework:spring-test:5.3.1 (test)
  - org.xmlunit:xmlunit-core:2.7.0 (test)

# Boot Starters offer

- Dependency Integration
- Versioning by starter parent



The screenshot shows a Maven project named "Hello World Demo" in an IDE. The left pane displays the `pom.xml` file content:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.4.0</version>
        <relativePath/> 
    </parent>
    <groupId>com.example</groupId>
    <artifactId>demo</artifactId>
    <version>0.0.1-SNAPSHOT</version>

```

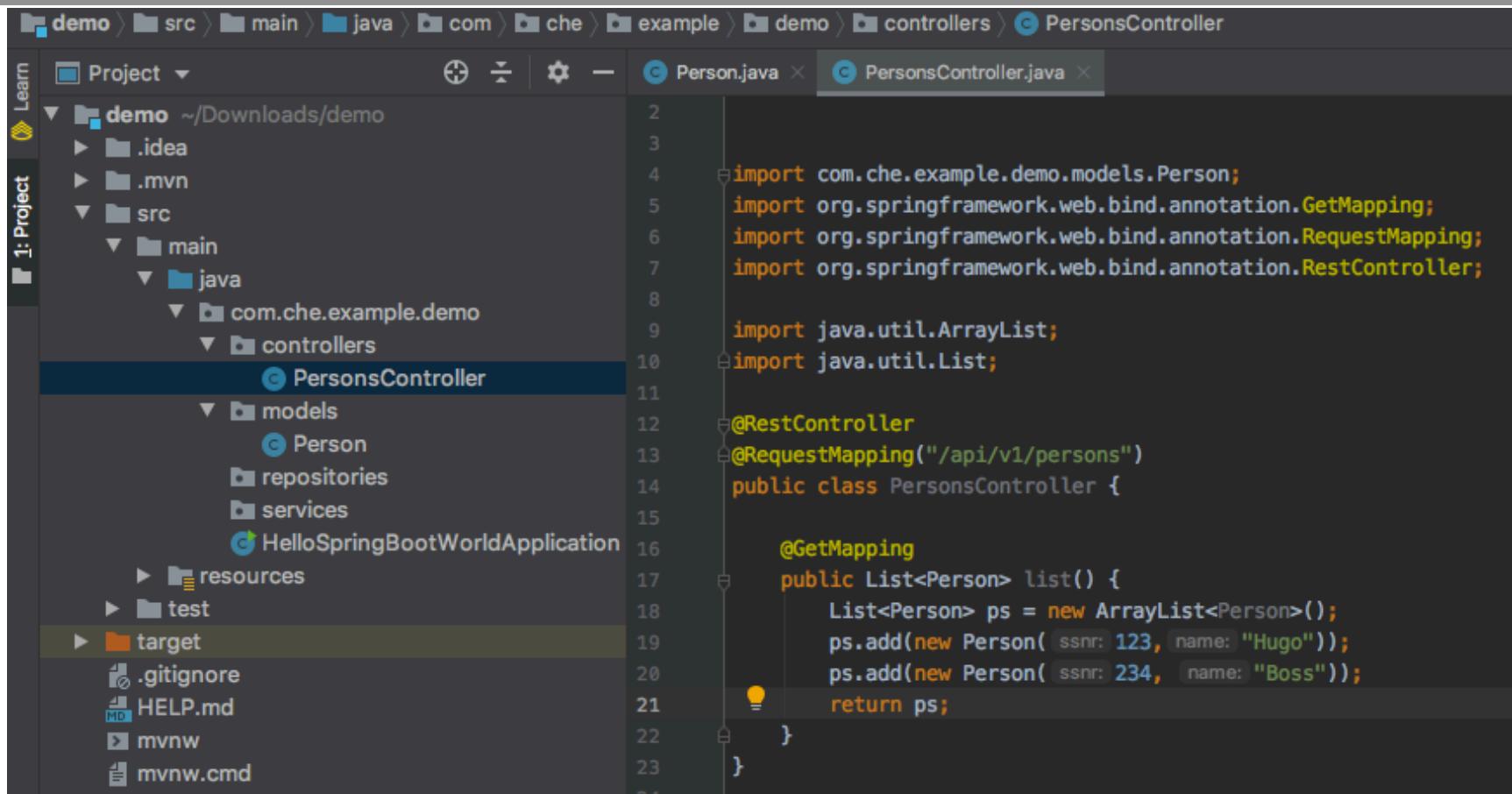
The right pane shows the Maven dependency tree. The tree starts with the parent dependency (`org.springframework.boot:spring-boot-starter-parent:2.4.0`) and includes its children: `org.springframework.boot:spring-boot-starter-web:2.4.0`, `org.springframework.boot:spring-boot-starter:2.4.0`, `org.springframework.boot:spring-boot:2.4.0`, `org.springframework.boot:spring-boot-autoconfigure:2.4.0`, `org.springframework.boot:spring-boot-starter-logging:2.4.0`, `jakarta.annotation:jakarta.annotation-api:1.3.5`, `org.springframework:spring-core:5.3.1 (omitted for duplicate)`, and `org.yaml:snakeyaml:1.27`.

# Add POJO models using Cmd/Ctrl+N

A screenshot of a Java IDE (IntelliJ IDEA) showing the code for a POJO model named Person. The code defines a class with private attributes ssnr and name, and methods for their getters and setters. The IDE's project structure on the left shows the file is located in the com.che.example.demo.models package under the demo project.

```
demo [~/Downloads/demo] - .../src/main/java/com  
demo > src > main > java > com > che > example > demo > models > Person  
Project Learn 1: Project  
demo ~/Downloads/demo  
|.idea  
.mvn  
src  
| main  
|> java  
|> com.che.example.demo  
|> controllers  
|> models  
|> Person  
|> repositories  
|> services  
HelloSpringBootWorldApplication  
|.resources  
test  
target  
.gitignore  
HELP.md  
mvnw  
mvnw.cmd  
pom.xml  
External Libraries  
Scratches and Consoles  
demo [~/Downloads/demo] - .../src/main/java/com  
Person.java < PersonsController.java  
1 package com.che.example.demo.models;  
2  
3 public class Person {  
4  
5     private int ssnr = -1;  
6     private String name = null;  
7  
8     @  
9     public Person(int ssnr, String name) {  
10         this.ssnr = ssnr;  
11         this.name = name;  
12     }  
13  
14     public int getSSNR() {  
15         return ssnr;  
16     }  
17  
18     public void setSSNR(int ssnr) {  
19         this.ssnr = ssnr;  
20     }  
21  
22     public String getName() {  
23         return name;  
24     }  
25  
26     public void setName(String name) {  
27     }
```

# Add controllers



The screenshot shows a Java code editor with two tabs open: Person.java and PersonsController.java. The PersonsController.java tab is active, displaying the following code:

```
import com.che.example.demo.models.Person;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import java.util.ArrayList;
import java.util.List;

@RestController
@RequestMapping("/api/v1/people")
public class PersonsController {

    @GetMapping
    public List<Person> list() {
        List<Person> ps = new ArrayList<Person>();
        ps.add(new Person( ssnr: 123, name: "Hugo"));
        ps.add(new Person( ssnr: 234, name: "Boss"));
        return ps;
    }
}
```

The project structure on the left shows a Spring Boot application named 'demo'. The 'src' directory contains 'main' which has 'java' and 'resources' sub-directories. The 'java' directory contains 'com.che.example.demo' which has 'controllers' and 'models' sub-directories. The 'controllers' directory contains 'PersonsController.java'. Other files in 'src/main/java' include 'HelloSpringBootWorldApplication.java', 'Person.java', and 'repositories'. The 'resources' directory contains 'application.properties'. The 'target' directory is also visible.

# Build, start, check Tomcat and get “nothing”

The screenshot shows a Java development environment with three tabs open: Person.java, PersonsController.java, and HelloSpringBootWorldApplication.java. The code in HelloSpringBootWorldApplication.java is:

```
package com.che.example.demo;  
import ...  
@SpringBootApplication  
public class HelloSpringBootWorldApplication {  
    public static void main(String[] args) { SpringApplication.run(HelloSpringBootWorldApplication.class, args); }  
}
```

Below the IDE is a browser window with the URL `localhost:8080`. The page displayed is a "Whitelabel Error Page" with the message: "This application has no explicit mapping for /error, so you are seeing this as a fallback." It also shows the timestamp "Mon Nov 16 21:36:44 CET 2020" and the error message "There was an unexpected error (type=Not Found, status=404)".

At the bottom of the browser window, there is a log output from the application:

```
INFO 5347 --- [           main] c.c.e.d>HelloSpringBootWorldApplication : Starting HelloSpringBootWorldApplication using Java 11.0.5 on ChEMc.local with PID  
INFO 5347 --- [           main] c.c.e.d>HelloSpringBootWorldApplication : No active profile set, falling back to default profiles: default  
INFO 5347 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)  
INFO 5347 --- [           main] o.apache.catalina.core.StandardService : Starting service [Tomcat]  
INFO 5347 --- [           main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.39]  
INFO 5347 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext  
INFO 5347 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 1397 ms  
INFO 5347 --- [           main] o.s.s.concurrent.ThreadPoolTaskExecutor : Initializing ExecutorService 'applicationTaskExecutor'  
INFO 5347 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''  
INFO 5347 --- [           main] c.c.e.d>HelloSpringBootWorldApplication : Started HelloSpringBootWorldApplication in 3.076 seconds (JVM running for 3.821)  
INFO 5347 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'  
INFO 5347 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'  
INFO 5347 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 2 ms
```

# or success on supported routes

A screenshot of a web browser displaying a JSON response. The URL in the address bar is "localhost:8080/api/v1/persons". The response shows two objects, indexed 0 and 1, each with "ssnr" and "name" fields.

	ssnr	name
0:	123	"Hugo"
1:	234	"Boss"

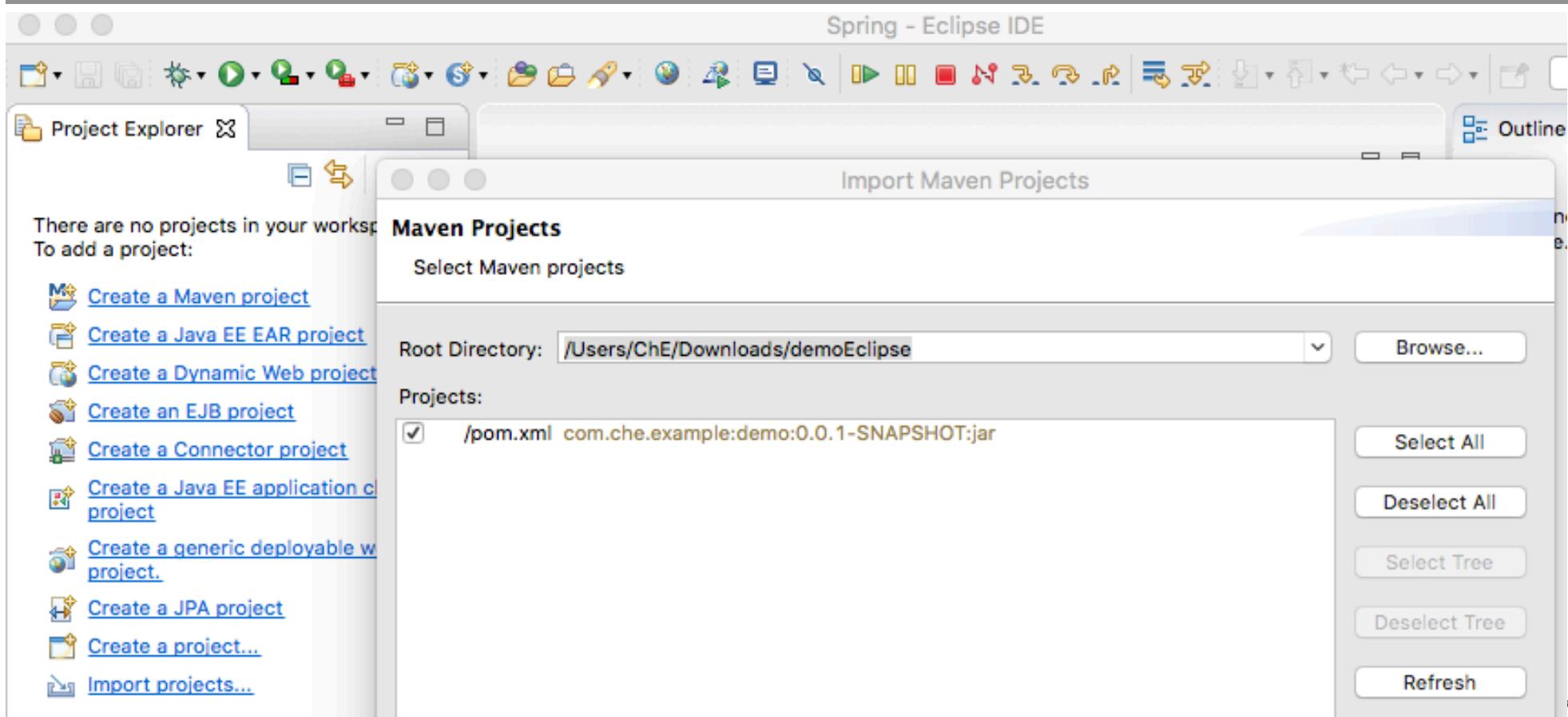
# Hello World without POJOs or API controllers

The screenshot shows a Java IDE interface with three tabs: Person.java, PersonsController.java, and HelloSpringBootWorldApplication.java. The HelloSpringBootWorldApplication.java tab is active, displaying the following code:

```
1 package com.che.example.demo;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5 import org.springframework.web.bind.annotation.GetMapping;
6 import org.springframework.web.bind.annotation.RequestParam;
7 import org.springframework.web.bind.annotation.RestController;
8
9 @SpringBootApplication
10 @RestController
11 public class HelloSpringBootWorldApplication {
12
13     public static void main(String[] args) { SpringApplication.run(HelloSpringBootWorldApplication.class, args); }
14
15     @GetMapping("/api/v1/hello")
16     public String hello(@RequestParam(value = "name", defaultValue = "World") String name) {
17         return String.format("Hello %s!", name);
18     }
19 }
```

Two browser windows are open, both showing the URL `localhost:8080/api/v1/hello`. The top window displays the response "Hello World!". The bottom window displays the response "Hello Sepp!" after a parameter was added to the URL: `localhost:8080/api/v1/hello?name=Sepp`.

# Btw, import extracted zip into Eclipse



# And start it

## Hello Eclipse!

**software  
inside**

# Open Questions / Back to the roots on

- Annotations
- Dependency Injections
- Builder Patterns
- Build Tools
- ..
- understanding concepts, best practices, techniques, methods, .., tools (in context with teaching top-down or bottom-up)