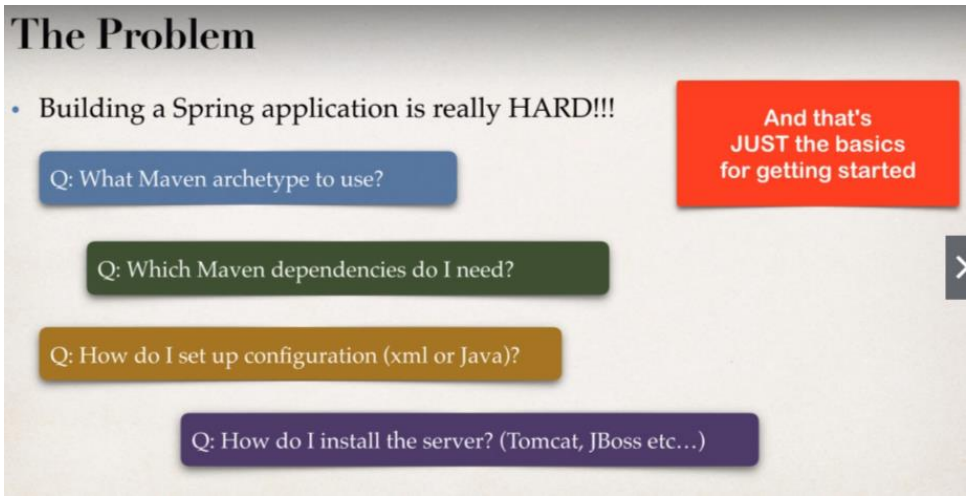
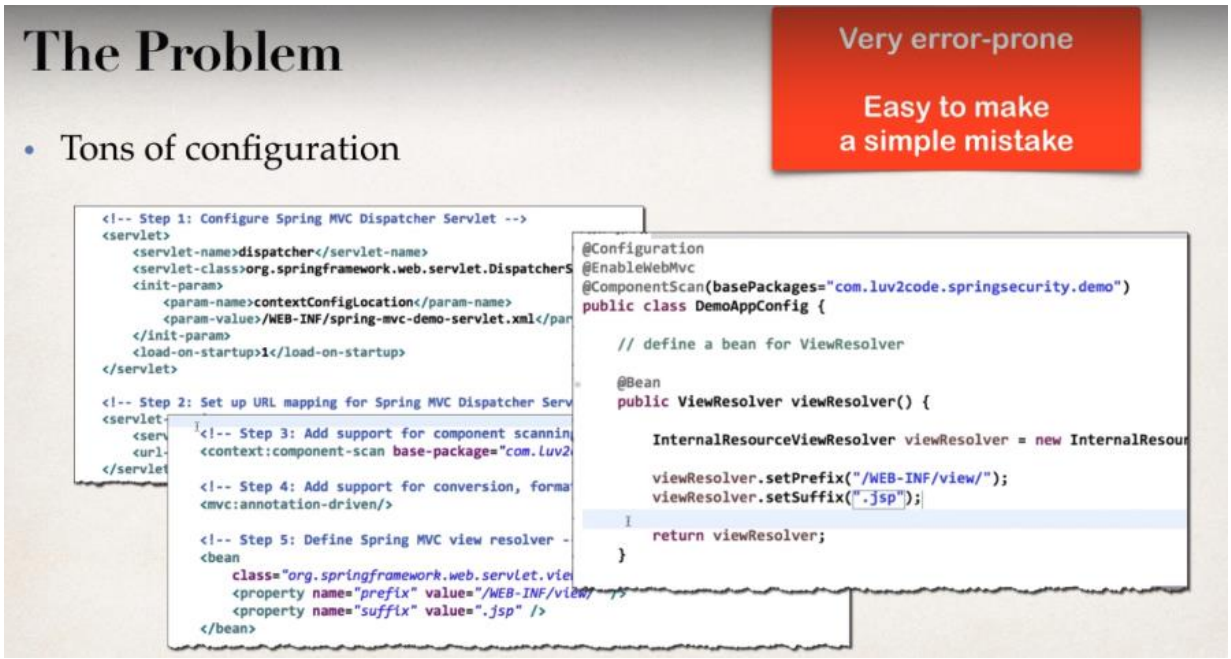
Section 70: Spring Boot - Overview

0 / 7 | 52min





Spring Boot is the 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.
    - Out of box we can make use of Tomcat, Jetty, Undertow,…

Spring boot provides the spring initializer, which helps to quickly create a starter Spring Project.

Spring Initializr

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

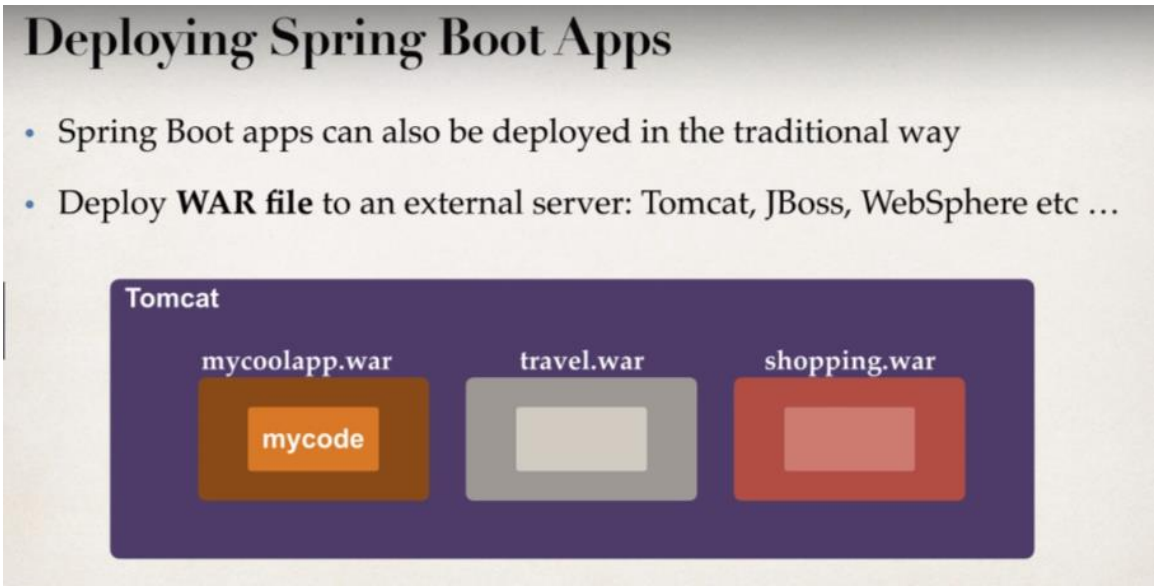
http://start.spring.io

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

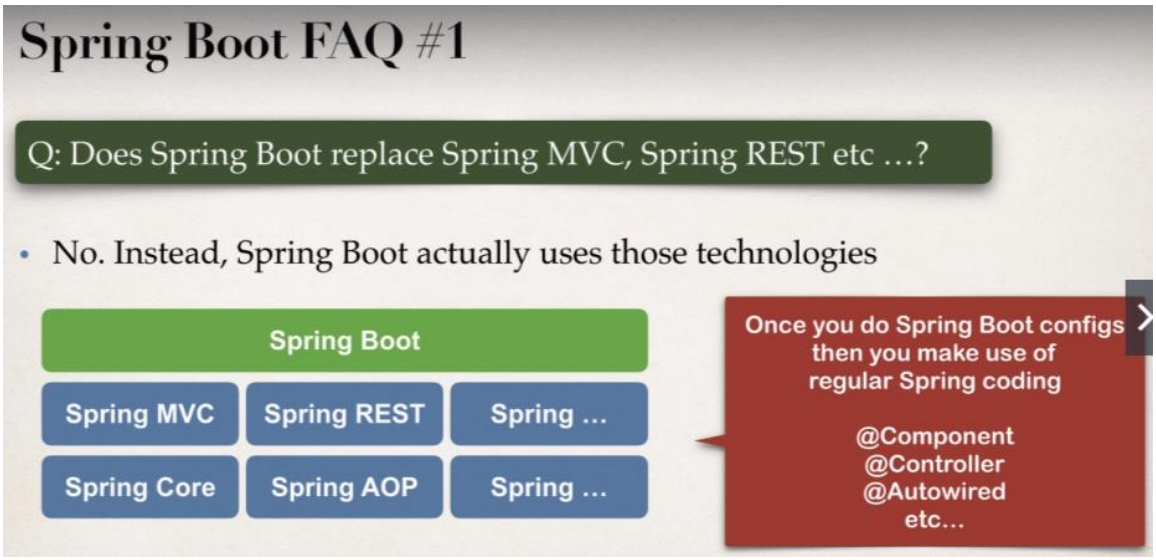


- **Spring Boot Initializer Demo**

1. Configure our project at Spring Initializer website

   Visit the website: https://start.spring.io/
2. Download the zip file
3. Unzip the file
4. Import Maven project into the IDE.

We will go to the file annotated with **`@SpringBootApplication`** annotation and run as **java application.**

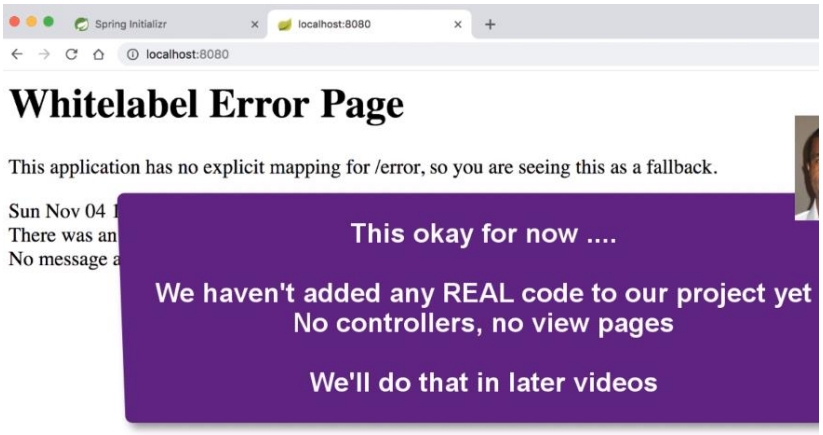


MydemoappApplication.java

mydemoapp ▸ src/main/java ▸ com.example.demo.mydemoapp ▸ MydemoappApplication

```java
package com.example.demo.mydemoapp;

import org.springframework.boot.SpringApplication;

@SpringBootApplication
public class MydemoappApplication {

    public static void main(String[] args) {
        SpringApplication.run(MydemoappApplication.class, args);
    }

}
```

Output:



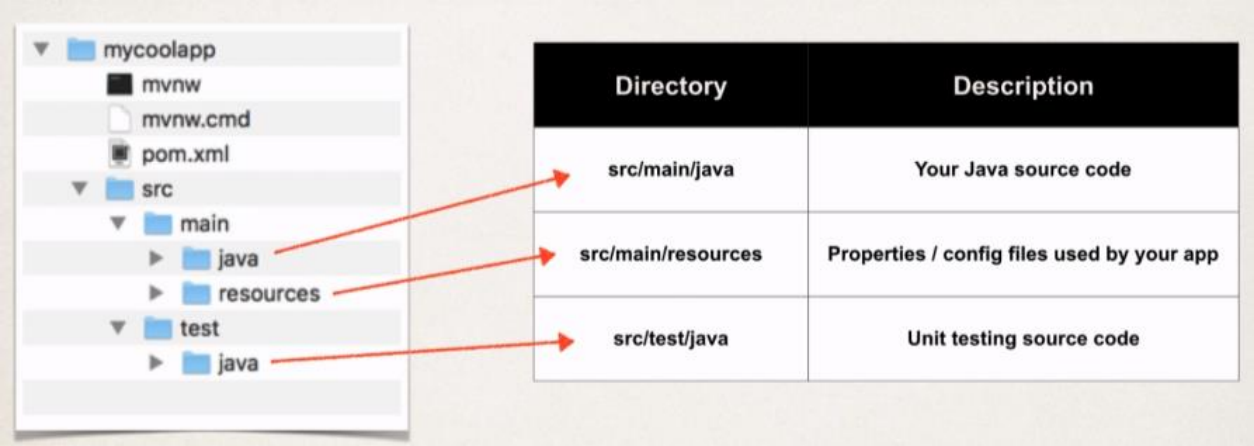
- **Create a REST controller with Spring Boot**

Output:

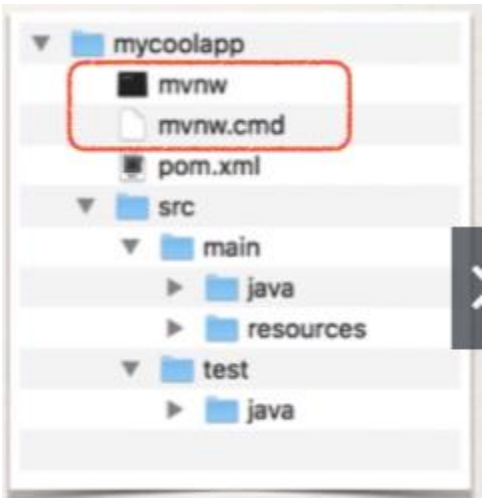
So, within no time we have successfully made our rest application and without performing any configuration.

➢ Exploring Spring Boot Project Structure:

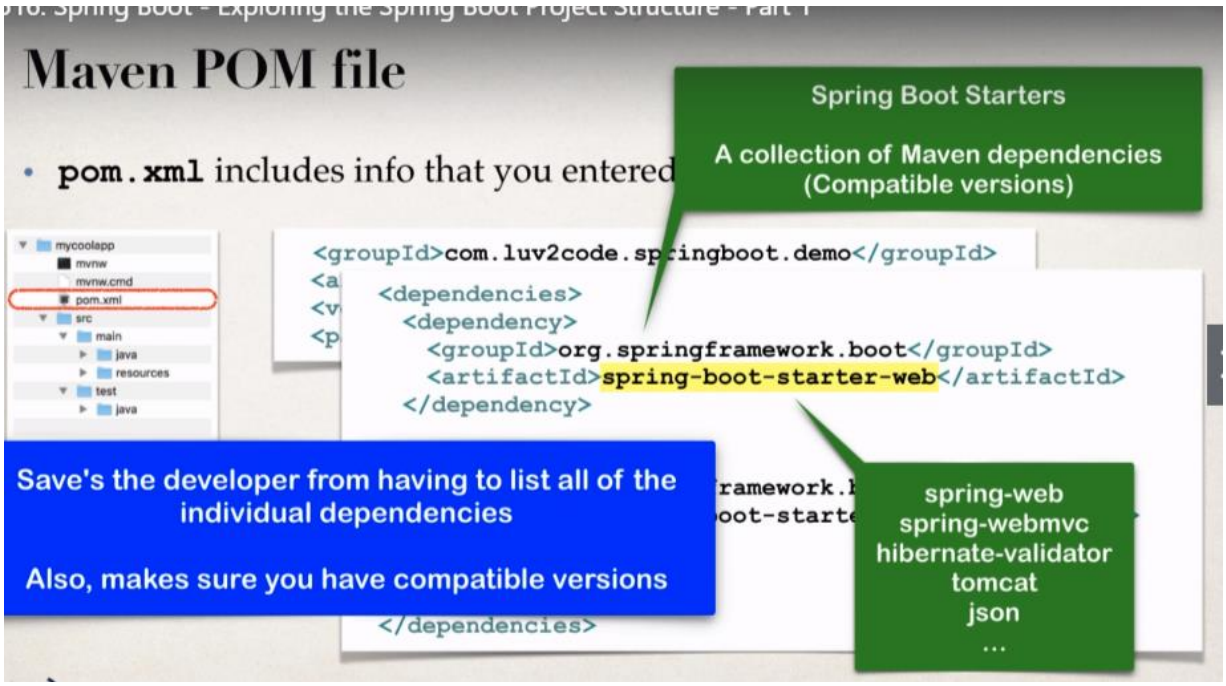
Spring boot make use of **Maven Standard Directory Structure.**

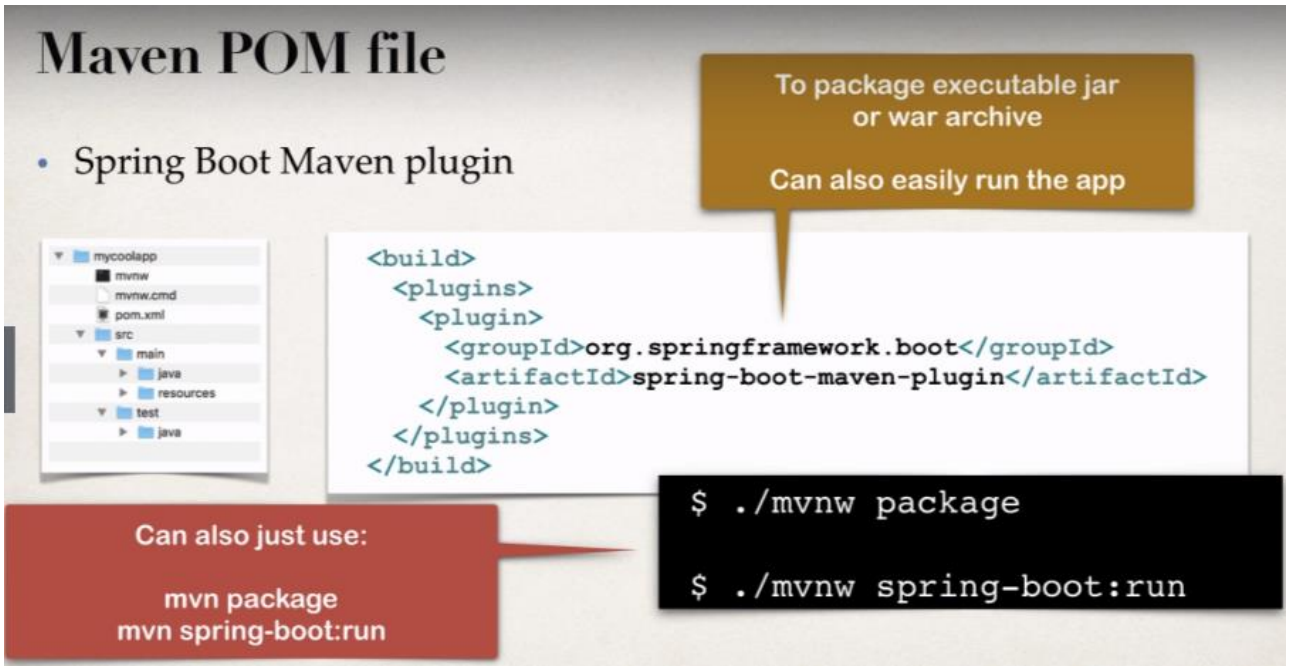


Maven Wrapper files:



- **`mvnw`** allows you to run a Maven Project.
- And there is no need to have Maven installed or present on your classpath.
- What they will do actually:
  - If correct version of Maven is NOT found on your computer
  - Then it will **`Automatically Downloads`** correct version and runs Maven.
- There are two files that are provided:
  - **mvnw.cmd** for MS windows.
  - **mvnw.sh** for linux/Mac
- If you already have Maven Installed previously.
- Then you can simply ignore/delete the **`mvnw`** files.

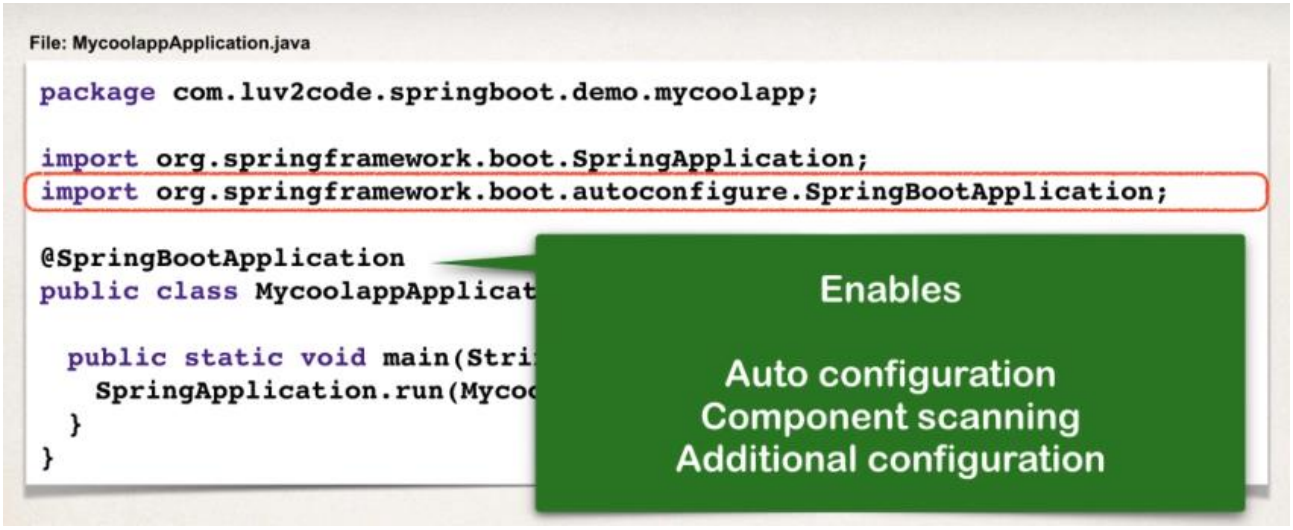




If maven is already installed in our system, then we can just use the below two commands:

```
mvn package
mvn spring-boot:run
```

Main Spring boot application class Created by Spring Initializer:



Annotations

- **`@SpringBootApplication`** is composed of the following annotations:

Annotation	Description
`@EnableAutoConfiguration`	Enables Spring Boot's auto-configuration support
`@ComponentScan`	Enables component scanning of current package Also recursively scans sub-packages
`@Configuration`	Able to register extra beans with `@Bean` or import other configuration classes

File: MycoolappApplication.java

```
package com.luv2code.springboot.demo.mycoolapp

import org.springframework.boot.SpringApplicat
import org.springframework.boot.autoconfigure.

@SpringBootApplication
public class MycoolappApplication {

  public static void main(String[] args) {
    SpringApplication.run(MycoolappApplication.class, args);
  }
}
```

Behind the scenes ...

Creates application context and registers all beans

Starts the embedded server Tomcat etc...

Bootstrap your Spring Boot application

More on Component Scanning

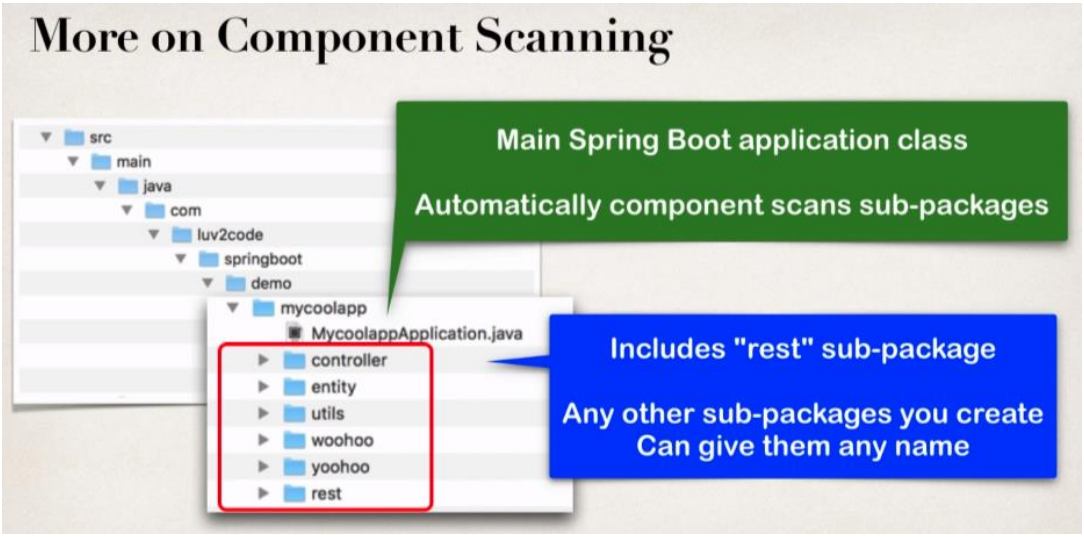
Best Practice

Place your main application class in the root package above your other packages

- This implicitly defines a base search package
  - Allows you to leverage default component scanning
  - No need to explicitly reference the base package name

Common pitfall in traditional Spring apps

Anyone use the wrong package name before in traditional Spring apps???

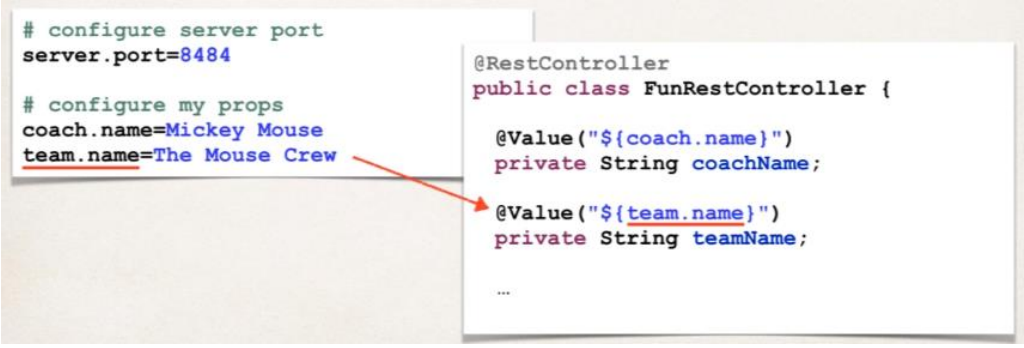


➢ **<u>Application Properties</u>:**

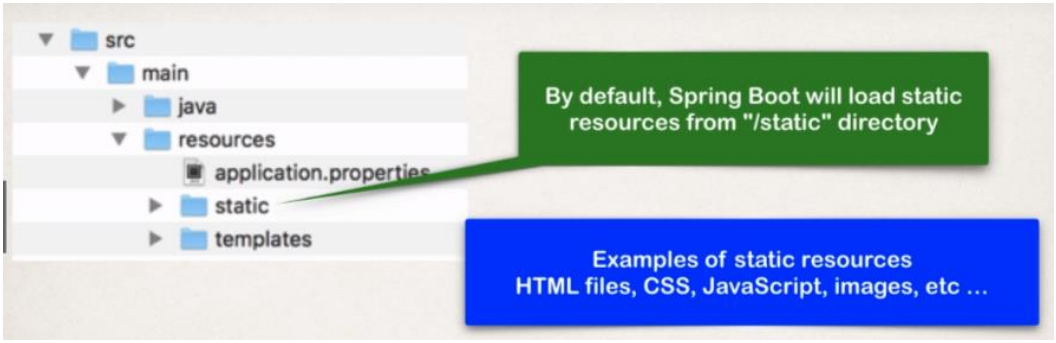
By default, Spring Boot will load properties from: **`application.properties`**

How to read data from **`Application.properties`** file:

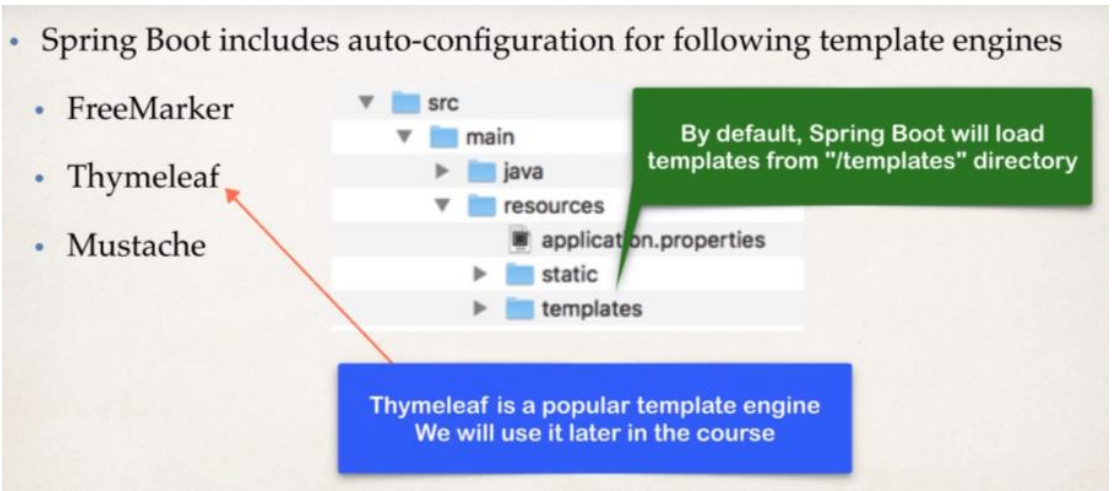
We can actually read this by making use of **`injection`**.



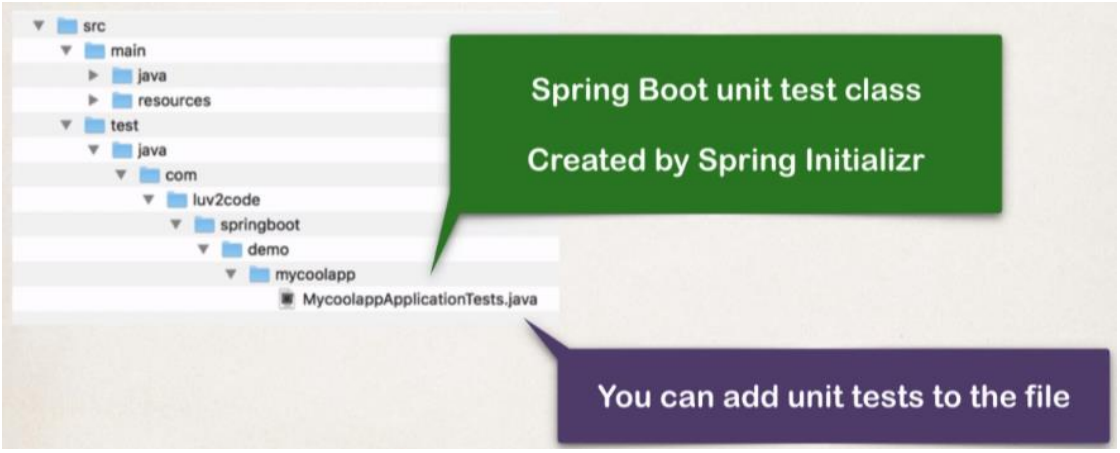
- **Static Content**



Templates



Unit Tests



- **Spring Boot Starters:**
  - A curated list of Maven Dependencies.
  - A collection of dependencies grouped together.
  - Tested and verified by the Spring Development team.
  - These starter makes it much easier for the developer to get started with spring.
  - Reduces the amount of Maven configuration that we need to do.

Let's consider an example of **Spring MVC**:

So, to build a Spring MVC app, you normally need dependencies;

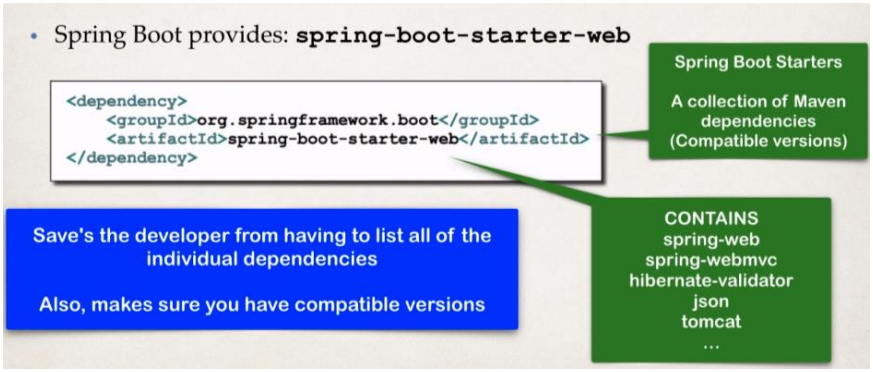
- Spring MVC
- Hibernate validator for some form validation
- And for a given web template, we need to put the dependency.

```
<!-- Spring MVC -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>5.1.2.RELEASE</version>
</dependency>

<!-- Form validation: Hibernate Validator -->
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-validator</artifactId>
    <version>6.0.13.Final</version>
</dependency>

<!-- Web template: JSP or Thymeleaf etc -->
<dependency>
    <!-- ... -->
</dependency>
```

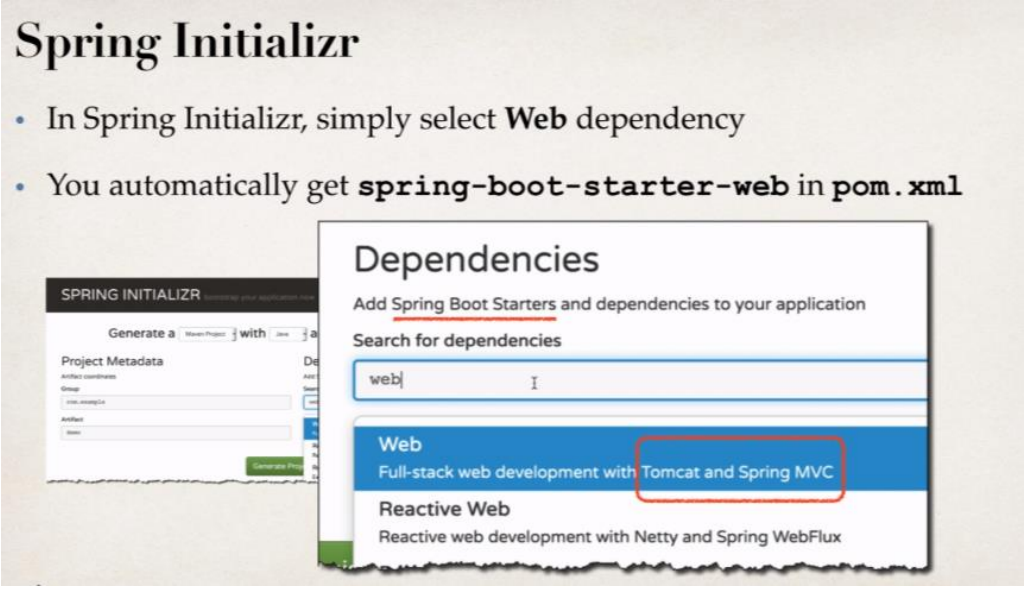
Here is the solution:

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

Add this Maven dependency to the Maven pom file. And this dependency entry actually contains other dependencies as it contains spring-web, spring-webmvc, hibernate-validator, json, tomcat.

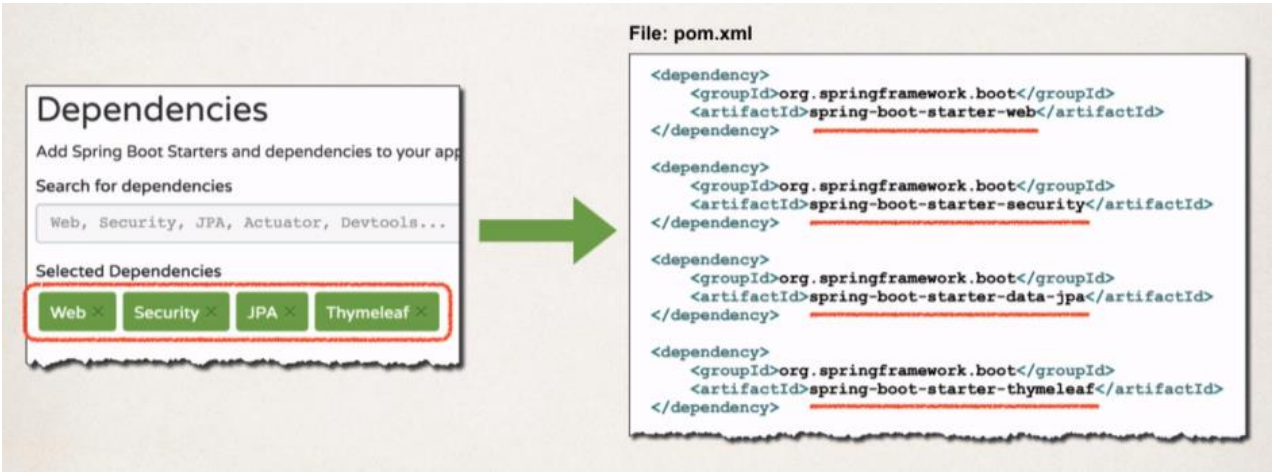


So, in the spring initializer we need to select web dependency:



Another Example:

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



Spring Boot Starters

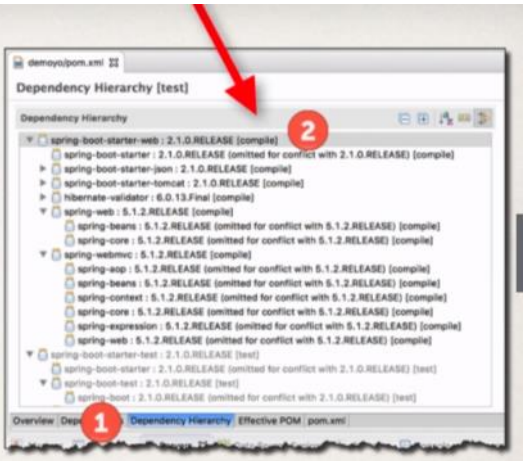
- There are 30+ Spring Boot Starters from the Spring Development team

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**
...	

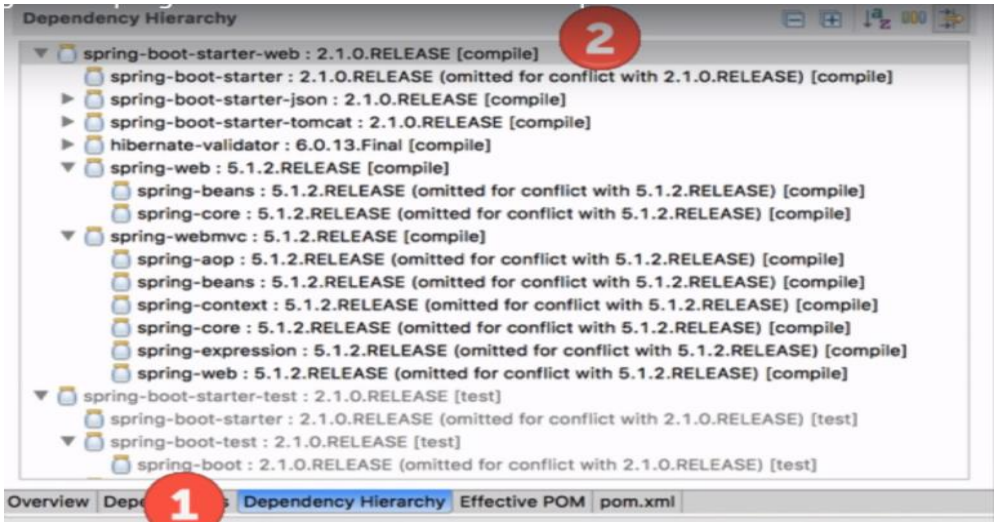
If we want a **`full list`** of Spring boot starters: Link

➢ **What is in the Starter dependency that we have just into pom.xml?**

- For Eclipse Users
- Open the pom.xml
- Select the tab **Dependency Hierarchy**
- Expand the desired starter



Ecilipse Way:



- **Spring Boot Starter Parent:**

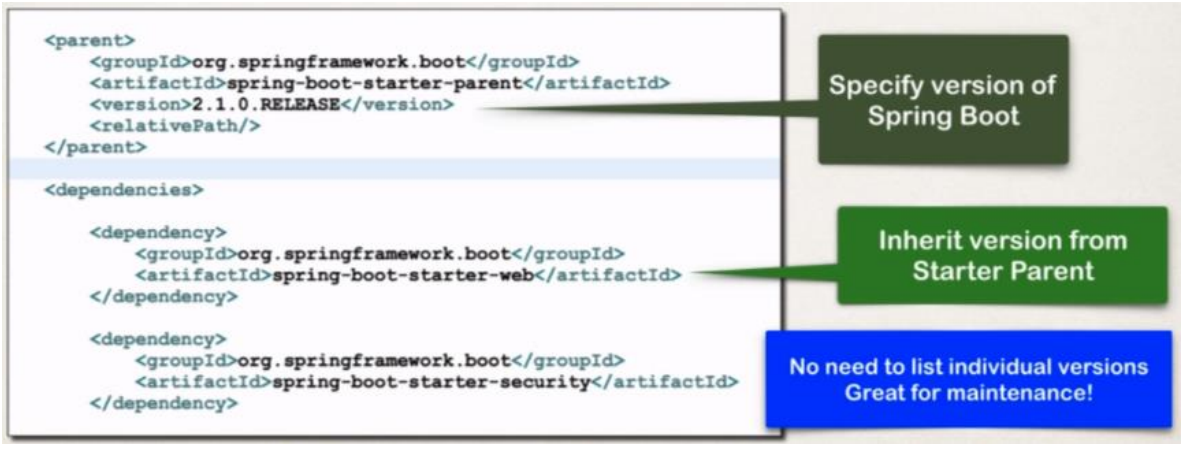
So, the Maven Defaults are actually defined in the Starter Parent.

- We get the Default compiler level
- Also have support for UTF- 8 source coding
- Others..

To override a default, set as a property

For ex in our project, we want to use java 12, or 15. then we can simply override that property.

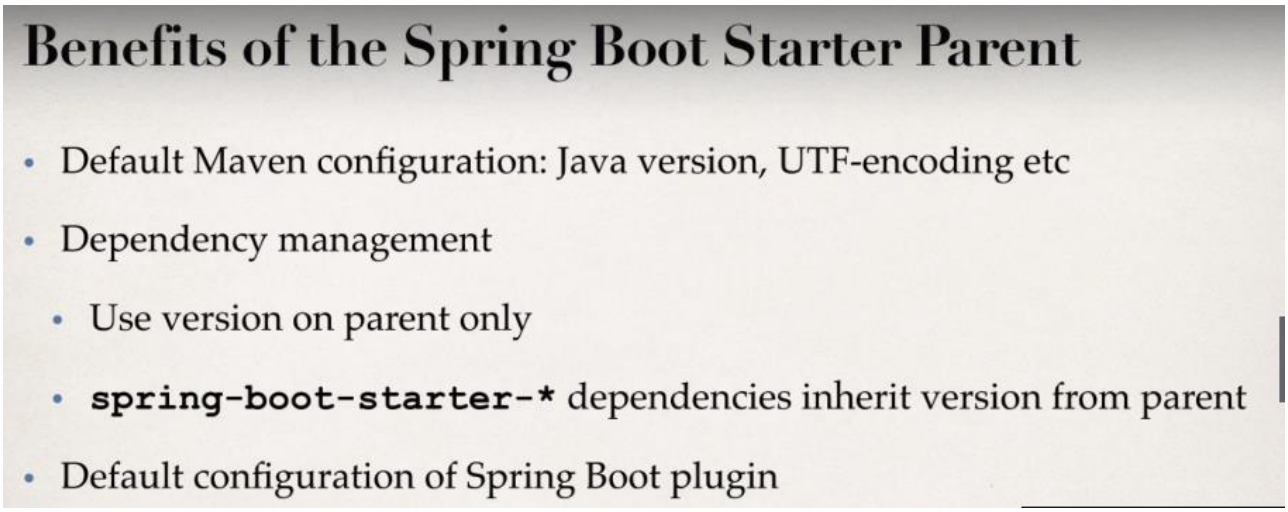
For **`spring-boot-starter-*`** dependencies, no need to list version.



Basically, we have here **`spring-boot-starter-parent`** where we specify the version that we are using and then for the actual dependencies, we simply inherit the version from the starter parent. So, there is no need to list individual versions and it's great for maintenance.

Starter parent also provides the default configuration of Spring boot plugin, in our build section of pom.xml file, we simply refer to the spring boot maven plugin and that's it. And now we can simply run it from our command line using the command:

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



Section 71: Spring Boot - Spring Boot Dev Tools and Spring Boot Actuator
0 / 4 | 35min

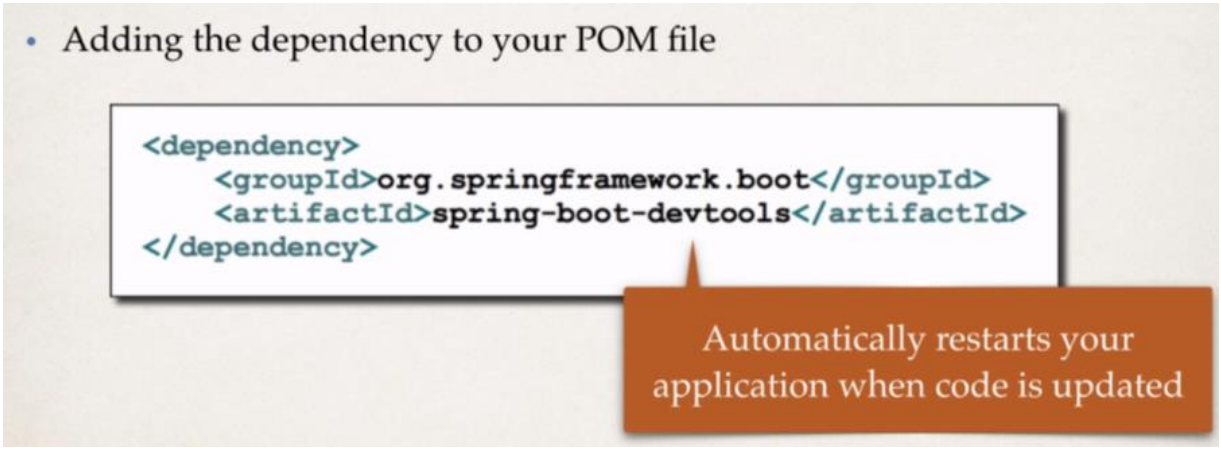
The Problem:

When running Spring Boot applications.

If we make changes to our source code, then we have to manually restart our application.

Solution: Spring boot Dev Tools

- **`Spring-boot-devtools`** to the rescue!
- Simply add the dependency to your pom.xml.



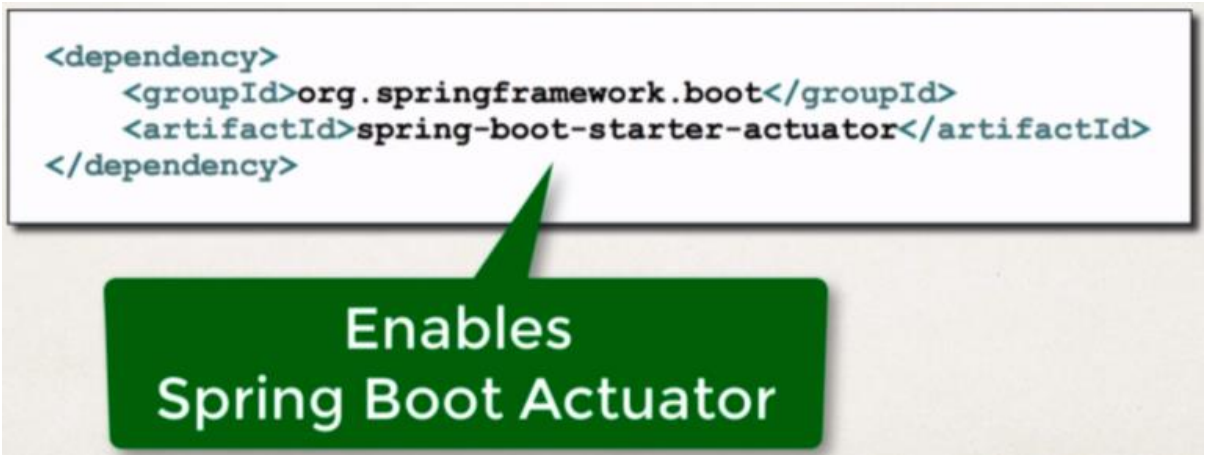
Spring boot Actuator Java Project Link

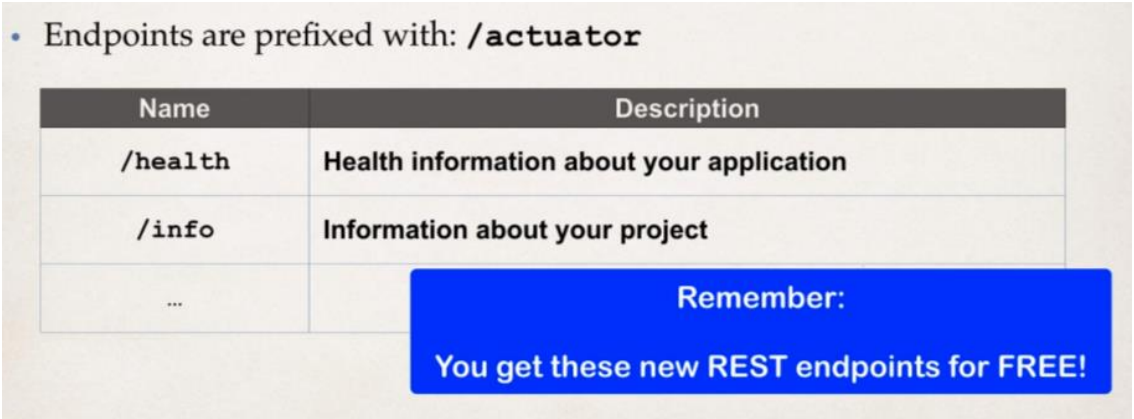
Problems:

- How can I monitor and Manage my application?
- How can I check the application health?
- How can I access application Metrics?

One Solution is: Spring boot Actuator

- It exposes endpoints to monitor and manage your application
- You easily get DevOps functionality
- Simply add the dependency to your POM file
- REST endpoints are automatically added to your application.





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

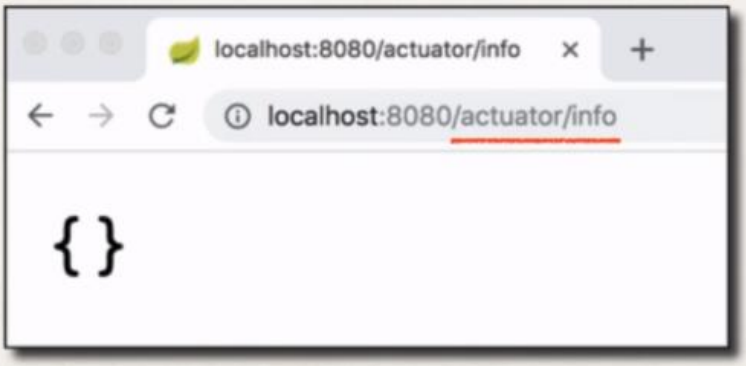
So, we can customize this “/info” endpoint.

- Update **application.properties** with your app info

File: src/main/resources/application.properties

```
info.app.name=My Super Cool App
info.app.description=A crazy and fun app, yoohoo!
info.app.version=1.0.0
```

- **/info** gives information about your application
- Default is empty



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
...	

Exposing Endpoints

- By default, only **/health** and **/info** are exposed
- To expose all actuator endpoints over HTTP

File: src/main/resources/application.properties

```
# Use wildcard "*" to expose all endpoints
# Can also expose individual endpoints with a comma-delimited list
#
management.endpoints.web.exposure.include=*
```

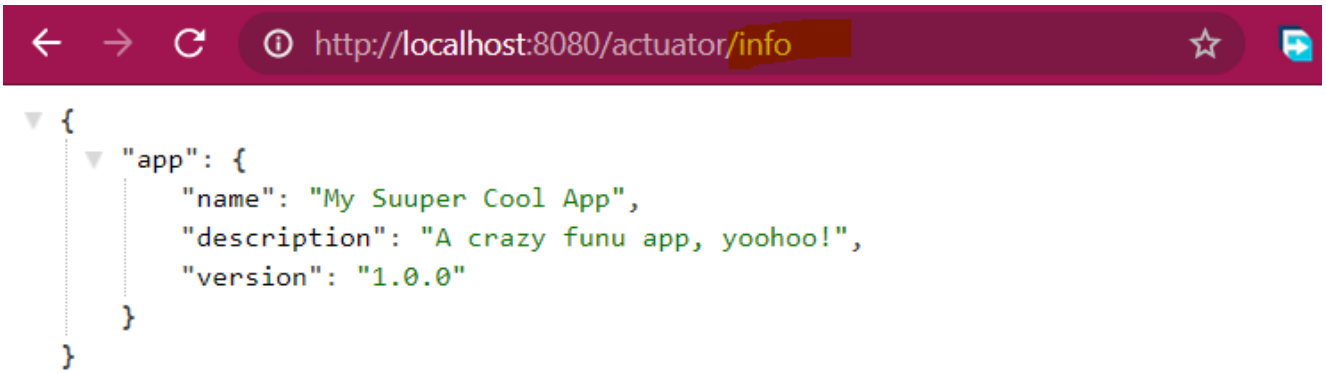
expose all endpoints

Development Process:

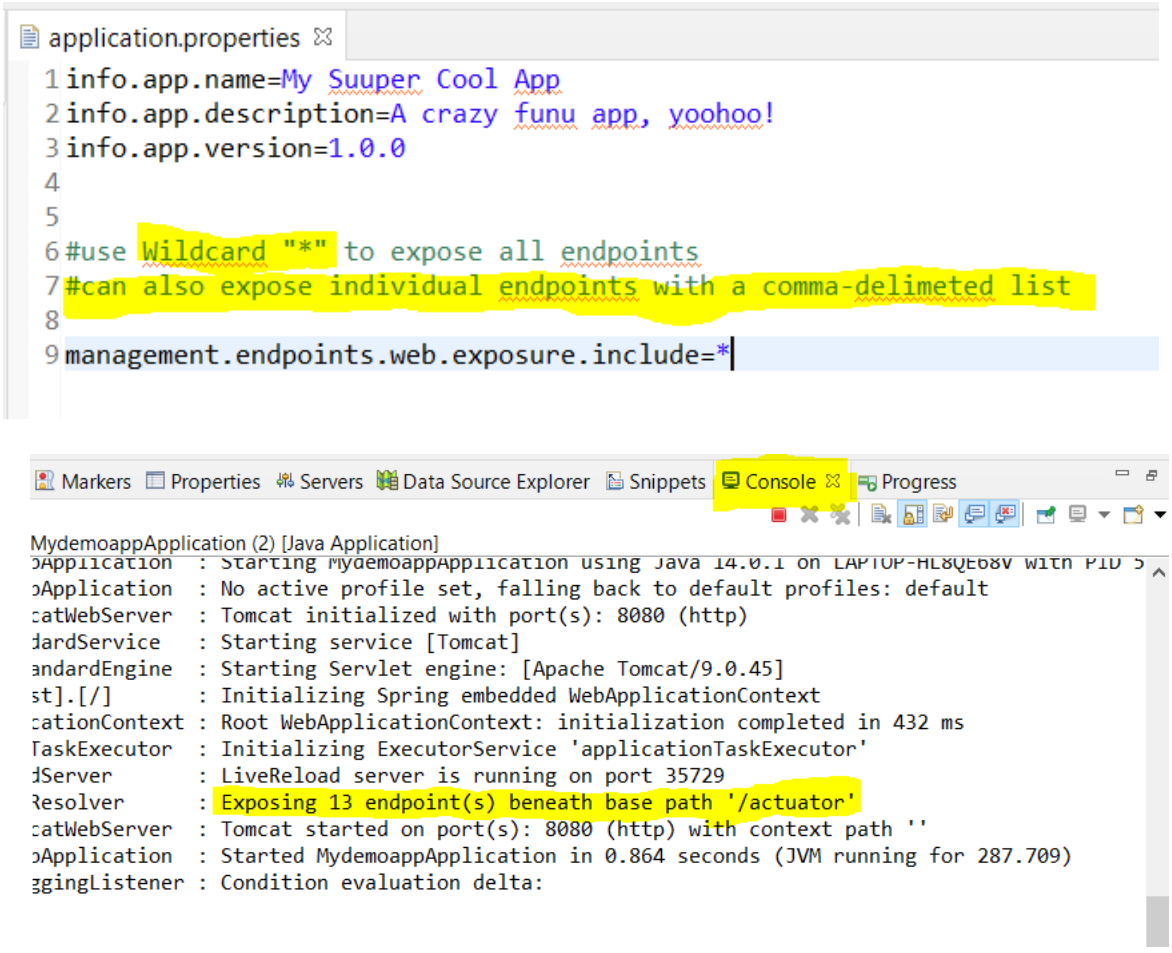
1. Edit pom.xml and add **spring-boot-starter-actuator**
2. View actuator endpoints for: **/health** and **/info**
3. Edit **application.properties** to customize **/info**.

application.properties

```
info.app.name=My Suuper Cool App
info.app.description=A crazy funu app, yoohoo!
info.app.version=1.0.0
```



Exposing all the other Endpoints of Spring Boot Actuator



We can test then using endpoints:

http://localhost:8080/actuator/mappings

http://localhost:8080/actuator/beans

http://localhost:8080/actuator/threaddump

Spring Boot Actuator Security:

Previously we learn how to expose all the spring boot actuator endpoints and we must be thinking about security,

We may not want to expose all of the endpoint information to anyone.

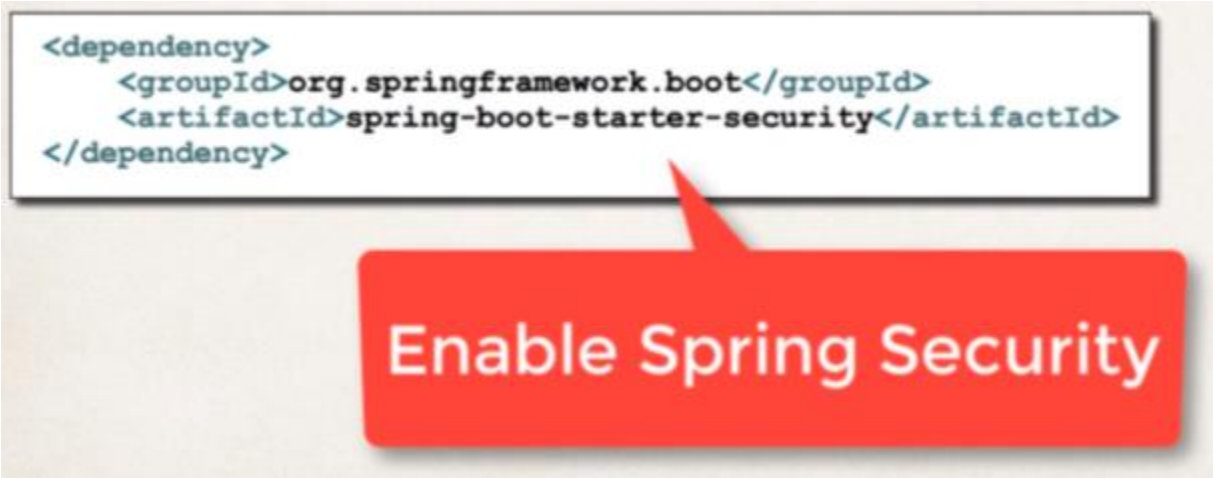
We will add Spring Security to project and endpoints are secured.

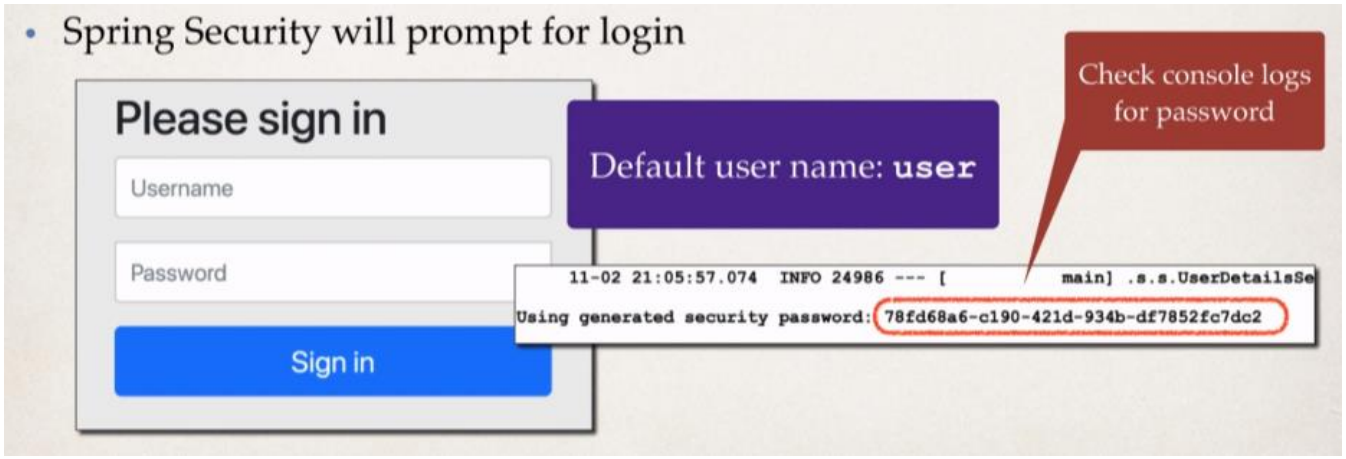
We need spring-boot-starter-security dependency.

Secured Endpoints:

- Now when we access: /actuator/beans
- Spring security will prompt for login

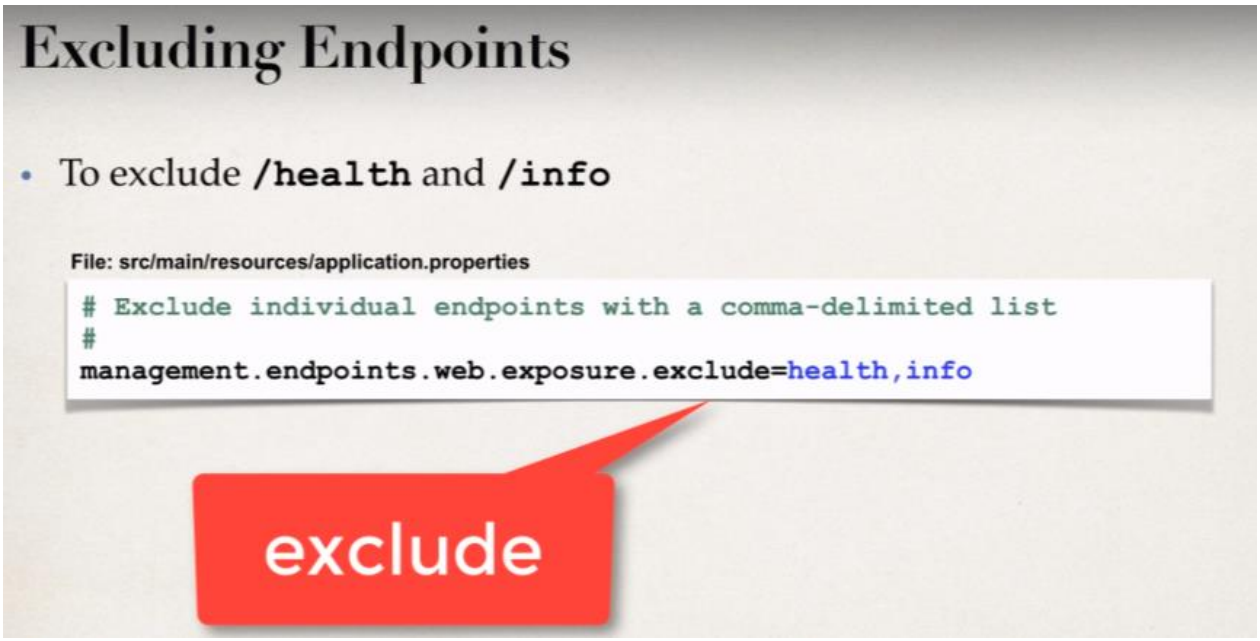
We can also override default username and generated password:





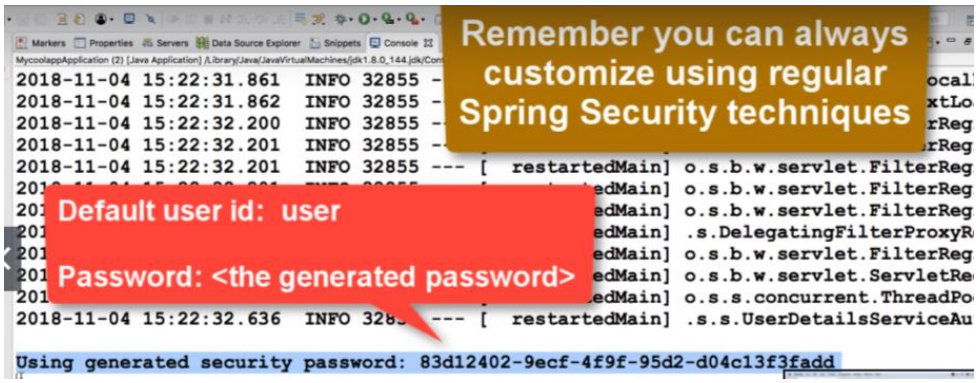
In the **`application.properties`** file:

```
spring.security.user.name=scott
spring.security.user.password=tiger
```

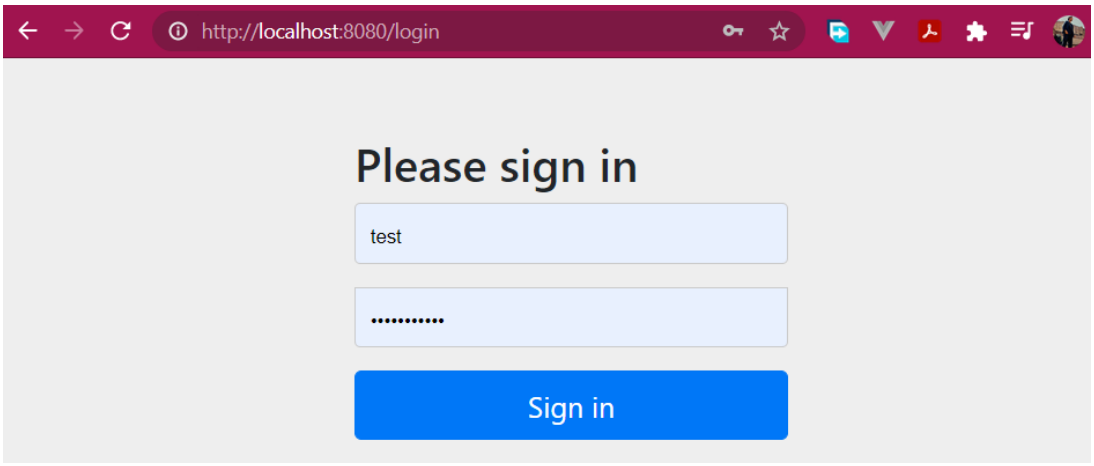


Development Process:

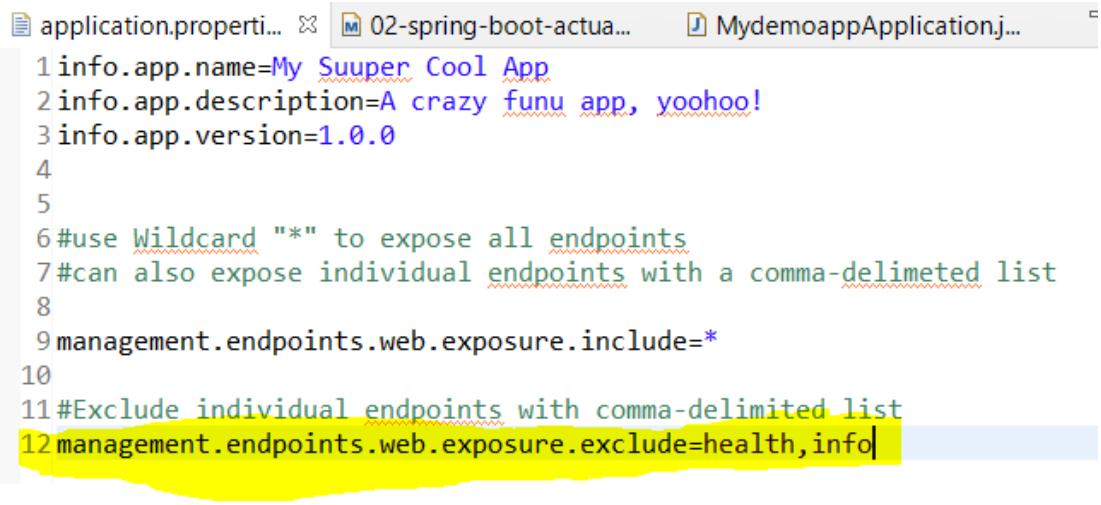
1. Edit pom.xml and add **`spring-boot-starter-security`**
2. Verify security on actuator endpoints for: **`/beans`** etc.
3. Disable endpoints for **`/health`** and **`/info`**.



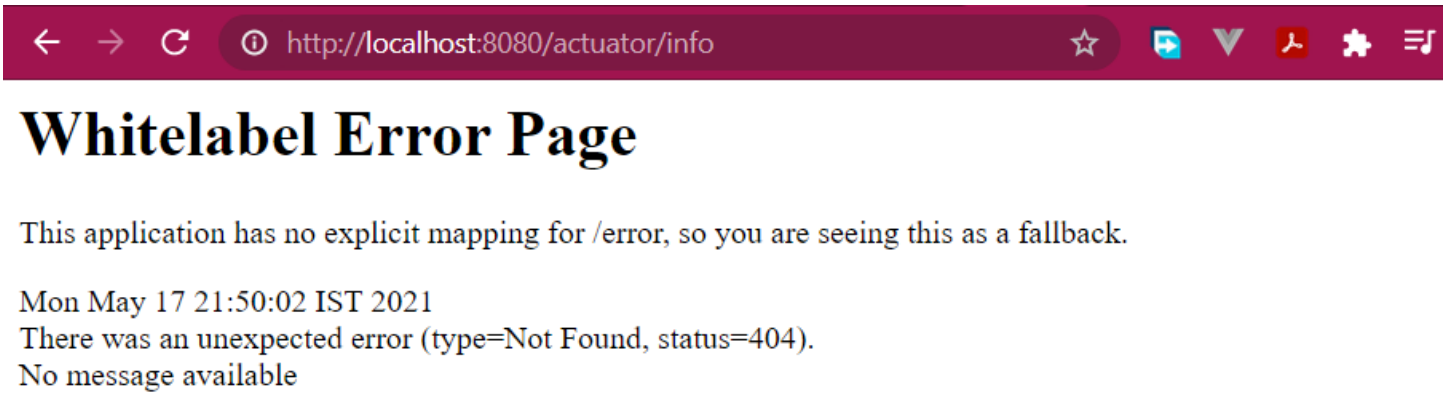
On hitting the endpoints: http://localhost:8080/actuator/mappings, we will get to see this login page.



But by default, /health and /info will not be secure so we will disable those endpoints.



Now /health and /info will not be available anywhere.



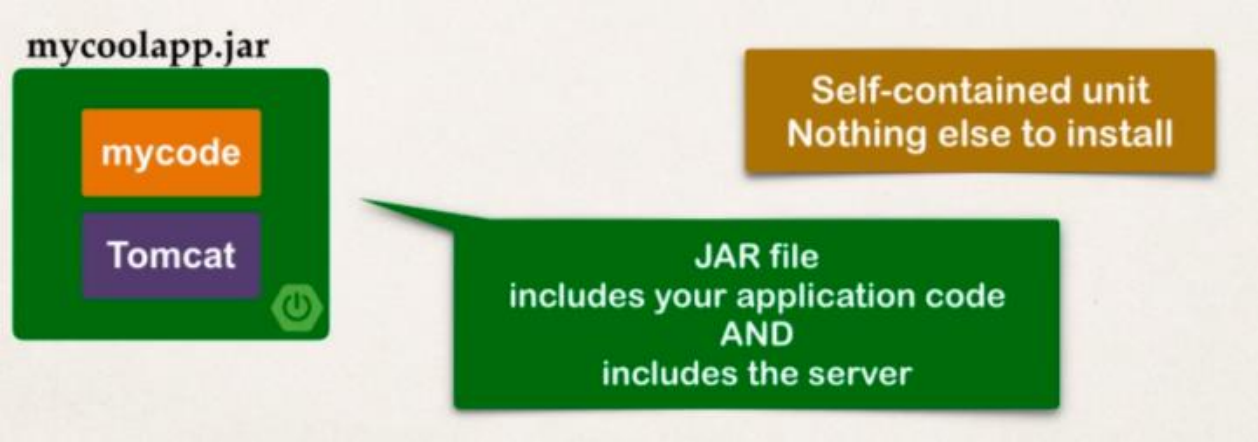
Section 72: Spring Boot - Running Spring Boot Apps from the Command Line

0 / 2 | 11min

Overview:

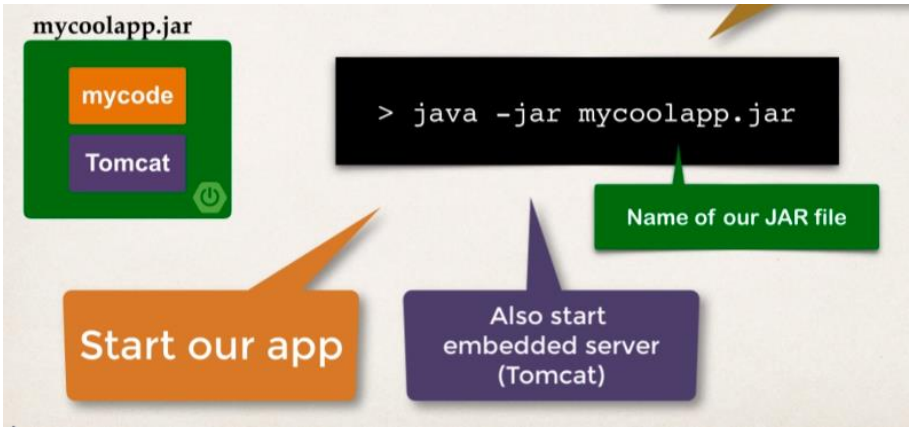
Running from the Command-Line:

- No need to have IDE open/running
- Since we are using Spring Boot, the server is embedded in our JAR File.
- So, there is no need to have separate server installed/running
- Since our spring boot apps are self-contained.



Two Options for running the app:

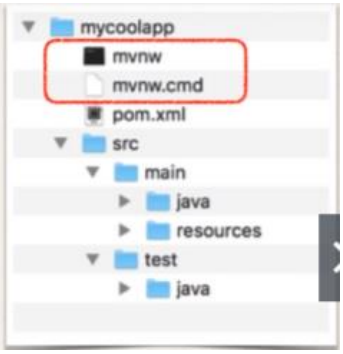
Option 1: User **`java -jar`**



Option 2: Use Spring boot maven Plugin

```
mvnw spring-boot:run
```

- **mvnw** allows you to run a Maven project
  - No need to have Maven installed or present on your path
  - If correct version of Maven is NOT found on your computer
    - **Automatically downloads** correct version and runs Maven



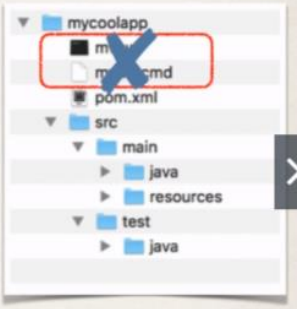
- **Automatically downloads** correct ve… and runs Maven
- Two files are provided
  - **mvnw.cmd** for MS Windows
  - **mvnw.sh** for Linux/Mac

```
> mvnw clean compile test
```

```
$ ./mvnw clean compile test
```

- If you already have Maven installed previously
  - Then you can ignore/delete the **mvnw** files
- Just use Maven as you normally would

```
$ mvn clean compile test
```



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

Development Process:

1. Exit the IDE
2. Package the app using **`mvnw package`**
3. Run app using **`java -jar`**
4. Run app using spring boot maven plugin, **`mvnw spring-boot:run.`**

`Go to the directory where the app is present then;`

```
D:\Java_Learning\spring_jar_sw\Spring-security-video-tutorial-files\CommandLine-Demo-Spring-boot>cd 03-spring-boot-commandline-demo

D:\Java_Learning\spring_jar_sw\Spring-security-video-tutorial-files\CommandLine-Demo-Spring-boot\03-spring-boot-commandline-demo>mvnw package
[INFO] Scanning for projects...
[INFO]
[INFO] ---------------------< com.example.demo:mydemoapp >---------------------
[INFO] Building mydemoapp 0.0.1-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- maven-resources-plugin:3.2.0:resources (default-resources) @ mydemoapp ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Using 'UTF-8' encoding to copy filtered properties files.
[INFO] Copying 1 resource
[INFO] Copying 0 resource
[INFO]
[INFO] --- maven-compiler-plugin:3.8.1:compile (default-compile) @ mydemoapp ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 2 source files to D:\Java_Learning\spring_jar_sw\Spring-security-video-tutorial-files\CommandLine-Demo-Spring-boot\03-spring-boot-commandlin
[INFO]
[INFO] --- maven-resources-plugin:3.2.0:testResources (default-testResources) @ mydemoapp ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Using 'UTF-8' encoding to copy filtered properties files.
[INFO] skip non existing resourceDirectory D:\Java_Learning\spring_jar_sw\Spring-security-video-tutorial-files\CommandLine-Demo-Spring-boot\03-spring-boot-co
[INFO]
[INFO] --- maven-compiler-plugin:3.8.1:testCompile (default-testCompile) @ mydemoapp ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 1 source file to D:\Java_Learning\spring_jar_sw\Spring-security-video-tutorial-files\CommandLine-Demo-Spring-boot\03-spring-boot-commandline-
```

Using **`mnvw package`** command to build the JAR File.

And the jar file will be created into the **`Target`** subdirectory:

```
2021-05-17 22:21:43.359  INFO 5884 --- [           main] c.e.d.m.MydemoappApplicationTests        : Starting MydemoappApplicationTests using Java 14.0.1 on LAPTOP-HL8QE68V with PID 5884
va_Learning\spring_jar_sw\Spring-security-video-tutorial-files\CommandLine-Demo-Spring-boot\03-spring-boot-commandline-demo)
2021-05-17 22:21:43.365  INFO 5884 --- [           main] c.e.d.m.MydemoappApplicationTests        : No active profile set, falling back to default profiles: default
2021-05-17 22:21:44.574  INFO 5884 --- [           main] o.s.s.concurrent.ThreadPoolTaskExecutor  : Initializing ExecutorService 'applicationTaskExecutor'
2021-05-17 22:21:44.938  INFO 5884 --- [           main] c.e.d.m.MydemoappApplicationTests        : Started MydemoappApplicationTests in 1.93 seconds (JVM running for 2.746)
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 2.721 s - in com.example.demo.mydemoapp.MydemoappApplicationTests
2021-05-17 22:21:45.400  INFO 5884 --- [extShutdownHook] o.s.s.concurrent.ThreadPoolTaskExecutor  : Shutting down ExecutorService 'applicationTaskExecutor'
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO]
[INFO] --- maven-jar-plugin:3.2.0:jar (default-jar) @ mydemoapp ---
[INFO] Building jar: D:\Java_Learning\spring_jar_sw\Spring-security-video-tutorial-files\CommandLine-Demo-Spring-boot\03-spring-boot-commandline-demo\target\mydemoapp-0.0.1-SNAPSHOT.jar
[INFO]
```

`And in the end, we will see BUILD SUCCESS log:`

```
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/shared/maven-dependency-tree/3.0
Downloaded from central: https://repo.maven.apache.org/maven2/org/ow2/asm/asm-util/8.0/asm-util-8.0.jar (85 kB
Downloaded from central: https://repo.maven.apache.org/maven2/org/vafer/jdependency/2.4.0/jdependency-2.4.0.jar
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/commons/commons-lang3/3.7/commons-lang
Downloaded from central: https://repo.maven.apache.org/maven2/org/jdom/jdom2/2.0.6/jdom2-2.0.6.jar (305 kB at 6
[INFO] Replacing main artifact with repackaged archive
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  24.179 s
[INFO] Finished at: 2021-05-17T22:22:02+05:30
[INFO] ------------------------------------------------------------------------
```

Now move into the **`target`** subdirectory:

```
D:\Java_Learning\spring_jar_sw\Spring-security-video-tutorial-files\CommandLine-Demo-Spring-boot\03-spring-boot-commandline-demo>cd target
```

Now we need to run the created jar file:

```
D:\Java_Learning\spring_jar_sw\Spring-security-video-tutorial-files\CommandLine-Demo-Spring-boot\03-spring-boot-commandline-demo\target>dir
 Volume in drive D is DATA
 Volume Serial Number is 2CA5-F7F9

 Directory of D:\Java_Learning\spring_jar_sw\Spring-security-video-tutorial-files\CommandLine-Demo-Spring-boot\03-spring-boot-commandline-demo\target

05/17/2021  10:22 PM    <DIR>          .
05/17/2021  10:22 PM    <DIR>          ..
05/17/2021  10:19 PM    <DIR>          classes
05/17/2021  10:21 PM    <DIR>          generated-sources
05/17/2021  10:21 PM    <DIR>          generated-test-sources
05/17/2021  10:21 PM    <DIR>          maven-archiver
05/17/2021  10:21 PM    <DIR>          maven-status
05/17/2021  10:22 PM        17,058,909 mydemoapp-0.0.1-SNAPSHOT.jar
05/17/2021  10:21 PM             4,285 mydemoapp-0.0.1-SNAPSHOT.jar.original
05/17/2021  10:21 PM    <DIR>          surefire-reports
05/17/2021  10:19 PM    <DIR>          test-classes
               2 File(s)     17,063,194 bytes
               9 Dir(s)  184,647,352,320 bytes free
```

Using the command **`java -jar mydemoapp-0.0.1-SNAPSHOT.jar`**

```
D:\Java_Learning\spring_jar_sw\Spring-security-video-tutorial-files\CommandLine-Demo-Spring-boot\03-spring-boot-commandline-demo\target>java -jar mydemoapp-0.0.1-SNAPSHOT.jar

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v2.4.5)

2021-05-17 22:28:22.950  INFO 13100 --- [           main] c.e.demo.mydemoapp.MydemoappApplication  : Starting MydemoappApplication v0.0.1-SNAPSHOT using Java 14.0.1 on LAPTOP-HL
rning\spring_jar_sw\Spring-security-video-tutorial-files\CommandLine-Demo-Spring-boot\03-spring-boot-commandline-demo\target\mydemoapp-0.0.1-SNAPSHOT.jar started by Anuj in D:\J
security-video-tutorial-files\CommandLine-Demo-Spring-boot\03-spring-boot-commandline-demo\target)
2021-05-17 22:28:22.954  INFO 13100 --- [           main] c.e.demo.mydemoapp.MydemoappApplication  : No active profile set, falling back to default profiles: default
2021-05-17 22:28:24.368  INFO 13100 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 8080 (http)
2021-05-17 22:28:24.380  INFO 13100 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2021-05-17 22:28:24.381  INFO 13100 --- [           main] org.apache.catalina.core.StandardEngine  : Starting Servlet engine: [Apache Tomcat/9.0.45]
2021-05-17 22:28:24.452  INFO 13100 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2021-05-17 22:28:24.453  INFO 13100 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 1338 ms
2021-05-17 22:28:24.706  INFO 13100 --- [           main] o.s.s.concurrent.ThreadPoolTaskExecutor  : Initializing ExecutorService 'applicationTaskExecutor'
2021-05-17 22:28:24.935  INFO 13100 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8080 (http) with context path ''
2021-05-17 22:28:24.949  INFO 13100 --- [           main] c.e.demo.mydemoapp.MydemoappApplication  : Started MydemoappApplication in 2.588 seconds (JVM running for 2.967)
```

And it runs the application on cmd.

Now to run via spring boot maven plugin:

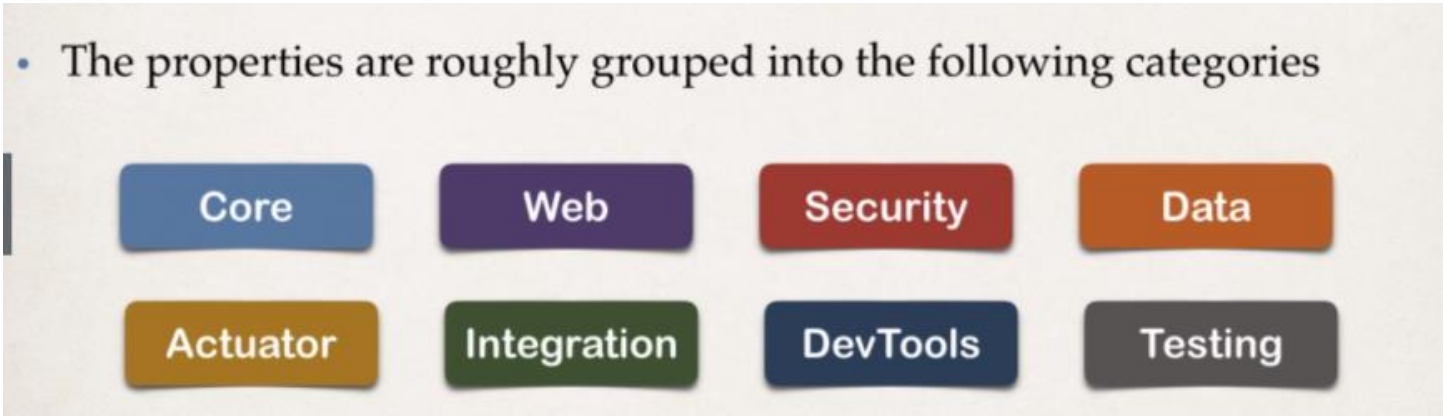
- **`cd..`**
- go to application folder; use command: **`mvnw spring-boot:run`**

And this will run the application.

- **Spring Boot Properties:**

1. Spring Boot can be configured in the application.properties file.
2. Server port, context path, actuator, security etc

- The properties are roughly grouped into the following categories



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
```

Set logging levels based on package names

Web Properties

http://localhost:7070/my-silly-app/fortune

File: src/main/resources/application.properties

```
# HTTP server port
server.port=7070

# Context path of the application
server.servlet.context-path=/my-silly-app

# Default HTTP session time out
server.servlet.session.timeout=15m
...
```

15 minutes

Default timeout 30 minutes

Actuator Properties

Actuator

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

Security Properties

File: src/main/resources/application.properties

```
# Default user name
spring.security.user.name=admin

# Password for default user
spring.security.user.password=topsecret
...
```

Data Properties

Data

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=scott

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

More on this in later videos

➢ Spring Employee CRUD Operations:

For setting up DAO service:

In traditional Spring

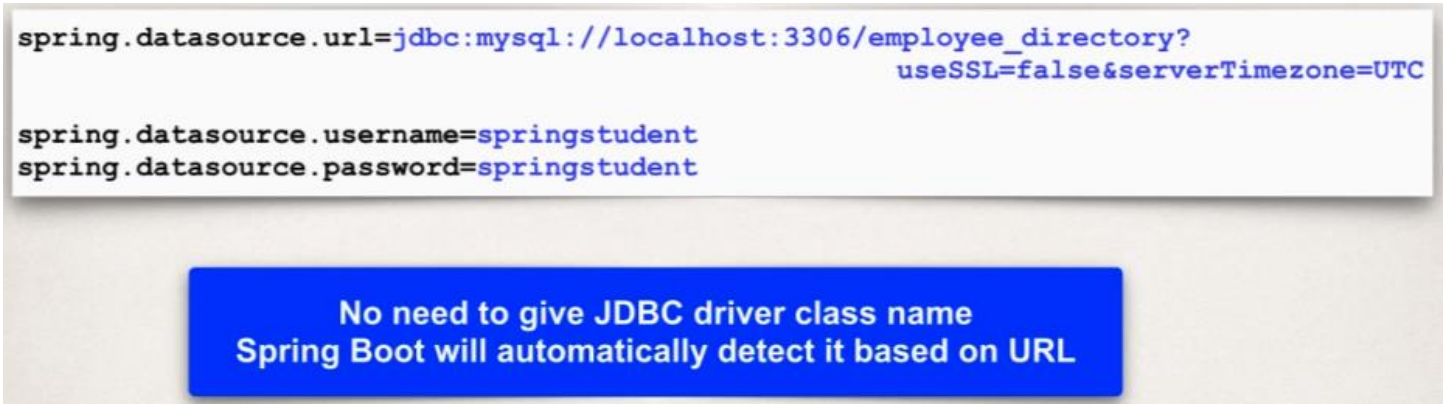
We normally had to do this configuration manually;

So, there should be easier solution to do all this as this is very error prone.

Spring boot comes to the rescue:

- Spring boot will automatically configure your data source for you.
- Based on entries from Maven pom file:
  - JDBC Driver: **`myql-connector-java`**
  - Spring Data (ORM): **`spring-boot-starter-data-jpa`**
- DB connection info from **`application.properties`**

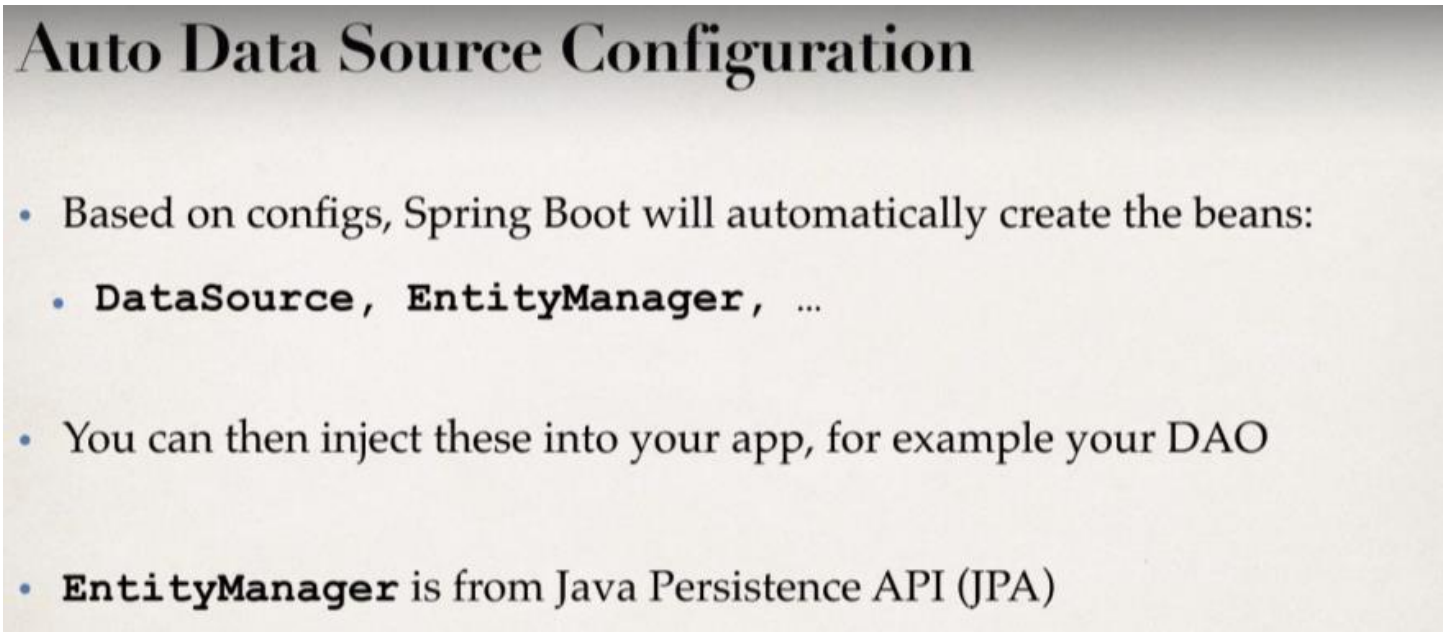
So, spring boot will actually create all these info to create data source for us.



There is additional data source properties that we can set:

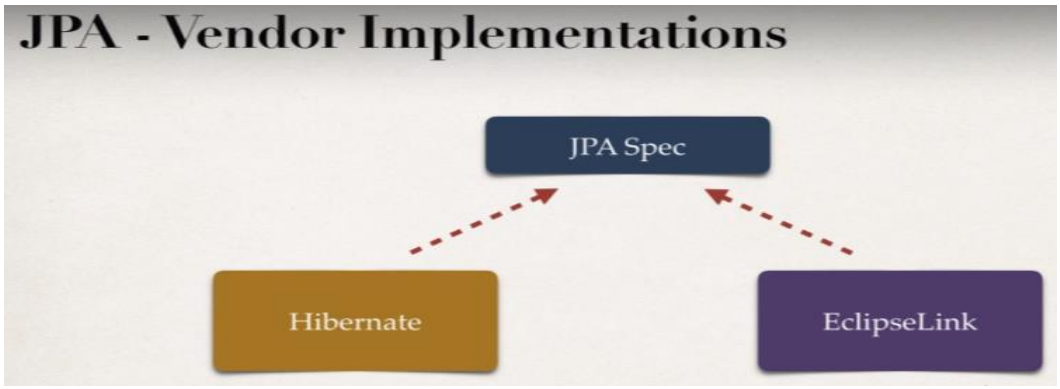
Properties are available to configure connection pool etc.

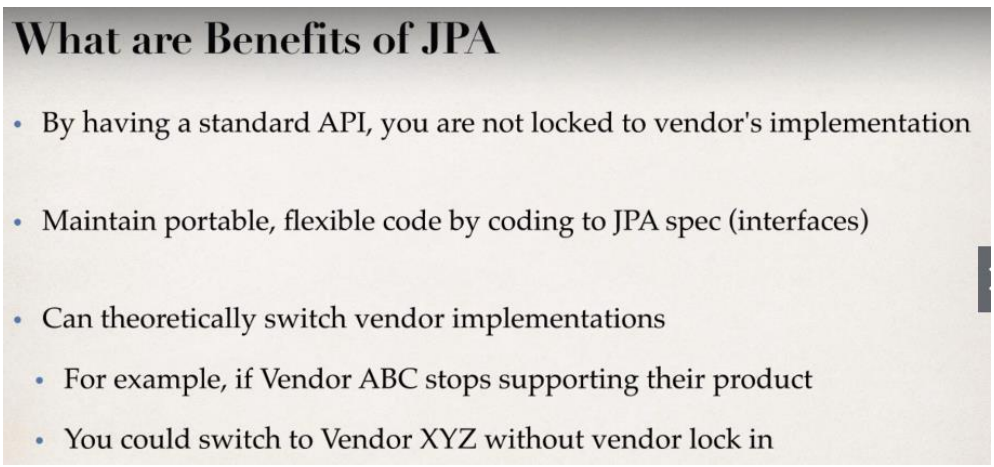
Link to follow more about spring boot properties. And go to **`spring.datasource.*`**



What is **JPA**?

- **Java Pesistence API (JPA)**
  - Standard API for Object-to-Relatioinal-Mapping (ORM)
- It's only a Specification
  - Defines a set of interfaces
  - Requires an implementation to be usable



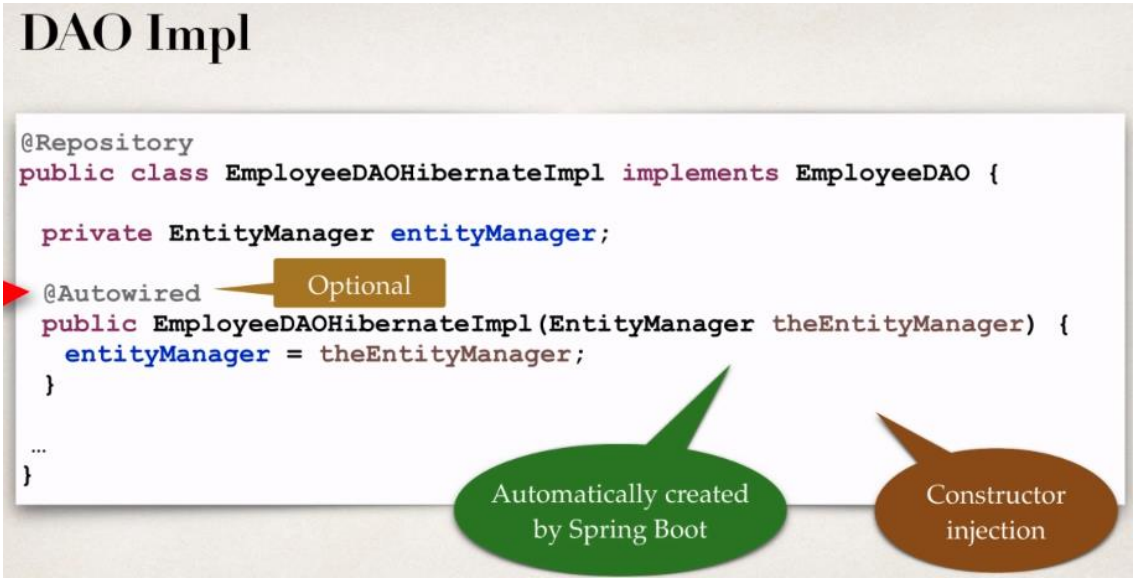


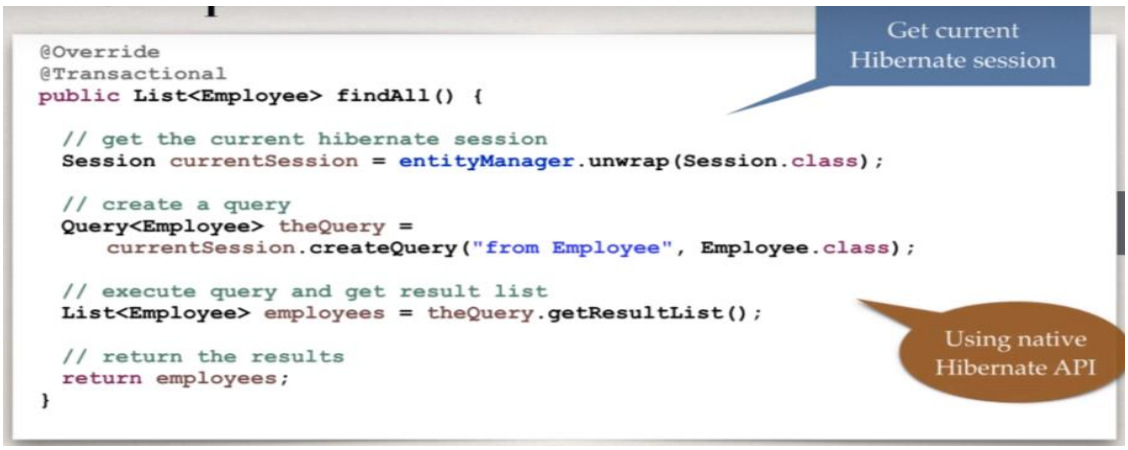
- In spring Boot, **`hibernate`** is default implementation of JPA.
- **`EntityManager`** is similar to Hibernate **`SessionFactory`**.
- **`EntityManager`** can serve as a wrapper for a hibernate **`Session`** Object.
- So, in our code we can inject **`EntityManager`** into our **`DAO`**

Various DAO techniques:

- Version 1: Use EntityManager but leverage native Hibernate API
- Version 2: Use EntityManager and standard JPA API
- Version 3: Spring Data JPA

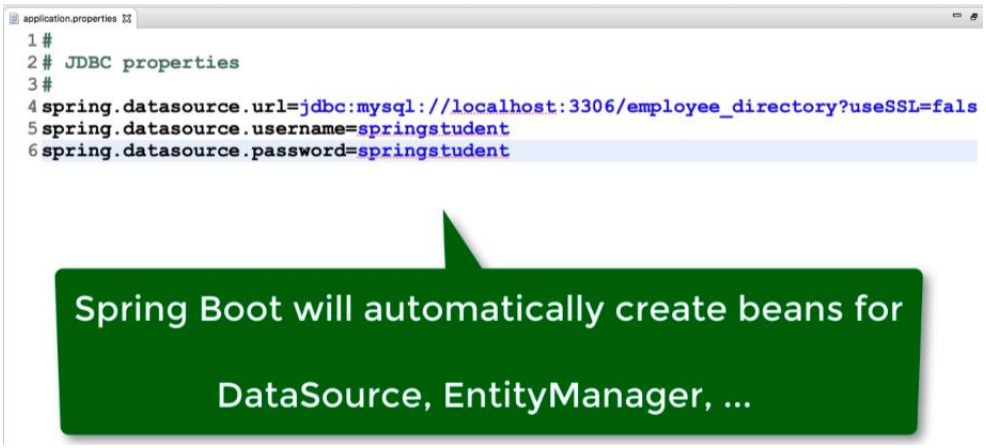
Version 1: Use EntityManager but leverage native Hibernate API Java Project link





Development Process:

- Update DB configs in **`application.properties`**

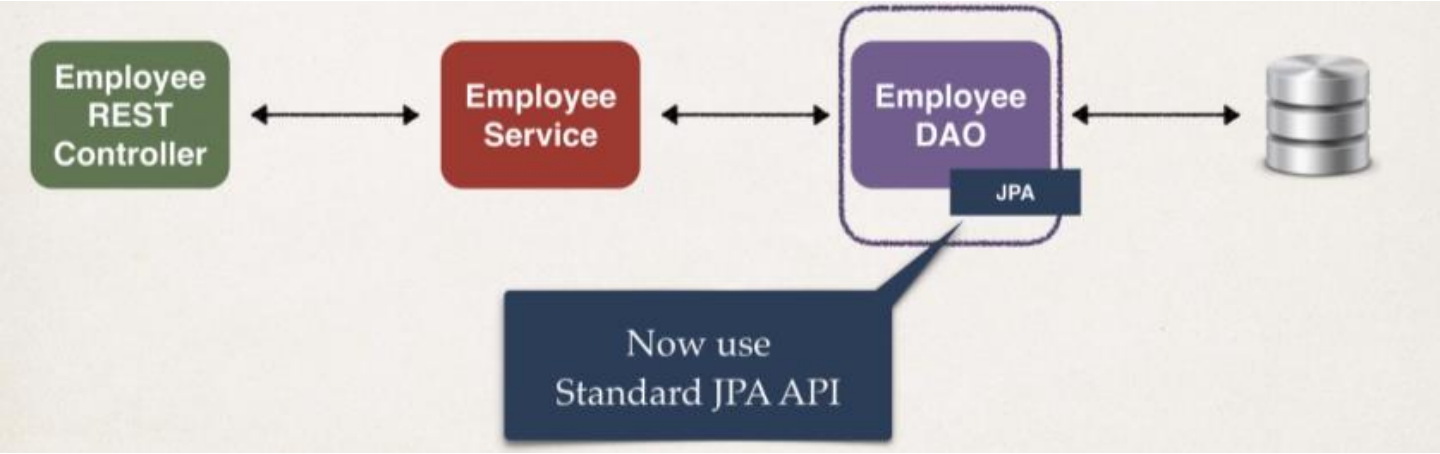


- Create **`Employee`** entity
- Create DAO interface.
- Create DAO implementation
- Create REST Controller to use DAO.

➢ **Create a JPA DAO in Spring boot:** Java Project Link

We have already covered version 1: Use EntityManager but leverage native Hibernate API

Here we will focus on the **usage of EntityManager and standard Data JPA API**



Benefits of JPA:

- By having a standard API, you are not locked to vendor's implementation.
- Maintain portable, flexible code
- Can theoretically switch vendor implementation
- If vendor ABC stops supporting their product, then we can easily switch to vendor XYZ without vendor lock in.

Standard JPA API

- The JPA API methods are similar to Native Hibernate API
- JPA also supports a query language: JPQL (JPA Query Language)
  - For more details on JPQL, see this link

www.luv2code.com/jpql

Comparing JPA to Native Hibernate Methods

Action	Native Hibernate method	JPA method
Create/save new entity	`session.save(...)`	`entityManager.persist(...)`
Retrieve entity by id	`session.get(...) / load(...)`	`entityManager.find(...)`
Retrieve list of entities	`session.createQuery(...)`	`entityManager.createQuery(...)`
Save or update entity	`session.saveOrUpdate(...)`	`entityManager.merge(...)`
Delete entity	`session.delete(...)`	`entityManager.remove(...)`

Development Process

1. Set up Database Dev Environment
2. Create Spring Boot project using Spring Initializr
3. Get list of employees
4. Get single employee by ID
5. Add a new employee
6. Update an existing employee
7. Delete an existing employee

Step-By-Step

Let's build a DAO layer for this

Using Standard JPA API

DAO Impl

Same interface for consistent API

```
@Repository
public class EmployeeDAOJpaImpl implements EmployeeDAO {

  private EntityManager entityManager;

  @Autowired
  public EmployeeDAOJpaImpl(EntityManager theEntityManager) {
    entityManager = theEntityManager;
  }

 ...
}
```

Automatically created by Spring Boot

Constructor injection

Get a list of employees

JPA API

Remember: No need to manage transactions ...

Handled at Service layer with @Transactional

```
@Override
public List<Employee> findAll() {

  // create a query
  TypedQuery<Employee> theQuery =
     entityManager.createQuery("from Employee", Employee.class);

  // execute query and get result list
  List<Employee> employees = theQuery.getResultList();

  // return the results
  return employees;
}
```

Using Standard JPA API

Delete an existing employee

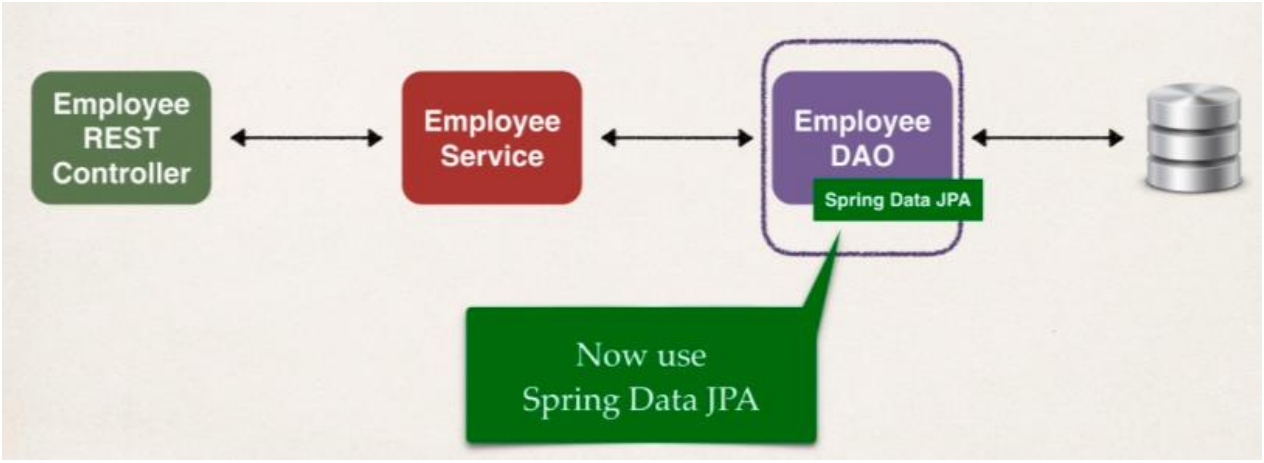
JPA API

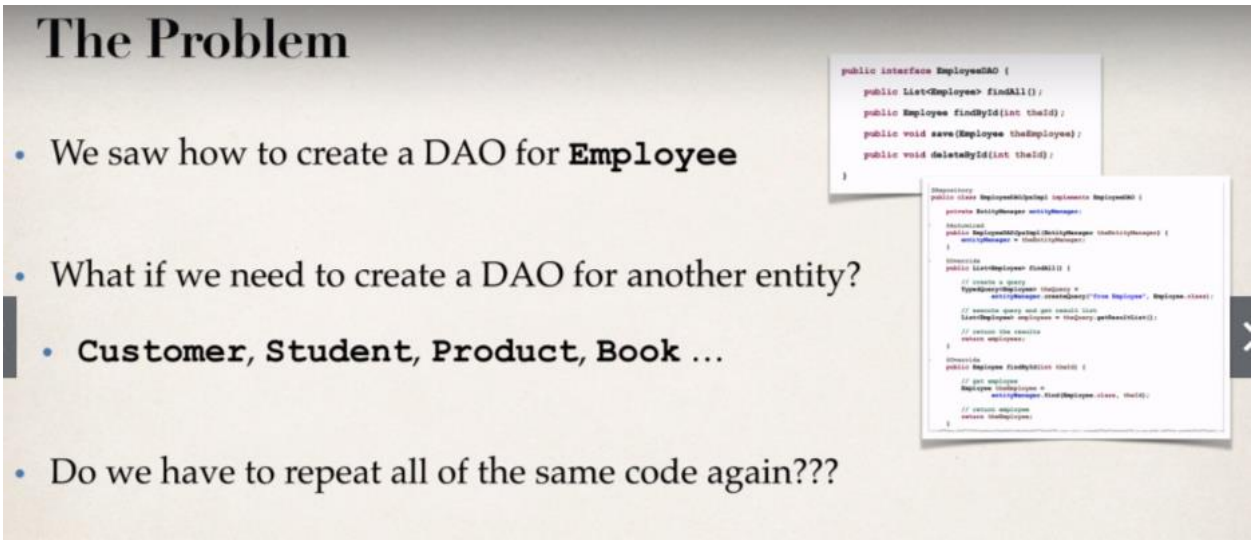
```
@Override
public void deleteById(int theId) {

  // delete object with primary key
  Query theQuery = entityManager.createQuery(
                                "delete from Employee where id=:employeeId");

  theQuery.setParameter("employeeId", theId);

  theQuery.executeUpdate();
}
```

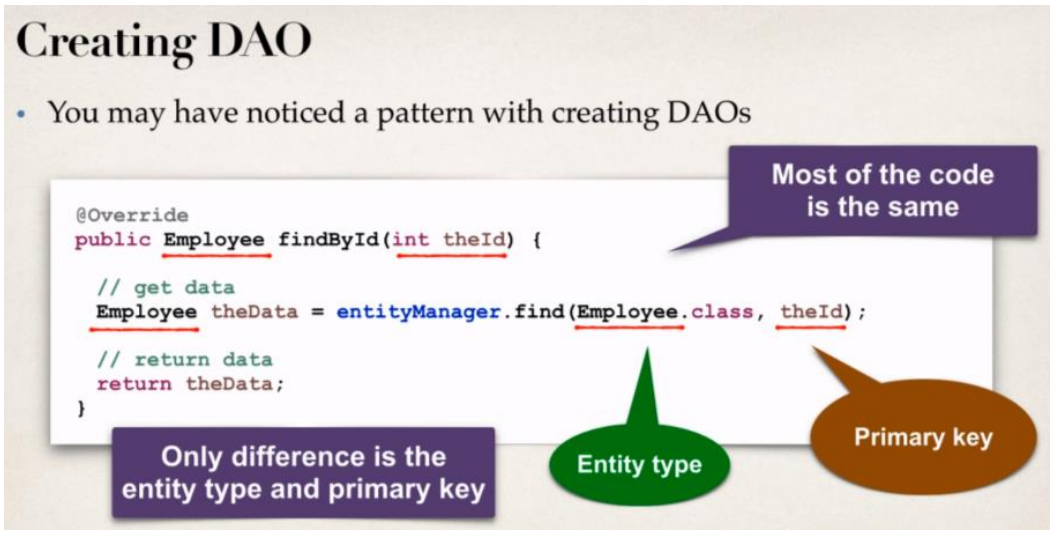
Section 76: Spring Boot - Spring Data JPA - Real-Time Project

0 / 3 | 24min

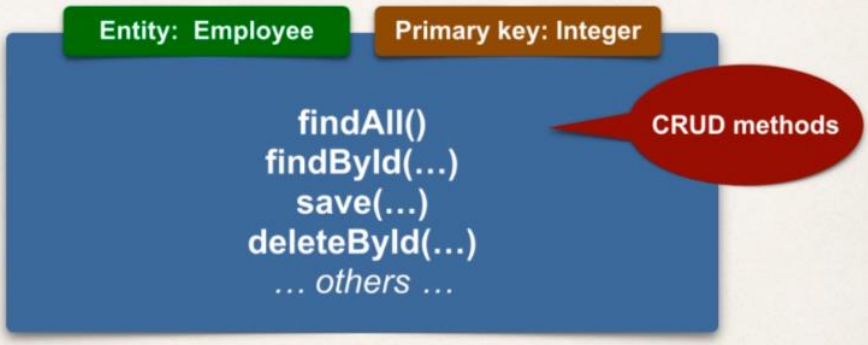
We will **Spring Data JPA:** Java Project Link







My Wish Diagram



Spring Data JPA - Solution

- **Spring Data JPA** is the solution!!!!

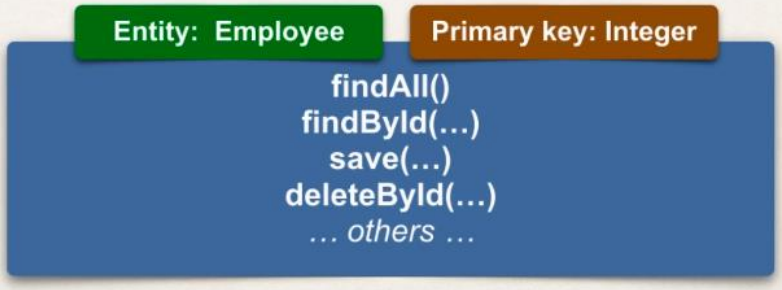
https://spring.io/projects/spring-data-jpa

- Create a DAO and just plug in your entity type and primary key
- Spring will give you a CRUD implementation for FREE …. like MAGIC!!
  - Helps to minimize boiler-plate DAO code … yaaay!!!

More than 70% reduction in code … depending on use case

JpaRepository

- Spring Data JPA provides the interface: **`JpaRepository`**
- Exposes methods (some by inheritance from parents)



Development Process

Step-By-Step

1. Extend **`JpaRepository`** interface

2. Use your Repository in your app

No need for implementation class

Step 1: Extend JpaRepository interface

Entity type

Primary key

```
public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

  // that's it ... no need to write any code LOL!

}
```

No need for implementation class

Get these methods for free

Entity: Employee

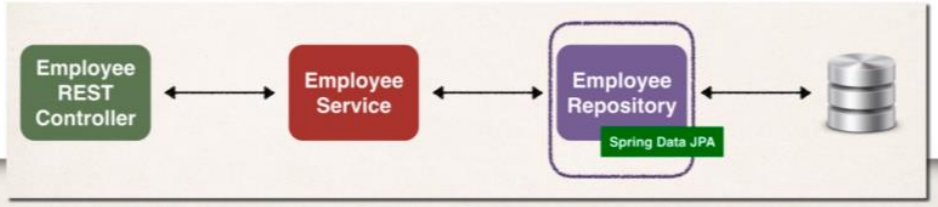
Primary key: Integer

findAll()
findById(...)
save(...)
deleteById(...)
... others ...

Step 2: Use Repository in your app

```
@Service
public class EmployeeServiceImpl implements EmployeeService {

  private EmployeeRepository employeeRepository;

  @Autowired
  public EmployeeServiceImpl(EmployeeRepository theEmployeeRepository) {
    employeeRepository = theEmployeeRepository;
  }
```

Our repository



```java
@Service
public class EmployeeServiceImpl implements EmployeeService {

  private EmployeeRepository employeeRepository;

  @Autowired
  public EmployeeServiceImpl(EmployeeRepository theEmployeeRepository) {
    employeeRepository = theEmployeeRepository;
  }

  @Override
  public List<Employee> findAll() {
    return employeeRepository.findAll();
  }
  ...
}
```

Our repository

Magic method that is available via repository

Minimized Boilerplate Code

Before Spring Data JPA

```java
public interface EmployeeDAO {

    public List<Employee> findAll();

    public Employee findById(int theId);

    public void save(Employee theEmployee);

    public void deleteById(int theId);

}
```

```java
@Repository
public class EmployeeDAOJpaImpl implements EmployeeDAO {

    private EntityManager entityManager;

    @Autowired
    public EmployeeDAOJpaImpl(EntityManager theEntityManager) {
        entityManager = theEntityManager;
    }

    @Override
    public List<Employee> findAll() {

        // create a query
        TypedQuery<Employee> theQuery =
                entityManager.createQuery("from Employee", Employee.class);

        // execute query and get result list
        List<Employee> employees = theQuery.getResultList();

        // return the results
        return employees;
    }

    @Override
    public Employee findById(int theId) {

        // get employee
        Employee theEmployee =
                entityManager.find(Employee.class, theId);

        // return employee
        return theEmployee;
    }
```

2 Files
30+ lines of code

After Spring Data JPA

```java
public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

    // that's it ... no need to write any code LOL!

}
```

1 File
3 lines of code!

No need for implementation class

Advanced Features

- Advanced features available for
  - Extending and adding custom queries with JPQL
  - Query Domain Specific Language (Query DSL)
  - Defining custom methods (low-level coding)

Visit for more info: Link

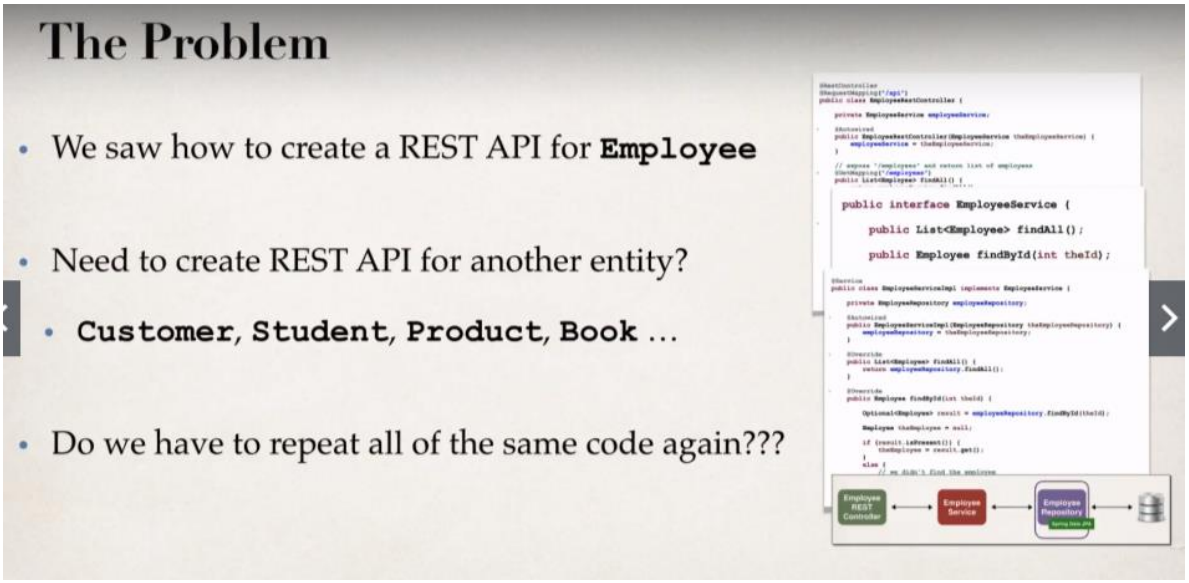
JPARepository make use of **Optional**; so instead of checking for nulls we can make use of **Optional** to see if value is present or not.

```
@Override
public Employee findById(int theId) {
	Optional<Employee> result = employeeRepository.findById(theId);

	Employee theEmployee = null;
	if(result.isPresent()) {
		theEmployee= result.get();
	}
	else {
		throw new RuntimeException("Did not find Employee id - "+theId);
	}
	return theEmployee;
}
```

Section 77: Spring Boot - Spring Data REST - Real-Time Project

0 / 3 | 30min

- Earlier we saw the magic of **Spring-data-JPA**
- This helped to eliminate Boilerplate code.



My Wish

- I wish we could tell Spring:

Create a REST API for me

Use my existing JpaRepository (entity, primary key)

Give me all of the basic REST API CRUD features for free

Spring Data REST - Solution

- **Spring Data REST** is the solution!!!! https://spring.io/projects/spring-data-rest
- Leverages your existing **`JpaRepository`**
- Spring will give you a REST CRUD implementation for FREE …. like MAGIC!!
  - Helps to minimize boiler-plate REST code!!!
  - No new coding required!!!

REST API

- Spring Data REST will expose these endpoints for free!

HTTP Method		CRUD Action
POST	/employees	Create a new employee
GET	/employees	Read a list of employees
GET	/employees/{employeeId}	Read a single employee
PUT	/employees/{employeeId}	Update an existing employee
DELETE	/employees/{employeeId}	Delete an existing employee

Get these REST endpoints for free

Spring Data REST – How Does it work?

- Spring Data REST will scan your project for JpaRepository.
- Expose REST APIs for each entity for your JpaRepository.

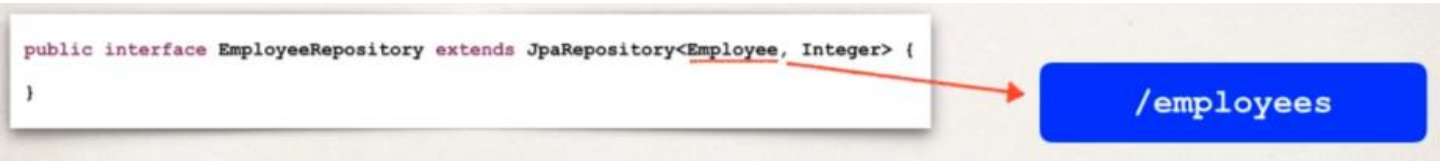
```
public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

}
```

REST Endpoints:

By Default, Spring Data REST will create the endpoints based on entity type.

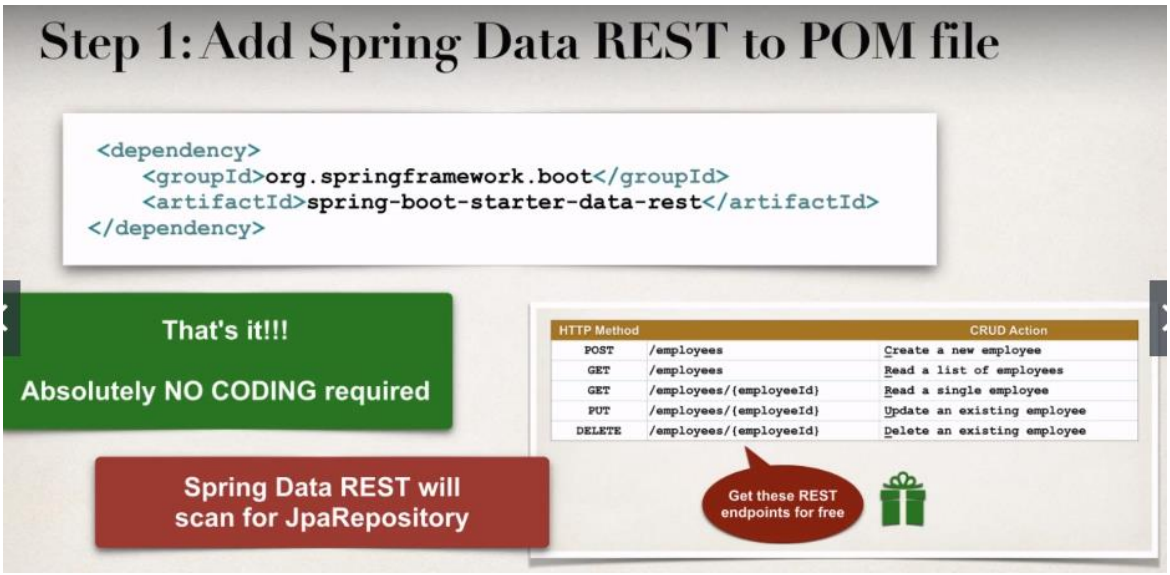
Simple plurized form

- First Character of Entity type is lowercase
- Then just add an “s” to the entity.



Development Process:

1. Add Spring Data REST to your Maven POM file.



In A Nutshell

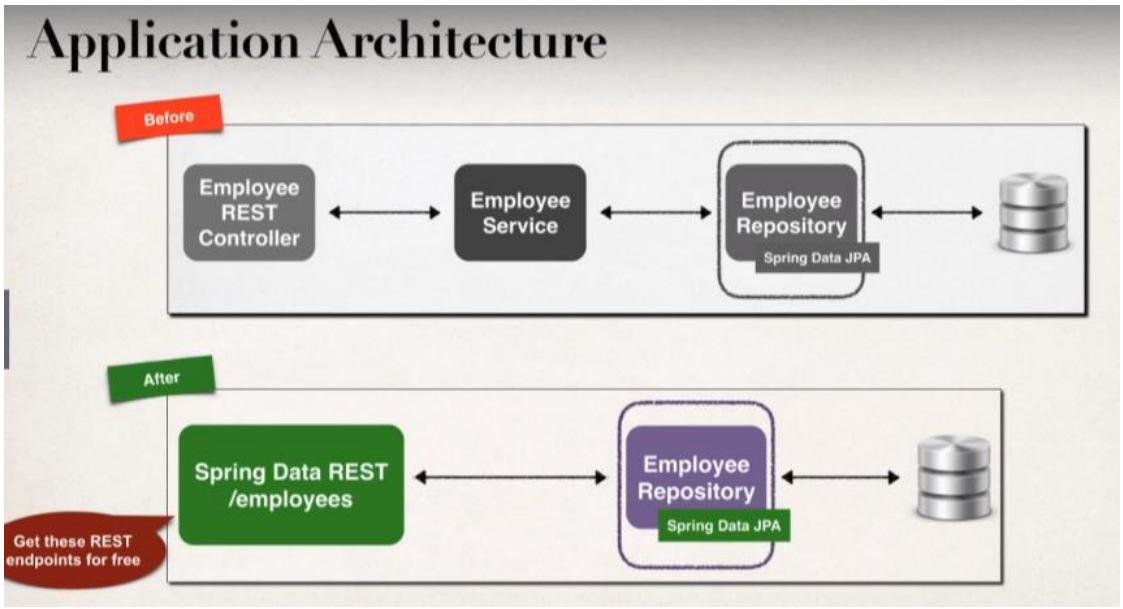
For Spring Data REST, you only need 3 items

We already have these two

1. Your entity: **`Employee`**
2. JpaRepository: **`EmployeeRepository extends JpaRepository`**
3. Maven POM dependency for: **`spring-boot-starter-data-rest`**

Only item that is new

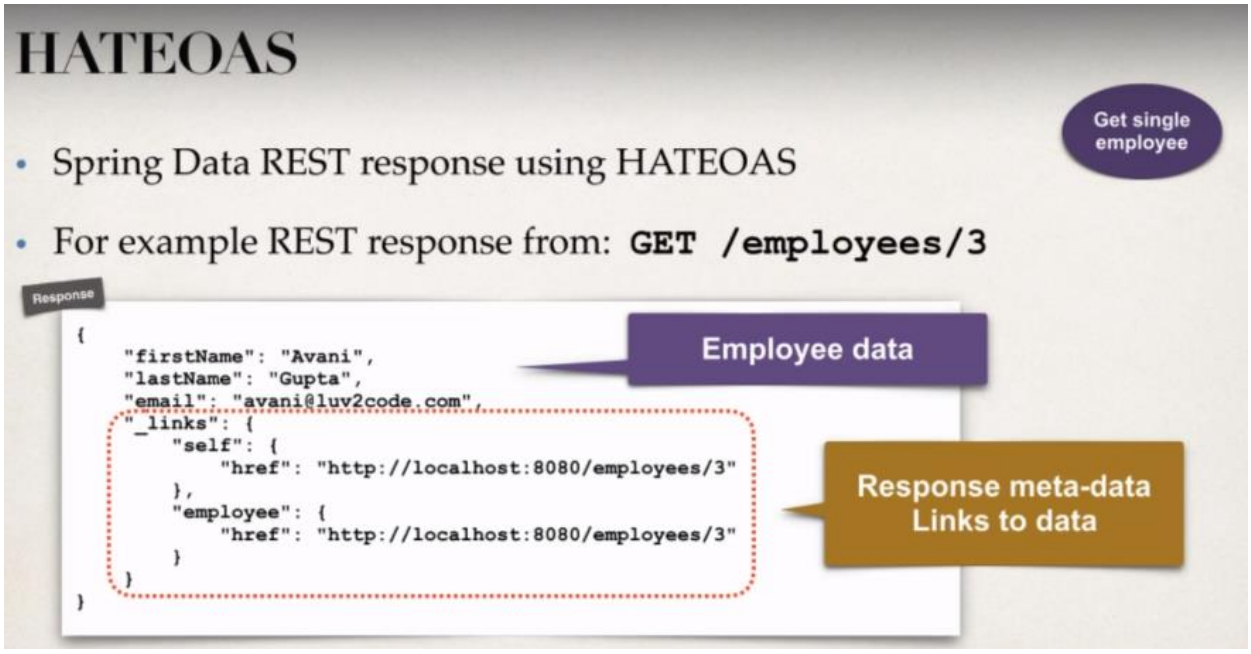
Application Architecture

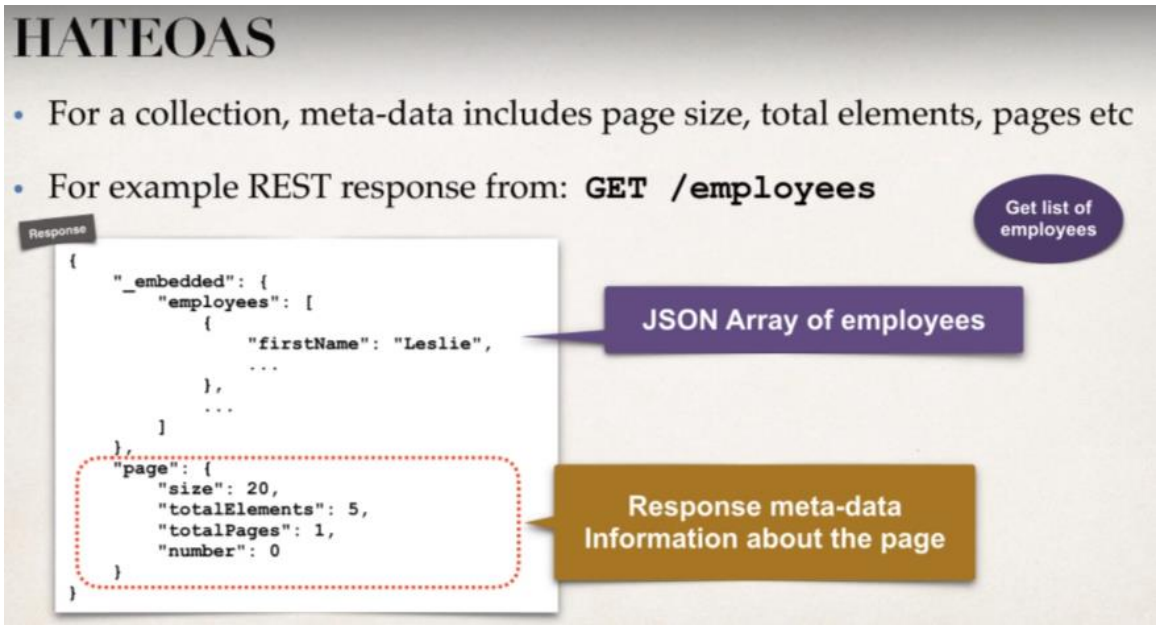


HATEOAS

- Spring Data REST endpoints are HATEOAS compliant
  - **HATEOAS**: Hypermedia as the Engine of Application State
- Hypermedia-driven sites provide information to access REST interfaces
  - Think of it as meta-data for REST data

https://spring.io/understanding/HATEOAS



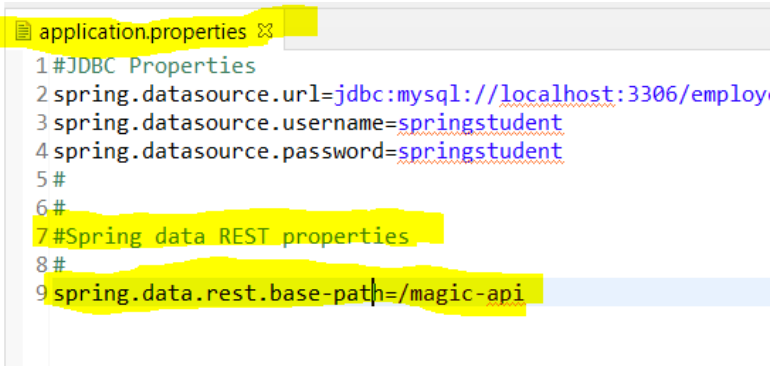


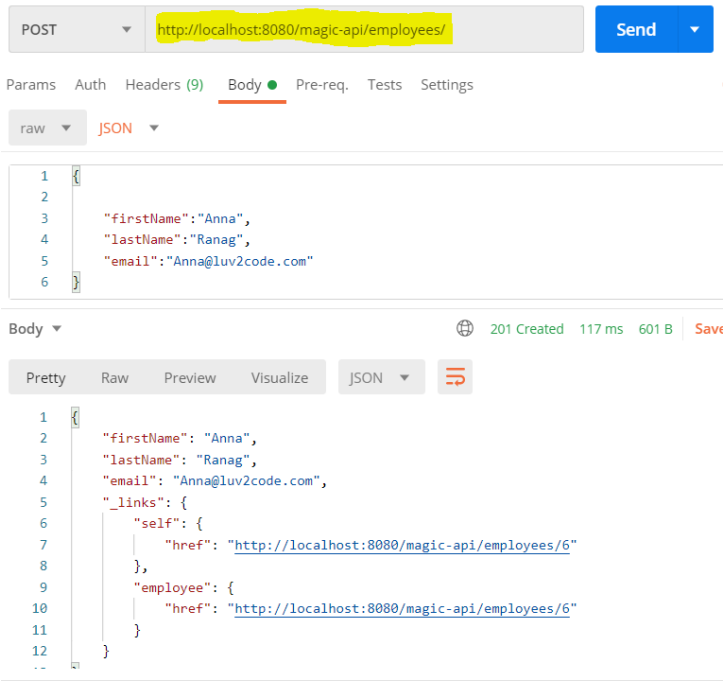
Advanced Features:

Spring Data REST advanced features

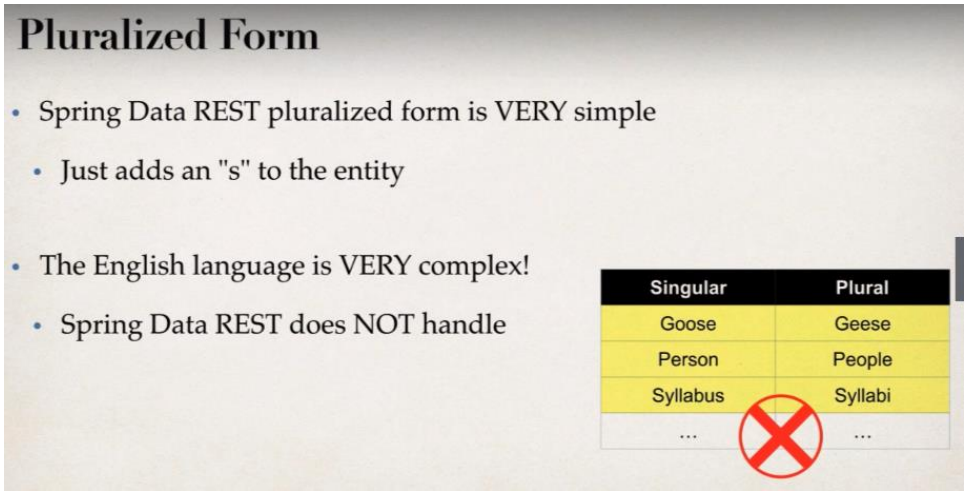
- Pagination, sorting and searching.
- Extending and adding custom queries with JPQL.
- We can also customize the REST APIs by making use of Query Domain Specific Language (Query DSL).

Customize the endpoint base path:



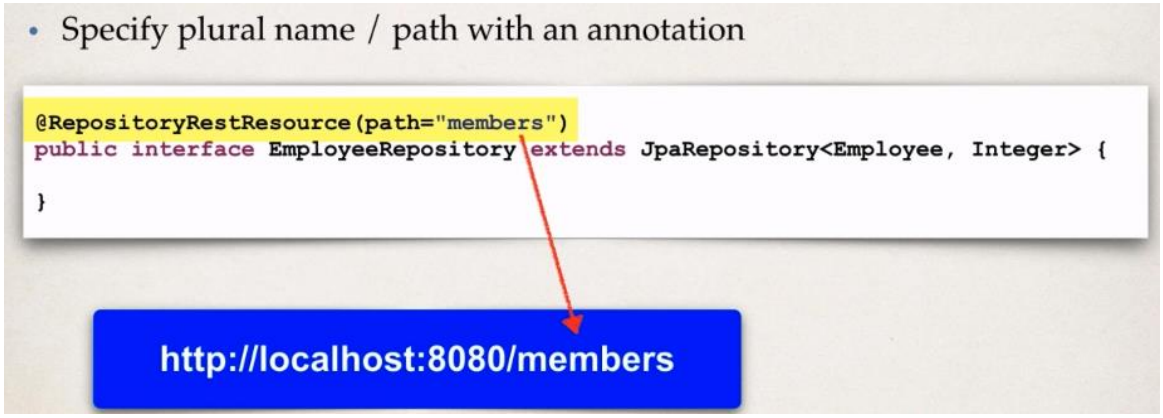


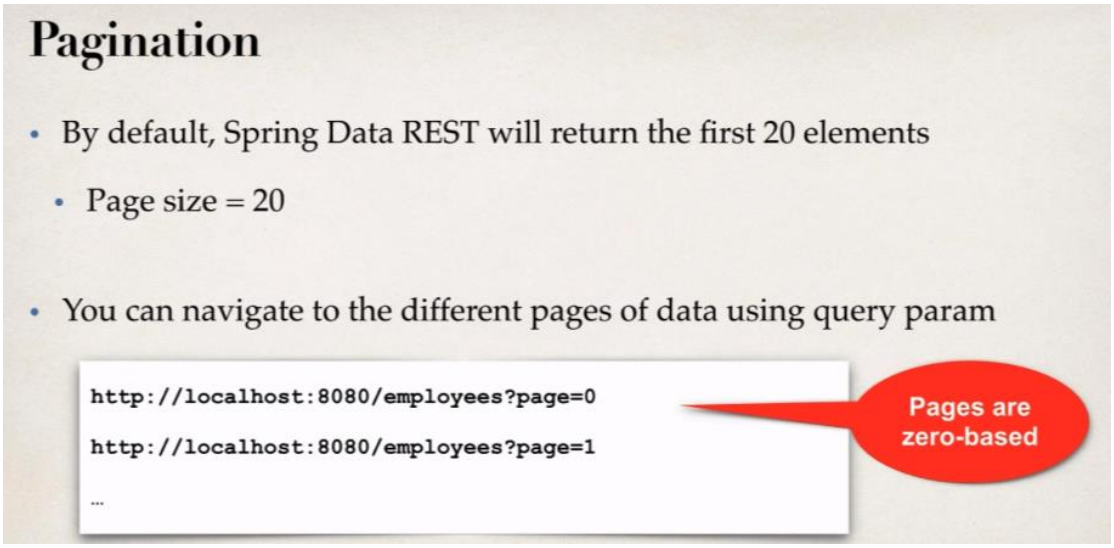
Spring Data REST Configuration, Pagination and Sorting:



- Spring Data REST does not handle complex pluralized forms.
  - In this case we need to specify plural name.
- What if we want to expose a different resource name?
  - Instead of **`/employees`**, we want to expose **`/members`**

Solution:





Spring Data REST Configuration

- Following properties available: application.properties

Name	Description
`spring.data.rest.base-path`	Base path used to expose repository resources
`spring.data.rest.default-page-size`	Default size of pages
`spring.data.rest.max-page-size`	Maximum size of pages
...	...

More properties available

www.luv2code.com/spring-boot-props

`spring.data.rest.*`

Sample Configuration

http://localhost:8080/magic-api/employees

File: application.properties

```
spring.data.rest.base-path=/magic-api

spring.data.rest.default-page-size=50
```

Returns 50 elements per page

Sorting

- You can sort by the property names of your entity
  - In our Employee example, we have: **`firstName`**, **`lastName`** and **`email`**
- Sort by last name (ascending is default) `http://localhost:8080/employees?sort=lastName`
- Sort by first name, descending `http://localhost:8080/employees?sort=firstName,desc`
- Sort by last name, then first name, ascending

`http://localhost:8080/employees?sort=lastName,firstName,asc`