Bộ môn: Kỹ Thuật Phần Mềm

Lập trình WWW *Java*

.

Spring Boot

Kỹ thuật Phần mềm – Phạm Quảng Tri

Spring Boot Introduction

Learn how to …

- Quickly develop Spring Boot applications
- 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
- Leverage all Java configuration (no xml) and Maven

Practical

- Introduction to Spring Boot development
- Not an A to Z reference
- For complete reference, see Spring Boot Reference Manual

https://spring.io/projects/spring-boot

Problems

» Building a Spring application is really HARD!!!

Q: What Maven archetype to use?

That's just basics for getting start

Q: Which Maven dependencies do I need?

Q: How do I set up configuration (xml or Java)?

Q: How do I install the server? (Tomcat, JBoss etc...)

Problems

» Tons of configuration

Very error-prone

```
<!-- 1. Configure Spring dispatcher servlet -->
<servlet>
    <servlet-name>dispatcher</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet
```

```
@Configuration
```

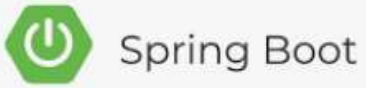
There should be an easier solution

```
<bean
        class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/view/" />
        <property name="suffix" value=".jsp" />
</bean>
<bean id="messageSource"
      class="org.springframework.context.support.ResourceBundleMessageSource" >
    <property value="resources/messages" name="basenames"/>
</bean>
```

Spring Boot Solution

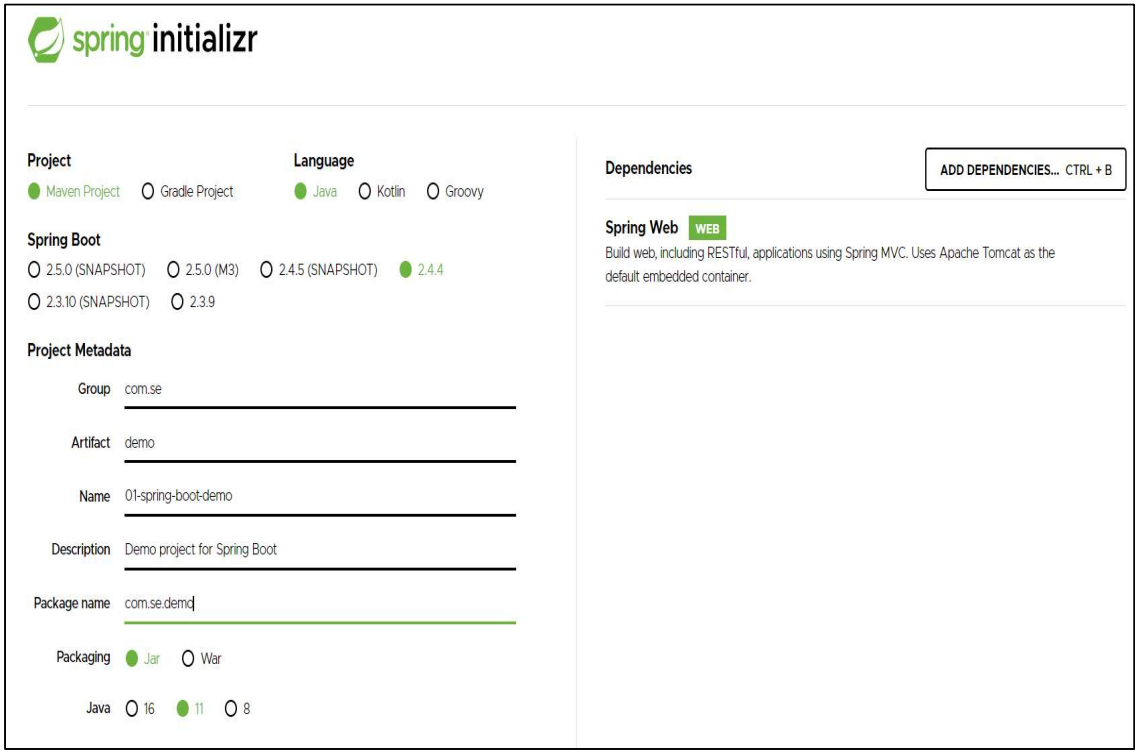
- Make it **easier to get started** with Spring development
- Minimize the amount of manual configuration
- Perform **auto-configuration based on props files** and **JAR classpath**
- Help to resolve **dependency conflicts** (Maven or Gradle)
- Provide an **embedded HTTP server** so you can get started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

Spring Initializr

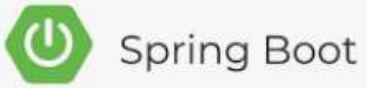


- Quickly create a starter Spring project
- Select your dependencies
- Creates a Maven/Gradle project
- Import the project into your IDE
- Eclipse, IntelliJ, NetBeans etc …

http://start.spring.io



Spring Boot Embedded Server

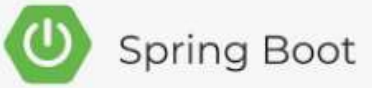


- Provide an embedded HTTP server so you can get started quickly

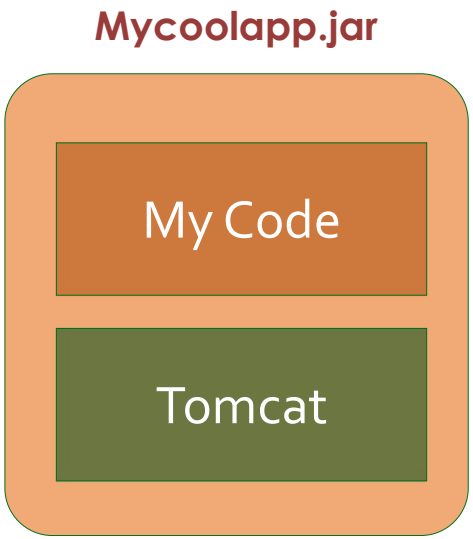
  Tomcat, Jetty, Undertow, …

- No need to install a server separately

Running Spring Boot Apps

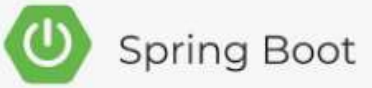


- Spring Boot apps can be run standalone (includes embedded server)
- Run the Spring Boot app from the IDE or command-line

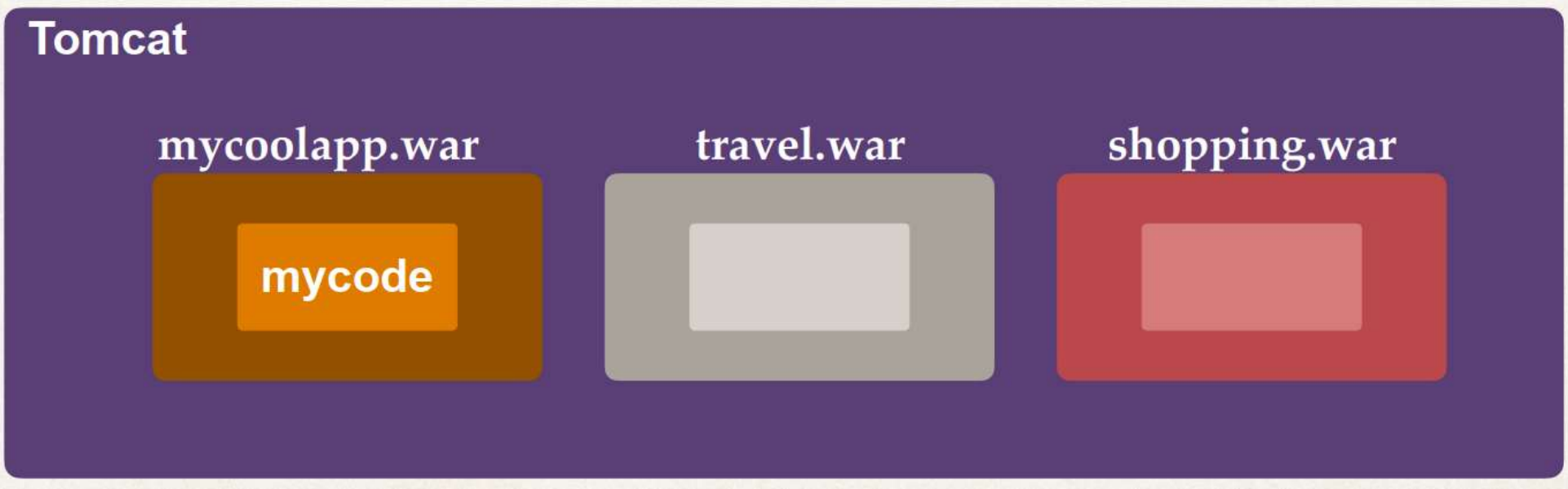



```
> java -jar mycoolapp.jar
```

Deploying Spring Boot Apps



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

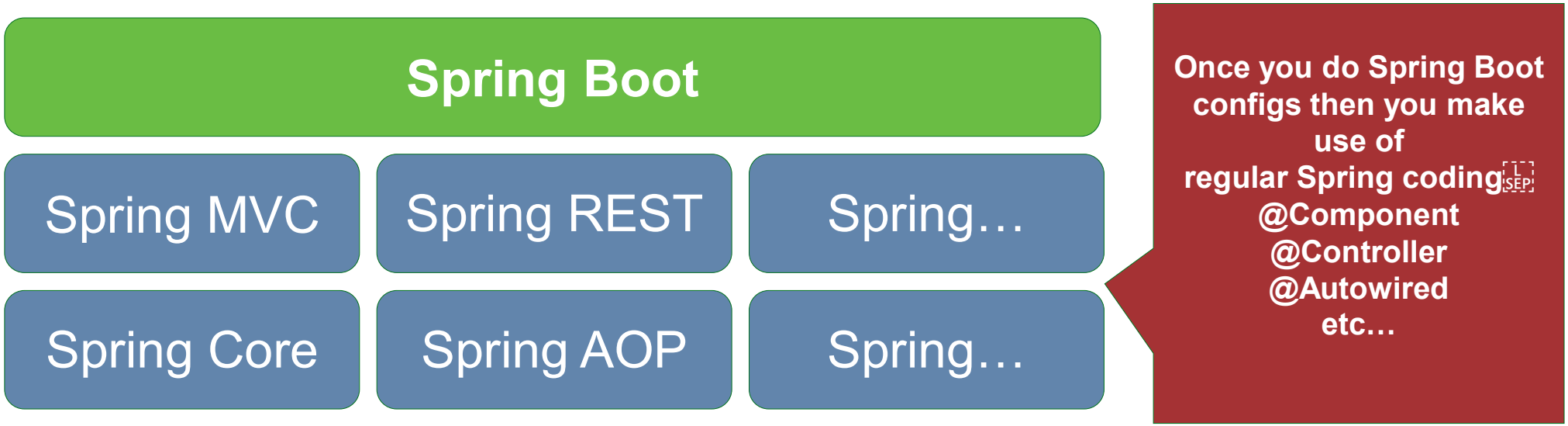


Spring Boot FAQ #1

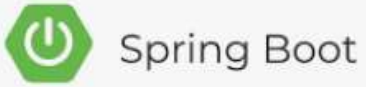
Spring Boot

Q: Does Spring Boot replace Spring MVC, Spring REST etc …?

» No. Instead, Spring Boot actually uses those technologies



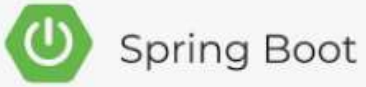
Spring Boot FAQ #2



Q: Does Spring Boot run code faster than regular Spring code?

- No.
- Behind the scenes, Spring Boot uses same code of Spring Framework
- Remember, Spring Boot is about making it easier to get started
- Minimizing configuration etc …

Spring Boot FAQ #3



Q: Do I need a special IDE for Spring Boot?

- No.
- You can use any IDE for Spring Boot apps … even use plain text editor
- The Spring team provides free Spring Tool Suite (STS) [IDE plugins]
- Some IDEs provide fancy Spring tooling support
- Not a requirement. Feel free to use the IDE that works best for you

Spring Boot Initializr Demo

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 …

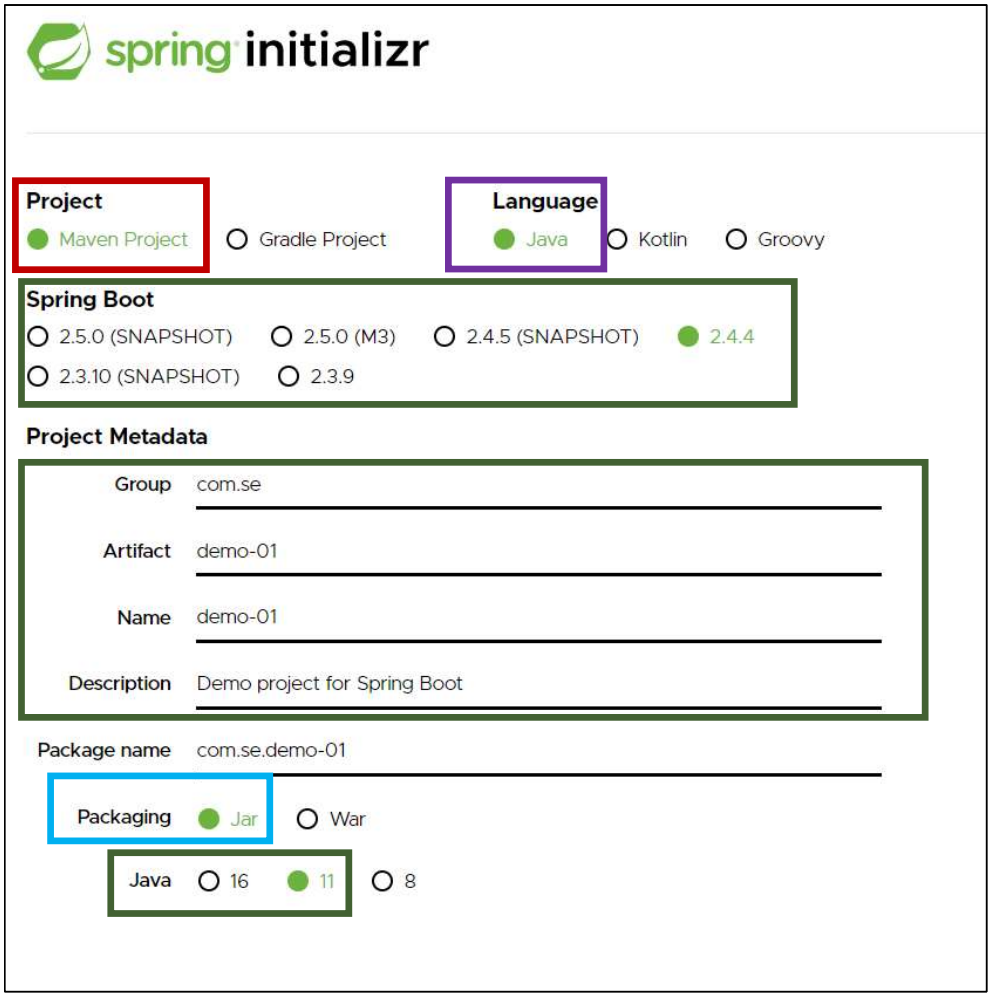
Development Process

- Configure our project at Spring Initializr website
- Download the zip file
- Unzip the file
- Import/open Maven project into our IDE

Step 1: Configure our project at Spring Initializr website

Spring Boot

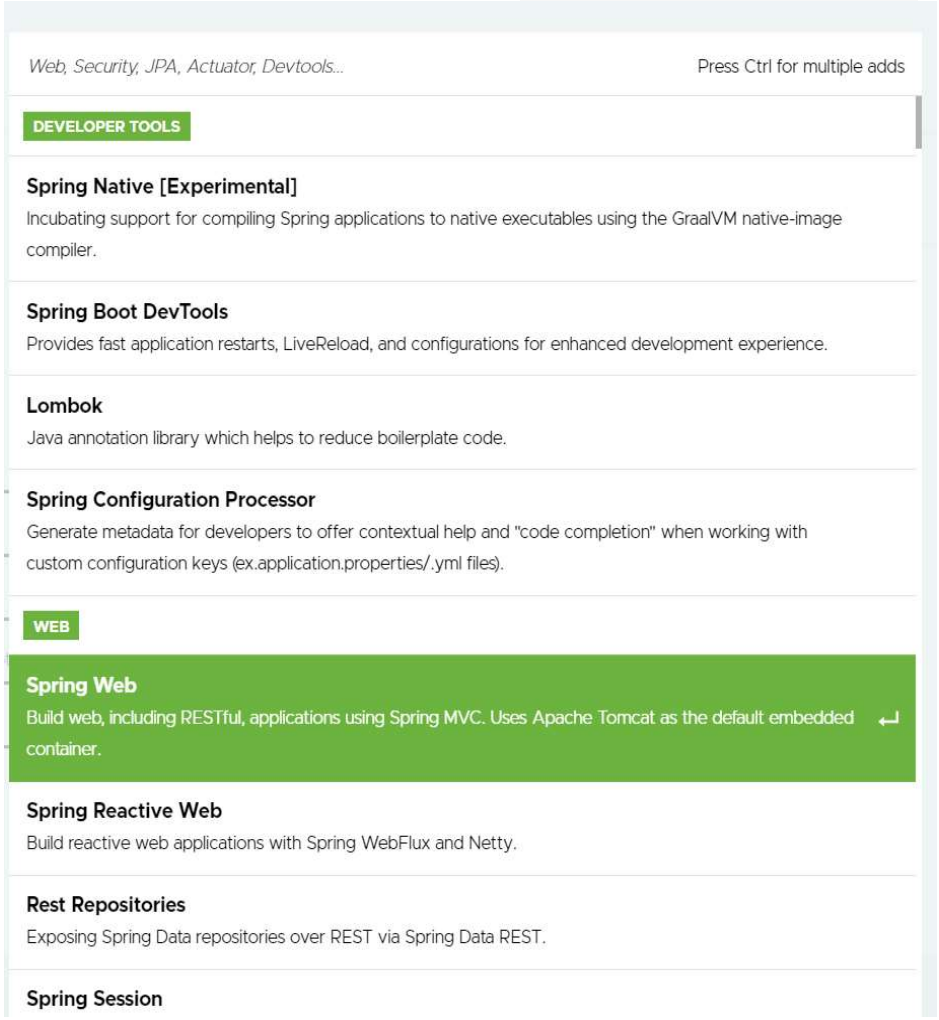
http://start.spring.io



Step 2: Configure our project at Spring Initializr website

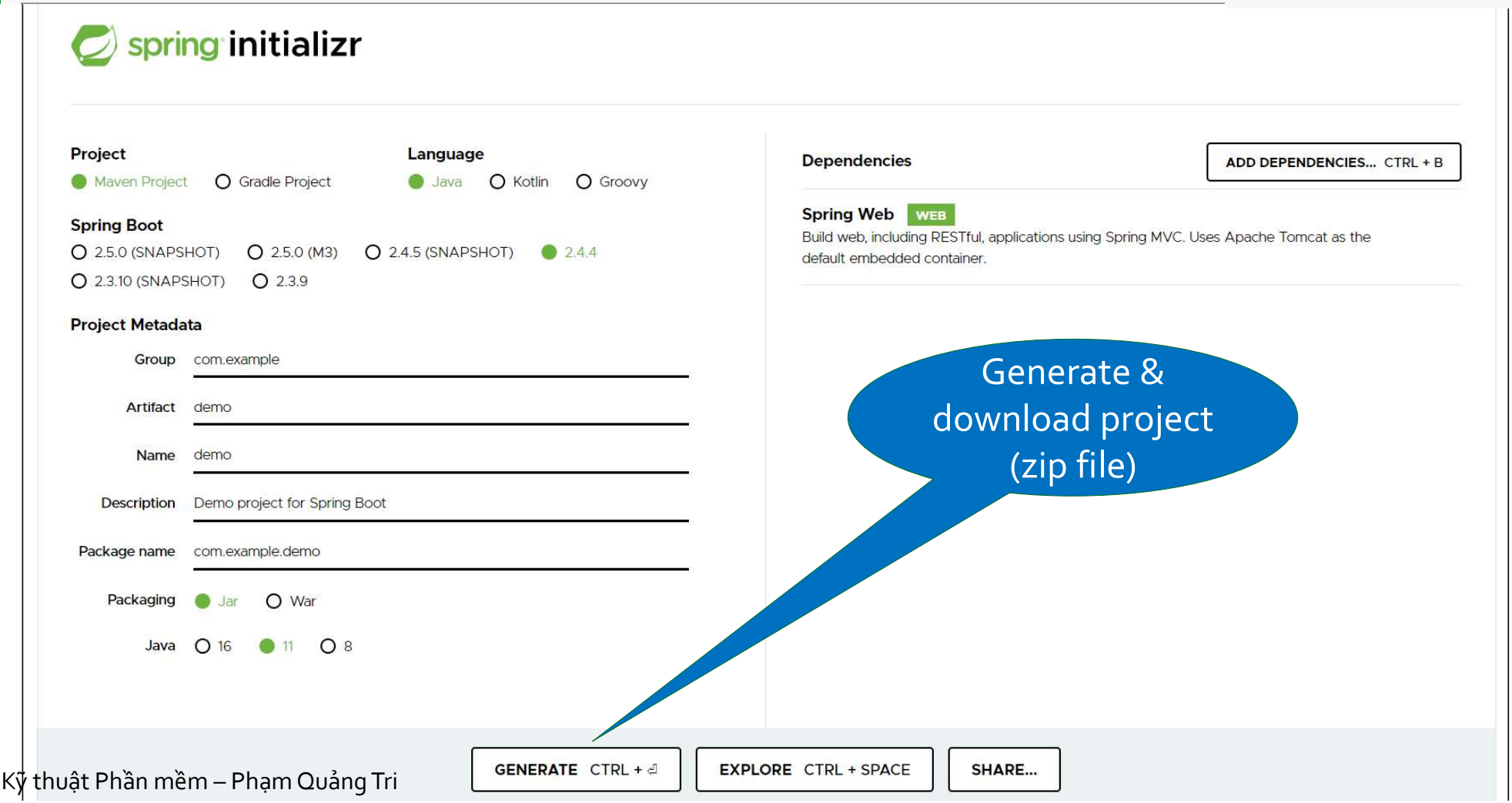
Spring Boot

http://start.spring.io

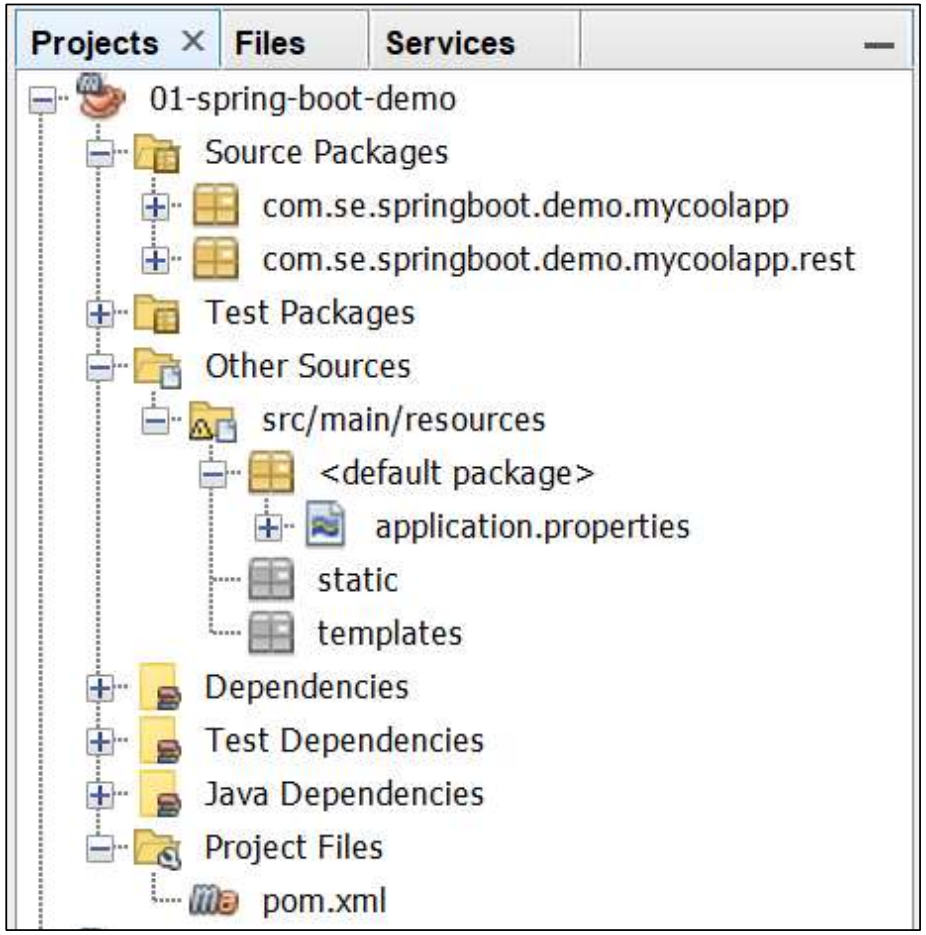


Step 2: Download the zip file

Spring Boot



Step 3, 4 : Unzip and open project into IDE



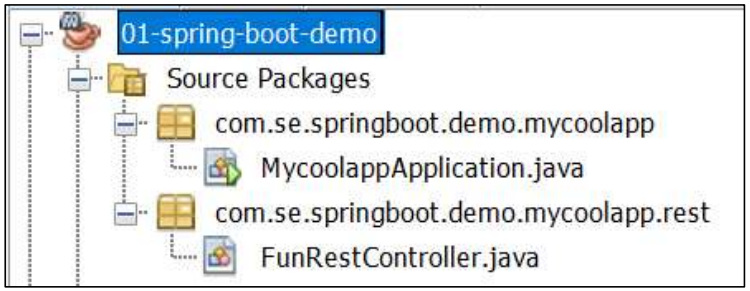
Create REST Controller

REST Controller

Spring Boot

» Let's create a very simple REST Controller

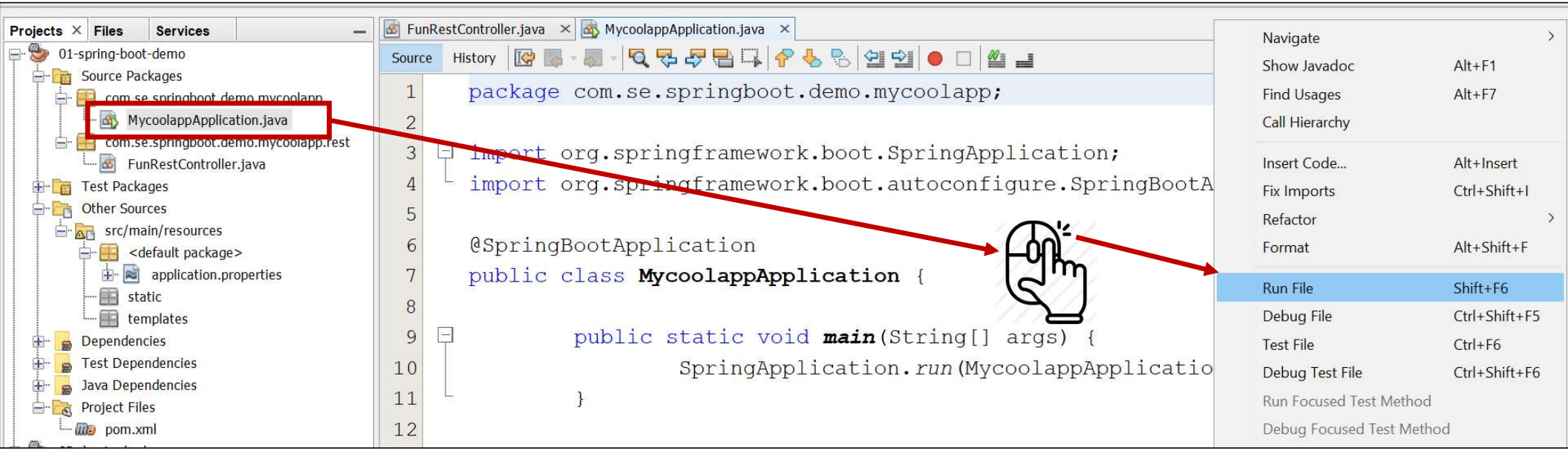
REST Controller



File: FunRestController (project: 01-spring-boot-demo)

```
@RestController
public class FunRestController {
//expose "/" that return Hello World
@GetMapping("/")
public String sayHello()
{
    return "Hello World! Time on server is " + LocalDateTime.now();}
}
```

REST Controller

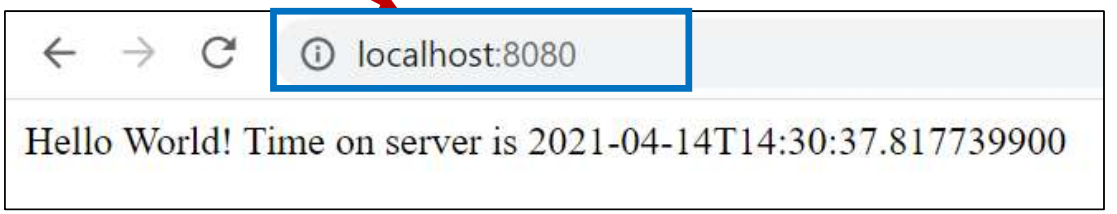


REST Controller

Spring Boot

File: FunRestController (project: 01-spring-boot-demo)

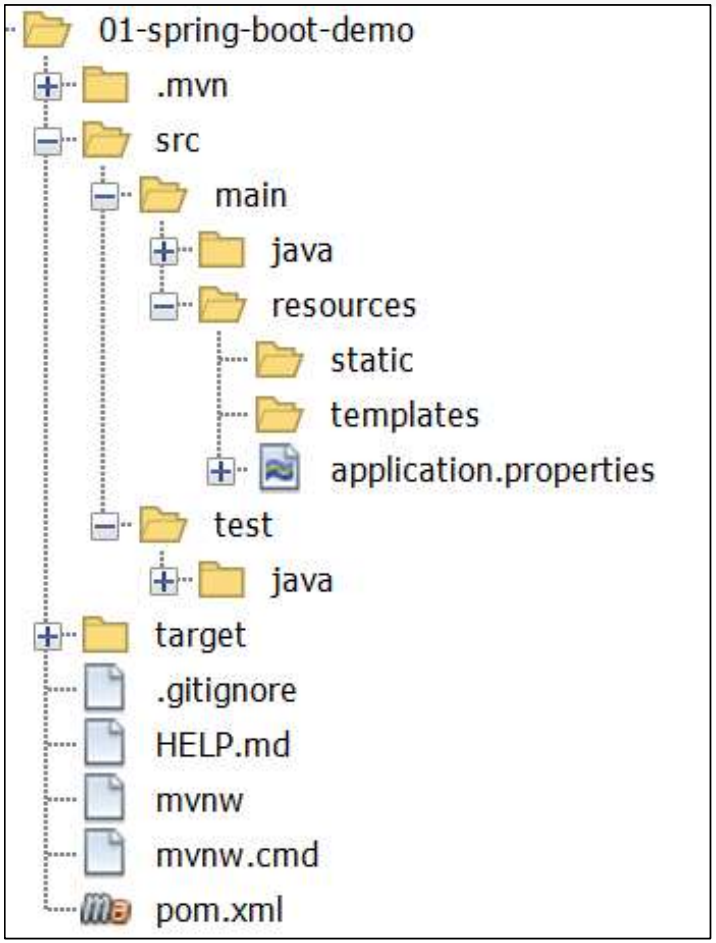
```java
@RestController
public class FunRestController {
//expose "/" that return Hello World
@GetMapping("/")
public String sayHello()
{
    return "Hello World! Time on server is " + LocalDateTime.now();}
}
```



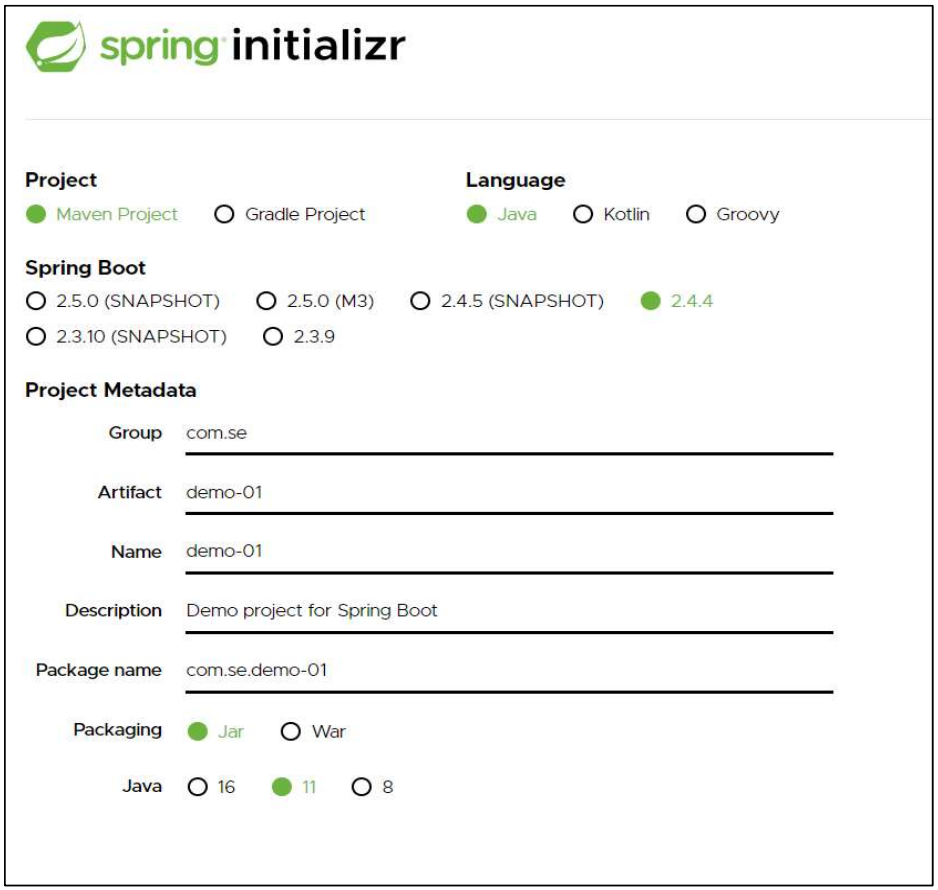
Spring Boot Project Structure

Let's explore the project structure

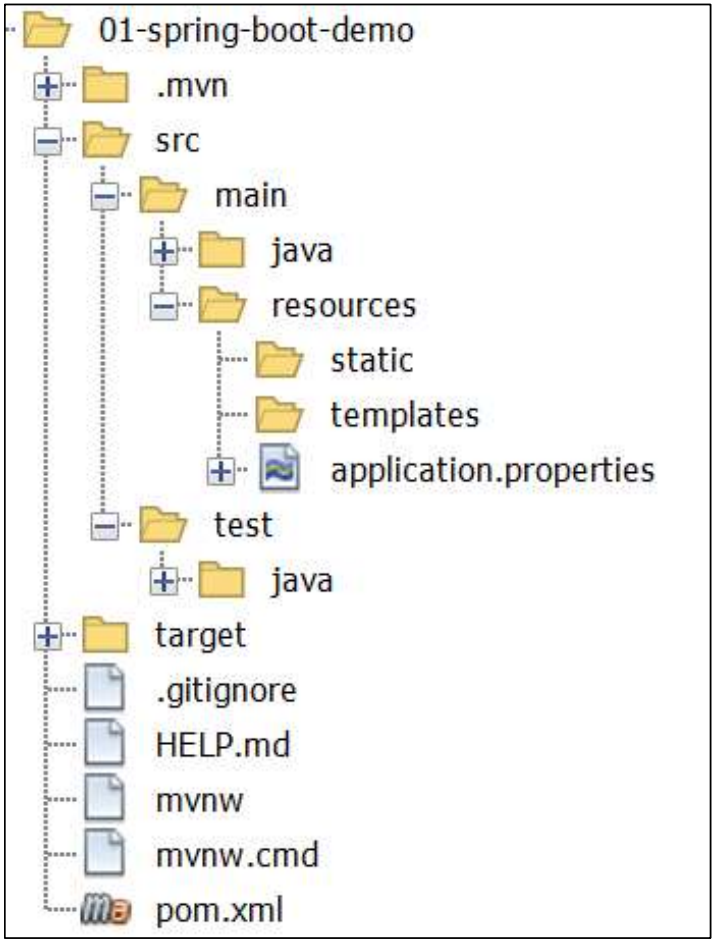
Spring Boot



http://start.spring.io



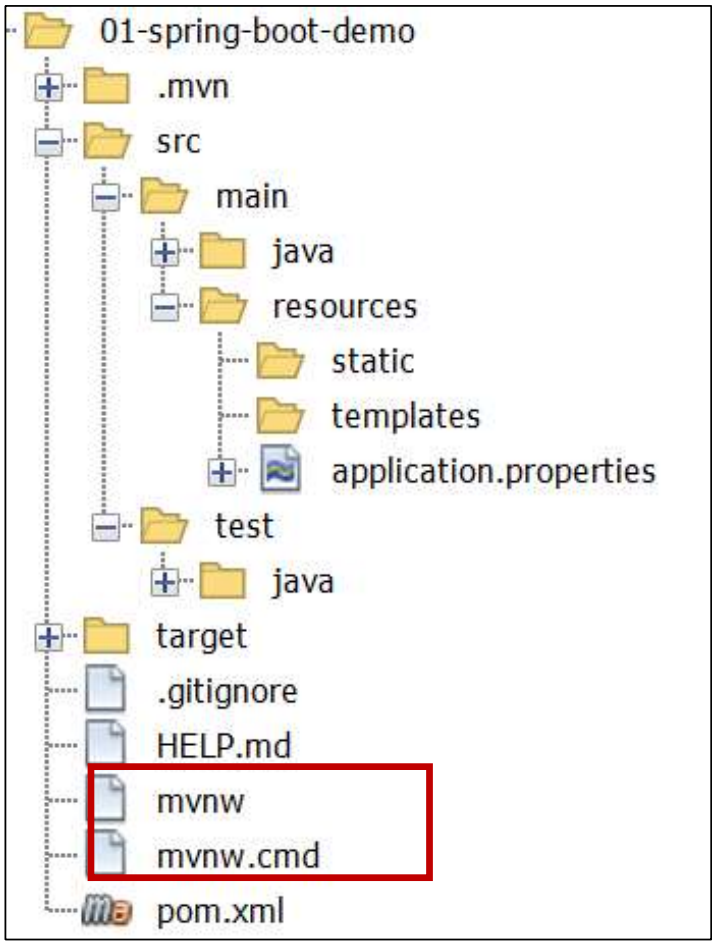
Maven Standard Directory Structure



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

Maven Wrapper files

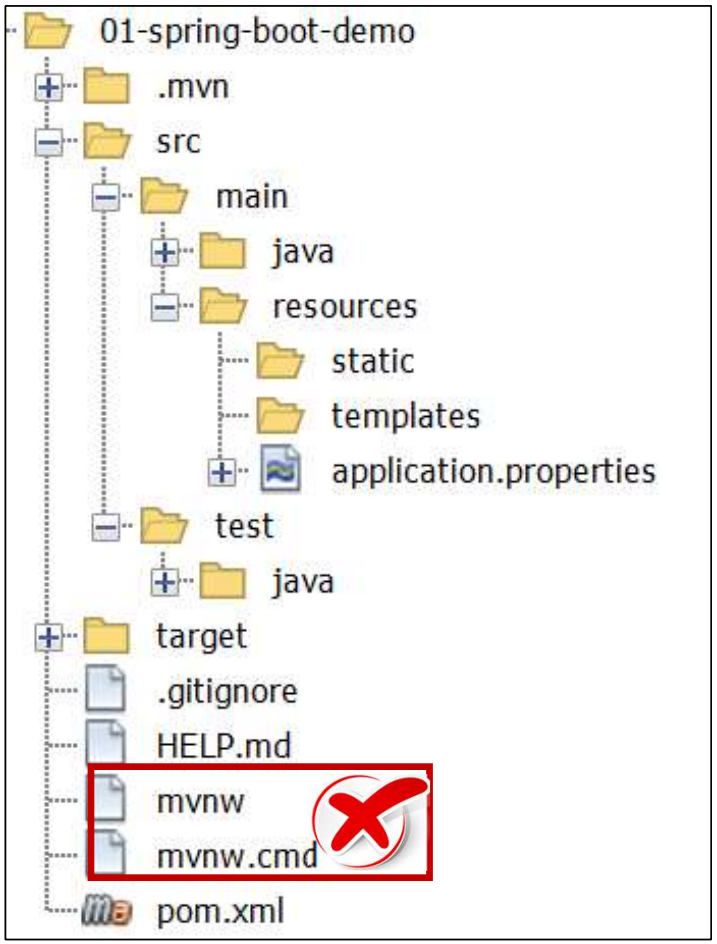
Spring Boot



- **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
- Two files are provided
  - **mvnw.cmd** for MS Windows
  - **mvnw.sh** for Linux/Mac

Maven Wrapper files

Spring Boot

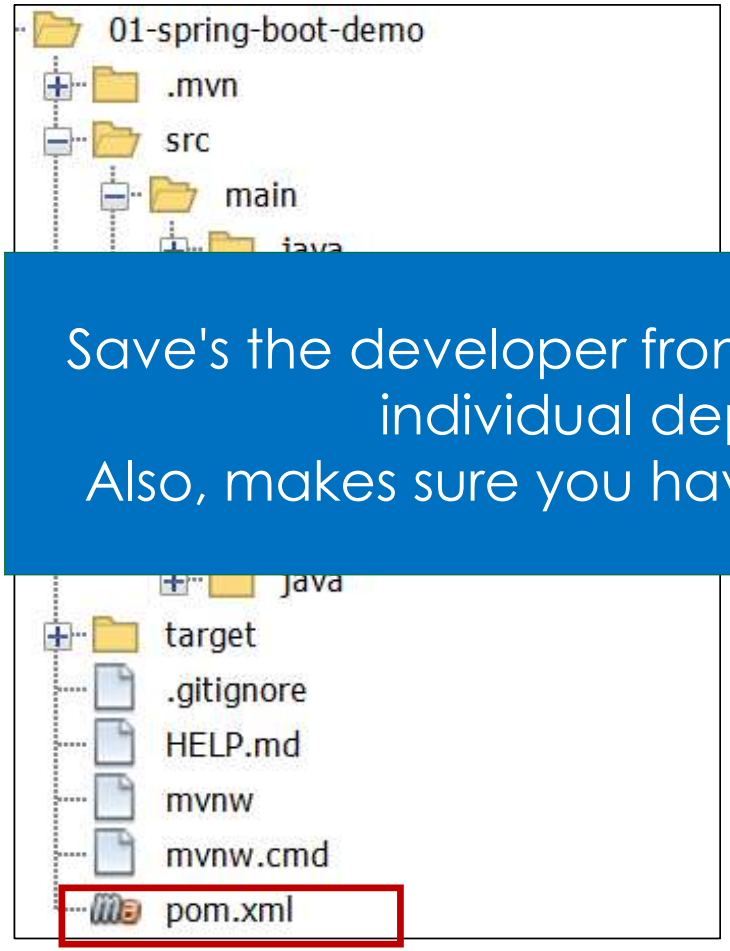


- If you already have Maven installed previously → then you can **ignore/delete** the mvnw files
- Just use Maven as you normally would

```
> mvnw clean compile test
```

Maven POM file

Spring Boot



» **pom.xml** includes info that you entered at Spring Initializr website

```
<groupId>com.se.springboot.demo</grou
<dependencies>
        <groupId>org.spr
        <artifactId>spri
        <scope>test</sc
    </dependency>
</dependencies>
```

Save's the developer from having to list all of the individual dependencies
Also, makes sure you have compatible versions

Spring Boot Starters
A collection of Maven dependencies
(Compatible versions)

Spring WEB
Spring WEB-MVC
Hiberbate Validator
Tomcat
JSON

Maven POM file

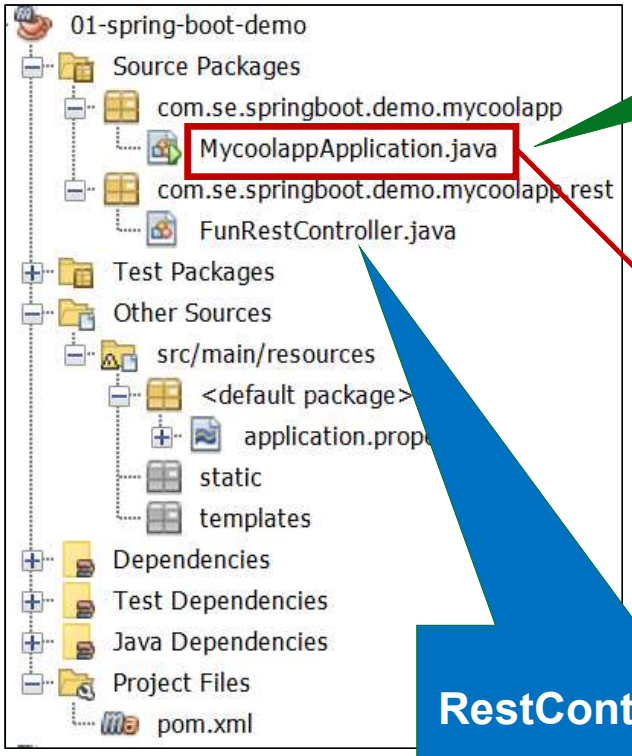
To package executable jar
or war archive
Can also easily run the app

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

```
> mvnw package
> mvnw spring-boot:run
```

Java Source Code

Spring Boot



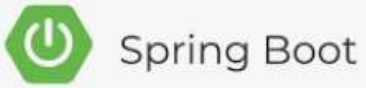
Main Spring Boot application class
Created by Spring Initializr

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;


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

RestController that we created

Java Source Code




```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.S

@SpringBootApplication
public class MycoolappApplication {

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

Composed of following annotations
@EnableAutoConfiguration
@ComponentScan
@Configuration

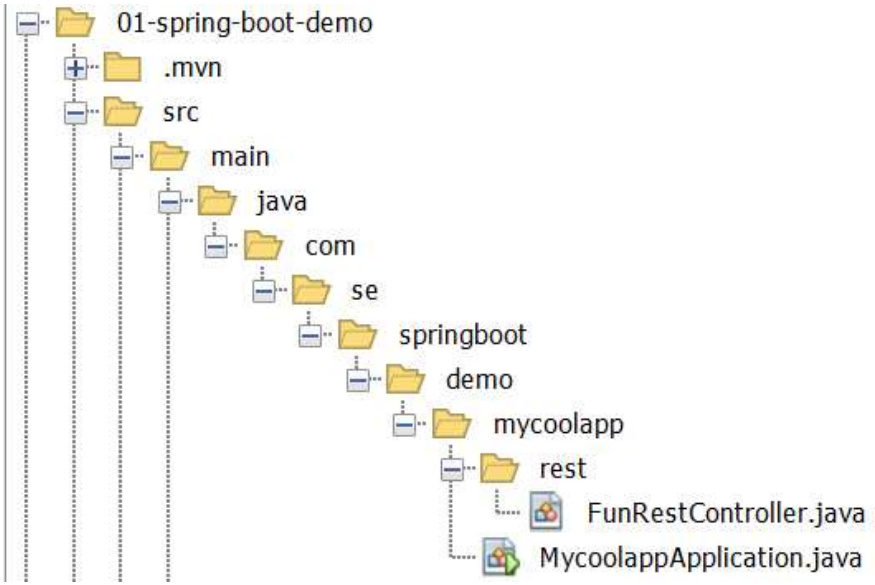
Bootstrap your Spring Boot application

More on Component Scanning – Best Practice

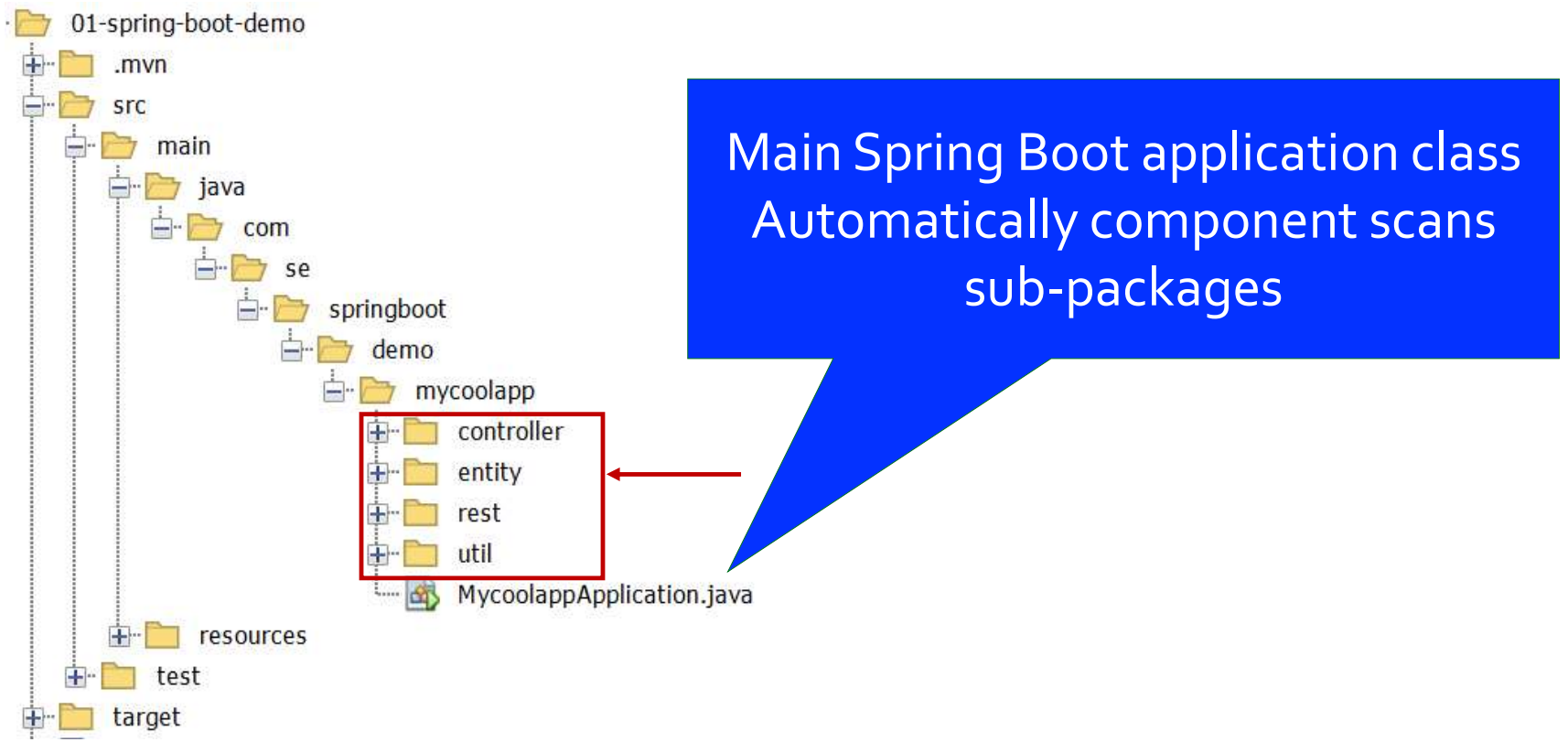
Place your main application class in the root package above your other packages

This implicitly defines a base search package

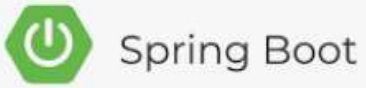
- Allows you to leverage default component scanning
- No need to explicitly reference the base package name



More on Component Scanning – Best Practice



More on Component Scanning – Best Practice



How a bout other packages?

- **`org.acme.iot.utils`**
- **`edu.cmu.wean`**

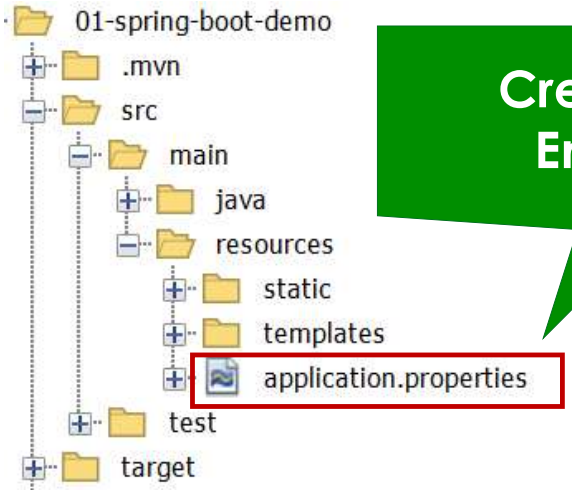
Explicitly list base packages to scan

```
package com.se.springboot.demo.mycoolapp;
…
@SpringBootApplication(
scanBasePackages={"com.se.springboot.demo.mycoolapp",
"org.acme.iot.utils",
"edu.cmu.wean"})
public class MycoolappApplication {
…
}
```

Application Properties

Spring Boot

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
```

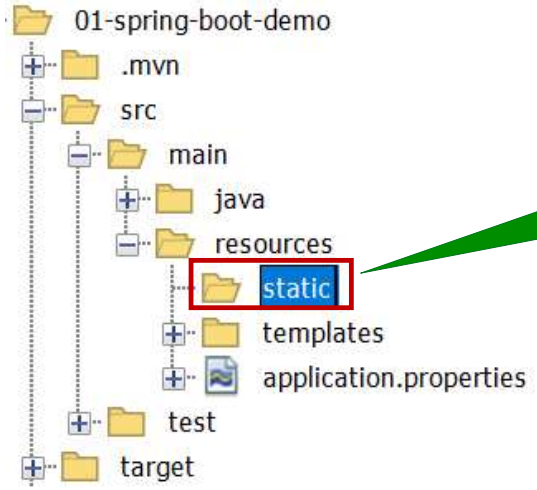
Also add your own custom properties

```
coach.name=Mickey Mouse
```

Static Content

Spring Boot

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



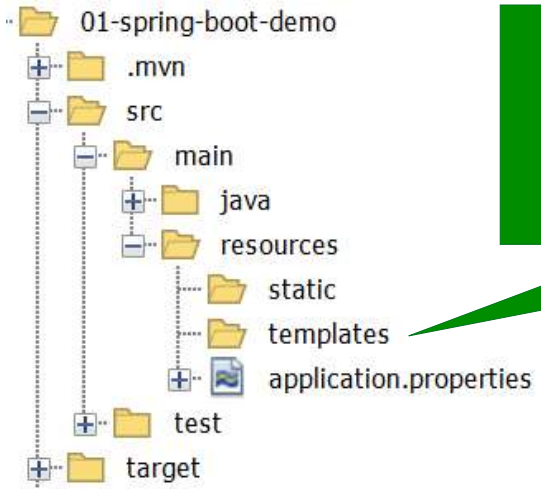
By default, Spring Boot will load static resources from "/static" directory

Examples of static resources
HTML files, CSS, JavaScript, images, etc …

Templates

Spring Boot includes auto-configuration for following template engines

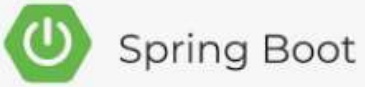
- FreeMarker
- Thymeleaf
- Mustache



By default, Spring Boot will load templates from "/templates" directory

Spring Boot Starters

Spring Boot Starters



- A **list** of **Maven dependencies**
- A collection of dependencies grouped together
- **Tested** and **verified** by the **Spring Development team**
- Makes it much **easier** for the **developer** to get started with Spring
- Reduces the amount of Maven configuration

Example: Spring MVC

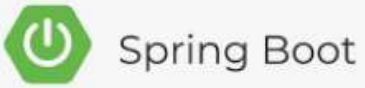
» when building a Spring MVC app, you normally need

```
<!-- 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>
```

Example: Spring MVC



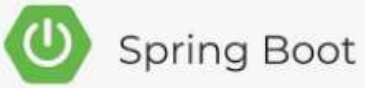
» With with Spring Boot Starter

A collection of Maven dependencies (Compatible versions)

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

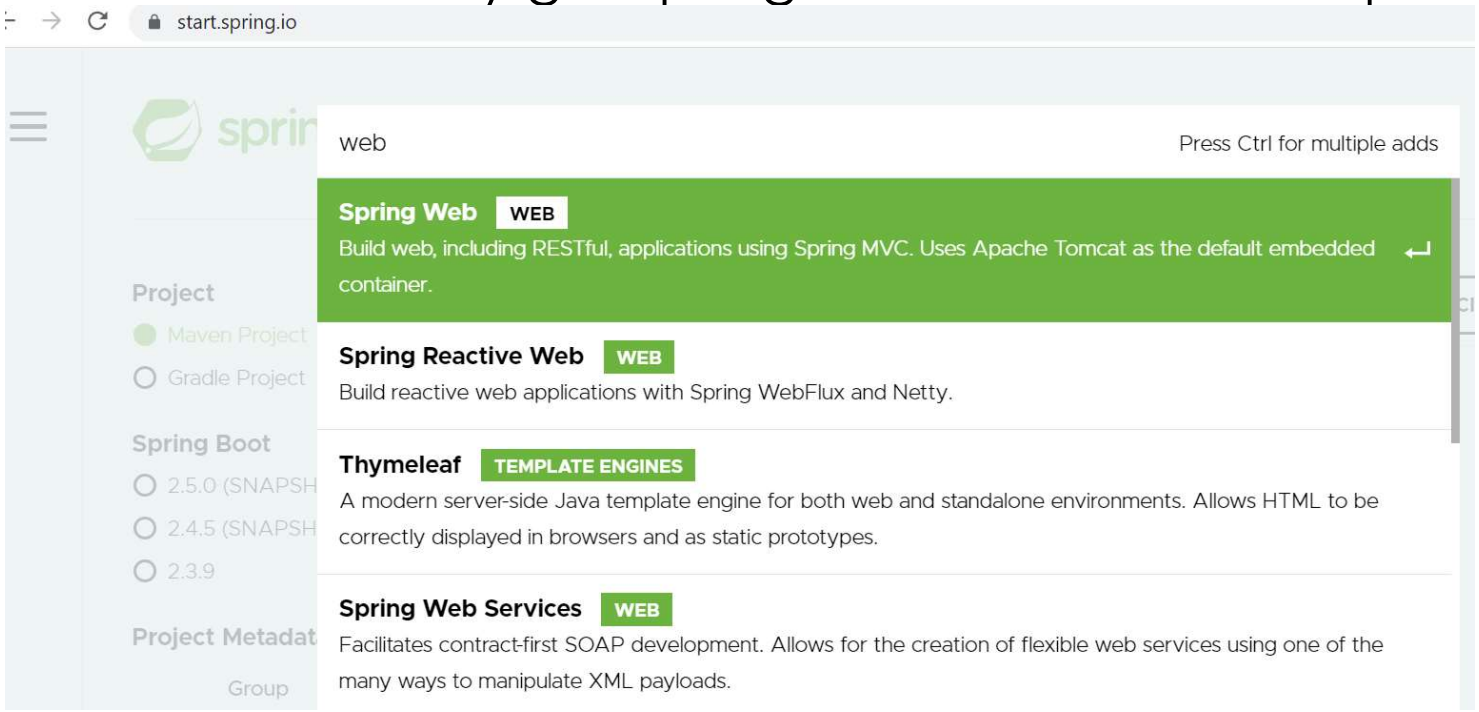
CONTAINS
spring-web
spring-webmvc
hibernate-validator
json

Spring Initializr

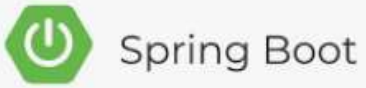


» In Spring Initializr, simply select Web dependency

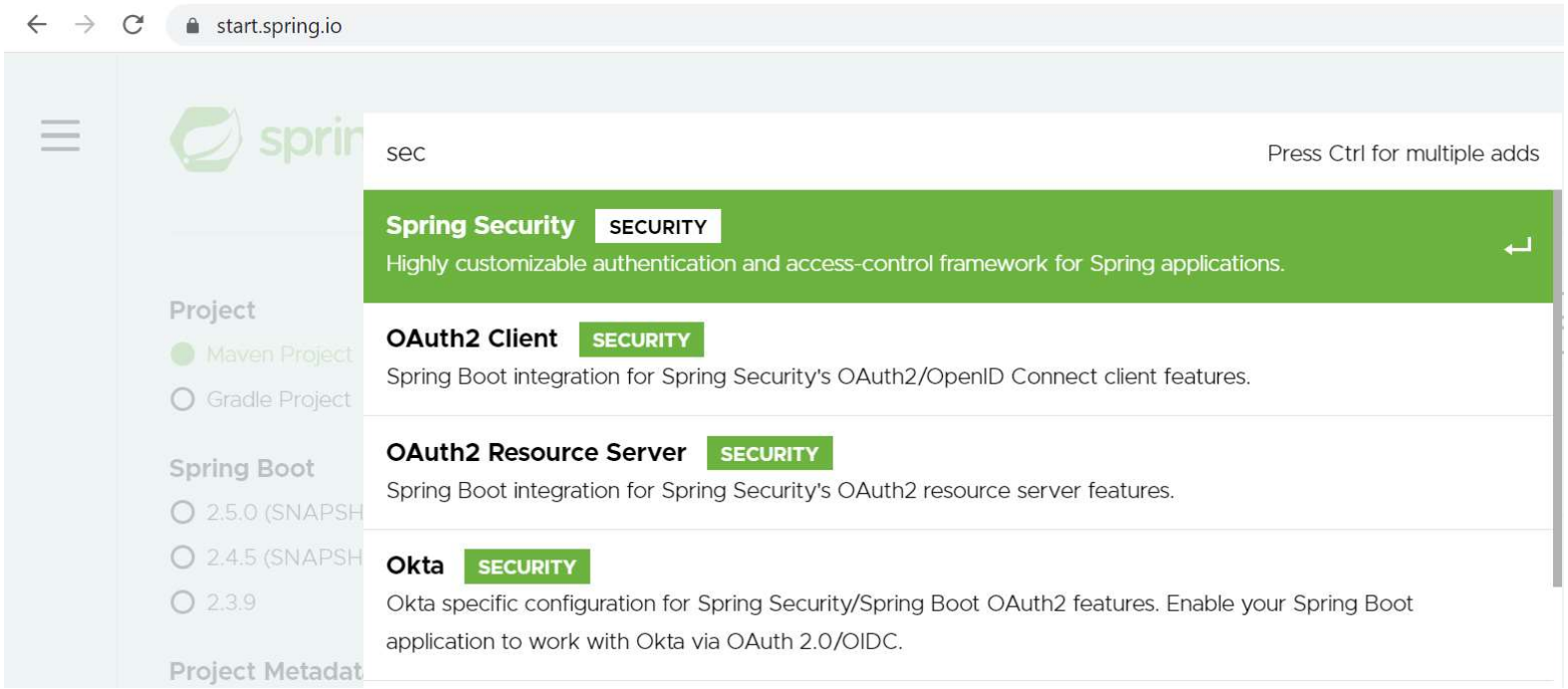
» You automatically get spring-boot-starter-web in pom.xml



Spring Initializr



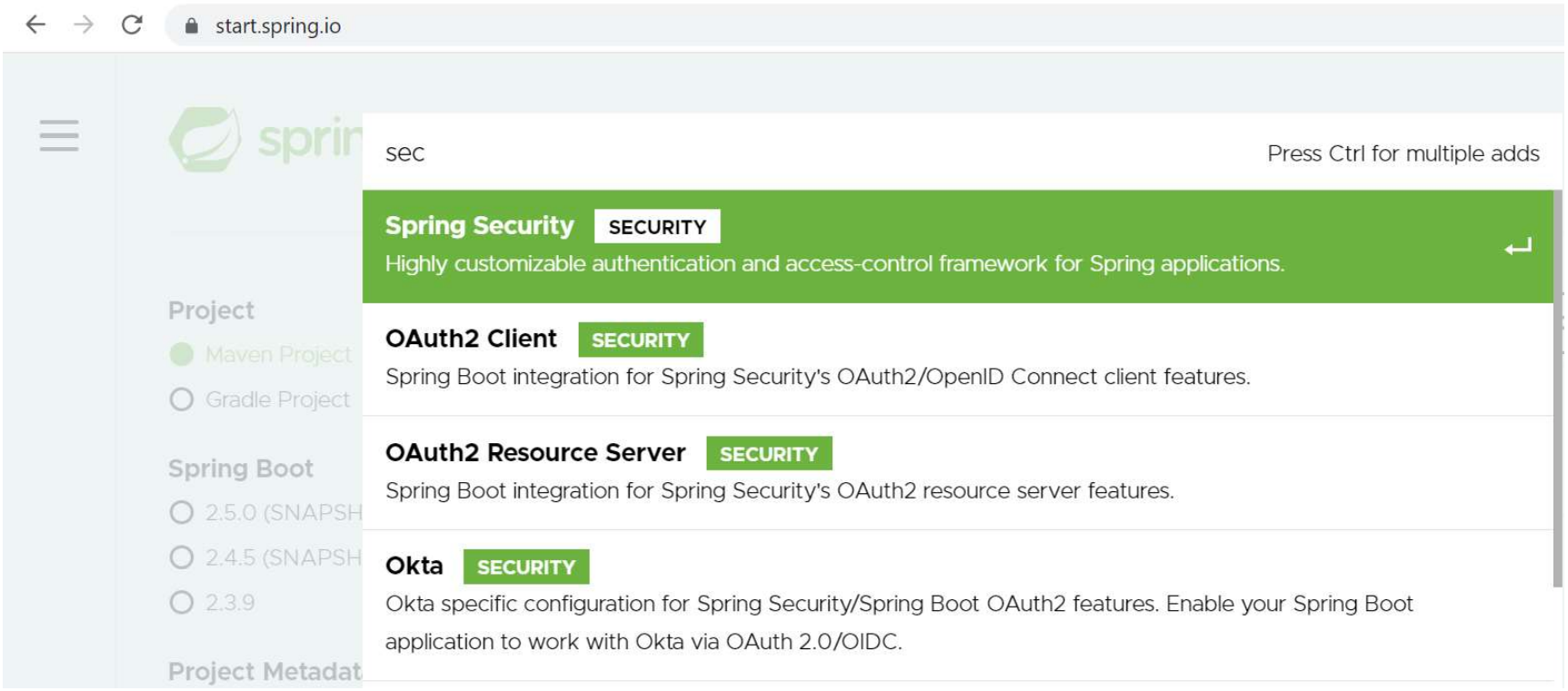
» If we are building a Spring app that needs : Web, Security, …



Spring Initializr

Spring Boot

» If we are building a Spring app that needs: Web, Security, …



Spring Initializr

» If we are building a Spring app that needs: Web, Security, …

```
<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>
</dependencies>
```

Spring Boot Starters

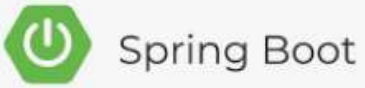
» There are 30+ Spring Boot Starters from the Spring Development team

» Link: https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/#using-boot-starter

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**
...	

Spring Boot Starter Parent

Spring Boot Starter Parent



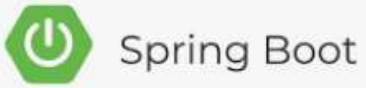
» Maven defaults defined in the Starter Parent

- UTF-8 source encoding
- Others

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.4.4</version>
    <relativePath/>
</parent>
```

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

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>2.4.4</version>
    <relativePath/>
</parent>
```

Specify version of Spring Boot

```
<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>
```

Inherit version from Starter Parent

Spring Boot Dev Tools

Problem

» When running Spring Boot applications

- If you make changes to your source code
- Then you have to manually restart your application ☹

» **Solution: `spring-boot-devtools`**

- Automatically restarts your application when code is updated
- Simply add the dependency to your POM file
- No need to write additional code ☺

Spring Boot Dev Tools

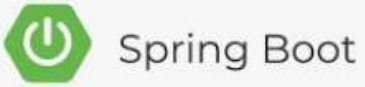
» Adding the dependency to your POM file

File: Pom.xml (project 02-dev-tools-demo)

```
<!-- ADD SUPPORT FOR AUTOMATIC RELOADING -->
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-devtools</artifactId>
</dependency>
```

Automatically restarts your application when code is updated

Spring Boot Dev Tools Documents

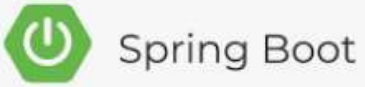


https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/#using-boot-devtools

Spring Boot: Run from Command-Line

(project: 04-command-line-demo)

Running from the Command-Line



- When running from the command-line

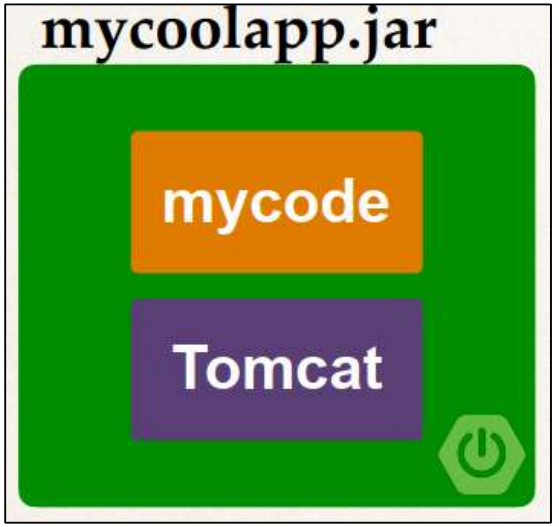
  No need to have IDE open/running

- 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**

Self-contained unit
Nothing else to install

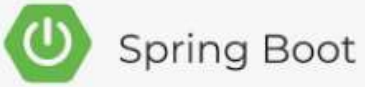


Running from the Command-Line

Two options for running the app

- Option 1: Use **java -jar**
- Option 2: Use Spring Boot Maven plugin - **mvnw spring-boot:run**

Development Process



- Exit the IDE (stop and close all project)
- Package the app using mvnw package
- Run app using java -jar
- Run app using Spring Boot Maven plugin, mvnw spring-boot:run

Package the app using mvnw package

Spring Boot

```
D:\CloudDisk\DataG\LTWWW_Java\NetbeanSpringBoot\04-command-line-demo>mvnw package
```

```
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  8.452 s
[INFO] Finished at: 2021-04-16T14:20:45+07:00
[INFO] ------------------------------------------------------------------------
```

```
D:\CloudDisk\DataG\LTWWW_Java\NetbeanSpringBoot\04-command-line-demo>cd target

D:\CloudDisk\DataG\LTWWW_Java\NetbeanSpringBoot\04-command-line-demo\target>dir
 Volume in drive D is DATA
 Volume Serial Number is 0056-533B

 Directory of D:\CloudDisk\DataG\LTWWW_Java\NetbeanSpringBoot\04-command-line-demo\target

04/12/2021  04:55 PM    <DIR>          generated-sources
04/12/2021  04:55 PM    <DIR>          generated-test-sources
04/12/2021  04:56 PM    <DIR>          maven-archiver
04/12/2021  04:55 PM    <DIR>          maven-status
04/16/2021  01:23 PM        17,065,932 mycoolapp-0.0.1-SNAPSHOT.jar
```

Run app using java -jar

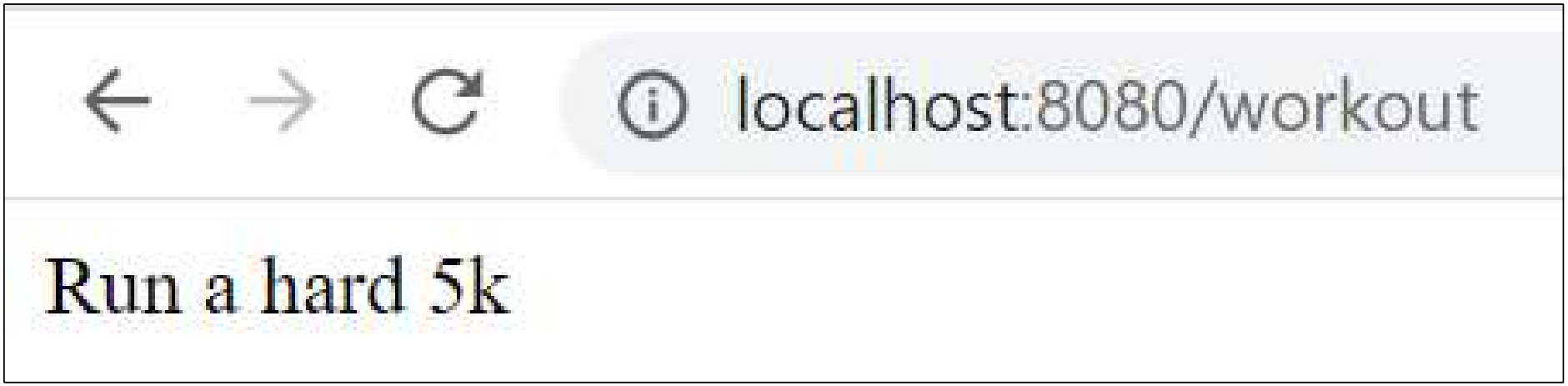
Spring Boot

```
D:\CloudDisk\DataG\LTWWW_Java\NetbeanSpringBoot\04-command-line-demo\target>java -jar mycoolapp-0.0.1-SNAPSHOT.jar
```

```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v2.4.4)

2021-04-16 14:33:15.501  INFO 18884 --- [           main] c.s.s.d.mycoolapp.MycoolappApplication
TOP-3LILFE3 with PID 18884 (D:\CloudDisk\DataG\LTWWW_Java\NetbeanSpringBoot\04-command-line-demo\t
G\LTWWW_Java\NetbeanSpringBoot\04-command-line-demo\target)
```

Run app using java -jar

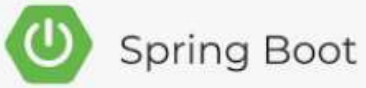


Run app using Spring Boot Maven plugin

Custom Application Properties

(project: 05-properties-demo)

Prolem



» You need for your app to be configurable … no hard-coding of values

» You need to read app configuration from a properties file

Solution: Application Properties file

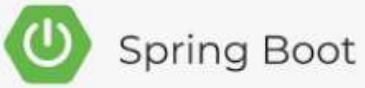
By default, Spring Boot reads information from a standard properties file located at: src/main/resources/application.properties

Standard Spring Boot file name

- You can define ANY custom properties in this file
- Your Spring Boot app can access properties using @Value

No additional coding or configuration required

Development Process



- Define custom properties in application.properties
- Inject properties into Spring Boot application using @Value

Step 1: Define custom properties in application.properties

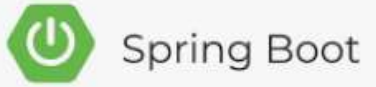
Spring Boot

File: src/main/resources/application.properties

```
#
# Define custom properties
#
coach.name=Mickey Mouse
team.name=The Cat Club
```

You can give ANY custom property names

Step 2: Inject properties into Spring Boot application




```
@RestController
public class FunRestController {
// inject properties for: coach.name and team.name
@Value("${coach.name}")
private String coachName;
@Value("${team.name}")
private String teamName;
```

File: src/main/resources/application.properties

```
#
# Define custom properties
#
coach.name=Mickey Mouse
team.name=The Cat Club
```

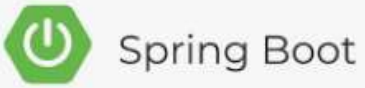
```
//expose endpoint for "teaminfo"
@GetMapping("/teaminfo")
public String getTeamInfor(){
  return "Coach:  " + coachName + " ---Team: "
          + teamName;
}
```

Spring Boot Properties

- Spring Boot can be configured in the application.properties file
- Server port, context path, actuator, security etc …
- Spring Boot has 1,000+ properties ☺!

List of Common Properties
https://docs.spring.io/spring-boot/docs/current/reference/html/appendix-application-properties.html#common-application-properties

Spring Boot Properties



Review some of the properties

```
#
#Change Spring Boot embedded server port
#default port 8080
#
server.port=7070
#
#
#
#
server.servlet.context-path=/mycoolapp
# Default HTTP session time out
server.servlet.session.timeout=15m
```

http://localhost:7070/mycoolapp/teaminfo

File: src/main/resources/application.properties

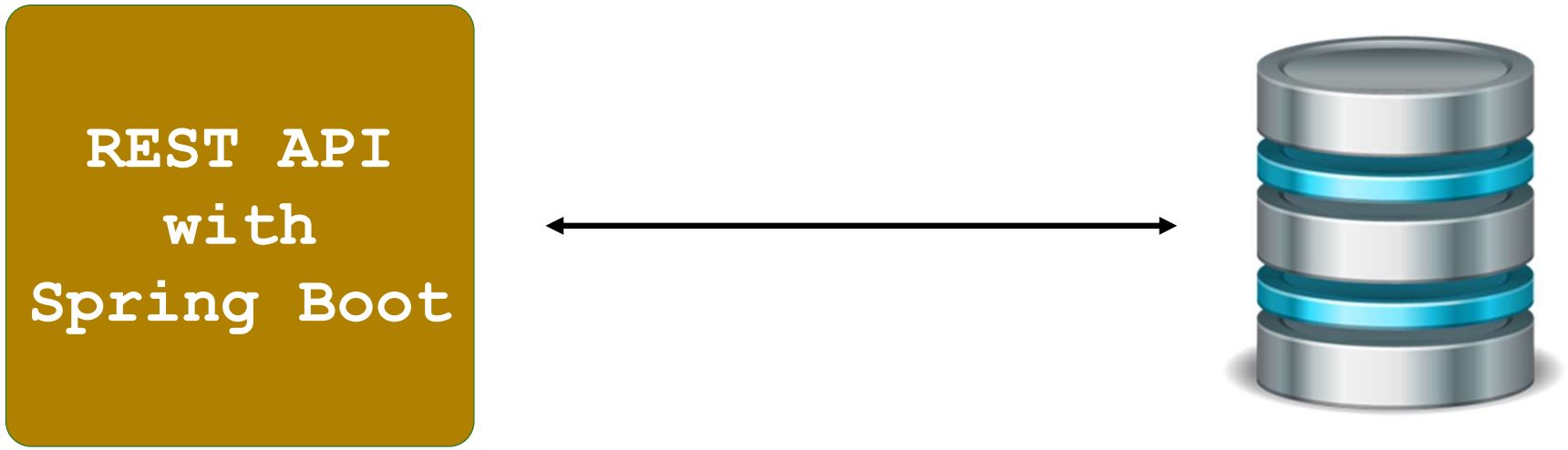
Spring Boot REST API - CRUD Project

(Project: 20-hibernate-crud-demo)

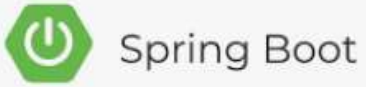
CRUD Project

Spring Boot

REST API with Spring Boot that connects to a database



API Requirements



Create a REST API for the Employee Directory

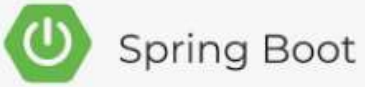
REST clients should be able to

- Get a list of employees
- Get a single employee by id
- Add a new employee
- Update an employee
- Delete an employee

REST API

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

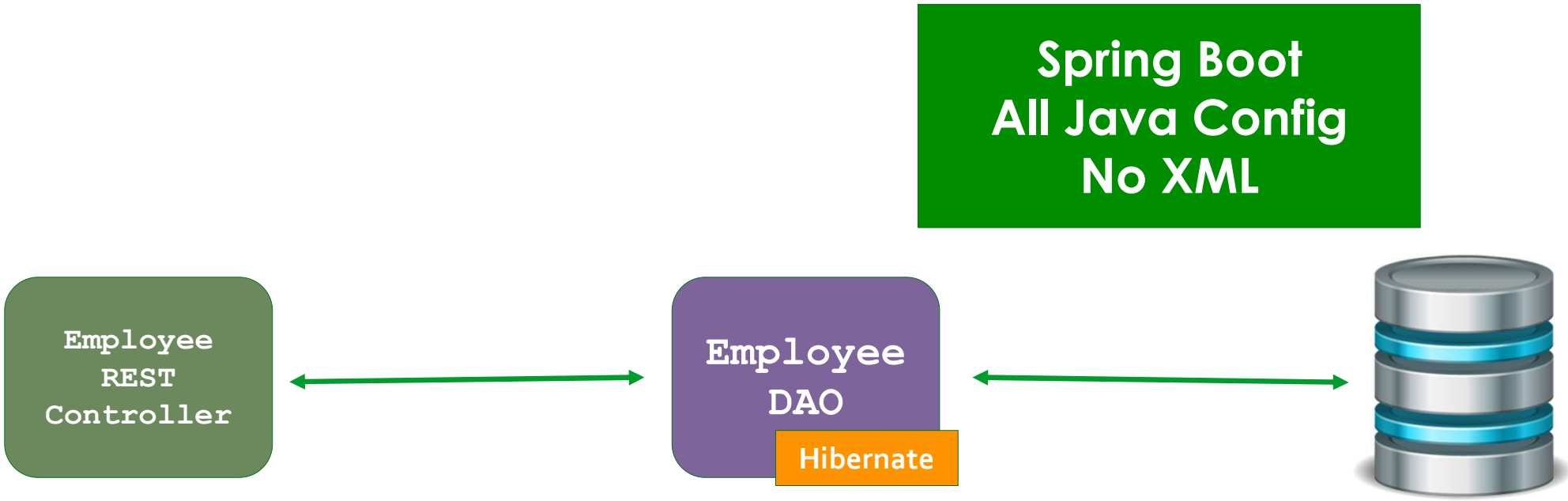
Development Process



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

Application Architecture

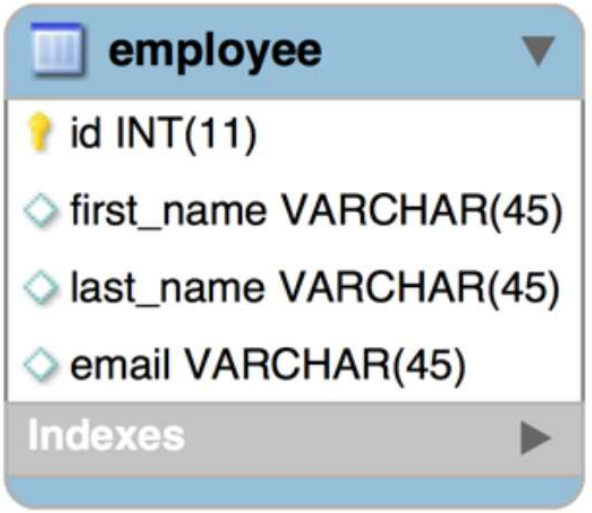
Spring Boot



Setup Database Table

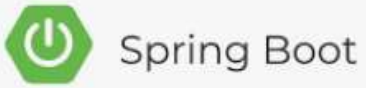
Spring Boot

1. Create database: employee_directory
2. Create table: employee
3. Input sample data



id	first_name	last_naem	email
1	John	Doe	john@gmail.com
2	David	Gueta	Gueta@secode.com
3	Mary	Public	mary@secode.com

Creating Spring Boot Project



Project
- (●) Maven Project
- ( ) Gradle Project

Language
- (●) Java
- ( ) Kotlin
- ( ) Groovy

Spring Boot
- ( ) 2.5.0 (SNAPSHOT)
- ( ) 2.5.0 (RC1)
- ( ) 2.4.6 (SNAPSHOT)
- (●) 2.4.5
- ( ) 2.3.11 (SNAPSHOT)
- ( ) 2.3.10

Project Metadata

Group	com.se
Artifact	springboot-cruddemo
Name	springboot-cruddemo
Description	Demo project for Spring Boot
Package name	com.se.springboot-cruddemo
Packaging	(●) Jar ( ) War
Java	( ) 16 (●) 11 ( ) 8

Dependencies — ADD DEPENDENCIES... CTRL + B

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

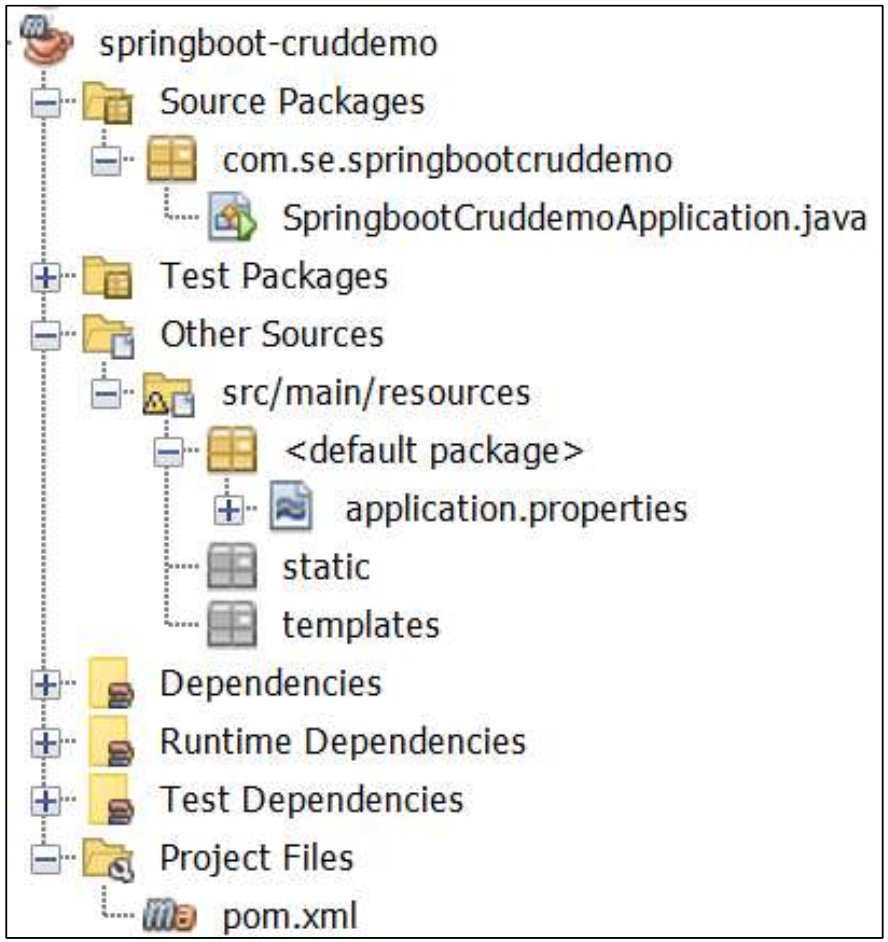
Spring Data JPA SQL
Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.

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

MS SQL Server Driver SQL
A JDBC and R2DBC driver that provides access to Microsoft SQL Server and Azure SQL Database from any Java application.

GENERATE CTRL + ⏎ | EXPLORE CTRL + SPACE | SHARE...

Creating Spring Boot Project

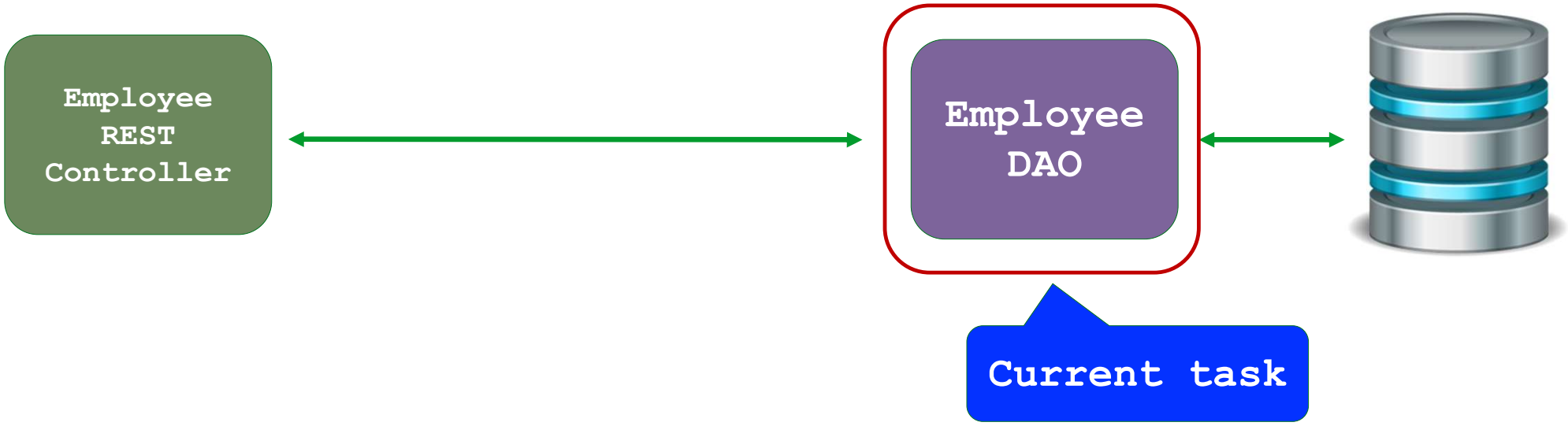


Create DAO in Spring Boot

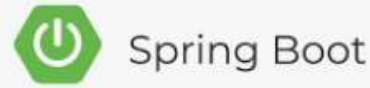
1. Set up Database Dev Environment
2. Create Spring Boot project using Spring Initializr
3. Get list of employees
4. Get single employee by ID
5. Add a new employee
6. Update an existing employee
7. Delete an existing employee

Let's build a DAO layer for this

Create DAO in Spring Boot



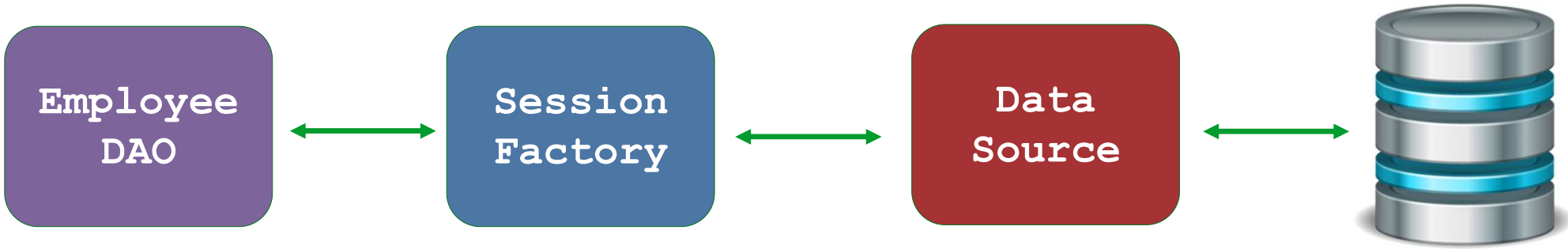
Create DAO in Spring Boot-Hibernate Session Factory



» In the past, our DAO used a Hibernate Session Factory

» Hibernate Session Factory needs a Data Source

- The data source defines database connection info



Traditional Spring

We normally had to do this configuration manually

XML

```
<!-- Step 1: Define Database DataSource / connection pool -->
<bean id="myDataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource"
      destroy-method="close">
        <property name="driverClass" value="com.microsoft.sqlserver.jdbc.SQLServerDriver" />
       <property name="jdbcUrl" value="jdbc:sqlserver://localhost:1433
              ;databaseName=web_customer_tracker" />
       <property name="user" value="sa" />
       <property name="password" value="sapassword" />
       <!-- these are connection pool properties for C3P0 -->
       <property name="minPoolSize" value="5" />
       <property name="maxPoolSize" value="20" />
       <property name="maxIdleTime" value="30000" />
 </bean>
<!-- Step 2: Setup Hibernate session factory -->
       <bean id="sessionFactory" class=
       "org.springframework.orm.hibernate5.LocalSessionFactoryBean">
       <property name="dataSource" ref="myDataSource" />
       <property name="packagesToScan" value="com.se.springdemo.entity" />
       <property name="hibernateProperties">
          <props>
             <prop key="hibernate.dialect">org.hibernate.dialect.SQLServer2012Dialect</prop>
              <prop key="hibernate.show_sql">true</prop>
          </props>
       </property>
</bean>
```

ALL JAVA

```
@Bean
public DataSource myDataSource() {
// create connection pool
ComboPooledDataSource myDataSource = new ComboPooledDataSource();
// set the jdbc driver
try {
        myDataSource.setDriverClass("com.microsoft.sqlserver.jdbc.SQLServerDriver");
}
catch (PropertyVetoException exc) {
        throw new RuntimeException(exc);
}
// for sanity's sake, let's log url and user ... just to make sure we are reading the data
logger.info("jdbc.url=" + env.getProperty("jdbc.url"));
logger.info("jdbc.user=" + env.getProperty("jdbc.user"));
// set database connection props
myDataSource.setJdbcUrl(env.getProperty("jdbc.url"));
myDataSource.setUser(env.getProperty("jdbc.user"));
myDataSource.setPassword(env.getProperty("jdbc.password"));
// set connection pool props
myDataSource.setInitialPoolSize(getIntProperty("connection.pool.initialPoolSize"));
myDataSource.setMinPoolSize(getIntProperty("connection.pool.minPoolSize"));
myDataSource.setMaxPoolSize(getIntProperty("connection.pool.maxPoolSize"));
myDataSource.setMaxIdleTime(getIntProperty("connection.pool.maxIdleTime"));
return myDataSource;
}
```

Spring Boot

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

application.properties

```
spring.datasource.url=jdbc:sqlserver://localhost:1433;databaseName=employee_directory
spring.datasource.username=sa
spring.datasource.password=sapassword
```

No need to give JDBC driver class name
Spring Boot will automatically detect it based on URL

Spring Boot will automatically create Beans for DataSource, EntityManager,...

Auto Data Source Configuration

» Based on configs, Spring Boot will automatically create the beans:

- `DataSource`, `EntityManager`, …
- You can then inject these into your app, for example your DAO
- `EntityManager` is from **Java Persistence API** (JPA)

» 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

» We can inject the `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

DAO Interface

File: EmployeeDAO.java

```
import java.util.List;
import com.se.springbootcruddemo.entity.Employee;
public interface EmployeeDAO {
        public List<Employee> findAll();
}
```

DAO Impl

File: EmployeeDAOHibernateImpl.java

```
@Repository
public class EmployeeDAOHibernateImpl implements EmployeeDAO {
    // define field for entitymanager
    private EntityManager entityManager;
    // set up constructor injection
    @Autowired
    public EmployeeDAOHibernateImpl(EntityManager theEntityManager) {
            entityManager = theEntityManager;}
    @Override
    @Transactional
    public List<Employee> findAll() {
        // get the current hibernate session
        Session currentSession = entityManager.unwrap(Session.class);
        // create a query - using native Hibernate API
        Query<Employee> theQuery =
            currentSession.createQuery("from Employee", Employee.class);
        // execute query and get result list
        List<Employee> employees = theQuery.getResultList();
        // return the results
        return employees;    }
}
```

Constructor Injection

Automatically created by Spring Boot

REST Controller

File: EmployeeRestController.java

```
@RestController
@RequestMapping("/api")
public class EmployeeRestController {
    private EmployeeDAO employeeDAO;

    @Autowired
    public EmployeeRestController(EmployeeDAO theEmployeeDAO) {
            employeeDAO = theEmployeeDAO;
    }
    // expose "/employees" and return list of employees
    @GetMapping("/employees")
    public List<Employee> findAll() {
            return employeeDAO.findAll();
    }
}
```

Test REST Controller

- Run **main** method in SpringbootCruddemoApplication.java class
- In web browser:

DAO: Find, Add, Update and Delete

File: EmployeeDAO.java

```
import java.util.List;
import com.se.springbootcruddemo.entity.Employee;
public interface EmployeeDAO {
        public List<Employee> findAll();

        public Employee findById(int theId);

        public void save(Employee theEmployee);

        public void deleteById(int theId);
}
```

DAO: Find, Add, Update and Delete

File: EmployeeDAO.java

```
import java.util.List;
import com.se.springbootcruddemo.entity.Employee;
public interface EmployeeDAO {
        public List<Employee> findAll();

        public Employee findById(int theId);

        public void save(Employee theEmployee);

        public void deleteById(int theId);
}
```

DAO Implement: Find, Add, Update and Delete

File: EmployeeDAOHibernateImpl.java

```
@Override
@Transactional
public Employee findById(int theId) {
    // get the current hibernate session
    Session currentSession = entityManager.unwrap(Session.class);
    // get the employee
    Employee theEmployee =
                    currentSession.get(Employee.class, theId);
    // return the employee
    return theEmployee;    }
```

DAO Implement: Find, Add, Update and Delete

File: EmployeeDAOHibernateImpl.java

```
@Override
@Transactional
public void save(Employee theEmployee) {
    // get the current hibernate session
    Session currentSession = entityManager.unwrap(Session.class);
    // save employee
    currentSession.saveOrUpdate(theEmployee);    }
```

DAO Implement: Find, Add, Update and Delete

File: EmployeeDAOHibernateImpl.java

```
@Override
@Transactional
public void deleteById(int theId) {
    // get the current hibernate session
    Session currentSession = entityManager.unwrap(Session.class);
    // delete object with primary key
    Query theQuery = currentSession.createQuery(
                    "delete from Employee where id=:employeeId");
    theQuery.setParameter("employeeId", theId);
    theQuery.executeUpdate();    }
```

REST Controller: Find, Add, Update and Delete

File: EmployeeRestController.java

```
// add mapping for GET /employees/{employeeId}
@GetMapping("/employees/{employeeId}")
public Employee getEmployee(@PathVariable int employeeId) {
    Employee theEmployee = employeeDAO.findById(employeeId);
    if (theEmployee == null) {
       throw new RuntimeException("Employee id not found - "
                + employeeId);   }
    return theEmployee; }
```

REST Controller: Find, Add, Update and Delete

File: EmployeeRestController.java

```
@PostMapping("/employees")
public Employee addEmployee(@RequestBody Employee theEmployee) {
    // also just in case they pass an id in JSON ... set id to 0
    // this is to force a save of new item ... instead of update
    theEmployee.setId(0);
    employeeDAO.save(theEmployee);
    return theEmployee; }
```

REST Controller: Find, Add, Update and Delete

File: EmployeeRestController.java

```
// add mapping for PUT /employees - update existing employee
@PutMapping("/employees")
public Employee updateEmployee(@RequestBody Employee theEmployee)
{
    employeeDAO.save(theEmployee);
    return theEmployee;
}
```

REST Controller: Find, Add, Update and Delete

File: EmployeeRestController.java

```
// add mapping for DELETE /employees/{employeeId} - delete employee
@DeleteMapping("/employees/{employeeId}")
public String deleteEmployee(@PathVariable int employeeId) {
    Employee tempEmployee = employeeDAO.findById(employeeId);
    // throw exception if null
    if (tempEmployee == null) {
            throw new RuntimeException("Employee id not found - "
                    + employeeId);
    }
    employeeDAO.deleteById(employeeId);
    return "Deleted employee id - " + employeeId;}
```

JPA DAO in Spring Boot

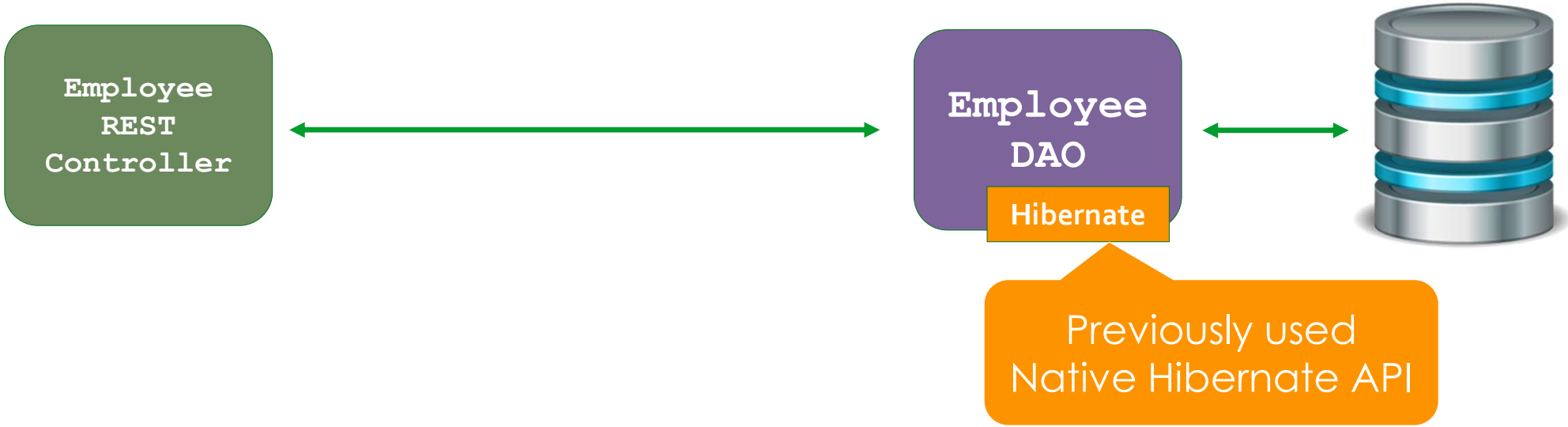
(Project: 21-jpa-crud-demo)

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

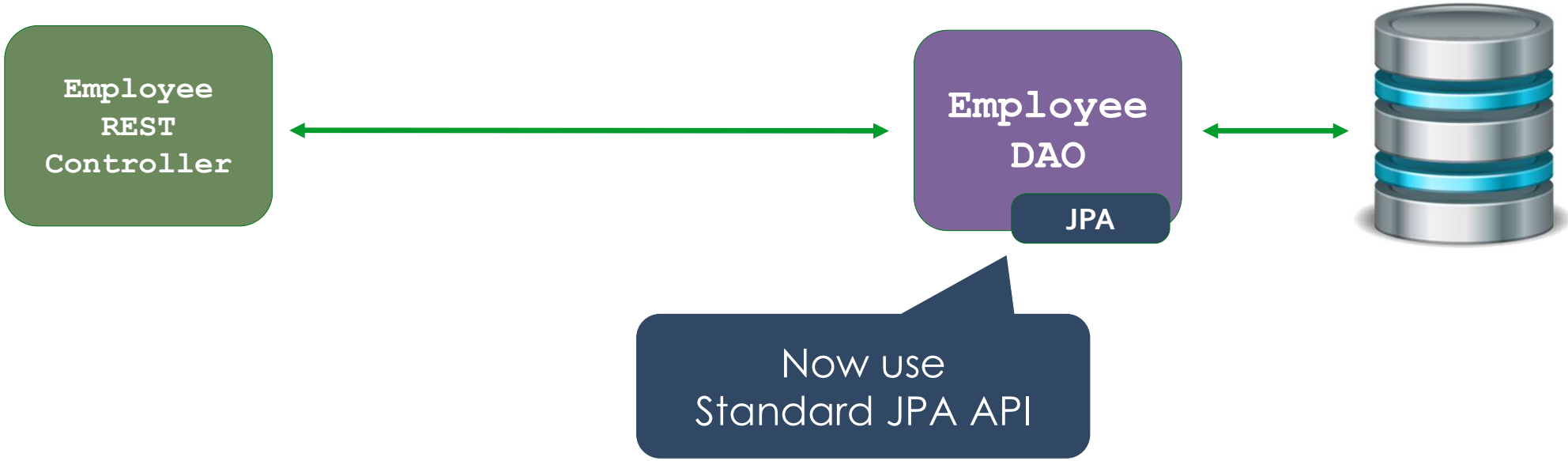
Application Architecture

Spring Boot



Application Architecture

Spring Boot



The Benefits of JPA

» By having a standard API, you are not locked to vendor's implementation

- Maintain portable, flexible code
- Can theoretically switch vendor implementations
- If Vendor ABC stops supporting their product
- 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

```
https://docs.oracle.com/javaee/7/tutorial/persistence-querylanguage.htm#BNBTG
```

Comparing JPA to Native Hibernate Methods

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

Create DAO in Spring Boot

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

Let's build a DAO layer for this

We just need to change the ADOImpl and Rest Controller

DAO Implement: List, Find, Add, Update and Delete

File: EmployeeDAOJpaImpl.java

```
@Repository
public class EmployeeDAOJpaImpl implements EmployeeDAO {
    // define field for entitymanager
    private EntityManager entityManager;
    // set up constructor injection
    @Autowired
    public EmployeeDAOJpaImpl(EntityManager theEntityManager) {
            entityManager = theEntityManager;}
    @Override
    @Transactional
    public List<Employee> findAll() {
        // get the current hibernate session
        // create a query
        Query theQuery =
                        entityManager.createQuery("from Employee");
        // execute query and get result list
        List<Employee> employees = theQuery.getResultList();
        // return the results
        return employees;}
```

DAO Implement: List, Find, Add, Update and Delete

File: EmployeeDAOJpaImpl.java

```
@Override
@Transactional
public Employee findById(int theId) {
    // get employee
    Employee theEmployee =
            entityManager.find(Employee.class, theId);
    // return employee
    return theEmployee;
}
```

DAO Implement: List, Find, Add, Update and Delete

File: EmployeeDAOJpaImpl.java

```
@Override
@Transactional
public void save(Employee theEmployee) {
    // save or update the employee
    Employee dbEmployee = entityManager.merge(theEmployee);
    // update with id from db ... so we can get generated id for save/insert
    theEmployee.setId(dbEmployee.getId());
}
```

DAO Implement: List, Find, Add, Update and Delete

File: EmployeeDAOJpaImpl.java

```
@Override
@Transactional
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(); }
```

REST Controller

File: EmployeeRestController.java

```
@Autowired
public EmployeeRestController(@Qualifier("employeeDAOJpaImpl")
        EmployeeDAO theEmployeeDAO) {employeeDAO = theEmployeeDAO;
}
```

Spring Data JPA in Spring Boot

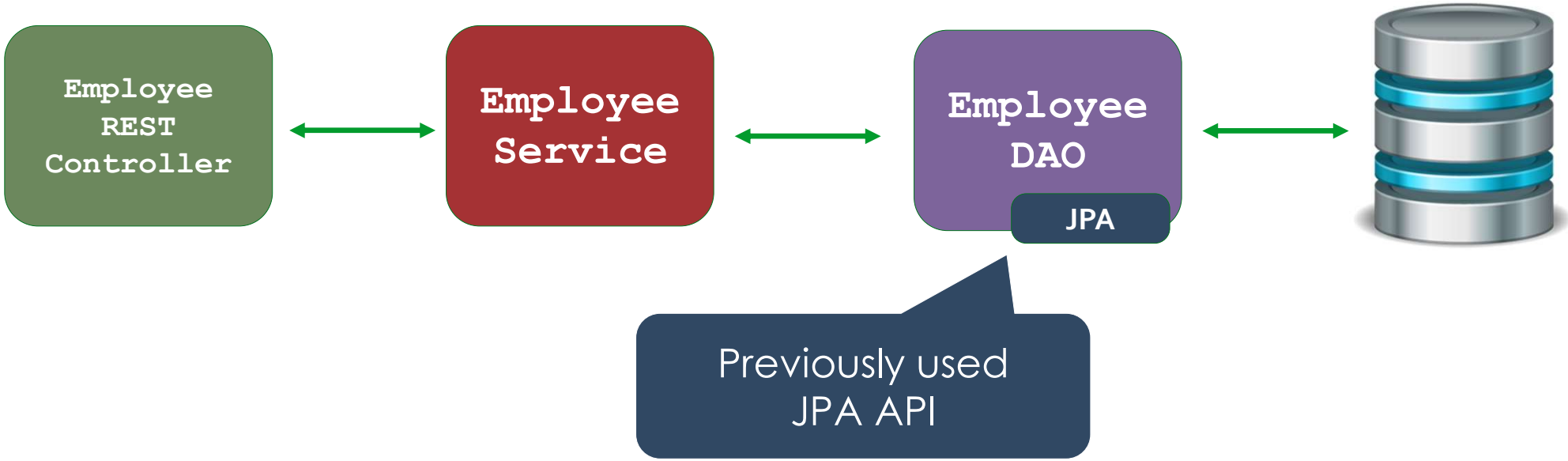
(Project: 22-spring-data-crud-demo)

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 

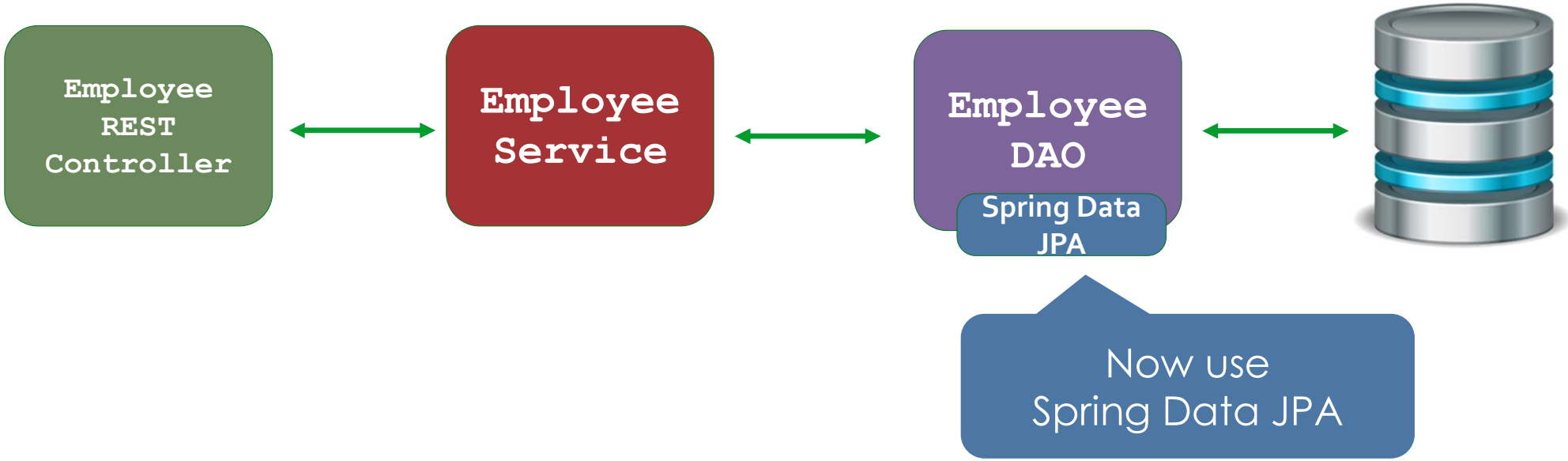
Application Architecture

Spring Boot

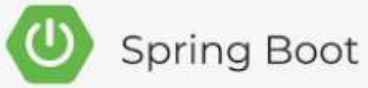


Application Architecture

Spring Boot



The Problem



- We saw how to create a DAO for Employee
- What if we need to **create a DAO** for **another entity**?
- Customer, Student, Product, Book …
- Do we have to **repeat all of the same code again**???

```java
import java.util.List;
import com.se.springbootcruddemo.entity.Employee;
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 EmployeeDAOHibernateImpl implements EmployeeDAO {
    // define field for entitymanager
    private EntityManager entityManager;
    // set up constructor injection
    @Autowired
    public EmployeeDAOHibernateImpl(EntityManager theEntityManager) {
            entityManager = theEntityManager;}
    @Override
    @Transactional
    public List<Employee> findAll() {
        // get the current hibernate session
        Session currentSession = entityManager.unwrap(Session.class);
        // create a query - using native Hibernate API
        Query<Employee> theQuery =
            currentSession.createQuery("from Employee", Employee.class);
        // execute query and get result list
        List<Employee> employees = theQuery.getResultList();
        // return the results
        return employees;    }

    @Override
    @Transactional
    public Employee findById(int theId) {
        // get the current hibernate session
        Session currentSession = entityManager.unwrap(Session.class);
        // get the employee
        Employee theEmployee =
                        currentSession.get(Employee.class, theId);
        // return the employee
        return theEmployee;   }
```

Creating DAO

Spring Boot

» You may have noticed a pattern with creating DAOs

```
@Override
@Transactional
public Employee findById(int theId) {
    // get employee
    Employee theEmployee = entityManager.find(Employee.class, theId);
    // return employee
    return theEmployee;  }
```

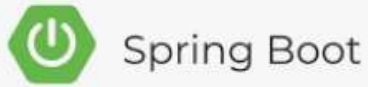
Most of the code is the same

Only difference is the entity type and primary key

Entity type

Primary key

We Wish



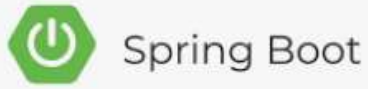
We could tell Spring:

Create a DAO for me

Plug in my entity type and primary key

Give me all of the basic CRUD features for free

Wish Diagram



Entity: Employee

Primary key: Integer

findAll()

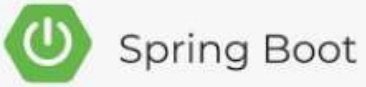
findById(...)

save(...)

deleteById(...)

... others ...

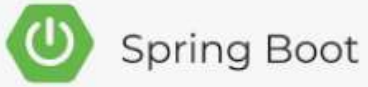
Spring Data JPA - Solution



- Spring Data JPA is the solution!!!!
- Create a DAO and just plug in your entity type and primary key
- Spring will give you a CRUD implementation for FREE …
- Helps to minimize boiler-plate DAO code …

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

JpaRepository



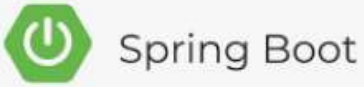
- Spring Data JPA provides the interface: JpaRepository
- Exposes methods (some by inheritance from parents)

findAll()
findById(…)
save(…)
deleteById(…)
… others …

- JpaRepository Docs

https://docs.spring.io/spring-data/jpa/docs/current/api/org/springframework/data/jpa/repository/JpaRepository.html

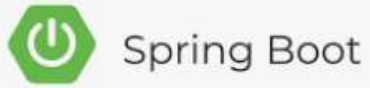
Development Process



- Extend JpaRepository interface
- Use your Repository in your app

No need for implementation class

Step 1: Extend JpaRepository interface



File: EmployeeRepository.java

```
public interface EmployeeRepository extends JpaRepository<Employee, Integer>
{
// that's it ... no need to write any code!
}
```

Entity: Employee

Primary key: Integer

findAll()

findById(...)

save(...)

deleteById(...)

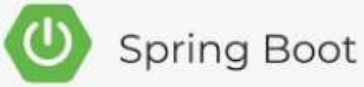
... others ...

Entity type

Primary key

Get these methods for free

Step 2: Use Repository in your app

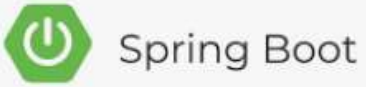


File: EmployeeServiceImpl.java

```
@Service
public class EmployeeServiceImpl implements EmployeeService {
    private EmployeeRepository employeeRepository;

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

Step 2: Use Repository in your app



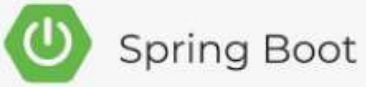
File: EmployeeServiceImpl.java

```
@Service
public class EmployeeServiceImpl implements EmployeeService {
    private EmployeeRepository employeeRepository;

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

Our repository

Step 2: Use Repository in your app

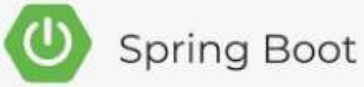


File: EmployeeServiceImpl.java

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

```
@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; }
```

Step 2: Use Repository in your app



File: EmployeeServiceImpl.java

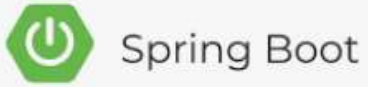
```
@Override
public void save(Employee theEmployee) {
        employeeRepository.save(theEmployee);
}
```

```
@Override
public void deleteById(int theId) {
        employeeRepository.deleteById(theId);
}
```

Spring Data REST in Spring Boot

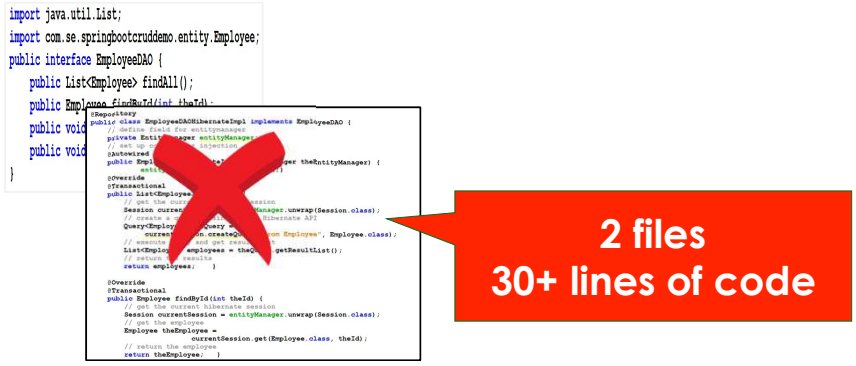
(Project: 23-spring-data-rest-crud-demo)

The Problem



- Earlier, we saw the magic of **Spring Data JPA**
- This helped to eliminate boilerplate code

Before Spring Data JPA



2 files
30+ lines of code

After Spring Data JPA

```
public interface EmployeeRepository extends JpaRepository<Employee, Integer>
{
// that's it ... no need to write any code!
}
```

1file
3 lines of code

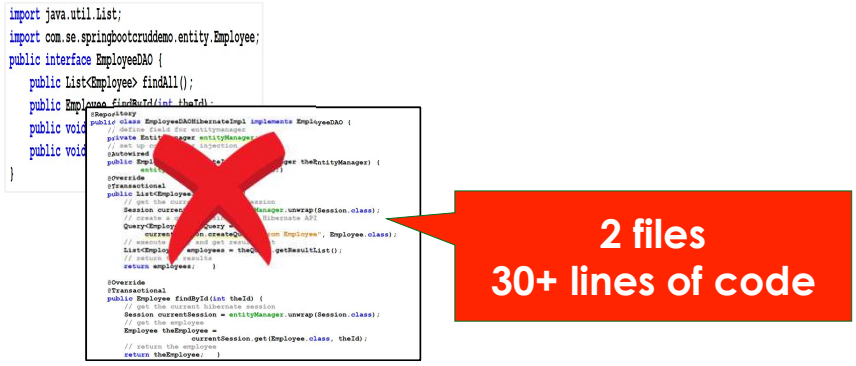
No need for implement class

The Problem

Spring Boot

Can this apply to REST APIs?

Before Spring Data JPA



2 files
30+ lines of code

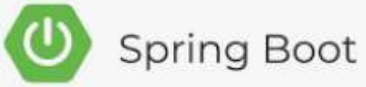
After Spring Data JPA

```
public interface EmployeeRepository extends JpaRepository<Employee, Integer>
{
// that's it ... no need to write any code!
}
```

1file
3 lines of code

No need for implement class

The Problem



» We saw how to create a REST API for Employee

Interface

```
public interface EmployeeService {
    public List<Employee> findAll();
    public Employee findById(int theId);
    public void save(Employee theEmployee);
    public void deleteById(int theId); }
```

REST Controller

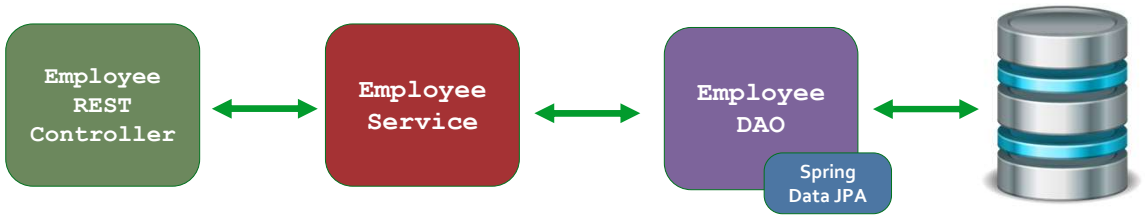
```
@RestController
@RequestMapping("/api")
public class EmployeeRestController {
    private EmployeeService employeeService;
    @Autowired
    public EmployeeRestController(EmployeeService theEmployeeService) {...3 lines }
    // expose "/employees" and return list of employees
    @GetMapping("/employees")
    public List<Employee> findAll() {...3 lines }
    // add mapping for GET /employees/{employeeId}
    @GetMapping("/employees/{employeeId}")
    public Employee getEmployee(@PathVariable int employeeId) {...6 lines }
    // add mapping for POST /employees - add new employee
    @PostMapping("/employees")
    public Employee addEmployee(@RequestBody Employee theEmployee) {...6 lines }
    // add mapping for PUT /employees - update existing employee
    @PutMapping("/employees")
    public Employee updateEmployee(@RequestBody Employee theEmployee) {...6 lines }
    // add mapping for DELETE /employees/{employeeId} - delete employee
    @DeleteMapping("/employees/{employeeId}")
    public String deleteEmployee(@PathVariable int employeeId) {...14 lines }
}
```

```
@Service
public class EmployeeServiceImpl implements EmployeeService {
    private EmployeeRepository employeeRepository;
    @Autowired
    public EmployeeServiceImpl(EmployeeRepository theEmployeeRepository) {
    @Override
    public List<Employee> findAll() {...2 lines }
    @Override
    public Employee findById(int theId) {...9 lines }
    @Override
    public void save(Employee theEmployee) {...3 lines }
    @Override
    public void deleteById(int theId) {...3 lines }
}
```

Service (implement of interface)

The Problem

» We saw how to create a REST API for Employee



Interface

```
public interface EmployeeService {
    public List<Employee> findAll();
    public Employee findById(int theId);
    public void save(Employee theEmployee);
```

Service (implement of interface

```
@Service
public class EmployeeServiceIm
    private EmployeeRepository
    @Autowired
    public EmployeeServiceImpl(EmployeeRepository theEmployeeRepository) {...2 lines }
```

REST Controller

```
@RestController
@RequestMapping("/api")
public class EmployeeRestController {
    private EmployeeService employeeService;
    @Autowired
    public EmployeeRestController(EmployeeService theEmployeeService) {...3 lines }
    // expose "/employees" and return list of employees
    @GetMapping("/employees")
    public List<Employee> findAll() {...3 lines }
    // add mapping for GET /employees/{employeeId}
    @GetMapping("/employees/{employeeId}")
    public Employee getEmployee(@PathVariable int employeeId) {...6 lines }
    // add mapping for POST /employees - add new employee
    @PostMapping("/employees")
    public Employee addEmployee(@RequestBody Employee theEmployee) {...6 lines }
    // add mapping for PUT /employees - update existing employee
    @PutMapping("/employees")
    public Employee updateEmployee(@RequestBody Employee theEmployee) {...6 lines }
    // add mapping for DELETE /employees/{employeeId} - delete employee
    @DeleteMapping("/employees/{employeeId}")
    public String deleteEmployee(@PathVariable int employeeId) {...14 lines }
}
```

The Problem

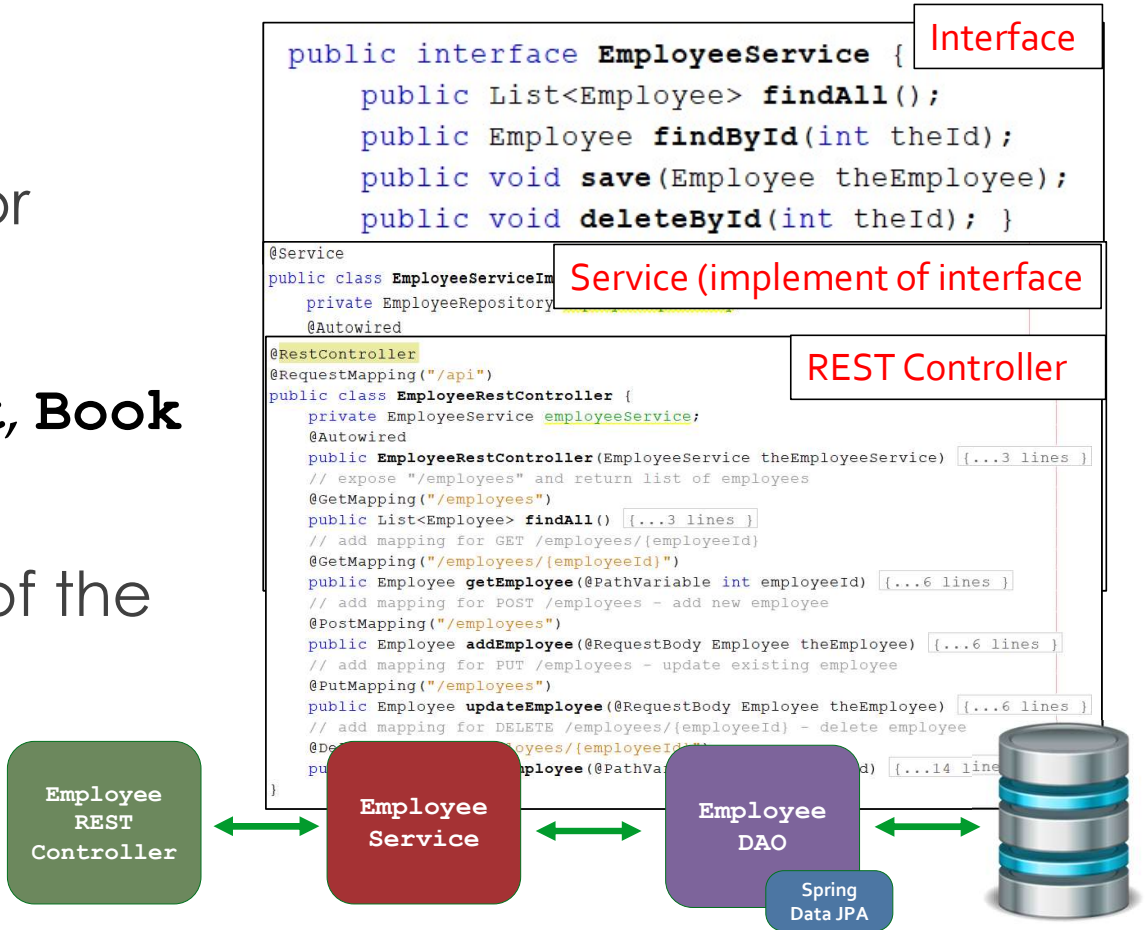
Spring Boot

- We saw how to create a REST API for Employee
- Need to create REST API for another entity?

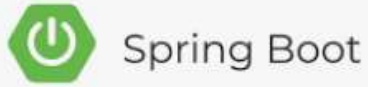
`Customer`, `Student`, `Product`, `Book`

...

- Do we have to repeat all of the same code again???

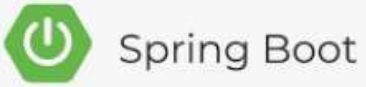


Spring Data REST - Solution



- Leverages your existing **`JpaRepository`**
- Spring will give you a REST CRUD implementation for FREE
- Helps to minimize boiler-plate REST code!!!
- No new coding required!!!

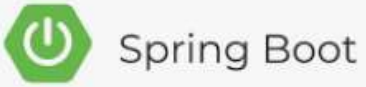
REST API



» 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

Spring Data REST - How Does It Work?



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

```
public interface EmployeeRepository extends JpaRepository<Employee, Integer>
{
// that's it ... no need to write any code!
}
```

REST Endpoints

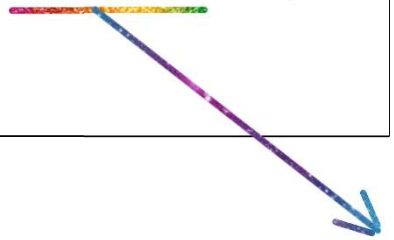
Spring Boot

» By default, Spring Data REST will create endpoints based on entity type

» **Simple** pluralized form

- First character of Entity type is **lowercase**
- Then just adds an **"s"** to the entity

```
public interface EmployeeRepository extends JpaRepository<Employee, Integer>
{
// that's it ... no need to write any code!
}
```



/employee

Development Process

Add Spring Data REST to your Maven POM file

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

Absolutely NO CODING required

Development Process

Add Spring Data REST to your Maven POM file

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

Absolutely NO CODING required

Spring Data REST will scan for JpaRepository

Development Process

Add Spring Data REST to your Maven POM file

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

Absolutely NO **CODING** required

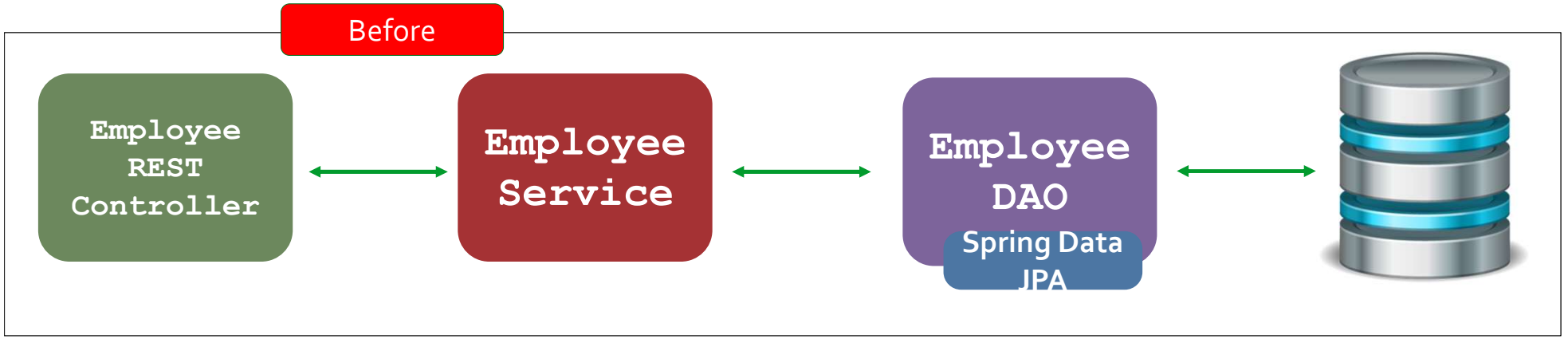
Spring Data REST will scan for JpaRepository

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

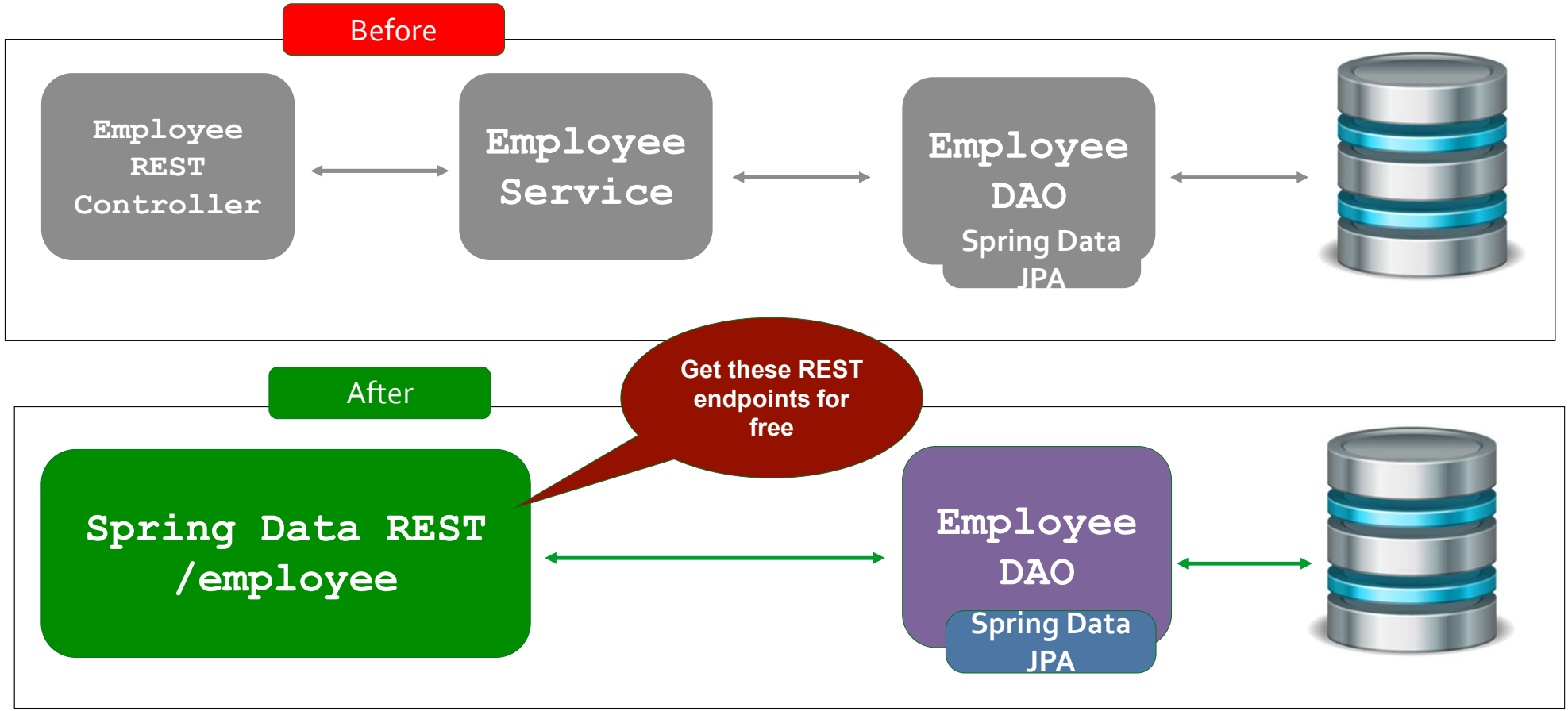
Get these REST endpoints for free

Application Architecture

Spring Boot

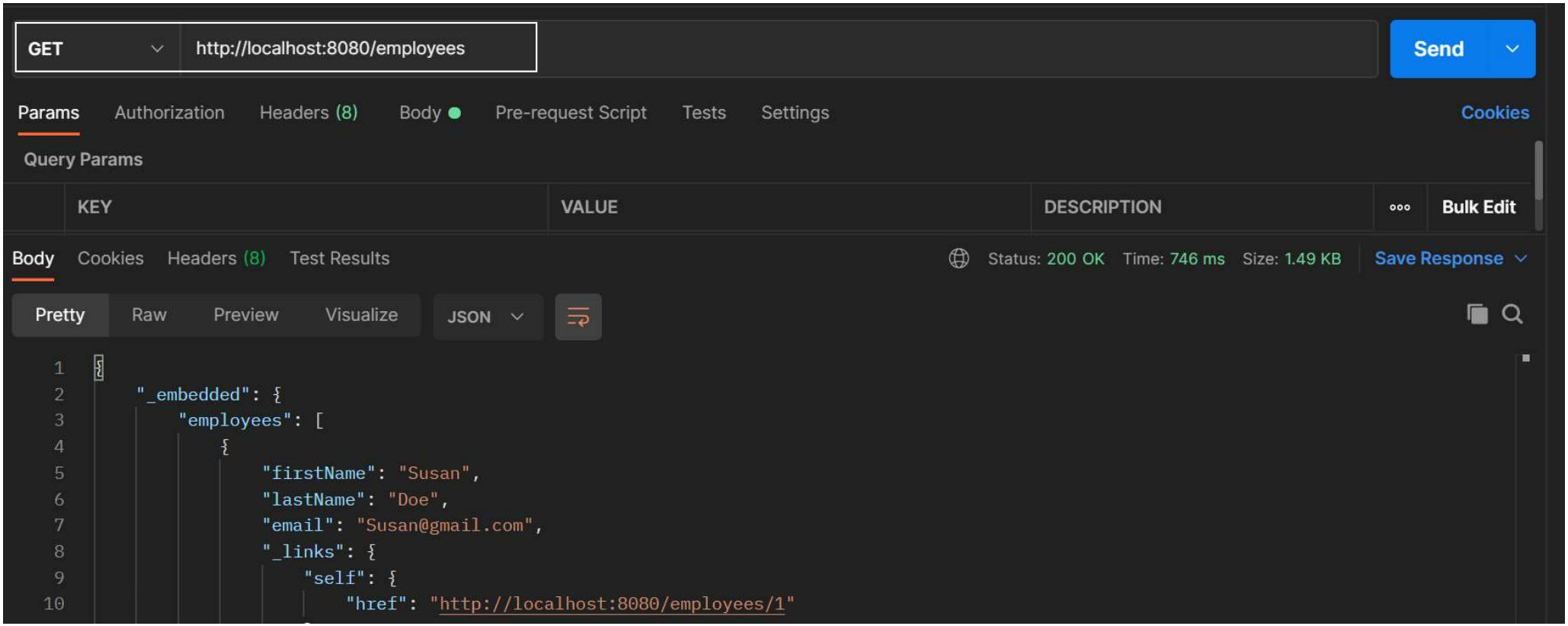


Application Architecture



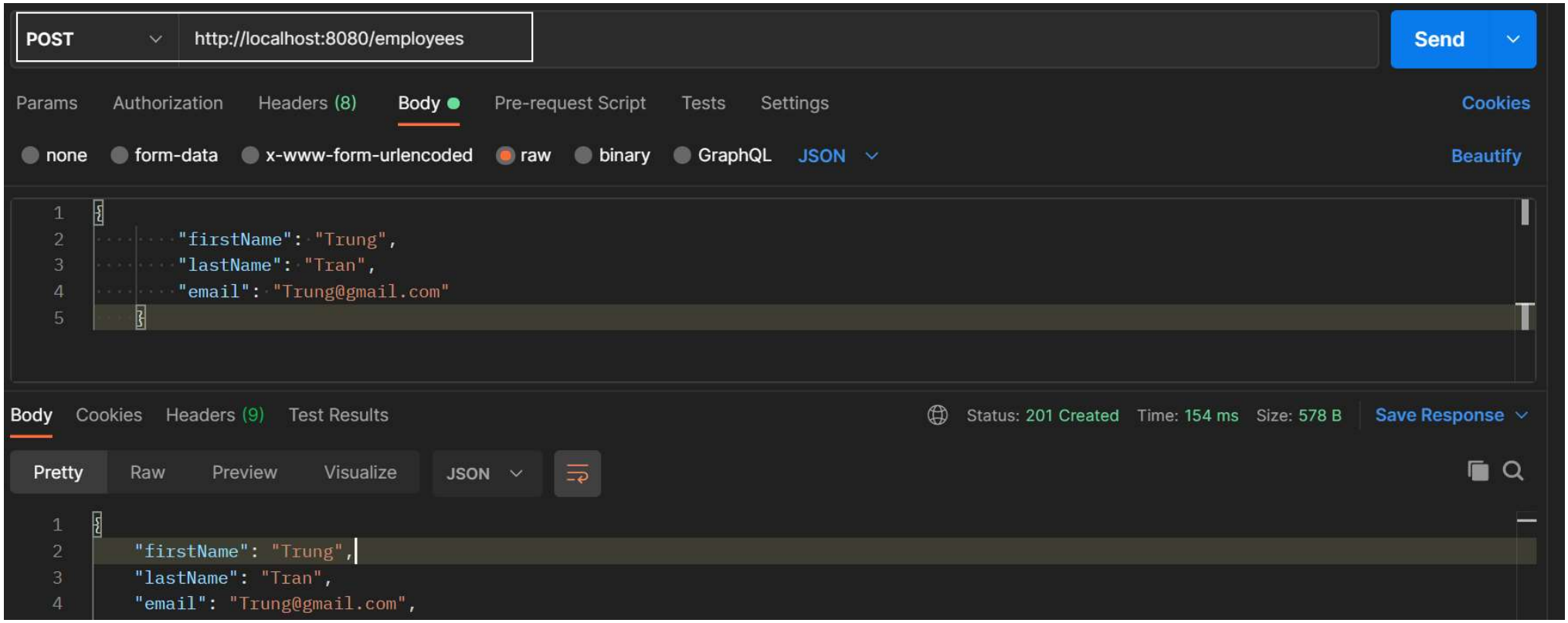
Test on Postman

Spring Boot



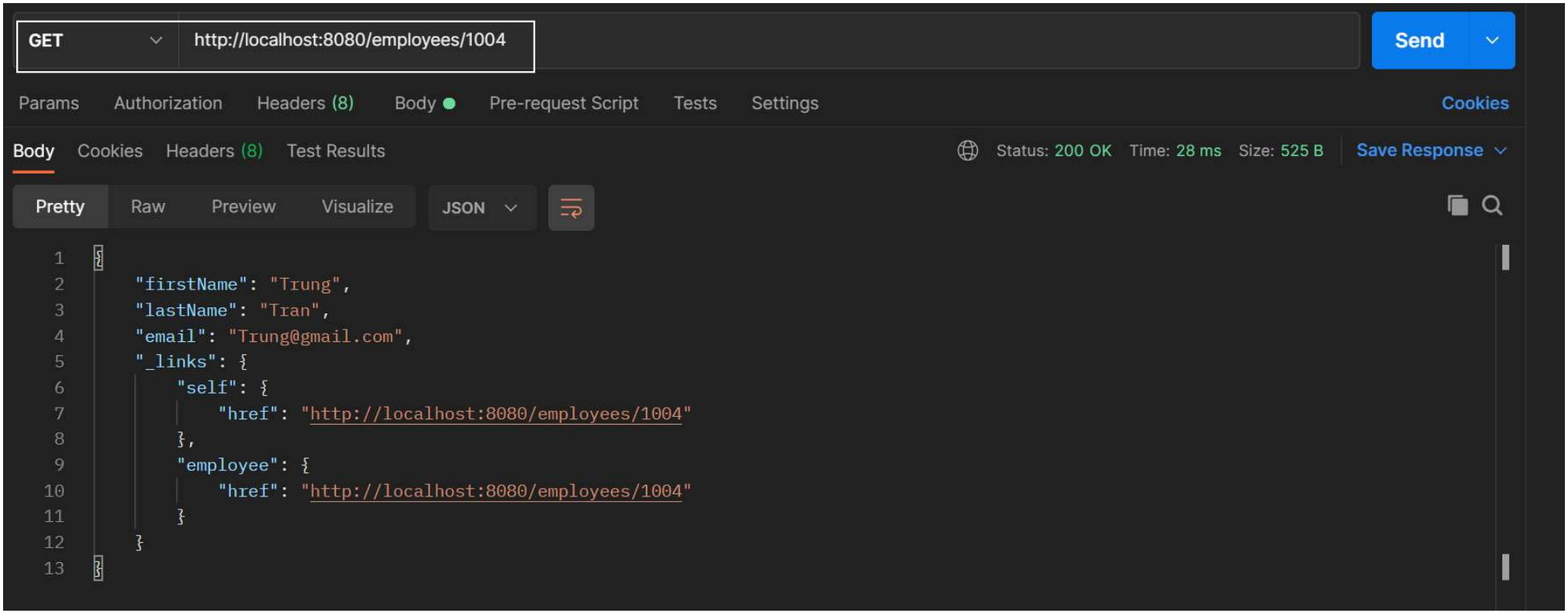
Test on Postman

Spring Boot



Test on Postman

Spring Boot



Spring Data REST Configuration, Pagination and Sorting

(Project: 24-spring-data-rest-crud-config-paging-sorting-demo)

REST Endpoints

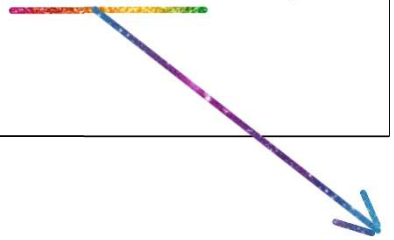
Spring Boot

» By default, Spring Data REST will create endpoints based on entity type

» **Simple** pluralized form

- First character of Entity type is **lowercase**
- Then just adds an **"s"** to the entity

```
public interface EmployeeRepository extends JpaRepository<Employee, Integer>
{
// that's it ... no need to write any code!
}
```



/employee

Pluralized Form

- Spring Data REST pluralized form is VERY simple
- Just adds an "s" to the entity
- The English language is VERY complex! → Spring Data REST does NOT handle

Singular	Plural
Goose	Geese
Person	People
Syllabus	Syllabi
...	...

Solution

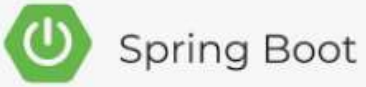
Spring Boot

» Specify plural name / path with an annotation

```
@RepositoryRestResource(path="members")
public interface EmployeeRepository extends JpaRepository<Employee, Integer>
{
// that's it ... no need to write any code!
}
```

http://localhost:8080/members

Pagination



» By default, Spring Data REST will return the first 20 elements

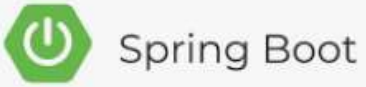
- **Page size = 20**

» You can navigate to the different pages of data using query param

```
http://localhost:8080/members?page=0
http://localhost:8080/members?page=1
```

Pages are zero-based

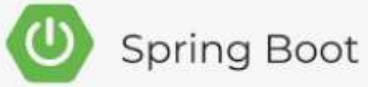
Spring Data REST Configuration



» Following properties available: application.properties

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

Sample Configuration

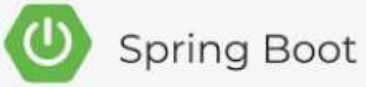


http://localhost:8080/magic-api/members?page=0

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

spring.data.rest.default-page-size=5
```

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/members?sort=lastName`
- Sort by first name, descending `http://localhost:8080/member?sort=firstName,desc`
- Sort by last name, then first name, ascending, page 1

`http://localhost:8080/members?sort=lastName,firstName,asc&page=1`

Thymeleaf with Spring Boot

(Project: 30-thymeleafdemo-helloworld)

What is Thymeleaf?

» Thymeleaf is a Java templating engine

www.thymeleaf.org

» Commonly used to generate the HTML views for web apps

» However, it is a general purpose templating engine

What is Thymeleaf?

» Thymeleaf is a Java templating engine

» Commonly used to generate the HTML vie

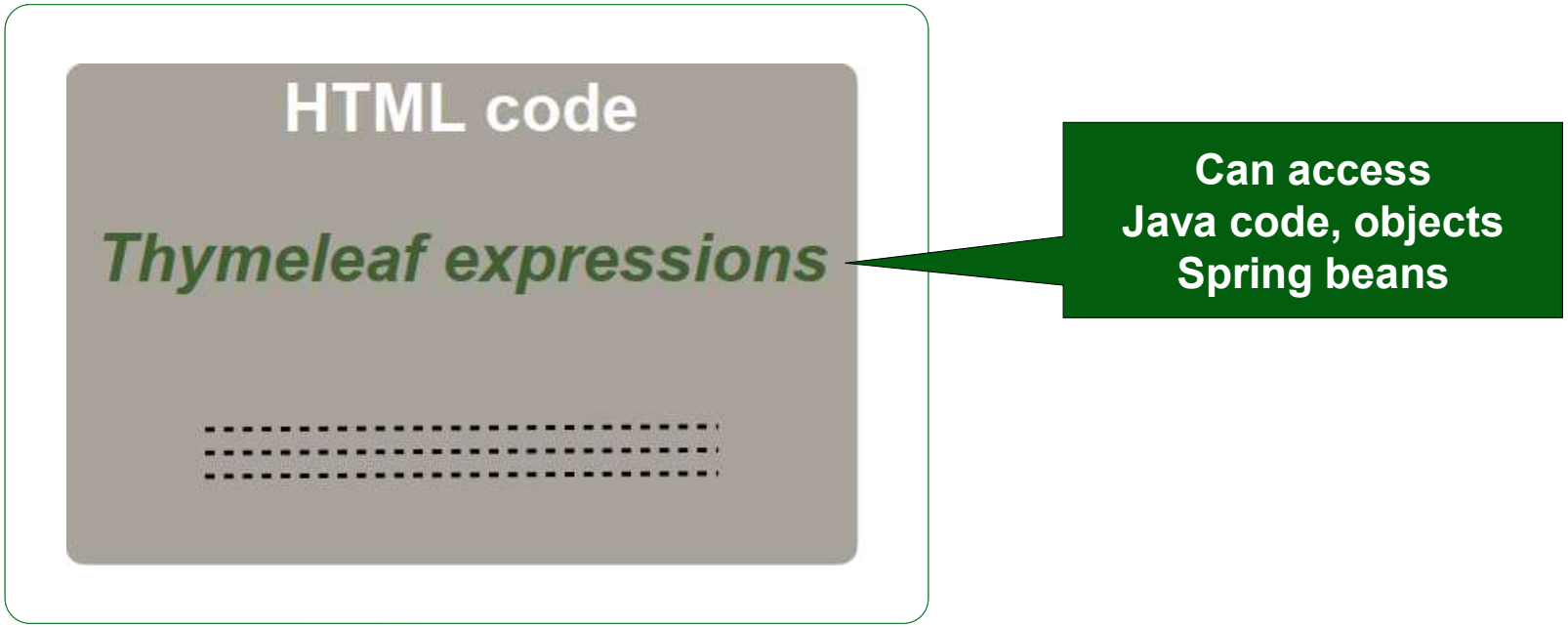
» However, it is a general purpose templating engine

www.thymeleaf.org

Separate project
Unrelated to spring.io

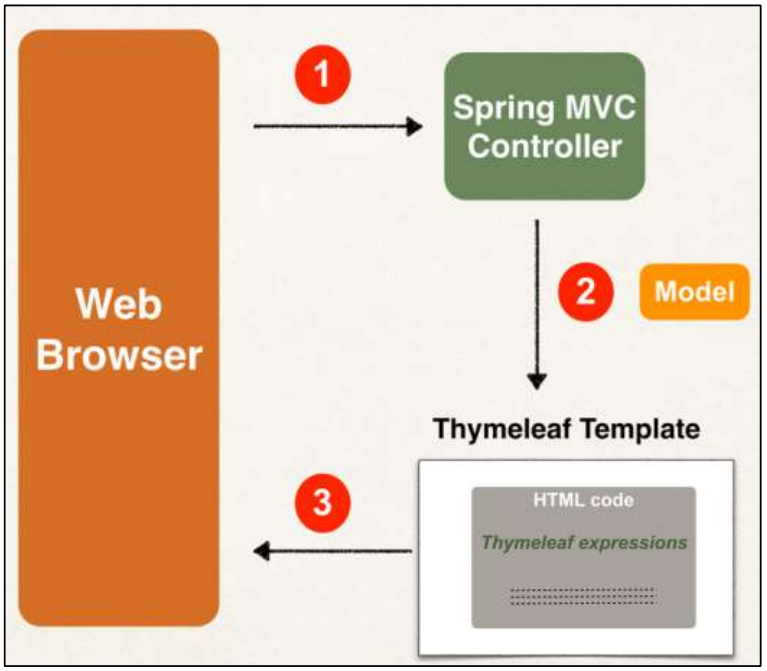
What is a Thymeleaf template?

- Can be an HTML page with some Thymeleaf expressions
- Include dynamic content from Thymeleaf expressions



Where is the Thymeleaf template processed?

- In a web app, Thymeleaf is processed on the server
- Results included in HTML returned to browser



Thymeleaf vs JSP

- Thymeleaf is similar to JSP
- Can be used for web view templates
- One key difference
  - JSP can only be used in a web environment
  - Thymeleaf can be used in web OR non-web environments

FAQ: Should I use JSP or Thymeleaf?

- Depends on your **project requirements**
- If you only need **web views** you can go **either way**
- If you need a **general purpose** template engine (non-web) use **Thymeleaf**

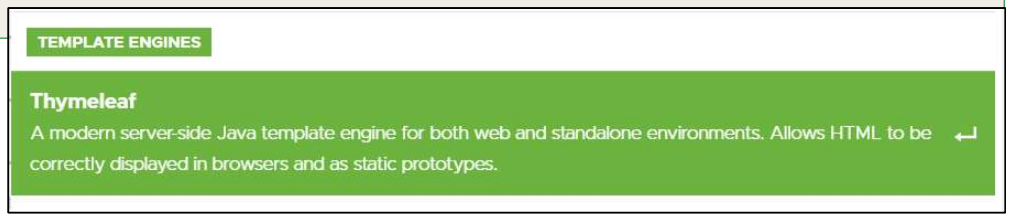
Demo – Development Process

- Add Thymeleaf to Maven POM file
- Develop Spring MVC Controller
- Create Thymeleaf template

Step 1: Add Thymeleaf to Maven pom file


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

Based on this, Spring Boot will auto configure to use **Thymeleaf templates**



Step 2: Develop Spring MVC Controller

File: DemoController.java

```
@Controller
public class DemoController {
    @GetMapping("/")
    public String sayHello(Model theModel) {
       theModel.addAttribute("theDate", new java.util.Date());
    return "helloworld"; }
}
```

src/main/resources/templates/helloworld.html

For web apps, Thymeleaf templates have a .html extension

Step 3: Create Thymeleaf template

File: src/main/resources/templates/helloworld.html

```
<!DOCTYPE HTML>
<html xmlns:th="http://www.thymeleaf.org">
<head> … </head>
<body>
      <p th:text="'Time on the server is ' + ${theDate}" />
</body>
</html>
```

To use Thymeleaf expressions

Thymeleaf expressions

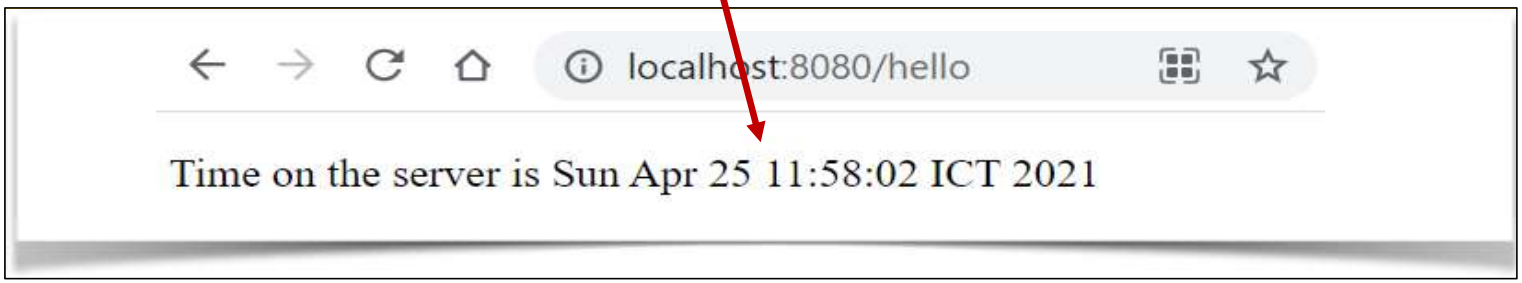
Step 3: Create Thymeleaf template

```
@Controller
public class DemoController {
    @GetMapping("/")
    public String sayHello(Model theModel) {
        theModel.addAttribute("theDate", new
java.util.Date());
    return "helloworld"; }
}
```

1

```
<!DOCTYPE HTML>
<html xmlns:th="http://www.thymeleaf.
<head> … </head>
<body>
      <p th:text="'Time on the server is ' + ${theDate}"
/>
</body>
</html>
```

2



Additional Features

- Looping and conditionals
- CSS and JavaScript integration
- Template layouts and fragments

CSS and Thymeleaf

(Project: 31-thymeleafdemo-helloworld-css)

Apply CSS Styles to our Page

» We have the option of using

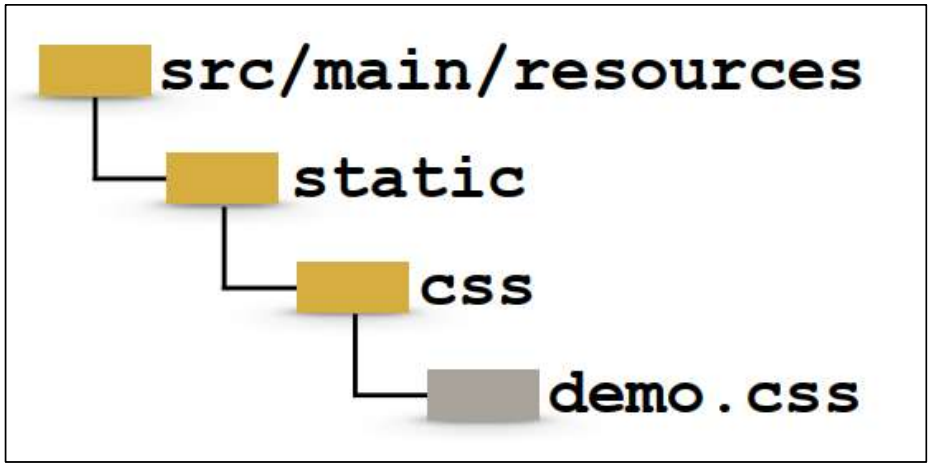
- Local CSS files as part of your project
- Referencing remote CSS files

Development Process

- Create CSS file
- Reference CSS in Thymeleaf template
- Apply CSS style

Step 1: Create CSS file

» Spring Boot will look for static resources in the directory **`src/main/resources/static`**

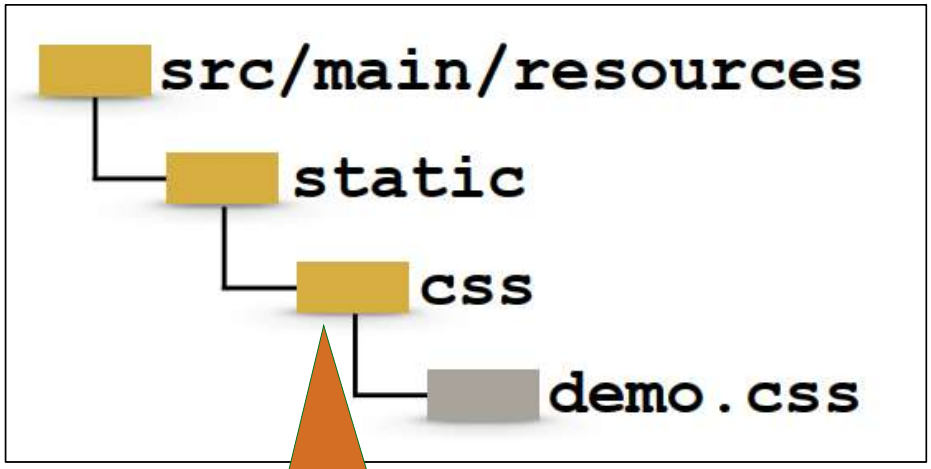


File: demo.css

```
.funny {
  font-style: italic;
  color: green;
}
```

Step 1: Create CSS file

» Spring Boot will look for **static resources** in the directory
`src/main/resources/static`



Can be any sub-directory name

You can create your own custom sub-directory
`static/css`
`static/images`
`static/js` ..

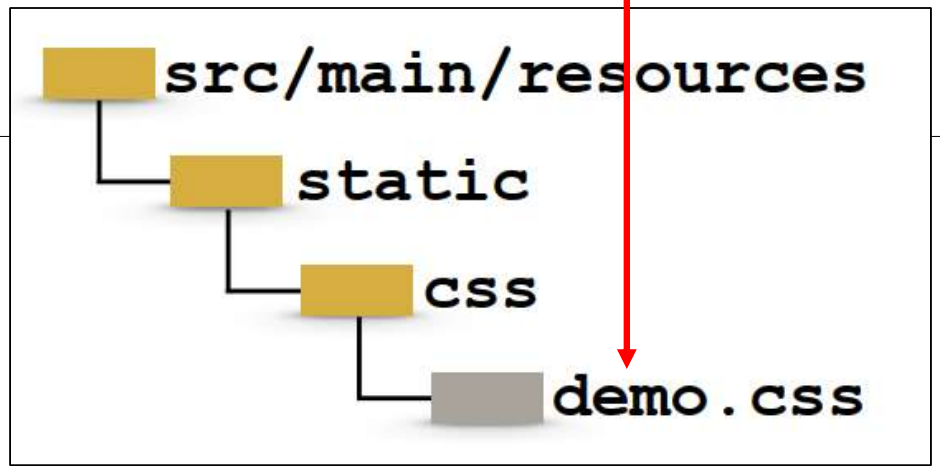
```
.funny {
  font-style: italic;
  color: green;
}
```

Step 2: Reference CSS in Thymeleaf template

File: helloworld.html

```
<head>
<title>Thymeleaf Demo</title>
  <!-- reference CSS file -->
  <link rel="stylesheet" th:href="@{/css/demo.css}" />
</head>
```

@ symbol
Reference context path of your application (app root)



Step 3: Apply CSS

File: demo.css

```
.funny {
   font-style: italic;
   color: green;
}
```

File: helloworld.html

```
<head>
   <title>Thymeleaf Demo</title>
   <!-- reference CSS file -->
   <link rel="stylesheet" th:href="@{/css/demo.css}" />
</head>
<body>
   <p th:text="'Time on the server is ' + ${theDate}"
    class="funny" />
</body>
```

Other search directories

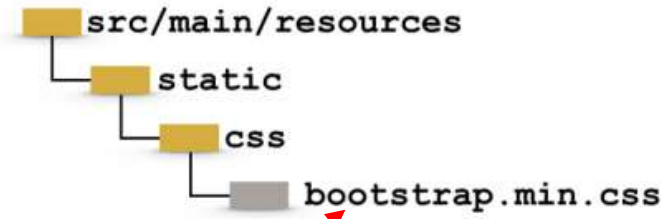
Spring Boot will search following directories for static resources:

```
/src/main/resource
/META-INF/resources
/resources
/static
/public
```

3rd Party CSS Libraries - Bootstrap

» Local Installation

Download Bootstrap file(s) and add to **`/static/css`** directory




```
<head>
… …
  <!-- reference CSS file -->
  <link rel="stylesheet" th:href="@{/css/bootstrap.min.css}" />
</head>
```

3rd Party CSS Libraries - Bootstrap

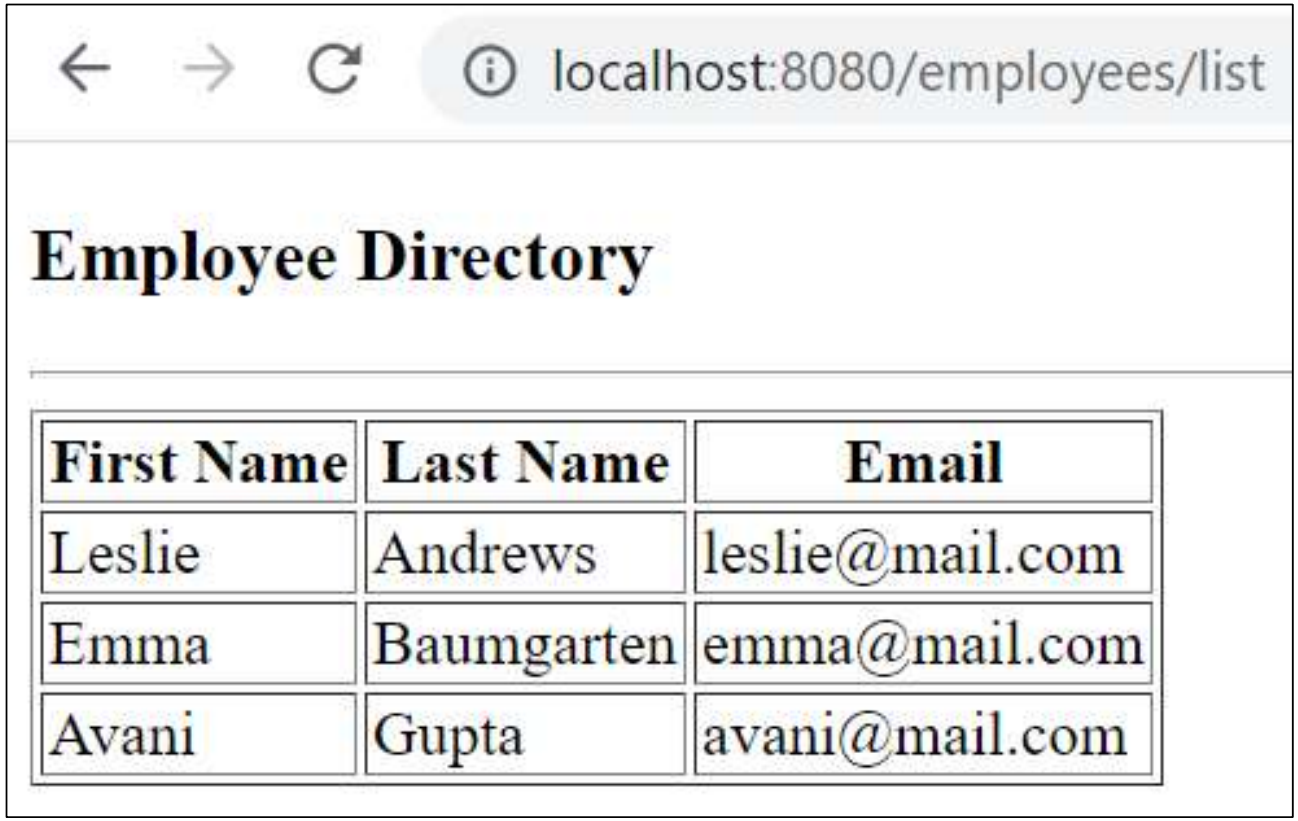
» Remote Files

```
<head>
… …
<!-- reference CSS file -->
<link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.2.1/css/bootstrap.min.css" />
… …
</head>
```

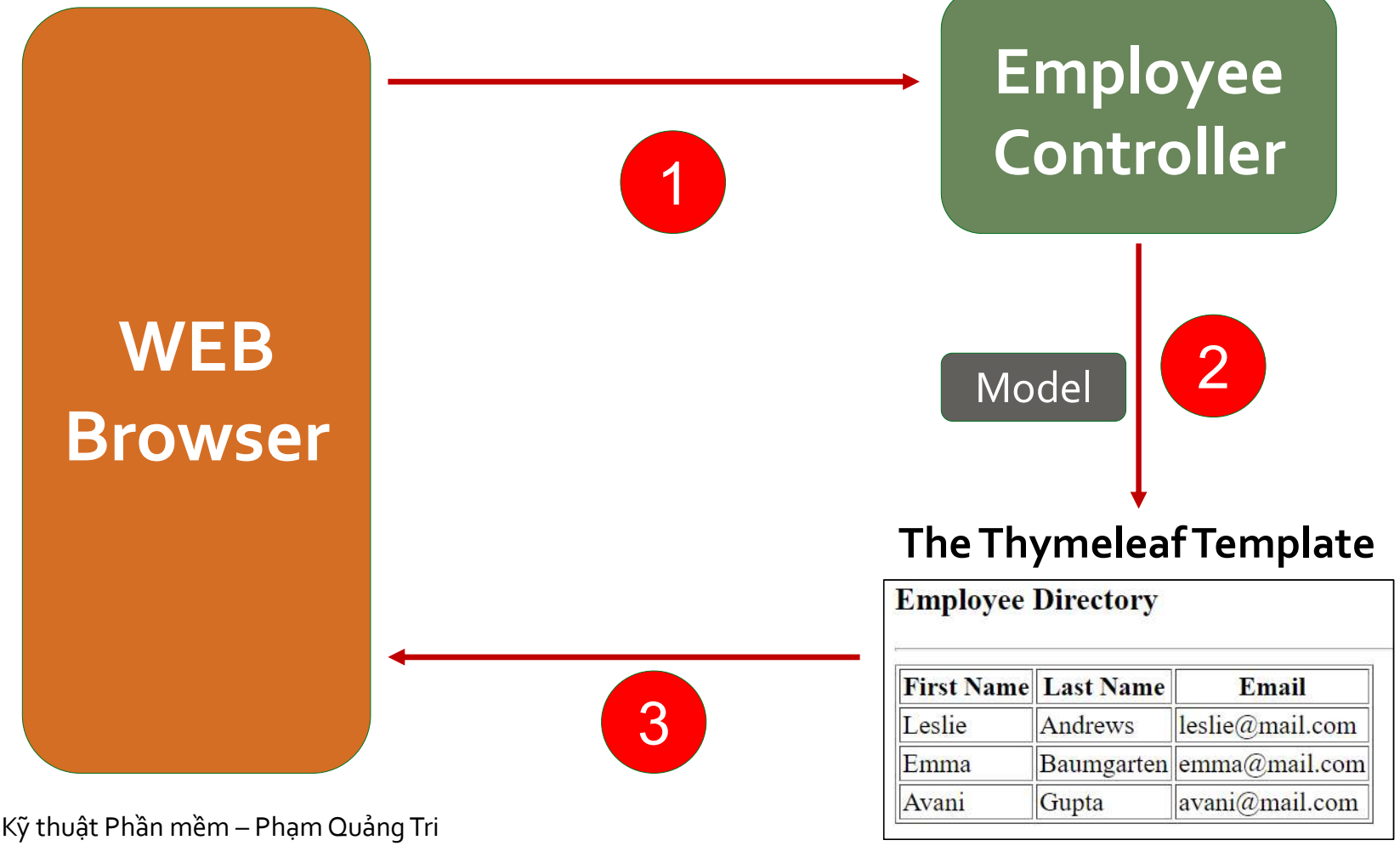
Create HTML Tables with Thymeleaf

(Project: 32-thymeleafdemo-employee-list)

HTML Tables


localhost:8080/employees/list
Employee Directory
First Name
Last Name
Email
Leslie
Andrews
leslie@mail.com
Emma
Baumgarten
emma@mail.com
Avani
Gupta
avani@mail.com

Application processing flow



Development Process

- Create Employee.java
- Create EmployeeController.java
- Create Thymeleaf template

Step 1: Create Employee.java


```
public class Employee {
    private int id;
    private String firstName;
    private String lastName;
    private String email;
    public Employee() {...3 lines }
    public Employee(int id, String firstName, String lastName, String email)
    public int getId() {...3 lines }
    public void setId(int id) {...2 lines }
    public String getFirstName() {...2 lines }
    public void setFirstName(String firstName) {...2 lines }
    public String getLastName() {...3 lines }
    public void setLastName(String lastName) {...2 lines }
    public String getEmail() {...2 lines }
    public void setEmail(String email) {...3 lines }
    @Override
    public String toString() {...3 lines }
```

Step 2: Create EmployeeController.java (1)

```
@Controller
@RequestMapping("/employees")
public class EmployeeController {
    private List<Employee> theEmployees;
    /*The PostConstruct annotation is used on a method that needs to be
    executed after dependency injection is done to perform any
    initialization*/
    @PostConstruct
    private void loadData() {
    // create employees
    Employee emp1 = new Employee(1, "Leslie", "Andrews", "leslie@mail.com");
    Employee emp2 = new Employee(2, "Emma", "Baumgarten", "emma@mail.com");
    Employee emp3 = new Employee(3, "Avani", "Gupta", "avani@mail.com");
    theEmployees = new ArrayList<>();
    // add to the list
    theEmployees.add(emp1);
    theEmployees.add(emp2);
    theEmployees.add(emp3);}
```

Step 2: Create EmployeeController.java (2)


```
@GetMapping("/list")
public String listEmployees(Model theModel) {
        // add to the spring model
        theModel.addAttribute("employees", theEmployees);
        return "list-employees";
}
```

The Thymeleaf template will access this data

src/main/resources/templates/list-employees.html

Step 3: Create Thymeleaf template (1)

File: list-employees.html

```
<!DOCTYPE HTML>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
...
<body>
<h3>Employee Directory</h3>
<hr>
<table border="1">
<!-- Build HTML table based on employees -->
</table>
</body>
</html>
```

Employee Directory

First Name	Last Name	Email
Leslie	Andrews	leslie@mail.com
Emma	Baumgarten	emma@mail.com
Avani	Gupta	avani@mail.com

Step 3: Create Thymeleaf template (2)

File: list-employees.html

```
<table border="1">
<thead>
<tr>
<th>First Name</th>
<th>Last Name</th>
<th>Email</th>
</tr>
</thead>
<tbody>
<tr th:each="tempEmployee : ${employees}">
<td th:text="${tempEmployee.firstName}" />
<td th:text="${tempEmployee.lastName}" />
<td th:text="${tempEmployee.email}" />
</tr>
</tbody>
</table>
```

```
@GetMapping("/list")
public String listEmployees(Model theModel) {
    // add to the spring model
    theModel.addAttribute("employees", theEmployees);
    return "list-employees";
}
```

Employee Directory

First Name	Last Name	Email
Leslie	Andrews	leslie@mail.com
Emma	Baumgarten	emma@mail.com
Avani	Gupta	avani@mail.com

Thymeleaf and Bootstrap

(Project: 33-thymeleafdemo-employees-list-css)

Make our Page Beautiful

Before

Employee Directory

First Name	Last Name	Email
Leslie	Andrews	leslie@mail.com
Emma	Baumgarten	emma@mail.com
Avani	Gupta	avani@mail.com

After

Employee Directory

First Name	Last Name	Email
Leslie	Andrews	leslie@mail.com
Emma	Baumgarten	emma@mail.com
Avani	Gupta	avani@mail.com

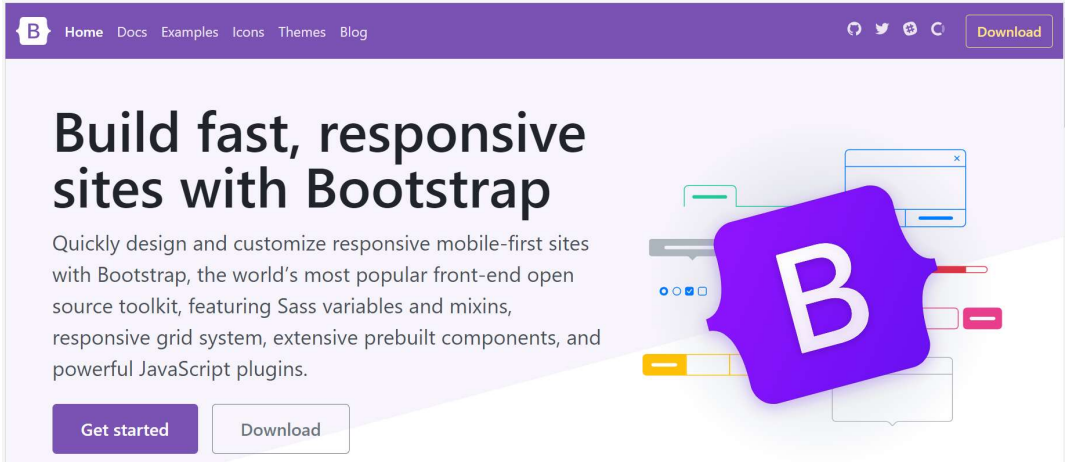
Bootstrap

Development Process

- Get links for remote Bootstrap file
- Add **links** in Thymeleaf template
- Apply **Bootstrap CSS** styles

Step 1: Get links for remote Bootstrap files

- Visit Bootstrap website: www.getbootstrap.com
- Website has instructions on how to Get Started



Step 1: Get links for remote Bootstrap files

CSS

Copy-paste the stylesheet `<link>` into your `<head>` before all other stylesheets to load our CSS.

Copy

```
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta3/dist/css/bootstrap.min.css
```

Step 2: Add links in Thymeleaf template

File: list-employees.html

```
<!DOCTYPE HTML>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
<!-- Bootstrap -->
 <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.2.1/css/bootstrap.min.css"
   integrity="sha384-GJzZqFGwb1QTTN6wy59ffF1BuGJpLSa9DkKMp0DgiMDm4iYMj70gZWKYbI706tWS"
   crossorigin="anonymous">
<title>Employee Directory</title>
</head>
```

Step 3: Apply CSS

File: list-employees.html

```
<table class="table table-bordered table-striped">
    <thead class="thead-dark">
        <tr>
            <th>First Name</th>
            <th>Last Name</th>
            <th>Email</th>
        </tr>
    </thead>
    <tbody ...7 lines />
</table>
```

Employee Directory

First Name	Last Name	Email
Leslie	Andrews	leslie@mail.com
Emma	Baumgarten	emma@mail.com
Avani	Gupta	avani@mail.com

Thymeleaf CRUD Project

(Project: 34-thymeleafdemo-employess-list-db)

Application Requirements

- Create a Web UI for the Employee Directory
- Users should be able to
  - Get a list of employees
  - Add a new employee
  - Update an employee
  - Delete an employee

Thymeleaf + Spring Boot

Application Requirements

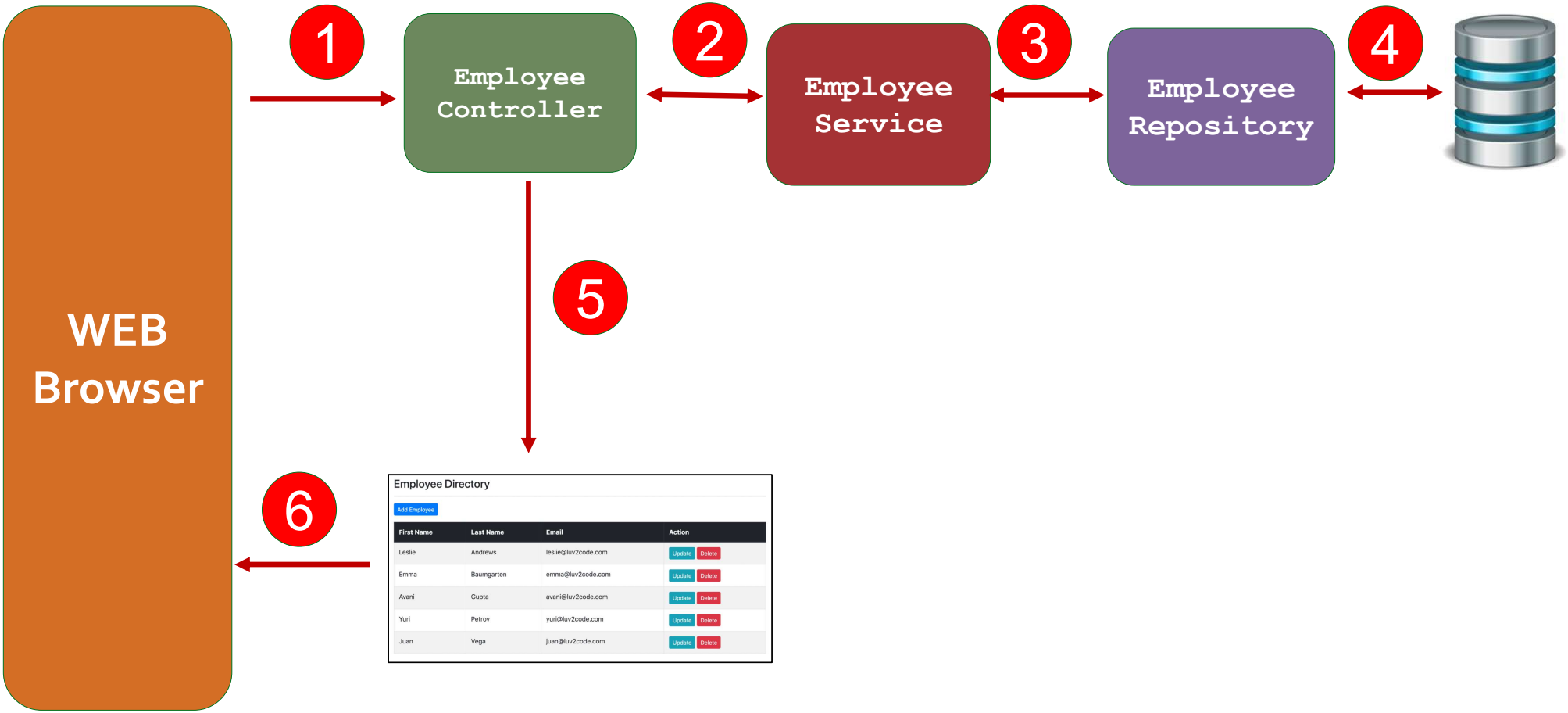
Employee Directory

Thymeleaf + Spring Boot

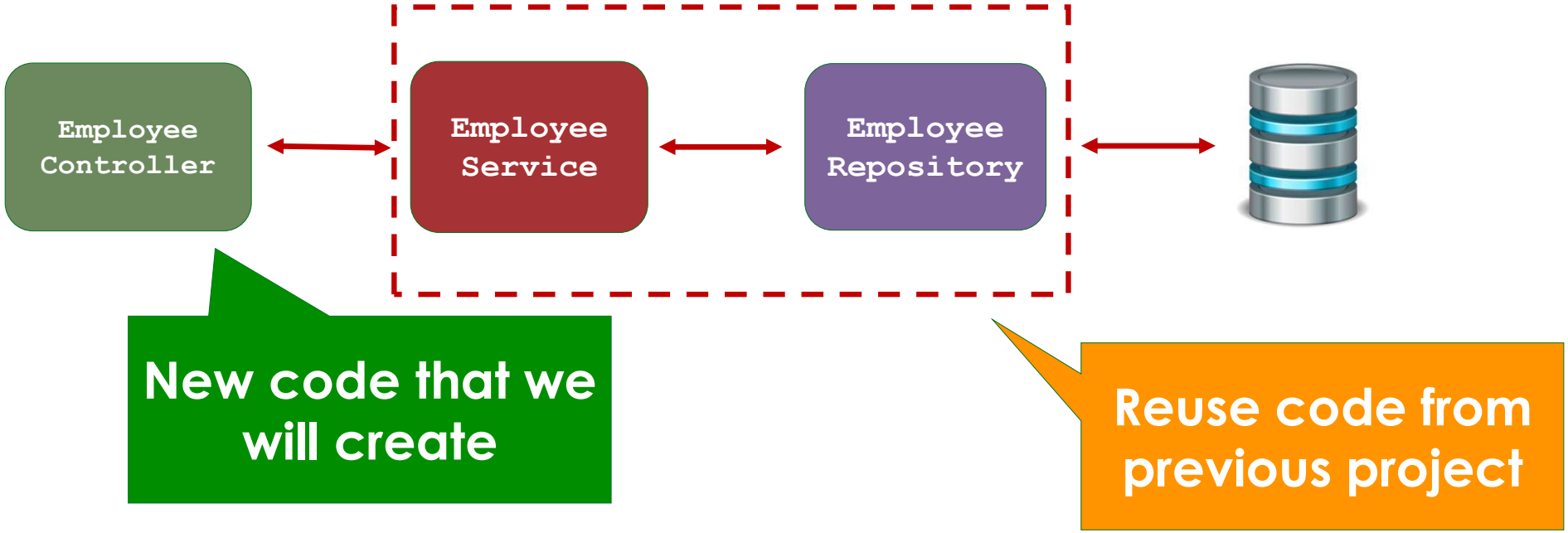
Add Employee

First Name	Last Name	Email	Action
Leslie	Andrews	leslie@luv2code.com	Update Delete
Emma	Baumgarten	emma@luv2code.com	Update Delete
Avani	Gupta	avani@luv2code.com	Update Delete
Yuri	Petrov	yuri@luv2code.com	Update Delete
Juan	Vega	juan@luv2code.com	Update Delete

Application processing flow



Application Architecture



Questions