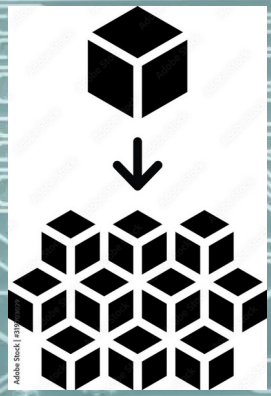
Angular Native Fundamentals

Full Stack Bootcamp (Day 4)

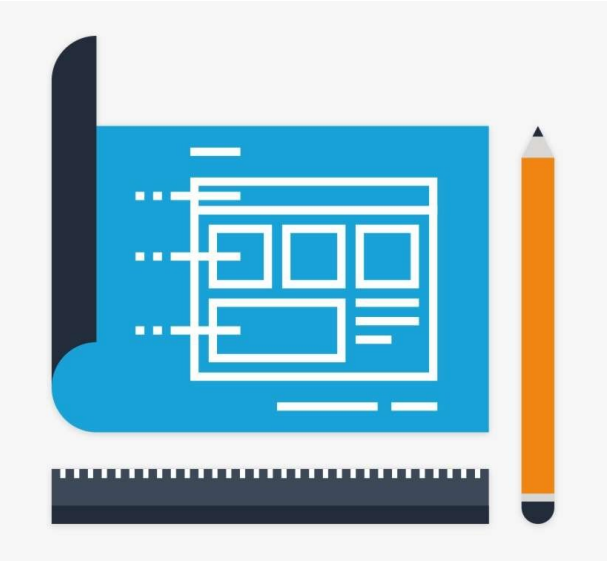


Agenda

- Day 1
  - ES6+
- Day 2
  - React Native
- Day 3
  - Angular
- Day 4
  - Springboot
  - SpringData
- Day 5
  - JSON
  - NoSQL
- Day 6
  - Relational
- Day 7
  - Junit
  - Mockito
- Day 8
  - Docker
- Day 9
  - Kubernetes
- Day 10
  - Images and tips

Welcome to the back-end nightmare.

- Architecture definitions
- Web services
- Type of architectures
  - Microservices based architecture
  - Event Driven Architecture
- What else to learn

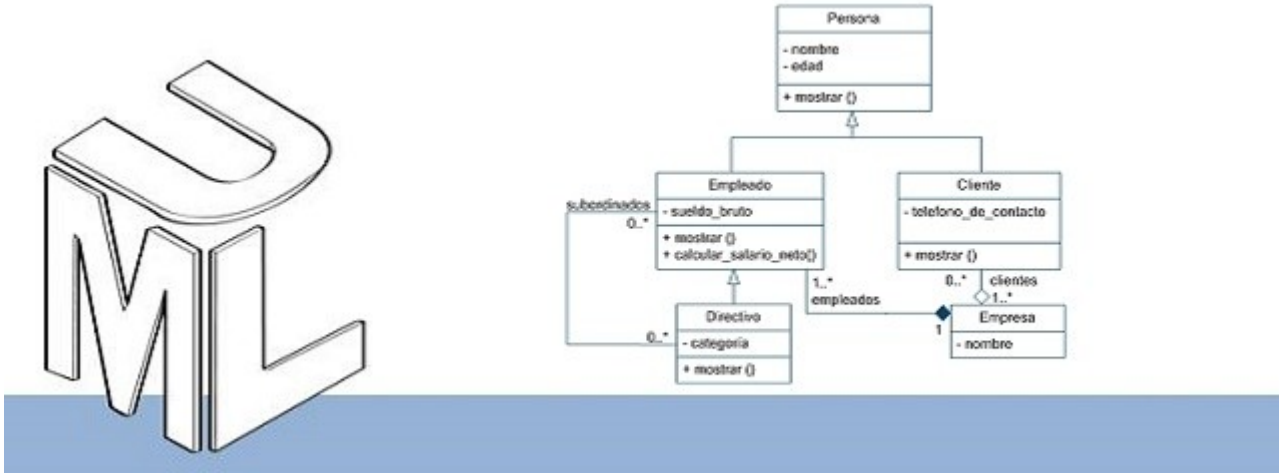


Architecture definitions

- Languages type:
  - Scripting
  - Procedural
  - OO
  - Functional
  - OO+Functional
- Applications type
  - Stand alone
  - Client/Server
  - Web based
- Microservices based Architecture

Architecture definitions

- Languages type:
  - Scripting
  - Procedural
  - OO
  - Functional
  - OO+Functional
- Applications type
  - Stand alone
  - Client/Server
  - Web based



Microservices based Architecture

- The spaghetti monster
  - Responsibility delegation (OOP)
- Layered applications
- Monolithic based application
- Decoupling applications

Microservices advantages

- Independence on processes
- Easier to maintain (when well documented)
- Security
- Flexible
- Integration with Agile

Eclipse IDE quick view

- Download and Installation
- “flavors” (MyEclipse, Eclipse, Spring Tools, Android Studio)
- Other Options (IntelliJ, Netbeans, VisualStudio Code)

Maven and Gradle Introduction

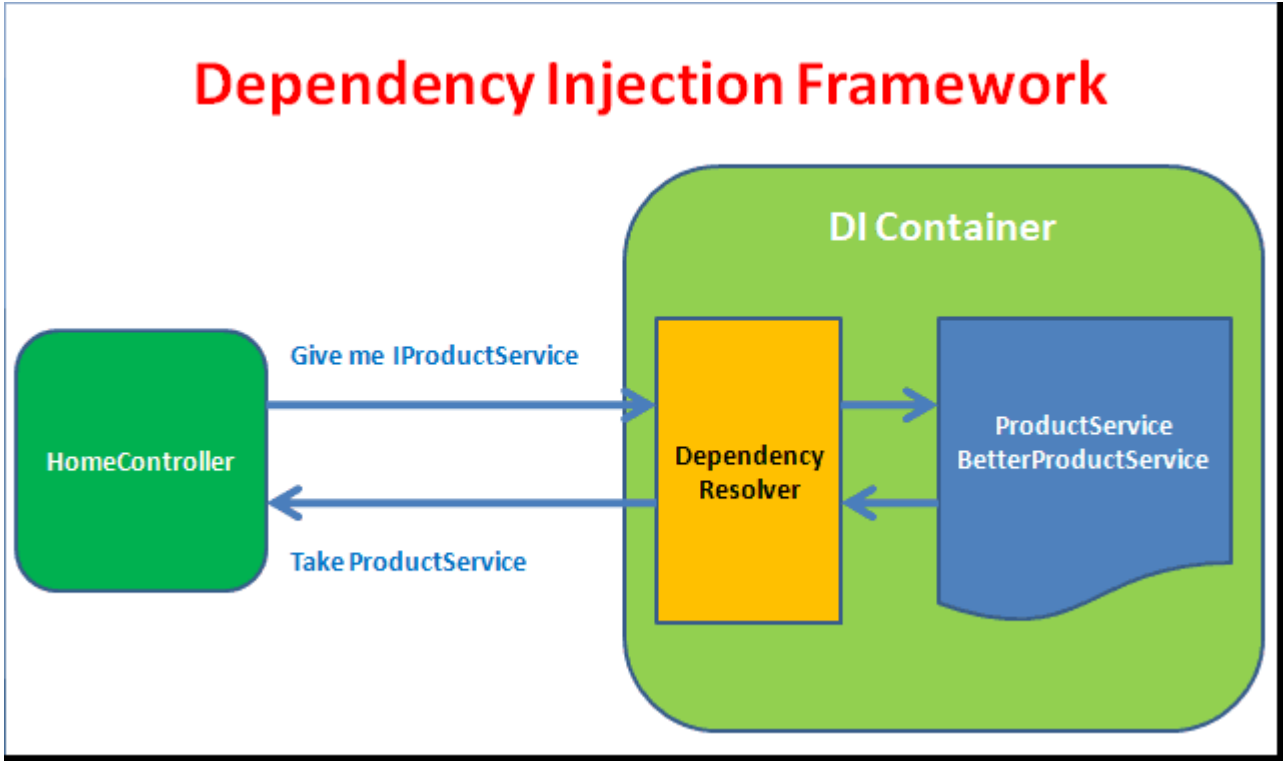
- Repositories
  - Global Maven repository
  - Mirrors
  - Corporate repositories
  - Local repositories

Git overview

- Whats a CVS?
  - CVS
  - SVN
- Branches
- Distributed Version System
  - git clone
  - git push
  - git add
  - git commit

Dependency Injection

- Definition
- Java EE
- Spring
- Annotations

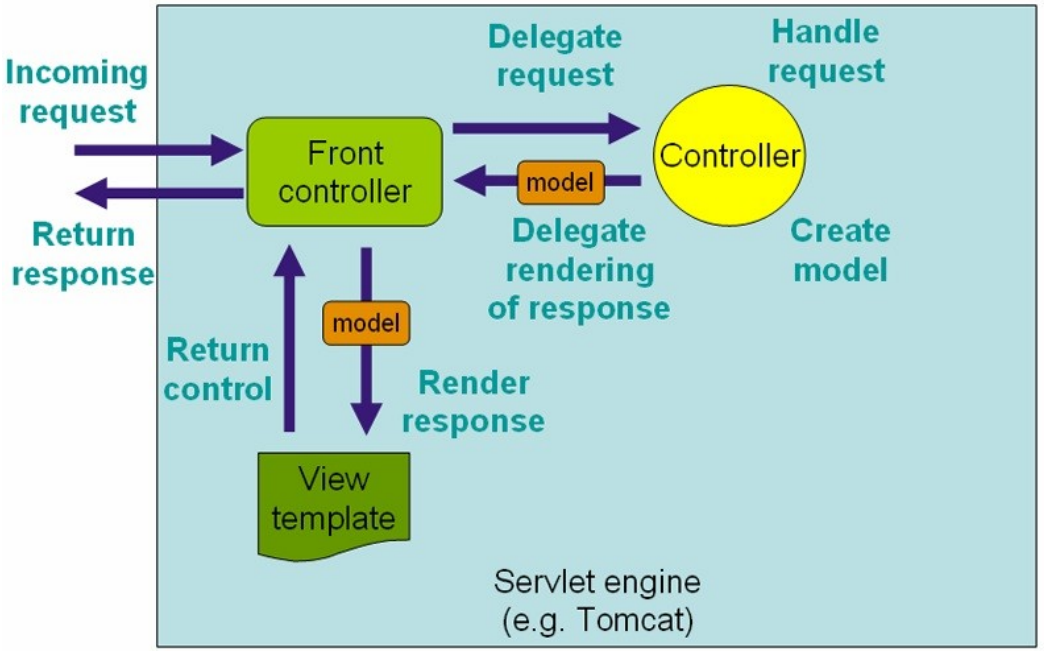


Exercise 1

- Install and set Eclipse IDE Spring tools

Spring MVC

- DispatcherServlet
- Configuration type
  - File based
  - Annotation based
- Beans
- Using Beans



Model View Controller

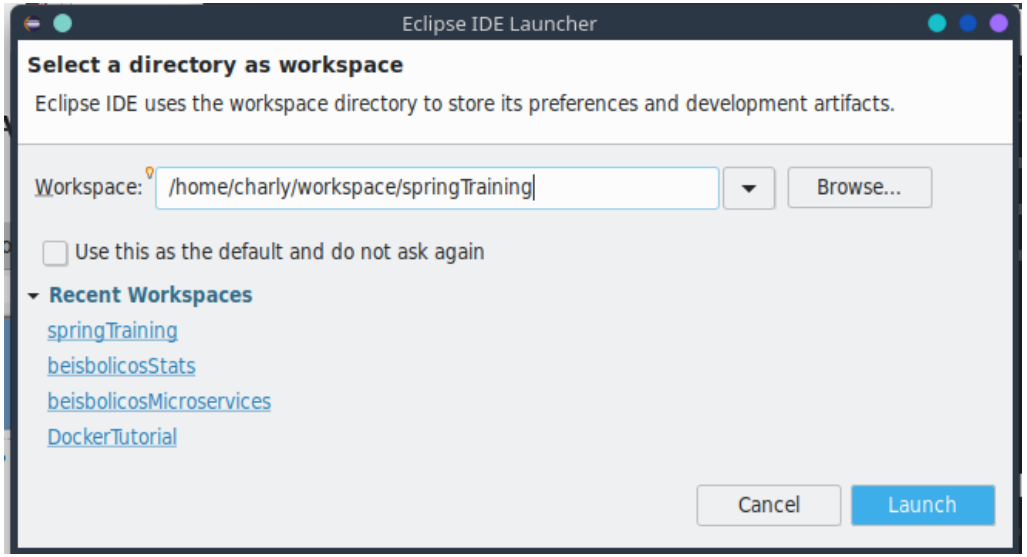
- Controllers are objects that will receive our requests and will process them
- Controllers are not used only for front end apps
- Controllers can redirect the servlet to a “View” that can be a web page or a webservice response, for this course we will be working with web services
- Views are the objects that “paint” the responses (usually a JSP or JSF) for this course our views are really the Angular or React front ends
- Model refers to the objects that represents a business object

Spring Boot (REST)

- ¿What's a web service?
- SOAP
- REST
- Http Verbs
  - @RestController
  - @GetMapping
  - @PostMapping
  - @DeleteMapping
  - @ResponseBody
  - @ResponseStatus
  - @ExceptionHandler

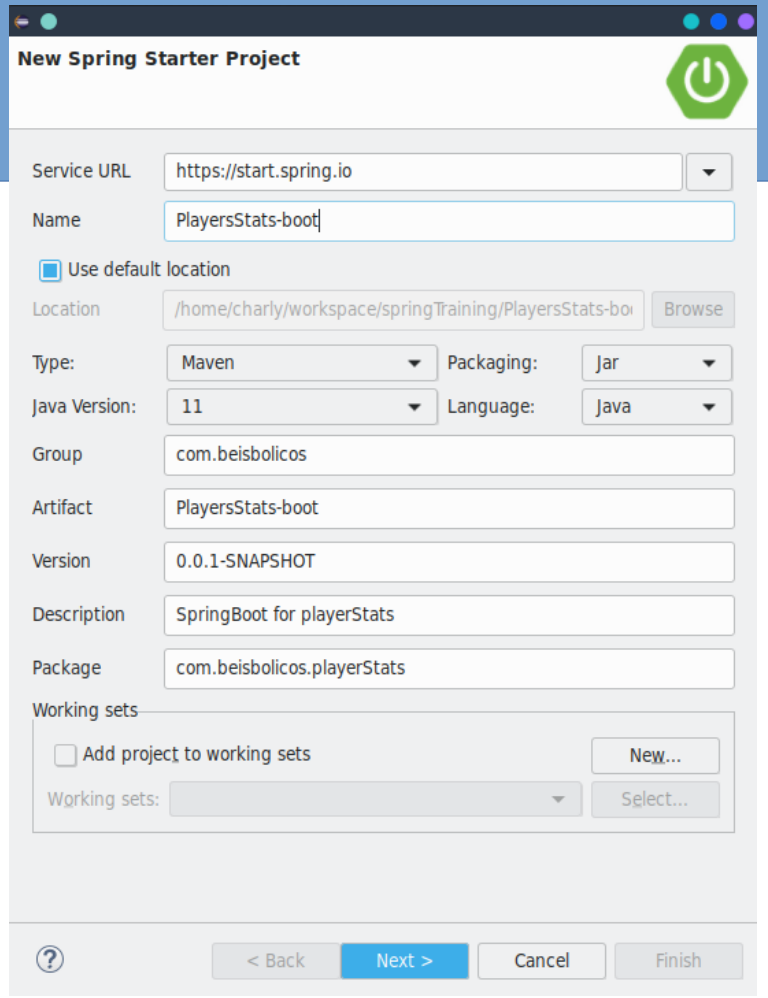
Hello World

- Understanding workplaces
- SpringTools

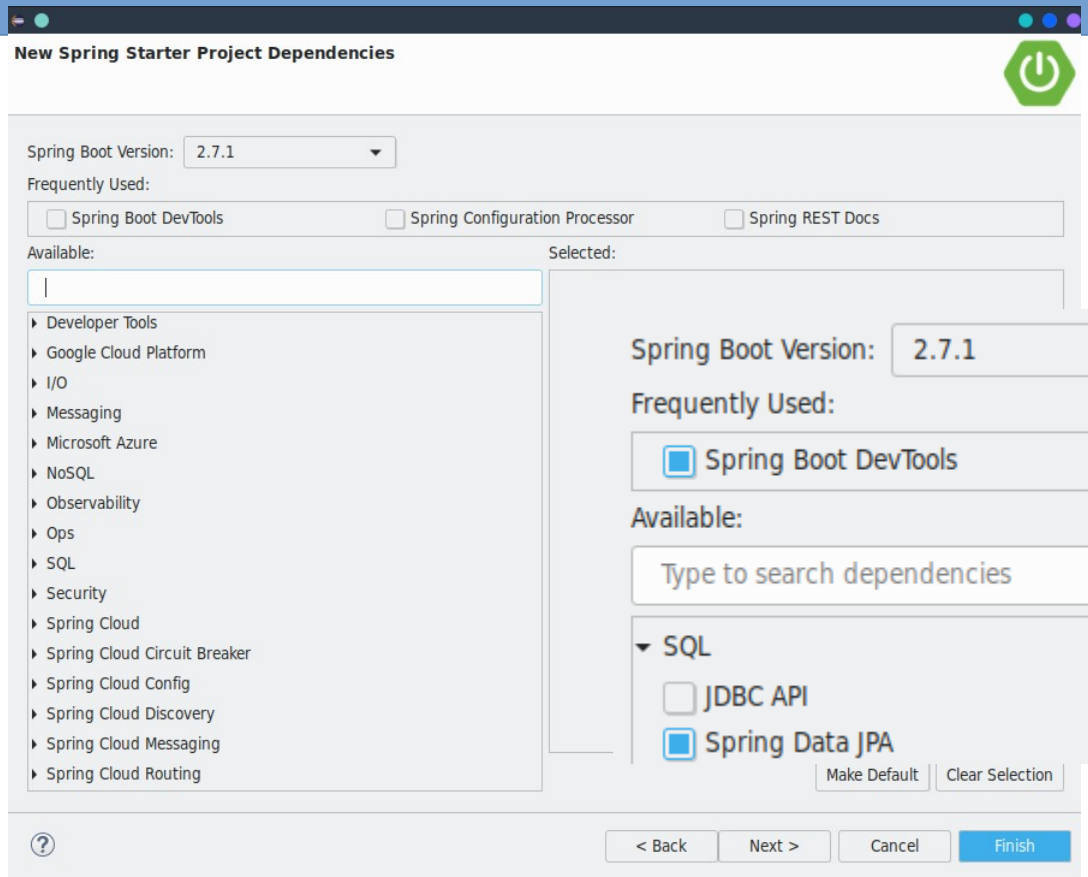


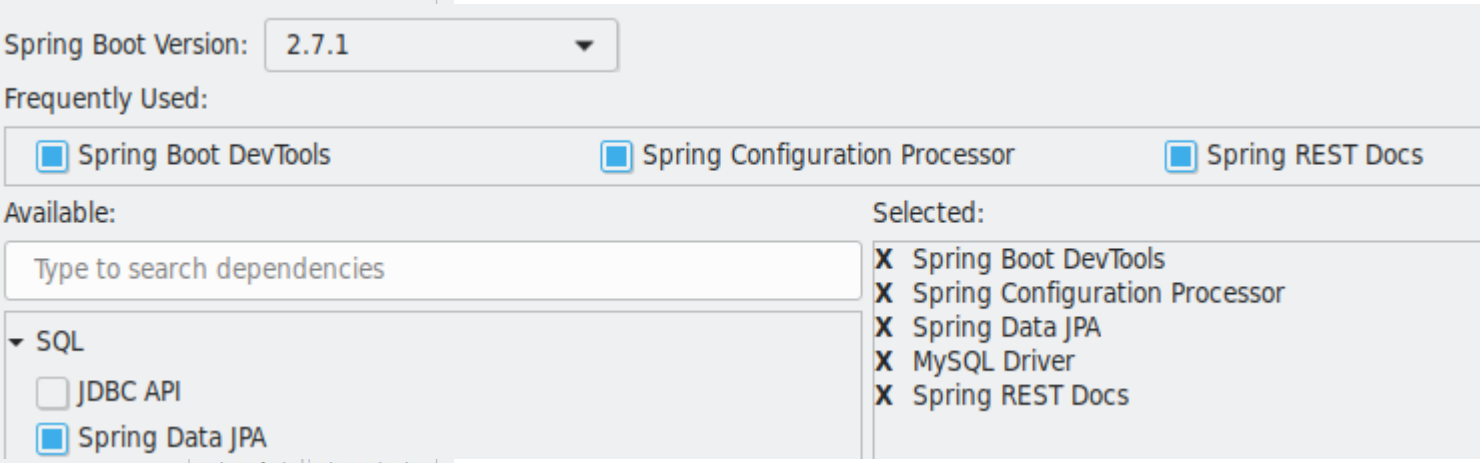
Create the project

- From the help Menu open the spring dashboard
- On the dashboard select “Create Spring Starter project”
- Select a name for your project



Create the project (2)





Overview

- Easy to create stand-alone, production-grade Spring based applications that you can “just run”.
  - It needs very little spring configuration.
- Create Java applications that can be started using java –jar or more traditional war deployments.
- Primary goals for the Spring Boot.
  - Provide a radically faster and widely accessible getting started experience for all Spring development.
  - Be opinionated out of the box, but get out of the way quickly as requirements start to diverge from the defaults.
  - Provide a range of non-functional features that are common to large classes of projects (e.g.embedded servers, security, metrics, health checks, externalized configuration).
  - Absolutely no code generation and no requirement for XML configuration.

Spring boot system requirements

- Even when it varies from version to version here are some recommendations to start with Spring boot
- Java 1.8 or above
- Tomcat 8 (or any servlet 3+ compatible app server)
- Maven/Gradle
- Java IDE (Eclipse, Jbuilder or Netbeans recommended)

Maven dependencies

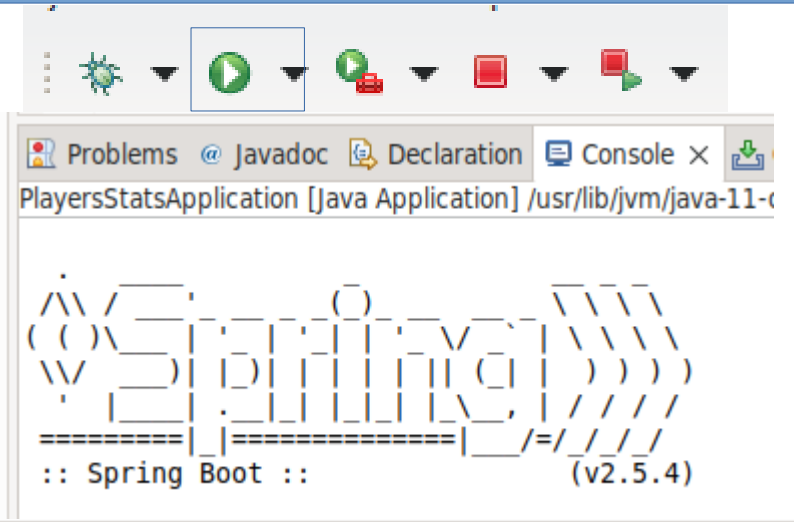
```
<dependencies>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
<groupId>org.springframework</groupId>
<artifactId>spring-web</artifactId>
</dependency>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-devtools</artifactId>
<scope>runtime</scope>
<optional>true</optional>
</dependency>
<dependency>
<groupId>mysql</groupId>
<artifactId>mysql-connector-java</artifactId>
<scope>runtime</scope>
</dependency>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-configuration-
processor</artifactId>
<optional>true</optional>
</dependency>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-test</
artifactId>
<scope>test</scope>
</dependency>
<dependency>
<groupId>org.springframework.restdocs</groupId>
<artifactId>spring-restdocs-mockmvc</
artifactId>
<scope>test</scope>
</dependency>
</dependencies>
```

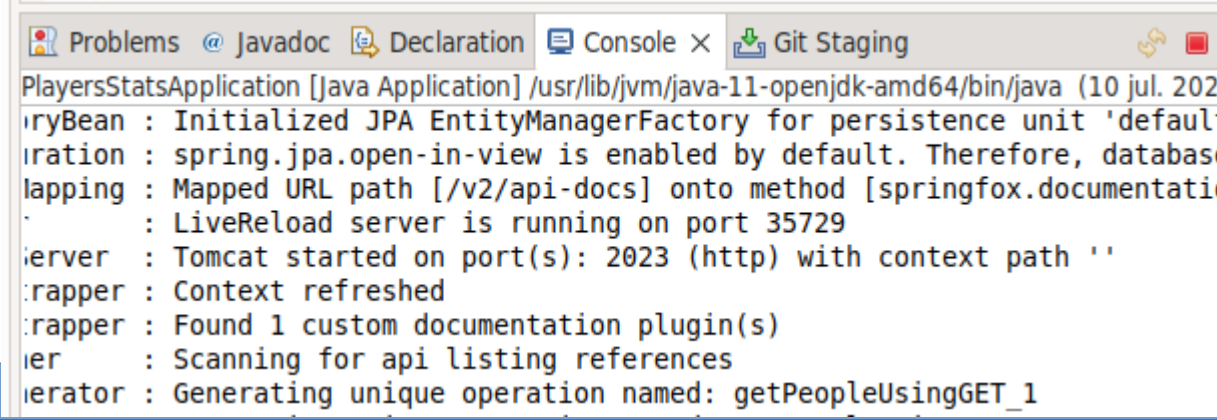
Hello World

```java
package com.beisbolicos.playerStats;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.web.bind.annotation.RequestMapping;

@SpringBootApplication
public class PlayersStatsBootApplication {

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

	@RequestMapping("/")
	public String helloWorld() {
		return "Hello World";
	}
}
```

Run your app

- Click on the run button on the toolbar and look the springboot message
- Look in the console for the URL and open it





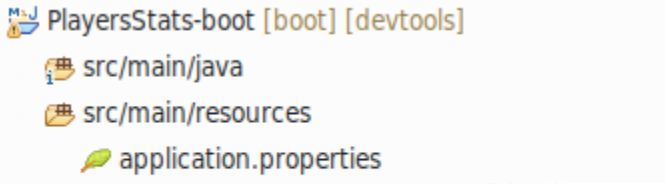
Configuring the project

- @Configuration annotation helps to create the configuration without XML files
- Application properties files is to set simple configuration variables needed.

Our back end app

- We’ll be working with MySQL (please install it before proceeding). If you are not an MySQL expert a good idea is to install MySQL workbench as well
- The configuration class will provide the DB Connection parameters and will linked to the Spring injected classes.
- Our application will provide a REST API for the “Baseball Data Bank”, we will be providing just the “player” table for the example.

Application.properties




```
1
2  server.port=2023
3
4  spring.datasource.url= jdbc:mysql://localhost:3306/baseballdb
5  spring.datasource.username=root
6  spring.datasource.password=Pqsfypqm1@
7
8  #spring.jpa.hibernate.ddl-auto=create-drop
9
10 spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
11 #spring.datasource.hikari.connection-timeout=60000
12 spring.jpa.database=mysql
13
14 spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5Dialect
15 spring.jackson.serialization.FAIL_ON_EMPTY_BEANS=false
```

Spring Data JPA (Hibernate)

- JPA it’s a relational database access framework that has several implementations
- JPA translates the databases to Java Objects
- JPA implementations:
  - Hibernate
  - Eclipse Link
  - Apache JPA

EntityManager

- EntityManager is a component that spring will configure atuomatically and has all the resources to connect the entities to the DB, that is made with the DataStore configuration

```
1  package com.beisbolicos.playerStats;
2
3  import javax.sql.DataSource;
12
13 @Configuration
14 @EnableJpaRepositories(basePackages = { "com.beisbolicos.playerStats.repo" })
15 @ComponentScan(value = "com.beisbolicos.playerStats.*")
16 @EntityScan(basePackages = { "com.beisbolicos.playerStats.entity" })
17 public class DataStoreSetup {
18
19     @Value("${spring.datasource.url}")
20     String databaseUrl;
21
22     @Value("${spring.datasource.username}")
23     String databaseUser;
24
25     @Value("${spring.datasource.password}")
26     String databasePassword;
27
28     @Bean
29     public DataSource dataSource() {
30
31         DriverManagerDataSource dataSource = new DriverManagerDataSource();
32         dataSource.setUrl(databaseUrl);
33         dataSource.setUsername(databaseUser);
34         dataSource.setPassword(databasePassword);
35         return dataSource;
36     }
37
38 }
```

The Entities (PeopleEntity)

- The entities are objects that will match a part of the RDB to a Java Object.

```
import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
@Entity
@JsonIgnoreProperties({"hibernateLazyInitializer", "handler"})
@Table(name="people")
public class PeopleEntity implements Serializable{
```

PeopleEntity

```
@Id
@Column(name = "playerid", nullable = false, unique = true)
private String playerId;

@Column(name="namefirst" )
private String nameFirst;

@Column(name="namelast")
private String nameLast;

@Column(name="birthcity")
private String birthCity;

@Column(name="birthcountry")
private String birthContry;

@Column(name="birthday")
private String birthDay;
```

```
@Column(name="birthmonth")
private String birthMonth;

@Column(name="birthyear")
private String birthYear;
```

Batting Entity

- Every player has a yearly batting statistics

```
@Entity
@IdClass(BattingKey.class)
@JsonIgnoreProperties({"hibernateLazyInitializer", "handler"})
@Table(name="batting")
public class Batting implements Serializable{
private static final long serialVersionUID = 1L;
@Id
@Column(name = "playerid", nullable = false)
private String playerId;
@Id
@Column(name = "yearid", nullable = false)
private int yearId;
@Id
@Column(name = "stint")
private String stint;
@Id
@Column(name = "teamid", nullable = false)
private String teamId;
@Column(name="sf")
private int sacrificeFly;
@Column(name="gidp")
private int groundInDoublePlay;
```

```
@Column(name="lgid")
private String lgId;
@Column(name="G")
private int games;
@Column(name="ab")
private int atBat;
@Column(name="r")
private int runs;
@Column(name="h")
private int hits;
@Column(name="2b")
private int
doubleHits;
@Column(name="3b")
private int
tripeHits;
@Column(name = "hr")
```

```
private int homeRuns;
@Column(name = "rbi")
private int runsBattedIn;
@Column(name = "sb")
private int stolenBases;
@Column(name = "cs")
private int caughtStealing;
@Column(name = "bb")
private int baseOnBalls;
@Column(name = "so")
private int strikedut;
@Column(name = "ibb")
private int
intentionalBaseOnBalls;
@Column(name="hbp")
private int hitByPitch;
@Column(name="sh")
private int sacrificeHit;
```

Queries

```
package com.beisbolicos.playerStats.repo;
import org.springframework.stereotype.Repository;
import com.beisbolicos.playerStats.entity.People;
import java.util.List;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;

@Repository
public interface PeopleRepository extends JpaRepository<People, String>{

@Query("select p  from People p, Batting b where p.playerId=b.playerId and
b.teamId = ?1 and b.yearId = ?2")
public List<People> findByTeamYear(String teamId, int yearId);

}
```

Using the entities and repositories

```
import java.util.List;

import javax.persistence.EntityNotFoundException;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import com.beisbolicos.playerStats.entity.People;
import com.beisbolicos.playerStats.repo.PeopleRepository;
import com.beisbolicos.playerStats.service.IPeopleService;
@Service
public class PeopleService implements IPeopleService {
@Autowired
PeopleRepository peopleRepository;
@Override
public void createPeople(People employee) {
peopleRepository.save(employee);
}
@Override
public void deletePeople(String id) {
peopleRepository.deleteById(id);
}
@Override
public List<People> getPeople() {
List<People> people = peopleRepository.findAll();
return people;
}
public List<People> getPeople(String teamId, int yearId) {
List<People>people = peopleRepository.findByTeamYear(teamId, yearId);
return people;
}
}
@Override
public People getPeopleById(String id) {
People people;
try {
people = peopleRepository.getById(id);
}catch (EntityNotFoundException e){
people = null;
}
return people;
}

@Override
public void updatePeople(People employee) {
peopleRepository.save(employee);

}
```

Displaying the results with the controller (1)

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RestController;

import com.beisbolicos.playerStats.entity.People;
import com.beisbolicos.playerStats.serviceImpl.PeopleService;

import io.swagger.annotations.ApiOperation;
import io.swagger.annotations.ApiResponse;
import io.swagger.annotations.ApiResponses;
import io.swagger.annotations.Example;
import io.swagger.annotations.ExampleProperty;
import io.swagger.annotations.SwaggerDefinition;
import io.swagger.annotations.Tag;

@RestController
@SwaggerDefinition(tags = {@Tag (name="Consultas", description="Diferentes consultas para la BDB")})

public class PeopleController {
```

Displaying with the controller (2)

```
@Autowired
PeopleService peopleService;

@PostMapping(value = "/people")


public ResponseEntity<Object> createEmployee(@RequestBody People people) {

peopleService.createPeople(people);
return new ResponseEntity<Object>("Successfully Saved", HttpStatus.OK);
}
```

Displaying with the Controller (3)

```
@GetMapping(value = "/people/{id}")
/**
 * Gets the player data according with the ID
 * @param id Gusually the first 5 letter of the lastname with 2 of the name and 2 digits to avoid collisions
 * @return Player or Manager data
 */
@ApiOperation(value = "Queries a people on the Baseball Data Bank",
              notes = "Queries a people on the Baseball Data Bank")
@ApiResponses(value = {
@ApiResponse(code = 200, message = "Successfully got the people",
examples = @Example(value = @ExampleProperty(mediaType="application/json",
value="{\"playerId\":\"allenjo02\","
+ "\"nameFirst\":\"Johnny\","
+ "\"nameLast\":\"Allen\","
+ "\"birthCity\":\"Lenoir\","
+ "\"birthContry\":\"USA\","
+ "\"birthDay\":\"30\","
+ "\"birthMonth\":\"9\","
+ "\"birthYear\":\"1904\"}"))),
@ApiResponse(code = 404, message = "Player or manager not found"),
@ApiResponse(code = 400, message = "Missing or invalid request body"),
@ApiResponse(code = 500, message = "Internal error")
}
)
public ResponseEntity<Object> getPeople(@PathVariable String id) {

People people = peopleService.getPeopleById(id);
return new ResponseEntity<Object>(people, HttpStatus.OK);
}
```

Displaying with the Controller (4)

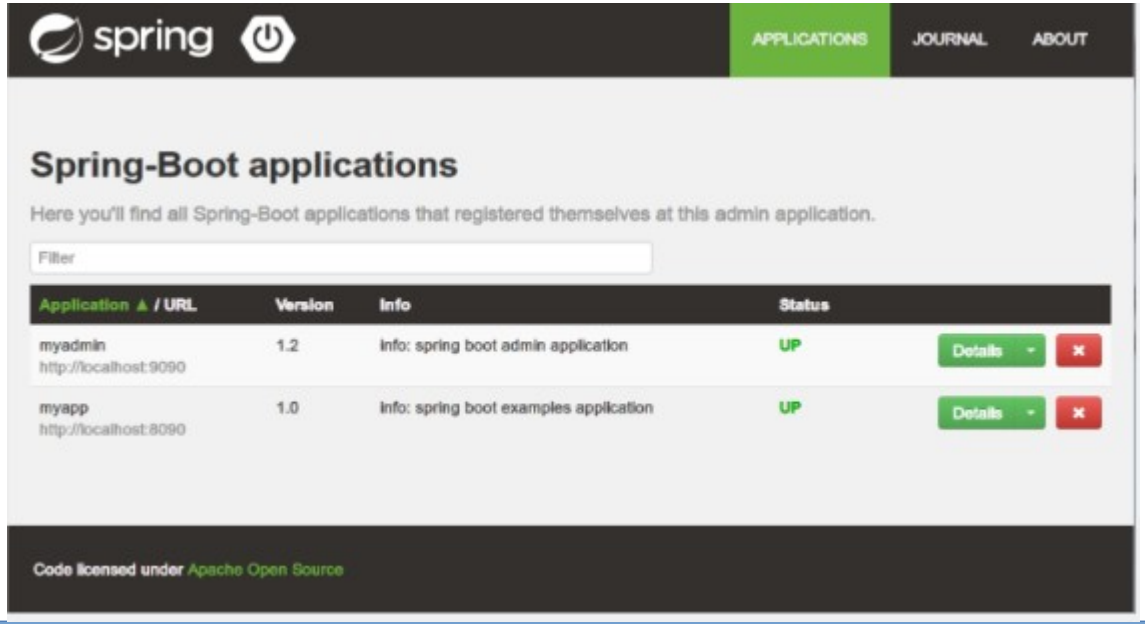
```
@PutMapping(value = "/people")
public ResponseEntity<Object> updateEmployee(@RequestBody People people) {

peopleService.updatePeople(people);
return new ResponseEntity<Object>("Successfully Updated", HttpStatus.OK);
}

@DeleteMapping(value = "/people/{id}")
public ResponseEntity<Object> deleteEmployee(@PathVariable String id) {

peopleService.deletePeople(id);
return new ResponseEntity<Object>("Successfully Deleted", HttpStatus.OK);
}
```

Displaying with the Controller (5)

```
@GetMapping(value="/people")
public ResponseEntity <Object> getPeople(){
List<People> people = peopleService.getPeople();
ResponseEntity<Object> listPeople = new ResponseEntity<>(people, HttpStatus.OK);
return listPeople;
}

@GetMapping(value="/people/{teamId}/{yearId}")
public ResponseEntity <Object> getPeople(@PathVariable String teamId, @PathVariable int yearId){
List<People> people = peopleService.getPeople(teamId, yearId);
ResponseEntity<Object> listPeople = new ResponseEntity<Object>(people, HttpStatus.OK);
return listPeople;
}

}
```

Spring boot Administrator

- Spring Boot Admin is a simple application to manage and monitor your Spring Boot Applications.
- The applications register with our Spring Boot Admin Client (via http) or are discovered using Spring Cloud (e.g. Eureka).



Spring boot admin application

- Add spring boot server libraries to project dependencies.
- Create SpringBootAdminApplication

```
<!-- https://mvnrepository.com/artifact/de.codecentric/spring-boot-admin-dependencies -->
<dependency>
    <groupId>de.codecentric</groupId>
    <artifactId>
            spring-boot-admin-dependencies
    </artifactId>
    <version>2.1.1</version>
    <type>pom</type>
</dependency>
```

```
1  package examples.spring.boot.admin;
2
3  import org.springframework.boot.SpringApplication;
8
9  @Configuration
10 @EnableAutoConfiguration
11 @EnableAdminServer
12 public class SpringBootAdminApplication {
13     public static void main(String[] args) {
14         SpringApplication.run(SpringBootAdminApplication.class, args);
15     }
16 }
```

Application properties for the server

```
# ========================
# Tomcat Configuration
# ========================
server.tomcat.max-threads=10
server.address=127.0.0.1
server.port=9090

# ========================
# Security Configuration
# ========================
security.user.name=admin
security.user.password=admin
management.security.role=SUPERU
SER
management.security.enabled=false
```

Adding client to our controller

- Add the maven dependencies

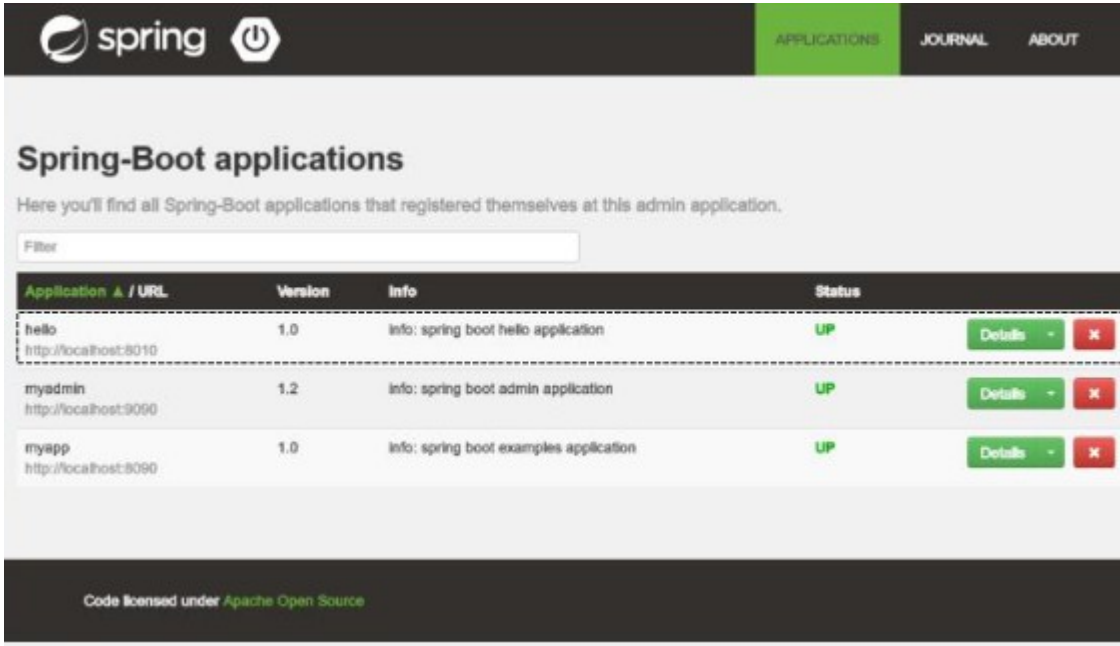
```
<!-- https://mvnrepository.com/artifact/de.codecentric/spring-boot-admin-starter-client -->
<dependency>
    <groupId>de.codecentric</groupId>
    <artifactId>spring-boot-admin-starter-client</artifactId>
    <version>3.0.0-M3</version>
</dependency>
```

Properties in the application.properties for spring boot client

```
# ======================
# JMX Configuration
#management.port=8011
#management.address=127.0.0.1
management.security.role=SUPERUSER
management.security.enabled=false
# ========================================
# Client Configuration for Spring Boot Admin
info.version=1.0
info.info=spring boot hello application
spring.boot.admin.client.name=hello
spring.boot.admin.url=http://127.0.0.1:9090
spring.boot.admin.username=admin
spring.boot.admin.password=admin
spring.boot.admin.client.health-url=http://localhost:8010/health
spring.boot.admin.client.service-url=http://localhost:8010
spring.boot.admin.client.management-url=http://localhost:8010
```

Spring boot admin checkup

- You don’t need to write code on your Spring boot App (that’s the beauty of Spring Boot)
- Run your Spring boot server
- Run the admin server application



Spring Cloud

- Microservices.Consuming Rest Services

```
@Bean
public RestTemplate restTemplate() {
    return new RestTemplate();
}
public class User {
    private Long id;
    private String username;
    private String firstname;
    private String lastname;
            …..
@Autowired
private RestTemplate restTemplate;
String url = "http://example.org/path/to/api";
User response = restTemplate.getForObject(url, User.class);
User[] response = restTemplate.getForObject(url, User[].class);
```

Kafka

- Kafka is an Open Source QUEUE manager
- QUEUES vs Topics
- JMS
- https://docs.spring.io/spring-kafka/reference/html/

Day 4 summary

- Back-end development introduction
- Spring boot
- REST-Controller
- JPA
- REST client
- Kafka Introduction