Campus BBVA
Tech University

Spring

Creación de implementación y cargador de la API

Implementación de la API

- Crear proyecto
- Definir la clase principal
- Empaquetar proyecto y arrancar el servidor
- Implementar el controlador
- Implementar la entidad
- Implementar el servicio
- Implementar la capa de acceso a datos

Creación de un servicio REST con Spring

Requisitos de entorno

- Java 1.8
- IDE: Eclipse | IntelliJ IDEA
- Maven
- Dependencias de Spring Boot o Spring Web

Creación de un servicio REST con Spring

Crear el proyecto

- Crear un proyecto maven nuevo indicando en el **pom.xml** que queremos usar Spring Web con Spring Boot:

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>1.4.1.RELEASE</version>
</parent>

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

Creación de un servicio REST con Spring

Definir clase principal

- SpringApplication

```
@SpringBootApplication
public class Application {

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

Creación de un servicio REST con Spring

Empaquetar el proyecto y lanzar el servidor

- Generar ficheros .class y .jar

```
$ mvn clean package
```

- Comprobar dependencias

```
$ mvn dependency:tree
```

- Arrancar la aplicación (servidor levantado y listo para recibir peticiones)

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

Campus BBVA
Tech University

Spring

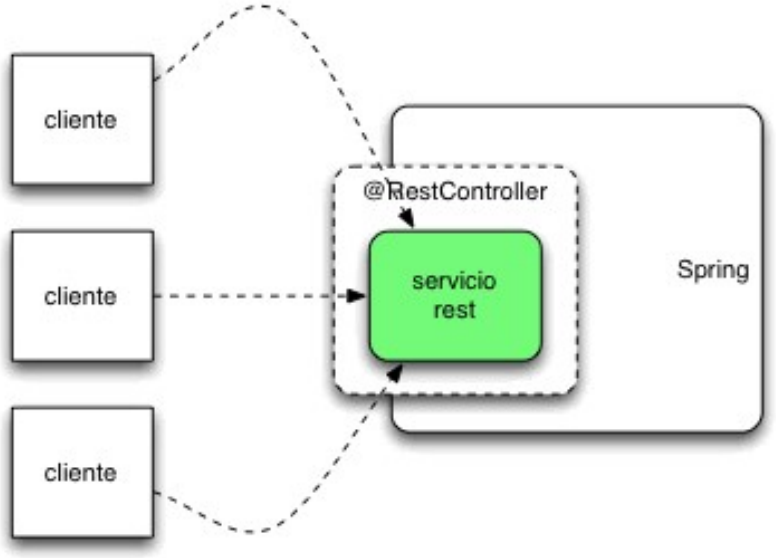
Configuración API: Entidad, Servicio, Controller

Implementación de la API

- Crear proyecto
- Definir la clase principal
- Empaquetar proyecto y arrancar el servidor
- Implementar el controlador
- Implementar el servicio
- Implementar la entidad
- Implementar la capa de acceso a datos

El Controlador - @RestController

- La clase anotada con @RestController será la encargada de gestionar las peticiones que se hagan a nuestra API.
- Indica que los datos devueltos por cada método se escribirán directamente en el cuerpo de la respuesta (response body).
- Sustituye al uso de @Controller + @ResponseBody.



El Servicio - @Service

- Funcionamiento parecido a **@Controller**
- Permite que Spring reconozca a la clase anotada como servicio al escanear los componentes de la aplicación

Implementación de la capa de acceso a datos

- Spring Boot + Spring data JPA

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                             http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <artifactId>spring-data-jpa</artifactId>
    <packaging>jar</packaging>
    <name>Spring Boot Spring Data JPA</name>
    <version>1.0</version>

    <parent>... </parent>

    <dependencies>
        <!-- jpa, crud repository -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>
    </dependencies>
</project>
```

@Entity

- Anotación que define objeto para persistencia en bases de datos basadas en JPA
- Permite asociar una clase a una tabla o colección
- Otras implementaciones: Spring Data, MongoDB, Spring Data Cassandra, etc…
- Anotar clases del modelo de persistencia

La Entidad - @Entity

- Model and JPA annotations

```
@Entity
public class Customer {

    // "customer_seq" is Oracle sequence name.
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "CUST_SEQ")
    @SequenceGenerator(sequenceName = "customer_seq", allocationSize = 1, name = "CUST_SEQ")
    Long id;

    String name;

          String email;

    @Column(name = "CREATED_DATE")
    Date date;

    //getters and setters, contructors
}
```

Configuration + Database Initialization

- Configure **Oracle** data source

```
# Oracle settings
spring.datasource.url=jdbc:oracle:thin:@localhost:1521:xe
spring.datasource.username=system
spring.datasource.password=password
spring.datasource.driver-class-oracle.jdbc.driver.OracleDriver
```

Configuration + Database Initialization

- Configure **MongoDB** data source

```
# Spring properties
spring:
  data:
    mongodb:
      host: localhost
      port: 27017
      uri: mongodb://localhost/test

# HTTP Server
server:
  port: 4444    # HTTP (Tomcat) port
```

application.yml*

Implementación de la capa de acceso a datos

- Spring Boot + Spring data JPA

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-mongodb</artifactId>
</dependency>
<dependency>
        <groupId>org.mongodb</groupId>
        <artifactId>mongo-java-driver</artifactId>
</dependency>
```

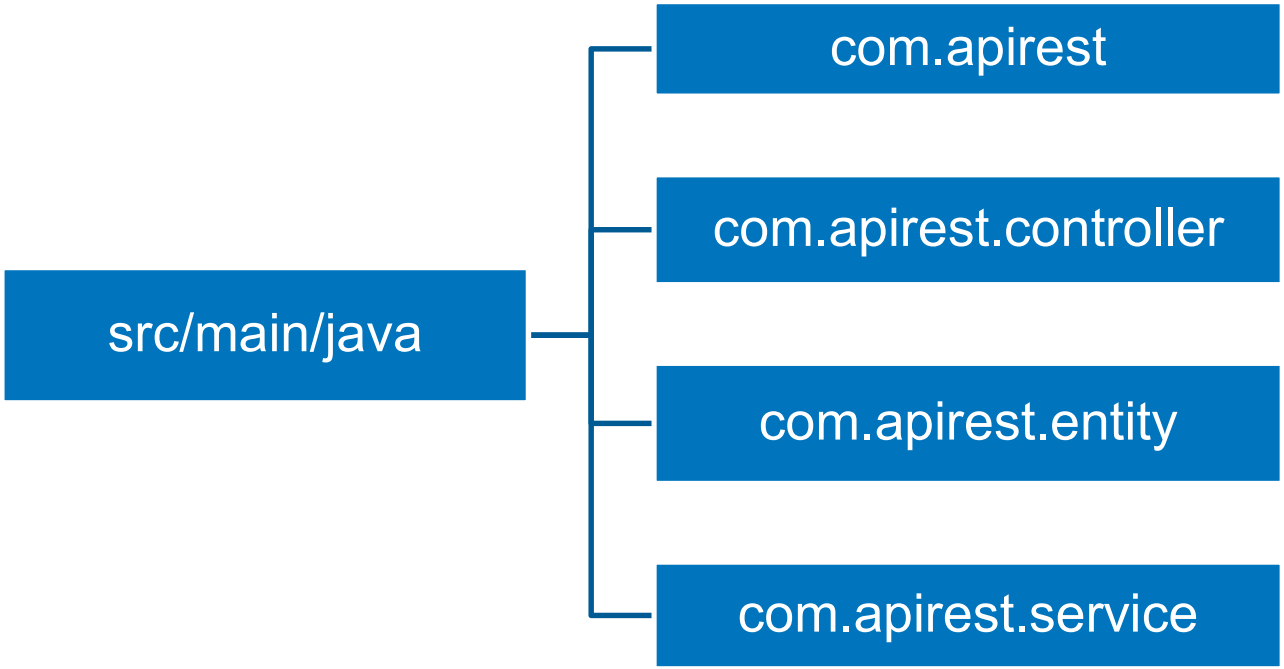
Implementación de la capa de acceso a datos

- Spring Data CrudRepository

```
import com.apirest.model.Customer;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.CrudRepository;
import org.springframework.data.repository.query.Param;

import java.util.Date;
import java.util.List;
import java.util.stream.Stream;

public interface CustomerRepository extends CrudRepository<Customer, Long> {

    List<Customer> findByEmail(String email);

    List<Customer> findByDate(Date date);

          // custom query example and return a stream
    @Query("select c from Customer c where c.email = :email")
    Stream<Customer> findByEmailReturnStream(@Param("email") String email);

}
```

Configuración API REST

- Estructura de la API



Anotaciones REST API

Anotaciones peticiones HTTP y mapeo de URLs

- **@GetMapping**
- **@PostMapping**
- **@PutMapping**
- **@DeleteMapping**
- **@PatchMapping**
- **@RequestMapping**(String URL, RequestMethod peticion)
- **@ResponseBody**

Configuración servicio REST con Spring - Entidad

```
@Entity
class Employee {

  private @Id @GeneratedValue Long id;
  private String name;
  private String role;

  Employee() {}

  Employee(String name, String role) {
    this.name = name;
    this.role = role;
  }
}
```

Configuración servicio REST con Spring - Controlador

```
@RestController
class EmployeeController {

private final EmployeeRepository repository;

EmployeeController(EmployeeRepository repository) {
    this.repository = repository;
  }

@GetMapping("/employees")
List<Employee> all() {
   return repository.findAll();
}
```

Configuración servicio REST con Spring – Controlador (II)

```
@PostMapping("/employees")
Employee newEmployee(@RequestBody Employee newEmployee) {
   return repository.save(newEmployee);
}

@GetMapping("/employees/{id}")
Employee one(@PathVariable Long id) {
   return repository.findById(id)
      .orElseThrow(() -> new EmployeeNotFoundException(id));
}
```

@RequestMapping y @ResponseBody

```
@Controller
public class SampleController {

    @Autowired
    private SampleService sampleService;

    public SampleController(SampleService sampleService) {
        this.sampleService = sampleService;
    }

    @RequestMapping(value = "/welcome/{userName}", method = RequestMethod.GET)
    @ResponseBody
    public String welcome(
            @PathVariable("userName") String userName
    )
    {
        return sampleService.welcome(userName);
    }
}
```