Microservices with Spring Boot + Spring Data

Using Spring Boot and Spring Data to quick develop HATEOAS microservices

Bernardo Silva

Terminology

- What is CRUD?
- What is REST?
- What is Spring?
- What is HATEOAS?
- What is Microservice?
- So… hands on!

What is CRUD?

- *"In computer programming, create, read, update and delete (as an acronym CRUD or possibly a Backronym) (Sometimes called SCRUD with an "S" for Search) are the four basic functions of persistent storage."*

- http://en.wikipedia.org/wiki/Create,_read,_update_and_delete

What is REST?

- "*Representational state transfer (REST) is an abstraction of the architecture of the World Wide Web; more precisely, REST is an architectural style consisting of a coordinated set of architectural constraints applied to components, connectors, and data elements, within a distributed hypermedia system.*"
- http://en.wikipedia.org/wiki/REST

Basic CRUD with REST

Operation	HTTP / REST
Create	PUT / **POST**
Read (Retrieve)	**GET**
Update (Modify)	**PUT** / PATCH
Delete (Destroy)	**DELETE**

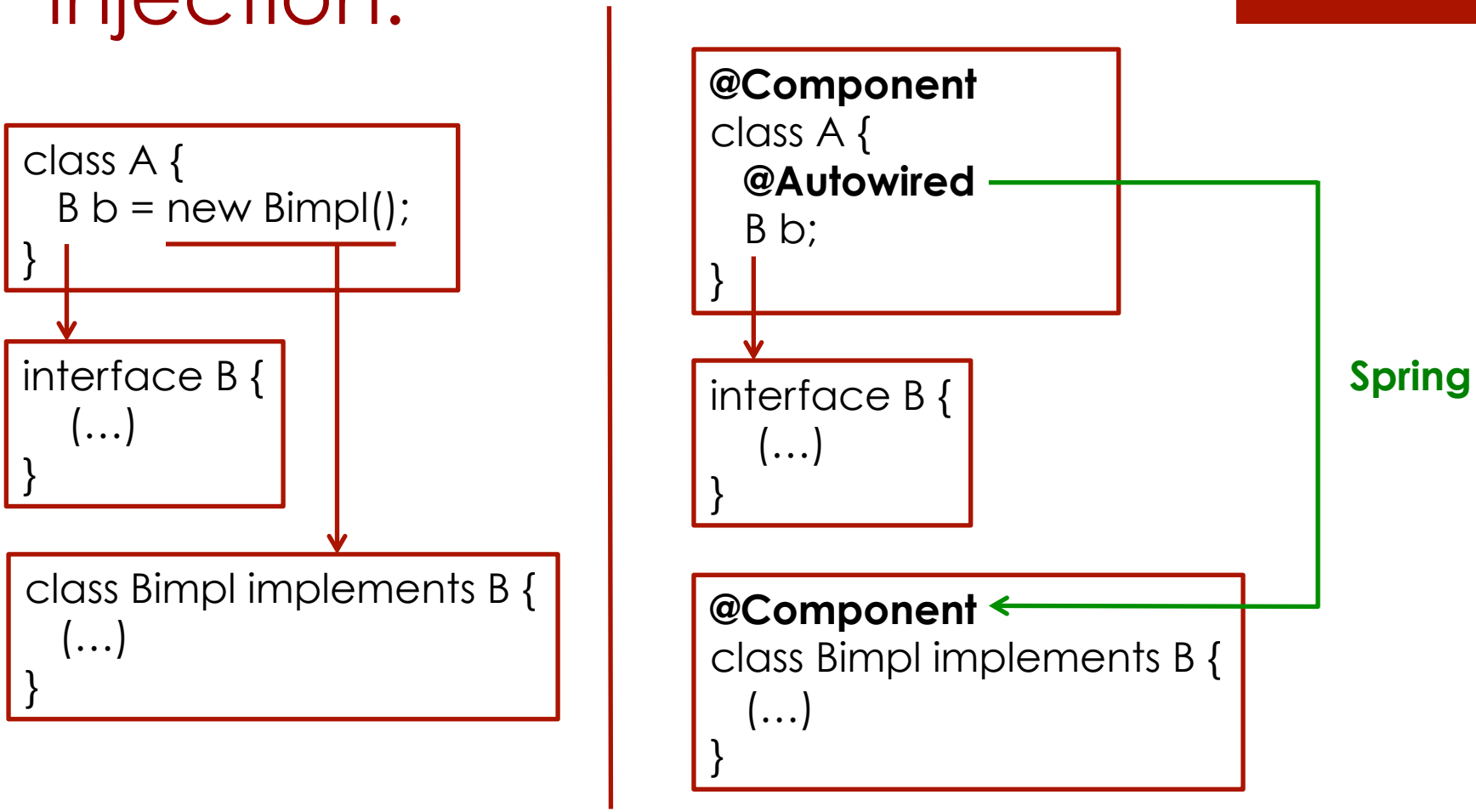
What is Spring?

- "*The Spring Framework is an open source application framework and inversion of control container for the Java platform.*"

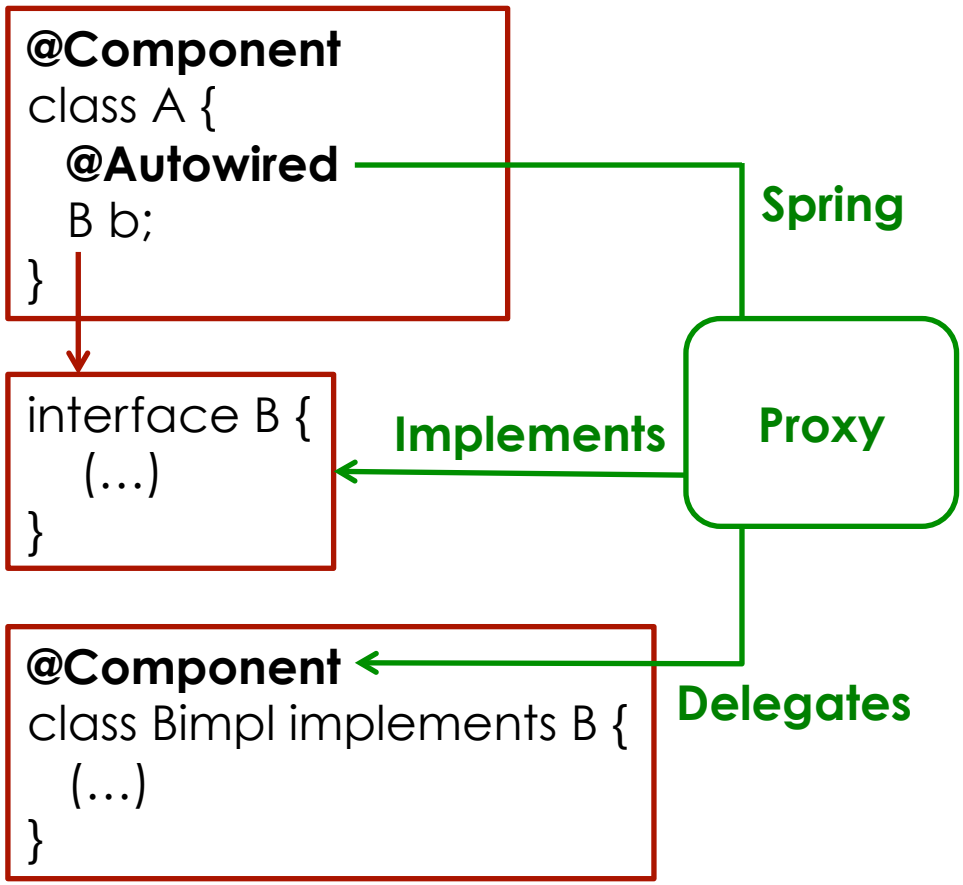
➢ http://en.wikipedia.org/wiki/Spring_Framework

- Inversion of control container / dependency injection
- Enrichment and proxying

Inversion of control container / dependency injection:



Enrichment and proxying:



What is HATEOAS?

- "*HATEOAS, an abbreviation for Hypermedia as the Engine of Application State, is a constraint of the REST application architecture that distinguishes it from most other network application architectures.*"

➢ http://en.wikipedia.org/wiki/HATEOAS

HATEOAS samples

XML

```
<person xmlns:atom="http://www.w3.org/2005/Atom">
  <firstname>Dave</firstname>
  <lastname>Matthews</lastname>
  <links>
    <atom:link rel="self" href="http://myhost/people/1" />
  </links>
</person>
```

JSON

```
{ firstname : "Dave",
  lastname : "Matthews",
  links : [ { rel : "self", href : "http://myhost/people/1" } ] }
```

What is Microservice?

- *"The term "Microservice Architecture" has sprung up over the last few years to describe a particular way of designing software applications as suites of independently deployable services."*

➢ http://martinfowler.com/articles/microservices.html

Microservices Reference

- The “ID” on a HATEOAS system is mainly an **URI**
  - Example: If you create a “teacher”, his “id” will be “**http://localhost:8080/teacher/1**” instead of just “**1**”.

- Then, each Microservice can **reference** other services entities.

So… hands on!

1) Using Gradle, you create a simple Spring Boot project:

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

2) You create a basic Spring Boot main class (it can be converted to a WAR later).

3) Then you boot doing:

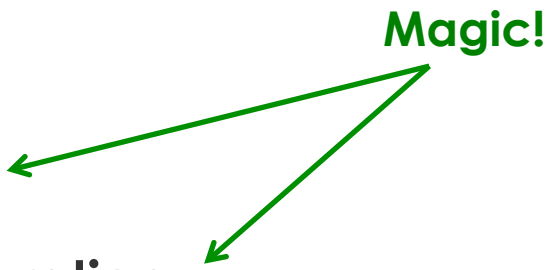
- **gradle bootRun**

build.gradle

- Buildscript dependencies:
  - **classpath("org.springframework.boot:spring-boot-gradle-plugin:1.1.5.RELEASE")**
- Body:
  - **apply plugin: 'spring-boot'**
- Dependencies:
  - **compile("org.springframework.boot:spring-boot-starter-web")**

Application.java

- Annotations:
  - **@Configuration**
  - **@ComponentScan**
  - **@EnableAutoConfiguration**

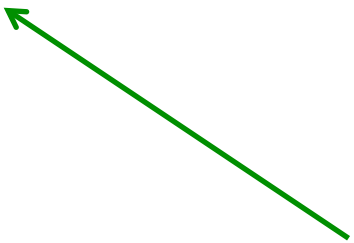


- Main method (unnecessary for WAR):
  - **SpringApplication.run(Application.class, args);**

Adding Hibernate (ORM)

- build.gradle (dependencies):

- **compile("org.springframework.boot:spring-boot-starter-data-jpa")**
- **compile("com.h2database:h2:1.3.176")**
- **compile("org.projectlombok:lombok:1.14.4")**

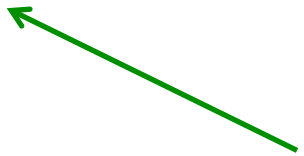


Just to make Entities more dumb

Adding Spring Data REST

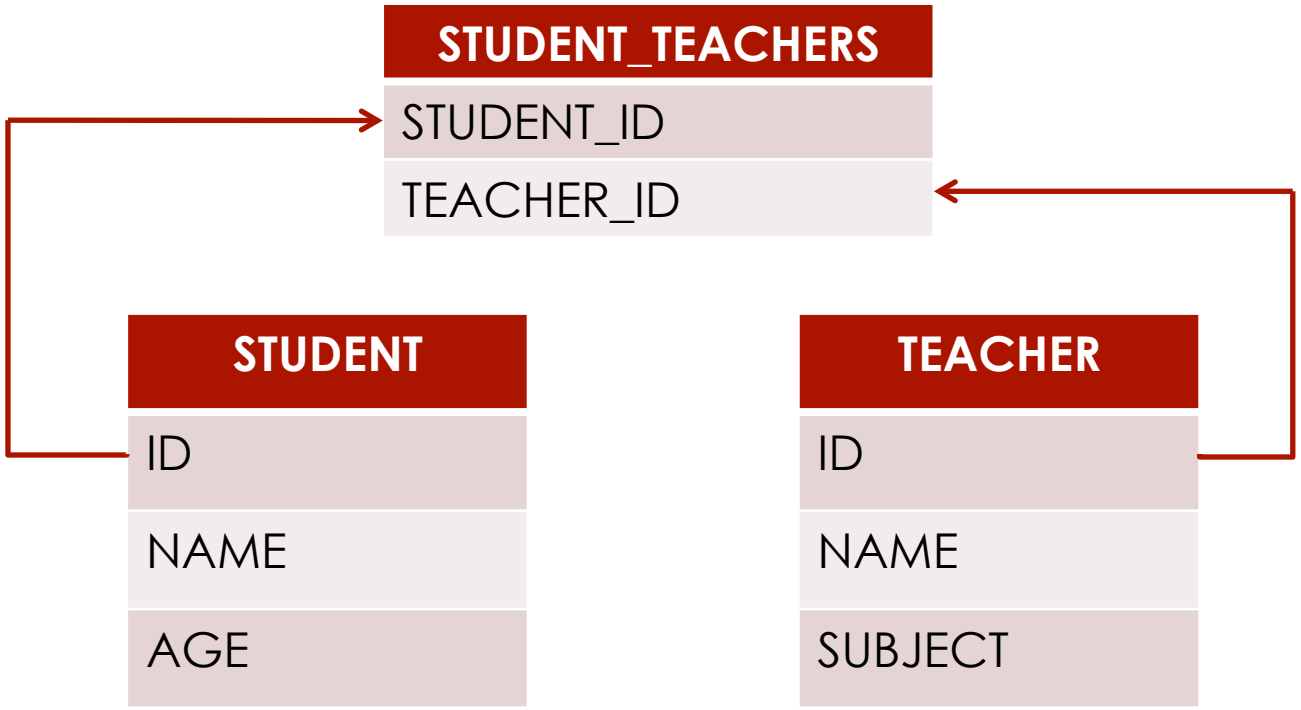
- build.gradle (dependencies):
  - **compile("org.springframework.data:spring-data-rest-webmvc")**

- Application.java:
  - **@Import(RepositoryRestMvcConfiguration.class)**



SUPER Magic!

Teacher and Student DDL



Adding “Teacher” entity

```
@Data
@Entity
@EqualsAndHashCode(exclude={"students"})
public class Teacher {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;
    private String subject;

    @ManyToMany
    @JoinTable(name = "STUDENT_TEACHERS",
            joinColumns = { @JoinColumn(name = "TEACHER_ID")},
            inverseJoinColumns = { @JoinColumn(name = "STUDENT_ID")})
    private Set<Student> students;
}
```

Adding “Student” entity

```
@Data
@Entity
@EqualsAndHashCode(exclude={"teachers"})
public class Student {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;
    private Integer age;

    @ManyToMany
    @JoinTable(name = "STUDENT_TEACHERS",
            joinColumns = { @JoinColumn(name = "STUDENT_ID")},
            inverseJoinColumns = { @JoinColumn(name = "TEACHER_ID")})
    private Set<Teacher> teachers;
}
```

Adding REST repositories

StudentRepository.java

```
@RepositoryRestResource(
        collectionResourceRel = "student", path = "student")
public interface StudentRepository extends
        PagingAndSortingRepository<Student, Long> {
}
```

TeacherRepository.java

```
@RepositoryRestResource(
        collectionResourceRel = "teacher", path = "teacher")
public interface TeacherRepository extends
        PagingAndSortingRepository<Teacher, Long> {
}
```

Run and be amazed!

- Try to get the root service description:
  - **GET**: http://localhost:8080/
- Try to create two students, two teachers and associate them!
  - **POST**: http://localhost:8080/student
- Use "PATCH" with "**text/uri-list**" to link entities
- Get JSON Schema of the entity:
  - **GET**: http://localhost:8080/student/schema
  - **Accept**: application/schema+json

Examples:

- **POST**: http://localhost:8080/student
  - { "name": "John", "age": 18 }
  - { "name": "Jorge", "age": 19 }
- **POST**: http://localhost:8080/teacher
  - { "name": "James", "subject": "Math" }
  - { "name": "Bob", "subject": "Biology" }
- **PUT/PATCH**: http://localhost:8080/student/1/teachers
  - Content-Type: **text/uri-list**

    http://localhost:8080/teacher/1

    http://localhost:8080/teacher/2
- **GET**: http://localhost:8080/teacher/1/students

QUESTIONS!

bernardo.silva@gmail.com
bernardo.silva@rackspace.com