

# Spring Boot

UA.DETI.IES

# Main topics

---

- ❖ Spring Boot
  - Dependencies, auto-configuration and runtime
- ❖ Spring MVC Architecture
- ❖ Spring WebFlux Architecture
- ❖ Java Persistence
- ❖ Spring Data
- ❖ Spring Data JPA
- ❖ Spring Data <others>

# What is Spring?

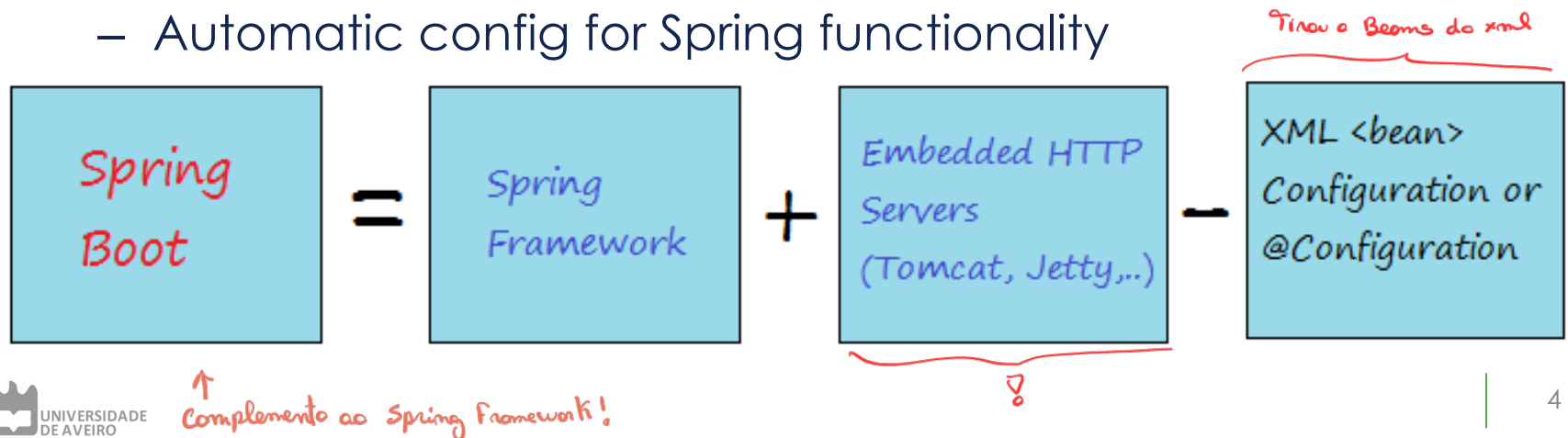
---

- ❖ Simply put, the Spring framework provides comprehensive **infrastructure support for developing Java applications**.
- ❖ It is packed with some nice features like Dependency Injection and out of the box modules like:
  - Spring JDBC
  - Spring MVC
  - Spring Security
  - Spring AOP
  - Spring ORM
  - Spring Test
- ❖ These modules **can reduce the development time** of an application.
  - For example, in the early days of Java web development, we needed to write a lot of code to insert a record into a data source.
  - But by using the JdbcTemplate of the Spring JDBC module we can reduce it to a few lines of code with only a few configurations.

<https://www.baeldung.com/spring-vs-spring-boot>

# What is Spring Boot?

- ❖ Extension of the Spring framework
  - that eliminated (even more) the boilerplate configurations required for setting up a Spring application.
  - It takes an opinionated view of the Spring platform, for a faster and more efficient development eco-system.
- ❖ Some features:
  - Opinionated 'starter' dependencies to simplify build and application configuration
  - Embedded server to avoid complexity
  - Automatic config for Spring functionality



# Spring Boot Main Goals

---

## ❖ Reducing development time

- as also Unit Test and Integration Test time
- to ease the development of production ready web applications very easily compared to existing Spring Framework, which really takes more time.

## ❖ Avoiding completely XML Configuration

## ❖ Providing **simple Annotation** (based on Spring' ones)

## ❖ Avoiding writing lots of import statements

## ❖ **Opinionated development** approach

- To provide some defaults to quick start new projects within no time



# Spring Boot Main Components

- ❖ **Spring Initilizr** - Web Interface to quick start the development of Spring Boot Applications.
- ❖ **Starters** *← dependências injetadas* - combine a group of common or related dependencies into single dependency
  - allow to add jars in the classpath
- ❖ **AutoConfigurator** - reduce the Spring Configuration
  - attempts to automatically configure the Spring application based on the jar dependencies
- ❖ **CLI** *← testes nas aplicações* - run and test Spring Boot applications from command prompt
  - groovy
- ❖ **Actuator** – provides EndPoints and Metrics
  - E.g., `http://localhost:8080/actuator/health`
    - `{"status":"UP"}`

# Spring Initializr <https://start.spring.io>



## Project

☐ Gradle - Groovy ☐ Gradle - Kotlin

☒ Maven

## Language

☒ Java ☐ Kotlin ☐ Groovy

## Spring Boot

☐ 3.2.0 (SNAPSHOT) ☐ 3.2.0 (RC2) ☐ 3.1.6 (SNAPSHOT) ☒ 3.1.5

☐ 3.0.13 (SNAPSHOT) ☐ 3.0.12 ☐ 2.7.18 (SNAPSHOT) ☐ 2.7.17

## Project Metadata

Group

Artifact

Name

Description

Package name

Packaging ☒ Jar ☐ War

## Dependencies

ADD DEPENDENCIES... ⌘ + B

No dependency selected

GENERATE ⌘ + ↵

EXPLORE CTRL + SPACE

SHARE...

# Spring Initializr

demo.zip

.gitignore

.mvn

HELP.md

mvnw

mvnw.cmd

**pom.xml**

src

DOWNLOAD

COPY

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
4   <modelVersion>4.0.0</modelVersion>
5   <parent>
6     <groupId>org.springframework.boot</groupId>
7     <artifactId>spring-boot-starter-parent</artifactId>
8     <version>3.1.5</version>
9     <relativePath/> <!-- lookup parent from repository -->
10  </parent>
11  <groupId>com.example</groupId>
12  <artifactId>demo</artifactId>
13  <version>0.0.1</version>
14  <description>demo</description>
15  <properties>
16    <java.version>1.8</java.version>
17  </properties>
18  <dependencies>
19    <dependency>
20      <groupId>org.springframework.boot</groupId>
21      <artifactId>spring-boot-starter</artifactId>
22    </dependency>
23  </dependencies>
```

.gitignore

.mvn

HELP.md

mvnw

mvnw.cmd

pom.xml

src

main

java

com

example

demo

**DemoApplication.java**

```
1 package com.example.demo;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5
6 @SpringBootApplication
7 public class DemoApplication {
8
9     public static void main(String[] args) {
10         SpringApplication.run(DemoApplication.class, args);
11     }
12
13 }
```



# Spring Boot main application

@SpringBootApplication

```
public class PayrollApplication {
```

```
    public static void main(String... args) {
```

```
        ApplicationContext ctx =
```

```
            SpringApplication.run(PayrollApplication.class, args);
```

```
    }
```

```
}
```

Enable component-scanning and auto-configuration

Bootstrap the application

# @SpringBootApplication

---

## ❖ Combines three other annotations:

### – @Configuration

- Designates a class as a configuration class using Spring's Java-based configuration

### – @ComponentScan

- Tells Spring Boot to scan the current package and its sub-packages in order to identify annotated classes (@Component, @Configuration, @Service, @Repository) and configure them as Spring beans

### – @EnableAutoConfiguration

- It enables the "magic" of Spring Boot auto-configuration avoiding writing the pages of XML configuration that would be required otherwise

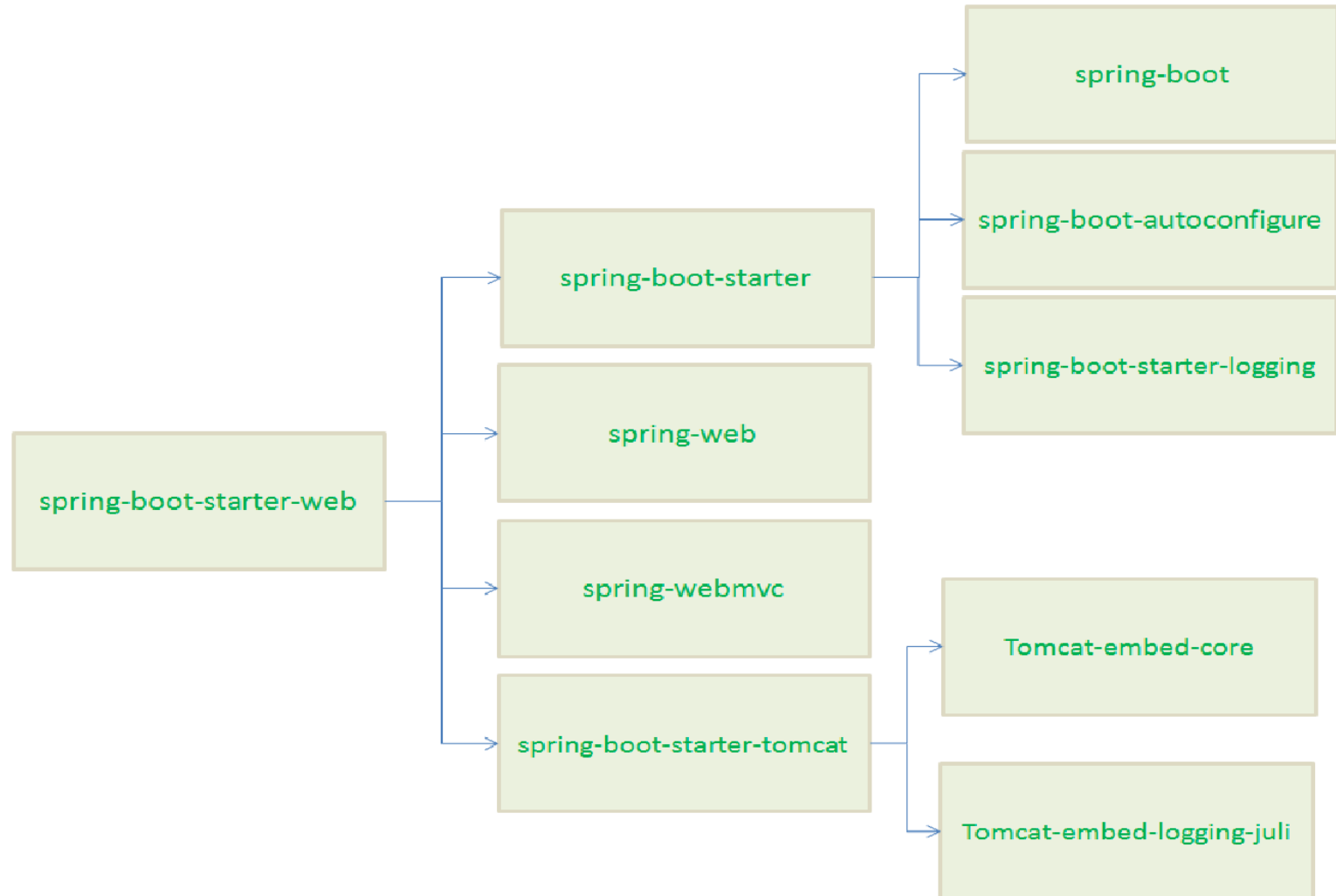
# Starters

---

- ❖ A set of pre-defined dependency descriptors
  - They combine a **group of common or related dependencies** into single dependencies
  - They avoid having to copy-paste loads of dependencies.
- ❖ For instance, to develop a Spring WebApplication with Tomcat we need to add the following minimal jar dependencies in pom.xml file
  - Spring core Jar file(spring-core-xx.jar)
  - Spring Web Jar file(spring-web-xx.jar)
  - Spring Web MVC Jar file(spring-webmvc-xx.jar)
  - Servlet Jar file(servlet-xx.jar)
- ❖ Instead, we just add "spring-boot-starter-web".

# Starters

↑ Configura tudo



# Starters

---

- ❖ Let's pretend for a moment that Spring Boot starter dependencies don't exist.
  - What kind of dependencies would you add to your build without Spring Boot?
  - Which Spring dependencies do you need to support Spring MVC?
  - Do you remember the group and artifact IDs for Thymeleaf?
  - Which version of Spring Data JPA should you use?
  - Are all of these compatible?
- ❖ All official starters follow a similar naming pattern
  - *spring-boot-starter-\**, where \* is a particular type of application.

<https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/#using-boot-starter>

# POM.xml – starter-parent

```
<?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
    https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>ies.spring</groupId>
  <artifactId>demo</artifactId>
  <version>0.0.1-SNAPSHOT</version>
```

default configurations for the application and a complete dependency tree to quickly build our Spring Boot project.

↗ starter...

```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>2.4.0</version>
  <relativePath /> <!-- lookup parent from repository -->
</parent>

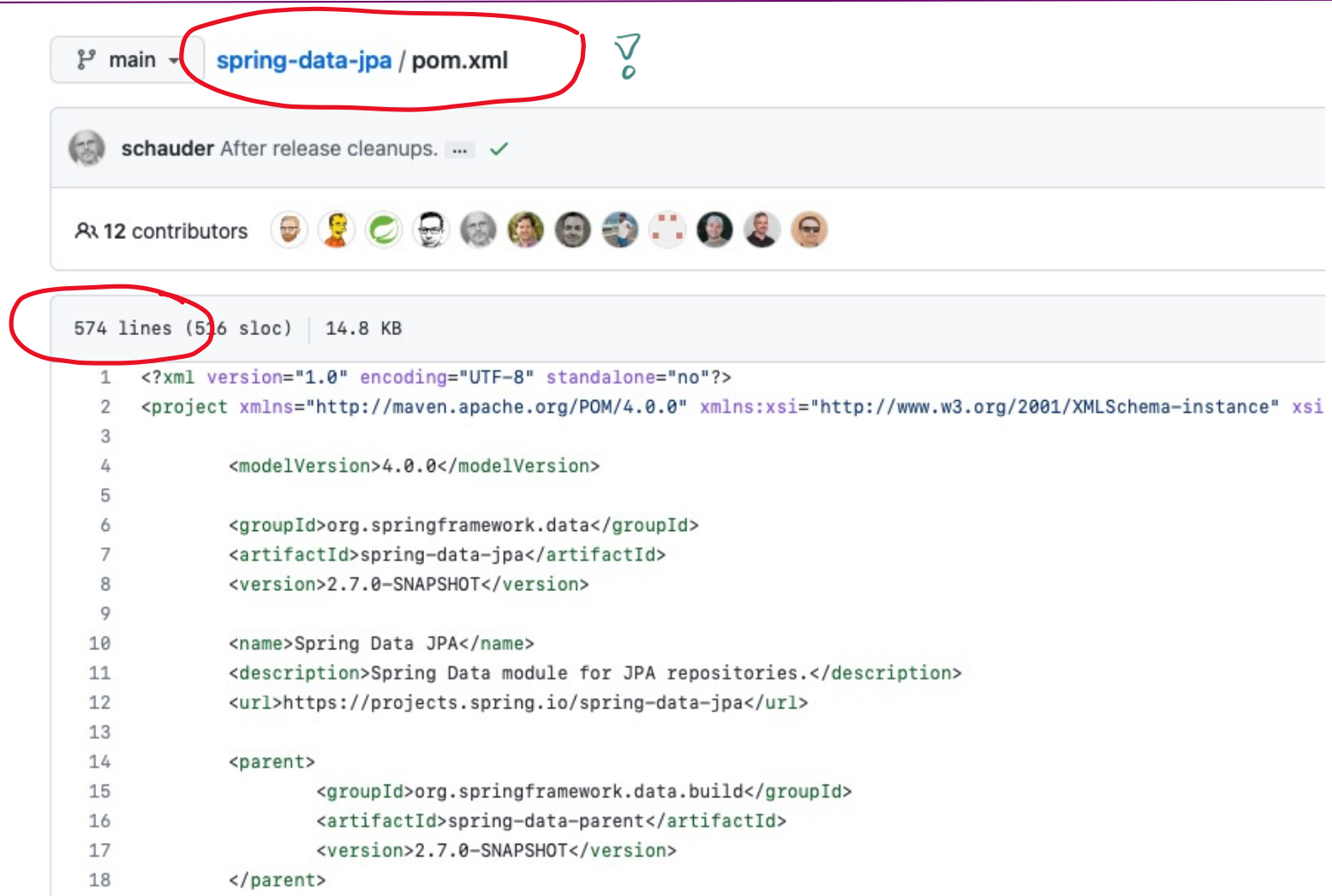
<!-- ... -->
```

# POM.xml – dependencies

```
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
  <dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <version>1.18.2</version>
    <scope>provided</scope>
  </dependency>
  ...
</dependencies>
<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
  </plugins>
</build>
</project>
```

No version

# github.com/spring-projects/spring-data-jpa/



main **spring-data-jpa / pom.xml** !

**schauder** After release cleanups. ... ✓

12 contributors

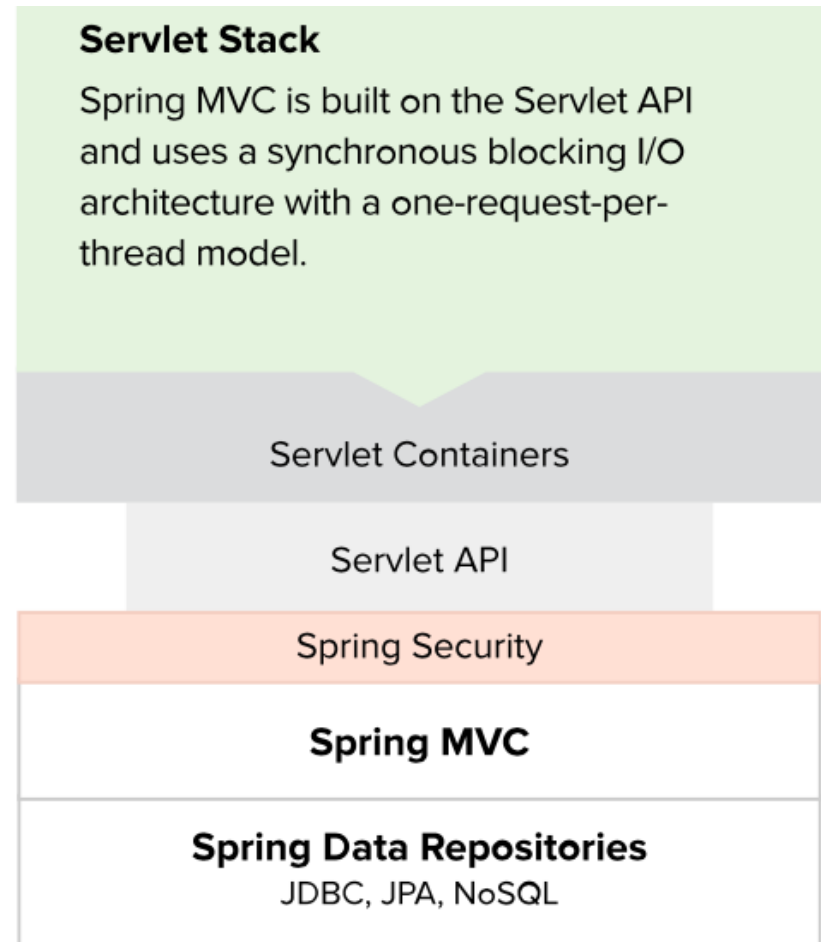
574 lines (516 sloc) | 14.8 KB

```
1 <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi
3
4     <modelVersion>4.0.0</modelVersion>
5
6     <groupId>org.springframework.data</groupId>
7     <artifactId>spring-data-jpa</artifactId>
8     <version>2.7.0-SNAPSHOT</version>
9
10    <name>Spring Data JPA</name>
11    <description>Spring Data module for JPA repositories.</description>
12    <url>https://projects.spring.io/spring-data-jpa</url>
13
14    <parent>
15        <groupId>org.springframework.data.build</groupId>
16        <artifactId>spring-data-parent</artifactId>
17        <version>2.7.0-SNAPSHOT</version>
18    </parent>
```



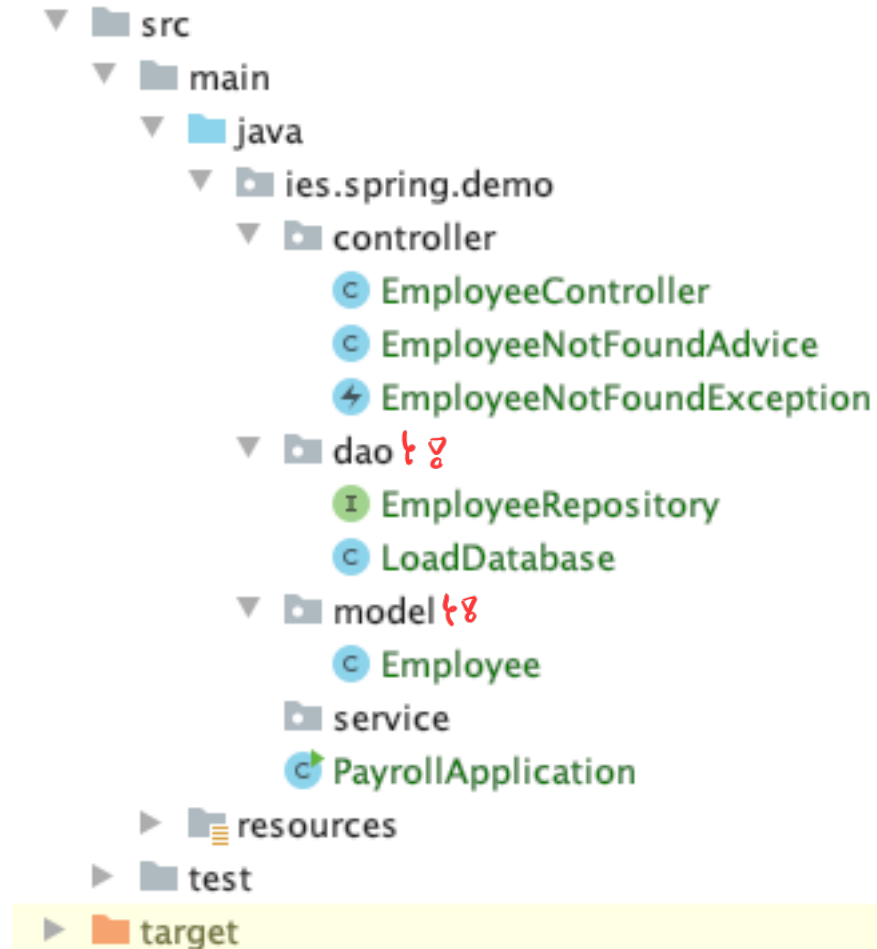
# The Spring Web MVC

- ❖ Spring MVC allows creating special **@Controller** or **@RestController** beans to handle incoming HTTP requests.
- ❖ Methods in the controller are mapped to HTTP by using **@RequestMapping** annotations. *↑ Services REST*



# The Spring Web MVC

## ❖ Project structure - example



# model.Employee.java

**@Entity**

**@Table(name = "employees")**

public class Employee {

private **@Id @GeneratedValue** Long id;

private String firstName;

private String lastName;

private String email;

public Employee() {  
}

public Employee(String firstName, String lastName, String email) {

    this.firstName = firstName;

    this.lastName = lastName;

    this.email = email;

}

// ...

↖ Não fazer os getters e setters !  
⊗ Posso não dar todos os setters e getters

**@Entity** denotes this is an entity object for the table name employees

The field id is the Primary Key and, hence, marked as **@Id**.

The field id is also marked with **@GeneratedValue**, which denotes that this is an Auto-Increment column.

# dao.EmployeeRepository

```
@Repository
public interface EmployeeRepository
    extends JpaRepository<Employee, Long> {
    public List<Employee> findByEmail(String email);
    // ... other methods
}
```

Where is this  
methods  
implemented?

↑ Não preciso de implementar o código...

Classe que se liga à classe  
que representa o modelo na base de dados! //

- ❖ The **JpaRepository** interface has two parameters:
  - the domain type that the repository will work with, and the type of its ID property (primary key).
- ❖ **EmployeeRepository** inherits methods for performing common persistence operations.
  - In addition, we may add other methods.
- ❖ The interface will be implemented automatically by Spring Boot at runtime when the application is started.

# Derived Query Methods

- ❖ Derived method names have two main parts separated by the first *By* keyword:
  - The first part – like *find* – is the introducer and the rest – like *ByName* – is the criteria.

```
public List<Employee> findByLastName(String lastname);  
    // or equivalents:  
public List<Employee> findByLastNameIs(String lastname);  
public List<Employee> findByLastNameEquals(String lastname);
```

ela faz toda a implementação  
↓  
✓

- ❖ Spring Data JPA supports ***find***, ***read***, ***query***, ***count*** and ***get***.

- we could have done *queryByName* and Spring Data would behave the same.

- ❖ We can also use *Distinct*, *First*, or *Top* to remove duplicates or limit our result set:

```
public List<Employee> findTop3ByFirstName(String firstname);
```

# Query methods: some examples

---

- findByLastnameAndFirstname
- findByLastnameOrFirstname
- findByStartDateBetween
- findByAgeLessThan
- findByStartDateAfter
- findByStartDateBefore
- findByAgeIsNull
- findByFirstnameLike
- findByFirstnameStartingWith
- findByAgeOrderByLastnameDesc
- findByAgeIn(Collection<Age> ages)
- findByFirstnameIgnoreCase

# Controller.EmployeeController

```
@RestController
@RequestMapping("/api/v1")
public class EmployeeController {
    @Autowired
    private EmployeeRepository employeeRepository;

    @GetMapping("/employees")
    public List<Employee> getAllEmployees(
        @RequestParam(required = false) String email,
        @RequestParam(required = false) String lastname) {
        if (email != null)
            return employeeRepository.findByEmail(email);
        else if (lastname != null)
            return employeeRepository.findByName(lastname);
        else
            return employeeRepository.findAll();
    }
    // ...
}
```

*Ten este cuidado para não quebrar serviços antigos*  
*⚠ IMPORTANTE*

# Controller.EmployeeController

---

```
// ...
```

```
@GetMapping("/employees/{id}")
public ResponseEntity<Employee>
    getEmployeeById(@PathVariable(value = "id") Long employeeId)
        throws ResourceNotFoundException {
    Employee employee = employeeRepository.findById(employeeId)
        .orElseThrow(() -> new ResourceNotFoundException("Employee not
found for this id :: " + employeeId));
    return ResponseEntity.ok().body(employee);
}
```

```
@PostMapping("/employees")
public Employee createEmployee(@Valid @RequestBody Employee employee) {
    return employeeRepository.save(employee);
}
```

```
// ...
```



# AutoConfigurator with H2



- ❖ To complete this configuration scenario, we may use an embedded H2 database

- ❖ In POM.xml

```
// ...  
<dependency>  
  <groupId>org.springframework.boot</groupId>  
  <artifactId>spring-boot-starter-data-jpa</artifactId>  
</dependency>  
<dependency>  
  <groupId>com.h2database</groupId>  
  <artifactId>h2</artifactId>  
  <scope>runtime</scope>  
</dependency>
```

- ❖ In application.properties file:

```
spring.datasource.url=jdbc:h2:mem:testdb  
spring.datasource.driverClassName=org.h2.Driver  
spring.datasource.username=user  
spring.datasource.password=password  
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
```

# AutoConfigurator with MySQL

## ❖ Using MySQL

```
create database demo
create user someuser ..
grant all on demo.* to someuser;
```

## ❖ In POM.xml

```
// ...
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <scope>runtime</scope>
</dependency>
// ...
```

## ❖ In application.properties file

```
spring.datasource.url=jdbc:mysql://localhost:3306/demo
spring.datasource.username=demo
spring.datasource.password=password
spring.jpa.hibernate.use-new-id-generator-mappings=false
spring.jpa.hibernate.ddl-auto = update
server.port=9000
```

## Running the application

- ❖ One of the biggest advantages of packaging an application as a jar and using an embedded HTTP server is that we can run it as:

```
~ java -jar target/demo-0.0.1-SNAPSHOT.jar
```

— or :

```
~ ./mvnw clean spring-boot:run }←
```

...

```

      _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
  ^ ^ / _ _ ' _ _ _ _ _ ( _ ) _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
( ( ) \ _ _ | ' _ | ' _ | | ' _ \ _ ' _ | \ _ _ _ _ _
 \ \ _ _ _ ) | _ | | | | | | | | ( _ | | ) ) ) )
  ' _ | _ _ _ | _ _ | _ | _ | _ | _ _ , | / / / / /
=====|_|=====| _ _ / = / _ / _ / _ /
:: Spring Boot ::                (v2.4.0)

```

```
2020-12-02 11:42:27.833 INFO 47912 --- [ restartedMain] n.g.s.s.Application
Starting Application on pc.lan with PID 47912
2020-12-02 11:42:27.889 INFO 47912 --- [ restartedMain] n.g.s.s.Application
No active profile set, falling back to default profiles: default
```

# Examples

```
~ curl http://localhost:9000/api/v1/employees
[]
~ curl -i -H "Content-Type:application/json" -d '{"firstName": "Maria",
"lastName": "Curia", "email": "mcuria@ua.pt"}'
http://localhost:9000/api/v1/employees
HTTP/1.1 200
Content-Type: application/json
Transfer-Encoding: chunked
Date: Wed, 02 Dec 2020 12:19:21 GMT
{"id":1,"firstName":"Maria","lastName":"Curia","email":"mcuria@ua.pt"}%
~ curl http://localhost:9000/api/v1/employees
[{"id":1,"firstName":"Maria","lastName":"Curia","email":"mcuria@ua.pt"}]

~ curl http://localhost:9000/api/v1/employees/1
{"id":1,"firstName":"Maria","lastName":"Curia","email":"mcuria@ua.pt"}

~ curl http://localhost:9000/api/v1/employees/33
{"timestamp":"2020-12-02T12:29:18.253+0000","message":"Employee not found
for this id :: 20","details":{"uri=/api/v1/employees/20"}}
```

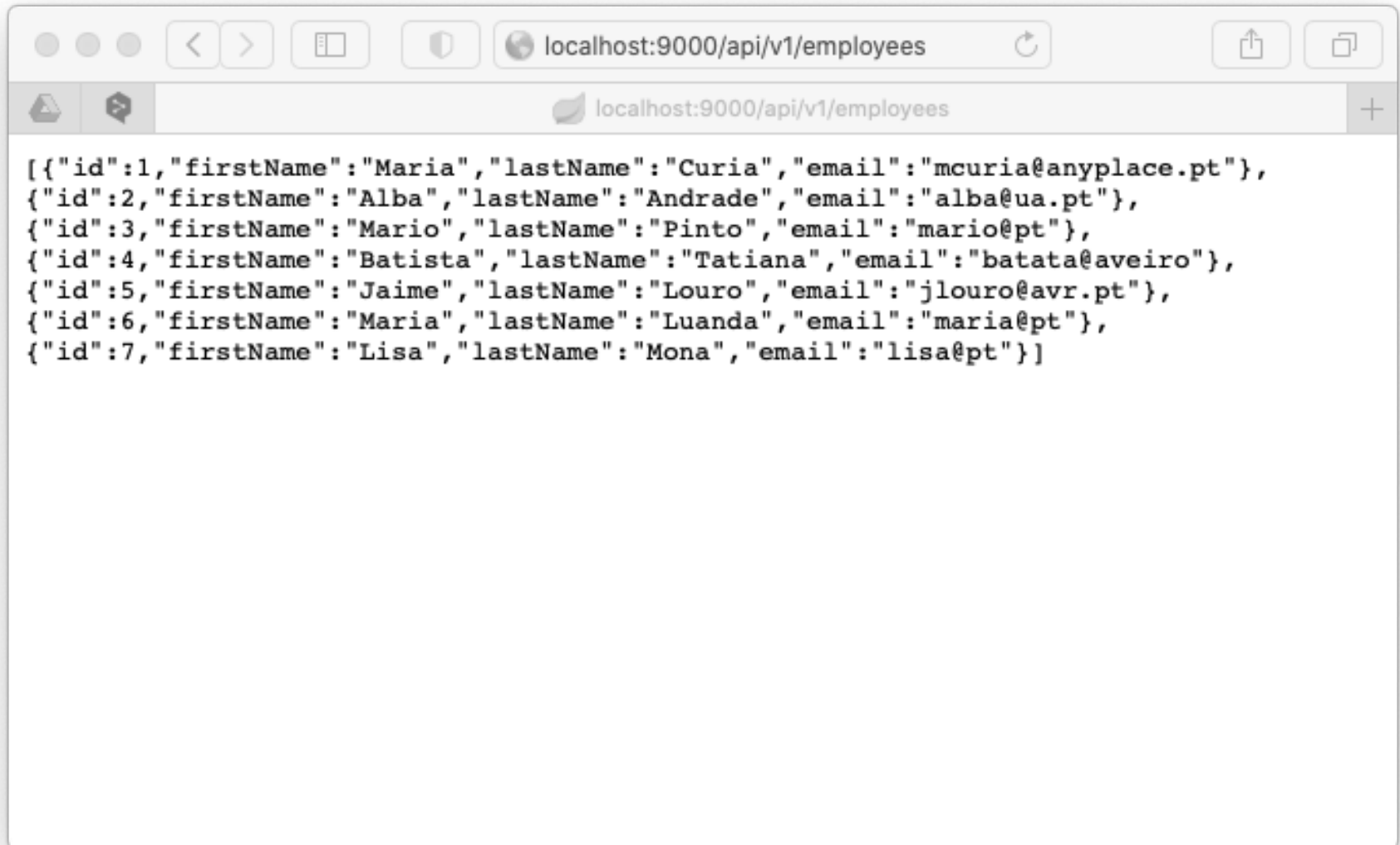
# Examples

```
~ curl -v localhost:9000/api/v1/employees
```

```
* Trying ::1...
* TCP_NODELAY set
* Connected to localhost (::1) port 9000 (#0)
> GET /api/v1/employees HTTP/1.1
> Host: localhost:9000
> User-Agent: curl/7.64.1
> Accept: */*
>
< HTTP/1.1 200
< Content-Type: application/json
< Transfer-Encoding: chunked
< Date: Wed, 02 Dec 2020 19:34:15 GMT
<
* Connection #0 to host localhost left intact
[{"id":1,"firstName":"Maria","lastName":"Curia","email":"mcuria@anyplace.pt"}, {"id":2,"
firstName":"Alba","lastName":"Andrade","email":"alba@ua.pt"}, {"id":3,"firstName":"Mario
","lastName":"Pinto","email":"mario@pt"}, {"id":4,"firstName":"Batista","lastName":"Tati
ana","email":"batata@aveiro"}, {"id":5,"firstName":"Jaime","lastName":"Louro","email":"j
louro@avr.pt"}, {"id":6,"firstName":"Maria","lastName":"Luanda","email":"maria@pt"}, {"id
":7,"firstName":"Lisa","lastName":"Mona","email":"lisa@pt"}]* Closing connection 0
```

# Examples

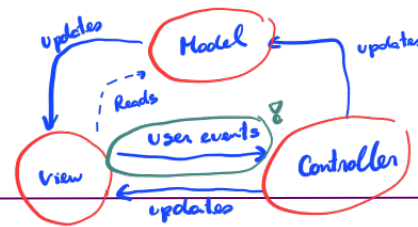
---



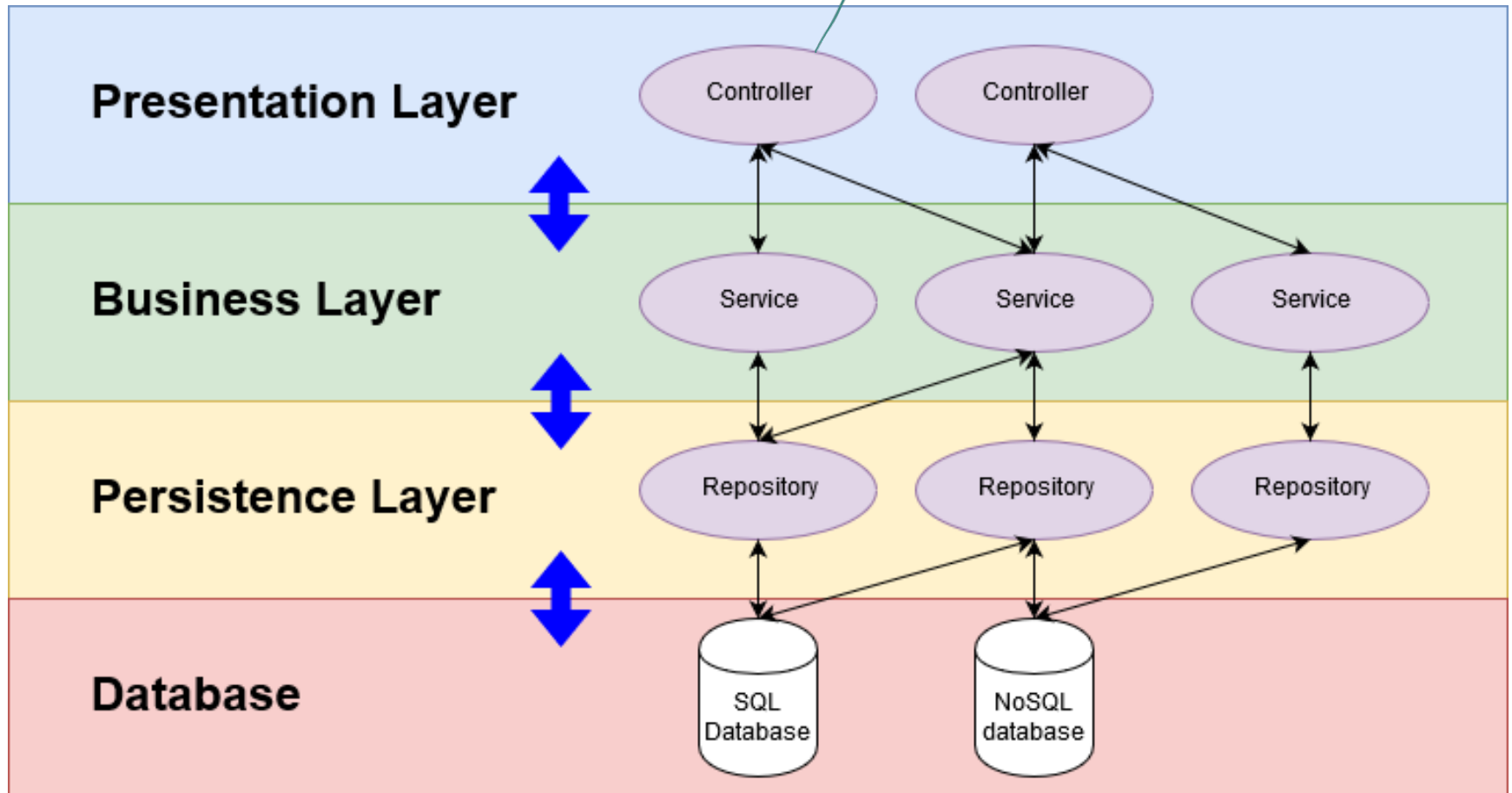
A screenshot of a web browser window. The address bar shows the URL `localhost:9000/api/v1/employees`. The page content displays a JSON array of seven employee objects. Each object contains `id`, `firstName`, `lastName`, and `email` fields. The browser interface includes standard navigation buttons and a tab bar at the top.

```
[{"id":1,"firstName":"Maria","lastName":"Curia","email":"mcuria@anyplace.pt"},
{"id":2,"firstName":"Alba","lastName":"Andrade","email":"alba@ua.pt"},
{"id":3,"firstName":"Mario","lastName":"Pinto","email":"mario@pt"},
{"id":4,"firstName":"Batista","lastName":"Tatiana","email":"batata@aveiro"},
{"id":5,"firstName":"Jaime","lastName":"Louro","email":"jlouro@avr.pt"},
{"id":6,"firstName":"Maria","lastName":"Luanda","email":"maria@pt"},
{"id":7,"firstName":"Lisa","lastName":"Mona","email":"lisa@pt"}]
```

# Data Persistence



Padrão MVC



<https://anchormen.nl/blog/big-data-services/spring-boot-tutorial/>

# Data Objects

← Dados do objeto

## ❖ DAO (Data Access Object)

- A structural pattern that isolate the application/business layer from the persistence layer using **an abstract API**.
- It is not a spring module in a strict sense, but rather conventions that should dictate you to write DAO, and to write them well.
- Example:

```
public interface Dao<T> {  
    Optional<T> get(long id);  
    List<T> getAll();  
    void save(T t);  
    void update(T t, String[] params);  
    void delete(T t);  
}
```

← Estrutura

## ❖ ORM (Object-relational mapping)

- The ORM package is related to the database access.
- It provides **integration layers for popular object-relational mapping APIs** (e.g., JDO, Hibernate).



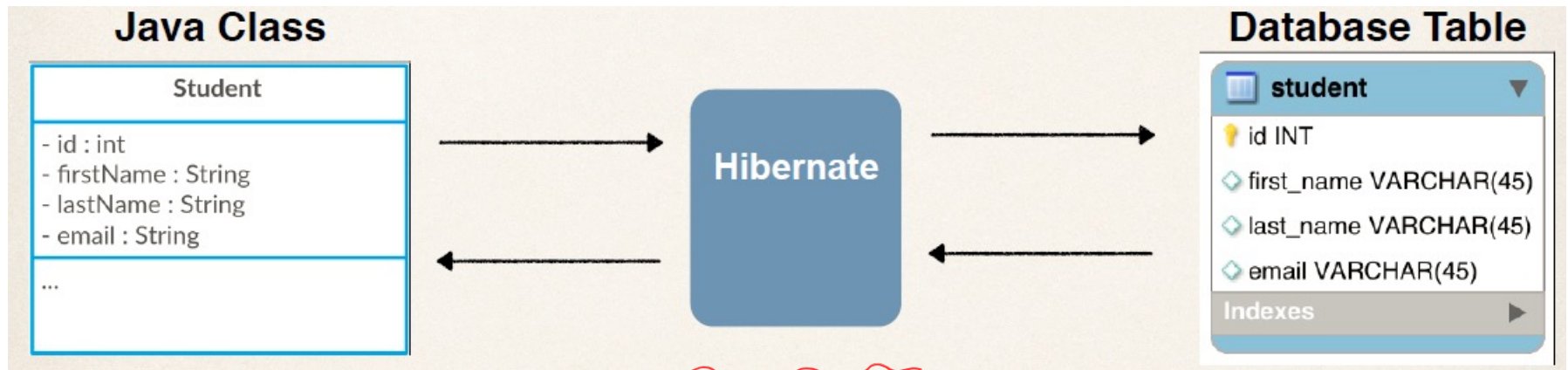
# Java/Jakarta Persistence API (JPA)

---

- ❖ The **Jakarta Persistence API (JPA)** – formerly Java Persistence API – is the standard specification for mapping Java objects to a relational database
  - It includes specifications, the entity and association mappings, the entity lifecycle management, and JPA's query capabilities
  - Mapping Java objects to database tables and vice versa is called Object-relational mapping (ORM).
- ❖ The **Java Persistence API (JPA)** is one possible approach to ORM.
  - Via JPA the developer can **map, store, update and retrieve data** from relational databases to Java objects and vice versa.
- ❖ Popular implementations are Hibernate, EclipseLink and Apache OpenJPA.

# Hibernate

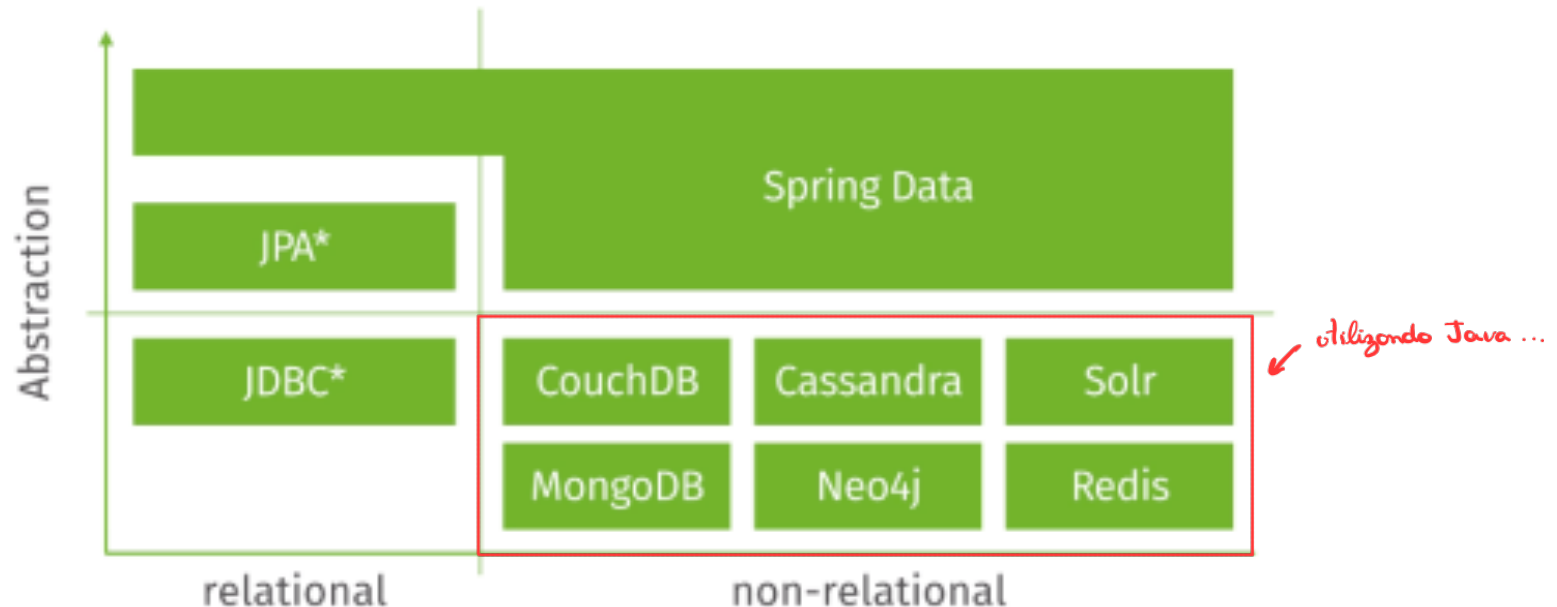
- ❖ Hibernate is a Java-based ORM tool
  - provides a framework for mapping application domain objects to the relational database tables and vice versa
- ❖ Hibernate provides a reference implementation of the Java/Jakarta Persistence API



# Spring Data Persistence

- ❖ An umbrella project having several sub-projects
  - Aiming to unify and ease the access to different kinds of persistence stores, from relational to NoSQL databases

## Spring Data



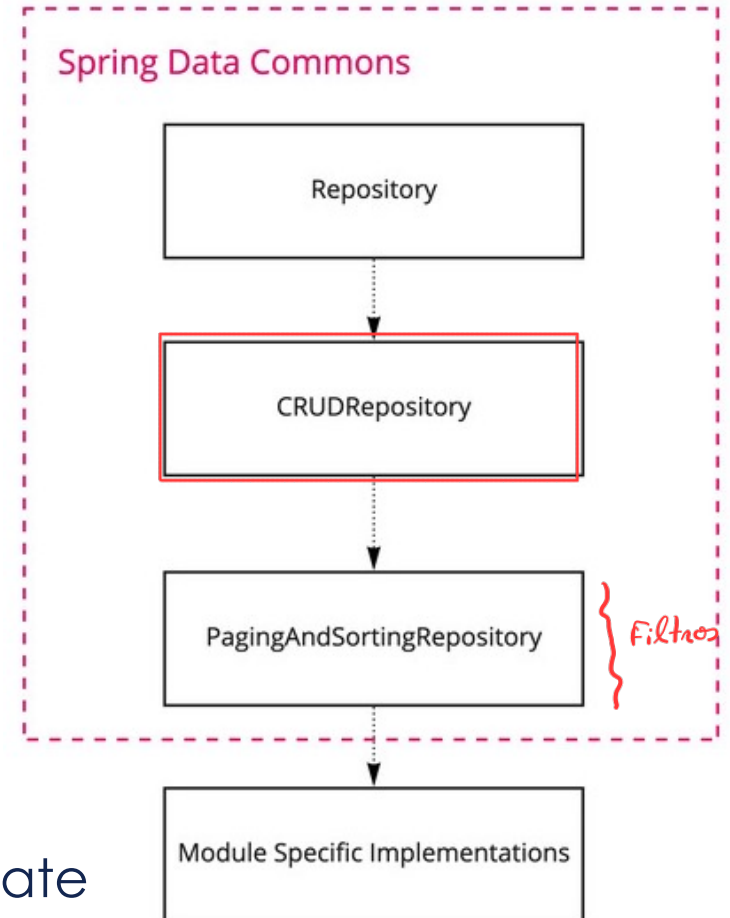
# Spring Data

## ❖ Main interfaces

- Repository, CRUDRepository, PagingAndSorting Repository

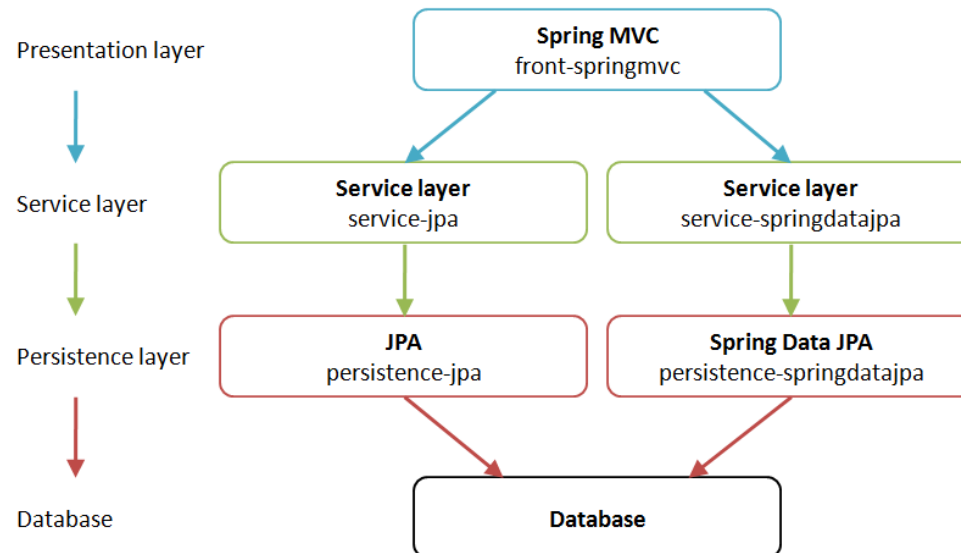
## ❖ Main annotations

- @Repository
  - a specialization of @Component
- @Id
- @Param
- @Transient
- @Transactional
- @CreatedBy, @LastModifiedBy,
- @CreatedDate, @LastModifiedDate



# Spring Data JPA

- ❖ An **abstraction** used to significantly reduce the amount of boilerplate code required to implement data access layers for various persistence stores.
- ❖ It adds its own features like a **no-code implementation of the repository pattern** and the **creation of database queries from method names**.
- ❖ It can also generate JPA queries on your behalf through method name conventions.



# Spring Data JPA

## ❖ JPA repositories tie to a particular JPA entity

@Repository

```
public interface UserRepository extends JpaRepository<User, Long> {}
```

## ❖ Main annotations

### – @Query

- With @Query, we can provide a JPQL implementation for a repository method:

```
@Query("SELECT COUNT(*) FROM Person p") long getPersonCount();
```

- Also, we can use named parameters:

```
@Query("FROM Person p WHERE p.name = :name") Person  
findByName(@Param("name") String name);
```

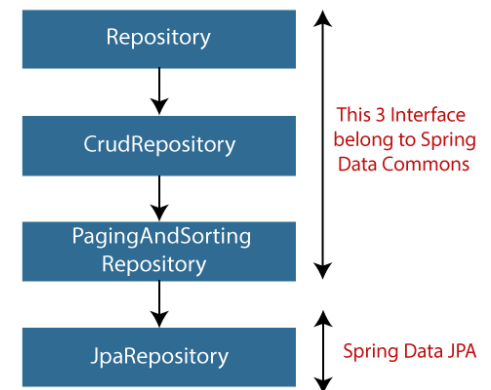
### – @Procedure

- call stored procedures from repositories.

### – @Lock

### – @Modifying

Spring Data Repository Interface



# Spring Data JPA with Hibernate

---

- ❖ Spring Boot configures Hibernate as the default JPA provider
  - To enable JPA in a Spring Boot application, we need the ***spring-boot-starter*** and ***spring-boot-starter-data-jpa*** dependencies
- ❖ Spring Boot can also auto-configure the ***dataSource*** bean, depending on the database we are using
  - For in-memory database (e.g. H2), Boot automatically configures the *dataSource*.
  - we only need to add the H2 dependency to the pom.xml file.

# @Entity

---

- ❖ Entities in JPA are nothing but POJOs representing data that can be persisted to the database
  - Assuming we have:

```
public class Customer {  
    private Long id;  
    private String username;  
    private String password;  
    private String full_name;  
    private Integer age;  
    ...  
}
```
  - We must ensure that the entity **has a no-arg constructor and a primary key**
  - Entity classes must not be declared final
- ❖ An entity represents a table stored in a database
  - Every instance of an entity represents a row in the table.



# @Entity

```
@Entity(name="customer")
@Table(name = "CUSTOMERS", schema = "CHAINS") // namespace
public class Customer {
    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    private Long id;

    private String username;
    @Column(nullable = false)
    private String password;
    @Column(name = "name", length=50, nullable = false)
    private String full_name;
    @Transient
    private Integer age; // not persistent (or static, final, transient)
    ...
}
```

↓ Tipo SQL

# Embeddable Classes

---

- ❖ User-defined classes that function as value types
  - As with other non-entity types, instances of an embeddable class can only be stored in the database as embedded objects, i.e. as part of a containing entity object.
- ❖ A class is declared as embeddable by marking it with the Embeddable annotation:

```
@Entity public class Company {  
    @Id @GeneratedValue  
    private Integer id;  
    private String name;  
    private String address;  
    @Embedded  
    private ContactPerson contactPerson;  
    // ...  
}
```

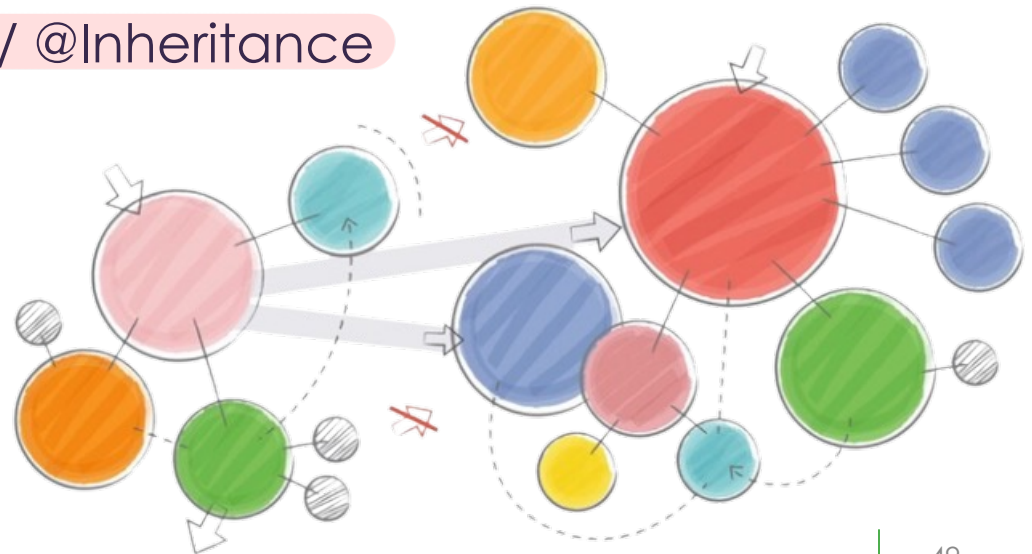
# Embeddable Classes

---

```
@Entity public class Company {
    @Id @GeneratedValue
    private Integer id;
    private String name;
    private String address;
    @Embedded
    @AttributeOverrides({
        @AttributeOverride( name = "firstName", column = @Column(name =
"contact_first_name")),
        @AttributeOverride( name = "lastName", column = @Column(name =
"contact_last_name")),
        @AttributeOverride( name = "phone", column = @Column(name =
"contact_phone"))
    })
    private ContactPerson contactPerson;
    // ...
}
```

# Relationships

- ❖ Every persistent field can be marked with one of the following annotations:
    - @OneToOne, @ManyToOne
      - for references of entity types
    - @OneToMany, @ManyToMany
      - for collections and maps of entity types
    - @MappedSuperClass / @Inheritance
      - for inherited types
- 



# @OneToMany / @ManyToOne

- ❖ The following entity classes demonstrate a bidirectional relationship:

```
@Entity  
class Customer {
```

```
...
```

```
    @OneToMany (mappedBy = "customer") // owned  
    Set<Order> orders;
```

```
}
```

```
@Entity  
class Order {
```

```
...
```

```
    @ManyToOne (optional = false) // owning  
    Customer customer;
```

```
...
```

```
}
```



Owned entity always maps to owning entity!

# @ManyToMany

```
@Entity
class Product {
    @Id
    @GeneratedValue (strategy = GenerationType.SEQUENCE)
    @Column (name = "pid")
    long prod_id;

    @Column (nullable = false)
    float price;

    @ManyToMany (mappedBy = "products", // owned
                fetch = FetchType.EAGER)
    Set<Order> orders;
}

@Entity
class Order {
    ...
    @ManyToMany (fetch = FetchType.EAGER) // owning
    @JoinTable (name = "OrderLines",
                joinColumns = @JoinColumn (name = "oid"),
                inverseJoinColumns = @JoinColumn (name = "pid"))
    Set<Product> products;
    ...
}
```



# Spring Data JPA @Query

- ❖ We can use the @Query annotation to execute both JPQL and native SQL queries

- SQL native – over *JDBC*

```
@Query( value = "SELECT * FROM USERS u WHERE u.status = 1",  
        nativeQuery = true)  
Collection<User> findAllActiveUsersNative();
```

- JPQL (*JPA Query Language*) – over *Hibernate*

```
@Query("SELECT u FROM User u WHERE u.status = 1")  
Collection<User> findAllActiveUsers();
```

```
@Query(value = "SELECT u FROM User u")  
List<User> findAllUsers(Sort sort);
```

Sorting

```
@Query(value = "SELECT u FROM User u ORDER BY id")  
Page<User> findAllUsersWithPagination(Pageable pageable);
```

Pagination

<https://www.baeldung.com/spring-data-jpa-query>

# JPA with MongoDB

---

## ❖ Define the model (as previously)

– e.g. Person

```
@Document(collection = "school")
public class Person {
    @Id
    private ObjectId id;
    private Integer ssn;
    @Indexed
    private String name;
}
```

## ❖ Adding Repository

```
@Repository
public interface PersonRepository
    extends MongoRepository<Person, String> {
    Person findByName(String name);
}
```



# JPA with MongoDB

---

## ❖ Adding connection info in application.properties

```
spring.data.mongodb.host=[host]
spring.data.mongodb.port=[port]
spring.data.mongodb.authentication-database=[authentication_database]
spring.data.mongodb.username=[username]
spring.data.mongodb.password=[password]
spring.data.mongodb.database=some_database
```

## ❖ Create the REST Controller

## ❖ Querying

```
@Query("{ 'name' : ?0' }") // in @Repository
Employee findByName(String name);
```

```
Query query = new Query(); // application
query.addCriteria(Criteria.where("age").lt(50).gt(20));
List<User> users = mongoTemplate.find(query, User.class);
```

# Spring Data Mongo Annotations

---

## ❖ @Document

- Mongo's equivalent of @Entity in JPA.

## ❖ @Field

```
@Document
class User {
    // ...
    @Field("email")
    String emailAddress;
    // ...
}
```

## ❖ @Query

```
@Query("{ 'name' : ?0 }")
List<User> findUsersByName(String name);
```

# Spring Data - summary

---

- ❖ Spring Data consists of many independent projects
  - Spring Data Commons
  - Spring Data JPA
  - Spring Data KeyValue
  - Spring Data LDAP
  - Spring Data MongoDB
  - Spring Data Redis
  - Spring Data REST
  - Spring Data for Apache Cassandra
  - Spring Data for Apache Solr
  - Spring Data Couchbase (community module)
  - Spring Data Elasticsearch (community module)
  - Spring Data Neo4j (community module)

# References

---

- ❖ <https://spring.io/projects/spring-boot>
- ❖ <https://spring.io/projects/spring-data>
- ❖ <https://www.baeldung.com/spring-tutorial>
- ❖ <https://www.baeldung.com/persistence-with-spring-series>
- ❖ <https://www.edureka.co/blog/spring-tutorial/>
- ❖ ... *and many others*