



# Introdução a Web Services REST com Spring Framework

*Parte 3 → Integração com Banco de Dados*

Prof. Me. Jorge Luís Gregório

[www.jlgregorio.com.br](http://www.jlgregorio.com.br)



# Agenda

- Iniciando o serviço do MySQL
- Configurando as dependências Maven
- Introdução ao Hibernate
  - Entidades Persistentes e Anotações
  - Repository
- Criando um CRUD (banco de dados)
- Testando com o Postman

A large, stylized green leaf graphic is positioned on the left side of the slide, partially overlapping the title box. It has a light green fill and a darker green outline, with a central vein and several smaller veins branching off.

# Iniciando o Serviço do MySQL

# Iniciando o serviço e criando o banco de dados

- Inicie o serviço do MySQL;
- Abra o terminal e digite o seguinte comando para entrar no MySQL:

```
mysql -u root
```

- Agora digite o seguinte comando para criar um banco de dados baseado no collate **latin1**:

```
create database spring_crud charset latin1 collate latin1_swedish_ci;
```


- Agora vamos usar o banco de dados, digite o seguinte comando:

```
use spring_crud;
```

```
MariaDB [(none)]> use spring_crud;  
Database changed  
MariaDB [spring_crud]> |
```

# Heidi SQL

- Se preferir usar uma ferramenta para administrar o banco de dados, sugerimos o **HEIDI SQL**, que é mais leve, mais simples e mais rápido;
- Acesse: <https://www.heidisql.com/>

 **HeidiSQL**

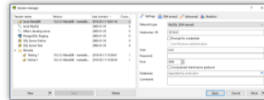

Home Downloads Images Forum Donate Bugtracker Help

---

### What's this?

HeidiSQL is free software, and has the aim to be easy to learn. "Heidi" lets you see and edit data and structures from computers running one of the database systems MariaDB, MySQL, Microsoft SQL, PostgreSQL and SQLite. Invented in 2002 by Ansgar, HeidiSQL belongs to the most popular tools for MariaDB and MySQL worldwide.

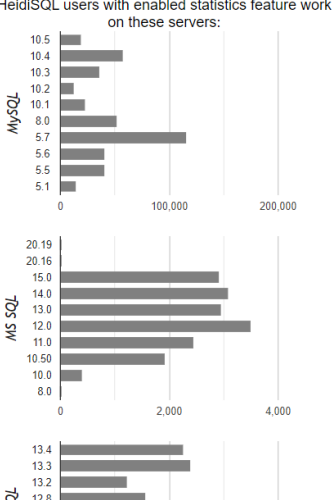
Download HeidiSQL, read further about [features](#), take part in [discussions](#) or see some [screenshots](#).



### News

30 May [HeidiSQL 11.3 with syntax highlighting in grid text editor](#)  
Get it from the download page Still need 32bit support? I am planning to drop the 32bit releases in a future release, for keeping the installer package small, and to decrease deployment efforts. ...

HeidiSQL users with enabled statistics feature work on these servers:



Version	Users (approx)
10.5	10,000
10.4	15,000
10.3	10,000
10.2	10,000
10.1	10,000
8.0	10,000
5.7	150,000
5.6	10,000
5.5	10,000
5.1	10,000

Version	Users (approx)
20.19	1,000
20.16	1,000
15.0	1,000
14.0	1,000
13.0	1,000
12.0	1,000
11.0	1,000
10.50	1,000
10.0	1,000
8.0	1,000

Version	Users (approx)
13.4	1,000
13.3	1,000
13.2	1,000
12.8	1,000

# Alterando o arquivo *properties.xml*

- Abra o arquivo *src/main/resources/application.properties* (se ele não existir, crie!);
- O conteúdo deste arquivo é:

```
spring.datasource.driverClassName=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/spring_crud?useTimeZone=true&serverTimeZone=UTC
spring.datasource.username=root
spring.datasource.password=mysqldba
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5InnoDBDialect
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=false
```

A large, stylized green leaf graphic is positioned on the left side of the slide, partially overlapping the title box. It has a light green fill and a darker green outline, with a central vein and several smaller veins branching out.

# Configurando as dependências

# Alterando o arquivo pom.xml

- É necessário adicionar duas dependências ao projeto:
  - **Spring Data JPA** – Suporte às classes para ORM
  - **MySQL Connector** – Driver de conexão do MySQL
  - **JavaX Persiste** – anotações para entidades persistentes - Hibernate;

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

<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
</dependency>

  <dependency>

<groupId>org.apache.aries.jpa.javax.persistence</groupId>
  <artifactId>javax.persistence_2.1</artifactId>
  <version>2.7.3</version>
</dependency>

<dependency>
  <groupId>org.springframework.data</groupId>
  <artifactId>spring-data-jpa</artifactId>
</dependency>
```



A large, stylized green leaf graphic is positioned on the left side of the slide, partially overlapping the central text box. The leaf is composed of several rounded, overlapping shapes in a light green color, with a thin green outline.

**Hibernate**

# O que é Hibernate ORM

- Hibernate ORM é um *framework* de Mapeamento Objeto-Relacional que implementa a especificação JPA – Java Persistence API;
- Ele é baseado no conceito de annotations e suporta todos os conceitos da linguagem Orientada a Objetos: herança, polimorfismo, associação, etc;
- Oferece diversas estratégias para criar, modificar e recuperar dados de um banco de dados relacional;
- É altamente escalável, configurável e extensível;
- Mais informações: <https://hibernate.org/orm/>

# Criando e Anotando a Entidade

- No pacote principal da aplicação, crie um pacote model;
- Crie uma classe java Person;
- Adicione os campos a seguir;
- Gere os métodos construtores, getters e setters;
- Veja as anotações →
- Note que essas anotações pertencem ao pacote ***javax.persistence***;
- Caso ele não importe corretamente, é possível adicionar dependências Maven pela sugestão de Código do IntelliJ IDEA

```
@Entity
@Table(name="person")
public class Person implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long id;

    @Column(name = "first_name", nullable = false, length = 40)
    private String firstName;

    @Column(name = "last_name", nullable = false, length = 40)
    private String lastName;

    @Column(name = "profession", nullable = false, length = 40)
    private String profession;
```

# Criando o Repository

- O padrão de projeto **Repository** é uma maneira de separar a lógica de negócio das operações de banco de dados;
- Um **Repository** faz a mediação entre o domínio da aplicação e a camada ORM, armazenando os dados de domínio em memória, como uma coleção de objetos;
- Isso traz uma série de vantagens, dentre os quais destaca-se a possibilidade de alterar regras de negócio sem necessariamente alterar a forma como os dados são persistidos ou recuperados do banco de dados;
- Mais informações: <https://martinfowler.com/eaaCatalog/repository.html>

# Criando o Repository

- No pacote principal da aplicação, crie um pacote chamado ***repository***;
- Nesse pacote, crie uma interface Java com o nome **PersonRepository**;
- O código dessa classe está ao lado;
- Esse código irá prover as operações básicas de CRUD;

```
package br.com.jlgregorio.crudpeople.repository;

import br.com.jlgregorio.crudpeople.model.Person;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface PersonRepository extends JpaRepository<Person, Long> {

}
```

# Executando a aplicação

- Nesse momento, se a aplicação for executada, a tabela people do banco de dados será criada;
- Faça um teste!

A large, stylized green leaf graphic is positioned on the left side of the slide, partially overlapping the title box. It has a light green fill and a darker green outline, with a central vein and several smaller veins branching out.

# Ajustando o Services

# O service agora deve manipular o model

- No pacote principal da aplicação crie, um pacote services;
- Neste pacote, crie uma classe chamada PersonService;
- Veja a lista de Imports:

```
package br.com.jlgregorio.crud.services;  
import br.com.jlgregorio.crud.model.Person;  
import br.com.jlgregorio.crud.repository.PersonRepository;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
import java.util.List;
```



# Métodos de PersonService

```
// this class is a service
@Service
public class PersonService {

    // dependency injection
    @Autowired
    PersonRepository repository;

    public Person save(Person person){
        return repository.save(person);
    }

    public Person findById(long id) throws Exception{
        return repository.findById(id).orElseThrow(() → new Exception("Not Found!"));
    }
}
```

# Métodos de PersonService (continuação)

```
public Person update(Person person) throws Exception {
    Person found = repository.findById(person.getId()).orElseThrow(() → new Exception("Not Found!"));
    found.setFirstName(person.getFirstName());
    found.setLastName(person.getLastName());
    found.setProfession(person.getProfession());
    return repository.save(found);
}

public List<Person> findAll(){
    return repository.findAll();
}

public void delete(Long id) throws Exception {
    Person found = repository.findById(id).orElseThrow( () → new Exception("Not Found!"));
    repository.delete(found);
}
```

A large, stylized green leaf graphic is positioned on the left side of the slide, partially overlapping the green box. It has a light green fill and a darker green outline, with a pointed tip and a curved shape.

**Controller**

# Criando o Controller

- No pacote principal da aplicação, crie um pacote *controller*;
- Neste pacote, crie a classe PersonController;
- Veja a lista de imports da classe PersonController:

```
import br.com.jlgregorio.crud.model.Person;  
import br.com.jlgregorio.crud.services.PersonService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.*;  
import java.util.List;
```

# Métodos de PersonController

```
@RestController
@RequestMapping("/people")
public class PersonController {

    @Autowired
    private PersonService service;

    @GetMapping
    public List<Person> findAll(){
        return service.findAll();
    }

    @GetMapping("/{id}")
    public Person findById(@PathVariable("id") long id) throws Exception{
        return service.findById(id);
    }
}
```

# Métodos de PersonController (continuação)

```
@PostMapping
public Person save(@RequestBody Person person){
    return service.save(person);
}

@PutMapping
public Person update(@RequestBody Person person) throws Exception {
    return service.update(person);
}

@DeleteMapping("/{id}")
public ResponseEntity<?> delete(@PathVariable("id") Long id) throws Exception {
    service.delete(id);
    return ResponseEntity.ok().build();
}
}
```

# Sobre as anotações *Mapping*

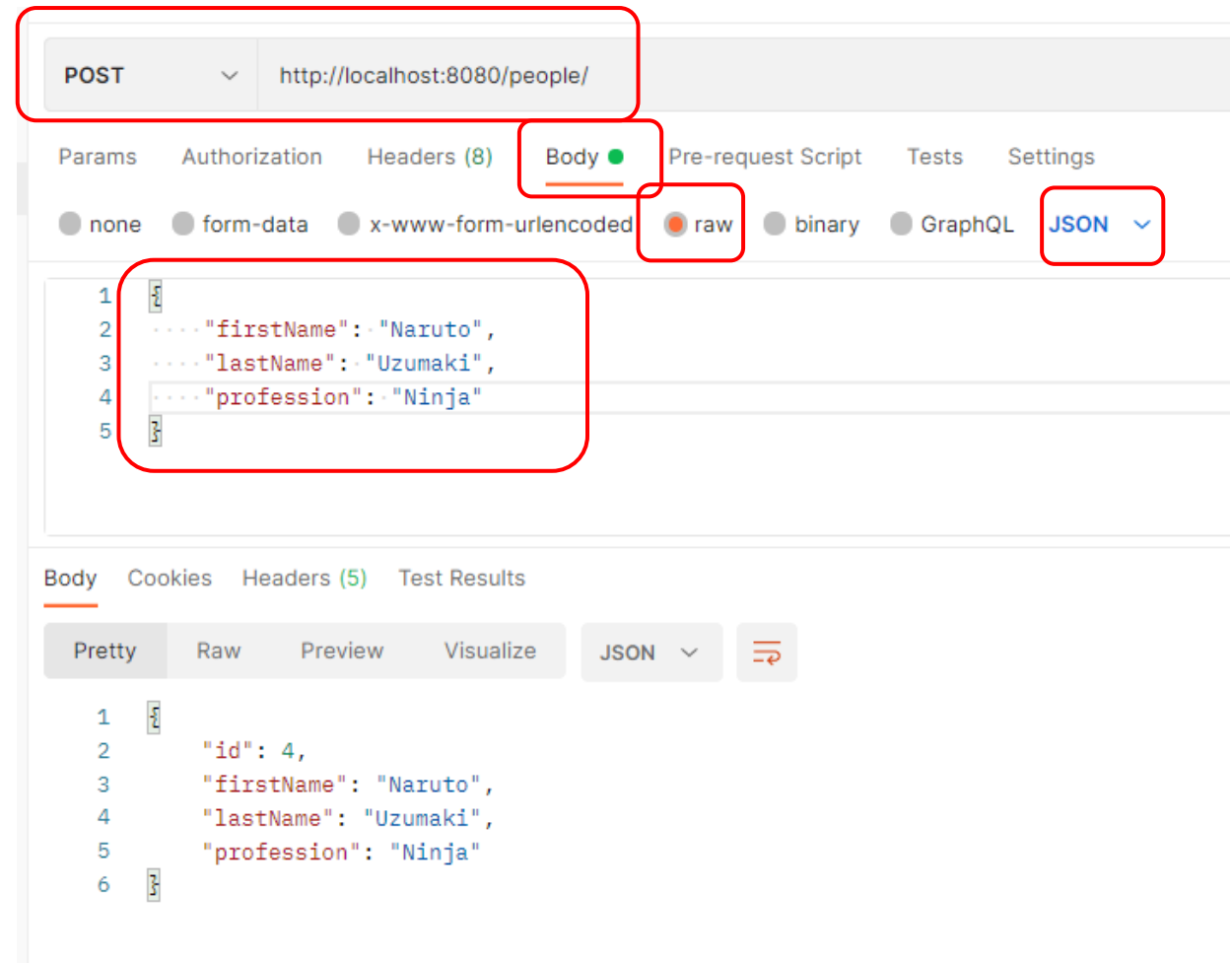
- Note que as anotações Mapping simplificam a construção dos controllers, pois consideram por padrão consumo e geração de dados no formato JSON;
- Assim, fica muito mais simples fazer o mapeamento das rotas

```
@PostMapping
public Person save(@RequestBody Person person){
    return service.save(person);
}
```

```
@DeleteMapping("/{id}")
public ResponseEntity<?> delete(@PathVariable("id") Long id) throws Exception {
    service.delete(id);
    return ResponseEntity.ok().build();
}
```

# Faça os testes no Postman!

- Seguindo a mesma estratégia do material anterior, faça os testes no Postman e veja os retornos!





# Desafio!

- Aproveitando o projeto existente, crie um CRUD de Veículos.

# Sobre mim

## JORGE LUÍS GREGÓRIO

- Professor da Faculdade de Tecnologia “Prof. José Camargo” – Fatec Jales, e da Escola Técnica Estadual Dr. José Luiz Viana Coutinho – Etec Jales;
  - Articulista do Jornal de Jales – Coluna “Fatecnologia”;
  - Apresentador do Tech Trends, podcast oficial da Fatec Jales;
  - Bacharel em Sistemas de Informação; Especialista em Desenvolvimento de Software para Web e Mestre em Ciência da Computação.
  - Trabalha com tecnologia desde 1998, tendo atuado como analista de suporte; administrador de redes de computadores; desenvolvedor de software, *webdesigner* e professor.
- 
- Site oficial: [www.jlgregorio.com.br](http://www.jlgregorio.com.br)
  - Perfil do LinkedIn: [www.linkedin.com/in/jlgregorio81](http://www.linkedin.com/in/jlgregorio81)
  - Currículo Lattes: <http://lattes.cnpq.br/3776799279256689>

