

# API REST





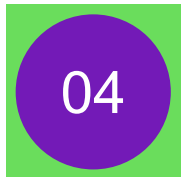
Iniciando projeto



Wrappers



**QUEUE**



MAP

# Iniciando projeto



## Projeto

☒ Projeto Maven ☐ Projeto Gradle

## Língua

☒ Java ☐ Kotlin ☐ Groovy

## Spring Boot

☐ 2.6.0 (INSTANTÂNEO) ☐ 2.6.0 (RC1) ☐ 2.5.7 (INSTANTÂNEO) ☒ 2.5.6  
☐ 2.4.13 (INSTANTÂNEO) ☐ 2.4.12

## Metadados de Projeto

Grupo

Artefato

Nome

Descrição

Nome do pacote

Embalagem ☒ .jar ☐ Guerra

Java ☐ 17 ☒ 11 ☐ 8

## Dependências

ADICIONAR DEPENDÊNCIAS... CTRL + B

### H2 Database

SQL

Fornecer um banco de dados na memória rápido que suporta acesso JDBC API e R2DBC, com uma pegada pequena (2 MB). Suporta modos integrados e de servidor, bem como um aplicativo de console baseado em navegador.

### Spring Data JPA

SQL

Persistir dados em armazenamentos SQL com Java Persistence API usando Spring Data e Hibernate.

### Spring Web

REDE

Crie aplicativos da web, incluindo RESTful usando Spring MVC. Usa Apache Tomcat como o contêiner integrado padrão.

### Spring Boot DevTools

FERRAMENTAS DE DESENVOLVIMENTO

Fornecer reinicializações rápidas de aplicativos, LiveReload e configurações para uma experiência de desenvolvimento aprimorada.

### Validação

I/O

Validação de feijão com validador Hibernate.

# Iniciando projeto

---

## Dependências

### **H2 Database SQL**

Fornece um banco de dados na memória rápida que suporta acesso JDBC API e R2DBC, com uma pegada pequena (2 MB). Suporta modos integrados e de servidor, bem como um aplicativo de console baseado em navegador.

### **Spring Data JPA SQL**

Persistir dados em armazenamentos SQL com Java Persistence API usando Spring Data e Hibernate.

### **Validation I/O**

Validação de feijão com validador Hibernate.

### **Spring Web WEB e Spring Boot DevTools DEVELOPER TOOLS**

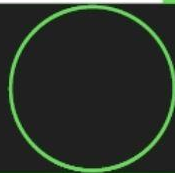
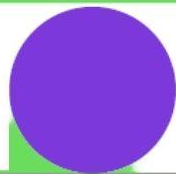
# Estrutura do projeto

---



---

**Model**



# Criando classe Model

## @Entity

```
public class Usuario implements Serializable {  
    private static final long serialVersionUID = 1L;
```

```
    @Id
```

```
    @GeneratedValue(strategy = GenerationType.IDENTITY)
```

```
    private Integer id;
```

```
    @NotEmpty
```

```
    @Length(min = 3, max = 100, message = "Valor entre 3 e 100 caracteres")
```

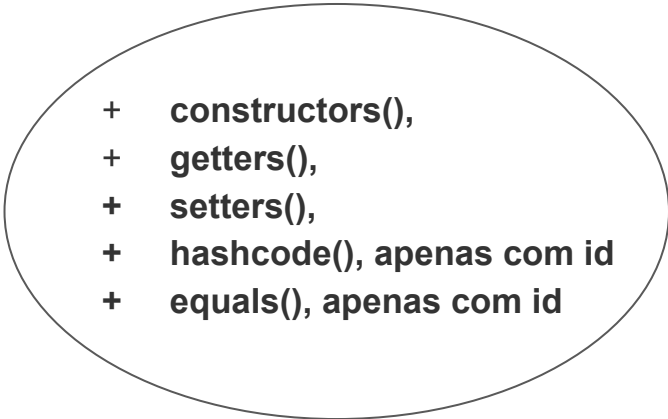
```
    private String nome;
```

```
    @NotEmpty
```

```
    @Length(min = 3, max = 100)
```

```
    private String senha;
```

```
}
```

- 
- + **constructors(),**
  - + **getters(),**
  - + **setters(),**
  - + **hashCode(), apenas com id**
  - + **equals(), apenas com id**

# Criando classe Model

---

## **Serialização**

A serialização é quando um objeto é transformado, em uma cadeia de bytes e desta forma pode ser manipulado de maneira mais fácil, seja através de transporte pela rede ou salvo no disco. Após a transmissão ou o armazenamento esta cadeia de bytes pode ser transformada novamente no objeto Java que o originou.

## ***Serializable***

Serializable que sinalizará a máquina virtual Java (JVM) que objetos daquela classe estão aptos a serem serializadas.

**SerialVersionUID** é um número que identifica a versão da classe que foi usada durante o processo de serialização.



# Criando classe Model

---

## @Entity

A anotação **@Entity** é utilizada para informar que uma classe também é uma entidade. A partir disso, a JPA estabelecerá a ligação entre a entidade e uma tabela de mesmo nome no banco de dados, onde os dados de objetos desse tipo poderão ser persistidos.

Uma entidade representa, na Orientação a Objetos, uma tabela do banco de dados, e cada instância desta entidade representa uma linha dessa tabela.

# Criando classe Model

---

## @Id

A anotação **@Id** é utilizada para informar ao JPA qual campo/atributo de uma entidade estará relacionado à chave primária da respectiva tabela no banco de dados. Essa é uma anotação obrigatória e um erro será gerado em tempo de execução caso ela não esteja presente.

## @GeneratedValue

A anotação **@GeneratedValue** é utilizada para informar que a geração do valor do identificador único da entidade será gerenciada pelo provedor de persistência. Essa anotação deve ser adicionada logo após a anotação **@Id**. Quando não anotamos o campo com essa opção, significa que a responsabilidade de gerar e gerenciar as chaves primárias será da aplicação

# Criando classe Model

---

## @NotEmpty

A anotação *@NotEmpty* faz uso da implementação *isValid ()* da classe *@NotNull* , que verifica se o tamanho / comprimento do objeto fornecido ( isso varia de acordo com o tipo de objeto sendo validado) é maior que zero.



**application.properties**



# application.properties

---

spring.datasource.url=jdbc:h2:mem:testdb

spring.datasource.driverClassName=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=password

spring.jpa.database-platform=org.hibernate.dialect.H2Dialect

spring.jpa.show-sql=true

spring.jpa.properties.hibernate.format\_sql=true

[Propriedades Comuns do application.properties](#)



# Instanciação DB



# Salvar no DB

---

```
import org.springframework.data.jpa.repository.JpaRepository;  
import com.primeira.api.apiRest.model.Usuario;
```

```
@Repository
```

```
public interface UsuarioRepository extends JpaRepository<Usuario, Integer> {
```

```
}
```

@Repository é uma anotação Spring que indica que a classe decorada é um repositório. Um repositório é um mecanismo para encapsular o comportamento de armazenamento, recuperação e pesquisa que emula uma coleção de objetos.

# Instanciação DB

---

@SpringBootApplication

public class ApiRestApplication implements **CommandLineRunner**{

    @Autowired

    UsuarioRepository usuarioRepository;

    public static void main(String[] args) {

        SpringApplication.run(ApiRestApplication.class, args);

    }

    @Override

    public void run(String... args) throws Exception {

        Usuario u1 = new Usuario(null, "Maria Silva", "marias", "1234");

        Usuario u2 = new Usuario(null, "Joao da Silva", "joao", "1234");

        usuarioRepository.saveAll(Arrays.asList(u1, u2));

    }

}



# Instanciação DB

---

## @Autowired

A anotação @Autowired fornece controle sobre onde e como a ligação entre os beans deve ser realizada. Pode ser usado para em métodos setter, no construtor, em uma propriedade ou métodos com nomes arbitrários e / ou vários argumentos.

No nosso exemplo a anotação vai permitir que Spring injete e gerenciar as instâncias do nosso repository.

# H2

<http://localhost:8080/h2-console/>

English ▼ Preferences Tools Help

---

Login

Saved Settings: Generic H2 (Embedded) ▼

Setting Name: Generic H2 (Embedded) Save Remove

---

Driver Class: org.h2.Driver

JDBC URL: jdbc:h2:mem:testdb

User Name: sa

Password:

Connect Test Connection



**GET**



# GET Usuário

---

@Service

```
public class UsuarioService {
```

```
    @Autowired
```

```
    private UsuarioRepository repository;
```

```
    public Usuario findById(Integer id) {
```

```
        Optional<Usuario> obj = repository.findById(id);
```

```
        return obj.orElse(null);
```

```
    }
```

```
    public List<Usuario> findAll() {
```

```
        return repository.findAll();
```

```
    }
```

```
}
```

# GET Usuário

---

## Optional

*Optional* é uma classe que foi implementada no Java 8, que tem o objetivo de simplificar os códigos, facilitando a vida dos desenvolvedores.

O *Optional* nos ajuda a evitar os erros *NullPointerException*, tirar a necessidade da verificação (if x != null) e também a escrever um código com menos linhas e mais bonito

# GET Usuário

---

```
@RestController
```

```
@RequestMapping(path= "/usuarios")
```

```
public class UsuarioController {
```

```
    @Autowired
```

```
    private UsuarioService service;
```

```
    @GetMapping(value =("/{id}")
```

```
    public ResponseEntity<Usuario> findById(@PathVariable Integer id){
```

```
        Usuario obj = this.service.findById(id);
```

```
        return ResponseEntity.ok().body(obj);
```

```
    };
```

```
    @GetMapping
```

```
    public ResponseEntity<List<Usuario>> findAll() {
```

```
        List<Usuario> list = service.findAll();
```

```
        return ResponseEntity.ok().body(list);
```

```
    }
```

```
}
```

# GET Usuário

---

## ResponseEntity

*ResponseEntity* representa toda a resposta HTTP: código de status, cabeçalhos e corpo . Como resultado, podemos usá-lo para configurar totalmente a resposta HTTP.

*ResponseEntity* é um tipo genérico. Consequentemente, podemos usar qualquer tipo como corpo de resposta.

VAMOS PRATICAR?