



EducaCiência FastCode

Fala Galera,

- Artigo: 38/2021 Data: Fevereiro/2021
- Público-alvo: Desenvolvedores – Iniciantes
- Tecnologia: Java
- Tema: Artigo 38- SpringBoot JPA H2 Get ID
- Link: <https://github.com/perucello/DevFP>

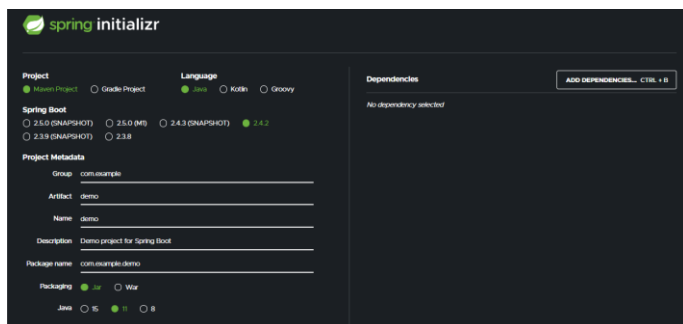
Neste artigo, abordaremos Spring Boot e iremos mapear o Método Get ID com repositório JPA Repository e Banco de Dados relacional H2.

Traremos uma série de artigos onde exploraremos os métodos de maneira individual até chegarmos em uma API com todos os métodos.

Lembrando que os fins são didáticos !

Para este ambiente , utilizaremos do Banco de Dados H2 , porém, virtual e assim já deixaremos um script de inserção dos dados no próprio código para que ao iniciarmos ele já esteja válido para manipularmos os dados de maneira objetiva.

Criaremos nosso projeto utilizando do link - <https://start.spring.io/> e abriremos na IDE Spring Tool Suíte 3 “STS”.





Com nosso Projeto já aberto no STS , iremos preparar nosso arquivo “pom”, ou seja, nossas dependências que iremos trabalhar no projeto.

```
<?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>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.4.2</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.project.jpa</groupId>
    <artifactId>JavaJPA</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>JavaJPA</name>
    <description>Spring Boot</description>
    <properties>
        <java.version>1.8</java.version>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

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

        <dependency>
            <groupId>javax.validation</groupId>
            <artifactId>validation-api</artifactId>
            <version>1.1.0.Final</version>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-devtools</artifactId>
            <scope>runtime</scope>
            <optional>true</optional>
        </dependency>
        <dependency>
            <groupId>com.h2database</groupId>
            <artifactId>h2</artifactId>
            <scope>runtime</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
</project>
```



Feito isso, vamos atualizar nosso Maven – para isso, basta executar “Maven Install”.

Vamos preparar agora nossos “pacotes” como demonstrados abaixo:

```
✓ JPA_Project_H2 [boot] [devtools] [JPA_Project_H2 main]
  ✓ src/main/java
    > com.project.jpa.JavaJPA
      > com.project.jpa.JavaJPA.controller
      > com.project.jpa.JavaJPA.model
      > com.project.jpa.JavaJPA.repository
    > src/main/resources
    > src/test/java
    > JRE System Library [JavaSE-1.8]
    > Maven Dependencies
    > bin
    > src
      > target
      mvnw
      mvnw.cmd
      > pom.xml
```

Feito isso, vamos criar nossas classes sendo:

```
✓ JPA_Project_H2 [boot] [devtools]
  ✓ src/main/java
    ✓ com.project.jpa.JavaJPA
      > JavaJpaApplication.java
    ✓ com.project.jpa.JavaJPA.controller
      > ClientesController.java
    ✓ com.project.jpa.JavaJPA.model
      > Cliente.java
    ✓ com.project.jpa.JavaJPA.repository
      > Clientes.java
  > src/main/resources
  > src/test/java
  > JRE System Library [JavaSE-1.8]
  > Maven Dependencies
  > bin
  > src
    > target
    mvnw
    mvnw.cmd
    pom.xml
```

Vamos detalhar:

- a) **Controller** – neste pacote teremos nossa classe “ClientesController” onde será escrito nosso código que manipularemos os métodos CRUD e nosso endpoint;
- b) **Model** – neste pacote teremos nossa classe “Cliente” onde criaremos a estrutura da nossa tabela Cliente no Banco de Dados
- c) **Repository** – teremos nossa “interface” que se estenderá da classe “Cliente” e receberá nosso Repository JPA.

Primeiramente, vamos criar a estrutura da nossa tabela, como a seguir:

```
package com.project.jpa.JavaJPA.model;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.validation.constraints.NotNull;
```

```
@Entity
public class Cliente {

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

    private String nome;

    @NotNull
    private String email;

    public Cliente() {
        super();
    }

    public Cliente(Long id, String nome, String email) {
        super();
        this.id = id;
        this.nome = nome;
        this.email = email;
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
```

```

        this.email = email;
    }

    @Override
    public int hashCode() {
        final int prime = 31;
        int result = 1;
        result = prime * result + ((id == null) ? 0 : id.hashCode());
        return result;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (getClass() != obj.getClass())
            return false;
        Cliente other = (Cliente) obj;
        if (id == null) {
            if (other.id != null)
                return false;
        } else if (!id.equals(other.id))
            return false;
        return true;
    }
}

```

Agora, podemos estender nossa interface, para isso , abra o pacote Repositório e clique em Clientes.java e insira o seguinte código:

```

package com.project.jpa.JavaJPA.repository;

import org.springframework.data.jpa.repository.JpaRepository;




import com.project.jpa.JavaJPA.model.Cliente;

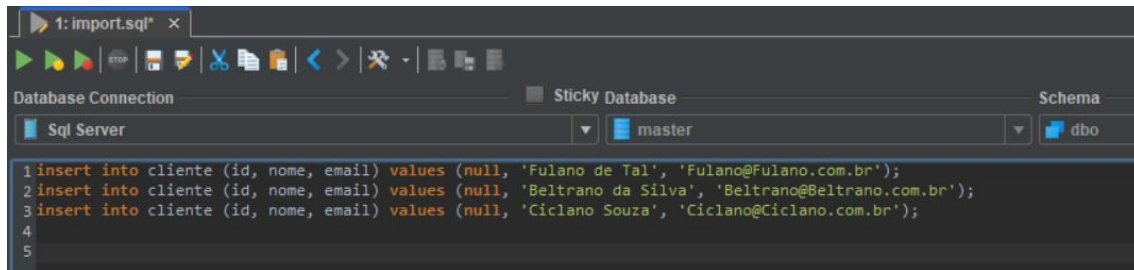
public interface Clientes extends JpaRepository<Cliente, Long> {

}

```

Feito isso, temos a estrutura do nosso banco de dados (tabela) pronta e agora , para que ao iniciarmos nossa aplicação do banco de dados relacional “H2” , vamos deixar um código sql em “resources” , para isso crie um arquivo chamado import.sql com os seguintes comandos descritos abaixo e “cole-o” em src/main/resources:

 src/main/resources
 application.properties
 **import.sql**



Feito isso, ao iniciarmos nossa API, estes dados já serão inseridos em nosso Banco de Dados H2.

Com a nossa estrutura do Banco de Dados criado, (entidade e repositório) agora, podemos focar em nosso CRUD.

Para este trabalho, abra o Script **ClientesController** que está no pacote Controller e insira os seguintes códigos:

```

package com.project.jpa.JavaJPA.controller;

import java.util.Optional;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import com.project.jpa.JavaJPA.model.Cliente;
import com.project.jpa.JavaJPA.repository.Clientes;

@RestController
@RequestMapping("api/JPA/clientes")
public class ClientesController {

    @Autowired
    private Clientes clientes;

    @GetMapping("/{id}")
    public ResponseEntity<Optional<Cliente>> buscar(@PathVariable Long id){
        Optional<Cliente> cliente = clientes.findById(id);
        if (clientes == null) {
            return ResponseEntity.notFound().build();
        }
        return ResponseEntity.ok(cliente);
    }
}

```

Com isso, temos nosso CRUD pronto onde os métodos que manipularemos são os seguintes:

- ⇒ **Get id** – este método retornará os dados de acordo com parâmetro passado, no nosso caso “id”, uma busca por “id” (Select)

Agora, basta iniciarmos nossa “Aplicação”

⇒ Run As \ Spring Boot App

```

:: Spring Boot ::
(v2.4.2)

2021-02-14 18:31:48.651 INFO 14160 --- [ restartedMain] c.p.jpa.JavaJPA.JavaJPAApplication : Starting JavaJPAApplication using Java 1.8.0_212-3-redhat on DESKTOP-F3A10P2 with PID 14160 (C:\Users\
2021-02-14 18:31:48.656 INFO 14160 --- [ restartedMain] c.p.jpa.JavaJPA.JavaJPAApplication : No active profile set, falling back to default profiles: default
2021-02-14 18:31:48.733 INFO 14160 --- [ restartedMain] e.DevToolsPropertyDefaultsPostProcessor : DevTools property defaults active! Set 'spring.devtools.add-properties' to 'false' to disable
2021-02-14 18:31:48.734 INFO 14160 --- [ restartedMain] e.DevToolsPropertyDefaultsPostProcessor : For additional web related logging consider setting the 'logging.level.web' property to 'DEBUG'
2021-02-14 18:31:49.764 INFO 14160 --- [ restartedMain] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFAULT mode.
2021-02-14 18:31:49.851 INFO 14160 --- [ restartedMain] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 70 ms. Found 1 JPA repository interfaces.
2021-02-14 18:31:50.704 INFO 14160 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
2021-02-14 18:31:50.717 INFO 14160 --- [ restartedMain] o.apache.catalina.core.StandardEngine : Starting service [Tomcat]
2021-02-14 18:31:50.717 INFO 14160 --- [ restartedMain] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.41]
2021-02-14 18:31:50.882 INFO 14160 --- [ restartedMain] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2021-02-14 18:31:50.883 INFO 14160 --- [ restartedMain] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 2147 ms
2021-02-14 18:31:50.959 INFO 14160 --- [ restartedMain] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Starting...
2021-02-14 18:31:51.142 INFO 14160 --- [ restartedMain] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Start completed.
2021-02-14 18:31:51.154 INFO 14160 --- [ restartedMain] o.s.b.a.h2.H2ConsoleAutoConfiguration : H2 console available at '/h2-console'. Database available at 'jdbc:h2:mem:0bd5d788-17d8-440b-ac08-79b
2021-02-14 18:31:51.402 INFO 14160 --- [ restartedMain] o.hibernate.jpa.internal.util.LogHelper : HH0000204: Processing PersistenceUnitInfo [name: default]
2021-02-14 18:31:51.553 INFO 14160 --- [ restartedMain] org.hibernate.Version : HH0000412: Hibernate ORM core version 5.4.27.Final
2021-02-14 18:31:52.007 INFO 14160 --- [ restartedMain] org.hibernate.annotations.common.Version : HCANN000001: Hibernate Commons Annotations (5.1.2.Final)
2021-02-14 18:31:52.236 INFO 14160 --- [ restartedMain] org.hibernate.dialect.Dialect : HH0000400: Using dialect: org.hibernate.dialect.H2Dialect
2021-02-14 18:31:53.047 INFO 14160 --- [ restartedMain] o.h.t.schema.internal.SchemaCreatorImpl : HH0000476: Executing import script 'file:/C:/Users/Usuario/Documents/Perucello%20SSD/00%20Artigos%20FI
2021-02-14 18:31:53.056 INFO 14160 --- [ restartedMain] o.h.e.t.j.p.i.JtaPlatformInitiator : HH0000490: Using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platform.internal.I
2021-02-14 18:31:53.078 INFO 14160 --- [ restartedMain] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
2021-02-14 18:31:53.120 INFO 14160 --- [ restartedMain] o.s.b.d.a.OptionalLiveReloadServer : LiveReload server is running on port 35729
2021-02-14 18:31:53.577 WARN 14160 --- [ restartedMain] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, database queries may be performed during view
2021-02-14 18:31:53.741 INFO 14160 --- [ restartedMain] o.s.s.concurrent.ThreadPoolTaskExecutor : Initializing ExecutorService 'applicationTaskExecutor'
2021-02-14 18:31:54.063 INFO 14160 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''
2021-02-14 18:31:54.076 INFO 14160 --- [ restartedMain] c.p.jpa.JavaJPA.JavaJPAApplication : Started JavaJPAApplication in 5.978 seconds (JVM running for 7.477)

```

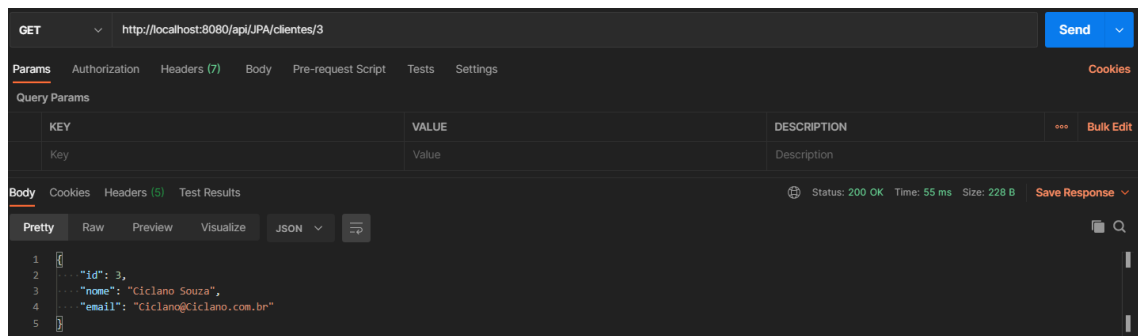
Nossa aplicação se iniciou como esperado.

Agora podemos testar , manipulando via Postman, para isso abra seu aplicativo e vamos testar os métodos que criamos em nossa API.

Utilizaremos como dissemos anteriormente, o Postman.

Vamos inserir os Endpoints de acesso e manipulá-lo como abaixo:

- ⇒ **Get id** – este método retornará os dados de acordo com parâmetro passado , no nosso caso “id” , uma busca por “id” (Select)
- ⇒ **Retorno esperado quando se obtém sucesso é 200 neste caso.**



Nossa API funcionou como nossa proposta, sendo assim finalizamos este artigo, onde os códigos estão disponíveis no GitHub para consumo.

Até mais !
Espero ter ajudado !