

Tutorial SpringBoot com JPA, Hibernate, H2 e Rest

Parte 2

Glauco Todeskco

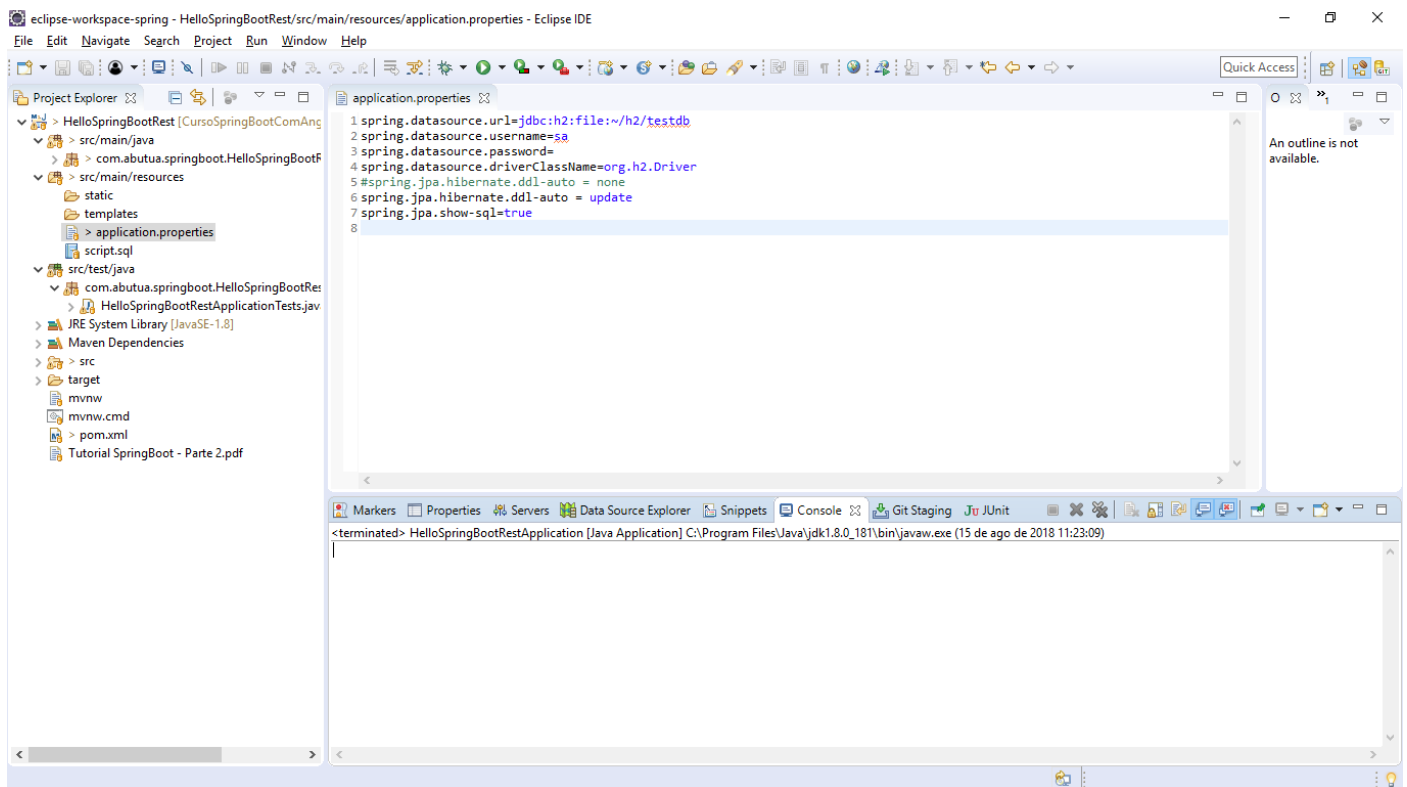
Download:

<https://github.com/glaucotodesco/CursoSpringBootComAngular/tree/Tutorial-Parte2>

Nessa segunda parte do tutorial o nosso serviço Rest vai retornar todos os registros de uma tabela no formato JSON.

1. Inicialmente vamos configurar o arquivo application.properties para que o SpringBoot acesse o banco de dados já configurado no pom.xml:

```
spring.datasource.url=jdbc:h2:file:~/h2/testdb
spring.datasource.username=sa
spring.datasource.password=
spring.datasource.driverClassName=org.h2.Driver
#spring.jpa.hibernate.ddl-auto = none
spring.jpa.hibernate.ddl-auto = update
spring.jpa.show-sql=true
```



2. Agora iremos construir uma entidade do JPA conforme a classe Produto abaixo. No momento todas as classes criadas devem estar no mesmo pacote ou subpacote da aplicação (método main):

```
package com.abutua.springboot.HelloSpringBootRest;
```

```
import javax.persistence.Entity;  
import javax.persistence.Id;
```

```
@Entity
```

```
public class Produto {
```

```
    @Id
```

```
    private long id;
```

```
    private String descricao;
```

```
    private double preco;
```

```
    public long getId() {
```

```
        return id;
```

```
    }
```

```
    public void setId(long id) {
```

```
        this.id = id;
```

```
    }
```

```
    public String getDescricao() {
```

```
        return descricao;
```

```
    }
```

```
    public void setDescricao(String descricao) {
```

```
        this.descricao = descricao;
```

```
    }
```

```
    public double getPreco() {
```

```
        return preco;
```

```
    }
```

```
    public void setPreco(double preco) {
```

```
        this.preco = preco;
```

```
    }
```

```
}
```

3. Crie agora uma interface de repositório para os produtos. No momento todas as interfaces criadas devem estar no mesmo pacote da aplicação (método main):

```
package com.abutua.springboot.HelloSpringBootRest;
```

```
import org.springframework.data.repository.CrudRepository;
```

```
public interface ProdutoRepository extends CrudRepository <Produto, Long> {
```

```
}
```

4. A última etapa de codificação é alterar a classe de Controle para retornar todos os produtos cadastrados:

```
package com.abutua.springboot.HelloSpringBootRest;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class Controller {

    @Autowired
    ProdutoRepositorio rep;

    @RequestMapping("/")
    public Iterable<Produto> index() {
        return rep.findAll();
    }
}
```

5. Ao fazer uma chamada pelo navegador teremos a seguinte saída:



6. Podemos inserir dados na tabela de Produto através da console do H2 acessando o seguinte endereço pelo navegador:

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

7. Defina as seguintes configurações para fazer o login na console do H2:

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:file:~/h2/testdb

User Name: sa

Password:

Connect Test Connection

8. Ao fazer o login execute o seguinte script SQL para inserir dados na tabela Produto:

```
Insert Into Produto Values (1,'Produto A',10.5);
Insert Into Produto Values (2,'Produto B',20.5);
Insert Into Produto Values (3,'Produto C',30.5);
Insert Into Produto Values (4,'Produto D',40.5);
Insert Into Produto Values (5,'Produto E',50.5);
```

H2 Console

localhost:8080/h2-console/login.do?sessionId=2568af8fc3a9363bc9d2751eae2d6794

Auto commit Max rows: 1000 Auto complete Off Auto select On

jdbc:h2:file:~/h2/testdb

PRODUTO

INFORMATION_SCHEMA

Users

H2 1.4.197 (2018-03-18)

Run Run Selected Auto complete Clear SQL statement:

```
Insert Into Produto Values (1,'Produto A',10.5);
Insert Into Produto Values (2,'Produto B',20.5);
Insert Into Produto Values (3,'Produto C',30.5);
Insert Into Produto Values (4,'Produto D',40.5);
Insert Into Produto Values (5,'Produto E',50.5);
```

Important Commands

?	Displays this Help Page
📜	Shows the Command History
🏃	Ctrl+Enter Executes the current SQL statement
👤	Shift+Enter Executes the SQL statement defined by the text selection
🔍	Ctrl+Space Auto complete
🔌	Disconnects from the database

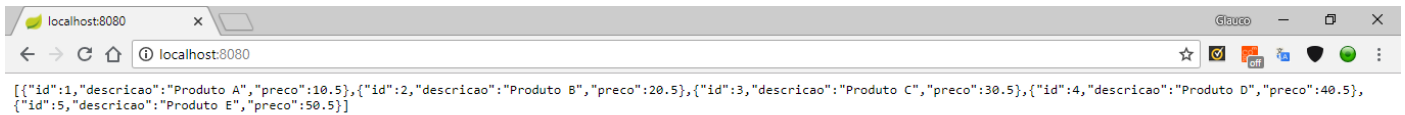
Sample SQL Script

Delete the table if it exists	DROP TABLE IF EXISTS TEST;
Create a new table with ID and NAME columns	CREATE TABLE TEST(ID INT PRIMARY KEY, NAME VARCHAR(255));
Add a new row	INSERT INTO TEST VALUES(1, 'Hello');
Add another row	INSERT INTO TEST VALUES(2, 'World');
Query the table	SELECT * FROM TEST ORDER BY ID;
Change data in a row	UPDATE TEST SET NAME='Hi' WHERE ID=1;
Remove a row	DELETE FROM TEST WHERE ID=2;
Help	HELP ...

Adding Database Drivers

Additional database drivers can be registered by adding the .jar file location of the driver to the the environment variables H2DRIVERS or CLASSPATH. Example (Windows): to add the database driver library

9. Após inserir os registros na tabela Produto, faça uma nova requisição pelo navegador:



10. Para não recriar todas as vezes a tabela Produto, altere no arquivo application.properties a propriedade spring.jpa.hibernate.ddl-auto = **update** para **none**, invertendo o comentário.

```
spring.datasource.url=jdbc:h2:file:~/h2/testdb
spring.datasource.username=sa
spring.datasource.password=
spring.datasource.driverClassName=org.h2.Driver
spring.jpa.hibernate.ddl-auto = none
#spring.jpa.hibernate.ddl-auto = update
spring.jpa.show-sql=true
```