

Tutorial SpringBoot com JPA, Hibernate, H2, Derby e Rest

Parte 2

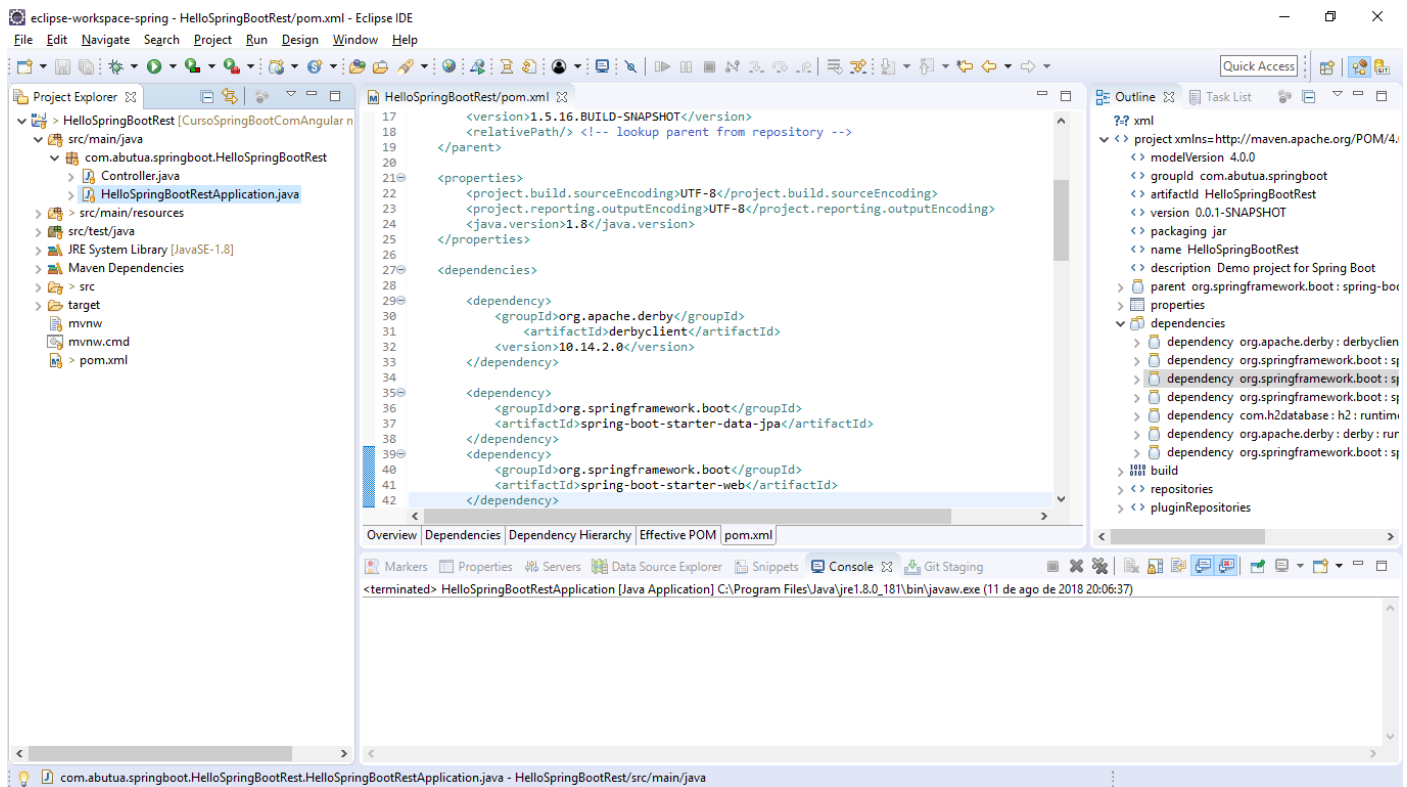
Glauco Todesco

Download:

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

1. Vamos acrescentar novas dependências no arquivo pom.xml do Maven, relacionadas ao acesso ao banco de dados do Derby.

```
<dependency>
  <groupId>org.apache.derby</groupId>
  <artifactId>derbyclient</artifactId>
  <version>10.14.2.0</version>
</dependency>
```



2. Vamos também deixar o banco Derby funcionando:

```
C:\WINDOWS\system32\cmd.exe

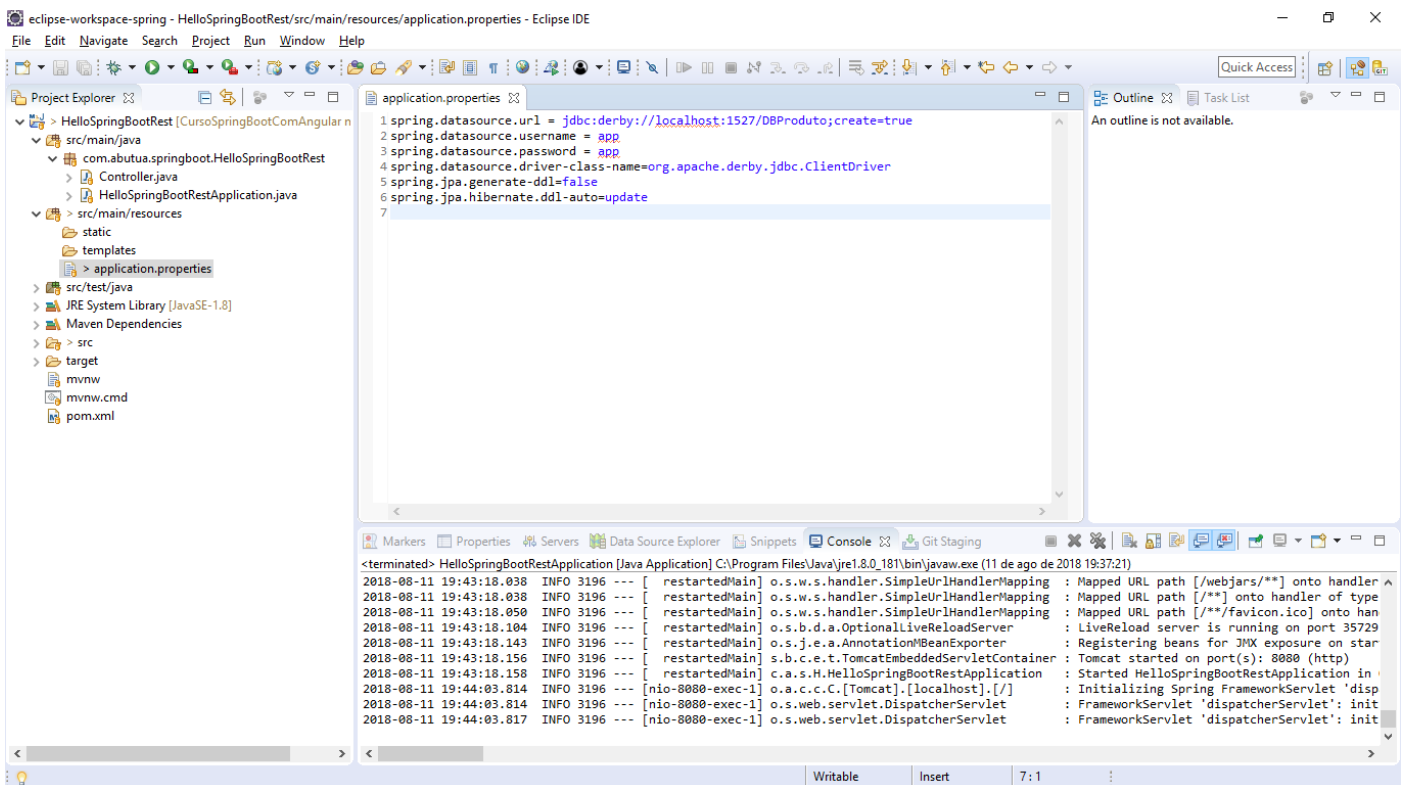
C:\Users\glauc\Desktop\db>set DERBY_INSTALL=C:\Users\glauc\Desktop\db\lib

C:\Users\glauc\Desktop\db>set CLASSPATH=C:\Users\glauc\Desktop\db\lib\lib\derbytools.jar;C:\Users\glauc\Desktop\db\lib\lib\derbynet.jar;.

C:\Users\glauc\Desktop\db>java -jar C:\Users\glauc\Desktop\db\lib\derbyrun.jar server start -noSecurityManager
Sat Aug 11 13:22:18 BRT 2018 : Apache Derby Servidor de Rede - 10.8.1.2 - (1095077) iniciado e pronto para aceitar conexões na porta 1527 em {3}
```

3. A próxima etapa é configurar o arquivo application.properties para que o SpringBoot acesse o banco de dados:

```
spring.datasource.url = jdbc:derby://localhost:1527/DBProduto;create=true
spring.datasource.username = app
spring.datasource.password = app
spring.datasource.driver-class-name=org.apache.derby.jdbc.ClientDriver
spring.jpa.generate-ddl=false
spring.jpa.hibernate.ddl-auto=update
```



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

```
package com.abutua.springboot.HelloSpringBootTest;
```

```
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;
```

```
    }
```

```
}
```

5. 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.HelloSpringBootTest;
```

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

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

```
}
```

6. 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();
    }
}
```

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



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

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

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

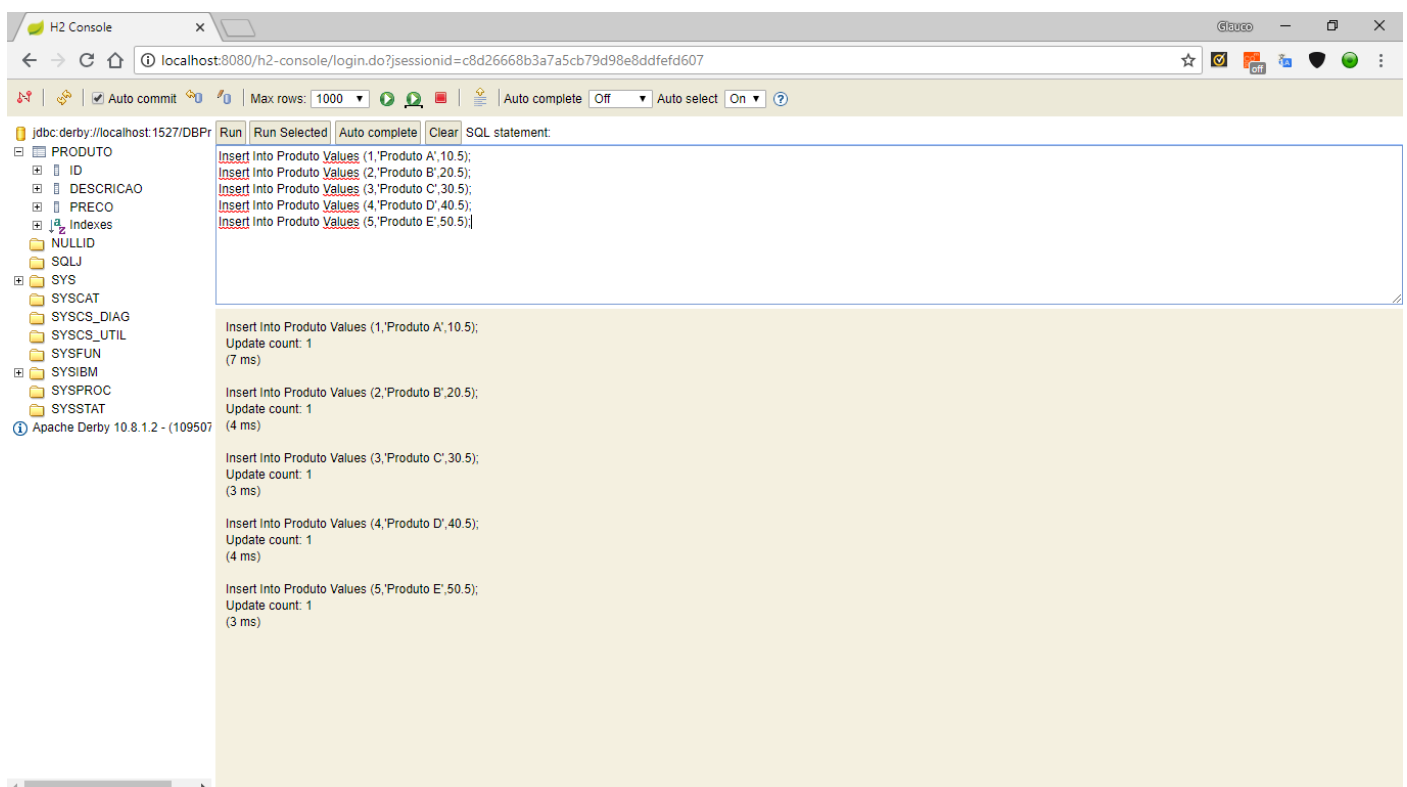


The screenshot shows the H2 Console application window. The address bar displays 'localhost:8080/h2-console'. The 'Login' dialog box is open, showing the following fields and buttons:

- Language:** English (dropdown menu)
- Preferences:** Preferences (link)
- Tools:** Tools (link)
- Help:** Help (link)
- Saved Settings:** Generic Derby (Server) (dropdown menu)
- Setting Name:** Generic Derby (Server) (text field) with **Save** and **Remove** buttons
- Driver Class:** org.apache.derby.jdbc.ClientDriver (text field)
- JDBC URL:** jdbc:derby://localhost:1527/DBProduto (text field)
- User Name:** app (text field)
- Password:** ... (password field)
- Buttons:** Connect and Test Connection

10. 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);
```



The screenshot shows the H2 Console application window after the login. The address bar displays 'localhost:8080/h2-console/login.do?sessionId=c8d26668b3a7a5cb79d98e8ddfed607'. The console shows the execution of the SQL script:

```
jdbc:derby://localhost:1527/DBPr
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);

Insert Into Produto Values (1,'Produto A',10.5);
Update count: 1
(7 ms)

Insert Into Produto Values (2,'Produto B',20.5);
Update count: 1
(4 ms)

Insert Into Produto Values (3,'Produto C',30.5);
Update count: 1
(3 ms)

Insert Into Produto Values (4,'Produto D',40.5);
Update count: 1
(4 ms)

Insert Into Produto Values (5,'Produto E',50.5);
Update count: 1
(3 ms)
```

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

