

- Como retornar o código de Status:
 - `ResponseEntity.ok()`
 - `ResponseEntity.notFound().build()`
 - `ResponseEntity.badRequest().build()`
- Os principais Status de Retorno
 - 200 - ok, 404 - not found, 201 - created, 400 - bad request
- Como definir cabeçalhos na nossa resposta
 - `ResponseEntity.notFound().header("chave", "valor").build()`
- Tratamento de Exceptions pelo Spring Boot
 - Controller utilizando a anotação `@RestControllerAdvice` - irá tratar as exceções da aplicação
 - Método no controller com anotação `@ExceptionHandler(ClassDaException.class)` irá tratar a exceção e montar uma resposta com `ResponseEntity`
- Refactoring - refatoração de código
- Utilização do DevTools para evitar reiniciar a aplicação a toda alteração

Localizado na pasta resources da aplicação, o arquivo “application.properties” tem o seguinte formato:

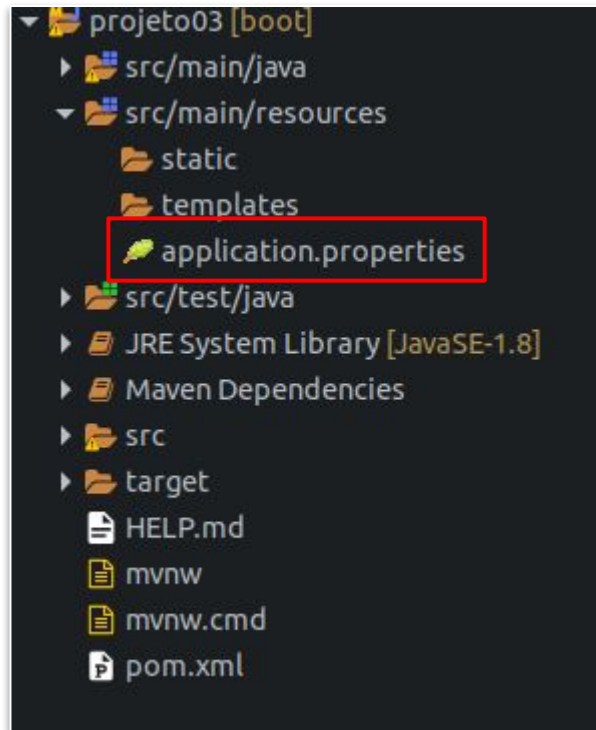
```
propriedade=valor
```

Sendo que “propriedade” pode ser uma “string” simples ou com separação por pontos, como nos nomes de pacotes:

```
prop.subprop=valor
```

Pode ser utilizado o padrão YAML, neste caso o nome do arquivo seria “application.yaml” e teria o formato:

```
prop:
  subprop:  valor
```



Propriedades comuns do Spring Boot:

server.port	Porta do servidor (8080)
spring.datasource.url	URL jdbc do banco
spring.datasource.username	Usuário para acesso ao banco
spring.datasource.password	Senha para acesso ao banco

```
application.properties
1 server.port=9090
2
3
```

```

  ____  _
 / ___|| | | |
| |___| | | |
 \___ \| | | |
  ___) | | | |
 / ___|| | | |
| |___| | | |
 \___) |_| |_|

:: Spring Boot ::      (v2.3.2.RELEASE)

2020-08-14 16:47:22.914 INFO 1156273 --- [main] o.s.j.projeto03.Projeto03Application : Starting Projeto03Application on bula-idx-notebo
2020-08-14 16:47:22.916 INFO 1156273 --- [main] o.s.j.projeto03.Projeto03Application : No active profile set, falling back to default p
2020-08-14 16:47:23.428 INFO 1156273 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 9090 (http)
2020-08-14 16:47:23.433 INFO 1156273 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2020-08-14 16:47:23.433 INFO 1156273 --- [main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.37]
2020-08-14 16:47:23.462 INFO 1156273 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationConte
2020-08-14 16:47:23.463 INFO 1156273 --- [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization compl
2020-08-14 16:47:23.564 INFO 1156273 --- [main] o.s.s.concurrent.ThreadPoolTaskExecutor : Initializing ExecutorService 'applicationTaskExe
2020-08-14 16:47:23.660 INFO 1156273 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s) 9090 (http) with con
2020-08-14 16:47:23.665 INFO 1156273 --- [main] o.s.j.projeto03.Projeto03Application : Started Projeto03Application in 0.947 seconds (
```

Propriedades customizadas

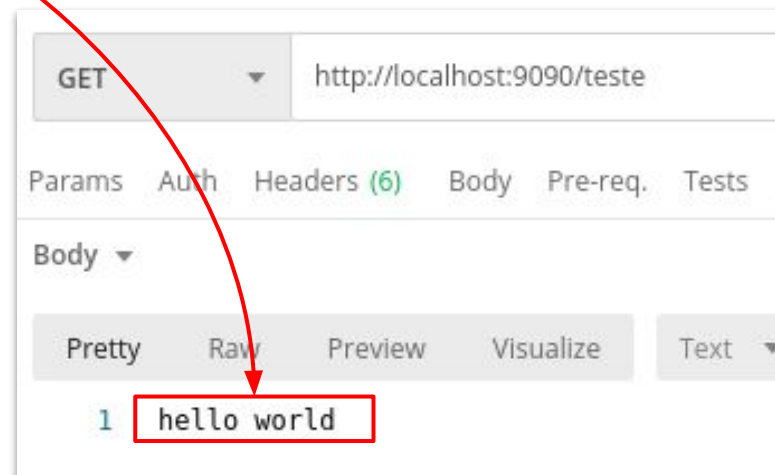
```
server.port=9090
```

```
minha-propriedade=hello world
```

```
@Value("${minha-propriedade}")  
private String propriedadeCustomizada;  
  
@GetMapping("/teste")  
public String retornaMinhaPropriedade() {  
    return propriedadeCustomizada;  
}
```

IMPORTANTE

O nome da propriedade deve estar entre \${..} na anotação @Value.



É possível ter uma classe complexa preenchida pelo arquivo de configuração do Spring Boot

application.properties

```
aplicacao.valor1=1500
aplicacao.valor2=Teste de valor
aplicacao.array1=a,b,c
aplicacao.array2[0]=1500
aplicacao.array2.1=1300
aplicacao.map.chave1=Primeira chave
aplicacao.map.chave2=Segunda Chave
aplicacao.lista=1,2,3
aplicacao.todos[0].id=1
aplicacao.todos[0].titulo=gravar
aplicacao.todos[0].descricao=gravar a aula
```

“Bean” de configuração


```
@Component
@ConfigurationProperties("aplicacao")
public class AplicacaoConfig {

    private int valor1;
    private String valor2;
    private String[] array1;
    private int[] array2;
    private Map<String,String> map;
    private List<Integer> lista;
    private List<Todo> todos;
}
```

Controller

```
@Autowired
private AplicacaoConfig aplicacaoConfig;

@GetMapping("/config")
public AplicacaoConfig retornaConfig() {
    return aplicacaoConfig;
}
```



GET http://localhost:9090/config

Params Auth Headers (6) Body Pre-req. Tests Settings

Body ▾

Pretty Raw Preview Visualize JSON

```
1 {
2   "valor1": 1500,
3   "valor2": "Teste de valor",
4   "array1": [
5     "a",
6     "b",
7     "c"
8   ],
9   "array2": [
10    1500,
11    1300
12  ],
13  "map": {
14    "chave1": "Primeira chave",
15    "chave2": "Segunda Chave"
16  },
17  "lista": [
18    1,
19    2,
20    3
21  ],
22  "todos": [
23    {
24      "id": 1,
25      "titulo": "gravar",
26      "descricao": "gravar a aula"
27    }
28  ]
29 }
```

1. Parâmetros de linha de comando

`java -jar minhaApp.jar --minha-propriedade=valor`

1. Atributos JNDI em `java:comp/env` (apenas para aplicações em servidores JEE)
2. Java System properties (`System.getProperties()`).
`System.setProperty(<prop>,<valor>)`

1. Variáveis do Sistema Operacional.

Win: `set propriedade=valor` / ou Painel de Controle - Variáveis do ambiente...

Linux: `propriedade=valor` / `export propriedade=valor`

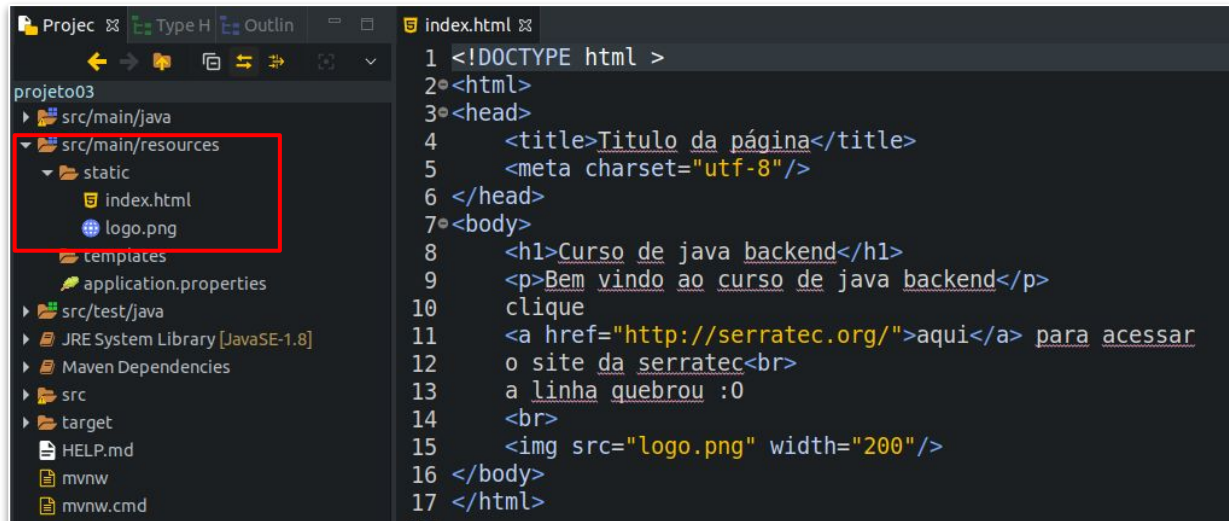
1. Um arquivo de profile específico **fora** do JAR quando for ativado profile
`application-{profile}.properties` / `application-dev.properties`

1. Um arquivo de profile específico **dentro** do JAR quando for ativado profile
`application-{profile}.properties` / `application-dev.properties`

1. O arquivo `application.properties` **fora** do jar
2. O arquivo `application.properties` **dentro** do jar
3. Valor da propriedade padrão do Spring

Arquivos e páginas html estáticas podem ser disponibilizadas pela aplicação se estiverem em uma destas pastas (/src/main/resources):

- /META-INF/resources/
- /resources/
- /static/
- /public/



Hibernate: Framework ORM - Object Relational Mapping

(*Mapeamento Objeto Relacional*)

- Diminuir a complexidade na integração de aplicações Java e bancos de dados relacionais
- Abstrair diversas operações sql (INSERT, UPDATE, DELETE)
- Facilitar a realização de consultas
- Interoperabilidade entre Bancos de dados



Diferenças entre a sintaxe SQL

```
-- oracle
select rownum linha, a.*    from
  ( select * from alunos order by nome ) a
) where
  linha > 10 and -- linha inicial
  linha < 20    -- linha final

--postgres, h2, mysql, sqlite
select * from alunos order by nome
  limit 10  -- linha inicial
 offset 10  -- itens por pagina

--sqlServer
select * from alunos order by nome
  offset 10 rows -- linha inicial
  fetch next 10 rows only -- itens por pagina
```

Hibernate “gera” o SQL específico para o banco configurado

```
//Hibernate
Session session = sessionFactory.openSession();
Query query = sess.createQuery("From Aluno");
query.setFirstResult(20);
query.setMaxResults(10);
List<Foo> fooList = fooList = query.list();
```



Java
Community
Process



HIBERNATE

← → ↺ 🏠 🔒 jcp.org/en/jsr/detail?id=338

 Java Community Process

 Community Development of Java Technology Specifications

JSRs Community Expert Group

Summary | Proposal | Detail (Summary & Proposal) | Nominations

JSRs: Java Specification Requests
JSR 338: Java™ Persistence 2.2

» JSRs by Platform



JSRs: Java Specification Requests
JSR 338: Java™ Persistence 2.2



Apache OpenJPA
Fundação Apache



Eclipselink - Fundação Eclipse
(antigo Toplink, doado pela Oracle)
Hoje é a implementação de referência



HIBERNATE

Hibernate, um dos principais
frameworks ORM.
Mantido pela RedHat

Spring Data tem por finalidade facilitar o acesso a fontes de dados no “estilo” Spring ;-)

- Spring Data JDBC
- Spring Data JPA
- Spring Data LDAP - Lightweight Directory Access Protocol (muito utilizado por serviços de autenticação e permissão)
- Spring Data MongoDB - Banco de dados NoSQL (json)
- Spring Data Redis - Banco em memória, distribuído para armazenamento de dados chave/valor
- Spring Data REST - permite consultar outros serviços REST
- Spring Data for Apache Cassandra - Banco de dados NoSQL
- Spring Data Elasticsearch - Servidor de indexação e buscas baseado no Apache Lucene

- Facilita a implementação de Repositórios em JPA
- Conceito de Repositórios para acesso aos dados
- Utiliza o Hibernate como implementação

Aplicação

CrudRepository

PagingAndSortingRepository

JpaRepository

Spring Data JPA

Hibernate

Banco de Dados

Incluir dependências no pom.xml

- Spring-boot-starter-data-jpa
Dependência principal do spring-data para JPA
Tem o Hibernate e outras bibliotecas como dependência
- h2 - banco de dados “embeded”
Feito inteiramente em java
Não precisa de instalação
Pode rodar somente em memória ou armazenar os dados em uma pasta

```
<dependencies>  
  <dependency>  
    <groupId>org.springframework.boot</groupId>  
    <artifactId>spring-boot-starter-web</artifactId>  
  </dependency>  
  
  <dependency>  
    <groupId>org.springframework.boot</groupId>  
    <artifactId>spring-boot-devtools</artifactId>  
  </dependency>  
  
  <dependency>  
    <groupId>org.springframework.boot</groupId>  
    <artifactId>spring-boot-starter-data-jpa</artifactId>  
  </dependency>  
  
  <dependency>  
    <groupId>com.h2database</groupId>  
    <artifactId>h2</artifactId>  
    <scope>runtime</scope>  
  </dependency>  
</dependencies>
```

Usando Postgres - Google: maven postgresql

The screenshot shows the Maven Repository page for the PostgreSQL JDBC Driver version 42.2.15. The page includes a search bar, a sidebar with 'Indexed Artifacts (17.7M)' and 'Popular Categories', and a main content area with details about the driver. The details section includes a license (BSD 2-clause), categories (PostgreSQL Drivers), organization (PostgreSQL Global Development Group), homepage (https://jdbc.postgresql.org), date (Aug 14, 2020), files (jar (979 KB)), repositories (Central), and a list of users (2,274 artifacts). The bottom of the page shows the Maven dependency configuration for the driver.

Home » org.postgresql » postgresql » 42.2.15

PostgreSQL JDBC Driver » 42.2.15
PostgreSQL JDBC Driver PostgreSQL

License: [BSD 2-clause](#)

Categories: [PostgreSQL Drivers](#)

Organization: PostgreSQL Global Development Group

HomePage: <https://jdbc.postgresql.org>

Date: (Aug 14, 2020)

Files: [jar \(979 KB\)](#) [View All](#)

Repositories: [Central](#)

Used By: **2,274 artifacts**

[Maven](#) [Gradle](#) [SBT](#) [Ivy](#) [Grape](#) [Leiningen](#) [Buildr](#)

```
<!-- https://mvnrepository.com/artifact/org.postgresql/postgresql -->
<dependency>
  <groupId>org.postgresql</groupId>
  <artifactId>postgresql</artifactId>
  <version>42.2.15</version>
</dependency>
```

```
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>

  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
  </dependency>

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

  <!-- https://mvnrepository.com/artifact/org.postgresql/postgresql -->
  <dependency>
    <groupId>org.postgresql</groupId>
    <artifactId>postgresql</artifactId>
    <version>42.2.15</version>
  </dependency>
</dependencies>
```

```
spring.datasource.url=jdbc:postgresql://localhost:5432/postgres
spring.datasource.username=postgres
spring.datasource.password=password
```


@Entity - identifica que a classe é uma entidade do banco

@Table - indica o nome da tabela

@Id - indica que o atributo será mapeado como chave primária

@Column - indica

```
@Entity
@Table(name = "todo" )
public class Todo {

    @Id
    private Integer id;

    @Column(name = "titulo", nullable = false, length = 20)
    private String titulo;

    @Column(name = "descricao", nullable = false, length=200)
    private String descricao;
```

- Opcional (será utilizado o nome do atributo como nome do campo, bem como seu tipo)
- Possui diversos parâmetros
 - name - nome do campo
 - nullable - indica se pode receber nulo
 - length - tamanho do campo
 - unique - se o valor do campo é único
 - scale e precision - utilizados para definição de tamanho de números

```
@Column(name = "descricao", nullable = false, length=200)  
private String descricao;
```

- Utilizada para indicar campos do tipo data
- Sempre associada há um atributo do tipo `java.util.Date` (não há suporte para outras classes)
- Usa as constantes de `TemporalType` para indicar o tipo SQL:
 - `TemporalType.DATE` para SQL DATE
 - `TemporalType.TIME` para SQL
 - `TemporalType.TIMESTAMP` para SQL TIMESTAMP

```
//SQL Date
@Temporal(TemporalType.DATE)
private Date dataNascimento;
//SQL Timestamp
@Temporal(TemporalType.TIMESTAMP)
private Date dataCadastro;
//SQL Time
@Temporal(TemporalType.TIME)
private Date horaCompromisso;
```

Permite usar valores gerados automaticamente, o atributo strategy pode ter os seguintes valores:

- GenerationType.IDENTITY - utiliza uma coluna própria do banco de dados que faça o auto incremento dos valores (no Postgres seria uma coluna do tipo Serial)

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

- GenerationType.SEQUENCE - utiliza uma **sequence** no banco requer uma anotação adicional para definir a **sequence**

```
@Id
@GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "sequence-generator")
@SequenceGenerator(name = "sequence-generator", sequenceName = "nome_da_sequence")
private Integer id;
```

- GenerationType.AUTO - tende variar de acordo com a implementação e até com relação a versão, por exemplo, no Hibernate 5.0 houve uma alteração e ele passou a usar TABLE quando se coloca AUTO

```
@Id
@GeneratedValue(strategy = GenerationType.AUTO)
private Integer id;
```

- GenerationType.TABLE - cria uma tabela com estrutura específica para o gerenciamento dos IDs

```
@Id
@GeneratedValue(strategy = GenerationType.TABLE, generator = "tabela_ids")
@TableGenerator(name = "tabela_ids", table = "ids_gerados")
private Integer id;
```

Para criar um repositório, basta criar uma interface e estender uma das interfaces:

- CrudRepository<T,ID> - operações de crud
- PagingAndSortingRepository<T,ID> - operações de paginação
- **JpaRepository<T,ID>** - operações específicas associadas a JPA



```
@Repository
public interface TodoRepository extends JpaRepository<Todo, Integer>{

}
```

CrudRepository

- save - Salva o objeto (INSERT) e retorna o objeto salvo (já com id gerada)
- findOne - retorna um objeto, recebendo o ID como parâmetro (SELECT * WHERE ID=)
- findAll() - retorna todos objetos da tabela (um objeto Iterable) (SELECT *)
- count() - retorna o total de registros na tabela (SELECT COUNT)
- delete - exclui o registro da tabela
- exists - retorna true se o ID existir na tabela (SELECT (COUNT(*)>0) WHERE ID=)

PagingAndSortingRepository

- findAll(Sort) - retorna todos os objetos (um objeto Iterable), mas com suporte a ordenação
- findAll(Pageable) - retorna um objeto Page contendo uma “página” de dados, com base nos valores do objeto Pageable passado

```
Sort sort = new Sort(new Sort.Order(Direction.ASC, "lastName"));
Pageable pageable = new PageRequest(0, 5, sort);
```


JpaRepository

- `findAll()` - retorna uma lista (List) contendo todos os objetos
- `findAll(Sort)` - retorna todos os objetos numa lista (List), mas com suporte a ordenação
- `save(Iterable)` - salva um conjunto de objetos passados num objeto Iterable (uma collection que implemente esta interface), retorna uma lista com os objetos inseridos (salvamento em lote)
- `flush` - JPA pode agrupar operações antes de enviá-las ao servidor de banco de dados, este método “força” que estas sejam enviadas
- `saveAndFlush` - faz as duas operações num único método ;-)
- `deleteInBatch(Iterable)` - recebe um conjunto de objetos passados num objeto Iterable e exclui todos do banco

```
private List<Todo> todos = new ArrayList<Todo>();

public List<Todo> getAll(){
    return todos;
}
private Todo findTodo(Integer id) throws TodoNotFoundException {
    Todo achado = null;
    for(Todo todo: todos) {
        if (todo.getId().equals(id)) {
            achado = todo;
            break;
        }
    }
    if (achado==null) {
        throw new TodoNotFoundException(id);
    }
    return achado;
}

public Todo getTodo(Integer id) throws TodoNotFoundException {
    return findTodo(id);
}
```

```
@Autowired
private TodoRepository todoRepository;

public List<Todo> getAll(){
    List<Todo> todos = todoRepository.findAll();
    return todos;
}

private Todo findTodo(Integer id) throws TodoNotFoundException {
    Optional<Todo> achado = todoRepository.findById(id);
    if (!achado.isPresent()) {
        throw new TodoNotFoundException(id);
    }
    return achado.get();
}

public Todo getTodo(Integer id) throws TodoNotFoundException {
    return findTodo(id);
}
```

```
public Todo addTodo(Todo todo) {
    todos.add(todo);
    return todo;
}

public Todo updateTodo(Integer id, Todo todoNovo) throws TodoNotFoundException {
    Todo achado = findTodo(id);
    if (todoNovo.getId() != null)
        achado.setId(todoNovo.getId());
    if (todoNovo.getTitulo() != null)
        achado.setTitulo(todoNovo.getTitulo());
    if (todoNovo.getDescricao() != null)
        achado.setDescricao(todoNovo.getDescricao());
    return achado;
}

public void deleteTodo(Integer id) throws TodoNotFoundException {
    Todo todo = findTodo(id);
    todos.remove(todo);
}
```

```
public Todo addTodo(Todo todo) {
    Todo todoSalvo = todoRepository.save(todo);
    return todoSalvo;
}

public Todo updateTodo(Integer id, Todo todoNovo) throws TodoNotFoundException {
    Todo achado = findTodo(id);
    if (todoNovo.getId() != null)
        achado.setId(todoNovo.getId());
    if (todoNovo.getTitulo() != null)
        achado.setTitulo(todoNovo.getTitulo());
    if (todoNovo.getDescricao() != null)
        achado.setDescricao(todoNovo.getDescricao());
    achado = todoRepository.save(achado);
    return achado;
}

public void deleteTodo(Integer id) throws TodoNotFoundException {
    Todo todo = findTodo(id);
    todoRepository.delete(todo);
}
```

GET http://localhost:8080/todo

Params Auth Headers (8) Body ● Pre-req. Tests Settings

Body ▾ 200 OK 20

Pretty Raw Preview Visualize JSON

1 []

GET http://localhost:8080/todo

Params Auth Headers (8) Body ● Pre-req. Tests Settings

Body ▾

Pretty Raw Preview Visualize JSON

```
1 [
2   {
3     "id": 1,
4     "titulo": "Almoço",
5     "descricao": "Ligar para o iFood"
6   },
7   {
8     "id": 2,
9     "titulo": "Aula",
10    "descricao": "Preparar o Material"
11  }
12 ]
```

POST http://localhost:8080/todo

Params Auth Headers (8) Body ● Pre-req. Tests Settings

raw JSON

```
1 {
2   "id": 1,
3   "titulo": "Almoço",
4   "descricao": "Ligar para o iFood"
5 }
```

Body ▾

Pretty Raw Preview Visualize

```
1 {
2   "id": 1,
3   "titulo": "Almoço",
4   "descricao": "Ligar para o iFood"
5 }
```

POST http://localhost:8080/todo

Params Auth Headers (8) Body ● Pre-req. Tests Settings

raw JSON

```
1 {
2   "id": 2,
3   "titulo": "Aula",
4   "descricao": "Preparar o Material"
5 }
```

Body ▾

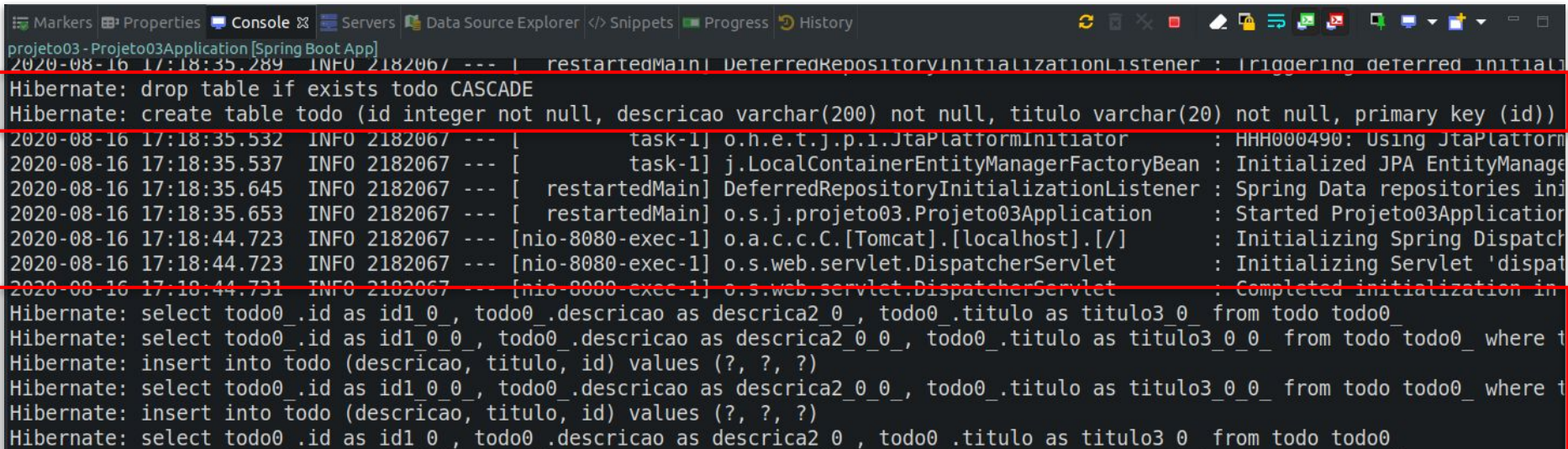
Pretty Raw Preview Visualize JSON

```
1 {
2   "id": 2,
3   "titulo": "Aula",
4   "descricao": "Preparar o Material"
5 }
```


Mas funciona realmente?

```
spring.datasource.url=jdbc:h2:mem:db;DB_CLOSE_DELAY=-1
spring.datasource.username=sa
spring.datasource.password=sa

spring.jpa.show-sql=true
```



Markers Properties Console Servers Data Source Explorer Snippets Progress History

projeto03 - Projeto03Application [Spring Boot App]

2020-08-16 17:18:35.289 INFO 2182067 --- [restartedMain] DeferredRepositoryInitializationListener : triggering deferred initialization

Hibernate: drop table if exists todo CASCADE

Hibernate: create table todo (id integer not null, descricao varchar(200) not null, titulo varchar(20) not null, primary key (id))

2020-08-16 17:18:35.532 INFO 2182067 --- [task-1] o.h.e.t.j.p.i.JtaPlatformInitiator : HHH000490: Using JtaPlatform

2020-08-16 17:18:35.537 INFO 2182067 --- [task-1] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory

2020-08-16 17:18:35.645 INFO 2182067 --- [restartedMain] DeferredRepositoryInitializationListener : Spring Data repositories initialized

2020-08-16 17:18:35.653 INFO 2182067 --- [restartedMain] o.s.j.projeto03.Projeto03Application : Started Projeto03Application

2020-08-16 17:18:44.723 INFO 2182067 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet

2020-08-16 17:18:44.723 INFO 2182067 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'

2020-08-16 17:18:44.731 INFO 2182067 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 1.001 seconds

Hibernate: select todo0_.id as id1_0_, todo0_.descricao as descricao2_0_, todo0_.titulo as titulo3_0_ from todo todo0_

Hibernate: select todo0_.id as id1_0_, todo0_.descricao as descricao2_0_, todo0_.titulo as titulo3_0_ from todo todo0_ where todo0_.id=1

Hibernate: insert into todo (descricao, titulo, id) values (?, ?, ?)

Hibernate: select todo0_.id as id1_0_, todo0_.descricao as descricao2_0_, todo0_.titulo as titulo3_0_ from todo todo0_ where todo0_.id=1

Hibernate: insert into todo (descricao, titulo, id) values (?, ?, ?)

Hibernate: select todo0_.id as id1_0_, todo0_.descricao as descricao2_0_, todo0_.titulo as titulo3_0_ from todo todo0_

Alterar a url no arquivo de propriedades













```
spring.datasource.url=jdbc:h2:file:./h2/banco;DB_CLOSE_DELAY=-1
spring.datasource.username=sa
spring.datasource.password=sa

spring.jpa.show-sql=true

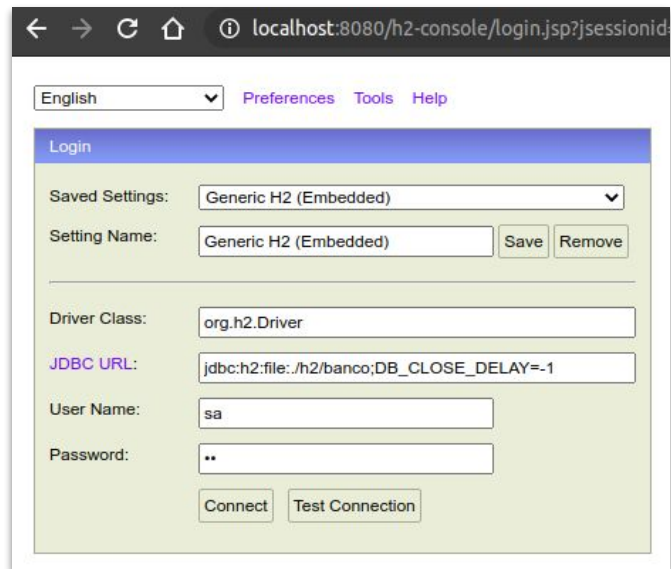
spring.h2.console.enabled=true
```

Após iniciar o projeto e fazer um “refresh” na estrutura do projeto (F5) a pasta h2 deve aparecer

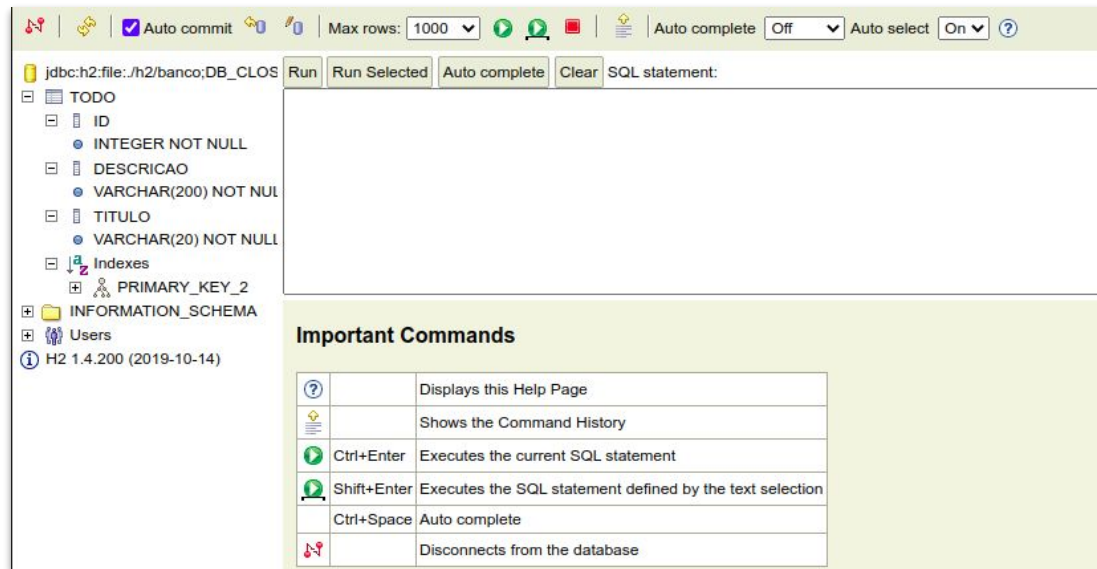
Depois basta acessar <http://localhost:8080/h2-console>

- ▶  src/main/java
- ▶  src/main/resources
- ▶  src/test/java
- ▶  JRE System Library [JavaSE-1.8]
- ▶  Maven Dependencies
- ▶  h2
- ▶  src
- ▶  target
- ▶  HELP.md
- ▶  mvnw
- ▶  mvnw.cmd
- ▶  pom.xml

Acessar com os mesmos dados utilizados no arquivo de propriedades (url, username e password)



The screenshot shows the H2 Console login page. At the top, there's a navigation bar with a language dropdown set to 'English' and links for 'Preferences', 'Tools', and 'Help'. Below this is a 'Login' section with a 'Saved Settings' dropdown menu currently showing 'Generic H2 (Embedded)'. Underneath, there's a 'Setting Name' field with the same text and 'Save' and 'Remove' buttons. The main login area contains four input fields: 'Driver Class' with 'org.h2.Driver', 'JDBC URL' with 'jdbc:h2:file:./h2/banco;DB_CLOSE_DELAY=-1', 'User Name' with 'sa', and 'Password' with two asterisks. At the bottom of this section are 'Connect' and 'Test Connection' buttons.



The screenshot shows the main interface of the H2 Console. At the top, there's a toolbar with icons for undo, redo, and other actions, along with a status bar showing 'Auto commit' is checked, 'Max rows' is 1000, 'Auto complete' is Off, and 'Auto select' is On. Below the toolbar is a tree view on the left showing the database structure: 'jdbc:h2:file:./h2/banco;DB_CLOSE' (expanded), 'TODO' (expanded), 'ID' (INTEGER NOT NULL), 'DESCRICAO' (VARCHAR(200) NOT NULL), 'TITULO' (VARCHAR(20) NOT NULL), 'Indexes' (expanded), 'PRIMARY_KEY_2', 'INFORMATION_SCHEMA', and 'Users'. The main area on the right is for SQL statements, with a 'Run' button and a 'SQL statement:' input field. At the bottom right, there's a section titled 'Important Commands' with a table of shortcuts.

Icon	Shortcut	Action
?		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

Run Run Selected Auto complete Clear SQL statement:

SELECT * FROM TODO

POST

http://localhost:8080/todo

Params Auth Headers (8) Body Pre-req. Tests

raw

JSON

```
1 {
2   "id":1,
3   "titulo": "Almoço",
4   "descricao": "Ligar para o iFood"
5 }
```

Body

Pretty

Raw

Preview

Visualize

JSON

```
1 {
2   "id": 1,
3   "titulo": "Almoço",
4   "descricao": "Ligar para o iFood"
5 }
```

Run

Run Selected

Auto complete

Clear

SQL statement:

SELECT * FROM TODO |

SELECT * FROM TODO;

ID	DESCRICAO	TITULO
----	-----------	--------

(no rows, 6 ms)

SELECT * FROM TODO;

ID	DESCRICAO	TITULO
1	Ligar para o iFood	Almoço

Quem criou as tabelas????

Duas propriedades referentes a DDL (Data Definition Language do SQL - CREATE, DROP, ALTER ...)

- `spring.jpa.generate-ddl` - indica se o spring deve ser responsável pela ddl (true ou false)
- `spring.jpa.hibernate.ddl-auto` - como o hibernate irá se comportar:
 - `none`: não realiza nenhuma operação
 - `validate`: valida a estrutura do banco com as entidades mapeadas, se houver diferenças
 - `update`: atualiza a estrutura do banco de acordo com as entidades (não exclui colunas, por segurança)
 - `create`: recria a estrutura do banco sempre
 - `create-drop`: cria a estrutura e a apaga ao final da sessão

O Spring tem um comportamento padrão diferente de acordo com o tipo de banco de dados.

- Bancos embodes (H2, HSQLDB e Derby) considera um ambiente de dev e usa `create-drop` por padrão
- Para outros bancos ele considera `none`

*Recomendação para ambiente de trabalho: deixar opções em `false` e `none` e, ou usar um script manual de banco de dados ou utilizar frameworks de gerenciamento de versão de banco, como o **liquibase** ou o **flyway***

- **spring.datasource.driverClassName** - classe do driver jdbc (org.h2.Driver ou org.postgres.Driver) - não é necessária pois o hibernate consegue identificar qual driver a partir da url, mas para alguns bancos ou situações específicas pode ser necessário configurar
- **spring.jpa.database-platform** - indica a classe do hibernate responsável por tratar o “dialeto” sql do banco
 - org.hibernate.dialect.PostgreSQLDialect
 - org.hibernate.dialect.H2Dialect
- **spring.jpa.properties.hibernate.format_sql** - se ao exibir o sql no console/log, ele deve estar formatado (identado) - (true ou false)

Incluir dependência do spring-boot-starter-validation no pom.xml

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

Incluir anotações de validação na entidade e no controller

```
@Entity
@Table(name = "todo")
public class Todo {

    @Id
    private Integer id;

    @NotNull
    @Size(min = 2, max = 20)
    @Column(name = "titulo", nullable = false, length = 20)
    private String titulo;
```

```
@GetMapping
public List<Todo> getAll() {
    return todoService.getAll();
}

@PostMapping
public Todo postTodo(@Valid @RequestBody Todo todo) {
    return todoService.addTodo(todo);
}
```

Cria o método “handler” para tratar a exceção de argumento inválido (@Valid força a validação)

```
@RestControllerAdvice
public class ExceptionsController {

    @ResponseStatus(HttpStatus.BAD_REQUEST)
    @ExceptionHandler(MethodArgumentNotValidException.class)
    public Map<String, String> handleValidationExceptions(MethodArgumentNotValidException ex) {
        //cria um map para tratar os erros
        Map<String, String> errosOcorridos= new HashMap<>();
        //recupera a lista de erros
        List<ObjectError> erros = ex.getBindingResult().getAllErrors();
        for(ObjectError erro:erros) { //para cada erro
            //pega o atributo onde ocorreu o erro
            String atributo = ((FieldError) erro).getField();
            //pega a mensagem de erro
            String mensagem= erro.getDefaultMessage();
            //adiciona no map
            errosOcorridos.put(atributo, mensagem);
        }
        //retorna o map
        return errosOcorridos;
    }
}
```


POST ▼ http://localhost:8080/todo

Params Auth Headers (8) Body ● Pre-req. Tests Settings

raw ▼ JSON ▼

```
1 {  
2   "id":1,  
3   "descricao": "Ligar para o iFood"  
4 }
```

Body ▼ 🌐 400 Bad Request

Pretty Raw Preview Visualize JSON ▼ 

```
1 {  
2   "titulo": "não deve ser nulo"  
3 }
```


POST ▼ http://localhost:8080/todo

Params Auth Headers (8) Body ● Pre-req. Tests Settings

raw ▼ JSON ▼

```
1 {  
2   "id":1,  
3   "titulo": "a",  
4   "descricao": "Ligar para o iFood"  
5 }
```

Body ▼ 🌐 400 Bad Request

Pretty Raw Preview Visualize JSON ▼ 

```
1 {  
2   "titulo": "tamanho deve ser entre 2 e 20"  
3 }
```

- Max - valor máximo inteiro (int)
- Min - valor mínimo inteiro (int)
- DecimalMax - Valor máximo numérico (BigDecimal)
 - `@DecimalMax(value = "100000.0", inclusive = true)`
 - `@DecimalMax("1.99")`
- DecimalMin - Valor mínimo numérico (BigDecimal)
 - `@DecimalMin(value = "0.0", inclusive = false)`
- Digits - numero de digitos permitidos, na parte inteira e na decimal (BigDecimal)
 - `@Digits(integer=3, fraction=2)`
- Future - valida se a data é no futuro (Date)
- Past - valida se a data é no passado (Date)
- NotNull - não pode ser nulo
- Null - deve ser nulo
- Pattern - deve ser validado por expressão regular
 - `@Pattern(regexp="\\(\\d{3}\\)\\d{3}-\\d{4}")`
- Size - define os limites aceitos, (String ou Coleções)
 - `@Size(min=2, max=10)`

Criar uma aplicação para gerenciar uma biblioteca de livros armazenando no banco de dados

- Entidade Livro
 - Id - identificador (número, sequence, chave primária)
 - título - título do livro (texto, obrigatório, mínimo 5 e máximo 30 caracteres)
 - tipo - tipo do livro (fantasia, técnico, romance) (texto, obrigatório, mínimo 3 máximo 20)
 - autor - nome do autor do livro (texto, obrigatório, mínimo 10, máximo 40)
 - data de publicação - data de publicação do livro (deve ser uma data anterior a data atual)
- API
 - Todas as operações CRUD no tópico /livro
- Bonus
 - No endpoint GET /livro possibilidade de passar um parâmetro ?ordem=campo para que a listagem venha ordenada por aquele campo

Pesquisar:

- Como passar a data em json para o controller (qual o formato)?
- Como ordenar no findAll do repositório