

# Módulo 06

Tainá Medeiros



# Spring Boot

É uma ferramenta de alavancagem de produtividade e qualidade.

O Spring Boot não apenas ajuda você a escolher o que você irá usar no seu projeto, como ele garante que as versões sejam compatíveis com o restante das bibliotecas.

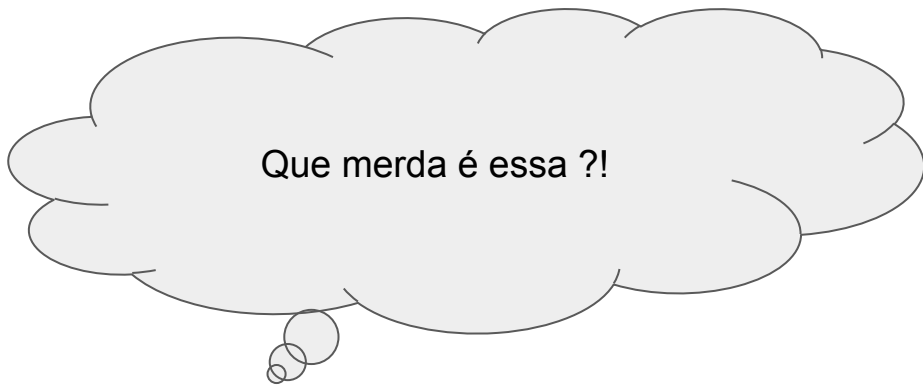
Ele também te ajuda com as configurações, ou seja, uma configuração padrão para cada item que você escolheu.

Resumidamente, é uma coisa linda!

Podem confiar em mim ;)

# Como funciona....

O Spring Boot escaneia o classpath do projeto buscando por algumas bibliotecas, dependendo da biblioteca que ele encontrar ele aplica um configuração padrão.



# Como funciona....


Por exemplo,

- A. se ele encontra as bibliotecas do Spring JPA e de algum banco de dados ele já sabe aquelas configurações padrões que todos nós vemos nas documentações oficiais como:
  - a. nome de usuário do banco, endereço, porta, senha, driver e tudo mais.
- B. Se ele encontra a biblioteca do Spring Web:
  - a. todas as configurações, locais das páginas web, container, porta....

Tudo isso será criado!

# Por onde começar...

1. Pode fazer “na mão”
2. Pode usar o Spring Boot pela linha de comando
3. Pode utilizar uma IDE
4. Pode utilizar o Spring Initializr.
  - a. <https://start.spring.io/>



Spring **Initializr**  
Bootstrap your application

**Project**

**Language**

**Spring Boot**

**Project Metadata**

**Maven Project** **Gradle Project**

**Java** **Kotlin** **Groovy**

**2.3.0 M2** **2.3.0 (SNAPSHOT)** **2.2.6 (SNAPSHOT)**

Group  
com.example

Artifact  
demo



**KEEP  
CALM  
AND  
USE THE  
FORCE**



# Iniciando...

1. Criar um projeto pelo Spring Initializr.
  
2. Com as seguintes dependências:
  - a. H2 Database
  - b. Spring Data JPA
  - c. Spring Web
  - d. Spring Boot DevTools



# IntelliJ IDEA

Version 2019.2.3

+ Create New Project

📄 Import Project

📁 Open

↶ Check out from Version Control ▼

- ☐ Create project from existing sources
- ☒ Import project from external model

🤖 Android Gradle

🌐 Eclipse

FB Flash Builder

🐘 Gradle

m Maven



Project

modulo06 ~/Downloads/Codenation-

.idea
.mvn
src
main
java
com.codenation.modulo06
Modulo06Application
resources
test
target
.gitignore
HELP.md
mvnw
mvnw.cmd
pom.xml

pom.xml x
Modulo06Application.java x

```

1      package com.codenation.modulo06;
2
3      import ...
8
9      @SpringBootApplication
10     public class Modulo06Application {
11
12         public static void main(String[] args) { SpringApplication.run(Modulo06Application.class, args); }
15
16         @Bean
17         public CommandLineRunner commandLineRunner() {
18             return args -> {
19                 System.out.println("----- It's ALIVEEEEEEEEEEE -----");
20             };
21         }
22     }
23

```

Run: Modulo06Application x

Console
Endpoints

```

2020-02-27 22:52:55.110 INFO 27503 --- [main] org.springframework.boot.SpringApplication : Starting application
2020-02-27 22:52:55.124 INFO 27503 --- [main] j.LocalContainerEntityManagerFactoryBean : Initialize
2020-02-27 22:52:55.269 WARN 27503 --- [main] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa
2020-02-27 22:52:55.429 INFO 27503 --- [main] o.s.s.concurrent.ThreadPoolTaskExecutor : Initializ
2020-02-27 22:52:55.677 INFO 27503 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat sta
2020-02-27 22:52:55.680 INFO 27503 --- [main] c.c.modulo06.Modulo06Application : Started Mo
----- It's ALIVEEEEEEEEEEE -----

```

# Entendo o Spring Data JPA

- O Spring Data JPA é um framework que nasceu para facilitar a criação dos nossos repositórios.
- Ele faz isso nos liberando de ter que implementar as interfaces referentes aos nossos repositórios (ou DAOs), e também já deixando pré-implementado algumas funcionalidades como, por exemplo,
  - ordenação das consultas
  - paginação de registros.

**Tem como objetivo facilitar nosso trabalho com persistência de dados de uma forma geral**

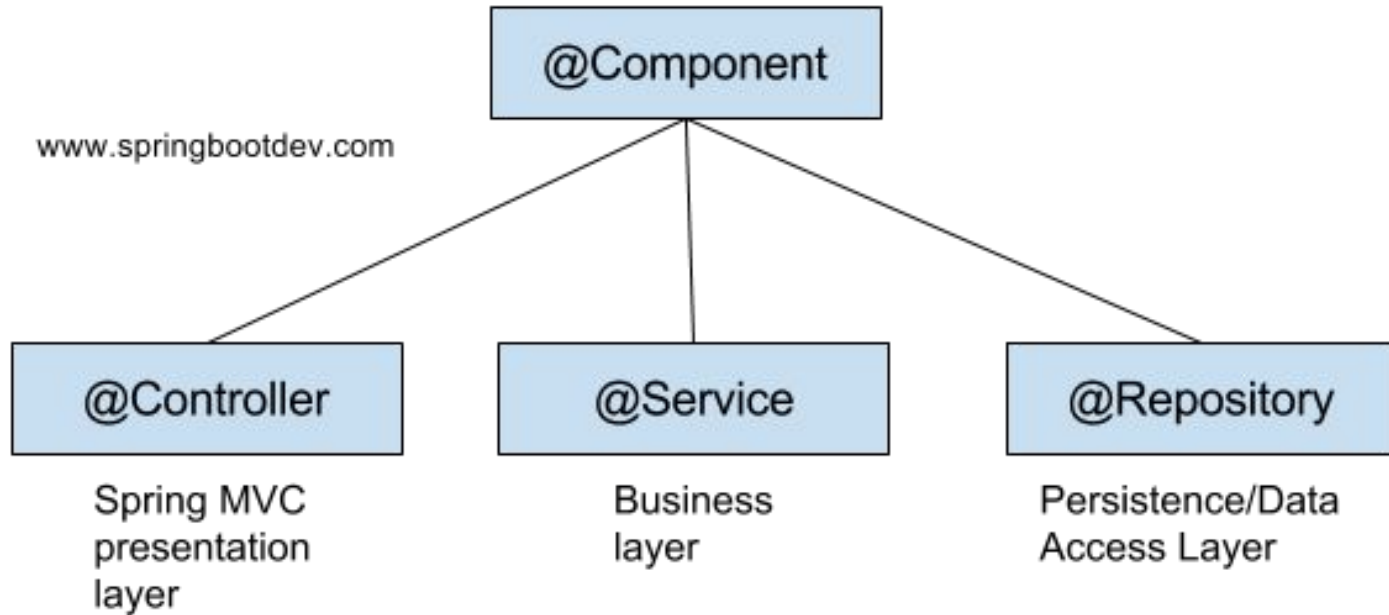
# Design Pattern

JPA é somente a tecnologia de acesso ao banco de dados.

Com ele você pode usar os Design Patterns (padrão de design)

- que não fazem parte do JPA especificamente.

# Design Pattern



# Design Pattern

**Repository** é um Design Pattern onde os dados são obtidos do banco de dados e ocorre também a regra de negócio. Este retorna objetos de domínio que seriam as Entidades (classes anotadas com @Entity).

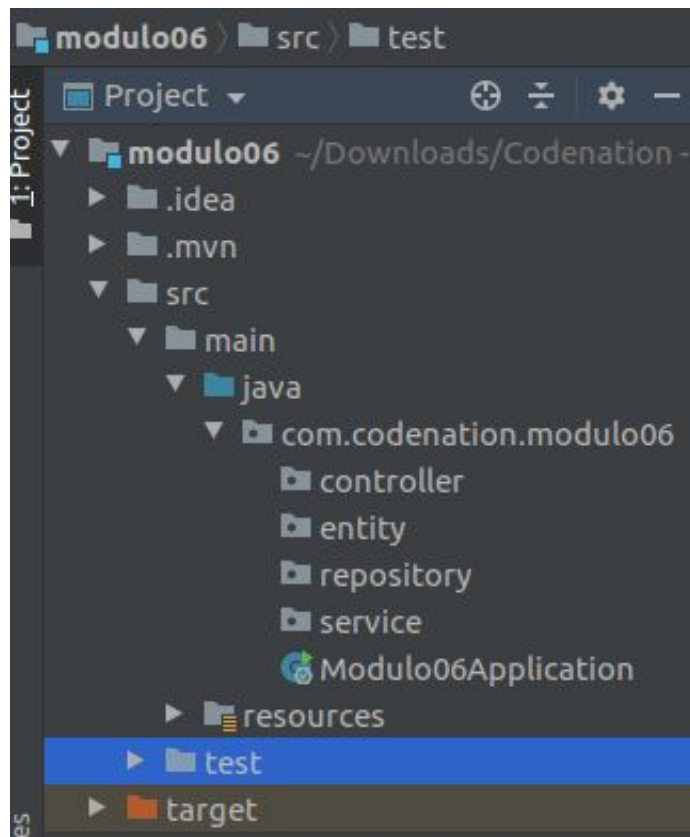
**DAO** é outro Design Pattern onde somente há a comunicação com o banco de dados sem regra de negócio.

**Service** seria outro Design Pattern onde há somente a regra de negócio e não tem acesso direto ao banco de dados.

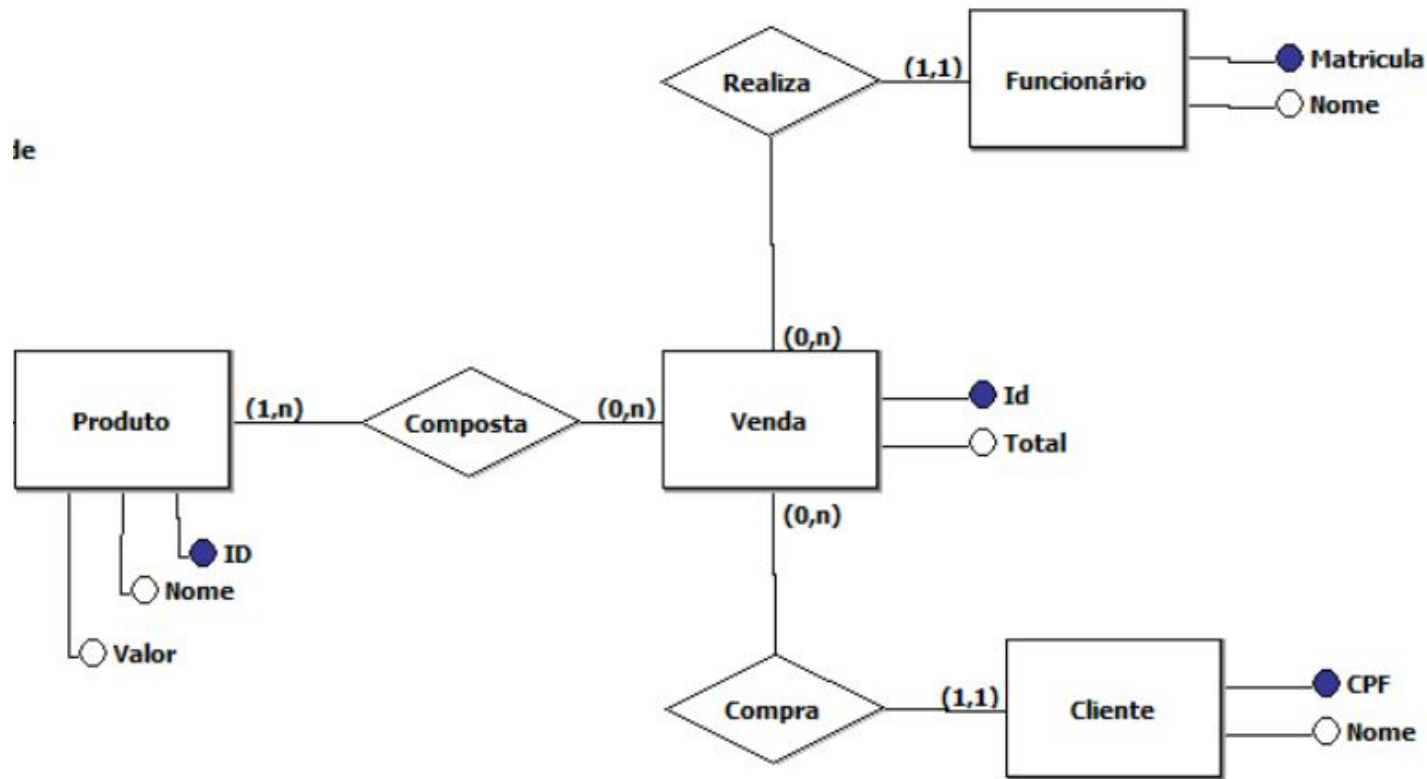
**Controller** Ele é utilizado para lidar com a ligação da View com as outras partes do sistema que são a regra de negócio e banco de dados.



# Criando a estrutura dos pacotes

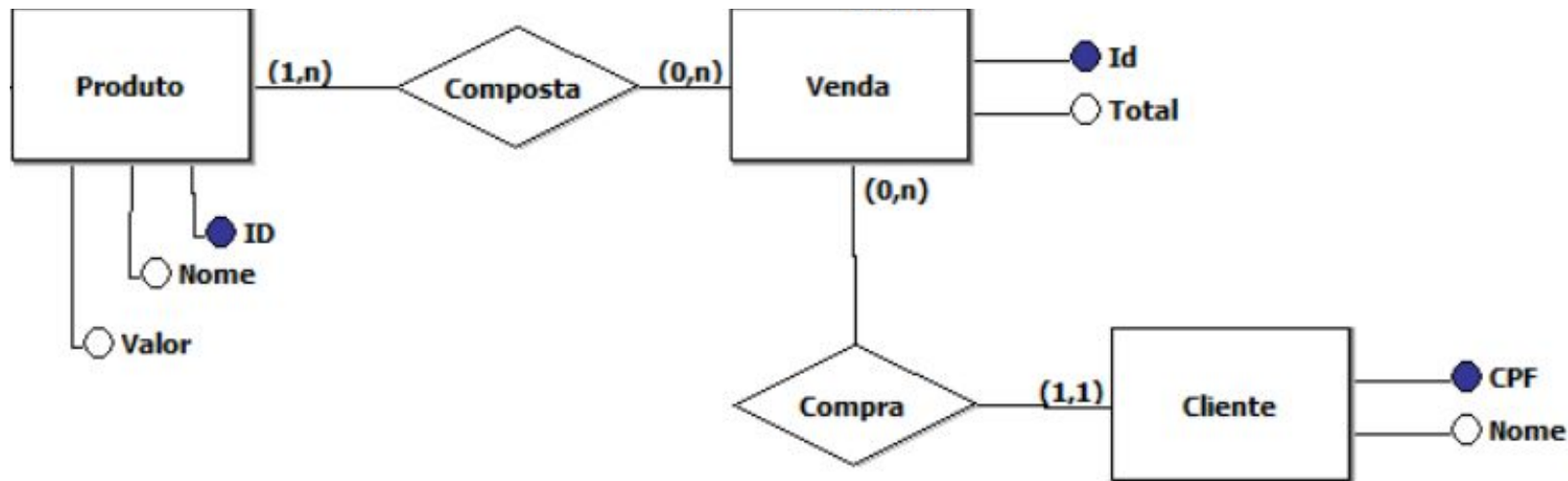


Lembra?





Vamos implementar...



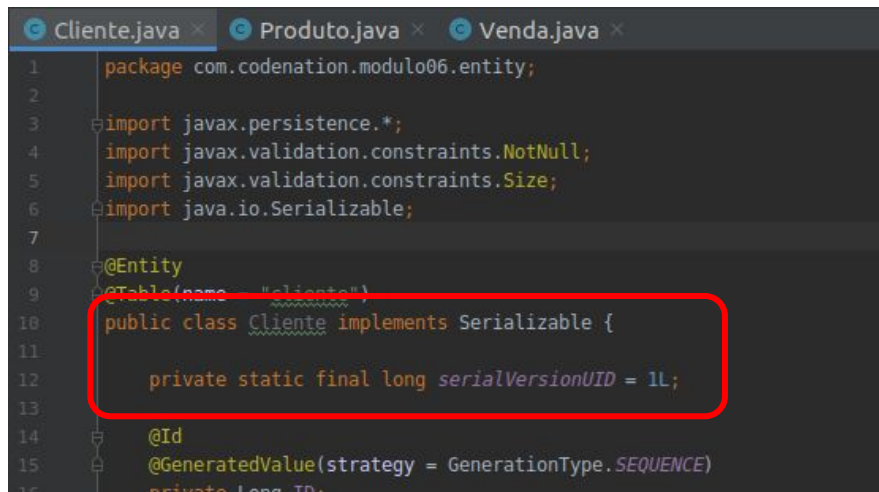
# Criando nossas entidades



Cliente.java x Produto.java x Venda.java x

```
1 package com.codenation.modulo06.entity;
2
3 import javax.persistence.*;
4 import javax.validation.constraints.NotNull;
5 import javax.validation.constraints.Size;
6 import java.io.Serializable;
7
8 @Entity
9 @Table(name = "cliente")
10 public class Cliente implements Serializable {
11
12     private static final long serialVersionUID = 1L;
13
14     @Id
15     @GeneratedValue(strategy = GenerationType.SEQUENCE)
16     private Long ID;
17
18     @NotNull
19     @Column(name = "cpf")
20     @Size(min = 11, max = 11)
21     private String cpf;
22
23     @NotNull
24     @Column(name = "nome")
25     @Size(min = 1, max = 100)
26     private String nome;
27
28     //Getter and Setter
29
30     public Long getID() {
31         return ID;
```

# Curiosidade...



```
1 package com.codenation.modulo06.entity;
2
3 import javax.persistence.*;
4 import javax.validation.constraints.NotNull;
5 import javax.validation.constraints.Size;
6 import java.io.Serializable;
7
8 @Entity
9 @Table(name = "cliente")
10 public class Cliente implements Serializable {
11
12     private static final long serialVersionUID = 1L;
13
14     @Id
15     @GeneratedValue(strategy = GenerationType.SEQUENCE)
16     private Long ID;
```

A serialização em Java é o processo no qual a instância de um objeto é transformada em uma sequência de bytes e é útil quando precisamos enviar objetos pela rede, salvar no disco, ou comunicar de uma JVM com outra.

Isso porque o estado atual do objeto é “congelado” e na outra “ponta” nós podemos “descongelar” este objeto sem perder nenhuma informação.

Criando os relacionamentos...



# Criando os relacionamentos...

Hibernate facilita o armazenamento e a recuperação de objetos Java através do Mapeamento Objeto-Relacional (Object/Relational Mapping - ORM)

@OneToMany

@ManyToMany

@ManyToOne

@OneToOne



# Criando os relacionamentos...

```
21     @OneToOne
22     @JoinColumn(name = "idCliente")
23     private Cliente cliente;
24
25     @OneToMany
26     private List<Produto> produtos;
27
```

```
46     public Cliente getCliente() {
47         return cliente;
48     }
49
50     public void setCliente(Cliente cliente) {
51         this.cliente = cliente;
52     }
53
54     public List<Produto> getProdutos() {
55         return produtos;
56     }
57
58     public void setProdutos(List<Produto> produtos) {
59         this.produtos = produtos;
60     }

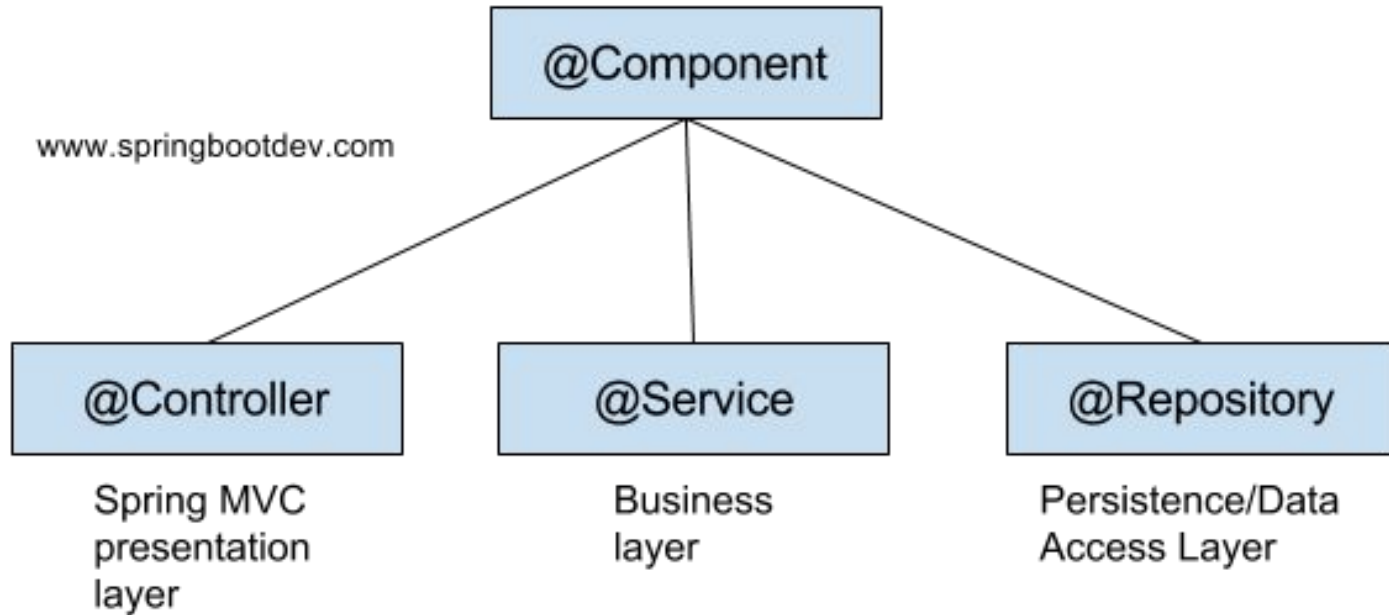
```

# Criando Design Pattern





# Criando Design Pattern



# Repository

Camada para interação com modelos e execução de operações de banco de dados

```
1 package com.codenation.modulo06.repository;
2
3 import com.codenation.modulo06.entity.Cliente;
4 import org.springframework.data.jpa.repository.JpaRepository;
5 import org.springframework.stereotype.Repository;
6
7 @Repository
8 public interface ClienteRepository extends JpaRepository<Cliente, Long> {
9
10 }
```

# Repository (Melhorando o código)

```
7  @Repository
8  public interface ClienteRepository extends JpaRepository<Cliente, Long> {
9
10     Cliente findById(long id);
11 }
```

# Service

O middleware entre o controlador e o repositório. Reúne dados do controlador, execute validação e lógica de negócios e chama o repositório para manipulação de dados.

```
8  @Service
9  public class ClienteService {
10
11      @Autowired
12      private ClienteRepository clienteRepository;
13
14      //Adicionar Cliente
15      public Object add(Object object) {
16          return clienteRepository.save((Cliente) object);
17      }
18
19      //Mostrar um Cliente
20      public Object get(long id) {
21          return clienteRepository.findById(id);
22      }
23
24      //Atualiza dados do Cliente
25      public Object update(Object object) {
26          return clienteRepository.save((Cliente) object);
27      }
28  }
```

# Service (Melhorando o código)

```
10  @Service
11  public class ClienteService {
12
13      @Autowired
14      private ClienteRepository clienteRepository;
15
16      //Adicionar Cliente
17      public Cliente add(Object object) {
18          return clienteRepository.save((Cliente) object);
19      }
20
21      //Pesquisar um Cliente
22      public Cliente get(long id) {
23          return clienteRepository.findById(id);
24      }
25
26      //Atualiza dados do Cliente
27      public Cliente update(Object object) {
28          return clienteRepository.save((Cliente) object);
29      }
30
31      //Conta quando clientes foram inseridos no banco
32      public long quantidaClientes() {
33          return clienteRepository.count();
34      }
35  }
```

# Controller

Contém a lógica do aplicativo e passa dados de entrada do usuário para o serviço

```
10  @RestController
11  @RequestMapping("/api")
12  public class ClienteController {
13      |
14      @Autowired
15      private ClienteService clienteService;
16
17      @PostMapping("/cliente")
18      public ResponseEntity<Cliente> addCliente(@RequestBody Cliente pessoa){
19          try{
20              return new ResponseEntity<>((Cliente)clienteService.add(pessoa), HttpStatus.CREATED);
21          }catch (Exception e){
22              return new ResponseEntity<>(HttpStatus.BAD_REQUEST);
23          }
24      }
```

# Controller

```
25
26     @GetMapping("/cliente/{id}")
27     public ResponseEntity<Cliente> getCliente(@PathVariable(value = "id") long id) {
28         try {
29             return new ResponseEntity<>((Cliente)clienteService.get(id), HttpStatus.OK);
30         } catch (Exception e){
31             return new ResponseEntity<>(HttpStatus.BAD_REQUEST);
32         }
33     }
34
35     @PutMapping("/pessoa/")
36     public ResponseEntity<Cliente> updateConta(@RequestBody Cliente conta){
37         try{
38             return new ResponseEntity<>((Cliente) clienteService.update(conta), HttpStatus.OK);
39         } catch (Exception e){
40             return new ResponseEntity<>(HttpStatus.BAD_REQUEST);
41         }
42     }
43 }
```

Se tentarmos executar o projeto, vai funcionar?





# Banco de Dados H2

O H2 é um banco de dados Open Source que funciona em memória com um console acessível pelo browser dentro do contexto da aplicação.

Como ele funciona em memória todo seu armazenamento é volátil, ou seja, a cada sobe e desce da aplicação ele será reconstruído.

Seu intuito é ser um banco de configuração rápida e fácil, visando favorecer a produtividade.

# Configurando o H2

Passo 01.

Integrando o H2 com o Spring Boot

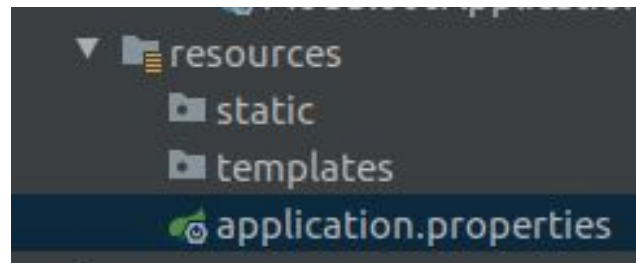
```
<dependency>  
  <groupId>com.h2database</groupId>  
  <artifactId>h2</artifactId>  
  <scope>runtime</scope>  
</dependency>
```



# Configurando o H2

Passo 02.

Configurar o acesso ao banco de dados



```
1  spring.h2.console.enabled=true
2  spring.h2.console.path=/h2
3  spring.datasource.url=jdbc:h2:mem:teste
4  spring.datasource.driver-class-name=org.h2.Driver
5  spring.datasource.username=sa
6  spring.datasource.password=password
7  spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
8  server.port=8181
9  spring.jpa.hibernate.ddl-auto=create
10 spring.jpa.show-sql=true
```

# Configurando o H2

Ativar o console web e para especificar um path para acessar respectivamente.

```
1  spring.h2.console.enabled=true
2  spring.h2.console.path=/h2
3  spring.datasource.url=jdbc:h2:mem:teste
4  spring.datasource.driver-class-name=org.h2.Driver
5  spring.datasource.username=sa
6  spring.datasource.password=password
7  spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
8  server.port=8181
9  spring.jpa.hibernate.ddl-auto=create
10 spring.jpa.show-sql=true
```

Configuram a conexão com o banco de dados em h2 que será criado ao iniciar a aplicação.

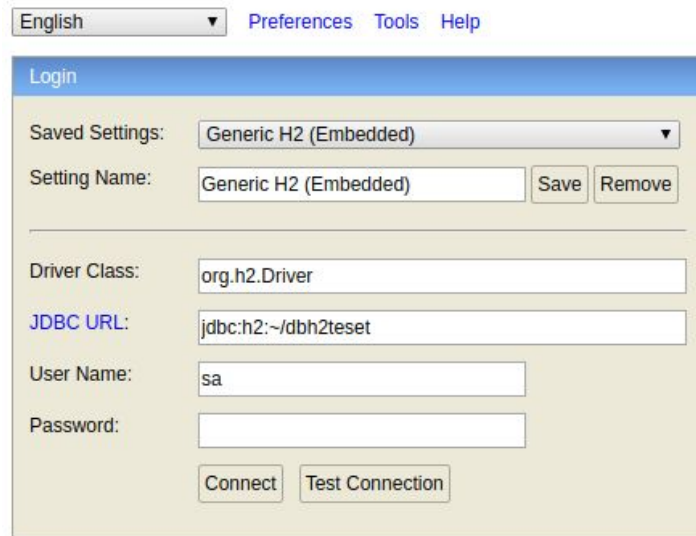
# Configurando o H2

Passo 03.

Acesse o console Web.

basta abrir o browser e digitar o seguinte local:

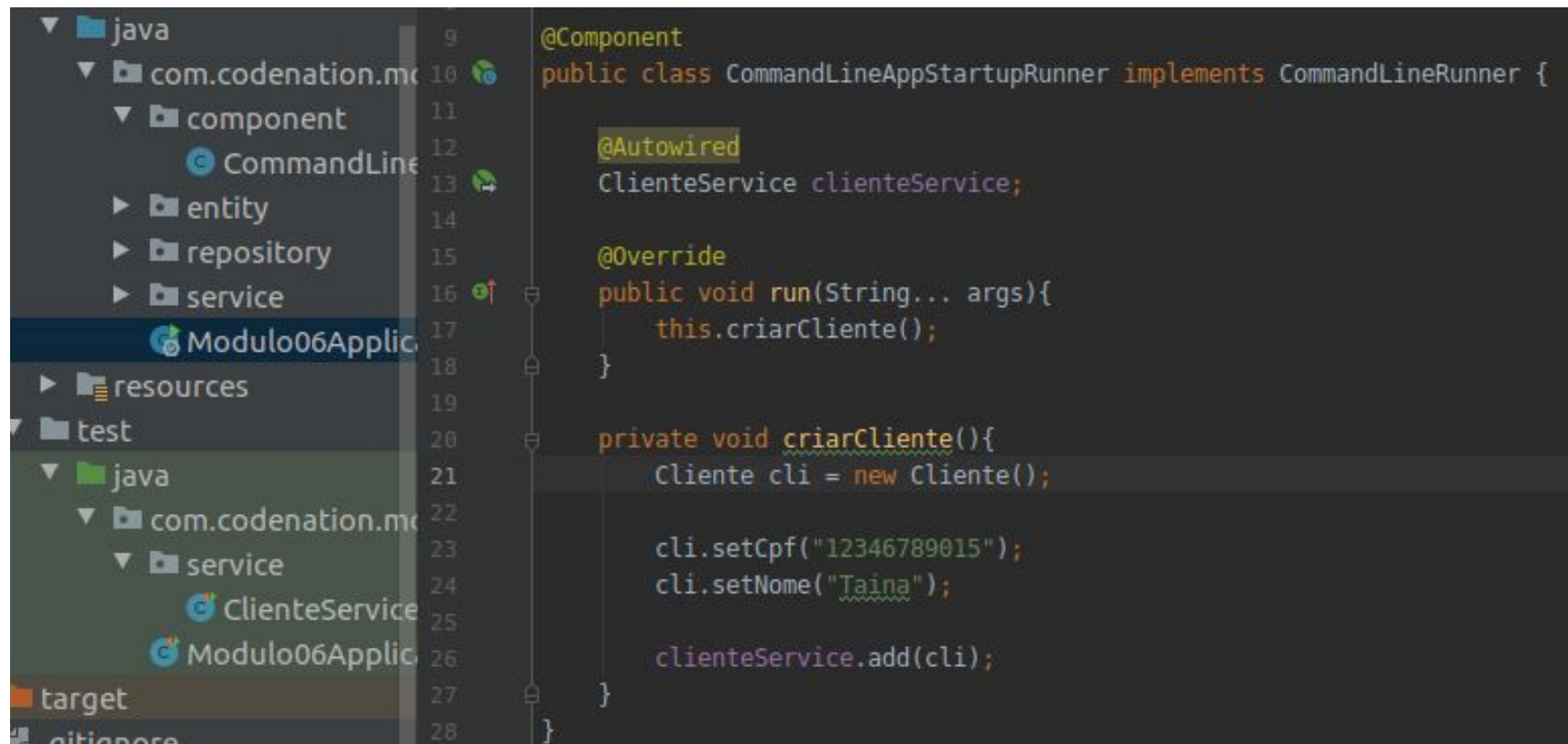
`http://localhost:8080/h2`



The screenshot shows the H2 database web console configuration page. At the top, there is a language dropdown menu set to 'English' and navigation links for 'Preferences', 'Tools', and 'Help'. The main section is titled 'Login' and contains the following fields and buttons:

- Saved Settings:** A dropdown menu showing 'Generic H2 (Embedded)'.
- Setting Name:** A text input field containing 'Generic H2 (Embedded)', with 'Save' and 'Remove' buttons to its right.
- Driver Class:** A text input field containing 'org.h2.Driver'.
- JDBC URL:** A text input field containing 'jdbc:h2:~/dbh2teset'.
- User Name:** A text input field containing 'sa'.
- Password:** An empty text input field.
- Buttons:** 'Connect' and 'Test Connection' buttons at the bottom.

# E para rodar?



The screenshot shows an IDE with a project structure on the left and a code editor on the right. The project structure includes a 'java' package with sub-packages 'com.codenation.module1', 'component', 'entity', 'repository', and 'service'. The 'component' package contains a 'CommandLineAppStartupRunner' class. The 'resources' folder contains a 'test' folder, which in turn contains a 'java' package with a 'com.codenation.module1' package, which contains a 'service' package. The 'service' package contains a 'ClienteService' class and a 'Modulo06Application' class. The code editor shows the implementation of the 'CommandLineAppStartupRunner' class, which implements the 'CommandLineRunner' interface. The class is annotated with '@Component' and '@Autowired'. It has a 'run' method that calls 'criarCliente()' and a 'criarCliente()' method that creates a 'Cliente' object, sets its 'cpf' and 'nome' attributes, and adds it to the 'clienteService'.

```
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28

@Component
public class CommandLineAppStartupRunner implements CommandLineRunner {

    @Autowired
    ClienteService clienteService;

    @Override
    public void run(String... args){
        this.criarCliente();
    }

    private void criarCliente(){
        Cliente cli = new Cliente();

        cli.setCpf("12346789015");
        cli.setNome("Taina");

        clienteService.add(cli);
    }
}
```

# Se tudo deu certo...

The screenshot shows a database client window with a toolbar at the top containing icons for undo, redo, and a checked 'Auto commit' box. To the right of the toolbar is a 'Max rows' dropdown set to '1000' and a green play button. On the left, a tree view shows the database structure for 'jdbc:h2:mem:teste', including tables like CLIENTE, PRODUTO, VENDA, and VENDA\_PRODUTOS, as well as INFORMATION\_SCHEMA, Sequences, and Users. The main area on the right contains a text editor with the SQL query 'SELECT \* FROM CLIENTE |'. Below the text editor are three buttons: 'Run', 'Run Selected', and 'Auto complete'. At the bottom, the results of the query are displayed in a table format, showing one row with columns ID, CPF, and NOME. Below the table, it indicates '(1 row, 10 ms)'.

Auto commit

Max rows: 1000

jdbc:h2:mem:teste

- + CLIENTE
- + PRODUTO
- + VENDA
- + VENDA\_PRODUTOS
- + INFORMATION\_SCHEMA
- + Sequences
- + Users

H2 1.4.200 (2019-10-14)

Run Run Selected Auto complete

SELECT \* FROM CLIENTE |

SELECT \* FROM CLIENTE;

ID	CPF	NOME
1	12346789015	Taina

(1 row, 10 ms)

# Como criar Query

O módulo JPA suporta a definição de uma consulta manualmente como uma String ou a derivação do nome do método.

Embora seja bastante conveniente obter uma consulta derivada do nome do método, pode-se enfrentar a situação na qual o analisador de nome do método não suporta a palavra-chave que deseja usar ou o nome do método se tornaria desnecessariamente feio. Portanto,

- você pode usar consultas nomeadas JPA por meio de uma convenção de nomenclatura,
- ou anotar seu método de consulta com `@Query`



# Como criar Query

Geralmente, o mecanismo de criação de consulta para JPA funciona como descrito em "Métodos de consulta".

```
public interface UserRepository extends Repository<User, Long> {  
  
    List<User> findByEmailAddressAndLastname(String emailAddress, String lastname);  
}
```

```
select u from User u where u.emailAddress = ?1 and u.lastname = ?2
```

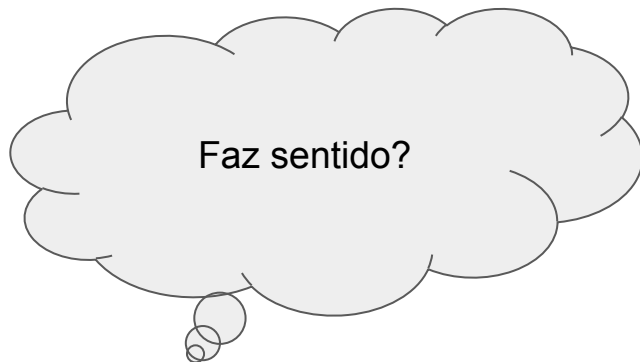
Keyword	Sample	JPQL snippet
And	findByLastnameAndFirstname	... where x.lastname = ?1 and x.firstname = ?2
Or	findByLastnameOrFirstname	... where x.lastname = ?1 or x.firstname = ?2
Is , Equals	findByFirstname , findByFirstnameIs , findByFirstnameEquals	... where x.firstname = ?1
Between	findByStartDateBetween	... where x.startDate between ?1 and ?2
LessThan	findByAgeLessThan	... where x.age < ?1
LessThanEqual	findByAgeLessThanEqual	... where x.age <= ?1
GreaterThan	findByAgeGreaterThan	... where x.age > ?1
GreaterThanEqual	findByAgeGreaterThanEqual	... where x.age >= ?1
After	findByStartDateAfter	... where x.startDate > ?1
Before	findByStartDateBefore	... where x.startDate < ?1

IsNull, Null	findByAge(Is)Null	... where x.age is null
IsNotNull, NotNull	findByAge(Is)NotNull	... where x.age not null
Like	findByFirstnameLike	... where x.firstname like ?1
NotLike	findByFirstnameNotLike	... where x.firstname not like ?1
StartingWith	findByFirstnameStartingWith	... where x.firstname like ?1 (parameter bound with appended %)
EndingWith	findByFirstnameEndingWith	... where x.firstname like ?1 (parameter bound with prepended %)
Containing	findByFirstnameContaining	... where x.firstname like ?1 (parameter bound wrapped in %)

OrderBy	findByAgeOrderByLastnameDesc	... where x.age = ?1 order by x.lastname desc
Not	findByLastnameNot	... where x.lastname <> ?1
In	findByAgeIn(Collection<Age> ages)	... where x.age in ?1
NotIn	findByAgeNotIn(Collection<Age> ages)	... where x.age not in ?1
True	findByActiveTrue()	... where x.active = true
False	findByActiveFalse()	... where x.active = false
IgnoreCase	findByFirstnameIgnoreCase	... where UPPER(x.firstname) = UPPER(?1)

# Usando @Query

Isso libera a classe de domínio das informações específicas da persistência e coloca a consulta na interface do repositório.



# Usando @Query

```
public interface UserRepository extends JpaRepository<User, Long> {  
    @Query("select u from User u where u.emailAddress = ?1")  
    User findByEmailAddress(String emailAddress);  
}
```

# Huuuuhullll



o entity tá ok  
o repository tá ok  
o service tá ok  
a main tá ok



.....brota agora os testes pra gente vê!!!!

# Testes Automatizados

"Apenas duas coisas são infinitas: o universo e a estupidez humana. E eu não tenho certeza do primeiro."

Escrevemos uma quantidade razoável de código. Elas funcionam corretamente?

Tudo indica que sim, até criamos um pequeno main para verificar isso e fazer as perguntas corretas.

Pode parecer que o código funciona, mas ele tem muitas falhas. Olhemos com mais cuidado.



# Testes Automatizados

## Testes unitários

- são testes que testam apenas uma classe ou método, verificando se seu comportamento está de acordo com o desejado.
- Em testes de unidade, verificamos a funcionalidade da classe e/ou método em questão passando o mínimo possível por outras classes ou dependências do nosso sistema.

# Testes Automatizados

## Testes de Integração

- Muitas vezes, principalmente quando estamos iniciando no mundo dos testes, é comum criarmos alguns testes que testam muito mais do que o necessário, mais do que apenas a unidade.
- Esses testes são responsáveis por testar o sistemas como um todo.

“Alô, Som! 1, 2, 3! Testando”

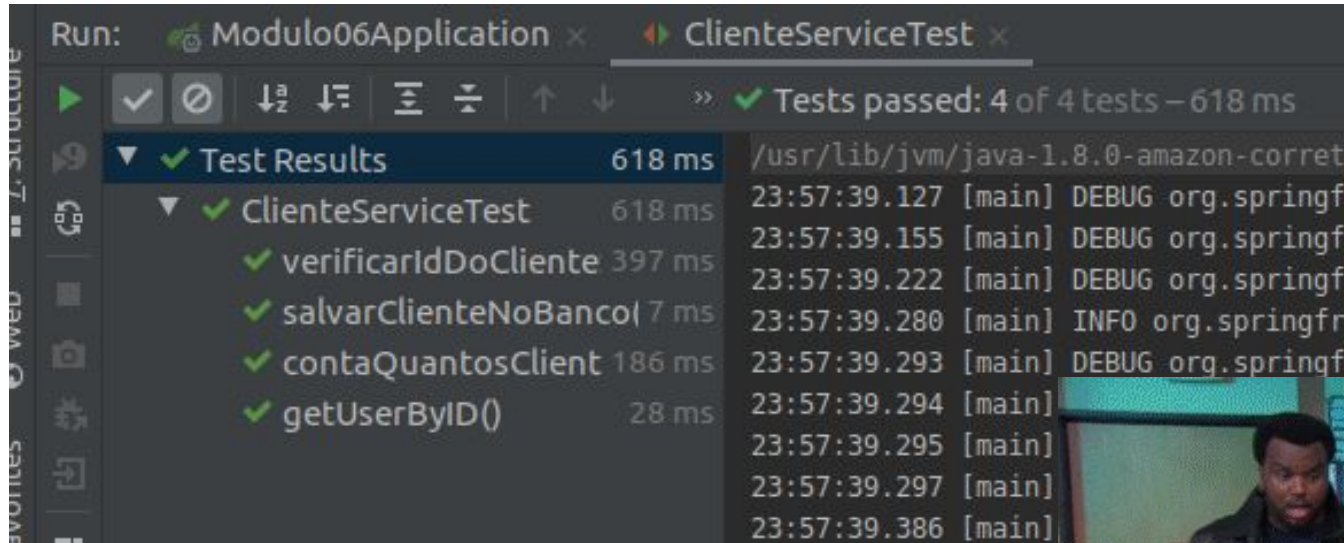
```
12  @SpringBootTest
13  @Transactional
14  public class ClienteServiceTest {
15
16      @Autowired
17      ClienteService clienteServiceTest;
18
19      @Test
20      void salvarClienteNoBanco() {
21          // given
22          Cliente cli = new Cliente();
23          cli.setCpf("12346789015");
24          cli.setNome("Taina");
25
26          // when
27          Cliente clienteSalvo = clienteServiceTest.add(cli);
28
29          // then
30          assertThat(clienteSalvo).isNotNull();
31      }
32  }
```

```
33      @Test
34      void verificarIdDoClienteSalvo() {
35          // given
36          Cliente cli = new Cliente();
37          cli.setCpf("12346789015");
38          cli.setNome("Taina");
39
40          // when
41          Cliente clienteSalvo = clienteServiceTest.add(cli);
42
43          // then
44          assertThat(clienteSalvo.getID()).isEqualTo(1);
45      }
```

```
47      @Test
48      public void getUserByID() {
49          // given
50          Cliente cli = new Cliente();
51
52          cli.setCpf("12346789015");
53          cli.setNome("Taina");
54
55          clienteServiceTest.add(cli);
56
57          // when
58          Cliente found = clienteServiceTest.get(cli.getID());
59
60          // then
61          assertThat(found.getNome())
62              .isEqualTo(cli.getNome());
63      }
```

```
65      @Test
66      public void contaQuantosClientesNoBanco() {
67          // given
68          Cliente cli = new Cliente();
69
70          cli.setCpf("12346789015");
71          cli.setNome("Taina");
72
73          clienteServiceTest.add(cli);
74
75          // when
76          long qtd = clienteServiceTest.quantidaClientes();
77
78          // then
79          assertThat(qtd).isEqualTo(1);
80      }
81  }
```

E no fim...



# GitHub

Link:

<https://github.com/tainajmedeiros/modulo6Codensation>