**Criando um projeto Spring Boot:**

FILE / NEW / OTHER

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

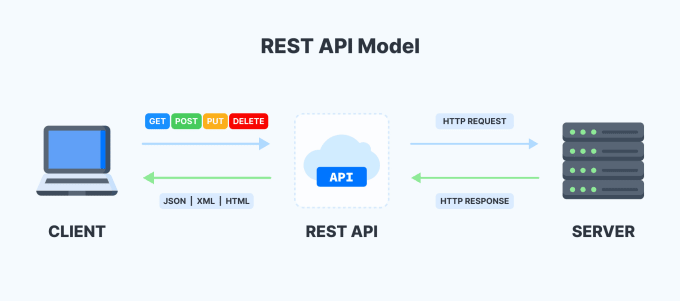
Interface gráfica do usuário, Texto, Aplicativo, Word

Descrição gerada automaticamente

Nesta API, vamos desenvolver um **ENDPOINT para controle de clientes**, que permita cadastrar, consultar, editar e excluir dados de clientes.

Exemplo:

/api/clientes



**Métodos:**

* **POST** /api/clientes (Serviço para cadastro de clientes)
* **PUT** /api/clientes (Serviço para edição de clientes)
* **DELETE** /api/clientes (Serviço para exclusão de clientes)
* **GET** /api/clientes (Serviço para consulta de clientes)

Para isso, vamos começar criando um controlador:

/controllers/ClientesController.java

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

**@RestController**

Annotation para mapear um classe do projeto Spring Boot definindo-a como um controlador de API que utiliza o padrão REST.

**package** br.com.cotiinformatica.controllers;

**import** org.springframework.web.bind.annotation.RestController;

@RestController

**public** **class** ClientesController {

}

**Criando os métodos do controlador:**

Definindo os ENDPOINTS da API.

**package** br.com.cotiinformatica.controllers;

**import** org.springframework.http.ResponseEntity;

**import** org.springframework.web.bind.annotation.DeleteMapping;

**import** org.springframework.web.bind.annotation.GetMapping;

**import** org.springframework.web.bind.annotation.PostMapping;

**import** org.springframework.web.bind.annotation.PutMapping;

**import** org.springframework.web.bind.annotation.RestController;

@RestController

**public** **class** ClientesController {

@PostMapping("/api/clientes")

**public** ResponseEntity<String> post() {

**return** **null**;

}

@PutMapping("/api/clientes")

**public** ResponseEntity<String> put() {

**return** **null**;

}

@DeleteMapping("/api/clientes")

**public** ResponseEntity<String> delete() {

**return** **null**;

}

@GetMapping("/api/clientes")

**public** ResponseEntity<String> getAll() {

**return** **null**;

}

@GetMapping("/api/clientes")

**public** ResponseEntity<String> getById() {

**return** **null**;

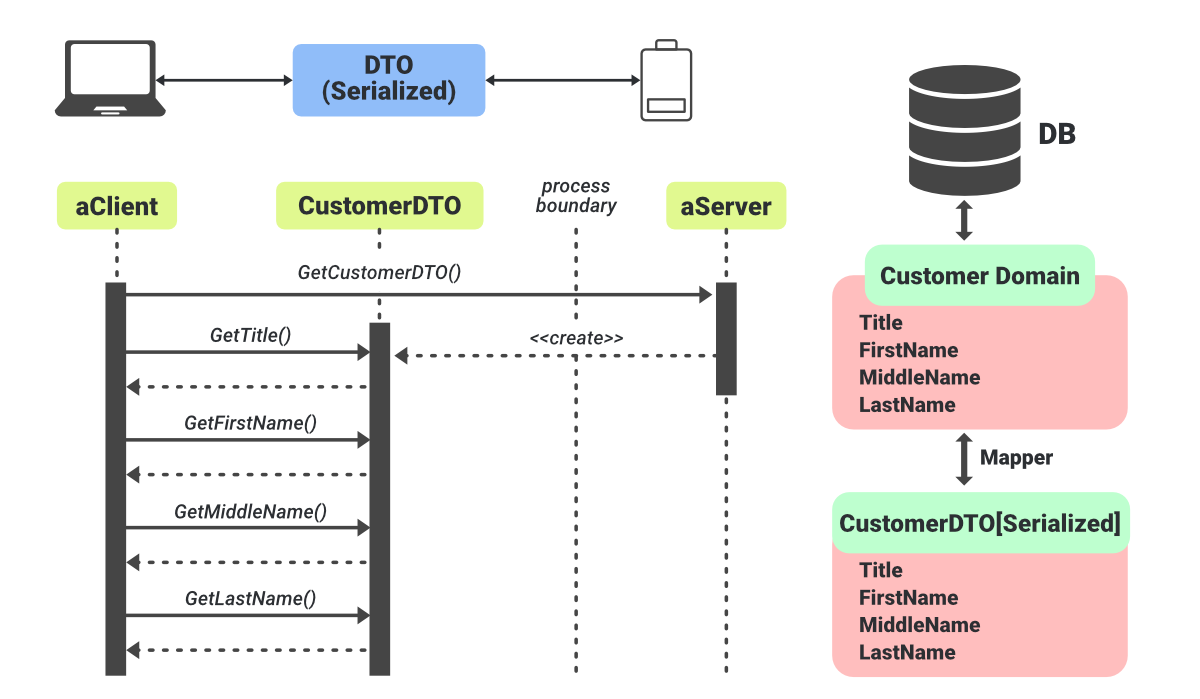
}

}

**DTOs – Data Transfer Objects**

Criando os objetos para transferência de dados com a finalidade

de modelar os dados de requisição e resposta da API.



Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

**package** br.com.cotiinformatica.dtos;

**import** lombok.AllArgsConstructor;

**import** lombok.Getter;

**import** lombok.NoArgsConstructor;

**import** lombok.Setter;

**import** lombok.ToString;

@Setter

@Getter

@NoArgsConstructor

@AllArgsConstructor

@ToString

**public** **class** PostClientesDTO {

**private** String nome;

**private** String cpf;

**private** String email;

**private** String telefone;

}

**package** br.com.cotiinformatica.dtos;

**import** lombok.AllArgsConstructor;

**import** lombok.Getter;

**import** lombok.NoArgsConstructor;

**import** lombok.Setter;

**import** lombok.ToString;

@Setter

@Getter

@NoArgsConstructor

@AllArgsConstructor

@ToString

**public** **class** PutClientesDTO {

**private** Integer idCliente;

**private** String nome;

**private** String cpf;

**private** String email;

**private** String telefone;

}

**package** br.com.cotiinformatica.dtos;

**import** java.util.Date;

**import** lombok.AllArgsConstructor;

**import** lombok.Getter;

**import** lombok.NoArgsConstructor;

**import** lombok.Setter;

**import** lombok.ToString;

@Setter

@Getter

@NoArgsConstructor

@AllArgsConstructor

@ToString

**public** **class** GetClientesDTO {

**private** Integer idCliente;

**private** String nome;

**private** String cpf;

**private** String email;

**private** String telefone;

**private** Date dataCadastro;

}

Voltando no controlador:

/controllers/ClientesController.java

package br.com.cotiinformatica.controllers;

import java.util.List;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.DeleteMapping;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.PathVariable;

import org.springframework.web.bind.annotation.PostMapping;

import org.springframework.web.bind.annotation.PutMapping;

import org.springframework.web.bind.annotation.RequestBody;

import org.springframework.web.bind.annotation.RestController;

import br.com.cotiinformatica.dtos.GetClientesDTO;

import br.com.cotiinformatica.dtos.PostClientesDTO;

import br.com.cotiinformatica.dtos.PutClientesDTO;

@RestController

public class ClientesController {

@PostMapping("/api/clientes")

public ResponseEntity<GetClientesDTO> post

(@RequestBody PostClientesDTO dto) {

return null;

}

@PutMapping("/api/clientes")

public ResponseEntity<GetClientesDTO> put

(@RequestBody PutClientesDTO dto) {

return null;

}

@DeleteMapping("/api/clientes/{id}")

public ResponseEntity<GetClientesDTO> delete

(@PathVariable("id") Integer idCliente) {

return null;

}

@GetMapping("/api/clientes")

public ResponseEntity<List<GetClientesDTO>> getAll() {

return null;

}

@GetMapping("/api/clientes/{id}")

public ResponseEntity<GetClientesDTO> getById

(@PathVariable("id") Integer idCliente) {

return null;

}

}

**Swagger**

Gerando a documentação mostrando os ENDPOINTS e métodos da API.

Incluindo no **pom.xml**

Desenho de rosto de pessoa visto de perto

Descrição gerada automaticamente com confiança média

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

<dependency>

<groupId>**io.springfox**</groupId>

<artifactId>**springfox-swagger-ui**</artifactId>

<version>**3.0.0**</version>

</dependency>

<dependency>

<groupId>**io.springfox**</groupId>

<artifactId>**springfox-boot-starter**</artifactId>

<version>**3.0.0**</version>

</dependency>

Criando a classe de configuração para o Swagger:

/config/SwaggerConfig.java

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

**package** br.com.cotiinformatica.config;

**import** java.util.Collections;

**import** org.springframework.context.annotation.Bean;

**import** org.springframework.context.annotation.Configuration;

**import** org.springframework.web.servlet.config.annotation.EnableWebMvc;

**import** springfox.documentation.builders.PathSelectors;

**import** springfox.documentation.builders.RequestHandlerSelectors;

**import** springfox.documentation.service.ApiInfo;

**import** springfox.documentation.service.Contact;

**import** springfox.documentation.spi.DocumentationType;

**import** springfox.documentation.spring.web.plugins.Docket;

**import** springfox.documentation.swagger2.annotations.EnableSwagger2;

@Configuration

@EnableWebMvc

@EnableSwagger2

**public** **class** SwaggerConfig {

@Bean

**public** Docket api() {

**return** **new** Docket(DocumentationType.***SWAGGER\_2***)

.select()

.apis(RequestHandlerSelectors

.*basePackage*("br.com.cotiinformatica"))

.paths(PathSelectors.*ant*("/\*\*"))

.build()

.apiInfo(apiInfo());

}

**private** ApiInfo apiInfo() {

**return** **new** ApiInfo(

"API para controle de clientes",

"Sistema Spring Boot API",

"Versão 1.0",

"http://www.cotiinformatica.com.br",

**new** Contact("COTI Informática",

"http://www.cotiinformatica.com.br", "contato@cotiinformatica.com.br"),

"Licença da API",

"http://www.cotiinformatica.com.br",

Collections.*emptyList*()

);

}

}

**Voltando no controlador e mapear comentários que descrevam as operações da API:**

package br.com.cotiinformatica.controllers;

import java.util.List;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.DeleteMapping;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.PathVariable;

import org.springframework.web.bind.annotation.PostMapping;

import org.springframework.web.bind.annotation.PutMapping;

import org.springframework.web.bind.annotation.RequestBody;

import org.springframework.web.bind.annotation.RestController;

import br.com.cotiinformatica.dtos.GetClientesDTO;

import br.com.cotiinformatica.dtos.PostClientesDTO;

import br.com.cotiinformatica.dtos.PutClientesDTO;

import io.swagger.annotations.Api;

import io.swagger.annotations.ApiOperation;

**@Api(tags = "Clientes")**

@RestController

public class ClientesController {

**@ApiOperation("Serviço para cadastro de clientes.")**

@PostMapping("/api/clientes")

public ResponseEntity<GetClientesDTO> post

(@RequestBody PostClientesDTO dto) {

return null;

}

**@ApiOperation("Serviço para edição de clientes.")**

@PutMapping("/api/clientes")

public ResponseEntity<GetClientesDTO> put

(@RequestBody PutClientesDTO dto) {

return null;

}

**@ApiOperation("Serviço para exclusão de clientes.")**

@DeleteMapping("/api/clientes/{id}")

public ResponseEntity<GetClientesDTO> delete

(@PathVariable("id") Integer idCliente) {

return null;

}

**@ApiOperation("Serviço para consulta de clientes.")**

@GetMapping("/api/clientes")

public ResponseEntity<List<GetClientesDTO>> getAll() {

return null;

}

**@ApiOperation("Serviço para consulta de cliente por id.")**

@GetMapping("/api/clientes/{id}")

public ResponseEntity<GetClientesDTO> getById

(@PathVariable("id") Integer idCliente) {

return null;

}

}

Por último, vamos configurar a porta do TOMCAT para executar o projeto:

/application.properties

**server.port=8082**

Interface gráfica do usuário, Aplicativo, Word

Descrição gerada automaticamente

Executando:

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

**Executando:**

Interface gráfica do usuário, Tabela

Descrição gerada automaticamente

**Acessando**

<http://localhost:8082/swagger-ui/index.html>

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

**Criando o banco de dados:**

PostGreSQL:

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Criando a classe de entidade

Modelo do banco de dados.

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

**package** br.com.cotiinformatica.entities;

**import** java.util.Date;

**import** lombok.AllArgsConstructor;

**import** lombok.Getter;

**import** lombok.NoArgsConstructor;

**import** lombok.Setter;

**import** lombok.ToString;

@Setter

@Getter

@NoArgsConstructor

@AllArgsConstructor

@ToString

**public** **class** Cliente {

**private** Integer idCliente;

**private** String nome;

**private** String cpf;

**private** String email;

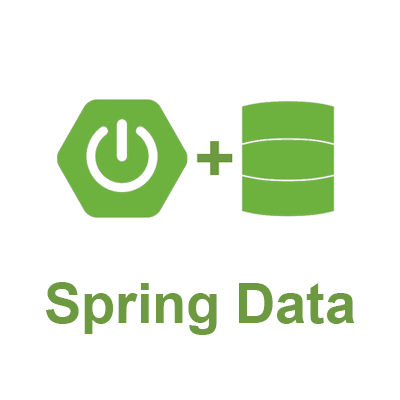
**private** String telefone;

**private** Date dataCadastro;

}

**Spring Data**

Conjunto de tecnologias do framework Spring para implementarmos repositórios de acesso a banco de dados nos projetos.



É composto de bibliotecas como:

* JDBC Template
* Hibernate e JPA (Java Persistence API)
* Drivers para bancos de dados não relacionais (NoSQL)

**Hibernate & JPA (Java Persistence API)**

São bibliotecas do Spring Data voltadas para conexão com bancos de dados utilizando um padrão chamado **ORM – Mapeamento Objeto Relacional**.



Primeiro, precisamos configurar as bibliotecas para o uso do Spring Data com Hibernate e JPA:

**/pom.xml**

Interface gráfica do usuário, Texto, Aplicativo, chat ou mensagem de texto

Descrição gerada automaticamente

<dependency>

<groupId>**org.springframework.boot**</groupId>

<artifactId>**spring-boot-starter-data-jpa**</artifactId>

</dependency>

<dependency>

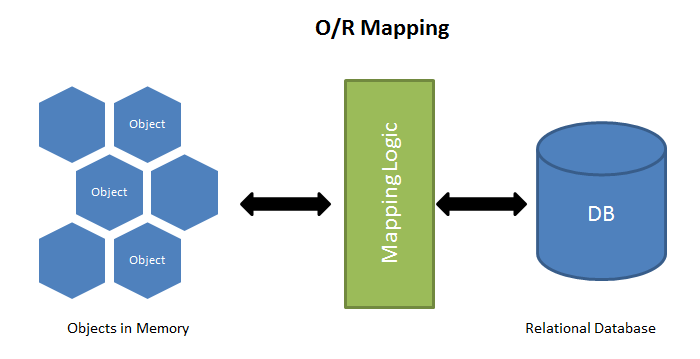
<groupId>**org.springframework.boot**</groupId>

<artifactId>**spring-boot-autoconfigure**</artifactId>

</dependency>

**Primeiro passo:**

Precisamos mapear as entidades do projeto, configurando quais tabelas do banco de dados serão associadas a cada classe de entidade do sistema.



Para fazermos o mapeamento, vamos utilizar a seguinte biblioteca:

javax.persistence (JPA)

**@Entity**

Define a classe como uma entidade do banco de dados. Toda classe mapeada como uma tabela do banco de dados deve obrigatoriamente ter essa anotação.

**@Table**

Define o nome da tabela do banco de dados

**@Id**

Define o campo chave primária da tabela, toda entidade mapeada deve obrigatoriamente ter uma chave primária.

**@GeneratedValue**

Define que um campo inteiro terá o seu conteúdo gerado automaticamente pelo banco de dados.

**@Column**

Mapeia as características de uma coluna da tabela.

**@Temporal**

Mapeia campos do tipo Data e Hora.

**package** br.com.cotiinformatica.entities;

**import** java.util.Date;

**import** javax.persistence.Column;

**import** javax.persistence.Entity;

**import** javax.persistence.GeneratedValue;

**import** javax.persistence.GenerationType;

**import** javax.persistence.Id;

**import** javax.persistence.Table;

**import** javax.persistence.Temporal;

**import** javax.persistence.TemporalType;

**import** lombok.AllArgsConstructor;

**import** lombok.Getter;

**import** lombok.NoArgsConstructor;

**import** lombok.Setter;

**import** lombok.ToString;

**@Entity**

**@Table(name = "cliente")**

@Setter

@Getter

@NoArgsConstructor

@AllArgsConstructor

@ToString

**public** **class** Cliente {

**@Id**

**@GeneratedValue(strategy = GenerationType.*IDENTITY*)**

**@Column(name = "idcliente")**

**private** Integer idCliente;

**@Column(name = "nome", length = 150, nullable = false)**

**private** String nome;

**@Column(name = "cpf", length = 11, nullable = false,**

**unique = true)**

**private** String cpf;

**@Column(name = "email", length = 50, nullable = false,**

**unique = true)**

**private** String email;

**@Column(name = "telefone", length = 14, nullable = false)**

**private** String telefone;

**@Temporal(TemporalType.*DATE*)**

**@Column(name = "datacadastro", nullable = false)**

**private** Date dataCadastro;

}

/application.properties

Mapeamento da conexão com o banco de dados (**DATA SOURCE**):

**server.port=8082**

**spring.datasource.url=jdbc:postgresql://localhost:5432/bd\_apiclientes**

**spring.datasource.driver-class-name=org.postgresql.Driver**

**spring.datasource.username=postgres**

**spring.datasource.password=coti**

**spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect**

**spring.jpa.hibernate.ddl-auto=update**

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

**spring.jpa.properties.hibernate.format\_sql=true**

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

**Executando o projeto:**

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Interface gráfica do usuário, Tabela, Excel

Descrição gerada automaticamente

Interface gráfica do usuário, Texto, Aplicativo, chat ou mensagem de texto

Descrição gerada automaticamente

Hibernate:

create table cliente (

idcliente serial not null,

cpf varchar(11) not null,

datacadastro date not null,

email varchar(50) not null,

nome varchar(150) not null,

telefone varchar(14) not null,

primary key (idcliente)

)

Hibernate:

alter table cliente

drop constraint UK\_r1u8010d60num5vc8fp0q1j2a

Hibernate:

alter table cliente

add constraint UK\_r1u8010d60num5vc8fp0q1j2a unique (cpf)

Hibernate:

alter table cliente

drop constraint UK\_cmxo70m08n43599l3h0h07cc6

Hibernate:

alter table cliente

add constraint UK\_cmxo70m08n43599l3h0h07cc6 unique (email)

**Criando os repositórios do projeto:**

Desenvolvendo os repositórios através de interfaces.

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

**@Repository**

Mapeando a interface como um repositório do Spring Data, que permitirá o uso dos métodos do Hibernate e JPA.

**package** br.com.cotiinformatica.repositories;

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

**import** org.springframework.stereotype.Repository;

**import** br.com.cotiinformatica.entities.Cliente;

**@Repository**

**public interface IClienteRepository**

**extends JpaRepository<Cliente, Integer> {**

}

**Publicando no GITHUB:**

Samsung@DESKTOP-P9F6D9F MINGW64 ~/Desktop/COTI - Aulas/2023 - Java WebDeveloper SQS 18h as 22h (Início em 09.01)/workspace/api\_clientes

**$ git init**

Samsung@DESKTOP-P9F6D9F MINGW64 ~/Desktop/COTI - Aulas/2023 - Java WebDeveloper SQS 18h as 22h (Início em 09.01)/workspace/api\_clientes (master)

**$ git branch -m main**

Samsung@DESKTOP-P9F6D9F MINGW64 ~/Desktop/COTI - Aulas/2023 - Java WebDeveloper SQS 18h as 22h (Início em 09.01)/workspace/api\_clientes (main)

**$ git add .**

Samsung@DESKTOP-P9F6D9F MINGW64 ~/Desktop/COTI - Aulas/2023 - Java WebDeveloper SQS 18h as 22h (Início em 09.01)/workspace/api\_clientes (main)

**$ git commit -m 'first commit'**

Samsung@DESKTOP-P9F6D9F MINGW64 ~/Desktop/COTI - Aulas/2023 - Java WebDeveloper SQS 18h as 22h (Início em 09.01)/workspace/api\_clientes (main)

**$ git remote add origin https://github.com/smendescoti/api\_clientes.git**

Samsung@DESKTOP-P9F6D9F MINGW64 ~/Desktop/COTI - Aulas/2023 - Java WebDeveloper SQS 18h as 22h (Início em 09.01)/workspace/api\_clientes (main)

**$ git push -u origin main**

Continua…