

Olá, meu nome é Carla Cristina e vou contar um pouquinho da minha aplicação. Vamos lá?

Quais as tecnologias do mundo Spring você usaria?

- Spring web
- Spring MVC
- Spring Data JPA
- Dev Tools
- Injeção de dependências
- Anotações de estereótipos

Quais classes seriam criadas nesse processo?

- Controller - que receberá o CRUD completo da aplicação, ou seja, as requisições do usuário (Get, Post, Put, Delete)
- Model - Onde será anotado os atributos da entidade, ou seja, as colunas da tabela e os seu tipo primitivo.
- Repository - É a interface que receberá o JPA Repository e nos auxiliará na busca dentro do banco de dados, nela podemos até montar um Query.

Qual foi o seu processo de decisão para realizar a implementação e o papel de cada tecnologia utilizada?

1. Escolher quais dependências fariam parte deste projeto:

```

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

  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
    <scope>runtime</scope>
    <optional>true</optional>
  </dependency>
  <dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <scope>runtime</scope>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
  </dependency>
</dependencies>

```

2. Configurar o application.properties, onde especificará a porta que utilizaremos a aplicação, o nome do banco de dados que será criado, caso já não exista um.

```

spring.jpa.hibernate.ddl-auto=update
spring.datasource.url=jdbc:mysql://localhost/bancoZup?createDatabaseIfNotExist=true&serverTimezone=UTC&useSSL=false
spring.datasource.username=root
spring.datasource.password=
spring.jpa.show-sql=true

```

3. Criar a model segundo a necessidade do projeto

```

@Entity
@Table(name="tb_usuario")
public class UsuarioModel {

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

    @Column
    @NotNull
    private String nome;

    @Column(unique = true)
    @NotNull
    private String email;

    @Column(unique = true)
    @NotNull
    private String cpf;

    @Column
    @NotNull
    private String dataNascimento;
}

```

As anotações @Column, @Id, @GeneratedValue são do javax.persistence e ajudarão a persistir as informações corretas no banco de dados. O unique = true fará com que nem o email e nem o cpf sejam duplicados.

#### 4 . Criando o repository

```

package com.projeto.zupProject;

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

public interface UsuarioRepository extends JpaRepository<UsuarioModel , Long>{

}

```

## 5. Controller “/cadastro”

```
@RestController
@RequestMapping("/cadastro")
@CrossOrigin(origins = "*" , allowedHeaders = "*")
public class UsuarioController {

    @Autowired
    private UsuarioRepository repository;

    @GetMapping
    public ResponseEntity<List<UsuarioModel>> buscarTodos(){
        return ResponseEntity.ok(repository.findAll());
    }

    // @GetMapping("/{id}")
    // public ResponseEntity<UsuarioModel> GetById(@PathVariable long id) {
    //     return repository.findById(id).map(resp -> ResponseEntity.ok(resp)).orElse(ResponseEntity.notFound().build());
    // }

    @PostMapping("/cadastrando")
    public ResponseEntity<UsuarioModel> Post(@RequestBody UsuarioModel usuarioModel){
        return ResponseEntity.ok(repository.save(usuarioModel));
    }

    // @PutMapping
    // public ResponseEntity<UsuarioModel> Put(@RequestBody UsuarioModel usuario) {
    //     return ResponseEntity.ok(repository.save(usuario));
    // }
```

- @RestController é uma anotação do spring MVC, que substituiu a anotação @Controller. Desta forma os dados serão retornados como JSON ou XML. Esta é a melhor forma de trafegar informações, usando uma API REST. O controller então irá receber as requisições, chamar a model e encontrar a view com as informações necessárias.
- @RequestMapping("/cadastro") define que qualquer requisição receberá este endpoint
- @GetMapping é o Read do CRUD e trará uma resposta na requisição, "OK" e @PostMapping é o Save, este enviará os valores da tabela.

Montando a aplicação cliente com Angular.

■ e2e	projeto angular criado, com componentes, model e service	4 days ago
■ src	criacao de componentes, service e modal	2 days ago
📄 .browserslistrc	projeto angular criado, com componentes, model e service	4 days ago
📄 .editorconfig	projeto angular criado, com componentes, model e service	4 days ago
📄 .gitignore	projeto angular criado, com componentes, model e service	4 days ago
📄 README.md	projeto angular criado, com componentes, model e service	4 days ago
📄 angular.json	projeto angular criado, com componentes, model e service	4 days ago
📄 karma.conf.js	projeto angular criado, com componentes, model e service	4 days ago
📄 package-lock.json	projeto angular criado, com componentes, model e service	4 days ago
📄 package.json	projeto angular criado, com componentes, model e service	4 days ago
📄 tsconfig.app.json	projeto angular criado, com componentes, model e service	4 days ago
📄 tsconfig.json	projeto angular criado, com componentes, model e service	4 days ago
📄 tsconfig.spec.json	projeto angular criado, com componentes, model e service	4 days ago
📄 tslint.json	projeto angular criado, com componentes, model e service	4 days ago

1. Utilizaremos a npm do node.js para instalar os seguintes pacotes:

```
npm install bootstrap@latest --save
```

```
npm install jquery@latest --save
```

```
npm install popper.js --save
```

Utilizarei bootstrap como framework e para a aplicação funcionar é necessário o jquery e o popper.js

2. Criando o model dentro do Angular. Esta classe será responsável em receber exatamente os atributos da nossa entidade @Entity

```

1  export class Usuario{
2
3      public nome: string
4      public email: string
5      public cpf: string
6      public dataNascimento: string
7
8  }
```

3. Criando o Service

Comando: `ng g s service/nomeService`

```

import { HttpClient, HttpHeaders } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { Observable } from 'rxjs';
import { Usuario } from '../model/usuario';

@Injectable({
  providedIn: 'root'
})
export class UsuarioService {

  usuario: Usuario = new Usuario ()
  constructor(private http: HttpClient) { }

  cadastrar(usuario: Usuario) : Observable<Usuario>{
    return this.http.post<Usuario>("http://localhost:8080/cadastro/cadastrando", usuario)
  }

  buscarTodos() : Observable<Usuario[]>{
    return this.http.get<Usuario[]>("http://localhost:8080/cadastro")
  }

  getIdUsuario(id:number) : Observable<Usuario>{
    return this.http.get<Usuario>(`http://localhost:8080/cadastro/${id}`)
  }
}

```

O service é onde a mágica acontece, para falar a verdade é a parte que eu mais gosto. É o lugar onde o back e o front se conectam, criando uma aplicação para o cliente completa.

Essas funções correspondem ao `@RestController` do nosso Spring. Cadastrar é o Post, buscar o get e o GetById se eu quiser encontrar algum usuário em específico. Passamos então a URL completa para encontrar exatamente o end point no Controller que desejamos.

#### 4. Criando os componentes para a utilização do conceito One Single Page Application

📁 cadastrar	criacao de componentes, service e modal	2 days ago
📁 login	criacao de componentes, service e modal	2 days ago
📁 model	criacao de componentes, service e modal	2 days ago
📁 navbar	criacao de componentes, service e modal	2 days ago
📁 pagina-inicial	criacao de componentes, service e modal	2 days ago
📁 service	criacao de componentes, service e modal	2 days ago
📄 app-routing.module.ts	projeto angular criado, com componentes, model e service	4 days ago
📄 app.component.css	projeto angular criado, com componentes, model e service	4 days ago
📄 app.component.html	projeto angular criado, com componentes, model e service	4 days ago
📄 app.component.spec.ts	projeto angular criado, com componentes, model e service	4 days ago
📄 app.component.ts	projeto angular criado, com componentes, model e service	4 days ago
📄 app.module.ts	criacao de componentes, service e modal	2 days ago

O Comando para criação de componentes é `ng g c nomeComponent`, nele é criado tanto o TypeScript, o CSS e o HTML do componente. Focarei no componente de cadastro e cadastros.

- Cadastro

```

usuario: Usuario = new Usuario()
senha: string

env = environment;

cpf = env.cpf
constructor(public usuarioService: UsuarioService, private router: Router) { }

ngOnInit() {
}

cadastrar() {
  if (this.usuario.nome == null || this.usuario.email == null || this.usuario.cpf == null || this.usuario.dataNascimento == null) {
    window.alert('Todos os campos são obrigatórios')
  } else {
    this.usuarioService.cadastrar(this.usuario).subscribe((resp: Usuario) => {
      this.usuario = resp
      window.alert("Usuario cadastrado com sucesso")
      this.router.navigate(['/login'])
    })
  }
}
}

```

A função cadastro checa se os campos estão nulos e mostra um alert caso estiverem. Caso contrário chamamos a função cadastrar do usuarioService, instanciamos o objeto e o subscribe recebe a informação como Json.

Futura implementação: Ainda preciso estudar melhor como usar o environment dentro da função, para que o cpf e o email não sejam duplicados dentro mesmo do front.

- Cadastros



```

export class LogarComponent implements OnInit {

  usuario: Usuario = new Usuario()
  listaUsuarios: Usuario[]

  idUsuario: number

  constructor(public usuarioService: UsuarioService,
    private router: Router) { }

  ngOnInit() {
    this.encontreTodos()
  }

  encontrarTodos(){
    this.usuarioService.buscarTodos().subscribe((resp: Usuario [])=>{
      this.listaUsuarios = resp
    })
  }

  encontrarId(){
    this.usuarioService.getByIdUsuario(this.idUsuario).subscribe((resp: Usuario)=>{
      this.usuario = resp
    })
  }
}

```

Neste componente, receberemos todos os cadastros. Então chamamos as funções `encontreTodos` e `encontreId` e colocamos dentro do `ngOnInit`. Ou seja, assim que abrirmos esse component receberemos todos os cadastros em uma lista. Bora ver como ficou?



# Orange TALENTS

localhost:4200 diz  
Usuário cadastrado com sucesso  
OK

## Cadastrar

Nome

E-mail

CPF

Data de Nascimento

Cadastrar

## Cadastros

#	Nome	CPF	Email	Data Nascimento
1	Carla Cristina	018.315.360-05	carla@carla.com	19/01/1996
2	Víctor Saade	269.938.150-01	victor@victor.com	18/06/1996

### Observações

Existem algumas aplicações que ainda não consegui fazer, mas independente do resultado continuo estudando e melhorando esta aplicação com outras camadas de controller.

Obrigada por esta oportunidade, foi uma grande oportunidade de provar os meus conhecimentos!

O repositório desta aplicação está no gitHub:

<https://github.com/CarlaCCP/zupProject>