|  |
| --- |
| **Projeto CRUD com Angular** |
| **Objetivo Geral: Tutorial de Criação de Projetos CRUD com Angular** |
| *Conteúdo:* |
| 1. Instalação das Ferramentas necessárias 2. Criação de Componentes Angular 3. Anexando o CRUD ao projeto angular |
| *Metodologia e Estratégia:* |
| 1. Aula expositiva dialogada com apoio de tutorial; 2. Exercícios de aplicação. |

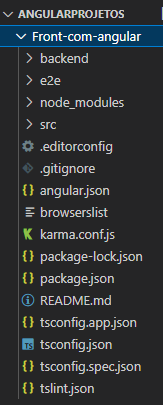
Criação de um projeto CRUD com o Angular

CRUD (Create, Read, Update e Delete em Inglês) é uma sigla utilizada para se referir às quatro operações básicas realizadas em banco de dados relacionais que são consulta, inclusão, alteração e exclusão dos registros.

Para criação desse projeto devemos foi desenvolvido um front para uma pagina de registro de eventos. Vamos realizar o crud no cadastro de pessoas no site. Para continuar deve-se baixar o projeto angular disponível em

<https://drive.google.com/file/d/1TgicDW0nROGXvkBIrsV2m9Iw_v8m2gpO/view?usp=sharing>

e extrair o arquivo .rar na WorkSpace dos projetos angular



Teste o Backend, Crie uma alumas informações no arquivo db.json e abra o postman

Lembre de iniciar o json-server

Digitando no terminal dentro da pasta backend:

npm i json-server

npm start

Com o json-server funcionando vamos começar a fazer o método CREATE

Cadastro create

Aprenderemos, agora, a criar uma requisição para um formulário do tipo login. No arquivo Front com angular.zip, em src>app>components, você encontrará as pastas separadas por temas, de acordo com a arquitetura do Angular.

1. Para acessar a tela de cadastro, utilize o caminho: src>app>components>cadastro>cadastro-create
2. Clique com o botão direito do mouse sobre a pasta cadastro , depois, em new file ” e crie um arquivo chamado cadastro.model.ts . Ao criarmos uma interação com o Back End, essa nova interação irá procurar um arquivo que possua seus atributos definidos. No arquivo db.json , é possível apenas adicionar dados novos, além de atualizar, listar e remover dados que já estão adicionados. Após criarmos o arquivo, devemos passar os atributos, que serão os mesmo que já estão no arquivo db.json . O id é opcional, pois o usuário não o fornece; é o próprio Back End, ao salvar os dados, que gera esse id.
3. export interface Cadastro{
4. id?: number
5. usuario: string
6. senha: string
7. }
8. Abra o projeto no terminal e crie o *service*digitando no terminal o comando **ng g s components/cadastro/cadastro**; para finalizar, aperte “*enter*”. O sistema criará um arquivo *service*, que é usado para implementar regras, acessar o Back-End e outros arquivos HTTP, entre outras funções.



1. Acesse o arquivo **cadastro-create.component.ts** e faça o seguinte procedimento:

•importe os arquivos *model*e *servisse* que você criou (linhas 2 e 4).

•instancie os dados da *model*(linha 13).

•No *constructor*, insira o cadastroService(linha 18) e o r*outer* (importado na linha 3).

Agora, podemos migrar o conteúdo dos demais arquivos para o arquivo typeScript(.ts).

import { Component, OnInit } from '@angular/core';

import { CadastroService } from '../cadastro.service';

import { Router } from '@angular/router';

import { Cadastro } from '../cadastro.model';

@Component({

  selector: 'app-cadastro-create',

  templateUrl: './cadastro-create.component.html',

  styleUrls: ['./cadastro-create.component.css']

})

export class CadastroCreateComponent implements OnInit {

  cadastro: Cadastro = {

    usuario: '',

    senha: ''

  }

   constructor(private cadastroService: CadastroService, private router: Router) { }

  ngOnInit(): void {

  }

}

1. No arquivo **cadastro-create.component.html**, adicione as seguintes linhas:
2. <section class="new-user" id="new-user">
3. <div class="container">
4. <div class="title-new-user">
5. <div class="row">
6. <div class="col-12">
7. <h2>Cadastre-se</h2>
8. </div>
9. <div class="col-md-6" style="margin: 0 auto;">
10. <div class="row-effect">
11. <p>Preencha o formulário</p>
12. </div>
13. </div>
14. </div>
15. </div>
16. <div class="row">
17. <div class="col-md-6" style="margin: 0 auto;">
18. <form>
19. <div class="form-row">
20. <div class="form-group col-md-12">
21. <mat-form-field>
22. <label>Usuário</label>
23. <input type="text" class="form-control" matInput[(ngModel)]="cadastro.usuario" name="usuario">
24. </mat-form-field>
25. </div>
26. </div>
27. <div class="form-group">
28. <mat-form-field>
29. <label>Senha</label>
30. <input type="password" class="form-control" matInput[(ngModel)]="cadastro.senha" name="senha">
31. </mat-form-field>
32. </div>
33. <div class="row">
34. <div class="col-12 col-md-12 d-flex justify-content-center" style="margin: 0 auto;">
35. <hr class="mb-4">
36. <div class="col-12 col-md-4 p-0">
37. <button class="btn btn-cadastrar" (click)="createCadastro()" color="warn">Cadastrar</button>
38. </div>
39. <div class="col-12 col-md-3 p-0">
40. <button class="btn btn-cadastrar" (click)="cancelarCadastro()" color="accent">Cancelar</button>
41. </div>
42. <div class="col-12 col-md-5 p-0">
43. <button class="btn btn-cadastrar" (click)="tabelasCadastro()" color="accent">Todos os cadastros</button>
44. </div>
45. </div>
46. </div>
47. </form>
48. </div>
49. </div>
50. </div>
51. </section>
53. <app-footer></app-footer>

Em [(ngModel)] (primeiro destaque, em azul), estamos passando os dados desse formulário para os atributos criados na *model*e instanciados no arquivo cadastro-create.component.ts.

Já a função (click) realizará algum evento por meio da interação do usuário, ou seja, o clique (segundo destaque, em azul). Se você salvar desse modo, o VSCode retornará erros no terminal, informando que os eventos do botão não foram criados.

6. Desse modo, antes de salvar, acesse o arquivo cadastro-create.component.tse adicione as seguintes linhas:

ngOnInit(): void { //já existe

  } //já existe acrescentar as funções abaixo

  createCadastro(): void{

    this.cadastroService.create(this.cadastro).subscribe(() =>{

      this.cadastroService.showMessege('Usuário Cadastrado')

    })

  }

  cancelarCadastro(): void{

    this.router.navigate([''])

  }

  tabelasCadastro(): void{

    this.router.navigate(['cadastro/tabela'])

  }

estamos criando o cadastro com a palavra “*create*”.

**Observação:** O *router* é um arquivo de configuração Front-End que serve para fazer a rota de acesso às páginas internas. Para ver as rotas, acesse o arquivo Front-com-angular\src\appapp-routing.module-ts.

1. Volte a ao arquivo cadastro.*service.ts* e adicione as seguintes linhas:

import { HttpClient } from '@angular/common/http';

import { Injectable } from '@angular/core';

import { MatSnackBar } from '@angular/material/snack-bar';

import { Observable } from 'rxjs';

import { Cadastro } from './cadastro.model';

@Injectable({

  providedIn: 'root'

})

export class CadastroService {

  baseUrl = "http://localhost:3001/login"

  constructor(private snackBar: MatSnackBar, private http: HttpClient) {

   }

   showMessege(msg:string):void{

    this.snackBar.open(msg, 'X',{

      duration: 6000,

      verticalPosition: "bottom"

    });

   }

   create(cadastro: Cadastro): Observable<Cadastro>{

    return this.http.post<Cadastro>(this.baseUrl, cadastro)

   }

}

Com esse código, estamos passando a URL e criando o método de “*create”*, a partir do qual o sistema adicionará as informações passadas pelo usuário, por meio de um *input* via POST, ao banco de dados.

Na linha 17, estamos criando um modal para apresentar uma mensagem quando a ação de deletar, atualizar ou criar um recurso for realizada (mensagem esta que ainda deverá ser configurada).

**Consulta de dados –READ**

Agora, estudaremos um exemplo de como criar uma requisição para consulta de dados (READ).

**1.**No arquivo **cadastro-read.component.html**, localizado dentro da pasta Front-com-angular\src\app\components\cadastro\cadastro-read, iremos inserir o atributo **\*matCellDef=”letrow”:{{row.usuario}}**.

<ng-container matColumnDef="id">

                <th mat-header-cell \*matHeaderCellDef mat-sort-header>Id</th>

                <td mat-cell \*matCellDef="let row">{{row.id}}</td>

              </ng-container>

              <ng-container matColumnDef="usuario">

                <th mat-header-cell \*matHeaderCellDef mat-sort-header>Usuário</th>

                <td mat-cell \*matCellDef="let row">{{row.usuario}}</td>

              </ng-container>

              <ng-container matColumnDef="senha">

                <th mat-header-cell \*matHeaderCellDef mat-sort-header>Senha</th>

                <td mat-cell \*matCellDef="let row">{{row.senha}}</td>

              </ng-container>

No Angular, um valor entre duas chaves representa uma variável, ou seja, um valor do typeScriptapresentado no HTML. O atributo **matCellDef** é utilizado para capturar o valor da tabela (na linha 8, é o nome das colunas e, na linha 9, o valor).

Na linha 22, há uma ação para os botões, ou seja, quando o usuário clicar em um botão para editar ou excluir dados no Front-End, uma outra ação será gerada no Back-End.

Perceba que, nas linhas 25 e 28, passamos o id específico que queremos editar. Na linha 25, ele levará à página em que poderemos editar e, também, deletar o id.

Testando pelo Postman, ao atualizar (PUT) ou deletar (DELETE) um conjunto de dados específicos, você precisará da URL e do id, por exemplo, http://localhost:3001/login/3. No Front-End, por outro lado, o usuário não vê esse processo.

   <ng-container matColumnDef="action">

                <th mat-header-cell \*matHeaderCellDef mat-sort-header>Ações</th>

                <td mat-cell \*matCellDef="let row">

                  <a routerLink="/cadastro/update/{{row.id}}">

                    <i class="material-icons">edit</i>

                  </a>

                  <a routerLink="/cadastro/delete/{{row.id}}">

                    <i class="material-icons">delete</i>

                  </a>

                </td>

              </ng-container>

              <tr mat-header-row \*matHeaderRowDef="displayedColumns"></tr>

              <tr mat-row \*matRowDef="let row; columns: displayedColumns;"></tr>

            </table>

**2.** No arquivo **cadastro-read.component.ts**, adicione o seguinte código, conforme destacado na imagem:

import { Component, OnInit } from '@angular/core';

import { Cadastro } from '../cadastro.model';

import { CadastroService } from '../cadastro.service';

@Component({

  selector: 'app-cadastro-read',

  templateUrl: './cadastro-read.component.html',

  styleUrls: ['./cadastro-read.component.css']

})

export class CadastroReadComponent implements OnInit {

  cadastros: Cadastro[]

  displayedColumns = ['id','usuario','senha','action'];

  constructor(private cadastroService: CadastroService) {}

  ngOnInit(): void {

    this.cadastroService.read().subscribe(cadastros => {

      this.cadastros = cadastros

    })

  }

}

No Angular, as colunas são passadas pelo arquivo **.ts**, diferentemente do que ocorre no HTML.

Cada valor (“Id”, “Usuário”, “Senha”, “Ação”) representa uma coluna.

**3.**Em seu arquivo **cadastro.service.ts**, adicione o código a seguir:

Perceba que a diferença entre adicionar (CREATE) e mostrar todos (READ) reside nos métodos utilizados (POST e GET).

**4.**Caso deseje listar apenas um único usuário, faça conforme mostrado na imagem a seguir e salve seu arquivo.

Os métodos listado acima podem ser compreendidos pelo seguintes códigos

   read(): Observable<Cadastro[]>{

    return this.http.get<Cadastro[]>(this.baseUrl)

   }

   readById(id: number): Observable<Cadastro>{

    const url =`${this.baseUrl}/${id}`

    return this.http.get<Cadastro>(url)

   }

Dentro do arquivo cadastro.service.ts

**Cadastro *update***

**1.**Acesse a pasta **cadastro-update**. Existe a opção de editar na tabela em que estão listados todos os dados. O processo de atualização consiste em alterar um dado que já está salvo, sobrescrevendo-o.

**2.**Acesse o arquivo **cadastro-update.component.html**e observe a sintaxe do código. Na imagem abaixo, analise as linhas 22 e 29, nas quais estamos passando o **[(ngModel)]**para substituir o valor dos atributos.

<form>

                    <div class="form-row">

                        <div class="form-group col-md-12">

                            <mat-form-field>

                                <label>Usuario</label>

                                <input type="text" class="form-control" matInput [(ngModel)]="cadastro.usuario" name="usuario">

                            </mat-form-field>

                        </div>

                    </div>

                    <div class="form-group">

                        <mat-form-field>

                            <label>Senha</label>

                            <input type="password" class="form-control" matInput [(ngModel)]="cadastro.senha" name="cpf">

                        </mat-form-field>

                    </div>

**3.**No arquivo **cadastro.service.ts** foram adicionadas as linhas de 37 a 40.

A linha 37 funciona de modo semelhante ao método de exclusão, pois ambos afetam apenas um único usuário ,uma vez que as alterações estão sendo feitas no id (lembre-se de que o id é único).

updateCadastro(cadastro: Cadastro): Observable<Cadastro>{

    const url = `${this.baseUrl}/${cadastro.id}`

    return this.http.put<Cadastro>(url, cadastro)

   }

Para p Modulo Delete

1. Acesse o arquivo **cadastro-delete.component.html**. Realize os seguintes ajustes
2. <div class="row">
3. <div class="col-md-6" style="margin: 0 auto;">
4. <form>
5. <div class="form-row">
6. <div class="form-group col-md-12">
7. <mat-form-field>
8. <label>Usuário</label>
9. <input type="text" class="form-control" matInput [value]="cadastro.usuario" name="usuario" disabled>
10. </mat-form-field>
11. </div>
12. </div>
13. <div class="form-group">
14. <mat-form-field>
15. <label>Senha</label>
16. <input type="password" class="form-control" matInput [value]="cadastro.senha" name="senha" disabled>
17. </mat-form-field>
18. </div>
19. <div class="row">
20. <div class="col-12 col-md-12 d-flex justify-content-center" style="margin: 0 auto;">
21. <hr class="mb-4">
22. <div class="col-12 col-md-4 p-0">
23. <button class="btn btn-cadastrar" (click)="deleteCadastro()" color="warn">Deletar</button>
24. </div>
25. <div class="col-12 col-md-3 p-0">
26. <button class="btn btn-cadastrar" (click)="cancelarCadastro()" color="accent">Cancelar</button>
27. </div>
28. <div class="col-12 col-md-5 p-0">
29. <button class="btn btn-cadastrar" (click)="tabelasCadastro()" color="accent">Todos os cadastros</button>
30. </div>

Perceba que os inputs estão desabilitados nessa página, o que

significa que o usuário não pode inserir dados Conforme

podemos verificar na linha 36 está previsto um clique para o

botão deletar o que confirma a exclusão delete O método de

funcionamento desse botão está sendo configurado no

cadastro.service.ts.

deleteCadastro(id: number): Observable<Cadastro>{

    const url = `${this.baseUrl}/${id}`

    return this.http.delete<Cadastro>(url)

   }

Quando vamos excluir um usuário, devemos passar o id

Logo após configurar o service adicione o código ao seu

arquivo cadastro-delete.component.ts

import { Component, OnInit } from '@angular/core';

import { Cadastro } from '../cadastro.model';

import { CadastroService } from '../cadastro.service';

import { Router, ActivatedRoute } from '@angular/router';

@Component({

  selector: 'app-cadastro-delete',

  templateUrl: './cadastro-delete.component.html',

  styleUrls: ['./cadastro-delete.component.css']

})

export class CadastroDeleteComponent implements OnInit {

  cadastro: Cadastro;

  constructor(private cadastroService: CadastroService,

              private router: Router,

              private route: ActivatedRoute

  ) {}

  ngOnInit(): void {

    const id = +this.route.snapshot.paramMap.get('id')

    this.cadastroService.readById(id).subscribe(cadastro => {

      this.cadastro = cadastro

    });

  }

  deleteCadastro(): void{

    this.cadastroService.updateCadastro(this.cadastro).subscribe(() => {

      this.cadastroService.showMessege('O usuário foi deletado')

      this.router.navigate(["/cadastro/tabela"]);

    });

  }

  cancelarCadastro(): void{

    this.router.navigate(['/cadastro'])

  }

  tabelasCadastro(): void{

    this.router.navigate(['/cadastro/tabela'])

  }

}

Salve todos os arquivos e faça teste em sua máquina