

1C_configurar ambiente de desenvolvimento

abra a pasta onde esta armazenado o xamp instalado no pc, [c:/xamp](#)

e na pasta htdocs crie uma pasta chamada api

c:/xamp/htdocs/api

criar uma pasta chamado php dentro da pasta api

copiar o arquivo .htaccess que foi disponibilizado como material de apoio no curso de api angular e colar dentro da pasta php

dentro da pasta api sera criado um projeto com o nome projeto-api ou api-angular etc

ng new api_angular

aula 2D-2E

criar conexão com o DB

criar arquivo conexao.php

implentar o seguinte codigo

conexão

```
<?php
// variaveis
$url = "localhost";
$usuario = "root";
$senha = "";
$base = "api";

// conexão

$conexao = mysqli_connect($url,$usuario,$senha,$base);
?>
```

criar o arquivo listar.php

1_ nesse arquivo: primeiro é chamado a conexão

2_ implementamos a nossa query “codigo da nossa pesquisa”

3_ executar a conexão, executa também a nossa query, pra fazer a pesquisa no DB

4_ criamos um vetor para armazenar dados e um variavel para a partir dela implementar nosso indice

5_ criamos um laço de repetição, e passamos a variavel executar, que recebeu a conexão para ser executada, e a nossa query, o resultado da query ficara armazenado em executar , e atravez desse laço de repetição nos vamos percorrer nossa listagem linha por linha, e armazenar os valores de cada campo contido em cada linha no array curso

6_ uma vez concluido a listagem e armazenada no array curso, vamos encapsular convertendo para json

7_ com o codigo var_dump(“curso”) exibiremos o conteudo na tela afim de fazer um teste

```
> listar.php
<?php
// incluir a conexão
include("conexao.php");

//query sql
$sql = "SELECT * FROM curso";




//Executar a conexão e a pesquisa do sql
$executar = mysqli_query($conexao,$sql);

// vetor
$curso=[];
$indice=0;

// laço de repetição
while($linha = mysqli_fetch_assoc($executar)){
    $curso[$indice]['idCurso'] = $linha['idCurso'];
    $curso[$indice]['nomeCurso'] = $linha['nomeCurso'];
    $curso[$indice]['valorCurso'] = $linha['valorCurso'];
    $indice++;
}
// incapsular em um json
json_encode(['dados'=>$curso]);
var_dump($curso);
?>
```

para visualizar o resultado na tela:
no navegador passe a seguinte url: <http://localhost/api>




Index of /api

Name	Last modified	Size	Description
 Parent Directory		-	
 api_angular/	2023-04-27 21:37	-	
 php/	2023-04-27 22:24	-	

Apache/2.4.54 (Win64) OpenSSL/1.1.1p PHP/8.0.25 Server at localhost Port 80

abra a pasta php, com um duplo click

Index of /api/php

Name	Last modified	Size	Description
 Parent Directory		-	
 conexao.php	2023-04-27 22:24	167	
 listar.php	2023-04-27 22:34	586	

Apache/2.4.54 (Win64) OpenSSL/1.1.1p PHP/8.0.25 Server at localhost Port 80

click no arquivo listar.php

nossa listagem aparecera em um forma jsom sem nenhum estilo, porem isso foi só pra testar a conexão e a query para listar dados

```
array(3) { [0]=> array(3) { ["idCurso"]=> string(1) "1" ["nomeCurso"]=> string(15) "api com angular" ["valorCurso"]=> string(5) "12.00" } [1]=> array(3) { ["idCurso"]=> string(1) "2" ["nomeCurso"]=> string(17) "[curso de ingles]" ["valorCurso"]=> string(4) "0.00" } [2]=> array(3) { ["idCurso"]=> string(1) "3" ["nomeCurso"]=> string(15) "[curso de java]" ["valorCurso"]=> string(7) "1200.00" } }
```

para o crud com php temos:

cadastar dados:

```
<?php
// incluir a conexão
include("conexao.php");
// obter os dados via json
$obterDados = file_get_contents("php://input");
// extrair dados do json
$extrair = json_decode($obterDados);
// separar os dados do json
$nomeCurso = $extrair->dados->nomeCurso; // na video aula a propriedade que aqui
// nomeamos com dados, la foi nomeada como cur
$valorCurso = $extrair->dados->valorCurso;

// sql que faz o cadastro no DB
$sql = "INSERT INTO curso(nomeCurso,valorCurso)VALUE('$nomeCurso',$valorCurso)";
mysqli_query($conexao,$sql);

// exportar um json dos dados cadastrados para verificação

$curso=[
    'nomeCurso' => $nomeCurso,
    'valorCurso'=> $valorCurso
]
json_encode(['dado'=>$curso]);
?>
```

listar dados:

```
> listar.php
<?php
// incluir a conexão
include("conexao.php");

//query sql
$sql = "SELECT * FROM curso";

//Executar a conexão e a pesquisa do sql
$executar = mysqli_query($conexao,$sql);

// vetor
$curso=[];
$indice=0;

// laço de repetição para percorrer a tabela e listar os dados
while($linha = mysqli_fetch_assoc($executar)){
    $curso[$indice]['idCurso']= $linha['idCurso'];
    $curso[$indice]['nomeCurso']= $linha['nomeCurso'];
    $curso[$indice]['valorCurso']= $linha['valorCurso'];
    $indice++;
}
// incapsular em um json
json_encode(['dados'=>$curso]);
var_dump($curso);
?>
```

atualizar dados:

```
<?php
include("conexao.php");
// obter os dados via json
$obterDados = file_get_contents("php://input");
// extrair dados do json
$extrair = json_decode($obterDados);
// separar os dados do json
$nomeCurso = $extrair->dados->idCurso;
$nomeCurso = $extrair->dados->nomeCurso; // na video aula a propriedade que aqui nomea
$nomeCurso = $extrair->dados->valorCurso;

// sql que faz o cadastro no DB
$sql = "UPDATE `curso` SET `nomeCurso`='$nomeCurso',`valorCurso`='$valorCurso' WHERE :";
mysqli_query($conexao,$sql);

// exportar um json dos dados cadastrados para verificação

$curso=[
    'idCurso' =>$idCurso,
    'nomeCurso' => $nomeCurso,
    'valorCurso'=> $valorCurso
]
json_encode(['dado'=>$curso]);
?>
```

excluir dados:

```
<?php
include("conexao.php");
// obter os dados via json
$obterDados = file_get_contents("php://input");

// extrair dados do json "decodificar os dados"
$extrair = json_decode($obterDados);

// separar os dados do json
$nomeCurso = $extrair->dados->idCurso;

// sql que faz o cadastro no DB
$sql = "DELETE FROM cursos WHERE idCurso=$idCurso";
mysqli_query($conexao,$sql);

// não precisa retornar dados json

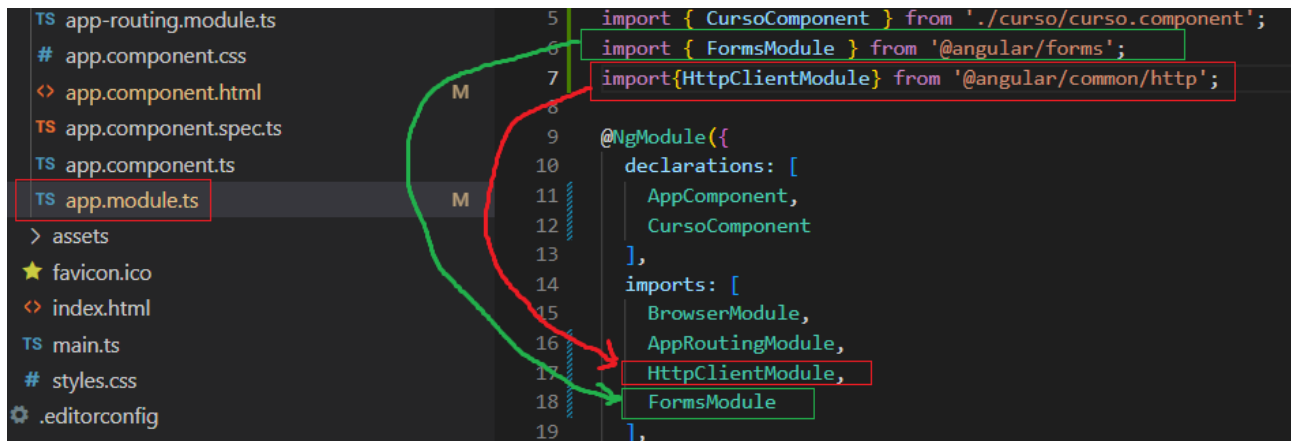
?>
```

aula 3C

module important para trabalhar com:

na imagem abaixo temos a declaração de 2 módulos muito importante pra trabalhar com Requisições em um DB
que é o HttpClientModule e o formsModule

formsModule: é imprescindível para realizar o two way data binding, poi é através desse modulo que conseguimos fazer a interpolação de dados, ou seja criar o vinculo entre o template e a classe, onde os dados vai ser armazenado e direcionado para seu destino seja para exibi-la na página web ou apenas para armazenar em um DB



httpClientModule:

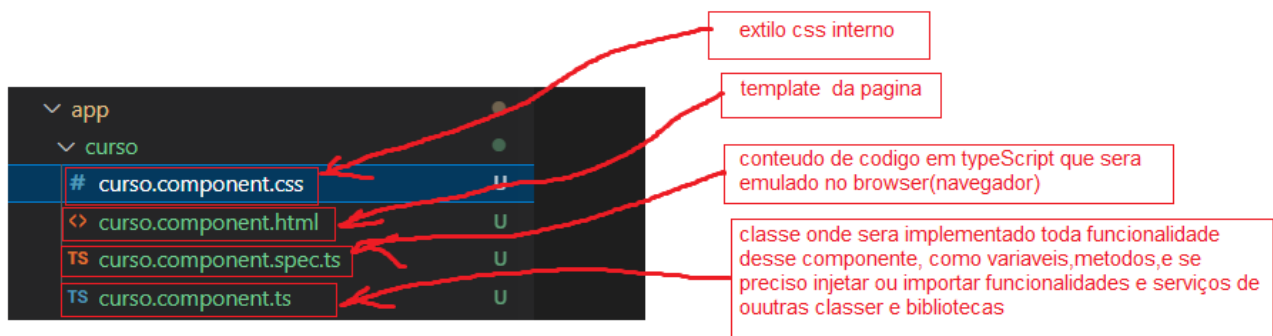
Realizar requisições HTTP é uma das tarefas mais comuns em SPAs atualmente. É por meio delas que a aplicação se comunica com APIs RESTful que lhes provêem dados e funcionalidades. Portanto, saber realizar esse tipo de ação no Angular é fundamental para todo desenvolvedor que utiliza esse framework.

Atualmente, a maioria das aplicações do tipo SPA (Single Page Applications) se comunicam com algum serviço no back-end por meio do protocolo HTTP. A fim de consumir dados e funcionalidades expostos por esse serviço, que normalmente é uma API RESTful, a SPA envia requisições HTTP utilizando um dos seus diferentes verbos (GET, POST, PUT, DELETE etc.) e trata os seus possíveis resultados (200, 404, 500, etc.).

Para que possamos fazer esse tipo de requisição no JavaScript dispomos hoje de duas APIs suportadas pelos browsers modernos: XMLHttpRequest (mais antiga) e fetch (mais recente e que utiliza [promises](#)). E quando se trata de um projeto **Angular**, contamos com um mecanismo nativo desse framework que oferece uma interface simplificada para realizar essa tarefa, mas que internamente utiliza o XMLHttpRequest. Trata-se da classe HttpClient, disponível no módulo HttpClientModule.

Obs.: SPA == single page application

composição-de-um-component:



3E_detalhando-o-conteudo-de-um-component-criar-template

como já foi até especificado acima o arquivo `curso.component.ts` do componente `curso`: é na classe contida nesse arquivo, que vai ser implementado as regras de negócios, funcionalidades como eventos de mouse, teclados, inicialização de algum comportamento na abertura da página,

construtor: usado para injetar algumas funcionalidade e criar instancias de objetos para ser usadas na classe em questão conforme e necessidade que o comportamento do nosso componente precisa

aula 4A_funcoes-ou-metodos_Crud-para-frontend

Já implementamos o crud no backend no começo desse documento, onde usamos como linguagem de programação o PHP, agora implementaremos as funções do crud no nosso componente, a partir de onde faremos a comunicação do frontend com o backend, ou seja: implementaremos no frontend as funções pelas quais, vamos enviar e requisitar os dados no nosso backend

```
1  import { Component } from '@angular/core';
2
3  @Component({
4    selector: 'app-curso',
5    templateUrl: './curso.component.html',
6    styleUrls: ['./curso.component.css']
7  })
8  export class CursoComponent {
9    cadastro():void{ }
10
11    // seria o mesmo que listar, pois atravez
12    // da listagem é que selecionamos os dados que queremos alterar
13    selecao(){ }
14    alterar(){ }
15    remover(){ }
16
17  }
```

aula 4B_chamar-o-crud-frontend-na-pagina-de-interação-com-usuario

```
1 <button (click)="cadastro()">Cadastrar</button>
2 <button (click)="selecao()">Selecionar</button>
3 <button (click)="alterar()">Alterar</button>
4 <button (click)="remover()">Excluir</button>

1 import { Component } from '@angular/core';
2
3 @Component({
4   selector: 'app-curso',
5   templateUrl: './curso.component.html',
6   styleUrls: ['./curso.component.css']
7 })
8 export class CursoComponent {
9
10  cadastro():void{ }
11  // seria o mesmo que listar, pois atravez
12  // da listagem é que selecionamos os dados que queremos alterar
13  selecao(){ }
14  alterar(){ }
15  remover(){ }
```

aula 4C_model-comportamento-igual-em-proj-MVC-cSharp

```
1
2
3 export class Curso{
4   constructor(nomeCurso:String, valorCurso:Number, idCurso?:Number){}
5
6 }
```

aula 4D_adicionar-httpClientModel-no-componente-curso

```
8 })
9 export class CursoComponent {
10  constructor(private httpClient:HttpClient){}
11  cadastro():void{ }
12  // seria o mesmo que listar, pois atravez
13  // da listagem é que selecionamos os dados que queremos alterar
14  selecao(){ }
```

aula 4E_configurar-url-base-criar-vetor-para-armazenar-os-cursos-que-serão-obtidos

```
6 selector: 'app-curso',
7 templateUrl: './curso.component.html',
8 styleUrls: ['./curso.component.css']
9 })
10 export class CursoComponent {
11  // URL base
12  // essa url é a base que possibilitara o acesso ao crud do backend,
13  // ou seja: os metodos que definimos em php,
14  url = "http://localhost/api/php/";
15
16  vetor:Curso[] | undefined;
17 }
```

url para que sera usada para comunicação entre o projeto do frontend app-php e o projeto do backend api-angular

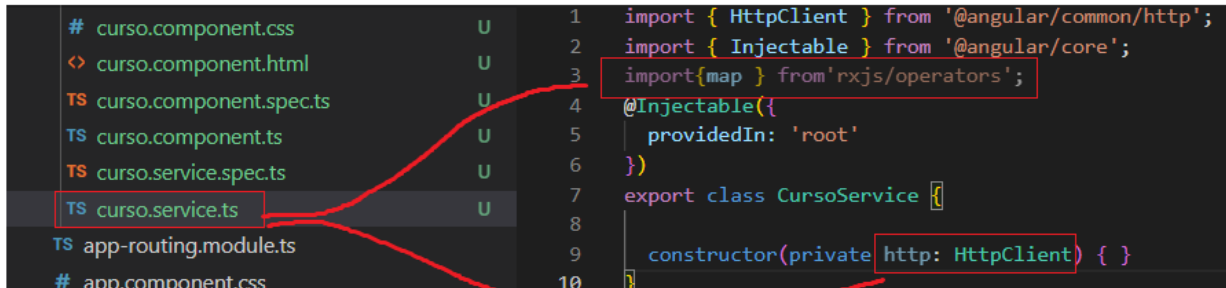
todos dados que recuperarmos do DB pelo backend, sera armazenado nesse vetor para que possamos manipula-los

aula 4F_implementar-a-camada-de-serviço

obs.: o primeiro “curso” do code abaixo indica a pasta onde o arquivo sera criado, o segundo “curso” do code abaixo indica o nome do arquivo: ficara curso.service.ts

ng g service curso/curso


aula 4G_importar-httpClient-e-o-map-para-o-serviço-curso



```
# curso.component.css      U
<> curso.component.html    U
TS curso.component.spec.ts U
TS curso.component.ts      U
TS curso.service.spec.ts   U
TS curso.service.ts        U
TS app-routing.module.ts   U
# app.component.css        U

1 import { HttpClient } from '@angular/common/http';
2 import { Injectable } from '@angular/core';
3 import { map } from 'rxjs/operators';
4 @Injectable({
5   providedIn: 'root'
6 })
7 export class CursoService {
8
9   constructor(private http: HttpClient) { }
10 }
```

aula 5A_implementar-serviço-listar-em-curso.service



```
php > src > app > curso > TS curso.service.ts > CursoService > obterCursos

import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { map } from 'rxjs/operators';
import { Curso } from './curso';
import { Observable } from 'rxjs';

@Injectable({
  providedIn: 'root'
})
export class CursoService {
  url = "http://localhost/api/php/";
  vetor: Curso[];

  constructor(private http: HttpClient) { }

  obterCursos(): Observable<Curso[]>{
    return this.http.get(this.url+"listar.php").pipe(
      map((res) => {
        this.vetor = res['cursos'];
        return this.vetor;
      })
    );
  }
}
```

aula 5B_implementar o intermediador que vai comunicar o serviço com o frontend

ou seja: esse vai ser o código de um método “função” que faz a comunicação entre template da listagem de curso, e o serviços (curso.service) onde de fato é feito a requisição dos dados na api, que foi construída em php contendo o CRUD que faz toda interação com o nosso DB

sendo assim implementaremos aqui o nosso curso.component.ts

```
import { HttpClient } from '@angular/common/http';
import { Component, OnInit } from '@angular/core';
import { Curso } from '../curso';
import { CursoService } from '../curso.service';

@Component({
  selector: 'app-curso',
  templateUrl: './curso.component.html',
  styleUrls: ['./curso.component.css']
})
export class CursoComponent implements OnInit {
  // url
  url = "http://localhost/api/php/";
  // vetor cursos
  vetor: Curso[];
  // objeto:
  curso = new Curso();

  constructor(private curso_Service: CursoService) {}
}
```

```
ngOnInit(): void {
  this.selecao();
}

selecao() {
  this.curso_Service.obterCursos().subscribe(
    (res: Curso[]) => {
      this.vetor = res;
    }
  )
}
```

aula 5C_exibir-curso-cadastrado- na-tabela-curso-do-DB de forma que possa ser visualizado no navegador

template:

```
<table style="border-width:3px;">
  <tr *ngFor="let v of vetor">
    <td>{{v.idCurso}}</td>
    <td>{{v.nomeCurso}}</td>
    <td>{{v.valorCurso}}</td>
  </tr>
</table>
```

aula 6A

Cadastrar:

nas imagens abaixo podemos ver o caminho que os dados fazem, até chegar no ultimo metodo , cadastrarCurso() onde os dados são enviados para a api, afim de serem gravados no banco de dados



Obs.:

o `.pipe()` permite percorrer os campos de um linha na tabela, para preencher os campos com os valores informado pelo usuario

o `map()` possibilita percorrer as colunas de uma tabela

após passar pelo metodo cadastrarCurso() os dados são enviados para a api, para ser mais especifico, para o metodo cadastrar da api, que esta no arquivo cadastrar.php, "cadastrar.php" pode ser visto no metodo cadastrarCurso() como complemento da url, passada para o method post

a imagem abaixo mostra a parte da api que manipulara esses dados para gravalos no banco de dados, essa parte da api é o arquivo cadastrar.php onde foi implementado o serviço que fara gravação dos dados no DB,

Remover dados “cursos” do DB:
curso.component.html

```
<tbody>
<tr *ngFor="let v of vetor">
  <td class="bordaBody">{{v.idCurso}} </td>
  <td class="bordaBody">{{v.nomeCurso}} </td>
  <td class="bordaBody">{{v.valorCurso}} </td>
  <td><button (click)="selecionarCurso(v)" style="background: blueviolet; color: aliceblue; padding: 4px;">Selecionar
  <td>
    <button (click)="alterar()" style="width:70px; background: orange; color: green;">alterar</button>
    <button (click)="remover()" style="width:70px; background: red; color: blue;">remover</button>
  </td>
</tr>
</tbody>
</table>
</div>
<form method="post" style="margin: 10px;" >
  <label style="margin-left: 10px;">nome do curso</label>
  <input style="margin-left: 10px;" type="text" name="nomeCurso" [(ngModel)]="curso.nomeCurso" >
  <label style="margin-left: 10px;">valor do curso</label>
  <input style="margin-left: 10px;" type="text" name="valorCurso" [(ngModel)]="curso.valorCurso" >
  <button style="margin-left: 10px; background: green; color: aliceblue;" (click)="cadastro()" > Cadastro </button>
  <p>{{curso.nomeCurso}}</p> <p>{{curso.valorCurso}}</p>
</form>
```

1º seleciona os dados, e os valores dos dados selecionados vão preencher o formulário

2º clique no botão remover, que chama o método remover()

(property) CursoComponent.curso: Curso

curso.component.ts

```
remover() {
  // metodo remover chamado pelo click no botão remover, na pagina de listagem de dados
  this.curso_Service.removerCurso(this.curso.idCurso).subscribe(
    (res: Curso) => {
      this.curso.nomeCurso = "";
      this.curso.valorCurso = 0;
      this.selecao();
    }
  )
}
```

o método remover chama o método removerCurso, do serviço cursoService no arquivo curso.service.ts, e passa pra ele o id do registro que foi selecionado para ser excluído.
obs.: lembrando que: apesar de quando alguém clicar em selecionar curso, só é preenchido a input de nomeCurso e valorCurso, internamente no código também, está sendo enviado para o método de selecionarCurso, o id do registro que está sendo selecionado como mostra a imagem abaixo

```
<tr *ngFor="let v of vetor">
  <td class="bordaBody">{{v.idCurso}} </td>
  <td class="bordaBody">{{v.nomeCurso}} </td>
  <td class="bordaBody">{{v.valorCurso}} </td>
  <td><button (click)="selecionarCurso(v)" style="background: blueviolet; color: aliceblue; padding: 4px;">Selecionar
```

```
selecionarCurso(c: Curso) {
  this.curso.idCurso = c.idCurso;
  this.curso.nomeCurso = c.nomeCurso;
  this.curso.valorCurso = c.valorCurso;
}
```

curso.service.ts

o metodo `removerCurso()` do arquivo `curso.service.ts` recebe o `id`, la no metodo `remover curso`, do arquivo `curso.component.ts` e envia o `id` para a api, mais precisamente para o arquivo `excluir.php`, que tem um codigo implementado, para receber esse `id`, e passar para uma query que tem um comando `sql` para excluir um registro relacionado a esse `id`

```
removerCurso (id:any):Observable<Curso>{  
  
  const url = `${this.url}excluir?idCurso=${id}`;  
  console.log(url); // testar se o id esta indo na url  
  return this.http.delete<Curso>(url);  
}
```

api_arquivo_excluir.php

```
<?php  
  
include("conexao.php");  
  
$idCurso=json_decode( $_GET["idCurso"]);  
  
// comando Sql  
$sql = "DELETE FROM cursos WHERE idCurso = $idCurso";  
  
// Executar a conexão e a pesquisa do sql  
$executar = mysqli_query($conexao,$sql);  
?>
```

metodo de alterar registro no banco de dados

```
curso.component.html
<tr *ngFor="let v of vetor">
  <td class="bordaBody">{{v.idCurso}} </td>
  <td class="bordaBody">{{v.nomeCurso}} </td>
  <td class="bordaBody">{{v.valorCurso}} </td>
  <td><button (click)="selecionarCurso(v)" style="background: blueviolet; color: aliceblue; padding: 4px;">Selecionar
  </td>
  <button (click)="alterar()" style="width: 70px; background: orange; color: green;">alterar</button>
  <button (click)="remover()" style="width: 70px; background: red; color: blue;">remover</button>
</tr>

curso.component.ts
alterar() {
  console.log("chego em curso.component.ts: " + JSON.stringify(this.curso));
  this.curso_Service.atualizarCurso(this.curso).subscribe(
    (res) => {
      // limpar os valores
      this.curso.nomeCurso = "";
      this.curso.valorCurso = 0;
      // atualizar a listagem
      this.selecao();
    }
  );
}

selecionarCurso(c: Curso) {
  this.curso.idCurso = c.idCurso;
  this.curso.nomeCurso = c.nomeCurso;
  this.curso.valorCurso = c.valorCurso;
}
```

arquivo curso.service.ts

```
atualizarCurso(c: Curso): Observable<Curso[]> {
  console.log("chego em curso.service.ts: " + c.idCurso);
  // executar alteração via url
  return this.http.put(this.url + 'alterar.php/', c);
  // percorrer o vetor para saber qual é o id do curso alterado para saber quem tem que ser alterado
  .pipe(map((res) => {
    const cursoAlterado = this.vetor.find((item) => {
      return item['idCurso'] === c['idCurso'];
    });
    // alterar o valor do vetor local
    if (cursoAlterado) {
      cursoAlterado['nomeCurso'] = c['nomeCurso'];
      cursoAlterado['valorCurso'] = c['valorCurso'];
    }
    return this.vetor;
  }));
}
```

recebe os dados do registro que sera alterado e envia para a api, no arquivo alterar.php

arquivo alterar.php
metodo alterar ou atualizar

```
<?php
// incluir a conexão
include("conexao.php");
$obterDados = file_get_contents("php://input");
//$obterDados=json_decode( $_GET["cursos"]);

$extrair = json_decode($obterDados);
// separar dados
$idCurso=(int)$extrair->idCurso; // o dado extraído do json sera convertido para int
$nomeCurso=$extrair->nomeCurso;
$valorCurso=(float)$extrair->valorCurso; // o dado extraído sera convertido para float

//query sql
$sql = "UPDATE cursos SET nomeCurso='$nomeCurso',valorCurso=$valorCurso WHERE idCurso=$idCurso";

// Executar a conexão e a pesquisa do sql
$executar = mysqli_query($conexao,$sql);

// vetor para exportar os daos cadastrado
$curso=[
    'idCurso'=> $idCurso,
    'nomeCurso' => $nomeCurso,
    'valorCurso'=> $valorCurso];
// incapsular em um json
echo(json_encode($curso));
?>
```