



Desenvolvimento Web&Mobile

Desenvolvimento Web para Server Side

Sara Monteiro

sara.monteiro.prt@msft.cesae.pt

Formulários e PreventDefault

Quando preenchemos um formulário, etc, estamos a submeter alguma informação e ao clicar no botão ele direcciona para a página onde vamos tratar os dados.

No entanto, o utilizador não precisa / não deve visualizar essa página.

Como prevenir que o utilizador vá para a página onde estamos a tratar os dados?

```
<form action="/myform">  
  <input type="text" placeholder="Nome"></br>  
  <input type="text" placeholder="Morada"></br>  
  <button type="submit">Submeter</button>  
</form>
```

Formulários e o PreventDefault

```
const form = document.querySelector('form');  
✓ form.addEventListener("submit", function(e){  
  e.preventDefault();  
  console.log('submetido!')  
})
```

Para prevenir que seja aberta uma nova página usamos o `preventDefault()` do objecto evento.

O código irá continuar a correr e poderemos depois trabalhar os dados internamente, mas a informação que estamos a dar ao browser é para parar a execução ali.

```
submetido!
```

```
>
```

Usar os dados do Submit

Através da submissão de formulários podemos trabalhar dados de uma forma muito mais orgânica do que usando o prompt.

Insira o Nome de um Animal

Lista de Animais

- gato
- cão
- tigre
- tigre

Usar os dados do Submit

```
<h1>Insira o Nome de um Animal</h1>
<form action="/myform">
  <input type="text" placeholder="Insira o Nome de um Animal"
    id="animal"><br>
  <button>Submeter</button>
</form>

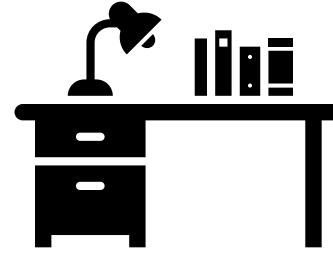
<h2>Lista de Animais</h2>
<ul id="animalList">

</ul>
```

```
const form = document.querySelector('form');
const input = document.querySelector('#animal');
const list = document.querySelector('#animalList');
```

```
✓ form.addEventListener("submit", function(e){
  e.preventDefault();
  const animalName = input.value;
  const newLi = document.createElement('li');
  newLi.innerText = animalName;
  list.append(newLi);
});
```

Exercício



1. Utilizando o ficheiro `gorcery.html` como ficheiro base, crie um evento que esteja à espera de uma submissão do formulário.
2. Quando o form é submetido, previna o comportamento por defeito.
3. Utilize a quantidade e o produto inserido e crie um novo li com os dados recolhidos. Anexe-os à lista.
4. Faça um reset dos inputs.

Input e Eventos de Alteração

```
<h3>Insira o Seu Texto</h3>
<input type="text" name="" id="myText">
</ul>
</body>
```

O evento 'input' é disparado sempre que fazemos alguma alteração no input seleccionado.

```
const input = document.querySelector('#myText');
input.addEventListener('input', function (e){
  console.log(input.value);
})
```

Temos também os eventos KeyUp e KeyDown sempre que o utilizador pressione ou largue uma tecla.

Event Bubbling e stopPropagation()

```
btn.addEventListener('click', function (e) {  
    alert('The button was clicked!');  
    e.stopPropagation();  
});
```



O Event Bubbling acontece quando temos no nosso HTML um elemento dentro de um elemento e a ambos aplicamos o mesmo evento. Por exemplo, um botão dentro de uma div ambos com um evento de click.

Para evitar que os eventos interfiram uns com os outros usamos o método stopPropagation()

[Documentação](#)

Event Delegation

- O Event Delegation é usado atribuindo o evento a um elemento pai que o estende para os seus filhos.
- É útil por exemplo em caso em que estamos a adicionar li's conforme o conteúdo que o utilizador vai inserindo e queremos que ele seja criado já com um evento de clique.
- Para o Event Delegation usamos o atributo target.

[Documentação](#)

```
<div>
  <button>Button 1</button>
  <button>Button 2</button>
  <button>Button 3</button>
</div>
```

```
const buttons = document.querySelectorAll('button')

buttons.forEach(button => {
  button.addEventListener("click", (event) => {
    console.log(event.target.innerText)
  })
})
```

← Sem delegação

Com delegação →

```
const div = document.querySelector('div')

div.addEventListener("click", (event) => {
  if(event.target.tagName === 'BUTTON') {
    console.log(event.target.innerText)
  }
})
```

The Call Stack

- É o mecanismo segundo o qual o JS interpreta e acompanha um script com várias funções.
- É como o JS sabe a função que está a ser corrida a cada momento, as que serão chamadas a seguir, etc.
- É como se fosse um dedo a apontar o sítio do script onde o JS está.



[Exemplo](#)

Web APIs

Em JS, só é executada uma função de cada vez. No entanto, o que acontece à nossa página quando estamos a processar alguma coisa que vá para o Backend, como por exemplo dados de um formulário?

- Os browsers vêm com Web API's (normalmente escritas em C+) que são capazes de tomar conta de certas tarefas do JS no background, permitindo que continuemos a nossa navegação.
- A Call Stack do JS reconhece estas funções de Web API e passa-lhes a informação.
- Assim que o browser acaba essas tarefas, elas são desenvolvidas à stack como callback.

[exemplo](#)

AJAX e APIs

AJAX

- Assíncrono JS e XML.
- Fazer pedidos para carregar informação à medida que vamos fazendo scroll, por exemplo.

API

- Application Programming Interface
- Interface que permite a um Website comunicar com outro / com outro elemento, etc.
- Comunicação com Endpoints que nos respondem com dados que nós podemos consumir na nossa aplicação

[Exemplo](#)

AJAX e APIs

AJAX

- Assíncrono JS e XML.
- Fazer pedidos para carregar informação à medida que vamos fazendo scroll, por exemplo.

API

- Application Programming Interface
- Interface que permite a um Website comunicar com outro / com outro elemento, etc.
- Comunicação com Endpoints que nos respondem com dados que nós podemos consumir na nossa aplicação

[Exemplo](#)

JSON

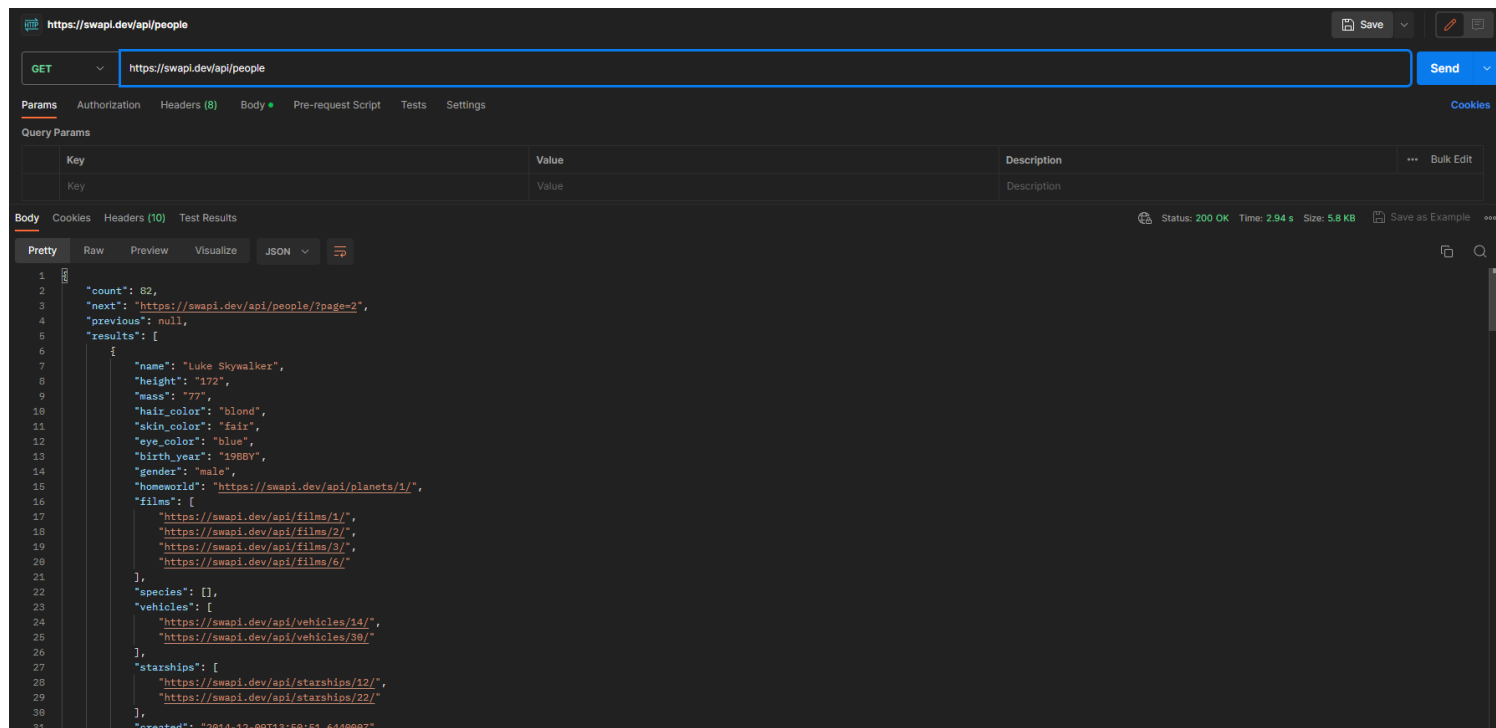
- JavaScript Object Notation
- O JSON não é só usado para JS, é usado também em outras APIs que forneçam dados para Web como Python, PHP, etc..
- O Objecto de JSON não pode ser usado directamente em JS, temos primeiro que o converter para um objecto JS através do método `JSON.parse(objectoJSON)`.

[documentação](#)

```
"browsers": {  
  "firefox": {  
    "name": "Firefox",  
    "pref_url": "about:config",  
    "releases": {  
      "1": {  
        "release_date": "2004-11-09",  
        "status": "retired",  
        "engine": "Gecko",  
        "engine_version": "1.7"  
      }  
    }  
  }  
}
```

Postman e teste de APIs

- O postman é a ferramenta mais conhecida para testar API's.
- Ele pode ser usado para testar APIs de dados online como a do [Star Wars](#) ou para testar a API customizada do backend da aplicação que estamos a construir.



Verbos HTTP

GET: Traz informação

POST: Serve para enviar dados para algum lado, normalmente para ser guardada numa base de dados.

Por exemplo quando submetemos um formulário iremos usar o POST.

PUT / PATCH: Actualiza informação.

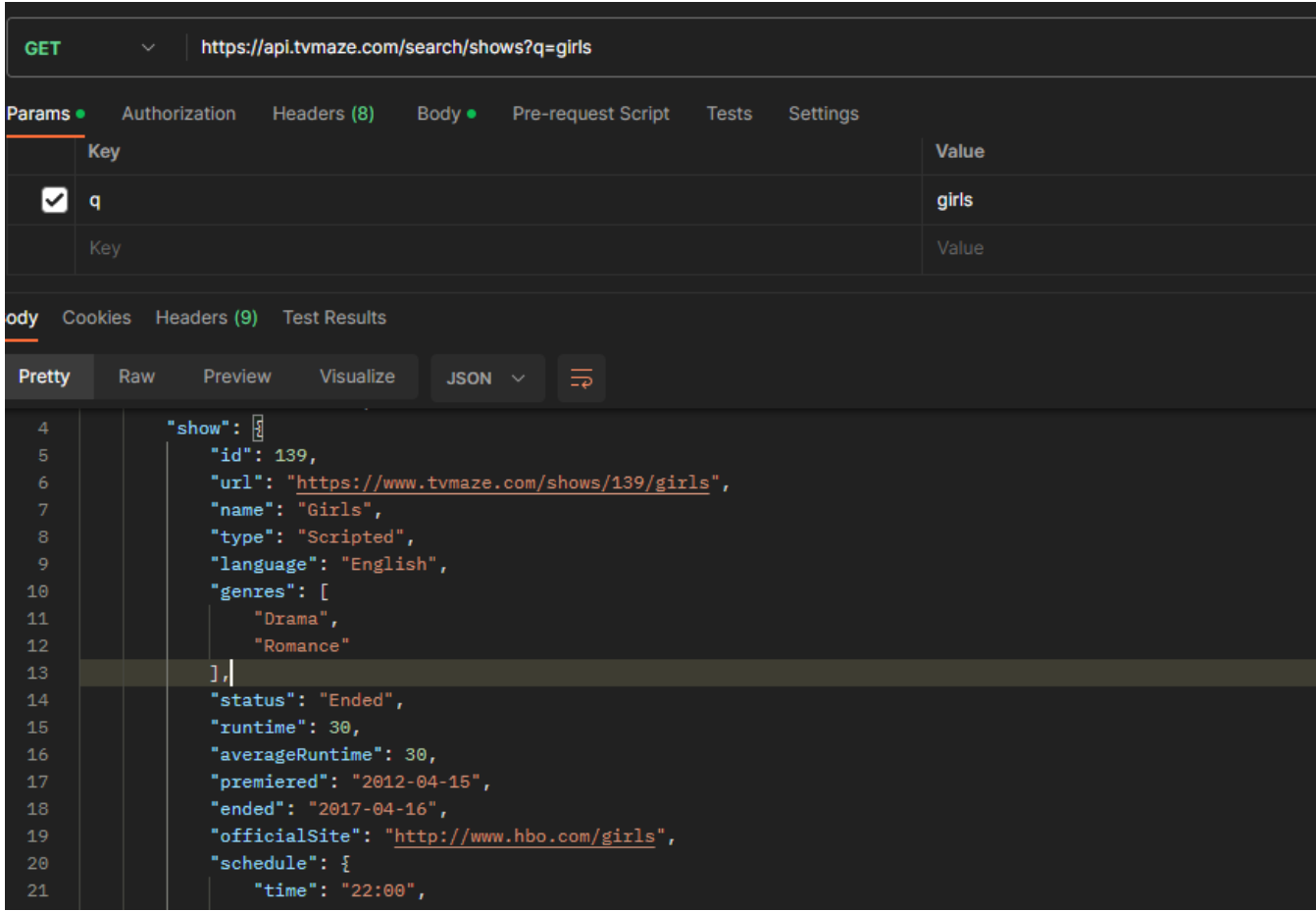
DELETE: Apaga informação

[documentação](#)

Query Strings

É um parâmetro que enviamos no url e que envia ao Backend a informação da pesquisa que queremos fazer

[exemplo](#)



The screenshot shows a REST client interface with a GET request to `https://api.tvmaze.com/search/shows?q=girls`. The 'Params' tab is active, showing a single parameter `q` with the value `girls`. The 'Body' tab is also active, displaying the JSON response in 'Pretty' format. The response is a JSON array with one object representing a TV show.

```
4  "show": {
5    "id": 139,
6    "url": "https://www.tvmaze.com/shows/139/girls",
7    "name": "Girls",
8    "type": "Scripted",
9    "language": "English",
10   "genres": [
11     "Drama",
12     "Romance"
13   ],
14   "status": "Ended",
15   "runtime": 30,
16   "averageRuntime": 30,
17   "premiered": "2012-04-15",
18   "ended": "2017-04-16",
19   "officialSite": "http://www.hbo.com/girls",
20   "schedule": {
21     "time": "22:00",
```

Fetch API

- nova forma do JS fazer pedidos a APIs
- suporta promises
- não suportado no IE

```
✓ const loadStarWars = async () => {  
  const res = await fetch("https://swapi.dev/api/films/1/");  
  const data = await res.json();  
  return data;  
};  
  
✓ const getData = async () => {  
  let result = await loadStarWars();  
  console.log(result);  
};  
  
getData();
```

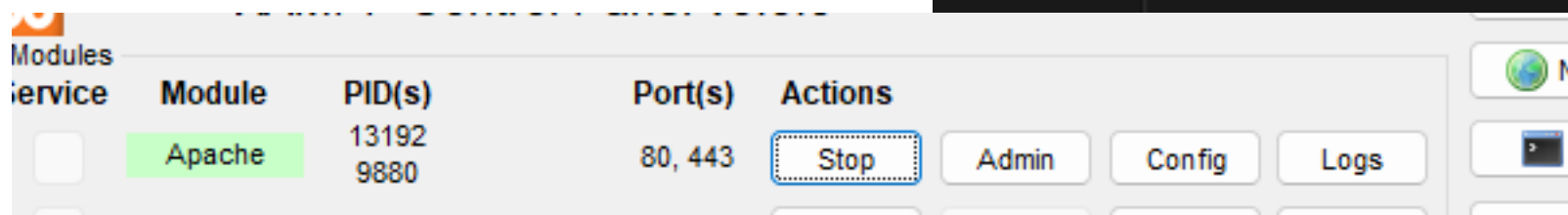
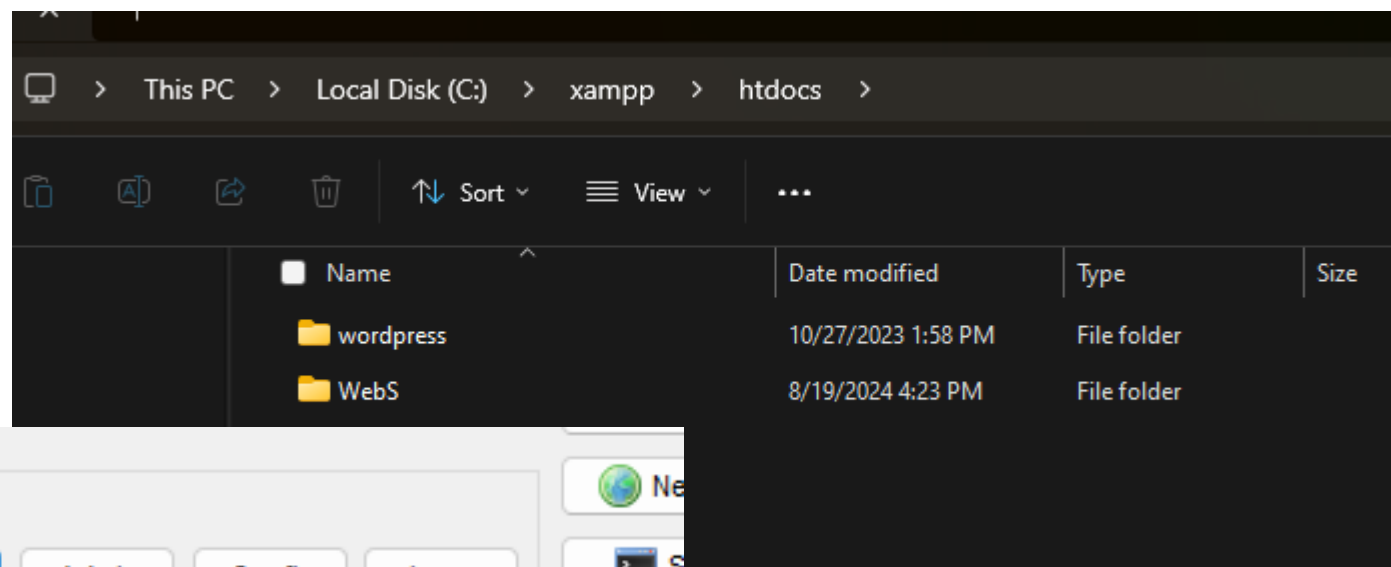
PHP

- Lançada em 1995
- Alimenta cerca de 80% dos sites porque a maioria do software popular como o Wordpress a usam
- Tem o Webhosting é o mais barato
- Tem evoluído muito nos últimos anos, nomeadamente com o OOP



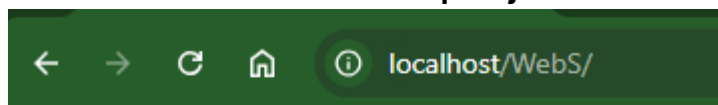
PHP: instalação

- Para usarmos o php precisamos de simular um servidor no nosso PC que interprete a linguagem. Usaremos o XAMPP.
- Instalar o [XAMPP](#)
- Alojarmos o nosso projecto na pasta htdocs do xampp e fazer start no servidor no xampp



PHP: usar no nosso projecto

- Finalmente abrimos o projecto que se encontra no htdocs no visual studio code
- Criamos um ficheiro chamado index.php e dentro criaremos a primeira linha de código em PHP
- Para correr o código e verificar se está tudo a funcionar abriremos o browser no nosso servidor e iremos seleccionar o nosso projecto



Hello World



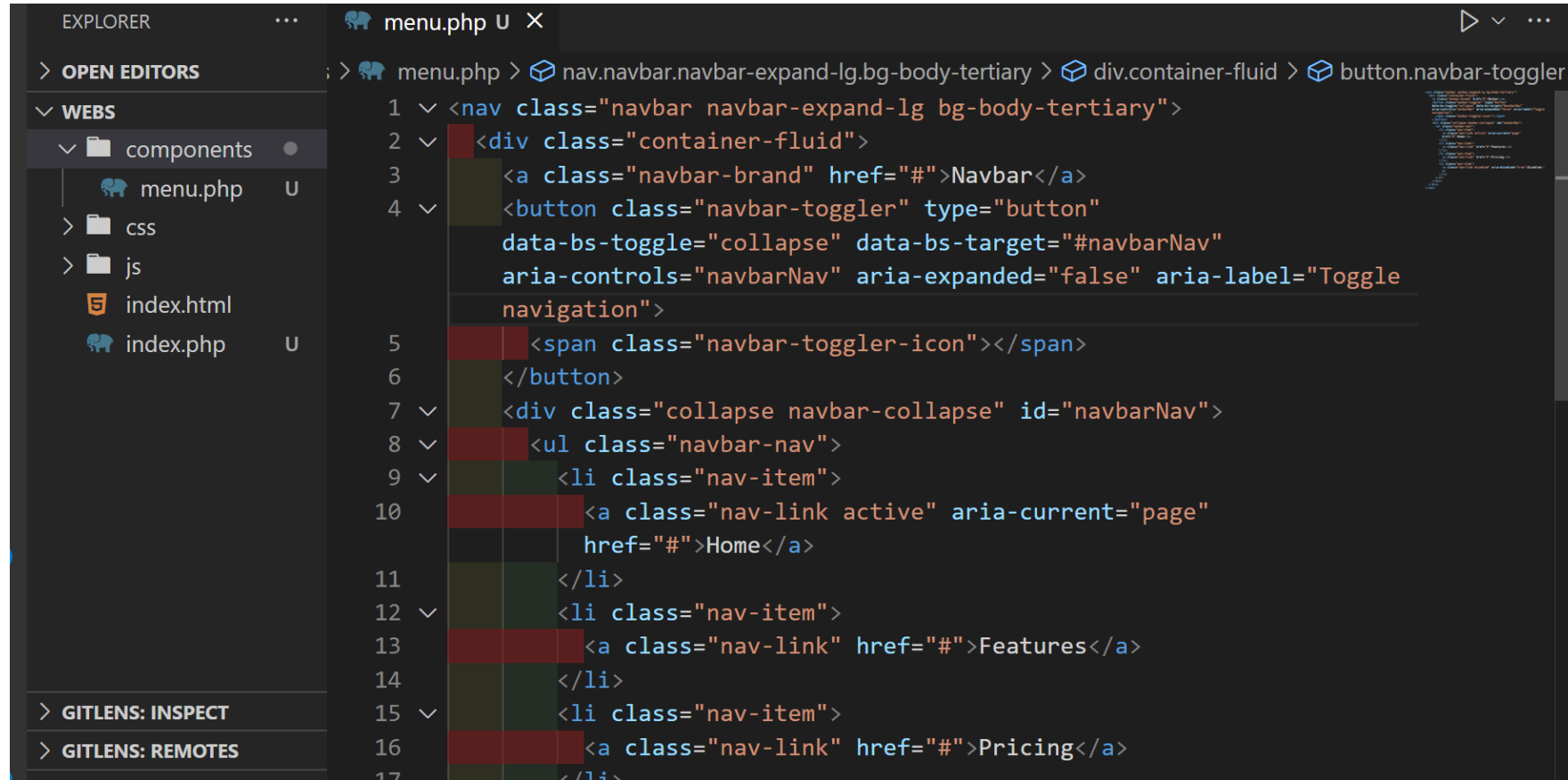
PHP: usar no nosso projecto

- De agora em diante iremos criar páginas com código FE (HTML, CSS e JS) de forma a conseguir colocar dados de php lá dentro (por exemplo, dados da base de dados). Para isso os ficheiros serão do tipo .php

```
index.php U X
index.php > html > body
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>PHP here we go</title>
7  </head>
8  <body>
9      <h1>Sou código Frontend</h1>
10     <?php
11         echo 'Sou código PHP';
12     ?>
13 </body>
14 </html>
15
16
```

PHP: o include

- O include permite-nos dividir o código php em vários ficheiros, criando componentes reutilizáveis.
Ex: menú e rodapé.
- Através do include podemos carregar outro ficheiro em determinado local da nossa página.

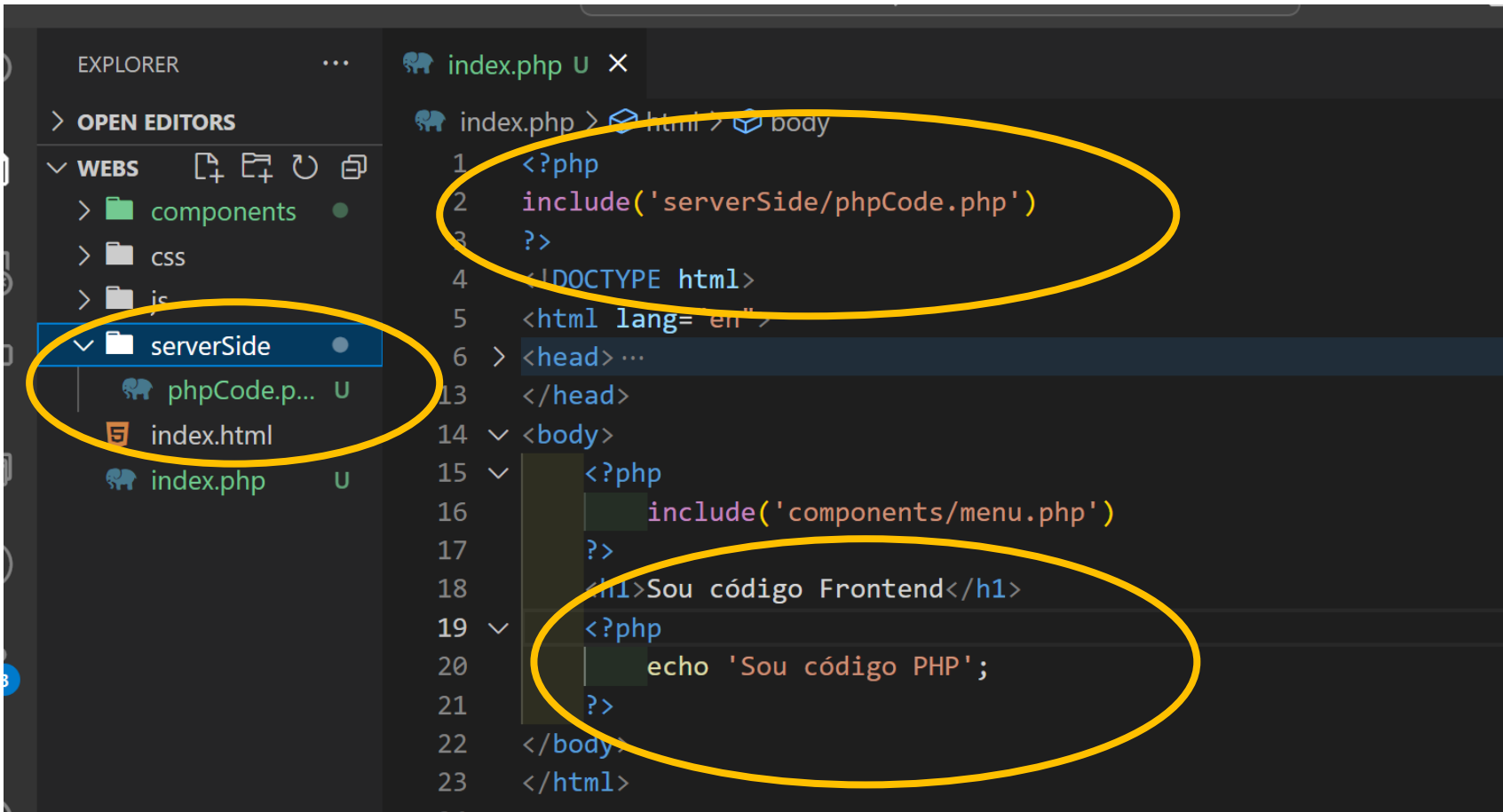


```
1 <nav class="navbar navbar-expand-lg bg-body-tertiary">
2 <div class="container-fluid">
3 <a class="navbar-brand" href="#">Navbar</a>
4 <button class="navbar-toggler" type="button"
  data-bs-toggle="collapse" data-bs-target="#navbarNav"
  aria-controls="navbarNav" aria-expanded="false" aria-label="Toggle
  navigation">
5 <span class="navbar-toggler-icon"></span>
6 </button>
7 <div class="collapse navbar-collapse" id="navbarNav">
8 <ul class="navbar-nav">
9 <li class="nav-item">
10 <a class="nav-link active" aria-current="page"
  href="#">Home</a>
11 </li>
12 <li class="nav-item">
13 <a class="nav-link" href="#">Features</a>
14 </li>
15 <li class="nav-item">
16 <a class="nav-link" href="#">Pricing</a>
17 </li>
```

PHP: o include

```
index.php U X
index.php > html > body
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>PHP here we go</title>
7      <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css"
        rel="stylesheet"
        integrity="sha384-QWTKZyjpPEjISv5WaRU90FeRpok6YctnYmDr5pN1yT2bRjXhOhJMhY6hW+ALEwIH"
        crossorigin="anonymous">
8      <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.
        min.js"
        integrity="sha384-YvpcrYf0tY3lHB60NNkmXc5s9fDVZLESaAA55NDzOxhy9GkcIdslK1eN7N6jIeHz"
        crossorigin="anonymous"></script>
9  </body>
10 </head>
11 <body>
12 <?php
13     include('components/menu.php')
14 >
15 <h1>Sou código Frontend</h1>
```


PHP: o include



```
1 <?php
2 include('serverSide/phpCode.php')
3 ?>
4 <!DOCTYPE html>
5 <html lang="en">
6 <head>...
13 </head>
14 <body>
15 <?php
16 include('components/menu.php')
17 ?>
18 <h1>Sou código Frontend</h1>
19 <?php
20 echo 'Sou código PHP';
21 ?>
22 </body>
23 </html>
```

Num ficheiro .php podemos incluir quantos includes quisermos, nomeadamente o que gera o código html para os menús, outro que crie a lógica server side para aquela página, etc..

PHP: uso de variáveis

phpCode.php U X

serverSide > phpCode.php > ...

```
1 <?php
2
3 $greetings = 'Hello Developers!';
4
5
```

index.php U X

index.php > html > body

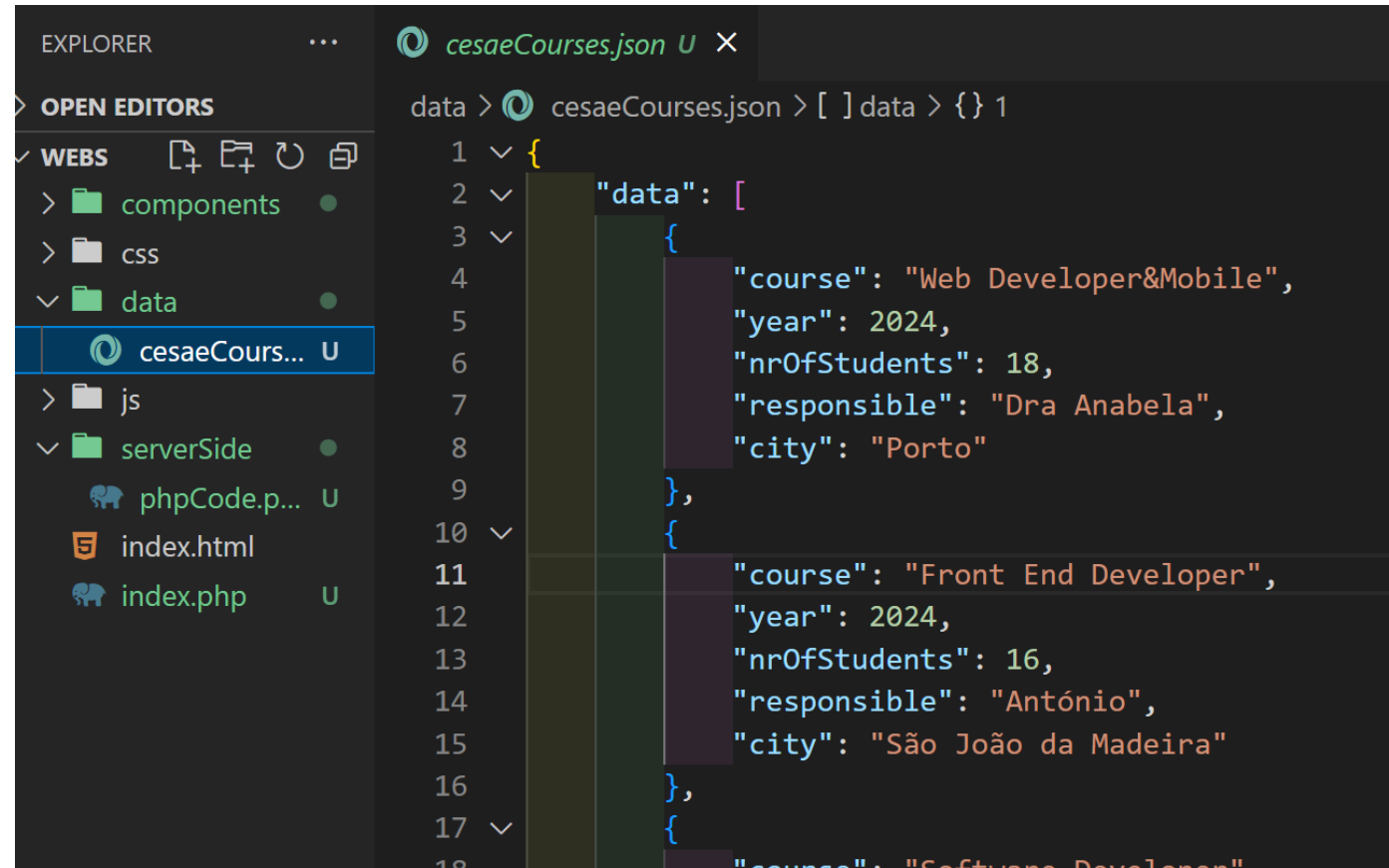
```
1 <?php
2 include('serverSide/phpCode.php')
3 ?>
4 <!DOCTYPE html>
5 <html lang="en">
6 > <head> ...
13 </head>
14 <body>
15 <?php
16 include('components/menu.php')
17 ?>
18 <h1>Sou código Frontend</h1>
19 <?php
20 echo 'Sou código PHP ';
21 echo $greetings;
22 ?>
23 </body>
24 </html>
```

PHP: interagir com dados

Uma das maiores vantagens dadas pelas linguagens de servidor é a possibilidade de interagir com dados guardados.

Normalmente tal acontece com uma base de dados mas usaremos ficheiros JSON para reproduzir o efeito, a título de exemplo.

Partindo do ficheiro dado `cesaeCourses.json` iremos trabalhar os dados e mostrar ao user.

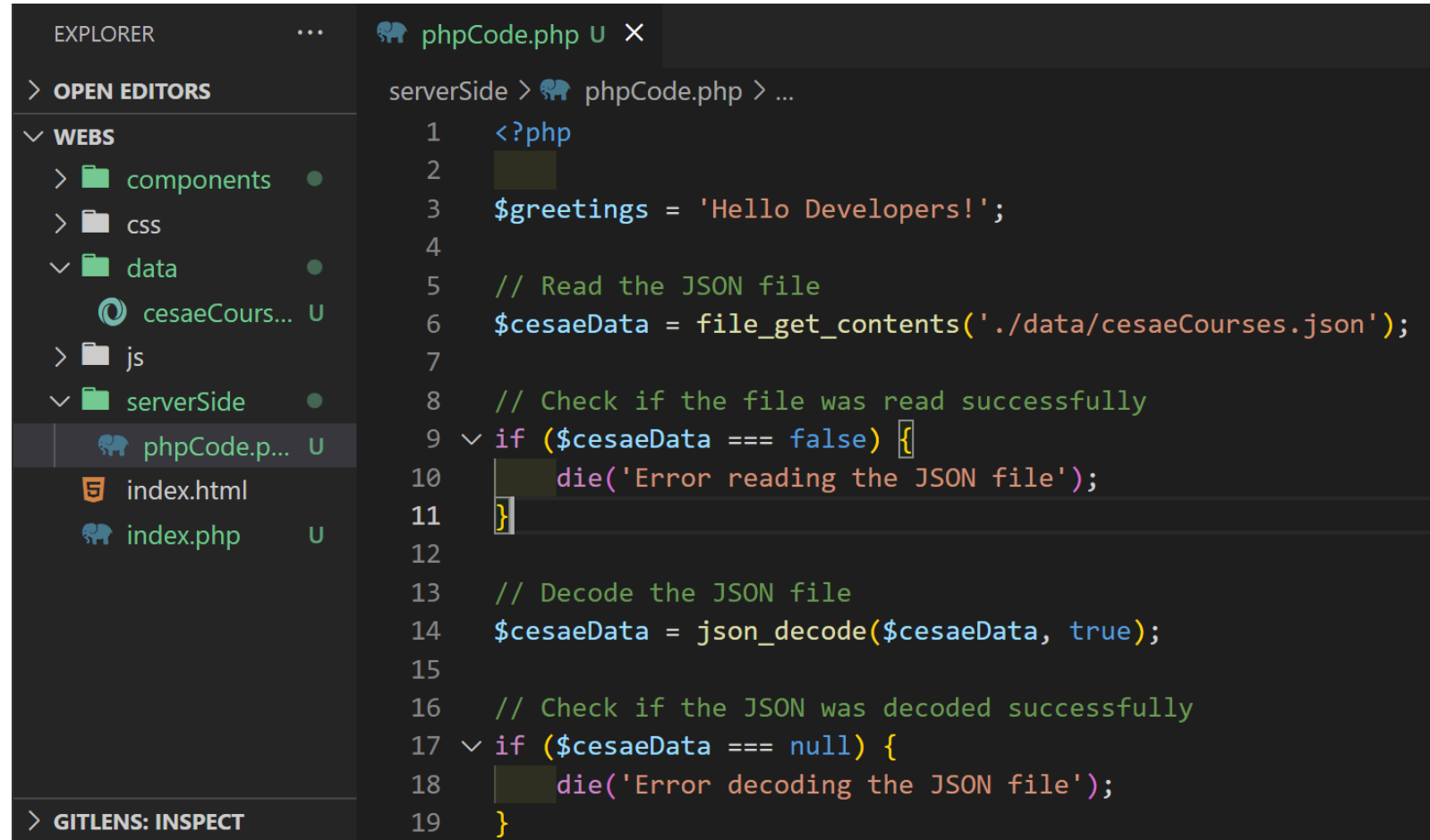


The screenshot shows a Visual Studio Code interface. On the left, the 'EXPLORER' sidebar displays a file tree with folders 'components', 'css', 'data', 'js', and 'serverSide'. The 'data' folder is expanded, showing the file 'cesaeCourses.json'. The main editor area displays the contents of 'cesaeCourses.json', which is a JSON array of course objects. The JSON is formatted with syntax highlighting and line numbers.

```
data > cesaeCourses.json > [ ] data > { } 1
1 {
2   "data": [
3     {
4       "course": "Web Developer&Mobile",
5       "year": 2024,
6       "nrOfStudents": 18,
7       "responsible": "Dra Anabela",
8       "city": "Porto"
9     },
10    {
11      "course": "Front End Developer",
12      "year": 2024,
13      "nrOfStudents": 16,
14      "responsible": "António",
15      "city": "São João da Madeira"
16    },
17    {
18      "course": "Software Developer"
```

PHP: interagir com dados

O primeiro passo será ir buscar os dados e transformar num formato que o php consiga interpretar. Para isso usaremos o `json_decode` que transforma os dados num objecto PHP.

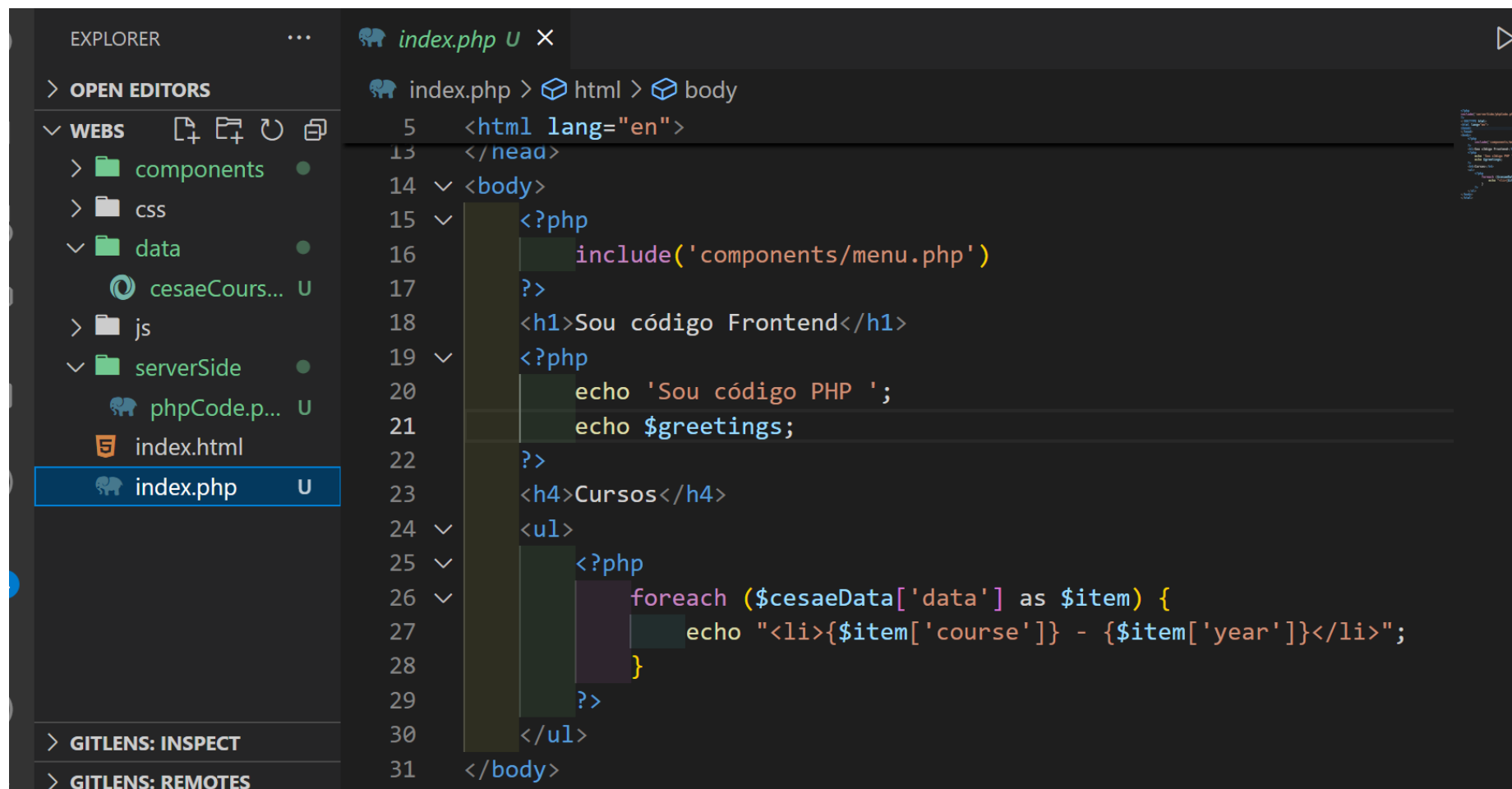


```
EXPLORER
> OPEN EDITORS
WEBS
  > components
  > css
  > data
    cesaeCours... U
  > js
  > serverSide
    phpCode.p... U
    index.html
    index.php U
  > GITLENS: INSPECT

serverSide > phpCode.php > ...
1  <?php
2
3  $greetings = 'Hello Developers!';
4
5  // Read the JSON file
6  $cesaeData = file_get_contents('./data/cesaeCourses.json');
7
8  // Check if the file was read successfully
9  if ($cesaeData === false) {
10     die('Error reading the JSON file');
11 }
12
13 // Decode the JSON file
14 $cesaeData = json_decode($cesaeData, true);
15
16 // Check if the JSON was decoded successfully
17 if ($cesaeData === null) {
18     die('Error decoding the JSON file');
19 }
```

PHP: interagir com dados

O passo seguinte será enviar e mostrar os dados quando a página carregar, por exemplo numa lista.



```
5 <html lang="en">
13 </head>
14 <body>
15 <?php
16     include('components/menu.php')
17 >
18 <h1>Sou código Frontend</h1>
19 <?php
20     echo 'Sou código PHP ';
21     echo $greetings;
22 >
23 <h4>Cursos</h4>
24 <ul>
25 <?php
26     foreach ($cesaeData['data'] as $item) {
27         echo "<li>{$item['course']} - {$item['year']}</li>";
28     }
29 >
30 </ul>
31 </body>
```

PHP: escrever dados num ficheiro

Para enviar dados que o user colocou para um ficheiro, usaremos o formulário e o método POST.

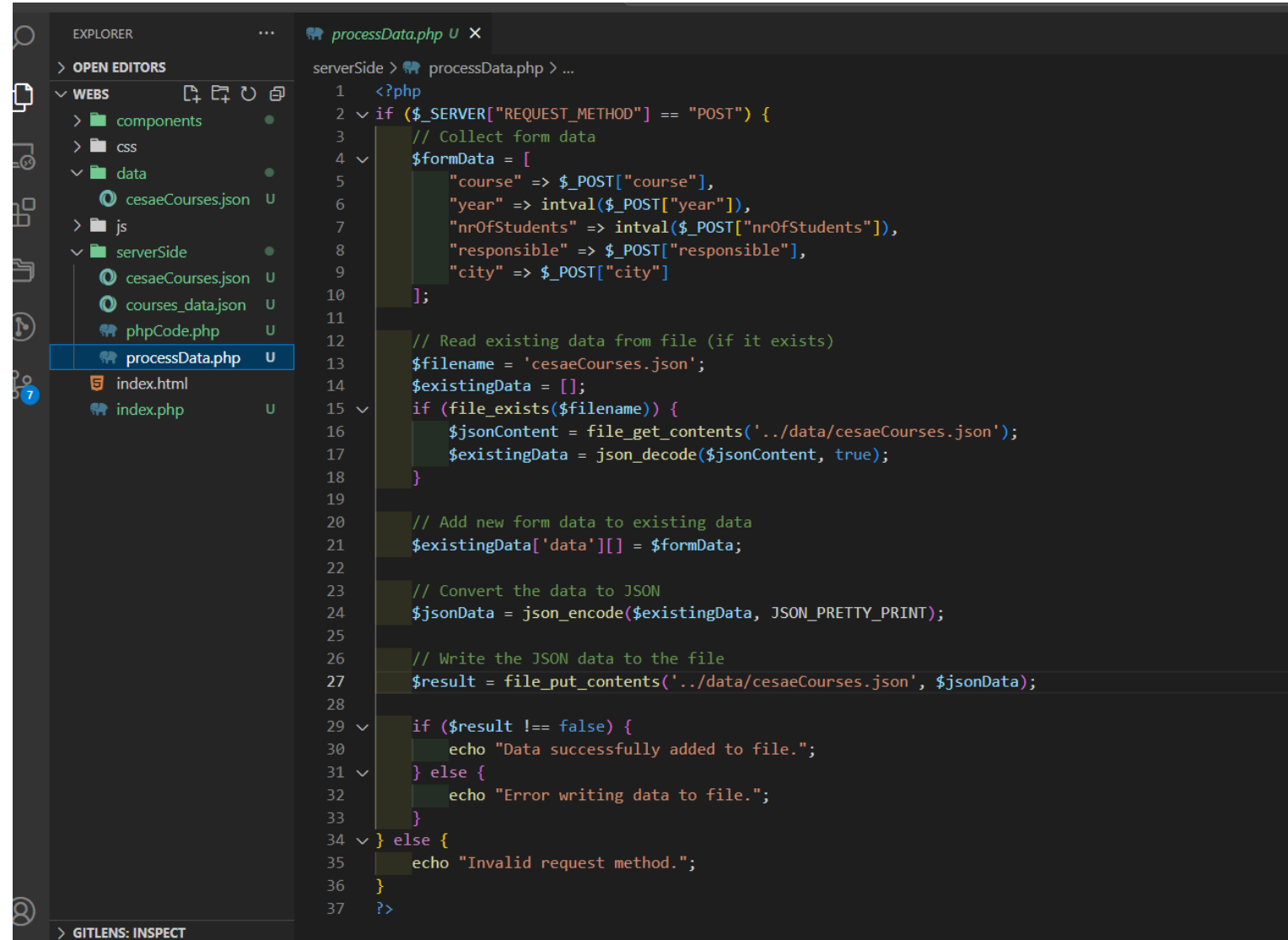
Para isso precisamos de identificar os campos que o php irá ler e que consta no campo name.

```
index.php U X
index.php > html > body > form
5  <html lang="en">
14 <body>
34 <form action="serverSide/processData.php" method="post">
35   <label for="course">Course Name:</label>
36   <input type="text" id="course" name="course" required><br><br>
37
38   <label for="year">Year:</label>
39   <input type="number" id="year" name="year" required><br><br>
40
41   <label for="nrOfStudents">Number of Students:</label>
42   <input type="number" id="nrOfStudents" name="nrOfStudents" required><br><br>
43
44   <label for="responsible">Responsible:</label>
45   <input type="text" id="responsible" name="responsible" required><br><br>
46
47   <label for="city">City:</label>
48   <input type="text" id="city" name="city" required><br><br>
49
50   <input type="submit" value="Submit">
51 </form>
52 </body>
53 </html>
54
```

PHP: escrever dados num ficheiro

Por último, criamos um ficheiro para processar os dados e nele convertemos os dados para JSON.

Temos então um sistema de dados a funcionar!

A screenshot of a code editor interface. On the left, the 'EXPLORER' sidebar shows a file tree with folders 'components', 'css', 'data', 'js', and 'serverSide'. Under 'serverSide', there are files 'cesaeCourses.json', 'courses_data.json', 'phpCode.php', 'processData.php' (highlighted), 'index.html', and 'index.php'. The main editor area shows the code for 'processData.php'. The code is a PHP script that checks if the request method is POST, collects form data, reads existing data from 'cesaeCourses.json', adds the new data, converts the array to JSON, and writes it back to the file. It includes error handling for file operations and invalid request methods.

```
1 <?php
2 if ($_SERVER["REQUEST_METHOD"] == "POST") {
3     // Collect form data
4     $formData = [
5         "course" => $_POST["course"],
6         "year" => intval($_POST["year"]),
7         "nrOfStudents" => intval($_POST["nrOfStudents"]),
8         "responsible" => $_POST["responsible"],
9         "city" => $_POST["city"]
10    ];
11
12    // Read existing data from file (if it exists)
13    $filename = 'cesaeCourses.json';
14    $existingData = [];
15    if (file_exists($filename)) {
16        $jsonContent = file_get_contents('../data/cesaeCourses.json');
17        $existingData = json_decode($jsonContent, true);
18    }
19
20    // Add new form data to existing data
21    $existingData['data'][] = $formData;
22
23    // Convert the data to JSON
24    $jsonData = json_encode($existingData, JSON_PRETTY_PRINT);
25
26    // Write the JSON data to the file
27    $result = file_put_contents('../data/cesaeCourses.json', $jsonData);
28
29    if ($result !== false) {
30        echo "Data successfully added to file.";
31    } else {
32        echo "Error writing data to file.";
33    }
34 } else {
35     echo "Invalid request method.";
36 }
37 ?>
```

Recursos

- <https://developer.mozilla.org/en-US/docs/Web/JavaScript>
- <https://www.php.net/docs.php>