07

Faça o que eu fiz na aula

Nesta aula vamos fazer o nome de usuário ser exibido na tela, para isso, vamos adotar uma regra de que o nome de usuário vai ser a primeira parte do e-mail, a parte antes do @.

Então vamos criar um arquivo chamado `contato.php` e dentro dele vamos criar uma classe `Contato` com a propriedade e-mail.

```
class Contato
{
    private $email;
}
```

Nesta classe, vamos adicionar o construtor para que essa propriedade seja inicializada.

```
public function __construct(string $email)
{
    $this->email = $email;
}
```

E para acessá-la, vamos utilizar a função getter:

```
public function getNomeUsuario(): string
{
}
```

Dentro dessa função podemos chamar a função `substr` que vai obter a nossa substring com o nome de usuário.

Para obtermos a substring, precisamos primeiro pegar qual a posição do caractere @, e para isso, vamos utilizar a função `strpos` :

```
$posicaoArroba = strpos($this->email, “@”);
```

Se o `strpos` não encontrar o @, ele vai retornar false, então vamos fazer uma condicional para tratar isso:

```
if ($posicaoArroba === false) {
return "Usuário inválido";
}
```

Agora que sabemos onde está o arroba, vamos pegar uma substring do começo da string até ele, e retornar esse resultado.

```
return substr($this->email, 0, $posicaoArroba);
```

No começo do arquivo `cadastro.php` vamos incluir este arquivo e instanciar a classe:

```
require 'Contato.php';

$contato = new Contato($_POST['email']);
```

E podemos chamar o método, lá na lista onde temos o texto do usuário:

```
<li class="list-group-item">Usuário: <?php echo $usuario->getNomeUsuario(); ?></li class="list-group
```

No entanto, utilizamos a instrução require para incluir os arquivos, e podemos usar o autoload, principalmente se o nosso sistema crescer para mais do que duas classes.

Vamos pegar as classes que temos, e logo depois da abertura da tag do PHP, adicionar o seguinte:

```
namespace App\Alura;
```

E vamos criar um diretório na raiz do projeto chamado `src`, e dentro dele, um diretório chamado `Alura` e mover os arquivos para dentro desse diretório.

Na raíz do projeto, vamos criar um arquivo `autoload.php`, e nele chamar a função `spl_autoload_register`, que aceita uma função como argumento:

```
spl_autoload_register(function ($classe) {

});
```

Agora dentro dessa função, vamos definir um prefixo de namespace que o nosso projeto vai ter que vamos chamar de `App\`

```
$prefixo = "App\\";
```

Vamos também informar o diretório raiz das classes no nosso programa:

```
$diretorio = __DIR__ . DIRECTORY_SEPARATOR . 'src' . DIRECTORY_SEPARATOR;
```

E vamos realizar uma comparação para ver se a classe que estamos carregando tem o prefixo `App\`. Para isso, precisamos pegar primeiro o tamanho do prefixo:

```
$tamanhoPrefixo = strlen($prefixo);
```

E fazer

```
if (strncmp($prefixo, $classe, $tamanhoPrefixo) !== 0) {
    return
}
```

Vamos fazer agora a substituição das barras invertidas do namespace pelo separador de diretório padrão do sistema operacional em que o PHP foi instalado com a função `str_replace` e juntar com o diretório para formar o nome do arquivo.

```
$arquivo = $diretorio . str_replace('\\', DIRECTORY_SEPARATOR, substr($classe, $tamanhoPrefixo)) . '
```

Precisamos fazer também uma condicional que só vai incluir o arquivo se ele existir:

```
if (file_exists($arquivo)) {
require $arquivo;
}
```

Com isso agora podemos voltar ao nosso arquivo `cadastro.php` e vamos apagar estas duas linhas do começo:

```
require ‘Contato.php;
require ‘Usuario.php’;
```

E trocá-las por esta:

```
require ‘autoload.php’;
```

Agora podemos chamar as classes assim:

```
$contato = new \App\Alura\Contato($_POST['email']);
$usuario = new \App\Alura\Usuario($_POST['nome']);
```

E se recarregarmos a página, ela vai ser exibida corretamente, sem precisarmos dar mais de um require no código.