

Namespaces, autoload e composer

- **Namespaces**: Particionamento lógico da estrutura de um projeto, resolve o problema de várias classes com o mesmo nome

Como usar?

- Em toda classe, na primeira linha, definir o namespace (é boa prática usar os nomes das pastas)

ex: namespace App\Model\;
class Usuario{

- Para instanciar um namespace, basta dar um use e indicar o namespace (se tiver autoload, não tem que dar os requires).

ex: use App\Model\Usuario;
\$usuario = new Usuario;

- **Autoload**: Carrega automaticamente os diretórios para nós, sem que precisemos usar o require, é sempre utilizado com namespaces.

Como usar?

- Na Composer.json, incluir o seguinte:

"autoload": {

"psr-4": {

Namespace ← "App" : "App/" } }

↗ Diretório dedicado

- o depois de montar o `composer.json`, basta *carregar* *instalar* um `Composer Update` no terminal
- e por último, chamar a `autoload` em algum lugar via `require`.

- Composer

- para começar, tudo o que precisamos é de um `composer.json` vazio no projeto.

• `composer.json`

- The `require` key

- ↳ comando que vai ~~ser~~ dizer ao `composer` quais pacotes você vai querer

ex: {

`"require": {`

`"monolog/monolog": "2.0.*"`

 } ↳ autor ↳ pacote ↳ versão

}

↳ última

- Para instalar os pacotes e seus aplicativos no `composer`, basta dar um `composer update`

Método Construtor -- `__construct()`

- ↳ função de nome reservado que será chamada assim que a classe for instanciada

Bases de Emails (abstração do PHPMailer)

- instalar PHP mailer no composer

```
"require": {  
    "phpmailer/phpmailer": "6.0.7"  
}
```

- na config.php

```
define("Mail", [  
    "host" => "",  
    "port" => "",  
    "user" => "",  
    "password" => "",  
    "from_name" => "",  
    "from_email" => ""  
]);
```

* definição de constantes globais

- na class email.php

```
<?php
```

```
namespace App\Support;
```

```
class Email {
```

```
    /** @var PHPMailer */
```

```
    private $email;
```

```
    /** @var stdClass */
```

```
    private $data;
```

```
    /** @var Exception */
```

```
    private $error;
```

Continuar

public function __construct() {

\$this->mail = new PHPMailer('true'); ** instancias ou*

\$this->data = new stdClass(); *obj.*

\$this->mail->isSMTP();

\$this->mail->isHTML();

** retiro o padding de msg*

\$this->mail->setLanguage("br");

\$this->mail->smtpAuth = true;

\$this->mail->smtpSecure = "tls";

** retiro o padding de auth*

\$this->mail->CharSet = "utf-8";

conferir na header.

\$this->mail->Host = Mail['host'];

\$this->mail->Port = Mail['port'];

** dados de quem*

\$this->mail->Username = Mail['user'];

vai enviar

\$this->mail->Password = Mail['password'];

public function add(\$subject, \$body, \$recipient_name, \$recipient_email)

\$this->data->subject = \$subject;

** dados de email.*

\$this->data->body = \$body;

\$this->data->recipient_name = \$recipient_name;

\$this->data->recipient_email = \$recipient_email;

return \$this;

continua

public function attach(\$filePath, \$fileName) {

\$this->data->attach[\$filePath] = \$fileName;

return \$this;

* adiciona arquivos

no email

}

auto-alimentação

* faz isso pra quando

quer mudar ser + fácil

public function send(\$fromName = Mail["from-name"],
\$fromEmail = Mail["from-email"]){

try {

\$this->mail->subject = \$this->data->subject;

\$this->mail->msgHtml(\$this->data->body);

\$this->mail->addAddress(\$this->data->reply-email, \$this->data->reply-name);

\$this->mail->setFrom(\$fromEmail, \$fromName);

if(!empty(\$this->data->attach)){

foreach(\$this->data->attach as \$path => \$name){

\$this->mail->addAttachment(\$path, \$name);

}

} \$this->mail->send();

return true;

} catch (Exception \$exception){

\$this->error = \$exception;

return false;}

public function error(){

return \$this->error; * devolve

}

o famoso \$e,

objeto exception (pra pegar erros)

Continua

- Onde eu for instanciar a classe

<?php

```
require __DIR__ . "/vendor/autoload.php";  
use Source\Support\Email;
```

```
$email = new Email();
```

```
$email->add(  
    "Este é o assunto",  
    "Esta é a mensagem",  
    "Nome de quem recebe",  
    "email de quem recebe"  
)*->send();
```

↳ o attach vai aqui caso eu queira

```
if (!$email->error()) {  
    var_dump(true);  
} else {  
    echo $email->error()->getMessage();  
}
```

```
}
```


Distribuição de logs com php

- pré requisitos

- ter acesso require do gerenciador no composer
- importar a minha classe PHPMailerHandler por require
- ter o phpmailer no projeto

- na pag:

<?PHP

//requires de auteloado, namespaces e etc...

→ instancia do meu agente de Log

```
$logger = new Logger("nome"); // Handler que gera log no Browser
$logger = pushHandler(new BrowserConsoleHandler($logger::Debug));
$logger = pushHandler(new StreamHandler("caminho", $logger::Warning));
$logger = pushHandler(new PHPMailerHandler("params", $logger::Crit));
```

→ Handler que gera log

em arquivo

Camadas que vão ↓
pro log.

→ Handler que envia log por email

aqui eu informo de

forma manual onde
pegar os dados extras
que eu quero no log

```
$logger → pushProcessor(function($record) {
    $record[extra][nome] = $_SERVER["HTTP_HOST"]
```

... quantos eu quiser

```
});
```

→ nível

```
ex: $logger → critical("Erro a nível tal", ["logger"] => true);
```

Mark com jquery

- baixar o jquery mark plugin e por no projeto
- incluir o arquivo onde for usar
- <SCRIPT>

```
$(document).ready(function() {
```

```
$("#cpf").mark('999.999.999-99')
```

```
});
```

PHP DOC

★ DOCBLOCKS ★

- Docblocks nada mais são que blocos de documentação no meio do seu código, mas que agem de forma "oficial" aos olhos do PHP, pois, com os docblocks inseridos no código, nós podemos obter as informações daquele método apenas de passar o mouse em cima dele.

- Como é feito

```
/**
```

```
*
```

```
*
```

```
*
```

```
*/
```

} abertura e
fechamento do
trecho de doc.

O docblock possui 3 partes:

- Sumário: descrição curta do método
- Descrição: descrição detalhada de como funciona
- Tags: são anotações pertinentes ao método
 - @param
 - @return → indicam que é uma tag

ex de uso

/**

* Essa função faz isso.

*

* Essa função faz isso com base nquilo

* e isso e aquilo dentro

*

* @param int \$id Aicl do usuario

* @return string \$nome () nome do usuario

*/

Validação com PHP

Na page:

```
if(isset($_POST['btn'])){  
    $valido = true
```

```
if(!$ctrl → validarSenha($_post['senha']) {  
    $valido = false;  
    $senhaInv = true;  
}
```

```
if($valido) {  
    Altera  
}
```

No html:

No input:

```
<? = $senhaInv? 'style="border: red solid 1px";' ?>
```

Em baixo:

```
<?PHP if($senhaInv){?>  
    <label style="color:red;"> Erro </label>  
<?PHP } ?>
```