DEPURAÇÃO, TESTE E IMPLANTAÇÃO

Douglas Nassif Roma Junior

/douglasjunior

/in/douglasjunior

douglasjunior.me

nassifrroma@gmail.com

Slides: https://git.io/vAd6S

AGENDA

- Introdução à Depuração
- Depuração com PHP Xdebug
- Xdebug com Visual Studio Code
- Testes Automatizados
- Otimização para Implantação

INTRODUÇÃO À DEPURAÇÃO

- **Depuração** (em inglês: *debugging, debug*) é o processo de encontrar e reduzir defeitos num aplicativo de software ou mesmo em hardware.

- Erros de software incluem aqueles que previnem o programa de ser executado e aqueles que produzem um resultado inesperado.

- A depuração começa com a tentativa de reprodução do problema, o que pode não ser uma tarefa simples, como em computação paralela.

INTRODUÇÃO À DEPURAÇÃO

- Após a reprodução, o problema deve ser reduzido até sua essência, para facilitar a depuração.

- É um processo iterativo em que para cada redução, uma nova execução é feita para assegurar a reprodução do problema.

- Como analogia, pode-se considerar esse processo de redução como uma forma de divisão e conquista.

DEPURAÇÃO COM PHP XDEBUG

- O **Xdebug** é uma extensão do PHP para auxiliar na depuração e desenvolvimento.
  - Contém um depurador de etapa única para usar com IDEs.
  - Incrementa a função `var_dump()` do PHP, adicionando rastreamentos de pilha de execução para Notices, Warnings, Errors e Exceptions.
  - Possui funcionalidade para gravar todas as chamadas de função e atribuição de variáveis para o disco;
  - Contém um *profiler*; e
  - Fornece funcionalidade de cobertura de código para uso com o PHPUnit.

DEPURAÇÃO COM PHP XDEBUG

- A forma de instalação do **Xdebug** varia de acordo com o sistema operacional e versão do PHP que você está utilizando.

- As instruções à seguir são indicadas para uso com o **Xampp** com **PHP** nas versões 7.0.x, 7.1x e 7.2.x.

- Pré-requisitos:
  - XAMPP for Windows: https://www.apachefriends.org/download.html
  - The VC14 builds require to have the Visual C++ Red. for Visual Studio 2015 x86 or x64 installed.
  - The VC15 builds require to have the Visual C++ Red. for Visual Studio 2017 x64 or x86 installed.

Fonte: https://gist.github.com/odan/1abe76d373a9cbb15bed

DEPURAÇÃO COM PHP XDEBUG

- Faça o Download do arquivo DLL referente a versão do PHP instalado:
  - PHP 7.0.x: https://xdebug.org/files/php_xdebug-2.5.5-7.0-vc14.dll
  - PHP 7.1.x: https://xdebug.org/files/php_xdebug-2.5.5-7.1-vc14.dll
  - PHP 7.2.x: https://xdebug.org/files/php_xdebug-2.6.0-7.2-vc15.dll

- Copie a DLL para o diretório `C:\xampp\php\ext`

- Edite o arquivo `C:\xampp\php\php.ini` e altere o valor da variável **output_buffering** para:

```
output_buffering=Off
```

DEPURAÇÃO COM PHP XDEBUG

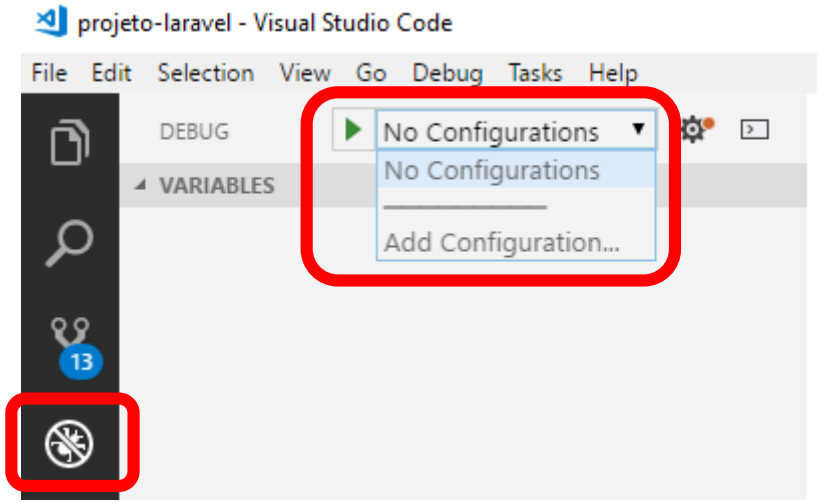
- Ainda no arquivo `C:\xampp\php\php.ini` adicione a seguinte configuração:

```
[XDebug]
zend_extension = "c:\xampp\php\ext\php_xdebug-2.6.0-7.2-vc15.dll"
xdebug.remote_autostart = 1
xdebug.profiler_append = 0
xdebug.profiler_enable = 0
xdebug.profiler_enable_trigger = 0
xdebug.profiler_output_dir = "c:\xampp\tmp"
;xdebug.profiler_output_name = "cachegrind.out.%t-%s"
xdebug.remote_enable = 1
xdebug.remote_handler = "dbgp"
xdebug.remote_host = "127.0.0.1"
xdebug.remote_log = "c:\xampp\tmp\xdebug.txt"
xdebug.remote_port = 9000
xdebug.trace_output_dir = "c:\xampp\tmp"
;36000 = 10h
xdebug.remote_cookie_expire_time = 36000
```

XDEBUG COM VISUAL STUDIO CODE

- Para integrar o **Xdebug** com o **Visual Studio Code**, primeiro é preciso instalar a extensão PHP Debug.

- Em seguida você já pode clicar na aba **Debug** e adicionar um novo **Perfil de Depuração**:



XDEBUG COM VISUAL STUDIO CODE

- No arquivo **“.vscode/launch.json”**, altere a propriedade “**program**” para apontar para o seu arquivo **index.php**.

```
{
  "version": "0.2.0",
  "configurations": [{
    "name": "Listen for XDebug",
    "type": "php",
    "request": "launch",
    "port": 9000
  }, {
    "name": "Launch currently open script",
    "type": "php",
    "request": "launch",
    "program": "public/index.php",
    "cwd": "${fileDirname}",
    "port": 9000
  }]
}
```

TESTES AUTOMATIZADOS

- Por padrão, o Laravel já vem integrado ao **PHPUnit**, inclusive com o arquivo **phpunit.xml** criado na raiz do projeto.

- O framework também contém métodos úteis que auxiliam na escrita de casos de testes de maneira expressiva.

- Por padrão, os arquivos de teste ficam dentro do diretório `tests`, que contem outros dois diretórios: `Feature` e `Unit`.

TESTES AUTOMATIZADOS

- **Testes de Unidade (Unit)**: São testes com foco em partes de código verdadeiramente pequenas e isoladas. Na maioria dos casos, testes de unidade são aplicados em apenas um único método por vez.

- **Teste de funcionalidade (Feature)**: Podem testar uma parte maior de seu código, incluindo como vários objetos interagem uns com os outros, ou ainda uma requisição HTTP completa para uma rota.

TESTES AUTOMATIZADOS

- **Criando testes de funcionalidade:**

```
php artisan make:test TarefaTest
```

- **Criando testes de unidade:**

```
php artisan make:test TarefaTest --unit
```

- **Para rodar os seus testes, execute no terminal:**

```
./vendor/bin/phpunit
```

TESTES AUTOMATIZADOS

- Por exemplo, para escrever um **teste de funcionalidade** para a rota de listagem de tarefas, você deve fazer algo como:

```php
public function testIndex() {
  $token = "Bearer {$this->loginBody->token}";

  $response = $this->withHeaders([
    'Content-Type' => 'application/json’,
    'Authorization' => $token,
  ])->json('GET', '/api/tarefas');

  $response->assertStatus(200)
    ->assertJsonStructure([
      'data' => [
        'resultado’,
        'metadados' => [
          'data_local', 'quantidade’
        ]
      ]
    ]);
}
```

TESTES AUTOMATIZADOS

- A lista completa de métodos *assert* disponíveis no Laravel você encontra em: https://laravel.com/docs/5.6/http-tests#available-assertions

- Adicionalmente, com o Laravel você também pode escrever testes de browser ou de banco de dados.

OTIMIZAÇÃO PARA IMPLANTAÇÃO

- Quando você estiver pronto para colocar o seu projeto em Produção, existem algumas coisas importantes que você precisa fazer para se certificar de que sua aplicação está rodando da forma mais eficiente possível.

- A primeira delas é otimizar a classe de **autoloader** criada pelo Composer:

```
composer install --optimize-autoloader
```

OTIMIZAÇÃO PARA IMPLANTAÇÃO

- Você também deve gerar o **cache** de configuração do Laravel, que faz com que o **Artisan** combine todos arquivos de configuração do Laravel em um único arquivo:

```
php artisan config:cache
```

- Quando você estiver em desenvolvimento, lembre-se de limpar o cache antes de modificar alguma configuração:

```
php artisan config:clear
```

OTIMIZAÇÃO PARA IMPLANTAÇÃO

- Semelhante o **cache** de configuração, você também deve fazer o **cache** de suas **rotas** para reduzir o tempo de carregamento durante as requisições:

```
php artisan route:cache
```

- Para desfazer:

```
php artisan route:clear
```

DÚVIDAS?

Douglas Nassif Roma Junior

/douglasjunior

/in/douglasjunior

douglasjunior.me

nassifrroma@gmail.com

Slides: https://git.io/vAd6S