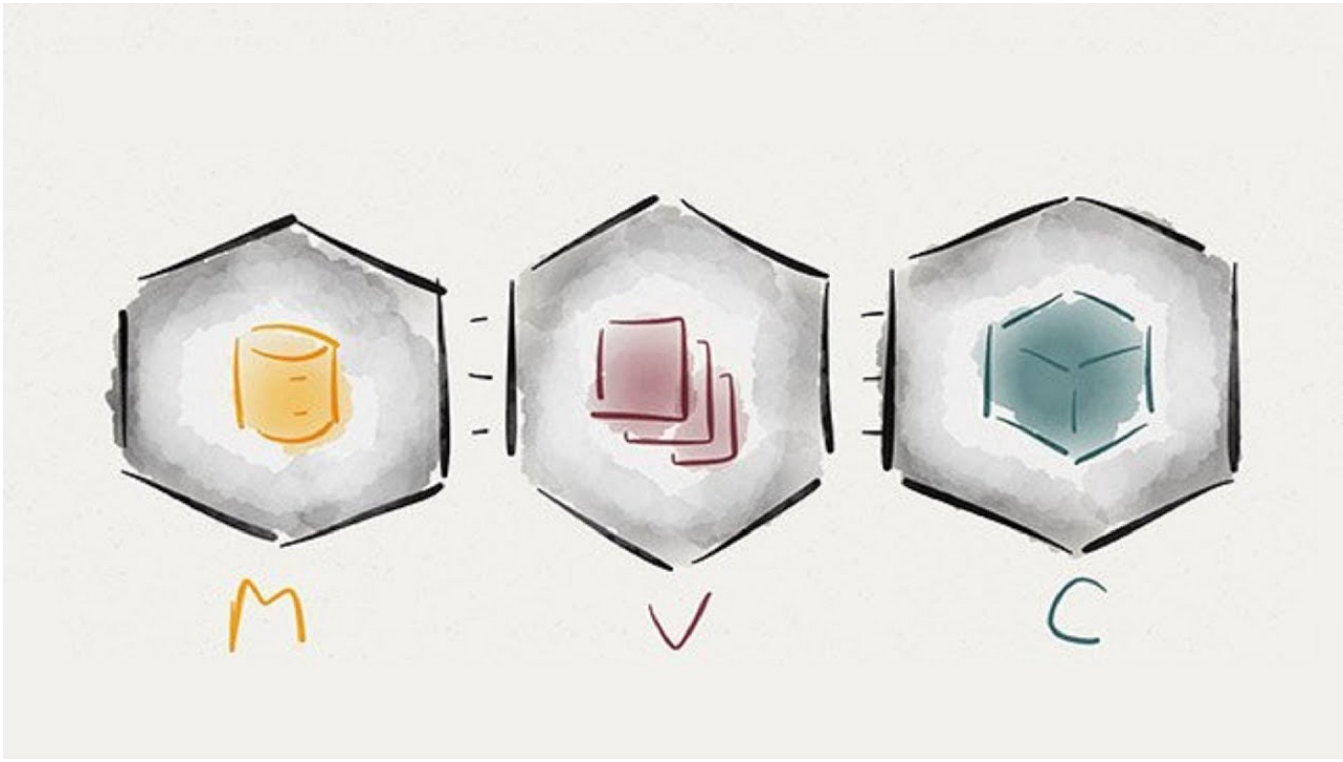
Jardel Gonçalves Follow
Feb 25, 2019 · 7 min read

# Construindo um simples framework MVC com PHP.




Neste artigo aprenderemos a construir um padrão de apresentação web sem uso de frameworks.

Código completo: https://github.com/JardelGoncalves/simple-framework-mvc-php

### Motivação

A principal vantagem de usar o padrão MVC (acrônimo para Model View Controller) é a separação das responsabilidades que você tem entre as camadas. Cada camada possui sua responsabilidade referente a uma parte do trabalho da sua aplicação. Outra vantagem é a facilidade nas manutenções posteriores da sua aplicação, pois em aplicações onde apenas um arquivo “faz tudo” a manutenção se torna uma tarefa complicada e cara.

- PostgreSQL

### Criando o Banco de dados

Para que nossa aplicação funcione corretamente, devemos criar nosso banco de dados. Acesse o PostgreSQL via pgAdmin ou terminal, crie um *database* com o nome `mvc_db` e com a seguinte tabela:

```
CREATE TABLE users (
    id SERIAL,
    name VARCHAR(100)
);
```

### A aplicação

Com o nosso framework, iremos desenvolver uma simples aplicação que lista todos os usuários no banco de dados PostgreSQL e buscar o usuário por ID.

A aplicação é extremamente simples, porém o foco está na construção do framework que pode ser utilizado em outras aplicações mais complexas.

Para iniciar o projeto, podemos criar uma pasta/diretório em sua máquina com o nome do seu projeto. Neste artigo chamaremos de `simple-mvc` e criaremos a seguinte estrutura de diretório.

```
simple-mvc
├── Application/
│   ├── controllers/
│   │   ├── Home.php
│   │   └── User.php
│   ├── core/
│   │   ├── App.php
│   │   ├── Controller.php
│   │   └── Database.php
│   ├── models/
│   │   └── Users.php
```

```
│   │       └── show.php
│   ├── .htaccess
│   └── autoload.php
│
└── public/
    ├── assets/
    │   ├── css/
    │   │   └── bootstrap.min.css
    │   ├── img/
    │   │   └── erro.png
    │   └── js/
    │       ├── bootstrap.min.js
    │       └── jquery.slim.min.js
    ├── .htaccess
    └── index.php
```

O código fonte do nosso framework ficará em `simple-mvc/Application/` . Quando o usuário acessar nossa aplicação, o diretório raiz será `simple-mvc/public` . A baixo, explicaremos a responsabilidade de cada diretório e arquivo em `simple-mvc/Application` .

- **controllers/** → Este diretório armazenará todos os controladores da aplicação que recebe os dados informados pelo usuário e decide o que fazer com eles e cada método deve realizar uma ação ou chamar `view` . Além disso, toda classe criada herda os métodos da classe `Controller` do arquivo armazenado em `Application/core/Controller.php` que será discutido em breve.
- **core/** → Neste diretório será armazenado três arquivos: `App.php` que é responsavel por tratar a URL decidindo qual controlador e qual método deve ser executado; `Controller.php` responsável por chamar o `model` , `view` e `pageNotFound` que são herdados para as classes no diretório `Application/controllers` ; E por último, o arquivo `Database.php` que armazena a conexão com o banco de dados.
- **models/** → Aqui fica a lógica das suas entidades, no nosso caso usaremos classes que irá interagir com o banco de dados e fornecer tais dados para as `views` .
- **views/** → As `views` serão responsável por interagir com o usuário. Uma das suas principais características é que cada `view` sabe como exibir um `model` .

- **autoload.php →** Neste arquivo, carregaremos de forma automática todas as classes no diretório `Application/` .

Sem mais delongas, vamos para o código 🙂.

### Navegando pelo código

O primeiro arquivo e um dos mais importante é `Application/core/App.php` que tem a responsabilidade de obter a URL, dividi-la por `/` e obtendo um array. O primeiro índice deste array contém o `controller` que iremos instanciar; O segundo índice contém o método que iremos chamar do `controller` que instanciamos e o restante são considerados como parâmetros do método. Abaixo você pode analisar o código:

```php
<?php
namespace Application\core;
/**
* Esta classe é responsável por obter da URL o controller, método (ação) e os parâmetros
* e verificar a existência dos mesmo.
*/
class App
{
  protected $controller = 'Home';
  protected $method = 'index';
  protected $page404 = false;
  protected $params = [];
  // Método construtor
  public function __construct()
  {
    $URL_ARRAY = $this->parseUrl();
    $this->getControllerFromUrl($URL_ARRAY);
    $this->getMethodFromUrl($URL_ARRAY);
    $this->getParamsFromUrl($URL_ARRAY);
    // chama um método de uma classe passando os parâmetros
    call_user_func_array([$this->controller, $this->method], $this->params);
  }
  /**
  * Este método pega as informações da URL (após o dominio do site) e retorna esses dados
  *
```

```
31      return $REQUEST_URI;
32    }
33    /**
34    * Este método verifica se o array informado possui dados na psoição 0 (controlador)
35    * caso exista, verifica se existe um arquivo com aquele nome no diretório Application/control
36    * e instancia um objeto contido no arquivo, caso contrário a variável $page404 recebe true.
37    *
38    * @param  array  $url   Array contendo informações ou não do controlador, método e parâmetros
39    */
40    private function getControllerFromUrl($url)
41    {
42      if ( !empty($url[0]) && isset($url[0]) ) {
43        if ( file_exists('../Application/controllers/' . ucfirst($url[0])  . '.php') ) {
44          $this->controller = ucfirst($url[0]);
45        } else {
46          $this->page404 = true;
47        }
48      }
49      require '../Application/controllers/' . $this->controller . '.php';
50      $this->controller = new $this->controller();
51    }
52    /**
53    * Este método verifica se o array informado possui dados na psoição 1 (método)
54    * caso exista, verifica se o método existe naquele determinado controlador
55    * e atribui a variável $method da classe.
56    *
57    * @param  array  $url   Array contendo informações ou não do controlador, método e parâmetros
58    */
59    private function getMethodFromUrl($url)
60    {
61      if ( !empty($url[1]) && isset($url[1]) ) {
62        if ( method_exists($this->controller, $url[1]) && !$this->page404) {
63          $this->method = $url[1];
64        } else {
65          // caso a classe ou o método informado não exista, o método pageNotFound
66          // do Controller é chamado.
67          $this->method = 'pageNotFound';
68        }
69      }
70    }
71    /**
72    * Este método verifica se o array informador possui a quantidade de elementos maior que 2
73    * ($url[0] é o controller e $url[1] o método/ação a executar), caso seja, é atrbuido
```

```
79    {
80      if (count($url) > 2) {
81        $this->params = array_slice($url, 2);
82      }
83    }
84  }
```

Outro arquivo importante, é `Application/core/Controller` que fornece para as classes filhas (as classes em `Application/controllers/*`) os métodos para instanciar um `model` dinâmicamente, exibir uma determinada `view` passando para a mesma os dados em um array que podem ser acessado pela variável `$data`. Em casos de métodos ou controladores não encontrados (informado pelo usuário) é chamado o método `pageNotFound` fornecido por essa classe. Abaixo está o código da mesma:

```php
<?php

namespace Application\core;

use Application\models\Users;

/**
* Esta classe é responsável por instanciar um model e chamar a view correta
* passando os dados que serão usados.
*/
class Controller
{

  /**
  * Este método é responsável por chamar uma determinada view (página).
  *
  * @param  string  $model   É o model que será instanciado para usar em uma view, seja seus mé
  */
  public function model($model)
  {
    require '../Application/models/' . $model . '.php';
    $classe = 'Application\\models\\' . $model;
    return new $classe();

```

```
30      * @param  string  $view   A view que será chamada (ou requerida)
31      * @param  array   $data   São os dados que serão exibido na view
32      */
33      public function view(string $view, $data = [])
34      {
35        require '../Application/views/' . $view . '.php';
36
37      }
38
39      /**
40      * Este método é herdado para todas as classes filhas que o chamaram quando
41      * o método ou classe informada pelo usuário nçao forem encontrados.
42      */
43      public function pageNotFound()
44      {
45        $this->view('erro404');
46      }
47    }
```

Como citado anteriormente, usaremos o PostgreSQL como banco de dados e abaixo é apresentado a classe de conexão com o banco de dados presente em `Application/core/Database.php`:

```
1   <?php
2
3   namespace Application\core;
4
5   use PDO;
6   class Database extends PDO
7   {
8     // configuração do banco de dados
9     private $DB_NAME = 'mvc_db';
10    private $DB_USER = 'postgres';
11    private $DB_PASSWORD = 'postgres';
12    private $DB_HOST = 'localhost';
13    private $DB_PORT = 5432;
14
15    // armazena a conexão
16    private $conn;
```

```
22    }
23
24    /**
25    * Este método recebe um objeto com a query 'preparada' e atribui as chaves da query
26    * seus respectivos valores.
27    * @param  PDOStatement  $stmt   Contém a query ja 'preparada'.
28    * @param  string        $key    É a mesma chave informada na query.
29    * @param  string        $value  Valor de uma determinada chave.
30    */
31    private function setParameters($stmt, $key, $value)
32    {
33      $stmt->bindParam($key, $value);
34    }
35
36    /**
37    * A responsabilidade deste método é apenas percorrer o array de com os parâmetros
38    * obtendo as chaves e os valores para fornecer tais dados para setParameters().
39    * @param  PDOStatement  $stmt         Contém a query ja 'preparada'.
40    * @param  array         $parameters   Array associativo contendo chave e valores para fornece
41    */
42    private function mountQuery($stmt, $parameters)
43    {
44      foreach( $parameters as $key => $value ) {
45        $this->setParameters($stmt, $key, $value);
46      }
47    }
48
49    /**
50    * Este método é responsável por receber a query e os parametros, preparar a query
51    * para receber os valores dos parametros informados, chamar o método mountQuery,
52    * executar a query e retornar para os métodos tratarem o resultado.
53    * @param  string   $query       Instrução SQL que será executada no banco de dados.
54    * @param  array    $parameters  Array associativo contendo as chaves informada na query e seu
55    *
56    * @return PDOStatement
57    */
58    public function executeQuery(string $query, array $parameters = [])
59    {
60      $stmt = $this->conn->prepare($query);
61      $this->mountQuery($stmt, $parameters);
62      $stmt->execute();
63      return $stmt;
64    }
```

Em `Application/models` criaremos um model que chamaremos de `Users.php` contendo apenas métodos estáticos e que interagem com o banco de dados, sendo um método que retorna todos os usuários na base de dados e o outro retorna apenas um usuário que possui um determinado `id`. Veja o código deste arquivo abaixo:

```php
<?php

namespace Application\models;

use Application\core\Database;
use PDO;
class Users
{
  /** Poderiamos ter atributos aqui */

  /**
  * Este método busca todos os usuários armazenados na base de dados
  *
  * @return   array
  */
  public static function findAll()
  {
    $conn = new Database();
    $result = $conn->executeQuery('SELECT * FROM users');
    return $result->fetchAll(PDO::FETCH_ASSOC);
  }

  /**
  * Este método busca um usuário armazenados na base de dados com um
  * determinado ID
  * @param    int     $id   Identificador único do usuário
  *
  * @return   array
  */
  public static function findById(int $id)
  {
    $conn = new Database();
    $result = $conn->executeQuery('SELECT * FROM users WHERE id = :ID LIMIT 1', array(
      ':ID' => $id
    ));

```

O prómixo arquivo é `Application/controllers/Home.php` que simplesmente herda de `Controller` todos os métodos. Na classe `Home` podem ser criados métodos que se responsabilize por uma determinada ação/processamento que inclui chamar uma determinada view (interface que interagem o usuário). No caso de `Home` existe o método `index` que é chamado toda vez em que o usuário informa na URL por exemplo `http://meusite.com/<controller>/index` substituindo o valor `<controller>` por exemplo por `home` ou quando é acessado `http://meusite.com/<controller>/` e no caso de `home`, por padrão pode ser chamado quando for acessado a URL `http://meusite.com` (exemplo). Veja o código desta classe abaixo:

```php
<?php

use Application\core\Controller;

class Home extends Controller
{
  /*
  * chama a view index.php do  /home   ou somente   /
  */
  public function index()
  {
    $this->view('home/index');
  }

}
```

Home.php hosted with ❤ by GitHub view raw

Application/controllers/Home.php

Normalmente um controller se responsabiliza por um model. No nosso caso, temos o model `Users` e o controlador que se responsabiliza pelo mesmo é `User` que pode ser encontrado em `Application/controllers/User.php` que possui dois métodos:

- index() → Que é chamado toda vez que a URL `http://<ip-ou-domino>/user/index` ou `http://<ip-ou-domino>/user/` é acessada e é retornado para a view (

- show() → É chamado quando o usuário acessa `http://<ip-ou-domino>/user/show/<id>` , caso não seja informado um ID é retornado a página de erro.

Veja o código da classe abaixo:

```php
<?php

use Application\core\Controller;

class User extends Controller
{
  /**
  * chama a view index.php da seguinte forma /user/index   ou somente   /user
  * e retorna para a view todos os usuários no banco de dados.
  */
  public function index()
  {
    $Users = $this->model('Users'); // é retornado o model Users()
    $data = $Users::findAll();
    $this->view('user/index', ['users' => $data]);
  }

  /**
  * chama a view show.php da seguinte forma /user/show passando um parâmetro
  * via URL /user/show/id e é retornado um array contendo (ou não) um determinado
  * usuário. Além disso é verificado se foi passado ou não um id pela url, caso
  * não seja informado, é chamado a view de página não encontrada.
  * @param  int   $id   Identificado do usuário.
  */
  public function show($id = null)
  {
    if (is_numeric($id)) {
      $Users = $this->model('Users');
      $data = $Users::findById($id);
      $this->view('user/show', ['user' => $data]);
    } else {
      $this->pageNotFound();
    }
  }
```

Com práticamente todas as classes criadas, devemos criar apenas os arquivos da `views` e para não deixar mais longo (porque já está! 😈) apresentaremos apenas a view `Application/views/user/show.php` e as demais views, os arquivos `.htaccess` e arquivos estáticos (css, imagens e js) você pode acessar o link do repositório que está no inicio deste artigo.

Como citado anteriormente, no arquivo `Application/core/Controller.php` o método `view()` recebe dois argumentos, o primeiro é a view que neste caso é a `Application/views/user/show.php` usaremos apenas o caminho após o diretório `views/` ou seja, apenas `user/show` sem a extensão `.php` e o segundo argumento é os dados que serão usados na view e que no caso será o usuário que possui um determinado `id`.

Esses dados são passados para em um array associativo com a chave `'user'` , você pode observar essas informações no controlador `User` citado anteriormente. É ideal entrar em detalhes deste método, pois na view usaremos uma variável (array) `$data['user']` , todo e qualquer dado que for passado no array da view ele pode ser obtido através da variável `$data` com a chave informada.

Como o método do model `Users` retorna um array, nós podemos acessar os dados deste array ( `$data['user']` ) com um `foreach` como podemos observar no código da view `user/show` abaixo:

```
<main>
  <div class="container">
    <div class="row">
      <div class="col-8 offset-2" style="margin-top:100px">

        <table class="table">
          <thead>
            <tr>
              <th scope="col">ID</th>
              <th scope="col">Name</th>
            </tr>
          </thead>
```

```
            </tr>
            <?php }?>
          </tbody>
        </table>
      </div>
    </div>
  </div>
</main>
```

Application/views/user/show.php

O próximo arquivo que fará todas essas classes funcionar, é o `autoload.php` encontrado na raiz do diretório `Application`. A sua principal responsabilidade é carregar de forma automática as classes utilizando SPL (*Standard PHP Library*). Abaixo podemos observar seu código:

```php
<?php

spl_autoload_register(function ($filename) {
  $file = '..' . DIRECTORY_SEPARATOR . $filename . '.php';
  if ( DIRECTORY_SEPARATOR === '/' ):
    $file = str_replace('\\', '/', $file);
  endif;

  if ( file_exists($file) ):
    require $file;
  else:
    echo 'Erro ao importar o arquivo!' ;
  endif;
});
```

autoload.php hosted with ❤ by GitHub view raw

Application/autoload.php

Está quasse tudo pronto!

Para finalizar e ser possivel testar nosso simples framework MVC, precisamos criar um arquivo `index.php` na raiz do diretório `public`. Toda vez que o usuário acessar uma rota em nossa aplicação este arquivo será responsável por fazer toda a aplicação

seria a parte do conteudo de cada view que difere entre elas) incialmente incluimos o nosso `autoload.php` e instanciamos um objeto `App`, ele é o principal arquivo do nosso framework e que faz toda a lógica da URL, chamada dos métodos, dos controladores e etc...

```php
<!DOCTYPE html>
<html lang="pt-br">
  <head>
    <meta charset="utf-8">
    <title>Simple Framework</title>
    <link rel="stylesheet" href="/assets/css/bootstrap.min.css">
  </head>
  <body>

  <?php
    require '../Application/autoload.php';

    use Application\core\App;
    use Application\core\Controller;

    $app = new App();

  ?>
  <script src="/assets/js/jquery.slim.min.js"></script>
  <script src="/assets/js/bootstrap.min.js"></script>
  </body>
</html>
```

index.php hosted with ❤ by GitHub view raw

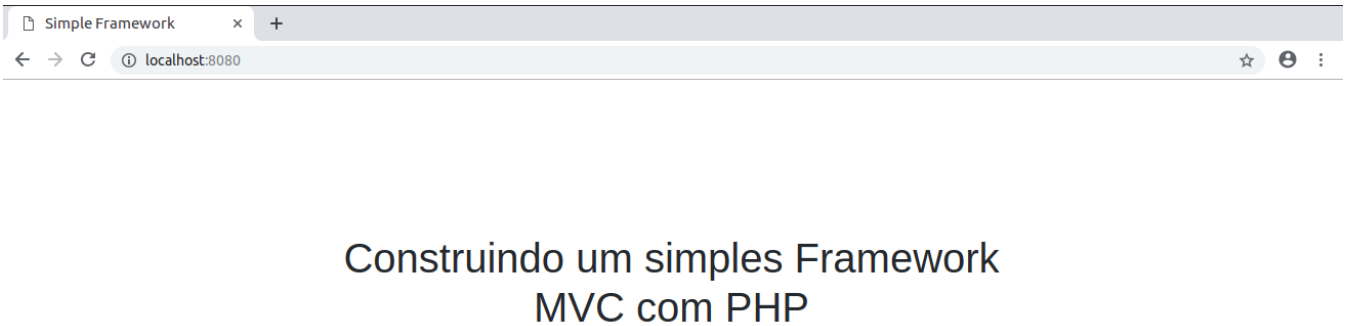
public/index.php

Quase lá!

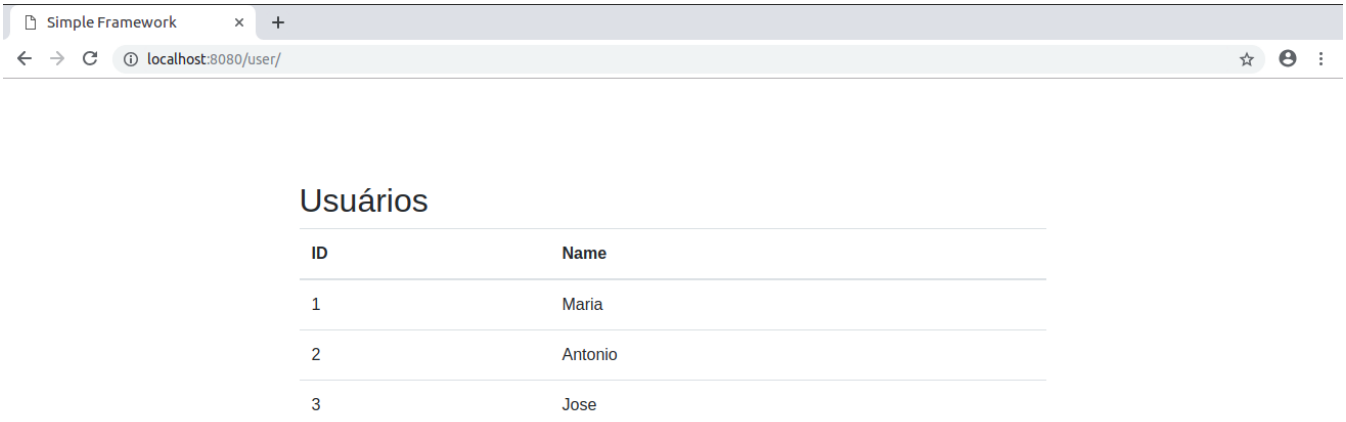
Caso você queria testar no Apache, lembre-se apenas de habilitar o módulo de reescrita. Porém, utilizaremos o servidor interno do PHP para testar. Na raiz do seu projeto (no meu caso, `simple-mvc`) digite o seguinte comando:

```
php -S localhost:8080 -t public/
```

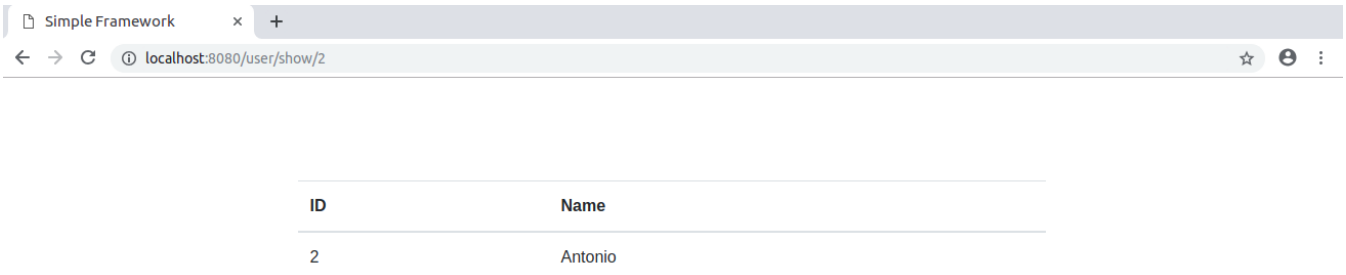
Primeiro teste que iremos fazer é acessar a raiz do nosso site, ou seja, `http://localhost:8080` . Ao acessar é exibido a seguinte página:




No segundo teste, vamos acessar `http://localhost:8080/user/` . Neste caso ele chamará o método `index()` do controlador `User` que retornará uma página listando todos os usuários contidos no banco de dados. Veja:



aquele determinado `id`.




É isso galera! Espero ter ajudado.

Abraços ♡