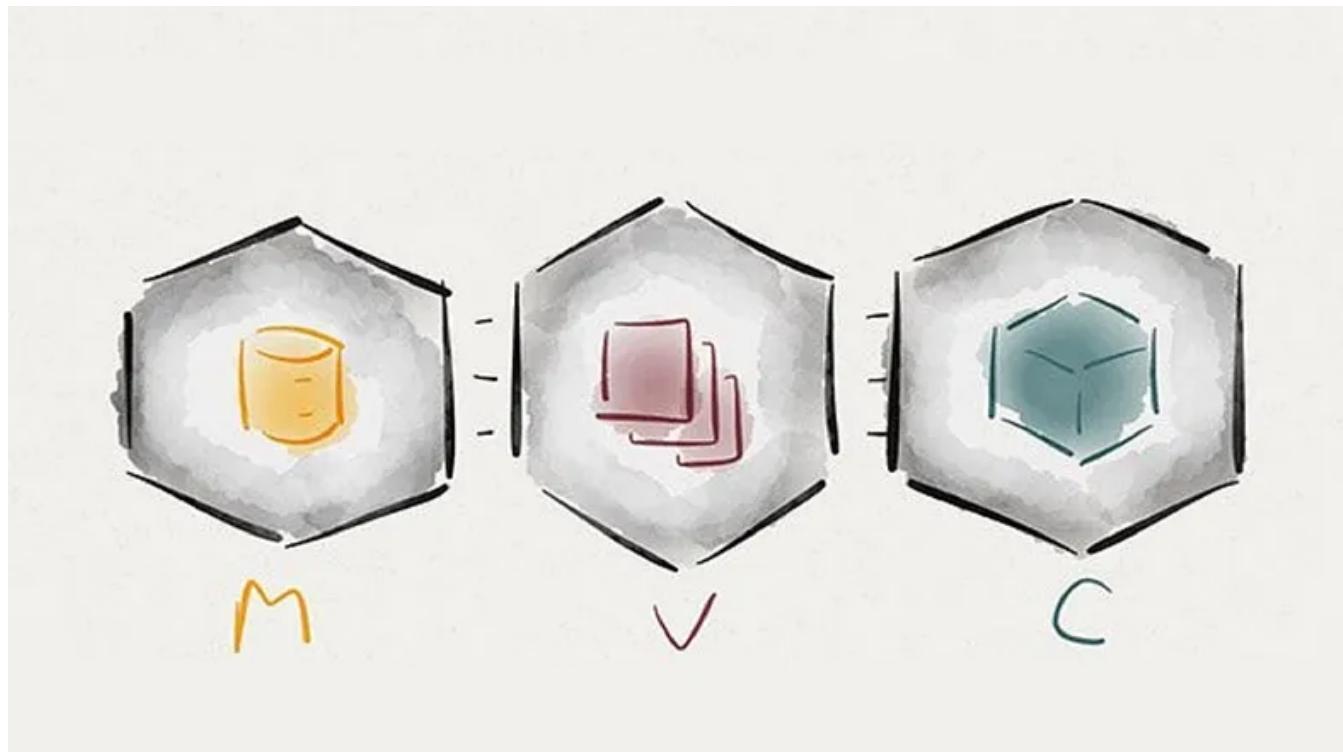


[Open in app](#)[Sign up](#)[Sign In](#)

# Construindo um simples framework MVC com PHP.

Jardel Gonçalves · [Follow](#)

7 min read · Feb 25, 2019

 [Share](#)

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.

## Requisitos

- PHP 7.2 (e o drive pgsql)
- 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
    └── views/
        ├── home/
        │   └── index.php
        └── user/
            ├── index.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 responsável 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`.
- **.htaccess** → Neste arquivo, apenas negaremos a navegação no diretório com a opção `Options -Indexes`.
- **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` queinstanciamos e o restante são considerados como parâmetros do método. Abaixo você pode analisar o código:

```

1 <?php
2 namespace Application\core;
3 /**
4 * Esta classe é responsável por obter da URL o controller, método (ação) e os parâmetros
5 * e verificar a existência dos mesmo.
6 */
7 class App
8 {
9     protected $controller = 'Home';
10    protected $method = 'index';
11    protected $page404 = false;
12    protected $params = [];
13    // Método construtor
14    public function __construct()
15    {
16        $URL_ARRAY = $this->parseUrl();
17        $this->getControllerFromUrl($URL_ARRAY);
18        $this->getMethodFromUrl($URL_ARRAY);
19        $this->getParamsFromUrl($URL_ARRAY);
20        // chama um método de uma classe passando os parâmetros
21        call_user_func_array([$this->controller, $this->method], $this->params);
22    }
23    /**
24     * Este método pega as informações da URL (após o domínio do site) e retorna esses dados
25     *

```

```
--  
26     * @return array  
27     */  
28     private function parseUrl()  
29     {  
30         $REQUEST_URI = explode('/', substr(filter_input(INPUT_SERVER, 'REQUEST_URI'), 1));  
31         return $REQUEST_URI;  
32     }  
33     /**  
34     * Este método verifica se o array informado possui dados na posição 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.
```

```
1 <?php  
2  
3 namespace Application\core;  
4  
5 use Application\models\Users;  
6  
7 /**  
8 * Esta classe é responsável por instanciar um model e chamar a view correta  
9 * passando os dados que serão usados.  
10 */  
11 class Controller  
12 {  
13  
14     /**  
15     * Este método é responsável por chamar uma determinada view (página).  
16     *  
17     * @param string $model É o model que será instanciado para usar em uma view, seja seus m  
18     */  
19     public function model($model)  
20     {  
21         require '../Application/models/' . $model . '.php';  
22         $classe = 'Application\\models\\' . $model;  
23         return new $classe();  
24     }  
25  
26  
27     /**  
28     * Este método é responsável por chamar uma determinada view (página).  
29     *  
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';
```

```
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 não forem encontrados.
42 */
43 public function pageNotFound()
44 {
45     <?php
46
47     namespace Application\core;
48
49     use PDO;
50
51     class Database extends PDO
52     {
53         // configuração do banco de dados
54         private $DB_NAME = 'mvc_db';
55         private $DB_USER = 'postgres';
56         private $DB_PASSWORD = 'postgres';
57         private $DB_HOST = 'localhost';
58         private $DB_PORT = 5432;
59
60         // armazena a conexão
61         private $conn;
62
63         public function __construct()
64         {
65             // Quando essa classe é instanciada, é atribuido a variável $conn a conexão com o db
66             $this->conn = new PDO("pgsql:dbname={$this->DB_NAME};host={$this->DB_HOST};port={$this->DB_PORT}");
67         }
68
69         /**
70          * Este método recebe um objeto com a query 'preparada' e atribui as chaves da query
71          * seus respectivos valores.
72          * @param PDOStatement $stmt  Contém a query ja 'preparada'.
73          * @param string        $key   É a mesma chave informada na query.
74          * @param string        $value  Valor de uma determinada chave.
75          */
76         private function setParameters($stmt, $key, $value)
77         {
78             $stmt->bindParam($key, $value);
79         }
80
81         /**
82          * A responsabilidade deste método é apenas percorrer o array de com os parâmetros
83          * obtendo as chaves e os valores para fornecer tais dados para setParameters().
84          * @param PDOStatement $stmt      Contém a query ja 'preparada'.
85          * @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      /**
1  <?php
2
3  namespace Application\models;
4
5  use Application\core\Database;
6  use PDO;
7  class Users
8  {
9      /** Poderiamos ter atributos aqui */
10
11     /**
12      * Este método busca todos os usuários armazenados na base de dados
13      *
14      * @return array
15      */
16     public static function findAll()
17     {
18         $conn = new Database();
19         $result = $conn->executeQuery('SELECT * FROM users');
20         return $result->fetchAll(PDO::FETCH_ASSOC);
21     }
22
23     /**
24      * Este método busca um usuário armazenados na base de dados com um
25      * determinado ID
26      * @param int $id Identificador único do usuário
27      *
28      * @return array
29      */
30     public static function findById(int $id)
31     {
32         $conn = new Database();
33         $result = $conn->executeQuery('SELECT * FROM users WHERE id = :ID LIMIT 1', array(
34             ':ID' => $id
35         ));
36
37         return $result->fetchAll(PDO::FETCH_ASSOC);
38     }
39
```

40 }

Users nhn hosted with ❤ by GitHub

[view raw](#)

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:

```

1 <?php
2
3 use Application\core\Controller;
4
5 class Home extends Controller
6 {
7     /*
8      * chama a view index.php do /home ou somente /
9     */
10    public function index()
11    {
12        $this->view('home/index');
13    }
14}
15 }
```

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 (`Application/views/user/index.php`) todos os usuários no banco de dados em uma variável `$data` que na verdade é um array associativo.

- `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:

```

1  <?php
2
3  use Application\core\Controller;
4
5  class User extends Controller
6  {
7      /**
8      * chama a view index.php da seguinte forma /user/index ou somente /user
9      * e retorna para a view todos os usuários no banco de dados.
10     */
11    public function index()
12    {
13        $Users = $this->model('Users'); // é retornado o model Users()
14        $data = $Users::findAll();
15        $this->view('user/index', ['users' => $data]);
16    }
17
18    /**
19     * chama a view show.php da seguinte forma /user/show passando um parâmetro
20     * via URL /user/show/id e é retornado um array contendo (ou não) um determinado
21     * usuário. Além disso é verificado se foi passado ou não um id pela url, caso
22     * não seja informado, é chamado a view de página não encontrada.
23     * @param int $id Identificado do usuário.
24     */
25    public function show($id = null)
26    {
27        if (is_numeric($id)) {
28            $Users = $this->model('Users');
29            $data = $Users::findById($id);
30            $this->view('user/show', ['user' => $data]);
31        } else {
32            $this->pageNotFound();
33        }
34    }
35
36
37 }
```

User.php hosted with ❤ by GitHub

[view raw](#)

Application/controllers/User.php

Com praticamente 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:

```
1 <main>
2   <div class="container">
3     <div class="row">
4       <div class="col-8 offset-2" style="margin-top:100px">
5
6         <table class="table">
7           <thead>
8             <tr>
9               <th scope="col">ID</th>
10              <th scope="col">Name</th>
11            </tr>
12          </thead>
13          <tbody>
14            <?php foreach ($data['user'] as $user) { ?>
15              <tr>
16                <td><?= $user['id'] ?></td>
17                <td><?= $user['name'] ?></td>
18              </tr>
19            </?php ?>
```

```

1
20      </tbody>
21    </table>
22  </div>
23</div>
24</div>
25</main>
```

show.php hosted with ❤️ by GitHub

[view raw](#)

```

1 <?php
2
3 spl_autoload_register(function ($filename) {
4     $file = '...' . DIRECTORY_SEPARATOR . $filename . '.php';
5     if ( DIRECTORY_SEPARATOR === '/') {
6         $file = str_replace('\\', '/', $file);
7     }
8
9     if ( file_exists($file) ) {
10        require $file;
11    } else {
12        echo 'Erro ao importar o arquivo!';
13    }
14});
```

autoload.php hosted with ❤️ by GitHub

[view raw](#)[Application/autoload.php](#)

Está quasse tudo pronto!

Para finalizar e ser possível 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 funcionar.

Este arquivo é bem simples, contém TAGs html que serão tudo aquilo em comum entre as views, ou seja, o cabeçalho e o rodapé da página, e no centro deste arquivo (que seria a parte do conteúdo de cada view que difere entre elas) inicialmente 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...

```

1 <!DOCTYPE html>
2 <html lang="pt-br">
3   <head>
```

```
4     <meta charset="utf-8">
5     <title>Simple Framework</title>
6     <link rel="stylesheet" href="/assets/css/bootstrap.min.css">
7   </head>
8   <body>
9
10  <?php
11    require '../Application/autoload.php';
12
13    use Application\core\App;
14    use Application\core\Controller;
15
16    $app = new App();
17
18  ?>
19  <script src="/assets/js/jquery.slim.min.js"></script>
20  <script src="/assets/js/bootstrap.min.js"></script>
21  </body>
22 </html>
```

index.php hosted with ❤ by GitHub

[view raw](#)



## Construindo um simples Framework MVC com PHP

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:

ID	Name
1	Maria
2	Antonio
3	Jose

E por último, acessaremos `http://localhost:8080/user/show/<id>` informando um `id` que no caso será o 2 e deve retornar uma view contendo apenas um usuário com aquele determinado `id`.

ID	Name
2	Antonio





• • •

É isso galera! Espero ter ajudado.

Abraços ♡

PHP

Mvc



Follow



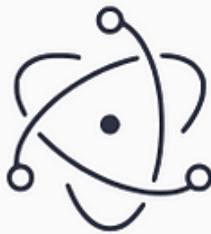
Written by **Jardel Gonçalves**

25 Followers

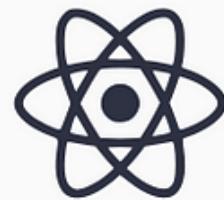
Front-End at loasys

---

## More from Jardel Gonçalves



# ELECTRON



# REACT.js



Jardel Gonçalves

## Construindo um template para criação de aplicações desktop com Electron e React.

O que é o Electron?

4 min read · Jul 10, 2019



14

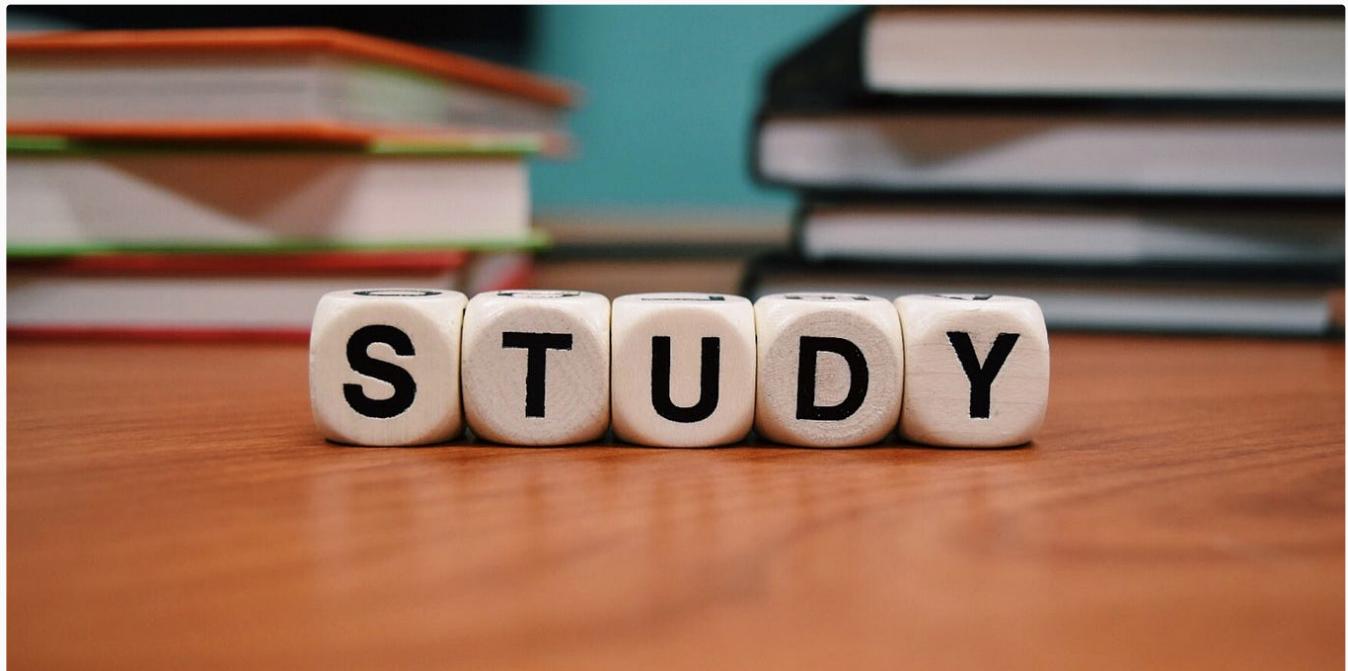


---

See all from Jardel Gonçalves

---

## Recommended from Medium



 .com software

### Simple Trick to Improve Your Classes in PHP

My secret trick how to code better. See how to make your classes better and avoid having too many responsibilities.

★ · 9 min read · Dec 13, 2022

 72

 1





Shahjalal in DevOps.dev

## PHP \_\_callStatic magic method and how it works?

In this article, we will learn about the PHP \_\_callStatic() magic method and how to use it to make our code more flexible.

1 min read · May 15



3

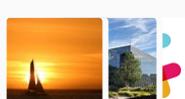


### Lists



#### Staff Picks

317 stories · 81 saves



#### Stories to Help You Level-Up at Work

19 stories · 40 saves



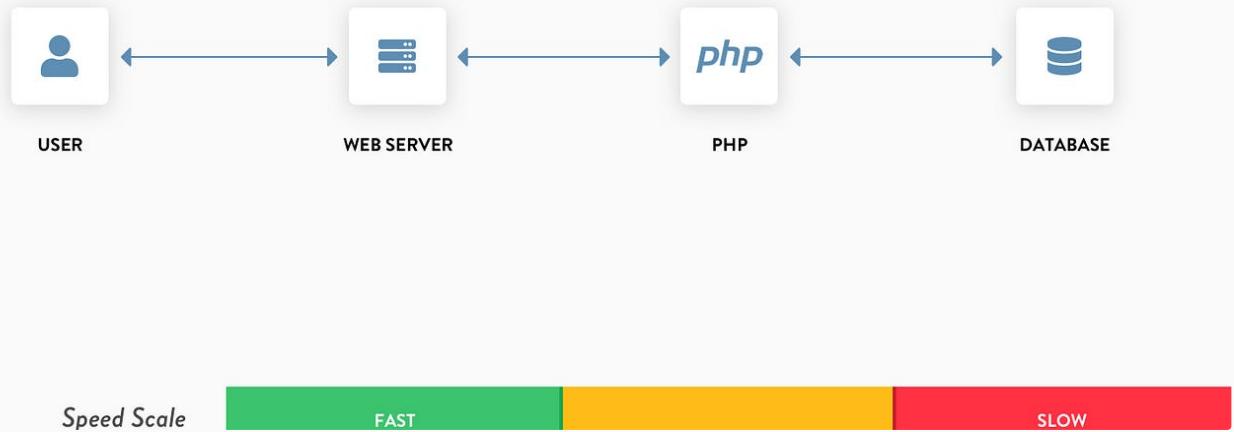
#### Self-Improvement 101

20 stories · 85 saves



#### Productivity 101

20 stories · 86 saves



 Ismat Babirli

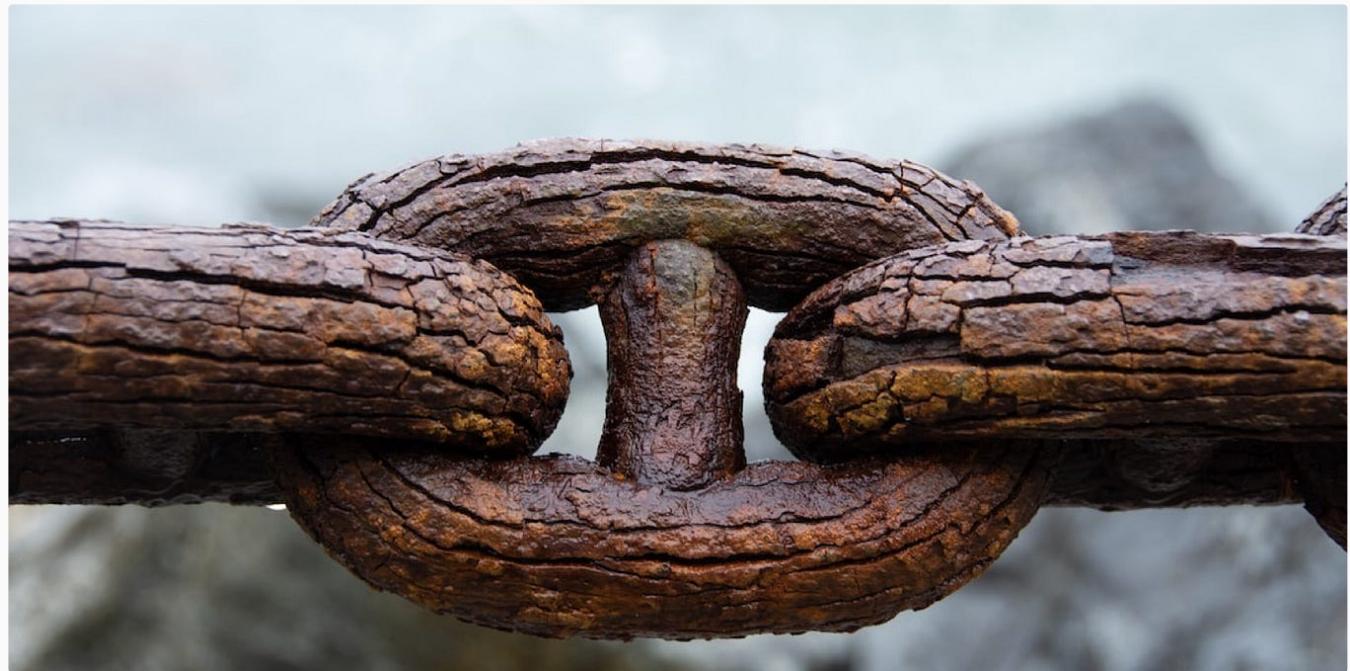
## Laravel Caching: The Key to Lightning-Fast App Performance in 2023

Caching is a powerful method that lets developers make their apps run faster by storing data that is used often in memory.

◆ · 6 min read · Jan 19

 27

 1



 Peter Hrobar

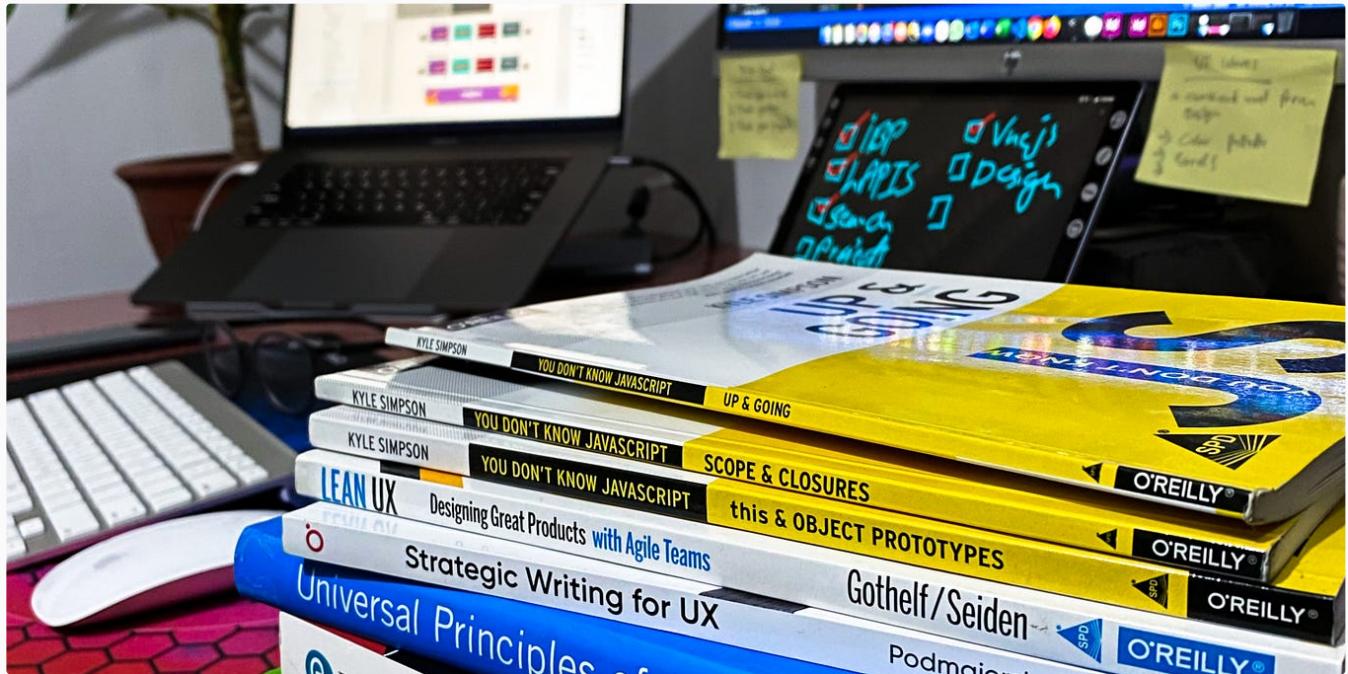
## Laravel event management system

The Laravel event management system provides a flexible and powerful mechanism for implementing event-driven architecture in your...

5 min read · May 16

14

1



 Ismat Babirli

## 2023: Laravel—The Game Changer for Web Development

As a 10-year experienced web developer, I can confidently say that Laravel is set to be the game changer for web development in 2023.

★ · 5 min read · Jan 12

163

15





 .com software

## Stop Caching Relations Inside Your Models in PHP

Database models are the heart of your application. Restrain yourself from caching anything inside!

◆ · 2 min read · Jan 24

 11

 1



See more recommendations