

XAMPP	2
CRUD	4
RESUMO DO TRABALHO COM O LARAVEL:	4
PROJETO Diz Aí	5
PROJETO finanças-app	20
Wireframe do projeto finanças-app	20
ETAPAS PROJETO finanças-app	20
PROJETO LARAVEL UTILIZANDO LIVEWIRE	50
PROCEDIMENTO PARA CLONAR PROJETO LARAVEL DO GITHUB	63
PUBLICAÇÃO NA INTERNET	64
Publicando no Heroku	64

AMBIENTE DE DESENVOLVIMENTO

Necessário a instalação do XAMPP, COMPOSER, VISUAL CODE e LARAVEL, veja abaixo os links necessários para a instalação dessas ferramentas:

- XAMPP: https://www.apachefriends.org/pt_br/index.html
- COMPOSER: <https://getcomposer.org/>
- Documentação: <https://laravel.com/docs/9.x>
- Para instalar o Laravel digite o seguinte comando do composer pelo CMD:
composer global require laravel/installer
- Instruções de como usar o Laravel no XAMPP:
<https://cynoteck.com/pt/blog-post/installing-laravel-8-on-windows-10-xampp/>
- Visual Code com extensões para Laravel

XAMPP

O XAMPP é um servidor apache para instalação local, ele contém o PHP na versão 8 e um banco de dados, ideal para testar um sistema em PHP no seu computador antes da publicação na web. Faça o download do XAMPP em <https://www.apachefriends.org/download.html> Após o download, execute o arquivo de instalação, ao final da instalação abra o painel de controle do XAMPP, clique em “start” para iniciar o serviço.

Os projetos deverão ser salvos na pasta “httdocs” dentro da pasta XAMPP e para acessar no navegador basta acessar a url <http://localhost/pasta/arquivo.php>

Para acessar o phpmyadmin que é o gerenciador do banco de dados local, basta acessar Pelo navegador a seguinte URL <http://localhost/phpmyadmin/> Utilize o banco de dados “test” ou crie uma nova base de dados.

Para mais informações sobre o XAMPP acesse:
https://www.apachefriends.org/faq_windows.html

Sobre MVC

Os 13 frameworks MVC mais usados no mercado
<https://mashable.com/2014/04/04/php-frameworks-build-applications/>
O que é o MVC
<https://pt.wikipedia.org/wiki/MVC>
<https://pt.slideshare.net/leopp/apresentao-mvc-13432782>
<https://www.devmedia.com.br/introducao-ao-padrao-mvc/29308>

FUNDAMENTOS DO LARAVEL

Laravel é o framework PHP livre e open-source mais popular no mundo, utiliza o padrão MVC (model, view, controller). Algumas características proeminentes do Laravel são sua sintaxe simples e concisa, um sistema modular com gerenciador de dependências dedicado, várias formas de acesso a banco de dados relacionais e vários utilitários indispensáveis no auxílio ao desenvolvimento e manutenção de sistemas.

A seguir alguns conceitos que utilizaremos no Laravel:

MIGRATIONS: Esse recurso gerencia as tabelas do banco de dados, é possível criar, editar, fazer backup, restore e excluir as tabelas de um banco de dados. Em uma migration você pode definir os campos e os relacionamentos das tabelas. Após escrever o código das migrations a execução é feita com comandos do php artisan, veja mais detalhes em

<https://laravel.com/docs/8.x/migrations>

ELOQUENT / MODELS: Models dentro da arquitetura MVC, são classes responsáveis pela leitura, escrita e validação dos dados nas tabelas do banco de dados. No Laravel usa-se o Eloquent como uma forma de abstrair as operações de banco de dados, portanto só usamos os models quando for necessário descrever alguma característica dos campos de uma tabela como por exemplo, relacionamentos.

Veja mais detalhes em <https://laravel.com/docs/8.x/eloquent>

CONTROLLERS: De forma resumida, um Controller é onde a lógica e as regras de negócio do sistema são manipuladas. A requisição é feita no Controller recebendo os dados do model, tratando-os e transmitindo-os para a view.

Um Controller é criado com comandos do php artisan.

Veja mais detalhes em <https://laravel.com/docs/8.x/controllers>

VIEWS / BLADE: Dentro do padrão MVC, uma view é onde os dados são exibidos para os usuários, é a parte visual do sistema. Uma View é um documento, que pode ser do tipo texto puro, JSON ou, mais comumente, HTML. No Laravel o mais comum de início é utilizarmos uma engine de template como o **Blade** em nossas views, o que nos permitirá usar HTML/PHP no mesmo arquivo de forma mais organizada. Com os **templates Blade** a exibição de dados dinâmicos dentro de um código HTML fica bem mais fácil.

Veja mais detalhes em <https://laravel.com/docs/8.x/blade>

ROTAS: No laravel, ao receber uma requisição em uma determinada URL o sistema de rotas define o que fazer, como por exemplo redirecionar ou enviar para o controlador (Controller) ou uma view, toda a navegação no sistema é definida nas rotas.

Veja mais detalhes sobre rotas em <https://laravel.com/docs/8.x/routing>

CRUD

CRUD é a composição da primeira letra de 4 funções básicas de um sistema que trabalha com banco de dados:

C: Create (criar) - criar um novo registro

R: Read (ler) - ler (exibir) as informações de um registro

U: Update (atualizar) - atualizar os dados do registro

D: Delete (apagar) - apagar um registro

Essas são as 4 ações para manipular as tabelas do banco de dados de seu sistema.

Do ponto de vista do desenvolvedor, ele precisará criar as tabelas (modelos) do banco de dados, funções (controles) que atualizarão o banco de dados e as interfaces (visões), como página web ou aplicativo mobile, em que os usuários irão interagir com os dados.

RESUMO DO TRABALHO COM O LARAVEL:

Essa é apenas uma sugestão de trabalho, conforme você for desenvolvendo as suas habilidades, você descobrirá qual metodologia é mais apropriada para você.

1. Planejar utilizando preferencialmente wireframes.
2. Iniciar por CMD o projeto e configurar o arquivo .env para estabelecer a conexão do banco de dados.
3. Escrever as migrations necessárias para a criação das tabelas do banco de dados, lembrando que se for utilizar o Jetstream para adicionar login e registro de usuários, não é necessário criar as tabelas de usuários.
4. Gerar os Models para cada tabela.
5. Criação dos controllers conforme definido nas rotas.
6. Criação das views e templates Blade para visualização das páginas.
7. Definir as rotas necessárias.
8. Realizar testes e fazer as correções necessárias.
9. Publicar o projeto.

PROJETO Diz Aí

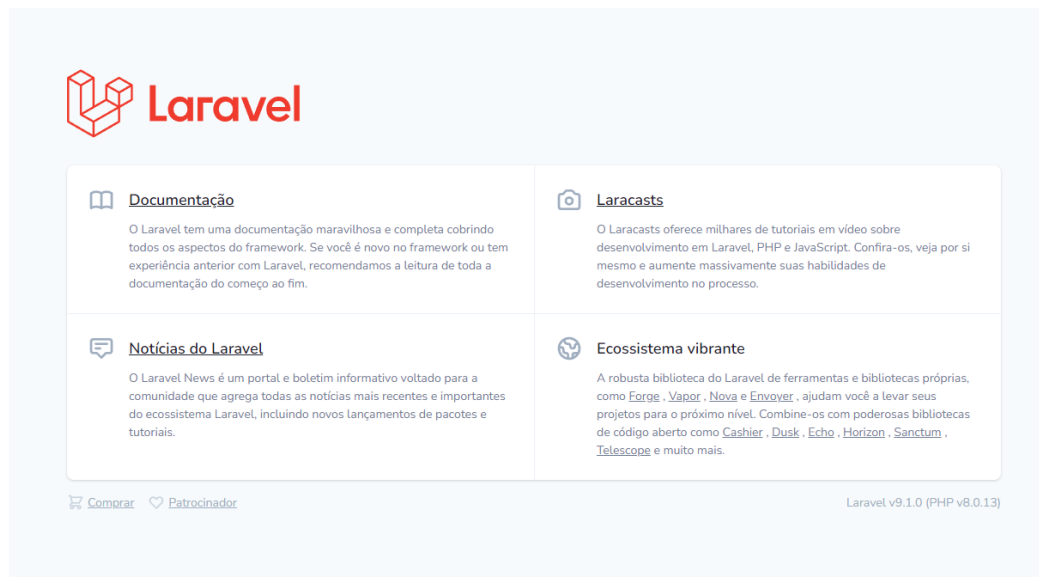
Esse é um projeto simples onde qualquer usuário sem precisar fazer login no site digita uma frase qualquer e escolhe um avatar entre as opções disponíveis, a grosso modo, é como se fosse uma versão anônima do Twitter.

1. Inicie o servidor Xampp e crie um banco de dados com nome “dizai”

2. Abra o CMD na pasta c:/xampp/htdocs digite:
composer create-project laravel/laravel dizai
Ou digite laravel new dizai

cd dizai

3. Dentro da pasta do projeto, no CMD digite:
php artisan serve - este comando inicia o servidor, para acessar o projeto basta abrir o navegador e acessar a URL <http://127.0.0.1:8000> ou <http://localhost:8000>
A partir daí, para visualizar qualquer alteração no projeto, é só acessar esse endereço, e se o navegador já estiver aberto com esta URL, basta atualizar pressionando F5.
A tela inicial do Laravel deverá aparecer.



Esta tela será alterada para o projeto que vamos desenvolver.

4. Abra o Visual Code digitando no CMD
Code .

5. Abra o arquivo .env na raiz do projeto. Esse arquivo contém as configurações do banco de dados do projeto. Encontre os dados referentes ao banco de dados e modifique-os conforme abaixo:

DB_CONNECTION=mysql

```
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=dizai
DB_USERNAME=root
DB_PASSWORD=
```

6. No Visual Code abra o arquivo de rotas routes > web.php e observe que o comportamento para a rota inicial do projeto é abrir o view “welcome”

```
Route::get('/', function () {
    return view('welcome');
});
```

7. Modifique esse carregamento conforme o código abaixo:

```
Route::get('/', function () {
    return view('dizai');
});
```

Esse código carregará a view “dizai” quando o site for acessado. Você precisa criar essa view pois ela não existe.

8. No Visual Code crie um novo arquivo com nome “dizai.blade.php” na pasta resources>views

9. Um arquivo de View é basicamente um HTML com recursos dinâmicos implementados pelo blade. Insira o seguinte código HTML que utiliza Bootstrap 5 e a biblioteca de ícones FontAwesome.:

```
<!doctype html>
<html lang="pt-BR">
  <head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">

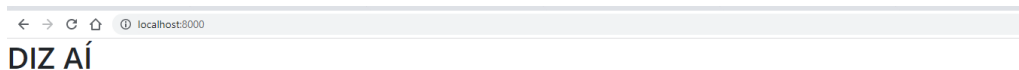
    <!-- Bootstrap CSS e fontawesome-->
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"
rel="stylesheet"
integrity="sha384-EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65V
ohhpuuCOmLASjC" crossorigin="anonymous">
    <link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.0.0/css/all.min.css"
integrity="sha512-9usAa10IRO0HhonpyAIVpjrylPvoDwiPUiKdWk5t3PyoIY1cOd4D
```

```
SE0Ga+ri4AuTroPR5aQvXU9xC6qOPnzFeg==" crossorigin="anonymous"
referrerpolicy="no-referrer" />
```

```
<title>DIZ AÍ</title>
</head>
<body>
  <h1>DIZ AÍ</h1>
```

```
<script
src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.9.2/dist/umd/popper.min.js"
integrity="sha384-IQsoLXI5PILFhosVNubq5LC7Qb9DXgDA9i+tQ8Zj3iwWAwPtgFT
xbJ8NT4GN1R8p" crossorigin="anonymous"></script>
  <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.min.js"
integrity="sha384-cVKIPhGWiC2AI4u+LWgxfKTRlcfu0JTxR+EQDz/bgldoEyl4H0zU
F0QKbrJ0EcQF" crossorigin="anonymous"></script>
</body>
</html>
```

10. Acesse novamente a URL principal, <http://localhost:8000/>
O resultado deverá ser esse:



11. Substitua a tag <h1> pelo código abaixo que irá criar um cabeçalho fixo na página

```
<div class="cabecalho fixed-top p-3">
  <h1 class="text-center m-0"><a class="link-warning" href="#">DIZ
  AÍ</a></h1>
  <p class="text-center text-secondary m-0">Fale o que der na telha mas seja
  educado!</p>
</div>
```

12. Logo acima da tag <title> adicione as regras CSS para criar um estilo ao cabeçalho fixo

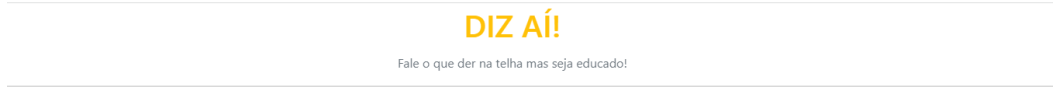
```
<style>
  .cabecalho{
```

```

background-color: #fff;
border-bottom: 1px solid #ddd;
box-shadow: 0px 0px 10px #aaa;
}
.cabecalho a{
text-decoration: none;
}
</style>

```

O resultado deverá ficar semelhante a este:



13. Logo abaixo do fechamento da DIV do cabeçalho, insira a seguinte DIV para os conteúdos:

```

<div class="container mt-5 pt-5">
  <div class="col-md-8 offset-md-2 mt-4">
    Insira o Conteúdo AQUI!!
  </div>
</div>

```

14. No lugar do texto “Insira o Conteúdo Aqui!!” vamos inserir 2 componentes CARDS para servir de modelo para o que os usuários escreveram. Substitua o texto pelos códigos abaixo:

```

<div class="card mb-3">
  <div class="row g-0">
    <div class="col-md-2 text-center">
      <span class="avatar text-warning"><i class="far
fa-grin-tongue"></i></span>
    </div>
    <div class="col-md-8">
      <div class="card-body">
        <h5 class="card-title">Codinome do usuário</h5>
        <p class="card-text">Um texto qualquer escrito no calor do
momento.</p>
        <p class="card-text"><small class="text-muted">Publicado em
99/99/9999 15:02</small></p>
      </div>
    </div>
  </div>
</div>

```



```

        </div>
    </div>
</div>
</div>

<div class="card mb-3">
    <div class="row g-0">
        <div class="col-md-2 text-center">
            <span class="avatar text-warning"><i class="far
fa-angry"></i></span>
        </div>
        <div class="col-md-8">
            <div class="card-body">
                <h5 class="card-title">Codinome do usuário</h5>
                <p class="card-text">Um texto qualquer escrito no calor do
momento.</p>
                <p class="card-text"><small class="text-muted">Publicado em
99/99/9999 15:02</small></p>
            </div>
        </div>
    </div>
</div>
</div>

```

Esse código vai renderizar dois cards, mais adiante você irá inserir um código para trazer esse conteúdo do banco de dados, tornando a aplicação dinâmica. O resultado deverá ficar semelhante a esse:



15. A próxima etapa é incluir um botão flutuante para abrir um modal com o formulário para inserção de novas frases. Adicione este código HTML para o botão antes da tag `<script>`:

```

<button type="button" class="botao-flutuante bg-warning"
data-bs-toggle="modal" data-bs-target="#modalFormulario">

```

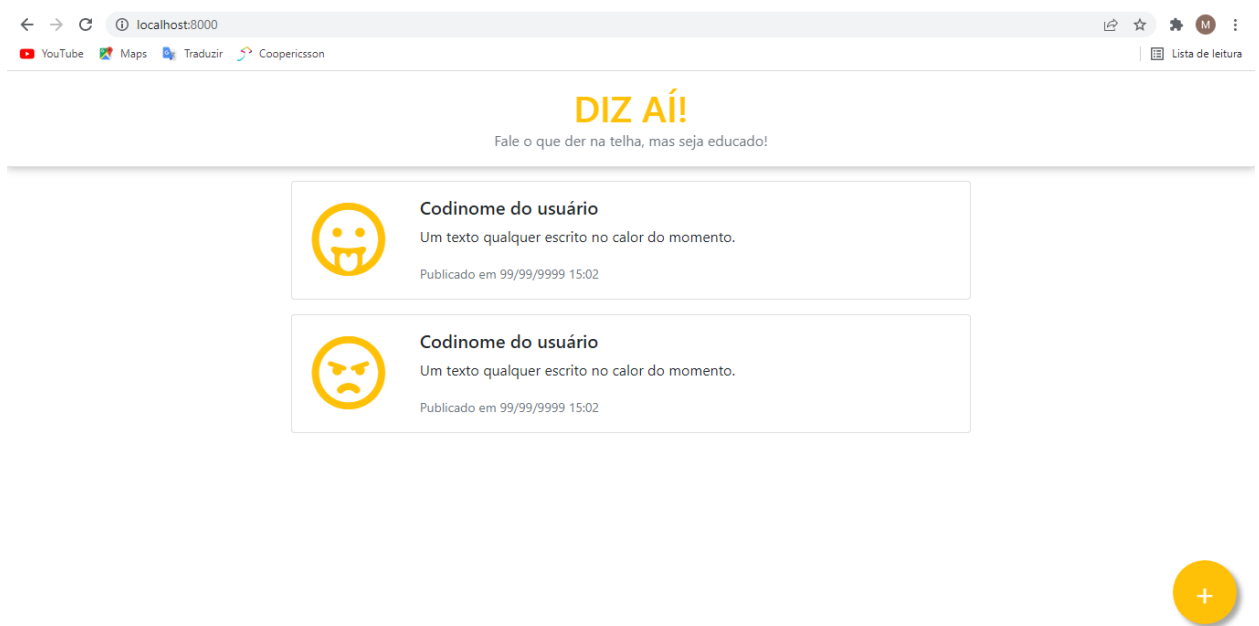
+

</button>

16. No código CSS adicione a regra abaixo para estilizar o botão:

```
.botao-flutuante{  
    display:block;  
    position: fixed;  
    bottom: 20px;  
    right: 20px;  
    width: 70px;  
    height: 70px;  
    border: none;  
    font-size: 35px;  
    color: #fff;  
    border-radius: 50%;  
    box-shadow: 5px 5px 5px #aaa;  
}
```

O resultado deverá ficar semelhante a esse:



17. Logo abaixo do botão, escreva o código do modal com o formulário para inserção do comentário:

```
<div class="modal fade" id="modalFormulario" tabindex="-1"  
aria-labelledby="modalFormularioLabel" aria-hidden="true">  
  <div class="modal-dialog">  
    <div class="modal-content">  
      <div class="modal-header">  
        <h5 class="modal-title" id="modalFormularioLabel">DIZ AÍ!!</h5>
```

```

        <button type="button" class="btn-close" data-bs-dismiss="modal"
aria-label="Close"></button>
    </div>
    <div class="modal-body">
        <form action="#">
            <input class="form-control" type="text" placeholder="Codinome"
name="codinome" required>
            <br>
            <p>Seu estado de espírito</p>
            <div class="form-check form-check-inline">
                <input class="form-check-input" type="radio" name="espírito"
value="far fa-grin-tongue" required>
                <label class="form-check-label"><i class="far fa-grin-tongue
fa-2x"></i></label>
            </div>
            <div class="form-check form-check-inline">
                <input class="form-check-input" type="radio" name="espírito"
value="far fa-grin-beam" required>
                <label class="form-check-label"><i class="far fa-grin-beam
fa-2x"></i></label>
            </div>
            <div class="form-check form-check-inline">
                <input class="form-check-input" type="radio" name="espírito"
value="far fa-sad-tear" required>
                <label class="form-check-label"><i class="far fa-sad-tear
fa-2x"></i></label>
            </div>
            <div class="form-check form-check-inline">
                <input class="form-check-input" type="radio" name="espírito"
value="far fa-surprise" required>
                <label class="form-check-label"><i class="far fa-surprise
fa-2x"></i></label>
            </div>
            <div class="form-check form-check-inline">
                <input class="form-check-input" type="radio" name="espírito"
value="far fa-angry" required>
                <label class="form-check-label"><i class="far fa-angry
fa-2x"></i></label>
            </div>

            <br><br>
            <label for="comentario" class="form-label">Diz Ai!!</label>

```

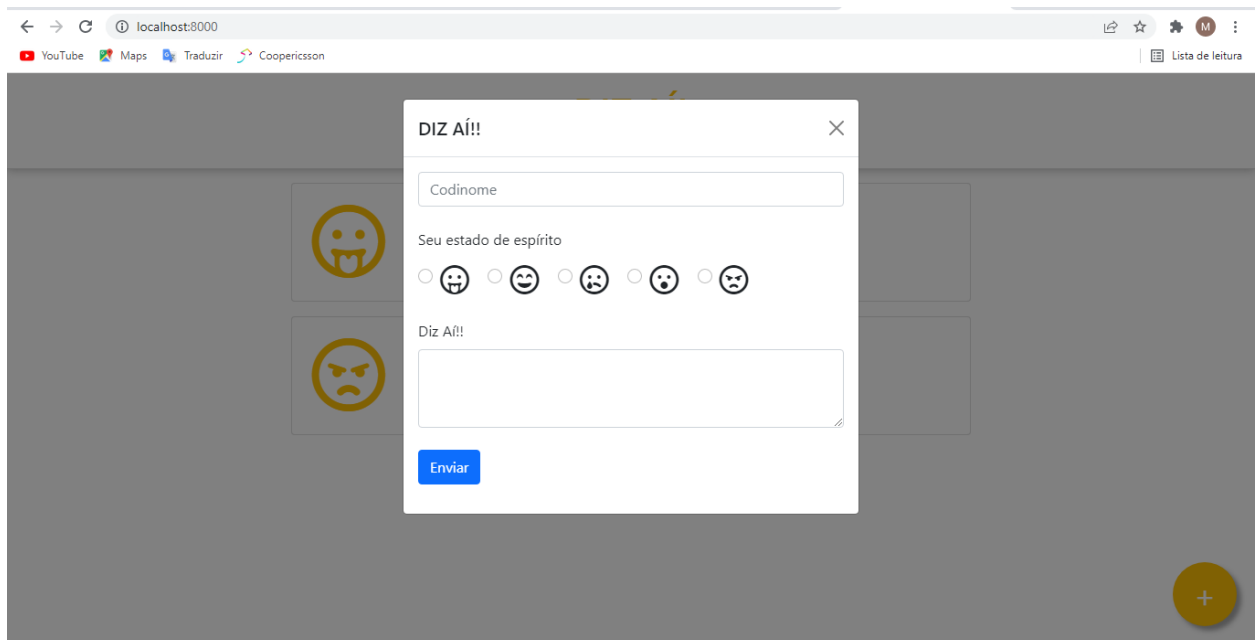
```

        <textarea class="form-control" id="comentario" name="comentario"
rows="3"></textarea>

        <br>
        <button type="submit" class="btn btn-primary
mb-3">Enviar</button>
    </form>
</div>
</div>
</div>
</div>

```

Experimente clicar no botão flutuante, o modal aberto com o formulário deverá ficar parecido com isso:



18. Criando a tabela no banco de dados. Todo o front-end está pronto, mas é uma página estática, não armazena e não carrega nada do banco de dados. Você vai criar e executar uma migration para criar a tabela **comentarios** com os mesmos campos existentes no formulário.

No CMD ou no terminal do Visual Code na pasta do projeto digite:

php artisan make:migration create_comentarios_table

Esse comando irá criar o arquivo create_comentarios_table.php na pasta database>migrations.

19. Abra o arquivo database>migrations>create_comentarios_table.php e modifique o método up() conforme o exemplo a seguir:

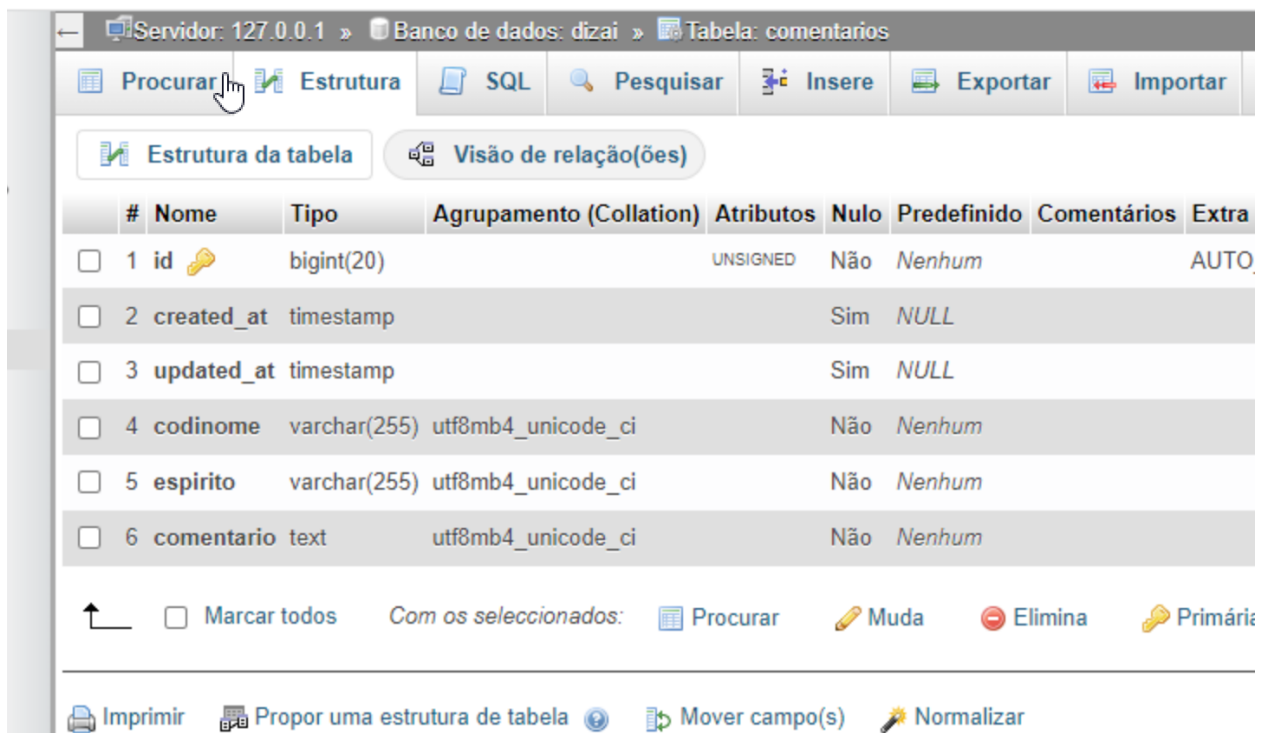
```
public function up()
{
    Schema::create('comentarios', function (Blueprint $table) {
        $table->id();
        $table->timestamps();
        $table->string('codinome');
        $table->string('espirito');
        $table->text('comentario');
    });
}
```

O método up() é executado quando a migration é iniciada e perceba que o método cria os campos na tabela.

20. No CMD ou terminal do Visual Code digite:

php artisan migrate

Esse comando vai executar as migrates pendentes e criar as tabelas que foram solicitadas.



#	Nome	Tipo	Agrupamento (Collation)	Atributos	Nulo	Predefinido	Comentários	Extra
<input type="checkbox"/>	1 id	bigint(20)		UNSIGNED	Não	Nenhum		AUTO
<input type="checkbox"/>	2 created_at	timestamp			Sim	NULL		
<input type="checkbox"/>	3 updated_at	timestamp			Sim	NULL		
<input type="checkbox"/>	4 codinome	varchar(255)	utf8mb4_unicode_ci		Não	Nenhum		
<input type="checkbox"/>	5 espirito	varchar(255)	utf8mb4_unicode_ci		Não	Nenhum		
<input type="checkbox"/>	6 comentario	text	utf8mb4_unicode_ci		Não	Nenhum		

Os campos criados para a tabela comentarios.

IMPORTANTE: Ao criar relacionamentos entre tabelas, verifique se a ordem de criação das tabelas está correta. Por exemplo, ao relacionar a tabela “post” com a tabela “user” a tabela “user” deve ser executada primeiro. Verifique o nome do arquivo de migration, se for necessário, renomeie o arquivo para garantir a ordem

correta.

21. Cada tabela deverá ter um Model para que exista a conexão entre o Controller (que usaremos mais adiante) com o banco de dados. É possível criar um controller e o model ao mesmo tempo, com os códigos necessários para fazer a conexão entre ambos.

No CMD ou terminal do Visual Code digite:

```
php artisan make:controller ComentarioController --resource  
--model=Comentario
```

Observe que o Model para a tabela comentarios é criada na pasta app>models e o controller na pasta app>Http>Controllers.

Por enquanto não acrescenta nenhum código no arquivo model, isso só será necessário se você precisar criar um relacionamento com outra tabela, proteger campos específicos ou acrescentar alguma característica especial para algum campo da tabela, essa necessidade depende muito das características do projeto.

22. Abra o arquivo app>Http>Controllers>ComentarioController.php

Note que o controller é criado importando o Model por esta instrução:

```
use app\Models\Comentario;
```

Também foram criados os métodos index(), create(), store(), show(), edit(), update() e destroy(). Essas funções não são obrigatórias, Laravel apenas adiantou o processo se o seu projeto necessitar. No projeto “Dizai” não vamos utilizar todas essas funções.

23. EXIBINDO OS COMENTÁRIOS SALVOS NO BANCO DE DADOS. Dentro do controller ComentarioController insira no método index() o código que vai consultar todos os comentários salvos. Digite o código abaixo dentro do método index():

```
public function index()  
{  
    $comentarios = Comentario::orderBy('created_at', 'desc')->get();  
    return view('dizai',['comentarios'=>$comentarios]);  
}
```

Explicando o código: **\$comentarios = Comentario::orderBy('created_at', 'desc')->get();** carrega na variável \$comentarios todos os registros da tabela comentarios que é definida no Model Comentario em ordem decrescente. **return view('dizai',['comentarios'=>\$comentarios]);** Carrega a view dizai e envia os dados na variável \$comentarios.

24. Abra o arquivo da view “dizai” para tratar os dados enviados pela variável “comentarios”.

25. Você vai inserir uma diretiva Blade para verificar se tem algum comentário para exibir, se não tiver, um parágrafo com uma mensagem que não há informações para exibir é mostrado, caso contrário os cards com os comentários são exibidos. Modifique a área de comentários para ficar como o código a seguir:

```
@if (count($comentarios) === 0)
```

```

        <p class="text-center">Não há comentários para exibir :-( </p>
    @else
        <div class="card">
            <div class="card-body">
                <div class="row">
                    <div class="col-md-2">
                        Avatar
                    </div>
                    <div class="col-md-8">
                        <h5 class="card-title">Codinome do usuário</h5>
                        <p class="card-text">
                            Um texto qualquer no calor do momento.
                        </p>
                        <p class="text-secondary">
                            Publicado em 99/99/9999
                        </p>
                    </div>
                </div>
            </div>
        </div>
    @endif

```

26. Agora que boa parte do FRONT-END está pronto, é hora de fazer o formulário gravar os comentários dos usuários. Para isso será necessário um método no controller para gravar os dados e definir a sua respectiva rota, o formulário irá submeter os dados para esse controller. Abra o arquivo `app>Http>Controllers>ComentarioController.php` e encontre o método `store()`. Esse método deverá ficar assim:

```

public function store(Request $request)
{
    $comentario = new Comentario;

    $comentario->codinome = $request->codinome;
    $comentario->espírito = $request->espírito;
    $comentario->comentario = $request->comentario;

    $comentario->save();
    return redirect('/');
}

```

Explicando o código:

store(Request \$request) essa função recebe um objeto do tipo Request, basicamente são os campos do formulário.

\$comentario = new Comentario; cria uma instância do Model Comentario, significa

que a variável \$comentario trata dos campos na tabela e a variável \$request trata os campos do formulário.

```
$comentario->codinome = $request->codinome;  
$comentario->espírito = $request->espírito;  
$comentario->comentario = $request->comentario;
```

Essas linhas vão atribuir o valor do campo na tabela do banco de dados ao valor no respectivo campo do formulário (\$comentario é igual a \$request).

```
$comentario->save();  
return redirect('/');
```

E finalmente salva os dados na tabela do banco de dados e recarrega a rota principal.

27. Esse código não é suficiente para fazer o formulário funcionar, é necessário criar uma rota e associar o action do formulário a essa rota. Abra o arquivo routes>web.php e adicione a seguinte rota:

```
Route::post('/novocomentario', [ComentarioController::class,  
'store'])->name('novocomentario');
```

Observe que essa rota é do tipo “post” que é o tipo recomendado para inserção de dados no banco de dados. O nome da rota é “novocomentario”, será necessário inserir essa informação no formulário.

28. Abra o arquivo resources>view>dizai.blade.php e localize a tag **<form action="#">**

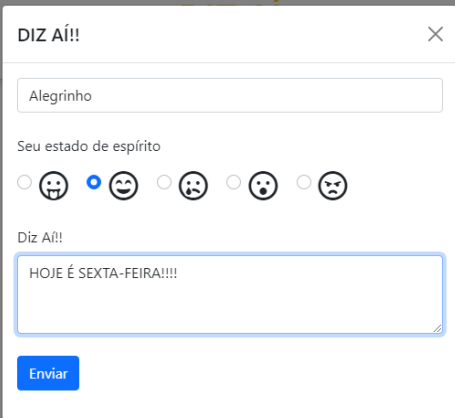
29. Modifique a tag para o que é mostrado abaixo:

```
<form method="POST" action="{{ route('novocomentario') }}">  
    @csrf
```

Veja que foi adicionado o atributo method com valor de post, também foi acrescentada a rota “novocomentario” como valor da action e por último um detalhe importante, a diretiva @csrf, essa diretiva adiciona ao formulário, um recurso de segurança para evitar ataques do tipo CSRF. Se você inspecionar o código o formulário fica assim:

```
<form method="POST" action="http://localhost:8000/novocomentario">  
    <input type="hidden" name="_token" value="uma-chave-qualquer">
```

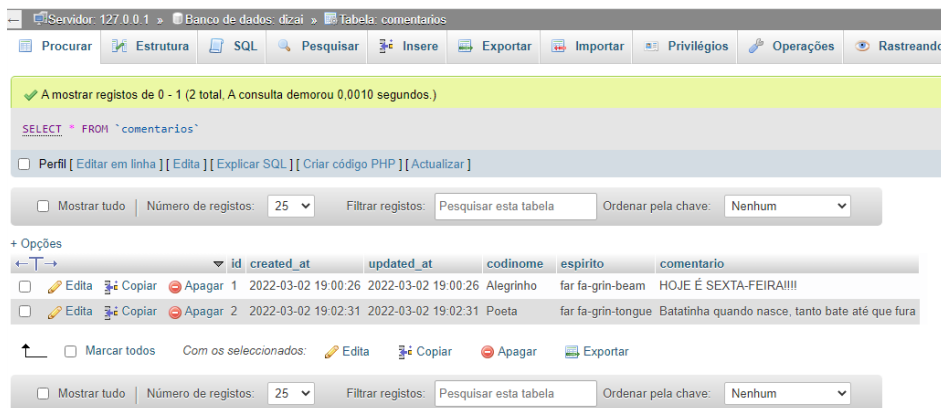

30. Faça um teste, digite alguma mensagem e veja o que acontece.



The modal dialog titled "DIZ AÍ!!" contains the following elements:

- A text input field with the placeholder text "Alegrinho".
- A section titled "Seu estado de espírito" with five radio button options, each accompanied by a different smiley face icon.
- A text input field with the placeholder text "Diz Aí!!".
- A blue button labeled "Enviar".

Observe que o que é mostrado ao final não corresponde ao que foi digitado, e aparece apenas uma caixa com modelo de comentário.



The screenshot shows a database management interface with the following components:

- Top navigation bar: Servidor: 127.0.0.1 » Banco de dados: dizai » Tabela: comentarios
- Toolbar: Procurar, Estrutura, SQL, Pesquisar, Inserir, Exportar, Importar, Privilegios, Operacoes, Rastreador.
- Status bar: A mostrar registros de 0 - 1 (2 total, A consulta demorou 0,0010 segundos.)
- SQL Editor: `SELECT * FROM `comentarios``
- Actions: Perfil, [Editar em linha], [Edita], [Explicar SQL], [Criar código PHP], [Atualizar]
- Filters: Mostrar tudo, Número de registros: 25, Filtrar registros: Pesquisar esta tabela, Ordenar pela chave: Nenhum
- Table with 6 columns: id, created_at, updated_at, codinome, espirito, comentario
- Table content:

	id	created_at	updated_at	codinome	espirito	comentario
<input type="checkbox"/>	1	2022-03-02 19:00:26	2022-03-02 19:00:26	Alegrinho	far fa-grin-beam	HOJE É SEXTA-FEIRA!!!!
<input type="checkbox"/>	2	2022-03-02 19:02:31	2022-03-02 19:02:31	Poeta	far fa-grin-tongue	Batatinha quando nasce, tanto bate até que fura

Bottom section: + Opções, ← T →, Marcar todos, Com os seleccionados: [Edita], [Copiar], [Apagar], [Exportar]

Apesar de existir dois comentários no banco de dados, a página só mostra o modelo.

Avatar

Codinome do usuário

Um texto qualquer no calor do momento.

Publicado em 99/99/9999



31. É necessário modificar a view para que seja mostrado todos os comentários. Abra o arquivo `resources>view>dizai.blade.php` modifique a estrutura **@else @endif** para que repita vários componentes cards. Após a diretiva **@else**, insira o componente card dentro de uma diretiva **@foreach** como a que está exibida abaixo:

```
@foreach ($comentarios as $comentario)
    <div class="card">
        ...
    </div>
@endforeach
```

32. Na área com o código CSS, dentro da estrutura `<style></style>` inclua a regra para o ícone de avatar:

```
.avatar {
    font-size: 80px;
}
```

33. Modifique o código de toda a estrutura **@foreach @endforeach** conforme o código abaixo:

```
@foreach ($comentarios as $comentario)
    <div class="card mb-4">
        <div class="card-body">
            <div class="row">
                <div class="col-md-2">
                    <span class="avatar text-warning"><i
class="{{ $comentario->espírito }}"></i></span>
```

```

</div>
<div class="col-md-8">
  <h5 class="card-title">{{$comentario->codinome}}</h5>
  <p class="card-text">
    {{$comentario->comentario}}
  </p>
  <p class="text-secondary">
    Publicado em {{
Carbon\Carbon::parse($comentario->created_at)->format('d/m/Y H:i:s') }}
  </p>
</div>
</div>
</div>
</div>
@endforeach

```

O laço @foreach vai repetir o seu código em quantos registros existirem na tabela, **\$comentarios as \$comentario** vai representar cada registro na variável \$comentario. Depois é só pedir o campo desta forma \$comentario->nome-do-campo-na-tabela. A exibição de datas usa o Carbon para facilitar como nesse exemplo:
Carbon\Carbon::parse(\$comentario->created_at)->format('d/m/Y')

34. Faça os últimos testes e ajuste o layout se for necessário.

Esse pequeno projeto está concluído, serviu apenas para trabalhar os recursos básicos de inserção e consulta a banco de dados. O próximo projeto vai trabalhar todos os aspectos de um CRUD.

PROJETO fincas-app


Wireframe do projeto fincas-app

LOGIN

Entrar

CADASTRO NOVO USUÁRIO

Cadastrar



Olá fulano

Seus dados


Extrato


Nova entrada

EXTRATO - SALDO R\$ 9.999,99

Receitas


Total - R\$ 9.999,00


Movimento 


Movimento 


Despesas


Total - R\$ 9.999,00


Movimento 


Movimento 

Movimento 

Movimento 

Movimento 

Movimento 



Olá fulano

Seus dados

Extrato

Nova entrada

NOVA ENTRADA

Enviar

Este projeto vai permitir que os usuários façam um controle financeiro diário, onde poderá ser cadastrada as receitas e despesas do usuário, ele também poderá controlar o seu saldo. Esse projeto vai permitir o uso de operações CRUD pois o usuário poderá Criar, Ler, Atualizar e Deletar os movimentos financeiros de qualquer movimento.

ETAPAS PROJETO fincas-app

1. Abra o servidor Xampp e inicie os serviços, clique nos botões "start".
2. Acesse o servidor e crie um banco de dados para o projeto com o nome "fincas"
3. Dentro da pasta c:/xampp/htdocs pelo CMD digite:

```
composer create-project laravel/laravel fincas-app
ou
laravel new fincas-app

cd fincas-app
```

20

php artisan serve - este comando inicia o servidor, para acessar o projeto basta abrir o navegador e acessar a URL <http://127.0.0.1:8000> ou <http://localhost:8000>
A partir daí, para visualizar qualquer alteração no projeto, é só acessar esse endereço, e se o navegador já estiver aberto com esta URL, basta atualizar pressionando F5.

4. Abra o arquivo **.env** na raiz do seu projeto e informe os dados do banco de dados assim (um banco de dados local não possui senha):

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=financas
DB_USERNAME=root
DB_PASSWORD=
```

5. **Criação das tabelas do banco de dados** - O Laravel tem um recurso para criar e fazer backup de tabelas chamado **MIGRATIONS**. É semelhante a um versionador, porém é voltado para o banco de dados do projeto. As operações CRUD podem ser feitas diretamente no Laravel sem que seja necessário o uso de comandos SQL, o Laravel usa o **Eloquent** que é uma abstração do banco de dados, permitindo um uso mais fácil. Mais adiante você verá como trabalhar com esses recursos.

Os migrations ficam localizados na pasta database>migrations.

Por enquanto você apenas irá criar as tabelas para o projeto com a seguinte estrutura:

- a. TABELA **fin_movimentos** | campos: id (chave primária), descricao, valor, tipo, data

6. No CMD ou no terminal do Visual Code na pasta do projeto digite:

php artisan make:migration create_fin_movimentos_table

Esse comando irá criar o arquivo create_fin_movimentos_table.php na pasta database>migrations.

7. Abra o arquivo create_fin_movimentos_table.php na pasta database>migrations.

Observe que esse arquivo possui os métodos up() e down().

O método up() é executado quando a tabela é criada e o método down() é executado quando for solicitado que a migration retorne um passo, é como se fosse o comando desfazer em um aplicativo qualquer. O método up() possui apenas os campos id que é a chave primária da tabela e o campo timestamp que o Laravel utiliza quando um dado é adicionado na tabela.

8. Modifique o código do método up() como exemplificado:

```
public function up()
{
    Schema::create('fin_movimentos', function (Blueprint $table) {
        $table->id();
```

```

        $table->timestamps();
        $table->float('valor', 12, 2);
        $table->text('descricao');
        $table->string('tipo');
        $table->date('data');
    });
}

```

Este código vai criar a tabela `fin_movimentos` com os campos `id` (chave primária), um campo do tipo `timestamps` para registrar a hora de qualquer inserção de registro (padrão do Laravel), o campo `valor` com 12 dígitos e 2 casas decimais, o campo `descricao` para o texto descritivo do movimento (texto longo), o campo `tipo` para informar se é despesa ou receita (texto curto) e o campo para a data do movimento.

IMPORTANTE: É ALTAMENTE RECOMENDÁVEL ESCREVER UMA NOVA MIGRATION PARA ACRESCENTAR UM NOVO CAMPO PARA A TABELA.

- No CMD ou terminal do Visual Code digite:

php artisan migrate

Esse comando vai executar as migrates pendentes e criar as tabelas que foram solicitadas.



A tabela `fin_movimentos` criada.

Mais adiante nesse projeto você irá adicionar dados nessa tabela.

Agora o próximo passo é criar toda a rotina de login e registro de usuários.

- ADICIONANDO UM MODEL PARA A TABELA CRIADA.** Cada tabela deverá ter um Model para que exista a conexão entre o Controller (que usaremos mais adiante) com o banco de dados. Enquanto a convenção é criar as tabelas do banco de dados no plural, a criação do Model deve ser feita no singular. No CMD ou terminal do Visual Code digite: **php artisan make:model Fin_movimento**

Observe que o Model para a tabela fin_movimentos é criada na pasta app>models.
NÃO ESQUEÇA DE SEMPRE CRIAR UM MODEL PARA CADA TABELA DO BANCO DE DADOS, EXCETO A TABELA DE USUÁRIOS QUE O LARAVEL JÁ CRIA COMO PADRÃO.

11. **Tela de Login/senha - criação de autenticação com o Laravel** - O Laravel utiliza o Jetstream e o Livewire para a criação de todo o processo de autenticação de forma automatizada o que aumenta e muito a produtividade no desenvolvimento do projeto.
Instalação do JETSTREAM: Com o CMD aberto dentro da pasta do projeto digite:
composer require laravel/jetstream

12. **Tela de Login/senha - Instalação do livewire** Com o CMD aberto dentro da pasta do projeto digite:

php artisan jetstream:install livewire

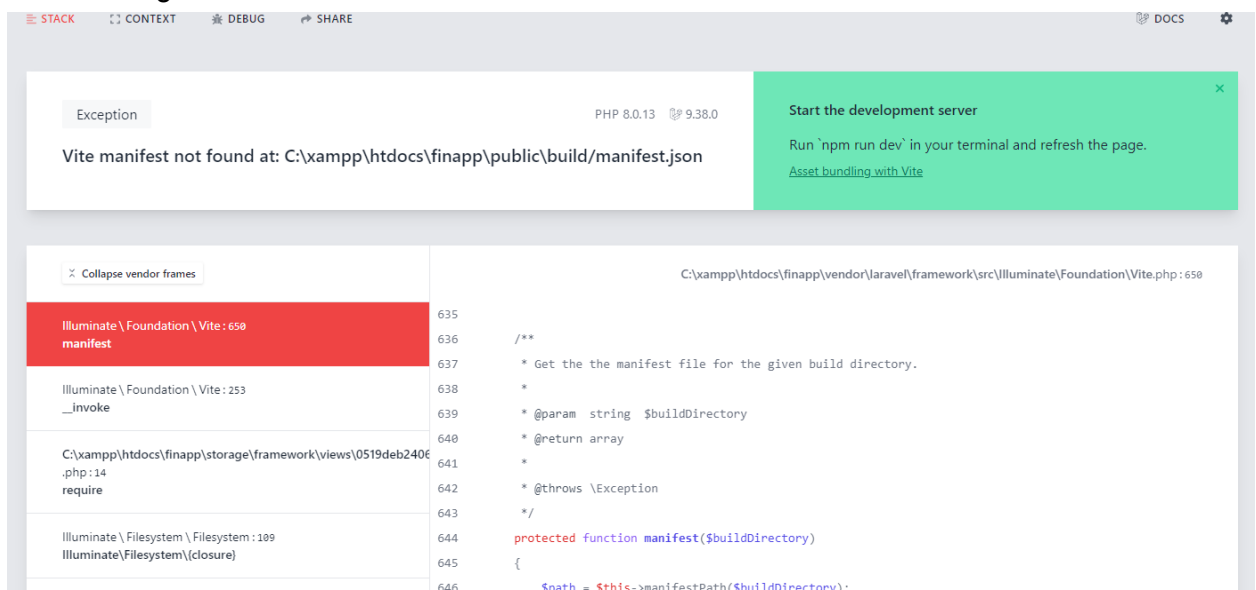
Após a instalação é necessário rodar dois comandos do npm, se você não tem o npm instalado, acesse o site do node para instalar o node e o npm <https://nodejs.org/en/>
Após a instalação do node, digite esses dois comandos no CMD ou terminal do Visual Code dentro da pasta do projeto:

npm install

npm run dev

Todos esses comandos vão instalar arquivos de MVC e tabelas do banco de dados para que todo o processo de login funcione no seu projeto.

ATENÇÃO! Se ao acessar a página de login ou registro de novos usuários aparecer uma mensagem de erro como essa:



Isso indica que houve um erro na instalação do vite. Vite é uma ferramenta de build que surgiu para substituir o construtor webpack. Digite no terminal os seguintes comandos

para resolver o problema:

```
npm remove laravel-mix  
composer require "innocenzi/laravel-vite:0.2.*"  
npm i -D vite vite-plugin-laravel  
php artisan vendor:publish --tag=vite-config
```

Para mais detalhes consulte:

<https://laravel-vite.dev/guide/quick-start.html#initial-setup>

Após a instalação desses recursos execute novamente:

```
npm install  
npm run dev
```

13. Abra a pasta do projeto no Visual Code e abra o arquivo de Migration

database/migrations/2014_10_12_000000_create_users_table.php

Este arquivo contém a descrição da tabela de usuários que será criada.

Observe que o método up() cria a tabela com os campos, id, name, email e vários outros.

```
public function up()  
{  
    Schema::create('users', function (Blueprint $table) {  
        $table->id();  
        $table->string('name');  
        $table->string('email')->unique();  
        $table->timestamp('email_verified_at')->nullable();  
        $table->string('password');  
        $table->rememberToken();  
        $table->foreignId('current_team_id')->nullable();  
        $table->string('profile_photo_path', 2048)->nullable();  
        $table->timestamps();  
    });  
}
```

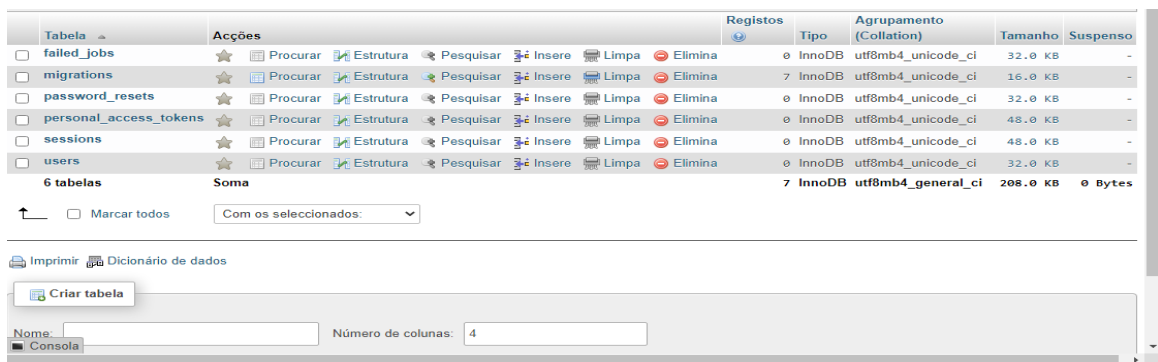
O método up() que cria a tabela com os campos, é executado quando o Migrations é rodado pelo CMD, e o método down() será rodado quando a reversão do Migrations for solicitada.

```
public function down()  
{  
    Schema::dropIfExists('users');  
}
```


14. No CMD ou terminal do Visual Code digite:

php artisan migrate

Esse comando irá gerar as tabelas descritas no arquivo de migração.

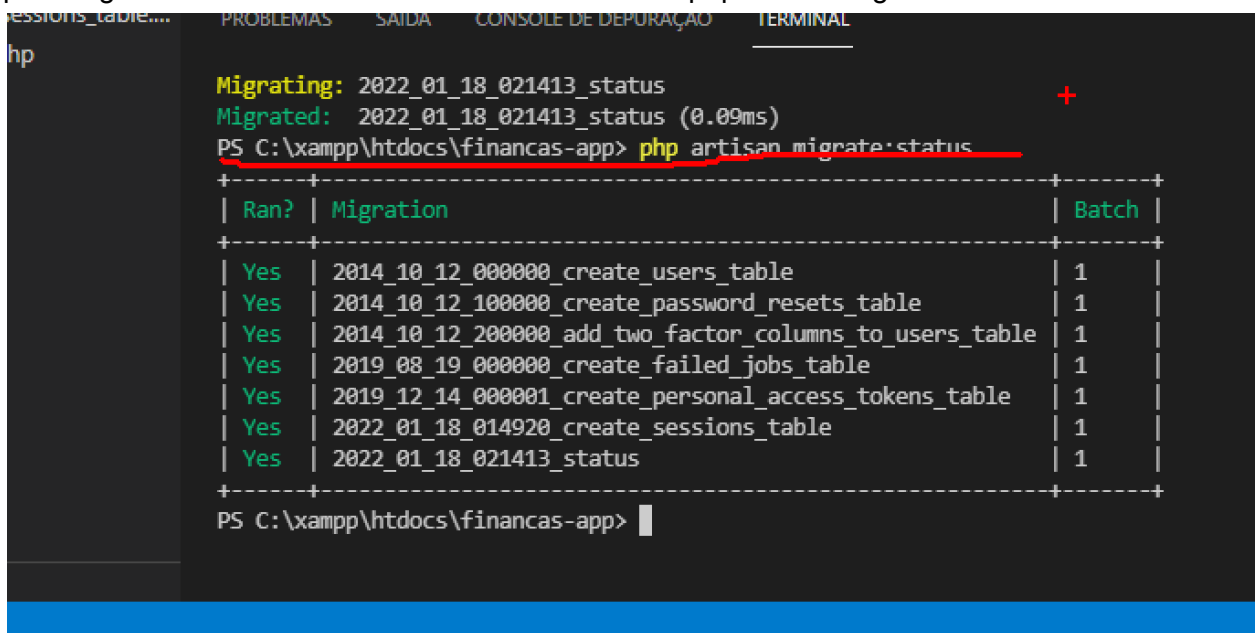


The screenshot shows the phpMyAdmin interface with a table list. The table 'failed_jobs' is selected. Below the table list, there is a section for creating a new table with fields for 'Nome' and 'Número de colunas'.

Tabela	Registos	Tipo	Agrupamento (Collation)	Tamanho	Suspensão
failed_jobs	0	InnoDB	utf8mb4_unicode_ci	32.0 KB	-
migrations	7	InnoDB	utf8mb4_unicode_ci	16.0 KB	-
password_resets	0	InnoDB	utf8mb4_unicode_ci	32.0 KB	-
personal_access_tokens	0	InnoDB	utf8mb4_unicode_ci	48.0 KB	-
sessions	0	InnoDB	utf8mb4_unicode_ci	48.0 KB	-
users	0	InnoDB	utf8mb4_unicode_ci	32.0 KB	-
6 tabelas	7	InnoDB	utf8mb4_general_ci	208.0 KB	0 Bytes

As tabelas criadas no banco de dados.

DICA: Para saber se todas as tabelas foram devidamente criadas no seu projeto você pode digitar no terminal do Visual Code ou no CMD `php artisan migrate:status`



The screenshot shows a terminal window with the command `php artisan migrate:status` executed. The output displays the status of each migration, indicating that all migrations were successful.

```
Migrating: 2022_01_18_021413_status
Migrated: 2022_01_18_021413_status (0.09ms)
PS C:\xampp\htdocs\financas-app> php artisan migrate:status

+-----+-----+-----+
| Ran? | Migration                                          | Batch |
+-----+-----+-----+
| Yes  | 2014_10_12_000000_create_users_table              | 1     |
| Yes  | 2014_10_12_100000_create_password_resets_table    | 1     |
| Yes  | 2014_10_12_200000_add_two_factor_columns_to_users_table | 1     |
| Yes  | 2019_08_19_000000_create_failed_jobs_table        | 1     |
| Yes  | 2019_12_14_000001_create_personal_access_tokens_table | 1     |
| Yes  | 2022_01_18_014920_create_sessions_table           | 1     |
| Yes  | 2022_01_18_021413_status                          | 1     |
+-----+-----+-----+
```

O status indicando que todas as tabelas do projeto foram criadas.

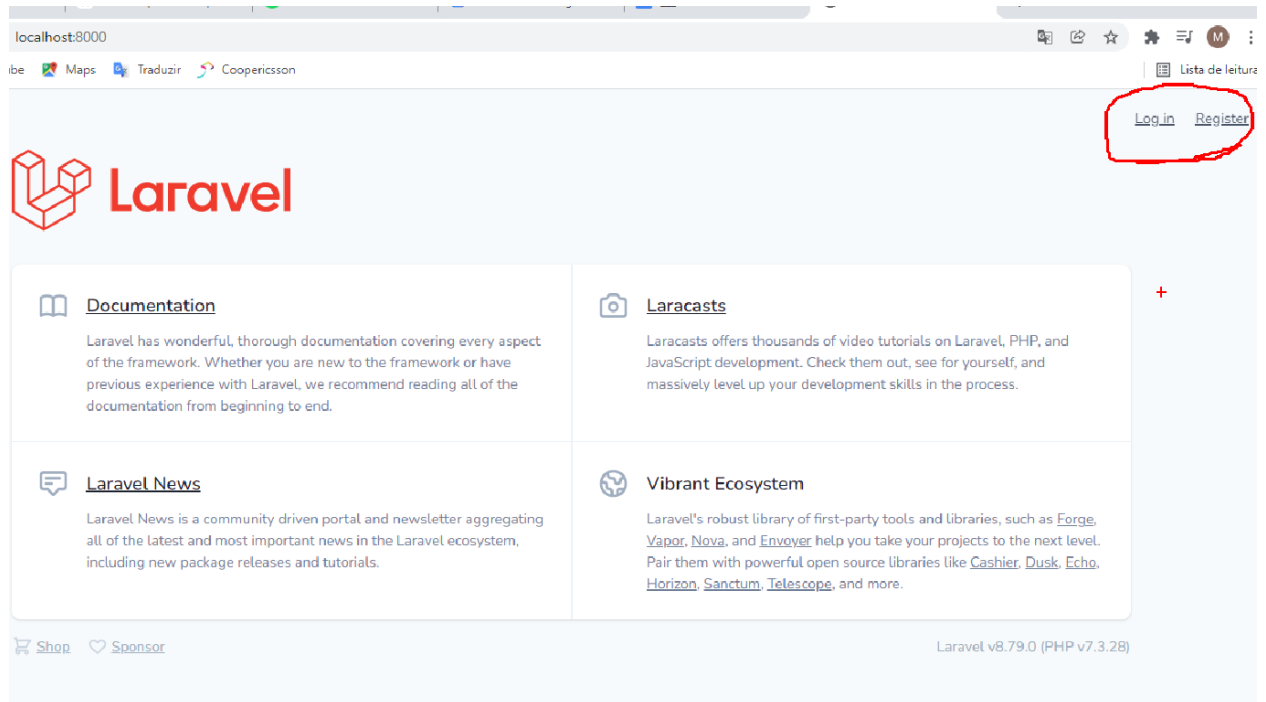
15. No terminal do Visual Code ou no CMD dentro da pasta do projeto digite:

php artisan serve

16. Acesse o site do projeto pelo navegador na seguinte URL:

localhost:8000

E veja como está o site.



A página do Laravel é apresentada indicando que o projeto está rodando. A diferença é que no canto superior direito é mostrado os links para login e registro de usuário. Esse modelo pode ser usado para um sistema que tenha uma página de apresentação e links de login para acessar um dashboard.

17. **Conhecendo a estrutura do projeto criado:** Abra o arquivo de rotas na pasta routes>web.php

Este arquivo contém a rota principal e a rota para o Dashboard

```
Route::get('/', function () {  
    return view('welcome');  
});  
  
Route::middleware(['auth:sanctum', 'verified'])->get('/dashboard', function () {  
    return view('dashboard');  
})->name('dashboard');
```

Observe os detalhes: A rota principal carrega a view “welcome”, essa view pode ser acessada no arquivo resources>views>welcome.blade.php

Todos os arquivos de uma view terminam com a extensão **.blade.php**, basicamente esses arquivos contém código HTML com diretivas blade para exibir conteúdo dinâmico. Veja que a pasta resources/views contém vários arquivos .blade.php. Mais adiante, nessa atividade, você escreverá código HTML com diretivas blade.

18. Experimente acessar **localhost:8000/dashboard**

Veja que o dashboard não é carregado, ao invés disso a tela de login é carregada.

Isso acontece porque a rota do dashboard contém um método middleware antes do método get: `middleware(['auth:sanctum', 'verified'])`
Isso significa que o carregamento do dashboard só acontece após a autenticação ser verificada.

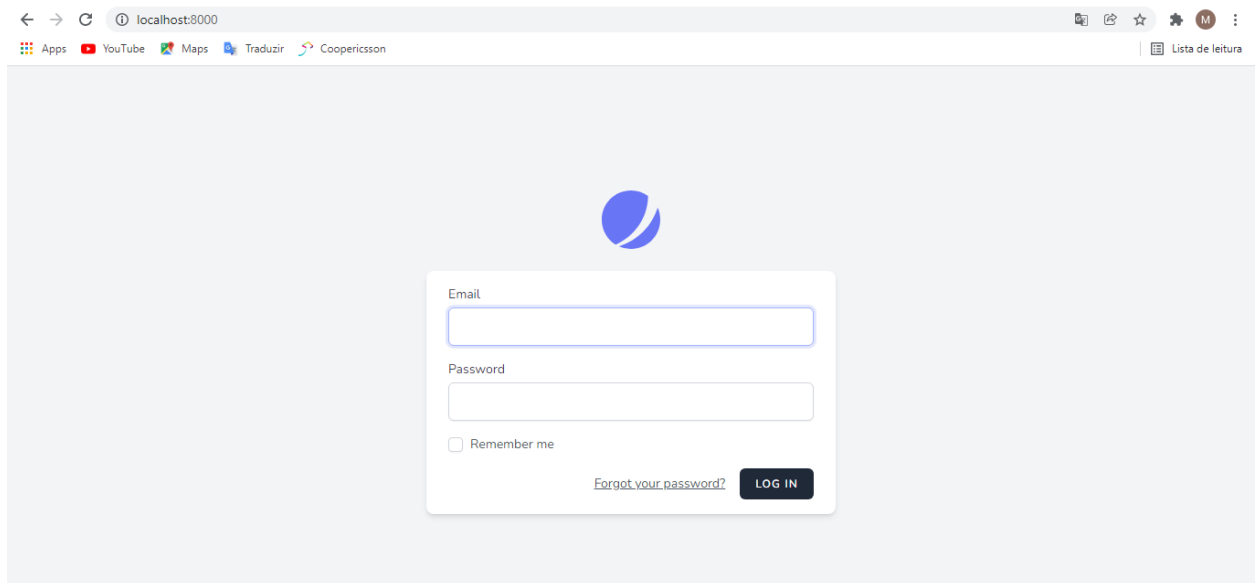
19. **Modificações necessárias nas rotas e nas views.** O wireframe do projeto inicia na tela de login e depois a tela de extrato é carregada. Abra o arquivo de rotas - `routes.web.php` - para fazer as modificações necessárias.

20. Modifique a rota principal conforme o código abaixo:

```
Route::get('/', function () {  
    return view('auth/login');  
});
```

Este código carrega a view login, ou seja, define a página de login como a página inicial do site.

21. Acesse a URL `localhost:8000` e veja que a página de login é carregada como a principal.



22. **Alterando o arquivo de login.** Abra o arquivo `resources>views>auth>login.blade.php` veja que o arquivo contém parte de código HTML para exibição do login.

A arquitetura do Laravel Jetstream é um pouco diferente de outros kits iniciais de aplicativos Laravel. As partes de autenticação são alimentadas pelo Laravel Fortify, que é um back-end de autenticação para o Laravel onde tudo já está pronto, você apenas utiliza as rotas.

O Fortify registra as rotas e controladores necessários para implementar todos os recursos de autenticação do Laravel, incluindo login, registro, redefinição de senha, verificação de e-mail e muito mais. Veja mais detalhes em <https://jetstream.laravel.com/2.x/concept-overview.html>

O Jetstream cria um layout "convidado" que é usado para definir o layout das páginas relacionadas à autenticação do Jetstream, como as páginas de login, registro e redefinição de senha do seu aplicativo. Ao usar a pilha Livewire, esse layout é definido `resources/views/layouts/guest.blade.php`. Ou seja, primeiro é carregado o arquivo `guest.blade.php` com o cabeçalho da página de login e depois é carregado o arquivo `login.blade.php` com o formulário de login.

23. Utilizando Bootstrap e FontAwesome nas páginas de login e registro de usuário.

Abra o arquivo `resources>views>layouts>gest.blade.php` e acrescente abaixo do comentário `<!-- styles -->` o carregamento do Bootstrap e do FontAwesome por tags de CDN. Se quiser copie o trecho abaixo:

```
<link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"
rel="stylesheet"
integrity="sha384-EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTWfSpd3yD65V
ohhpUuCOMLASjC" crossorigin="anonymous">
<link rel="stylesheet"
href="https://pro.fontawesome.com/releases/v5.10.0/css/all.css"
integrity="sha384-AYmEC3Yw5cVb3ZcuHtOA93w35dYTsvhLPVnYs9eStHfGJvOvK
xVfELGroGkvsg+p" crossorigin="anonymous"/>
```

24. Inclua abaixo do comentário `<!-- Scripts -->` o carregamento do código javascript do Bootstrap por tag de CDN. Se quiser copie o trecho abaixo:

```
<script
src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.9.2/dist/umd/popper.min.js"
integrity="sha384-IQsoLXl5PILFhosVNubq5LC7Qb9DXgDA9i+tQ8Zj3iwWAwPtgFT
xbJ8NT4GN1R8p" crossorigin="anonymous"></script>
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.min.js"
integrity="sha384-cVKIPhGWiC2AI4u+LWgxfKTRlcfu0JTxR+EQDz/bglIdoEyl4H0zU
F0QKbrJ0EcQF" crossorigin="anonymous"></script>
```

25. Repita o carregamento dos arquivos CSS e JS conforme demonstrado nos itens 17 e 18 no arquivo `resources>views>layouts>app.blade.php`
Dessa forma o dashboard do projeto também utilizará Bootstrap e FontAwesome.

26. Alterando o formulário de login. Abra o arquivo `resources>views>auth>login.blade.php` e apague esse trecho que exibe o logotipo do

Jetstream **<x-jet-authentication-card-logo />**


27. Inclua antes do trecho `<x-slot name="logo">` um título com um ícone de dinheiro com o seguinte código:

APP FINANÇAS

28. Na linha 42 do arquivo login.blade.php, abaixo do início da diretiva @if acrescente um link para a página de cadastro de novo usuário com o seguinte código:

[Cadastre-se]({{ route('register') }})

O resultado deverá ficar semelhante a esse:

 APP FINANÇAS

Email

Password

☐ Remember me

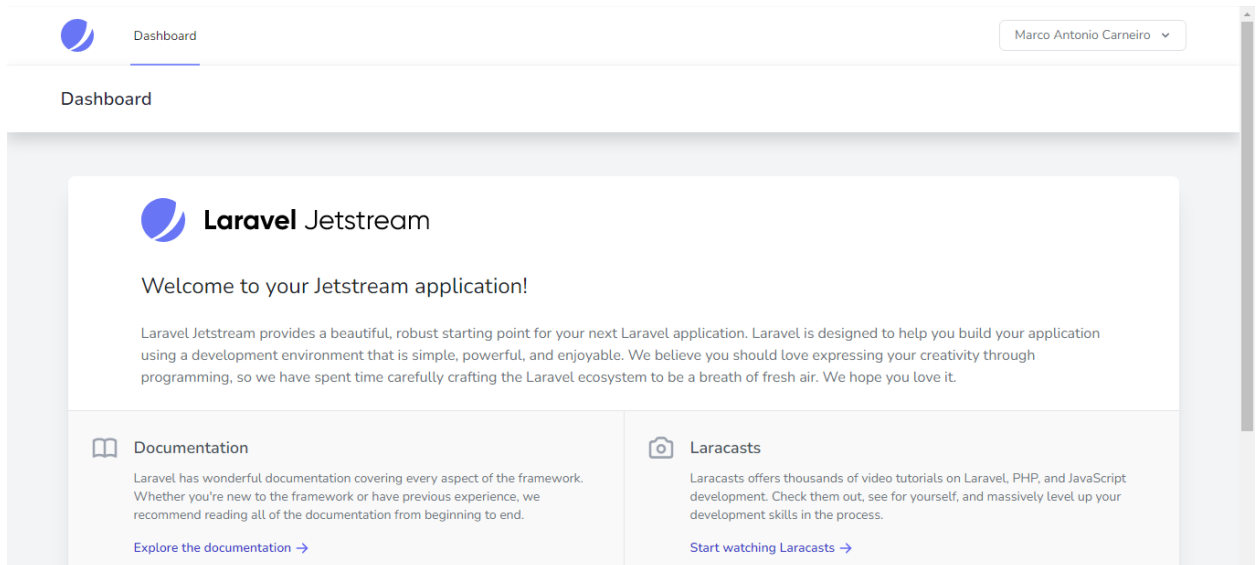
[Cadastre-se](#) [Esqueceu sua senha?](#) LOG IN

29. **DESAFIO:** Baseado nas alterações de logotipo feita no arquivo login.blade.php você deverá alterar o logotipo e traduzir os textos nos arquivos da pasta resources>views>auth :
- confirm-password.blade.php**
 - forgot-password.blade.php**
 - register.blade.php**
 - reset-password.blade.php**

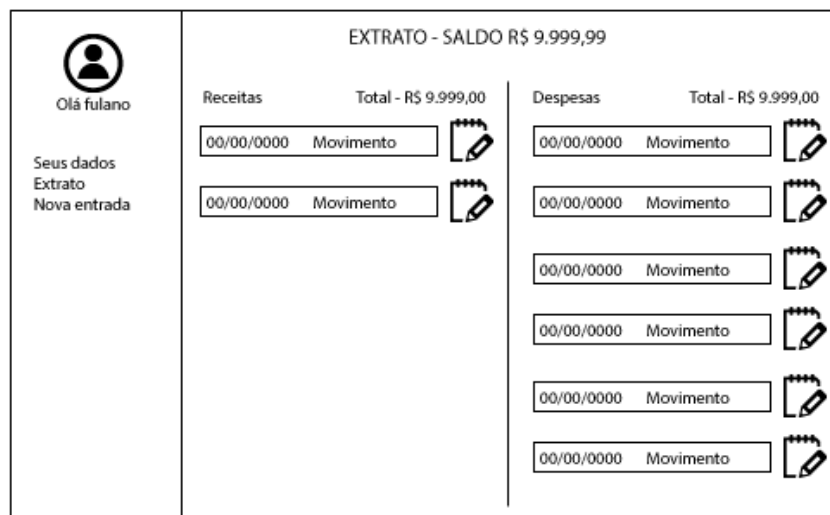
two-factor-challenge.blade.php e verify-email.blade.php

Essas são as views envolvidas em todo o processo de login e registro de usuário.

30. Acesse o site e crie uma conta de usuário (anote para não esquecer).
Observe que ao criar a conta o site já direciona a página de dashboard



Essa tela de dashboard precisa ser alterada para algo perto disso:



Essa página deve ter uma coluna de navegação e uma coluna de receitas e outra de despesas e acima o saldo deverá aparecer.

31. **REAPROVEITAMENTO DE LAYOUTS EM DIVERSAS VIEWS.** É muito comum em páginas web que a estrutura básica se repita em vários arquivos. O Blade permite que diversas views herdem uma mesma estrutura, possibilitando assim, o reaproveitamento de código. As views dashboard e a view de inserção e edição de movimentações vão

aproveitar da mesma estrutura da view “app”.

Dentro da pasta resources>views crie um novo arquivo com nome **app.blade.php**

32. No arquivo app.blade.php adicione o código abaixo:

```
<!DOCTYPE html>
<html lang="{{ str_replace('_', '-', app()->getLocale()) }}">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <meta name="csrf-token" content="{{ csrf_token() }}">

    <title>@yield('title')</title>

    <!-- Fonts -->
    <link rel="stylesheet"
href="https://fonts.googleapis.com/css2?family=Nunito:wght@400;600;700&displa
y=swap">

    <!-- Styles -->
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"
rel="stylesheet"
integrity="sha384-EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTWfSpd3yD65V
ohhpuuCOmLASjC" crossorigin="anonymous">
    <link rel="stylesheet"
href="https://pro.fontawesome.com/releases/v5.10.0/css/all.css"
integrity="sha384-AYmEC3Yw5cVb3ZcuHtOA93w35dYTsvhLPVnYs9eStHfGJvOvK
xVfELGroGkvsg+p" crossorigin="anonymous"/>

  </head>
  <body>
    <div class="container-fluid">
      <div class="row">
        @yield('navegacao')

        @yield('extrato')

      </div>
    </div>

    <!-- Scripts -->
```

```

<script
src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.9.2/dist/umd/popper.min.js"
integrity="sha384-IQsoLXI5PILFhosVNubq5LC7Qb9DXgDA9i+tQ8Zj3iwWAwPtgFT
xbJ8NT4GN1R8p" crossorigin="anonymous"></script>
<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.min.js"
integrity="sha384-cVKIPhGWiC2AI4u+LWgxfKTRlcfu0JTxR+EQDz/bglIdoEyl4H0zU
F0QKbrJ0EcQF" crossorigin="anonymous"></script>
</body>
</html>

```

Observe as seguintes diretivas blade:

@yield('title') - Essa diretiva vai exibir na view, o conteúdo de uma seção com nome “title”

@yield('navegacao') - Essa diretiva vai exibir na view, o conteúdo de uma seção com nome “navegacao”

@yield('conteudo') - Essa diretiva vai exibir na view, o conteúdo de uma seção com nome “conteudo”

33. ALTERAÇÕES DA VIEW DASHBOARD: Abra o arquivo

resources>views>dashboard.blade.php e apague todo o código existente.

34. Na primeira linha do arquivo dashboard.blade.php inclua a seguinte diretiva:

@extends('app')

Essa diretiva herda todo o código da view app.blade.php. Isso significa que qualquer outra view também poderá usar a estrutura da view app.blade.php

35. Em seguida acrescente a seguinte diretiva:

@section('title', 'Extrato')

Essa diretiva exibe no @yield('title') o valor 'Extrato' da seção 'title'

A diretiva **@section** suporta valores de texto, variáveis e estruturas HTML.

Todos esses valores são enviados para as diretivas **@yield** existentes no template (arquivo app.blade.php)

36. Acrescente após o @section extrato, as sections “navegacao” e extrato. O código final deverá ficar igual a este:

@extends('app')

@section('title', 'Extrato')

@section('navegacao')

<div class="col-3 bg-secondary text-white-50 vh-100">

<p class="text-center mt-3 mb-5 fw-bold fs-4">


```

        <i class="fas fa-money-bill-wave"></i>
        APP FINANÇAS
    </p>

    <p class="text-center mt-3 mb-5">
        <i class="fas fa-user fa-4x"></i><br>
        Olá, Usuário
    </p>

    <nav class="nav flex-column text-center">
        <a class="nav-link text-white" href="#">Extrato</a>
        <a class="nav-link text-white" href="#">Novo Movimento</a>
    </nav>
</div>
@endsection

@section('conteudo')
    <div class="col-9 text-secondary">
        <p class="mb-5 p-2 fw-bold fs-4 border-bottom border-secondary">
            EXTRATO - Saldo R$ 9.999,99
        </p>

        <div class="row">
            <div class="col-md-6">
                <p class="fs-2">Receitas</p>
                {{-- lista de receitas --}}
                <div class="card mb-3">
                    <div class="card-header">
                        99/99/9999
                    </div>
                    <div class="card-body">
                        Descrição do movimento
                    </div>
                    <div class="card-footer">
                        <a href="#" ><i class="fas fa-edit"></i></a> &nbsp; &nbsp; &nbsp;
                        <a href="#" ><i class="fas fa-trash-alt"></i></a>
                    </div>
                </div>

                {{-- total de receitas --}}
                <p class="fw-bold">Total: R$ 9.999,00</p>
            </div>

```

 APP FINANÇAS


Olá, Usuário

Extrato

Novo Movimento

EXTRATO - Saldo R\$ 9.999,99

Receitas

99/99/9999
Descrição do movimento
 
Total: R\$ 9.999,00

Despesas

99/99/9999
Descrição do movimento
 
Total: R\$ 9.999,00

34

novos movimentos e assim ter dados para exibir.

38. O primeiro passo é definir uma rota para essa view. Abra o arquivo routes>web.php

39. Acrescente uma nova rota com nome “novo-movimento”, essa rota só pode ser acessada se o usuário estiver logado, será necessário o método “middleware”.

Acrescente o código abaixo:

```
Route::middleware(['auth:sanctum', 'verified'])->get('/novo_movimento', function ()
{
    return view('novo_movimento');
})->name('novo_movimento');
```

40. **Criando a View com o formulário para inserção de novos movimentos.** Na pasta resources>views crie um novo arquivo com nome novo_movimento.blade.php e adicione o código abaixo:

```
@extends('app')
```

```
@section('title', 'Inserir movimento')
```

```
@section('navegacao')
```

```
<div class="col-3 bg-secondary text-white-50 vh-100">
```

```
<p class="text-center mt-3 mb-5 fw-bold fs-4">
```

```
<i class="fas fa-money-bill-wave"></i>
```

```
APP FINANÇAS
```

```
</p>
```

```
<p class="text-center mt-3 mb-5">
```

```
<i class="fas fa-user fa-4x"></i><br>
```

```
Olá, Usuário
```

```
</p>
```

```
<nav class="nav flex-column text-center">
```

```
<a class="nav-link text-white" href="#">Extrato</a>
```

```
<a class="nav-link text-white" href="#">Novo Movimento</a>
```

```
</nav>
```

```
</div>
```

```
@endsection
```

```
@section('extrato')
```

```
<div class="col-md-6 offset-md-2 mt-5">
```

```
<div class="panel panel-default">
```

```
<div class="panel-heading">
```

```

        <h3 class="panel-title text-secondary"><strong>Suas Finanças - Incluir
movimento </strong></h3>
    </div>
    <div class="panel-body">
        <form id="frm-novo-movimento" action="#" method="POST">
            <div class="form-group">
                <label for="movimento">Movimento</label>
                <input type="text" class="form-control" id="movimento"
name="movimento" placeholder="Descrição" required>
            </div>

            <br>

            <div class="form-check form-check-inline">
                <input class="form-check-input" type="radio" name="tipo"
id="opcao1" value="Despesa">
                <label class="form-check-label" for="opcao1">Despesa</label>
            </div>
            <div class="form-check form-check-inline">
                <input class="form-check-input" type="radio" name="tipo"
id="opcao2" value="Receita">
                <label class="form-check-label" for="opcao2">Receita</label>
            </div>

            <div class="form-group mt-4">
                <label for="valor">Valor</label>
                <input type="Number" class="form-control" id="valor"
name="valor" required>
            </div>

            <button type="submit" class="mt-4 btn btn-sm
btn-primary">Inserir</button>
        </form>
    </div>
</div>
@endsection

```

41. Para que a navegação funcione você deve alterar os links da seção navegação das views dashboard e novo_movimento para o seguinte código:

```

<nav class="nav flex-column text-center">
    <a class="nav-link text-white" href="{{route('dashboard')}}">Extrato</a>

```

```
<a class="nav-link text-white" href="{{route('novo_movimento')}}">Novo
Movimento</a>
</nav>
```

42. Teste a navegação entre as duas Views.

43. **MIGRATION PARA ADIÇÃO DE NOVA COLUNA - Relacionamento um para muitos.**

Uma modificação será necessária para a tabela **fin_movimentos**, pois é necessário criar uma relação de um usuário para vários movimentos, isso se faz através de uma definição de chave estrangeira por uma migration de alteração de tabela. No CMD ou terminal do Visual Code digite:

```
php artisan make:migration add_user_id_to_fin_movimentos_table
```

Esse comando vai criar um migration que irá adicionar o campo “user_id” na tabela “fin_movimentos”.

44. Abra o arquivo **add_user_id_to_fin_movimentos_table.php** na pasta database>migrations. No método up(), na linha 17 onde está o comentário vazio “//”, apague o comentário e adicione a seguinte instrução:

```
$table->foreignId('user_id')->constrained();
```

No método down(), na linha 29 onde está o comentário vazio “//”, apague o comentário e adicione a seguinte instrução:

```
$table->foreignId('user_id')->constrained()->onDelete('cascade');
```

O método up() vai adicionar a nova coluna de chave estrangeira “user_id” e o método down() vai remover todos os registros relacionados a um usuário.

O próximo passo é fazer uma alteração nos models do projeto para determinar esse relacionamento.

45. Abra o arquivo app>Models>Fin_movimento.php

46. Logo abaixo da instrução use HasFactory; adicione o seguinte método user():

```
public function user()
{
    return $this->belongsTo('App\Models\User');
}
```

Este método estabelece uma ligação com o Model User.

47. Abra o arquivo app>Models>User.php para estabelecer uma ligação com o Model Fin_movimento.

48. Adicione o método abaixo antes chave de fechamento da classe User:

```
public function fin_movimentos()
```

```
{
    return $this->hasMany('App\Models\Fin_movimento');
}
```

Esse método estabelece que o Model User tem uma relação para vários registros do Model Fin_movimento. Todas essas alterações são importantes para que o formulário de novos movimentos possa inserir dados na tabela fin_movimentos tendo o relacionamento com a tabela de usuários, se um usuário for removido, os registros deste usuário na tabela fin_movimentos serão automaticamente removidos também.

49. No CMD ou terminal do Visual Code digite: **php artisan migrate:fresh**
Essa linha vai atualizar as colunas com as alterações especificadas.

#	Nome	Tipo	Agrupamento (Collation)	Atributos	Nulo	Predefinido	Comentários	Extra	Ações
1	id	bigint(20)		UNSIGNED	Não	Nenhum		AUTO_INCREMENT	Muda Elimina Mais
2	created_at	timestamp			Sim	NULL			Muda Elimina Mais
3	updated_at	timestamp			Sim	NULL			Muda Elimina Mais
4	valor	double(12,2)			Não	Nenhum			Muda Elimina Mais
5	descricao	text	utf8mb4_unicode_ci		Não	Nenhum			Muda Elimina Mais
6	tipo	varchar(255)	utf8mb4_unicode_ci		Não	Nenhum			Muda Elimina Mais
7	data	date			Não	Nenhum			Muda Elimina Mais
8	user_id	bigint(20)		UNSIGNED	Não	Nenhum			Muda Elimina Mais

Observe que a tabela fin_movimentos agora possui a coluna user_id

50. CRIANDO O CONTROLLER PARA OS MOVIMENTOS FINANCEIROS.

Os controllers são parte fundamental da lógica do sistema, no caso de um sistema de controle financeiro devemos criar um controller para cuidar de todo o CRUD envolvido. No CMD ou terminal do Visual Code digite:

php artisan make:controller MovimentoController

Este comando vai criar na pasta Http>Controllers o arquivo MovimentoController.php. Neste arquivo iremos adicionar as lógicas para inserir, editar, excluir e exibir os movimentos financeiros do usuário. Observe que o Laravel cria a classe vazia, iremos trabalhar alguns métodos mais adiante.

51. FAZENDO O FORMULÁRIO DE INSERÇÃO DE MOVIMENTO FUNCIONAR.

O processamento de todo formulário acontece no atributo “action” da tag <form>

O atributo action geralmente é uma URL que processa todos os dados coletados no formulário. No Laravel essa URL é definida nos arquivos de rotas e indica um Controller que coleta os dados e os processa, enviando os dados para a tabela.

52. Abra o arquivo novo_movimento.blade.php para adicionar algumas alterações no formulário.

53. Modifique a tag do início do formulário de

```
<form id="frm-novo-movimento" action="#" method="POST"> Para  
<form id="frm-novo-movimento" action="novomovimento" method="POST">
```

Esse valor do atributo action "novomovimento" significa a URL que vai tratar os dados do formulário, no caso do Laravel é uma rota que aciona um método de um controller que vai fazer a gravação dos dados no banco.

54. **Adicionando segurança ao formulário.** Abaixo da tag <form> no arquivo novo_movimento.blade.php digite a diretiva @csrf , a modificação deverá ficar assim:

```
<form id="frm-novo-movimento" action="#" method="POST">  
    @csrf
```

Sem proteção CSRF, um site malicioso pode criar um formulário HTML que aponta para a rota do seu aplicativo e envia o próprio endereço de e-mail ou outro dado do usuário malicioso.

A diretiva @csrf gera automaticamente um "token" CSRF para cada sessão de usuário ativa gerenciada pelo aplicativo. Esse token é usado para verificar se o usuário autenticado é a pessoa que realmente está fazendo as solicitações ao aplicativo. Como esse token é armazenado na sessão do usuário e é alterado sempre que a sessão é regenerada, um aplicativo mal-intencionado não consegue acessá-lo.

O próximo passo é criar um método no controller que salva os dados do formulário no banco de dados e em seguida definir a rota que irá acessar o controller.

55. Abra o arquivo **app>Http>Controllers>MovimentoController.php**

56. Abaixo da instrução **use Illuminate\Http\Request;**

Adicione essa instrução: **use app\Models\Fin_movimento;**

Essa instrução indica que esse controller se relaciona com o model Fin_movimento, dessa forma será possível gravar dados na tabela de movimentos.

57. Dentro da classe MovimentoController adicione o seguinte método no lugar do comentário "//"

```
public function store(Request $request)  
{  
    $movimento = new Fin_movimento;  
  
    $movimento->valor = $request->valor;  
    $movimento->descricao = $request->descricao;  
    $movimento->tipo = $request->tipo;
```

```

    $movimento->data = date('Y-m-d');
    $user = auth()->user();
    $movimento->user_id= $user->id;

    $movimento->save();
    return redirect('dashboard');
}

```

Por padrão o método store é utilizado para salvar dados no banco, esse nome é apenas uma convenção, o método poderia se chamar gravar que também iria funcionar.

Explicando esse método:

store(Request \$request) significa que o método store recebe dados do tipo Request (dados de formulário) através da vária \$request que guarda todos os dados preenchidos nos campos do formulário.

\$movimento = new Fin_movimento; significa que a variável \$movimento é uma instância do Model Fin_movimento, ou seja, representa a tabela de movimentos do banco de dados. As linhas

\$movimento->valor = \$request->valor;

\$movimento->descricao = \$request->descricao;

\$movimento->tipo = \$request->tipo;

Associam os valores dos campos da tabela (variável \$movimento) aos mesmos valores dos campos do formulário (variável \$request).

\$movimento->data = date('Y-m-d'); associa ao campo data da tabela de movimentos a data atual no formato indicado para gravação na tabela, ano-mês-dia.

\$user = auth()->user(); armazena na variável \$user os dados do usuário logado

\$movimento->user_id= \$user->id; associa o id do usuário logado ao campo user_id da tabela de movimentos.

\$movimento->save(); salva os dados na tabela do banco de dados e **return redirect('dashboard');** redireciona para a rota “dashboard”.

58. Abra o arquivo routes>web.php, abaixo da instrução use

Illuminate\Support\Facades\Route; adicione a seguinte instrução:

use App\Http\Controllers\MovimentoController;

Essa instrução estabelece uma ligação com o controller MovimentoController o que significa que será possível criar rotas com qualquer método deste controller.

59. Ainda dentro do arquivo routes>web.php adicione a seguinte rota:

Route::post('/novomovimento', [MovimentoController::class, 'store']);

Essa é uma rota do tipo post que executa o método “store” do controller

MovimentoController, podemos dizer que a URL “novomovimento” é um apelido para o método “store” do controller MovimentoController.

No formulário, quando o usuário clicar no botão submit, o método store será acionado.

60. Faça um teste, acesse o formulário de inserção, digite alguns dados, clique no botão submit (enviar) e depois veja se os dados foram salvos na tabela fin_movimentos.

Suas Finanças - Incluir movimento

Movimento

Pagamento conta de luz

☒ Despesa ☐ Receita

Valor

140,50

Inserir

Dados inseridos no formulário.

✓ A mostrar registros de 0 - 0 (1 total, A consulta demorou 0,0004 segundos.)

SELECT * FROM `fin_movimentos`

☐ Perfil [Editar em linha] [Editar] [Explicar SQL] [Criar código PHP] [Atualizar]

☐ Mostrar tudo | Número de registros: 25 | Filtrar registros: Pesquisar esta tabela

+ Opções

	id	created_at	updated_at	valor	descricao	tipo	data	user_id
<input type="checkbox"/>	1	2022-02-03 00:48:38	2022-02-03 00:48:38	140.00	pagamento conta de luz	Despesa	2022-02-03	1

↑ ☐ Marcar todos Com os seleccionados: ☐ Editar ☐ Copiar ☐ Apagar ☐ Exportar

☐ Mostrar tudo | Número de registros: 25 | Filtrar registros: Pesquisar esta tabela

Operações resultantes das consultas

☐ Imprimir ☐ Copiar para área de transferência ☐ Exportar ☐ Mostrar gráfico ☐ Criar visualização

O movimento inserido na tabela.

61. **Exibindo os movimentos cadastrados no dashboard.** Por enquanto o Dashboard só exibe o modelo criado. Para exibir os movimentos cadastrados do usuário logado é necessário criar um novo método no controller MovimentoController.php que vai coletar os movimentos do usuário e enviar para a view. Não podemos esquecer de modificar a rota existente para que acesse o método no Controller. Abra o arquivo MovimentoController.php.

62. Para exibir os movimentos cadastrados pelo usuário logado você vai criar um método com nome getMovimentos (mas pode ter outro nome se você quiser), esse método vai consultar os movimentos de despesa, receita e seus respectivos totais. Escreva o seguinte código antes do fechamento da classe:

```
public function getMovimentos()
{
```

```

$user = auth()->user();
$despesas = Fin_movimento::where('tipo', 'Despesa')
->where('user_id', $user->id)->get();
$receitas = Fin_movimento::where('tipo', 'Receita')
->where('user_id', $user->id)->get();

$totaldespesas = $despesas->sum('valor');
$totalreceitas = $receitas->sum('valor');

$parametros = [
    'despesas'=> $despesas,
    'receitas'=>$receitas,
    'totaldespesas'=>$totaldespesas,
    'totalreceitas'=>$totalreceitas
];

return view('dashboard', $parametros);
}

```

Explicando o código:

\$user = auth()->user(); obtém os dados do usuário logado.

\$despesas = Fin_movimento::where('tipo', 'Despesa')
->where('user_id', \$user->id)->get(); consulta na tabela fin_movimentos os registros do usuário logado onde o campo tipo é igual a “Despesa”, o resultado é armazenado na variável **\$despesas**, a mesma coisa é feita na variável **\$receitas** para as receitas.

\$totaldespesas = \$despesas->sum('valor'); armazena na variável **\$totaldespesas** a soma do campo valor, veja que a mesma coisa é feita para as receitas na variável **\$totalreceitas**.

Finalmente a view dashboard é renderizada enviando os dados pela variável **\$parametros**, note que essa variável é uma array associativa, os itens dessa array serão exibidos na view dashboard.

63. Abra o arquivo routes>web.php para alterar a rota do dashboard. Essa rota simplesmente exibe a view. Será necessário indicar que essa rota acessa o método getMovimentos do controller MovimentoController.php. Mude o código da rota para o código abaixo:

```

Route::middleware(['auth:sanctum', 'verified'])
->get('/dashboard', [MovimentoController::class, 'getMovimentos'])
->name('dashboard');

```

64. **EXIBINDO OS DADOS OBTIDOS PELO CONTROLLER NA VIEW.** Abra o arquivo `resources>views>dashboard.blade.php`

65. Localize no código o comentário `{{-- lista de receitas --}}`
Apague toda a estrutura que está abaixo deste comentário, até o comentário `{{-- total de receitas --}}`

66. Adicione esta estrutura:

```
@if (count($receitas) === 0)
    <p>Não há receitas para esse usuário.</p>
@else
    @foreach ($receitas as $receita)
        <div class="card mb-3">
            <div class="card-header">
                {{$receita->data}}
            </div>
            <div class="card-body">
                {{$receita->descricao}}
            </div>
            <div class="card-footer">
                <strong>R$ {{$receita->valor}}</strong> &nbsp; &nbsp; &nbsp;
                <a href="#" ><i class="fas fa-edit"></i></a> &nbsp; &nbsp; &nbsp;
                <a href="#" ><i class="fas fa-trash-alt"></i></a>
            </div>
        </div>
    @endforeach
@endif
```

Esse código contém algumas diretivas blade, vamos entender a função de cada parte.

@if (count(\$receitas) === 0) @else @endif

Essa estrutura verifica se a quantidade de itens da variável `$receitas` é igual a 0 (diretiva **@if**), se essa condição for verdadeira, significa que não há nenhuma receita lançada, então a mensagem “Não há receitas para esse usuário” é exibida.

Caso contrário (diretiva **@else**) uma diretiva de repetição **@foreach** é exibida na página. A diretiva **@foreach** faz uma iteração em todos os itens de uma variável do tipo array ou um objeto com diversos valores, enquanto houver valores na variável, o Laravel vai repetir uma determinada estrutura com diferentes valores, a vantagem é escrever a estrutura apenas uma vez para que seja repetida quantas vezes for necessária. O resultado deverá ser semelhante a esse:



Veja mais detalhes sobre as diretivas blade em <https://laravel.com/docs/8.x/blade#blade-directives>

67. Repita esse procedimento entre os comentários `{{-- lista de despesas --}}` e `{{-- total de despesas --}}`, modifique \$receitas por \$despesas e \$receita por \$despesa. Todas essas variáveis foram enviadas pelo Controller MovimentoController.
68. Procure o comentário `{{-- total de receitas --}}` e modifique a tag `<p>` por esta:


```
<p class="fw-bold">Total: R$ {{$totalreceitas}}</p>
```

 O termo `{{ $totalreceitas }}` vai exibir a soma do campo valor. Faça o mesmo procedimento para o total de despesas.
69. EXIBINDO O SALDO DO USUÁRIO. Para exibir o saldo do usuário basta subtrair o total de receitas pelo total de despesas. Localize no código o seguinte trecho:


```
EXTRATO - Saldo R$ 9.999,00
```

 e substitua por


```
{{ $totalreceitas - $totaldespesas }}
```
70. Acesse o site e faça alguns lançamentos de despesas e receitas, o resultado deverá ficar semelhante a esse:



71. **EXIBINDO O NOME DO USUÁRIO.** Se você quiser, pode enviar essa informação pelo método `getMovimentos` do controller `MovimentoController`, é mais organizado, mas nesse exemplo vamos fazer diretamente na view pois aceita códigos php tranquilamente. No arquivo `resources>views>dashboard.blade.php` localize na seção “navegacao” o texto “Olá usuário”, modifique conforme o código abaixo:

Olá, <?php \$user = auth()->user(); echo \$user->name; ?>

72. Acesse o Dashboard e veja se está tudo OK.

73. **ADICIONANDO UM LINK PARA LOGOUT.** Como no Jetstream todos os recursos para trabalhar com a autenticação já estão prontos, basta adicionar algum código na view. Abra o arquivo `resources>views>dashboard.blade.php` e acima do fechamento da tag `nav` adicione o código abaixo:

```
<form method="POST" action="{{ route('logout') }}">
    @csrf
    <a class="nav-link text-light" href="{{ route('logout') }}"
    onclick="event.preventDefault();
        this.closest('form').submit(); " role="button">
        <i class="fas fa-sign-out-alt"></i>

        {{ __('Log Out') }}
    </a>
</form>
```

74. Faça essas mesmas alterações da seção “navegacao” na mesma seção “navegacao” no arquivo `resources>views>novo_movimento.blade.php` e experimente fazer logout e

cadastrar um novo usuário e fazer lançamentos para esse novo usuário.

75. **EDITANDO UM MOVIMENTO.** Para a página de edição funcionar, será necessário dois novos métodos no controller MovimentoController e também uma nova view.

Abra o arquivo **app>Http>Controllers>MovimentoController.php**

76. Antes da chave de fechamento do controller insira o método abaixo:

```
public function editar($id)  
{  
    $movimento = Fin_movimento::findOrFail($id);  
    return view ('editar', ['movimento' => $movimento]);  
}
```

Explicando o código:

public function editar(\$id) o método recebe o id do movimento a ser editado
\$movimento = Fin_movimento::findOrFail(\$id); retorna para a variável \$movimento todos os dados do movimento que corresponde com o id. A função findOrFail retorna os dados de um único registro, se não encontrar nada, retorna um erro.
return view ('editar', ['movimento' => \$movimento]); carrega a view edit enviando os dados do registro, esses dados irão preencher o formulário de edição.

77. A view editar será praticamente igual a view novo_movimento, abra o arquivo **app>resources>views>novo_movimento.blade.php** e salve como **editar.blade.php**. Veja que o projeto contém 2 views praticamente iguais. Em um ambiente de trabalho profissional o ideal seria a criação de um template com o formulário de novo movimento e edição, mas nessa atividade vamos utilizar uma cópia para simplificar o processo.

78. No arquivo editar.blade.php faça as seguintes alterações:

A seção title deverá ficar assim: **@section('title', 'Editar movimento')**

Modifique a tag <form> para isso:

<form id="frm-edita-movimento" action="updatemovimento" method="POST">

79. Inclua abaixo da diretiva @csrf a seguinte diretiva:

@method('PUT')

Essa diretiva instrui o formulário que o método que será trabalhado é o método 'PUT' utilizado em formulários que vão processar um update.

80. No campo de descrição acrescente o atributo value dessa forma:

value="{{ \$movimento->descricao }}" esse atributo vai mostrar o valor do campo conforme o que está gravado no banco de dados.

81. Modifique o bloco com as tags input radio para esse código:

```
<div class="form-check form-check-inline">  
    @if ($movimento->tipo == 'Despesa')
```

```

        <input class="form-check-input" type="radio" name="tipo" id="opcao1"
value="Despesa" checked>
    @else
        <input class="form-check-input" type="radio" name="tipo" id="opcao1"
value="Despesa">
    @endif
    <label class="form-check-label" for="opcao1">Despesa</label>
</div>
<div class="form-check form-check-inline">
    @if ($movimento->tipo == 'Receita')
        <input class="form-check-input" type="radio" name="tipo" id="opcao2"
value="Receita" checked>
    @else
        <input class="form-check-input" type="radio" name="tipo" id="opcao2"
value="Receita">
    @endif
    <label class="form-check-label" for="opcao2">Receita</label>
</div>

```

Basicamente esse código verifica o tipo de despesa para apresentar a opção certa marcada (atributo checked).

82. Finalmente no campo de valor acrescente um atributo value conforme abaixo:

```

{{ $movimento->valor }}

```

83. Adicionando a rota para acessar o formulário de edição. Abra o arquivo routes>web.php e adicione a seguinte rota:

```

Route::middleware(['auth:sanctum', 'verified'])
->get('/editar/{id}', [MovimentoController::class, 'editar']->name('editar'));

```

Observe que essa rota aguarda o id do registro, isso será informado na view dashboard.

84. Abra a view dashboard.blade.php e dentro da estrutura @foreach da coluna de receitas, modifique o link de edição para esse código:

```

<a href="{{ route('editar', [$receita->id]) }}" ><i class="fas fa-edit"></i></a>

```

Faça o mesmo procedimento na coluna de despesas mas utilize a variável \$despesa

85. Faça um teste, acesse o dashboard e clique em algum link de edição, o formulário com os dados do registro deverão ser carregados.

DICA: Se o código estiver correto e você tiver visualizando um erro 404, experimente limpar o cache de rotas com o seguinte comando do artisan:
php artisan optimize

86. **Model, Controller e rota para fazer a atualização dos dados.** Abra o Model utilizado para a tabela movimentos, arquivo app>Models>Fin_movimento.php

Abaixo de **use HasFactory**; inclua a seguinte instrução:

protected \$guarded = [];

Por razões de segurança, todos os modelos Eloquent são protegidos contra atribuição em massa por padrão.

protected \$guarded = []; indica que todos os campos poderão ser atualizados. Se você especificar alguns campos nessa lista, somente eles não serão atualizados, podemos dizer que esta array funciona como uma lista negra.

87. Salve o arquivo.

88. Abra o arquivo **app>Http>Controllers>MovimentoController.php** e inclua o método abaixo:

```
public function updatemovimento(Request $request)  
{  
    Fin_movimento::findOrFail($request->id)->update($request->all());  
    return redirect('/dashboard');  
}
```

Explicando o código:

public function updatemovimento(Request \$request) o método “updatemovimento” consta no atributo action do formulário de edição, significa que quando o usuário clicar no botão submit, esse método será acionado conforme definido na rota. A variável **Request \$request** recebe os valores dos campos do formulário.

Fin_movimento::findOrFail(\$request->id)->update(\$request->all()); atualiza o registro definido em “id” com todos os campos enviados por request.

return redirect('/dashboard'); após a atualização, redireciona para o dashboard, mostrando os dados atualizados.

89. **ROTA PARA A AÇÃO DE ATUALIZAR OS REGISTROS.** Por fim, abra o arquivo **routes>web.php** e inclua seguinte rota:

```
Route::middleware(['auth:sanctum', 'verified'])  
->put('/updatemovimento/{id}', [MovimentoController::class,  
'updatemovimento']->name('updatemovimento'));
```

90. Salve os arquivos e faça uma modificação em algum movimento.

91. **MELHORANDO A EXIBIÇÃO DAS DATAS NO DASHBOARD.** Observe que no dashboard as datas são exibidas no formato **ano-mês-dia** que é o formato que as datas são gravadas no banco de dados. Para uma melhor experiência do usuário é necessário modificar o formato para **dia/mês/ano**. Abra o arquivo **app>resources>views>dashboard.blade.php** e encontre no código a parte onde é exibida as datas para as receitas **\$receita->data**, modifique para **{{ \Carbon\Carbon::parse(\$receita->data)->format('d/m/Y') }}**

Faça o mesmo procedimento para as despesas

```
{{ \Carbon\Carbon::parse($despesa->data)->format('d/m/Y') }}
```

92. LÓGICA PARA DELETAR MOVIMENTOS - Controllers, View e Rotas.

Controller: Abra o arquivo `app>Http>Controllers>MovimentoController.php` e inclua o método abaixo:

```
public function destroy($id)
{
    Fin_movimento::findOrFail($id)->delete();
    return redirect('/dashboard');
}
```

Entendendo o código:

public function destroy(\$id) A função recebe o id do registro para ser deletado.

Fin_movimento::findOrFail(\$id)->delete(); Localiza o registro e exclui.

return redirect('/dashboard'); Após excluir o registro, retorna para a dashboard

93. Rota: Abra o arquivo `routes>web.php` e inclua a rota abaixo:

```
Route::middleware(['auth:sanctum', 'verified'])
->delete('/deletar/{id}', [MovimentoController::class, 'destroy']->name('deletar');
```

94. View: Abra o arquivo `app>resources>views>dashboard.blade.php` localize na coluna de receitas o link para apagar o movimento

```
<a href="#" ><i class="fas fa-trash-alt"></i></a>
```

Modifique esse link incluindo o formulário abaixo:

```
<form action="{{route('deletar',[$receita->id])}}" method="POST" class="d-inline">
    @csrf
    @method('DELETE')
    <button type="submit" class="btn btn-link"
        onclick="if (!confirm('Excluir este movimento?')) { return false }">
        <i class="fas fa-trash-alt"></i></button>
</form>
```

Entendendo o formulário:

action="{{route('deletar',[\$receita->id])}}" esta action vai executar a rota para exclusão de registros, que por sua vez ativa o Controller responsável.

class="d-inline" esta é uma classe do Bootstrap que modifica a aparência do formulário para um padrão semelhante a um link.

@csrf diretiva blade para ativar a proteção contra ataques CSRF.

@method('DELETE') diretiva blade para forçar a utilização do método DELETE para o formulário.

onclick="if (!confirm('Excluir este movimento?')) { return false }" evento de javascript no ícone da lixeira com uma execução de um IF ternário, ou seja, um IF

simples em uma única linha. Este IF verifica se a função `confirm()` com a mensagem “Excluir este movimento?” foi cancelada, caso verdadeiro, não executa o envio do formulário (`return false`), se o usuário clicou em OK, o formulário continua sendo processado, ou seja, exclui o registro.

95. Faça alguns testes, inclua movimentos, edite-os, exclua-os, veja se o aplicativo funciona como o esperado. Experimente cadastrar um novo usuário e incluir novos movimento, verifique se o aplicativo funciona.
- Se tudo estiver como o esperado é hora de publicar o projeto na internet.

PROJETO LARAVEL UTILIZANDO LIVENWIRE

Livewire permite interações em tempo real sem recarregar as páginas, assim como as bibliotecas Javascript React e Vue. Essas bibliotecas que são muito utilizadas, tem uma configuração difícil e uma curva de aprendizado longa, Livewire tem o mesmo objetivo, porém a sua utilização é muito simples. Toda a documentação você encontra em <https://laravel-livewire.com/>

O passo a passo a seguir vai montar um pequeno site onde as pessoas possam digitar pensamentos e reflexões, muito parecido com o projeto feito anteriormente “Diz Ai”, esse novo projeto terá o nome de “Mind Drops” e utilizará o Livewire.

1. Inicie o projeto com o composer: **composer create-project laravel/laravel mind-drops**
2. Abra a pasta do novo projeto no Visual Code e digite no terminal:
composer require livewire/livewire
Este comando vai instalar o Livewire no projeto Mind Drops
3. No terminal digite: **php artisan livewire:publish**
Cria o arquivo **livewire.php** dentro da pasta config. Este arquivo define as configurações gerais do Livewire como o caminho dos controllers e das views, etc. Abra esse arquivo apenas para observar as configurações, não é necessário modificar.
4. **php artisan livewire:publish --assets**
Para publicar os arquivos de assets - css e js
5. Na raiz do projeto abra o arquivo **composer.json**
Inclua esse trecho de código (COM ASPAS)
"@php artisan vendor:publish --force --tag=livewire:assets --ansi"
Dentro da array **post-autoload-dump** (NÃO ESQUEÇA A VÍRGULA NO FINAL DO ITEM ANTERIOR)

6. Após a instalação, crie na pasta das views uma pasta com nome **layout** e nessa pasta adicione o arquivo **app.blade.php** com o seguinte código:

```
<!DOCTYPE html>
<html lang="pt-BR">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Mind Drops</title>
  @livewireStyles
</head>
<body>

  <div class="container">
    {{$slot}}
  </div>

  @livewireScripts
</body>
</html>
```

Os componentes serão renderizados em **\$slot**

7. **MODELS E TABELAS DO PROJETO.** Abra o phpMyAdmin do servidor Xampp e crie um novo banco de dados com nome **mind_drops** , esse será o banco de dados usado no projeto.
8. Abra o arquivo .env e adicione os dados do banco de dados.
9. No terminal do Visual Code digite:
php artisan make:model Drop -m
Esse comando vai criar o Model com nome Drop e também o migration para essa tabela (atributo -m)
Veja que os arquivos foram criados na pasta models e na pasta migrations.
10. Abra o arquivo app>models>Drop.php e acrescente logo abaixo de use HasFactory;
protected \$fillable=['content']; essa linha define que apenas a coluna content será preenchida na tabela.
11. Abra o arquivo de migrations na caminho database>migrations>create_table_drops.php para definir que colunas serão criadas na tabela. Acrescente no método up() as linhas abaixo:

```

$stable->string('content');
$stable->unsignedBigInteger('user_id');
//relacionamento com a tabela de usuários
$stable->foreign('user_id')->references('id')->on('users')->onDelete('cascade');

```

12. No terminal digite **php artisan migrate** esse comando vai criar as tabelas do banco de dados.

13. **RELACIONAMENTO DE UM PARA MUITOS COM A TABELA DE USUÁRIOS e DROPS.** Abra o arquivo **app>Models>User.php** e acrescente dentro da estrutura da classe a função abaixo:

```

public function drops()
{
    return $this->hasMany(Drop::class);
}

```

14. Abra o arquivo **app>Models>Drop.php** e acrescente dentro da classe o seguinte método que cria o relacionamento de muitos para um:

```

public function user()
{
    return $this->belongsTo(User::class);
}

```

15. **Componentes e databinds.** No terminal digite

php artisan livewire:make ShowDrops

Esse comando vai criar o componente ShowDrops.

Observe que são criados os arquivos **app>Http>Livewire>ShowDrops.php** e o arquivo **resources>views>livewire>show-drops.blade.php**, ou seja, o comando criou um controller e uma view.

16. Abra o controller **app>Http>Livewire>ShowDrops.php** e adicione a propriedade pública **\$mensagem**, o código deverá ficar assim:

```

<?php
namespace App\Http\Livewire;
use Livewire\Component;
class ShowDrops extends Component
{
    public $mensagem = 'Olá Mind Drops!';
    public function render()
    {
        return view('livewire.show-drops');
    }
}

```

17. Abra o arquivo `resources>views>livewire>show-drops.blade.php` e deixe o código assim:

```
<div>
```

```
    Estabelecendo ligação com a propriedade pública. <br>
```

```
    <input type="text" name="mensagem" id="mensagem"
    wire:model="mensagem">
```

```
    <p>Atualiza o valor da propriedade pública</p>
```

```
    Este é o valor atual: {{$mensagem}}
```

```
</div>
```

18. Abra o arquivo de rotas e adicione o carregamento do controller e a rota, o arquivo de rotas deverá ficar assim:

```
<?php
```

```
use Illuminate\Support\Facades\Route;
```

```
use App\Http\Livewire\{
```

```
    ShowDrops
```

```
};
```

```
Route::get('/', function () {
```

```
    return view('welcome');
```

```
});
```

```
Route::get('drops', ShowDrops::class);
```

19. Inicie o servidor e acesse a rota, veja que a mensagem inicial é exibida sem a necessidade de enviá-la no carregamento da view, o livewire acessa as propriedades públicas dos controller de forma muito simples.

20. Digite um novo valor no campo e veja que o valor da propriedade pública foi atualizado sem que fosse necessário escrever código javascript, na verdade o Livewire escreve esse código “por baixo dos panos” sem que você se preocupe com isso.

OBSERVAÇÃO: Todo código do componente deve estar entre <div></div>.

21. **UTILIZANDO FACTORY PARA GERAR USUÁRIOS PARA TESTE.** Para testar o componente ShowDrops precisamos cadastrar usuários e alguns drops, mas para não fazer todo o processo de páginas de cadastro vamos usar um recurso do Laravel chamado Factory, com ele podemos gerar registros nas tabelas do banco de dados sem precisar criar formulários. Esse recurso é somente para testes.

No terminal digite: **php artisan make:seed UserSeed**

Observe que o arquivo é criado em `database>seeders>UserSeed.php`

22. Abra o arquivo **database>seeders>UserSeed.php**.

23. Acrescente a referência ao model antes da declaração da classe:

```
use App\Models\User;
```

24. Dentro do método `run()` digite: **User::factory(10)->create();**

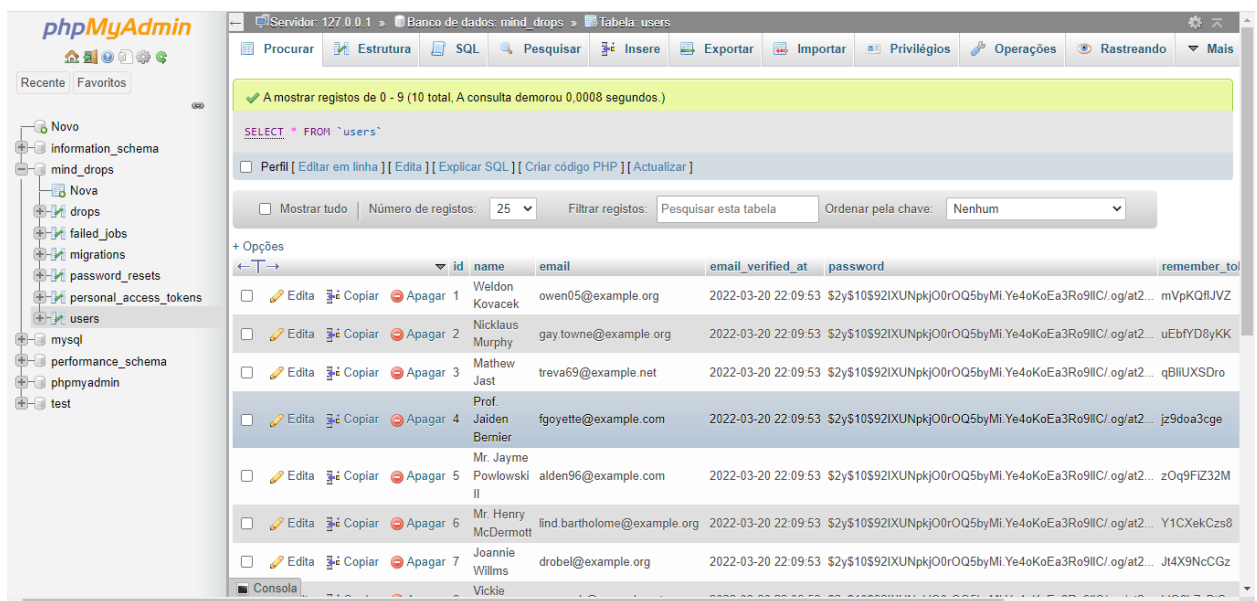
Essa instrução vai criar 10 novos usuários.

25. Abra o arquivo **database>seeders>DatabaseSeeder.php** e no método `run()` inclua o código abaixo:

```
$this->call([  
    UserSeed::class,  
]);
```

26. No terminal digite: **php artisan db:seed**

Este comando vai processar os seeders criados. Observe pelo phpMyAdmin os novos usuários criados.



The screenshot shows the phpMyAdmin interface. On the left, the database structure is visible, including 'mind_drops' and 'users' tables. The main panel displays the 'users' table with 10 records. The table has columns: id, name, email, email_verified_at, password, and remember_token. The records are as follows:

	id	name	email	email_verified_at	password	remember_token
<input type="checkbox"/>	1	Weldon Kovacek	owen05@example.org	2022-03-20 22:09:53	\$2y\$10\$92IXUNpkj0rOQ5byMi.Ye4oKoEa3Ro9llC/.og/at2...	mVpKQfJvZ
<input type="checkbox"/>	2	Nicklaus Murphy	gaytowne@example.org	2022-03-20 22:09:53	\$2y\$10\$92IXUNpkj0rOQ5byMi.Ye4oKoEa3Ro9llC/.og/at2...	uEbYD8yKK
<input type="checkbox"/>	3	Mathew Jast	trev69@example.net	2022-03-20 22:09:53	\$2y\$10\$92IXUNpkj0rOQ5byMi.Ye4oKoEa3Ro9llC/.og/at2...	qBliUXSDro
<input type="checkbox"/>	4	Prof. Jaiden Bernier	fgoyette@example.com	2022-03-20 22:09:53	\$2y\$10\$92IXUNpkj0rOQ5byMi.Ye4oKoEa3Ro9llC/.og/at2...	jz9doa3cge
<input type="checkbox"/>	5	Mr. Jayme Powlowski II	alden96@example.com	2022-03-20 22:09:53	\$2y\$10\$92IXUNpkj0rOQ5byMi.Ye4oKoEa3Ro9llC/.og/at2...	zOq9FIZ32M
<input type="checkbox"/>	6	Mr. Henry McDermott	lind.bartholome@example.org	2022-03-20 22:09:53	\$2y\$10\$92IXUNpkj0rOQ5byMi.Ye4oKoEa3Ro9llC/.og/at2...	Y1CXekCzs8
<input type="checkbox"/>	7	Joannie Wilms	drobel@example.org	2022-03-20 22:09:53	\$2y\$10\$92IXUNpkj0rOQ5byMi.Ye4oKoEa3Ro9llC/.og/at2...	Jt4X9NcGz
<input type="checkbox"/>	8	Vickie				
<input type="checkbox"/>	9					
<input type="checkbox"/>	10					

27. Agora que a tabela tem 10 usuários você pode inserir 2 Drops do usuário 1 diretamente pelo phpMyAdmin apenas para fazermos um teste. Na tabela drops insira 2 drops para o usuário 1. Veja exemplo abaixo:

user_id bigint(20) unsigned Weldon Kovacek - 1

content varchar(255)

created_at timestamp ☒

updated_at timestamp ☒

Executar

id bigint(20) unsigned

user_id bigint(20) unsigned 1 - Weldon Kovacek

content varchar(255)

created_at timestamp ☒

updated_at timestamp ☒

28. Modifique o código do componente ShowDrops conforme abaixo:

```
<?php
namespace App\Http\Livewire;

use Livewire\Component;
use Livewire\WithPagination;
use App\Models\Drop;

class ShowDrops extends Component
{
    public $content = 'Olá Mind Drops!';

    public function render()
    {
        $drops = Drop::with('user')->get();
        return view('livewire.show-drops', ['drops' => $drops]);
    }
}
```

OBSERVAÇÃO: a linha `$drops = Drop::with('user')->get();` faz um select dos drops do usuário logado.

29. Modifique o código da view show-drops.blade.php conforme a seguir:

```
<div>
  <h3>Mind Drops </h3>
  <hr>
  @foreach($drops as $drop)
    {{ $drop->user->name }} - {{ $drop->content }}
  @endforeach
</div>
```

30. Acesse a rota e veja que a página exibe os drops. Esse início serviu apenas para uma pequena introdução ao Livewire, os próximos passos vão tratar da construção deste projeto.

31. **INSTALAÇÃO DO JETSTREAM** Esse pacote vai ser necessário para trabalhar com recursos de login e usuário e identificar curtidas e postagens por usuário. Digite no terminal: **composer require laravel/jetstream**
Esse comando instala o pacote jetstream no seu projeto.

32. No terminal digite: **php artisan jetstream:install livewire**
Esse comando vai instalar alguns recursos do jetstream para o livewire como configurações de variáveis de seções e também vai processar algumas migrations necessárias.

33. No terminal digite: **npm install** para instalar os recursos de javascript utilizados pelo jetstream.

34. No terminal digite **npm run dev**

35. No terminal digite o comando para rodar as migrations criadas pelo jetstream:
php artisan migrate

36. Modifique a rota principal em routes>web.php para:

```
Route::get('/', function () {
    return view('auth.login');
});
```

Dessa forma o endereço principal do app vai solicitar um login.

37. Acrescente no início do arquivo de rotas o carregamento do componente ShowDrops

```
use App\Http\Livewire\{
    ShowDrops
};
```


38. Modifique a rota dos drops conforme abaixo:

```
Route::get('drops', ShowDrops::class)->middleware('auth');
```

39. No navegador tente acessar **localhost:8000/drops**

Observe que você foi redirecionado para a tela de login, isso significa que essa rota só será acessada se o usuário estiver logado.

40. Código do model User. Adicione os métodos abaixo antes do caractere “}” de fechamento da classe para definir os relacionamentos da tabela users:

```
public function drops()  
{  
    return $this->hasMany(Drop::class);  
}  
  
public function likes()  
{  
    return $this->hasMany(Like::class);  
}
```

41. Código do model Drop (observe os comentários):

```
<?php  
  
namespace App\Models;  
use Illuminate\Database\Eloquent\Factories\HasFactory;  
use Illuminate\Database\Eloquent\Model;  
  
class Drop extends Model  
{  
    use HasFactory;  
  
    //campos "preenchíveis"  
    protected $fillable=['content', 'user_id'];  
  
    //Relacionamento com a tabela de usuários  
    public function user()  
    {  
        return $this->belongsTo(User::class);  
    }  
  
    //Relacionamento com a tabela de likes  
    public function likes()  
    {  
        //retorna as curtidas apenas do usuário autenticado - se estiver autenticado  
        return $this->hasMany(Like::class)->where(function($query){
```

```

        if(auth()->check()){
            $query->where('user_id', auth()->user()->id);
        }
    });
}
}

```

42. Código do componente ShowDrops em **app>Http>Livewire>ShowDrops.php**
(veja os comentários)

```

<?php
namespace App\Http\Livewire;

use Livewire\Component;
use Livewire\WithPagination;
use App\Models\Drop;

class ShowDrops extends Component
{
    //Evita o reload da página ao usar paginação
    use WithPagination;

    public $content = 'Olá Mind Drops!';

    //validação
    protected $rules = [
        'content' => 'required|min:3|max:255'
    ];

    public function render()
    {
        //Retorna os registros do usuário com paginação de 5 - mais novos para
        antigos
        $drops = Drop::with('user')->latest()->paginate(5);
        return view('livewire.show-drops', ['drops' => $drops]);
    }
}

```

```

//Método para gravar um Drop no banco
public function create()
{
    $this->validate();

    //Insere o conteúdo de acordo com o relacionamento
    //definido do método drops() do model user
    auth()->user()->drops()->create(
        [
            'content' => $this->content,
        ]
    );

    $this->content = "";
}

//Método para gravar curtidas do BD
public function like($idDrop)
{
    $drop = Drop::find($idDrop);

    $drop->likes()->create([
        'user_id' => auth()->user()->id
    ]);
}

//Método para remover curtidas do BD
public function unlike(Drop $drop)
{
    $drop->likes()->delete();
}
}

```

43. Código da view resources>views>livewire>show-drops.blade.php

```

<div>
    <h3>Mind Drops </h3>

    <form wire:submit.prevent="create">
        <input type="text" name="content" id="content" wire:model="content">
        <?php //Mensagem de erro da validação feita no componente ShowDrops ?>
        @error('content') {{ $message }} @enderror
    </form>
</div>

```

```

        <button type="submit">Criar um Drop</button>
    </form>
    <hr>

    @if(count($drops) == 0)
        <p>Nenhum Drop para ser mostrado.</p>
    @else
        @foreach($drops as $drop)
            <p>{{ $drop->user->name }} - {{ $drop->content }} &nbsp;
                @if($drop->likes->count())
                    <?php // link com databind com o método unlike(envia o id do drop) do
componente ShowDrops ?>
                    <a href="#" wire:click.prevent="unlike( {{ $drop->id }} )" <i
class="fa-solid fa-heart"></i></a>
                @else
                    <?php // link com databind com o método like(envia o id do drop) do
componente ShowDrops ?>
                    <a href="#" wire:click.prevent="like( {{ $drop->id }} )" <i
class="fa-regular fa-heart"></i></a>
                @endif
            </p>
        @endforeach

        <?php //Gera os links de paginação - definidos no componente ShowDrops
?>
        {{ $drops->links() }}
    @endif
</div>

```

44. Alterar a rota padrão após a autenticação em config>fortify.php comentar o padrão e alterar para a rota drops conforme exemplo abaixo:

```

/* 'home' => RouteServiceProvider::HOME, */
'home' => 'drops',

```

Essa definição estabelece qual rota será acessada quando o usuário fizer o login.

45. Códigos dos models e migrations para as tabelas drops e likes

MODEL LIKE

```

<?php
namespace App\Models;

```

```

use Illuminate\Database\Eloquent\Factories\HasFactory;

```

```

use Illuminate\Database\Eloquent\Model;

class Like extends Model
{
    use HasFactory;

    //campos "preenchíveis"
    protected $fillable = ['user_id','drop_id'];

    //Relacionamento com a tabela de usuários
    public function user()
    {
        return $this->belongsTo(User::class);
    }

    //Relacionamento com a tabela de drops
    public function drop()
    {
        return $this->belongsTo(Drop::class);
    }
}

MODEL DROP
<?php
namespace App\Models;
use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Drop extends Model
{
    use HasFactory;
    //campos "preenchíveis"
    protected $fillable=['content', 'user_id'];

    //Relacionamento com a tabela de usuários
    public function user()
    {
        return $this->belongsTo(User::class);
    }

    //Relacionamento com a tabela de likes
    public function likes()

```

```

{
    //retorna as curtidas apenas do usuário autenticado - se estiver autenticado
    return $this->hasMany(Like::class)->where(function($query){
        if(auth()->check()){
            $query->where('user_id', auth()->user()->id);
        }
    });
}
}

```

MIGRATION DA TABELA drops - método up()

```

public function up()
{
    Schema::create('drops', function (Blueprint $table) {
        $table->id();
        $table->unsignedBigInteger('user_id');
        $table->string('content');
        $table->timestamps();
        $table->foreign('user_id')
            ->references('id')
            ->on('users')
            ->onDelete('cascade');
    });
}

```

MIGRATION DA TABELA likes - método up()

```

public function up()
{
    Schema::create('likes', function (Blueprint $table) {
        $table->id();
        $table->unsignedBigInteger('user_id');
        $table->foreign('user_id')->references('id')->on('users');
        $table->unsignedBigInteger('drop_id');
        $table->foreign('drop_id')->references('id')->on('drops');
        $table->timestamps();
    });
}

```

46. Código de routes>web.php

```

<?php
use Illuminate\Support\Facades\Route;
use App\Http\Livewire\{
    ShowDrops
};

```

```
use App\Http\Livewire\User\UploadPhoto;
```

```
Route::get('/', function () {  
    return view('auth.login');  
});
```

```
Route::get('drops', ShowDrops::class)->middleware(['auth:sanctum', 'verified'])  
    ->name('drops');
```

```
Route::middleware(['auth:sanctum', 'verified'])->get('/dashboard', function () {  
    return view('dashboard');  
})->name('dashboard');
```

```
Route::middleware(['auth:sanctum', 'verified'])->get('/upload',  
UploadPhoto::class)->name('upload');
```

PROCEDIMENTO PARA CLONAR PROJETO LARAVEL DO GITHUB

- 1- Faça o clone do seu projeto na pasta xampp/htdocs
- 2- Pelo phpMyAdmin do Xampp, crie um banco de dados para o seu projeto.
- 3- Abra a pasta do projeto no Visual Code.
- 4- Faça uma CÓPIA do arquivo .env.example e renomeie este arquivo para .env
- 5- No arquivo .env coloque o nome do seu app e o nome do seu banco de dados.
- 6- No terminal do Visual Code digite `composer install`
- 7- No terminal do Visual Code digite `npm install`
- 8- No terminal do Visual Code digite `php artisan key:generate`
- 9- No terminal do Visual Code digite `php artisan migrate`
este comando vai gerar as tabelas do banco de dados totalmente vazias.
- 10- No terminal do Visual Code digite `php artisan serve`
e teste o seu projeto, você vai precisar criar um usuário novamente

Depois desse procedimento você pode continuar trabalhando normalmente no seu computador e salvar as alterações do projeto no github.

PUBLICAÇÃO NA INTERNET

Você deve ter acesso ao SSH do servidor que é o correspondente ao CMD do seu computador local, isso será necessário para você fazer as instalações e as migrations através dos comandos do composer e php artisan.

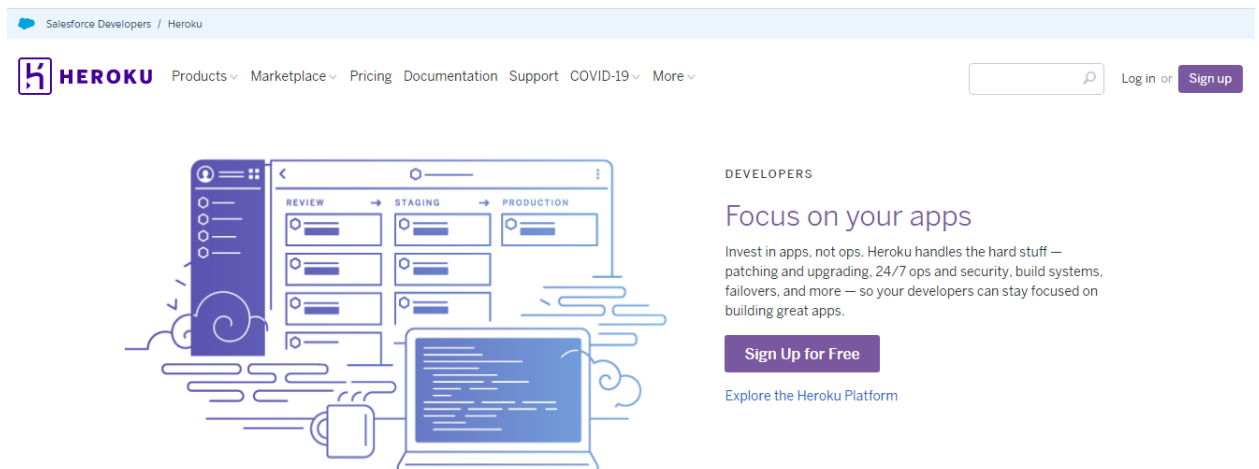
Esse artigo mostra como fazer isso em um servidor compartilhado:

<https://suporte.hostgator.com.br/hc/pt-br/articles/115004145214-Como-instalar-o-Laravel->

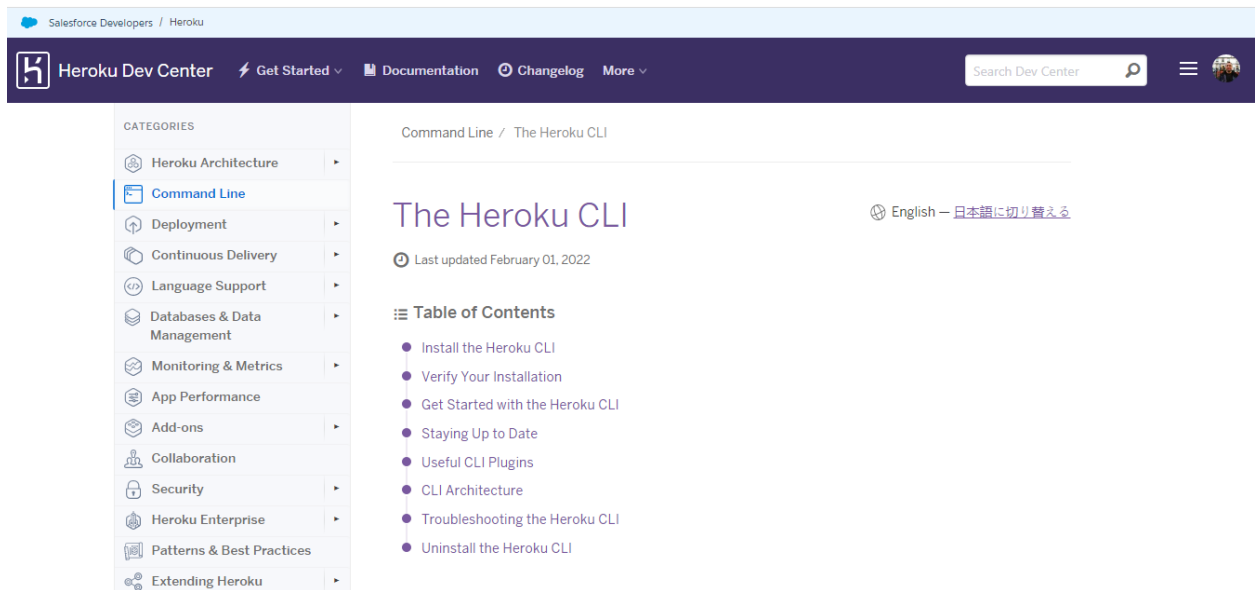
Publicando no Heroku

Heroku é uma plataforma de nuvem como serviço que suporta várias linguagens de programação com um custo bem baixo. Uma das primeiras plataformas em nuvem, o Heroku está em desenvolvimento desde junho de 2007, quando suportava apenas a linguagem de programação Ruby, mas agora suporta Java, Node.js, Scala, Clojure, Python, PHP e Go. É possível publicar um site utilizando banco de dados e outros recursos gratuitamente, você só começa a pagar quando os serviços precisarem de máquinas mais potentes. Para usar você precisa ter uma conta no GITHUB senão é impossível utilizar. A seguir veja um passo a passo para publicar o projeto “Diz Ai” no Heroku.

1. Acesse o site do Heroku para criar uma conta.



2. Faça o download do Heroku CLI, necessário para as operações.



3. Após a instalação, digite heroku no CMD, se tudo estiver certo, o help do Heroku CLI será exibido.

```
Prompt de Comando
VERSION
heroku/7.52.0 win32-x64 node-v14.15.3

USAGE
$ heroku [COMMAND]

COMMANDS
access          manage user access to apps
addons          tools and services for developing, extending, and operating your app
apps            manage apps on Heroku
auth           check 2fa status
authorizations  OAuth authorizations
autocomplete    display autocomplete installation instructions
buildpacks      scripts used to compile apps
certs           a topic for the ssl plugin
ci              run an application test suite on Heroku
clients         OAuth clients on the platform
config          environment variables of apps
container       Use containers to build and deploy Heroku apps
domains         custom domains for apps
drains          forward logs to syslog or HTTPS
features        add/remove app features
git             manage local git repository for app
help            display help for heroku
keys            add/remove account ssh keys
labs            add/remove experimental features
local           run Heroku app locally
logs            display recent log output
maintenance     enable/disable access to app
members         manage organization members
```

4. Na pasta do projeto digite:
git init
git add .
git commit -m "Upload inicial"

5. No CMD digite:

heroku login

Será solicitado o seu login e senha do Heroku.

```
C:\Windows\System32\cmd.exe
Microsoft Windows [versão 10.0.19042.685]
(c) 2020 Microsoft Corporation. Todos os direitos reservados.

C:\xampp\htdocs\dizai>git init
Reinitialized existing Git repository in C:/xampp/htdocs/dizai/.git/

C:\xampp\htdocs\dizai>git add .

C:\xampp\htdocs\dizai>git commit -m "Deploy para o Heroku"
On branch master
Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean

C:\xampp\htdocs\dizai>heroku login
» Warning: heroku update available from 7.52.0 to 7.59.3.
heroku: Press any key to open up the browser to login or q to exit:
Opening browser to https://cli-auth.heroku.com/auth/cli/browser/3317cecf-0de8-4f30-8899-3aec29adac0?requestor=SFMyNTY.g
2gDbQAAAA4yMDEuNzUuMTg4LjE2Mm4GALCB8Fd_AWIAAVGA.B8K4-OY2XBPsEHeULy38SqcKgKvmFwMhmCrthWH3_Y0
Logging in... done
Logged in as marco.carneiro7@gmail.com

C:\xampp\htdocs\dizai>
```

O Heroku cli estabeleceu uma conexão do repositório git com a sua conta.

6. Digite **heroku create**

```
C:\xampp\htdocs\dizai>git add .

C:\xampp\htdocs\dizai>git commit -m "Deploy para o Heroku"
On branch master
Your branch is up to date with 'origin/master'.

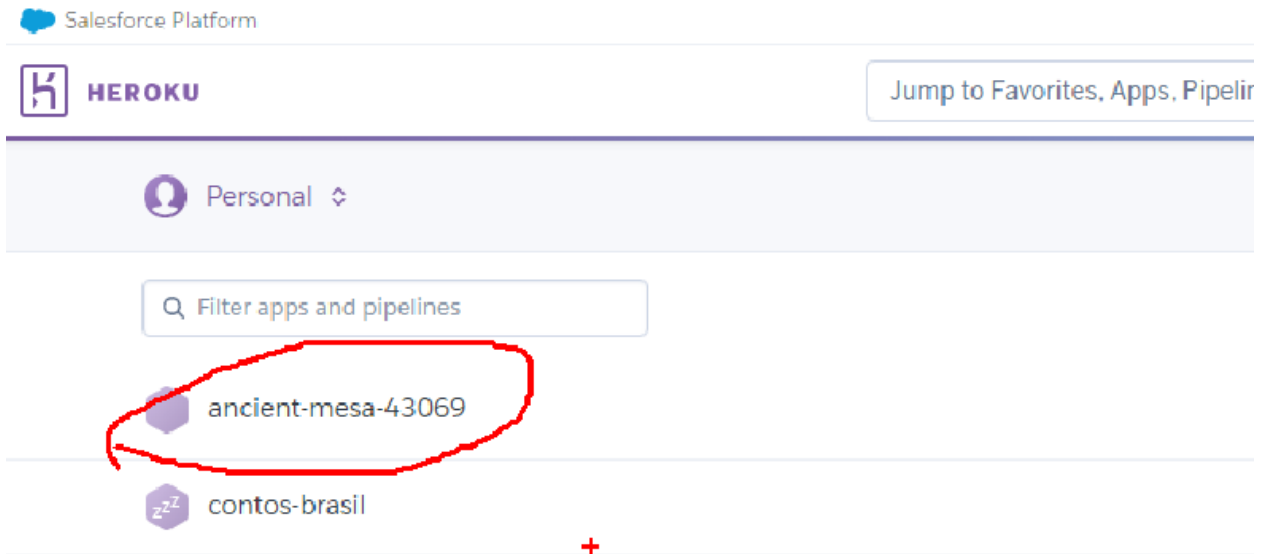
nothing to commit, working tree clean

C:\xampp\htdocs\dizai>heroku login
» Warning: heroku update available from 7.52.0 to 7.59.3.
heroku: Press any key to open up the browser to login or q to exit:
Opening browser to https://cli-auth.heroku.com/auth/cli/browser/3317cecf-0de8-4f30-8899-3aec29adac
2gDbQAAAA4yMDEuNzUuMTg4LjE2Mm4GALCB8Fd_AWIAAVGA.B8K4-OY2XBPsEHeULy38SqcKgKvmFwMhmCrthWH3_Y0
Logging in... done
Logged in as marco.carneiro7@gmail.com

C:\xampp\htdocs\dizai>heroku create
» Warning: heroku update available from 7.52.0 to 7.59.3.
Creating app... done, ⬢ ancient-mesa-43069
https://ancient-mesa-43069.herokuapp.com/ | https://git.heroku.com/ancient-mesa-43069.git

C:\xampp\htdocs\dizai>
```

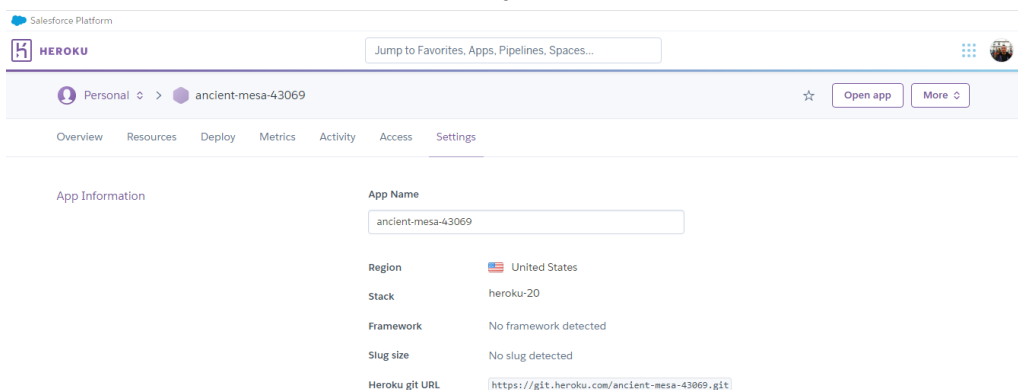
O heroku criou um novo app com base no repositório do github, o trecho destacado é a URL do seu projeto. Acesse o seu dashboard no Heroku e veja que esse projeto é adicionado.

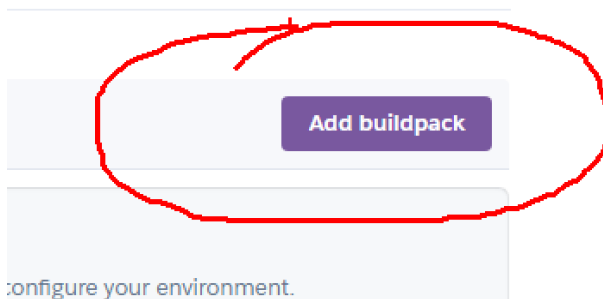


O novo projeto no dashboard do Heroku.

O Heroku analisa o arquivo `composer.json` na raiz da pasta e consegue determinar que o projeto é baseado em PHP, dessa forma o ambiente criado é um ambiente PHP. Entretanto, muitos projetos Laravel utilizam o Node também, nesse caso será necessário configurar o ambiente como multiplataforma, ou seja, um ambiente tanto para PHP quanto Node, para isso será necessário configurar o heroku com 2 buildpacks.

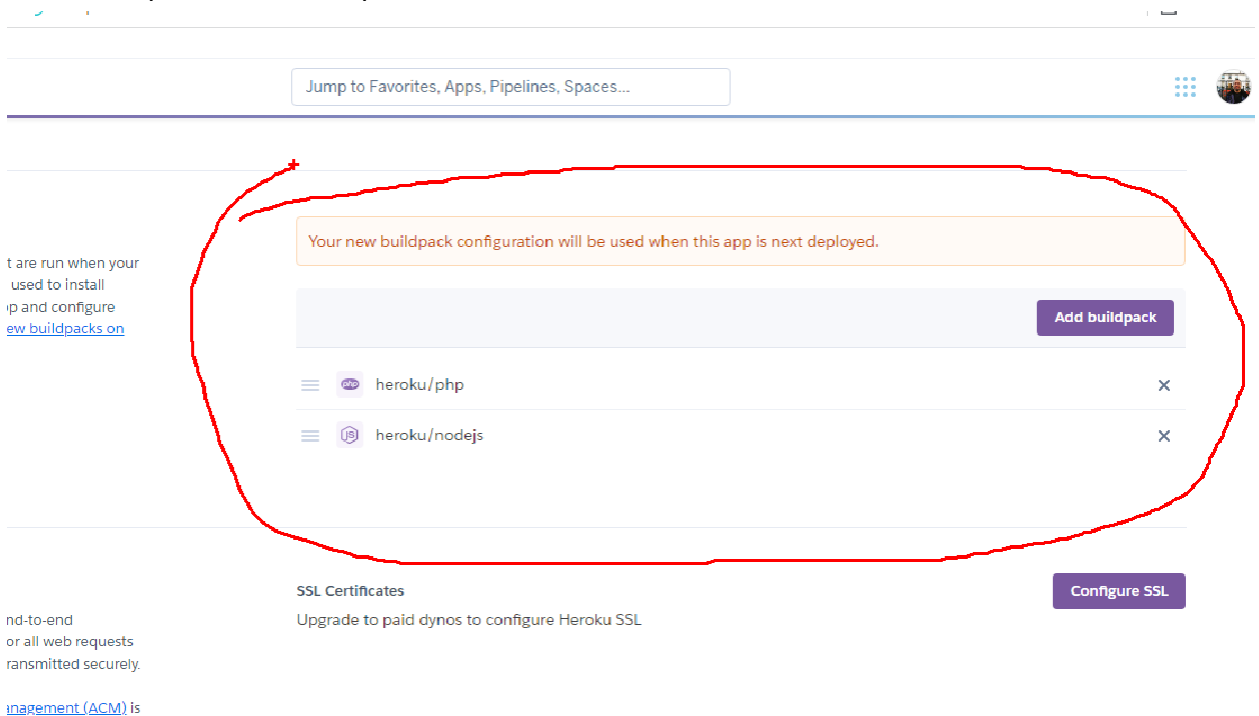
7. No dashboard do Heroku clique no projeto, os detalhes serão exibidos.





Clique no botão “add buildpack”.

8. Adicione o pacote PHP e o pacote Node.

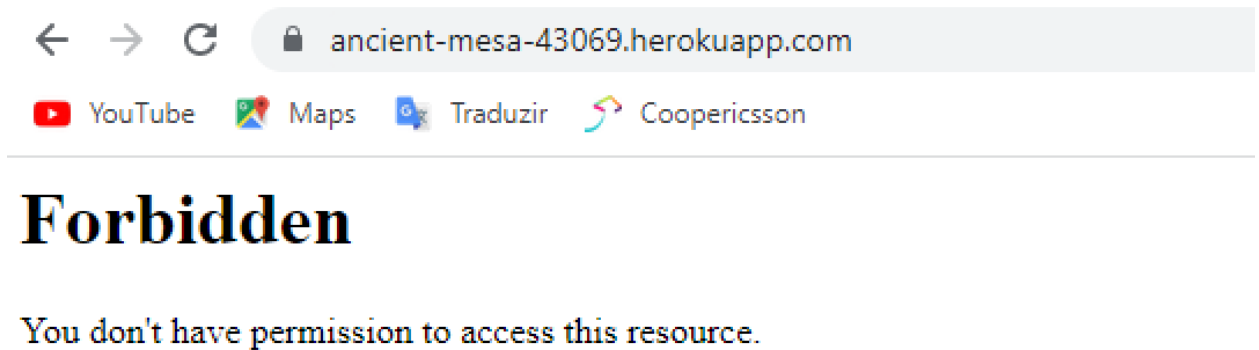


Os buildpacks PHP e Node adicionados ao ambiente.

9. No CMD digite: **git push heroku master**

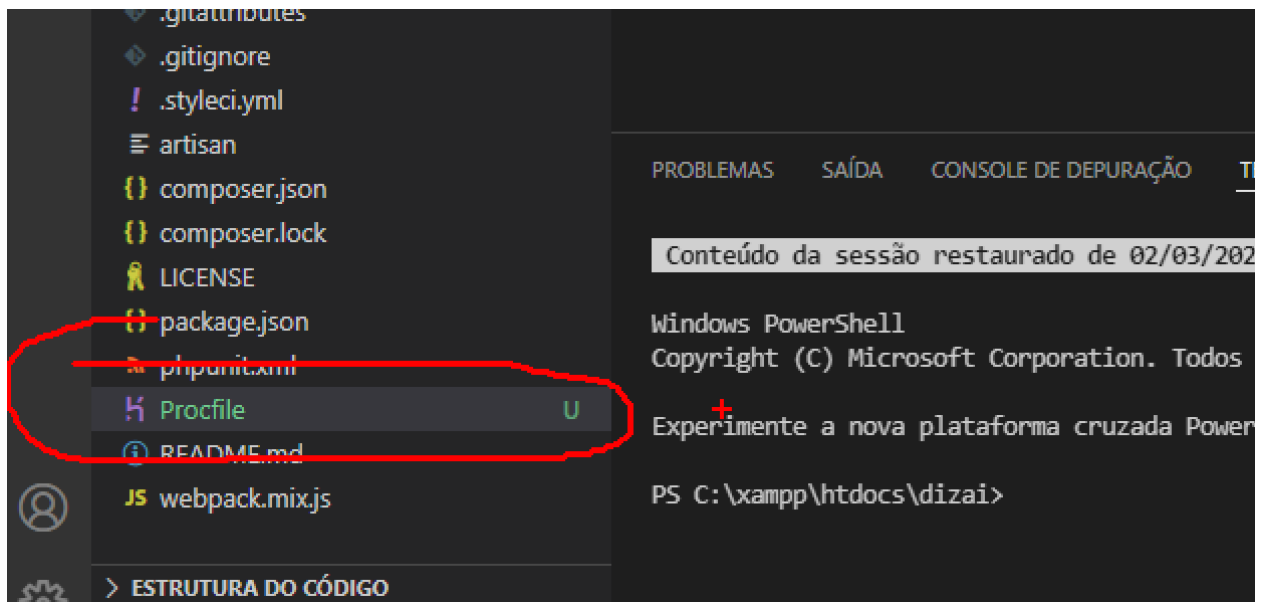
Nesse momento os arquivos vão subir para o Heroku e ao mesmo tempo o composer, o node e todas as dependências necessárias também serão instaladas.

10. Ao acessar a URL uma mensagem de “Forbidden” será exibida.



Faltam mais alguns detalhes para implementar como o tipo de servidor, a pasta raiz, banco de dados etc.

11. Para especificar que o servidor será do tipo apache - necessário para executar o PHP - você precisará criar um arquivo com nome “Procfile”. Dentro da pasta do projeto, utilizando o Visual Code, crie um arquivo com o nome “Procfile”.



Observe que o Visual Code indica que esse arquivo é do Heroku, veja o logotipo à esquerda do nome.

12. Dentro desse arquivo escreva o seguinte código:

web: vendor/bin/heroku-php-apache2 public/

Esse código determina que o servidor será o apache 2 e a pasta raiz do projeto é a

pasta “public” que é um padrão do Laravel.

13. No CMD digite os comandos abaixo para fazer um novo commit e enviar o arquivo para o Heroku:

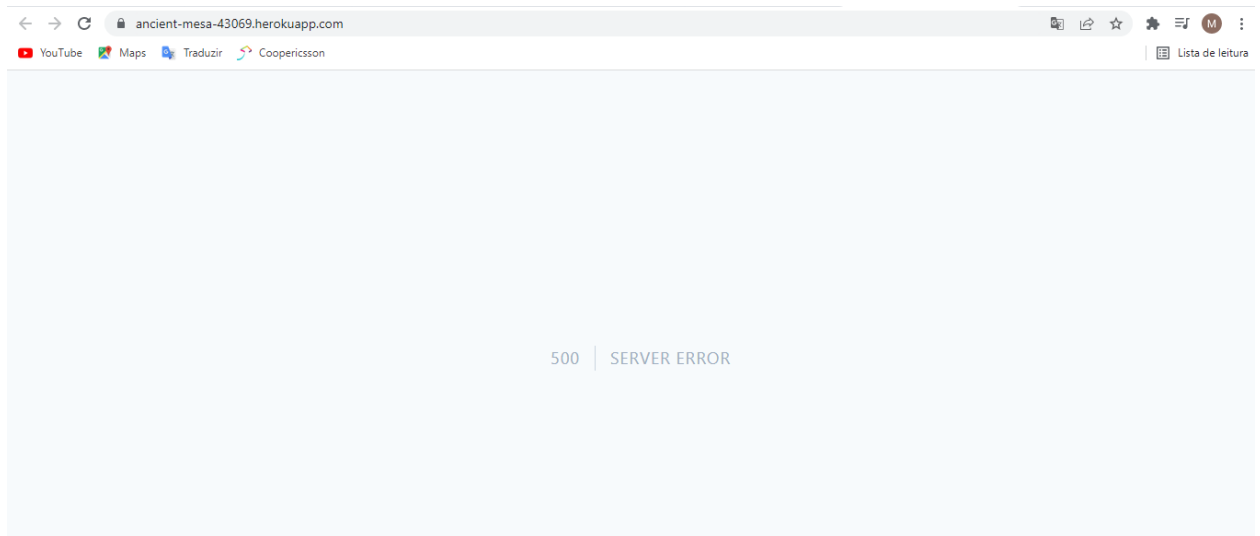
git add . Procfile

git commit -m "Upload do Procfile"

git push heroku master

Após isso, todas as dependências serão atualizadas e o ambiente será preparado.

14. Acesse novamente a URL do projeto, veja que agora um erro 500 é exibido, veja que é uma tela que segue o padrão do Laravel.



15. É necessário especificar no arquivo **.env** um banco de dados válido e também a URL do projeto no Heroku. Por padrão o Heroku utiliza o banco de dados Postgres.

No CMD digite:

heroku addons:create heroku-postgresql:hobby-dev

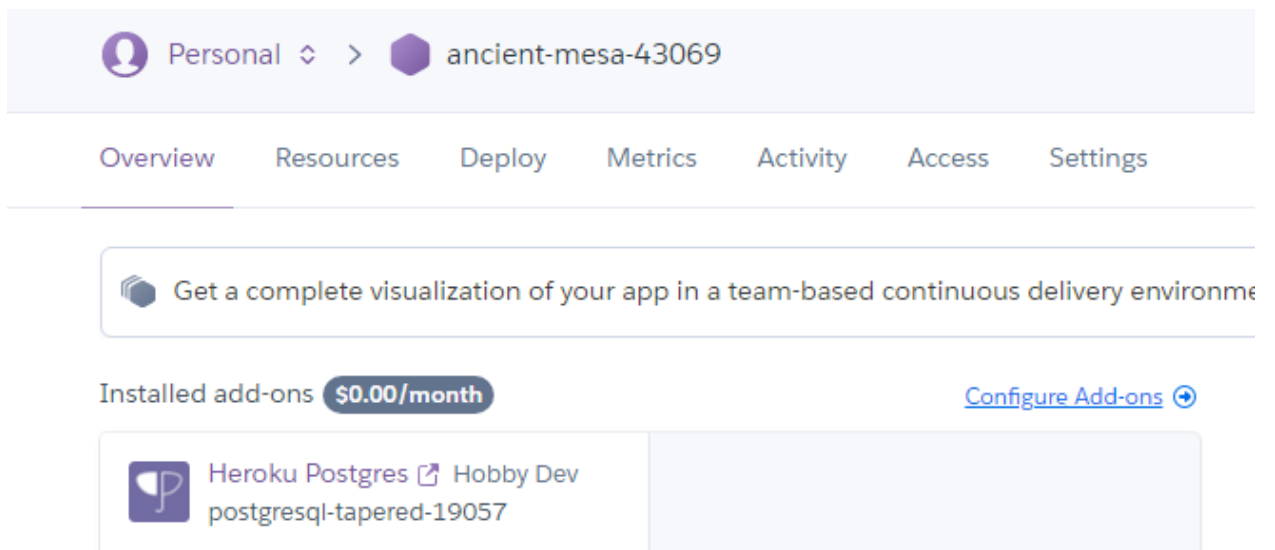
```
C:\Windows\System32\cmd.exe

notifications  display notifications
orgs           manage organizations
pg            manage postgresql databases
pipelines     manage pipelines
plugins       list installed plugins
ps            Client tools for Heroku Exec
psql          open a psql shell to the database
redis         manage heroku redis instances
regions       list available regions for deployment
releases      display the releases for an app
reviewapps    manage reviewapps in pipelines
run           run a one-off process inside a Heroku dyno
sessions      OAuth sessions
spaces        manage heroku private spaces
status        status of the Heroku platform
teams         manage teams
update        update the Heroku CLI
webhooks      list webhooks on an app

C:\xampp\htdocs\dizai>heroku addons:create heroku-postgresql:hobby-dev
» Warning: heroku update available from 7.52.0 to 7.59.3.
Creating heroku-postgresql:hobby-dev on ancient-mesa-43069... free
Database has been created and is available
! This database is empty. If upgrading, you can transfer
! data from another database with pg:copy
Created postgresql-tapered-19057 as DATABASE_URL
Use heroku addons:docs heroku-postgresql to view documentation

C:\xampp\htdocs\dizai>
```

16. Acesse o dashboard do projeto e clique no recurso de postgres.



17. Clique em Settings>View Credenciais

Os dados do seu banco de dados serão exibidos

ADMINISTRATION

Database Credentials

Get credentials for manual connections to this database.

Cancel

Please note that **these credentials are not permanent**.

Heroku rotates credentials periodically and updates applications where this database is attached.

Host	ec2-3-230-238-86.compute-1.amazonaws.com
Database	d436k19lmvrnri
User	taoonxzvslvvza
Port	5432
Password	be0f4b20d78d5f8e6ffbb3994ebd2e9f4eb1584e8a277f27fd529c187591bc6e
URI	postgres://taoonxzvslvvza:be0f4b20d78d5f8e6ffbb3994ebd2e9f4eb1584e8a277f27fd529c187591bc6e@ec2-3-230-238-86.compute-1.amazonaws.com:5432/d436k19lmvrnri
Heroku CLI	heroku pg:psql postgresql-tapered-19057 --app ancient-mesa-43069

18. Abra o arquivo .env pelo Visual Code e faça as seguintes alterações:
Em APP_URL copie e cole a URL do seu projeto.

```
APP_NAME=Dizai
APP_ENV=local
APP_KEY=base64:E5cIrYLVSWaaSwW7BAGrYBDNwq3AdNv/hxq+1x+AivU=
APP_DEBUG=true
APP_URL=https://ancient-mesa-43069.herokuapp.com/
```

19. Modifique as informações do banco de dados conforme o dashboard.

```
10
11 DB_CONNECTION=pgsql
12 DB_HOST=ec2-3-230-238-86.compute-1.amazonaws.com
13 DB_PORT=5432
14 DB_DATABASE=d436k19lmvrnri
15 DB_USERNAME=taoonxzvslvvza
16 DB_PASSWORD=be0f4b20d78d5f8e6ffbb3994ebd2e9f4eb1584e8a277f27fd529c187591bc6e
17
```

20. No CMD digite os seguintes comandos para atualizar os arquivos no servidor:

git add .env -f

git add .

git commit -m "Implementação do banco de dados"

git push heroku master

O processo de atualização do ambiente vai acontecer novamente.

21. Acesse novamente a URL do projeto, veja que agora um erro de consulta ao banco de dados é exibido.

Symfony Exception

PDOException > QueryException

HTTP 500 Internal Server Error

SQLSTATE[42P01]: Undefined table: 7 ERROR: relation "comentarios" does not exist
LINE 1: select * from "comentarios" order by "created_at" desc
^ (SQL: select * from "comentarios" order by "created_at" desc)

Exceptions 2 Stack Traces 2

Illuminate\Database\QueryException

- in /app/vendor/laravel/framework/src/Illuminate/Database/Connection.php (line 712)
- in /app/vendor/laravel/framework/src/Illuminate/Database/Connection.php -> runQueryCallback (line 672)
- in /app/vendor/laravel/framework/src/Illuminate/Database/Connection.php -> run (line 376)
- in /app/vendor/laravel/framework/src/Illuminate/Database/Query/Builder.php -> select (line 2513)
- in /app/vendor/laravel/framework/src/Illuminate/Database/Query/Builder.php -> runSelect (line 2501)
- in /app/vendor/laravel/framework/src/Illuminate/Database/Query/Builder.php -> Illuminate\Database\Query\{closure} (line 3035)
- in /app/vendor/laravel/framework/src/Illuminate/Database/Query/Builder.php -> onceWithColumns (line 2502)
- in /app/vendor/laravel/framework/src/Illuminate/Database/Eloquent/Builder.php -> get (line 632)
- in /app/vendor/laravel/framework/src/Illuminate/Database/Eloquent/Builder.php -> getModels (line 616)

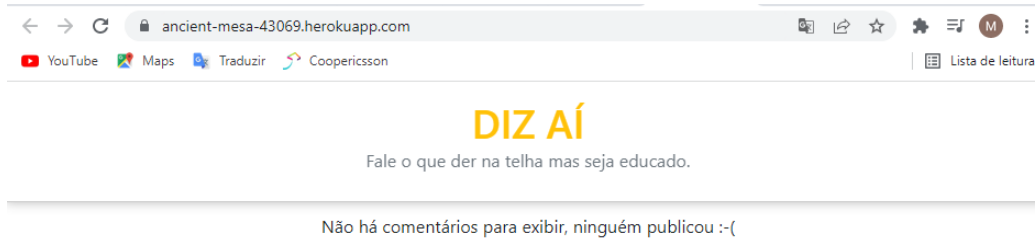
Isso significa que a nossa aplicação está funcionando, porém as tabelas não foram encontradas. Falta rodar as **migrations** para que as tabelas sejam criadas.

22. Digite no CMD:

heroku run "php artisan migrate"

Perceba que é possível executar qualquer instrução do composer e do artisan diretamente no servidor Heroku da seguinte forma: heroku run "instrução"

23. Acesse novamente o site, veja que agora está funcionando perfeitamente.



Faça os ajustes finais e teste a publicação online.

Uma última dica: se após a publicação, o site não gerar links internos com HTTPS

Então siga o procedimento deste artigo na documentação do Laravel, que o problema estará solucionado: <https://laravel.com/docs/9.x/requests#configuring-trusted-proxies>

Veja aqui um tutorial sobre o deploy no Heroku

<https://www.youtube.com/watch?v=kY4u39a6Ueg>

DICAS DIVERSAS

FUNDAMENTOS BÁSICOS

- A pasta “public” contém o site renderizado, esta é a pasta que deverá ser acessada para visualizar o projeto e também para armazenar arquivos estáticos como CSS e JS.
- O arquivo routes/web contém as rotas do projeto, ou seja, nesse arquivo você define qual a primeira view que será carregada ou qual o controller principal
O arquivo .env contém os dados os dados críticos do desenvolvimento como banco de dados e outras informações que não deverão ser compartilhadas em ferramentas de versionamento como o github

- Não remova o arquivo env.example da raiz do seu projeto, ele é utilizado para a criação de um arquivo .env em projetos clonados no github. O arquivo .env contém as informações de banco de dados e da chave para o projeto, por isso esse arquivo não é gerenciado pelo github. O arquivo env.example serve para criar um novo arquivo .env
- Para iniciar um projeto Laravel que foi clonado do GITHUB deve-se digitar no terminal dentro da pasta do projeto:
composer install ou composer install --ignore-platform-reqs
 Se for solicitado digite:
composer update ou composer update --ignore-platform-reqs
 Esse comando vai atualizar todos os pacotes PHP que forem necessários.
php artisan key:generate
 Esse comando vai gerar uma chave para o projeto no arquivo .env (obrigatório).
- Para consultar todos os recursos do artisan, digite no CMD ou terminal do Visual Code:
php artisan list
- Para que o LARAVEL grave corretamente os horários do **timestamp** você deve alterar a configuração de timezone no arquivo config>app.php para
 'timezone' => 'America/Sao_Paulo',
 E também atualizar essas configurações no servidor com
php artisan config:cache
- Siga essa instrução <https://laravel.com/docs/8.x/requests#configuring-trusted-proxies>
 Se após a publicação o Laravel não escrever **HTTPS** em todos os links.
 Abra o arquivo app>Http>Middleware>TrustProxies.php
 e altere protected \$proxies; para
protected \$proxies = '*';
 Isso indica que o Laravel vai confiar em todos os proxies do seu servidor.
- Criação de funções globais no Laravel - HELPERS
<https://pt.stackoverflow.com/questions/353879/como-criar-fun%C3%A7%C3%B5es-dispon%C3%ADveis-globalmente-no-laravel>
- **UPLOAD DE IMAGENS**
 - Crie uma pasta com nome “uploads” dentro da pasta “public” (pode ser o nome que você quiser). O código abaixo mostra como fazer um upload com novo nome de arquivo
 - insira esse código no Controller:

```
if($request->file('bgimage')){
    $randomTxt = Str::random(20);
    $file= $request->file('bgimage');
    $filename= $randomTxt.date('YmdHi').$file->getClientOriginalName();
    $file-> move(public_path('public/uploads'), $filename);
}
```

```
$pesquisa->bgimage = $filename;  
}
```

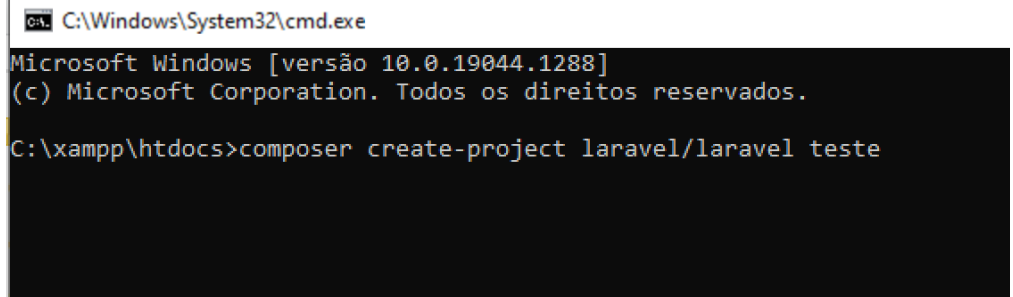
NOTA: nesse exemplo o campo de formulário e o campo no banco de dados se chama “bgimage” e a variável **\$pesquisa** é a instância do Model.

- Para testes de variáveis use **dd** ao invés de **var_dump()**, veja abaixo um exemplo de teste da propriedade \$photo na action storagePhoto de um componente

```
public function storagePhoto()  
{  
    dd($this->photo);  
}
```

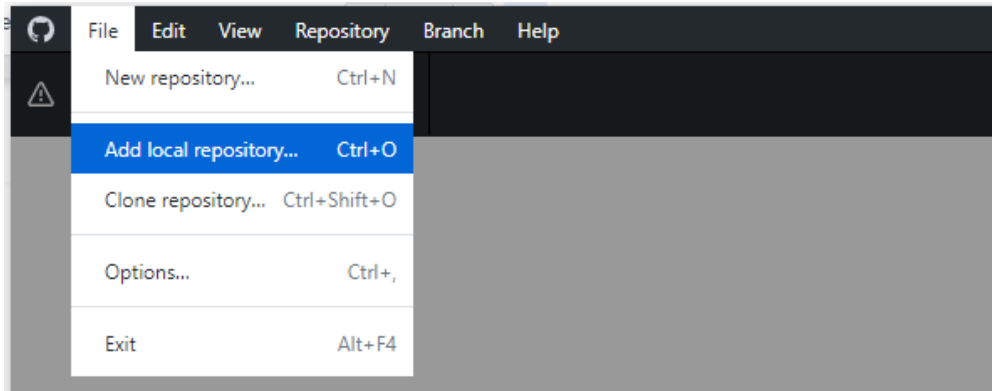
CRIANDO UM NOVO PROJETO E ADICIONANDO-O AO GITHUB

- Crie um novo projeto com a instrução do composer:
composer create-project laravel/laravel nome-projeto

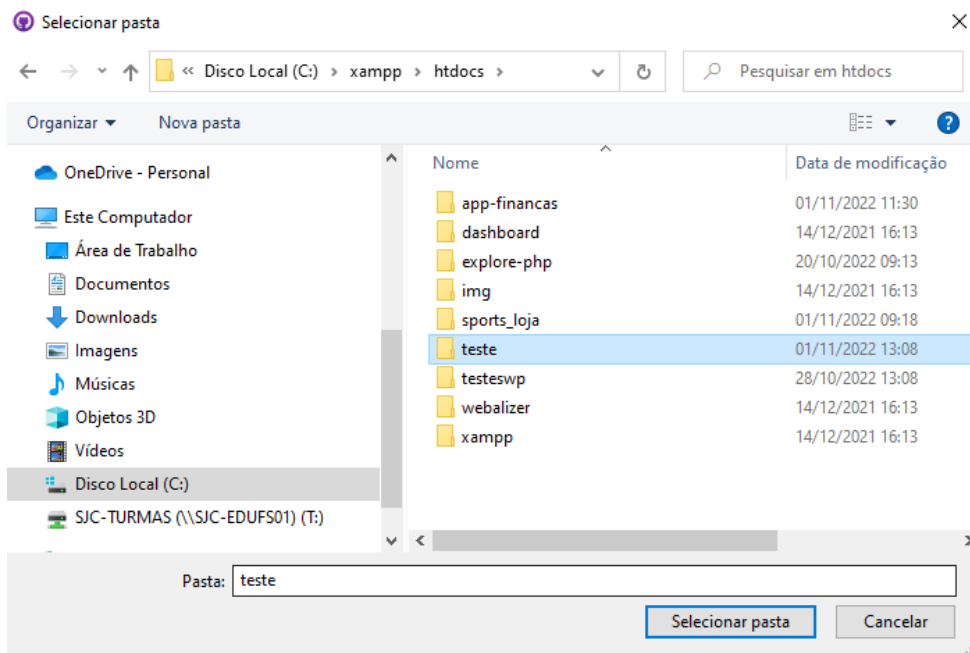


```
C:\Windows\System32\cmd.exe  
Microsoft Windows [versão 10.0.19044.1288]  
(c) Microsoft Corporation. Todos os direitos reservados.  
C:\xampp\htdocs>composer create-project laravel/laravel teste
```

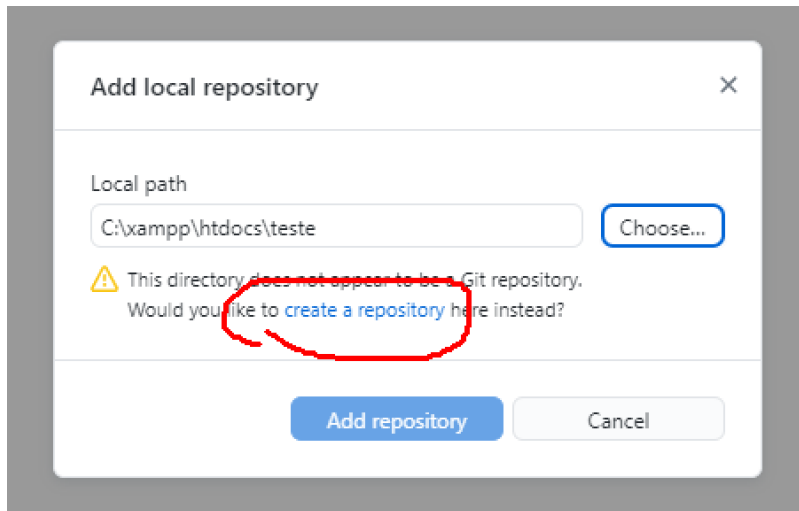
- Após a criação do projeto, abra o GITHUB Desktop. No menu File, clique em “Add local Repository”.



- Selecione a pasta criada pelo processo.

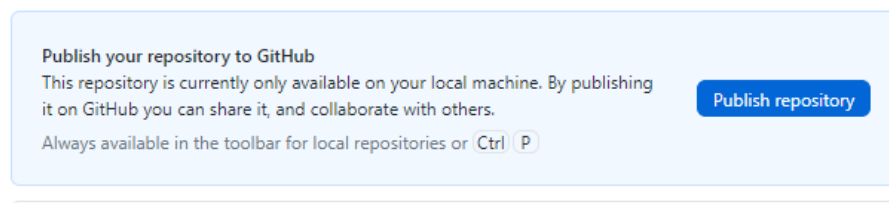


- Em seguida vai aparecer um aviso para que um repositório seja criado. Clique em “create repository”.



- Preencha o formulário para a criação no novo repositório.

- Após a criação do repositório, clique no botão “Publish repository”.



- Confirme o nome e a descrição do repositório. Se quiser que o repositório seja público, desmarque a opção “Keep this code private”.

Publish repository

GitHub.com | GitHub Enterprise

Name: teste

Description: Apenas um teste

☒ Keep this code private

Publish repository | Cancel

- Acesse o github.com e verifique se o repositório foi criado.

marcocarneiro / teste Public

Code | Issues | Pull requests | Actions | Projects | Wiki | Security | Insights | Settings

main | 1 branch | 0 tags

Go to file | Add file | Code

File/Folder	Status	Time
marcocarneiro	Initial commit	1 minute ago
app	Initial commit	1 minute ago
bootstrap	Initial commit	1 minute ago
config	Initial commit	1 minute ago
database	Initial commit	1 minute ago
lang/en	Initial commit	1 minute ago
public	Initial commit	1 minute ago
resources	Initial commit	1 minute ago
routes	Initial commit	1 minute ago
storage	Initial commit	1 minute ago
tests	Initial commit	1 minute ago
.editorconfig	Initial commit	1 minute ago
.env.example	Initial commit	1 minute ago
.gitattributes	Initial commit	1 minute ago

About

Apenas um teste

Readme

AGPL-3.0 license

0 stars

1 watching

0 forks

Releases

No releases published

Create a new release

Packages

No packages published

Publish your first package

Languages

PHP 81.3% | Blade 17.4%

CONTROLLERS

- Para criar um Controller no seu projeto digite o seguinte comando no CMD ou terminal do Visual Code:
php artisan make:controller NomeController

- **Upload de arquivos com Request.** Necessário uma View com um formulário com atributo enctype e action apontando para uma rota do tipo post que executa um método de algum controller. Trecho do controller no método que vai gravar dados no banco, nesse exemplo a instância do Model tem nome de \$movimento:

```
//upload de imagens
if($request->hasfile('imagem') && $request->file('imagem')->isValid())
{
    $requestImage = $request->imagem;
    $extensao = $requestImage->extension();
    //nome do arquivo inicia com nome do usuário
    $nomeImagem = $user->name .
md5($requestImage->getClientOriginalName()) . '.' . $extensao;

    $request->imagem->move(public_path('imagens'), $nomeImagem);
    $movimento->imagem = $nomeImagem;
}
```

PARA EXIBIR NA VIEW (variável \$receita é o objeto dentro de um @foreach):

```
<br>
```

- **CONSULTA INNER JOIN.** Utilize essa técnica quando precisar exibir os dados de uma tabela combinando com outra. No exemplo abaixo uma consulta é feita no controller retornando os dados da tabela fin_movimentos e o nome do usuário com base no campo user_id da tabela fin_movimentos, essa tabela não contém o campo name, essa informação vem da tabela users.

MÉTODO NO CONTROLLER

```
public function testeJoin()
{
    $movimentos = Fin_movimento::join('users', 'users.id', '=',
'fin_movimentos.user_id')
        ->get(['fin_movimentos.*', 'users.name']);

    return view ('teste', ['movimentos' => $movimentos]);
}
```

EXIBINDO O RESULTADO NA VIEW

```
@foreach ($movimentos as $movimento)
    <div class="card mb-5">
        <div class="card-header">
            Nome: {{ $movimento->name }}
        </div>
        <div class="card-body">
```



```

        {{$movimento->descricao}}
    </div>
</div>
@endforeach

```

Perceba que o campo **name** não existe na tabela **fin_movimentos**, essa informação vem da tabela **users** graças à técnica do **INNER JOIN**.

- **GRAVAÇÃO / LEITURA DADOS JSON.** No exemplo abaixo os dados são construídos com todos os campos de Request exceto os campos “_token”, “id_pesquisa”, “data_hora_inicio” e “ip”. Em seguida os dados são codificados no formato JSON e associados ao campo “dados” do Model Resultado, o termo **JSON_UNESCAPED_UNICODE** é necessário para que os textos sejam acentuados dentro do padrão UTF-8

```

$dados = $request->except(['_token', 'id_pesquisa', 'data_hora_inicio', 'ip']);
$resultado->dados = json_encode($dados, JSON_UNESCAPED_UNICODE);

```

MODELS / MIGRATIONS

- Para criar um Model para uma tabela do seu banco de dados, digite o seguinte comando no CMD:
php artisan make:model Nome-da-tabela-no-singular
- Para criar um model e o seu migration correspondente:
php artisan make:model NomeDoModel -m
- **IMPORTANTE:** Ao criar relacionamentos entre tabelas, verifique se a ordem de criação das tabelas está correta. Por exemplo, ao relacionar a tabela “post” com a tabela “user” a tabela “user” deve ser executada primeiro. Verifique o nome do arquivo de migration, se for necessário, renomeie o arquivo para garantir a ordem correta.

- Para adicionar uma nova coluna a uma tabela existente, digite no terminal do VSCODE:
php artisan make:migration add_descricao_to_nomeDaTabela_table

Os métodos up() e down() deverão ficar semelhantes ao exemplo abaixo:

```

public function up()
{
    Schema::table('nomeDaTabela', function($table) {
        $table->text('descricao');
    });
}

public function down()
{
    Schema::table('nomeDaTabela', function($table) {

```

```

        $table->dropColumn('descricao');
    });
}

```

- **Relacionamentos** : Exemplo de método up() com chave estrangeira - Relacionamentos

```

public function up()
{
    Schema::create('avaliacoes', function (Blueprint $table) {
        $table->id();
        $table->unsignedBigInteger('id_cliente');
        $table->foreign('id_cliente')->references('id')->on('clientes');
        $table->unsignedBigInteger('id_fornecedor');
        $table->foreign('id_fornecedor')->references('id')->on('fornecedores');
        $table->integer('nota');
        $table->timestamps();
    });
}

```

- **Relacionamentos** : Exemplo de relacionamento entre 2 tabelas, no caso com a tabela de usuários e a tabela de posts. A seguir o código que deve ser inserido nos respectivos models.

NO MODEL User:

```

public function posts()
{
    return $this->hasMany(Post::class);
}

```

NO MODEL Post:

```

public function user()
{
    return $this->belongsTo(User::class);
}

```

- Não esqueça de incluir as ligações dos models nos controllers e dos controllers nas rotas. Isso deve ser declarado antes das classes logo após a tag de abertura do php “<?php”.

Exemplos:

Ligando as rotas ao controller MovimentoController

```
use App\Http\Controllers\MovimentoController;
```

Ligando o Controller ao Model Fin_movimento

```
use App\Models\Fin_movimento;
```

- **Relacionamentos:** Exemplo de chave primária podendo ser NULL
`$table->unsignedBigInteger('id_opc_resposta')->nullable()->default(NULL);`
`$table->foreign('id_opc_resposta')->references('id')->on('opc_respostas');`

ROTAS

- Tutorial sobre rotas API:
<https://www.twilio.com/blog/criar-e-consumir-uma-api-restful-no-php-laravel>
- Limpar o cache das rotas:
php artisan optimize

BLADE / VIEWS

- Dica - diferentes maneiras de passar variáveis para views:
<https://www.geeksforgeeks.org/different-ways-for-passing-data-to-view-in-laravel/>
- Para exibir a URL principal do site nos arquivos blade, utilize
`{{ url('/') }}`
- Para o blade renderizar o HTML dentro de uma variável, a sintaxe deve ser `{!! $variável !!}`
IMPORTANTE: Certifique-se que códigos maliciosos não sejam renderizados.
- Exemplo de código Blade para exibir dentro de um link, a classe “active” do bootstrap, somente se a rota atual coincidir com o link (utilização de IF ternário)
` is('datas_comemorativas')) ? 'active' : " }}"`
`href="{{route('datas_comemorativas')}}">Datas Comemorativas`
- COMO UTILIZAR SLOTS
<https://spatie.be/docs/laravel-blade-x/v2/basic-usage/using-slots>
- Este link contém muito mais dicas e macetes do Blade:
<https://blog.especializati.com.br/laravel-blade/>
- **O que são componentes no blade?**
<https://medium.com/@huriellopes/componentes-no-laravel-framework-2249562d0113>

TINYMCE

- Biblioteca javascript para criação de campos de edição com muitos recursos, como formatação, inserção de tabelas, imagens, etc.
 Veja este link que mostra a instalação no Laravel:
<https://www.tiny.cloud/docs/tinymce/6/laravel-composer-install/>

LIVEWIRE / JETSTREAM

- Instalação do Livewire no projeto Laravel
composer require livewire/livewire
Este comando vai instalar o Livewire no projeto Mind Drops
php artisan livewire:publish
Cria o arquivo **livewire.php** dentro da pasta config. Este arquivo define as configurações gerais do Livewire como o caminho dos controllers e das views, etc. Abra esse arquivo apenas para observar as configurações, não é necessário modificar.
php artisan livewire:publish --assets Para publicar os arquivos de assets - css e js
Na raiz do projeto abra o arquivo **composer.json**
Inclua esse trecho de código (COM ASPAS)
"@php artisan vendor:publish --force --tag=livewire:assets --ansi"
Dentro da array **post-autoload-dump** (NÃO ESQUEÇA A VÍRGULA NO FINAL DO ITEM ANTERIOR)
- Para compilar os assets para produção deve-se digitar no terminal **npm run prod**
- Criação de um componente Laravel
php artisan livewire:make NomedoComponente
- O componente é renderizado nas views com a tag livewire, deve-se usar o mesmo nome gerado nas views, exemplo: **<livewire:nome-do-componente />**
- Para definir que rota deverá ser carregada após o login, você deve modificar o arquivo **config>fortify.php**, no exemplo abaixo a rota de nome “drops” é carregada após o login.

```
/* 'home' => RouteServiceProvider::HOME, */  
    'home' => 'drops',
```
- **DICA: ACRESCENTAR COLUNAS NA TABELA DE USUÁRIOS**
No arquivo de migração **dataHora_create_users_table.php** acrescente a nova coluna, exemplo: **\$table->string('tipo');** no model acrescente a nova coluna na array fillable, neste exemplo será adicionado o campo “tipo” para definir se o usuário é tipo administrador ou leitor. Exemplo:
protected \$fillable = [
 'name',
 'email',
 'password',
 'tipo'
];

];

No arquivo **resources>views>auth>register.blade.php** adicione o campo para a coluna, exemplo:

```
<div class="mt-4">
    <x-jet-label for="tipo" value="{{ __('Tipo de usuário') }}" />
    <select id="tipo" class="block mt-1 w-full" name="tipo" required>
        <option value="">Selecione</option>
        <option value="administrador" >Administrador</option>
        <option value="leitor" >Leitor</option>
    </select>
</div>
```

Abra o arquivo **app>fortify>CreateNewUser.php** e inclua o campo desejado na função create() conforme o exemplo abaixo (campo “tipo”):

```
Validator::make($input, [
    'name' => ['required', 'string', 'max:255'],
    'email' => ['required', 'string', 'email', 'max:255', 'unique:users'],
    'password' => $this->passwordRules(),
    'tipo' => ['required', 'string', 'max:255'],
    'terms' => Jetstream::hasTermsAndPrivacyPolicyFeature() ? ['accepted',
'required'] : "",
    ]->validate();
return User::create([
    'name' => $input['name'],
    'email' => $input['email'],
    'tipo' => $input['tipo'],
    'password' => Hash::make($input['password']),
]);
```

Faça o teste para registrar um novo usuário

- Para modificar o layout utilizado por um componente, deve-se especificar no método render() do componente conforme o exemplo:

```
public function render()
{
    return view('livewire.show-posts')
        ->layout('pasta.arquivo-blade');
}
```

Veja mais detalhes em:

<https://laravel-livewire.com/docs/2.x/rendering-components>

- EXEMPLO DE MÉTODO DO COMPONENTE PARA GRAVAR E TAMBÉM ATUALIZAR, BASEADO NO ID DO REGISTRO - propriedade "reg"

```
public function saveUpdate()
{
    $instanciaModel = new NomeDoModel;
    $instanciaModel->user_id = auth()->user()->id;
    $instanciaModel->titulo = $this->titulo;
    $instanciaModel->slug = Str::slug($this->titulo, '-');
    if ($this->reg == 0) {
        $instanciaModel->save();
        $this->reg = $instanciaModel->id;
    } else {
        NomeDoModel::find($this->reg)->update([
            'titulo' => $this->titulo,
            'slug' => Str::slug($this->titulo, '-'),
        ]);
    }
}
```

- PASSANDO PARÂMETROS PARA OUTRO COMPONENTE

O exemplo a seguir envia o parâmetro \$reg para o componente 'show-perguntas' somente quando este for atualizado, ou seja, maior que 0 que é o valor inicial no componente pai.

```
@if($reg > 0)
    <livewire:show-perguntas :reg="$reg">
@endif
```

O componente filho deverá ter uma propriedade pública com nome "reg"

Veja mais detalhes em:

<https://laravel-livewire.com/docs/2.x/rendering-components>

- Veja esse artigo sobre componentes clonados/aninhados:
<https://gist.github.com/webdevmatics/11290b6e9ae75469f6f017741bfb679f#file-clone-for-m-blade-php>
- Artigo sugerindo uma abordagem no Livewire para formulários complexos:
<https://github.com/livewire/livewire/issues/313>
- As rotas do Jetstream mais usadas :
 - login
 - register
 - forgot-password
 - logout

- user/profile
- **ROTA user/profile** - essa rota está programada para utilizar a rota dashboard, não vai funcionar se a rota não existir no arquivo Routes > web.php
Para personalizar a aparência, modifique os arquivos de views dentro da pasta profile.
Maiores informações acesse:
<https://jetstream.laravel.com/2.x/features/profile-management.html>

LINKS IMPORTANTES

VALIDAÇÃO DE CAMPOS DO BRASIL (CNPJ, CPF, CELULAR, PIS, etc)
<https://github.com/LaravelLegends/pt-br-validator>

DIRETIVAS BLADE
<https://laravel.com/docs/8.x/blade#blade-directives>

COMO MONTAR CONSULTAS AO BANCO DE DADOS
<https://laravel.com/docs/8.x/queries>

RELACIONAMENTO DO TIPO MUITOS PARA MUITOS
Um relacionamento de muitos para muitos entre 2 tabelas vai precisar de uma terceira tabela.
Veja um artigo na documentação do Laravel sobre o assunto:
<https://laravel.com/docs/9.x/eloquent-relationships#many-to-many>
Um vídeo onde esse tópico é trabalhado
<https://www.youtube.com/watch?v=Z5xzJc9VHzY>