Laravel com VueJs

1. Baixar o Laravel em https://laravel.com
2. Instalar o composer e o PHP 7 ou se achar melhor, instalar o Xampp.
3. Comando para criar o projeto com o nome de “blog”

```
composer create-project --prefer-dist laravel/laravel blog "5.5.*"
```

4. A configuração da conexão do banco de dados é feita no arquivo “.env”

Os arquivos deste projeto podem ser baixado em: https://tinyurl.com/projIlton
Caso o projeto não contenha as pastas “vendor” e a pasta de módulos do NodeJS, será necessário executar os seguintes comandos: composer update (*para criar a pasta vendor*) e
npm install (*para criar a pasta de módulos do node js*)

Já de início podemos implementar as funcionalidades referentes a autenticação/login do usuário, executando na console o seguinte comando:

```
php artisan make:auth
```

Criando um CRUD de usuários

1 - Criar uma classe Controler com o nome de UsuariosController, na pasta Admin:

```
php artisan make:controller Admin/UsuariosController –resource
```

Será criada a classe controle com a assinatura de todos os métodos para um CRUD. Abra este arquivo e faça as seguintes alterações:
- importe a classe “User” da Model, que já vem nativa no Laravel.

```
use App\User;
```

- Implementação do método index da classe UsuariosController:

```
7   use App\User;
8
9   class UsuariosController extends Controller
10  {
11
12      public function index()
13      {
14          $listaMigalhas = json_encode([
15              ["titulo"=>"Home", "url"=>route('home')],
16              ["titulo"=>"Lista de Usuário", "url"=>""]
17              ]);
18
19
20          //transformar em Json somente na view.
21          $listaModelo = User::select('id','name','email')->paginate(10);
22
23
24          return view('admin.usuarios.index',compact('listaMigalhas','listaModelo'));
25      }
```

Aqui já implementamos o array do “bredcomb” (lista de migalhas) e o array contendo a lista de usuários já cadastrados.

Para a implementação do cadastro do usuário, digite o seguinte código no método store() desta classe.

```
42      public function store(Request $request)
43      {
44          $data = $request->all();
45          $validacao = \Validator::make($data,[
46              'name' => 'required|string|max:255',
47              'email' => 'required|string|email|max:255|unique:users',
48              'password' => 'required|string|min:6|confirmed'
49          ]);
50
51          if ($validacao->fails())
52              return redirect()->back()->withErrors($validacao)->withInput();
53
54          $data['password'] =bcrypt($data['password']);
55
56          User::create($data);
57
58          return redirect()->back();
59      }
```

Na linha 52, temos o “withInput()” na função “redirect()”. Serve para manter no formulário as informações que já foram digitadas, caso a validação acuse algum erro de preenchimento.

A função da linha 56, grava os dados no banco.

O método show(), para buscar os dados do usuário:

```
67          public function show($id)
68          {
69              return User::find($id);//retorna um Json
70          }
```

O método update(), para alterar os dados do usuário, por enquanto pode ficar da seguinte forma:

```
90      public function update(Request $request, $id)
91      {
92          $data = $request->all();
93          $validacao = \Validator::make($data,[
94              'name' => 'required|string|max:255',
95              'email' => 'required|string|email|max:255|unique:users',
96              'password' => 'required|string|min:6|confirmed'
97          ]);
98
99          if ($validacao->fails())
100             return redirect()->back()->withErrors($validacao)->withInput();
101
102         User::find($id)->update($data);
103
104         return redirect()->back();
105     }
```

E, por fim, o método de exclusão:

```
113     public function destroy($id)
114     {
115         User::find($id)->delete();
116         return redirect()->back();
117     }
```

2 – criar a rota

Ao criar uma classe controller é necessário definir a rota de acesso para ela. Esta definição se faz no arquivo "routers/web.php", da seguinte forma:

Route::resource('usuario', "UsuarioController@index")

No caso de rotas que deverão serem protegidas, restringindo o acesso somente a usuários administradores, elas deverão ficar dentro de um grupo de rotas com acesso restrito, da seguinte forma:

```
22  Route::middleware(['auth'])->prefix('admin')->namespace('Admin')->group(function(){
23      Route::resource('usuarios', 'UsuariosController');
24  });
```

3 – criar a view de Usuário

Na pasta “resources/views”, criar a pasta “admin” e dentro dela a pasta “usuarios”. Dentro desta pasta, crie o arquivo “index.blade.php”.

Este arquivo será a tela que listará os usuários cadastrados. Para cada usuário teremos os botões “editar”, “ver detalhes” e “excluir”. Ao clicar em um destes botões, os dados/formulário será exibido em uma janela (modal).

No topo desta lista teremos o botão “criar”, para cadastrar um novo usuário, que será incluído posteriormente.

A estrutura inicial deste arquivo, deverá ser a seguinte:

```
@extends('layouts.app')

@section('content')

@endsection
```

Dentro do bloco @section, deverá ter o código abaixo para listar os usuários:

```
4   <pagina tamanho="12">
5       @if($errors->all())
6           <div class="alert alert-danger alert-dismissible text-center" role="alert">
7           <button type="button" class="close" data-dismiss="alert" aria-label="Close"><span aria-hidden="true">&times;</span></button>
8               @foreach ($errors->all() as $key => $value)
9                   <li><strong>{{$value}}</strong></li>
10              @endforeach
11          </div>
12      @endif
13      <painel titulo="Lista de usuários">
14          <migalhas v-bind:lista="{{$listaMigalhas}}"></migalhas>
15          <tabela-lista
16              v-bind:titulos="['#','Nome','E-mail','Data']"
17              v-bind:itens="{{json_encode($listaModelo)}}"
18              ordem="asc" ordemcol="2"
19              criar="#criar" detalhe="/admin/usuarios" editar="/admin/usuarios" deletar="/admin/usuarios" token="{{ csrf_token() }}" modal="sim">
20          </tabela-lista><!-- se, por exemplo, mantermos o valor "#criar" no atributo "criar", então o botão "criar" será exibido no componente "
21          <div align="center"><!-- paginação -->
22              {{$listaModelo->links()}}
23          </div>
24      </painel>
25  </pagina>
```

Linha 4 - <pagina> é um componente que implementaremos mais à frente.
Linhas 5 a 12 implementamos a exibição de mensagem de erro, caso ocorra na edição dos dados de um usuário.
Linha 13 - <painel> é um componente que implementaremos mais à frente. Ele está recebendo o atributo “título” como parâmetro.
Linha 14 - temos o componente “<migalhas>” que também será implementado. Ele está recebendo como parâmetro a lista de migalhas que implementamos na classe UsuariosController.

Linha 15 – componente “<tabela-lista>”. Este componente recebe uma série de parâmetros. Que serão explicados após implementarmos o seu código.
Linhas 21 a 23 – paginação.

Ainda neste arquivo temos que implementar os códigos das janelas que se abrirão (modais) ao clicar em cada um dos links (editar, detalhes e deletar).

Para estas janelas, implementaremos um componente com o nome de “modal”. O código a ser implementado na view de usuário para abrir a janela com o formulário de cadastro, deverá ser o seguinte:

```
27  <modal nome="adicionar" titulo="Adicionar">
28      <formulario id="formAdicionar" css="" action="{{route('usuarios.store')}}" method="post" enctype="" token="{{ csrf_token() }}">
29          <div class="form-group">
30          <label for="name">Nome</label>
31          <input type="text" class="form-control" id="name" name="name" value="{{old('name')}}">
32          </div>
33          <div class="form-group">
34              <label for="email">E-mail</label>
35              <input type="email" class="form-control" id="email" name="email" value="{{old('email')}}">
36          </div>
37          <div class="form-group">
38              <label for="password">Senha</label>
39              <input type="password" class="form-control" id="password" name="password" value="{{old('senha')}}">
40          </div>
41      </formulario>
42      <span name="botoes">
43          <button form="formAdicionar" class="btn btn-info">Adicionar</button>
44      </span>
45  </modal>
```

Linha 27 – iniciamos o componente <modal>, passando como parâmetros um nome e um título.
Linha 28 – estamos utilizando um componente <formulario> que também teremos que implementar, com seus devidos parâmetros (id, css, action, method, enctype e token). A action envia os dados para o método “store()”, da classe UsuarioController.
Linha 31 – a função “old()” está pegando a informação enviada pela classe “UsuarioController”, por meio da função “WithInput()”. Recupera os dados informados pelo usuário, caso a validação detecte a falta de preenchimento de campos obrigatórios.

As outras janelas (modal), são muito semelhantes a esta.

4 - Rodar Servidor PHP:

```
php artisan serve
```

5 – Criar componentes

Os componentes do framework Vue.js, ficam na pasta "resources/assets/js/components". Nesta pasta o Laravel já disponibiliza um componente de exemplo. Os componentes, após implementados precisam estar declarados no arquivo app.js

Após implementar os componentes é necessário deixar ativo um “observador”, que observará as alterações nos componentes, de modo que possa repercurti-las na view. Para ativar o observador, execute o seguinte comando:

```
npm run watch
```

Um componente é referenciado na página html, por meio de uma tag com o seu nome. Por exemplo, o componente “<tabela-lista>” (linha 15). O conteúdo do componente será exibido a partir da sua tag na página html.

Segue exemplo do componente <tabela-lista>:

TabelaLista.vue

```
1  <template>
2    <div>
3
4      <div class="form-inline">
5        <a v-if="criar && !modal" v-bind:href="criar">Criar</a>
6        <modallink v-if="criar && modal" tipo="button" nome="adicionar" titulo="Criar" css=""></modallink>
7        <div class="form-group pull-right">
8          <input type="search" class="form-control" placeholder="Buscar" v-model="buscar" >
9        </div>
10     </div>
11
12     <table class="table table-striped table-hover">
13       <thead>
14         <tr>
15           <th style="cursor:pointer" v-on:click="ordenaColuna(index)" v-for="(titulo,index) in titulos" :key="titulo.id">{{titulo}}</th>
16
17           <th v-if="detalhe || editar || deletar">Ação</th>
18         </tr>
19       </thead>
20       <tbody>
21         <tr v-for="(item,index) in lista" :key="item.id">
22           <td v-for="i in item" :key="i.id">{{i}}</td>
23
24           <td v-if="detalhe || editar || deletar">
25             <form v-bind:id="index" v-if="deletar && token" v-bind:action="deletar + item.id" method="post">
26               <input type="hidden" name="_method" value="DELETE">
27               <input type="hidden" name="_token" v-bind:value="token">
```

Linha 5 – observe o parâmetro “criar”, que foi enviado pela pagina html, conforme pode-se conferir na linha 19 do código representado na página 3 desta apostila. Este é apenas um deles, há vários outros.

```
19          criar="#criar" detalhe="/admin/usuarios" edit
```

Todos os parâmetros são recebidos pelo componente por meio da varável “props”, da seguinte forma:

```
67   <script>
68     export default {
69       props:['titulos','itens','ordem','ordemcol','criar','detalhe','editar','deletar','token','modal'],
70       data: function(){
71         return {
72           buscar:'',
73           ordemAux: this.ordem || "asc",
74           ordemAuxCol: this.ordemcol || 0
75         }
76       },
```

Aqui está sendo mostrado também parte da implementação do recurso de ordenação dos itens listados na página html.

6 - Importar fontes e demais arquivos css

Para importar fontes e demais arquivos css, sem que seja necessário baixar e acrescentar ao seu projeto, bastar utilizar os links CDN destes arquivos e colocá-los no arquivo "resources/assets/sass/app.scss"
Exemplo: @import url("https://fonts.googleapis.com/css?family=Raleway:300,400,600");

7 – Comando para criar uma classe Model (exemplo classe Artigo)

php artisan make:model Artigo -m //o "-m" é para criar a migração

Obs.: não é necessário criar a classe Model de Usuário, uma vez que o Laravel gera a classe “User”, quando executado o comando para a criação das funcionalidades de autenticação.

8 – Comando para criar tabelas do Banco de Dados (lembrando que antes disso é necessário criar as classes de migração, conforme fizemos no item anterior, ao acrescentar “-m” no comando que cria a classe Model)

```
php artisan migrate
```

9 - instalar o VueX (para possibilitar a edição de registros)

```
npm i vuex --save-dev
```

No **app.js**, incluir as seguintes linhas:

```
import Vuex from 'Vuex';
Vue.use(Vuex);

const store = new Vuex.Store({
   state:{
        itens:{}
   },
   mutations:{
        setItens(state,obj){
             state.itens = objs;
        }
   }
});
```

Por fim, inclua a variável **state** na constante "app" neste mesmo arquivo (app.js).

10 – exibir mensagens de erro em português

laravel-5.5-pt-BR-localization:
Seguir as instruções que estão no link abaixo:
https://github.com/enniosousa/laravel-5.5-pt-BR-localization
Necessário instalar o Git.
Configurar o Framework para utilizar o idioma português como Default
// Linha 81 do arquivo config/app.php

```
'locale' => 'pt-BR',
```

O arquivo de configuração do idioma fica na pasta “resources/lang”.

11 – criar a tela que exibe informações detalhadas do usuário

Utilizar o método “show()” da classe UsuarioController, conforme exemplo de implementação da página 2.

Na página 4, observe que na linha 19 do código estamos passando um parâmetro, em “detalhe”, que será lido pelo componente “<tabela-lista>”. Mais à frente mostrarei o código deste componente.

```
18              ordem="asc" ordemcol="2"
19              criar="#criar" detalhe="/admin/usuarios" editar="/
20          </tabela-lista><!-- se, por exemplo, mantermos o valor
```

Com estes parâmetros da linha 19, o componente montará a url de acesso à página de detalhes do usuário.

12 – Adicionar nova coluna na tabela do banco de dados

`php artisan make:migration [nome da migração] --table=[nome da tabela]`

Abra o arquivo que foi gerado e implemente a alteração a ser feita na tabela.

Exemplo para adicionar o campo autor, do tipo “enum”

```
public function up()
{
    Schema::table('users', function (Blueprint $table) {
        $table->enum('autor',['N','S'])->default('N');
    });
}
```

Toda implementação feita na função “up()” deverá ser feita a implementação inversa na função “down()”.

```
public function down()
{
    Schema::table('users', function (Blueprint $table) {
        $table->dropColumn('admin');
```

Após fazer esta implementação, execute o comando:

`php artisan make:migrate`

13 – Crirar regras de acesso, conforme o perfil do usuário

As regras são implementadas na classe “AuthServiceProvider”, que fica dentro de “app/providers”.

```
public function boot()
{
    $this->registerPolicies();

    Gate::define('eAdmin', function ($user) {
        return $user->admin == "S";
    });
```

Neste exemplo, foi definida uma regra com o nome de “eAdmin”, que retorna TRUE se o usuário for administrador.

Nesse outro caso, verifica-se se é administrador, se não for, verifica se é autor.

```
Gate::define('autor', function ($user) {
    return ($user->admin == "S" ? true : $user->autor == "S");
});
```

Os bloqueios de acesso podem ser implementados no template, nas rotas ou nas classes de controle.

Nas rotas, por exemplo, podemos fazer da seguinte forma:

```
Route::middleware(['auth'])->prefix('admin')->namespace('Admin')->group(function(){

  Route::resource('artigos', 'ArtigosController');
  Route::resource('usuarios', 'UsuariosController');
  Route::resource('autores', 'AutoresController');
  Route::resource('adm', 'AdminController')->middleware('can:eAdmin');

});
```

Observe que utilizamos o nome da regra que criamos (eAdmin) para definir que somente administradores poderão acessar a view “adm”.

Para bloquear o acesso na view, faremos da seguinte forma:

```
@can('eAdmin')
  <div class="col-md-4">
    <caixa qtd="{{$totalUsuarios}}" titulo="Usuários" url="{{route('usuarios.index')}}
  </div>
  <div class="col-md-4">
    <caixa qtd="{{$totalAutores}}" titulo="Autores" url="{{route('autores.index')}}"
  </div>
  <div class="col-md-4">
    <caixa qtd="{{$totalAdmin}}" titulo="Admin" url="{{route('adm.index')}}" cor="gre
  </div>
@endcan
```