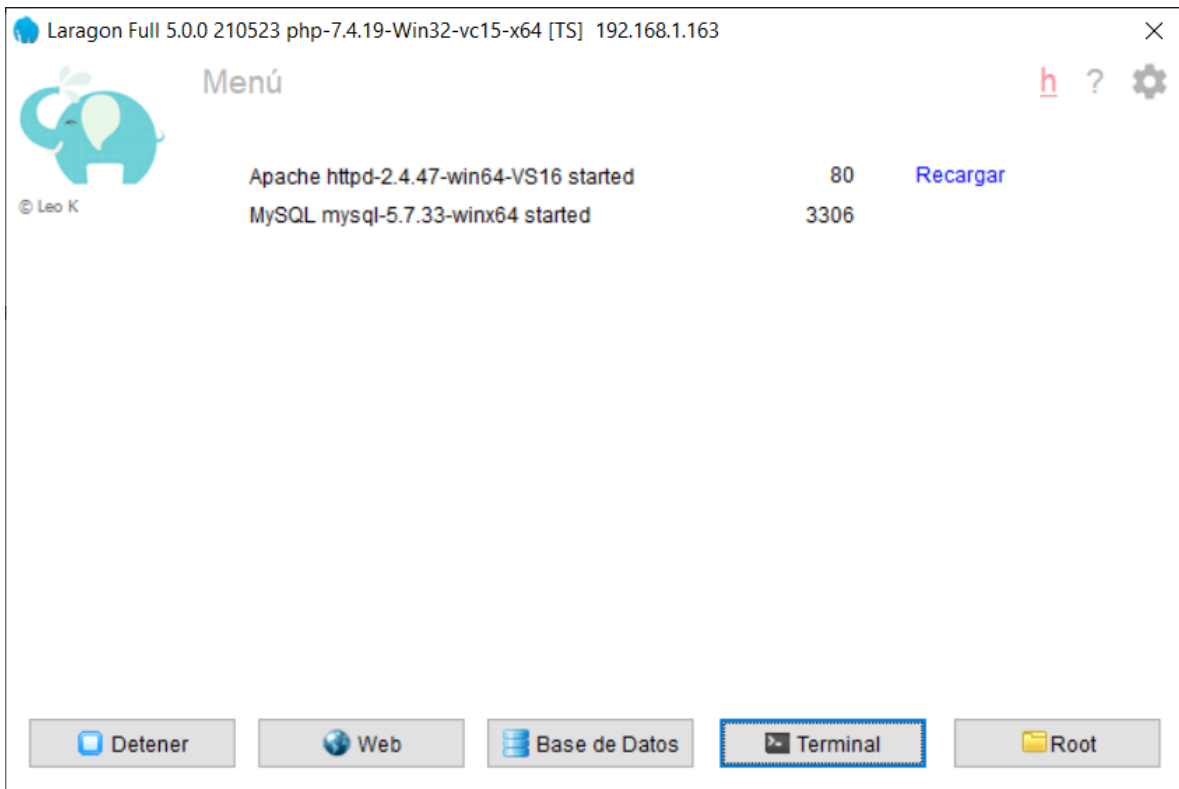


Proyecto: Página web de bienes raíces (Terracota Hábitat) Manual de Programador

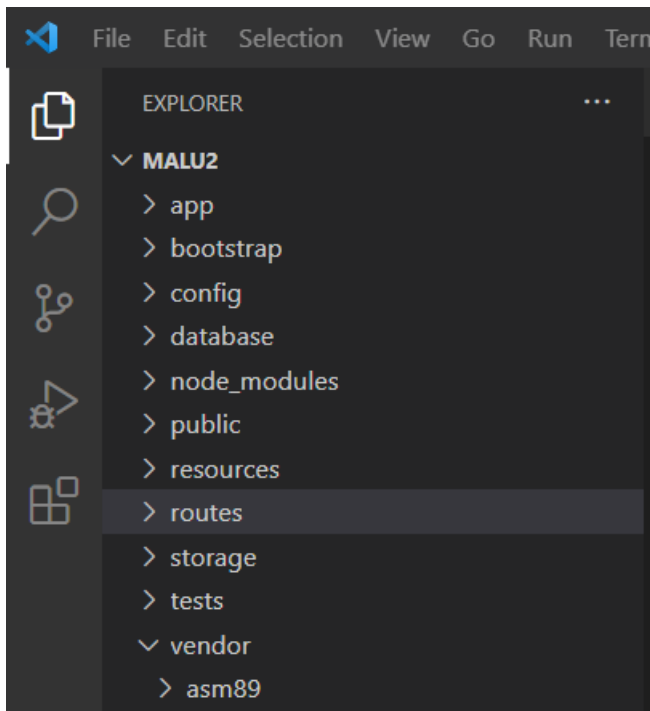
Para este proyecto utilizamos Laragon, y con este nuestro proyecto será en la versión 7.30.4.

Para comenzar con la programación, lo primero que hacemos será generar nuestro proyecto de Laravel:

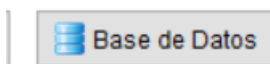


En esta ventana damos click derecho y ponemos creación rápida de sitio web, y le damos un nombre a nuestro proyecto, en mi caso "MALU2"

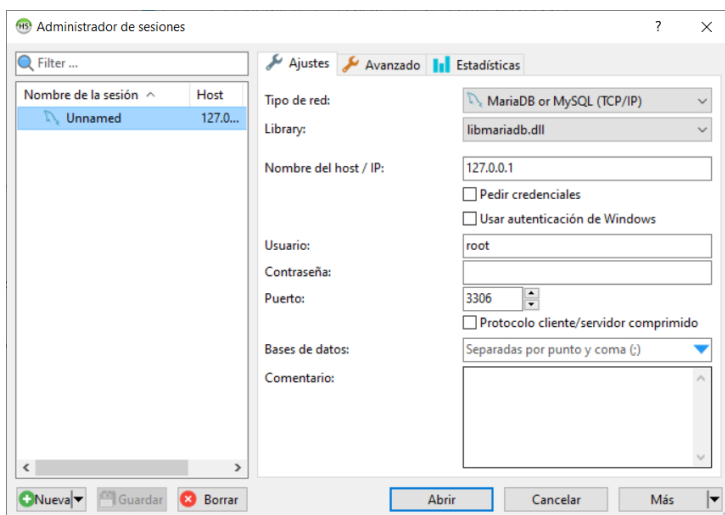
Tiendo esto se nos va a generar una carpeta que es recomendable manipular con Visual Code, se verá algo así:



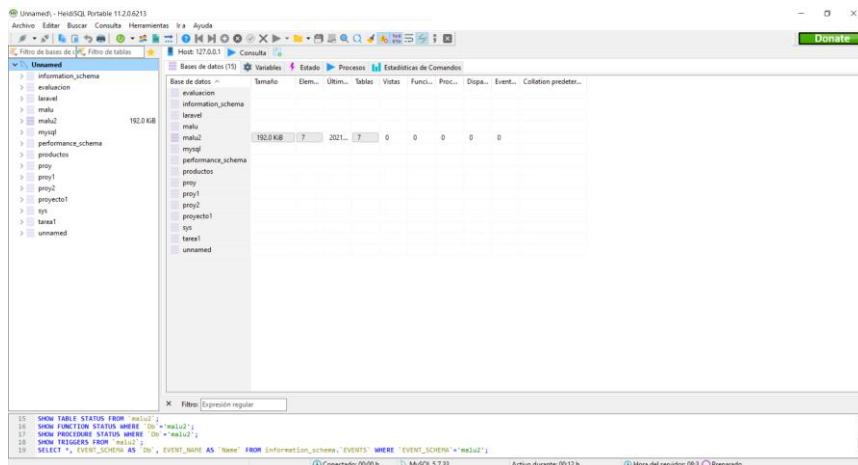
Ahora en nuestra ventana de Laragon,, damos click en el botón de base de datos:



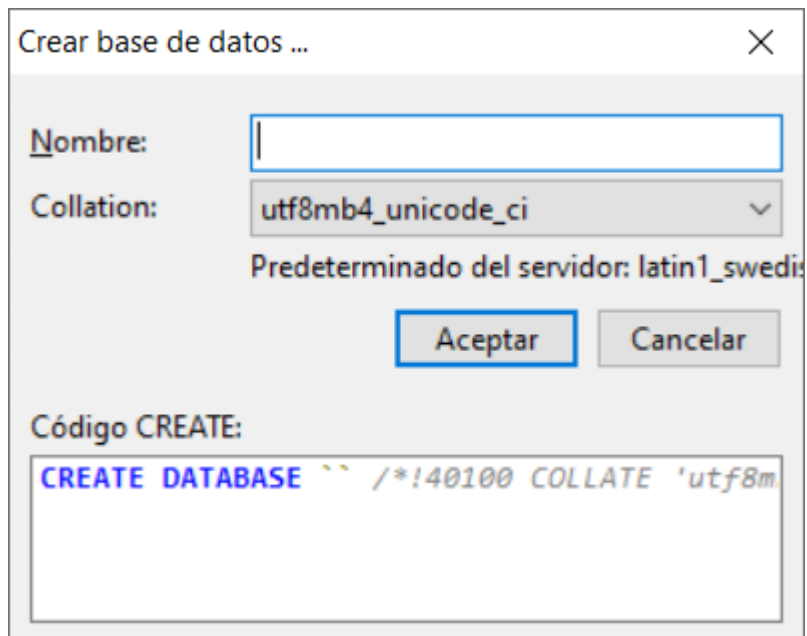
, nos abrirá esta ventana:



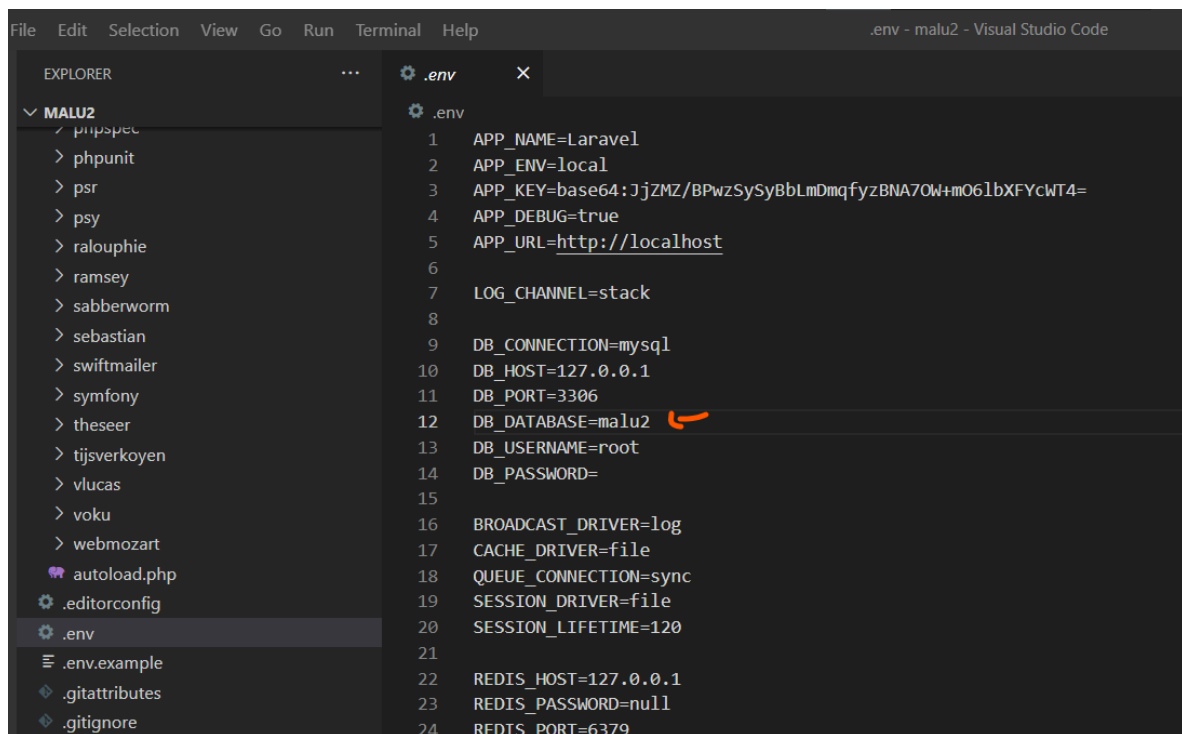
Damos abrir y click derecho crear nueva base de datos:



Damos nombre y seleccionamos caracteres:



Una vez tenemos esto, regresamos a nuestra carpeta en VCode, y buscamos el archivo llamado .env:



```
File Edit Selection View Go Run Terminal Help .env - malu2 - Visual Studio Code

EXPLORER
MALU2
├── composer
├── phpunit
├── psr
├── psy
├── ralouphie
├── ramsey
├── sabberworm
├── sebastian
├── swiftmailer
├── symfony
├── theseer
├── tijsverkoyen
├── vlucas
├── voku
├── webmozart
├── autoload.php
├── .editorconfig
├── .env
├── .env.example
├── .gitattributes
└── .gitignore

.env
1 APP_NAME=Laravel
2 APP_ENV=local
3 APP_KEY=base64:JjZMZ/BPwzSySyBbLmDmqfyzBNA7OW+mO6lbXFYcWT4=
4 APP_DEBUG=true
5 APP_URL=http://localhost
6
7 LOG_CHANNEL=stack
8
9 DB_CONNECTION=mysql
10 DB_HOST=127.0.0.1
11 DB_PORT=3306
12 DB_DATABASE=malu2
13 DB_USERNAME=root
14 DB_PASSWORD=
15
16 BROADCAST_DRIVER=log
17 CACHE_DRIVER=file
18 QUEUE_CONNECTION=sync
19 SESSION_DRIVER=file
20 SESSION_LIFETIME=120
21
22 REDIS_HOST=127.0.0.1
23 REDIS_PASSWORD=null
24 REDIS_PORT=6379
```

En la parte de DB_DATABASE: pondremos el nombre de la base de datos que acabamos de crear para que esa sea la base de datos de nuestro proyecto actual.

Ahora antes de comenzar es importante tener en cuenta que en nuestro proyecto en un futuro vamos a necesitar un login y un registro de usuarios por lo que añadiremos de una vez el paquete Authentication que laravel tiene en su documentación los pasos son:

1.- estos comandos en consola (Recordando estar en la carpeta del proyecto)

- composer require laravel/ui:^2.4
- php artisan ui vue --auth
- npm install && npm run devc
- php artisan route:list
- php artisan migrate:fresh

2.- Revisar que en el proyecto se hayan creado las vistas Login, Register y el HomeController en controladores

Una vez que tenemos esto listo, pasaremos a crear nuestra vista inicial en la vista home, que es el directorio raíz con ruta '/' y muestra la página de inicio de nuestro proyecto.

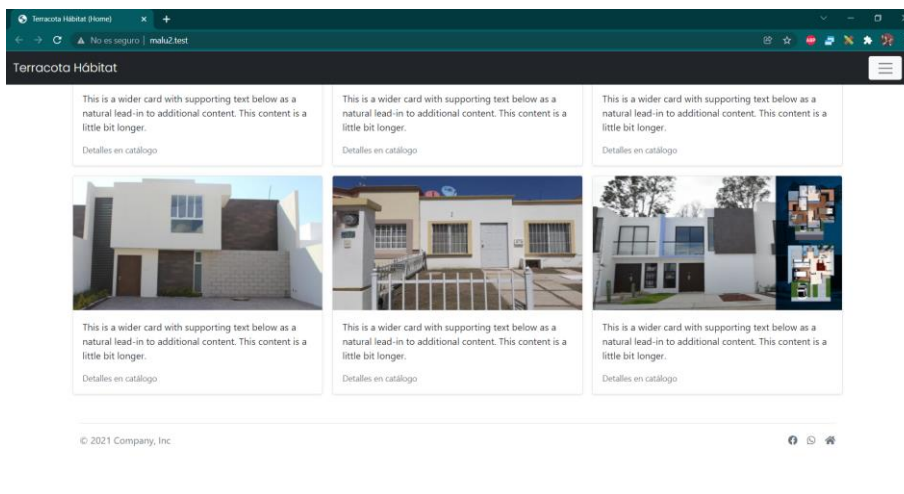
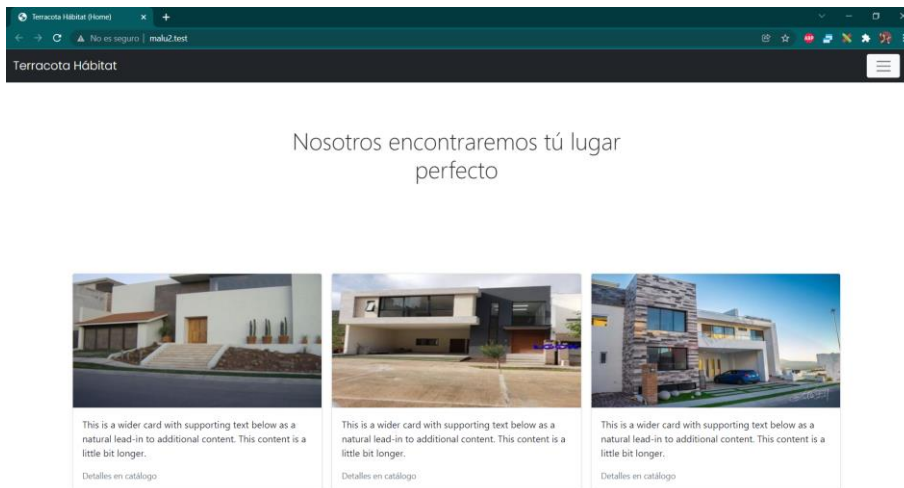
```
File Edit Selection View Go Run Terminal Help home.blade.php - maluz - Visual Studio Code

EXPLORER
MALUZ
  app
  bootstrap
  config
  database
  node_modules
  public
  resources
  js
  lang
  sass
  views
    auth
    layouts
    agregarprop.blade.php
    catalogo.blade.php
    contactanos.blade.php
    footer.blade.php
    home.blade.php
    imagenPropiedad.blade.php
    layout.blade.php
    layoutAdmin.blade.php
    layoutUs.blade.php
    nosotros.blade.php
    pdf.blade.php
    perfil.blade.php
    propiedad.blade.php
    usuarios.blade.php
    verpropiedad.blade.php
    welcome.blade.php
  routes
  storage
  OUTLINE

home.blade.php
1 <!DOCTYPE html>
2 <html lang="{{ str_replace('_', '-', app()->getLocale()) }}">
3
4 <head>
5   <meta charset="utf-8">
6   <meta name="viewport" content="width=device-width, initial-scale=1">
7   <script src="https://kit.fontawesome.com/924c212a80.js" crossorigin="anonymous"></script>
8
9   <title>Terracota Hábitat (Home-Logeado) </title>
10
11 <!-- Fonts -->
12 <link rel="preconnect" href="https://fonts.googleapis.com">
13 <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
14 <link href="https://fonts.googleapis.com/css2?family=Poppins:wght@300&display=swap" rel="stylesheet">
15 <link href="https://fonts.googleapis.com/css2?family=Lato:wght@700&display=swap" rel="stylesheet">
16 <link href="https://fonts.googleapis.com/css2?family=Montserrat:wght@300&display=swap" rel="stylesheet">
17
18 <!-- Styles -->
19 <style type="text/css">
20   .body {
21     width: 100%;
22     background-color: white;
23   }
24
25   .textol {
26     color: white;
27     font-family: 'Lato', sans-serif;
28     position: relative;
29     font-size: 60px;
30     top: -550px;
31     width: 600px;
32     left: 70px;
33   }
34
35   .desc1 {
36     color: white;
37     font-family: 'Montserrat', sans-serif;
38     position: relative;
```

Para este proyecto se utilizó Bootstrap, en cuestión de diseño y maquetación, la pagina inicial quedó de esta forma:





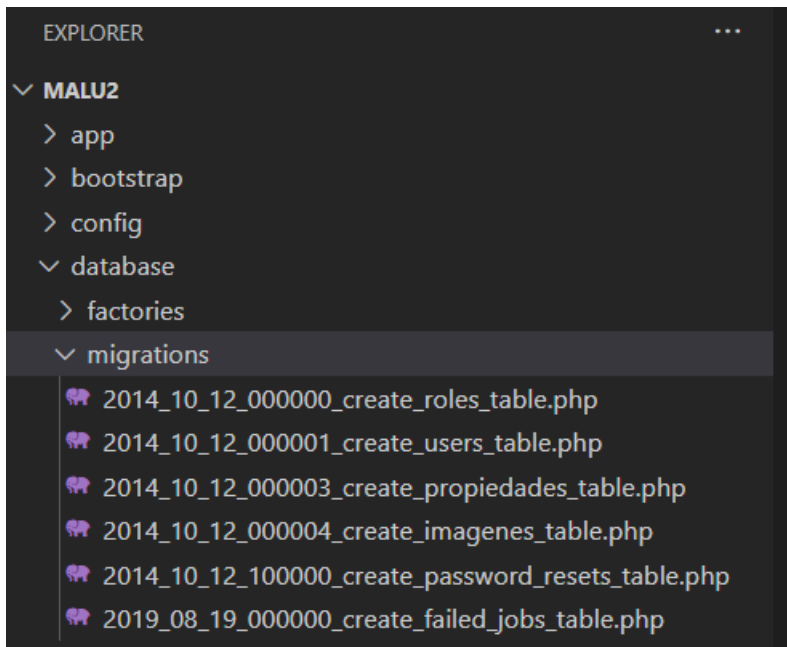
Teniendo esta vista lista, haremos nuestras tablas para comenzar con la parte de recopilación de datos y darle vida a nuestra página web, para eso tendremos que correr los siguientes comandos en nuestra consola creando las migraciones, vista, controlador que necesitaremos para el manejo de los datos en las tablas.

php artisan make:model NOMBRE_TABLA -mc

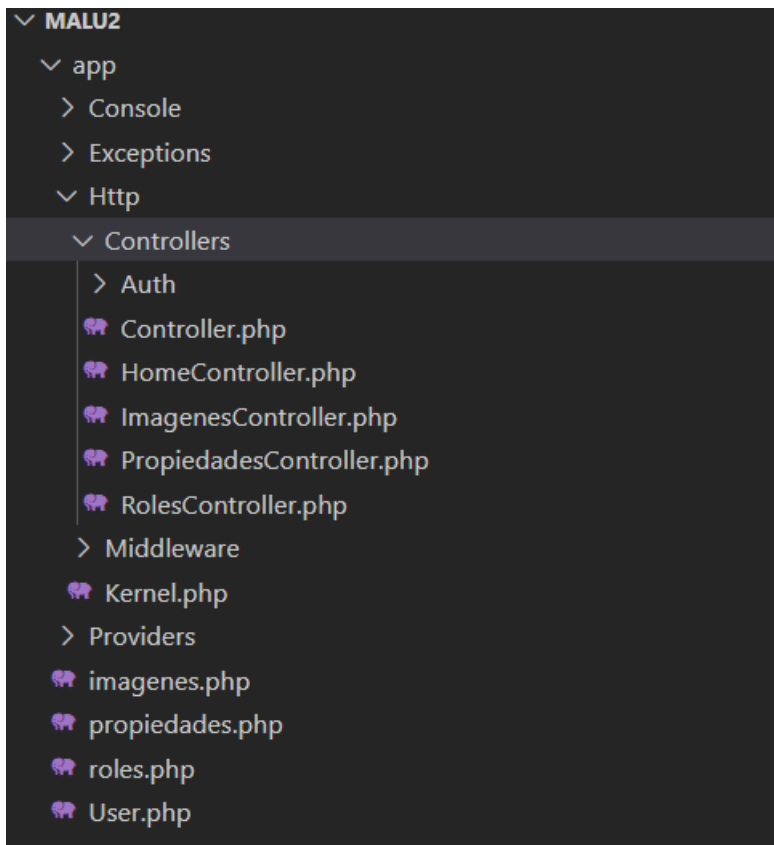
después ponemos el comando

php artisan migrate:refresh

y en nuestra carpeta aparecerán estos elementos:



También se crearon:



Para este proyecto en específico se crearon las tablas imágenes, propiedades y roles porque users se crea con el paquete de autenticación.

En el archivo de migrations de cada tabla hay que definir los atributos y tipos que queremos que tengan las tablas que recién creamos, aquí quedaron así:

```
class CreateRolesTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('roles', function (Blueprint $table) {
            $table->id('idRol');
            $table->string('name');
            $table->timestamps();
        });
    }
}
```

```
class CreateUsersTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    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->string('nick')->unique();
            $table->foreignId('idRol')->default(1)->references('idRol')->on('roles')->onUpdate('cascade')->onDelete('cas');
            $table->rememberToken();
            $table->timestamps();
        });
    }
}
```



```

class CreatePropiedadesTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('propiedades', function (Blueprint $table) {
            $table->id('idP');
            $table->string('name');
            $table->string('location');
            $table->string('price');
            $table->string('description');
            $table->string('terrain');
            $table->timestamps();
        });
    }
}

```

```

class CreateImagenesTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('imagenes', function (Blueprint $table) {
            $table->id();
            $table->string('name');
            $table->foreignId('idP')->default(1)->references('idP')->on('propiedades')->onUpdate('cascade')->onDelete('casc');
            $table->timestamps();
        });
    }
}

```

Ahora para este proyecto tendremos 3 tipos de usuarios el denominado “Invitado”, “Usuario” y el “Administrador” cree 3 menús distintos que se cargaran dependiendo del usuario que se encuentre en sesión, se cargan con una condición en cada vista realizada:

Menú

Home

Registrate

Iniciar Sesión

Catálogo

Acerca de ▼

Hola Pilar García Delgadillo

Home

Mi perfil

Catálogo

Cerrar Sesión

Acerca de ▼

Menú de Admin! Hola Pili!

Home

Añadir propiedad

Mi perfil

Mostrar propiedades

Edición de usuarios

Catálogo

Cerrar Sesión

Acerca de ▼

```
agregarprop.blade.php  home.blade.php X
resources > views > home.blade.php > html > body > div.container.border-bottom > a.btn.btn-secondary.b1.
54     </style>
55     <link rel="stylesheet" type="text/css" href="estilos.css">
56     <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css"
57     <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.m
58
59 </head>
60
61 <body>
62     @if (Auth::guest())
63     @include('layout')
64     @else
65
66     @if(Auth::user()->idRol==1)
67     @include('layoutUs')
68     @else
69     @include('layoutAdmin')
70     @endif
71     @endif
72
```

Para tener siempre en registro en la tabla roles, los tipos de roles existentes, así como un usuario administrador base que será el encargado de manipular la página, para eso creamos semillas que nos permiten tener datos pre cargados en la base de datos para crear la semilla hacemos:

make:seeder -Crear la ruta de semillas (ejecución de una vez)

php artisan make:seeder TABLASEEDER - crear una semilla

en este caso se crearon 2 roles y user:

```
database
├── factories
├── migrations
├── seeds
│   ├── DatabaseSeeder.php
│   ├── roles.php
│   └── users.php
└── .gitignore
```

En el DatabaseSeeder definimos las semillas de las tablas a utilizar:

```
DatabaseSeeder.php X
database > seeds > DatabaseSeeder.php > DatabaseSeeder > run
1  <?php
2
3  use Illuminate\Database\Seeder;
4
5  class DatabaseSeeder extends Seeder
6  {
7      /**
8       * Seed the application's database.
9       *
10      * @return void
11      */
12     public function run()
13     {
14         $this->call(roles::class);
15         $this->call(users::class);
16     }
17 }
18
```

Y en cada semilla insertamos los datos a la tabla:

```
DatabaseSeeder.php  roles.php 2 X
database > seeds > roles.php > ...
1  <?php
2
3  use Illuminate\Database\Seeder;
4
5  class roles extends Seeder
6  {
7      /**
8       * Run the database seeds.
9       *
10     * @return void
11     */
12     public function run()
13     {
14         DB::table('roles')->insert(['name'=>'Usuario',]);
15         DB::table('roles')->insert(['name'=>'Administrador',]);
16     }
17 }
18
```

```
DatabaseSeeder.php  roles.php 2  users.php 1 X
database > seeds > users.php > ...
1  <?php
2
3  use Illuminate\Database\Seeder;
4
5  class users extends Seeder
6  {
7      /**
8       * Run the database seeds.
9       *
10      * @return void
11      */
12     public function run()
13     {
14         DB::table('users')->insert(['name'=>'Pili',
15         'email'=>'123@hmsk.com',
16         'password' => bcrypt('laravel'),
17         'nick'=>'pili',
18         'idRol'=>'2',]);
19     }
20 }
21 }
22
```

Y al finalizar recargamos las semillas a nuestro proyecto con este comando:

```
php artisan migrate:fresh --seed
```

Lo siguiente es crear el enlace a las vistas y funciones, nuestras funciones serán creadas por practicidad y organización todas en `HOMECONTROLLER` y nuestras rutas deben estar definidas en el archivo `web.php`

```

HomeController.php 1 web.php X
routes > web.php > ...
9 | Web Routes
10 | -----
11 |
12 | Here is where you can register web routes for your application. These
13 | routes are loaded by the RouteServiceProvider within a group which
14 | contains the "web" middleware group. Now create something great!
15 |
16 | */
17 |
18 | Route::get('/', function () {
19 |     return view('welcome');
20 | });
21 |
22 | Auth::routes();
23 |
24 | Route::get('/home', 'HomeController@index')->name('home');
25 | Route::get('/nosotros', function () { return view('nosotros');});
26 | Route::get('/contactanos', function () { return view('contactanos');});
27 |
28 | Route::get('/usuarios', 'HomeController@verUsuarios');
29 | Route::get('/catalogo', 'HomeController@catalogo');
30 | Route::get('/eliminar/{id}', 'HomeController@eliminar');
31 | Route::post('/guardausr', 'HomeController@guardausr');
32 | Route::post('/updateUsr', 'HomeController@updateUsr');
33 | Route::get('/perfil', function () { return view('perfil');});
34 | Route::get('/agregarprop', function () { return view('agregarprop');});
35 | Route::post('/guardaProp', 'HomeController@guardaProp');
36 | Route::post('/guardaImg', 'HomeController@guardaImg')->middleware('auth');
37 | Route::get('/verpropiedad', 'HomeController@verProp');
38 | Route::get('/eliminarP/{id}', 'HomeController@eliminarP');
39 | Route::post('/guardaP', 'HomeController@guardaP');
40 | Route::get('/propiedad/{id}', 'HomeController@detalleProp');
41 | Route::get('/PruebaPDF/{id}', 'HomeController@PruebaPDF');
42 |

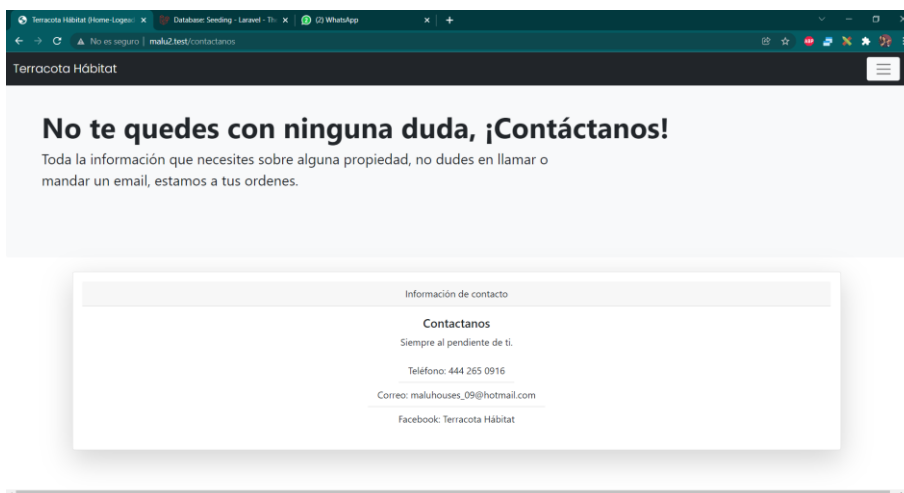
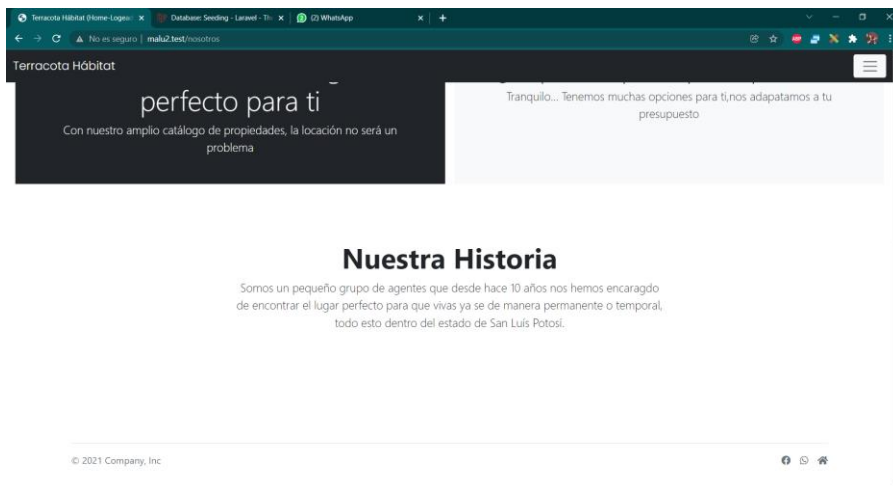
```

La primera vista creada fue la de nosotros y contáctanos que no tienen información cargada de la base de datos simplemente son vistas informativas, la ruta es de tipo Get y regresamos las vistas designadas. Quedaron así:

Acerca de ▾

Nosotros

Contáctanos

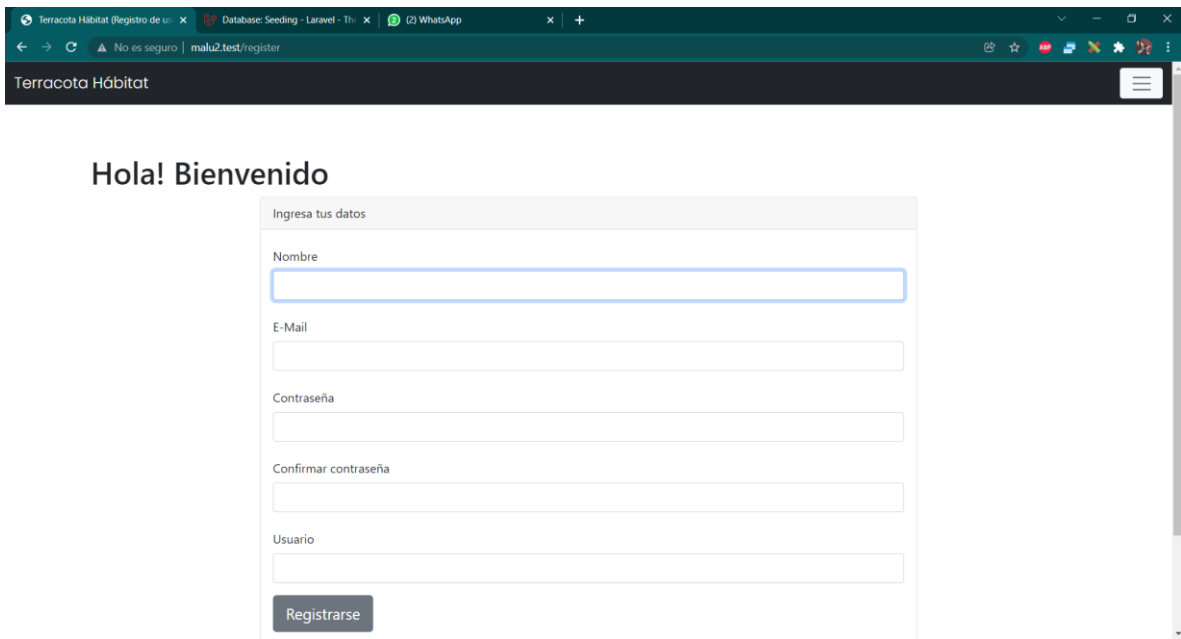


Un ejemplo de como en el menú le damos lugar a la ruta para cargar la vista que definimos en web.php sería algo así:

```
<ul class="dropdown-menu" aria-labelledby="offcanvasNavbarDropdown">
  <li><a class="dropdown-item" href="/nosotros">Nosotros</a></li>
  <li>
    <hr class="dropdown-divider">
  </li>
  <li><a class="dropdown-item" href="/contactanos">Contáctanos</a></li>
</ul>
```

Login y registro

Esta parte lo más lógico sería crear una función que recopile datos de un formulario, pero gracias al paquete de autenticación, estas funciones ya están definidas y simplemente se modificaron los datos a guardar y las vistas pre-definidas.



The screenshot shows a web browser window with the URL `malu2.test/register`. The page title is "Terracota Hábitat". Below the header, there is a greeting "Hola! Bienvenido". The main content is a registration form titled "Ingresa tus datos". The form contains the following fields:

- Nombre (Name): A text input field with a blue border.
- E-Mail: A text input field.
- Contraseña (Password): A text input field.
- Confirmar contraseña (Confirm password): A text input field.
- Usuario (Username): A text input field.

At the bottom of the form is a button labeled "Registrarse" (Register).

Terracota Hábitat (Registro de u... Database: Seeding - Laravel - Th... (2) WhatsApp

No es seguro | malu2.test/login

Terracota Hábitat

Hola! Porfavor inicia sesión

Ingresa tus credenciales

Correo

Contraseña

☐ Recuerdame

Login [Olvidaste tu contraseña?](#)

© 2021 Company, Inc

Funciones de administrador

Como administradores debemos tener oportunidad de ver qué usuarios se encuentran registrados, así como modificar sus roles, añadir propiedades al catálogo, modificar información de las propiedades previamente agregadas.

Lo primero será el registro de usuarios para ver cuantos tenemos en base:

Terracota Hábitat (Administrar u... Database: Seeding - Laravel - Th... (2) WhatsApp

No es seguro | malu2.test/usuarios

Terracota Hábitat

Registro de usuarios

Nombre	Email	Nick	Rol	Modificar Info	Eliminar
Pili	123@hmsk.com	pili	Administrador	Editar	Eliminar
Pilar García Delgadillo	pilig.del07@gmail.com	pili2	Usuario	Editar	Eliminar

© 2021 Company, Inc

Desglosa los datos de los usuarios y nos da la opción de darlo de baja o modificar sus datos en tiempo real, necesitaremos una función en homecontroller que recopile

todo lo existente en nuestra tabla Users y lo mande a la vista para ahí comenzar con la lectura algo así:

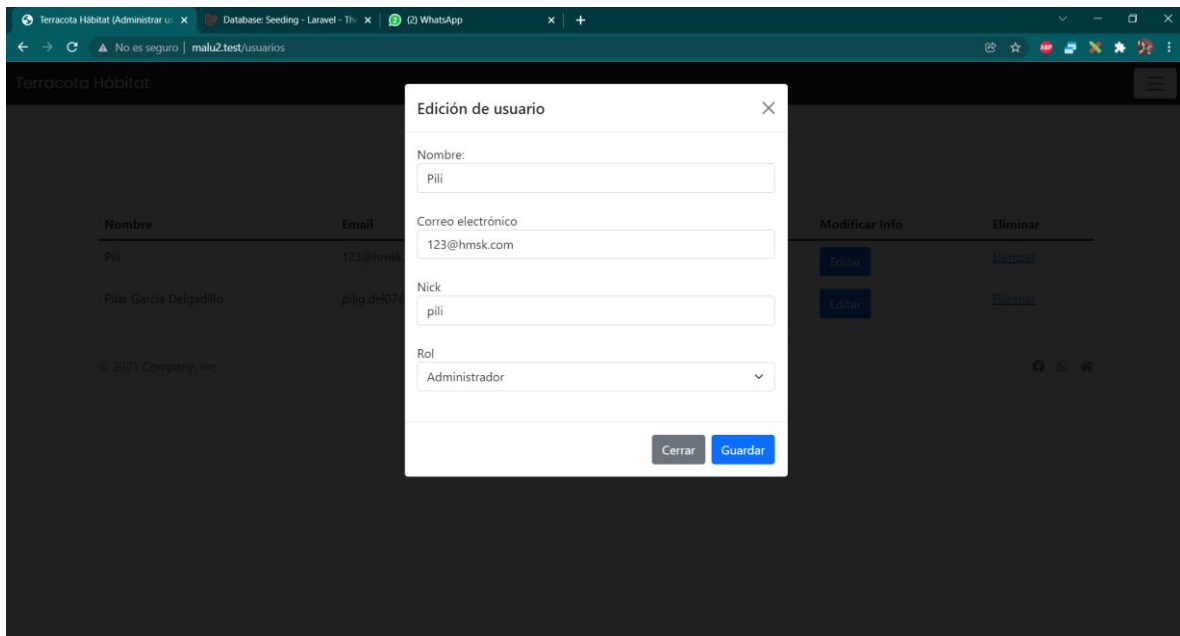
```
public function verUsuarios()
{
    $usuarios=user::all();
    return view('usuarios')->with('usuarios',$usuarios);
}
```

Creamos una variable de tipo User que almacena todos los datos de la tabla usuarios y los manda con el nombre usuarios a la vista de usuarios así que en la vista usuarios para desplegar la lectura de todos nuestros atributos que queremos que el administrador sea capaz de observar:

```
@foreach($usuarios as $u)
    <tr>
        <td scope="row">{{ $u->name }}</td>
        <td>{{ $u->email }}</td>
        <td>{{ $u->nick }}</td>
        @if($u->idRol== 1)
            <td>Usuario</td>
            <input type="hidden" id="idRol_{{ $u->id }}" value="1">
        @else
            <td>Administrador</td>
            <input type="hidden" id="idRol_{{ $u->id }}" value="2">
        @endif
        <input type="hidden" id="name_{{ $u->id }}" value="{{ $u->name }}">
        <input type="hidden" id="email_{{ $u->id }}" value="{{ $u->email }}">
        <input type="hidden" id="nick_{{ $u->id }}" value="{{ $u->nick }}">
        <input type="hidden" id="idd" value="{{ $u->id }}">
        <td><button type="button" class="btn btn-primary btnModal" id="{{ $u->id }}" data-bs-toggle="modal" data-target="#{{ $u->id }}">
            Editar
        </button></td>
        <td><a href="/eliminar/{{ $u->id }}">Eliminar</a></td>
    </tr>
@endforeach
</tbody>
</table>
```

Utilizamos un for each para avanzar a través de todos los datos como un for normal en programación y accedemos a cada atributo con la notación u->atributo y con dobles paréntesis para la impresión en pantalla de la variable.

Para el botón de edición vamos a llamar a un modal que cambia los datos que se ven en pantalla pertenecientes a cada usuario



Se crea con un identificador y un modal de Bootstrap.

```
<div class="modal fade" id="Modal" tabindex="-1" aria-labelledby="exampleModalLabel" aria-hidden="true">
  <div class="modal-dialog">
    <div class="modal-content">
      <div class="modal-header">
        <h5 class="modal-title" id="exampleModalLabel">Edición de usuario</h5>
        <button type="button" class="btn-close" data-bs-dismiss="modal" aria-label="close"></button>
      </div>
      <form action="/guardausr" method="POST">
        <div class="modal-body">
          @csrf
          <input type="hidden" name="identificador" id="id2" value="0">
          <label for="">Nombre:</label>
          <input class="form-control" type="text" name="name" id="name" value="0">
          <br>
          <label for="">Correo electrónico</label>
          <input class="form-control" type="text" name="email" id="email" value="0" >
          <br>
          <label for="">Nick</label>
          <input class="form-control" type="text" name="nick" id="nick" value="0" >
          <br>
          <label for="">Rol</label>
          <select class="form-select" name="idRol" id="idRol" required>
            <option value="1">Usuario</option>
            <option value="2">Administrador</option>
          </select>
          <br>
        </div>
        <div class="modal-footer">
          <button type="button" class="btn btn-secondary" data-bs-dismiss="modal">Cerrar</button>
          <button type="submit" class="btn btn-primary">Guardar</button>
        </div>
      </form>
    </div>
  </div>
</div>
```

```

public function guardausr(Request $request)
{
    if(isset($request->identificador))
        $usuarios=user::find($request->identificador);
    else
        $usuarios=new user();

    $usuarios->name=$request->name;
    $usuarios->email=$request->email;
    $usuarios->nick=$request->nick;
    $usuarios->idRol=$request->idRol;

    $usuarios->save();

    if(isset($request->identificador))
        return redirect('/usuarios');

    return redirect()->back()->with('success','Usuario registrada con exito');
}

```

El formulario manda a llamar a una función que guarda la nueva información otorgada por el administrador para reflejar el cambio en la base de datos.

```

<script>
    $(document).ready(function(){
        var myModal = document.getElementById('Modal');
        myModal.addEventListener('shown.bs.modal', function () {});
        // $('#btnModal').click(ola);
        $("#btnModal").click(function(){
            var id = this.id;
            name = $("#name_" + id).val();
            email = $("#email_" + id).val();
            nick = $("#nick_" + id).val();
            idRol = $("#idRol_" + id).val();

            $("#id2").attr('value',id);
            $("#name").attr('value',name);
            $("#idRol option[value='"+idRol + "']").attr("selected",true);
            $("#nick").attr('value',nick);
            $("#email").attr('value',email);
        });
    });
</script>
</div>

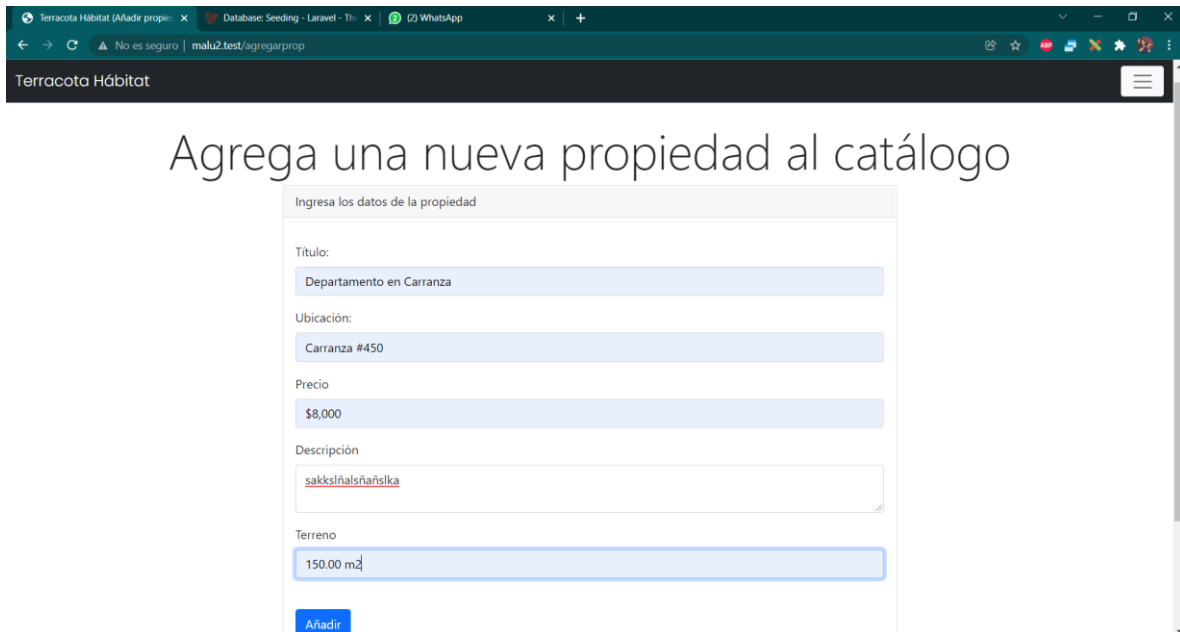
```

Acoplado javascript, para que se escriban los valores actuales en el formulario. Pero para la eliminación sencillamente tenemos una función dentro del HomeController que nos va a permitir eliminar el registro de la tabla.

```
public function eliminar($id)
{
    $usuarios=user::find($id);
    $usuarios->delete();
    return redirect('/usuarios');
}
```

Añadir propiedad

Tenemos un formulario para que el administrador registre los datos de la propiedad que desea dar de alta y luego se verá en el catálogo, esta en 2 pasos información e imagen de la propiedad



The screenshot shows a web browser window with the URL `malu2.test/agregarprop`. The page title is "Terracota Hábitat". The main heading is "Agrega una nueva propiedad al catálogo". Below the heading is a form titled "Ingresa los datos de la propiedad". The form contains the following fields:

- Título:** A text input field containing "Departamento en Carranza".
- Ubicación:** A text input field containing "Carranza #450".
- Precio:** A text input field containing "\$8,000".
- Descripción:** A text input field containing "sakkstñalsñññsika".
- Terreno:** A text input field containing "150.00 m2".

At the bottom of the form is a blue button labeled "Añadir".


```

public function guardaProp(Request $request)
{
    //dd($request);
    $propiedad = new propiedades();

    $propiedad->name=$request->name;
    $propiedad->location=$request->location;
    $propiedad->price=$request->price;
    $propiedad->description=$request->description;
    $propiedad->terrain=$request->terrain;
    $propiedad->save();
    $idP= $propiedad->id;
    return view('imagenPropiedad')->with('idPropiedad',$idP)->with('success','Propiedad registrada con éxito!');
    // return redirect()->back()->with('success','Propiedad registrado con éxito!');
}

```

Y para la imagen tenemos que:

```

public function guardaImg(Request $request)
{
    //dd($request);
    $img = new imagenes();

    $path=$request->file('imagen')->store('public');
    $file = basename($path);
    $img->name=$file;
    $img->idP=$request->idP;

    $img->save();
    return view('home')->with('success','Propiedad e imagen registradas con éxito!');
}

```

Para que funcionen las imágenes hay que ejecutar estos comandos:

php artisan storage:link

y para la visualización de la imagen se debe poner esta ruta en src:

```

with('propiedades',$prop)->with('imagenes',$img);
}
```



```

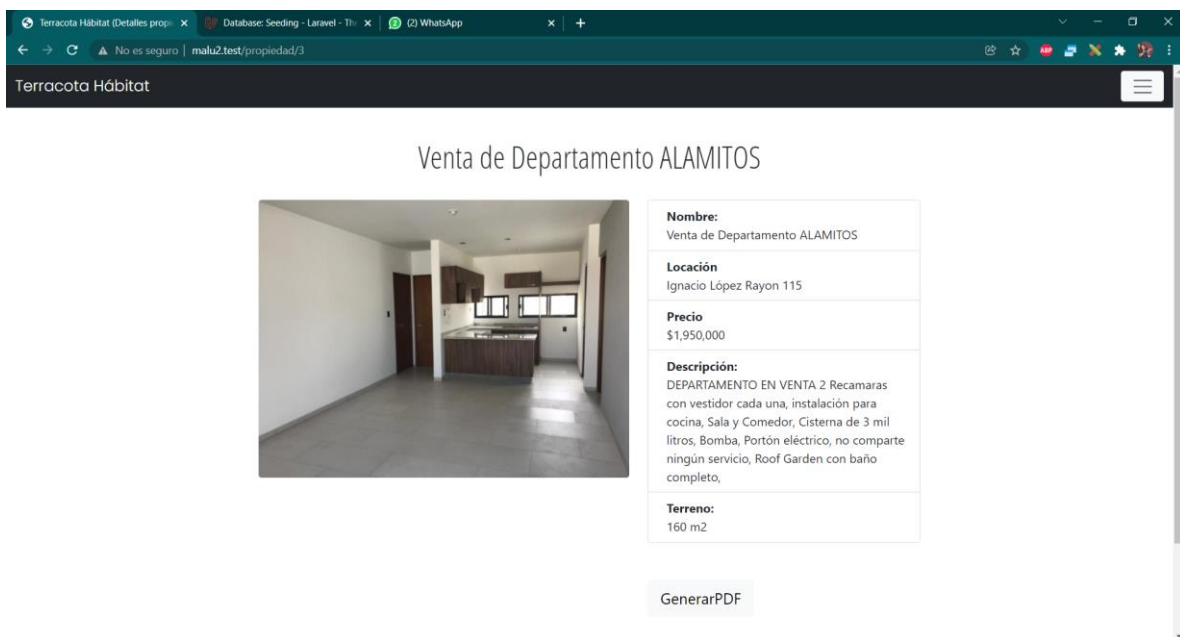
<div class="album py-5 bg-light">
  <div class="container-fluid">
    <div class="row row-cols-1 row-cols-sm-2 row-cols-md-3 g-3">
      @foreach($propiedades as $p)
        <div class="col">
          <div class="card shadow-sm">
            @foreach($imagenes as $i)

              @if($p->idP == $i->idP)
                {{{$p->name}}}</title>

            <div class="card-body">
              <div class="card-text">
                <ul class="list-group list-group-flush">
                  <li class="list-group-item">Locación: {{{$p->location}}}</li>
                  <li class="list-group-item">Precio: {{{$p->price}}}</li>
                  <li class="list-group-item">Terreno: {{{$p->terrain}}}</li>
                </ul>
              </div>
              <div class="d-flex justify-content-between align-items-center">
                <small class="text-muted"><a href="/propiedad/{{{$p->idP}}}" class="stretched-link">Más inf
              </div>
            </div>
          </div>
        </div>
      @endforeach
    </div>
  </div>
</div>

```

Ahora para ver los detalles de cada imagen en grande hicimos otra función que se llama detalle propiedad que recibe el identificador de nuestra propiedad para buscar su información en la tabla de imágenes y propiedades.



```

public function detalleProp($id){
    $p = DB::table('propiedades')->select('idP','name','location','price','description','terrain')->where('idP','=',$id)->get();
    $i= DB::table('imagenes')->select('name')->where('idP','=',$id)->get();
    // dd($p);
    // dd($i);
    return view('propiedad',array('propiedad' => $p,'imagen'=> $i));
}

```

Generar pdf

En nuestras vistas ampliadas de las propiedades tenemos un botón que genera un pdf con la información que estamos viendo en pantalla sobre la propiedad en cuestión.

160 m2

GenerarPDF

Para esto usamos una biblioteca llamada DomPDF, los pasos para su instalación son:

1. Comando en consola: `composer require barryvdh/laravel-dompdf`
2. En nuestro archivo de configuración "app.php" en la parte de providers añadimos

```
App\Providers\AppServiceProvider::class,  
App\Providers\AuthServiceProvider::class,  
// App\Providers\BroadcastServiceProvider::class,  
App\Providers\EventServiceProvider::class,  
App\Providers\RouteServiceProvider::class,  
  
Barryvdh\DomPDF\ServiceProvider::class,  
  
],  
  
/*
```

3. En el mismo archivo pero en aliases ponemos

```
'Validator' => Illuminate\Support\Facades\Validator::class,  
'View' => Illuminate\Support\Facades\View::class,  
'PDF' => Barryvdh\DomPDF\Facade::class,  
  
],
```

4. En el controlador donde creemos nuestros pdf ponemos USE PDF

Y creamos la siguiente función:

```

public function PruebaPDF($id){
    $p = DB::table('propiedades')->select('idP','name','location','price','description','terrain')->where('idP','=',$id);
    // $pdf = PDF::loadView('pdf',array('propiedad' => $p));

    $pdf =PDF::loadView('pdf',compact('p'));
    return $pdf->download('invoice.pdf');
}

```

Impornate crear una vista de pdf donde pondremos que información se debe imprimir de la que estamos recibiendo en la función loadView

The screenshot shows the Visual Studio Code editor with a file named `pdf.blade.php` open. The editor has a tab bar at the top with several files: `HomeController.php 1`, `pdf.blade.php X`, `catalogo.blade.php`, `usuarios.blade.php`, and `web.php`. The main editor area shows the content of `pdf.blade.php` with the following code:

```

1  <!DOCTYPE html>
2  <html>
3      <head>
4          <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
5          <title>Hola HTML</title>
6      </head>
7      <body>
8          <p> {{ $p[0]->name }}</p>
9          <p> {{ $p[0]->location }}</p>
10         <p> {{ $p[0]->price }}</p>
11         <p> {{ $p[0]->description }}</p>
12         <p> {{ $p[0]->terrain }}</p>
13     </body>
14 </html>
15

```

Mostrar propiedades

Mismo concepto que la edición de usuarios pero ahora trabajando con la tabla de propiedades

Registro de propiedades

Nombre	Locación	Precio	Descripción	Terreno	Editar	Eliminar
Renta de Casa SATELITE	República de Guatemala #150	\$8,000	CASA EN RENTA P.B. Baño Completo, Sala, Comedor, 1 Recamara con closet, Jardín, Cocina Integral, Patio Servicio, Cochera para 2 autos. P.A. 2 Recamaras con closet y baño completo.	150.00 m2	Editar	Eliminar
Venta de Departamento ALAMITOS	Ignacio López Rayon 115	\$1,950,000	DEPARTAMENTO EN VENTA 2 Recamaras con vestidor cada una, instalación para cocina, Sala y Comedor, Cisterna de 3 mil litros, Bomba, Portón eléctrico, no comparte ningún servicio, Roof Garden con baño completo,	160 m2	Editar	Eliminar
Departamento en Carranza	Carranza #450	\$8,000	sakksiñalsñañsika	150.00 m2	Editar	Eliminar

Terracota Hábitat (Editar propiedad) x Database: Seeding - Laravel - Th x (3) WhatsApp x +

← → ↻ No es seguro | malu2.test/verpropiedad

Terracota Hábitat

Nombre	Locación	Precio
Renta de Casa SATELITE	República de Guatemala #150	\$8,000
Venta de Departamento ALAMITOS	Ignacio López Rayon 115	\$1,950,000
Departamento en Carranza	Carranza #450	\$8,000

© 2021 Company, Inc.

Edición de datos de propiedad

Nombre:

Renta de Casa SATELITE

Locación:

República de Guatemala #150

Precio:

\$8,000

Descripción:

CASA EN RENTAP.B. Baño Completo, Sala, Comedor, 1 Recama

Terreno:

150.00 m2

Cerrar

Guardar

```

public function eliminarP($id)
{
    $propiedad=propiedades::where('idP',$id)->delete();
    $img= imagenes::where('idP',$id)->delete();
    return redirect('/verpropiedad');
}
public function guardaP(Request $request)
{
    if(isset($request->identificador))
        //$prop = propiedades::where('idP', $request->identificador)->first();
        $prop1=DB::table('propiedades')->where("propiedades.idP","=", $request->identificador)
            ->update(["propiedades.name"=>$request->name,"propiedades.location"=>$request->location,"propiedades.p
    else{
        $prop=new propiedades();
        $prop->name=$request->name;
        $prop->location=$request->location;
        $prop->price=$request->price;
        $prop->description=$request->description;
        $prop->terrain=$request->terrain;
        $prop->save();
    }

    if(isset($request->identificador))
        return redirect('/verpropiedad');

    return redirect()->back()->with('success','Propiedad registrado con exito');
}

```

Edición de información de perfil

Para un usuario es importante modificar sus datos personales, se muestran y en un formulario se pueden modificar los datos que el desee.

Terracota Hábitat (Mi perfil) x +

← → ↻ ⚠ No es seguro | malu2.test/perfil

Terracota Hábitat

Mi perfil

Hola pili2

Nombre: Pilar García Delgadillo

Usuario: pili2

Email: pili2.del07@gmail.com

[Cambiar contraseña](#)

Editar

¡Cambia tu información cuando necesites!

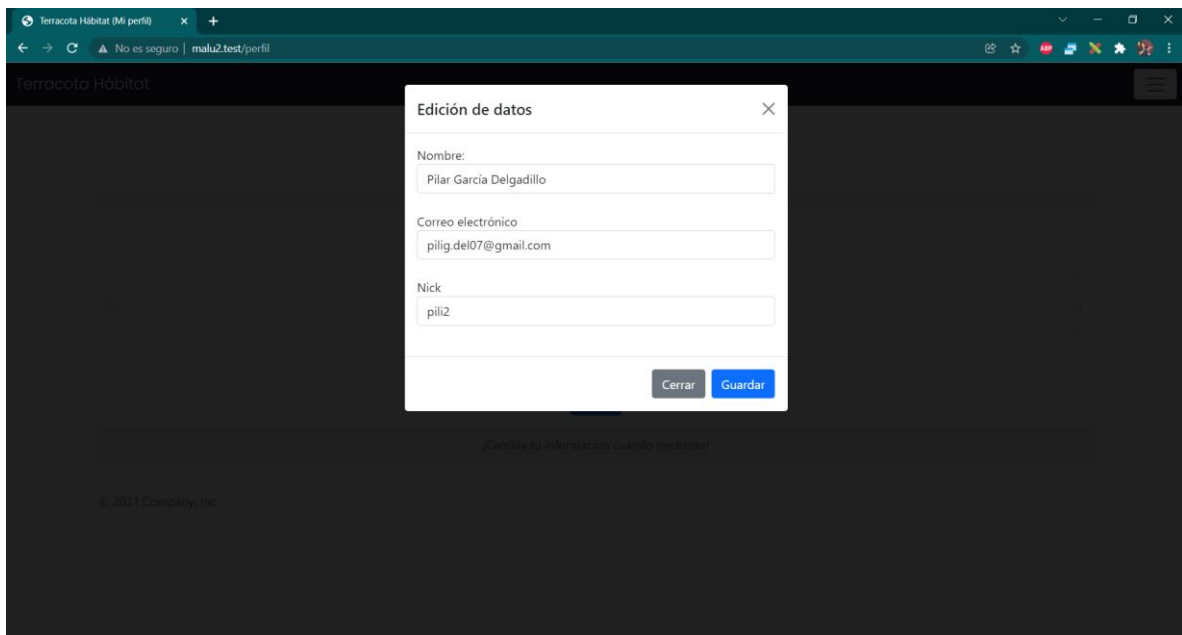
© 2021 Company, Inc

```
public function updateUsr(Request $request)
{
    if(isset($request->identificador))
        $usuarios=user::find($request->identificador);
    else
        $usuarios=new user();

    $usuarios->name=$request->name;
    $usuarios->email=$request->email;
    $usuarios->nick=$request->nick;

    $usuarios->save();

    if(isset($request->identificador))
        return redirect('/perfil');
}
```



Básicamente fueron los elementos y comandos necesarios para la creación de este proyecto.