Tópicos Avanzados de Programación

Semestre enero – junio 2024

JOSE ARTURO BUSTAMANTE LAZCANO

Contenido

[Introducción 3](#_Toc159848438)

[Requisitos para Laravel 6](#_Toc159848439)

[Iniciando un Proyecto 11](#_Toc159848440)

[Comando para iniciar un proyecto 11](#_Toc159848441)

[Extensiones VSCode 11](#_Toc159848442)

[Rutas en Laravel ¿Qué son y para que sirven? 11](#_Toc159848443)

[Comando de Instalación Auth 14](#_Toc159848444)

[Middleware 15](#_Toc159848445)

[FORMULARIOS EN LARAVEL 17](#_Toc159848446)

[Heredar Estilo 17](#_Toc159848447)

[Links de Navegación 17](#_Toc159848448)

[Link de Navegación móvil 18](#_Toc159848449)

[Casos de envío de datos 19](#_Toc159848450)

[Caso 1 enviar los datos sin un name y un método 19](#_Toc159848451)

[Caso 2: Enviar los datos con name 19](#_Toc159848452)

[Caso 3: especificar el método al formulario en post 20](#_Toc159848453)

[Caso 4: Acceder a los datos que ingreso el usuario. 20](#_Toc159848454)

[INSERCIÓN DE DATOS EN BD 21](#_Toc159848455)

[Modelos en Laravel 10 21](#_Toc159848456)

[Migraciones 21](#_Toc159848457)

[Eloquent 22](#_Toc159848458)

[Unidad 2 - Componentes y librerías. 26](#_Toc159848459)

[Introducción a Filament 26](#_Toc159848460)

[Ejecutamos 30](#_Toc159848461)

[Crear modelo 33](#_Toc159848462)

[Migraciones 38](#_Toc159848463)

[Generar los Resources 43](#_Toc159848464)

[Editar el Resource 46](#_Toc159848465)

[Paso 1 Crear Proyecto 47](#_Toc159848466)

[Paso 2 Abrir el composer.json cambiar la línea 64 a dev 47](#_Toc159848467)

[Paso 3 Cambiar el nombre de la bd en .env 47](#_Toc159848468)

[Paso 4 Instalar Filament 47](#_Toc159848469)

[Paso 5 importar la migración para crear la bd y Crear un usuario para entrar al admin 48](#_Toc159848470)

[Paso 6 Crear Modelos: Los modelos nos incorporan la interface, migration y vistas 48](#_Toc159848471)

[Paso 7 Modificar las Migraciones y agregarles los campos 49](#_Toc159848472)

[Paso 8 Poner los modelos globales en unguard 50](#_Toc159848473)

[Paso 9 Las relaciones 50](#_Toc159848474)

[Paso 10 Crear el CRUD con Filament creando un Resource 51](#_Toc159848475)

# Introducción

Introducción a Laravel:

Laravel es un popular marco de trabajo (framework) de código abierto para el desarrollo de aplicaciones web en PHP. Diseñado con un enfoque en la elegancia y la simplicidad, Laravel facilita la creación de aplicaciones web robustas y mantenibles. A continuación, se ofrecen algunos conceptos clave y características que hacen de Laravel una elección popular en el desarrollo web moderno:

Instalación:

Laravel se instala fácilmente a través de Composer, el gestor de dependencias de PHP. Con el siguiente comando, puedes crear un nuevo proyecto Laravel:

Bash

Copy code

composer create-project --prefer-dist laravel/laravel nombre-de-tu-proyecto

Artisan CLI:

Laravel incluye una herramienta de línea de comandos llamada Artisan, que simplifica tareas comunes de desarrollo, como migraciones de base de datos, generación de controladores y mucho más. Por ejemplo, puedes crear un nuevo controlador con el siguiente comando:

bash

Copy code

php artisan make:controller MiControlador

Enrutamiento:

Laravel ofrece un sistema de enrutamiento simple y expresivo. Puedes definir rutas en el archivo routes/web.php para manejar solicitudes HTTP. Aquí hay un ejemplo:

php

Copy code

Route::get('/bienvenida', 'ControladorBienvenida@index');

Controladores:

Los controladores en Laravel manejan la lógica de la aplicación. Pueden interactuar con modelos y vistas. Un ejemplo básico de un controlador podría ser:

php

Copy code

<?php

namespace App\Http\Controllers;

class ControladorBienvenida extends Controller

{

public function index()

{

return view('bienvenida');

}

}

Vistas:

Laravel utiliza el motor de plantillas Blade para crear vistas. Las vistas son responsables de mostrar el contenido HTML. Aquí tienes un ejemplo simple de una plantilla Blade:

html

Copy code

<html>

<body>

<h1>Hola, {{ $nombre }}</h1>

</body>

</html>

Eloquent ORM:

Eloquent es el ORM de Laravel que simplifica la interacción con la base de datos mediante el uso de modelos y consultas orientadas a objetos.

Middleware:

Laravel utiliza middleware para procesar solicitudes antes de llegar a las rutas de la aplicación. Esto es útil para tareas como autenticación, autorización y manipulación de solicitudes.

Autenticación y Autorización:

Laravel proporciona un sistema completo de autenticación con funciones como registro, inicio de sesión y restablecimiento de contraseña. También incluye capacidades de autorización para controlar el acceso a recursos.

Migraciones y Seeders:

Laravel facilita la gestión de la base de datos con migraciones, que permiten versionar el esquema de la base de datos, y seeders, que se utilizan para llenar la base de datos con datos de prueba.

Pruebas:

Laravel fomenta las pruebas unitarias y de características, proporcionando herramientas para facilitar la escritura y ejecución de pruebas automatizadas.

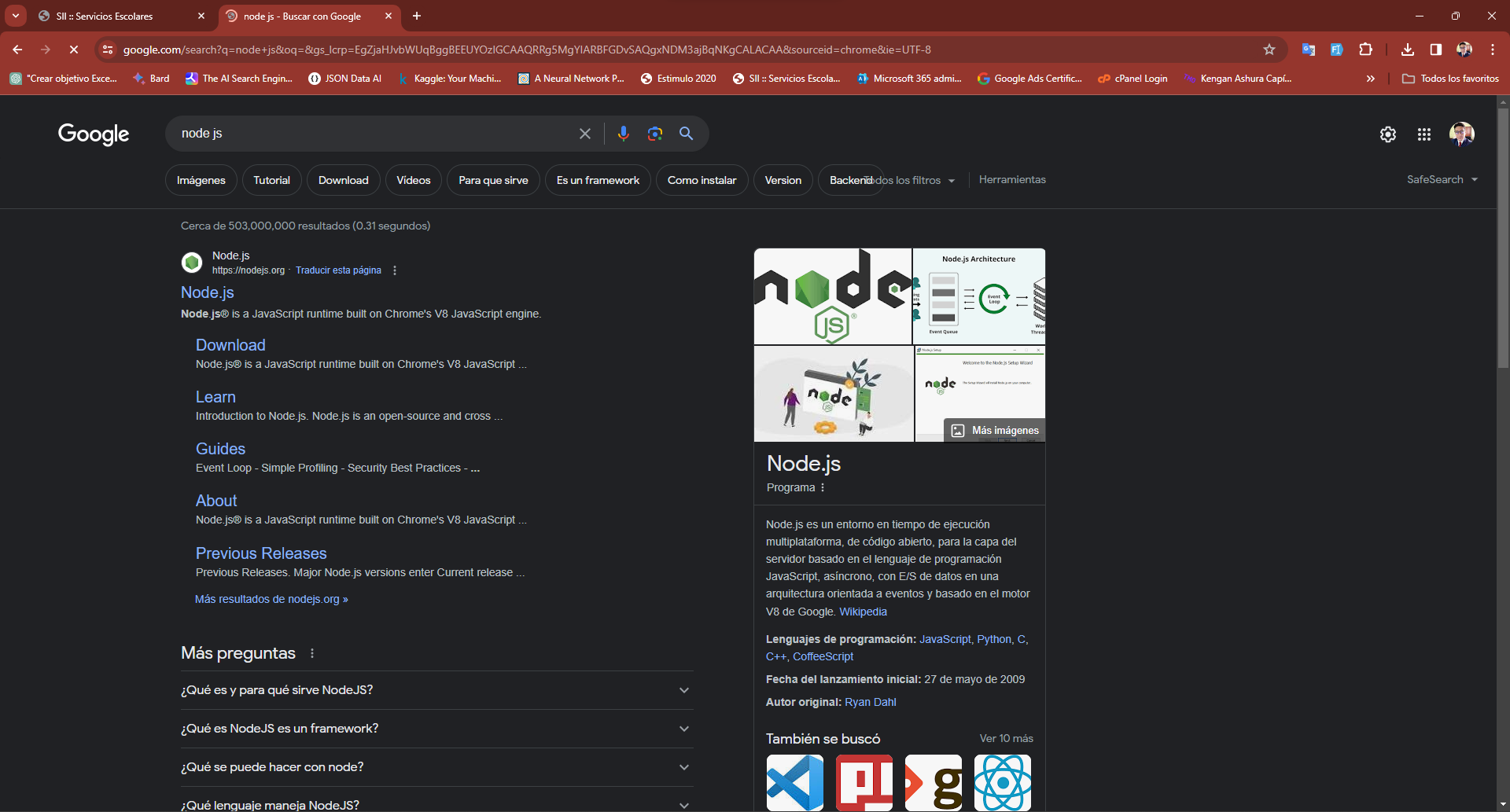
Laravel es una herramienta poderosa y flexible que ha ganado popularidad debido a su sintaxis elegante, su amplia comunidad y su enfoque en la productividad del desarrollador.

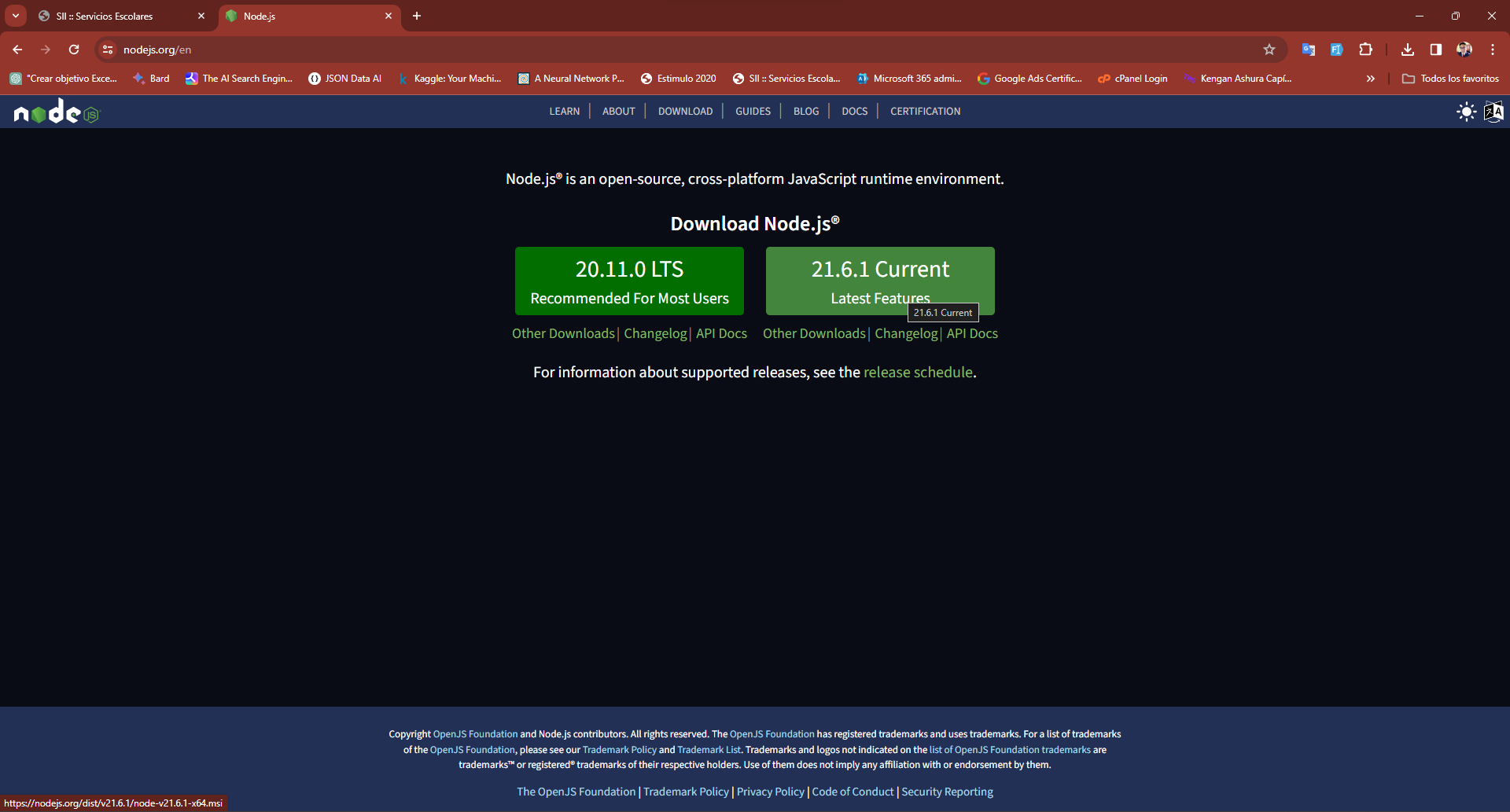
# Requisitos para Laravel

* VS Code
  + Laravel Artisan
  + Laravel Blade formatter
  + Laravel Blade Snippets
  + Laravel Create View
  + Laravel Extension Pack
  + Laravel Extra Intellisense
  + Laravel goto view
  + Laravel Intellisense
  + Laravel Snippets
  + Laravel-goto-components
  + Laravel-jump-cotroller
  + PHP Intelephense
  + Prettier-Code Formatter
  + Tailwind CSS InstelliSense
  + Vscode-icons
  + Background-cover
* Node Js - https://nodejs.org/dist/v21.6.1/node-v21.6.1-x64.msi
* Laragon - <https://github.com/leokhoa/laragon/releases/download/6.0.0/laragon-wamp.exe>
* Composer - <https://getcomposer.org/Composer-Setup.exe>

Proceso de Instalación

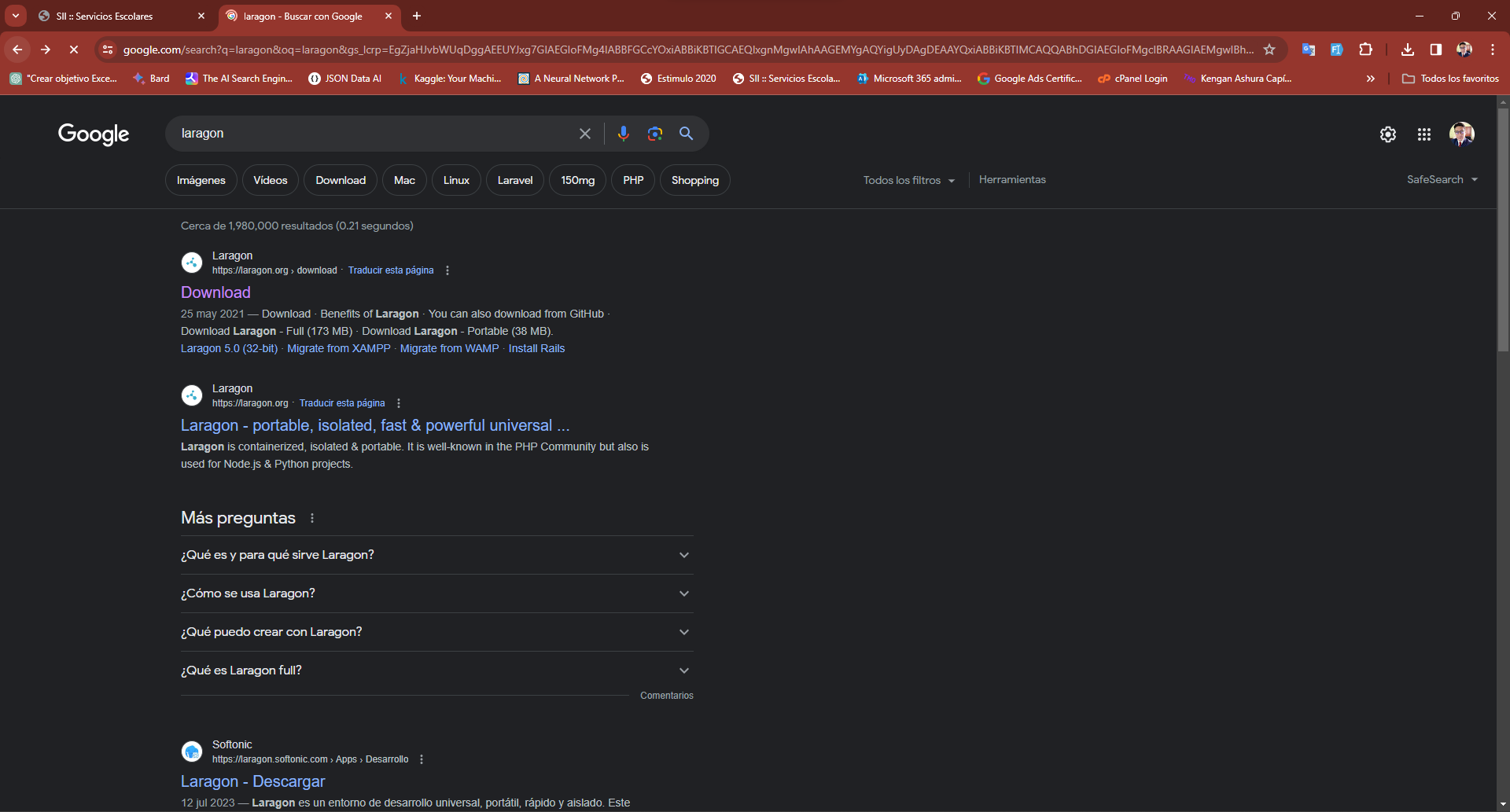
Instalación Node JS

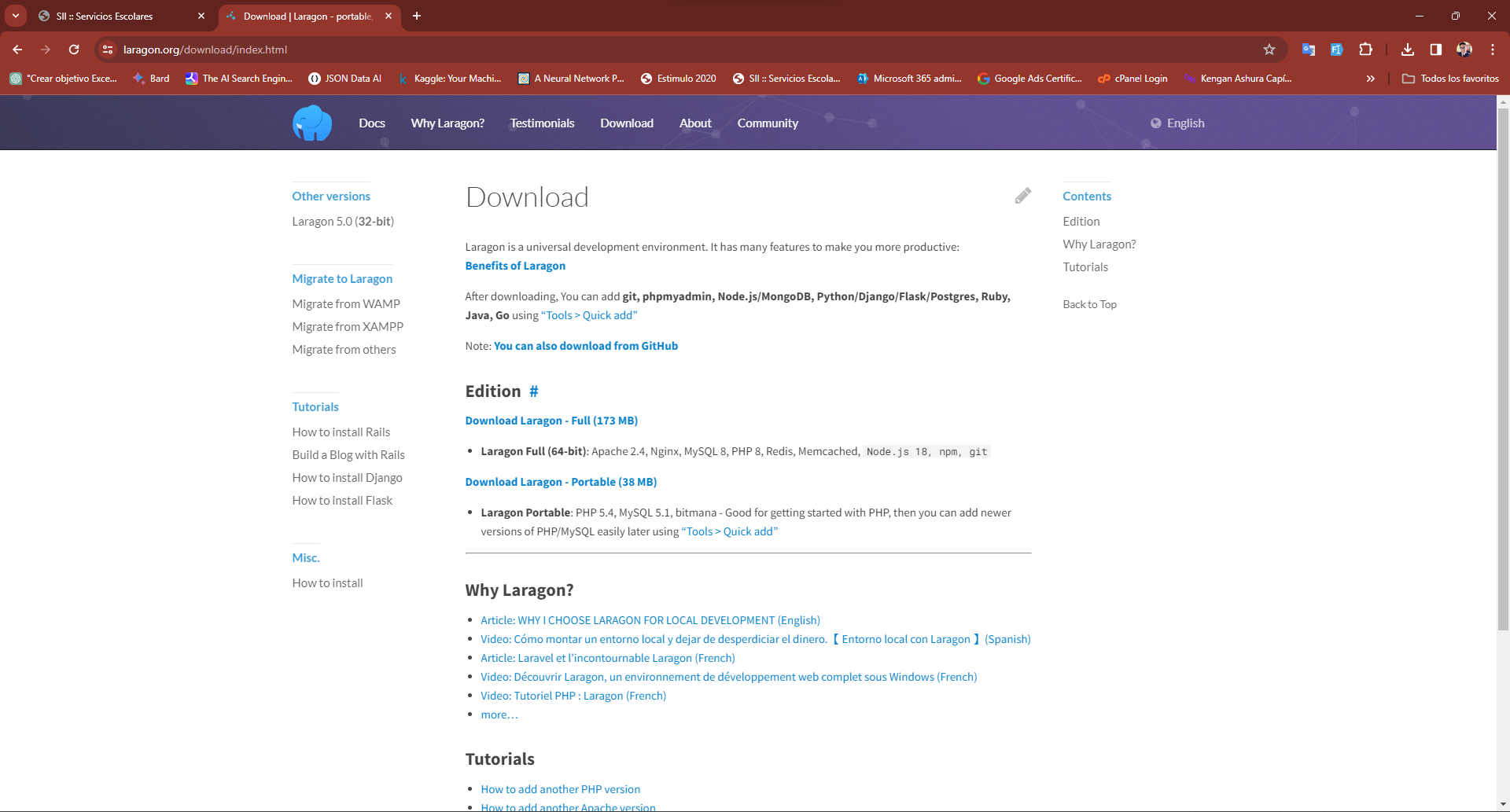


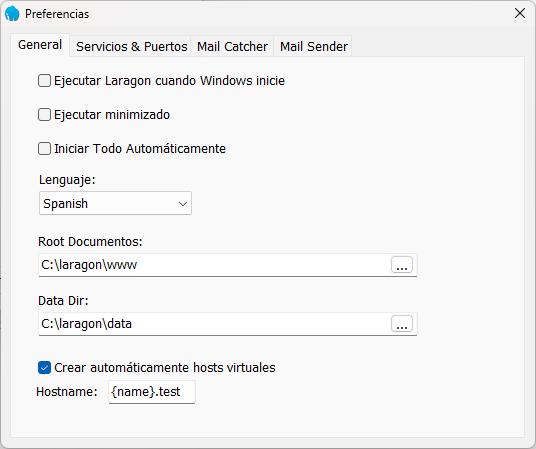


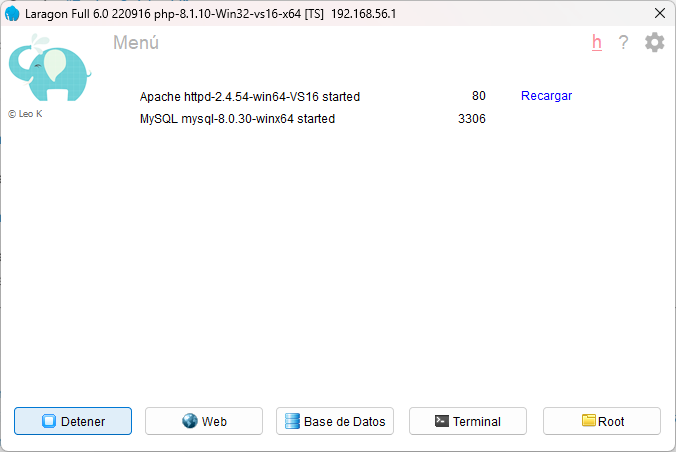


Instalación Laragon

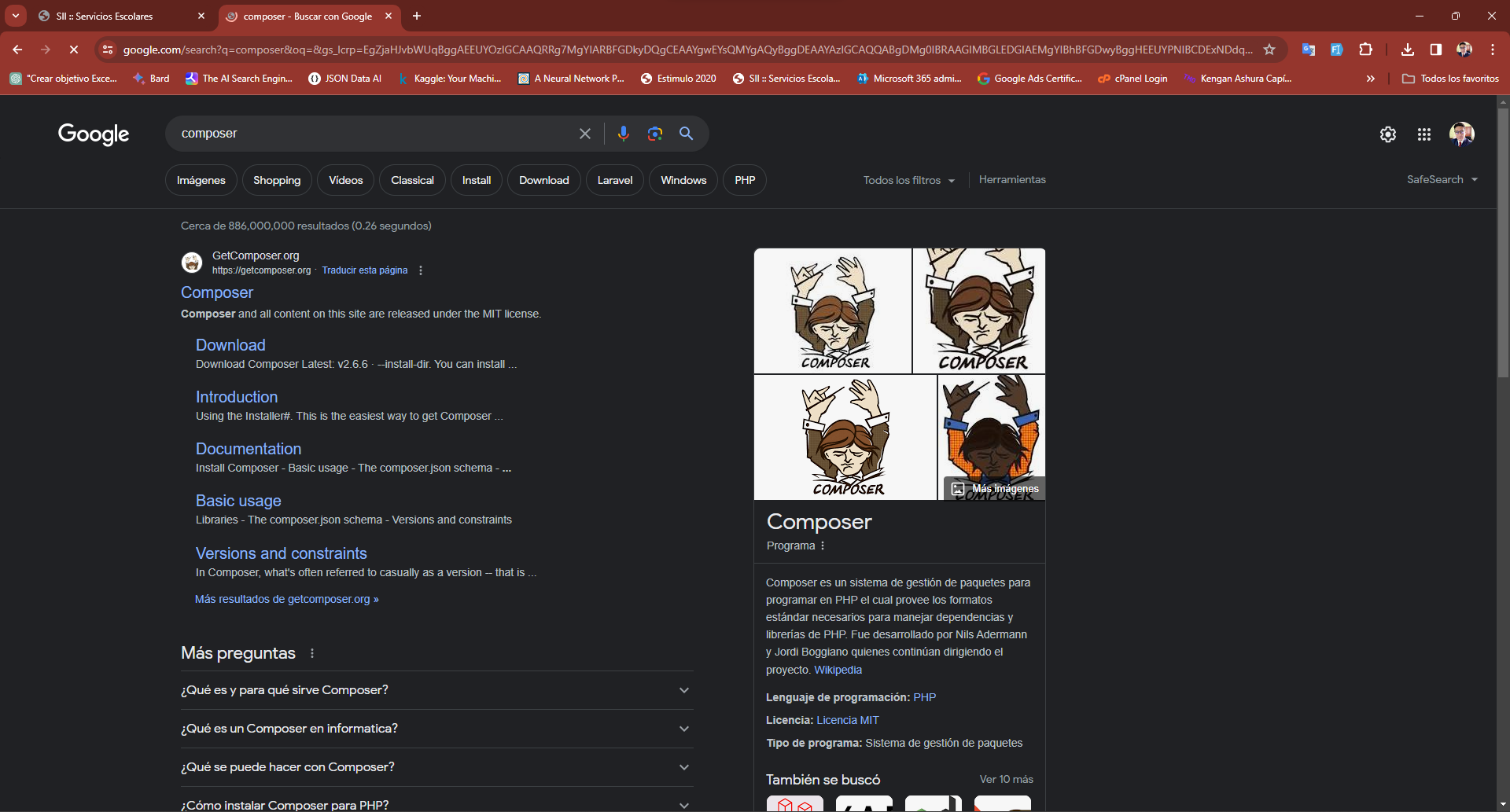


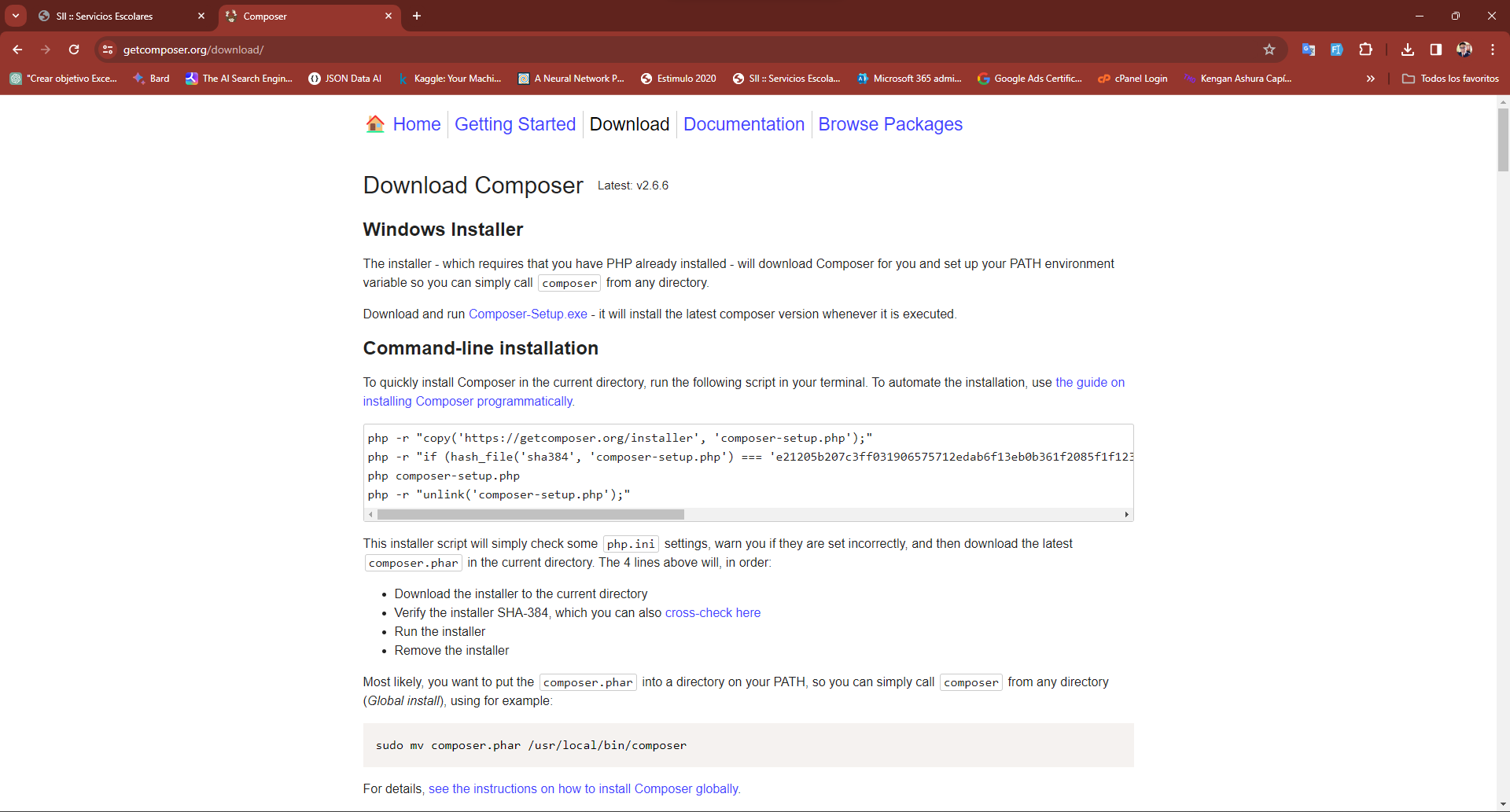






Instalación Composer

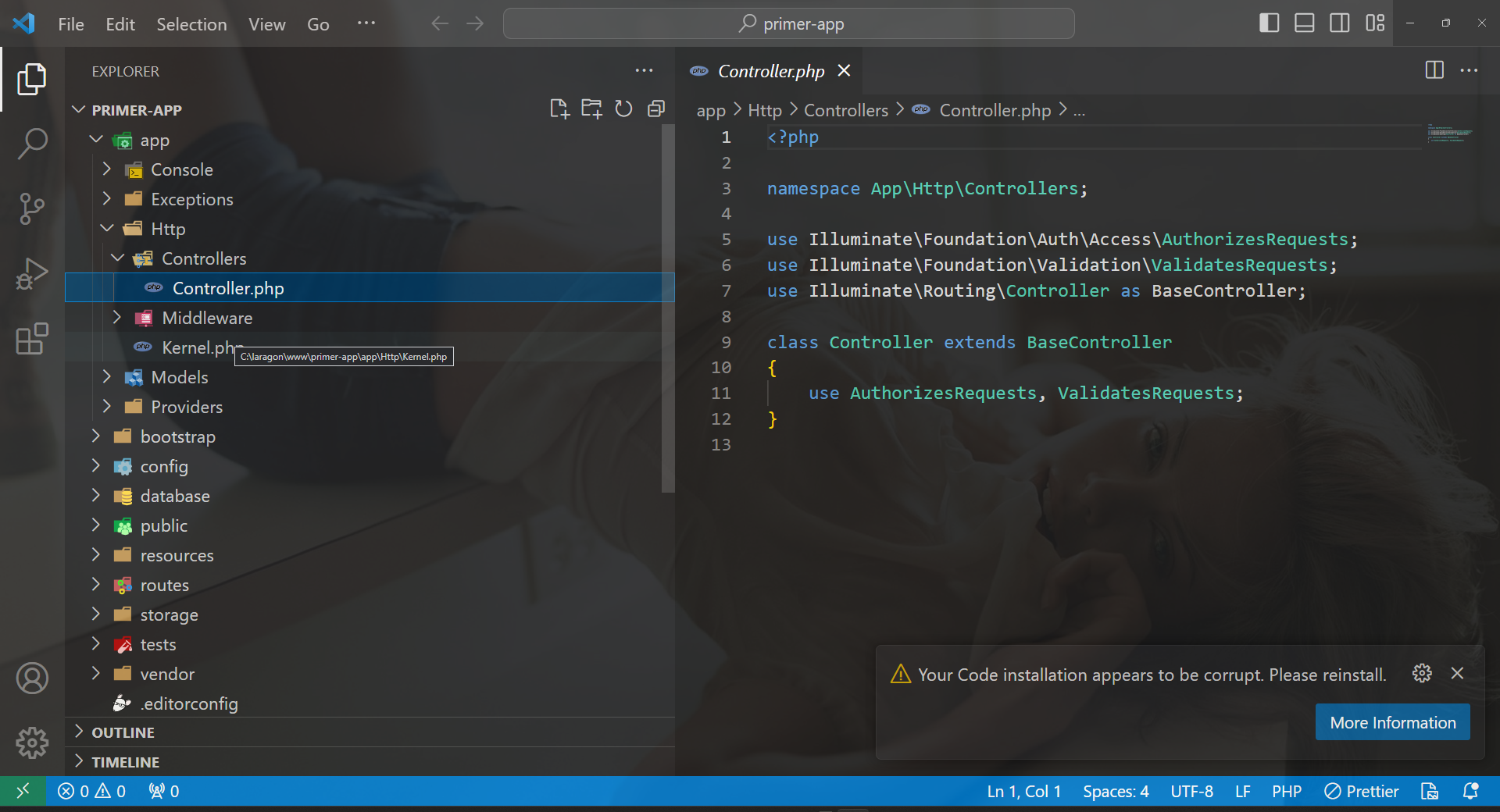




# Iniciando un Proyecto

## Comando para iniciar un proyecto

composer create-project laravel/laravel example-app



## Extensiones VSCode

* Laravel Blade formatter
* Laravel Blade Snippets
* Laravel Goto View
* Laravel Snippets
* PHP Intelephense
* Tailwind CSS IntelliSense
* Alpine.js IntelliSense
* GitHub Copilot

## Rutas en Laravel ¿Qué son y para que sirven?

Las rutas en Laravel son una herramienta fundamental para el desarrollo de aplicaciones web. En el primer capítulo del curso "Laravel desde cero", aprenderás qué son las rutas en Laravel y para qué sirven. Las rutas permiten definir las URLs de una aplicación y mapearlas a controladores que procesan las solicitudes del usuario. En este capítulo, conocerás cómo crear rutas básicas y avanzadas, definir parámetros en las rutas y cómo utilizar middleware para aplicar filtros y restricciones en las solicitudes.

Ejemplo 1: Vamos a crear una ruta llamada ejemplo que nos muestre un texto que diga ‘Bienvenidos al ejemplo 1’.

Paso 1: ir a la carpeta routers/web.php

Paso 2: definir la ruta con el código

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

    return view('welcome');

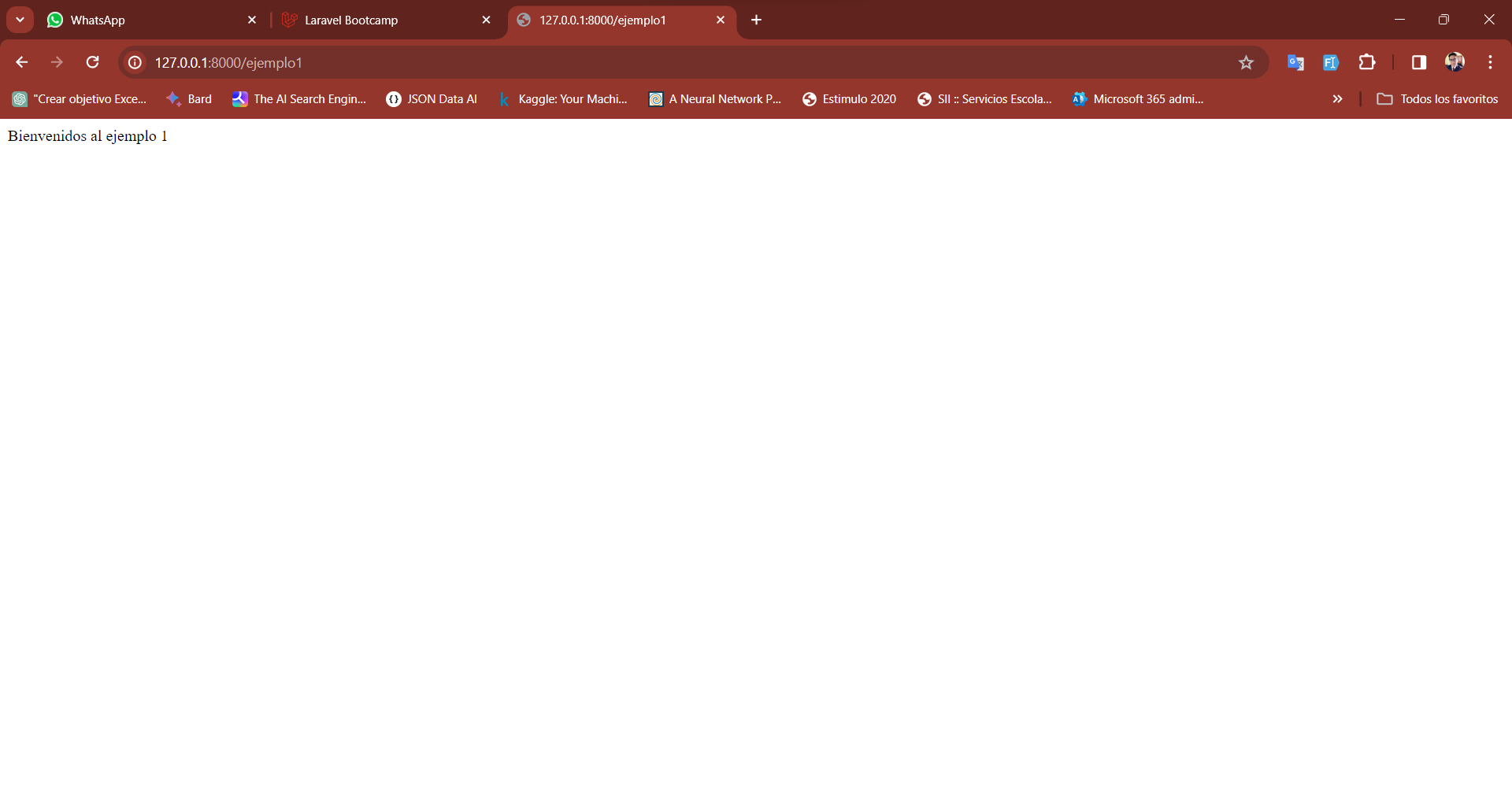
});

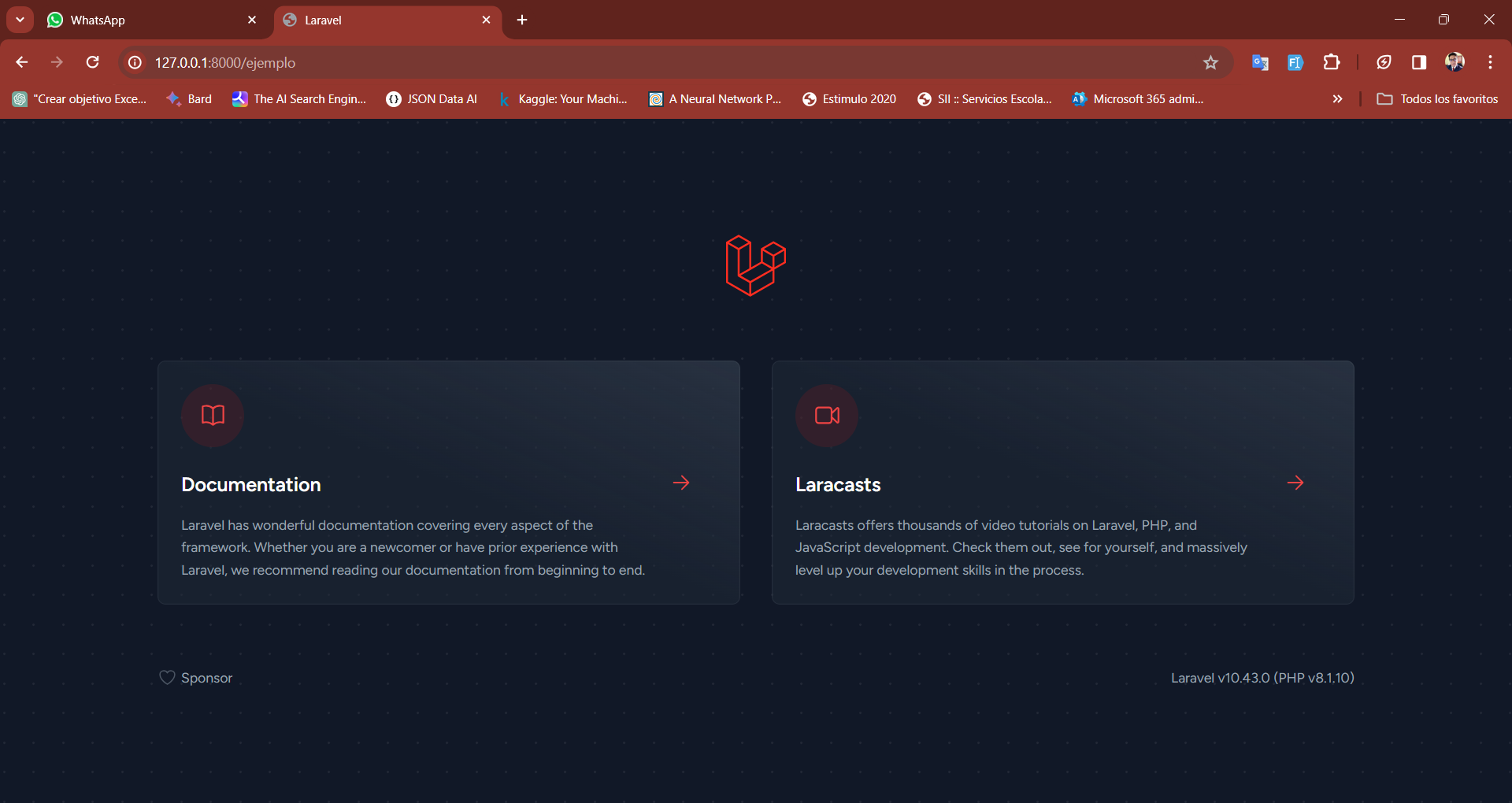
Route::get('ejemplo1', function () {

    return 'Bienvenidos al ejemplo 1';

});

Resultado:





RUTAS

Cuando iniciamos un proyecto en laravel por default tenemos una ruta llamada welcome, en este momento tenemos mas rutas por que instalamos la auth de usuarios, las rutas son la activación de las paginas que despliega laravel. Ejemplo:

# Comando de Instalación Auth

\*/ composer require laravel/breeze --dev

\*/ php artisan breeze:install blade --dark

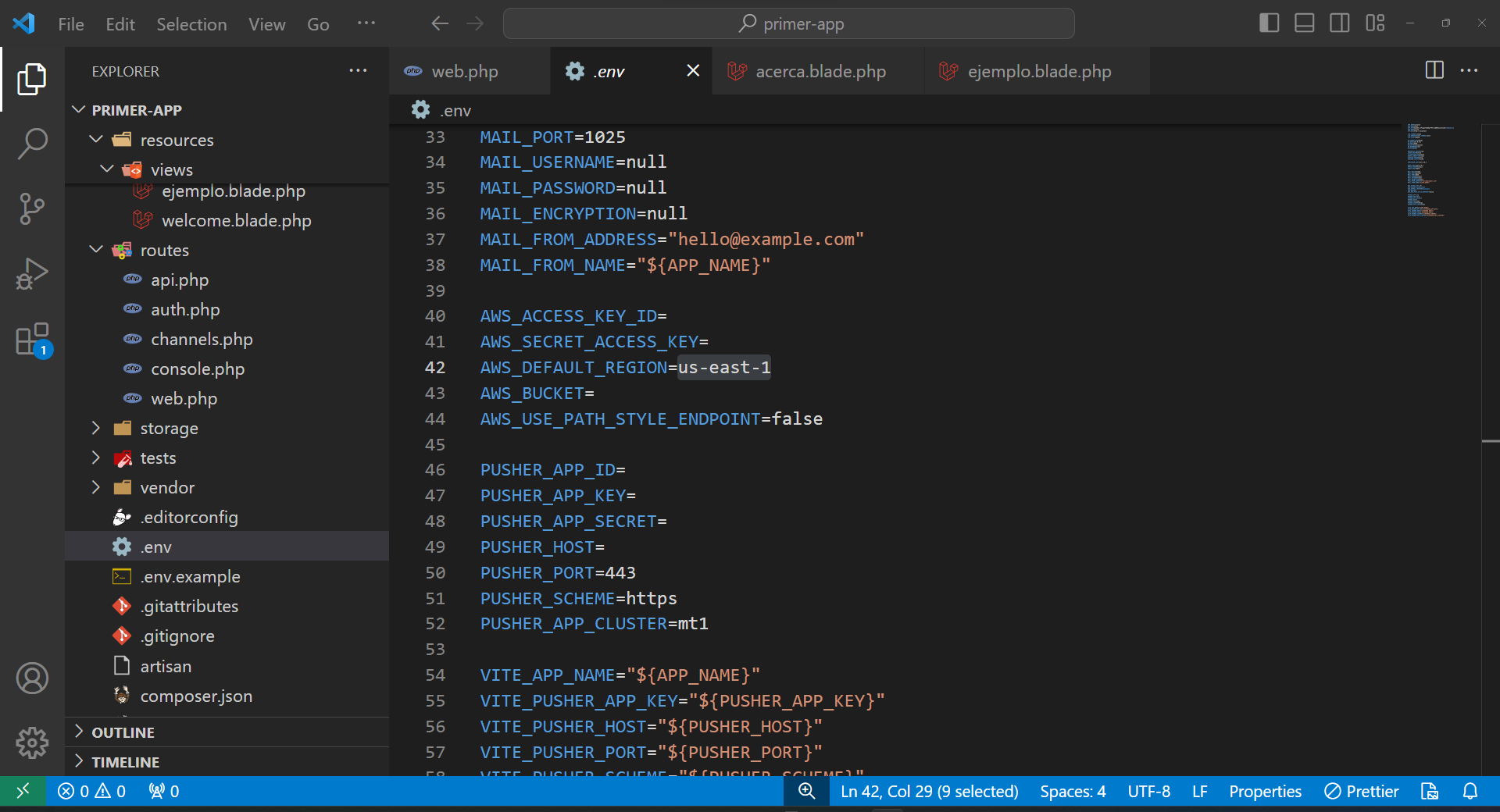
Abrimos el archivo env para colocar BD y corremos la migración

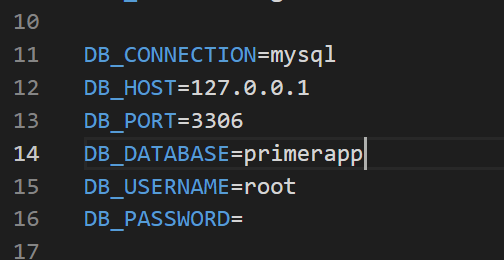
\*/ php artisan migrate

Ahora Creamos un registro de usuario.

Nota para la Auth.

Después de instalar la Auth debemos configurar nuestro archivo .env para determinar la base de datos de mi proyecto y corroborar las tablas de usuario.



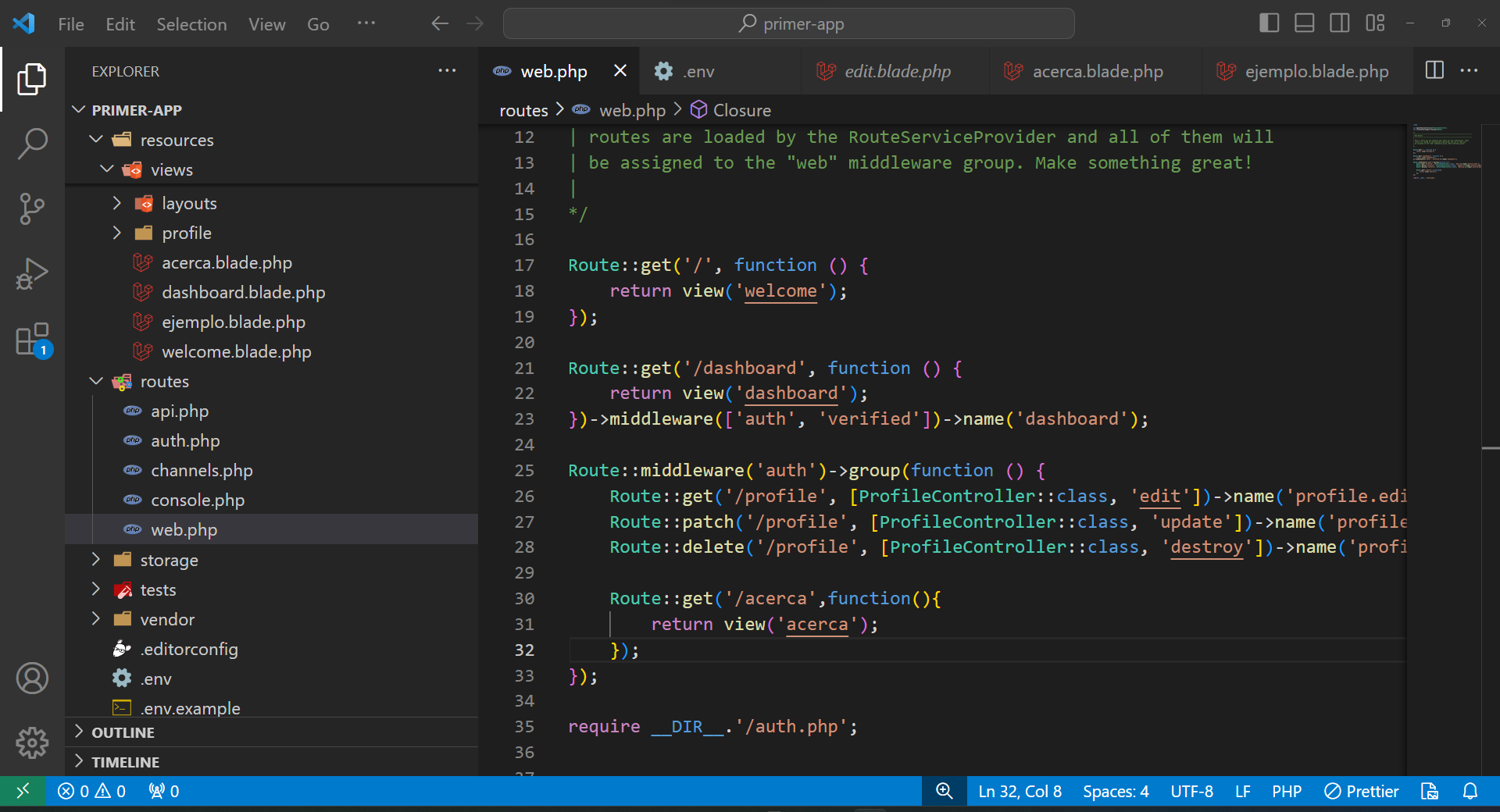


# Middleware

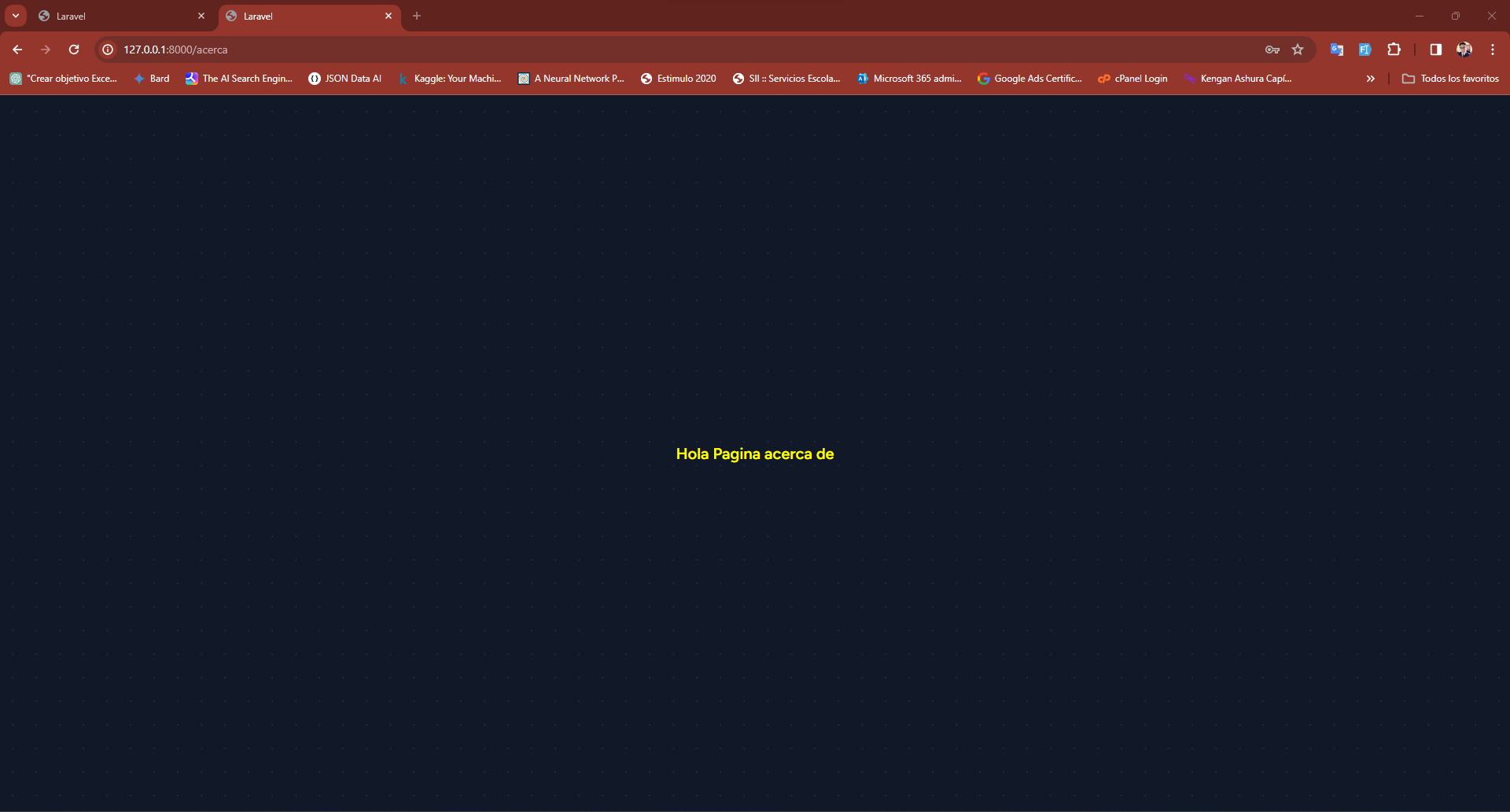
Es una capa intermedia que se sitúa entre la solicitud HTTP entrante y la aplicación. Se utiliza para realizar acciones específicas antes o después de que la solicitud alcance la lógica de la aplicación principal. Los middlewares son una parte fundamental del sistema de manejo de solicitudes HTTP en Laravel y ofrecen una manera conveniente de filtrar, modificar o manipular la solicitud y la respuesta.

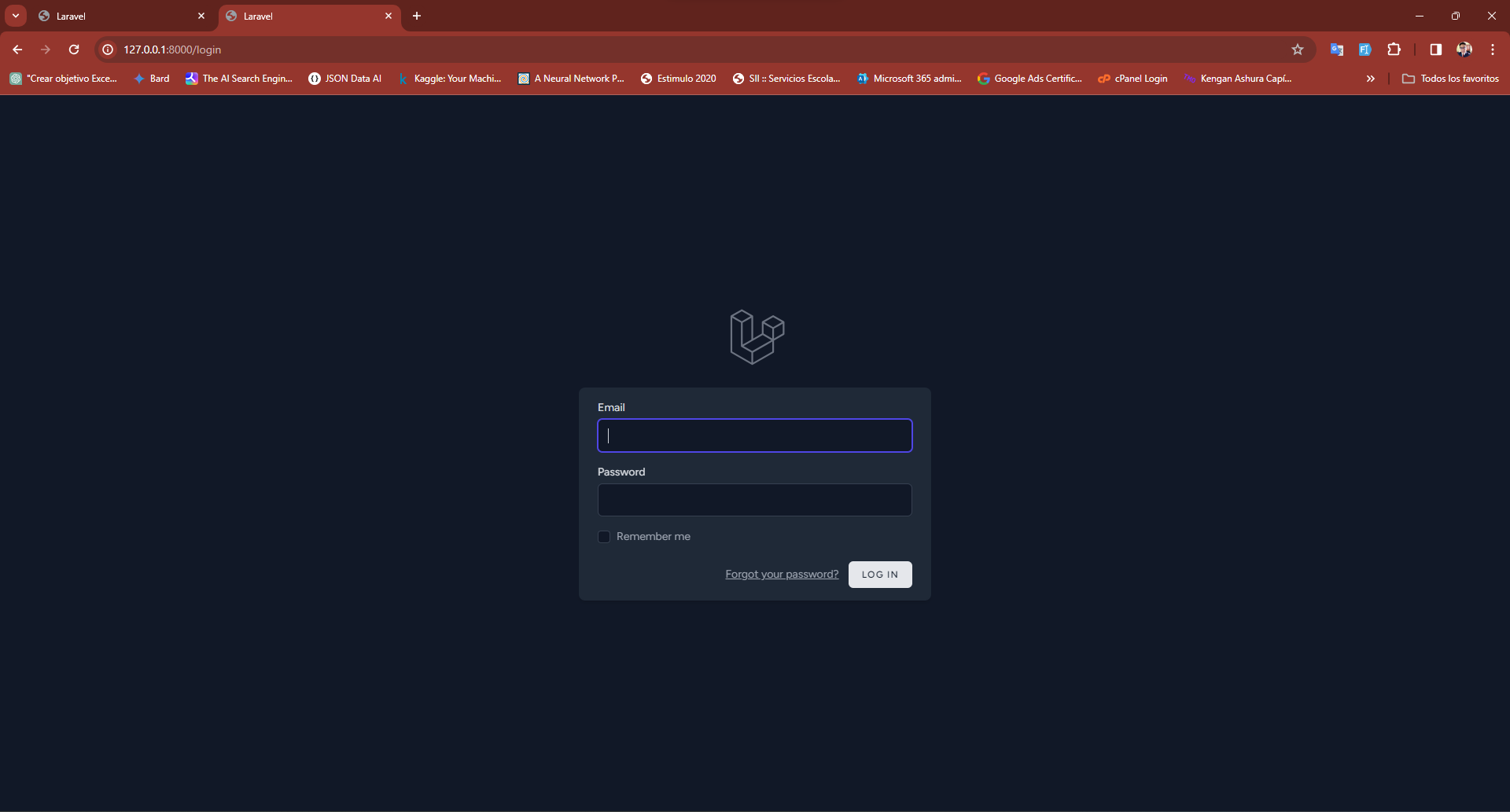
Practica Pasar la pagina Acerca de a un middleware

Primer paso meter la ruta dentro del middleware



Y probamos con sesión activa podemos ver la ruta sin ella nos solicitara el login.





# FORMULARIOS EN LARAVEL

Paso1 Creamos la ruta para nuestro servicio llamado chirps

//Creando la ruta a chirps

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

    return 'Bienvenidos a la pagina chirps';

})->name('chirps.index');

## Heredar Estilo

Ahora vamos a darle el estilo de plantilla que tiene nuestro sitio para ello hay que tomar las clases de Tailwind que usa laravel. Esta la tomare en layout/dashboard.blade.php y se las pondré a mi archivo chirps / index.blade.php

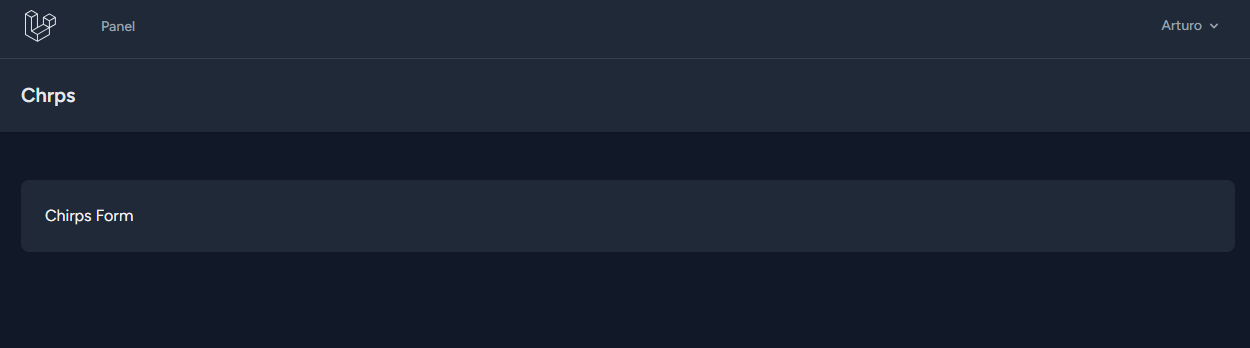
<x-app-layout>

<h1>Bienvenidos al primer ejemplo de una vista.</h1>

</x-app-layout>

## Links de Navegación

Agregaremos unos Links de navegación para pasar entre el dashboard y los chirps y para que se diferencien aun mas le agregare un forms al chirps.



Nos vamos al archivo de navegación que esta en layouts/navigation.blade.php en la sección de Navigation Links y agregamos a chirps ahora hay que recordar que en mis rutas yo le coloque un name así que es hora de usarlo:

<!-- Navigation Links -->

                <div class="hidden space-x-8 sm:-my-px sm:ms-10 sm:flex">

                    <x-nav-link :href="route('dashboard')" :active="request()->routeIs('dashboard')">

                        {{ \_\_('Dashboard') }}

                    </x-nav-link>

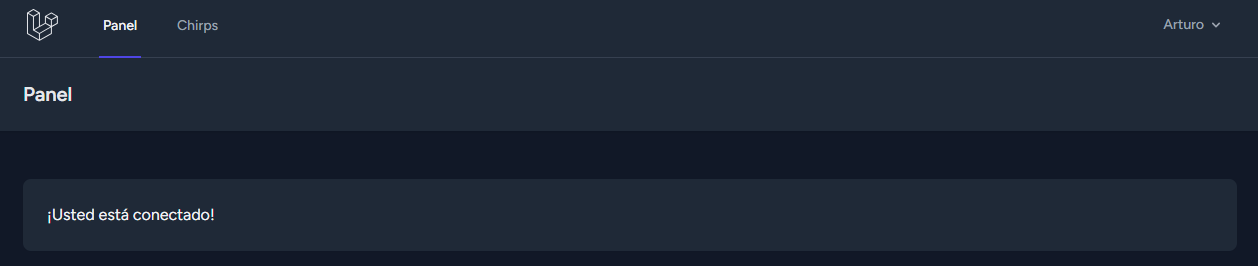
                    <x-nav-link :href="route('chirps.index')" :active="request()->routeIs('chirps.index')">

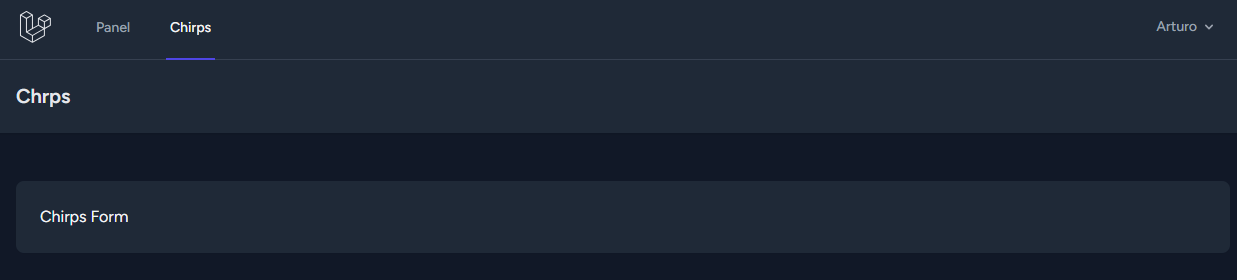
                        {{ \_\_('Chirps') }}

                    </x-nav-link>

                </div>

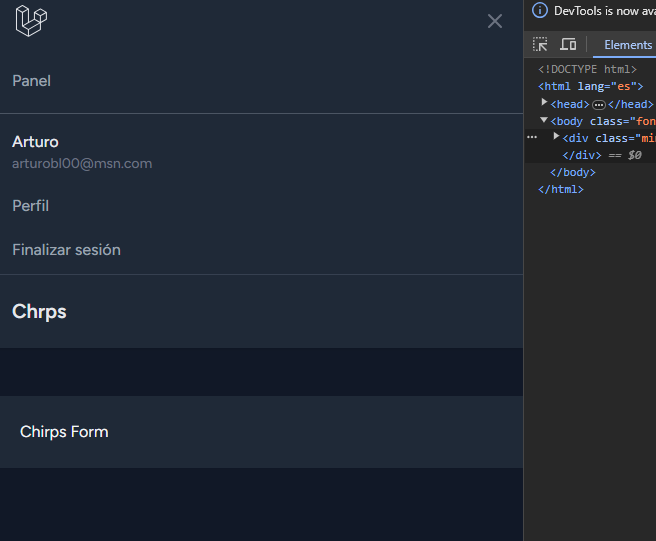
            </div>





## Link de Navegación móvil

Si hacemos responsivo el sitio nos damos cuenta que Chirps no aparece



Editamos la sección

<!-- Responsive Navigation Menu -->

    <div :class="{'block': open, 'hidden': ! open}" class="hidden sm:hidden">

        <div class="pt-2 pb-3 space-y-1">

            <x-responsive-nav-link :href="route('dashboard')" :active="request()->routeIs('dashboard')">

                {{ \_\_('Dashboard') }}

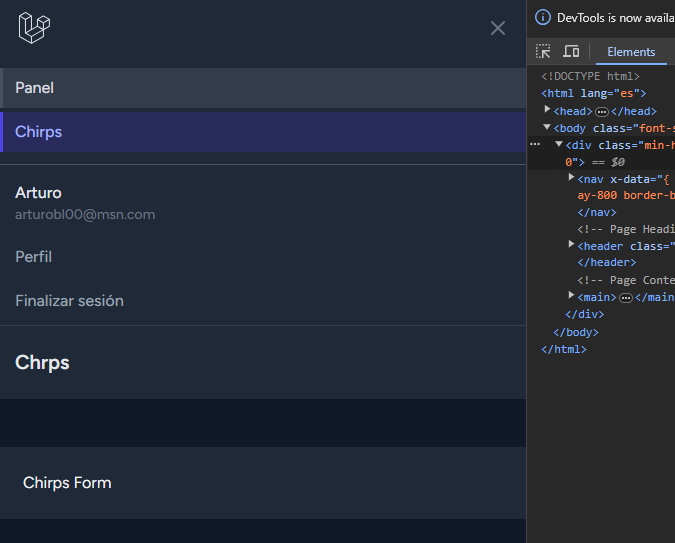
            </x-responsive-nav-link>

            <x-responsive-nav-link :href="route('chirps.index')" :active="request()->routeIs('chirps.index')">

                {{ \_\_('Chirps') }}

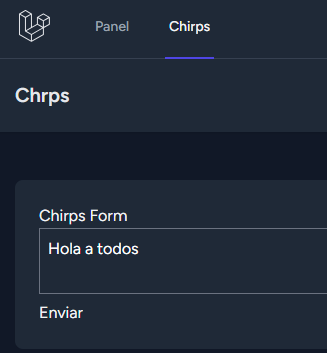
            </x-responsive-nav-link>

        </div>



## Casos de envío de datos

### Caso 1 enviar los datos sin un name y un método



En este caso el resultado de la url queda así: <http://127.0.0.1:8000/chirps>? No hay datos que se estén recibiendo.

Caso 2: Enviar los datos con name pero sin especificar un método que por default laravel usa get.

Resultado: <http://127.0.0.1:8000/chirps?messege=Arturo>

### Caso 3: especificar el método al formulario en post

Para que funcione con el método post no solo se debe poner el método en el formulario también debemos crear una ruta post en nuestro web.php. Aparte al formulario hay que poner un token @csfr.

Paso uno la ruta

Route::post('/chirps', function(){

        return 'Procesando el chirps';

    });

Paso dos form en post y token

<form method="post">

@csrf

      <textarea class="bg-gray-800 text-white border border-gray-700 p-2 w-full h-24" name="messege"></textarea>

      <button>Enviar</button>

</form>

Caso 4: Acceder a los datos que ingreso el usuario. En laravel tenemos un método que se llama request() este se coloca en la función de la ruta a esta le pasamos el name del formulario y la almacenamos dentro de una variable.

 Route::post('/chirps', function(){

        $messege = request('messege');

        //Insertar en la base de datos

        return 'Procesando el chirps - '. $messege;

    });

Voy a dar un poco de formato a mi formulario

<div class="py-12">

        <div class="max-w-7xl mx-auto sm:px-6 lg:px-8">

            <div class="bg-white dark:bg-gray-800 overflow-hidden shadow-sm sm:rounded-lg">

                <div class="p-6 text-gray-900 dark:text-gray-100">

                    {{ \_\_("Chirps Form") }}

                    <form method="post">

                        @csrf

                        <textarea

                        placeholder="{{\_\_('Whats on you mind?')}}"

                        class="bg-gray-800 text-white border border-gray-700 p-2 w-full h-24" name="messege"></textarea>

                        <x-primary-button>Enviar</x-primary-button>

                    </form>

                </div>

            </div>

        </div>

    </div>

## INSERCIÓN DE DATOS EN BD

Lo que necesitamos para que laravel envíe datos de un formulario a una tabla de la base de datos es crear un modelo.

## Modelos en Laravel 10

Los modelos se refieren a una clase de PHP que interactúa con una tabla de la base de datos. Los modelos en Laravel están diseñados para facilitar el acceso y la manipulación de datos en la base de datos. Cada modelo está asociado con una tabla específica en la base de datos y se utiliza para realizar operaciones como la recuperación de registros, la creación de nuevos registros, la actualización de registros existentes y la eliminación de registros.

Comando

\*/ php artisan make:model Chirp -mrc

Para nombrar un modelo debemos usar PascalCase y deben estar escritos en singular los modelos trabajan con una migración y un controlador la migración manda o crea la tabla y el controlador contiene las opciones crud.

## Migraciones

Las migraciones se crean y gestionan mediante la línea de comandos de Artisan, el sistema de comandos de Laravel. Al generar una migración, se crea un archivo en el directorio database/migrations que contiene dos métodos: up y down.

**up:** Este método especifica las operaciones que deben realizarse cuando se ejecuta la migración. Por lo general, se utiliza para crear tablas, agregar columnas o realizar otros cambios en la estructura de la base de datos.

**down:** Este método especifica las operaciones que deben realizarse cuando se revierte la migración. Es útil para deshacer los cambios realizados en el método up.

El comando para ejecutar las migraciones una vez creadas y modificadas método **UP**:

\*/ php artisan migrate

El comando para revertir las migraciones método **down:**

\*/ php artisan migrate:rollback

Para nuestro ejemplo agregaremos un campo a la tabla que crearemos llamado message el código queda así

public function up(): void

    {

        Schema::create('chirps', function (Blueprint $table) {

            $table->id();

            $table->string('message');

            $table->timestamps();

        });

    }

Ahora necesitamos vincular el mensaje con quien lo escribió para eso vamos a realizar una relación con foreignId()

Detallamos los datos

foreugnId: establece el nombre del campo que se va a relacionar o llave foránea

constrained: restringe la llave foránea para asegurar la integridad referencial

cascadeOnDelete: para eliminar los registros que acompañan al registro de la llave foránea.

public function up(): void

    {

        Schema::create('chirps', function (Blueprint $table) {

            $table->id();

            $table->string('message');

            $table->foreignId('user\_id')->constrained()->cascadeOnDelete();

            $table->timestamps();

        });

    }

## Eloquent

Eloquent es el ORM (Object-Relational Mapping) incluido en Laravel. Proporciona una implementación activa de los patrones ActiveRecord para trabajar con bases de datos relacionales. En otras palabras, Eloquent permite interactuar con la base de datos utilizando objetos en lugar de escribir consultas SQL directamente.

Lo modelos pertenecen a Eloquent y me permiten

* User::get()
* User::where()
* User::orderBy()
* User::créate()
* User:;update()
* User::delete()

Para usar el modelo nos vamos al archivo de rutas web.php y lo importamos

use App\Http\Controllers\ProfileController;

use Illuminate\Support\Facades\Route;

use App\Models\Chirp;

Ahora en la ruta post de nuestro formulario vamos a usar el modelo créate de eloquent

Route::post('/chirps', function(){

        //Insertar en la base de datos

        Chirp::create([

            'message' => request('messege'),

            'user\_id' => auth()->id(),

        ]);

        return to\_route('chirps.index');

    });

Explicamos mandamos a llamar al modelo Chirp con su propiedad créate dento le pusimos un arreglo que nos incluye el nombre del campo de la tabla y se le asigna el request que ya explicamos el cual lo toma del name del elemento de nuestro formulario y el método auth que toma el id del usuario.

Si ahorita ejecutamos nos mandara un error esto por que no hemos activado en el Modelo el método fillable.

Para que esto funcione además en el modelo Chirp debemos agregar el método fillable que me permitirá identificar cuales son los campos permitidos.

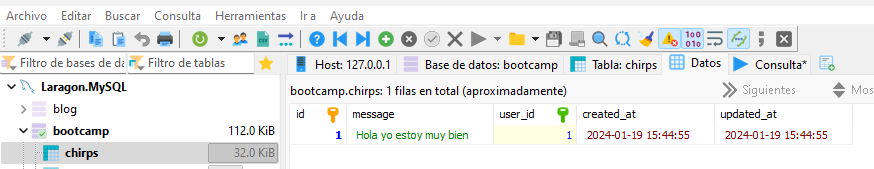
protected $fillable = [

        'message',

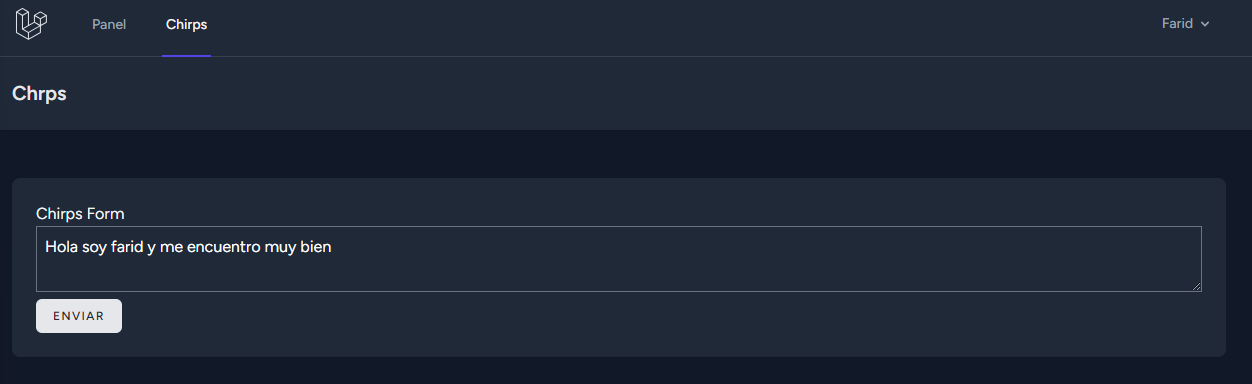
        'user\_id',

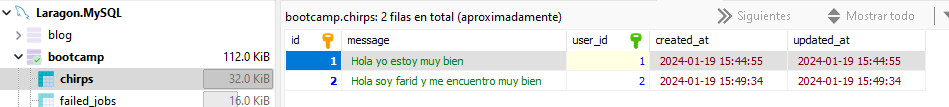
    ];

Ejecutamos nuestro formulario y notamos que el registro fue enviado



Ahora probemos creando otro usuario para que el registro tenga un id = 2 y probemos nuestros formulario.





Ya Tenemos un Formulario, una ruta, una migración, un modelo y podemos insertar datos en una tabla.

Examen unidad 1

Tomando en cuenta los conceptos adquiridos hasta el momento crea el siguiente agregado:

1.- Crea un MVC llamado Examen con autentificación de usuario.

2.- Crea un formulario donde se pueda registrar un Asunto y un mensaje.

3.- Agrega el contenido del formulario en una migración llamada Examen que registre User\_id, Asunto y Mensaje.

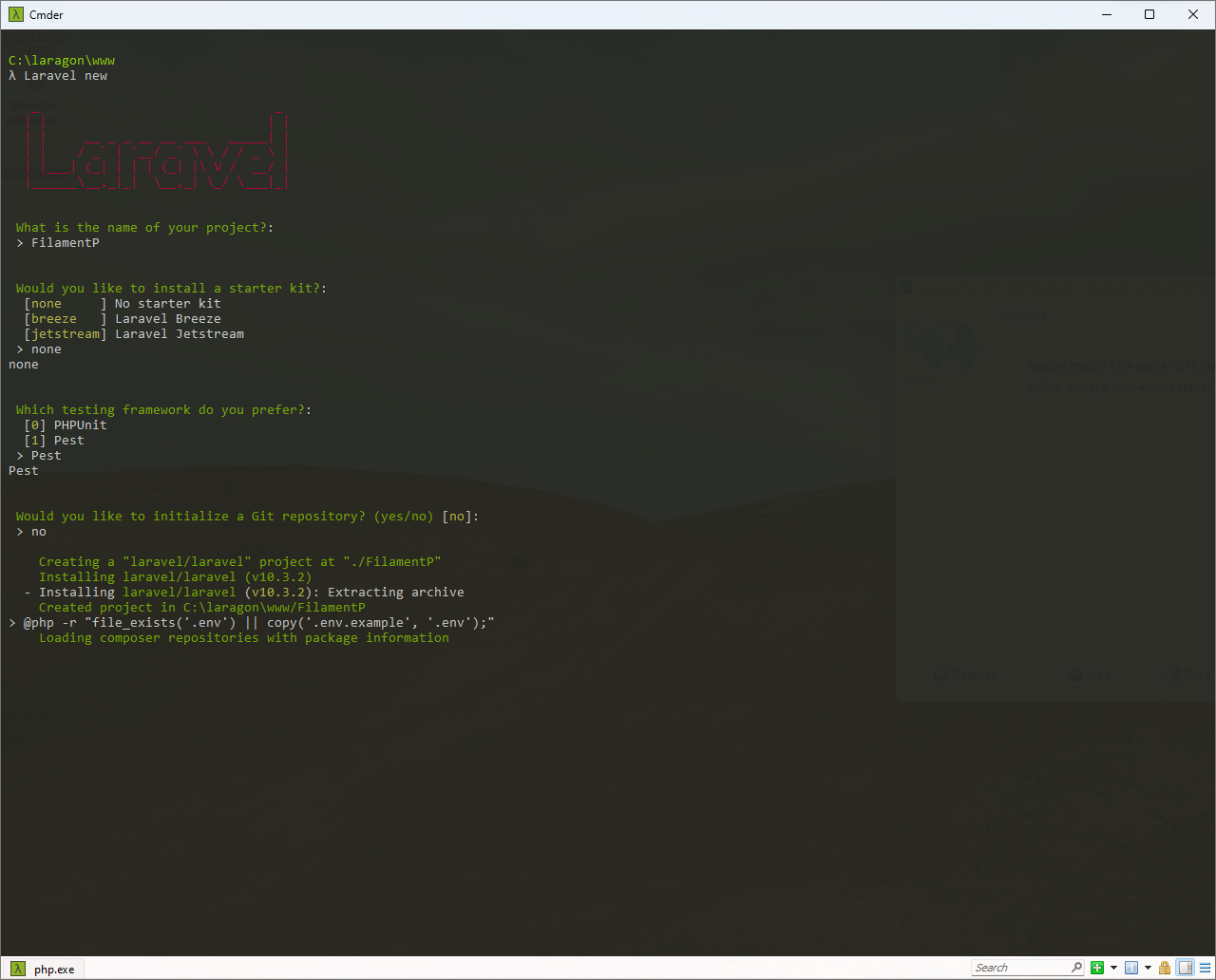
4.- Integra en el proyecto Primer-app tu Programa y adjunta las evidencias en el portafolio mínimo 5 imágenes del proceso.

Imágenes:

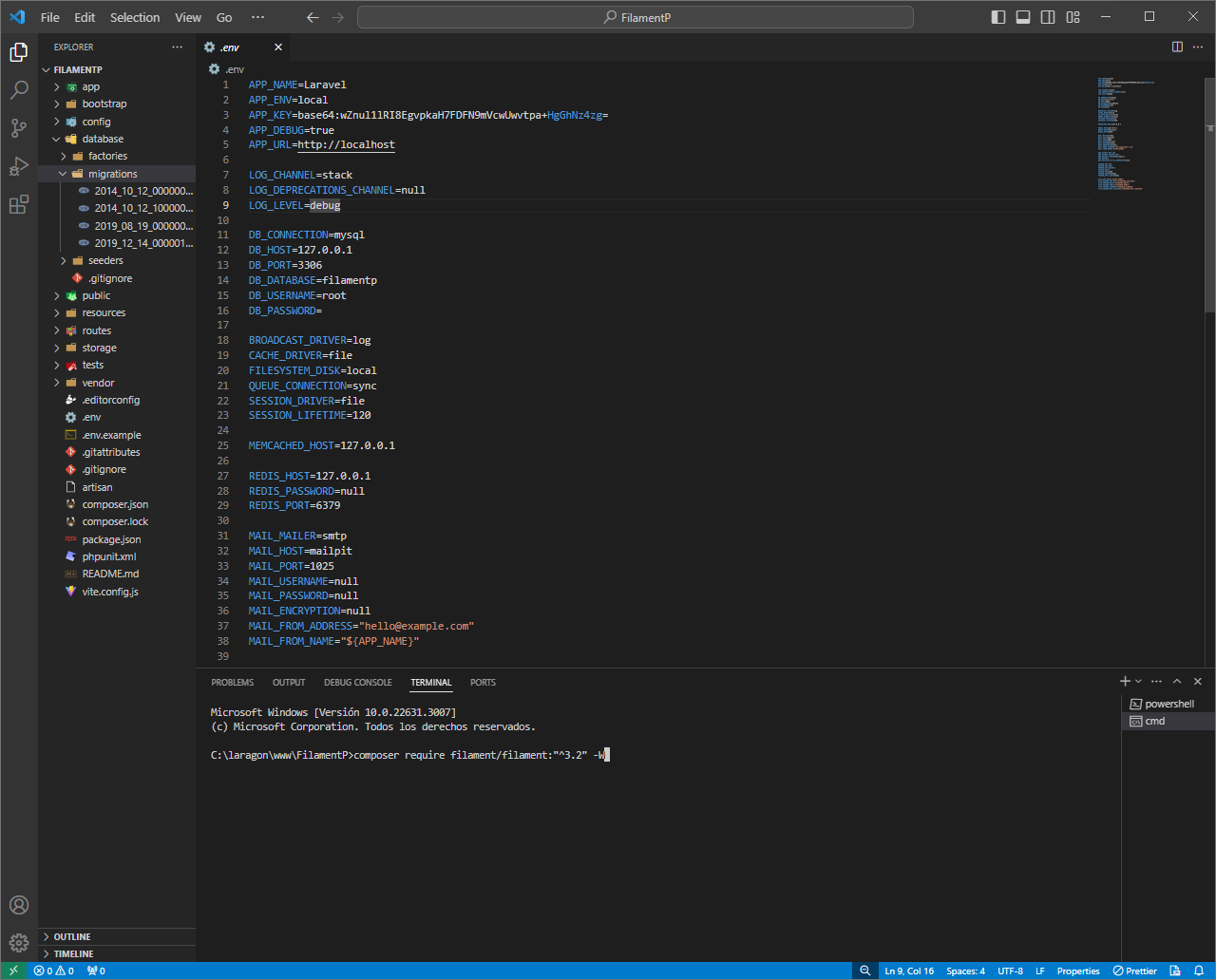
# Unidad 2 - Componentes y librerías.

## Introducción a Filament

Instalación por comandos

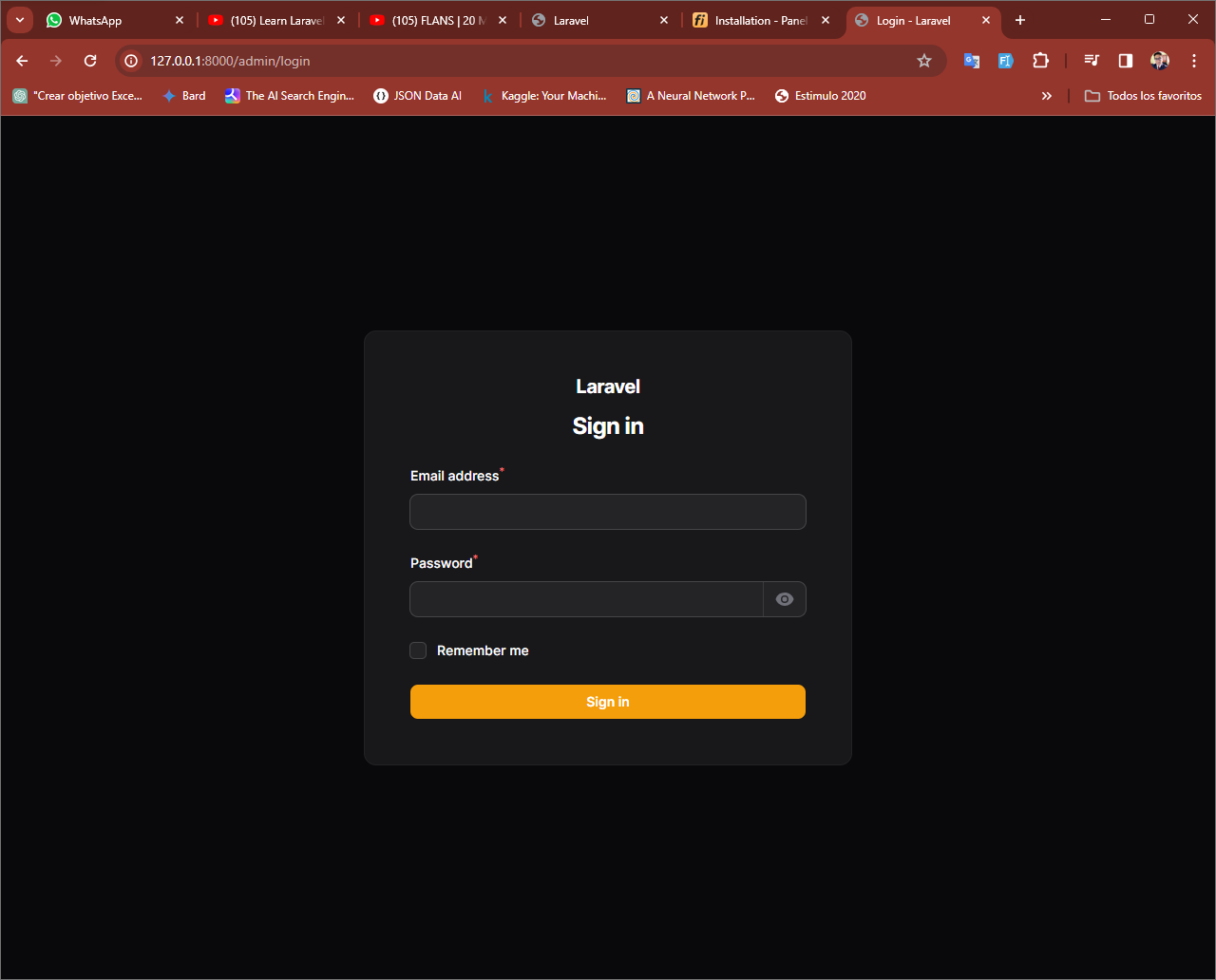


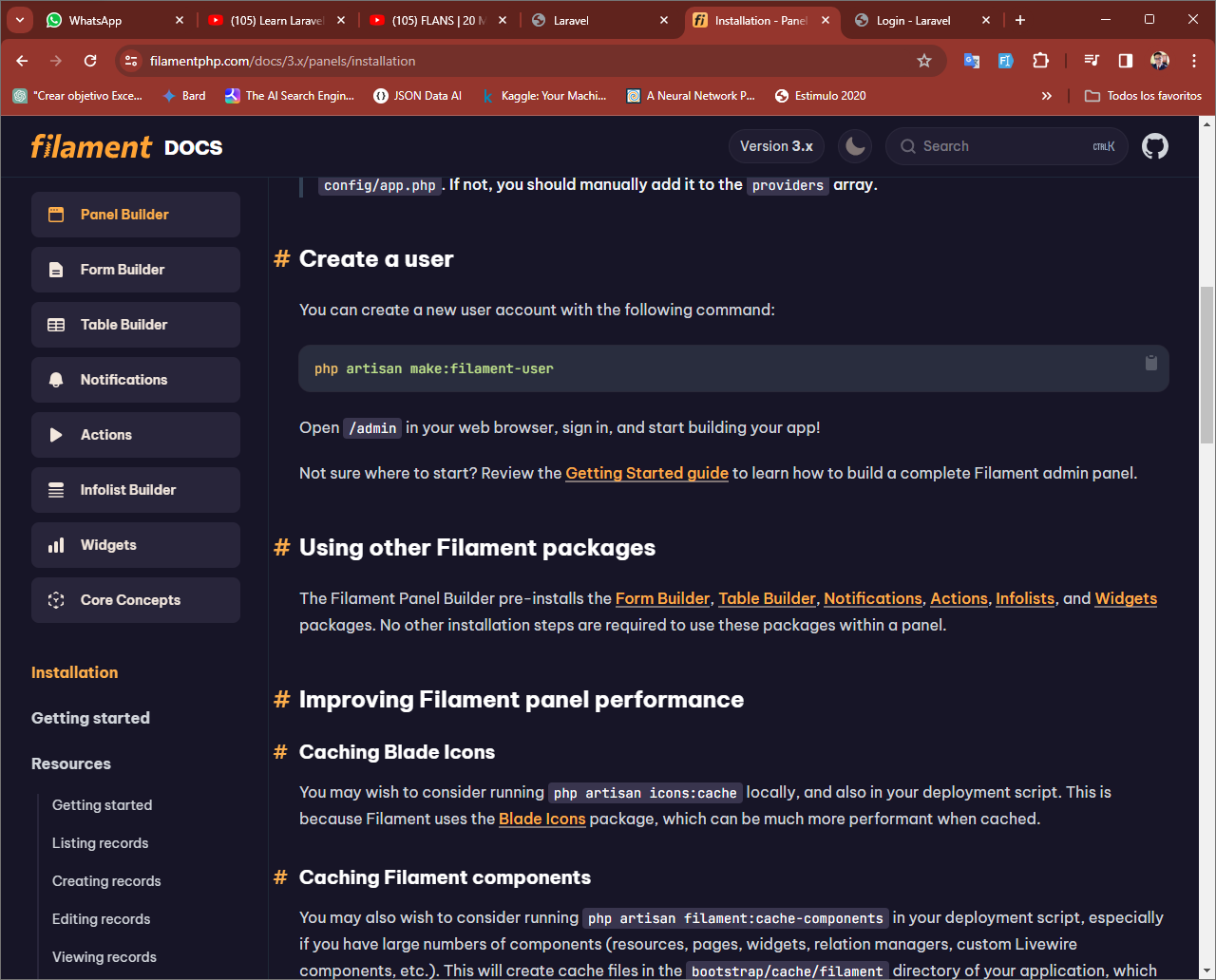


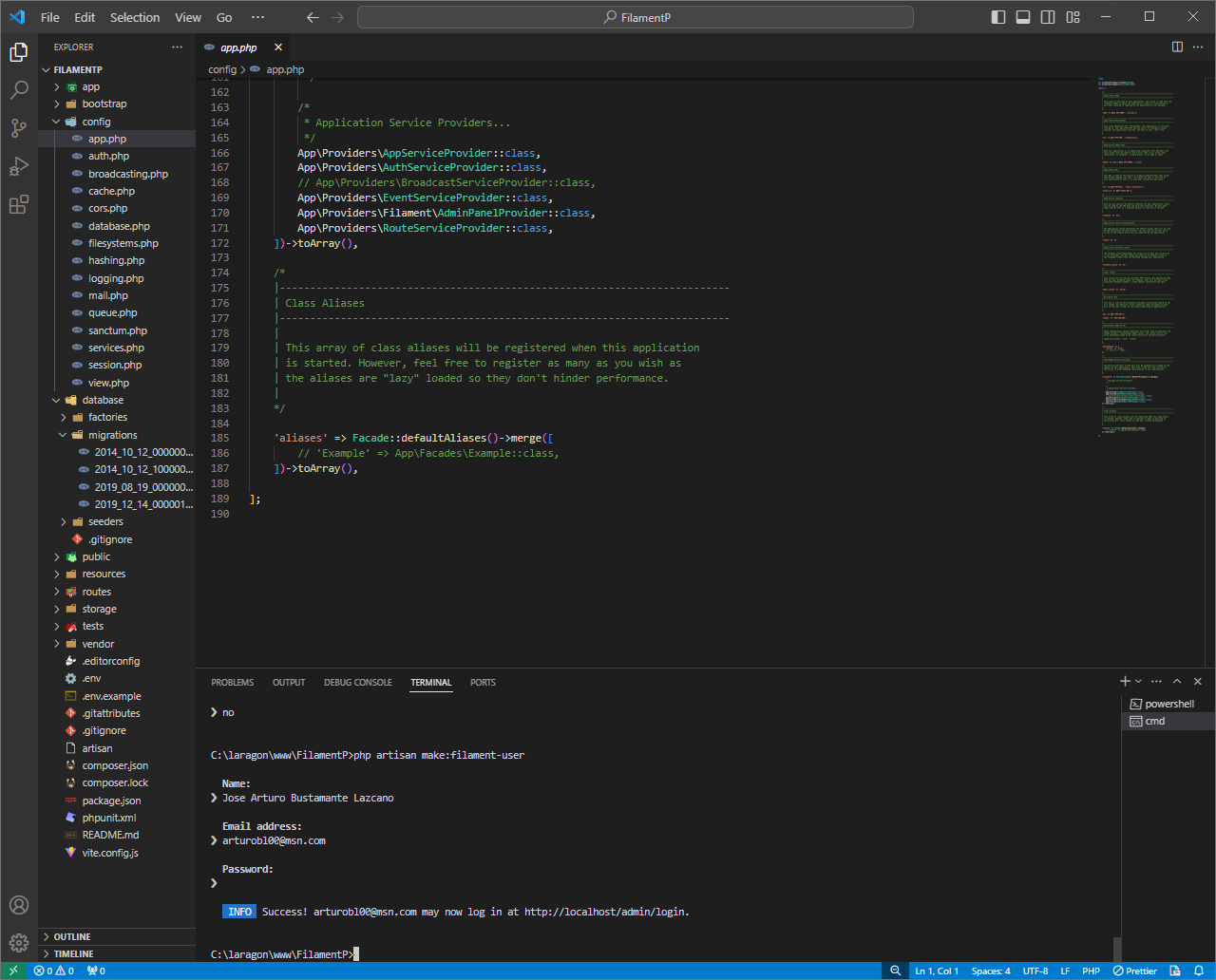




## Ejecutamos

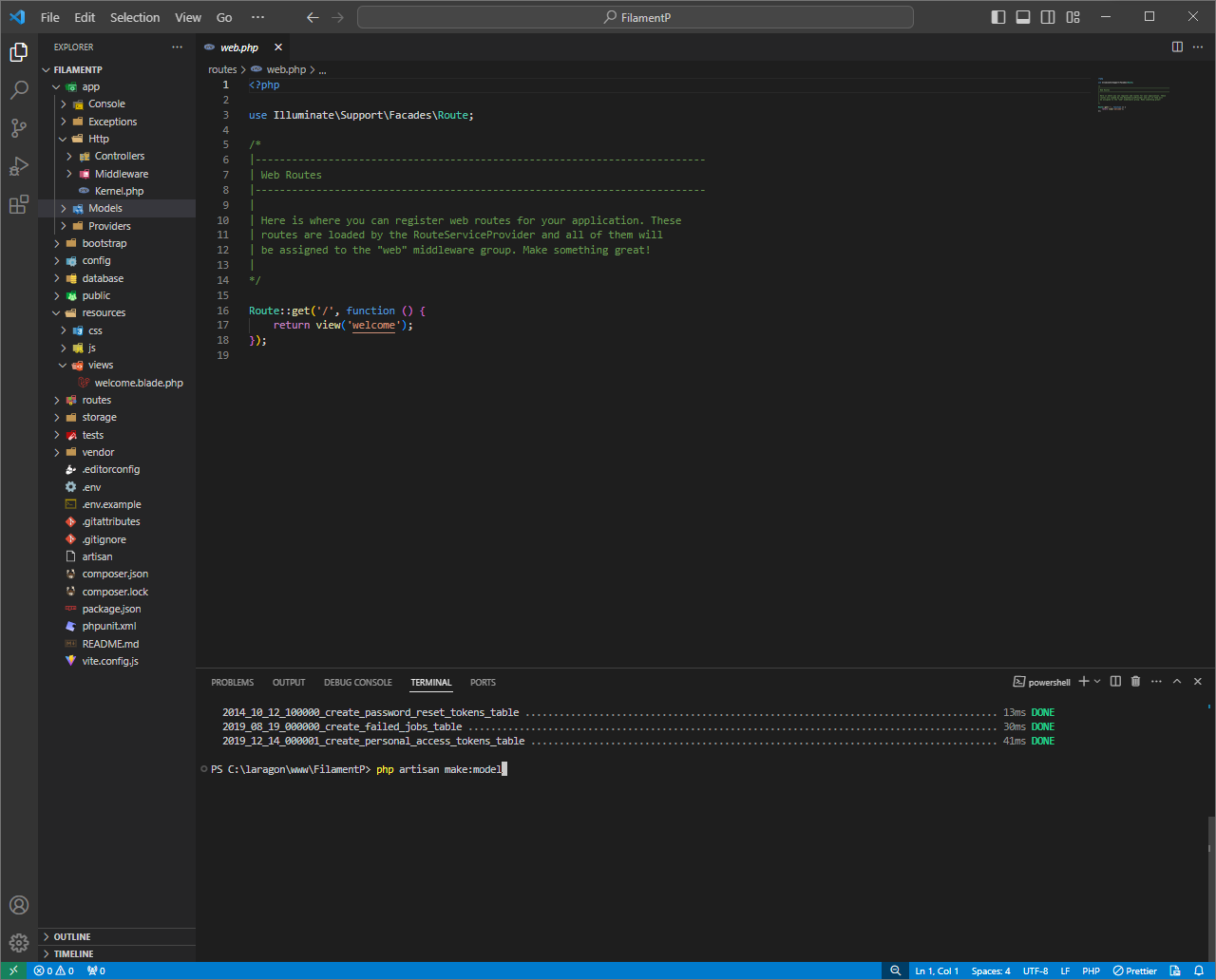


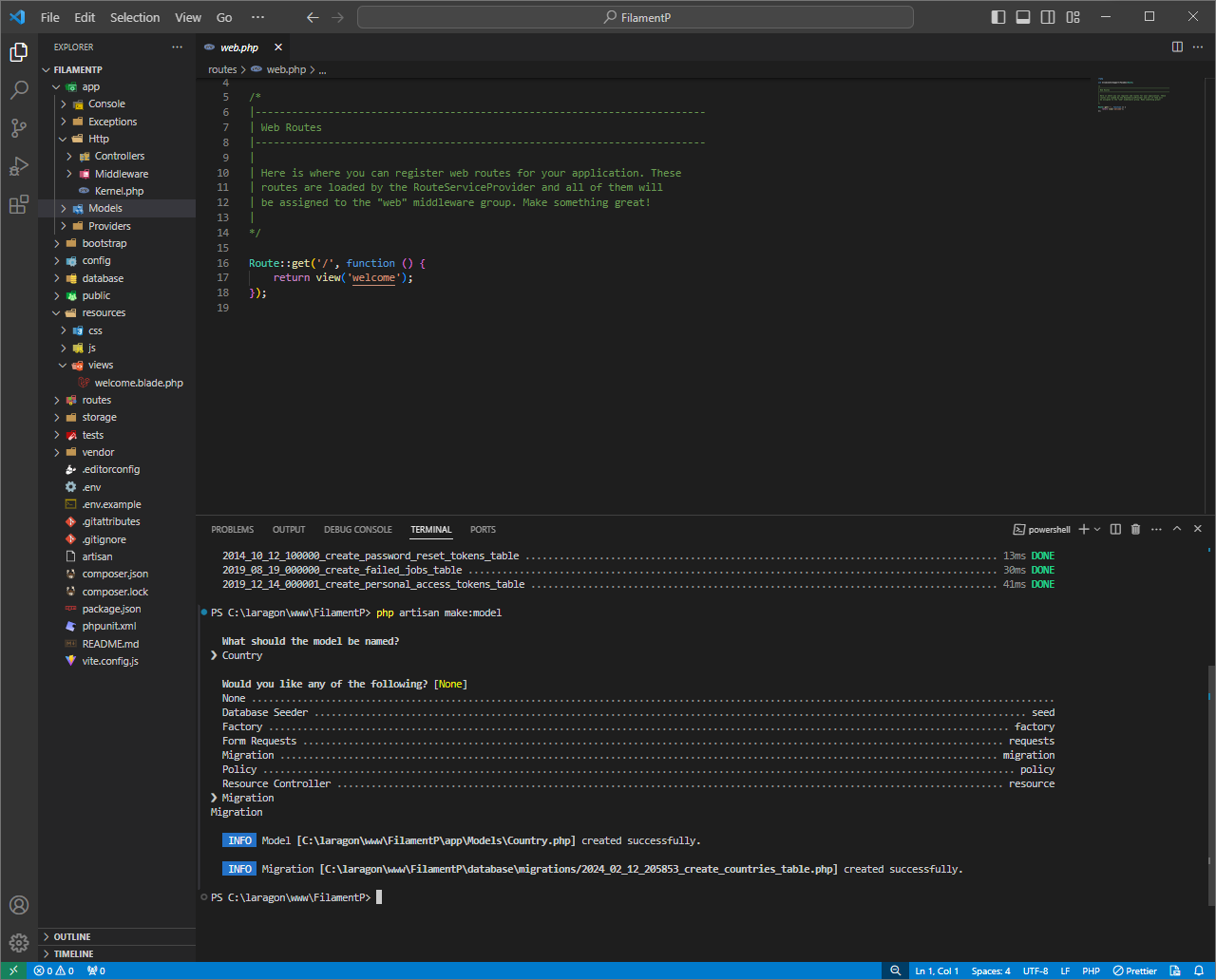


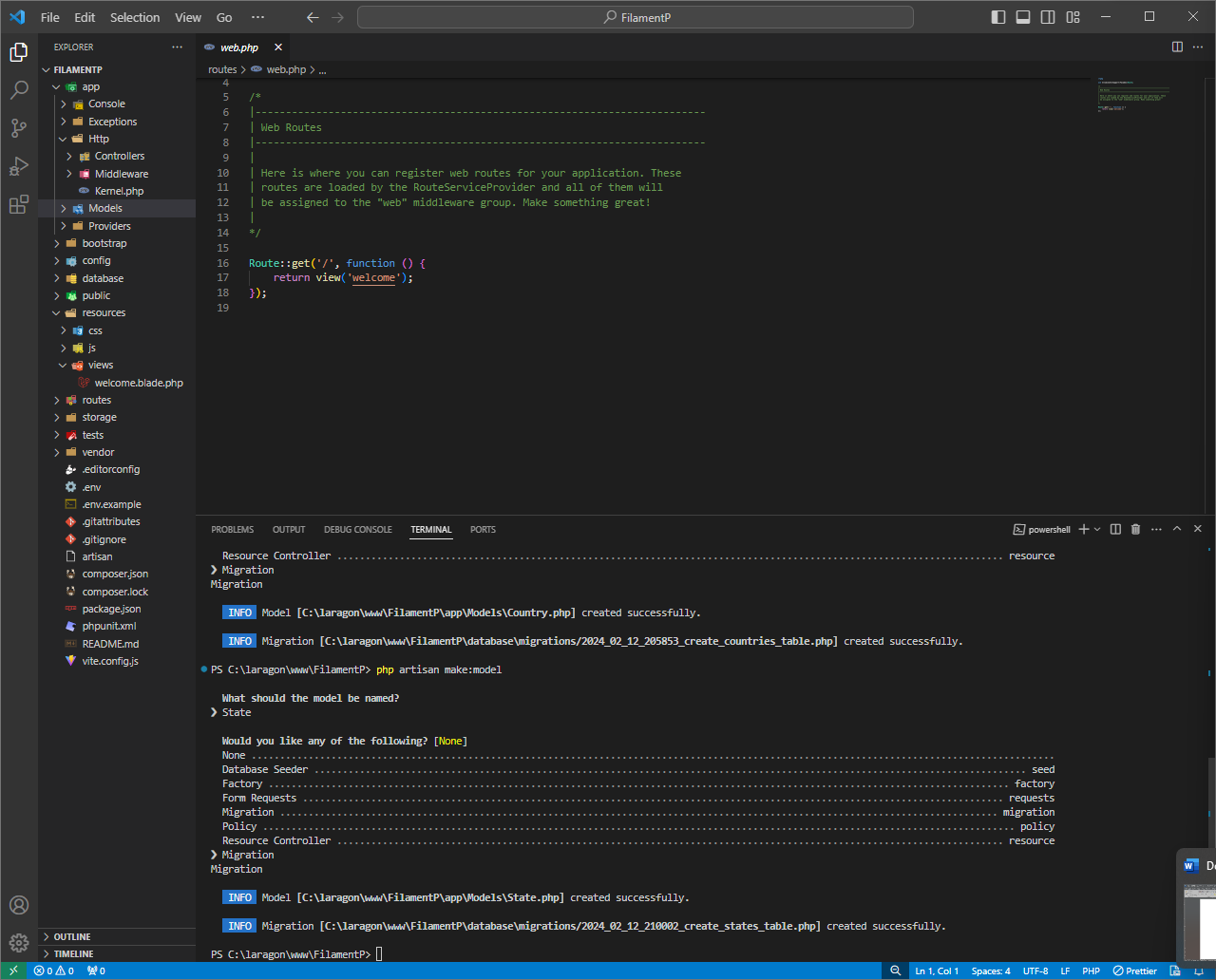


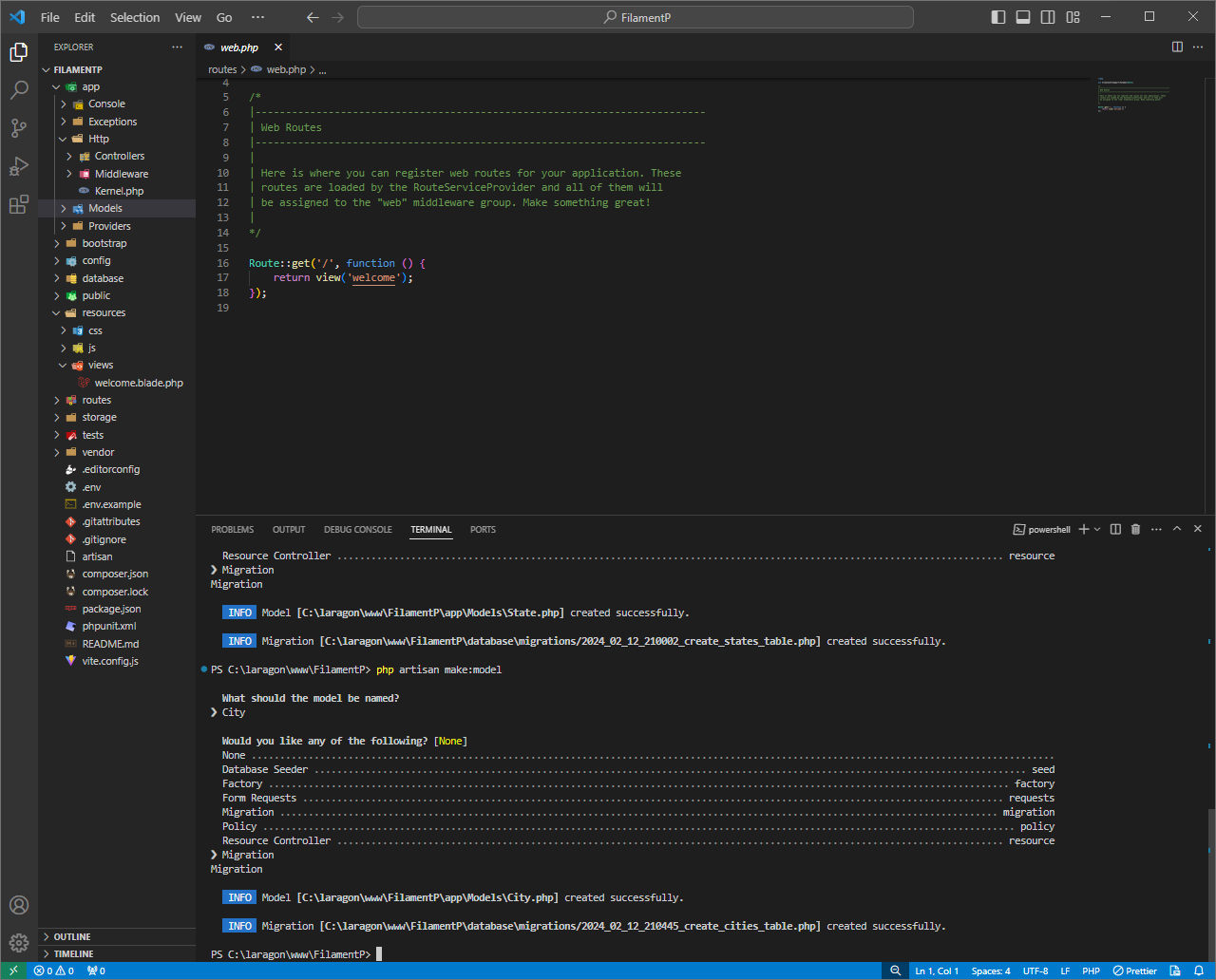
## Crear modelo

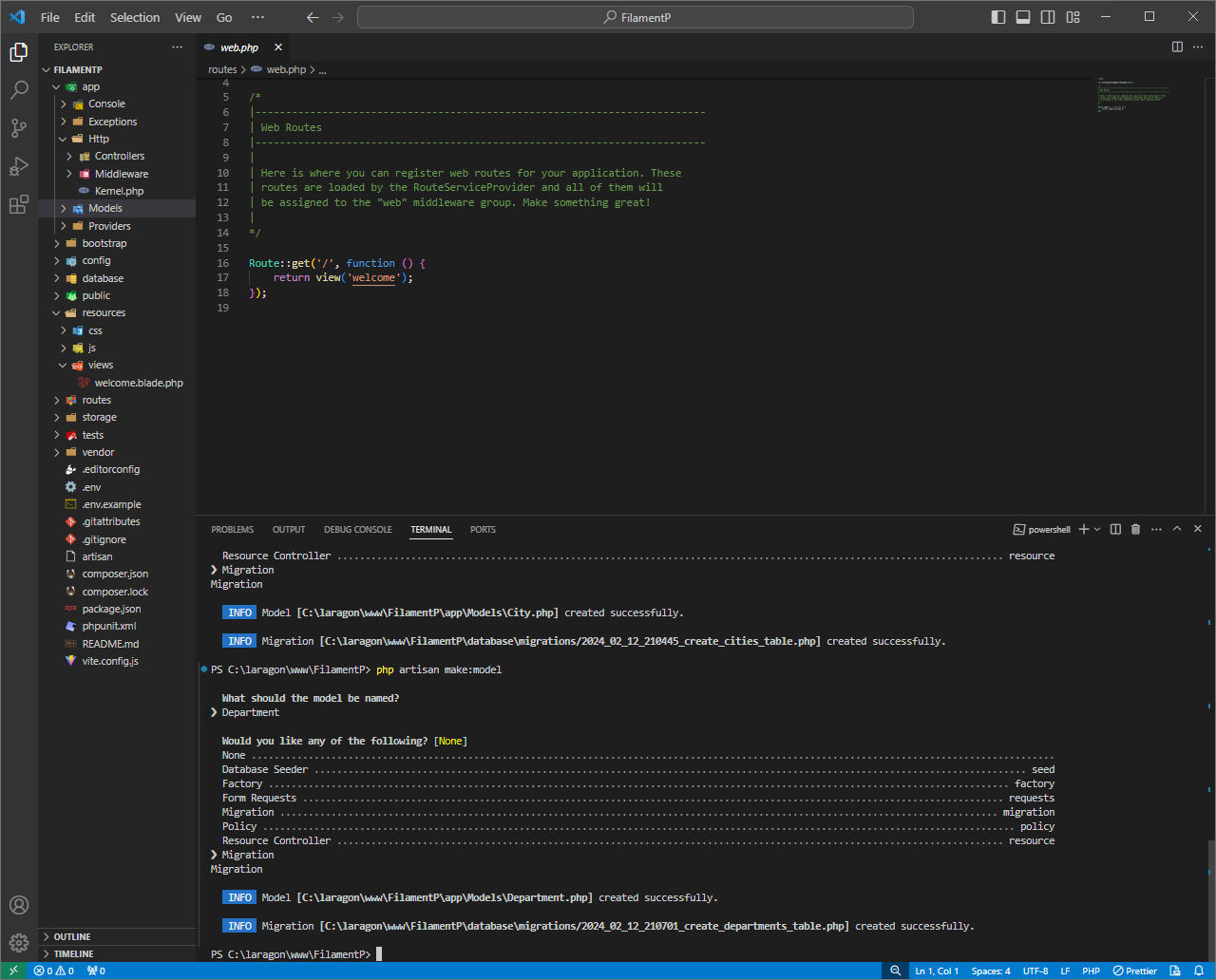
Los Modelos serán Country, State, City, Department, Employee.

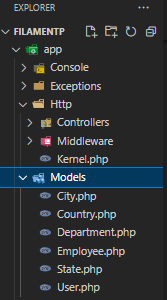


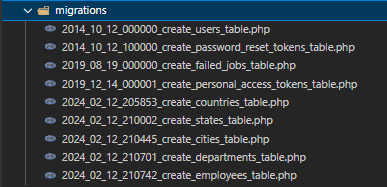




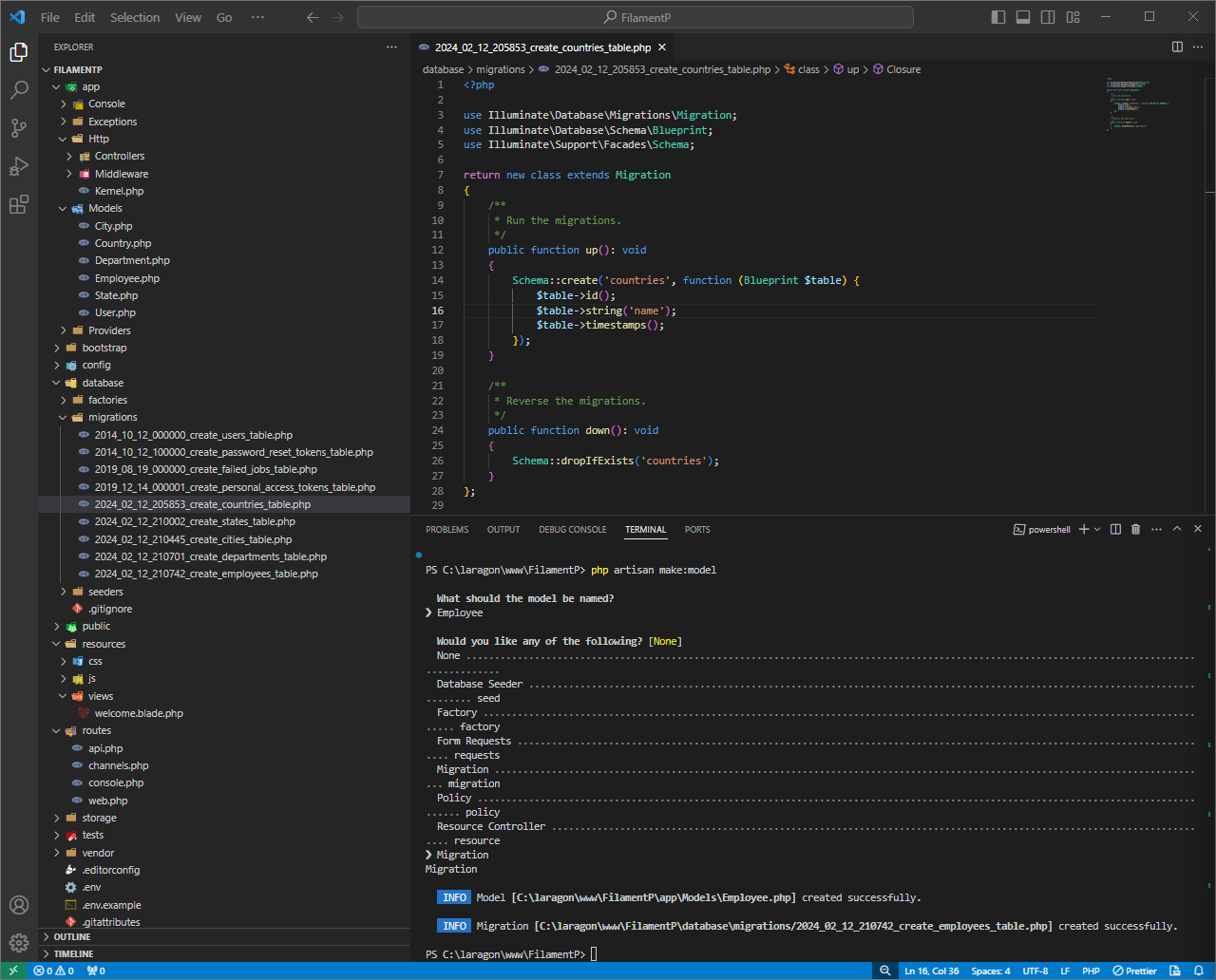


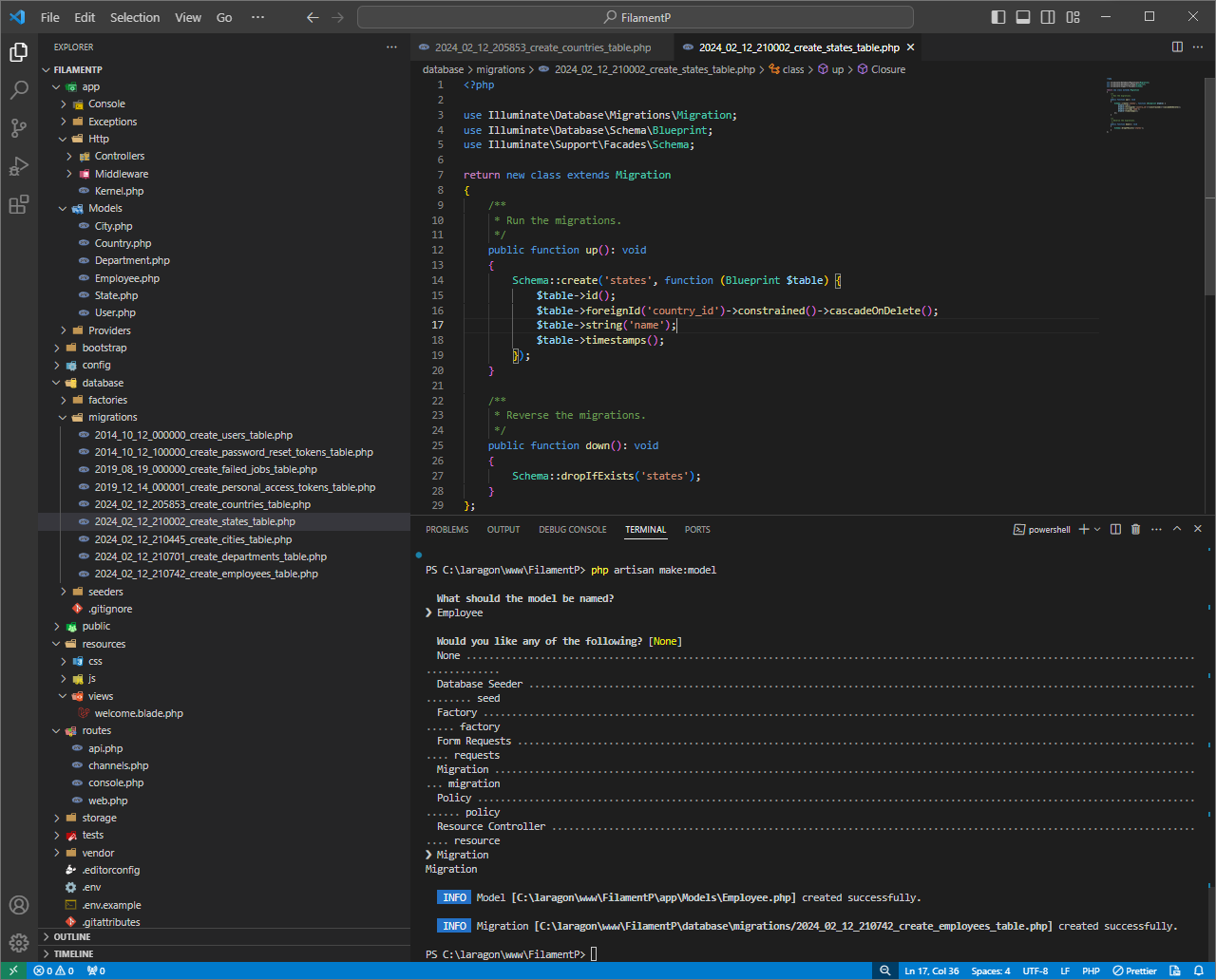


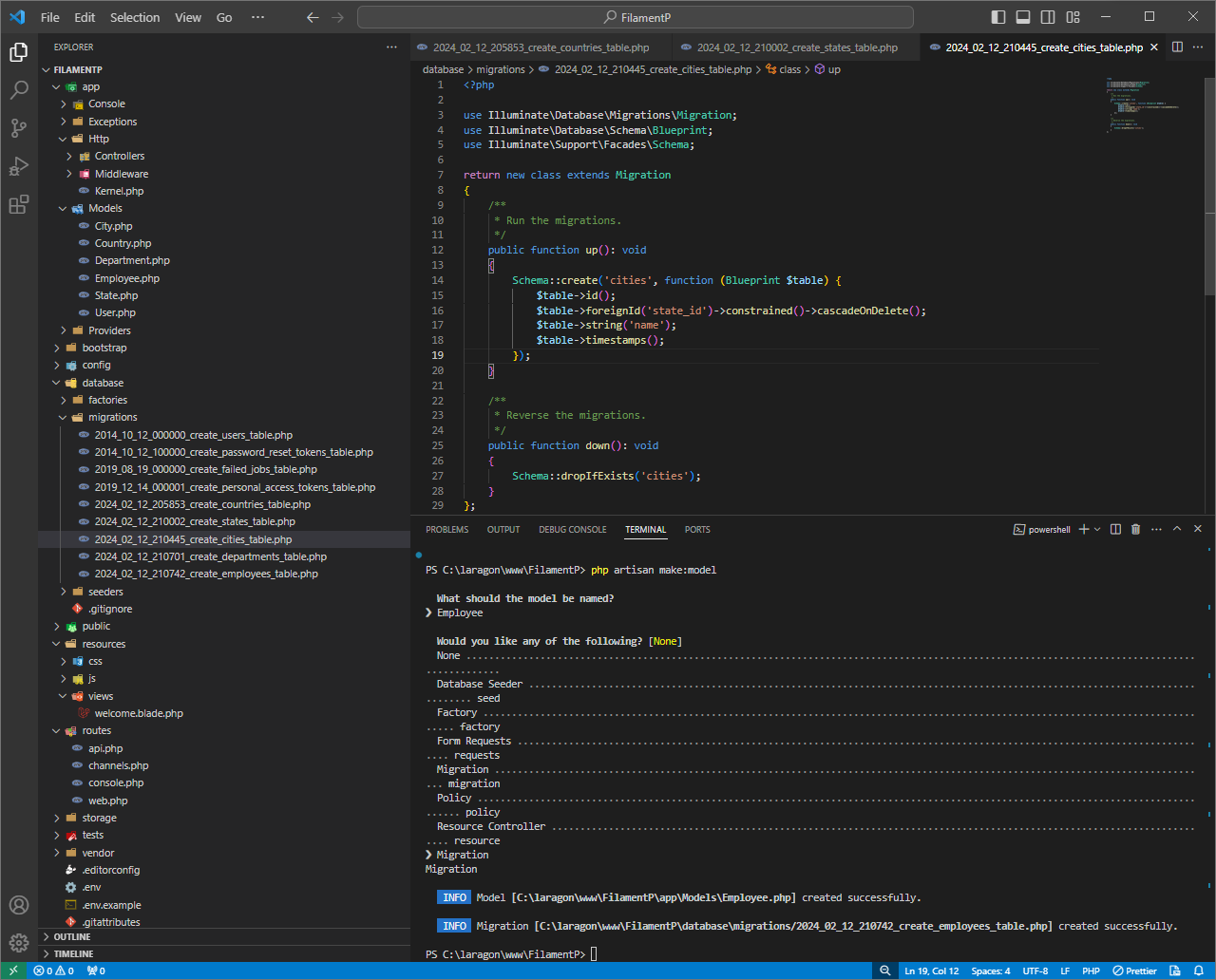


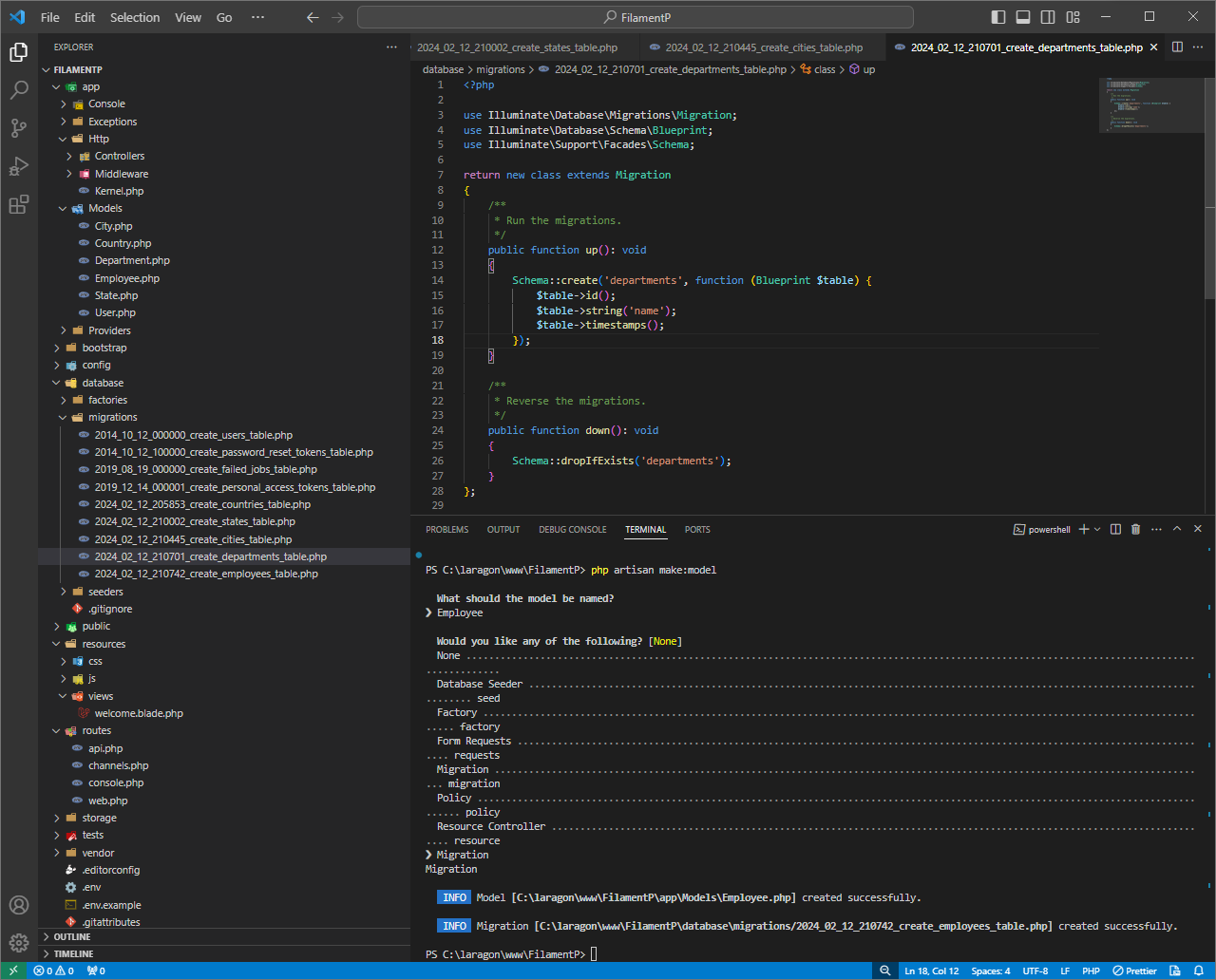


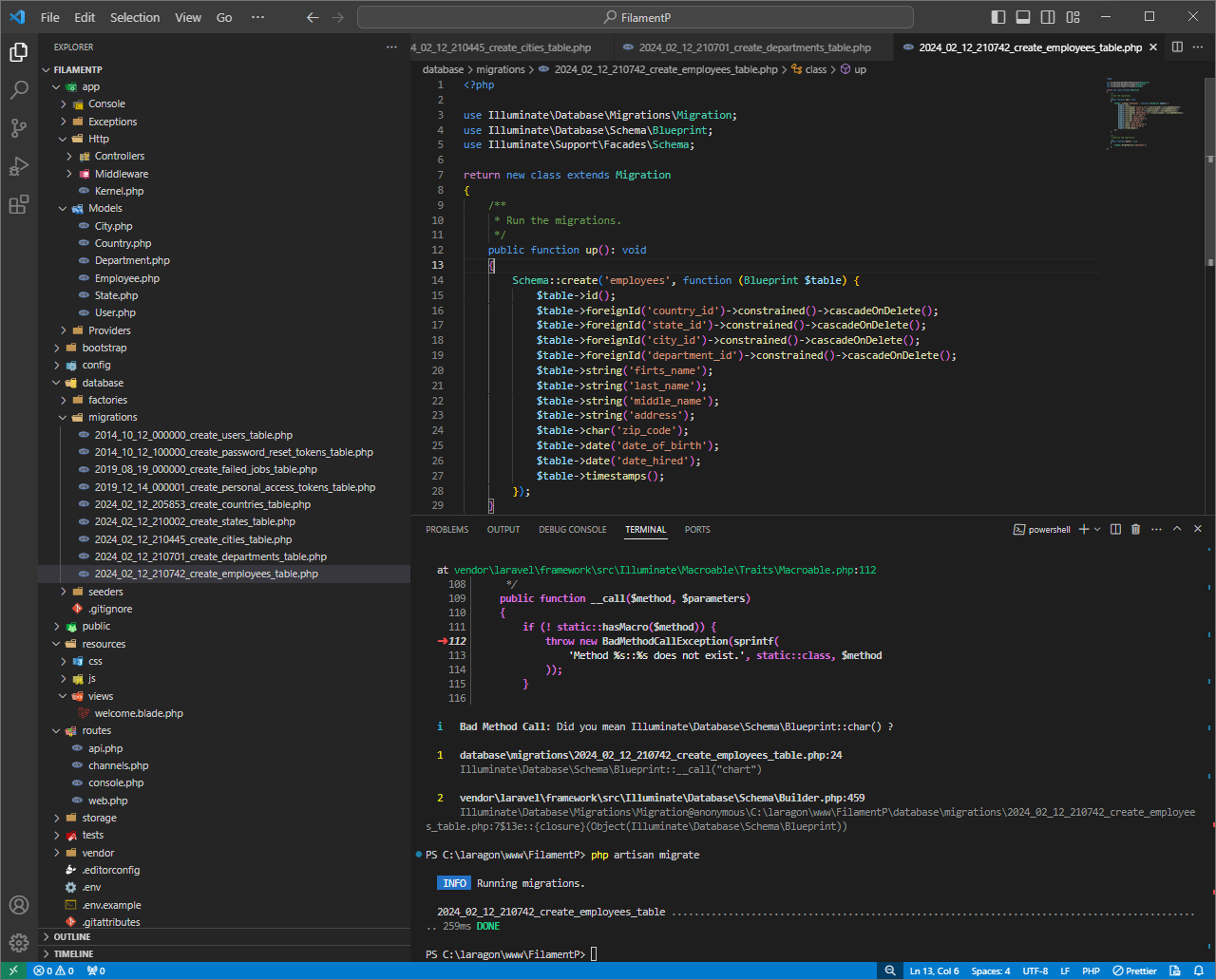
## Migraciones





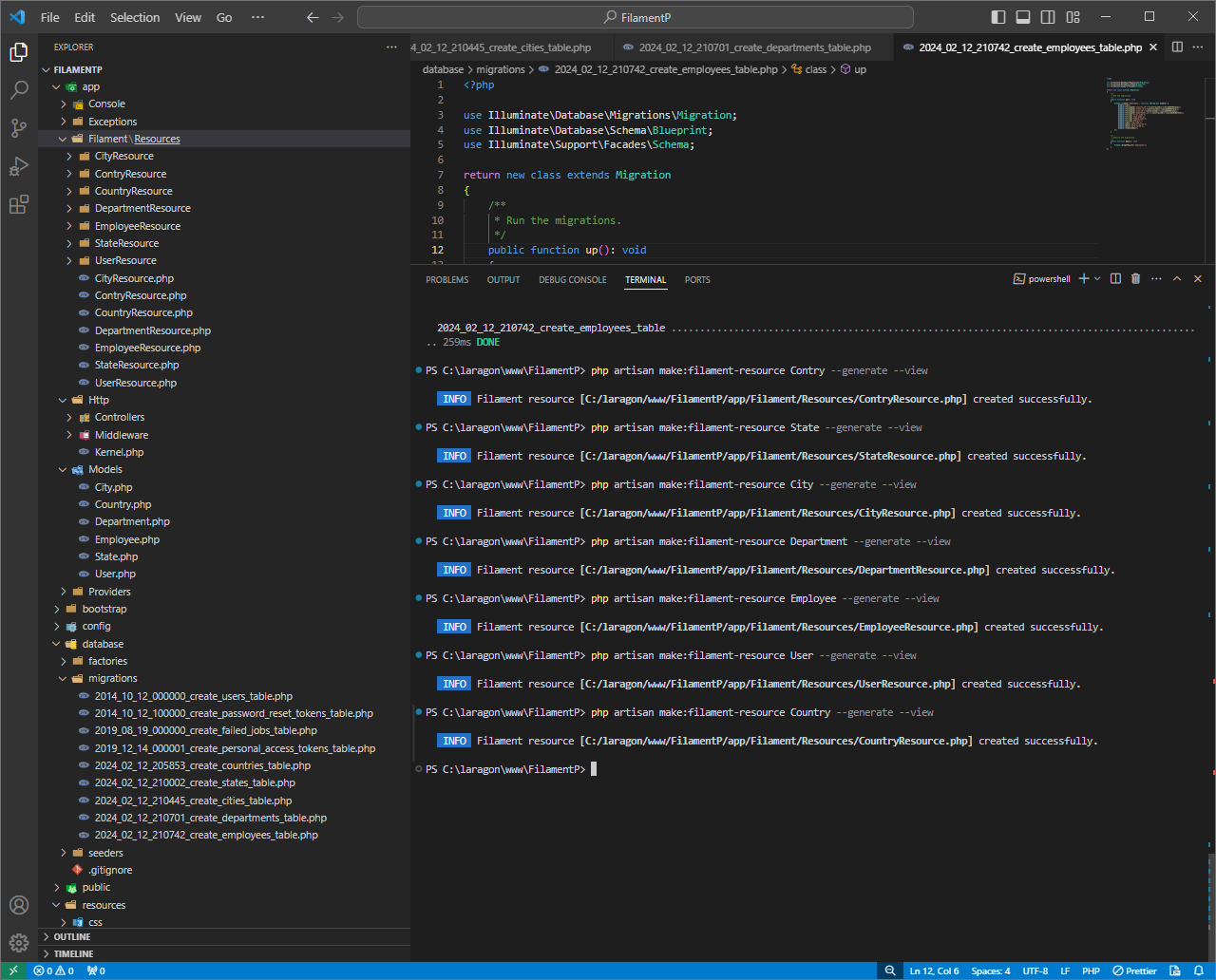


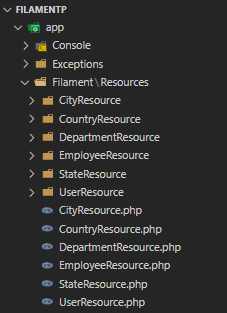


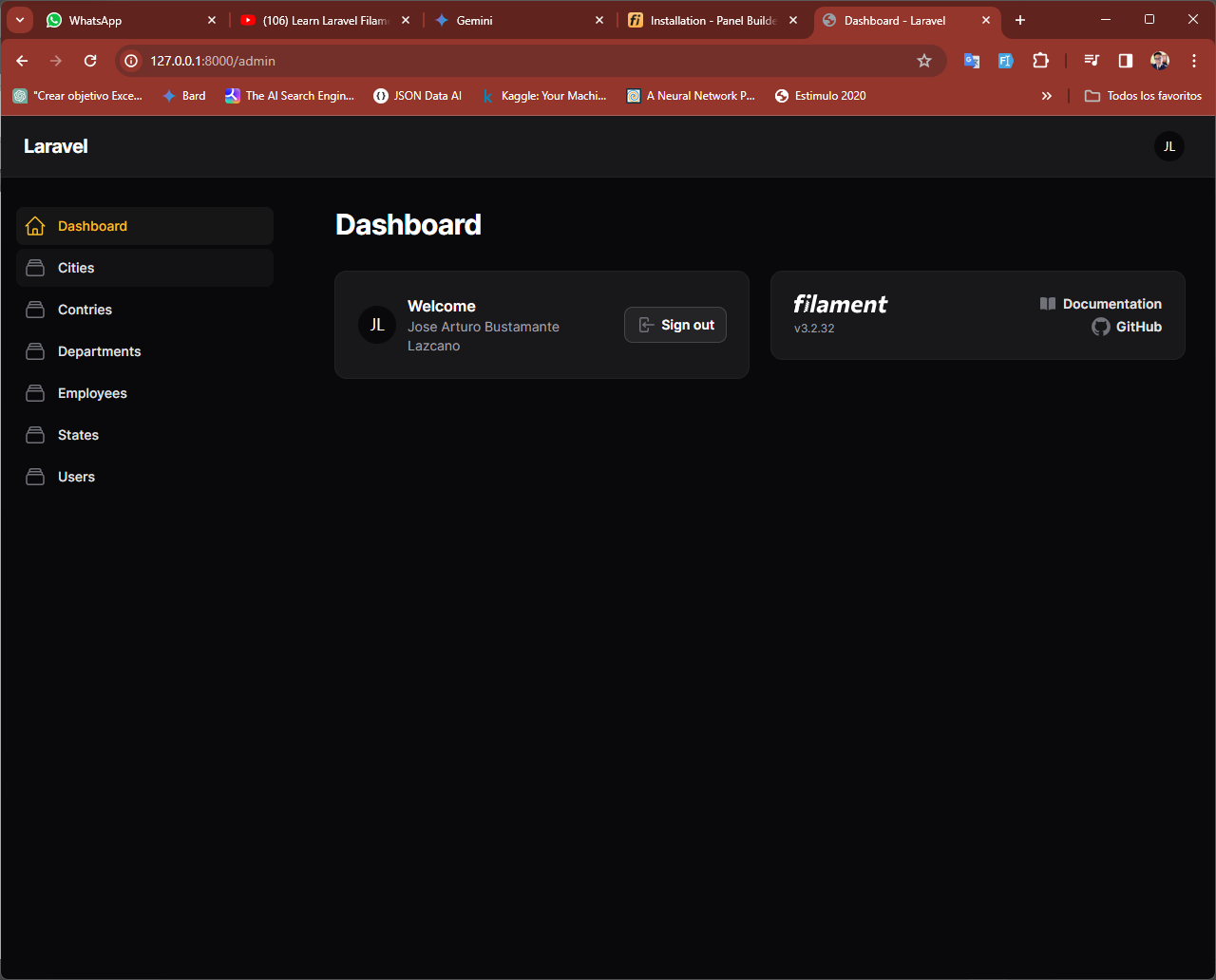


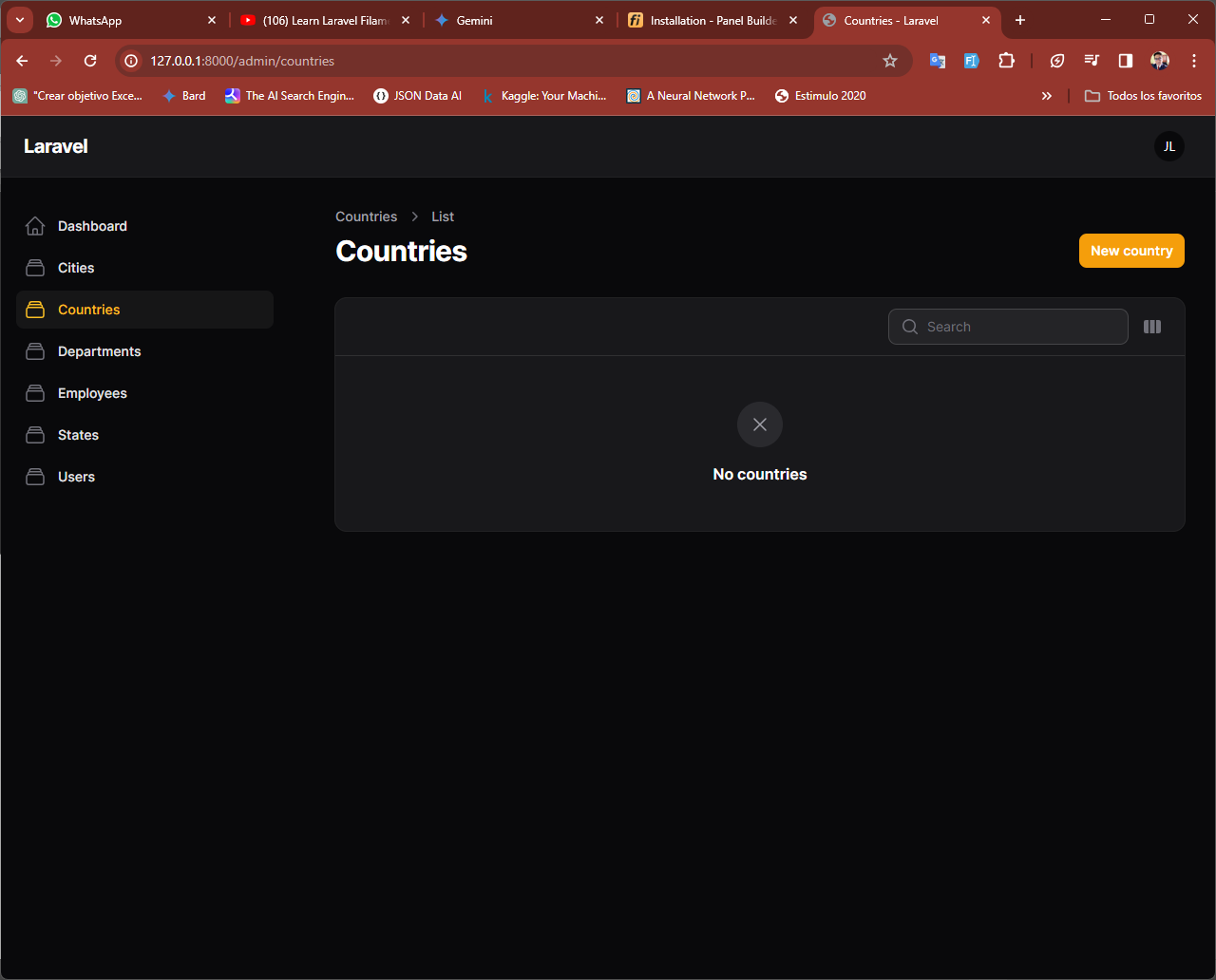


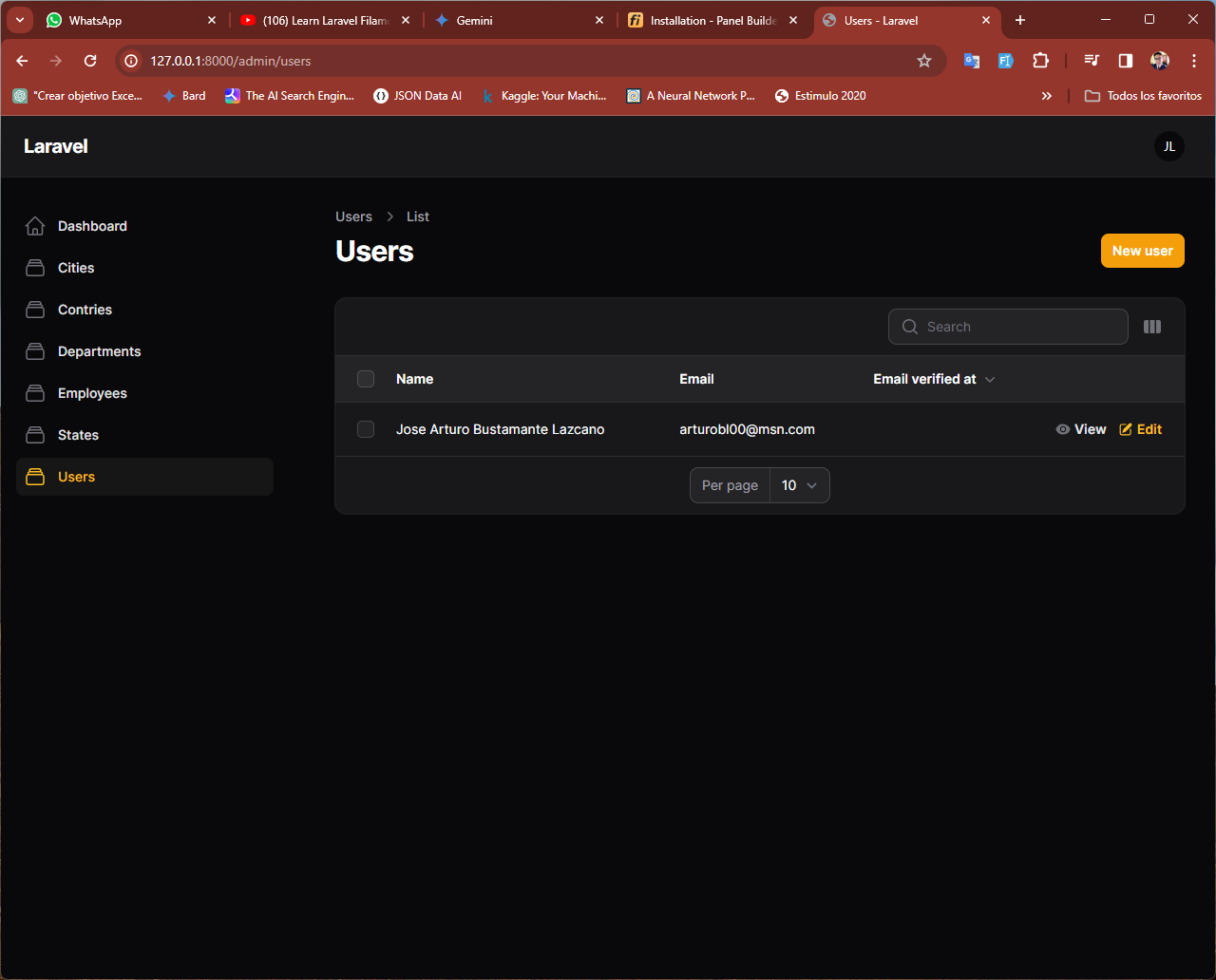
## Generar los Resources





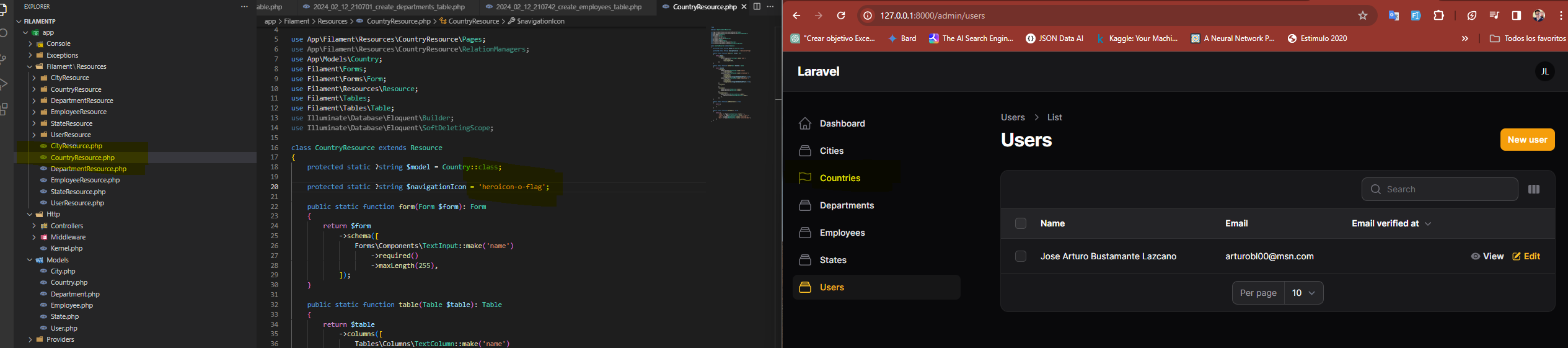






## Editar el Resource

Cambio de icono para Countries.

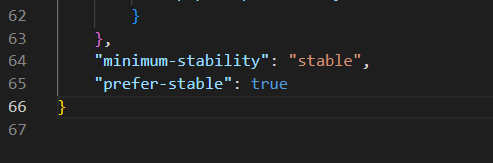


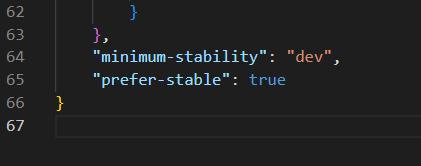
Laravel con Filament

# Paso 1 Crear Proyecto

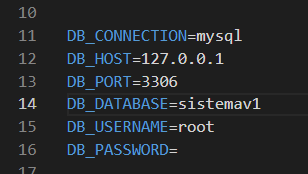
Composer create-project laravel/laravel sistemav1

# Paso 2 Abrir el composer.json cambiar la línea 64 a dev





# Paso 3 Cambiar el nombre de la bd en .env



# Paso 4 Instalar Filament

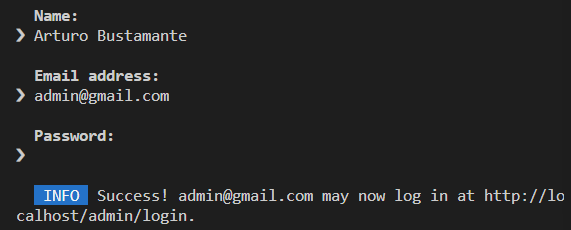
composer require filament/filament:"^3.2" -W

 php artisan filament:install --panels

# Paso 5 importar la migración para crear la bd y Crear un usuario para entrar al admin

Php artisan migrate

php artisan make:filament-user

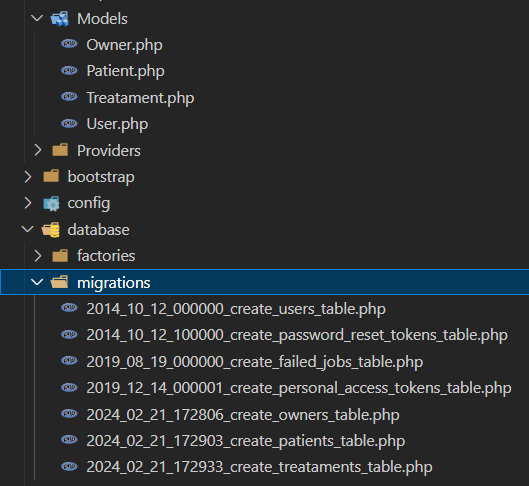


# Paso 6 Crear Modelos: Los modelos nos incorporan la interface, migration y vistas

**PS C:\laragon\www\sistemav1> php artisan make:model Owner -m**

**PS C:\laragon\www\sistemav1> php artisan make:model Patient -m**

**PS C:\laragon\www\sistemav1> php artisan make:model Treatment -m**



# Paso 7 Modificar las Migraciones y agregarles los campos

Mifration Owners

public function up(): void

    {

        Schema::create('owners', function (Blueprint $table) {

            $table->id();

            $table->string('email');

            $table->string('name');

            $table->string('phone');

            $table->timestamps();

        });

    }

Migration Patients

public function up(): void

    {

        Schema::create('patients', function (Blueprint $table) {

            $table->id();

            $table->date('date\_of\_birth');

            $table->string('name');

            $table->foreignId('owner\_id')->constrained('owners')->cascadeOnDelete();

            $table->string('type');

            $table->timestamps();

        });

    }

Migration Treatments

public function up(): void

    {

        Schema::create('treataments', function (Blueprint $table) {

            $table->id();

            $table->string('description');

            $table->text('notes')->nullable();

          $table->foreignId('patient\_id')->constrained('patients')->cascadeOnDelete();

            $table->unsignedInteger('price')->nullable();

            $table->timestamps();

        });

    }

# Paso 8 Poner los modelos globales en unguard

Nos dirimos a app/providers/AppServiceProvider.php

Importamos:

use Illuminate\Database\Eloquent\Model;

Por ultimo agregamos el unguard

public function boot(): void

    {

        //

        Model::unguard();

    }

# Paso 9 Las relaciones

class Owner extends Model

{

    use HasFactory;

    #Los Owner tienen Patients

#Los owners son los dueños los patients son las mascotas

    public function patients(): HasMany

    {

        return $this->hasMany(Patient::class);

    }

}

class Patient extends Model

{

    use HasFactory;

    #Los Patient pertenecen a un Owner

    public function owner(): BelongsTo

    {

        return $this->belongsTo(Owner::class);

    }

    #Los Patient tienen varios Treatments

    public function treatments(): HasMany

    {

        return $this->hasMany(Treatament::class);

    }

}

class Treatament extends Model

{

    use HasFactory;

    #Un Treatment pertece a un Patient

    public function patient(): BelongsTo{

        return $this->belongsTo(Patient::class);

    }

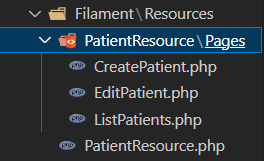
}

# Paso 10 Crear el CRUD con Filament creando un Resource

**PS C:\laragon\www\sistemav1> php artisan make:filament-resource Patient**

Y solo en este paso hay que crear de nuevo la migración para actualizar en la bd las migraciones de nuestros modelos.

**PS C:\laragon\www\sistemav1> php artisan migrate**



Ahora vamos a editar la función de formulario y la función de tabla muy fácil

Formulario de Patients

public static function form(Form $form): Form

    {

        return $form

            ->schema([

                //Vamos a crear un formulario con los campos que defini

                //y un modal para relacionar datos con owner

                Forms\Components\TextInput::make('name')->required()->maxLength(255),

                Forms\Components\Select::make('type')->options([

                    'cat' => 'Cat',

                    'dog' => 'Dog',

                    'hamster' => 'Hamster',

                    'guinea pig' => 'Guinea Pig',

                    'parrot' => 'Parrot',

                    'fish' => 'Fish',

                    'turtle' => 'Turtle',

                    'canary' => 'Canary',

                    'horse' => 'Horse',

                    'ferret' => 'Ferret',

                    'mouse' => 'Mouse',

                    'snake' => 'Snake',

                    'gerbil' => 'Gerbil',

                    'lizard' => 'Lizard',

                    'chinchilla' => 'Chinchilla',

                    'tarantula' => 'Tarantula',

                    'hermit crab' => 'Hermit Crab',

                    'potbellied pig' => 'Potbellied Pig',

                    'iguana' => 'Iguana',

                    'cockatiel' => 'Cockatiel',

                    'goldfish' => 'Goldfish',

                    'hedgehog' => 'Hedgehog',

                    'gecko' => 'Gecko',

                    'budgerigar' => 'Budgerigar',

                    'axolotl' => 'Axolotl',

                    'rabbit' => 'Rabbit',

                    'sugar glider' => 'Sugar Glider',

                    'bearded dragon' => 'Bearded Dragon',

                    'parakeet' => 'Parakeet',

                    'betta fish' => 'Betta Fish',

                    'tortoise' => 'Tortoise',

                    'parrotlet' => 'Parrotlet',

                    'cockatoo' => 'Cockatoo',

                    'rat' => 'Rat',

                ])->required(),

                Forms\Components\DatePicker::make('date\_of\_birth')

                ->required()

                ->maxDate(now()),

#Atención aquí creo la relación y el modal

                Forms\Components\Select::make('owner\_id')

                ->relationship('owner', 'name')

                ->searchable()

                ->preload()

                ->createOptionForm([

                    Forms\Components\TextInput::make('name')

                    ->label('Name Owner')

                    ->required()

                    ->maxLength(255),

                    Forms\Components\TextInput::make('email')

                    ->label('Email address')

                    ->email()

                    ->required()

                    ->maxLength(255),

                    Forms\Components\TextInput::make('phone')

                    ->label('Phone number')

                    ->tel()

                    ->required(),

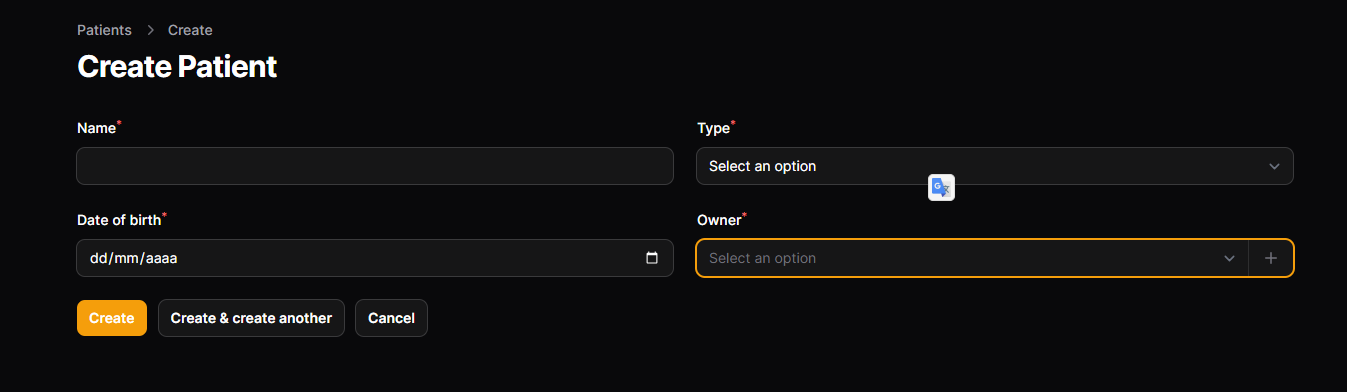
                ])

                ->required(),

            ]);

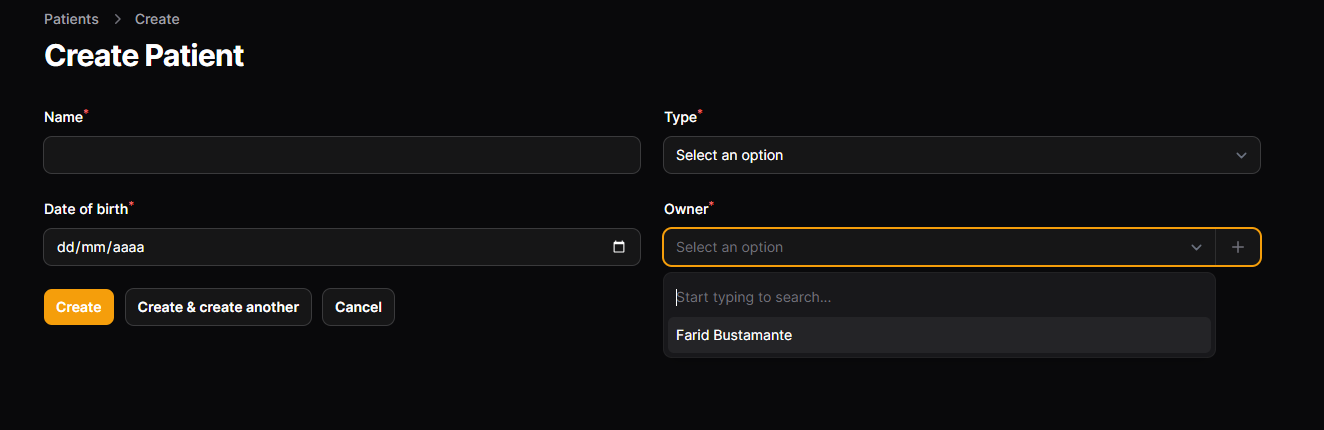
    }

Así queda mi formulario



Modal





Ahora la tabla para mostrar los registros

public static function table(Table $table): Table

    {

        return $table

            ->columns([

                //Diseñaremos la vista de consulta de datos

                Tables\Columns\TextColumn::make('name')

                ->searchable(),

                Tables\Columns\TextColumn::make('type'),

                Tables\Columns\TextColumn::make('date\_of\_birth'),

# Atención aquí tomo los datos que quiero de la relación

                Tables\Columns\TextColumn::make('owner.name')

                ->label('Dueño')

                ->searchable(),

# Atención aquí tomo los datos que quiero de la relación

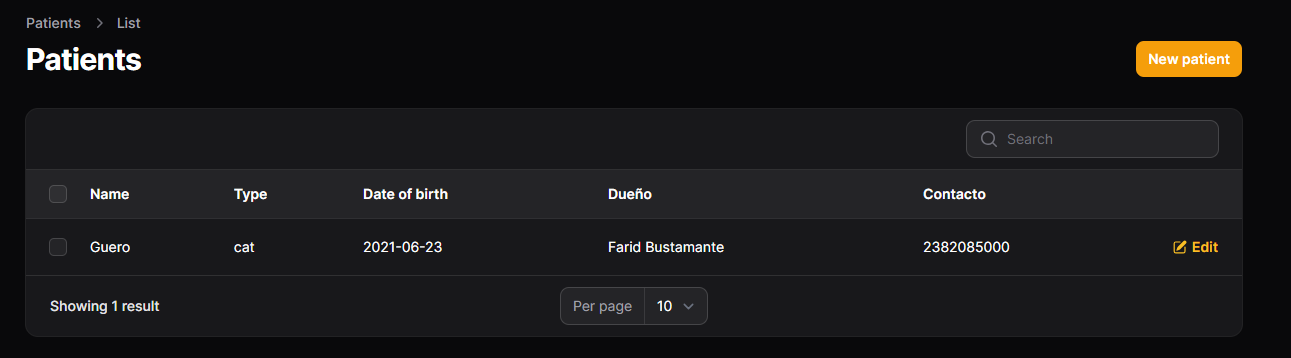
                Tables\Columns\TextColumn::make('owner.phone')

                ->label('Contacto')

                ->searchable(),

            ])

Así queda el formulario



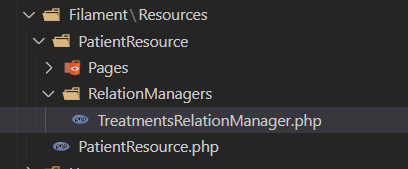
Ahora como le hago para vincula la información de una tabla con otra tabla abajo por ejemplo busco a mi paciente ya que lo encuentro entro al registro y me gustaría ver dentro los tratamientos que tiene asignado o por que no asignar uno nuevo para eso usamos

**PS C:\laragon\www\sistemav1> php artisan make:filament-relation-manager PatientResource treatments description**

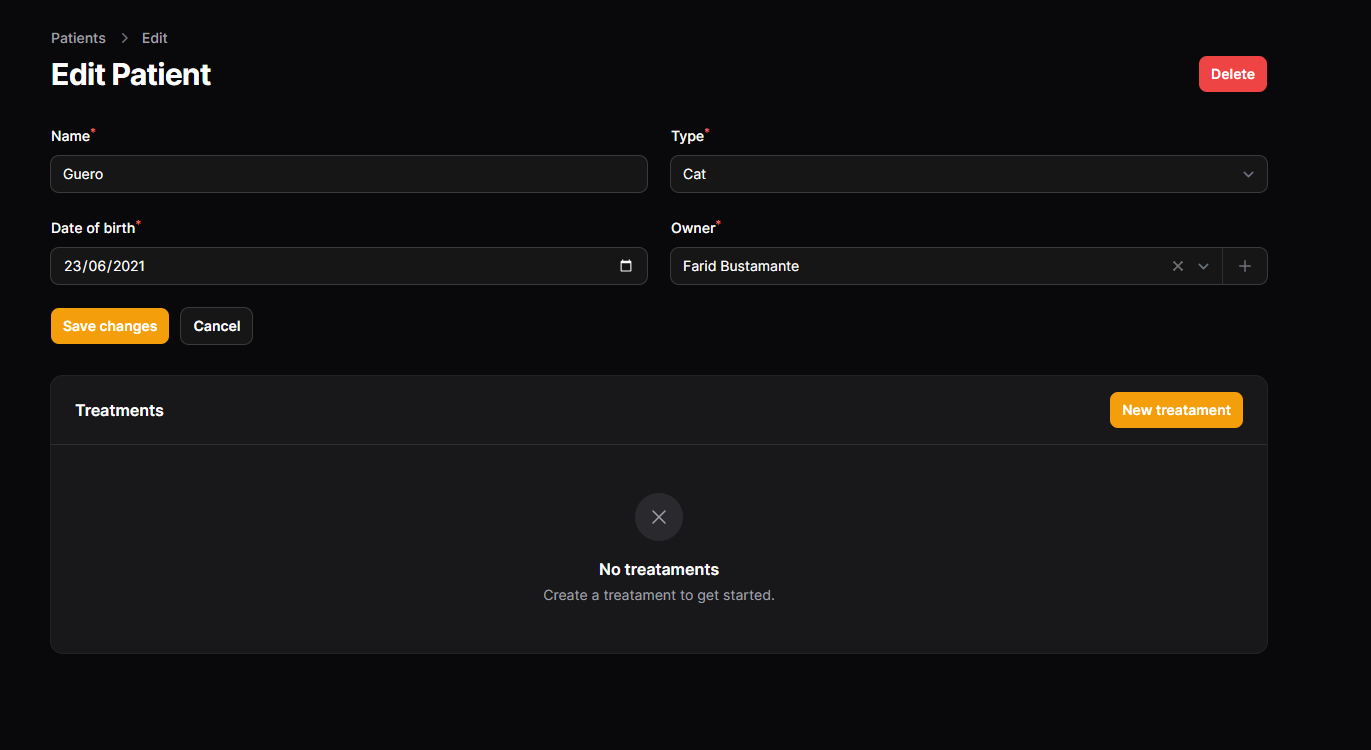
Donde PatientResourcer es la insterface donde lo voy a meter

treatments es la relación que cree en el modelo esto es muy importante

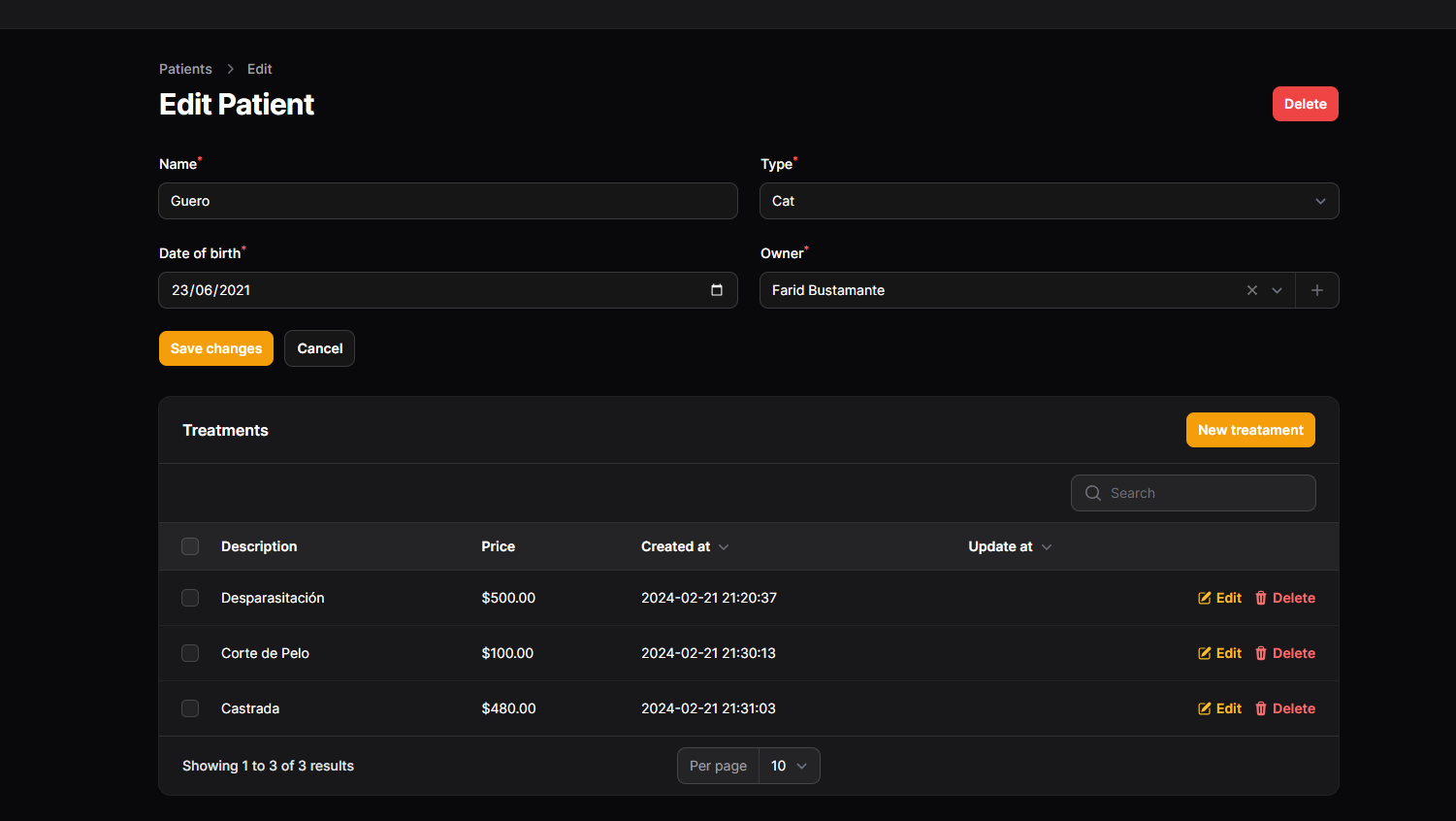
description es un campo que defini en la migración pero al editar puedo meter todos o mas



Ahora se ve así



Editamos el formulario para meter mas campos



public function form(Form $form): Form

    {

        return $form

            ->schema([

                Forms\Components\TextInput::make('description')

                    ->required()

                    ->maxLength(255)

                    ->columnSpan('full'),

                Forms\Components\Textarea::make('notes')

                    ->required()

                    ->maxLength(65535)

                    ->columnSpan('full'),

                Forms\Components\TextInput::make('price')

                    ->numeric()

                    ->prefix('$'),

            ]);

    }

    public function table(Table $table): Table

    {

        return $table

            ->recordTitleAttribute('description')

            ->columns([

                Tables\Columns\TextColumn::make('description')

                ->searchable(),

                Tables\Columns\TextColumn::make('price')

                ->money('usd'),

                Tables\Columns\TextColumn::make('created\_at')

                ->sortable(),

                Tables\Columns\TextColumn::make('update\_at')

                ->sortable(),

            ])

            ->filters([

                //

            ])

            ->headerActions([

                Tables\Actions\CreateAction::make(),

            ])

            ->actions([

                Tables\Actions\EditAction::make(),

                Tables\Actions\DeleteAction::make(),

            ])

            ->bulkActions([

                Tables\Actions\BulkActionGroup::make([

                    Tables\Actions\DeleteBulkAction::make(),

                ]),

            ]);

    }

Para usar el money instale el siguiente comando

**PS C:\laragon\www\sistemav1> php artisan make:cast MoneyCast**

<?php

namespace App\Casts;

use Illuminate\Contracts\Database\Eloquent\CastsAttributes;

use Illuminate\Database\Eloquent\Model;

class MoneyCast implements CastsAttributes

{

    /\*\*

     \* Cast the given value.

     \*

     \* @param  array<string, mixed>  $attributes

     \*/

    public function get(Model $model, string $key, mixed $value, array $attributes): mixed

    {

        return round(floatval($value) / 100, precision: 2);

    }

    /\*\*

     \* Prepare the given value for storage.

     \*

     \* @param  array<string, mixed>  $attributes

     \*/

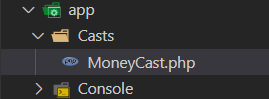
    public function set(Model $model, string $key, mixed $value, array $attributes): mixed

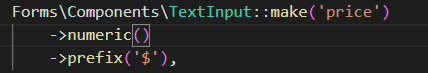
    {

        return round(floatval($value) \* 100);

    }

}

****

****