Laravel

Juliana Nicole Abril Cabrera
Camila Verónica Granda Salamea
Emily Pamela Romero Araujo

Viernes, 8 de Julio

1 Abstract

Este boletín explora Laravel presenta información para crear un proyecto y un ejemplo de CRUD utilizando POSTMAN para operaciones básicas sin necesidad de frontend.

2 Introducción

Laravel es creado por Taylor Otwell en 2011, con el objetivo de proporcionar una alternativa más avanzada y flexible a los marcos de desarrollo PHP existentes. Rápidamente se distinguió por su elegancia, sencillez y, sobre todo, por su enfoque en el desarrollador.
Laravel es un framework moderno para desarrollar aplicaciones web y se destaca por su sintaxis clara. Proporciona una estructura organizada que simplifica la creación de aplicaciones.

3 Proyecto

Una vez completada la instalación de Laravel, sigue estos pasos para crear un nuevo proyecto:

1. **Abrir la Terminal (cmd):**
   Abre el símbolo del sistema (cmd) y navega hasta la carpeta donde está corriendo XAMPP. Puede ser la "Carpeta Pública" definida al inicio del curso o la carpeta htdocs.

2. **Crear un Proyecto Nuevo:**
   Ejecuta el siguiente comando para crear un nuevo proyecto Laravel:

```
laravel new CRUD
```

Figure 1: Creación del nuevo proyecto en consola.

Se nos va a preguntar por unas configuraciones del proyecto. Como se puede ver entre los "[ ]", hay una opción predeterminada. Vamos a elegir esa opción en cada configuración, por lo que a todas las preguntas daremos simplemente ***'enter'***.

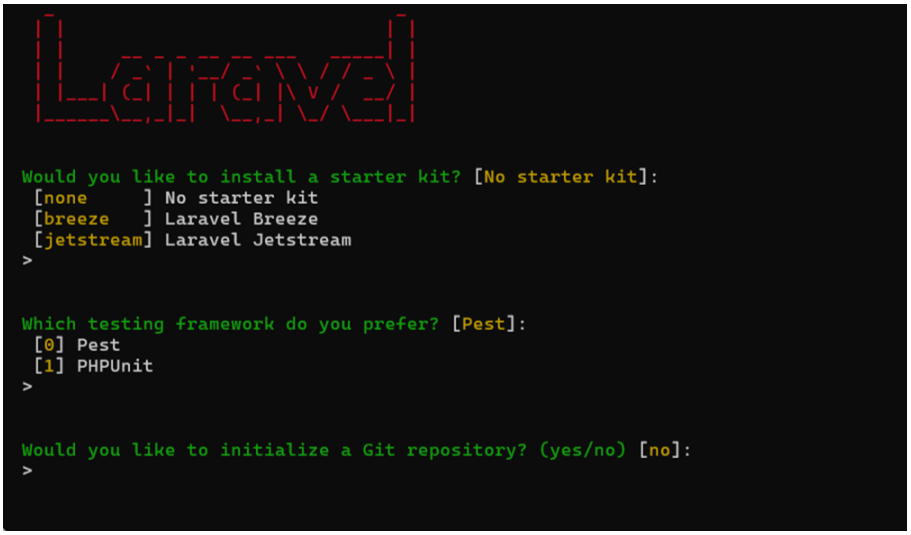

Figure 2: Proceso de creación del proyecto.

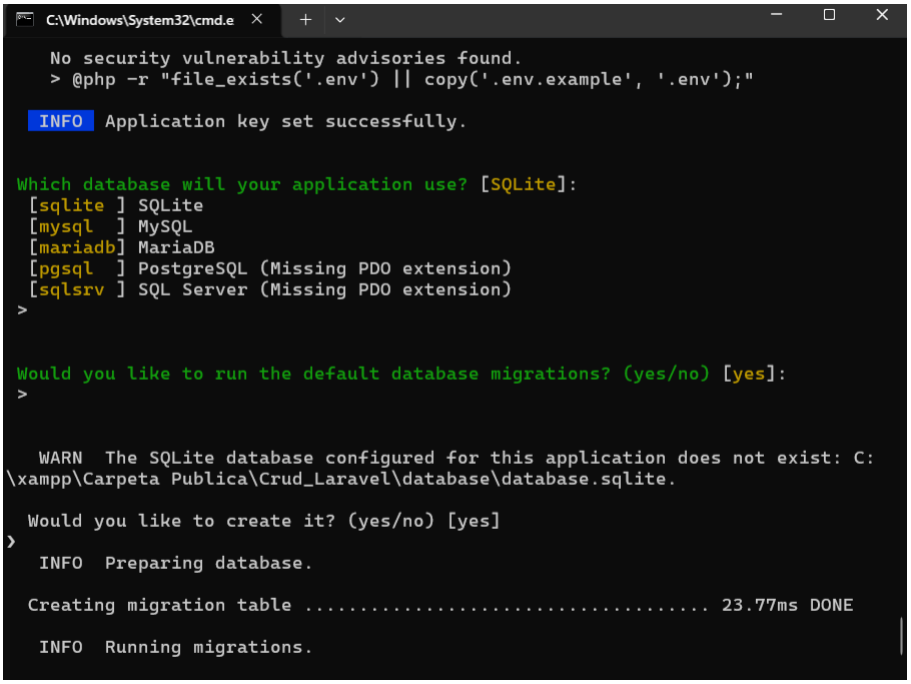

Figure 3: Proceso de creación del proyecto.

Este comando generará una nueva carpeta llamada CRUD con toda la estructura y archivos necesarios para comenzar a trabajar en tu aplicación Laravel.

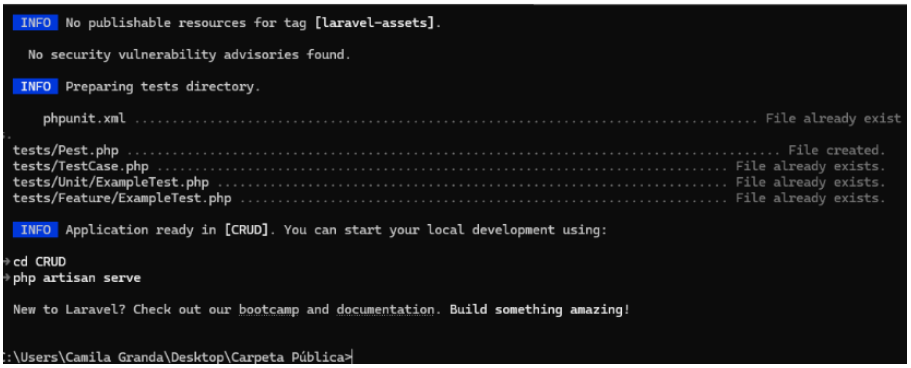

Figure 4: Nuevo Proyecto creado.

3. Después nos dirigimos a VSC, y abrimos la carpeta que fue creada anteriormente.

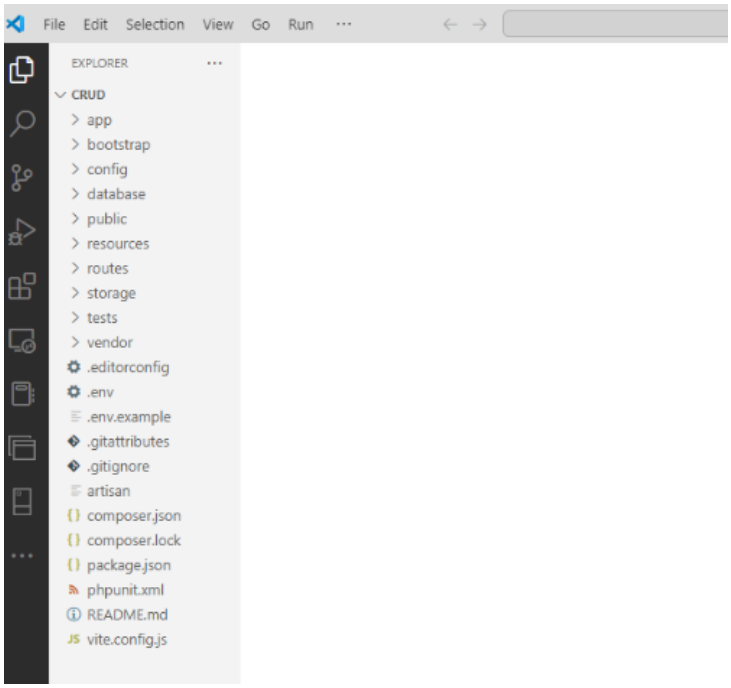

Figure 5: Proyecto en VSC

4. Una vez hecho esto, abrimos la terminal de Visual Studio Code y ejecutamos el comando:

```
php artisan install:api
```

Ejecutamos este comando para empezar nuestro proyecto de Laravel que no tiene interfaz gráfica. Utilizaremos Postman para interactuar y probar el servicio web. Después, se nos preguntará si queremos modificar la base de datos, a lo que responderemos afirmativamente.

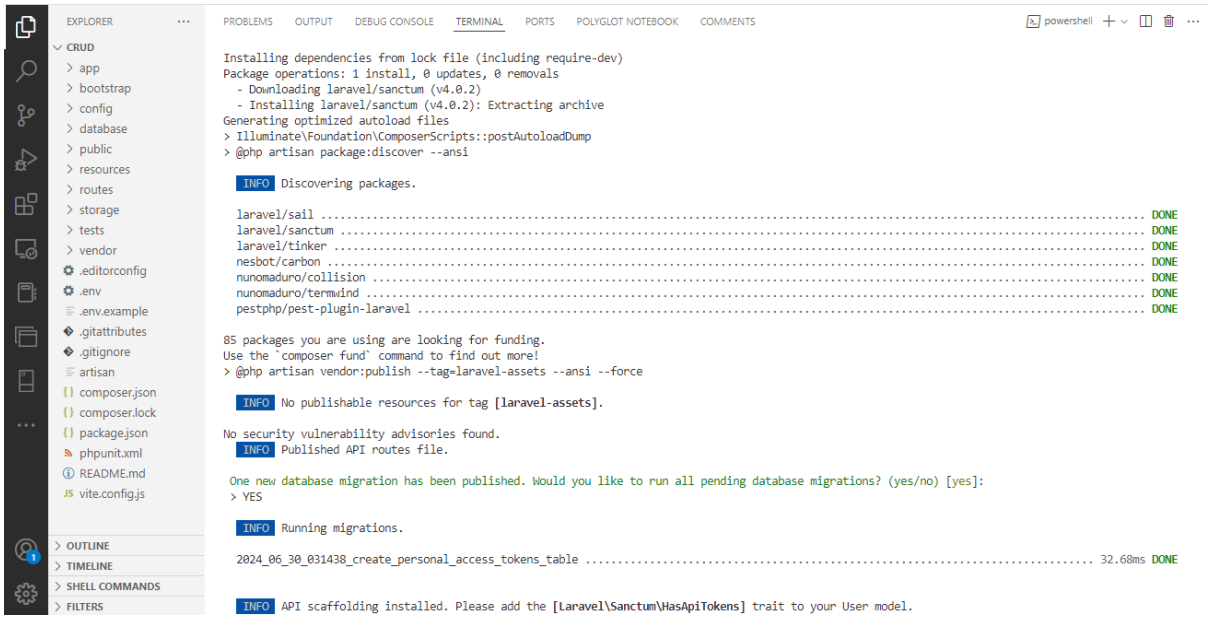

Figure 6: Instalación de API

Esto hará que en nuestro proyecto en la carpeta routes, se creará un nuevo archivo llamado **'api.php'** y en la carpeta bootstrap se encuentra un archivo **'app.php'** el cual se agregará la ruta del archivo recién creado y el cuál debería quedar de la siguiente manera:

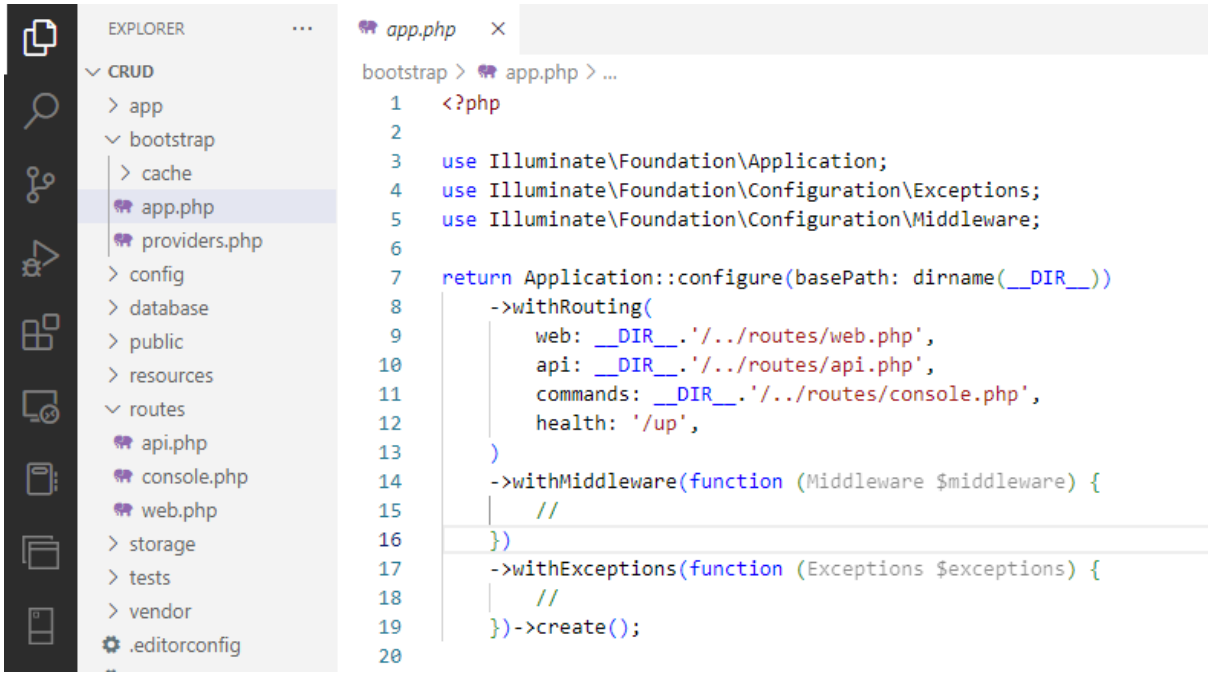

Figure 7: Modificación en el archivo de app.php

5. A continuación, podemos proceder a realizar el CRUD. Para ello, es necesario crear una migración que nos permita consultar, modificar, ingresar y borrar datos de una base de datos. Cabe destacar que, al instalar Composer, se incluye una base de datos SQLite. Para crear la migración, abrimos la terminal de Visual Studio Code y ejecutamos el siguiente comando:

```
php artisan make:migration create_student_table
```

Esto lo que va hacer es que se va a crear un nuevo archivo en **'./database/migrations'**, y esto es lo que ha creado:

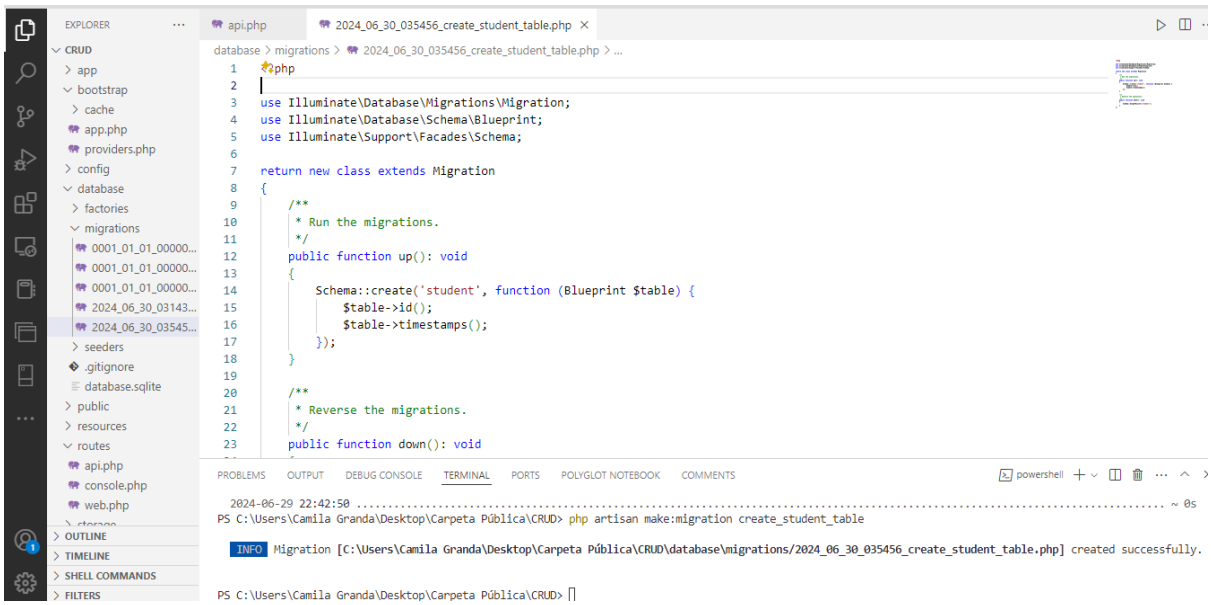

Figure 8: Creación de Migración

Donde se ha creado una tabla **'student'**, con los campos **'id' y 'timestamp'**, que al momento de registrar un nuevo dato estos se generarán de manera automática sin necesidad de que nosotros ingresemos datos a esos campos. A continuación se modificará por lo siguiente:

```
Schema::create('student', function (Blueprint $table) {
    $table->id();
    $table->string('name');
    $table->string('email');
    $table->string('phone');
    $table->string('address');
    $table->timestamps();
});
```

Es recomendable que al momento de copiar código para que este alineado:

(a) Señalamos el código que se quiere alinear.
(b) Hacemos clic derecho.
(c) Y seleccionamos la opción **Format Document**.

A continuación se debe migrar la tabla a SQLite para poder visualizarla, para ello se abre la terminal en VSC y se coloca el siguente comando:

```
php artisan migrate
```

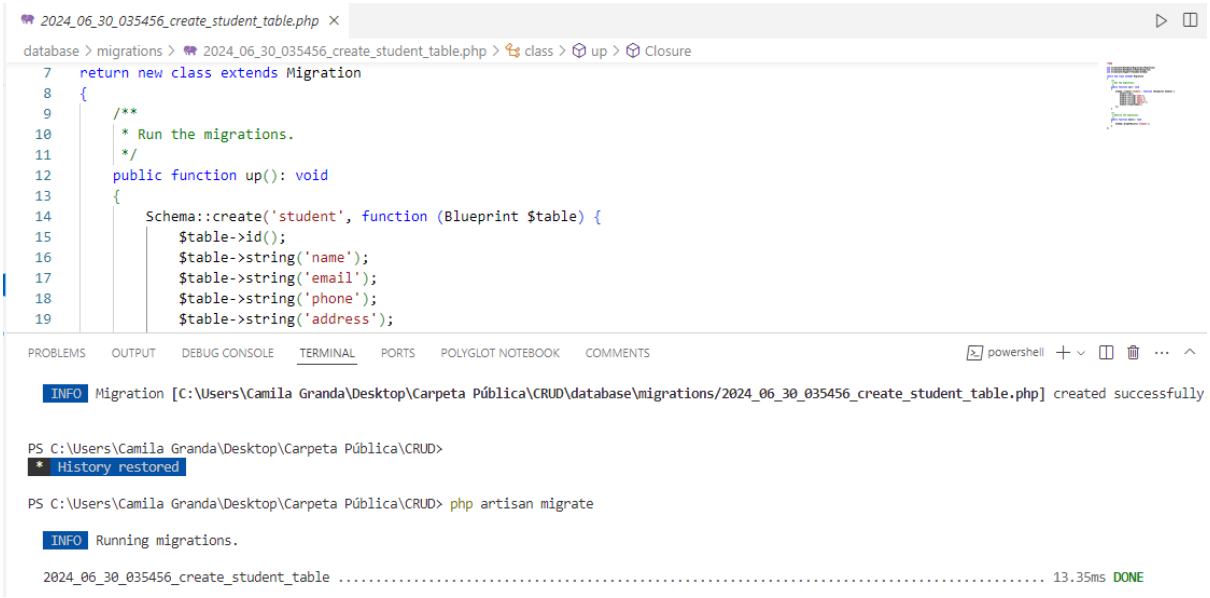

Figure 9: Migración de Tabla

Ahora se puede visualizar la tabla en SQLite de la siguiente manera:

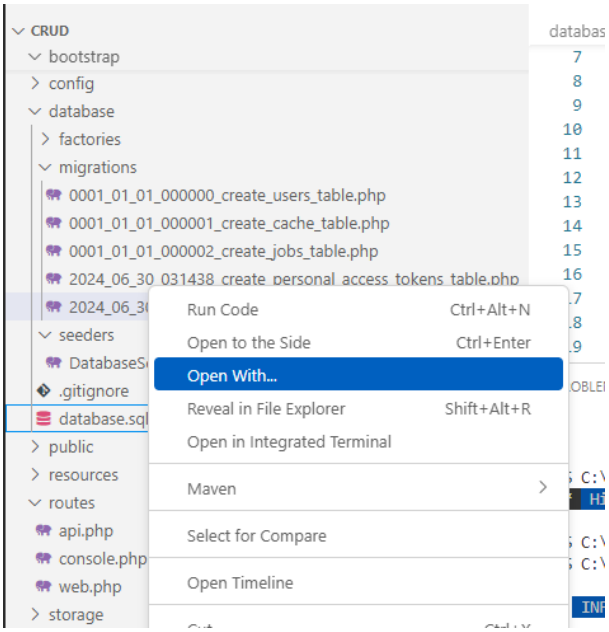

Figure 10: Ingreso a SQLite

Y se coloca la opción de **'SQL Viewer'**, luego de ello tabla **'student'**

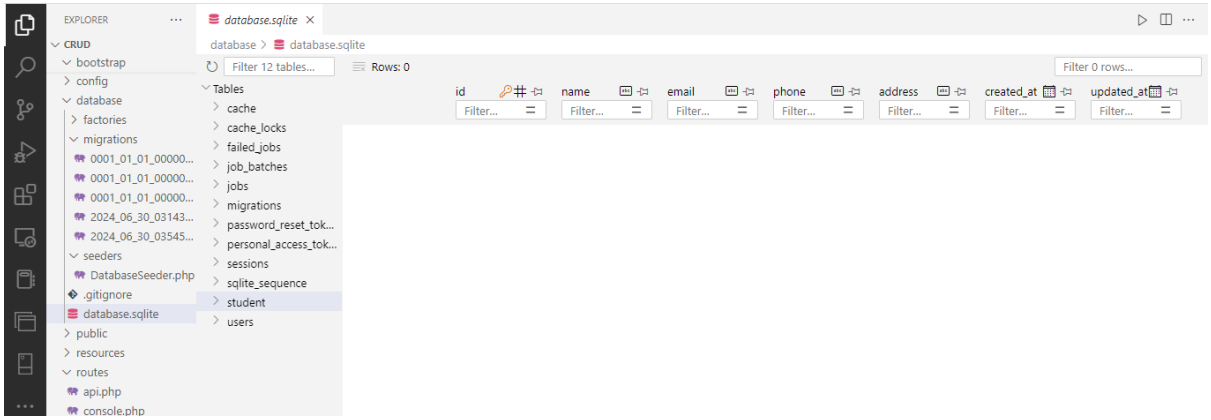

Figure 11: Tabla Estudiante

Para interactuar con nuestra tabla, hay que ingresar código, para ello nos ubicamos en la terminal nuevamente y ejecutamos el siguiente comando, la primera letra de **'Student'** siempre es con mayúscula, debido a que por lo general los modelos empiezan de esa manera:

```
php artisan make:model Student
```

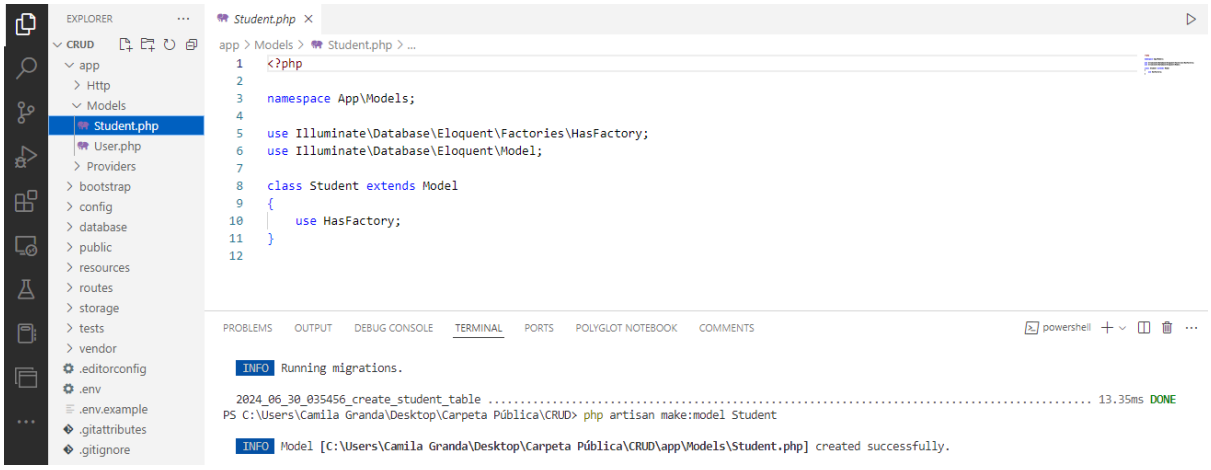

Figure 12: Modelo Estudiante

Este archivo será modificado por el siguente código:

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Student extends Model
{
    use HasFactory;

    protected $table = 'student';

    protected $fillable = [
        'name',
        'email',
        'phone',
        'address',
    ];
}
```

Una vez que tengo el modelo creado, se procede a la creación del controlador, para ello se abre nuevamente la terminal y se ejecuta el siguiente comando:

```
php artisan make:controller Api/studentController
```

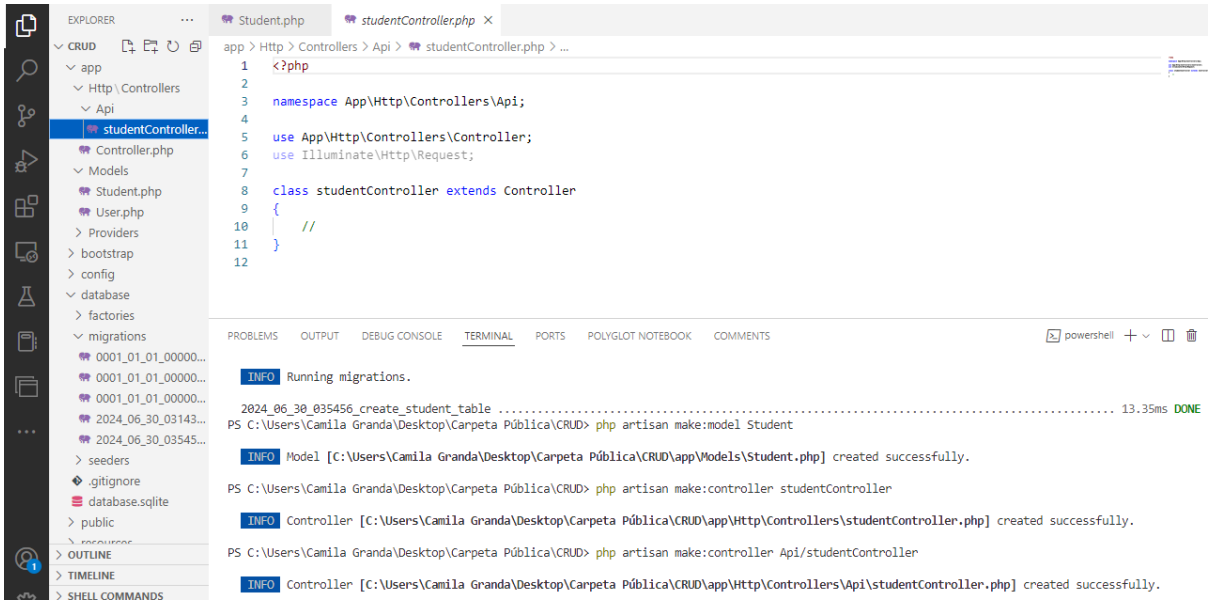

Figure 13: Controlador Estudiante

Este **'studentController'** tendrá los métodos o funciones que se van a ejecutar cuando se visite una url, a continuación se modificará el código por lo siguiente:

```php
<?php

namespace App\Http\Controllers\Api;

use App\Http\Controllers\Controller;
use App\Models\Student;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Validator;

class studentController extends Controller
{
    public function index(){
        $students = Student::all();
        $data = [
            'students' => $students,
            'status' => 200
        ];
        return response()->json($data, 200);
    }

    public function store(Request $request){
        $validator = Validator::make($request->all(), [
            'name' => 'required:max:255',
            'email' => 'required|email|unique:student',
            'phone' => 'required|digits:10',
            'address' => 'required'
        ]);

        if($validator->fails()){
            $data =[
                'message' => 'Error en la validación de los datos',
                'errors' => $validator->errors(),
                'status' => 400
            ];
            return response()->json($data, 400);
        }
```

```
        $student = Student::create([
            'name' => $request->name,
            'email' => $request->email,
            'phone' => $request->phone,
            'address' => $request->address
        ]);

        if(!$student){
            $data = [
                'message' => 'Error al crear el estudiante',
                'status' => 500
            ];
            return response()->json($data, 500);
        }

        $data = [
            'student' => $student,
            'status' => 201
        ];
        return response()->json($data, 201);
    }

    public function show($id){
        $student = Student::find($id);
        if(!$student){
            $data = [
                'message' => 'Estudiante no encontrado',
                'status' => 404
            ];
            return response()->json($data, 404);
        }

        $data = [
            'student' => $student,
            'status' => 200
        ];
        return response()->json($data, 200);
    }

    public function destroy($id){
        $student = Student::find($id);
        if(!$student){
            $data = [
                'message' => 'Estudiante no encontrado',
                'status' => 404
            ];
            return response()->json($data, 404);
        }

        $student->delete();
        $data = [
            'message' => 'Estudiante eliminado',
            'status' => 200
        ];
        return response()->json($data, 200);
    }

    public function update(Request $request, $id){
        $student = Student::find($id);
```

```
        if(!$student){
            $data = [
                'message' => 'Estudiante no encontrado',
                'status' => 404
            ];
            return response()->json($data, 404);
        }

        $validator = Validator::make($request->all(), [
            'name' => 'required:max:255',
            'email' => 'required|email|unique:student',
            'phone' => 'required|digits:10',
            'address' => 'required'
        ]);

        if($validator->fails()){
            $data =[
                'message' => 'Error en la validación de los datos',
                'errors' => $validator->errors(),
                'status' => 400
            ];
            return response()->json($data, 400);
        }

        $student->name = $request->name;
        $student->email = $request->email;
        $student->phone = $request->phone;
        $student->address = $request->address;

        $student->save();

        $data = [
            'message' => 'Estudiante actualizado',
            'student' => $student,
            'status' => 200
        ];
        return response()->json($data, 200);
    }
}
```

Luego de ello para que funcione en el postman hay que modificar el archivo **'api.php'** ubicado en **'./routes/api.php'**

```
<?php

use App\Http\Controllers\Api\studentController;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Route;

Route::get('/students', [studentController::class, 'index']);
Route::post('/students', [studentController::class, 'store']);
Route::get('/students/{id}', [studentController::class, 'show']);
Route::delete('/students/{id}', [studentController::class, 'destroy']);
Route::put('/students/{id}', [studentController::class, 'update']);
```

Una vez hecho esto, arrancamos el servidor con el siguente comando en la terminal de VSC:

```
php artisan serve
```

Luego de ello nos dirigimos a postman, elegimos **POST** y colocaremos el siguiente url:

```
http://localhost:8000/api/students
```

Dando así el siguiente resultado:

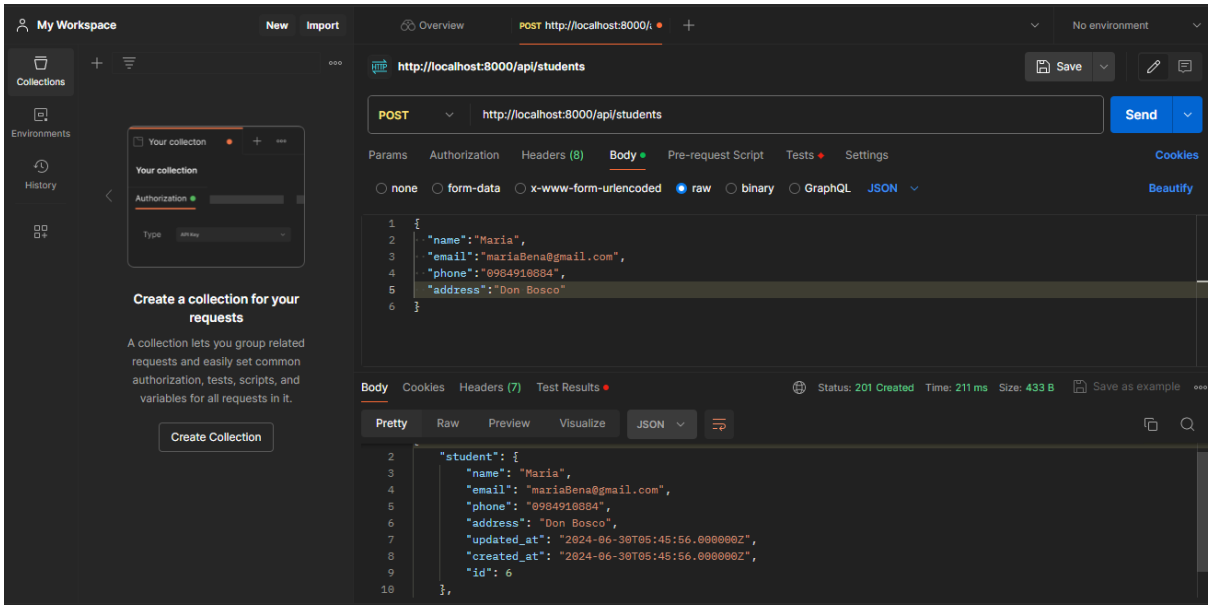

Figure 14: POST en Postman

Así mismo colocamos **GET** para observar que se haya guardado correctamente. Dando así el siguiente resultado:

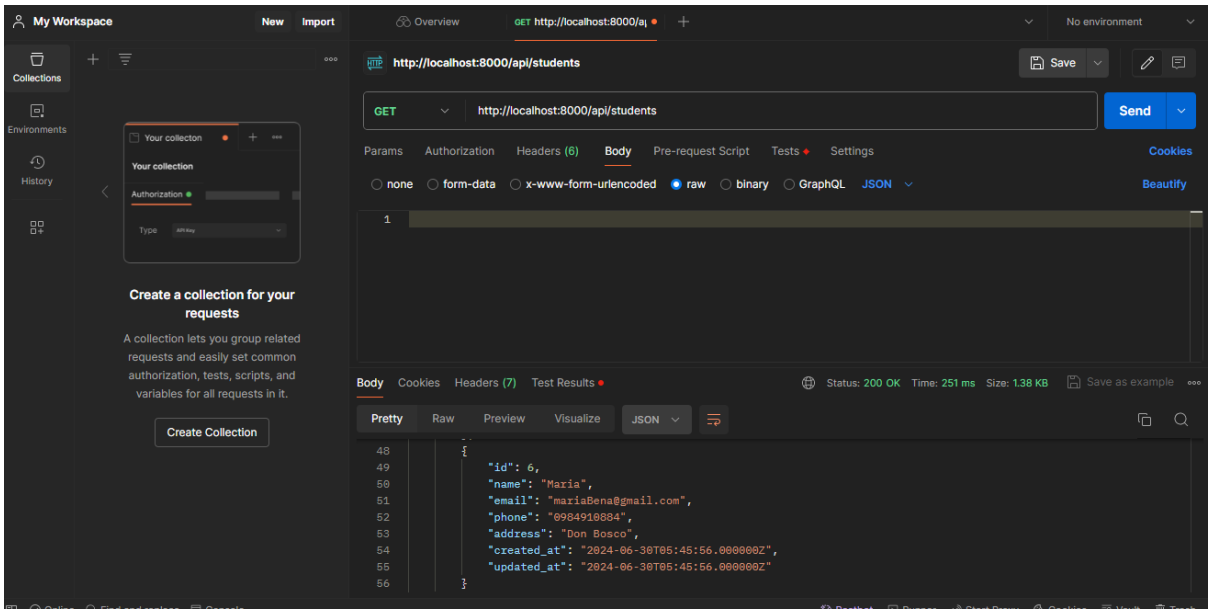

Figure 15: GET en Postman

También se puede visualizar el método GET dado un id, de tal manera que en Postman mostraría los siguientes resultados:

```
http://localhost:8000/api/students/6
```

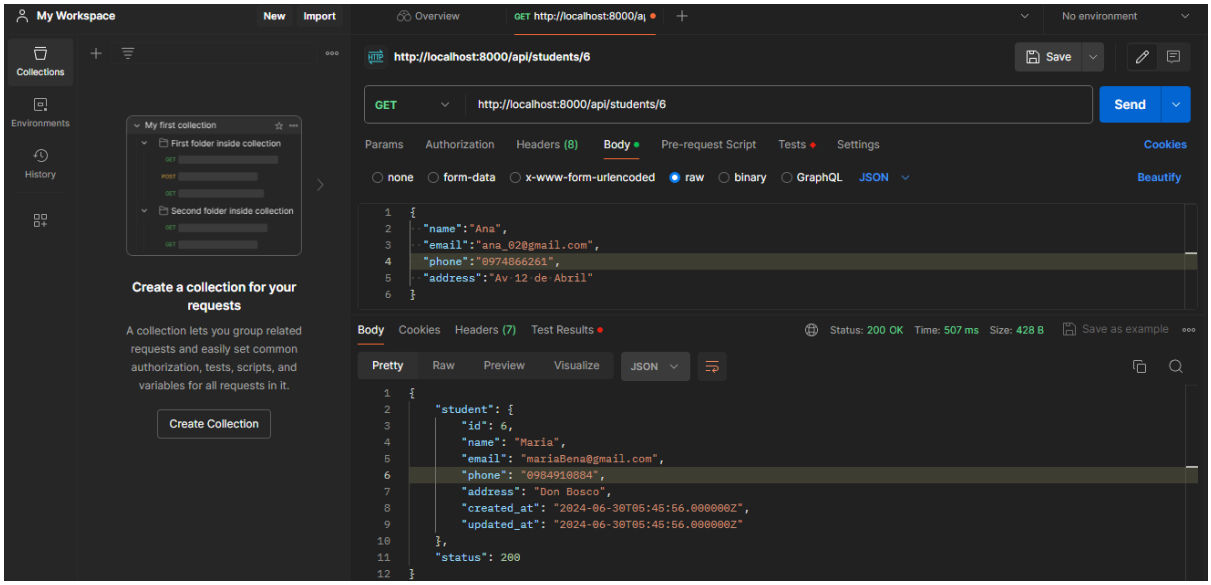

Figure 16: GET dado un id en Postman

De igual manera se puede observar en la base de datos, los datos creados:

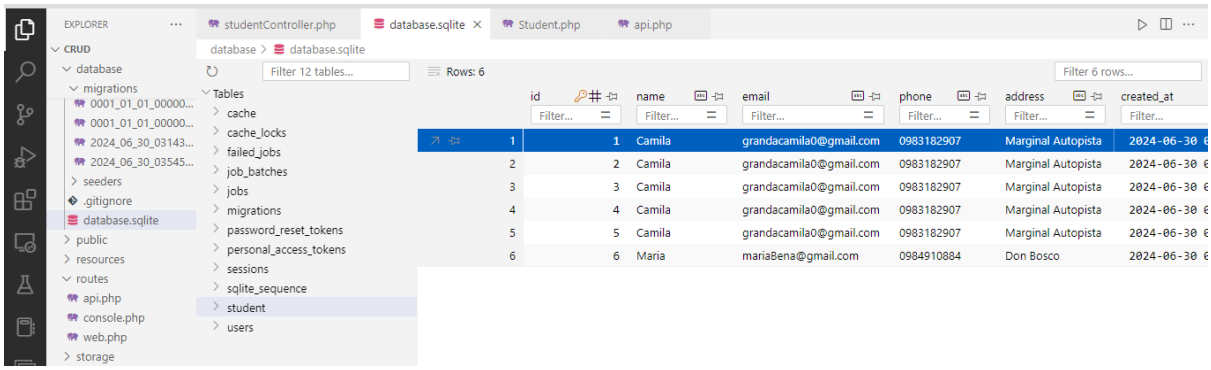

Figure 17: Base de Datos SQLite

También se puede visualizar el método DELETE dado un id, de tal manera que en Postman mostraría los siguientes resultados:

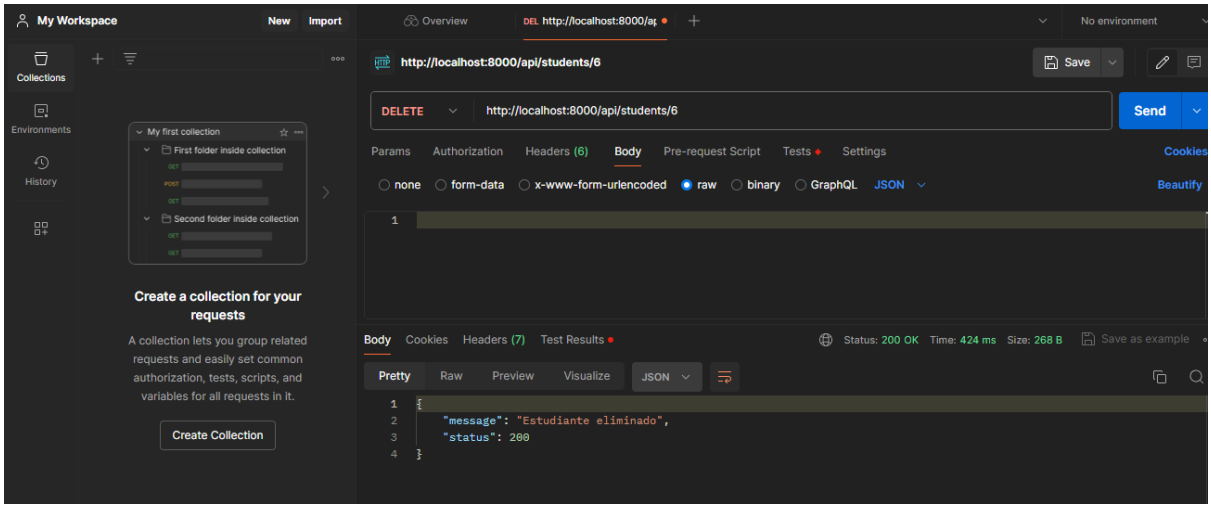

Figure 18: DELETE en Postman

Finalmente se visualiza el método PUT dado un id, de tal manera que en Postman mostraría los siguientes resultados:

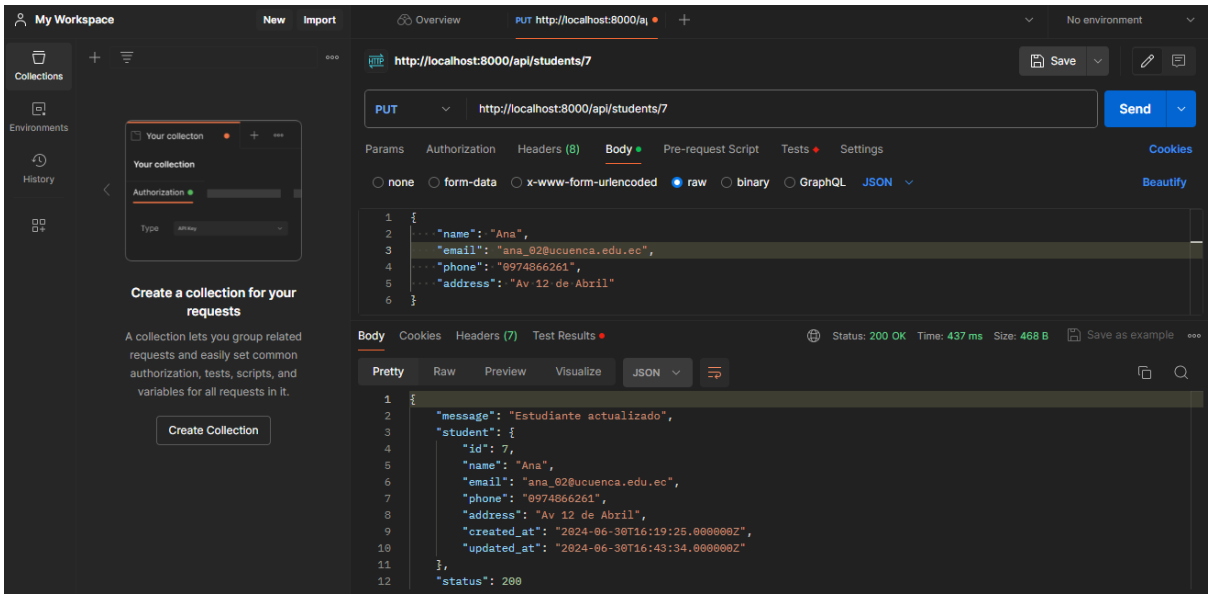

Figure 19: PUT en Postman