CONFIGURACIÓN BACKEND LARAVEL 9:

1. **Creación del proyecto larabel:**

composer create-project laravel/laravel api

2. **Creación de la base de datos para los personajes en mysql:**

CREATE DATABASE starwars_db CHARACTER SET utf8 COLLATE utf8_spanish_ci;

3. **Para manejar los datos de los personajes creamos el modelo Personaje:**

php artisan make:model Personaje –m

4. **Creamos controlador necesario:**

php artisan make:controller Api/PersonajeController --api

5. **Configuramos fichero de migración correspondiente a nuestra base de datos, añadiendo los campos necesarios:**

```php
public function up(): void
{
    Schema::create('personajes', function (Blueprint $table) {
        $table->id();
        $table->string('name');
        $table->string('height');
        $table->string('mass');
        $table->string('hair_color');
        $table->text('skin_color');
        $table->string('eye_color');
        $table->string('birth_year');
        $table->text('gender');
        $table->string('homeworld');
        $table->timestamps();
    });
}
```

6. **Para habilitar la asignación masiva en nuestra tabla, modificamos el modelo de personajes, añadiendo la siguiente propiedad:**

   protected $fillable = ['name', 'height', 'mass', 'hair_color', 'skin_color', 'eye_color', ‘birth_year’, ‘gender’, ‘homeworld’];

7.Modificamos el código de nuestro fichero controlador PersonajeController:

```php
class PersonajeController extends Controller
{
    /**
     * Display a listing of the resource.
     */
    public function index()
    {
        $personajes = Personaje::all();
        return $personajes;
    }

    /**
     * Store a newly created resource in storage.
     */
    public function store(Request $request)
    {
        $personaje = new Personaje();
        $personaje->name = $request->name;
        $personaje->height = $request->height;
        $personaje->mass = $request->mass;
        $personaje->hair_color = $request->hair_color;
        $personaje->skin_color = $request->skin_color;
        $personaje->eye_color = $request->eye_color;
        $personaje->birth_year = $request->birth_year;
        $personaje->gender = $request->gender;
        $personaje->homeworld = $request->homeworld;
        $personaje->save();
    }
```

```php
public function show(string $id)
{
    $personaje = Personaje::find($id);
    return $personaje;
}

/**
 * Update the specified resource in storage.
 */
public function update(Request $request, string $id)
{
    $personaje = Personaje::findOrFail($request->id);
    $personaje->name = $request->name;
    $personaje->height = $request->height;
    $personaje->mass = $request->mass;
    $personaje->hair_color = $request->hair_color;
    $personaje->skin_color = $request->skin_color;
    $personaje->eye_color = $request->eye_color;
    $personaje->birth_year = $request->birth_year;
    $personaje->gender = $request->gender;
    $personaje->homeworld = $request->homeworld;

    $personaje->save();
    return $personaje;
}
```

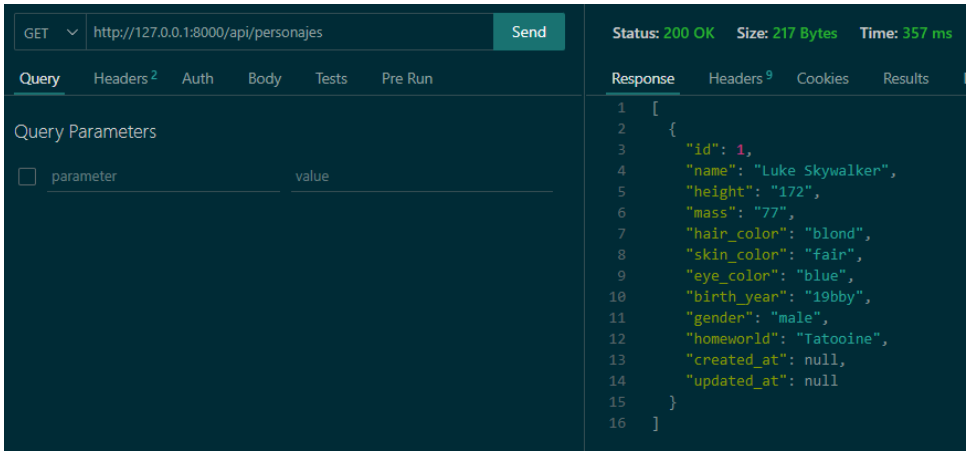
```php
public function destroy(string $id)
{
    $personaje = Personaje::destroy($id);
    return $personaje;
}
```

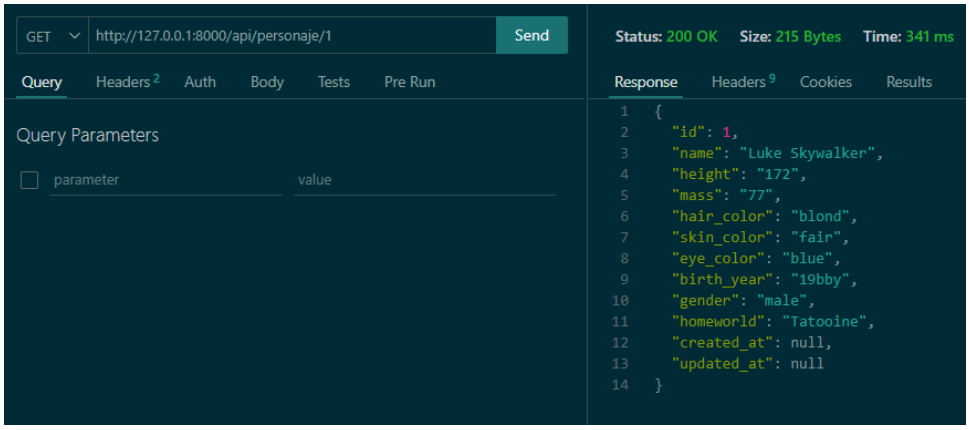
8.Agregación de las rutas necesarias en el fichero:

```php
use App\Http\Controllers\Api\PersonajeController;

/*
|--------------------------------------------------------------------------
| API Routes
|--------------------------------------------------------------------------
|
| Here is where you can register API routes for your application. These
| routes are loaded by the RouteServiceProvider and all of them will
| be assigned to the "api" middleware group. Make something great!
|
*/

Route::middleware('auth:sanctum')->get('/user', function (Request $request) {
    return $request->user();
});

Route::controller(PersonajeController::class)->group(function (){
    Route::get('/personajes', 'index');
    Route::post('/personaje', 'store');
    Route::get('/personaje/{id}', 'show');
    Route::put('/personaje/{id}', 'update');
    Route::delete('/personaje/{id}', 'destroy');
});
```

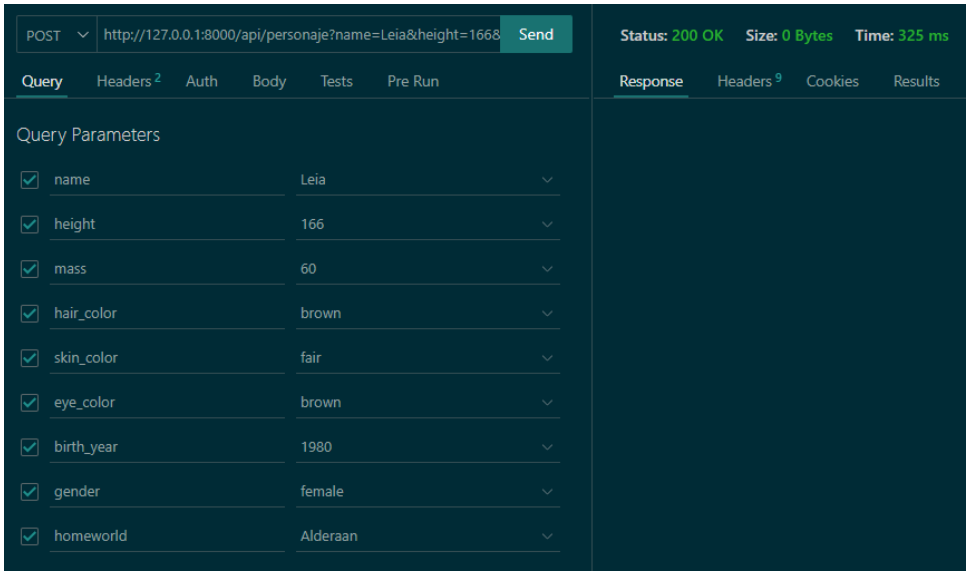
- **Para testear nuestra API, utilizaremos un simulador de llamadas a API, en nuestro caso hemos utilizado la extensión de Thunder en VS Code:**

  GET: Obtenemos dato especifico de la base de datos

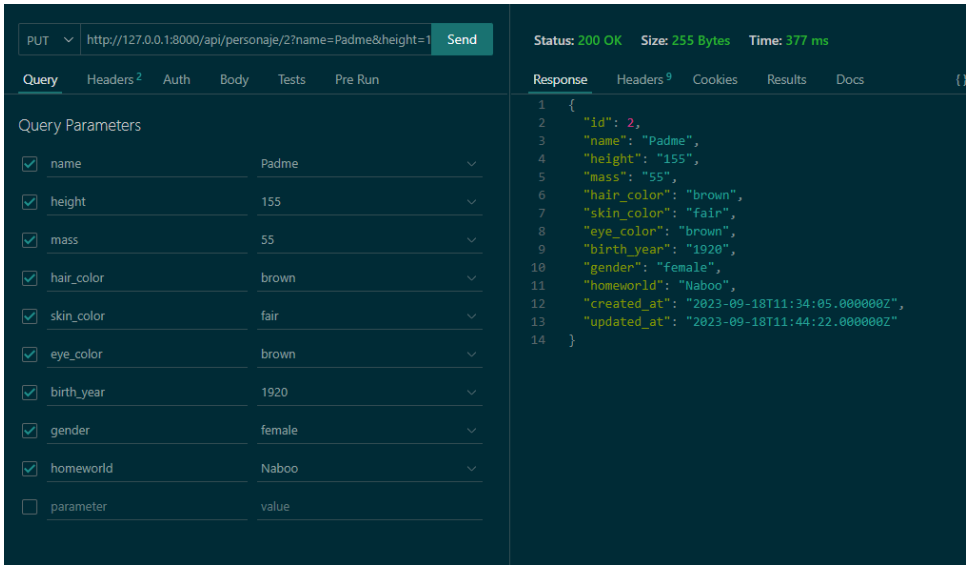




POST: Se solicita la subida de recursos a la base de datos



PUT: Actualiza recurso específico de nuestra base de datos



DELETE: Borra recurso específico de nuestra base de datos