

UNIVERSIDAD POLITÉCNICA DE TECÁMAC.

Materia: Cliente/Servidor.

Docente: Emmanuel Torres Servín.

Trabajo: API.

Team:

- Natali Joselin Alemán Perez.
- Luz Alexia Fuentes Cortes.
- Marco Joel Ángel Velasco.
- José Eduardo de la Cruz Medina.

Grupo: 1523IS.

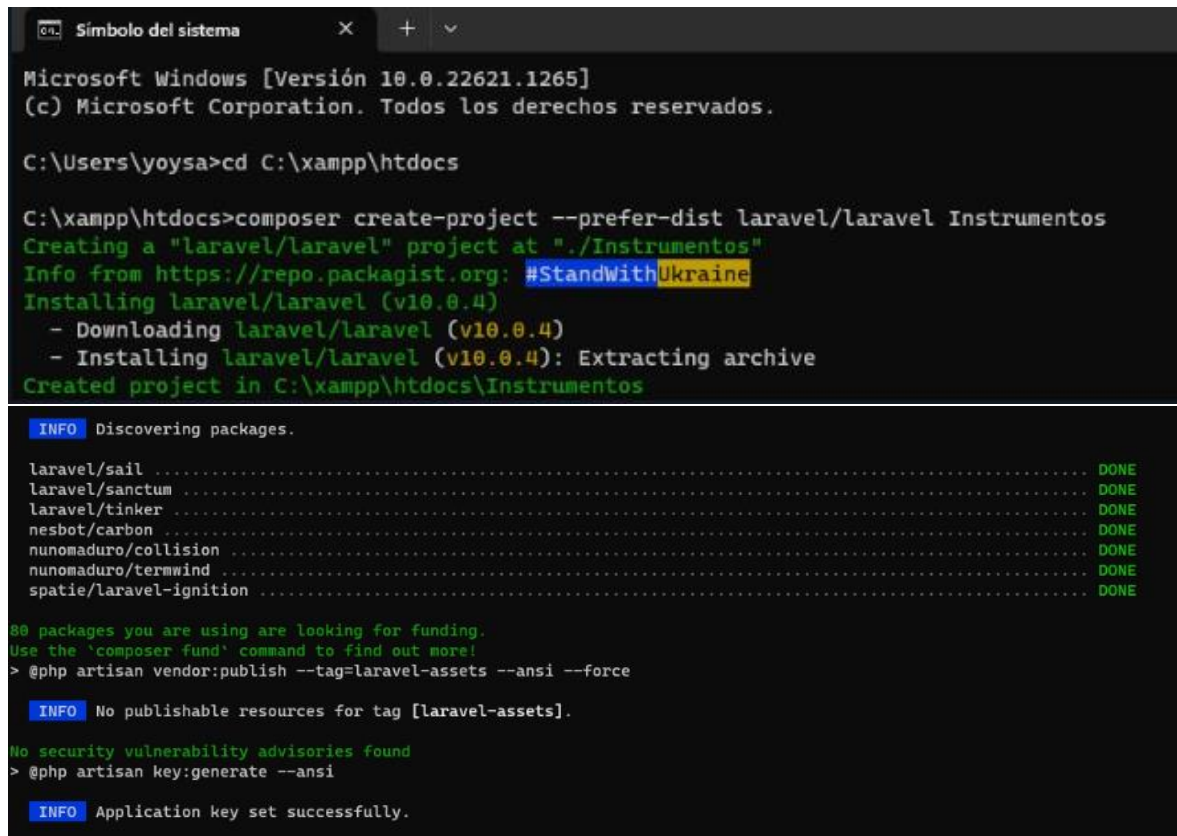
Fecha de entrega: 13 de Marzo del 2023.

Índice:

Reporte del proceso del diseño del modelo de acceso y presentación de datos.....	2
Programas de aplicación del modelo de acceso y presentación de datos.....	8
Programas de aplicación MVC.....	9
Interfaz de programación de aplicaciones API.....	10
Link de repositorio GitHub:	11

Reporte del proceso del diseño del modelo de acceso y presentación de datos.

Creación del proyecto:



```

Microsoft Windows [Versión 10.0.22621.1265]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\yoysa>cd C:\xampp\htdocs

C:\xampp\htdocs>composer create-project --prefer-dist laravel/laravel Instrumentos
Creating a "laravel/laravel" project at "./Instrumentos"
Info from https://repo.packagist.org: #StandWithUkraine
Installing laravel/laravel (v10.0.4)
  - Downloading laravel/laravel (v10.0.4)
  - Installing laravel/laravel (v10.0.4): Extracting archive
Created project in C:\xampp\htdocs\Instrumentos

INFO Discovering packages.

laravel/sail ..... DONE
laravel/sanctum ..... DONE
laravel/tinker ..... DONE
nesbot/carbon ..... DONE
nunomaduro/collision ..... DONE
nunomaduro/termwind ..... DONE
spatie/laravel-ignition ..... DONE

80 packages you are using are looking for funding.
Use the 'composer fund' command to find out more!
> @php artisan vendor:publish --tag=laravel-assets --ansi --force

INFO No publishable resources for tag [laravel-assets].

No security vulnerability advisories found
> @php artisan key:generate --ansi

INFO Application key set successfully.
  
```

Creación de la base de datos:

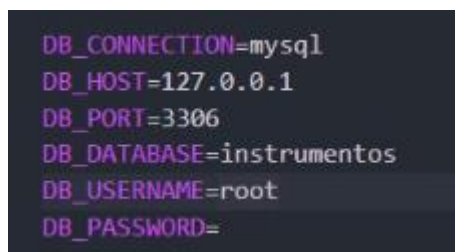


Bases de datos

Crear base de datos

instrumentos utf8mb4_general_ci Crear

Conexión de la base de datos en laravel:



```

DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=instrumentos
DB_USERNAME=root
DB_PASSWORD=
  
```

Creación de la migración:

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
PS C:\xampp\htdocs\Instrumentos> php artisan make:migration create_table_instrumento
[INFO] Migration [C:\xampp\htdocs\Instrumentos\database\Migrations\2023_03_11_220125_create_table_instrumento.php] created successfully.

```

Creación de la tabla instrumento en la migración:

```

1  <?php
2
3  use Illuminate\Database\Migrations\Migration;
4  use Illuminate\Database\Schema\Blueprint;
5  use Illuminate\Support\Facades\Schema;
6
7  return new class extends Migration
8  {
9      /**
10       * Run the migrations.
11       */
12     public function up(): void
13     {
14         Schema::create('instrumento', function (Blueprint $table) {
15             $table->id();
16             $table->string('nombre', 30);
17             $table->integer('cantidad');
18             $table->string('marca', 30);
19             $table->float('precio', 8, 2);
20             $table->timestamps();
21         });
22     }
23
24     /**
25      * Reverse the migrations.
26      */
27     public function down(): void
28     {
29         Schema::dropIfExists('instrumento');
30     }
31 };
32

```

Corriendo la migración:

```

PS C:\xampp\htdocs\Instrumentos> php artisan migrate
[INFO] Preparing database.

Creating migration table ..... 27ms DONE

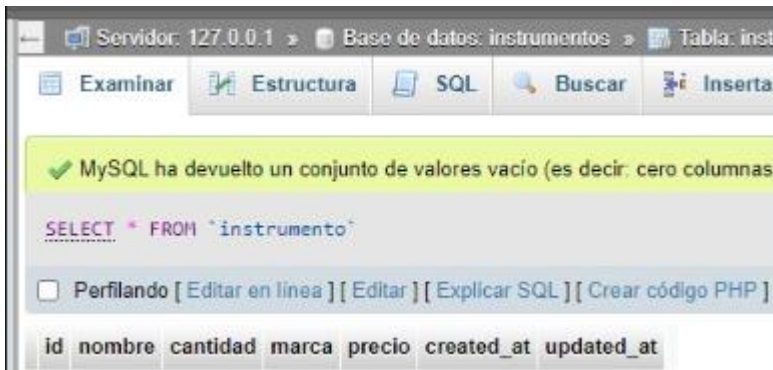
[INFO] Running migrations.

2014_10_12_000000_create_users_table ..... 30ms DONE
2014_10_12_100000_create_password_reset_tokens_table ..... 39ms DONE
2019_08_19_000000_create_failed_jobs_table ..... 27ms DONE
2019_12_14_000001_create_personal_access_tokens_table ..... 41ms DONE

```

Tabla	Replicación	Acción	Filas	Tipo	Cotejamiento	Tamaño	Residuo a depurar
<input type="checkbox"/> failed_jobs	✓	★ Examinar Estructura Buscar Insertar Vaciar Eliminar	0	InnoDB	utf8mb4_unicode_ci	32.0 KB	-
<input type="checkbox"/> instrumento	✓	★ Examinar Estructura Buscar Insertar Vaciar Eliminar	0	InnoDB	utf8mb4_unicode_ci	16.0 KB	-
<input type="checkbox"/> migrations	✓	★ Examinar Estructura Buscar Insertar Vaciar Eliminar	5	InnoDB	utf8mb4_unicode_ci	16.0 KB	-
<input type="checkbox"/> password_reset_tokens	✓	★ Examinar Estructura Buscar Insertar Vaciar Eliminar	0	InnoDB	utf8mb4_unicode_ci	16.0 KB	-
<input type="checkbox"/> personal_access_tokens	✓	★ Examinar Estructura Buscar Insertar Vaciar Eliminar	0	InnoDB	utf8mb4_unicode_ci	48.0 KB	-
<input type="checkbox"/> users	✓	★ Examinar Estructura Buscar Insertar Vaciar Eliminar	0	InnoDB	utf8mb4_unicode_ci	32.0 KB	-
6 tablas	Replicación	Número de filas	5	InnoDB	utf8mb4_general_ci	160.0 KB	0 B

☐ Seleccionar todo
 Para los elementos que están marcados: ▼



Creación del modelo:

```
PS C:\xampp\htdocs\Instrumentos> php artisan make:model Instrumento
```

INFO Model [C:\xampp\htdocs\Instrumentos\app\Models\Instrumento.php] created successfully.

```
app > Models > Instrumento.php
1  <?php
2
3  namespace App\Models;
4
5  use Illuminate\Database\Eloquent\Factories\HasFactory;
6  use Illuminate\Database\Eloquent\Model;
7
8  class Instrumento extends Model
9  {
10     protected $table = 'instrumento';
11     protected $fillable = ['nombre', 'cantidad', 'marca', 'precio'];
12     use HasFactory;
13 }
```

Creación del controlador:

```
PS C:\xampp\htdocs\Instrumentos> php artisan make:controller InstrumentoController -r
```

INFO Controller [C:\xampp\htdocs\Instrumentos\app\Http\Controllers\InstrumentoController.php] created successfully.

Modificación del controlador para guardar datos en la base de datos.

```
public function store(Request $request)
{
    //Instanciar la clase Instrumento
    $instrumento = new Instrumento();
    //Asignar los valores de la petición al objeto
    $instrumento->nombre = $request->input('nombre');
    $instrumento->cantidad = $request->input('cantidad');
    $instrumento->marca = $request->input('marca');
    $instrumento->precio = $request->input('precio');
    //Guardar el objeto en la base de datos
    $instrumento->save();
}

/**
 * Display the specified resource.
 */
public function show(string $id)
{
    return Instrumento::findOrFail($id)->get();
}
```

Creación de la ruta en la API:

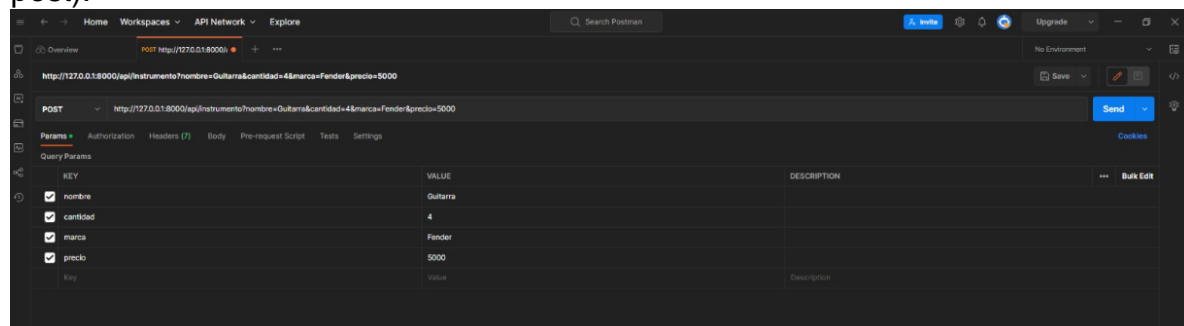
```
Route::resource('instrumento', InstrumentoController::class);
```

Verificación de que la ruta es correcta:

```
PS C:\xampp\htdocs\Instrumentos> php artisan route:list

GET|HEAD / ..... ignition.executeSolution ..... Spatie\LaravelIgnition > ExecuteSolutionController
POST / ..... ignition/health-check ..... Spatie\LaravelIgnition > HealthCheckController
GET|HEAD / ..... ignition/update-config ..... Spatie\LaravelIgnition > UpdateConfigController
POST /api/instrumento ..... instrumento.index ..... InstrumentoController@index
POST /api/instrumento ..... instrumento.store ..... InstrumentoController@store
GET|HEAD /api/instrumento/create ..... instrumento.create ..... InstrumentoController@create
GET|HEAD /api/instrumento/{instrumento} ..... instrumento.show ..... InstrumentoController@show
PUT|PATCH /api/instrumento/{instrumento} ..... instrumento.update ..... InstrumentoController@update
DELETE /api/instrumento/{instrumento} ..... instrumento.destroy ..... InstrumentoController@destroy
GET|HEAD /api/instrumento/{instrumento}/edit ..... instrumento.edit ..... InstrumentoController@edit
```

Verificando en postman (insertando datos en la base de datos a través del método post).



Datos insertados correctamente (API funcionando):

Mostrando filas 0 - 0 (total de 1, La consulta tardó 0,0004 segundos.)

```
SELECT * FROM `instrumento`
```

Perfilando [Editar en línea] [Editar] [Explicar SQL] [Crear código PHP] [Actualizar]

Mostrar todo | Número de filas: 25 | Filtrar filas: Buscar en esta tabla

Opciones extra

	id	nombre	cantidad	marca	precio	created_at	updated_at
<input type="checkbox"/> Editar <input type="checkbox"/> Copiar <input type="checkbox"/> Borrar	2	Guitarra	4	Fender	5000.00	2023-03-11 23:18:06	2023-03-11 23:18:06

Seleccionar todo | Para los elementos que están marcados: Editar Copiar Borrar Exportar

Modificación del método index para mostrar los valores agregados:

```
public function index()
{
    return Instrumento::all();
}
```

```
[{"id":2,"nombre":"Guitarra","cantidad":4,"marca":"Fender","precio":5000,"created_at":"2023-03-11T23:18:06.000000Z","updated_at":"2023-03-11T23:18:06.000000Z"}]
```

Modificación del método destroy para eliminar un registro de la base de datos:

```
public function destroy(string $id)
{
    $instrumento = Instrumento::findOrFail($id);
    $instrumento->delete();
}
```

Eliminando el registro con id = 2.

http://127.0.0.1:8000/api/instrumento/2

DELETES http://127.0.0.1:8000/api/instrumento/2

Params Authorization Headers (0) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE	DESCRIPTION
key	value	Description

Registro eliminado:

Servidor: 127.0.0.1 » Base de datos: instrumentos » Tabla: instrumento

Examinar Estructura SQL Buscar Insertar Exportar Importar


✓ MySQL ha devuelto un conjunto de valores vacío (es decir: cero columnas). (La consulta tardó 0,0004 segundos)

```
SELECT * FROM `instrumento`
```

☐ Perfilando [[Editar en línea](#)] [[Editar](#)] [[Explicar SQL](#)] [[Crear código PHP](#)] [[Actualizar](#)]

id	nombre	cantidad	marca	precio	created_at	updated_at
----	--------	----------	-------	--------	------------	------------

Operaciones sobre los resultados de la consulta

 [Crear vista](#)

Programas de aplicación del modelo de acceso y presentación de datos.

El modelo de acceso y presentación de datos es un patrón de diseño utilizado en la programación de aplicaciones para separar la lógica de acceso a datos de la lógica de presentación de datos. Este patrón ayuda a mantener un código más limpio, modular y fácil de mantener.

Una API (Interfaz de Programación de Aplicaciones) es un conjunto de funciones, protocolos y herramientas que permiten a los desarrolladores crear aplicaciones que se comuniquen con otras aplicaciones o servicios. La implementación del modelo de acceso y presentación de datos en una API puede mejorar la escalabilidad y la flexibilidad de la aplicación, ya que permite a los desarrolladores realizar cambios en la lógica de acceso a datos sin afectar la lógica de presentación.

A continuación, se presentan algunos ejemplos de programas de aplicación del modelo de acceso y presentación de datos en una API:

- Utilizar un patrón DAO (Objeto de Acceso a Datos) para separar la lógica de acceso a datos de la lógica de presentación en una API RESTful. El patrón DAO se utiliza para encapsular el acceso a datos y proporcionar una interfaz abstracta para interactuar con los datos. De esta manera, la lógica de presentación puede interactuar con los datos sin conocer los detalles de implementación de la capa de acceso a datos.
- Implementar un patrón MVC (Modelo-Vista-Controlador) para separar la lógica de acceso a datos de la lógica de presentación en una API. El patrón MVC se utiliza para dividir la aplicación en tres componentes principales: el modelo, la vista y el controlador. El modelo es responsable de la lógica de acceso a datos, la vista es responsable de la presentación de datos y el controlador es responsable de la lógica de negocio. De esta manera, la lógica de acceso a datos se puede modificar sin afectar la lógica de presentación.
- Utilizar una arquitectura de microservicios para separar la lógica de acceso a datos de la lógica de presentación en una API. En una arquitectura de microservicios, cada servicio se encarga de una función específica y se comunica con otros servicios a través de API. De esta manera, la lógica de acceso a datos se puede escalar y modificar de forma independiente de la lógica de presentación.

Programas de aplicación MVC.

El patrón de arquitectura MVC (Modelo-Vista-Controlador) se puede aplicar a la construcción de una API.

En una API, el modelo representaría la lógica de negocio y los datos de la aplicación, la vista sería la representación del resultado de la API y el controlador manejaría las solicitudes de los clientes y los procesos de la aplicación.

Un ejemplo de cómo implementar un programa de aplicación MVC en una API sería el siguiente:

- **Modelo:** La lógica de negocio y los datos se almacenarían en una base de datos o en un servicio externo, como un sistema de gestión de contenido o una plataforma de pago.
- **Vista:** La representación del resultado de la API se generarían utilizando una plantilla, como JSON o XML, que se enviaría al cliente como respuesta a la solicitud.
- **Controlador:** El controlador sería responsable de manejar las solicitudes de los clientes y procesarlas de acuerdo con la lógica de negocio de la aplicación. Esto podría incluir la validación de datos de entrada, la manipulación de datos en la base de datos y la generación de respuestas de API adecuadas.

En general, es importante tener en cuenta que la aplicación de MVC en una API puede variar según la tecnología y los requisitos específicos de la aplicación. Sin embargo, la estructura básica de separar la lógica de negocio y los datos del procesamiento de solicitudes y la generación de respuestas sigue siendo el enfoque recomendado para el desarrollo de API.

Interfaz de programación de aplicaciones API.

Una interfaz de programación de aplicaciones (API, por sus siglas en inglés) es un conjunto de reglas, protocolos y herramientas que se utilizan para construir aplicaciones de software. Una API define una serie de operaciones, comandos y protocolos que pueden ser utilizados por los desarrolladores de software para acceder a una funcionalidad específica de una aplicación o sistema.

Las APIs permiten que diferentes aplicaciones interactúen entre sí de manera eficiente y efectiva, ya que proporcionan una forma estándar de comunicación entre ellas. Esto significa que los desarrolladores pueden crear aplicaciones que utilizan una API para acceder a la funcionalidad de otra aplicación o sistema, sin tener que conocer los detalles internos de cómo funciona ese sistema.

Por ejemplo, una API de redes sociales puede permitir a los desarrolladores acceder a los datos de usuario de una plataforma de redes sociales, como Facebook o Twitter. Esto significa que los desarrolladores pueden crear aplicaciones que utilizan estos datos para proporcionar servicios adicionales, como la integración de redes sociales en una aplicación de mensajería o la creación de una aplicación de análisis de datos de redes sociales.

En resumen, las APIs son herramientas esenciales para la creación de aplicaciones de software modernas, ya que permiten que diferentes aplicaciones y sistemas se comuniquen y trabajen juntos de manera efectiva.

Link de repositorio GitHub:

<https://github.com/MarcoJoel1234/API.git>