

Computo III

Formularios con Laravel

En Laravel, los formularios son una manera de recopilar información del usuario a través de una interfaz gráfica. Un formulario típicamente contiene una serie de campos de entrada (como cuadros de texto, botones de opción, menús desplegables, etc.) que el usuario puede completar y enviar al servidor para su procesamiento.

En Laravel, se puede crear fácilmente formularios HTML utilizando la clase Form. Esta clase proporciona métodos para crear diferentes tipos de elementos de formulario, como campos de entrada de texto, casillas de verificación, botones de radio, menús desplegables, etc. Además, Laravel también proporciona la capacidad de validar y procesar los datos del formulario enviados por el usuario utilizando el sistema de validación y controladores de Laravel.

Validaciones en formularios

Antes de crear la validación, se muestra la migración correspondiente:

```
database > migrations > 2023_04_10_144345_create_costumers_table.php
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('costumers', function (Blueprint $table) {
15              $table->id();
16              $table->string('name');
17              $table->string('email');
18              $table->string('phone');
19              $table->string('address')->nullable();
20              $table->timestamps();
21          });
22      }
23
24      /**
25       * Reverse the migrations.
26       */
27      public function down(): void
28      {
29          Schema::dropIfExists('costumers');
30      }
31  };
32
```

Seguido abrir el controlador correspondiente, específicamente en el método store() que es donde se controlan los errores de validación del formulario, y es en este método en donde se agregara el código de validación.

Se hará uso de la variable request que se pasa como parámetro a la función store(), la cual contiene los campos del formulario; el método validate() recibirá un array con los campos del formulario y las validaciones correspondientes.

Mostrar errores en formulario

```
1 <div class="box box-info padding 1">
2   <div class="box-body">
3
4     <div class="form-group">
5       {{ Form::label('name') }}
6       {{ Form::text('name', $customer->name, ['class' => 'form-control', 'placeholder' => 'Name
7       @error('name')
8         <p class="error-message">{{ $message }}</p>
9       @enderror
10    </div>
11    <div class="form-group">
12      {{ Form::label('email') }}
13      {{ Form::text('email', $customer->email, ['class' => 'form-control', 'placeholder' => 'Em
14      @error('email')
15        <p class="error-message">{{ $message }}</p>
16      @enderror
17    </div>
18    <div class="form-group">
19      {{ Form::label('phone') }}
20      {{ Form::text('phone', $customer->phone, ['class' => 'form-control', 'placeholder' => 'Ph
21      @error('phone')
22        <p class="error-message">{{ $message }}</p>
23      @enderror
24    </div>
25    <div class="form-group">
26      {{ Form::label('address') }}
27      {{ Form::text('address', $customer->address, ['class' => 'form-control', 'placeholder' =>
28      @error('address')
29        <p class="error-message">{{ $message }}</p>
30      @enderror
31    </div>
32  </div>
33 </div>
34 <div class="box-footer mt20">
35   <button type="submit" class="btn btn-primary">{{ ('Submit') }}</button>
36 </div>
37 </div>
38
39 <style>
40   p {color: red;}
41 </style>
```

Debajo del input a validar escribir la directiva @error el cual permite imprimir el mensaje de error que corresponde a este campo.

Validaciones en Laravel

Existen varias formas de validar una aplicación para cubrir aspectos de seguridad como SQL Injection, ataques XSS o CSRF, algunas de ellas son:

- Validación de lado del cliente (Javascript y etiquetas HTML).
- Validación a nivel de base de datos (Migraciones y modelos).
- Validación de formularios (Request).

Validación del lado del cliente

Se puede validar que los campos de un formulario sean requeridos al agregar el atributo required

```
<form action="#">
  Username:
  <input type="text" name="username" required>
  <input type="submit">
</form>
```

El atributo required es un atributo booleano. Cuando esta presente, este especifica que un campo debe ser rellenado antes de ser enviado el contenido del formulario.

El atributo required trabaja con los siguientes tipos de input:

- text
- search
- url
- tel
- email
- password
- data pickers
- number
- checkbox

El atributo pattern

Con required solo se necesita de cualquier valor en el elemento <input> para ser válido, pero utilizando el atributo pattern en conjunto, se logra que se verifique no solo la presencia de un valor, sino que este valor debe contener un formato, una longitud o un tipo de dato específico. Esto último se logra definiendo un patrón con expresiones regulares.

```
<label for="tel">Teléfono 10 dígitos empezando por 503</label>  
<input type="text" pattern="^503\d{8}$">
```

Validación de formularios con plugins JQuery

El mejor plugin JQuery para validar formularios es formvalidation. Sin embargo formvalidation es un plugin que tiene un costo dependiendo la licencia que se requiera utilizar. Formvalidation es compatible con los formularios de los frameworks CSS más populares:

- Bootstrap
- Foundation
- Pure
- Semantic UI
- Otros

Validación del lado del servidor.

Pero podemos especificar que solo querremos aceptar números haciendo una condición de la siguiente manera:

```
web.php X
routes > web.php
23
24 Route::get('/usuario/{id}', function ($id=0) {
25     return 'ID USuario: '. $id;
26 }->where('id', '[0-9]+');
```

En este ejemplo estamos haciendo uso de expresiones regulares el nos brinda un mundo de posibilidades.

BLADE

Blade es un motor de plantillas (template engine) en el framework de PHP Laravel que permite definir la interfaz de usuario de una aplicación web de una manera fácil.

Blade proporciona una sintaxis simple y legible para definir vistas, que se compilan en código PHP y se ejecutan en el servidor para generar el HTML que se enviará al navegador.

Algunas de las características de Blade incluyen:

- Herencia de plantillas: permite definir una plantilla base que puede ser extendida por otras plantillas, lo que permite reutilizar código y mantener una estructura consistente en toda la aplicación.
- Secciones: permiten definir secciones en una plantilla que pueden ser rellenas por otras vistas que la extienden, lo que facilita la creación de componentes reutilizables.
- Directivas: Blade proporciona una serie de directivas (como @if, @foreach, @include, entre otras) que permiten escribir lógica de control de flujo y reutilizar secciones de código.

Se tiene una ruta en la aplicación que devuelve una vista que muestra una lista de usuarios. Se puede definir la vista de la siguiente manera utilizando Blade:

```

users.blade.php X
resources > views > users.blade.php
1  @extends('layouts.app')
2
3  @section('content')
4      <h1>List of Users</h1>
5
6      <ul>
7          @foreach ($users as $user)
8              <li>{{ $user->name }}</li>
9          @endforeach
10     </ul>
11 @endsection

```

En este ejemplo, se está utilizando la directiva `@extends` para indicar que esta vista extiende la plantilla base `layouts.app`. La directiva `@section` permite definir el contenido de la sección `content` que se rellenará en la plantilla base.

Dentro de la sección `content`, se está utilizando la sintaxis de Blade para escribir un bucle `@foreach` que recorre la lista de usuarios y muestra su nombre utilizando la sintaxis `{{ $user->name }}`, que se traduce en código PHP como `echo $user->name;`.

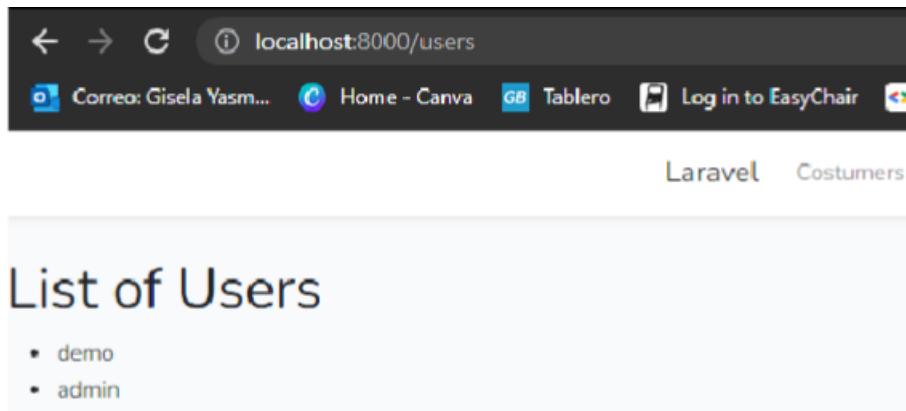
Ahora, en el controlador tenemos una función que devuelve esta vista y le pasamos la lista de usuarios, la vista se renderizará con los datos correspondientes:

```

users.blade.php  UserController.php X
app > Http > Controllers > UserController.php
1  <?php
2
3  namespace App\Http\Controllers;
4
5  use Illuminate\Http\Request;
6
7  /**
8   * Class CostumerController
9   * @package App\Http\Controllers
10  */
11  class UserController extends Controller
12  {
13      public function index()
14      {
15          $users = User::all();
16
17          return view('users', ['users' => $users]);
18      }
19  }

```

Al ejecutar la aplicación la salida se deberá visualizar de la siguiente manera:



Rest API Laravel

Laravel le permite crear API RESTful de manera fácil y rápida. Este podría ser el back-end de una aplicación front-end, una fuente de datos para una aplicación móvil o un servicio para otras aplicaciones o API. REST significa transferencia de estado representacional y es un estilo arquitectónico para la comunicación de red entre aplicaciones, que se basa en un protocolo sin estado (generalmente HTTP) para la interacción.

Verbos HTTP

Los verbos HTTP representan acciones

En las API RESTful, se utilizan los verbos HTTP como acciones y los puntos finales son los recursos sobre los que se actúa.

Verbos HTTP por su significado semántico:

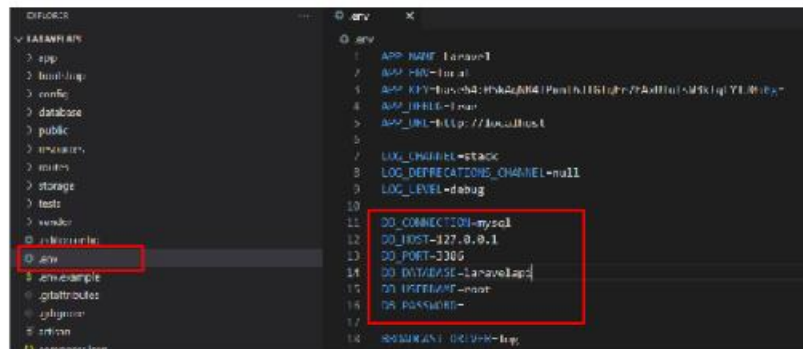
- GET: recuperar recursos
- POST: crear recursos
- PUT: actualizar recursos
- DELETE: eliminar recursos

Pasos para crear una API REST

1. Crear un proyecto de laravel

```
C:\Users\Gisela Espinoza\Documents\crudLaravel>composer create-project laravel/laravel laravelapi
Creating a "laravel/laravel" project at "../laravelapi"
```

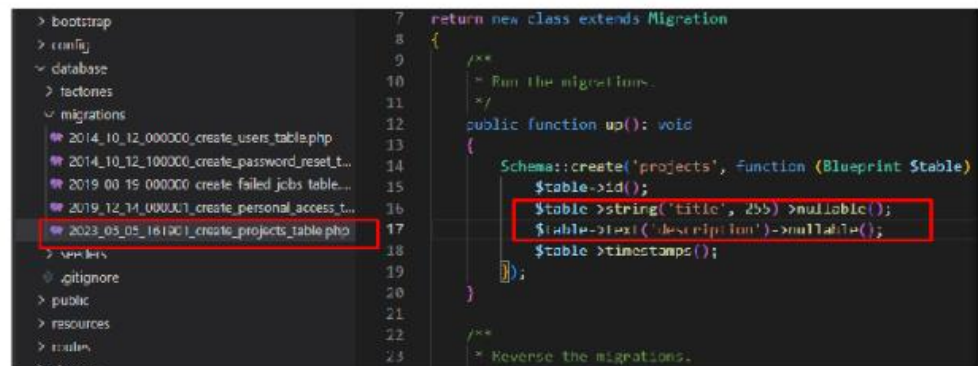
2. Configurar base de datos



3. Creación de una tabla o las tablas de la base de datos

```
C:\Users\Gisela Espinoza\Documents\crudLaravel\laravelapi>php artisan make:migration create_projects_table

INFO: Migration [C:\Users\Gisela Espinoza\Documents\crudLaravel\laravelapi\database\Migrations\2023_05_05_161901_create_projects_table.php] created successfully.
```



4. Ejecutar la migración

```
C:\Users\Gisela Espinoza\Documents\crudLaravel\laravelapi>php artisan migrate

INFO: Preparing database.

Creating migration table ..... 37ms DONE

INFO: Running migrations.

2023_05_05_161901_create_projects_table ..... 16ms DONE
2019_12_14_000001_create_personal_access_tokens_table ..... 48ms DONE
2019_08_19_000000_create_failed_jobs_table ..... 38ms DONE
2014_10_12_100000_create_password_reset_tokens_table ..... 76ms DONE
2014_10_12_000000_create_users_table ..... 53ms DONE
```


5. Agregar el modelo y el controlador

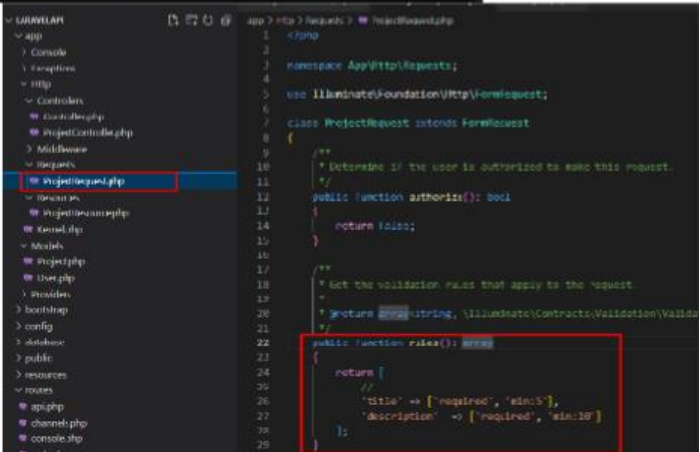
```
C:\Users\Gisela Espinoza\Documents\crudLaravel\laravelapi>php artisan make:model Project
INFO: Model [C:\Users\Gisela Espinoza\Documents\crudLaravel\laravelapi\app\Models/Project.php] created successfully.
```

```
C:\Users\Gisela Espinoza\Documents\crudLaravel\laravelapi>php artisan make:controller ProjectController --no-model
INFO: Controller [C:\Users\Gisela Espinoza\Documents\crudLaravel\laravelapi\app\Http\Controllers/ProjectController.php] created successfully.
```

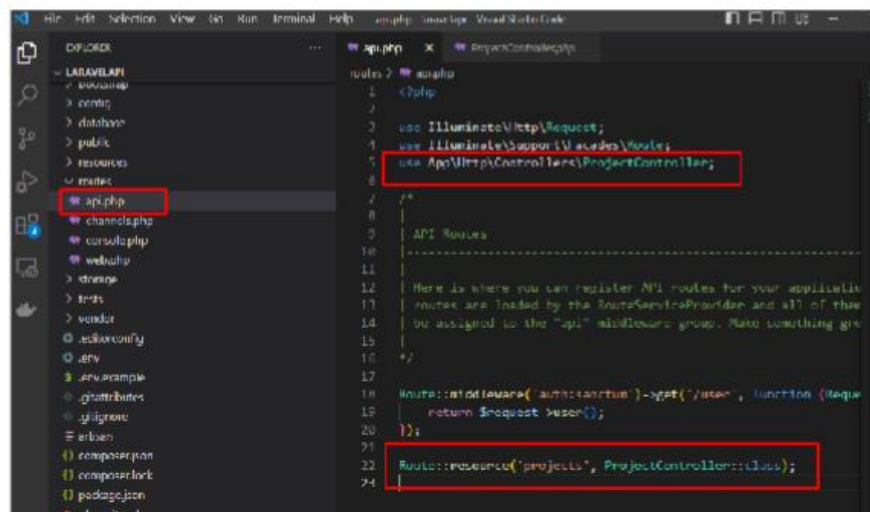
6. Crear ProjectRequest para validar datos y ProjectResource para mostrar los datos limpios

```
C:\Users\Gisela Espinoza\Documents\crudLaravel\laravelapi>php artisan make:request ProjectRequest
INFO: Request [C:\Users\Gisela Espinoza\Documents\crudLaravel\laravelapi\app\Http\Requests/ProjectRequest.php] created successfully.

C:\Users\Gisela Espinoza\Documents\crudLaravel\laravelapi>php artisan make:resource ProjectResource
INFO: Resource [C:\Users\Gisela Espinoza\Documents\crudLaravel\laravelapi\app\Http/Resources/ProjectResource.php] created successfully.
```



7. Agrega la ruta de recursos



```
routes.php
1 <?php
2
3 use Illuminate\Http\Request;
4 use Illuminate\Support\Facades\Route;
5 use App\Http\Controllers\ProjectController;
6
7 /*
8 |
9 | API Routes
10 |
11 | Here is where you can register API routes for your application.
12 | These routes are loaded by the RouteServiceProvider and all of them
13 | will be assigned to the "api" middleware group. Make something great
14 |
15 | */
16
17
18 Route::middleware('auth:sanctum')->get('/user', function (Request $request) {
19     return $request->user();
20 });
21
22 Route::resource('projects', ProjectController::class);
23
24
```

8. Actualizar el código del controlador | [Index](#)

```
ProjectController.php X ProjectRequest.php Project.php
app > Http > Controllers > ProjectController.php
1  <?php
2
3  namespace App\Http\Controllers;
4
5  use Illuminate\Http\Request;
6  use App\Http\Requests\ProjectRequest;
7  use App\Http\Resources\ProjectResource;
8  use App\Models\Project;
9
10
11 class ProjectController extends Controller
12 {
13     /**
14      * Display a listing of the resource.
15      */
16     public function index()
17     {
18         //
19         $projects= Project::latest()->paginate(10);
20         return ProjectResource::collection($projects);
21     }
22
23 }
```

Pasos para crear una API REST

8. Actualizar el código del controlador | [Store](#)

```
public function store(Request $request)
{
    //
    $projects=Project::create($request->all());
    return new ProjectResource($projects);
}
```

8. Actualizar el código del controlador | [Update](#)

```
public function update(Request $request, Project $project)
{
    //
    $project->update($request->all());
    return new ProjectResource($project);
}
```

8. Actualizar el código del controlador | [Destroy](#)

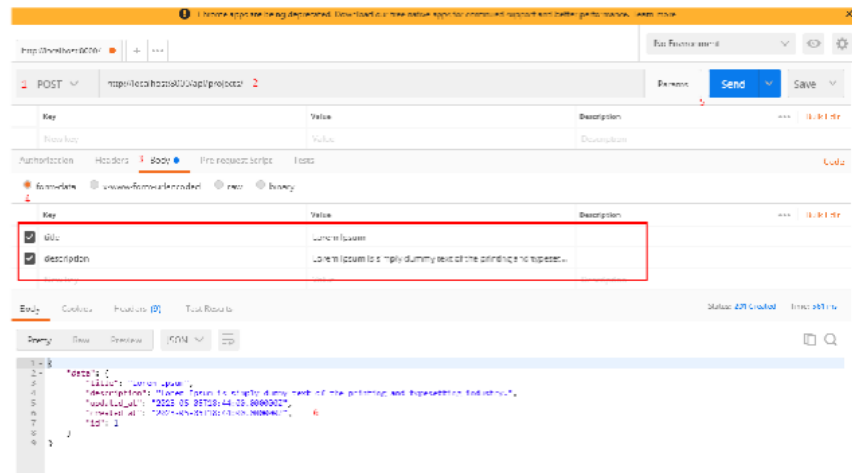
```
public function destroy(Project $project)
{
    //
    $project->delete();
    return response(null, 204);
}
```

9. Ejecutar la aplicación

```
C:\Users\Gisela Espinoza\Documents\crudLaravel\laravelapi>php artisan serve
[INFO] Server running on [http://127.0.0.1:8000].
Press Ctrl+C to stop the server
```

Prueba API REST | POSTMAN | Método POST

1. Seleccionar el método a usar
2. Escribir la URL
3. Seleccionar Body (form-data)
4. Escribir los datos de envío
5. Clic en Send
6. Visualizar resultados



¿Qué es la autenticación de la API?

Una capa de autenticación se utiliza comúnmente en aplicaciones para permitir a los usuarios iniciar y cerrar sesión, proteger la privacidad y limitar los permisos de los usuarios.

La implementación de una capa de autenticación puede ser útil para proteger información confidencial y garantizar que solo los usuarios autorizados tengan acceso a los recursos protegidos.

En este contexto, la "Autenticación de una API basada en tokens" se presenta como un método efectivo de autenticación. Los tokens son una forma de validar la identidad de un usuario y se utilizan para permitir el acceso a los recursos protegidos. Este enfoque tiene varias ventajas, como ser fácil de implementar, proporcionar un buen nivel de seguridad y ser rápido y eficaz. Por estas razones, se considera la forma más utilizada de autenticación en la industria a día de hoy.

Por qué implementar la autenticación basada en tokens

La Autenticación Basada en Token es una forma común de autenticación que funciona mediante el intercambio de tokens entre el cliente y el servidor. Los tokens se utilizan para validar la identidad del usuario y proporcionar acceso a los recursos protegidos. Esta forma de autenticación se considera segura porque los tokens pueden ser cifrados y firmados para evitar la manipulación o la suplantación de identidad. Sin embargo, es importante tener en cuenta que la seguridad de la Autenticación Basada en Token depende en gran medida de la forma en que se implemente. Por ejemplo, es importante utilizar algoritmos de cifrado y firma seguros para los tokens, así como garantizar que los tokens no sean almacenados en lugares vulnerables. Además, es importante considerar otros factores de seguridad, como la gestión de contraseñas y la protección contra ataques.

Token de seguridad

La Autenticación Basada en Token es una forma común de autenticación que funciona mediante el intercambio de tokens entre el cliente y el servidor. Los tokens se utilizan para validar la identidad del usuario y proporcionar acceso a los recursos protegidos. Esta forma de autenticación se considera segura

porque los tokens pueden ser cifrados y firmados para evitar la manipulación o la suplantación de identidad.

Sin embargo, es importante tener en cuenta que la seguridad de la Autenticación Basada en Token depende en gran medida de la forma en que se implemente. Por ejemplo, es importante utilizar algoritmos de cifrado y firma seguros para los tokens, así como garantizar que los tokens no sean almacenados en lugares vulnerables. Además, es importante considerar otros factores de seguridad, como la gestión de contraseñas y la protección contra ataques

Rest API Laravel

Muchos puntos de conexión de la API REST necesitan autenticación o devuelven información adicional si se realiza el proceso de autenticación.

Para autenticar la solicitud, envía un token en el encabezado Authorization de la solicitud. En el ejemplo siguiente, solo basta con reemplazar YOUR_TOKEN por una referencia al token.

```
curl --request GET \  
--url "https://api.pets.com/pets" \  
--header "Authorization: Bearer YOUR-TOKEN" \  

```

Implementación de autenticación

El diagrama anterior muestra de una forma sencilla la implementación de la autenticación en una base de datos y una API. La idea principal es crear una tabla de usuario con el correo electrónico y la contraseña de cada usuario y un endpoint api que genere un token si el email y la contraseña están en la base de datos.

El token generado se utilizará como mecanismo de autenticación en cada uno de los endpoints de la API. Para ello, se debe validar la existencia del token en el header de la petición y su validez.