React+Laravel

Conectar un frontend desarrollado en React con un backend en Laravel es una práctica común en el desarrollo web moderno.

---

1. Estructura del Proyecto

Puedes organizar tu proyecto de dos maneras:

- **Opción 1:** Tener el frontend (React) y el backend (Laravel) en repositorios o carpetas separadas.
- **Opción 2:** Integrar React dentro del proyecto de Laravel (por ejemplo, en la carpeta `resources/js`).

Para este ejemplo, asumiremos que React y Laravel están en proyectos separados.

---

2. Configurar el Backend (Laravel)

a. Habilitar CORS (Cross-Origin Resource Sharing)

Como React y Laravel correrán en servidores o puertos diferentes, necesitas habilitar CORS en Laravel para permitir solicitudes desde el frontend.

1. Instala el paquete `fruitcake/laravel-cors` (si no está incluido en Laravel 11):

```
composer require fruitcake/laravel-cors
```

2. Publica el archivo de configuración de CORS:

```
php artisan vendor:publish --tag="cors"
```

3. Configura CORS en `config/cors.php`. Asegúrate de permitir el origen de tu aplicación React:

```
'paths' => ['api/*'], // Rutas a las que se aplicará CORS
'allowed_origins' => ['http://localhost:3000'], // Origen de tu aplicación
React
'allowed_methods' => ['*'], // Métodos permitidos
'allowed_headers' => ['*'], // Cabeceras permitidas
```

4. Aplica el middleware `\Fruitcake\Cors\HandleCors::class` en `bootstrap/app.php`:

```
$app->middleware([
    \Fruitcake\Cors\HandleCors::class,
    // Otros middlewares globales
]);
```

b. Crear APIs en Laravel

Crea las rutas y controladores necesarios para exponer los endpoints que consumirá React.

1. Define rutas en `routes/api.php`:

```
use App\Http\Controllers\Api\PostController;

Route::get('/posts', [PostController::class, 'index']); // Ejemplo: Obtener
todos los posts
```

```
Route::post('/posts', [PostController::class, 'store']); // Ejemplo: Crear un post
```

2. Crea un controlador para manejar las solicitudes:

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

3. Implementa la lógica en el controlador:

```
namespace App\Http\Controllers\Api;

use App\Http\Controllers\Controller;
use App\Models\Post;
use Illuminate\Http\Request;

class PostController extends Controller
{
    public function index()
    {
        $posts = Post::all();
        return response()->json($posts);
    }

    public function store(Request $request)
    {
        $post = Post::create($request->all());
        return response()->json($post, 201);
    }
}
```

c. Probar las APIs

Usa herramientas como Postman o Insomnia para probar que tus endpoints funcionen correctamente.

---

3. Configurar el Frontend (React)

a. Instalar Axios o Fetch

Para hacer solicitudes HTTP desde React, puedes usar `axios` o `fetch`.

1. Instala Axios:

```
npm install axios
```

2. Crea un archivo de configuración para Axios (opcional):

```
// src/api/axios.js
import axios from 'axios';

const api = axios.create({
    baseURL: 'http://localhost:8000/api', // URL de tu backend Laravel
    headers: {
        'Content-Type': 'application/json',
    },
});

export default api;
```

b. Consumir APIs en React

En tus componentes de React, usa Axios o Fetch para consumir las APIs de Laravel.

Ejemplo con Axios:

```
import React, { useEffect, useState } from 'react';
import axios from 'axios';

const PostList = () => {
    const [posts, setPosts] = useState([]);

    useEffect(() => {
        // Obtener los posts desde Laravel
        axios.get('http://localhost:8000/api/posts')
            .then(response => {
                setPosts(response.data);
            })
            .catch(error => {
                console.error('Error fetching posts:', error);
            });
    }, []);

    return (
        <div>
            <h1>Posts</h1>
            <ul>
                {posts.map(post => (
                    <li key={post.id}>{post.title}</li>
                ))}
            </ul>
        </div>
```

```
    );
};

export default PostList;
```

c. Manejar Envío de Datos

Para enviar datos a Laravel (por ejemplo, un formulario):

```
const createPost = () => {
    const postData = {
        title: 'Nuevo Post',
        content: 'Contenido del post',
    };

    axios.post('http://localhost:8000/api/posts', postData)
        .then(response => {
            console.log('Post creado:', response.data);
        })
        .catch(error => {
            console.error('Error creando el post:', error);
        });
};
```

---

4. Ejecutar los Proyectos

- **Backend (Laravel):** Ejecuta el servidor de Laravel:

  ```
  php artisan serve
  ```

  Esto levantará el servidor en `http://localhost:8000`.

- **Frontend (React):** Ejecuta el servidor de desarrollo de React:

  ```
  npm start
  ```

  Esto levantará el servidor en `http://localhost:3000`.

---

5. Desplegar la Aplicación

Cuando estés listo para desplegar:

- **Backend:** Sube tu proyecto Laravel a un servidor (por ejemplo, usando Laravel Forge, Heroku, o cualquier servicio de hosting).
- **Frontend:** Compila tu aplicación React:

  ```
  npm run build
  ```

  Luego, sube los archivos generados en la carpeta `build` a un servidor web (por ejemplo, Netlify, Vercel, o un servidor estático).

---

6. (Opcional) Integrar React en Laravel

Si prefieres tener React dentro de tu proyecto Laravel:

1. Coloca tu proyecto React en la carpeta `resources/js`.
2. Compila los archivos de React usando Laravel Mix:

   ```
   npm run dev
   ```

3. Incluye el archivo compilado en tus vistas de Blade:

   ```
   <script src="{{ mix('js/app.js') }}"></script>
   ```

Con estos pasos, habrás conectado exitosamente un frontend en React con un backend en Laravel. ¡Buena suerte con tu proyecto! 🚀