

Komunikasi Dengan Backend Menggunakan Axios

I Putu Tatar Nirartha

Axios

Axios adalah sebuah library open source untuk melakukan request HTTP

```
//GET  
axios.get("api/getAllData")
```

```
//POST  
axios.post("api/postData", payload);
```

```
//PUT  
axios.put("api/putData", payload);
```

```
//PATCH  
axios.patch("api/patchData", payload);
```

```
//DELETE  
axios.delete("api/deleteData/" + payload);
```

Import Axios

Setelah menginstall axios kita harus menambahkan modul axios ke dalam file action.js

```
import axios from 'axios';
```

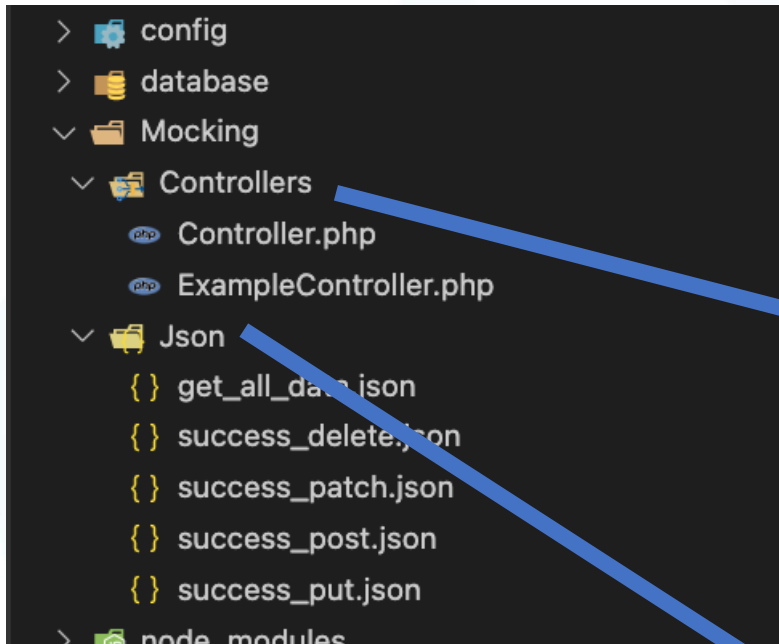
API (Application Programming Interface)

API sendiri merupakan interface yang dapat menghubungkan satu aplikasi dengan aplikasi lainnya.

API bertindak sebagai perantara antara pengguna dan server web.

Cara kerja API bisa di umpamakan seperti rumah makan. Ada tiga aspek penting dalam API yaitu pengguna atau kamu, pelayan (API), dan dapur (server web). Sebagai pelanggan, kamu akan memberi tahu pelayanan tentang makanan pesananmu. Lalu pelayan (API) akan memintanya ke dapur. Setelahnya, pelayan (API) akan mengantarkan makanan sesuai pesananmu.

Membuat Mocking API



Buat folder baru pada root folder dengan nama "Mocking".
Lalu di dalam folder Mocking, buat lagi folder dengan nama "Json" dan "Controllers".

Berisikan controller dari mocking
yang akan kita buat

Berisikan file json untuk response
dari mocking api

Membuat File Controller

Pada folder
Mocking/Controller
kita buat file baru bernama
Controller.php dengan isi
seperti disamping ini

```
<?php

namespace Mocking\Controllers;

use Illuminate\Routing\Controller as BaseController;

class Controller extends BaseController
{
    public function getData(){
        $path = base_path()."/Mocking/Json/get_all_data.json";
        $json = json_decode(file_get_contents($path), true);
        return response()->json($json);
    }
    public function postData(){
        $path = base_path()."/Mocking/Json/success_post.json";
        $json = json_decode(file_get_contents($path), true);
        return response()->json($json);
    }
    public function putData(){
        $path = base_path()."/Mocking/Json/success_put.json";
        $json = json_decode(file_get_contents($path), true);
        return response()->json($json);
    }
    public function patchData(){
        $path = base_path()."/Mocking/Json/success_patch.json";
        $json = json_decode(file_get_contents($path), true);
        return response()->json($json);
    }
    public function deleteData(){
        $path = base_path()."/Mocking/Json/success_delete.json";
        $json = json_decode(file_get_contents($path), true);
        return response()->json($json);
    }
}
```

File controller ini akan
berisikan method-method
yang akan dipanggil ketika
API dipanggil

Membuat File JSON Untuk Response

Pada folder Mocking/Json kita buat file json yang akan digunakan sebagai response dari api

```
// get_all_data.json
{
  "data": [
    {
      "id": 1,
      "name": "Air Mineral"
    },
    {
      "id": 2,
      "name": "Permen"
    },
    {
      "id": 3,
      "name": "Donat"
    }
  ]
}
```

Menambahkan Folder Mocking Pada composer.json

```
19         "laravel/sail" : "1.0.1",
20         "mockery/mockery": "^1.4.4",
21         "nunomaduro/collision": "^5.10",
22         "phpunit/phpunit": "^9.5.10"
23     },
24     "autoload": {
25         "psr-4": {
26             "App\\": "app/",
27             "Database\\Factories\\": "database/factories/",
28             "Database\\Seeders\\": "database/seeders/",
29             "Mocking\\": "Mocking/"
30         }
31     },
32     "autoload-dev": {
33         "psr-4": {
```

Kita harus menambahkan folder Mocking kita pada composer.json, jika tidak maka akan terjadi error 500

Lalu kita jalankan “**composer install**” pada terminal

Menambahkan Mocking API Pada api.php

```
Route::get('/getAllData', 'Mocking\Controllers\Controller@getData');  
Route::post('/postData', 'Mocking\Controllers\Controller@postData');  
Route::put('/putData', 'Mocking\Controllers\Controller@putData');  
Route::patch('/patchData', 'Mocking\Controllers\Controller@patchData');  
Route::delete('/deleteData/{id?}', 'Mocking\Controllers\Controller@deleteData');
```

Kita tambahkan mocking api yang telah dibuat pada file api.php

Menambahkan Button pada Component Untuk Men-trigger Action

```
<button class="mx-1 btn btn-success" @click="getData">GET</button>
```

Button pada component

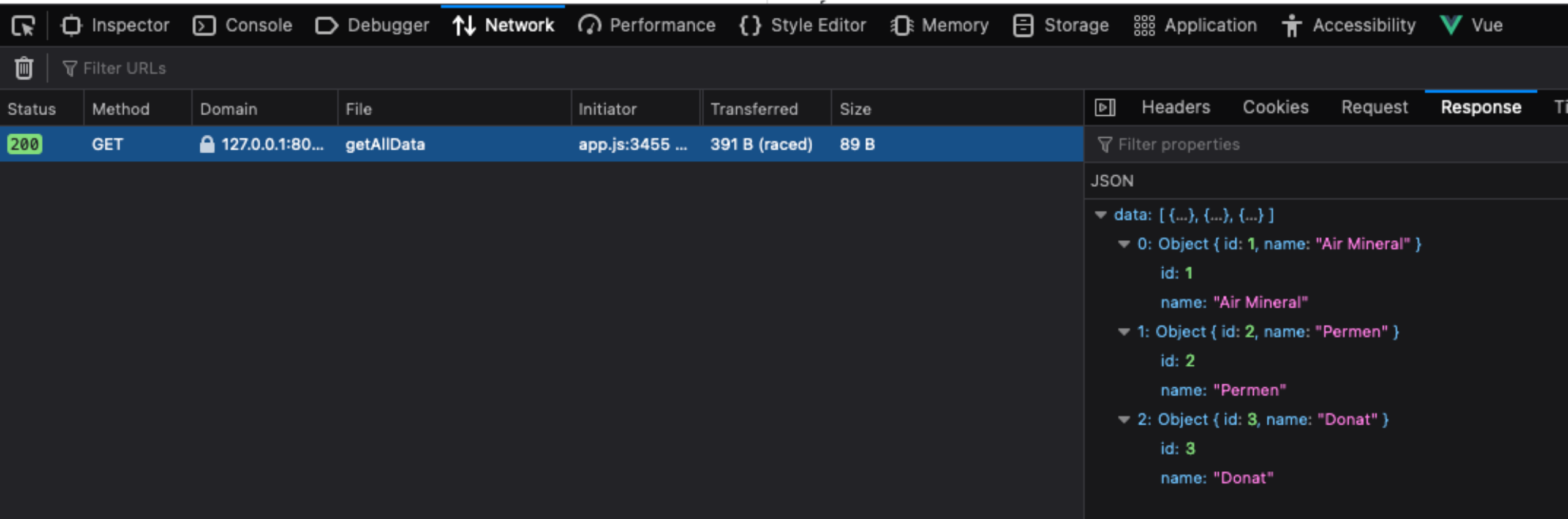
```
getData(){  
  this.$store.dispatch("example/getAllData")  
}
```

Method yang di trigger oleh button ketika di klik

```
const getAllData = async (context) => {  
  let response = await axios.get("api/getAllData");  
  
  context.commit("UPDATE_DATA", response.data);  
}
```

Action yang di dispatch oleh method getData

Hasil Ketika Memanggil API



The screenshot shows the Chrome DevTools Network tab. The top toolbar includes icons for Inspector, Console, Debugger, Network (active), Performance, Style Editor, Memory, Storage, Application, Accessibility, and Vue. Below the toolbar is a filter bar with a trash icon and a 'Filter URLs' input. The main table lists network requests. The selected request is a GET request to 127.0.0.1:80... with the file 'getAllData' and initiator 'app.js:3455 ...'. The transferred size is 391 B (raced) and the size is 89 B. The right sidebar shows the 'Response' tab with a 'Filter properties' input. The response is a JSON array of three objects:

```
JSON
▼ data: [ {...}, {...}, {...} ]
  ▼ 0: Object { id: 1, name: "Air Mineral" }
    id: 1
    name: "Air Mineral"
  ▼ 1: Object { id: 2, name: "Permen" }
    id: 2
    name: "Permen"
  ▼ 2: Object { id: 3, name: "Donat" }
    id: 3
    name: "Donat"
```

Dokumentasi dan Contoh

- Dokumentasi Axios: <https://axios-http.com/docs/intro>
- Contoh: <https://github.com/tatarnirartha/contoh-axios>

Terima **Kasih**