

Pertemuan 11

Konsep REST API dan REST API Client dengan Javascript

Pendahuluan

Terkadang ketika kita membuat frontend(website) menggunakan javascript atau menggunakan framework javascript(React, Vue, Svelte) kita akan membutuhkan suatu cara untuk melakukan komunikasi dari frontend ke backend(server) seperti kode php kita dulu yang melakukan komunikasi dengan database mysql.

Untuk pertemuan kali ini kita akan belajar tentang RESTFUL API, Sebelum bahas RESTFUL API apa sih API, API adalah Application Programming Interface, tujuan dari API dalam konteks ini ialah berbagi data antar aplikasi yang berbeda, nah RESTFUL API ini merupakan suatu desain arsitektur yang terdapat di API itu tadi, RESTFUL API ini sering digunakan ketika frontend(website) memerlukan suatu data yang hanya ada di backend(server).

Cara kerja RESTFUL API ini yaitu REST Client(user) akan melakukan akses pada data pada REST server, data-data yang diakses tadi memiliki identifier yang berbeda berbentuk URI (<https://api.github.com/users/octocat/>) setiap URI memiliki data yang berbeda dengan yang lain.

Konsep ini menggunakan serialize dan deserialize, jadi di server akan men-serialize data, lalu di client nanti akan deserialize data-nya, serialize merupakan suatu konversi/translasi agar dapat di baca oleh bahasa pemrograman lain dan deserialize ini sama dengan serialize tetapi kebalikan-nya.

kita akan membuat website yang menampilkan todo, todo ini berasal dari server orang lain.


URI yang akan kita pakai:

- <https://jsonplaceholder.typicode.com/todos>

Setiap API memiliki schema, setiap API memiliki schema yang berbeda-beda jadi kita harus mempelajari-nya dulu sebelum membuat aplikasi kita.

Ini merupakan schema RESTFUL API yang akan kita pakai, URI(Endpoint) ini return data yang berbentuk JSON, JSON ini hampir mirip dengan Javascript Object, data yang berikan API ini bisa di deserialize menjadi array yang berisi Javascript Object, kalau tidak tau array tolong lihat pada pertemuan 1 atau 9.

<https://jsonplaceholder.typicode.com/todos>



```
1  [  
2      {  
3          "userId": xxx,  
4          "id": xxx,  
5          "title": "xxx",  
6          "complete": true,  
7      },  
8      {  
9          "userId": xxx,  
10         "id": xxx,  
11         "title": "xxx",  
12         "complete": false,  
13     }  
14     ...  
15 ]
```

Kita nanti hanya fokus di id, title, dan complete, kalau temen-temen tidak bisa bahasa inggris id itu unique identifier ini digunakan untuk membedakan dengan data lain, title merupakan judul, dan complete merupakan selesai tetapi dalam konteks ini apakah todo yang dimaksud selesai atau belum.

Untuk mengambil data dari URI tadi kita akan menggunakan function `fetch()` yang disediakan oleh javascript, function ini memiliki return value `Promise`, `Promise` ini termasuk di asynchronous yang berarti non-blocking function tetapi beda dengan multi-threading, multi-threading mengeluarkan thread langsung di process dan terkadang hidup-nya cukup lama kalau asynchronous ini hanya si program menjalankan kode lain dan juga kode yang jalan-nya lama dan kode yang lama tersebut nanti bisa di-ambil ketika kita menggunakan keyword `await`, `await` ini nanti akan men-stop eksekusi kode lain dan menanti kode yang jalan-nya lama itu tadi sampai selesai, tidak semua tipe data bisa di `await` hanya `Promise` saja yang bisa di `await`.

Praktek

Petama-tama kita siapkan html-nya dulu



```
1 <!DOCTYPE html>
2 <html lang="en">
3
4   <head>
5     <meta charset="UTF-8">
6     <meta http-equiv="X-UA-Compatible" content="IE=edge">
7     <meta name="viewport" content="width=device-width, initial-scale=1.0">
8     <title>List todo</title>
9   </head>
10
11   <body>
12     <h1>Todos</h1>
13     <div id='app'>
14       <!-- -->
15     </div>
16   </body>
17
18 </html>
```

Nah setelah selesai kita kasih `<script>` tag di bagian bawah, hal pertama yang akan kita lakukan mengambil html elemen yang memiliki id app.

lalu kita tambah function yang membantu kita ketika ngoding.

```
1  </body>
2    <script>
3      // kita ambil html element yang ber-id app
4      const app = document.getElementById('app');
5
6      // Function utility
7      function checkMark(boolean) {
8        return boolean ? '✓' : '✗';
9      }
10
11     // function ini digunakan untuk append todo ke id app element
12     function appendTodoToApp(todoObject) {
13       const id = todoObject.id;
14       const judul = todoObject.title;
15       const selesai = checkMark(todoObject.complete);
16       const html = `<div id="todo-`${id}`">
17         <h2>`${judul}</h2>
18         <span>Status : `${selesai}</span>
19       </div>`;
20
21       app.innerHTML += html;
22     }
23   </script>
24
25 </html>
```

function checkMark(boolean) itu hanya melakukan konversi terhadap boolean agar menjadi ✓ atau ✗

Dan appendTodoToApp(todoObject) merupakan function yang akan melakukan penambahan html elemen di html elemen yang ber-id app menggunakan data yang diberikan yaitu todoObject, karena kita tidak peduli kita langsung menambahkan html elemen yang baru menggunakan String

Note : tanda += ini di konteks ini merupakan penambahan String, kalau angka nanti ditambahkan langsung

misal kalau string

```
let nama = "Zagar";
```

```
nama += " Dangure";
```

```
console.log(nama); // nanti muncul "Zagar Dangure"
```

Kalau angka / int / double / float

```
let hasil = 4;
```

```
hasil += 2;
```

```
console.log(hasil); // nanti muncul 6
```



```
1 // kita melakukan request ke server
2 // karena function ini mengembalikan/return promise dapat menggunakan
3 // method then dan catch untuk asynchronous programming
4 fetch('https://jsonplaceholder.typicode.com/todos')
5   // hal pertama kita deserialize(kebalikan serialize) dulu
6   .then(res => res.json())
7   // karena method json tadi return promise juga, jadi kita dapat nambah
8   // method then yang berisi penambahan html element ke client
9   // dengan menggunakan data yang sudah di sediakan
10  .then(data => {
11    data.forEach(todo => appendTodoToApp(todo));
12  });
```

Yang terakhir kita tinggal menggunakan function fetch yang sudah disediakan Javascript untuk melakukan request ke <https://jsonplaceholder.typicode.com/todos>, karena ini mengembalikan Promise kita dapat menambahkan method then setelah function fetch, di gambar di-atas ini sering disebut method chaining karena kita tidak men-definisikan variabel kita langsung selipkan pemanggilan method.

Note : Kalau ada masalah bisa di `console.log(nama_variabel)` lalu buka dev tool di browser lalu menuju ke tab console, kalau itu belum bisa membantu menyelesaikan masalah bisa tanya sekalian nama error yang muncul di console.

Note : Konsep seperti ini sering dipakai untuk website yang modern dan interaktif agar tidak ada reload di halaman.