antares;

Fetch API: GET Requests

Dahil sa pagiging popular ng Ajax (dating acronym para sa *asynchronous JavaScript and* XML, pero hindi na ngayon kasi wala nang masyadong gumagamit ng XML sa Web platform, hindi gaya dati), napasama sa ECMAScript 2015 standard ang Fetch API, isang API na nagpapadali sa Ajax. Sa article na ito, tingnan natin kung paano natin magagamit ang Fetch API para magpadala ng `GET` request sa server.

DISCLAIMER: May mahabang intro at history ang article na ito. Puwede ka nang dumiretso sa mismong tutorial

Kung wala kang idea tungkol sa Ajax, isa itong technique sa Web development. So traditionally bago nauso ang Ajax, kapag kailangan mong ma-display sa UI ang mga update sa data, or gusto mong magpasa ng data sa server, kailangang i-reload ang buong page. Pero nang mauso ang Ajax, lahat ng pagkuha at pagpapadala ng data sa server ay nangyayari na sa background; hindi na kailangang i-reload iyong buong page. Dito na nauso iyong mga umiikot na loader GIF para ipakita sa user na pina-process pa ang action nila.

So for years, ginagawa ito ng mga Web developers gamit ang `XMLHttpRequest`. Noon, sobrang makabago itong technique na ito. Pero sobrang komplikado rin nito. Napakaraming kailangang i-setup na **boilerplate code**.

Boilerplate Code

> Isang set ng code na kailangan mong isama sa program para i-setup ang isang feature or technique. Kadalasan nang kina-copy-paste ito dahil bukod sa pagse-setup ng mga kailangan mo, wala na itong ibang ginagawa. May mga pagkakataon na maiisip mong

kalat din ito sa code, pero wala kang magagawa dahil kailangan mong gamitin.

Then, dumating ang *jQuery* at pinadali ang buhay ng lahat. Bukod sa kapangyarihan nitong gawing consistent sa lahat ng browsers ang mga features ng Web platform, may feature din ang jQuery para sa Ajax. Gamit ang function nitong `ajax()`, hindi na kailangan ng mga Web developer na paulit-ulit na pakisamahan ang boilerplate code na minsan nilang sinulat.

Pero tapos na ang panahon ng inconsistencies sa Web platform. Namamayagpag na ngayon ang Web standards sa tulong ng W3C at WHATWG. Sa halos lahat ng pagkakataon, consistent na ang features ng Web sa lahat ng major browsers (i.e, Chrome, Firefox, Safari, Edge, UC Browser, etc.). At unti-unti na ring bumababa ang bilang ng mga gumagamit ng jQuery. Nang ilabas ang ECMAScript 2015 standard (bagong version ng JavaScript), isinama nito ang mga features na galing sa jQuery, kasama na ang `ajax()` function, na ngayon ay buhay sa katauhan ng Fetch API.

HTTP Requests

Kapag nagse-send tayo ng requests sa server, meron tayong tinatawag na **HTTP Verbs**. Basically, sinasabi nito kung ano ang gusto nating gawin ng server; verbs, ibig sabihin action words sila, gusto natin na gawin ng server iyong action na pinadala natin sa request. So may pitong HTTP verbs:

- `GET` - "Pakikuha itong specific na data"
- `POST` - "Pakilagay itong data na ito sa database"
- `PUT` - "Paki-update nitong data na ito; kung wala pang ganito sa database, gumawa ka ng bago"
- `PATCH` - "Paki-update nitong data na ito, pero itong specific part lang na ito. 'Wag mong baguhin lahat"
- `DELETE` - "Pakitanggal na nitong data na ito"
- `HEAD` - Kagaya ito ng `GET` requests, pero hindi mo makukuha iyong data. Instead, makukuha mo lang ay iyong HTTP headers, i.e., iyong data na sine-send ng server pabalik para malaman mo kung anong nangyari sa server habang ine-execute n'ya iyong action na ni-send mo. Dito nakikita sa HTTP headers kung anong code ang binalik ng server (200 OK, 404 Not Found, etc.) kasama na ang iba pang data na useful for debugging.
- `OPTIONS` - Para naman itong `--help` sa mga program. Kapag nag-send ka ng `OPTIONS` request sa server, ibabalik nito sa iyo ang mga available na actions sa `GET`, `POST`, `PUT`, `PATCH`, `DELETE`, at `HEAD`. Kaya lang, bihira lang daw ang mga nag-i-implement ng ganitong action sa mga API nila.[1]

Ideally, magagamit natin lahat ito, kasi gano'n naman talaga dapat. Kaya lang, malupit ang mundo, at sa totoong buhay, `GET` at `POST` lang ang madalas na ginagamit. May iba na gumagamit din ng `PUT` at `DELETE`. Pero wala pa akong nakitang gumamit ng `PATCH`, `HEAD`, at `OPTIONS`. Nagulat pa nga ako no'ng nalaman kong may ganiyan pa pala; ang akala ko apat lang.

Sa susunod na tutorial (yes, nasa intro pa lang po tayo), tingnan natin kung paano natin gagamitin ang `GET` requests para kumuha ng data sa server.

Fetch API Tutorial

Sa tutorial na ito, gagawa tayo ng simpleng program: kapag na-click iyong button, magse-send tayo ng request sa server para kumuha ng data, then idi-display natin iyon sa isang HTML table.

Data Format

Para sa tutorial na ito, gagamitin natin ang JSON Placeholder API (https://jsonplaceholder.typicode.com) para hindi na tayo magse-setup ng sarili nating server. Kaya siguraduhing may Internet connection kapag ita-try na itong tutorial na ito.

Kapag nag-send tayo ng `GET` request sa JSON Placeholder API, ganito ang format ng data na *gusto* nating makuha natin pabalik:

Expected Data

```
[
    {
        "first_name": "Annalise",
        "last_name": "Keating",
        "email": "ak@murd.er"
    }
]
```

Marami pang ibang sample data na binibigay ang JSON Placeholder API, pero iyan lang ang kailangan natin.

User Interface

So kailangan natin ng user interface. Simple lang ang gagawin natin; isang button lang saka table.

‘Wag na muna po tayong mag-design ngayon, pakiusap lang. Hindi tayo matatapos.

index.html

```
<button onclick="sendGetRequest()">Send GET Request</button>

<table>
    <thead>
        <tr>
            <th>First Name</th>
            <th>Last Name</th>
            <th>E-Mail</th>
        </tr>
    </thead>
    <tbody id="table-body"></tbody>
</table>
```

sa `<tbody>` natin ilalagay iyong data mamaya. Pansinin din iyong `onclick` attribute sa button. Mamaya, ide-declare natin iyang function na iyan.

Ang `sendGetRequest()` Function

I-declare na natin ang `sendGetRequest` function sa JavaScript. Sa loob ng function na ito, dito natin ilalagay ang `fetch()`:

index.js

```
const serverURL = 'http://jsonplaceholder.typicode.com/users';
function sendGetRequest() {
    const request = fetch(serverURL);
}
```

Suggested:
Function Parameters at Return Statements

Sa example na ito, tinawag natin ang `fetch()` function at binigyan ito ng URL para sa server kung saan siya magse-send ng request. Kapag tinawag natin ang `fetch` function, magre-return ito ng isang `Promise` object.

Sa JavaScript, ang mga `Promise` object ang ginagamit para mag-execute ng code sa “background”.

> Technically, hindi talaga nag-e-execute sa background ang mga `Promise` kasi hindi gano'n gumagana ang JavaScript kasi single-threaded ito, hindi gaya ng ibang languages. Pero para sa example natin, sabihin na nating nag-e-execute ito sa "background".

JavaScript `Promise`

May mga methods ang `Promise` objects na magagamit natin para makapagdagdag tayo ng actions sa mga operation na nagra-run sa "background".

`Promise`

> The Promise object represents the *eventual* completion (or failure) of an asynchronous operation, and its resulting value.*Mozilla Developer Docs*

- `then()` - isa itong method na gagawa ng action kapag natapos na ang operation na nagra-run sa "background".
- `catch()` - isa itong method na gagawa ng action kapag natapos na ang operation sa "background" dahil **nagkaroon ng error.**
- `finally()` - isa itong method na mag-e-execute kapag natapos na ang lahat ng operations ng `Promise` object, kasama na ang `then()` at `catch()`.

Gusto ko ring banggitin sa inyo saglit na itong mga method na ito ay mga example ng tinatawag nating *higher-order functions*. Sa normal na functions, nagpapasa tayo ng parameters na either number, string, object, or array. Pero sa mga higher-order functions, sa halip na normal na data structures ang tanggapin nila bilang parameters, tumatanggap sila ng functions. Ibig sabihin puwede tayong magpasa ng functions sa mga functions. *mindf$ck™*

Sa mga methods ng `Promise`, kapag nagpasa tayo ng mga function, ie-execute nila ang mga function

na iyon sa mga specific na phase ng promise execution. Kapag nagpasa tayo ng function sa `then()`, ie-execute nito ang pinasa nating function pagkatapos na ma-execute ang promise. Kapag nagpasa naman tayo ng function sa `catch()`, ie-execute nito ang function natin kapag nagkaroon ng error. At kapag nagpasa naman tayo ng function sa `finally()`, ie-execute nito ang function natin kapag natapos na ang operation sa `Promise` object, ang `then()` function, at ang `catch()` function.

Pagre-retrieve ng data

Matatandaang naka-store sa variable na `request` ang request natin sa server:

index.js

```
1 const serverURL = 'http://jsonplaceholder.typicode.com/users';
2 function sendGetRequest() {
3     const request = fetch(serverURL);
4 }
```

Kapag tinawag natin ang `fetch()`, agad-agad itong magpapadala ng request sa server URL, at ang ire-return nito ay isang `Promise` na magiging pangakong panghahawakan natin, isang pangako na matatapos ang operation sa takdang panahon. Pero dahil hindi natin alam kung kailan ito matatapos, kailangan nating pagbilinan ang `request` variable kung ano ang gagawin niya kapag natanggap na natin ang response ng server.

Dahil lahat ng natatanggap natin galing sa server ay naka-string, at JSON ang kailangan natin, kailangan nating i-parse ang string papuntang JSON. Para magawa ito, kailangan nating hingin ang tulong ng `then()` at magpasa rito ng function na magpa-parse ng string:

index.js

```
1 const serverURL = 'http://jsonplaceholder.typicode.com/users';
2 function sendGetRequest() {
3     const request = fetch(serverURL);
4
5     const jsonPromise = request.then(function(response) {
6         return response.json();
7     })
8 }
```

Isa-isahin natin kung ano ang nangyayari dito:

- Para makuha ang response ng server, nagpasa tayo ng function sa `then()`. Kapag in-execute ng `then()` ang function natin, ibibigay nito ang response ng server sa function natin, at iso-store ito sa `response` argument.
- Ang `response` variable ay isang object sa JavaScript na may mga methods na kasama. Built-in dito ang `json()` function na magko-convert ng response ng server into JSON object.
- Pagkatapos na ma-execute ang function na pinasa natin, magre-return ang `request.then()` ng isa pang Promise object, kaya nilagay natin ito sa `jsonPromise`. Nasa loob ng `jsonPromise` ang JSON object na nakuha natin galing sa server.

Medyo may pagka-komplikado ang mga Promise sa JavaScript. Pero isang technique na ginamit ko dati, hanggang ngayon, para hindi ako malito ay ganito: iniisip ko na parang time capsule ang mga Promise object. Kung may result tayo na gustong makuha sa mga operation sa background, ang makukuha lang natin ay Promise objects, at iyong result na gusto nating makuha ay nasa loob noon. Ang tanging paraan para makuha kung anuman ang nasa loob ng time capsule na iyon ay kung gagamitin natin ang `then()` function; iyon lang iyong tanging opening na mayro'n ang mga Promise object.

Kaya para ma-display natin iyong data, kailangan muna nating makuha iyon sa loob ng `jsonPromise`. At paano natin gagawin iyon? Tatawagin natin ulit ang `then()` function.

index.js

```
1 jsonPromise.then(function(jsonObject){
2     console.log(jsonObject);
3 })
```

Matatandaan na nasa loob ito ng `sendGetRequest()` function na mag-e-execute kapag napindot ang button. Kapag na-click iyong button na iyon, makikita mo sa console (Chrome DevTools or F12 Developer Tools sa Firefox) ang JSON object na natanggap natin mula sa server.

Ang kailangan lang natin ay tatlong pieces of information: first name, last name, at email address. Gagawa tayo ngayon ng function na magko-convert ng JSON object na iyan into HTML table rows. Generally, ganito ang format na susundin natin per row:

index.html

```
1 <tr>
```

```
    <td><!--First Name--></td>
    <td><!--Last Name--></td>
    <td><!--Email--></td>
</tr>
```

Kaya gagawa tayo ng function na gagawa nito. Tawagin natin itong `jsonToHtmlString()`:

index.js

```
function jsonToHtmlString(jsonObject) {
    return `
    <tr>
        <td>${jsonObject.first_name}</td>
        <td>${jsonObject.last_name}</td>
        <td>${jsonObject.email}</td>
    </tr>
    `
}
```

Sa function na ito, gumamit din tayo ng tinatawag nating *template strings*. Kasama ito sa mga bagong features ng JavaScript sa ES2015. Sa halip na gumawa tayo ng maraming string concatenation (`"string1 " + variable + " string2"`), puwede nating gamitin ang template string para gawin ang tinatawag nating **string interpolation**.

String Interpolation

> In computer programming, string interpolation (or variable interpolation, variable substitution, or variable expansion) is the process of evaluating a string literal containing one or more placeholders, yielding a result in which the placeholders are replaced with their corresponding values... String interpolation allows easier and more intuitive string formatting and content-specification compared with string concatenation. *Wikipedia (https://en.m.wikipedia.org/wiki/String_interpolation)*

Basically, sa string interpolation, kukunin ng compiler or interpreter lahat ng variable na nasa string at isa-substitute ang value ng variable doon.

Sa halip na double quotes ("") or single quotes (''), sa template literals, gumagamit tayo ng backticks (`) or iyong nasa left side ng `1` sa keyboard. Lahat naman ng nasa loob ng `${}` ay mai-interpret bilang JavaScript code.

Ngayong nagawa na natin ang `jsonToHtml()` function, puwede na natin itong magamit sa loob ng `sendGetRequest()`.

Pagdi-display ng Data

Sa ngayon, ito ang laman ng `sendGetRequest()` function:

index.js

```
1 const serverURL = 'http://jsonplaceholder.typicode.com/users';
2 function sendGetRequest() {
3     const request = fetch(serverURL);
4
5     const jsonPromise = request.then(function(response) {
6         return response.json();
7     })
8 }
```

Tatawagin natin nang isang beses pa ang `then()` method sa `jsonPromise` para makuha ang mismong JSON object at mai-display ito.

index.js

```
1 jsonPromise.then(function(jsonObject){
2     const people = jsonObject;
3 });
```

Kung matatandaan ninyo, array ang laman ng JSON object; array ito ng mga object na may first name, last name, at email. So ang kailangan natin ay i-transform ang bawat item sa array na ito into HTML string. Kaya gagawin natin ito gamit ang ni-declare natin kaninang `jsonToHtmlString()`.

index.js

```
jsonPromise.then(function(jsonObject){
    const people = jsonObject;

    const htmlStrings = jsonObject.map(person => jsonToHtmlString(person))
});
```

Ang `map()` method ay isang higher-order function na makikita sa mga array sa JavaScript. Bale ang gagawin nito, iisa-isahin niya iyong mga items ng array, pagkatapos papalitan niya iyong value. Sa case na 'to, nagpasa tayo ng function sa `map()`; kung anuman ang i-return ng function na pinasa natin, iyon ang magiging value ng item natin.

> Ang `map()` method ay *non-mutative*; ibig sabihin, hindi niya binabago ang laman ng variable. Gumagawa siya ng bagong object. Kaya naman, kahit ilang beses mong tawagin ang `map()`, hindi magbabago ang laman ng `jsonObject`. Mai-store lang ang result ng `map()` sa `htmlStrings`.

I-try nating i-output sa console ang value ng `htmlStrings`:

Console Output

```
[
    "<tr>\
        <td>Annalise</td>\
        <td>Keating</td>\
        <td>ak@murd.er</td>\
    </tr>",
    "<tr>\
        <td>Wesley</td>\
        <td>Gibbins</td>\
        <td>wes@murd.er</td>\
    </tr>",
    ...
]
```

Dahil kino-convert ng `jsonToHtmlString()` ang JSON objects into HTML, makikita natin na nagkaroon na tayo ng array ng strings galing sa array ng JSON objects. Puwede na natin itong gamitin para i-output sa front end ang data:

index.js

```
document.getElementById("table-body")
    .innerHTML = htmlStrings.join('');
```

Sa code na ito, pagdudugtung-dugtungin lang natin lahat ng items ng `htmlStrings` gamit ang `join()` function. Pagkatapos, ilalagay natin ito sa table gamit ang `innerHTML` property. Once na malagyan ng value ang `innerHTML`, mag-a-appear na sa screen ang output natin.

Footnotes

1. ^ Galing ito sa *Zalando RESTful API and Event Scheme Guidelines*.

Tingnan ang pinakabagong version ng artikulong ito sa
https://celestialcinnamon.github.io/antares-blog/tl/Fetch-API-GET-Requests/

See me outside

- Antares on Facebook (https://facebook.com/antaresprogramming)
- Antares on Github (https://github.com/celestialcinnamon/antares-blog)

- See me on Facebook (https://facebook.com/dorkas.rubio)
- See me on Twitter (please don't)
- See me on Instagram (https://instagram.com/melancholicapoptosis)
- See my portfolio (https://celestialcinnamon.github.io)
- Send me an email: francoisoibur21@gmail.com

Ang Antares Programming

Ang Antares Programming ay isang blog para sa mga Pilipino tungkol sa mga bagay tungkol sa Web at software development na hindi madalas maituro sa mga university at college. Dahil kinikilala ng Antares Programming ang epekto ng wikang kinalakhan o *mother tongue* sa pagkatuto, karamihan ng mga artikulo sa site na ito ay nasa Filipino. Umaasa ang writer nito na darating ang panahon na magkakaroon ng mas maraming materyal sa iba pang mga wika ng Filipinas. Pero sa ngayon, sapat na ang pagsisikap na ito.

Nga pala, hindi laging ganito "kalalim" (kapormal) ang Filipino sa site na ito. ☺