Fetch API: POST Requests

Sa pangalawang bahagi ng Fetch API series, talakayin naman natin ang pagse-send ng `POST` requests sa server gamit ang Fetch API.

Ang post na ito ay pangalawang bahagi ng Fetch API series. Para sa unang bahagi, tinalakay natin ang `GET` requests.

Kung matatandaan n'yo, sa nakaraang post about sa `GET` requests, nakita natin kung paano kukuha ng data sa server. Pero paano kung gusto nating mag-send ng data sa server? May mga pagkakataon, gaya ng user registration, na kailangan nating mag-send ng data sa server. Trabaho naman iyan ng `POST` requests.

Setup

Sa example na ito, may example tayong server sa endpoint na ito:

```
https://www.example.com
/api/user/
```

Kadalasan sa mga REST API, iisa lang ang endpoint para sa isang collection ng data. Halimbawa, para sa isang table sa database, `tbl_users`, may isang endpoint na nakalaan, `/api/user/`. Lahat ng actions sa `tbl_users` ay mangyayari lang sa `/api/users/` na URL. Kung gusto mong kumuha ng data sa `tbl_users`, magse-send ka ng `GET` request sa `/api/users/`. Kung gusto mo namang mag-send ng data, `POST` request ang ise-send mo. Nag-iiba ang function ng bawat endpoint depende sa type ng request mo, depende sa HTTP verb na ginamit mo sa request.

REST API

> Representational State Transfer. Isa itong service na nagpo-provide ng access sa isang dataset o database gamit ang HTTP.

Sabihin nating iyong REST API na ginagamit natin, `www.example.com/api/`, ay tumatanggap ng `POST` requests. Ganito ang tinatanggap niyang format ng data:

```
{
    "name" : {
        "first": "First name
ng user. Required.",
        "middle": "Middle
name ng user. Optional.",
        "last": "Surname ng
user. Required."
    },
    "email": "E-mail address
ng user. Required."
}
```

Sa data na ise-send natin sa server, kailangan na sumusunod sa format na iyan ang data natin.

After natin mag-send ng data, magse-send sa atin pabalik ang server ng response. Sa response na ito, makikita natin kung successful ba o hindi ang data natin. Sa example API natin, kapag successful ang request natin, magse-send sa atin ang server ng response na `200 OK`. Kung hindi naman, magse-send ito ng response na may error code na alinman sa mga ito:

- `400 Bad Request` Kapag ganito, ibig sabihin nagkaroon ng error na hindi alam kung saan galing.
- `401 Unauhorized` Ibig sabihin, kailangan munang mag-log in bago makapag-send ng request.
- `403 Forbidden` Ibig sabihin, hindi ka puwedeng mag-send ng request.
- `408 Request timeout` Masyadong matagal ang inabot ng request, either mabagal ang Internet connection, o may problema sa data na sinend mo dahil masyadong malaki
- `409 Conflict` May isa pang client na nag-send ng data na kaparehas ng sa iyo.
- `429 Too many requests` Sobra na sa limit ang request mo. Nangyayari ito sa mga free trial ng API na may bayad, o kaya kapag sobrang dami mong sinend na request sa isang period of time. Karamihan sa mga API ay may requests per hour na limit.

Sa article na ito, hindi natin iha-handle ang errors na iyan, idi-display lang natin sa `.status`. Pero in real life, kailangan nating

i-handle ang mga errors na iyan.

User Interface

So gusto nating makapag-input ang user ng information niya. Tandaan, kailangan natin ng apat na data: first name, middle name, surname, at email address. Kaya gagawa tayo ng HTML form na magsa-submit ng data sa server.

index.html

```
<form method="POST"
action="/actions
/submit_data"
id="form_ajax">
    <label
for="first_name">First
Name</label>
    <input type="text"
id="first_name"
name="first_name" required
/>

    <label
for="middle_name">Middle
Name</label>
    <input type="text"
id="middle_name"
name="middle_name" required
/>

    <label
for="last_name">Last
Name</label>
    <input type="text"
id="last_name"
name="last_name" required />

    <label
for="email_address">Email
Address</label>
    <input type="email"
id="email_address"
name="email_address"
required />

    <button
type="submit">Submit
Data</button>
</form>

<p class="status"></p>
```

May form tayo rito na may ID na `form_ajax`.

Nilagyan natin ng ID para magkaroon tayo ng reference sa JavaScript mamaya. Pansinin ang mga attributes ng form. Ang attribute na `method="POST"` ang magsasabi na POST request ang ise-send natin, at ang `action` attribute naman ang magsasabi kung saan natin ipapadala ang data na makukuha sa form. Pansinin na magkaiba ang nakalagay na URL sa `action` at ang API endpoint na gagamitin natin (`http://www.example.com/api/user/`).

Actually, itong mga attribute na ito, ang `method` at `action` ay ginagamit natin bilang fallback features. Noong wala pang JavaScript, ito talaga ang ginagamit, pero nire-reload kasi nito ang buong page, na ayaw natin kaya tayo gumagamit ng Ajax. Pero kailangan pa rin nating lagyan ng ganitong features ang form natin para sa kapakanan ng mga users na naka-disable ang JavaScript dahil sa kung anumang dahilan. Mahalaga rin ito kasi minsan nagkaka-error sa script natin. May mga times na kapag nagka-error, hindi na gumagana lahat, kaya mahalaga na may fallback features sa form para masigurong mase-send pa rin ang data ng user kahit may error sa scripts.

Ang URL na nakalagay sa `action` attribute ay isang page. Itong page na ito ang magse-send ng data sa `www.example.com` just in case hindi gumana iyong ajax. Kaya ikaw ang gagawa ng `/actions/submit_data` page. Hindi natin iyan tatalakayin dito, sorry. Pero kung marunong kang gumawa ng back-end features, madali lang para sa iyo na i-figure out kung paano iyan gagawin. After all, hindi mo naman aaralin ang ajax kung wala kang kahit kaunting alam sa back end.

May paragraph din tayo, `.status`. Dito natin idi-display ang magiging status ng request, kasama na ang errors at success notifications.

Retrieval ng Data

Sa user interface, may button tayo na magsa-submit ng form kapag na-click ito. Kapag na-submit ang form, magre-reload ang buong page, kaya dapat nating pigilan iyon. Mag-a-attach tayo ng event handler sa form para sa `submit` event:

index.js

```
const form =
document.getElementById('form_ajax

form.addEventListener('submit',
function(event) {
    event.preventDefault();
})
```

Ang `event.preventDefault()` ang pipigil sa page na mag-reload. Kaya hindi magse-send ang data. Pero pansinin na kapag naka-disable

ang JavaScript, o may kung anumang mali sa scripts natin, hindi mag-e-execute ang part na ito kaya magre-reload ang page at magse-send pa rin ang data.

Next, kailangan nating makuha ang data na ise-send natin. Kung babalikan natin ang `index.html`, mapapansin mong lahat ng `<input />` tags ay may `name` attribute. Ito ang gagamitin natin para makuha ang data.

index.js

```
const form =
document.getElementById('form_ajax');

form.addEventListener('submit',
function(event) {
    event.preventDefault();

    const first_name =
form['first_name'].value;
    const middle_name =
form['middle_name'].value;
    const last_name =
form['last_name'].value;
    const email_address =
form['email_address'].value;
})
```

Kapag ginagamit natin ang `form`, puwede itong maging array ng mga `<input />` elements. Kapag ginamit natin ang `form['first_name']`, makukuha natin ang textbox na may hawak ng data, hindi ang mismong data. Kaya kailangan nating kunin ang `value` property ng textbox para makuha iyong mismong input ng user.

Format ng Data

Matatandaan na may kailangan tayong sunding format:

Format ng Data

```
{
    "name" : {
        "first": "First name
ng user. Required.",
        "middle": "Middle
name ng user. Optional.",
        "last": "Surname ng
user. Required."
    },
    "email": "E-mail address
ng user. Required."
}
```

Kailangan nating i-format sa ganiyang paraan ang data. Hindi naman ito mahirap.

index.js

```
form.addEventListener('submit',
function(event) {
    ...

    const formattedData = {
        name: {
            first: first_name,
            middle: middle_name,
            last: last_name
        },
        email: email_address
    }
})
```

Pagse-send ng Request

Ngayon kailangan na nating i-send ang data. Tandaan, ang endpoint natin ay `https://www.example.com/api/users/`.

index.js

```
const endpoint =
'https://www.example.com
/api/users/';

fetch(endpoint, {
    method: "POST",
    body:
JSON.stringify(formattedData)
})
```

Sa example na ito, nadagdagan ng arguments ang `fetch()` function. Sa nakaraang example natin, nagpapasa lang tayo ng URL sa `fetch()` function. Pero dahil gusto nating mag-send ng data, magpapasa tayo ng isa pang parameter: ang `options` parameter. Maraming laman ang `options` parameter, pero dalawa lang muna ang gagamitin natin. Kung hindi natin ise-specify ang values ng ibang mga properties ng `options` parameter, may defaults naman.

Sa example na ito, ginamit natin ang dalawang properties na ito:

- `method` Ito ang HTTP verb na gagamitin natin. Sa kasong ito, dahil gusto nating mag-send ng data, `POST` ang ginamit nating verb.
- `body` Ito ang data na ise-send natin sa server.

May isang limitation sa `fetch()` function. Kinakailangang `FormData` object ang ipasa natin sa `body` property. Pero naka-JSON ang data natin, kaya paano iyon? Bukod sa `FormData`, tumatanggap din ng JSON ang `body`, pero kailangang naka-string ito, kaya naman natin tinawag ang `JSON.stringify()` method. Gagawin nitong string ang data na nasa `formattedData`.

Challenge

Itutuloy ko pa sana ang tutorial na ito hanggang sa dulo, pero naisip ko kaparehas na lang naman iyon ng sa unang part ng tutorial na ito. Kaya kunin natin itong magandang chance para makapag-practice na rin. Nagawa na natin ang unang part nito, ang pagse-send ng data. Bakit hindi mo subukang ituloy ang natitirang parts ng tutorial na ito? Ito lang ang kailangan mong gawin:

- Gamit ang `then()` method sa `Promise` object na nire-return ng `fetch()`, i-display ang success status ng ajax sa `<p class="status"></p>`
- Gamit ang `catch()` method sa `Promise` object, i-display ang error ng ajax sa `<p`

```
class="status"></p>
```

Ano ang magiging premyo kapag nagawa mo? Validation at self-satisfaction. Salamat sa pagbabasa ng article na ito at sa pagsuporta sa Antares Programming. Magkita ulit tayo sa susunod na artikulo.

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