ASINHRONA RAZMENA PODATAKA SA SERVEROM

Novi Sad, 2021

Miroslav Zarić

TOKOM VREMENA RAZLIČITI PRISTUPI

- AJAX
  - XMLHttpRequest API
    - eventualno uz jQuery
- Fetch API
- Axios

AJAX

- AJAX - Asynchronous JavaScript And XML
  - Termin promovisan 2005. godine, suštinski opisuje sposobnost da se pomoću JavaScripta izvrši upit (request) ka serveru, sačeka odgovor i osveži stanje interfejsa (web stranice) na osnovu rezultata upita
  - Iako se u samom nazivu koristi XML, suštinski nikada nije bio ograničen samo na upotrebu XML formata
  - Zasniva se na korišćenju XMLHttpRequest API-ja

XMLHttpRequest API

- **XMLHttpRequest (XHR) -** objekti koji se koriste za obavljvanje komunikacije sa serverima.
  - Ovo omogućava da se komunikacija obavi u pozadini (asinhrono) bez potrebe da se iznova iscrtava cela stranica.
- Na ovaj način osvežava se samo deo stranice za koji je došlo do promene podataka.
- Ovaj objekat se može koristiti za razmenu podataka u bilo kom formatu (ne samo u XML-u kako to ime sugeriše)

Upotreba XMLHttpRequest-a

1. Kreirati XMLHttpRequest objekat
2. Dodati odgovarajuće event-listenere (minimalno one koji treba da reaguju na odgovor servera - dakle *reqListener* bi bila vaša funkcija)
3. Otvoriti konekciju
4. Poslati zahtev

```
var oReq = new XMLHttpRequest();
oReq.addEventListener("load", reqListener);
oReq.open("GET", "http://www.example.org/
example.txt");
oReq.send();
```

Upotreba XMLHttpRequest-a

- Zahtev može da se pošalje asinhrono (što je i podrazumevano stanje) ili sinhrono (ovo drugo treba izbegavati, jer većina browsera je ovu funkcionalnost proglasila za *deprecated*)
- Tip zahteva se postavljao opcionim trećim parametrom u pozivu *open* metode

Događaji XMLHttpRequest-a

- ***abort***
  - okida se kada se zahtev odbaci, npr, kada se programski pozove metoda XMLHttpRequest.abort().
  - odgovarajući event handler se postavlja preko ***onabort*** property-ja.
- ***error***
  - okida se kada se pojavi greška u izvršavanju upita.
  - odgovarajući event handler se postavlja preko ***onerror*** property-ja.
- ***load***
  - okida se kada se XMLHttpRequest komunikacija sa serverom uspešno obavi.
  - odgovarajući event handler se postavlja preko ***onload*** property-ja.
- ***loadend***
  - okida se kada se zahtev završi, bilo uspešno ili neuspešno.
  - odgovarajući event handler se postavlja preko ***onloadend*** property-ja.
- ***loadstart***
  - okida se kada započne prijem podataka.
  - odgovarajući event handler se postavlja preko ***onloadstart*** property-ja.
- ***progress***
  - okida se periodično kada pristignu dodatni podaci.
  - odgovarajući event handler se postavlja preko ***onprogress*** property-ja.
- ***timeout***
  - okida se kada istekne unapred postavljeno vreme za završetak komunikacije sa serverom.
  - odgovarajući event handler se postavlja preko ***ontimeout*** property-ja.

Metode XMLHttpRequest-a

- ***XMLHttpRequest.abort***()
  - Prekida zahtev koji je već poslan.
- ***XMLHttpRequest.getAllResponseHeaders()***
  - Vraća sva zaglavlja iz primljenog odgovora, odvojena novim redom, null ukoliko odgovor nije primljen.
- ***XMLHttpRequest.getResponseHeader(****headerName****)***
  - Vraća traženo zaglavlje iz responsa ili null ukoliko odgovor još nije primljen ili traženi parametar u zaglavlju ni ne postoji.
- ***XMLHttpRequest.open()***
  - Inicijalizuje izvršavanje zahteva.
- ***XMLHttpRequest.overrideMimeType()***
  - omogućava da se prinudno postavi tip odgovora, bez obzira šta je server naveo za tip u odgovoru.
- ***XMLHttpRequest.send()***
  - Šalje pripremljeni zahtev.
    BITNO: Kada je zahtev asinhron (a to je podrazumevano stanje) - ova funkcija se odmah i završava - pre slanja se moraju postaviti callback funckije koje će odraditi odgovarajući event handling kada odgovor stigne.
- ***XMLHttpRequest.setRequestHeader()***
  - Omogućava postavljanje odgovarajući polja u zaglavlje zahteva.
    BITNO: Mora se pozvati NAKON metode open(), ali obavezno pre send().

Property-ji XMLHttpRequest-a

- ***XMLHttpRequest.onreadystatechange***
  - sadrži naziv event handler funkcije koja će biti pozvana kad god se detektuje promena readyState atributa.

- ***XMLHttpRequest.readyState*** **Read only**
  - Numerička vrednost koja opisuje trenutno stanje zahteva
  - **0 UNSENT**
  - **1 OPENED**
  - **2 HEADERS_RECEIVED**
  - **3 LOADING**
  - **4 DONE**

- ***XMLHttpRequest.response*** **Read only**
  - Vraća sadržaj odgovora - tip zavisi od toga šta je postavljeno u *XMLHttpRequest.responseType*

- ***XMLHttpRequest.responseText*** **Read only**
  - Vraća DOM sstring koji sadrži odgovor u obliku teksta (null ako zahtev nije uspeo, ili još nije izvršen send())

- ***XMLHttpRequest.responseType***
  - Definiše tip sadržaja u odgovoru

- ***XMLHttpRequest.responseXML*** **Read only**
  - Vraća Document objekat koji sadži odgovor na zahtev, ili null ukoliko zahtev nije uspeo ili se sadržaj ne može parsirati kao XML ili HTML.

Property-ji XMLHttpRequest-a

- ***XMLHttpRequest.responseURL*** **Read only**
  - Serijalizvoani oblik URL-a odgovora ili null

- ***XMLHttpRequest.status*** **Read only**
  - Vraća statusni kod odgovora

- ***XMLHttpRequest.statusText*** **Read only**
  - Vraća opisni tekst statusa odgovora (npr. ”200 OK”).

- ***XMLHttpRequest.timeout***
  - Broj milisekundi pre nego što se zahtev automatski terminira.

- ***XMLHttpRequestEventTarget.ontimeout***
  - EventHandler koji se poziva kada istekne postavljeno vreme za timeout.

- ***XMLHttpRequest.upload*** **Read only**
  - XMLHttpRequestUpload objekat koji reprezentuje podatke koji se uploaduju.

- ***XMLHttpRequest.withCredentials***
  - Boolean koji govori da li cross-site Access-Control zahteve treba obaviti korišćenjem kredencijala (cookies ili authorizaciona polja u zaglavlju).

XMLHttpRequest - praćenje napretka zahteva

```
var oReq = new XMLHttpRequest();

oReq.addEventListener("progress", updateProgress);
oReq.addEventListener("load", transferComplete);
oReq.addEventListener("error", transferFailed);
oReq.addEventListener("abort", transferCanceled);

oReq.open();

// … konfigurisati i poslati zahtev

// progress on transfers from the server to the client (downloads)
function updateProgress (oEvent) {
  if (oEvent.lengthComputable) {
    var percentComplete = oEvent.loaded / oEvent.total * 100;
    // ...
  } else {
    // Unable to compute progress information since the total size is unknown
  }
}
function transferComplete(evt) {
  console.log("The transfer is complete.");
}
function transferFailed(evt) {
  console.log("An error occurred while transferring the file.");
}
function transferCanceled(evt) {
  console.log("The transfer has been canceled by the user.");
}
```

Fetch API

- Noviji API, obezbeđuje interfejse za pribavljanje resursa, pa i preko mreže.

- Suštinski za one koji su već koristili XMLHttpRequest upotreba Fetch API-ja je vrlo slična

- Fetch obezbeđuje generičku definiciju Request i Response objekata (kao i svega ostalog povezanog sa mrežnim zahtevima). Osmišljen je tako da bude upotrebljiv gde god se pojavi potreba u budućnosti.

Fetch API

- Za slanje zahteva i prihvatanje traženog resursa koristi se WindowOrWorkerGlobalScope.fetch() metoda.

- Ova fetch() metoda je implementirana u više interfejsa, primerice u Window i WorkerGlobalScope.

- fetch() metod da zahteva jedan argument - putanju do resursa koji se želi dobaviti.

- Vraća **Promise** objekat koji se rezolvira u **Response** objekat — u momentu kada server pošalje *response header* i to čak i kada sam response predstavlja error HTTP status.
  - Kao drugi, opcioni argument pri pozivu se može proslediti i inicijalizacioni objekat (koji sadrži Request specifikaciju).

Fetch API - Request

- Request interfejs reprezentuje zahtev koji se šalje serveru.
- Korespondirajući Request objekat može se kreirati eksplicitno koristeći Request() konstruktor, ali je verovatnije da će biti vraćen kao rezultat poziva neke druge API operacije ili pristupa nekom drugom property-ju npr. FetchEvent.request .

Request properties

- ***Request.cache*** Read only
  - cache mode zahteva (default, reload, no-cache).
- ***Request.context*** Read only **deprecated**
- ***Request.credentials*** Read only
  - sadrži kredencijale zahteva (omit, same-origin, include). Podrazumevano je same-origin.
- ***Request.destination*** Read only
  - String (iz RequestDestination enumeracije) koji opisuje tip sadržaja koji se pribavlja.
- ***Request.headers*** Read only
  - Sadrži polja zagalavlja zahteva.
- ***Request.integrity*** Read only
  - Sadrži vrednost za proveru intergriteta subresursa (npr., sha256-BpfBw7ivV8q2jLiT13fxDYAe2tJllusRSZ273h2nFSE=).

Request properties

- ***Request.method*** Read only
  - Sadrži metod zahteva (GET, POST, etc.)
- ***Request.mode*** Read only
  - sadrži mode zahteva (cors, no-cors, same-origin, navigate.)
- ***Request.redirect*** Read only
  - sadrži informaciju o tome kako treba obraditi redirekciju. Može imati vrednost *follow*, *error*, ili *manual*.
- ***Request.referrer*** Read only
  - Informacija o referalu zahteva (preko koga je zahtev došao do servera.
- ***Request.referrerPolicy*** Read only
  - Sadrži informaciju o načinu postupanja sa referalima (e.g., no-referrer).
- ***Request.url*** Read only
  - sadrži URL zahteva.

- Request implementira i Body, tako da sadrži i dodatne property-je:
- ***body*** Read only
  - jednostavan ReadableStream of the body contents.
- ***bodyUsed*** Read only
  - boolean vrednost koja pokazuje da li je body već iskorišten u repsonsu

Request metode

- ***Request.clone()***
  - Klonira tekući request objekat

- Kroz Request su dostupne i Body metode

- ***Body.arrayBuffer()***
  - Vraća Promise koji se rezolvira u ArrayBuffer reprezentaciju tela zahteva.
- ***Body.blob()***
  - Vraća promise objekat koji se rezolvira u Blob reprezentaciju tela zahteva.
- ***Body.formData()***
  - Vraća promise koji se rezolvira u FormData reprezentaciju tela zahteva.
- ***Body.json()***
  - Vraća promise objekat koji se rezolvira kao JSON rreprezentacija tela zahteva.
- ***Body.text()***
  - Vraća promise objekat koji se rezolvira kao USVString (text) rreprezentacija tela zahteva.

Fetch API - Response

- Response interfejs reprezentuje odgovor koji je dobijen od servera.
- Korespondirajući Response objekat može se kreirati eksplicitno koristeći Response.Response() konstruktor, ali je verovatnije da će biti vraćen kao rezultat poziva neke druge API operacije.

Response properties

- ***Response.headers*** Read only
  - Sadržaj zaglavlja u odgovoru.
- ***Response.ok*** Read only
  - Boolean koji pokazuje da li je zahtev bio uspešan(status u rasponu 200–299).
- ***Response.redirected*** Read only
  - Pokazuje da li je odgovor dobijen kao rezultat redirekcije (URL polje ima više od jednog zapisa).
- ***Response.status*** Read only
  - Statusni kod odgovora
- ***Response.statusText*** Read only
  - Statusni tekst odgovora
- ***Response.trailers***
  - Promise objekat koji se rezolvira u Headers objekat a sadrži vrednosti iz HTTP Trailer zaglavlja.
- ***Response.type*** Read only
  - Tip odgovora ***(e.g., basic, cors).***
- ***Response.url*** Read only
  - URL sa kojeg je odgovor poslan

- Kao i zahtev i odgovor sadrži Body.body i Body.bodyUsed

Response metode

- ***Repsonse.clone()***
  - Klonira tekući response objekat
- ***Response.error()***
  - Vraća Response objekat koji je povezan sa mrežnom greškom.
- ***Response.redirect()***
  - Kreira novi response sa novim URL-om.

- Kroz Request su dostupne i Body metode

- ***Body.arrayBuffer()***
  - Uzima response tok i čita ga do kraja i vraća Promise koji se rezolvira u ArrayBuffer reprezentaciju tela zahteva.
- ***Body.blob()***
  - Uzima response tok i vraća promise objekat koji se rezolvira u Blob reprezentaciju tela zahteva.
- ***Body.formData()***
  - Uzima response tok i vraća promise koji se rezolvira u FormData reprezentaciju tela zahteva.
- ***Body.json()***
  - Uzima response tok i vraća promise objekat koji se rezolvira kao JSON rreprezentacija tela zahteva.
- ***Body.text()***
  - Uzima response tok i vraća promise objekat koji se rezolvira kao USVString (text) rreprezentacija tela zahteva.

Primer upotrebe response metode

```
const image = document.querySelector('.my-image');
fetch('flowers.jpg')
.then(response => response.blob())
.then(blob => {
  const objectURL = URL.createObjectURL(blob);
  image.src = objectURL;
});
```

```
// Function to do an Ajax call
const doAjax = async () => {
  const response = await fetch('Ajax.php'); // Generate the Response object
  if (response.ok) {
    const jsonValue = await response.json(); // Get JSON value from the response body
    return Promise.resolve(jsonValue);
  } else {
    return Promise.reject('*** PHP file not found');
  }
}
```

Upotreba Fetch API

- U osnovnoj verziji Fetch API je vrlo jednostavan za upotrebu.
  - Važno je zapamtiti da se Promise objekat uvek rezolvira na Response kada se od servera primi bilo kakav odgovor, pa makar on bio i Error.

```
fetch('http://example.com/movies.json')
  .then(response => response.json())
  .then(data => console.log(data));
```

Upotreba Fetch API

```
// Example POST method implementation:
async function postData(url = '', data = {}) {
  // Default options are marked with *
  const response = await fetch(url, {
    method: 'POST', // *GET, POST, PUT, DELETE, etc.
    mode: 'cors', // no-cors, *cors, same-origin
    cache: 'no-cache', // *default, no-cache, reload, force-cache, only-if-cached
    credentials: 'same-origin', // include, *same-origin, omit
    headers: {
      'Content-Type': 'application/json'
      // 'Content-Type': 'application/x-www-form-urlencoded',
    },
    redirect: 'follow', // manual, *follow, error
    referrerPolicy: 'no-referrer', // no-referrer, *no-referrer-when-downgrade, origin, origin-
when-cross-origin, same-origin, strict-origin, strict-origin-when-cross-origin, unsafe-url
    body: JSON.stringify(data) // body data type must match "Content-Type" header
  });
  return response.json(); // parses JSON response into native JavaScript objects
}

postData('https://example.com/answer', { answer: 42 })
  .then(data => {
    console.log(data); // JSON data parsed by `data.json()` call
  });
```