XMLHttpRequest

Komunikacija između web pregledača i HTTP servera na kojima se nalaze fajlovi web sajtova zasniva se na međusobnom smenjivanju zahteva i odgovora. Web pregledači upućuju zahteve, a HTTP serveri takve zahteve obrađuju i formiraju odgovore, propraćene eventualnim podacima. S obzirom na to da je osnovni predmet interesovanja u ovom kursu frontend programiranje, primarno se bavimo različitim načinima na koje se korišćenjem web pregledača serveru mogu uputiti zahtevi i na taj način poslati ili zatražiti određeni podaci.

U prethodnoj lekciji imali ste prilike da se upoznate sa osnovnim načinima na koje je korišćenjem web pregledača serveru moguće uputiti HTTP zahtev:

- otvaranjem neke web stranice;
- klikom na neki link na stranici;
- prosleđivanjem forme.

Sve su to osnovni načini koje je moguće koristiti kako bi se serveru uputio GET ili POST HTTP zahtev. Ipak, sve pobrojane pristupe karakteriše jedna zajednička osobina – svi oni proizvode potpuno osvežavanje stranice koju korisnik pregleda u web pregledaču. Pored ovih osnovnih pristupa, web pregledači izlažu i skup funkcionalnosti koje je moguće koristiti za slanje HTTP zahteva korišćenjem JavaScript koda koji mi samostalno pišemo. Takvi pristupi omogućavaju da se HTTP zahtevi pošalju bez potrebe za osvežavanjem kompletne stranice – odnosno, da se obavi slanje HTTP zahteva koje je potpuno nevidljivo za korisnika koji pregleda web stranicu. Na taj način otvaraju se mogućnosti za moćnu manipulaciju sadržajem i strukturom dokumenta, koji je moguće ažurirati parcijalno u zavisnosti od potreba i namene web sajta. Skup funkcionalnosti koji tako nešto omogućava drugačije se naziva AJAX i on će biti predmet lekcije koja je pred vama.

Šta je AJAX?

AJAX je skraćenica za **A**synchronous **J**avaScript **A**nd **X**ML. Reč je zapravo o pojmu pod kojim je objedinjen skup različitih tehnologija koje omogućavaju da se korišćenjem programskog koda koji mi pišemo uputi HTTP zahtev serveru. Tako se AJAX zapravo odnosi na upotrebu različitih tehnologija sa kojima smo se mi već susretali – HTML, CSS, JavaScript, DOM, XML i JSON. Pored nabrojanih tehnologija, presudne komponente koje omogućavaju komunikaciju sa serverom korišćenjem JavaScript jezika jesu i aplikativni programski interfejsi koje web pregledači izlažu na korišćenje kodu koji pišemo. Dva najznačajnija takva API-a su:

- XMLHttpRequest API;
- Fetch API.

Dva navedena API-a u kombinaciji sa nešto ranije nabrojanim tehnologijama čine osnovu AJAX modela. Tako AJAX sam po sebi nije tehnologija, već samo pojam koji se koristi da označi specifičan način upotrebe svih pobrojanih tehnologija, što na kraju omogućava web aplikacijama da obavljaju parcijalno ažuriranje podataka, bez potrebe za osvežavanjem kompletne stranice.

Pojam *Asynchronous JavaScript And XML* skovao je 2005. godine Jesse James Garrett. Ovaj pojam se na naš jezik najjednostavnije može prevesti kao *Asinhroni JavaScript i XML*. Reč *asinhroni* u ovom pojmu odnosi se isključivo na činjenicu da AJAX omogućava upućivanje HTTP zahteva bez potrebe za osvežavanje kompletne stranice. Bitno je znati i to da, iako je pojam *XML* obuhvaćen samim nazivom AJAX, to ne znači da je AJAX ograničen na korišćenje XML jezika za razmenu podataka između klijenata i servera. Danas se umesto XML-a mnogo češće kao tekstualni format za razmenu koristi JSON. Oba jezika bila su predmet prvog dela ovog kursa, a kao što možete da naslutite, sada je došao trenutak da naučeno i praktično iskoristimo na primerima realizacije AJAX modela.

U ovoj lekciji bavićemo se korišćenjem XMLHttpRequest API-a, dok će naredna lekcija biti posvećena Fetch API-u.

Šta je XMLHttpRequest?

Pojam XMLHttpRequest u osnovi se odnosi na jedan objekat koji omogućava komunikaciju sa serverom. Takva komunikacija ogleda se u mogućnosti slanja i primanja podataka u JSON, XML, HTML ili čistom tekstualnom formatu.

`XMLHttpRequest` veoma često se skraćeno naziva `XHR`.

Kako bi se mogle koristiti funkcionalnosti `XMLHttpRequest` objekta, njega je prvo potrebno kreirati. Kreiranje se obavlja korišćenjem istoimene konstruktorske funkcije:

```
let xhr = new XMLHttpRequest();
```

Prikazanom naredbom obavljeno je kreiranje `XMLHttpRequest` objekta. Osnovni način njegove upotrebe zasniva se na sledećim koracima:

- kreiranje i inicijalizacija zahteva;
- slanje zahteva;
- slušanje različitih događaja koje `XMLHttpRequest` objekat emituje, a koji govore o različitim stanjima kroz koje prolazi obrada upućenog zahteva;
- dobijanje odgovora i obrada eventualnih podataka koji su zajedno sa odgovorom pristigli od servera.

Pitanje

Osnovni objekat koji omogućava slanje HTTP zahteva serveru korišćenjem JavaScript jezika zove se:

a) XMLHttpRequest
b) AJAX
c) Post
d) Get

Objašnjenje

Pojam `XMLHttpRequest` u osnovi se odnosi na istoimeni objekat koji omogućava komunikaciju sa serverom.

Inicijalizacija i slanje zahteva korišćenjem XMLHttpRequest objekta

Prvi korak u korišćenju `XMLHttpRequest` objekta odnosi se na kreiranje, odnosno inicijalizaciju zahteva, što se obavlja upotrebom metode `open()`.

XMLHttpRequest metoda open()

Inicijalizacija novog HTTP zahteva obavlja se korišćenjem metode `open()`, `XMLHttpRequest` objekta. Njena sintaksa izgleda ovako:

```
xhr.open(method, url, [async, user, password])
```

Metoda `open()` može da prihvati dva obavezna i tri opciona parametra:

`method` – `string` vrednost kojom se definiše HTTP metoda zahteva; to znači da ovaj parametar može imati vrednosti kao što su `"GET"`, `"POST"`, `"PUT"`, `"DELETE"`...; obavezan parametar;

`url` – vrednost tipa `string` ili `URL`, kojom se definiše adresa na koju će biti upućen zahtev; obavezan parametar;

`async` – logička vrednost (`true` ili `false`) kojom se definiše da li će zahtev biti upućen asinhrono ili sinhrono; vrednost `true` rezultuje stvaranjem asinhronog zahteva, a vrednost `false` proizvodi sinhroni zahtev; ovo je opcioni parametar, koji kao svoju podrazumevanu vrednost ima `true`; to praktično znači da pozivanje metode `open()` ne blokira izvršavanje koda koji se nalazi nakon njenog poziva; većina web pregledača više ne podržava upućivanje sinhronih zahteva;

`user`, `password` – parametri kojima se definiše korisničko ime i lozinka, respektivno, ukoliko server zahteva autentifikaciju; reč je o opcionim parametrima koji kao podrazumevanu vrednost imaju `null`.

Na osnovu upravo prikazane sintakse metode `open()`, možemo napisati kod za inicijalizaciju zahteva:

```
let xhr = new XMLHttpRequest();
xhr.open('GET', 'users.php');
```

Sada je nakon kreiranja `XMLHttpRequest` objekta obavljen poziv `open()` metode sa dva obavezna parametra. Njima je definisano da će biti kreiran HTTP GET zahtev i da će on biti upućen na relativnu URL adresu `users.php`. To praktično znači da će zahtev biti obrađen od strane fajla `users.php` koji se na serveru nalazi u istom folderu kao i fajl sa ovakvim JavaScript kodom. Naravno, metodi `open()` moguće je proslediti i neku apsolutnu URL adresu:

```
let xhr = new XMLHttpRequest();
xhr.open('GET', 'http://www.mysite.com/users.php');
```

Definisanje apsolutnih URL putanja praktikuje sa samo kada se zahtev šalje na adresu koja pripada nekom drugom domenu.

Oba prikazana primera su ilustrovala inicijalizaciju zahteva koji će biti obrađen od strane serverskih skripti (skripti koje su konkretno napisane PHP jezikom). Objekat `XMLHttpRequest` omogućava da se obaviti i direktno čitanje sadržaja nekog tekstualnog fajla, u šta se ubrajaju i `.xml` i `.json` fajlovi, pa je moguće napisati i nešto ovako:

```
let xhr = new XMLHttpRequest();
xhr.open('GET', 'users.json');
```

Na ovaj način će nam `XMLHttpRequest` objekat omogućiti da pročitamo sadržaj jednog JSON fajla.

Slanje zahteva korišćenjem XMLHttpRequest objekta

Prethodni redovi ilustrovali su osobine metode `open()`. Iako njen naziv navodi na zaključak da se njenim korišćenjem otvara konekcija, bitno je znati da se metoda `open()` koristi isključivo za inicijalizaciju zahteva, a ne za njegovo slanje, odnosno otvaranje konekcije.

Kako bi se inicijalizovani HTTP zahtev poslao serveru, neophodno je koristiti metodu `send()`.

XMLHttpRequest metoda send()

Metoda kojom se obavlja slanje HTTP zahteva korišćenjem `XMLHttpRequest` objekta je metoda `send()`:

```
xhr.send([body])
```

Metoda `send()` može da prihvati jedan parametar koji nije obavezan:

`body` – podaci koji će biti postavljeni u telo zahteva

U prethodnoj lekciji ste mogli da pročitate da GET zahtevi tipično nemaju telo. Dakle, iako je to moguće uraditi, postavljanje podataka u telo slanjem GET zahteva nema nikakvu upotrebnu vrednost. Stoga se `body` parametar `send()` metode uglavnom koristiti prilikom upućivanja POST zahteva.

Nakon upoznavanja `send()` metode, možemo da napišemo minimalnu logiku koja će obaviti slanje HTTP zahteva serveru:

```
let xhr = new XMLHttpRequest();
xhr.open('GET', 'countries.json');
xhr.send();
```

Slušanje različitih događaja XMLHttpRequest objekta

Prilikom slanja HTTP zahteva, razumljivo je očekivati određeni odgovor od servera, upakovan u format HTTP odgovora. Stoga je na neki način neophodno definisati logiku koja će se aktivirati kada se od servera dobije odgovor na upućen zahtev. To se postiže korišćenjem nekoliko događaja `XMLHttpRequest` objekta.

XMLHttpRequest događaji

`XMLHttpRequest` objekat obezbeđuje nekoliko različitih načina kako bi se obradio serverski odgovor. Svi oni se zasnivaju na slušanju određenih događaja, koji se emituju u različitim situacijama, odnosno fazama obrade zahteva. Tri osnovna događaja `XMLHttpRequest` objekta su:

`load` – aktivira se kada je zahtev u potpunost obrađen i kada je kompletna sadržina HTTP odgovora na raspolaganju;

`error` – aktivira se kada zahtev ne može biti upućen zbog problema sa mrežnom konekcijom i ostalih sličnih problema tehničke prirode koji sprečavaju web pregledač da uputi zahtev;

`progress` – aktivira se periodično tokom preuzimanja podataka koji čine serverski HTTP odgovor; svako aktiviranje ovog događaja propraćeno je i informacijom o količini do sada preuzetih podataka odgovora, tako da se ovaj događaj efikasno može koristiti za praćenje napredovanja preuzimanja podataka iz serverskog odgovora.

Pretplatu na slušanje tri upravo opisana događaja moguće je postići na dva načina, što je uostalom slučaj i sa većinom drugih događaja koje nam web pregledači stavljaju na raspolaganje. Prvi način podrazumeva korišćenje odgovarajućih svojstava koja poseduje `XMLHttpRequest` objekat:

- `onload`
- `onerror`
- `onprogress`

Ovakvim svojstvima mogu se dodeliti reference na funkcije koje će se aktivirati kada dođe do pojave konkretnog događaja:

```
xhr.onload = function () {
      console.log('Response loaded: ' + xhr.response);
};
xhr.onerror = function () {
      console.log('Network Error');
};
xhr.onprogress = function (event) {
      console.log('Received ' + event.loaded + ' of ' + event.total);
};
```

Drugi način za prijavu na upravo prikazane događaje jeste korišćenjem metode `addEventListener()`:

```
xhr.addEventListener("load", function () {
      console.log('Response loaded: ' + xhr.response);
});
xhr.addEventListener("error", function () {
      console.log('Network Error');
});
xhr.addEventListener("progress", function () {
      console.log('Received ' + event.loaded + ' of ' + event.total);
});
```

Obrada odgovora dobijenog od servera

U prethodnim redovima ste mogli da vidite da se u slučaju uspešne obrade zahteva od strane servera podaci isporučuju u okviru funkcije koja se pridružuje `load` događaju. Ipak, pored samih podataka, unutar funkcije za obradu `load` događaja moguće je doći do još nekih važnih informacija o HTTP odgovoru.

Svojstva XMLHttpRequest objekta za dobijanje podataka i zaglavlja HTTP odgovora

Najznačajnija svojstva `XMLHttpRequest` objekta čije vrednosti je moguće dobiti nakon emitovanja `load` događaja su:

`xhr.status` – statusni kod HTTP odgovora; npr. `200` znači da je zahtev uspešno obrađen, `404` da traženi resurs na postoji itd.; u prethodnoj lekciji ste mogli da pročitate više o statusnim odgovorima;

`xhr.statusText` – statusna poruka koja prati statusni kod; tako poruka `OK` prati status `200`, a poruka `Not Found` kod `404`; i o statusnim porukama ste mogli da čitate u prethodnoj lekciji;

`xhr.response` – sadržaj tela HTTP odgovora, odnosno podaci koje server isporučuje klijentu.

Podrazumevano, sadržaj tela HTTP odgovora dobija se kao običan tekst (*plain text*). Ipak, ukoliko je unapred poznato koji podaci se očekuju od servera, to je moguće definisati svojstvom `xhr.responseType` i na taj način automatski obaviti konverziju ili parsiranje podataka. Svojstvo `xhr.responseType` može imati sledeće vrednosti:

`""` (prazan string) – podaci se dobijaju kao običan tekst;
`"text"` – podaci se dobijaju kao običan tekst;
`"arraybuffer"` – podaci se automatski prevode u ArrayBuffer objekat;
`"blob"` – podaci se automatski prevode u Blob objekat;
`"document"` – podaci se automatski parsiraju u `XMLDocument` objekat;
`"json"` – podaci se automatski parsiraju iz JSON formata u JavaScript objekat.

`XMLHttpRequest` objekat poseduje i svojstva `responseText` i `responseXML` koja je moguće koristiti za dobijanje sadržaja tela HTTP odgovora u tekstualnom, odnosno objektnom `XMLDocument` obliku. Ipak, reč je o starim svojstvima koja su zamenjena pristupom koji podrazumeva kombinaciju svojstava `xhr.response` i `xhr.responseType`.

Pored podataka, `XMLHttpRequest` omogućava da se dobiju i informacije iz zaglavlja HTTP odgovora. Za obavljanje takvog posla, moguće je koristiti sledeće metode:

`getResponseHeader(name)` – koristi se za čitanje jednog zaglavlja po imenu koje se prosleđuje kao parametar;

`getAllResponseHeaders()` – koristi se za čitanje svih zaglavlja.

Na osnovu osobina upravo prikazanih svojstava za dobijanje podataka od servera, ukoliko na primer od servera očekujemo podatke u JSON formatu, možemo napisati ovako nešto:

```
let xhr = new XMLHttpRequest();
xhr.open('GET', 'countries.json');
xhr.responseType = 'json';
xhr.send();


xhr.onload = function () {
    let countries = xhr.response;

    for (let i = 0; i < countries.length; i++) {
        console.log(countries[i].name);
    }
};
```

Pre slanja zahteva definisali smo tip podataka odgovora postavljanjem vrednosti `xhr.responseType` svojstva na `json`. Na taj način će parsiranje JSON teksta biti obavljeno automatski, pa će vrednost svojstva `response` unutar funkcije za obradu `load` događaja biti JavaScript objekat kreiran na osnovu tekstualnih JSON podataka.

Primer 1 – Kompletan kod za čitanje sadržaja jednog JSON fajla korišćenjem XMLHttpRequest objekta

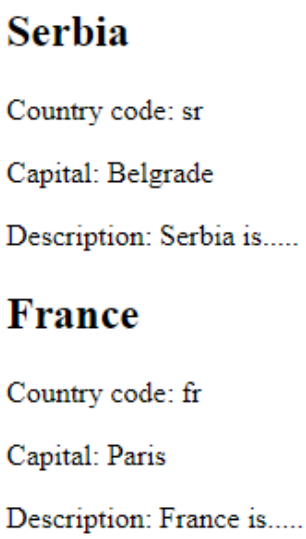
Zbog bolje preglednosti i čitljivosti, u nastavku će biti prikazan kompletan primer za čitanje sadržaja jednog JSON fajla korišćenjem `XMLHttpRequest` objekta:

```
let xhr = new XMLHttpRequest();
xhr.open('GET', 'countries.json');
xhr.responseType = 'json';
xhr.send();
xhr.onload = function () {
    let countries = xhr.response;
    for (let i = 0; i < countries.length; i++) {
        document.body.innerHTML += '<h2>' + countries[i].name + '</h2>';
        document.body.innerHTML += '<p>Country code: ' +
countries[i].countryCode + '</p>';
        document.body.innerHTML += '<p>Capital: ' + countries[i].capital +
'</p>';
        document.body.innerHTML += '<p>Description: ' +
countries[i].description + '</p>';
    }
};
xhr.onerror = function () {
    console.log('Network Error');
};
xhr.onprogress = function (event) {
    console.log('Received ' + event.loaded + ' of ' + event.total);
};
```

Sadržaj JSON fajla koji se na ovaj način čita izgleda ovako:

```
[
    {
        "countryCode": "sr",
        "name": "Serbia",
        "capital": "Belgrade",
        "description": "Serbia is....."
    },
    {
        "countryCode": "fr",
        "name": "France",
        "capital": "Paris",
        "description": "France is....."
    }
]
```

Nakon čitanja sadržaja JSON fajla, u web pregledaču se dobija prikaz kao na slici 8.1.



Slika 8.1. Izgled web stranice u pregledaču nakon čitanja i obrade dobijenih podataka

Slanje podataka serveru korišćenjem XMLHttpRequest objekta

U prethodnim redovima ilustrovana je primena `XMLHttpRequest` objekta za postizanje slanja zahteva za određenim podacima. Upravo zbog toga su u primerima kreirani GET HTTP zahtevi, čija namena je zahtevanje određenih podataka od servera. Podjednako važan segment rada sa podacima ogleda se u mogućnosti njihovog slanja serveru. Procedura slanja veoma je slična upravo prikazanim primerima, s tim što se umesto GET HTTP zahteva kreira POST HTTP zahtev.

Jednostavan primer kreiranja POST HTTP zahteva i slanja tekstualnih JSON podataka serveru može da izgleda ovako:

```
var user = {
      name: "Ben",
      surname: "Torrance"
}

let json = JSON.stringify(user);


let xhr = new XMLHttpRequest();
xhr.open("POST", 'users.php');
xhr.setRequestHeader('Content-type', 'application/json; charset=utf-8');

xhr.send(json);
```

U primeru se prvo obavlja kreiranje JavaScript objekta kojim se predstavlja jedan korisnik (`user`). Takav JavaScript objekat se zatim serijalizuje u JSON string korišćenjem metode `JSON.stringify()`, o kojoj je već bilo reči u prvom delu ovog kursa. Kreira se `XMLHttpRequest` objekat i obavlja inicijalizacija POST zahteva, a zatim se JSON podaci postavljaju unutar tela HTTP zahteva, prosleđivanjem promenljive `json` metodi `send()` (nešto ranije u ovoj lekciji je prikazana sintaksa metode `send()`).

Postavljanje zaglavlja prilikom slanja zahteva

Prikazani primer ilustruje i mogućnost definisanja zaglavlja HTTP zahteva. Tako nešto se može postići korišćenjem metode `setRequestHeader()`:

```
setRequestHeader(name, value)
```

`name` – naziv zaglavlja
`value` – vrednost zaglavlja

U primeru je korišćenjem metode `setRequestHeader()` obavljeno postavljanje zaglavlja `Content-type`, kojim se serveru govori koji tip podataka može da očekuje u telu zahteva. Bitno je razumeti da postavljanje `Content-type` zaglavlja nije obavezno, ali u nekim situacijama može biti korisno, jer serverska logika takvu informaciju može da iskoristi za automatsko parsiranje podataka.

Primer 2 – Kompletan primer slanja JSON podataka serveru

U nastavku će biti prikazan kompletan primer slanja JSON podataka korišćenjem `XMLHttpRequest` objekta:

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-
scale=1.0">
        <title>Sending data using</title>
    </head>
```

```
    <body>
        <script>
            var user = {
                name: "Ben",
                surname: "Torrance"
            }
            let json = JSON.stringify(user);
            let xhr = new XMLHttpRequest();
            xhr.open("POST",
'http://examples.wizardbit.com/ajax/users.php');
            xhr.setRequestHeader('Content-type', 'application/json;
charset=utf-8');
            xhr.send(json);
            xhr.onload = () => document.write(xhr.response);
        </script>
    </body>
</html>
```

Zahtev se sada upućuje skripti sa serverskom logikom (napisanom PHP jezikom), koja je specijalno kreirana za potrebe ovog primera. Server će prihvatiti podatke koje klijent na ovaj način upućuje i formiraće odgovarajući odgovor, koji se na klijentu dobija unutar funkcije koja se aktivira prilikom emitovanja `load` događaja. U prikazanom primeru je definisano da će podaci sa servera direktno biti prikazani unutar tela (`body` elementa) HTML stranice. Uspešna obrada zahteva od strane servera na stranici će proizvesti sledeću poruku:

```
Hello from the backend! You have sent me the following user:
Ben Torrance
```

Slanje podataka forme korišćenjem XMLHttpRequest objekta

Za metodu `send()` se može reći da je vrlo svestrana kada je reč o tipu podataka koji može da prihvati kao parametar. U upravo prikazanom primeru njoj je prosleđen `string` koji predstavlja JSON u tekstualnom obliku. Ipak, ona može da prihvati podatke i u raznim drugim objektnim oblicima – `FormData`, `Blob`, `BufferSource`... U nastavku će biti prikazan primer slanja podataka HTML forme serveru, ali ne na tradicionalan način, već korišćenjem objekta `XMLHttpRequest`.

Kako bi se podaci forme predstavili u objektnom JavaScript obliku, pa samim tim i prosledili serveru korišćenjem `XMLHttpRequest` objekta, koristi se jedan poseban objekat – `FormData`.

FormData

`FormData` objekat koristi se za kreiranje parova ključeva i vrednosti koji predstavljaju elemente forme i njihove vrednosti u objektnom obliku. `FormData` omogućava preuzimanje podataka iz postojećih formi, ali i kreiranje potpuno novih parova ključeva i vrednosti, čime se otvara put za manipulaciju podacima formi pre nego što oni budu prosleđeni serveru.

Stoga je `FormData` objekat moguće konstruisati na dva načina:

u potpunosti samostalno ili

korišćenjem reference na postojeći HTML `form` element.

Samostalno kreiranje `FormData` objekta:

```
var formData = new FormData();
formData.append("name", "Ben");
formData.append("message", "This is message...");
```

Kod ilustruje samostalno kreiranje `FormData` objekta, korišćenjem istoimene konstruktorske funkcije. Parovi ključeva i vrednosti se dodaju korišćenjem metode `append()`. Na ovaj način je kreirana objektna reprezentacija jedne HTML forme, koja bi, na primer, mogla imati dva `input` elementa sa `name` atributima, `name` i `message`, respektivno.

`FormData` objekat je moguće konstruisati i na osnovu neke postojeće HTML forme, tako što se konstruktorskoj funkciji `FormData()` prosleđuje referenca na DOM element koji predstavlja formu:

```
var formData = new FormData(someFormElement);
```

Sada će `FormData` objekat automatski biti popunjen parovima ključeva i vrednosti na osnovu polja koje poseduje forma čija je referenca prosleđena korišćenjem promenljive `someFormElement`.

Osnovni kod za slanje podataka forme korišćenjem `XMLHttpRequest` objekta može da izgleda ovako:

```
let formData = new FormData(document.forms.contact);

let xhr = new XMLHttpRequest();
xhr.open("POST", "contact.php");
xhr.send(formData);
```

U primeru se prvo kreira `FormData` objekat na osnovu forme koja već postoji unutar HTML koda. Zatim se konstruiše `XMLHttpRequest` objekat, inicijalizuje zahtev i `FormData` objekat kao parametar prosleđuje metodi `send()`, čime se podaci forme smeštaju unutar tela HTTP zahteva.

Primer 3 – Slanje podataka forme korišćenjem XMLHttpRequest objekta

Upravo prikazani primer predstavljao je samo mali isečak kompletnog primera za slanje podataka forme serveru korišćenjem JavaScript jezika. Stoga će u nastavku biti prikazan kompletan primer sa svim potrebnim elementima kako bi se podaci forme prosledili serveru korišćenjem `XMLHttpRequest` objekta. Pri tom će biti obavljeno i dodavanje jednog para ključa i vrednosti koji inicijalno ne postoje unutar same HTML forme.

Kompletan primer izgleda ovako:

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-
scale=1.0">
        <title>Sending Form data using XMLHttpRequest object</title>
        <style>
            body {
                font-family: sans-serif;
           }
            form input[type="text"],
            form textarea {
                display: block;
                margin-bottom: 8px;
                width: 360px;
            }
        </style>
    </head>
    <body>
        <form name="contact">
            <label for="name">Name: </label>
            <input type="text" name="name" id="name">
            <label for="message">Message: </label>
            <textarea name="message" id="message" cols="30"
rows="10"></textarea>
            <input type="submit" value="Submit">
        </form>

        <div id="response">
        </div>
        <script>
            document.forms.contact.addEventListener("submit", function (e) {
                e.preventDefault();

                let formData = new FormData(document.forms.contact);

                formData.append("contact-code",
Math.random().toString(36).substring(7));

                let xhr = new XMLHttpRequest();
                xhr.open("POST",
"http://examples.wizardbit.com/ajax/contact.php");
                xhr.send(formData);

                xhr.onload = () => {
                    var responseDiv = document.getElementById("response");
                    responseDiv.innerHTML = xhr.response;
                };
            });
        </script>
    </body>
</html>
```

Kod ilustruje kompletnu HTML stranicu sa svim elementima koji su potrebni kako bi se podaci forme prosledili serveru korišćenjem `XMLHttpRequest` objekta. Neophodno je da obratite pažnju na nekoliko stvari:

forma je identična kao u primeru iz prethodne lekcije – poseduje dva elementa za unos podataka: `input` element `name` i `textarea` element `message`;

`div` element sa `id`-jem `response` koristi se za prikaz odgovora koji se dobije od servera;
u JavaScript kodu je prvo obavljena pretplata na `submit` događaj koji će biti emitovan prilikom prosleđivanja forme;

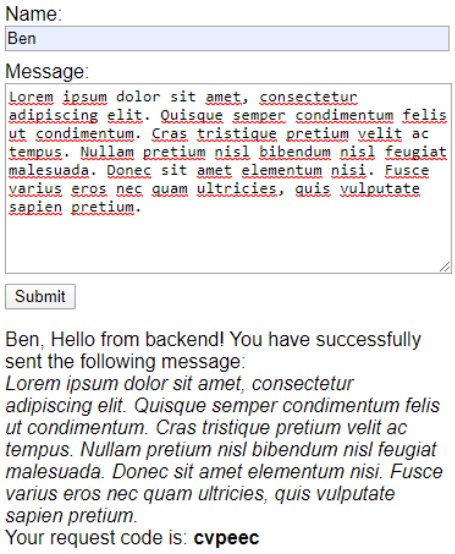
podrazumevana akcija prosleđivanja otkazana je pozivanjem metode `e.preventDefault()`, što znači da web pregledač neće samostalno obaviti prosleđivanje podataka ovakve forme;

unutar JavaScript koda obavljeno je dodavanje još jednog para ključa i vrednosti, koji ne postoje unutar HTML koda; reč je o kodu (*code*) zahteva koji se nasumično generiše;

forma se prosleđuje PHP skripti koja je specijalno napravljena za ovaj primer i koja se nalazi na eksternom serveru;

podaci koji se dobiju od servera kao odgovor na ovakav zahtev postavljaju se kao sadržaj `div` elementa sa `id`-jem `response`.

Uspešno prosleđivanje podataka forme stvoriće prikaz kao na slici 8.2.



Slika 8.2. Primer uspešnog prosleđivanja podataka forme korišćenjem JavaScripta

CORS

U dosadašnjem toku lekcije ilustrovano je slanje nekoliko GET i POST HTTP zahteva. Neki od njih su korišćeni za čitanje sadržaja fajlova na lokalnom serveru, dok su drugi bili namenjeni slanju podataka na udaljene servere. Prilikom komunikacije sa serverom korišćenjem JavaScript programskog jezika, bitno je poznavati još jedan veoma važan koncept, koji je od presudnog značaja za ishod komunikacije između klijenata i servera na webu. Reč je o pojmu CORS.

CORS je skraćenica za Cross-Origin Resource Sharing, što se na naš jezik najjednostavnije može prevesti kao *deljenje resursa između dva različita domena*. Ipak, u izvornom nazivu se umesto pojma domena koristi jedan za nas potpuno nov pojam – **origin**.

Origin je zapravo spoj tri različite informacije: protokola, domena i porta. Na primer, JavaScript logika za upućivanje zahteva serveru, koja se u prethodnim primerima nalazila na lokalnom HTTP serveru, imala je sledeći origin:

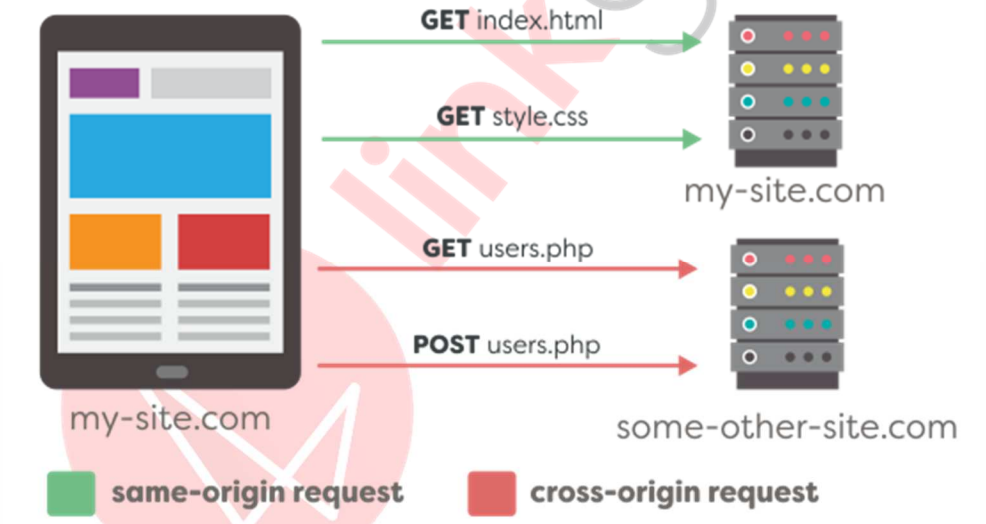
```
http://localhost
```

JavaScript logika koja se nalazi na nekom produkcionom serveru može imati sledeći origin:

```
https://www.mysite.com
```

Iz primera je jasno da origin predstavlja deo URL adrese, bez konkretne putanje, čime se omogućava definisanje jasnog porekla JavaScript koda (odnosno, njegovo vezivanje za protokol, domen i port).

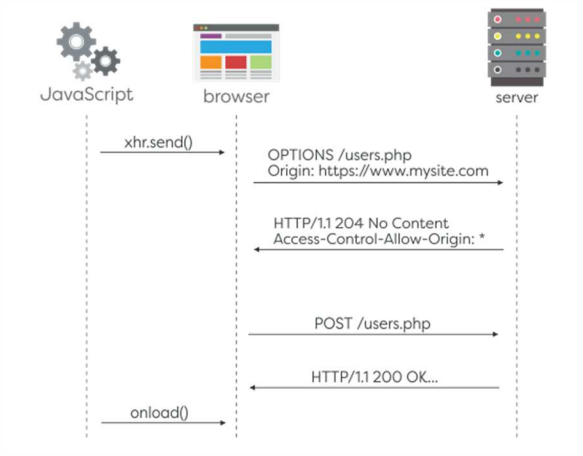
Cross-origin HTTP zahtev nastaje kada JavaScript pokuša da kontaktira sa nekim serverom koji ima različit origin. Na primer, cross-origin HTTP zahtev može nastati ukoliko naša klijentska JavaScript logika koja se nalazi na adresi `http://localhost` pokuša da uputi zahtev serveru na adresi `https://www.mysite.com`.



Slika 8.3. Same-origin i cross-origin zahtevi

Slika 8.3. ilustruje razliku između same-origin i cross-origin zahteva (zelenom bojom su prikazani same-origin, a crvenom cross-origin zahtevi).

Moderni web pregledači podrazumevano blokiraju cross-origin HTTP zahteve iz sigurnosnih razloga. Ipak, kako bi serverima ostavili mogućnost da samostalno odluče da li žele da prihvate određeni cross-origin HTTP zahtev, web pregledači pre upućivanja konkretnog zahteva serveru šalju jedan poseban zahtev koji se naziva **preflight**. Svrha takvog zahteva jeste omogućavanje serveru da navede izvore sa kojih želi da prihvati zahteve (slika 8.4).



Slika 8.4. Uprošćeni način funkcionisanja CORS-a

Na slici 8.4 se može videti nekoliko veoma važnih aspekata CORS mehanizma.

1. Kako bi web pregledač utvrdio da li HTTP server dozvoljava prijem zahteva sa konkretnog izvora (origina), on upućuje preflight zahtev; preflight zahtev se realizuje kao HTTP zahtev upućen korišćenjem OPTIONS metode; web pregledač zahtevu obavezno pridodaje i origin zaglavlje.
2. Web pregledač je u obavezi da u odgovoru priloži zaglavlje `Access-Control-Allow-Origin` i da kao njegovu vrednost navede sve origine koje prihvata, ili zvezdicu, ukoliko prihvata zahteve sa svih origina.
3. Tek nakon dobijanja odgovora na preflight zahtev i ukoliko HTTP server prihvati konekciju sa klijentskog origina, sledi upućivanje konkretnog zahteva.
4. Server odgovara na konkretan zahtev i web pregledač isporučuje podatke metodi koja je pridružena `onload` svojstvu ukoliko je za upućivanje zahteva korišćen `XMLHttpRequest` objekat.

Kako sve ovo izgleda u praksi, biće prikazano u narednom primeru.

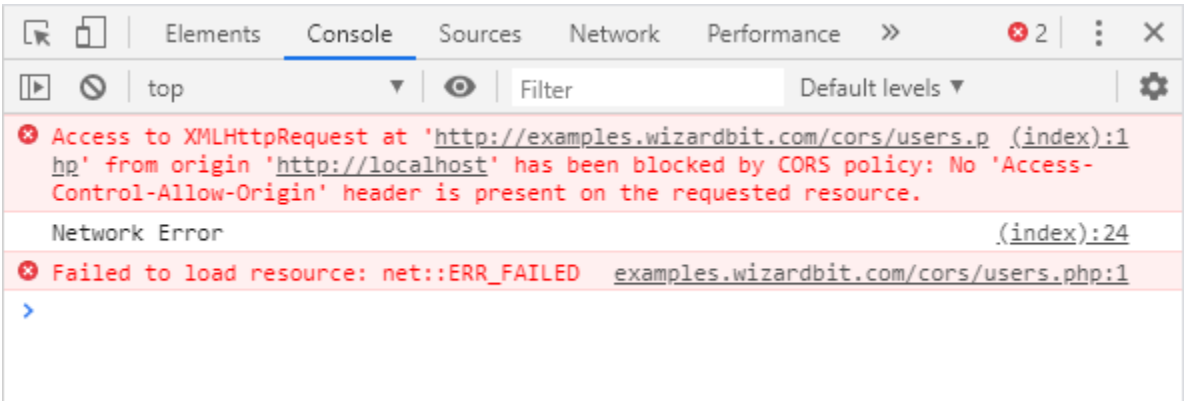
Primer 4 – Cross-origin HTTP zahtevi

U dosadašnjem toku lekcije, u primerima koji su prikazani, već su upućivani cross-origin HTTP zahtevi. Ipak, serverska logika koja je obrađivala takve zahteve bila je specijalno kreirana tako da ih prihvati bez problema. Sada će biti prikazan primer u kojem tako nešto neće biti slučaj.

```
            let xhr = new XMLHttpRequest();
            xhr.open('GET', 'http://examples.wizardbit.com/cors/users.php');
            xhr.responseType = 'json';
            xhr.send();
            xhr.onload = function () {
                var person = xhr.response;
                document.write(person.name + ' ' + person.age + ' ' +
person.city);
            };
            xhr.onerror = function () {
```

```
                console.log('Network Error');
            };
            xhr.onprogress = function (event) {
                console.log('Received ' + event.loaded + ' of ' +
event.total);
            };
```

Primer ilustruje upućivanje cross-origin GET HTTP zahteva. Zahtev je cross-origin zato što je origin JavaScript logike `http://localhost`, a origin servera `http://examples.wizardbit.com`. Kad se uputi zahtev, unutar konzole web pregledača se dobija poruka kao na slici 8.5.



Slika 8.5. Poruka koja se unutar konzole dobija kao produkt blokiranja zahteva od strane web pregledača usled kršenja CORS modela

Pored `users.php` fajla kome se u primeru upućuje zahtev, na serveru postoji i fajl `users-allow-cors.php` sa skriptom koja prihvata cross-origin HTTP zahteve. Stoga je dovoljno napraviti izmenu na putanji na koju se upućuje zahtev i primer će proizvesti drugačiji rezultat:

```
xhr.open('GET', 'http://examples.wizardbit.com/cors/users-allow-cors.php');
```

Rezultat sada izgleda ovako:

```
John 30 New York
```

Restrikcije web pregledača koje se tiču cross-origin HTTP zahteva takođe su razlog zbog kog se korišćenjem `XMLHttpRequest` objekta ne može pristupiti fajlovima koji se nalaze na fajl sistemu korisnika.

Rezime

- AJAX je skraćenica za **A**synchronous **J**avaScript **A**nd **X**ML.
- AJAX je skup različitih tehnologija koje omogućavaju da se korišćenjem programskog koda koji mi pišemo uputi HTTP zahtev serveru.
- Dva najznačajnija Web API-a koja omogućavaju komunikaciju sa serverom korišćenjem JavaScript jezika su XMLHttpRequest API i Fetch API.
- Pojam XMLHttpRequest u osnovi se odnosi na jedan objekat koji omogućava slanje HTTP zahteva i obradu HTTP odgovora.
- Inicijalizacija novog HTTP zahteva obavlja se korišćenjem metode `open()`, `XMLHttpRequest` objekta.
- Metoda kojom se obavlja slanje HTTP zahteva korišćenjem `XMLHttpRequest` objekta je metoda `send()`.
- `XMLHttpRequest` objekat obezbeđuje nekoliko različitih događaja koji se aktiviraju kako bi se obradio serverski odgovor; ti događaji su `load`, `error` i `progress`.
- Podaci koje je server poslao zajedno sa HTTP odgovorom mogu se pročitati unutar funkcije za obradu `load` događaja.
- Definisanje zaglavlja HTTP zahteva može se postići korišćenjem metode `setRequestHeader()`.
- `FormData` objekat koristi se za kreiranje parova ključeva i vrednosti koji predstavljaju elemente forme i njihove vrednosti u objektnom obliku.
- CORS je skraćenica za Cross-Origin Resource Sharing, što se na naš jezik najjednostavnije može prevesti kao deljenje resursa između dva različita domena.
- Origin je spoj tri različite informacije: protokola, domena i porta.
- Cross-origin HTTP zahtev nastaje kada JavaScript pokuša da kontaktira sa nekim serverom koji ima različit origin.
- Kako bi serveri mogli samostalno da odluče da li žele da prihvate određeni cross-origin HTTP zahtev, web pregledači, pre upućivanja konkretnog zahteva, serveru šalju jedan poseban zahtev koji se naziva preflight.