Fetch API

Prethodna lekcija bila je posvećena komunikaciji sa serverom upotrebom `XMLHttpRequest` objekta. Reč je o osnovnom pristupu za ostvarivanje takve komunikacije. Ipak, upotreba `XMLHttpRequest` objekta ima i modernu alternativu, koja se ogleda u korišćenju Fetch API-a.

Korišćenjem Fetch API-a, moguće je postići sve ono što je prikazano u prethodnoj lekciji, ali na nešto drugačiji, jednostavniji i moderniji način. Stoga će u lekciji pred vama svi već prikazani scenariji čitanja i slanja podataka biti realizovani korišćenjem Fetch API-a.

Šta je Fetch API?

Fetch API je skup funkcionalnosti koji je namenjen komunikaciji sa serverom. Pri tome, Fetch API se može doživeti kao moderna alternativa pristupu koji je ilustrovan u prethodnoj lekciji, a koji podrazumeva korišćenje `XMLHttpRequest` objekta.

Osnova Fetch API-a jeste metoda `fetch()`.

Metoda fetch()

Metoda `fetch()` poseduje sledeću sintaksu:

```
let promise = fetch(url, [options])
```

Metoda `fetch()`može da prihvati dva parametra, od čega je samo prvi parametar obavezan:

- `url` – putanja na kojoj se nalazi fajl
- `options` – parametri za konfigurisanje zahteva koji se upućuje

Ukoliko se drugi parametar izostavi, metoda `fetch()` se koristi za upućivanje GET zahteva. Za upućivanje svih ostalih vrsta zahteva neophodno je definisati i vrednosti drugog parametra (`options`), kojim je pored HTTP metode moguće konfigurisati i razne druge osobine zahteva.

Metoda `fetch()` kao svoju povratnu vrednost emituje Promise objekat; stoga je za dobijanje rezultata izvršavanja ovakve metode neophodno koristiti pristupe za konzumiranje Promisea: `then()`, `catch()` i `finally()` metode ili ključne reči `async/await`.

U nastavku ove lekcije prvo će biti ilustrovano slanje HTTP zahteva kojim se od servera traže neki podaci, a zatim i primeri slanja podataka serveru.

Pitanje

Metoda `fetch()` poseduje:

a) dva obavezna parametra
b) jedan obavezan i jedan opcioni parametar
c) tri obavezna parametra
d) jedan obavezan i dva opciona parametra

Objašnjenje:

Metoda `fetch()` može da prihvati dva parametra, od čega je samo prvi parametar, kojim se definiše URL zahteva, obavezan.

Upućivanje GET zahteva korišćenjem Fetch API-a

GET HTTP zahtev se korišćenjem Fetch API-a može uputiti na sledeći način:

```
fetch('countries.json').then(function (response) {
    return response.text();
});
```

Pošto metoda `fetch()` kao svoju povratnu vrednost emituje `Promise` objekat, njegovo ispunjenje je u primeru konzumirano definisanjem callback funkcije koja je kao parametar prosleđena metodi `then()`.

Promise se ispunjava odmah nakon što udaljeni HTTP server pošalje zaglavlje odgovora. Drugim rečima, razrešenje Promisea koji metoda `fetch()` emituje kao povratnu vrednost ne podrazumeva preuzimanje kompletnog sadržaja. Kao svoju povratnu vrednost, Promise `fetch()` metode emituje još jedan specifičan objekat Fetch API-a. Reč je o objektu `Response`.

Response objekat

`Response` je objekat istoimene ugrađene klase Fetch API-a. On sadrži osnovne informacije o HTTP odgovoru koji isporučuje server:

- `status` – statusni kod odgovora
- `ok` – `boolean` promenljiva čija vrednost ukazuje na to da li je zahtev uspešno obrađen; vrednost ovog svojstva je `true` za sve statusne kodove klase 200 (odnosno, za statusne kodove između 200 i 299)
- `headers` – objekat sa kolekcijom svih zaglavlja odgovora, čije vrednosti je moguće pojedinačno dobiti pozivanjem metode `get()` (na primer `response.headers.get('Content-Type')`)

Pored svojstava kojima je moguće dobiti informacije iz zaglavlja i statusne linije HTTP odgovora, `Response` objekat poseduje i različite metode, koje je moguće koristiti za dobijanje sadržaja HTTP odgovora, odnosno za dobijanje podataka koji se nalaze u telu HTTP odgovora:

- `response.text()` – čita podatke kao običan tekst
- `response.json()` – parsira JSON podatke iz tela odgovora i isporučuje JavaScript vrednost, objekat ili niz, u zavisnosti od JSON sadržaja
- `response.formData()` – parsira podatke iz tela odgovora i isporučuje `FormData` objekat
- `response.blob()` – parsira podatke iz tela odgovora i isporučuje `Blob` objekat
- `response.arrayBuffer()` – parsira podatke iz tela odgovora i isporučuje `ArrayBuffer` objekat

Sve metode za čitanje podataka iz tela odgovora kao svoju povratnu vrednost emituju `Promise` objekte.

Nakon upoznavanja osobina `Response` objekta, možemo da nastavimo sa realizacijom primera:

```
fetch('countries.json').then(function (response) {
    return response.json();
}).then(function (countries) {
      // code for managing data

});
```

Nad `Response` objektom je pozvana metoda `json()` i na taj način je inicirano čitanje JSON podataka iz tela odgovora i njihovo parsiranje. S obzirom na to da metoda `json()` kao svoju povratnu vrednost emituje `Promise` objekat, njegova konzumacija je obavljena nadovezivanjem još jednog poziva metode `then()`. Njena callback funkcija aktiviraće se kada podaci iz tela odgovora budu u potpunosti preuzeti. Podaci će biti automatski parsirani i prosleđeni kao ulazni parametar callback funkcije (u primeru je to parametar `countries`).

Kako bi primer slanja GET zahteva i čitanja podataka isporučenih od servera u potpunosti bio kompletan, neophodno je u njega inkorporirati i logiku za obradu eventualnih grešaka. Bitno je znati da se `Promise` koji kao povratnu vrednost emituje `fetch()` metoda ne ispunjava samo kada web pregledač nije u stanju da uputi zahtev, na primer, zbog mrežnih problema. Kako bi se obradili takvi scenariji, dovoljno je definisati poziv metode `catch()` na dnu lanca nadovezivanja metoda za konzumiranje Promisea:

```
fetch('countries.json').then(function (response) {
    return response.json();
}).then(function (countries) {

      // code for managing data

}).catch(function (err) {
    console.log('Fetch problem: ' + err.message);
});
```

Ipak, bitno je znati da ukoliko dođe do neke druge greške prilikom obrade HTTP zahteva od strane servera, `Promise` objekti koje isporučuje Fetch API neće biti odbijeni. Na primer, ukoliko traženi resurs ne postoji, do odbijanja Promisea neće doći. Kako bi se obradili i takvi scenariji, neophodno je iskoristiti pristupe za inspekciju statusne linije i zaglavlja HTTP odgovora, prikazanih nešto ranije:

```
fetch('countries.json').then(function (response) {

    if (!response.ok) {
        throw Error("Error while reading file.");
    }


    return response.json();
}).then(function (countries) {

    // code for managing data

}).catch(function (err) {
    console.log('Fetch problem: ' + err.message);
});
```

Sada je unutar callback funkcije kojom se obrađuje razrešenje `Promise` objekta `fetch()` metode postavljen i jedan uslovni blok kojim se proverava da li je zahtev uspešno obrađen, inspekcijom vrednosti promenljive `ok`. Ona će imati vrednost `true` za sve statusne kodove iz klase 200, čime efikasno možemo utvrditi da li je server uspešno obradio naš zahtev. Ukoliko se zahtev ne obradi uspešno, u kodu se obavlja podizanje izuzetka. Takav izuzetak će biti obrađen na identičan način kao i odbijanje Promisea – aktiviranjem callback funkcije koja je kao parametar prosleđena `catch()` metodi.

Primer 1 – Slanje GET zahteva i obrada dobijenih podataka

U nastavku će biti prikazan kompletan primer slanja GET HTTP zahteva korišćenjem Fetch API-a:

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-
scale=1.0">
        <title>AJAX</title>
    </head>
    <body>
        <script>
            fetch('countries1.json').then(function (response) {
                if (!response.ok) {
                    throw Error("Error while reading file.");
                }
                return response.json();
            }).then(function (countries) {
                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>';
```

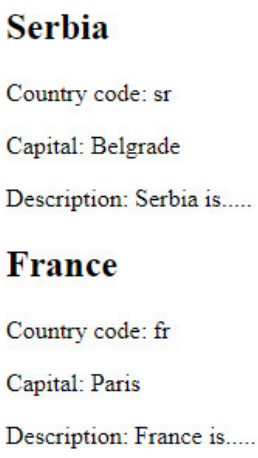
```
                    }
                }).catch(function (err) {
                    console.log('Fetch problem: ' + err.message);
                });
            </script>
        </body>
    </html>
```

Kod ilustruje kompletan HTML dokument sa logikom za upućivanje GET HTTP zahteva i čitanje sadržaja jednog JSON fajla. JSON fajl se nalazi na istom serveru kao i klijentska logika. Takođe, ne zaboravite da je neophodno da ovakav HTML dokument postavite na lokalni HTTP server, kako web pregledač ne bi blokirao zahtev za čitanjem.

Sadržaj `countries.json` fajla koji se čita izgleda ovako:

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

Nakon izvršavanja prikazanog koda, unutar web pregledača biće dobijen rezultat kao na slici 9.1.



Slika 9.1. Izgled web stranice u pregledaču nakon čitanja i obrade dobijenih podataka

Identično je bilo moguće postići i korišćenjem `async/await` pristupa:

```
            (async () => {

                try {
                    let response = await fetch('countries.json');
                    if (!response.ok) {
                        throw new Error("Error while reading file.");
                    }
                    let countries = await response.json();

                    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>';
                    }

                } catch (err) {
                    console.log('Fetch problem: ' + err.message);
                }

            })();
```

Sada je kompletna funkcionalnost za slanje GET zahteva i obradu dobijenog odgovora smeštena unutar jedne samopozivajuće async funkcije. S obzirom na to da je samopozivajuća, ona će biti automatski pozvana, a činjenica da je reč o async funkciji nam omogućava da unutar nje koristimo `await` ključnu reč za obradu asinhronih operacija koje kao svoju povratnu vrednost emituju `Promise`.

Slanje podataka serveru korišćenjem Fetch API-a

Prethodni redovi su ilustrovali način na koji se Fetch API koriti kako bi se serveru uputio zahtev za podacima. Naredni redovi će ilustrovati korišćenje Fetch API-a za slanje podataka serveru.

Slanje podataka serveru obavlja se formiranjem POST HTTP zahteva. Kako bi se takav zahtev kreirao korišćenjem `fetch()` metode, neophodno je pored putanje, definisane prvim parametrom, definisati i vrednost drugog parametra – `options`.

Konfigurisanje metode fetch()

Metoda `fetch()` kao svoj drugi parametar može da prihvati objekat kojim se konfiguriše zahtev koji će biti upućen serveru. Najznačajnija svojstva takvog objekta su:

- `method` – definiše HTTP metodu zahteva; tako vrednosti ovoga svojstva mogu biti `"GET"`, `"POST"`, `"DELETE"`...
- `headers` – definiše zaglavlja HTTP zahteva
- `body` – definiše telo HTTP zahteva; može da prihvati vrednosti različitih tipova u zavisnosti od formata podataka `String`, `Object`, `Blob`, `BufferSource`, `FormData`, `URLSearchParams`...

Na osnovu upravo prikazanih osobina objekta za konfigurisanje HTTP zahteva, možemo napisati prvi primer za upućivanje POST zahteva:

```
fetch('users.php', {
      method: 'POST',
      headers: {
            'Content-Type': 'application/json;charset=utf-8'
      },
      body: JSON.stringify(user)
});
```

Metodi `fetch()` je prosleđen i drugi parametar, koji poseduje tri svojstva. Stoga sada na praktičnom primeru možete videti na koji način se obavlja konfigurisanje različitih segmenata zahteva. Svojstvom `method` definisana je HTTP metoda zahteva. Zatim, svojstvom `headers` definisan je objekat sa jednim svojstvom, kojim se postavlja vrednost `Content-Type` zaglavlja. Na taj način će serveru biti stavljeno do znanja da očekuje podatke u JSON obliku. Na kraju, konkretni podaci spakovani su unutar tela zahteva, definisanjem vrednosti `body` svojstva.

Primer 2 – Slanje podataka korišćenjem Fetch API-a

Prikazani kod ilustruje samo deo kompletnog primera za slanje podataka serveru korišćenjem Fetch API-a. Evo koda kompletne HTML stranice:

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-
scale=1.0">
        <title>Fetch API - Sending data</title>
    </head>
    <body>
        <script>
            var user = {
                name: "Ben",
                surname: "Torrance"
            }
            fetch('http://examples.wizardbit.com/ajax/users.php', {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/json;charset=utf-
8'
                },
                body: JSON.stringify(user)
            }).then(function (response) {
                return response.text();
            }).then(function (text) {
                document.write(text);
            }).catch(function (err) {
                console.log('Fetch problem: ' + err.message);
            });
        </script>
    </body>
</html>
```

U prikazanom primeru, serveru se šalju JSON podaci koji su dobijeni serijalizacijom jednog JavaScript objekta. Nad `Response` objektom koji isporučuje `Promise` `fetch()` metode, poziva se metoda `text()`, kako bi se dobio serverski odgovor u čistom tekstualnom formatu. Nakon uspešnog slanja podataka serveru, unutar web pregledača se dobija sledeća poruka:

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

Primer se, naravno može realizovati i upotrebom `async/await` pristupa:

```
                    (async () => {
                       try {
                           let response = await
fetch('http://examples.wizardbit.com/ajax/users.php', {
                               method: 'POST',
                               headers: {
                                   'Content-Type':
'application/json;charset=utf-8'
                               },
                               body: JSON.stringify(user)
                           });
                           let text = await response.text();
                           document.write(text);
                       } catch (err) {
                           console.log('Fetch problem: ' +
err.message);
                       }
                   })();
```

Za kraj ove lekcije biće prikazan još jedan primer slanja podataka serveru korišćenjem Fetch API-a. Ovoga puta biće to podaci iz jedne forme.

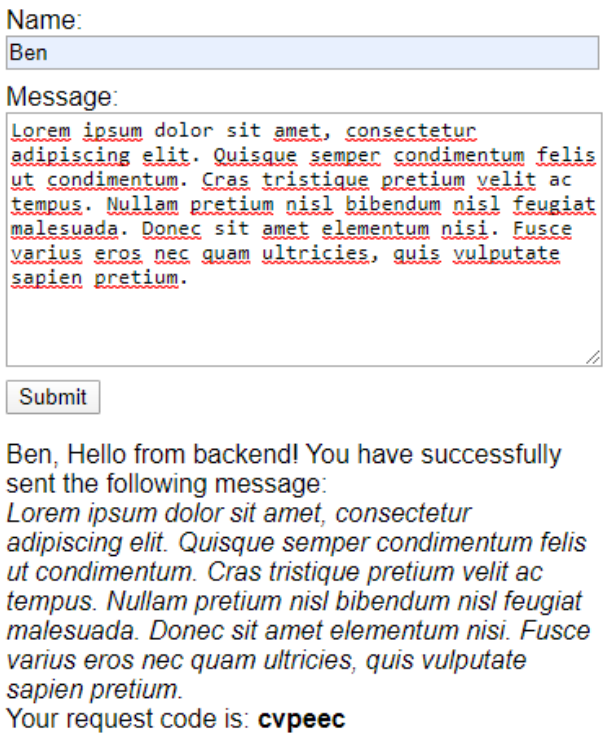
Primer 3 – Slanje podataka forme korišćenjem Fetch API-a

Primer je identičan onom iz prethodne lekcije, s tim što se sada za slanje podataka forme koristi Fetch API:

```
<!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 Fetch API</title>
        <style>
            body {
                font-family: sans-serif;
            }
            form input[type="text"],
            form textarea {
                display: block;
                margin-bottom: 8px;
```

```
                    width: 360px;
                }
                #response {
                    margin-top: 16px;
                }
            </style>
        </head>
        <body>
            <form name="contact">
                <label for="name">Name: </label>
                <input type="text" name="name" id="name"
autocomplete="off">
                <label for="message">Message: </label>
                <textarea name="message" id="message" cols="30" rows="10"
autocomplete="off"></textarea>
                <input type="submit" value="Submit">
            </form>
            <div id="response">
            </div>
            <script>
                document.forms.contact.addEventListener("submit", function
(e) {
                    e.preventDefault();
                    sendFormData();
                });
                async function sendFormData() {
                    let formData = new FormData(document.forms.contact);
                    formData.append("contact-code",
Math.random().toString(36).substring(7));
                    let result = await
fetch("http://examples.wizardbit.com/ajax/contact.php", {
                        method: "POST",
                        body: formData
                    });
                    let text = await result.text();
                    var responseDiv = document.getElementById("response");
                    responseDiv.innerHTML = text;
                }
            </script>
        </body>
    </html>
```

Primer proizvodi efekat kao na slici 9.2.



Slika 9.2. Primer uspešnog prosleđivanja podataka forme korišćenjem JavaScripta

Kada je reč o cross-origin zahtevima, za zahteve koji se upućuju korišćenjem Fetch API-a važi identično pravilo kao i kod zahteva upućenih korišćenjem `XMLHttpRequest` objekta.

Rezime

- Fetch API je moderan skup funkcionalnosti koji je namenjen komunikaciji sa serverom.
- Fetch API interno koristi Promise.
- Osnova Fetch API-a jeste metoda `fetch()`.
- Metoda `fetch()` može da prihvati dva parametra, pri čemu se prvi odnosi na putanju na koju će biti upućen zahtev, a drugi na podešavanja za konfigurisanje zahteva.
- Razrešenje Promisea koji metoda `fetch()` emituje kao povratnu vrednost ne podrazumeva preuzimanje kompletnog sadržaja, već samo zaglavlja odgovora.
- `Response` objekat istoimene, ugrađene klase Fetch API-a sadrži osnovne informacije o HTTP odgovoru koji isporučuje server.
- Kada se drugi parametar metode `fetch()`izostavi, obavlja se upućivanje GET HTTP zahteva.
- Drugim parametrom `fetch()` metode moguće je konfigurisati zahtev, definisanjem HTTP metode, sadržaja tela zahteva i njegovih zaglavlja.