FRONTEND MĀJASLAPAS IZSTRĀDE

9. lekcija - saziņa ar serveri

LATVIJAS UNIVERSITĀTE
BIZNESA, VADĪBAS UN EKONOMIKAS FAKULTĀTE

VADĪBAS UN UZŅĒMĒJDARBĪBAS MĀCĪBU CENTRS

NACIONĀLAIS ATTĪSTĪBAS PLĀNS 2020

EIROPAS SAVIENĪBA
Eiropas Sociālais fonds

IEGULDĪJUMS TAVĀ NĀKOTNĒ

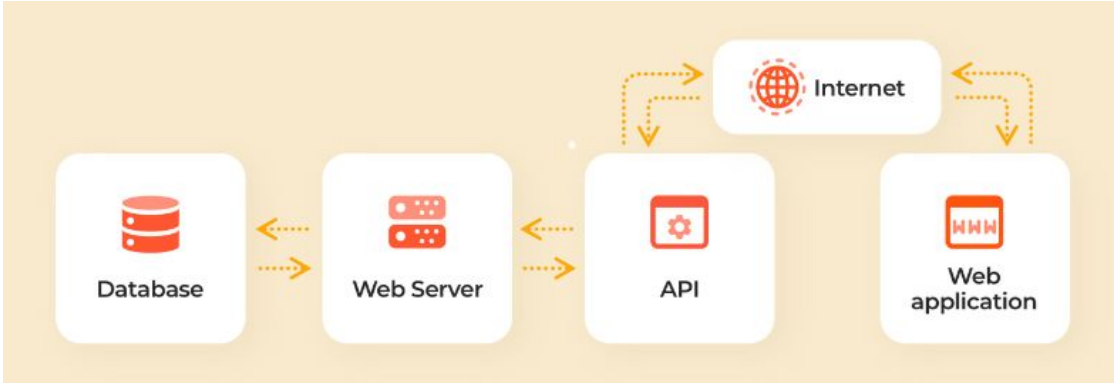
ESF projekts Nr. 8.4.1.0/16/I/001 ''Nodarbināto personu profesionālās kompetences pilnveide"

Lietojumprogrammas interfeiss (API)

Serveris kā API

Gluži kā mājaslapas apmeklētājam ir piejamas pogas, izvēlnes un ievadlauki, lai iegūtu vēlamo informāciju - no JavaScript puses mums ir pieejams līdzvērtīgs interfeis serverī ar kuru varam mijiedarboties jeb API.
Pieprasot un sūtot datus serverim no pārlūkprogrammas, izmantojam servera API (Application Programming Interface) jeb servera lietojumprogrammas interfeisu.
Serveris veic datu iegūšanu un saglabāšanu datubāzē - katram API piemīt nosacījumi kā pareizi pieprasīt un nosūtīt datus no pārlūkprogrammas, lai serveris varētu atgriezt aplikācijai nepieciešamo informāciju.
Katras aplikācijas servera konfigurācija un formāts datu pieprasīšanai/sūtīšanai būs atšķirīgs un dokumentēts katrā attiecīgajā projektā.
Lekcijas piemērā izmantosim bezmaksas API, lai demonstrētu šo saziņu starp pārlūkprogramm un serveri:

[Randomuser.me dokumentācija]

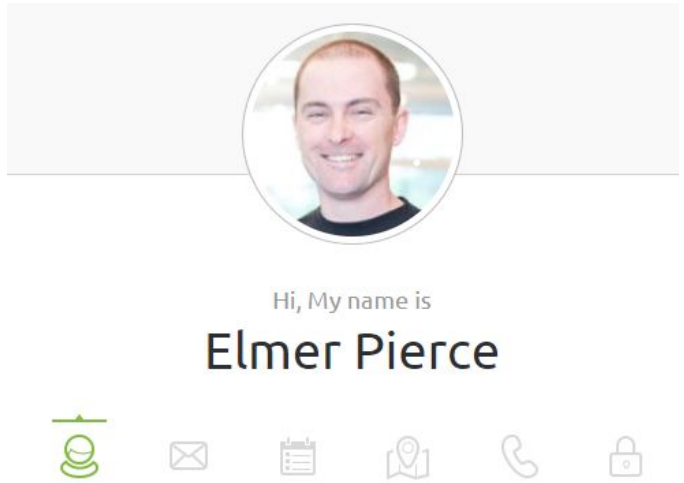


Fetch metode

fetch(“resource_URL”) metodei obligāts ir tikai viens parametrs - resursa URL, ko vēlamies iegūt no servera.

Izmantojot piemēru ar randomuser.me API, kā parametru varam norādīt **“https://randomuser.me/api/”**, kas atgriezīs JSON objektu ar nejauši izvēlētiem, mākslīga lieotāja datiem.
fetch(“https://randomuser.me/api/”)


```
▼{results: [{gender: "female", name: {title: "Madame", first: "Liv", last: "Colin"},…}],…}
  ▶info: {seed: "cca1ab230b6d1d73", results: 1, page: 1, version: "1.3"}
  ▼results: [{gender: "female", name: {title: "Madame", first: "Liv", last: "Colin"},…}]
    ▼0: {gender: "female", name: {title: "Madame", first: "Liv", last: "Colin"},…}
       cell: "079 163 98 05"
      ▶dob: {date: "1982-02-08T16:30:48.130Z", age: 40}
       email: "liv.colin@example.com"
       gender: "female"
      ▶id: {name: "AVS", value: "756.3557.3454.15"}
      ▶location: {street: {number: 9892, name: "Grande Rue"}, city: "Wikon", state: "Glarus", country: "Switzerland",…}
      ▶login: {uuid: "9289f2f5-6a1a-4beb-85cf-a7b67ddac745", username: "whitemouse536", password: "sigma",…}
      ▶name: {title: "Madame", first: "Liv", last: "Colin"}
       nat: "CH"
       phone: "078 743 35 27"
      ▶picture: {large: "https://randomuser.me/api/portraits/women/46.jpg",…}
      ▶registered: {date: "2009-09-16T02:38:13.966Z", age: 13}
```

API parametri

Lai iegūtu specifiskākus datus no servera, pievienojot URL parametrus, varam pieprasīt datus, kas atbilst parametru vērtību kritērijiem.

URL pievienojam parametrus aiz **?** simbola.
fetch("https://randomuser.me/api/**?**")

Serveris sagaida key/value pāri jeb parametra nosaukumu un vērtību pēc kuriem atlasīt datus no datubāzes.
Dotajā piemēra API serveris var saņemt pieprasījumu, piem., atgriezt tikai sieviešu dzimuma lietotāju datus:
fetch("https://randomuser.me/api/**?gender=female**")

Varam arī norādīt cik personu datu piemērus atgriezt:
fetch("https://randomuser.me/api/**?results=50**")

Kā arī varam kombinēt šos parametrus iekš URL ar **&** simbolu:
fetch("https://randomuser.me/api/**?results=50&gender=female**")

Iegūto datu apstrāde - Promise objekts

Līdz šim brīdim esam apskatījuši kā pieprasīt datus no API, bet pagaidām tos varam apskatīt tikai DevTools -> Network cilnē.
Lai ar tiem darbotos mūsu JavaScript kodā, nepieciešams vispirms sagaidīt atbildi no servera un tad varam piekļūt datiem.
Šādas operācijas tiek dēvētas par asinhronām jeb tiek veikta darbība, bet nav zināms, kad tā atgriezīs kādu rezultātu. JavaScript ir pieejamas vairākas metodes kā rīkoties ar asinhronām darbībām - viena no tām ir **Promise** objekts.
fetch() metode atgriež šādu Promise objektu, kas mums ļauj izmantot **then()** metodi, lai izpildītu kodu tikai pēc tam, kad esam saņēmuši atbildi no servera.
fetch().then()
then() sagaida parametru, kas būs funkcija ko izsaukt attiecīgajā brīdī. Šajā funkcijā kā parametrs tiek ievietots **response** datu straume jeb servera pilna atbilde:

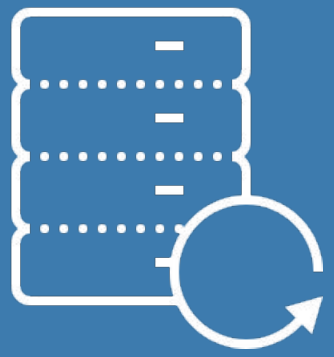
```
fetch("resource_URL")
   .then((response) => {
         // kods ko izpildīt, kad saņemta servera atbilde
   })
```

MDN dokumentācija par Promise pielietojumu

Iegūto datu apstrāde - Readabble stream

Tā kā serveris atgriež datus nelielās daļās(chunks) līdz visi dati ir nosūtīti, ar **response** datu straumi (ReadableStream) nav iespējams darboties kā līdz šim ierasto JavaScript objektu.
Ja sagaidām JSON tipa atbildi no servera, varam izmantot datu straumes **json()** metdoi, lai no tās iegūtu objektu. Tā kā pati **json()** metode arī ir asinhrona un nav zināms, kad tā beigs savu darbību - tā argriež Promise. Līdz ar to varam izmantot **then()** metodi, lai izpildītu kodu, kad ir iegūts rezultāts.
json() metodes gala rezultāts tiks padots secīgajai **then()** funkcijai kā parametrs:

```
fetch("resource_URL")
   .then((response) => {
      return response.json()
   }
   .then((responseObj) => {
         // kods ko izpildīt, kad iegūts JavaScript objekts no json() metodes
   })
```



Datu sūtīšana uz API

Konfigurācijas objekts

Nosūtot datus serverim, varam izmantot jau iepazīto **fetch()** metodi. Veicot datu sūtīšanu, fetch metodē nepieciešams pievienot vēl vienu parametru - konfigurācijas objektu. Šajā konfigurācijas objektā kā īpašības varam norādīt metodi ar kādu vēlamies mijaiedarboties ar servera datiem(method), sūtāmie dati (body) un informācija serverim par šo pieprasījumu (headers):

```
fetch("resource_URL", {
  method: // metode,
  body: // dati,
  headers: // informācija serverim,
})
```

Iepriekšējos piemēros jau noklusēti tika iestatīs method: "GET". "GET" metodei nav obligāti nepieciešams norādīt body un headers īpašības, tāpēc varam vienkāršos datu pieprasījuma gadījumos iztikt bez konfigurācijas objekta.

POST metode

Norādot method: **”POST”** varam nosūtīt serverim datus un tie tiks saglabāti daubāzē kā jauns ieraksts.

```
fetch(“resource_URL”, {
  method: ”POST”,
  body: // dati,
  headers: // informācija serverim,
})
```

Turpmākajos piemēros izmantosim API, kas var simulēt datu sūtīšanu kā JSON un ievietošanu datubāzē - JSON palceholder API.

```
fetch(“https://jsonplaceholder.typicode.com/posts”, {
  method: ”POST”,
  body: JSON.stringify(
        title: 'User title',
        body: 'Content text',
        userId: 1,
  ),
  headers: 'Content-type': 'application/json; charset=UTF-8'
})
```

Tā kā šis API ir gatavs saņemt JSON objekta datus, izmantojam JSON.stringify() metodi, lai pārvērstu JavaScript objektu JSON formātā.
headers sadaļā norādām nosūtāmo datu tipu **application/json** un izmantotos simbolus kopumu **charset=UTF-8**.

PUT metode

Norādot method: **”PUT”** varam nosūtīt serverim datus un datubāzē tiks atrasts ieraksts kuru aizvietot ar nosūtītajiem datiem.

```
fetch(“https://jsonplaceholder.typicode.com/posts”, {
  method: ”PUT”,
  body: JSON.stringify(
          id: 1,
          title: 'Updated title',
          body: 'Same old content text',
          userId: 1,
  ),
  headers: 'Content-type': 'application/json; charset=UTF-8'
})
```

Izmantojot **”PUT”** metodi, jānosūta atbilstošā datubāzes ieraksta pilna objekta struktūra, pat ja tiek mainīti tikai daļa no datiem.
Tā kā katram datubāzes ierakstam tiek piešķirts arī unikāls identifikātors, to nepieciešams norādīt body objekta īpašībā id, lai varētu veikt izmaiņas attiecīgajā ierakstā.

PATCH metode

Norādot method: **”PATCH”** varam nosūtīt serverim datus un datubāzē tiks atrasts ieraksts kurā veikt labojumus ar nosūtītajiem datiem.

```
fetch(“https://jsonplaceholder.typicode.com/posts”, {
  method: ”PATCH”,
  body: JSON.stringify(
          id: 1,
          body: 'Patched-up content text'
  ),
  headers: 'Content-type': 'application/json; charset=UTF-8'
})
```

Izmantojot **”PATCH”** metodi, varam nosūtīt serverim tikai daļēju izmaināmā ieraksta saturu un tas tiks papildināts.

PATCH metode

Norādot method: **”DELETE”** varam nosūtīt serverim datus par ierakstu ko dzēst no datubāzes.

```
fetch(“https://jsonplaceholder.typicode.com/posts”, {
  method: ”DELETE”,
  body: JSON.stringify(
          id: 1,
  ),
  headers: 'Content-type': 'application/json; charset=UTF-8'
})
```

Izmantojot **”DELETE”** metodi, varam nosūtīt serverim tikai identifikātoru ierakstam ko dzēst. Šis API pieļauj arī īsinātu pierakstu šai darbībai:

```
fetch(“https://jsonplaceholder.typicode.com/posts/1”, {
  method: ”DELETE”,
})
```

Kur dzēšamā ieraksta identifikātors norādīts URL un nav nepieciešams izmantot body vai headers īpašību.

VADĪBAS UN UZŅĒMĒJDARBĪBAS MĀCĪBU CENTRS

NACIONĀLAIS ATTĪSTĪBAS PLĀNS 2020

EIROPAS SAVIENĪBA
Eiropas Sociālais fonds

IEGULDĪJUMS TAVĀ NĀKOTNĒ

ESF projekts Nr. 8.4.1.0/16/I/001 ''Nodarbināto personu profesionālās kompetences pilnveide"

VADĪBAS UN UZŅĒMĒJDARBĪBAS MĀCĪBU CENTRS

Programmas nosaukums