Problem i kommunikationen

Hittills har vi utgått från att processen att hämta en resurs från en server har fungerat bra. Men vi kan stötta på många olika problem i kommunikationen med servern.

Kommunikationsproblem kan delas upp i:

HTTP-fel, servern fick förfrågan, och skickade en respons, men status-värdet inte är 200-299, vilket signalerar på att något gick snett.

Nätverksfel, ingen respons har tagits emot (vanligtvis för att servern aldrig fick förfrågan)

HTTP-fel

I dessa fall (**HTTP-fel**) tog servern emot förfrågan och skickade en *response*, som dock inte hade ett status mellan 200 och 299.

Klientfel, status kod: 4xx.

Dessa fel beror på att klienten har gjort något fel, t.ex: 400 Bad Request, 401 Unauthorized, 403 Forbidden, 404 Not Found.

Serverfel, status kod: 5xx

Dessa fel beror på att något inte fungerade hos servern, t.ex: 500 Internal Server Error, 502 Bad Gateway, 503 Service Unavailable, 504 Gateway Timeout

Nätverksfel

I dessa fall (**nätverksfel**) förfrågan inte fram till servern, så ingen respons skickades. Exempel på denna typ av problem:

Timeout, förfrågan tog för lång tid att besvara (typisk vid dålig uppkoppling)

DNS-fel: Vi har skrivit fel på URL:et... eller den har ändrats (mindre sannolikt)

Nätverksproblem: Problem med nätverksanslutningen (routern, ingen internet, brandvägg...)

Servern är nere: Servern är otillgänglig eller överbelastad

Felhantering

Det är viktigt att hantera kommunikationsfel så att vår webbsida alltid är "under kontroll".

> (det är faktiskt viktigt att kontrollera även eventuella fel som kan uppstå inom koden, men det hinner vi inte prata om i DU3).

HTTP-fel och Nätverksfel hanteras på olika sätt.

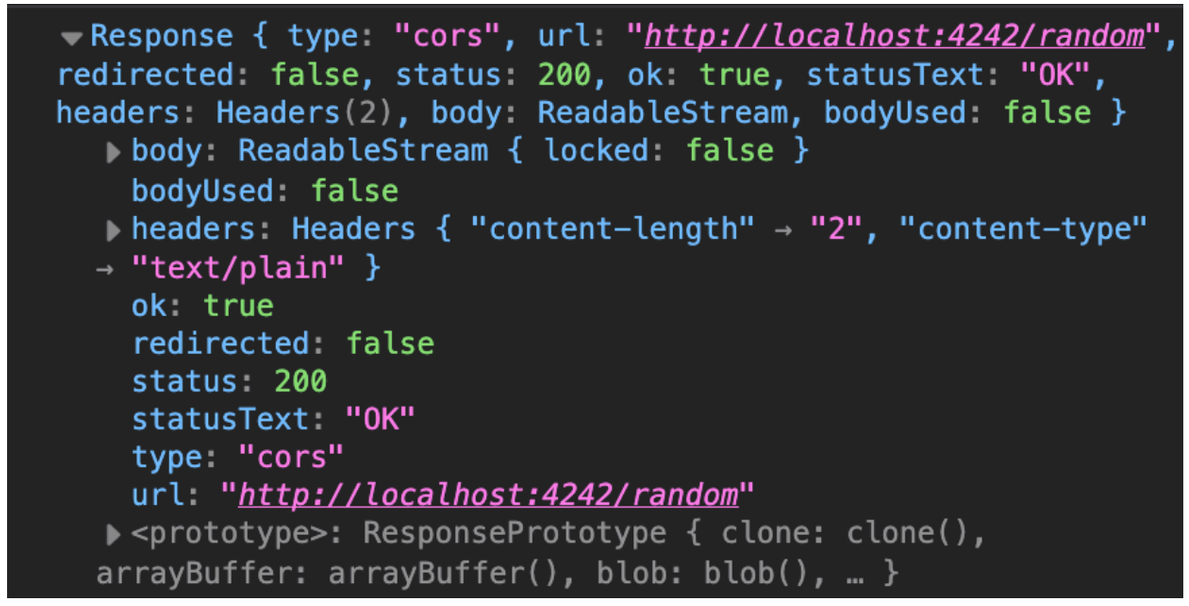
Felhantering: HTTP-fel

Vår promise har uppfyllts

Vid ett HTTP-fel tar vi emot en response, som dock innehåller ett status som inte ligger mellan 200 och 299.

Eftersom promiseResponse löstes till en *promise* så kommer funktionen responseHandler att anropas. Så felhanteringen bör ske där.

Response-objektet:



Attributet ok kommer att vara true om status är 200-299, false annars. Så om ok är false, så vet vi att något gick snett.

Felhantering: HTTP-fel

Vår promise har uppfyllts

Vid ett HTTP-fel tar vi emot en response, som dock innehåller ett status som inte ligger mellan 200 och 299.

Eftersom promiseResponse löstes till en *promise* så kommer funktionen responseHandler att anropas. Så felhanteringen bör ske där.

Om vi tittar på response-objektet igen:

```
▼Response { type: "cors", url: "http://localhost:4242/random",
redirected: false, status: 200, ok: true, statusText: "OK",
headers: Headers(2), body: ReadableStream, bodyUsed: false }
  ▶body: ReadableStream { locked: false }
    bodyUsed: false
  ▶headers: Headers { "content-length" → "2", "content-type"
  → "text/plain" }
    ok: true
    redirected: false
    status: 200
    statusText: "OK"
    type: "cors"
    url: "http://localhost:4242/random"
  ▶<prototype>: ResponsePrototype { clone: clone(),
  arrayBuffer: arrayBuffer(), blob: blob(), … }
```

Om det blev ett HTTP-fel så kommer ok att vara **false**

Om det blev ett HTTP-fel så kommer status inte att ligga inom 200-299

Felhantering: HTTP-fel

Vår promise har uppfyllts

```
const request = new Request(resource-url);

const responsePromise = fetch(request);
responsePromise.then(handleResponse);

function handleResponse (response) {

  if (!response.ok) {
    // hantera felet, t.ex:
    //   informera användaren via skärmen
    //   anropa en särskild funktion
    //   etc.

  } else {
    const resourcePromise = response.text();
    resourcePromise.then(handleResource);
  }
}

function handleResource (resource) {
  // do something with resource
}
```

Felhantering: Nätverksfel

Vår promise har avvisats

Vid ett nätverksfel löses inte promiseResponse, den blir istället avvisad (rejected). Så handleResponse kommer inte att anropas.

Det som anropas är istället den andra funktionen som kan anges i then()
(se 04 - Hur hanteras en promise?)

```
const request = new Request(resource-URL);
const promiseResponse = fetch(request);
promiseResponse.then(responseHandler, rejectHandler);
...
```

Felhantering: Nätverksfel

Vår promise har avvisats

```
const request = new Request(resource-URL);
const promiseResponse = fetch(request);
promiseResponse.then(responseHandler, rejectHandler);
```

rejectHandler kommer att anropas med själva felet som argument.

Vi är vanligtvis i denna kurs inte intresserade av själva felet utan av att det blev ett fel, så vi kan informera användaren.

Felhantering: Nätverksfel

Vår promise har avvisats

När vi har en Promise-chain, som t.ex:

```
const request = new Request(server-URL);
fetch(request)
  .then(response => response.text())
  .then(resource => {
     // do something with the resource
   })
  .catch(rejectHandler)

function rejectHandler (error) {
  // do something...
}
```

Kan vi, istället för att skriva rejectHandler vid varje then(), skriva en .catch() metod i slutet, som kommer att hantera avvisandet av alla promises.

OBS: Igen, använd inte Promise-chain om ni inte läser på om dem.