TDP013 - Webbprogrammering och interaktivitet

Föreläsning 3:
JS i webläsaren, AJAX, CORS, projekt och etikuppgift

Anders Fröberg
Institutionen för Datavetenskap (IDA)

LINKÖPINGS UNIVERSITET

Återblick från föreläsning 1 och 2

- JavaScript
    - Repetition av callbacks
- Node.js
    - Serverramverk skrivet i Javascript
    - Stöd för nästan allt i ES6 (om man arbetar med definierar sin kod som module)
- Mocha
    - Ett testramverk för Node.js
- Code coverage med Istanbul och C8
- MongoDB
- HTML+CSS

Callbacks och asynkrona anrop

Event-loop

- Node.js använder bara en tråd och alla requests körs i denna
- Om Node.js väntade på varje rad kod att exekvera innan den fortsatte skulle det innebära att alla som gjorde anrop till servern fick vänta

```
let data = longRunningProcess()
```

Om vi är oförsiktiga och använder kod som ovan på fel ställe kan responsen bli mycket långsam...

Asynkrona anrop

- Kör en funktion utan att pausa
- Kan utnyttja **callbacks** eller **Promises**
- Asynkrona funktioner markeras med **async**

```
async function doSomething(){
    // något tidskrävande
}
```

  - async returner implicit ett Promise

- För att vänta (inte att föredra!) på en **async**-function används **await**

```
await doSomething()
```

  - Kan användas för att få asynkrona anrop att bete sig seriellt
  - Måste i sin tur användas i en async-funktion

Asynkrona anrop: Promise

- Objekt som representerar ett “löfte” (Promise)
- Ett Promise-objekt har ett state
    - **pending**
    - **fulfilled**
    - **rejected**
- Promise tar två callbacks som parametrar: **resolve** och **reject**
- När state ändras körs en av dessa funktioner
- resolve-parameterns värde sätts via **.then(...)**
- reject-parameterns värde sätts via **.catch(...)**
- Flera **.then(...)** kan definieras för samma Promise

Asynkrona anrop: Promise

```
function loadData(){
    return [
        {'title': 'Gone in 60 seconds', 'year': 2000},
        {'title': 'Pulp Fiction', 'year': 1994}
    ]
}

let p = new Promise((resolve, reject) => {
    let data = loadData()
    if(data !== null){
        resolve(data)
    } else {
        reject('Failed to load data')
    }
})

p.then((x) => {
    // ‘then’ kallas på om vi lyckas
    console.log('Data loaded successfully:')
    console.log(JSON.stringify(x, null, 2))
}).catch((msg) => {
    // catch kallas på om vi misslyckas
    console.log(`Something went wrong: ${msg}`)
})
```

Vad är callbacks?

- Funktioner som **argument** till funktioner
- Lämnar över ansvaret för att fånga upp data och event till den kallade funktionen
- Stor del av både JavaScript och tredjeparts-bibliotek
- *”Om jag ger dig mitt pass, skulle du kunna hämta paketet jag beställt, lämna det utanför min dörr och sen ringa mig?”*

Torrsims Demo

JavaScript

LINKÖPINGS UNIVERSITET

JavaScript (ES6)

- Mer eller mindre fullständigt stöd i alla moderna webbläsare
  - Chrome, Firefox, Safari, Opera, Edge, Brave, …
- Finns senare versioner men med mer begränsat stöd
- JavaScript körs (generellt sett) endast i en tråd
- Objektorienterad programmering stöds men kostar mer minne då varje objekt definierar sina egna funktioner (prototyping)

index.html

```html
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>A document</title>
    <script src="script.js">
  </head>
  <body>
    <h1>The Heading</h1>
    <p>The paragraph</p>
  <body>
</html>
```

script.js

```js
window.onload = () => {
  console.log('Page loaded!');
  let names = ['Bob', 'Dylan', 'Woodie',
               'Guthrie'];
  for(let name of names){
    hello(name);
  }
}

function hello(name){
  console.log(`Hello ${name}!`);
}
```

Viktigaste tillskotten i ES6

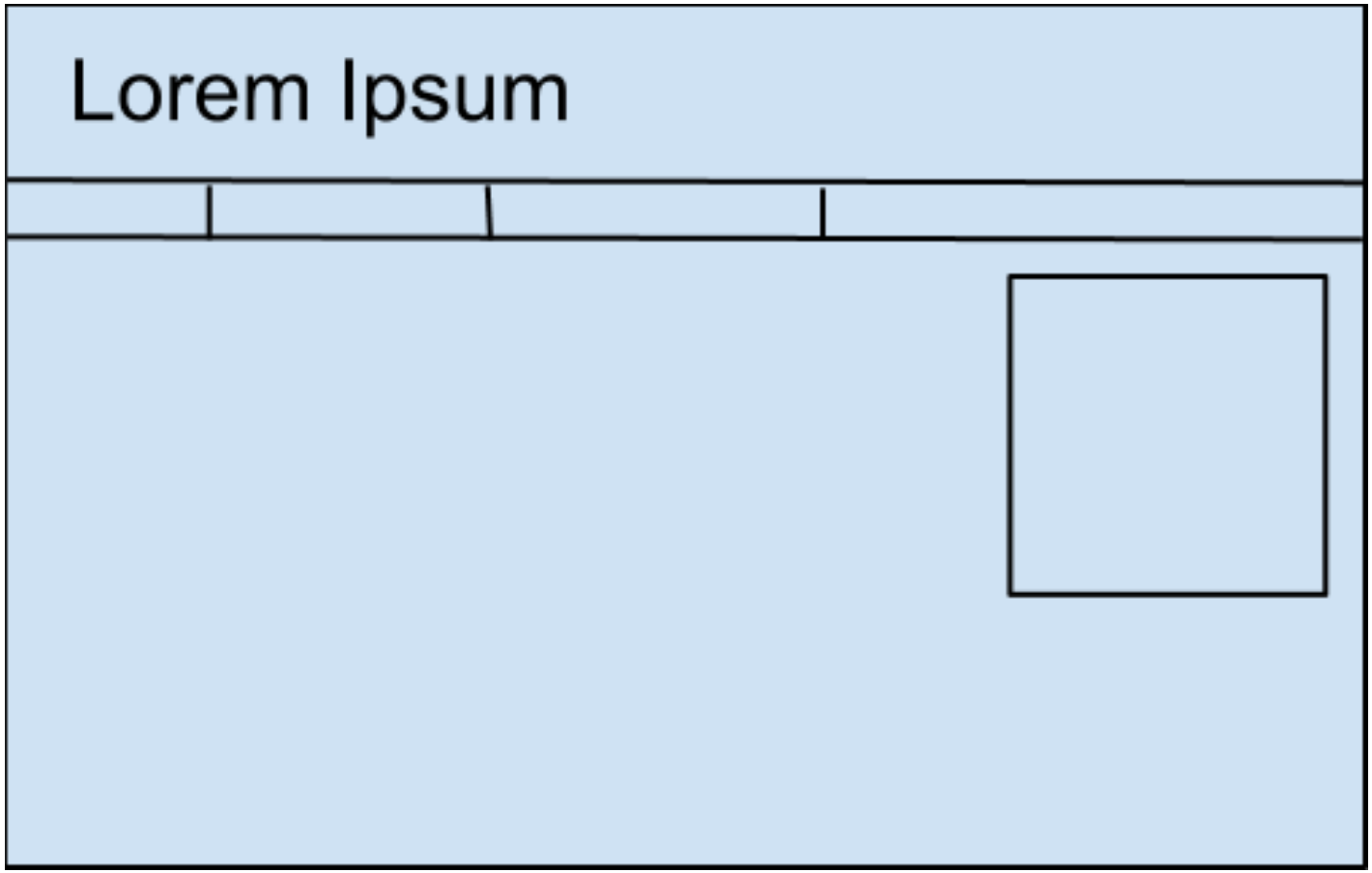
- arrow functions
- let + const
- iterators + for..of
- promises
- template strings
- classes
- modules

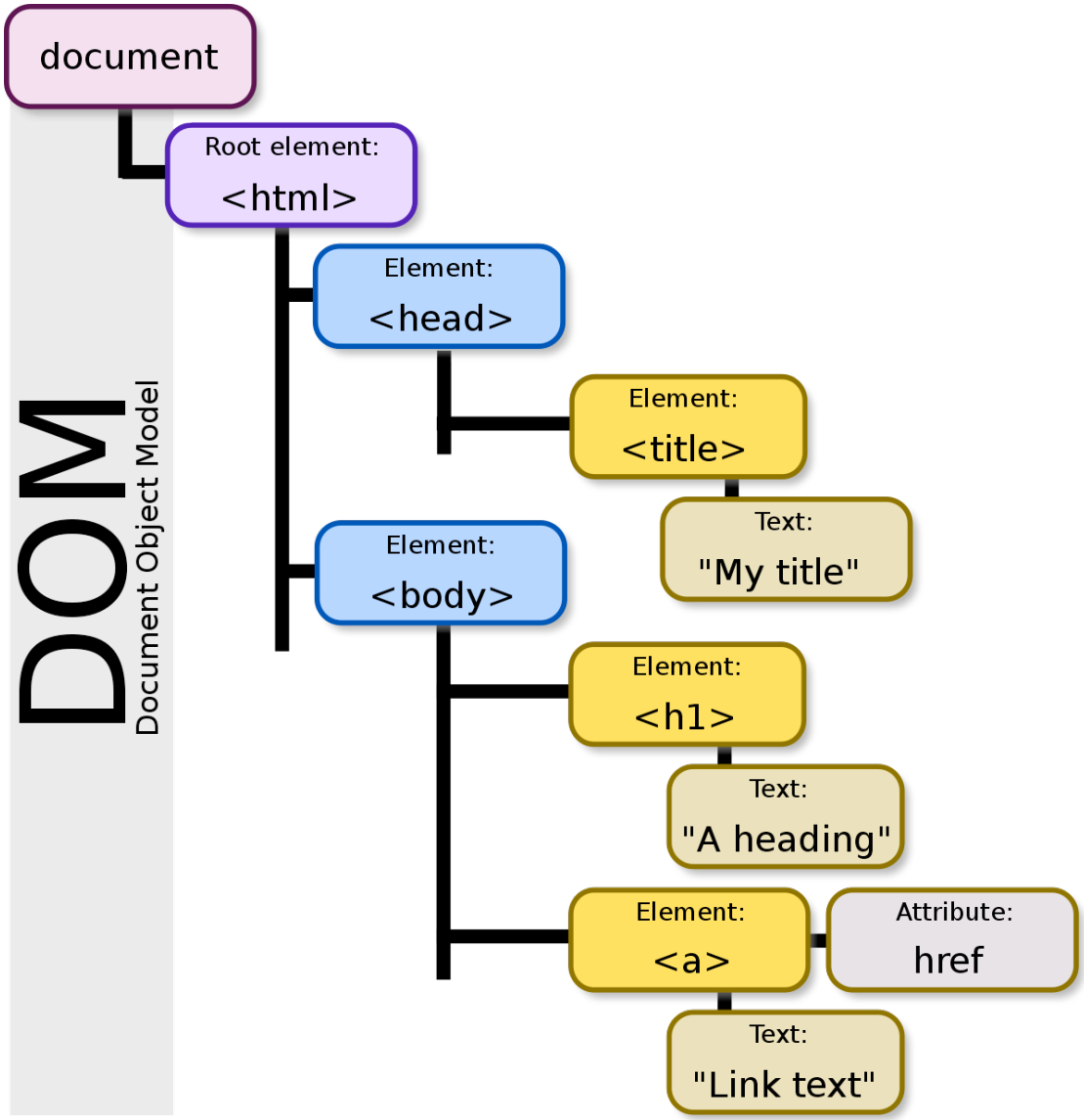
Syntax

```
// Deklarera en global variabel
let foo = 'Hello';

// En funktion
function helloWorld(repeat) {
  // Deklarera en lokal variabel
  let bar = 'World!';

  // Logga till konsol
  console.log(foo);

  let greeting = '';
  // for-loop
  for(let i=0; i<repeat; i++){
    // Slå samman strängarna och lägg till i greeting
    greeting += foo + ' ' + bar;
  }
  return greeting;
}

// funktionsanrop
helloWorld(12);
```

Document object model (DOM)

Hemsida





https://en.wikipedia.org/wiki/Document_Object_Model

Noder

- Varje element i ett HTML dokument är en nod i DOM-trädet (inklusive **<!-- kommentarer -->**)
- Det finns 12 olika typer av noder
- Element, TextNode och AttributeNode är de tre typer som generellt sett är intressanta för webbdesign

Navigera i DOM

- För att göra ändringar i DOM-trädet med JavaScript måste man kunna få tag i specifika element, ex:
  - document.getElementById('param') returnerar elementet med angivet ID
  - document.getElementByTagName('param') returnerar en lista med element med en viss tag
  - document.querySelector(<css selector>) returnerar det fösta elementet baserat på en CSS selector.
  - document.querySelectorAll(<css selector>) hämtar en lista av element baserat på en CSS selector.

Operationer på noder

- element.childNodes returnerar en lista med alla noder direkt under element i DOM-trädet.
- element.parentNode returnerar den nod som finns direkt ovanför element i DOM-trädet.
- element.nextSibling returnerar den nod som finns direkt till höger och på samma nivå som element i DOM-trädet.
- element.previousSibling returnerar den nod som finns direkt till vänster och på samma nivå som element i DOM-trädet.

Operationer på noder

- document.createElement('param') skapar ett nytt element baserat på en tag uttryckt som en sträng.
- document.createTextNode('param') skapar en ny TextNode från en sträng.
- element.appendChild(child) placerar det angivna elementet child sist i listan av noder direkt under element.
- element.removeChild(child) tar bort ett element från listan av noder direkt under det specificerade elementet. Noden måste finnas i listan över elementets barn.

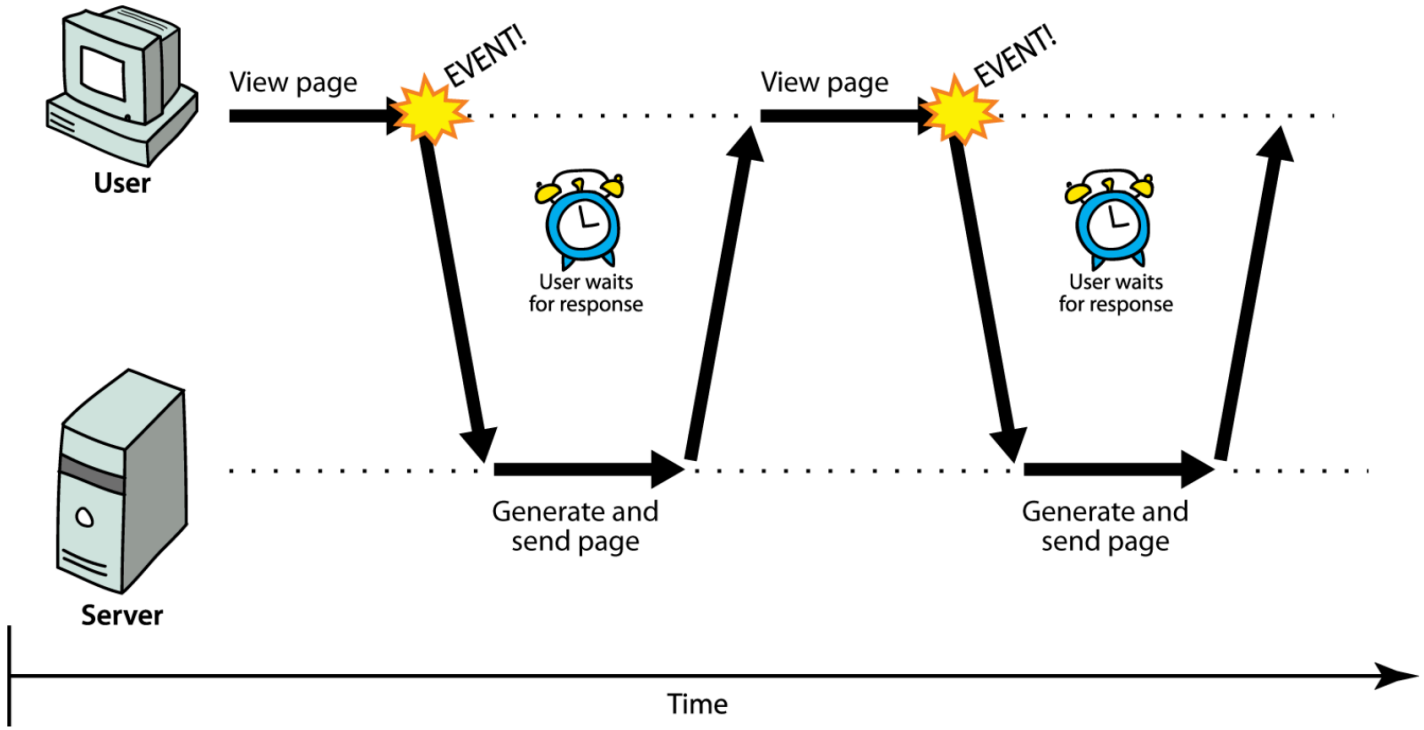
Livekodning

HTTP-anrop

Hämta och skicka data på webben

LINKÖPINGS UNIVERSITET

Synkrona anrop på webben

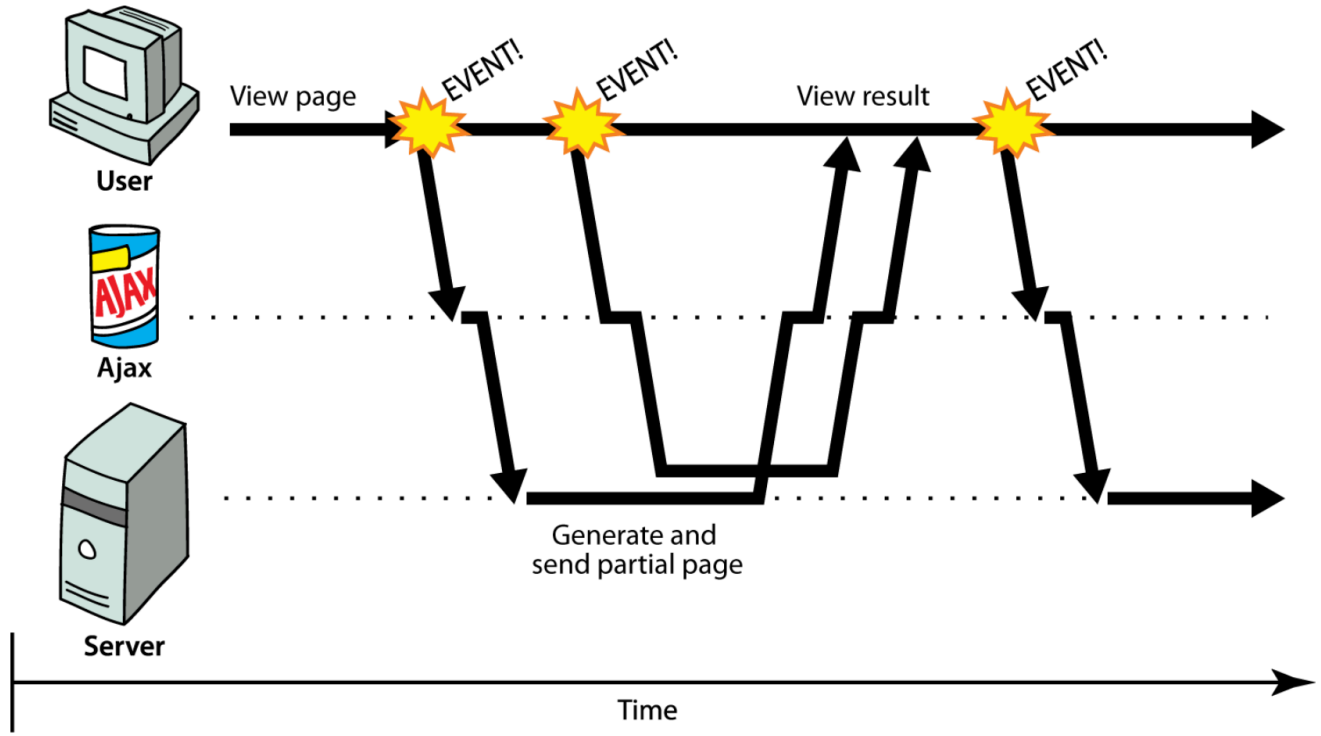


Användaren måste vänta på svar, och kan inte göra något under tiden.

Hela sidan uppdateras.

Bild: https://www.rose-hulman.edu/Class/csse/csse290-WebProgramming/201330/Slides/lecture21-ajax.shtml

Asynkrona anrop på webben



Användaren kan göra annat i väntan på svar från servern.

Endast de påverkade delarna av sidan ändras.

Bild: https://www.rose-hulman.edu/Class/csse/csse290-WebProgramming/201330/Slides/lecture21-ajax.shtml

AJAX

- *Asynchronous Javascript and XML*
- Möjliggör asynkrona anrop på webben via JavaScript
- Görs lite olika beroende på vilken webbläsare som används, men skillnaderna är idag mycket små
- Bibliotek som jQuery kan förenkla i vissa sammanhang men är inte nödvändiga
- Det som kommer tillbaka från servern är (oftast) JSON, XML, binära filer eller text

AJAX: Skicka request

```
0 UNSENT
1 OPENED
2 HEADERS_RECEIVED
3 LOADING
4 DONE
```

```
let xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = () => {
  if (this.readyState == 4 && this.status == 200) {
    let data = JSON.parse(this.responseText);
    console.log(data);
  }
};
xhttp.open('GET', 'https://gorest.co.in/public/v1/users', true);
xhttp.send();
```

”true” gör anropet asynkront

AJAX: Skicka request

```
function reqListener() {
  let data = JSON.parse(this.responseText);
  console.log(data);
}

function reqError(err) {
  console.log('Fetch Error :-S', err);
}

let oReq = new XMLHttpRequest();
oReq.onload = reqListener;
oReq.onerror = reqError;
oReq.open('GET', 'https://gorest.co.in/public/v1/users', true);
oReq.send();
```

HTTP-metoder

HTTP-metoder

- Kommunicerar önskad handling
- Vanligaste metoderna
  - **GET** – Be servern att returnera en viss resurs.
  - **HEAD** – Be servern att skicka information om en utpekad resurs (utan att skicka själva innehållet).
  - **POST** – Skicka information till servern som ändrar information på servern ELLER skicka information som är olämpligt att inkludera som en del av URL:en.
  - **PUT** – Lägg till eller uppdatera en resurs.
  - **DELETE** – Radera den utpekade resursen.
  - **OPTIONS** – Be servern att returnerar en lista över de HTTP-kommandon som servern stöder.

HTTP-metoder

- Kommunicerar önskad handling
- Vanligaste metoderna
  - **GET** – Be servern att returnera en viss resurs.
  - **HEAD** – Be servern att skicka information om en utpekad resurs (utan att skicka själva innehållet).
  - **POST** – Skicka information till servern som ändrar information på servern ELLER skicka information som är olämpligt att inkludera som en del av URL:en.
  - **PUT** – Lägg till eller uppdatera en resurs.
  - **DELETE** – Radera den utpekade resurs.
  - **OPTIONS** – Be servern att returnerar en lista över de HTTP-kommandon som servern stöder.

AJAX: Skicka med data med GET

```
let xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
  if (this.readyState == 4 && this.status == 200) {
    let data = JSON.parse(this.responseText);
    console.log(data);
  }
};
xhttp.open('GET','https://gorest.co.in/public/v1/users', true);
xhttp.send();
```

AJAX: Skicka med data POST

```
let xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
  if (this.readyState == 4 && this.status == 200) {
    let data = JSON.parse(this.responseText);
    console.log(data);
  }
};
xhttp.open('POST', 'https://gorest.co.in/public/v1/users', true);
xhttp.setRequestHeader('Content-type', 'application/json',
                       'Authorization', ‘Bearer <access token>');
xhttp.send('{"name": "John Doe", "gender": "male",
             "email": "john.doe@noone.com", "status": "active"}');
```

jQuery

```
let callback = (data) => {
  $(‘#content').text = data;
}

$.ajax({
  url: 'http://localhost:8888/',
   type: 'POST',
   data: {
     name: 'Marcus',
     filter: 'Employee'
   },
   success: callback
 });
```

jQuery behöver inte användas i någon av labbarna!

fetch(...)

AJAX med promises

Why promises or callback-hell

```
fetchResource(
  url,
  function (result) {
    // Do something with the result
    fetchResource(
      newUrl,
      function (result) {
        // Do something with the new result
        fetchResource(
          anotherUrl,
          function (result) {
            // Do something with the new result
          },
          failureCallback
        );
      },
      failureCallback
    );
  },
  failureCallback
);
```

```
fetchResource(url)
  .then(handleResult, failureCallback)
  .then(handleNewResult, failureCallback)
  .then(handleAnotherResult, failureCallback);
```

fetch: AJAX med promises

```
const url = 'https://gorest.co.in/public/v1/users'
fetch(url, { 'method': 'GET' })
    .then(resp => resp.json()) // parsa och gå vidare till nästa 'then'
    .then(data => {
        console.log(data);
    });
```

Alternativ med await

```
const url = 'https://gorest.co.in/public/v1/users'
let resp = await fetch(url, { 'method': 'GET' })
if (resp.ok) { // HTTP status 200-299
    let data = await resp.json();
    console.log(data);
}
```

CORS

Cross-Origin Resource Sharing

LINKÖPINGS UNIVERSITET

CORS

- Restriktioner på grund av säkerhetsskäl
    - “Cross-site scripting”
    - Risk för injections
    - Kan komma runt autentisering
- AJAX kräver att alla anrop görs till exakt samma domän som klienten kör!
    - Om er sida ligger på domänen http://example.com så får man endast anropa tjänster på http://example.com/...
- CORS används för att servern explicit ska ge rättigheter för vissa domäner

Cross-Site Scripting & CORS

- AJAX kräver att anropen görs till exakt samma domän som klienten kör!
    - Om er sida ligger på domänen http://example.com så får man endast anropa tjänster på den domänen
- Restriktionen finns av säkerhetsskäl
    - “Cross-site scripting”
    - Risk för injections
    - Kan komma runt autentisering
    - Intercepts
- CORS (Cross-Origin Resource Sharing) används för att servern explicit ska ge rättigheter för vissa domäner
- Vi kommer inte fördjupa oss i alla detaljer kring detta, utan fokusera på hur vi implementerar det

CORS

- *Cross-Origin Resource Sharing*
- Webbläsare använder i regel "same-origin policy"
- Innan GET/POST anropet skickas ett **OPTIONS**-anrop till servern
- Om rätt headers returneras så tillåter webbläsaren att man genomför GET/POST
- Ett relativt ”snyggt” sätt att göra det på som minimerar för mycket kodändringar i redan existerande system

CORS

- Vid ett cross domain-anrop skickar klienten först ett anrop med metoden OPTIONS.
- Header i svaret från servern beskriver vad som är tillåtet.
- Klienten ansvarar sedan för att bara skicka tillåtna requests
- Sker i regel helt automatiskt

CORS: Response headers

- På **serversidan** lägger man till vad och vilka domäner som ska tillåtas utifrån som skrivs som respons i headers
- Exempel:

```
let headers = {};
headers['Access-Control-Allow-Origin'] = '*';
headers['Access-Control-Allow-Methods'] = 'POST, GET, OPTIONS';
res.writeHead(200, headers);
res.end();
```

- Måste på läggas till i alla utgående "response" som man vill göra tillgängliga
- OBS: Vi väljer här att sätta '*' vilket tillåter alla domäner att anropa servern. I en produktionsmiljö specificerar man i regel domäner som ska få skicka anrop.

CORS: Response headers

- Hur kan man snabba upp och förenkla processen med headers?

```
if(req.method == 'OPTIONS'){
  let headers = {};
  headers['Access-Control-Allow-Origin'] = '*';
  headers['Access-Control-Allow-Methods'] = 'POST, GET, OPTIONS';
  res.writeHead(200, headers);
  res.end();
} else {
   // vid POST, GET, etc.
}
```

CORS: Response headers

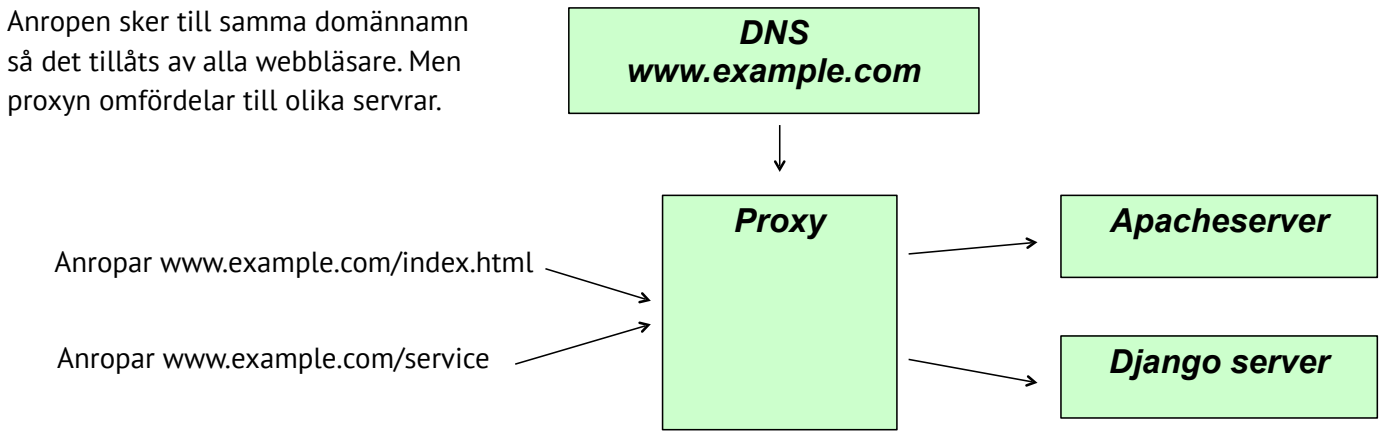
- I Express.js kan vi göra det på ett enkelt sätt med .**use(...)**
- .use(...) kallas på varje gång appen tar emot ett request, oavsett vilken route som används

```
app.use((req, res, next) => {
  res.header('Access-Control-Allow-Origin', '*');
  res.header('Access-Control-Allow-Headers', 'Origin, X-Requested-With,
             Content-Type, Accept');
  next();
});
```

- Finns libs som underlättar arbetet med CORS ytterligare
    - ...men det är en god idé att kontrollera att man inte öppnar upp för mycket!

Kan man klara sig utan CORS?

- Ja, men det blir mer komplicerat
- Man kan använda en “proxy” som hanterar alla anrop till domänen
- En proxy kan även ha andra fördelar så som ”caching” samtidigt som det fungerar med alla webbläsare
- Out-of-scope i denna kurs!



Hur kontrollerar vi att CORS fungerar?

- Anropa din server från en extern domän
    - Enkelt om din server ligger online
- Alternativ 1:
    - Skapa en fil som gör ett anrop mot localhost
    - Öppna sedan filen direkt i webbläsaren!
    - **OBS**: Fungerar inte alltid för alla webbläsare
- Alternativ 2:
    - Gör ett anrop med OPTIONS och kontrollera innehåll i headers

Projektet

LINKÖPINGS UNIVERSITET

Projektet

- Projektet genomförs i par (eller enskilt) enligt webreg
- För godkänt ska de grundläggande kraven vara uppfyllda
- För högre betyg finns ytterligare krav
- Deadlines:
    - **Onsdag 16:e oktober 13–17** (projektpresentationer)
    - **Fredag 19:e oktober** (inlämning av kod)
- Se hemsidan för andra redovisningsalternativ och kompletteringar

Projektet: Kortfattat

- En social webbplats
- Användare ska kunna registrera sig, logga in och logga ut på en personlig sida
- Data ska sparas i en databas (MongoDB)
- Lösenord får inte skickas som ren text
- Alla beslut kring interaktion och design ska vara **tydligt genomtänkta**
- Testning av backend ska ske med Mocha och Istanbul/C8

Projektet: Högre betyg

1. Möjlighet för vänner att chatta med varandra i realtid med HTML5 WebSockets och socket.io-plugin till Node.js
2. Använda ett klientramverk för att bygga din applikation (*)

- **Krav för betyg 4:** Grundläggande kraven + 1 **eller** 2
- **Krav för betyg 5:** Grundläggande kraven + 1 **och** 2

Etikuppgift

Etikuppgift

- Leta reda på ett så kallat *värdedrivet* företags etiska kod eller etiska policy
- Besvara frågorna i den anpassade versionen av Gibbs reflektionsmodell och skriv en reflektion på ca 1 sida (~500 ord)
- Inkludera eller länka till etisk kod eller policy
- Lägg upp dokumenten på git och skapa en tag
- Seminarieuppgiften görs enskilt. Synka med er labbpartner och inte samma företag

- Välj företag innan **onsdag 27:e september**
- Deadline för rapport: **onsdag 2:a oktober**
- Seminarium: **onsdag 2:a oktober 13:15–17:00**

LINKÖPINGS UNIVERSITET