Backend för Frontendutvecklare

Föreläsning 1

NORTH LINK

Vad ska vi lära oss?

- Node.js
- Skapa en webserver med express.js
- HTTP, REST, CRUD
- Säkerhetshot - hur de funkar och hur man hanterar dem
- Vad är en databas och dess roll i en webbapplikation
- Introduktion till att skapa och använda en databas

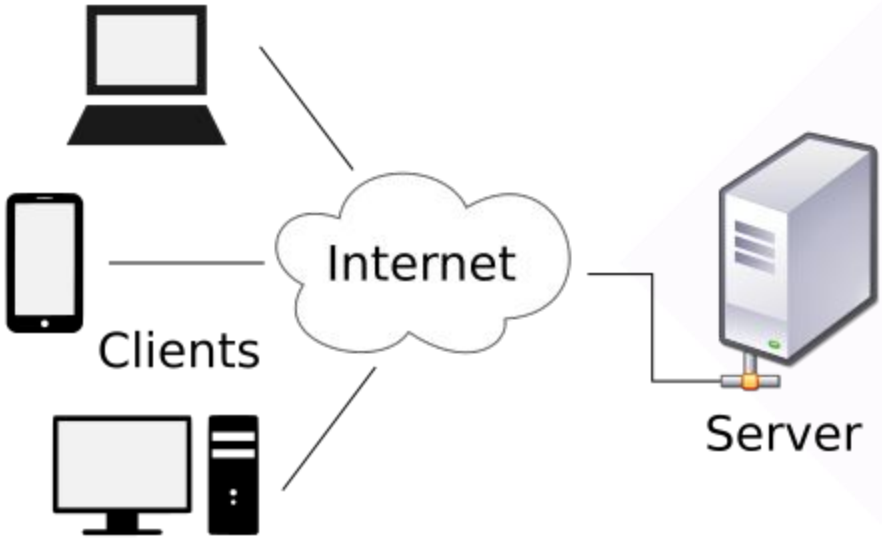
Om kursen

- Kursen är 9 veckor
- Kunskapskontrollen görs genom en obligatorisk laboration som ger IG/G
- I slutet av kursen har vi en inlämning som ger IG/G/VG
- Får att få godkänt på kursen måste man minst ha fått godkänt på både laborationen och inlämningen
- Det förutsätts att ni är bekväma i JavaScript vid det här laget - därför kommer tempot att vara högre än tidigare
- Det kommer också förväntas att ni kan läsa dokumentation själva

Vad är backend?

- Frontend kod körs på klienten i webbläsaren (alltid JavaScript)
- Backend kod körs på en eller flera servrar (i vilket språk som helst)
- Servern ansvarar för att svara på klientens frågor - t.ex. vad är det som finns på /home routen?
- Den koden som ligger till grund för all logik på servern kallas för backend

Webb



Node.js

- Node är en open-source JavaScript runtime som låter oss köra JavaScript kod utan en webbläsare.
- Node är designat för att bygga skalbara nätverksapplikationer
- Första releasen var 2009 - sen dess har användningen av runtimen exploderat och idag är det väldigt vanligt att folk använder node.js för att skapa servrar.
- https://nodejs.org/api/

Installera node.js

Ni har antagligen redan node.js installerat på er dator nu. Men om ni inte skulle ha det är detta er chans!

Det enklaste sättet (enligt mig) att hantera node är genom ett annat program som heter Node Version Manager (nvm). Det låter oss enkelt installera den exakta versionen av node vi vill använda, samt tillåter oss att ha flera versioner installerade samtidigt.

Installation och dokumentation för nvm hittar ni på https://github.com/nvm-sh/nvm

Om ni inte gillar nvm kan ni också ladda ner node.js från deras egna sida https://nodejs.org/en/download/

Hello world i node.js

Som ni förhoppningsvis har förstått har node.js ingenting med webbläsare att göra. Dvs för att köra node.js kod måste vi använda oss av node.js runtimen.

script.js

```
const myText = 'Hello world';

console.log(myText);
```

```
> node script.js // -> 'Hello world'
```

NORTH LINK

Node.js vs JS i webbläsaren

Många variabler som ni är vana vid när ni kodar frontend existerar inte i node.js miljön. T.ex `document` och `window` är variabler som är direkt relaterade till webbläsaren.

Node.js kommer med flera moduler som vi kan ladda in och använda direkt (utan att installera något extra). T.ex. finns det en filsystems modul som låter oss manipulera filer på datorn. Denna modulen heter `fs` och laddas in enkelt genom att skriva

```
const fs = require('fs');
```

Idag i node.js använder vi oss av `require/module.exports` syntaxen istället för `import/export`

Vanliga node.js moduler

- fs - För manipulering av filsystemet på datorn
- http - För att skapa webservrar och göra HTTP requests
- https - Samma som ovan
- os - Hjälpfunktioner för operativsystem specifika metoder
- path - Hjälpfunktioner för att hantera paths för filer
- querystring - Hjälpfunktioner för att hantera URL query strängar (t.ex. ?page=1)
- stream - För att jobba med strömmande data

Listan på alla moduler som finns inbyggda i node.js hittar ni i dokumentationen

https://nodejs.org/api/

Övning - Hello World med en webbserver

Följ den väldigt korta guiden på
https://nodejs.org/en/docs/guides/getting-started-guide/

Därefter går vi igenom den tillsammans.

Övning - Hello world med en fil

Återanvänd koden från förra exemplet. Men nu ska ni istället ladda in texten “hello world” från en fil på er dator.

Skapa en fil “hello.txt” och fyll den med texten “hello world”. Använd därefter `fs` modulen och metoden `readFile` för att läsa in texten i filen - och därefter skicka värdet som svar i responsen.

Testa att ändra texten i filen och verifiera att allting funkar.

Node.js callback pattern

Node.js använder sig av en standard för alla sina callbacks metoder som ser likadan ut överallt. Väldigt många tredje-parts bibliotek adapterar därefter den här standarden.

En callback tar oftast två argument. Första argumentet innehåller ett värde om ett fel skedde. Vi brukar kalla argumentet för `err`. Om inget fel skedde är `err` satt till `null`. Andra argumentet innehåller datan vi är intresserade av. När vi anropar en metod som tar en callback, så är callback argumentet också oftast sist.

```
fs.readFile('myfile.txt', (err, data) => {
    if (err) return handleError(err);
    console.log(data);
});
```

https://nodejs.org/en/knowledge/getting-started/control-flow/what-are-callbacks/

Async vs sync

Om ni tittade på dokumentationen för fs modulen är det möjligt att ni hittade en metod `readFileSync`. Denna metod kanske verkade enklare att använda eftersom den inte kräver någon callback. `readFileSync` är som namnet indikerar en synkron funktion, medans `readFile` är asynkron. I node.js är det vanligt att vi får tillgång till två metoder som gör samma sak, den enda skillnaden är att ena metoden är synkron och den andra asynkron.

Detta betyder INTE att vi ska använda den synkrona metoden för att den är “enklare”. Det finns nästan aldrig ett tillfälle då det är rekommenderat att använda den synkrona metoden (gör aldrig det om du inte är 100% på vad du håller på med).

Blocking

Anledningen till att vi aldrig ska använda en synkron metod i node.js (när en asynkron finns tillgänglig) är att i nästan alla fall så är den synkrona metoden blockerande.

När vi anropar en blockerande metod i node.js så kommer hela processen att vänta på att metoden blir färdig. Det här blir katastrof i t.ex. en webbserver där flera klienter kan göra anrop till en server samtidigt. Har man blockerande beteende i koden så betyder det att alla klienter behöver vänta på varandra. Den asynkrona metoden är däremot icke-blockerande och kommer tillåta processen att jobba med annat tills den underliggande processen som hanterar metoden är färdig.

Läs mer på https://nodejs.org/en/docs/guides/blocking-vs-non-blocking/

Express.js

Med node.js är det enkelt att skapa en HTTP server med `http` modulen. Men även om det är enkelt, kan det snabbt bli lite bökigt när vi ska börja hantera routing och data. Därför är det ofta att man använder ett ramverk för att göra det ännu enklare att hantera vanlig logik. Ett av de mest populära ramverken är `express.js`.

Vi kommer i resten av kursen att använda oss av express.

https://expressjs.com/

Övning - Hello world med express

Följ guiden på https://expressjs.com/en/starter/hello-world.html och skapa en enkel hello world webbserver (sista hello world jag lovar).

När du är färdig, läs https://expressjs.com/en/starter/basic-routing.html och därefter skumma igenom express dokumentationen.

Express routing

När vi gör ett anrop mot en express server kommer den att titta på url:en vi anropar och försöker matcha den mot en route (snarlikt till hur t.ex. react router funkar). Om den inte hittar en match skickar den 404 till klienten och avslutar anropet. Men om en matchande route hittas kommer funktionen vi har angett för routen att anropas.

```
app.get('/hello', (req, res) => {
    // I will be called on a GET request to localhost:8080/hello
});
app.listen(8080);
```

https://expressjs.com/en/guide/routing.html

Route parametrar

Express har stöd för parametrar i routsen.

Exempel:

```
app.get('/users/:userId/books/:bookId', function (req, res) {
  res.send(req.params)
});
```

```
Route path: /users/:userId/books/:bookId
Request URL: http://localhost:3000/users/34/books/8989
req.params: { "userId": "34", "bookId": "8989" }
```

NORTH LINK

Express route handlers

När en route matchar och vår funktion anropas får den alltid två argument - `request` och `response`.

`request` objektet innehåller all info om HTTP requestet (som kommer från klienten). Headers, querystrings, data osv.

`response` objektet är det objekt där vi sätter all information som ska finnas i responsen. Statuskoder, headers, data osv.

request

https://expressjs.com/en/4x/api.html#req

Viktiga attributer/metoder

- `req.body` - Data som skickades med anropet
- `req.method` - HTTP verbet som användes
- `req.params` - Route värdena vi har angett i routingen
- `req.query` - Värdena i query strängen
- `req.get(field)` - Returnerar ett HTTP header värde

response

https://expressjs.com/en/4x/api.html#res

Viktiga attributer/metoder

- `res.end()` - Avslutar ett svar
- `res.json(body)` - Skickar data formatterat som JSON. Sätter även rätt headers
- `res.redirect(path)` - Berättar för klienten att de ska försöka på en annan route
- `res.render(view)` - Render en vy (mer om detta senare)
- `res.send(body)` - Skickar data
- `res.status(code)` - Sätter statuskoden för svaret

Express - middlewares

I express är något som kallas för middlewares populärt att användas. En middleware är egentligen inte mer än en funktion som körs på requestet innan vår egna route hantering. Middlewarens roll är att hantera vanlig logik så att vi slipper göra det för varje route. T.ex. att parsa JSON data, hantera cookies, sessions osv.

Det finns mängder med tredjeparts-middlewares. Vi kommer att titta på ett flertal under kursens gång.

Att använda en middleware är enkelt.

```
app.use(middleware());
```

Express - inbyggda middlewares

Express har tre inbyggda middlewares som alla är väldigt användbara.

- `express.json` - Hanterar JSON data som skickas med en request om Content-Type är satt till `application/json`. Sparar datan i req.body
- `express.urlencoded` - Gör samma som ovan fast för `application/x-www-form-urlencoded` data (form data).
- `express.static` - Hjälper oss att hantera statiska filer (t.ex. bilder, css, frontend JS, osv.)

Express.json Exempel

Det enklaste sättet att ta emot JSON data i ett anrop är att använda `express.json`

Vi använder den enkelt genom att lägga till

```
app.use(express.json())
```

i vår kod. Det är viktigt att du lägger raden ovanför dina routes.

Statiska filer

Express har en inbyggd middleware (`express.static`) för att hantera statiska filer (HTML, CSS, frontend JS, bilder, osv). I våra projekt har vi ofta en mapp “public” som innehåller alla våra statiska filer (men den kan heta vad som helst). Ofta vill vi att våra användare ska kunna komma åt allting i denna mappen. Det gör vi enkelt med `express.static`.

```
app.use(express.static('public'));
```

Exempel

Se exempel nedan för en väldigt enkel webbserver som använder static middlewaren

https://github.com/LinkNorth/EC-backend/tree/master/examples/simple_webserver

Testa vår server

Oftast kopplar vi ihop vår backend med frontend kod med hjälp av AJAX. Men om man börjar med att bygga backenden är det lite jobbigt att inte kunna testa sina routes innan man får upp frontenden.

Vi behöver såklart ingen frontend för att använda vår server. Allt vi behöver är ett sätt att skapa en HTTP request. Det finns som vanligt mängder med sätt att göra detta på. Men det enklaste sättet för enklare requests är att använda programmet `curl` i vår terminal.

Curl

Med `curl` kan vi enkelt göra ett HTTP request direkt från vår terminal.

> curl http://www.google.com

Vanliga argument vi kan ange till curl är -H för att ange en header, -d för att skicka data och -v för att få mer information om svaret. Vi behöver också ange -X för att berätta vilket HTTP verb vi vill använda.

För att göra ett HTTP request med JSON data till vår server

```
curl localhost:3000/hello -XPOST -H 'Content-Type: application/json'
-d '{"value": "test"}' -v
```

Övning - JSON API

Skapa en express server som har tre routes

- POST /uppercase
- POST /lowercase
- POST /capitalize

Varje route tar emot JSON data i formatet `{"value": "text here"}`. Varje route returnerar ett JSON objekt som ser likadant ut - men värdet för value nyckeln ska vara transformerat baserat på routen. T.ex. `uppercase` routen returnerar värdet i uppercase osv. Om datan är i felaktigt format eller inte skickas med ska servern svara med 400.

Exempel - `POST /uppercase {value: "hello"} -> {value: "HELLO"}`

NORTH LINK

Skriva egna middlewares

Det är enkelt att skriva våra egna middlewares.

```
// Our own middleware
app.use(function(req, res, next) {
    console.log(req.method);
    next();
});

app.get('/Hello', (req, res) => res.send('Hello world!'));
```

https://expressjs.com/en/guide/writing-middleware.html

Övning - Skriva middlewares

Lägg till två middlewares i koden för förra uppgiften.

Första middlewaren ska logga alla anrop som sker till servern i formatet METHOD - URL

Andra middlewaren ska kolla att JSON datan är i korrekt format. Om den är felaktig ska requestet avslutas med 400.