Föreläsning 3

Mer om node och express

Hämta data från en server

När vi gör anrop om data från server är det väldigt sällan som servern kan ge oss all data som existerar. Vad man istället brukar få (eller implementera) är en pagination som ger oss ett visst antal resultat åt gången. När man behöver fler resultat får man göra ett nytt anrop till servern och be om nästa “sida” med resultat.

Det ser ofta ut som följande

```
GET /getData?page=1&size=20

GET /getData?page=2&size=20
```

Övning - lista länder

Du ska visa en lista med länder. Skapa en frontend som gör ett ajax anrop till en server och ber om en lista på länder. Backenden ska stödja pagination.

T.ex. /countries?page=1&size=20 ska ge oss de första 20 länderna i listan.

Använd data som finns på https://gist.github.com/marijn/396531/188caa065e3cd319fed7913ee3eecf5eec541918

Tips är att kopiera datan (det finns i JSON/JS format lite längre ner i konversationen) och lägga den i en fil som ni sedan läser in

Svårare: Lägg till filtrering (t.ex. att landets namn ska börja på en viss substräng)

Env variabler & config

När vi skapar en server har vi ofta värden som kan förändras. T.ex. portar vi använder, information relaterat till de databaser vi använder, m.m.

Ofta vill vi inte hårdkoda denna informationen i vår kod, då det är vanligt att de behöver ha annorlunda värden i produktions miljö vs utvecklings/test miljö.

Det finns två vanliga sätt att lösa detta problemet.

- Miljövariabler
- Config filer

Config filer

En config fil är precis vad det låter som - en fil som innehåller konfigurationsinformation. Det kan vara vilket filformat som helst (det som är enklast att hantera är att föredra). För node använder vi ofta JSON filer.

`require` funktionen hanterar json filer, därför för att ladda in en konfiguration från t.ex. `config.json` behöver vi enbart göra

```
const config = require('./config.json');
```

OBS: commita aldrig en config fil, om du vill spara strukturen på filen, gör en kopia och kalla den `config.sample.json` (som innehåller dummy variabler). Commita därefter enbart den.

Miljövariabler

Miljövariabler (environment variables) är ett annat vanligt sätt att specificera unika värden när vi startar vår server. En miljövariabel sätts i vår terminal när vi startar vårt program.

```
> NODE_ENV=production npm start
```

Variabeln kommer därefter att vara tillgänglig inne i vår kod genom objektet `process.env`

T.ex. `process.env['NODE_ENV']`

Övning - konfigurera port

I en av dina existerande servar, testa att konfigurera din port först med hjälp av miljövariabler, därefter med hjälp av en konfigurationsfil.

Cookies

Cookies är ett sätt för servern att be klienten att spara en liten bit data. Ofta används cookies för att komma ihåg saker som är relevant till inlogg. Men kan också användas för tracking, annan form av session hantering förutom login (t.ex. kundvagnar), m.m.

För att be en klient att spara en cookie behöver vi skicka en `Set-Cookie` header.

Därefter kan klienten välja att spara cookien, och i framtida requests skicka värdet i `Cookie` headern.

Express cookies

Express har inbyggda metoder för att enklare skapa cookie headers.

```
res.cookie(name, value [, options])
```

För att parsa cookies kan vi använda oss av cookie-parser middlewaren.

Därefter sparas värdena i `req.cookies`

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

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

Övning - Cookies

Skapa två `GET` routes - `setCookie` och `getCookie`

`setCookie` ska ta emot en querysträng med nyckeln “name”. Du ska därefter sätta värdet på en cookie “name” till vad som är angett i querysträngen.

`getCookie` ska returnera “Hello NAME” där NAME är det värde som är satt i cookien “name”.

Använd en webbläsare för att testa funktionaliteten.

Glöm inte cookie parsing middlewaren.

Express router

En Express router fungerar som en “mini-applikation”. Den låter oss gruppera routes och middlewares tillsammans. T.ex. säg att vi vill ha en middleware på några specifika routes, då kan vi skapa en router och applicera middlewaren med `router.use(middleware)`.

En router kan också användas för att slippa upprepa långa paths i vår routing.

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

Express router

Att skapa en express router är så här enkelt:

```
const router = express.Router();

router.get('/hello', (req, res) => res.send('Hello from router'));

app.use('/api', router);
```

GET /api/hello -> “Hello from router”

Node EventEmitter

En eventemitter har vi använt oss av tidigare (`addEventListener` jobbar med en typ av eventemitter).

I node.js finns det en `EventEmitter` class vi kan använda själva för att skapa vår egen eventemitter. Denna classen används även av de flesta objektet i node.js.

Rekommenderad läsning: https://nodejs.org/api/events.html#events_events

EventEmitter

```
const EventEmitter = require('events');

class MyEmitter extends EventEmitter {}

const myEmitter = new MyEmitter();
myEmitter.on('event', () => {
  console.log('an event occurred!');
});

myEmitter.emit('event');
```

EventEmitter - once

Om vi bara är intresserade av ett event en gång kan vi använda oss av `.once`

```
const myEmitter = new MyEmitter();
let m = 0;
myEmitter.once('event', () => {
  console.log(++m);
});
myEmitter.emit('event');
// Prints: 1
myEmitter.emit('event');
// Ignored
```

NORTH LINK

EventEmitter - req/res

Båda request och response objekten implementerar EventEmitter classen. Om vi ttar t.ex. på https://nodejs.org/api/http.html#http_class_http_serverresponse så kan vi se att responsen skickar två events “close” och “finish”.

Request objektet har liknande events.

Övning - middleware loggning avancerad

Skapa en middleware som för varje request loggar

METHOD - URL - STATUS - ResponseTime

Där ResponseTime är den tiden det tog för servern att svara på requestet.

Tips: titta på eventsen som response objektet skickar

Övning - REST API

Skapa ett API där vi kan ha två olika resurser. Det ska finnas routes för att skapa båda resurserna separat. Det ska också finnas ett sätt att koppla ihop de två resurserna.

Exempel

Resurserna är bilar, och personer (eller användare). Varje person kan ha en eller flera bilar - varje bil kan ha en eller flera personer kopplat till sig.

Vanligt löser man det med en POST /user/:userId/car/:carId

När vi hämtar ena resursen, ska vi få ut alla andra resurser som också är kopplade.