Föreläsning 5

Streams

Node Streams

En stream (ström) är en abstrakt definition på hur man jobbar med streaming data. Streaming data låter oss skicka mindre delar av vår data åt gången.

I node används konceptet streams lite överallt. Till exempel är både request och response objektet streams. Det är viktigt att man har en förståelse för hur en ström i node.js fungerar.

https://nodejs.org/api/stream.html#stream_stream

Stream typer

Det finns fyra olika bas typer av strömmar i node.js

- Writable - En ström vi kan skriva till (filer, HTTP response, konsolen)
- Readable - En ström vi kan läsa från (filer, HTTP request, konsolen)
- Duplex - En ström som är både Writable och Readable
- Transform - En duplex ström som transformerar data

Alla strömmar implementerar EventEmitter classen

Writable stream

En writable stream har två viktiga metoder `write()` och `end()`.

Till exempel är response objektet i express en writable stream. När vi skriver `res.end()` är det metoden på strömmen vi anropar - och inte något som express har lagt till.

`write()` låter oss skriva till strömmen - vi kan kalla på metoden hur många gånger vi vill.

`end()` avslutar strömmen - metoden ska enbart anropas en gång.

Writable exempel

```
app.get('/hello', (req, res) => {

    res.write('Hello');

    res.write(' World!');

    res.end()

});
```

Readable stream

En readable stream emittar två viktiga events - `data` och `end`

`data` eventet skickas när det finns ny data tillgänglig - datan kommer att vara första argumentet i eventhanteraren.

`end` eventet skickas när strömmen är avslutad.

Readable exempel

```
app.post('/hello', (req, res) => {
    let body = '';

    req.on('data', chunk => {
        body += chunk;
    });

    req.on('end', () => {
        console.log(body);
        res.end();
    });
});
```

Övning - JSON parsing middleware

Skriv din egna JSON parsing middleware.

Middlewaren ska kolla om Content-Typen är application/json - om content typen är satt och datan inte är giltig JSON ska du svara med 400.

stdin, stdout

I node har vi tillgång till två strömmar som kallas för “standard input” (stdin) och “standard output” (stdout).

stdin och stdout är ett sätt att låta oss läsa och skriva data till konsolen.

När vi skriver `console.log` sker det en `process.stdout.write` kommando i bakgrunden.

Genom att lyssna på stdout kan vi skicka meddelanden till vårt program genom konsolen.

stdin, stdout exempel

```
process.stdin.on('data', data => {

    process.stdout.write(data.toString().toUpperCase());

});
```

Process argument

När vi kör ett program med node kan vi få ut om man har angett några argument till programmet.

T.ex. node test.js foo 1 bar - har tre argument, “foo”, 1 och “bar”.

Vi kan komma åt dem genom att använda process.argv

Detta är en array med alla argument, notera att arrayen innehåller “node” samt namnet på vår fil också.

Övning - “curl”

Skriv ett program som lyssnar på process.stdin. När du skriver in en URL till programmet ska du göra ett HTTP anrop till url:en och logga statuskoden på svaret till konsolen.

Du kan antingen använda dig av process.argv för att göra ett enstaka request, alternativt kan du använda process.stdin för att lyssna på allt man skriver i konsolen.

fs streams

fs modulen har inbyggda metoder för att använda streams när vi jobbar med filer.

```
fs.createReadStream

fs.createWriteStream
```

Värd avancerad läsning

https://github.com/substack/stream-handbook

Delar är utdaterade, men förklaringen av koncepten är fortfarande bra.

Avancerad funktionalitet - pipes

I node går det att “pipa” strömmar. Detta innebär att vi skickar all info i en ström till en annan. Vi pipar en “readable” ström till en “writable”.

T.ex.

```
req.pipe(res)
```

Detta tar all info i requestet och skickar det tillbaka direkt i responsen.

```
fs.createReadStream('image.jpg').pipe(res);
```

Detta skickar en bild till responsen

Avancerade Övningar - Streams

I bash finns det många väldigt användbara program vi kan köra som t.ex.

- cp
- cat
- head

Många av dessa är förvånansvärt enkla att implementera med hjälp av node streams. Testa att återskapa deras funktionalitet med hjälp av node streams. Du behöver inte implementera allt som programmen stödjer.

Övning - cp

Skriv ett program som kopierar en fil

```
node cp.js foo.txt bar.txt
```

Tips - titta på createReadStream, createWriteStream och pipe

Övning - cat

Skriv ett program som läser flera filer och skriver innehållet i alla filer till konsolen

```
node cat.js first.txt second.html third.css
```

Övning - head

Skriv ett program head.js som läser av de n första raderna i en fil

```
node head.js myfile.txt 10
```

Fler övningar

Fler bash program du testa implementera

- mv (var försiktig när du tar bort filer)
- rm (var försiktig när du tar bort filer)
- ls
- touch
- mkdir
- wc

Om du inte vet vad ett kommando gör så är detta ett utmärkt tillfälle att bli mer bekväm med bash! Google är din bästa vän