Internetværk og Web-programmering
Web-apps:
Multi-page vs. Single Page

Brian Nielsen

Distributed, Embedded, Intelligent Systems

2 stil arter for web-applikationer

Klassiske ”web-sider”

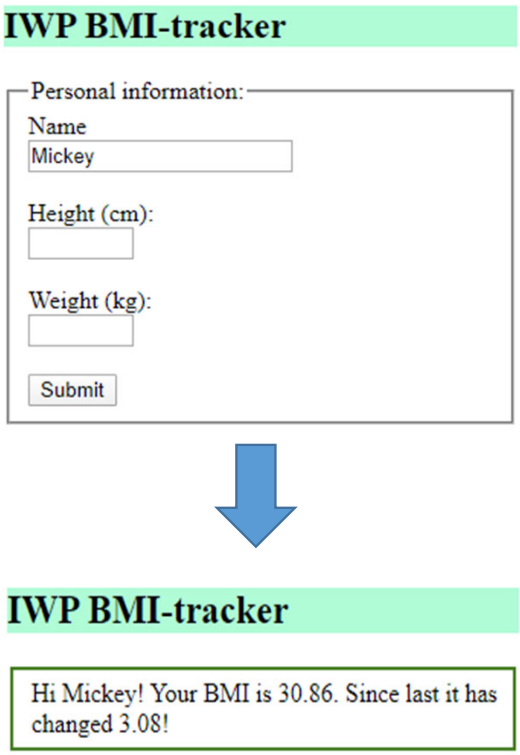
- Bruger udfylder og indsender ”formular”
- Klient og server kommunikerer via HTML
- Begrænset klient-side scripting
- Ingen interaktion med server uden et click på en ”submit” knap
- Server-side scripting (klassisk PHP) genererer dynamisk (beregninger og DB-opslag) en respons web-side, som nyt, helt, og selvstændigt HTML dokument
  - Flere ”tunge” dokumenter transporteres til/fra server
- ”Old-school” (men simpel og stadig arbejdshesten bag mange applikationer),
- God til søgemaskine optimering, bookmarks
- ”Fler-sidede applikationer”

Moderne web-applikationer

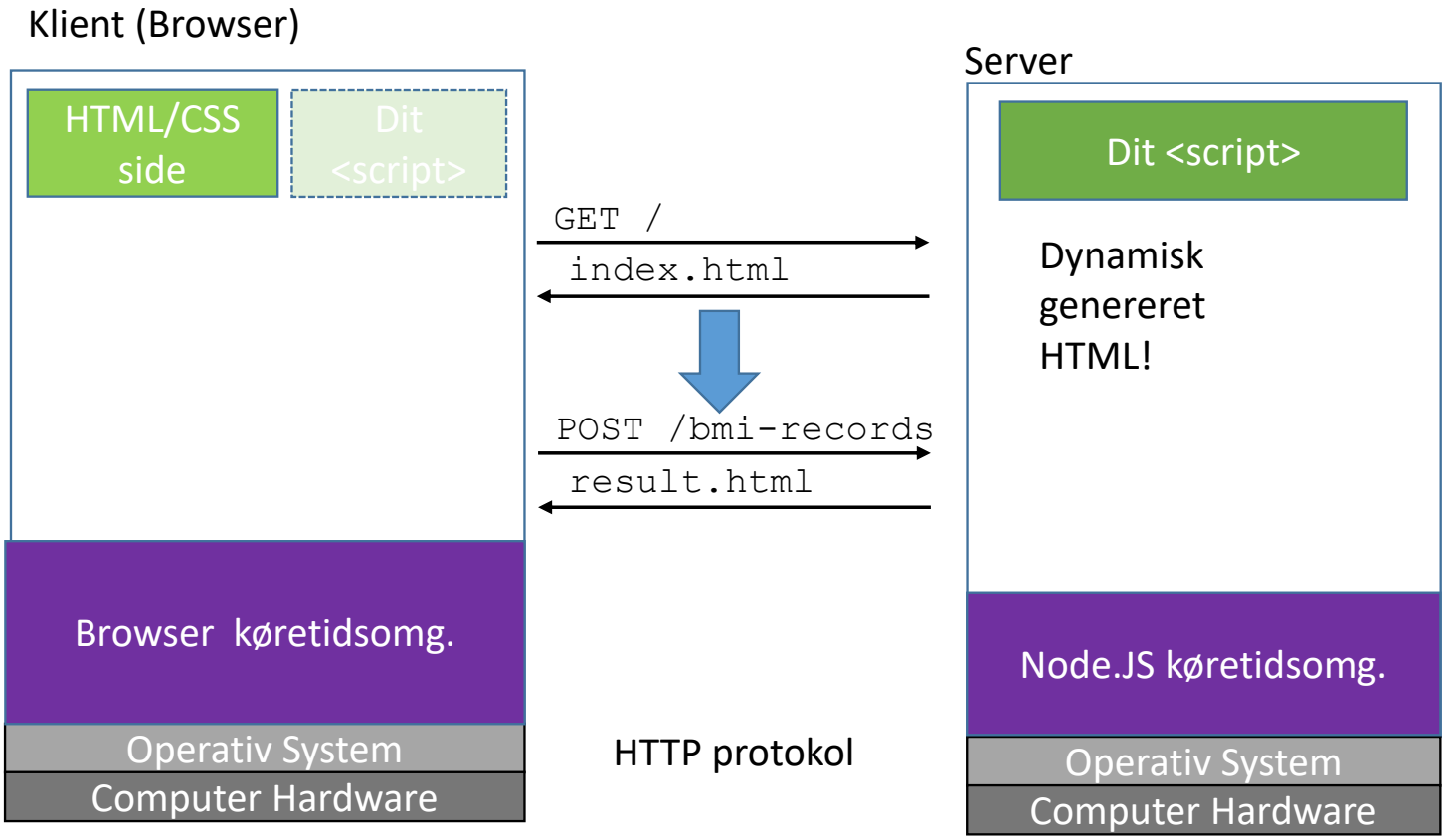
- En oplevelse af at arbejde med en ”rigtig” applikation
- Meget client-side scripting til bruger-interaktion
- Server interaktion foregår ofte i baggrunden via HTTP (REST) API og JSON (AJAX)
  - Flere, hurtigere kald
- Dynamisk omskrivelse på klient-side af applikationens HTML side vha DOM og events.
  - Kun den opdaterede del ændres.
- I det ekstreme ”Single Page Web-application”

Kan blandes til ”Hybride” varianter

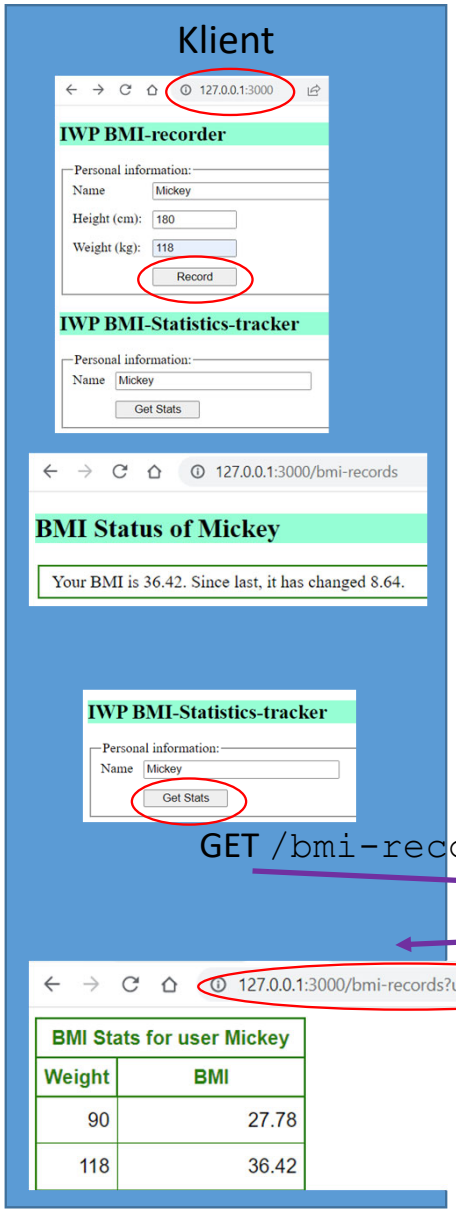
Klassiske Web-side applikationer



Nyt selvstændigt html dokument
Evt. navigér tilbage og start forfra!



Struktur af BMI-SITE applikation



GET /

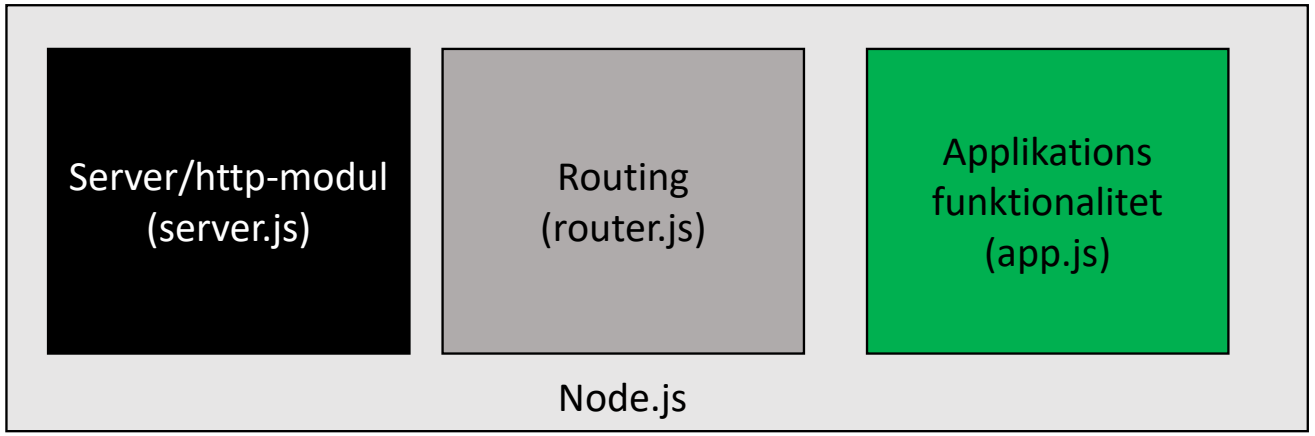
Front page html

POST /bmi-records

Html page

GET /bmi-records?userName=Mickey

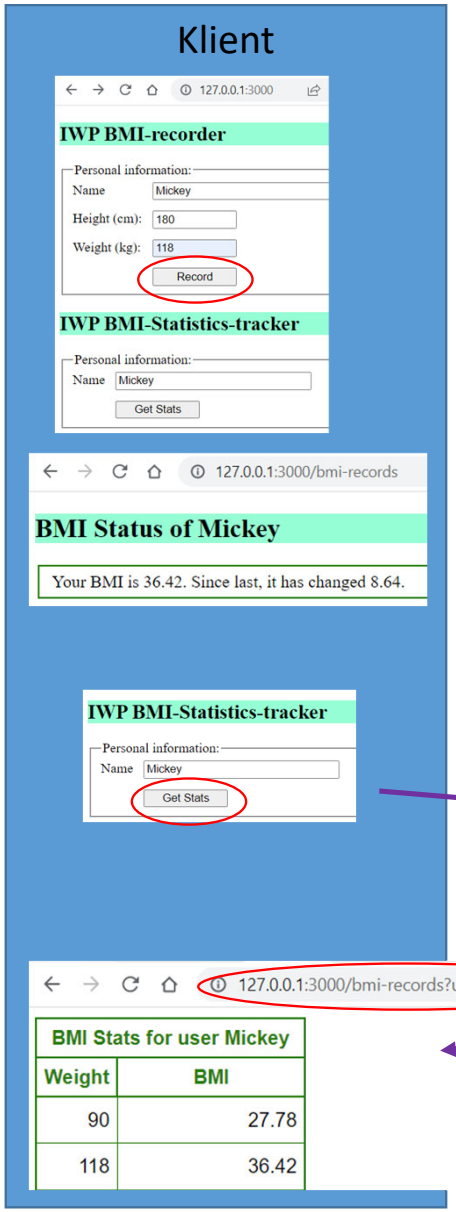
Html page

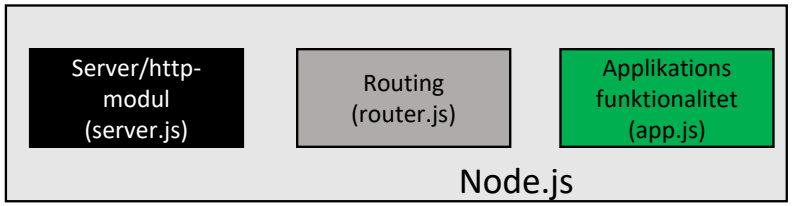


Overordnet logik:

- Browser udpeger applikation med URL: 127.0.0.1:3000 /
- Server leverer forside (bmi.html)
- Bruger angiver navn, højde, og vægt
- Klient (Browser)
  - klient side HTML validering af formular
  - formular data indsendes i POST (data i http body)
- Server
  - validerer formular input,
  - gemmer data i ”DB” (array)
  - Renderer side med resultat, og sender html tilbage i HTTP respons
- I statistik formular indsendes data via GET, userName i query string

Adfærd af BMI-SITE applikation





POST `/bmi-records`

html page

```
validatedData=validateBMIRecordForm
                        (formData)
status=recordBMI(validatedData)
html=renderHTMLBMIUpdatePage(status)
```

GET `/bmi-records?userName=Mickey`

html page

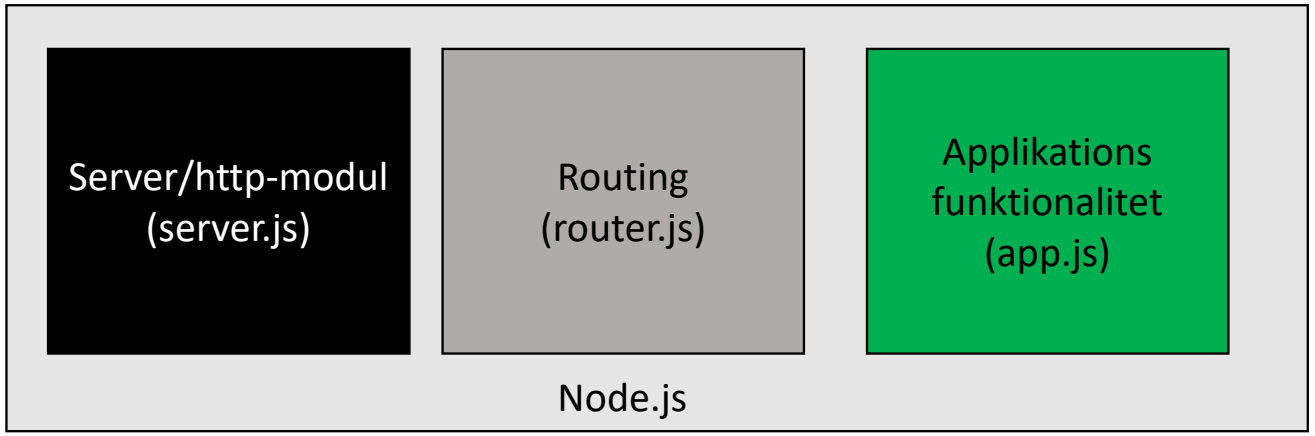
```
validatedFormData=validateBMIStatForm
                        (formData)
Html=renderHTMLBMIStatPage(
                        validatedFormData)
```

server.js og router.js er BLACK BOXE … indtil videre ☺

Fil-Struktur af applikationen

Fil-struktur

```
node/
  server.js
  app.js
  router.js
  PublicResources/
    css/
       simple.css
    js/
    html/
       help.html
       bmi.html
    img/
      bmi.png …
```



- Server har adgang til de scripts og filer den skal bruge i node kataloget.
- Klienter har kun adgang til filer i ”`PublicResources`” (håndhæves af server-modul)
  - ondsindede brugere kunne forsøge at lave requests til filer udenfor dette område! Fx `PublicResources/../../../passwords.txt`
  - (credentials er self. lagret i krypteret form)

Dynamisk serverside generering af HTML

- HTML sider ligger ikke nødvendigvis (sjældent) som statiske filer på server
  - Dog sommetider som ”skabeloner/templates”
- Script på serversiden genererer en streng som indeholder html-dokumentet; strengen sendes som svar i stedet for filen
  - Evt baseret på data udlæst fra en database.

Se eksempelkode

```
function renderHTMLBMIUpdatePage(bmiStatus){
  let page=renderHTMLHdr("BMI Status",["/css/simple.css"]);
  page+=`<body><section>
<h1> BMI Status of ${bmiStatus.userName} </h1>
<output>
Your BMI is ${bmiStatus.bmi}. Since last, it has changed ${bmiStatus.delta}.
</output>

</section></body>`;
 return page;
}
```

HTTP Beskeder

carriage return character

line-feed character

request line (GET, POST, HEAD commands)

header lines

carriage return, line feed at start of line indicates end of header lines

```
GET /index.html HTTP/1.1\r\n
Host: www-net.cs.umass.edu\r\n
User-Agent: Firefox/3.6.10\r\n
Accept: text/html,application/xhtml+xml\r\n
Accept-Language: en-us,en;q=0.5\r\n
Accept-Encoding: gzip,deflate\r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7\r\n
Keep-Alive: 115\r\n
Connection: keep-alive\r\n
\r\n
```

status line (protocol status code status phrase)

header lines

data, e.g., requested HTML file

```
HTTP/1.1 200 OK\r\n
Date: Sun, 26 Sep 2010 20:09:20 GMT\r\n
Server: Apache/2.0.52 (CentOS)\r\n
Last-Modified: Tue, 30 Oct 2007 17:00:02
   GMT\r\n
ETag: "17dc6-a5c-bf716880"\r\n
Accept-Ranges: bytes\r\n
Content-Length: 2652\r\n
Keep-Alive: timeout=10, max=100\r\n
Connection: Keep-Alive\r\n
Content-Type: text/html; charset=ISO-8859-
   1\r\n
\r\n
data data data data data ...
```

URL og URLSearchParams Objekterne [DF11.9]

```
function SearchParamsDemo(){
  let url = new URL("http://example.com");
  url.pathname="bmi-records";
  console.log(url.toString());       //=>http://example.com/bmi-records
  let params = new URLSearchParams();
  params.append("userName", "Mickey");
  params.append("weight", "100");
  console.log(params.toString());    // =>userName=Mickey&weight=100
  url.search = params;
  console.log(url.href);             // =>http://example.com/bmi-records?userName=Mickey&weight=100


  let url2= new URL("http://example.com/bmi-records?userName=Mickey&weight=100");
  let sp=new URLSearchParams(url2.search); //or simply sp=url2.searchParams;

  console.log ( sp.has("weight") );//=> true
  console.log ( sp.has("sex") );   //=> false
  console.log ( sp.get("weight") );//=> 100
}
```

HTTP/1.1 Metoder (a.k.a ”verbs”)

- **GET:**
  - Anmoder om overførsel af en repræsentation of den ønskede ressource
  - ”Læsning”
- **POST:**
  - Udfør en resource-specifik behandling på den ønskede ressource
  - ”Ændring”
    - Fx, tilføje data til ressourcen (fx indtastet i en “HTML form”)
- **PUT:**
  - Oprette (eller erstatte) tilstanden på den ønskede ressource) i sit hele så den svarer til den medsendte repræsentation
- **DELETE**
  - Sletter ressourcen (eller fjerner forbindelsen imellem URL navn og ressourcen)
- HEAD, PATCH, CONNECT, OPTIONS, TRACE

Når serveren modtager et request med en given metode kalder den en **funktion** som ”du” eller (web-server programmøren) har lavet!

”Spilleregler” for web-arkitekturen forudsætter at funktionen (som udføres på server siden) skal respektere hensigten bag HTTP metoderne

- Fx må GET ikke ændre ressourcen.
- Caching, forudindlæsning (”pre-fetching”)

Sikre metoder: udførslen af **funktionen** må ikke ”skade” ressourcen, eller give unormal stor belastning på server

- Klient kan gentage dem
- GET, HEAD, OPTIONS, TRACE skal programmeres så de er sikre
- Et PUT umiddelbart efterfulgt af et GET skal give den værdi der netop er oprettet

- Begrebet ”Idempotente” operationer introduceres senere

DIN KODE SKAL RESPEKTERE DISSE