Endpoint: slutningen på url skal være tilsvarende noget på serveren.
Kigge på server: npx json-server -watch "data/dbjson" --port "8000"

- udskift data/dbjson og port efter behov.

Eksempler på servere (fil kollektion):

Readme.txt filerne er din ven, de forklarer godt hvad der sker.

Server applikation:

Klient rendering.
(server1 fil)
data:

- Server data/database
  - db.json indeholder en kollektion af data (start én).
  - Kan udvides til at indeholde flere kollektioner.
    - Dette er gjort efterfølgende på linje 63 ved start ved komma..
  - Fordel med json - det minder om java.

node modules indeholder:

- eksterne moduler som projektet skal bruge.

public mappe (static):

- App.js, linje 13 laver public til static. VIGTIGT!
  - Middleware
  - Skal ligge efter statement på linje 5, const app = express();
- Indeholder offentligt tilgængelige ting, fx css eller billeder.

app.js

- endpoint kan ses på linje 26.

- linje 77, post (arrow funktion)
  - Får action fra post.
  - req (request) - data der bliver sendt med når man trykker create.
  - res (responce) - hvad der skal sendes tilbage til klient.
  - Alt indhold i funktionen ligger i linje 78 til 92
    - Linje 79 kigger på den tilkoblede tabe i create.html og leder efter “title” (name=”title” i tabel)
  - Linje 83 laver fetch til 'http://localhost:8000/blogs/' (port og kollektionen (kigger i givende port))
    - fetch parameter i rækkefølge:
      - Endpoint
      - POST, man skriver i.
      - headers - content der kommer
      - body, parser json (blog) (laver javascript til json)
    - fetch returnerer et promise (rejected, pending, result)

- then (arrow funktion)- kan kun udføres hvis fetch giver result (linje 87)
    - handling inde i fetch laver kun en console.log, men kan også lave andet hvis nødvendigt.
    - En god idé ville være at lave en redirect til en side der ville give mening.
- hvis fetch får reject, så går den i catch og udfører dennes handling (linje 91)
  - npx kommando fortæller man hvilken port man bruger
- Linje 89 slutter forbindelse - res.end();
- Slutter på linje 94
- blog er javascript objekt

- linje 44 kører funktion til linje 66

- linje 66 indeholder nyt endpoint
    - fetch på et specifikt givende id
    - linje 71 laver json object om til javascript og returnere et promise som der bliver checket på.
    - så kører den linje 72 hvis der er result.
    - hvis nogle fejl fanger den catch i linje 73.

- linje 97 indeholder nyt endpoint.
    - Den modtager et 'id' i dette endpoint.
    - denne funktion bruges til at slette et element med et givende 'id'
    - har en .then og .catch for at se/fange fejl

    - Callback, spørgsmål, uddybes?

code.js

- linje 9 laver en udskrivning i html som en tabel `<table>`
    - Kører foreach hvor den kører igennem ovenstående blog.
- linje 54 (single thread)
    - hvis man ikke laver en await i linje 55, så kører den forkert. Det gør at man kører funktionen igennem, før at man kører andet der skal bruge den information man går efter.
    - Man kan kun lave await i async.
- linje 45 er async.
    - den kalder på port 3000
        - fungerer med app.js linje 57

index.html

- script står øverst, men kan også ligges i bunden.
    - Dette er grundet onload på linje 8, hvor den siger at man først kører "udskrivAlle();"
- placeringen af script fortæller om den bliver kørt først eller sidst, alt efter om den ligger først eller sidst i html filen.
    - Med fordel kan man altid smide den til sidst, så der ikke sker nogle fejl.

Server.ejs:

Server rendering.
Ved rendering bliver det hele lavet om til html kode.

Overordnet mappestruktur:

- Data (json)
- Node modules
- Public (skal bruges)
- Views

app.js:

- linje 1-17 skal bruges som standard i starten, når der er tale om ejs
- linje 7 app.set('view engine','ejs'); fortæller at vi arbejder med ejs og view engine
- Husk linje 10 for at parse til body.

- Linje 30 starter man rendering af index filen (arrow funktion)
    - Dette er noget man gør med alle sider for at rendere.
    - Endpoint
    - Man bruger denne form for kode i sammenhæng med at det er ejs filer.
- Linje 39 indeholder nyt endpoint
    - fetch på 'http://localhost:8000/blogs'
    - linje 41 laver json object om til javascript og returnere et promise som der bliver checket på.
    - så kører den linje 42 hvis der er result.
    - hvis nogle fejl fanger den catch i linje 43.

Klient vs server:

- Klient har “ansvar” for at lave siden.
    - Har adgang til at rette lokalt i koden.
- Server har “ansvar” for at lave siden.
    - Har ikke adgang til at rette lokalt i koden.
    - Mere sikker ved rendering
    - Mulighed for at checke mere data igennem hurtigere. Fx API

Hastighed på de forskellige er anderledes alt efter hvilken opgave den skal udføre.

Potentielt spørgsmål: Hvad er fordele og ulemper på klient og server.

Html gloser:
Form:

- formular til indsamling af data.
    - submit (endpoint) bruges til indsendelse af formularens data.

Single thread er hurtigere en multi. Non blocking (asynkron).
Asynkron skal bruge .then og .catch for at fejlfinde i det.
Ejs = embedded javascript

API'er

En API er ikke en server!

ServerApi1

Start json server op for at det kan fungere.

Ingen brug af html.
Den bliver kun brugt til at lægge data ind og trække ud.
Den er mellemmanden mellem klient og databasen.
Kan kun bruges hvis json server er startet.

I denne fil skal vi betragte db.json som vores API.

Klient går altid på port 3000.

app.js er api'en i dette projekt.
Henter data gennem json.

Client1 fil:

Denne fil viser et godt eksempel på hvordan det fungerer bedre. Beskrivelse om hvad der sker står i filens Readme.txt

—----------

Sæt at man bruger link til dyn i index. (linje 23).
Derefter kører den skrevne script i head (linje 4), og så kører den onload på uskrivAlle1 (async) fra scriptfilen (linje 35).

—----------

API vs server:

- Api sender kun data
- Server kan sende filer, data, funktioner osv.