



World

Anders Dall
Simon Lervad
Tinna María Richter
Rikke Domino Isaksen

Erhvervsakademi Kolding, IBA
Webudvikling, 2. semester

Niels Müller Larsen
Mile Ninkovic

Indholdfortengelse

Indledning.....	3
Problemformulering	4
Metodeovervejelser	4
Gantt	4
Trello	4
Git	5
Wireframe	5
Database	6
Desktop research.....	7
Analyse.....	7
Konstruktion & evaluering	9
Node.js.....	9
MongoDB.....	10
Node.js & MongoDB	12
Konklusion	18
Litteraturliste	19
Bilag.....	21

Indledning

Der skal laves en samlet database, som indeholder informationer omkring jordens verdensdele. I databasen finder man 3 collections; som består af lande, byer og sprog. Hver collection indeholder objekter, som hver især indeholder informationer vedrørende det land, sprog eller by, som man skriver ind. Databasen er koblet sammen med en server, som giver en hjemmeside mulighed for at læse, skrive samt redigere og opdatere data i databasen. Hele projektet er udarbejdet vha. værktøjerne; MongoDB og Node.js, der sammen giver en god mulighed for at oprette og vedligeholde servere og databaser.

Problemformulering

Hvordan kan man lave en dynamisk hjemmeside, hvor data om verdens kontinenter skal fremvises og redigeres vha. en database?

Metodeovervejelser

Gantt

Ved at anvende Gantt, som er plandreven, opdeler man projektets opgaver i mindre dele og laver en tidsestimering for skabe et overblik over de opgaver, der skal laves (Busch A., 2015, s. 48 - 49) (Busch 2015, s. 48 - 49)([Bilag I - Gantt](#)).

Trello

For at gøre arbejdsprocessen mere iterativ er der anvendt Trello, som giver mulighed for at gå tilbage i arbejdsprocessen. Trello er en god og fleksibel måde at visualisere og planlægge sine projekter. Den giver bedre mulighed for at organisere projekter eller personlige opgaver

Der er valgt at anvende Trello ([Bilag II - Trello](#)), som et slags "SCRUM board". På den måde giver det et overblik over, hvilke opgaver der skal udføres og arbejdes på. For at skabe sammenhæng mellem Gantt kortets opgaver, er det de samme opgaver, som er skrevet ind, på den måde bliver der også sikret, at der ikke er overset noget (Trello n.d.)(Okholm 2018).

Git

For at gøre opdelingen af opgaverne overskuelige, blev der arbejdet i Git. Ved at anvende git er alt kodning gemt på GitHub, som gruppemedlemmerne har adgang til. På den måde kan man samarbejde omkring kodningen af produktet og undgå at komme til at arbejde med forskellige versioner.

Wireframe

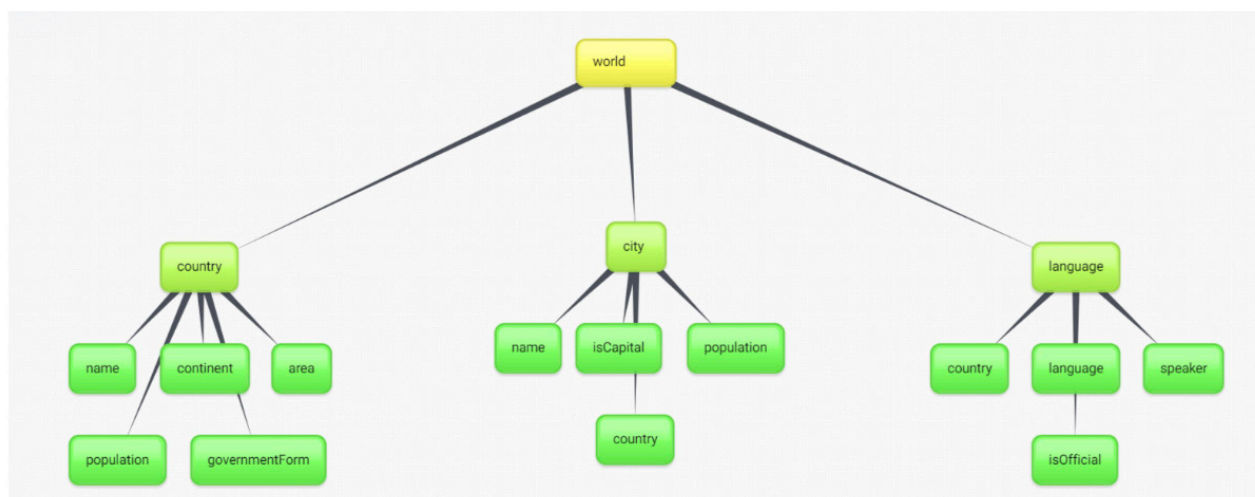
For at visualisere hjemmesidens opbygning blev der lavet en wireframe, hvilket giver en forståelse for, hvor elementerne skal placeres samt få det samlede indtryk af hjemmesiden

(toolmaster n.d.) Wireframen blev holdt i low fidelity og uden farver, da det er selve indholdet der, skal fokuseres på og ikke designet ([Bilag III - wireframe](#)).

Database

På hjemmesiden skal der være en database. I databasen vi har kaldt "world", er der oprettet 3 collections; country, city og language. Da databasen kan virke uoverskuelig, er der valgt at visualisere databasens indhold vha. et mindmap, som forklarer, hvilke felter der ligger under hver collections.

Det er ikke nok at oprette en database i systemet. Den skal også have indhold, ellers giver den en fejlmelding, når man prøver at køre den på localhost. Derfor er der lavet et dokument med collections og felter, der kunne kopieres ind i databasen, så den ikke var tom.



Desktop research

For at kunne udføre opgaven skulle der søges efter informationer omkring de forskellige lande. Det blev gjort gennem desktop research, hvor man her får nogle informationer på en hurtig og nem måde. Når man arbejder med desktop research arbejder man med sekundær data, hvilket vil sige data, som i forvejen har været samlet. Her er der fundet sider, som indeholder informationer omkring de forskellige lande, byer og sprog. Hvilket er anvendt til at indsætte i databasen. Ved at finde de rigtige sider har det sparet en del tid, da man ikke skulle ind på flere forskellige sider (International n.d.).

Analyse

I forbindelse med projektet blev der anvendt Gantt, som er plandreven, men arbejdsprocessen er iterativ. Hvilket har gjort, at projektet kunne opdeles i mindre bider, samtidig med at man har kunne gå tilbage i processen. På den måde har man kunne lave opgaverne, teste om det virkede for evt. at gå tilbage igen, hvis noget gik galt. De forskellige opgaver har været fordelt mellem gruppemedlemmerne, som har arbejdet med dem. Ved at anvende Git har alle haft mulighed for at arbejde med koden og foretage ændringer, der har dog været nogle udfordringer med Git. Hvis flere gruppemedlemmer har arbejdet i den samme fil eller har haft en forældet udgave, opstod der merge konflikter, som skulle rettes. Derudover har der været problemer med, at man hverken kunne push eller pull, hvilket dog blev løst ved at slette projektet mappen og clone den på ny.

Før hjemmesiden overhovedet har kunnet blive kodet, er der lavet en wireframe. Wireframe laves så alle har været enige i, hvordan hjemmesiden skal se ud, samt skabe et overblik over, hvordan den skulle kodes.

I projektet er der anvendt MongoDB med forskellige collections, som skal anvendes på siden. Når man anvender MongoDB bliver databasens indhold gemt lokalt, hvis man ville have dataen ude på en server kan man med fordel anvende Node.js, som giver muligheden at vise databasens indhold. Ved at skrive de retter koder kan man også få Node.js til at ændre og tilføje dataen.

Konstruktion & evaluering

Node.js

Node.js er server-side JavaScript miljø. Fordelen ved Node.js er at man kan kode både server- og klient-side, hvilket gør den meget fleksibel. Når man arbejder med Node.js er det oftest i forbindelse med web applikationer/hjemmesider, som giver mulighed for at lave dynamisk indhold på en side. Node.js er god til REST / JSON-programmerings interfaces, samt databaser eller webservices, fordi den kan snakke med forskellige systemer.

Når man arbejder med Node.js og databaser, er den mest anvendte database, MongoDB, som er open-source og tillader server-side data validation (W3School n.d.)(Capan n.d.). I projektet er der anvendt Node.js, fordi der skulle laves en dynamisk hjemmeside, hvor der blev arbejdet med databaser.

Der var i starten problemer med, hvordan man skulle skrive de forskellige elementer ind i Node.js, fordi Node.js ikke skal anvende en browser. Derfor var det svært at manipulere DOM elementerne direkte. Løsningen var at skrive de forskellige informationer ind vha. variabler og HTML elementer.

```
let dynamic = "";
for (var i = 0; i < obj.length; i++) {
  let heading = `<h2>${obj[i].continent}</h2>\n`;
  let name = `<p>Name: ${obj[i].name}</p>\n`;
  let continent = `<p>Continent: ${obj[i].continent}</p>\n`;
  let area = `<p>Area: ${obj[i].area}</p>\n`;
  let population = `<p>Population: ${obj[i].population}</p>\n`;
  let governmentForm = `<p>The government form: ${obj[i].governmentForm}</p>\n`;
  dynamic += heading + name + continent + area + population + governmentForm;
}
```

MongoDB

MongoDB er en document database, hvilket vil sige, at den gemmer data i filer, som minder meget om JSON. Det gør det muligt at sende data mellem forskellige systemer (Zetcode n.d.) (w3school n.d.)

MongoDB er anvendt, fordi den giver mulighed for at ligge lokalt på ens computer og det kan arbejde sammen med Node.js.

I starten var der udfordringer med at læse hele databasen på en gang. Databasen består af tre "collections": country, city og language. Når der blev skrevet nye data ind i databasen, skulle den sammenligne dataen fra POST med det data, som allerede var i databasen.

På den måde skulle man være i stand til at tilføje et nyt land.

Hvis landet allerede eksisterer, skal information opdateres.

Samtidig skulle man kunne tilføje nye byer, men kun hvis landet er oprettet, ellers skal brugeren underrettes og slå fejl. Det samme med language, hvis landet hvor man taler sproget ikke eksistere, så skal den ikke skrives ind i databasen.

Problemet blev løst ved at lave en stor funktion, hvor der blev oprettet tre tomme variabler som objekter. Her blev collections læst igennem en af gangen, og dataen fra hver collection bliver sat ind i hver deres variabel. Funktionen blev afsluttet, og dataen kunne sendes med over i "res.write()" funktion.

Det er løst på følgende måde:


```

continents(req, res, asset) {
  const mongo = require('mongodb');
  const dbname = "world";
  const constr = `mongodb://localhost:27017`;
  asset = asset.substring(1);

  mongo.connect(constr, { useNewUrlParser: true, useUnifiedTopology: true
  },function (error, con) {
    if (error) {
      throw error;
    }
    const db = con.db(dbname); // make dbname the
    current db
    /* Retrieve,
     * reads cities from the database
     */
    let obj;
    let objTwo;
    let objThree;
    db.collection("country").find().toArray(function (err, data) {
      if (err) {
        throw err;
      }
      res.writeHead(httpStatus.OK, { // yes, write
        relevant header
        "Content-Type": "text/html; charset=utf-8"
      });
      obj = data;
      db.collection("city").find().toArray(function (err, data) {
        if (err) {
          throw err;
        }
        res.writeHead(httpStatus.OK, { // yes,
          write relevant header
          "Content-Type": "text/html; charset=utf-8"
        });
        objTwo = data;
        db.collection("language").find().toArray(function (err,
          data) {
          if (err) {
            throw err;
          }
          res.writeHead(httpStatus.OK, { //
            yes, write relevant header
            "Content-Type": "text/html; charset=utf-8"
          });
          objThree = data;
          console.log(obj);
          console.log(objTwo);
          console.log(objThree);
          res.write(countryList.cities(obj, objTwo, objThree,
            asset));
          con.close();
          res.end();
        });
      });
    });
  });
}

```

Node.js & MongoDB

Ved at anvende Node.js og MongoDB sammen har det givet mulighed for at lave en dynamisk hjemmeside, hvor man kan hive data fra en database og tilføje dem på en side.

I forbindelse med arbejdet med Node.js, opstod der overvejelser omkring, hvordan man kunne gøre tingene nemmere, så man ikke skulle gentage sig selv flere gange. Der skulle oprettes forbindelse til en database med 3 forskellige collections med hver deres data. Til at starte med blev den samme kode gentaget flere gange, hvilket er unødvendigt, da den eneste forskel er collection navnet, der har samme navn, som dens url. Der blev derfor valgt at lave en parameter i funktionen, der bliver modtaget i forbindelse med sidens dannelse, som bliver sendt hen til funktionen, der skaber en forbindelse til databasen.

```
dbRead(req, res, asset) {
  asset = asset.substring(1);
  const mongo = require('mongodb');
  const dbname = "world";
  const constr = `mongodb://localhost:27017`;

  mongo.connect(constr, { useNewUrlParser: true, useUnifiedTopology: true }, function (error, con) {
    if (error) {
      throw error;
    }
    const db = con.db(dbname); // make dbname the current db
    /* Retrieve,
     * reads cities from the database
     */

    db.collection(asset).find().sort({continent : 1, country : 1, }).toArray(function (err, city) {

      if (err) {
        throw err;
      }
      res.writeHead(httpStatus.OK, { // yes, write relevant header
        "Content-Type": "text/html; charset=utf-8"
      });
      if (asset === "country") {
        res.write(experimental1.cities(city)); // home made templating for native node
      }
      if (asset === "city") {
        res.write(experimental2.cities(city)); // home made templating for native node
      }
      if (asset === "language") {
        res.write(experimental3.cities(city)); // home made templating for native node
      }
      con.close();
      res.end();
    });
  });
},
```

Ved at anvende denne parameter, og fjerne '/' fra url navnet, kunne man nøjes med en funktion, hvor der var 3 forskellige if statements alt efter sidens url.

Da der skulle arbejdes med informationer fra en form, opstod der nogle udfordringer, da der var tvivl om, hvordan man skulle hente de forskellige data fra formen. Der blev her prøvet forskellige muligheder, men efter utallige forsøg og vejledning, kunne man skrive informationerne ind i en kontaktformular, som efterfølgende blev sat ind i en database. Det blev løst ved at fortælle, hvilket data den skulle sende med videre, og hvor den skulle få det fra.

Problemet opstod, fordi der ikke var skrevet de rigtige navne til input felterne, og fordi den ikke henviste til den rigtige side.

```
receiveData(req, res, data) {
  let obj = lib.makeWebArrays(req, data); // home made GET and POST objects
  res.writeHead(httpStatus.OK, { // yes, write relevant header
    "Content-Type": "text/html; charset=utf-8"
  });
  res.write(experimental.receipt(obj)); // home made templating for native node

  const mongo = require('mongodb');
  const dbname = "world";
  const constr = `mongodb://localhost:27017`;
  let newCountry = { name: obj.POST.country,
    continent: obj.POST.continent,
    area: obj.POST.area,
    population: obj.POST.population,
    governmentForm: obj.POST.governmenform
  };
  let query = { name: newCountry.name };

  mongo.connect(constr, { useUrlParser: true, useUnifiedTopology: true}, function (error, con) {
    if (error) {
      throw error;
    }
    console.log(`Connected to server`);
    const db = con.db(dbname); // make dbname the current db
    /* Update,
    * updates/inserts city in the database
    */
    db.collection("country").updateOne(query, {"$set": newCountry}, {upsert: true}, function (err, collection) {
      if (err) {
        throw err;
      }
      console.log("City inserted/updated");
      con.close();
    });
  });
  res.end();
}
```

For at få informationen sat ind i databasen skulle der laves 2 variabler, som skulle sættes op mod hinanden. Står landet i databasen i forvejen skal den opdatere databasen, er den der ikke, skal det tilføjes til databasen. Når landet er tilføjet til databasen, skal den vises under den pågældende tabel. Men der opstod en fejl, da landene blev tilføjet under hinanden og gav en uendelig loop med divs.

[Home](#)[Countries](#)[Cities](#)[Language](#)

Europe

Countries

Denmark

Denmark

Continent	Europe
Area	42933
Population	5603000
Government form	Democracy

Spoken languages

Danish

Country	Denmark
Speakers	99
Is official	Yes

Cities

Laeborg

Russia

Russia

Continent	Europe
Area	17100000
Population	145934462
Government form	Democracy

Cities

Germany

Germany

Continent	Europe
Area	357386
Population	83783942
Government form	Democracy

Cities

Derfor blev der undersøgt, hvordan div-taggene skulle lukkes, så landene blev printet ud efter hinanden og ikke indeni hinanden. Der blev lavet et loop for "countries", "languages" "cities", men fordi alt content på siden var inde i et loop, kunne man ikke bare afslutte vores "div", da vores content så blev udenfor den samlede boks. Loopet blev derfor undersøgt og opdelt dem fra hinanden, for at undersøge, hvor henne der skulle lukkes.

```
console.log("test");
country += `
    <button class="play1"><h2>${obj2[q].name}</h2></button>
    <div class="cities">
        <div class="hiddenCity">
            <table>
                <tr>
                    <td>Is capital</td>
                    <td>${obj2[q].isCapital}</td>
                </tr>
                <tr>
                    <td>Population</td>
                    <td>${obj2[q].population}</td>
                </tr>
                <tr>
                    <td>Country</td>
                    <td>${obj2[q].country}</td>
                </tr>
            </table>
        </div>
    </div>
`;
    }
}

country += `</div></div>`;

countries += country;

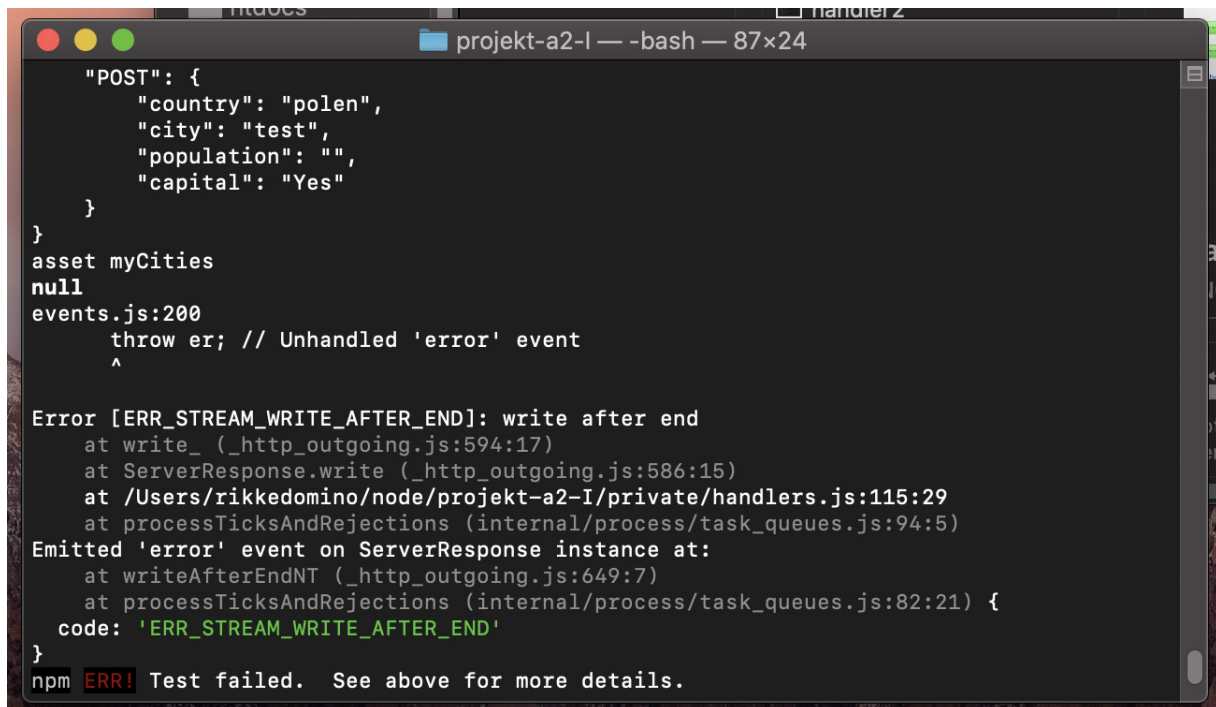
return htmltop + countries + htmlbot;
}
```

Udover at skrive indholdet pænt ind på siden, skal det være muligt at tilføje og ændre informationen. Landene skal kunne indsættes, samt byerne til landene og det pågældende sprog, men kun hvis landene i forvejen eksisterer i databasen, ellers skal de ikke tilføjes. I begyndelsen blev de 2 opgaver opdelt i hver deres objekt, men for at gøre det nemmere, blev de sat ind i den samme. Herefter blev en variabel anvendt for at indikere, hvilken collection der skulle ændres.

```
if (asset === "myCities") {  
  
  /* Update,  
  * updates/inserts city in the database  
  */  
  db.collection("city").findOne(findCountry).then(doc => {  
    console.log(doc);  
    if(doc === null){  
      console.log("Landet er der ikke");  
      res.write(notThere.receipt(obj.POST));  
      con.close();  
      res.end();  
    }else{  
      console.log("Landet er der");  
      db.collection("city").findOne(findCity).then(doc => {  
        db.collection("city").updateOne(findCity, {"$set": newCity}, {upsert: true}, function (err, collection) {  
          if (err) {  
            throw err;  
          }  
          console.log("City updated");  
          res.write(experimental.receipt(obj));  
          con.close();  
          res.end();  
        });  
      });  
    }  
  });  
}  
});  
}
```

I de forskellige collections blev der, ligesom i indsættelse af landene, sat 2 variabler op imod hinanden for først at undersøge om landet er der. Herefter blev der undersøgt om byen, var der i forvejen eller om, den skulle skrives ind. Er landet der ikke, bliver informationerne ikke sat ind på siden, men er informationerne der, bliver de sat ind på siden. I forbindelse med if statements var der udfordringer med at få den ind i statementet, hvis landet ikke var der i forvejen. Efter mange forsøg, blev der fundet ud af, at der ikke skulle være gåseøjne rundt om null. Da ændringerne blev foretaget, kunne funktionen komme ind i if statementet.

For at gøre brugeren opmærksom på om informationen er der eller ej, skal der åbnes 2 forskellige sider, vha. res.write. Her opstod der problemer, da der kom en fejlmeddelelse på consolen. Fejlen blev løst ved at skrive res.end() efter funktionen.



```
projekt-a2-I — bash — 87x24

"POST": {
  "country": "polen",
  "city": "test",
  "population": "",
  "capital": "Yes"
}
}
asset myCities
null
events.js:200
  throw er; // Unhandled 'error' event
  ^

Error [ERR_STREAM_WRITE_AFTER_END]: write after end
    at write_ (_http_outgoing.js:594:17)
    at ServerResponse.write (_http_outgoing.js:586:15)
    at /Users/rikkedomino/node/projekt-a2-I/private/handlers.js:115:29
    at processTicksAndRejections (internal/process/task_queues.js:94:5)
Emitted 'error' event on ServerResponse instance at:
    at writeAfterEndNT (_http_outgoing.js:649:7)
    at processTicksAndRejections (internal/process/task_queues.js:82:21) {
  code: 'ERR_STREAM_WRITE_AFTER_END'
}
npm ERR! Test failed.  See above for more details.
```

Konklusion

For at lave en dynamisk hjemmeside, er der blevet gjort brug af Node.js og MongoDB. Hjemmesiden har på forsiden et verdenskort, der er inddelt i 7 kontinenter, hvor man vælger et kontinent for at se databasens indhold under det pågældende kontinent. MongoDB bliver anvendt til at samle informationer omkring kontinentets lande, byer og sprog. Ved at anvende Node.js kan man tilgå databasen og udskrive informationerne på en hjemmeside. Node.js er også anvendt, så man kan tilføje og ændre i oplysningerne på siden vha. en formular. I formularen kan man ikke tilføje byer eller sprog til et land, som ikke er i databasen i forvejen. Ved at tilføje og ændre data til databasen bliver siden dynamisk.

Litteraturliste

Busch, A.M. (2015) Kommunikation i multimediedesign. Kbh.: Hans Reitzel

Capan, T. (n.d.) Why The Hell Would I Use Node.Js? A Case-by-Case Tutorial [online] available from <<https://www.toptal.com/nodejs/why-the-hell-would-i-use-node-js>> [17 February 2020]

International, B. (n.d.) 'How To Use Desk Research'. available from <<https://www.b2binternational.com/publications/desk-research/>> [20 February 2020]

Okholm, T. (2018) GUIDE: Få styr på et større projekt ved hjælp af Trello [online] available from <<https://www.altomdata.dk/guide-faa-styr-paa-et-stoerre-projekt-ved-hjaelp-af-trello>> [22 November 2019]

toolmaster (n.d.) Teknisk Ordbog - Wireframe - Toolmaster.Dk [online] available from <<https://www.toolmaster.dk/ordbog/44-wireframe>> [20 February 2020]

Trello (n.d.) About | What Is Trello? [online] available from <<https://trello.com/about>> [17 February 2020]

W3School (n.d.) Node.Js Introduction [online] available from <https://www.w3schools.com/nodejs/nodejs_intro.asp> [17 February 2020]

w3school (n.d.) Node.Js MongoDB Get Started [online] available from <https://www.w3schools.com/nodejs/nodejs_mongodb.asp> [17 February 2020]

Zetcode (n.d.) MongoDB JavaScript Tutorial - Programming MongoDB in JavaScript [online] available from <<http://zetcode.com/javascript/mongodb/>> [19 February 2020]

Bilag

Bilag I - Gantt

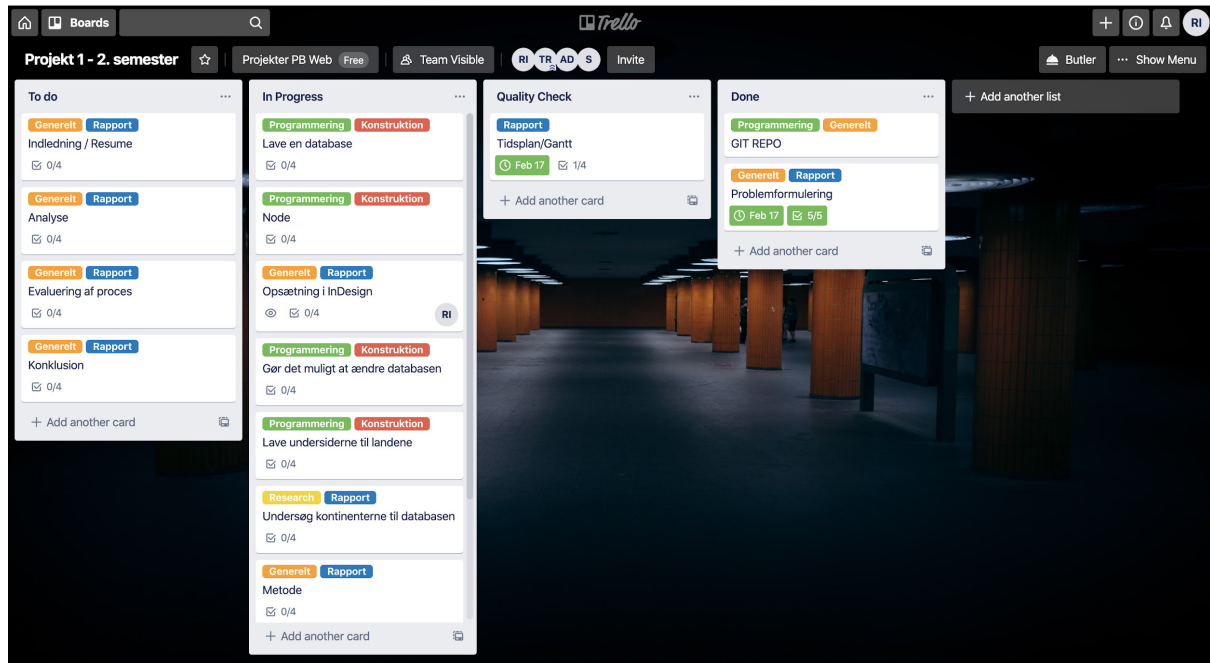
Bilag II - Trello

Bilag III - Wireframe

Bilag I - Gantt

Tidsplan										
Opgaver	Fredag	Lørdag	Søndag	Mandag	Tirsdag	Onsdag	Torsdag	Fredag	Lørdag	Søndag
Mødes										
Anders						10 til 14	Kage			
Simon				Kage						
Tinna										
Rikke										
Arbejder på kode		Alle	Alle	Simon	Anders	Tinna	Rikke	Simon	Alle	Alle
Generelt										
Indledning										
Problemformulering										
Metode										
Research										
Analyse										
Konstruktion										
Evaluering af analyse										
Konklusion										
InDesign										
Præsentation(powerpoint)										
Research										
Undersøge kontinenterne til databasen										
Konstruktion										
Lave en database										
Node										
Lav en forside										
Lav undersider til landene										
Gør det muligt at ændre databasen vha. formular										

Bilag II - Trello



Bilag III - Wireframe

Menu

Countries

Languages

Cities

Verdenskort

Asia

Europe

South America

North America

Australia

Africa

Antarctica