Projektuppgift

DT162G Javascriptbaserad webbutveckling

Projekt JavaScriptramverk
Bloggwebbplats med React

Cristina Löfqvist

Mittuniversitetet
MID SWEDEN UNIVERSITY

Campus Härnösand Universitetsbacken 1, SE-871 88. **Campus Sundsvall** Holmgatan 10, SE-851 70 Sundsvall.
Campus Östersund Kunskapens väg 8, SE-831 25 Östersund.
Phone: +46 (0)771 97 50 00, Fax: +46 (0)771 97 50 01.

MITTUNIVERSITETET
Avdelningen för informationssystem och -teknologi

Författare: Cristina Löfqvist, crlo1900@student.miun.se
Utbildningsprogram: Webbutveckling, 120 hp
Huvudområde: Datateknik
Termin, år: VT, 2020

Sammanfattning

Detta projekt har gått ut på att utveckla en webbplats som konsumerar webbtjänster med hjälp av Node.js, Express och MongoDB och använder ett JavaScript baserat front end-ramverk i form av React. Resultatet av projektet har blivit en webb-applikation med följande funktionalitet:

- Bloggportal med möjlighet att registrera användarkonto.
- Loggin funktionalitet.
- Möjlighet för registrerade och inloggade användare att publicera blogginlägg.
- Möjlighet för registrerade och inloggade användare att radera egna publicerade blogginlägg.
- Möjlighet för registrerade och inloggade användare att uppdatera användaruppgifter.
- Display av publicerade blogginlägg.
- Funktionalitet att logga ut.

Projektet har inneburit en hel del utmaningar. Framför allt gällande att sätta sig in i React biblioteket och hur asynkrona anrop fungerar.

Slutprodukten blev en fungerande bolgg-webbplats med en samling användbara React komponenter.

Innehållsförteckning

Terminologi

Akronymer/Förkortningar

GUI	Graphical User Interface
DOM	Document Objekt Model

1 Introduktion

1.1 Bakgrund och problemmotivering

JavaScript är ett av de mest använda programmerings-språket inom webbutvecklings-området. Det kan utföra en uppsjö fantastiska, väl användbara saker. Men nackdelarna är bland annat att det ofta skapar komplicerad logik och leder till dubbelarbete för att få till Graphical User Interface GUI med HTML och CSS. En liten ändring på ett element kan få effekter för hela sidans komposition. Genom React kan man skapa självständiga komponenter som enkelt kan återanvändas i och utanför projektet och dessa går även ganska enkelt att plocka up och vidareutveckla för bättre GUI.

1.2 Övergripande syfte

Att utforska hur man kan konsumera webbtjänster och upptäcka fördelarna man kan vinna på att använda React som ett Javascript baserat ramverk för front-end utveckling.

1.3 Avgränsningar

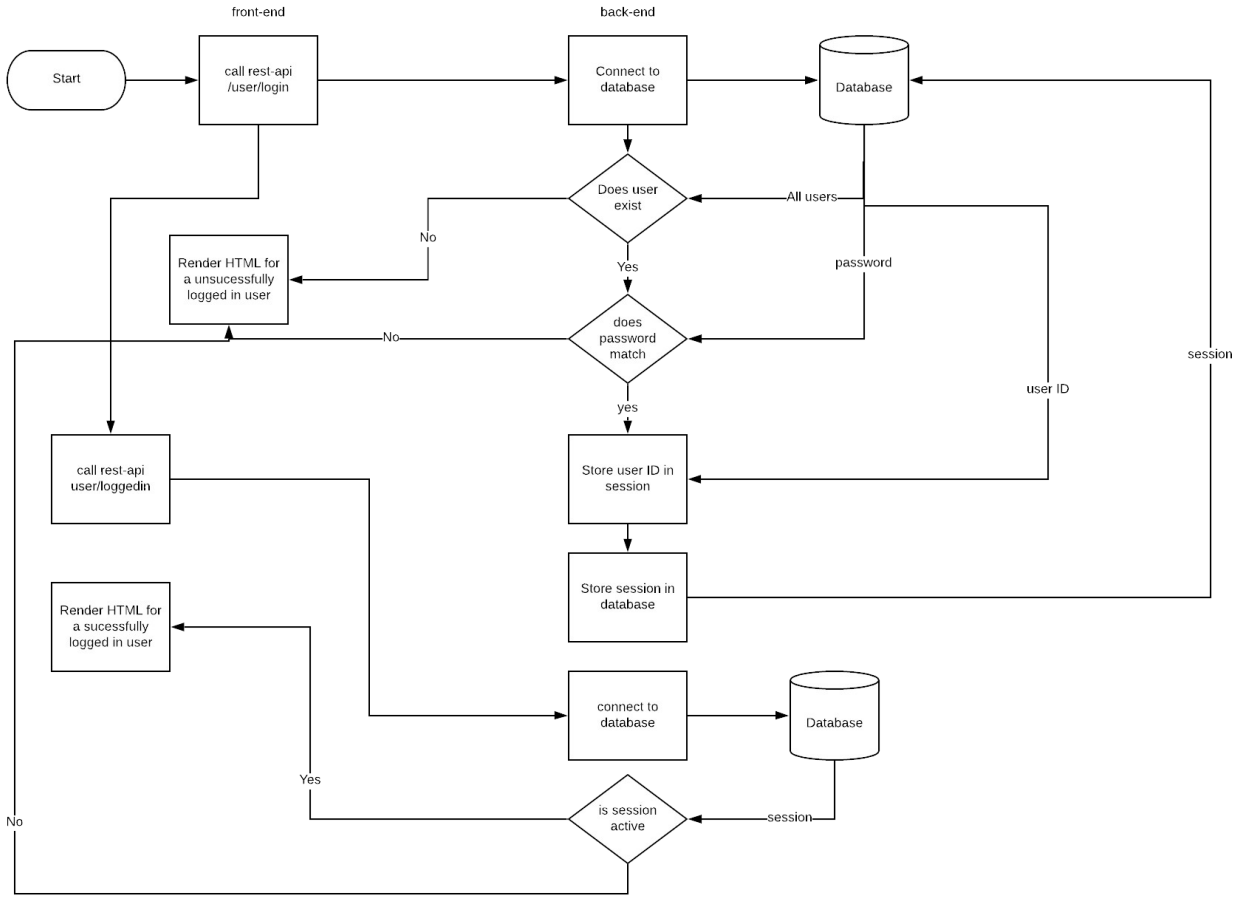
Projektet har fokus på att skapa en bloggportal för text innehåll och har inte försetts med möjlighet till blogginlägg med bild, ljud eller video innehåll. Det finns heller ingen implementerad funktionalitet för att kontrollera om korrekt input matas in av användare eller generera eventuella felmeddelande.

1.4 Konkreta och verifierbara mål

Att skapa kunskaper i användning av React som ett JavaScript baserat ramverk för front-end utveckling samt att konsumera webbtjänster med Node.js, express och MongoDB och producera en webb-applikation som använder sig av detta.

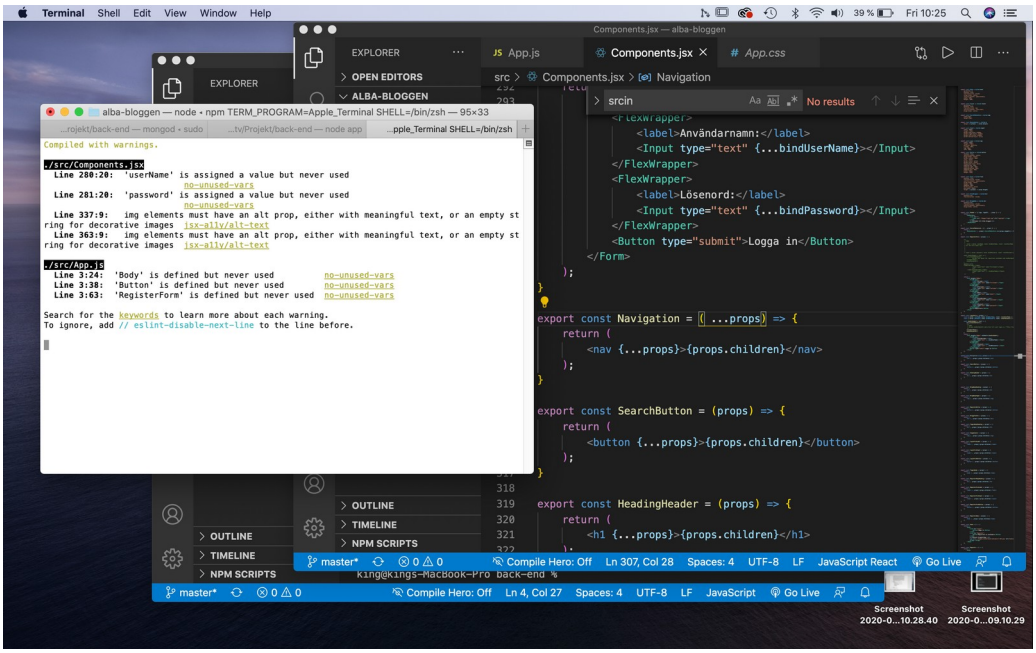
2 Teori

Applikationen är uppdelad i två delar. En front-end del och en back-end del.



Figur 1.0 Flödesschema över applikationen.

2.1 Frontend



Figur 1.1 ovan visar hur react applikationen startas för att köras lokalt vid utvecklingsskedet av applikationen. Genom att navigera till src mappen i den react applikation som skall köras och ange: `npm start`

2.1.1 React

Grundstenarna för uppbyggnaden av webbplatsens front-end utgörs av React och React styled components. React är ett JavaScriptbibliotek [2] som i motsats till klassisk programmering där man skiljer på logik och markup i olika filer, bygger på komponenter som kan innehålla och kombinera båda delar i samma komponent[3]. Med Tillägg av modulen styled components kan man även inkludera stilmallar på komponent-nivån. På så sätt får man ett objekt som innehåller samtliga delar det vill säga marcup, stilmall och logik. Allt i ett [4].

2.1.2 JSX

JSX är en syntax utbyggnad till JavaScript som med fördel kan användas till React för att beskriv hur User interface UI ska se ut [3]. JSX producerar React element som sedan renderas till Document Object Model (DOM) [3].

```
const username = posts[i].userName;
const userNamePost = <h5>Hello, {username}</h5>;

ReactDOM.render(
element,
document.getElementById('userNamePost')
);
```

Figur 1.2 Kodexempel på JSX React.

Figur 1.1 visar hur en variabel deklareras i JSX och används genom att omge variabelnamn username med måsvingar för att sedan renderas till DOM. Här skapas en variabel som innehåller värdet av användarnamn och skriver sedan ut det som ett h5 element till DOM. Man kan även se i exemplet ovan att html används direkt, det är en av styrkorna och funktionen med JSX att det är en javascript extension som tillåter html direkt i koden.

Vilket Javascript uttryck som helst går att översätta till JSX [3]. I motsatts till webbläsarens DOM element så är React element rena objekt. React bygger på komponenter som gör det möjligt att dela upp User Interface i självständiga enheter som går att återanvända [3]. React komponenter fungerar i huvudsak så att de översätter rå data till HTML [6].

Komponenter i React kan skapas på två olika sätt. Dels genom funktionella komponenter som är vanliga JavaScript funktioner med JSX och dels genom ES6 klass definierade komponenter [3].

2.1.3 Props och state

I React används något som heter props och state. Dessa utgörs av data som HTML byggs upp från [6]. Både props och state fungerar som rena JavaScript objekt. Båda innehåller information som påverkar resultatet av det som kommer att renderas. Men dessa skiljer sig åt i ett stort avseende. Nämligen att props skickas med till komponenten (ungefär som parametrar i funktioner) och state upprätthålls inuti komponenten (ungefär som variabler som deklareras inuti funktioner) [5].

2.1.4 Hooks

React hooks har använts för att tillhandahålla ett sätt att hantera logik i funktionella komponenter och samtidigt dela icke user interface (UI) logik och beteende över applikationen [10].

När funktionella komponenter implementeras kommer en komponent just vara en funktion. En funktion har ingen möjlighet att spara undan ett så kallat state, som i en klass komponent hade representerats av ett attribut. Men med hjälp av så kallade hooks kan states skapas även för funktionella komponenter. Det finns flera olika hooks för att utföra olika arbeten. Men för att just skapa ett state används hooken useState.

Kod exempel:

```
const useInput = initialValue => {
const [value, setValue] = useState(initialValue)

return {
value,
setValue,
reset: () => setValue(""),
```

```
bind: {
value,
onChange: event => {
setValue(event.target.value);
}
}
}
```

// Funktionen som heter useInput tar argumentet initialValue. I funktions-skopet utförs följande: En array med value och setValue där value representerar ett state och setValue är en funktion för att ändra värdet av value. Detta fås av att man använder hooken useState på följande vis: const[value, setValue] = useState(initialValue) . Argumentet till useState, (initial value) är startvärdet på statet (value). Sedan returneras ett objekt(mellan måsvingarna) innehållande följande: Värdet av statets value, funktionen setValue, den anonyma funktionen reset samt objektet bind som innehåller statets value och en onchageeventhandler som anropar setvalue med argumentet av värdet av eventet [10].

Denna används genom:

```
/**
* Hook to get values from input fields
*/
const { value: newEmail, bind: bindNewEmail, reset:
resetNewEmail } = useInput('');
```

Först så skapas ett objekt (samma antal som antalet inputfält i formet). Genom följande:`const { value: newEmail, bind: bindNewEmail, reset: resetNewEmail } = useInput('');`

Det som sker på denna raden är att ett objekt skapas som innehåller nyckeln value som får värdet av inputfältet som den representerar. ”newEmail” i detta fallet. En nyckel ”bind” som kommer representera ”on change event” eventet och nyckel ”reset” som kommer vara en funktion som resetar inputfältet. Allt detta returneras från useInput i.e useInput anropas och argumentet/startvärdet sätts till tomt. Sedan skapas en funktion som heter handleSubmit som tar ett event som argument. Denna funktionen handleSubmit kommer att anropas då elemetet form submittas. För att veta vilket värde som är vilket läggs följande till för ett input fält: `<Input type="text" {...bindNewEmail}></Input>`

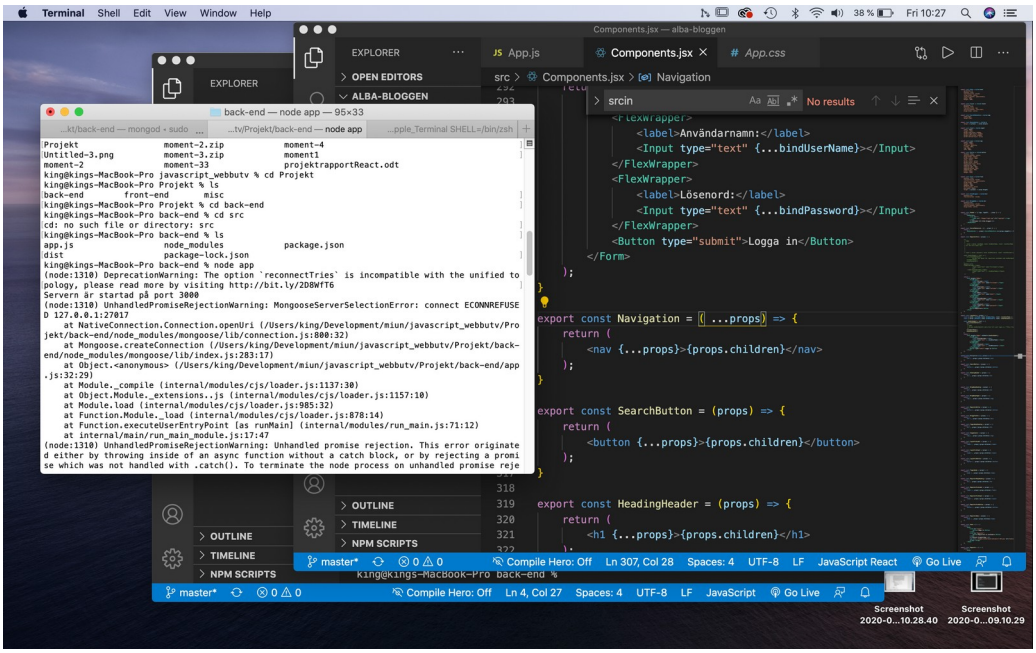
[10].

2.1.5 Routing

React används för att skapa så kallade single paged applicaions. Vilket innebär att det finns inga undersidor i den bemärkelsen att det finns flera separata html filer för undersidorna. Då react renderar, alltså bygger, HTML element så är det samma sida som visas i browsern hela tiden fast innehållet kan bytas ut för att skapa sidor som ser olika ut. För att åstadkomma fler olika sidor så används react routing [2].

2.2 Backend

Projektets back-end del använder sig av en Node.js baserad server där REST-APIerna har implementerats, samt av en Mongodatabas där information angående användaruppgifter och blogginlägg lagrats.



Figur 2.0 ovan visar hur node.js applikationen där rest APIer finns, startas i termnialen, För att köras lokalt vid utvecklingsskedet av applikationen, genom att navigera till mapp en där app.js finns och köra kommandot: node app .Dessa APIer innehåller kopplingar till mongo-databasen för att lagra information.

2.2.1 Node.js

Node.js utvecklades 2009 av Ryan Dahl och är ett utvecklings ramverk som är baserat på Googles V8 JavaScript motor som en miljö på serversidan som matchade klient-sidans miljö i webbläsaren. Node.js bygger på Javascript. Node.js gör det möjligt att skapa skript som kör både på server och klient - sidan. Detta är en stor fördel eftersom man då kan ta JavaScript som är skrivet på klient-sidan och enkelt adaptera det till server-sidan [1]. Utvecklare kan då

enklare navigera projekt mellan klient och server sidan och även återanvända kod emellan dessa miljöer [1].

Node modulerna som använts i projektet är express, bycrypt, connect-mongo, express-session och mongoose. För att installera och spara Node moduler för ett projekt användes npm (Node package manager) exempelvis för att installera express har följande exekverats i rotkatalogen för projektet: npm install –save express@4.0.0. Npm har massvis med funktionalitet som kan hjälpa utvecklaren att till exempel skapa ett nytt Node.js projekt, genom att skapa den struktur som krävs för att köra en Node server eller att skapa ett nytt React projekt liknande hur ett Node,.js projekt skapas.

2.3 Node.js moduler

2.3.1 Express

Express modulen packar ihop och utvidgar funktionaliteten av Node.js httpmodul i ett gränssnitt som är lätt att använda [1]. Express modulen har använts för att skapa de REST-APIer som krävdes för att uppnå den önskade funktionaliteten för webbplatsen. Följande funktioner implementerades via ett eller flera REST-APIer:

- Autentisering av användare
- Skapandet av nya användare
- Hämta alla blogginlägg som finns sparade i databasen
- Skapa nya blogginlägg
- Hämta ett specifikt blogginlägg
- Ta bort ett specifikt blogginlägg
- Hämta information angående en specifik användare
- Logga ut en inloggad användare
- Uppdatera email av en användare.

Nedan visas hur expressmodulen har importerats genom:

```
var express = require("express");
```

och instansierats genom: `var app = express();`

2.3.2 Autentisering

För att identifiera användare och förse med olika möjligheter till tillgång av uppgifter och innehåll beroende på om en användare är inloggad eller ej så har applikationen som skapats utformats så att den tillhandahåller loginfunktionalitet där vissa kriterier verifierar en användare [1].

För att skapa Autentisering har så kallade sessioner använts. En session är en form av lagringsutrymme av information som ligger på serversidan. En session är i aktivt tillstånd en viss angiven tid[9].Varje gång som webbplatsen från front-end sidan anropar ett REST-API eller skall rendera HTML beroende på om en användare är inloggad eller ej så anropas först ett REST-API som tillhandahåller information om den givna användaren har en aktiv session eller inte. Alltså om en användare har en aktiv session, menas det att denna användare är autentiserad. För att kunna se om en användare har en aktiv session eller inte så kommer varje gång front-end delen gör en HTTPrequest, så skickas session id till webbservern detta sessions id paras ihop med den session informationen som webbservern lagrat i mongodatabasen[9].

Detta har implementerats med hjälp av följande node moduler:

- Bcrypt
- Connect-mongo
- Express-session

2.3.3 Bcrypt

Bcrypt modulen är en Node.js modul som tillhandahåller funktionalitet bland annat för autentisering av användare och säker lagring av användaruppgifter. Följande funktion har används för säker lagring av lösenord i Mongodatabasen:

```
bcrypt.hash(req.body.password, 10)
```

Det bcrypts metod hash gör i anropet visar här ovan är att den hashar ett lösenord med ett auto-genererat salt, anledningen till att man vill hasha ett lösenord är att man inte vill spara lösenord i en databas i plain-text utfall att databasen blir hackad. Här är ännu ett exempel på ett asynkront anrop, bcrypt.hash körs asynkront eftersom applikationen annars riskerar att bli hängande och vänta på hasning och saltning av lösenordet vilket kan ta lång tid beroende på lösenordets längd, komplexitet, saltets längd och komplexitet[8]. Om Node.js applikationen hänger och väntar på att hashningen av ett lösenord skall bli klar kan en anropande front-end del bli hängandes och laddningstiden av eventuella HTML element bli påverkad.

Bcrypt har även använts för autentisering av användare. När man autentiserar en användare jämförs ett angivet lösenord med det hashade lösenordet i databasen, och för att användaren inte skall behöva dölja sitt lösenord för hand tillhandahåller bcrypt funktionalitet för att jämföra ett lösenord i klartext (angivet av en användare vid ett inloggningsförsök) och ett hashat lösenord (lösenord för en angiven användare lagrat i exempelvis en mongodatabas).

2.3.4 Express-session

Express-session har använts som en viktig del i projektets funktionaliteter för registrering och inloggning. Något som leder till säkerhetsbrister med HTTP autentisering är att inloggningen fortskrider så länge kriterierna för inloggningen lagras. Det är därför fördelaktigt att implementera en egen autentisering som lagras i sessioner som löper ut efter önskvärd tid [1]. Express session är en Node modul som agerar som en bro mellan databasen och applikationen och finns inuti Express modulen [1].

2.3.5 Connect-mongo

Connect-mongo är en MongoDB session lagring för Connect och Express och används i projektet för att lagra sessioner de sessioner som skapats med express-session modulen i mongodatabasen [1].

2.3.6 Mongoose

Applikationen kopplar upp till mongodatabasen genom metoden:

```
mongoose.connect(`mongodb:${server}/${database}`)
```

Mongoose är ett Object Document Model (ODM) bibliotek som tillhandahåller extra funktionalitet för anslutning till en mongodatabas. Mongoose förenklar några av svårigheterna som finns kring att utföra databasanrop från en Node.js applikation genom metoder för databasoperationer som save, find, update och remove [1].

2.4 Databasen

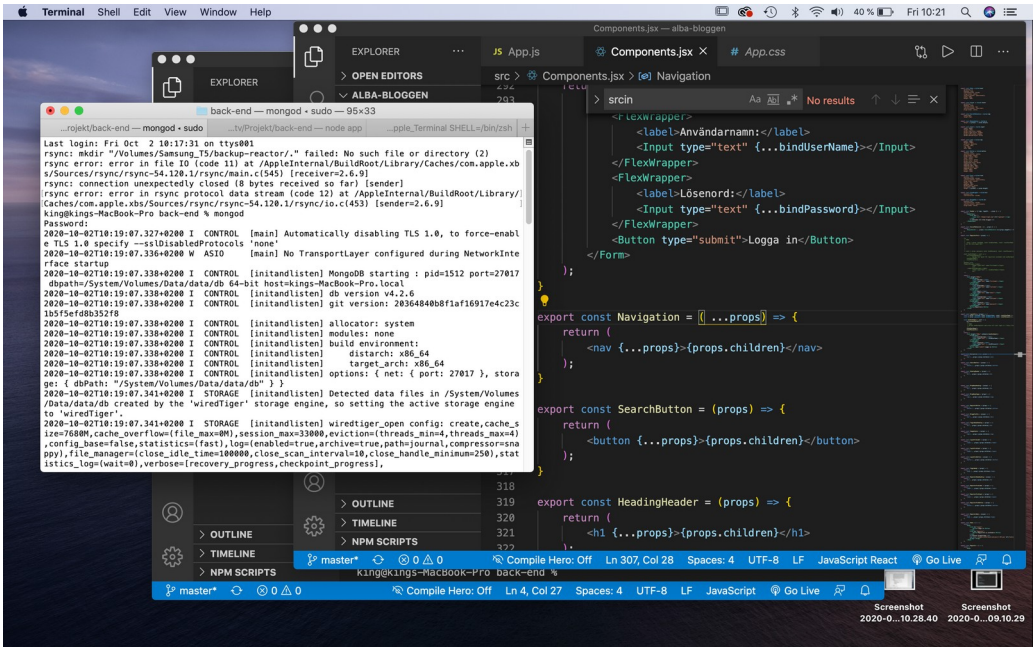
2.4.1 MongoDB

MongoDB är databasen som använts för projektet. MongoDB är en databas av typen NoSQL. Namnet Mongo härleds ifrån ordet ”humongous” som understryker skalbarheten och prestandan som MongoDB erbjuder [1]. I motsatts till traditionella relationsdatabaser som lagrar datan i kolumner och rader så lagrar MongoDB data i form av JSON liknande objekt [1]. Detta är fördelaktigt eftersom det bortarbetar behovet av att omvandla data från rader till objekt och vice versa [1]. MongoDB är en av de databaser som tillhandahåller högst prestanda och dess back-end har stöd för hög trafik och skalbarhet över mulitpla servrar[1].

Ett MongoDB schema definierar fält och fälttyperna för dokumenten i så kallade kollektioner [1]. Detta kan vara väldigt användbart eftersom det gör att

lagringen av data blir mer strukturerad och det blir möjligt att göra så kallad ”typkastning”alltså att man kan bestämma typen för dokumenten i objektmodellen/kollektionerna i mongoDB databasen. Det vill säga om det sa vara av typen nummer, string, array, buffer, date, objectId, Mixed eller boolean. Objekten/dokumenten i schema-modellen går även att validera [1]. För varje fält i ett schema måste man definiera en specifik värdetyp. Ett schema måste definieras för varje olika dokumenttyp som ska användas och endast dokument av en typ bör lagras i varje kollektion [1].

Mongo db startas via terminalen genom att ange: mongod om det finns en global installation på datorn som i detta fallet.



Figur 2.1 ovan visar hur Mongo databasen startas för att köras lokalt vid utvecklingsskedet av applikationen.

3 Metod

3.1 Verifiering

Applikationen testkördes kontinuerligt under utvecklingsfasen lokalt via localhost. Under utvecklingsfasen testades REST-API vi applikationen Postman.

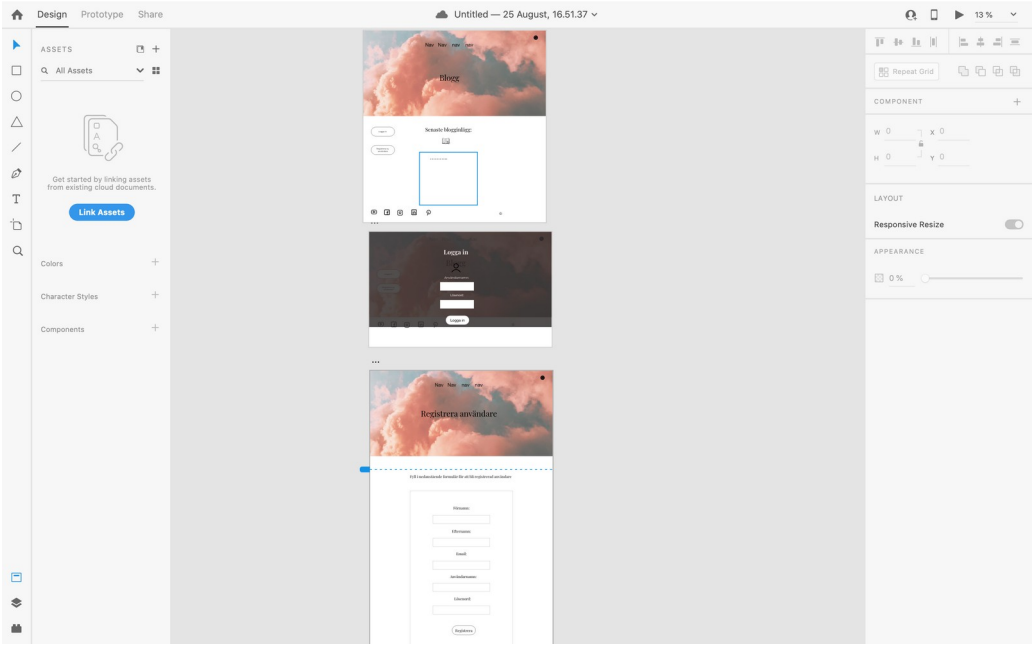
3.2 Verktyg

Följande verktyg och miljöer har använts under projektet:

- Utvecklingsmiljön var Visual Studio Code.
- Node.js har använts som webbserver.
- Mongod har använts för att starta Mongodatabas-hanteraren.
- Npm har använts för att installera de Node moduler som använts på back-end sidan av projektet.
- React komponenter har använts för att bygga DOM.

4 Konstruktion

Inledningsvis av projektet skapades mockups och genom den ringades in vilka React komponenter och sub-komponenter som skulle behövas. Dessa komponenter namngavs. Tankesättet kring att staka ut vilka komponenter som skulle behövas var att varje komponent ideellt sett bara ska utföra en sak.



Figur 3.0. printscreen av mocup för webbplatsen.

Figur 3.0 visar hur komponenterna enkelt kunde identifieras från mockup modellen.

Därefter arrangerades dessa efter de hierarkier som fanns enligt nedan:

- GridBody
  - Header
    - Navigation
      - Link
    - SearchButton
    - HeadingHeader

    - Logo
- BlogBody
    - BlogBodyHeading
    - BlogBodyImage
    - LoginButton
    - RegisterButton
    - BlogTextDiv
- ToggleBody
    - ToggleBodyHeading
    - ImageAvatar
    - LoginForm
        - LoginFormLabel
        - LoginFormInput
        - LoginFormButton
- RegiserBody
    - RegisterBodyHeading
    - RegisterForm
        - RegisterFormlabel
        - RegisterFormInput
        - RegisterFormButton

Sedan selekterades ut vilka av komponenterna som kunde kategoriseras som så kallade porps eller state.

Efter det skapades en statisk version av applikationen som renderade datamodellen. För att skapa komponenter som var återanvändbara och föra över data så användes porps. Med props kan man göra att komponenterna ärver data i nedåtstigande led i hierarkin [5].

5 Resultat

5.1 Tekniska problem

Under projektet har det upplevts ganska klurigt och krävande att arbeta med <form> och få applikationen att fungera kring detta med React.

Det framkom att det fanns problem med förståelsen kring hur Node servern utför uppgifter asynkront.

Till en början skapades webbskisser för att göra inloggningens GUI med en ”modal” där en ruta för inloggning visas i webbläsaren som lägger sig något svart/transparent över övriga element i webbläsaren. Detta fick väljas bort senare under utvecklingsfasen då det upplevdes som för komplext utifrån den givna tiden och försening som uppstod i projektet.

Flera problem uppstod med REST-API anrop. En hel del tid fick läggas på att studera hur verktyg som Postman fungerar samt hur de olika node modulerna som användes i projektet fungerade och vilken funktionalitet de hade och hur de kunde användas på bästa sätt.

Under projektet har det också upplevts vara en stor utmaning att få till en bra studie- och arbetsmiljö. Projektet har dragit ut på tiden eftersom det varit kantat av störande moment på grund av vård av barn etcetera. Men insikten har tillkommit att det finns begränsningar i vad man kan styra över och oftast får man arbeta med de förutsättningar som ges och detta har gett lärdomen att kunna prestera trots knepiga förutsättningar.

6 Slutsatser

Det har varit mycket givande och riktigt roligt att arbeta med Node.js applikation och med React som front-end ramverk. Det blir tydligt att det på långsikt innebär ett mer effektiv arbetsätt att bygga upp egna react komponenter som kan återanvändas om om och till senare projekt istället för långa rader av HTML och CSS kod. Genom detta projektarbetet nätvek skapats och ett github repo för react komponenter som förhoppningsvis ska växa allt eftersom. Det har också varit enormt givande att bekanta sig med utvecklings-ramverket Node.js och dess många fördelar och användningsområden. Nya kunskaper och erfarenheter har inhämtats som kan bidra till användbara kompetenser i branschen.

Källförteckning

[1] Dayley Brad 2014. ”Node.js, MongoDB and AngularJS web Development”. Pearson Education. Inc.

[2] React ”Hello world” https://reactjs.org/docs/hello-world.html Publicerad: 2020. Hämtad: 2020-08-25.

[3] React ”Introducing JSX”https://reactjs.org/docs/introducing-jsx.html Publicerad: 2020. Hämtad: 2020-08-25.

[4] Styled components ”Visual components for the primitive ages”https://styled-components.com/ Publicerad: 2020-05-26. Hämtad: 2020-08-25.

[5] React ”Component State”https://reactjs.org/docs/faq-state.html Publicerad: 2020. Hämtad: 2020-08-26.

[6] GitHub ”Props vs State”https://github.com/uberVU/react-guide/blob/master/props-vs-state.md Publicerad: 2017-07-06. Hämtad: 2020-08-26.

[7] Nodemailer ”Nodemailer”https://nodemailer.com/about/ Publicerad: 2020. Hämtad: 2020-08-1

[8] MDN ”Node.bcrypt.js” https://www.npmjs.com/package/bcrypt Publicerad: 2020-06. Hämtad: 2020-08-11.

[9] SP Lessons ”JavaScript Session” https://www.splessons.com/lesson/javascript-session/ Publicerad: Hämtad: 2020-07-10.

[10] R ”Simplify react forms with hooks” https://rangle.io/blog/simplifying-controlled-inputs-with-hooks/ Publicerad: 2019-01-25. Hämtad: 2020-10-02.