Sisällönhallintajärjestelmä infonäytöille

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Ammattikorkeakoulututkinnon opinnäytetyö

Tietotekniikan koulutusohjelma

Hämeen ammattikorkeakoulu Riihimäki, kevät 2017

Jukka Kari

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Tietotekniikan koulutusohjelma
Riihimäki

Tekijä Jukka Kari **Vuosi** 2017

Työn nimi Sisällönhallintajärjestelmä infonäytöille

Työn ohjaaja Petri Kuittinen

TIIVISTELMÄ

Tämän opinnäytetyön tavoitteena on kertoa infonäytön sisällönhallintajärjestelmän toteutuksen työprosessi ja se, mitä infonäytöllä ja sisällönhallintajärjestelmällä tarkoitetaan. Käyn myös läpi sitä, missä infonäyttöjä käytetään. Tavoitteena oli tehdä oma infonäytön sisällönhallintajärjestelmä itse suunniteltuna ja toteutettuna kerryttäen samalla omaa osaamista ohjelmistokehityksessä. Suunnittelu oli tapahtunut pitkällä aikavälillä päässä ideoiden. Idean sain alun perin työharjoittelupaikan töiden yhteydessä, jossa tein töitä info- ja mainosnäyttöjen parissa.

Kerron, mitä työvälineitä ja teknologioita valitsin projektiin käytettäväksi ja miksi. Sain lopulta aikaan toimivan infonäyttöjen sisällönhallintajärjestelmän, johon älykkyyttä toi erityisesti sisällölle asetettavat parametrit, joilla voidaan määritellä, milloin sisältöä halutaan näyttää.

Pohdin myös, millä tavalla sovellusta voisi jatkokehittää ja mitä voisi tehdä paremmin. Projektista sain omaa työllistämistä varten hyvää näyttöä ohjelmoinnin osaamisesta, joka oli tarkoituskin.

Avainsanat Infonäyttö, sisällönhallintajärjestelmä, React, Node.js, näyttöviestintä

Sivut 37 sivua

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Degree Programme in Information Technology
Riihimäki

Author Jukka Kari **Year** 2017

Subject Digital Signage CMS

Supervisor Petri Kuittinen

ABSTRACT

The purpose of this thesis was to go through the process of creating a content management system for digital signage and to discuss what digital signage and content management system stand for. I also examined targets where digital signage is used. The aim of this project was to design and create a content management system by myself while building my skills in software development. I had been planning this project in my head for a long time already. I got the idea for this thesis when I was working as an intern for a company that specializes in digital signage.

This thesis contains a description of the tools and technologies I chose to use in this project and the reasons for choosing them. I managed to create a working content management system for digital signage, which has some intelligence built-in, where you can set when you want to show the content on displays with different parameters.

In this thesis, I also explore how to further develop the application and what I could have done better. I managed to create an application to showcase my programming skills, which was also what I was after.

Keywords Digital signage, content management system, React, Node.js

Pages 37 pages

SISÄLLYS

1 JOHDANTO

Infonäyttöjen, jumboscreenien ja digitaalisten mainostaulujen käyttö on ollut jo pitkään esillä mainonnan ja yritysten ulkoisen ja sisäisen viestinnän yleistyessä. Näytön sisältö on helpompi vaihtaa kuin perinteisen mainostaulun, jonne pitää viedä erikseen esimerkiksi paperinen juliste. Näyttöjen yleistymisen takia aloin kiinnostua oman sisällönhallintajärjestelmän tekemisestä infonäytöille.

Opinnäytetyön tavoitteena oli luoda sisällönhallintajärjestelmä infonäytöille, joka sisältää hieman ”älykkyyttä” siten, että sisällölle voidaan määritellä tietyt ehdot, milloin sisältöä näytetään näytöllä. Sovellus toteutetaan käyttäen Node.js-ohjelmaa, jolla voidaan tehdä web-applikaatioita. Ohjelma on koodattu käyttäen JavaScript-ohjelmointikieltä, jota käytetään projektissa sekä palvelimen että selaimen puolella. Sisältöjen ja näyttöjen tiedot tallennetaan MongoDB-tietokantaan.

Opinnäytetyön tekemisen päällimmäisenä tarkoituksena oli opetella käyttämään minulle uusia työvälineitä ja teknologioita, joita käytetään nykyään web-sovelluskehityksessä, kuten Reactin käyttö käyttöliittymän toteuttamisessa.

Opinnäytetyössä käydään läpi, kuinka suunnittelen ja toteutan oman sisällönhallintajärjestelmän infonäytöille ja mitä työvälineitä ja tekniikoita olen valinnut käyttöön ja miksi. Kerron myös hieman sisällönhallintajärjestelmistä ja infonäytöistä. Lopussa myös pohditaan jatkokehitysmahdollisuuksia ja mitä olisi voinut tehdä toisin.

2 SISÄLLÖNHALLINTAJÄRJESTELMÄT

Sisällönhallintajärjestelmistä käytetään usein lyhennettä ”CMS”, joka tulee englanninkielen sanoista ”Content Management System”. Kuten nimestä saattaa päätellä, sisällönhallintajärjestelmällä hallitaan jonkun kohteen tai ohjelman sisältöä. Sisällönhallintajärjestelmä otetaan käyttöön usein sen takia, että sillä voidaan helpottaa esimerkiksi web-sivun sisällön muokkaamista ilman, että pitäisi tietää mitään ohjelmoinnista. (Wikipedia, 2017.)

Suosituimpia websisällönhallintajärjestelmiä on mm. WordPress, Joomla ja Drupal, joista selvästi suosituin on WordPress, joka on käytössä 28%:lla kaikista websivuista (W3tecs, 2017). Näillä sisällönhallintajärjestelmillä voidaan ylläpitää web-sivuja, jotka voivat sisältää mm. kuvia, videoita tai

vaikkapa blogin. Yhteistä näillä kaikilla on myös se, että kaikki edellä mainitut sisällönhallintajärjestelmät on ohjelmoitu PHP-ohjelmointikielellä.

Suomessa info- ja mainosnäytöille on myös kehitetty monta omaa web-sisällönhallintajärjestelmää. Yksi näistä on FirstView, jota käytetään useissa Suomen S-ryhmän ketjuissa (FirstView, n.d.). Yhteistä infonäyttöjen sisällönhallintajärjestelmissä on se, että sisältöä voi lisätä kuvina, videoina tai HTML-tiedostoina. Monet sisältävät myös sisältöpohjia joihin voi lisätä omat tekstit ja kuvat palvelun omassa editorissa. Myös sisällön ajastus päivän ja kellonajan mukaan on mahdollista.

3 INFONÄYTÖT YLEISESTI

Infonäytöistä puhuessa käytetään usein sanoja ”digital signage”. Yksinkertaisin määritelmä digital signagelle on ”etäohjattava digitaalinen näyttö, joka on tyypillisesti sidoksissa myynnin, markkinoinnin ja mainostamisen kanssa” (Jimmy, 2013).

Paikkoja joissa voit törmätä digital signage -ratkaisuihin, ovat esimerkiksi kauppakeskukset, liikkeet, koulut, työpaikat, tienvarret, koti, julkinen liikenne ja messut.

Digital signage -näytöt koostuvat suuresta osasta erilaisia näyttöratkaisuja. Näyttöjen käyttötarkoituksia on myös monia, mm. mainostus ja jonkun tiedon esittäminen. Yleisimmin käytetyt sisällön muodot ovat kuvat, videot ja tieto. Tietoa on esimerkiksi julkisen liikenteen aikataulut tai työpaikan projektin eteneminen. Tieto voidaan toistaa esimerkiksi HTML-sivuna näytöllä.

Digital signagesta mainostajille tekee haluttavampaa se, että näytöistä voi vaihtaa sisällön nopeammin ja kohdistaa mainontaa esimerkiksi vain yhdelle päivälle tiettyyn kellonaikaan, toisin kuin normaalissa ei digitaalisessa mainostaulussa.

Digital signage -näyttöjä löytyy kaiken kokoisia eri käyttötarkoituksiin. Suurimmat ovat kooltaan useita tuhansia neliömetrejä, joita löytyy esimerkiksi joistain ostoskeskuksista ja stadioneilta. Pienimmät näytöt voivat olla taas vain muutaman tuuman kokoisia.

Digital signagea ei ole esimerkiksi tavalliset mainostaulut tai mainostaulut joissa vaihtuu paperinen juliste.

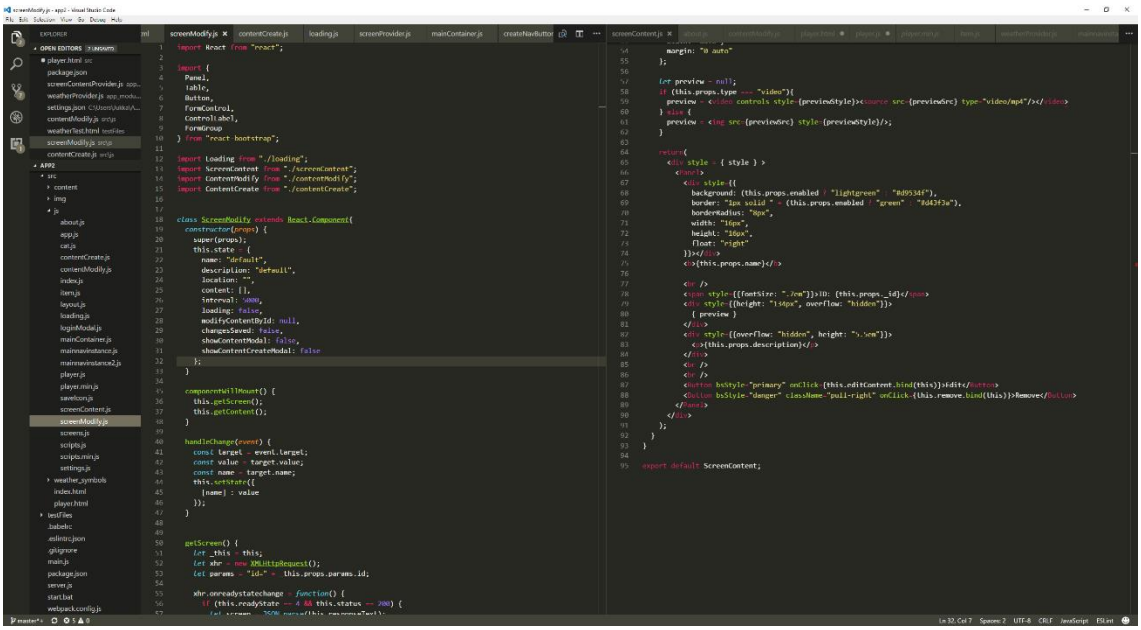
4 TYÖVÄLINEET JA TEKNOLOGIAT

4.1 Visual Studio Code

Visual Studio Code on Microsoftin kehittämä avoimen lähdekoodin tekstieditori, joka on rakennettu Electronin päälle. Electronilla voidaan rakentaa työpöytäsovelluksia käyttäen JavaScriptiä, HTML:ää ja CSS:ää. Visual Studio Coden voi asentaa Microsoftin Windows-, Applen Mac OS- ja Linux -alustoille. (Peter Bright, 30.4.2015.)

Visual Studio Codeen löytyy paljon ladattavia lisäosia esimerkiksi syntax highlighter JSX-tiedostoille, joita tässä projektissa käytettiin (Kuva 1).

Tein projektin alussa editorivertailuja ja päädyin käyttämään Visual Studio Codea pienen testailun jälkeen. Päätökseen vaikutti se, että ohjelmaan on sisäänrakennettu Git-hallinta, enkä ollut aiemmin ohjelmoinut mitään projektia Visual Studio Codella. Visual Studio Code pitää myös itsensä automaattisesti päivitettynä.



Kuva 1. Visual Studio Code editori

4.2 Git

Git on maailman eniten käytetty versionhallintajärjestemä ja kovaa vauhtia tulossa versiohallinnan standardiksi (Kayla, n.d.). Se on ilmainen ja avointa lähdekoodia, sitä voidaan käyttää niin isoissa kuin pienissäkin ohjelmistoprojekteissa nopeuttamaan ja selkeyttämään työntekoa (Git-scm n.d.).

Git pitää kirjaa kaikista tiedostoihin tehdyistä muutoksista. Git on hajautettu versionhallintajärjestelmä, mikä tarkoittaa sitä, että kaikilla jotka työskentelevät projektin parissa gitissä, on pääsy koko projektin historiaan, ei pelkästään tiedostojen nykyiseen tilaan. (Peter Bell, 2015.)

Päädyin käyttämään gittiä, koska se on yleisimmin käytössä ollut versionhallintajärjestelmä eteen tulleissa työpaikoissa.

4.3 **MongoDB**

MongoDB on avoimen lähdekoodin dokumenttipohjainen NoSQL-tietokantaohjelma. Dokumentit tallennetaan JSON-tyylisesti siten, että taulussa olevien dokumenttien ei kaikkien tarvitse sisältää samoja kenttiä. (MongoDB, n.d.)

NoSQL-tietokanta eroaa relaatiotietokannoista mm. niin, että taulujen väleillä ei ole suoraan liitoksia. Kaikki liitokset pitää tehdä ohjelman puolella.

Valitsin MongoDB:n projektiin mukaan, koska se sopi käyttötarpeeseeni, sitä käytetään usein Node.js:n kanssa ja minulla oli jo hieman kokemusta sen käytöstä.

4.4 **Node.js**

Node.js on palvelinpuolen JavaScript-ajoympäristö, mikä on rakennettu googlen avoimen lähdekoodin JavaScript-moottorilla nimeltä V8. Nodella voidaan tehdä myös työpöytäsovelluksia, esimerkiksi Electronia tai Nw.js -ohjelmia käyttäen. Noden alkuperäinen kehittäjä on Ryan Dahl, joka kehitti Noden vuonna 2009. (TrainingDotCom, 2016.)

Valitsin Noden projektiin, koska minulla oli jo aiempaa kokemusta sen käytöstä.

4.5 **Npm**

Npm (Node package manager) on pakettienhallintaohjelma JavaScript-ohjelmistojen kehittäjille. Npm mahdollistaa pakettien (kutsutaan myös moduuleiksi) jakamisen muiden käyttäjien kesken. (Npmjs, n.d.)

Valitsin npm:n käyttöön projektiin, koska se tulee jo valmiiksi Node.js-asennuksen mukana ja minulla oli jo aiempaa kokemusta sen käytöstä.

4.6 Express.js

Express.js on pieni avoimenlähdekoodin sovelluskehys Node.js:lle, joka tarjoaa sarjan ominaisuuksia web- ja mobiilisovelluksille. (Expressjs.com, n.d.) Expressillä käsitellään palvelimelle tulevat pyynnöt.

Valitsin Expressin projektiin, koska Express on Noden yksi suosituimmista sovelluskehyksistä web-applikaatiolle.

4.7 React.js

React on avoimen lähdekoodin JavaScript-kirjasto, jolla rakennetaan web-käyttöliittymiä komponenttien avulla. Reactin on kehittänyt Facebook.

React-komponentti voi olla lomakkeen input, button tai mikä tahansa elementti käyttöliittymässä (SurviveJS, n.d.). React-komponentit voivat sisältää jokainen oman tilan (state), jolla voidaan esimerkiksi määritellä komponentille omat muuttujat, joita voidaan käyttää komponentin piirtämiseen tilan mukaan. React-komponentille voidaan myös antaa ominaisuuksia (props), joihin komponentti ei voi itse vaikuttaa. Reactin kanssa käytetään usein JSX:ää, jolla voidaan lisätä JavaScript-koodin sekaan XML:ää. Reactia voi käyttää myös ilman JSX:ää. Alla esimerkkikoodi React-komponentista, joka sisältää tilan ”isBlue”, jota saa vaihdettua buttonia painamalla.

```
import React from "react";
import ReactDOM from "react-dom";

class Cat extends React.Component {
  constructor(props){
    super(props);
    this.state = {
      isBlue : false
    };
  }
  toggleBackgroundColor(){
    this.setState({
      isBlue : !this.state.isBlue
    });
  }
  render(){
    return (
      <div style={{backgroundColor : (this.state.isBlue ?
"#aaf" : "#fff")}}>
        <h1>Cat</h1>
        <p>Name: {this.props.name}</p>
        <p>Weight: {this.props.weight} kg</p>
        <p>Breed: {this.props.breed}</p>
```

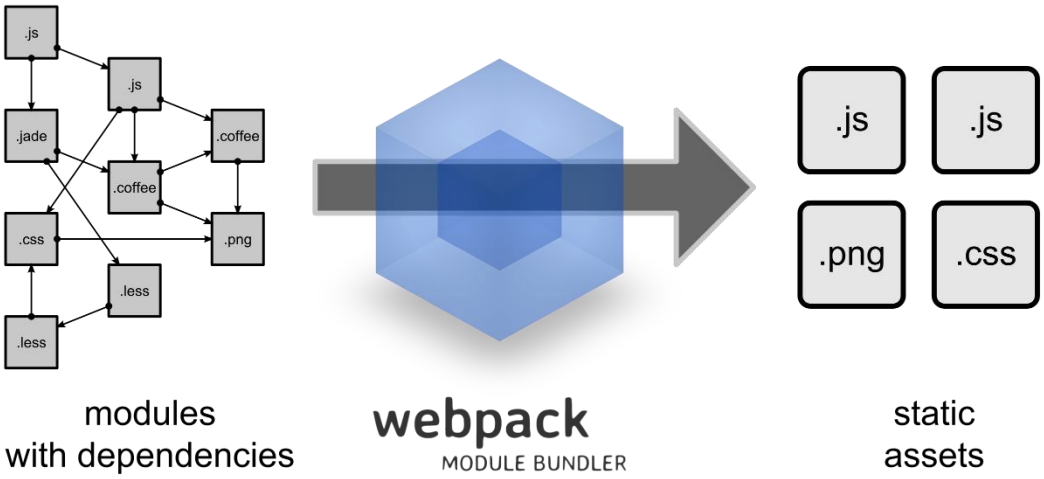
```
        <button
onClick={this.toggleBackgroundColor.bind(this)}> Blue
background on/off </button>
      </div>
    );
  }
}

ReactDOM.render(
  <Cat name="Kalle" weight="10" breed="Siamese" />,
  document.getElementById("app")
);
```

Reactista on myös React Native -versio, jolla voi rakentaa JavaScriptiä ja Reactia käyttäen mobiiliapplikaatioita joka ei kuitenkaan ole ”web-mobiiliapplikaatio” tai ”HTML5-applikaatio”. Tyylittely ei tapahdu käyttäen CSS:ää vaan tyylit määritellään suoraan JavaScript-objekteilla. Tyylin määrittely tapahtuu kuitenkin CSS:stä tuttujen nimien kanssa sanat yhdistettynä isoilla kirjaimilla ilman väliviivoja. Esimerkiksi CSS:llä taustan väri määriteltäisiin punaiseksi ”background-color : red ” ja JavaScript-objektissa se määriteltäisiin ilman väliviivaa: ”backgroundColor : ’red’”. React Native käyttää siis samoja palikoita applikaation rakentamiseen kuin normaalit iOS ja Android mobiiliapplikaatiot (React Native, n.d.).

Päädyin käyttämään Reactia projektissa, sillä React-osaajille vaikuttaa olevan tällä hetkellä kysyntää työmarkkinoilla, sekä AngularJS-kirjastoon verrattuna React vaikutti hieman kiinnostavammalta. Tykkäsin myös enemmän Reactin tavasta lisätä HTML:ää JavaScriptin sekaan kuin Angularin tavasta lisätä JavaScriptiä HTML:n sekaan.

4.8 **Webpack**

Webpack on eri tiedostojen kuten JavaScript-moduulien yhteen niputtaja. Se voi auttaa mm. web-sivustojen latausaikoihin, kun halutut tiedostot on saatu nippuun yhteen tiedostoon niin, että palvelimelle ei tarvitse tehdä joka tiedostolle omaa kutsua. Webpackiin löytyy myös lisäosia, joilla saa esimerkiksi kuvatiedostot optimoitua. Alla havainnekuva Webpackin toiminnasta (Kuva 2).



Kuva 2. Webpack module bundler (Webpack, n.d.).

4.9 **Babel**

Babel on avoimen lähdekoodin JavaScript kääntäjä, joka kääntää ei vielä tuettua, seuraavan sukupolven JavaScriptiä nykyselaimille sopivaan muotoon. Babelista löytyy myös oma lisäosa Reactia varten, joka lisää tuen JSX-syntaksille. (Babeljs.io, n.d.)

Otin Babelin käyttöön projektissa, jotta pääsen käyttämään JavaScriptin uusimpia ominaisuuksia projektissani.

4.10 **Bootstrap**

Bootstrap on avoimen lähdekoodin tuote responsiivisten käyttöliittymien ja nettisivujen tekoon. Alunperin Bootstrapin on kehittänyt Mark Otto ja Jacob Thornton, jotka olivat vielä Bootstrapin julkaisun yhteydessä töissä Twitterillä. (Jake, 2013.)

Bootstrap sisältää CSS-, kuva- ja JavaScript-tiedostot käyttöliittymiä varten. Tässä opinnäytetyössä on kuitenkin käytetty vain CSS tiedostoa, josta saa tyylit suureen osaan HTML elementeistä. Alla esimerkki, jossa on Chrome-selaimessa Bootstrapillä tyylitelty button ja tyylitön button (Kuva 3).

Kuva 3. Bootstrap button ja normaali button

4.11 YR.no REST API

YR on norjalaisten ilmatieteen laitoksen ja valtion viestintäyhtiö NRK:n omistama säänennustepalvelu. YR.no nettisivu tarjoaa säätiedot yli 10 miljoonasta paikasta ympäri maailmaa. Tiedot ja ohjeet ilmaisen säätiedon hakemiseen on ilmoitettu tällä hetkellä suurimmaksi osaksi vain norjaksi. Palvelun käyttöohjeiden tarjoamista englanniksi ei ole, koska YR pelkää, että palveluun tulisi liikaa liikennettä. (YR, 2016.)

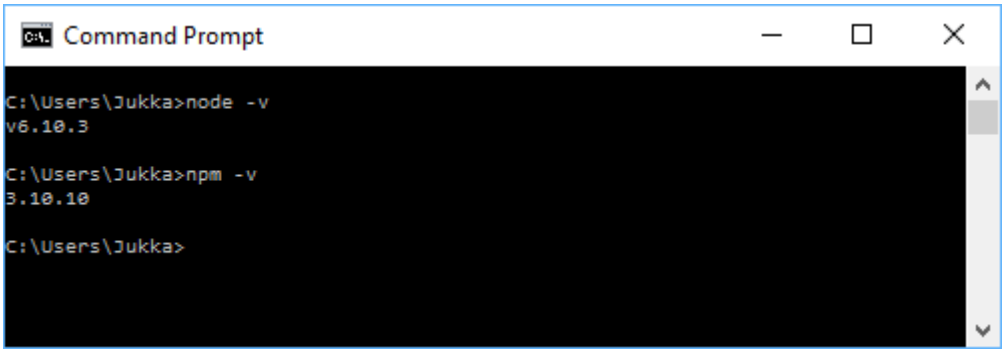
Päädyin käyttämään YR:n sääpalvelua, koska se riittää tarpeisiini ja se on ilmainen.

5 OHJELMIEN ASENNUS

Tämä luku kertoo, miten tärkeimmät käyttämäni ohjelmat projektin kehityksessä asennetaan. Käyttöjärjestelmänä toimii Windows 10.

5.1 Node.js

Node.js-ohjelman asennuspaketin saa ladattua osoitteesta https://nodejs.org/en/download/. Asennuksen yhteydessä asentuu koneelle myös Noden käyttämä npm, joka on lisätty asennuspakettiin. Npm ei kuitenkaan kuulu Node-ohjelman ytimeen. Asennuksen jälkeen voi testata Noden toimivuuden komentorivillä kuvan 4 esimerkin mukaan.



Kuva 4. Node.js asennus

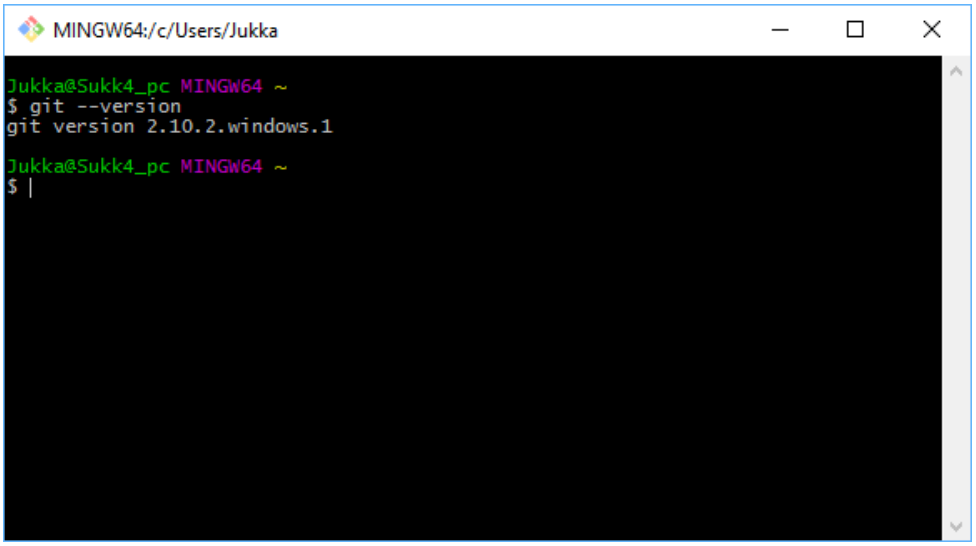
5.2 Visual Studio Code

Visual Studio Coden asennuspaketti ladataan osoitteesta https://code.visualstudio.com/Download. Määrittelin vielä omaan käyttööni Visual Studio Codeen väriteemaksi ”Monokai” ja tabien koon kahden välilyönnin kokoisiksi. Lisäsin myös ESLint-pluginin, joka näyttää virheet koodissa. Alla ES-Lint-pluginin konfigurointitiedosto ”.eslintrc.json”.

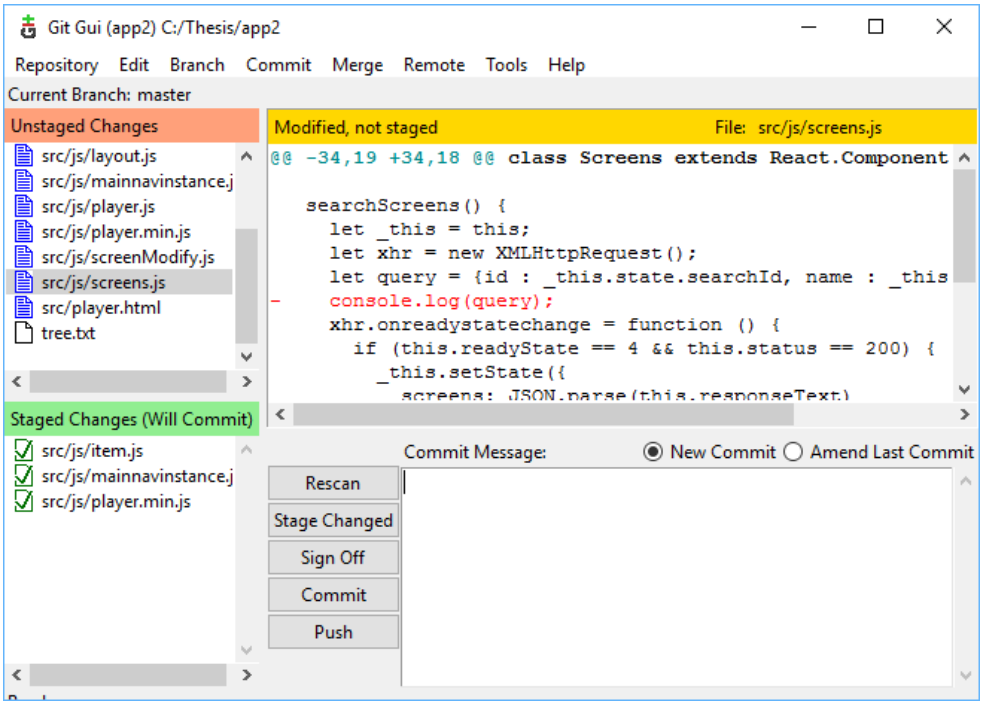
```
{
    "env": {browser": true, es6": true},
    "parserOptions": {
        "ecmaFeatures": {
            "experimentalObjectRestSpread": true,
            "jsx": true
        },
        "sourceType": "module"
    },
    "plugins": ["react"],
    "rules": {
        "indent": ["error",2, { "SwitchCase": 1 }],
        "linebreak-style": ["error","windows"],
        "quotes": ["error","double"],
        "semi": ["error","always"]
    }
}
```

5.3 Git

Gitin asennuspaketti ladataan osoitteesta https://git-scm.com/downloads. Asensin Gitin mukana myös Git Bashin (kuva 5) ja Git GUI:n (kuva 6), joilla gittiä voi käyttää joko komentoriviltä tai graafisella käyttöliittymällä. Myös GitHubille olisi oma työpöytäsovellus, mutta en nähnyt tarpeelliseksi sen käyttöä.



Kuva 5. Git Bash MinGW



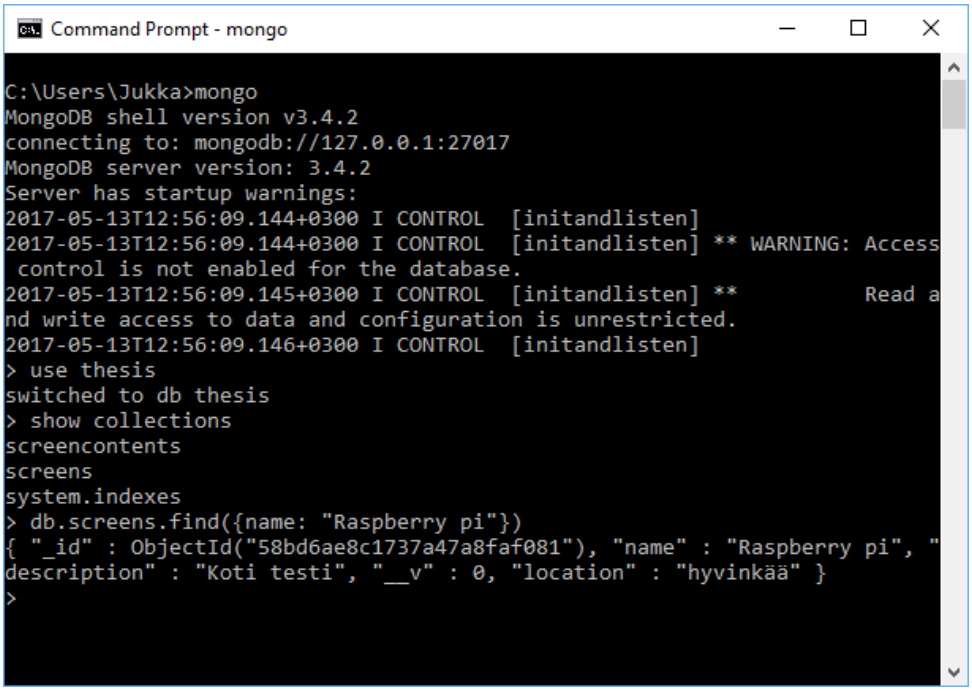
Kuva 6. Git Gui käyttöliittymä.

5.4 **MongoDB**

MongoDB:n asennuspaketti ladataan osoitteesta https://www.mongodb.com/download-center. Kun MongoDB on asennettu, tarvitsee MongoDB vielä oman hakemiston, minne tietokannat tallennetaan. Oletusarvona MongoDB:lle määritelty hakemisto on Windows-ympäristössä ”C:\data\db”. MongoDB:n asennus ei kuitenkaan välttämättä itse luo kansioita, vaan ne pitää itse tehdä manuaalisesti. Datakansion sijainnin manuaalisen määrityksen voi tehdä MongoDB:n avaamisen yhteydessä komennolla ”mongod —dbpath c:\haluamasi\sijainti” tai muokkaamalla MongoDB:n config-tiedostoa.

Käyttääksesi komentoriviltä komentoja ”mongod” ja ”mongo” ilman, että menet oikeaan hakemistoon, pitää polut olla lisättynä Windowsin PATH-ympäristömuuttujaan mongod.exe:lle ja mongo.exe:lle. Molempien oletushakemisto Windows-ympäristössä on ”C:\Program Files\MongoDB\Server\3.4\bin”.

Asennuksen onnistumisen jälkeen toimivuutta voi testata komentorivillä komennolla ”mongod”, joka käynnistää MongoDB-prosessin. Kun MongoDB on päällä, voidaan tietokantoja hallita komennolla ”mongo”, joka avaa hallintakonsolin (Kuva 7).



Kuva 7. MongoDB:n hallinta komentoriviltä

6 SOVELLUKSEN SUUNNITTELU

Sovelluksen suunnittelu alkoi päässäni jo työharjoittelussa, missä kuulin, että olisi kätevää, jos näyttöviestinnässä olisi enemmän ”älyä”, kuten vaikka sään mukaan tapahtuva sisällön näyttäminen. En varsinaisesti kirjoittanut mitään sen suurempaa suunnitelmaa, valitsin vain kiinnostavia palikoita, joista voisin lähteä toteuttamaan sisällönhallintajärjestelmää.

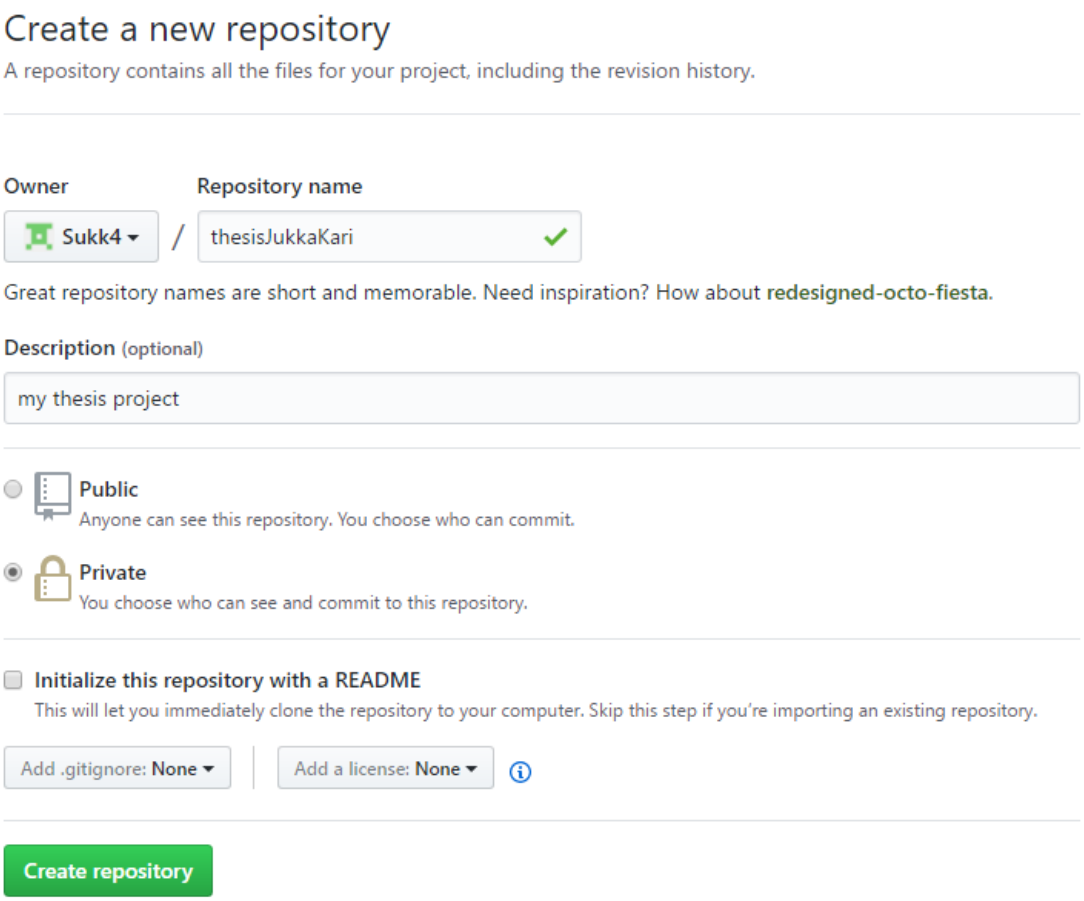
Työharjoittelussa minulle oli selvinnyt suunnilleen miltä sisällönhallintajärjestelmän pitäisi näyttää ja kuinka se toimii. Näiden tietojen pohjalta lähdin pohtimaan ja rakentamaan omaa käyttöliittymääni. En piirtänyt mitään suunnitelmaa, vaan pidin ideat päässäni ja tein muutoksia suunnitelmiin projektin tekemisen yhteydessä.

Ennen projektin aloittamista kunnolla, hankin vielä osaamista Reactista ja Webpackistä. Tein muutaman tutoriaalin ja luin dokumentaatiota, etten tekisi isoja virheitä, joita voisi olla myöhemmin vaikea korjata.

7 SOVELLUKSEN TOTEUTUS

7.1 Projektin luonti ja konfigurointi

Aloitin projektin luonnin tekemällä GitHubiin yksityisen repositoryn (Kuva 8). Kun projekti oli luotu GitHubiin, loin projektikansion omalle koneelle ja tein uuden git repositoryn komentorivin kautta juuri luotuun kansioon. Lisäsin aluksi kansioon tyhjän README.md-tiedoston ja .gitignore-tiedoston, joka kertoo gitille mitä tiedostoja ei tallenneta. Lisäsin .gitignore-tiedostoon mm. kansion ”node_modules”, jonne npm asentaa tarvittavat kirjastot ja moduulit.



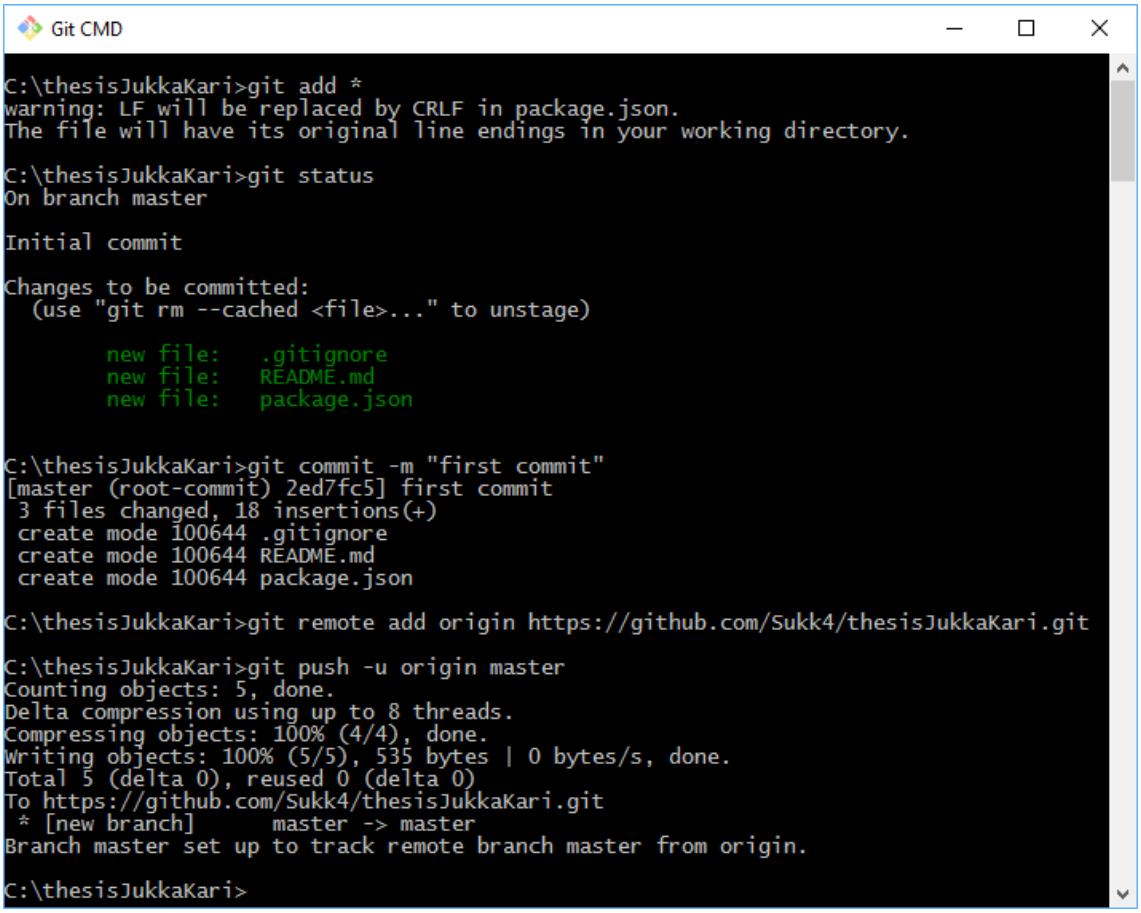
Kuva 8. GitHub projektin luonti

Seuraavaksi ajoin komentorivillä komennon ”npm init”, jolla luodaan automaattisesti package.json-tiedosto projektille vastaamalla kysymyksiin. Alla esimerkki package.json-tiedostosta.

```
{
  "name": "thesisjukkakari",
  "version": "1.0.0",
  "description": "digital signage cms",
```

```
  "main": "main.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Jukka Kari",
}
```

Kun olin saanut projektin pohjan valmiiksi, tallensin muutokset, lisäsin ulkopuoliseksi työhakemistoksi aiemmin GitHubiin luodun repositoryn ja tallensin projektin GitHubiin kuvassa näkyvillä komennoilla (kuva 9).

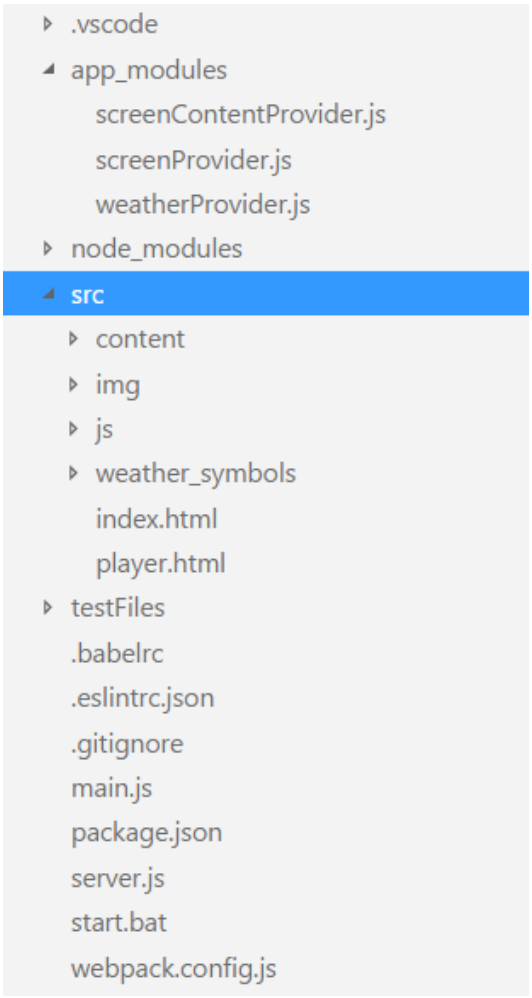


Kuva 9. Projektin tallennus GitHubiin (Jukka Kari, 2017)

7.2 Kansiorakenne

Projektin juuri sisältää määrittelytiedostoja projektille, palvelimen koodin sekä start.bat-tiedoston, joka käynnistää palvelimen nodemonilla, MongoDB:n, Webpackin ja selaimen oikeaan osoitteeseen. ”/app_modules”-kansio sisältää kaikki projektin omat moduulit palvelimen puolelle. ”/node_modules”-kansio sisältää kaikki npm:n avulla asennetut moduulit. ”/src”-kansio sisältää HTML-tiedostot sisällönhallinnalle ja sisällöntoistolle. ”/src”-kansio sisältää myös oman kansion ”/src/content” näytöille tallennettavaa sisältöä varten, ”/src/img”-kansion, joka sisältää kuvia sisäl-

lönhallintajärjestelmää varten, ”/src/js” -kansion, joka sisältää kaikki selainpuolen JavaScript-tiedostot ja ”/src/weather_symbols”-kansion, joka sisältää kuvat sääsymboleille. ”/testFiles”-kansio sisältää tiedostoja, joilla voi testata sisällön lisäämistä näytölle ja toistoa näytöllä. (Kuva 10)



Kuva 10. Kansiorakenne kuvattuna Visual Studio Code:ssa

7.3 Uuden riippuvuuden lisääminen projektiin

Uuden JavaScript-kirjaston tai riippuvuuden lisääminen projektiin käy npm-komennon kautta. Kirjoittamalla komentorivillä esim. ”npm install react –save”, tallentaa npm package.json tiedostoon Reactin projektin riippuvuudeksi ja asentaa Reactin ja sen riippuvuudet ”node_modules”-kansioon. Vain kehitystä varten olevat kirjastot lisätään esim. komennolla ”npm install nodemon –save-dev”, joka lisäisi nodemonin kehitysympäristön riippuvuudeksi. Alla esimerkki package.json-tiedostosta uuden asennetun moduulin jälkeen.

```
{
  "name": "thesisjukkakari",
  "version": "1.0.0",
  "description": "digital signage cms",
```

```
  "main": "main.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Jukka Kari",
  "dependencies": {
    "react": "^15.4.2"
  },
  "devDependencies": {
    "nodemon": "^1.11.0"
  }
}
```

Mikäli haluaa lisätä riippuvuuden package.json-tiedostoon käsin, pitää riippuvuus asentaa erikseen komentoriviltä komennolla ”npm install”.

7.4 Palvelimen pohja

Palvelimen puolella kaikkia kutsuja ja kyselyitä hoitaa server.js-tiedosto. Tiedostossa määritellään mitä moduuleita on käytössä. Mongoose:lle määritellään käytettävä MongoDB-tietokanta.

Palvelimella on toteutettu käyttäen express.js-kehystä, joka hoitaa tässä tapauksessa palvelimelle tulevat HTTP GET- ja POST-pyynnöt. Expressille on myös lisätty middlewareksi body-parser ja multer, jotka hoitavat kyselylomakkeet ja tiedostojen vastaanottamisen palvelimelle. Applikaatiolle on määritelty kuunneltavaksi portti 8088. Alla pari esimerkkiä, kuinka GET- tai POST-pyyntö hoidetaan.

```
app.get("/", (req, res) => {
  res.sendFile(__dirname + "/src/index.html");
});
app.get("/createScreen", (req, res) => {
  screenProvider.create((err, screen) => {
    if(err){
      res.send(err);
    } else {
      res.send(JSON.stringify(screen));
    }
  });
});
app.post("/weather", (req, res) => {
  weatherProvider.getWeather(req.body.location, (err, data) =>
{
    if (err) {
      console.log(err);
      res.send(err);
    } else {
```

```
      res.send(data);
    }
  });
});
```

7.5 **Navigointi**

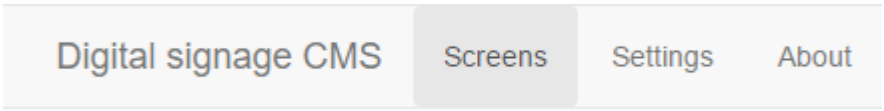
7.5.1 React Router

Sisällönhallintapaneelin index.js-tiedosto pitää sisällään reitittäjän ”React Router”, jolla saadaan web-applikaatiosta ”single page application” (SPA) eli yhden sivun aplikaatio, vaikka sivuja onkin oikeasti monta. Tämä poistaa turhat sivulataukset. React Router pitää myös huolen applikaation sivuhistoriasta hashHistory:llä. Jokaiselle sivulle on oma Route-komponentti, johon on määritelty sivun polku. Kaikki sivut on lisätty Routen sisään, johonka on määritelty komponentiksi Layout.

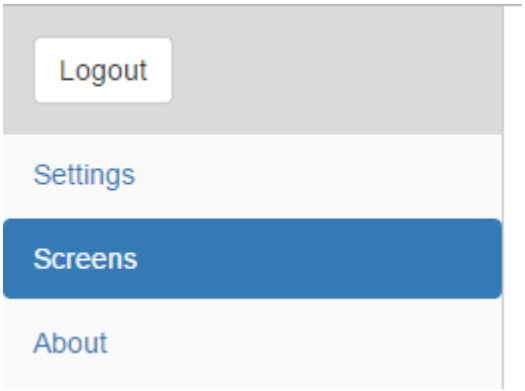
```
import React from "react";
import ReactDOM from "react-dom";
import {Router, Route, IndexRoute, hashHistory} from "react-
router";

import Layout from "./layout";
import Settings from "./settings";
import About from "./about";
import Screens from "./screens";
import ScreenModify from "./screenModify";
const app = document.getElementById("app");

ReactDOM.render(
    <Router history={hashHistory}>
        <Route path="/" component={Layout}>
            <IndexRoute component={About}/>
            <Route path="/settings" component={Settings}/>
            <Route path="/screens" component={Screens}/>
            <Route path="/screen/:id" component={ScreenMod-
ify}/>
            <Route path="/about" component={About}/>
        </Route>
    </Router>,
app);
```

7.5.2 Navigaatiopalkki

Linkit React Routerille ja sivun päänavigaatio on tiedostossa ”mainnavinstance.js”. Navigaatio on toteutettu käyttäen Bootstrapin elementtejä, sekä linkit käyttäen ”react-router-bootstrap”-kirjastoa (Kuva 11) (Kuva 12).



Kuva 11. Sivun lopullinen navigaatiopalkki



Kuva 12. Sivulle suunniteltu vaihtoehtoinen navigaatiopalkki

Alla esimerkkikoodi navigaatiopalkista.

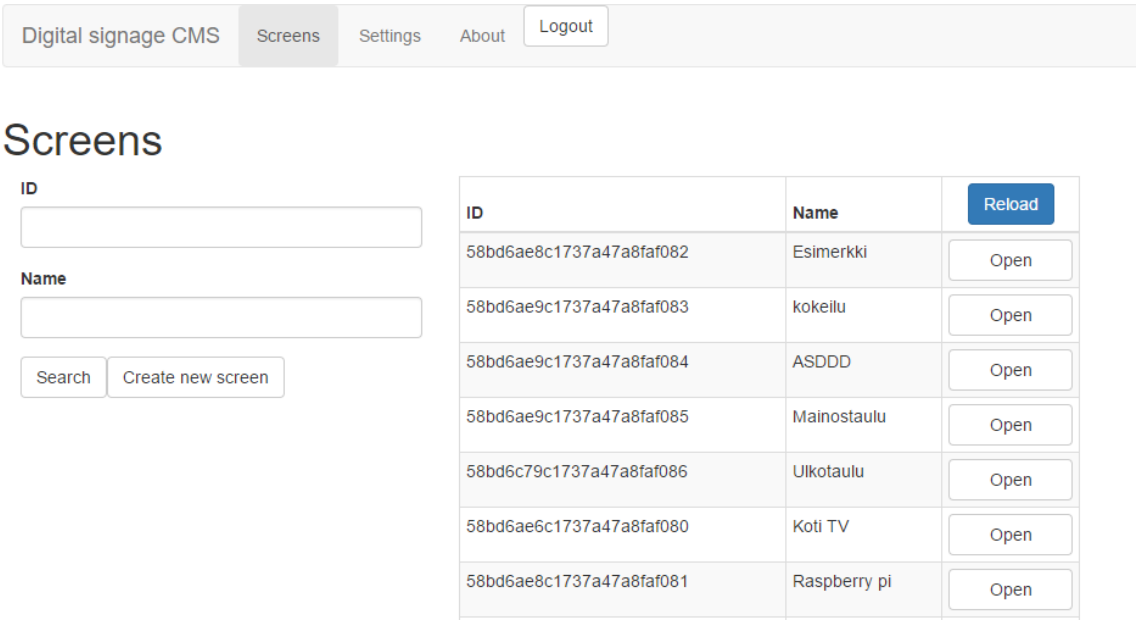
```
render() {
  return (
    <div>
      <Navbar>
        <Navbar.Header>
          <Navbar.Brand>
            <a href="#">Digital signage CMS</a>
          </Navbar.Brand>
        </Navbar.Header>
        <Nav bsStyle="pills">
          <LinkContainer to="/screens">
            <NavItem eventKey={2}>Screens</NavItem>
          </LinkContainer>
          <LinkContainer to="/settings">
            <NavItem eventKey={3}>Settings</NavItem>
          </LinkContainer>
          <LinkContainer to="/about">
            <NavItem eventKey={4}>About</NavItem>
          </LinkContainer>
        </Nav>
      </Navbar>
```

```
    </div>
  );
}
```

7.6 Näyttöjen haku

7.6.1 Selaimella

Kun Screens-sivu avataan, haetaan tauluun heti kaikki näytöt. Näyttöjä voi hakea erikseen näytön id:n ja nimen perusteella, tai vain selaten taulusta (Kuva 13). Kirjoittamalla kenttiin hakukriteerit ja painamalla ”Search”, lähetetään palvelimelle AJAX-pyyntö funktiolla ”searchScreens”. Kun palvelin on käsitellyt haun, lähettää palvelin löydetyt näytöt selaimelle ja selain päivittää taulun. ”Reload”-painikkeella haetaan kaikki näytöt uudelleen.



Kuva 13. Näyttöjen haku ja hallinta.

Screens-sivulla on oma React-komponentti. Komponentilla on oma tila, joka sisältää listan näytettävistä näytöistä sekä hakukenttien sisällön.
Kun hakukenttään kirjoitetaan, päivittää komponentti oman tilansa metodilla ”handleChange”. Alla esimerkit metodeista “handleChange” ja “searchScreens”.

```
handleChange(event) {
  const target = event.target;
  const value = target.value;
  const name = target.name;

  this.setState({
```

```
    [name] : value
  });
}

searchScreens() {
  let _this = this;
  let xhr = new XMLHttpRequest();
  let query = {id : _this.state.searchId, name :
_this.state.searchName};
  xhr.onreadystatechange = function () {
    if (this.readyState == 4 && this.status == 200) {
      _this.setState({
        screens: JSON.parse(this.responseText)
      });
    }
  };
  xhr.open("POST", "/screensQuery.txt", true);
  xhr.setRequestHeader("Content-type", "application/json");
  xhr.send(JSON.stringify(query));
}
```

Kaikkien näyttöjen näyttäminen taulussa tapahtuu seuraavasti Screens-komponentin ”render”-metodissa:

```
this.state.screens.map(function (item, i) {
  return (
    <tr key={i}>
      <td>{item._id}</td>
      <td>{item.name}</td>
      <td>
        <LinkContainer to={{ pathname: "/screen/" + item._id
}}>
          <Button style={{display: "block", margin: "auto"}}>
Open </Button>
        </LinkContainer>
      </td>
    </tr>
  );
})
```

7.6.2 Palvelimella

Palvelimella AJAX-pyyntöön vastaa server.js-tiedostossa määritellyt kohdat, jotka ohjaavat haun screenProvider-moduulille.

Näyttöjä voi etsiä metodilla ”findByQuery” johon voi lisätä hakuparametreiksi JavaScript-objektin, joka sisältää id:n ja nimen. Ilman parametrejä voi

hakea kaikki näytöt metodilla ”findAll”. Alla esimerkki haun toteutuksesta palvelimen puolella.

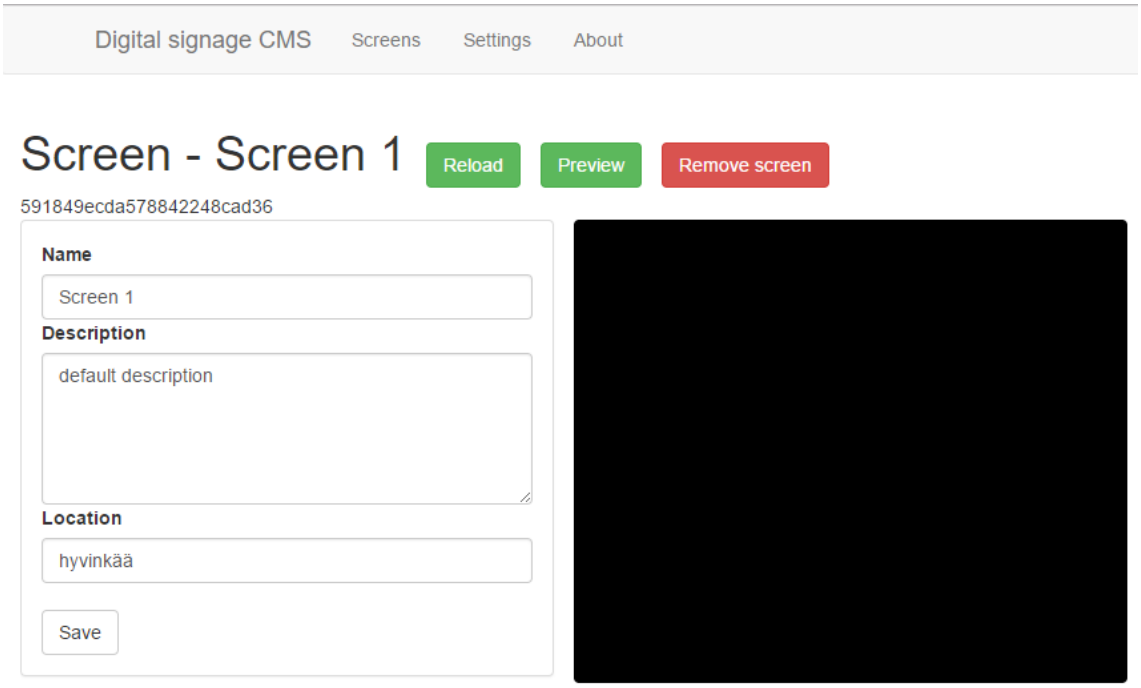
```
static findByQuery( QUERY, callback ){
  let query = {};
  if (QUERY.id !== "") query._id = QUERY.id;
  if (QUERY.name !== "") query.name = QUERY.name;
  screen.find(
    query
  , (err, data) => {
    callback(err, data);
  });
}
```

7.7 **Näyttöjen hallinta**

7.7.1 Selaimessa

Uuden näytön voi lisätä Screens-välilehdessä painamalla haun alla olevaa ”Create new screen”-painiketta (Kuva 13) Painikkeen painamiseen on liitetty funktio, joka tekee pyynnön palvelimelle, joka luo uuden näytön. Kun näyttö on luotu, lähettää palvelin näytön tiedot selaimelle ja selain avaa näytön muokkaussivun (Kuva 14). Näytön muokkaussivun osoite on ”/screen/näytönId”.

Näytön nimeä, kuvausta ja sijaintia voi muuttaa muokkaussivulla. Muutokset tallennetaan AJAX-pyynnöllä. Kun pyyntö on suoritettu, ilmoitetaan näytön tallennuksen onnistuminen pienellä ”muutokset tallennettu” ilmoituksella, jota näytetään muutama sekunti.



Kuva 14. Näytön muokkaus

Näytön voi poistaa näytön muokkaussivulla painamalla ”Remove Screen”-painiketta (kuva 14). Näyttöä poistaessa poistetaan myös kaikki näytölle annettu sisältö. Näytön poiston yhteydessä varmistetaan vielä, oletko varma, että haluat poistaa näytön ja kaiken sen sisällön. Kun näyttö on poistettu onnistuneesti, ohjataan selain sivulle ”Screens”.

Näytön sisältöä voi esikatsella mustasta laatikosta tai painamalla ”Preview”-painiketta, joka avaa sisällön toistajan uuteen ikkunaan (Kuva 14).

7.7.2 Palvelimella

Näyttöjen hakuun palvelimelta palvelimella on oma moduuli (screenProvider.js). Se sisältää luokan ”ScreenProvider”, joka sisältää joukon staattisia metodeita, joita voidaan kutsua ilman, että server.js-tiedostossa luodaan erikseen oma olio luokasta.

Jokainen näyttö tallennetaan MongoDB-tietokantaan käyttäen mongoose-kirjastoa. Tiedoston alussa luodaan näytölle oma malli tietokantaan alla olevan koodin mukaan.

```
import mongoose from "mongoose";

const schema = new mongoose.Schema({
  _id : mongoose.Schema.Types.ObjectId,
  name : {type: String},
  description : {type: String},
```

```
  location: {type: String},     // city for weather example :
"helsinki"
});

const screen = mongoose.model("Screen", schema);
```

Uuden näytön luomisessa generoidaan näytölle oma id ja annetaan oletus-arvot nimi, kuvaus ja sijainti. Tämä tapahtuu luokan “create”- metodissa.

```
static create ( callback ){
  let newScreen = new screen({
    _id : new mongoose.Types.ObjectId(),
    name : "default name",
    description : "default description",
    location : "hyvinkää"
  });
  newScreen.save((err, screen) => {
    if (err){
      console.log(err);
      callback(err);
    }
    else {
      callback(null, screen);
    }
  });
}
```

Näytön tietoja voi päivittää metodilla ”findByIdAndUpdate”, joka päivittää annetun näytön annettujen arvojen mukaan id:n perusteella.

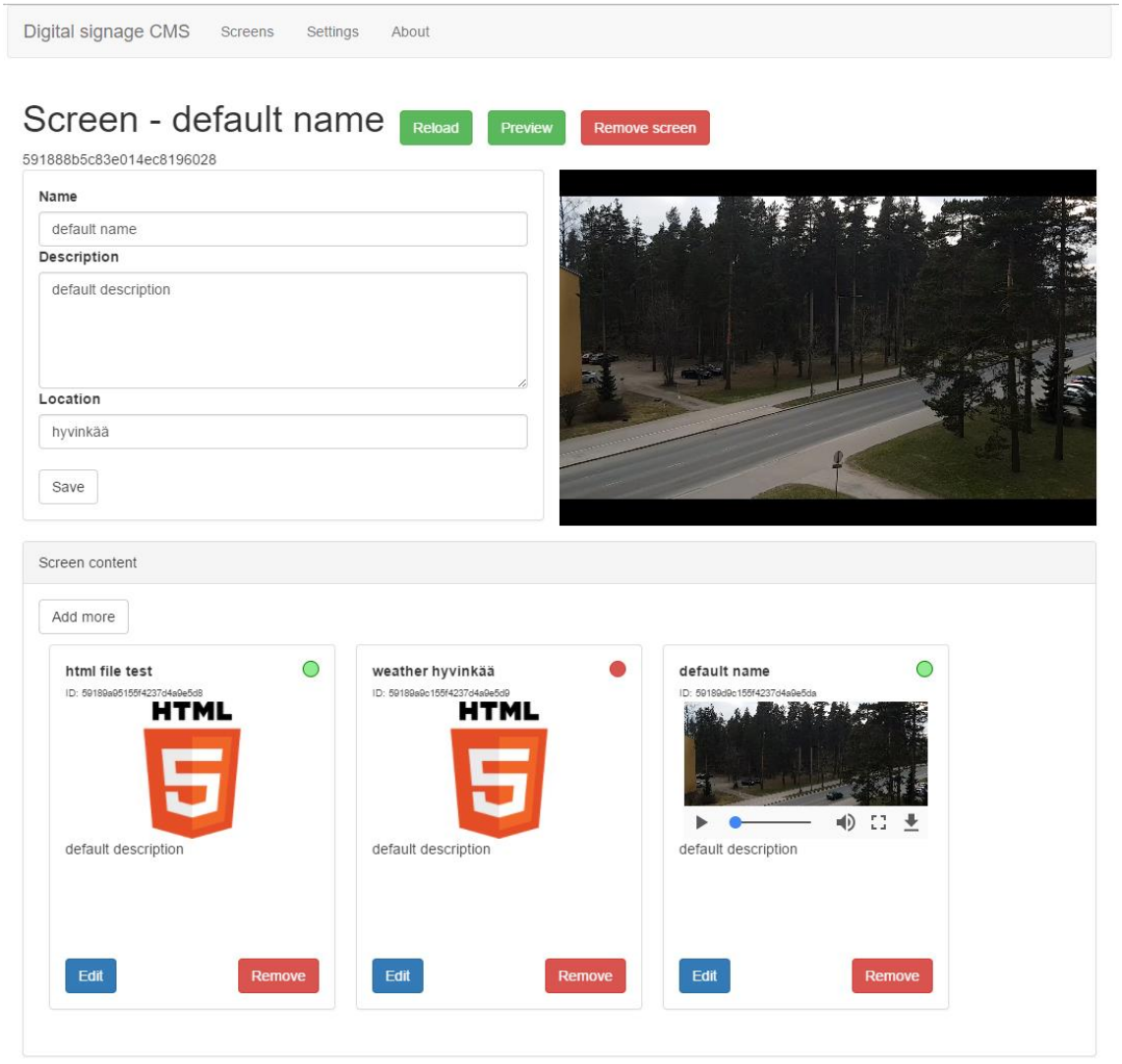
```
static findByIdAndUpdate( SCREEN , callback ){
  screen.findByIdAndUpdate(
    SCREEN.id,
    { $set: {name : SCREEN.name, description :
SCREEN.description, location: SCREEN.location} },
    (err, data) => {
      callback(err, data);
    }
  );
}
```

Näyttöjä voi poistaa id:n perusteella metodilla removeById. Server.js-tiedostossa on määritelty poistettavaksi myös kaikki poistettavan näytön sisältö.

7.8 **Näytön sisällönhallinta**

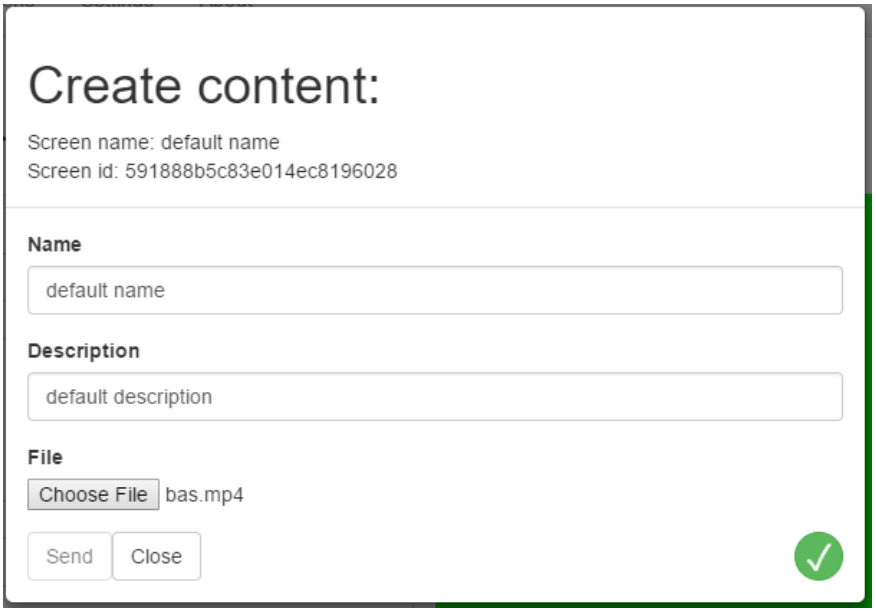
7.8.1 Selaimella

Näytön sisällönhallinta tapahtuu näytönhallinnan alapuolella osassa ”Screen content” (kuva 15). Sisältöä voi poistaa painamalla sisällön kohdalla ”Remove”-painiketta, joka varmistaa vielä, että ”oletko varma, että haluat poistaa tämän sisällön”. Sisällön oikeassa ylänurkassa on myös punainen tai vihreä pallo, jolla näkee onko sisältö määritelty näytettäväksi.



Kuva 15. Näytön sisällönhallinta.

Uutta sisältöä voi lisätä painamalla painiketta ”Add more” (kuva 15), jolloin aukeaa kehys, jossa voidaan määritellä sisällölle nimi, kuvaus ja tiedosto (kuva 16). Kehyksen ja sisällön lisäämisen koodi on tiedostossa ”contentCreate.js”. Kehyksen luonnissa on käytetty Bootstrapin modal-tyyliä.



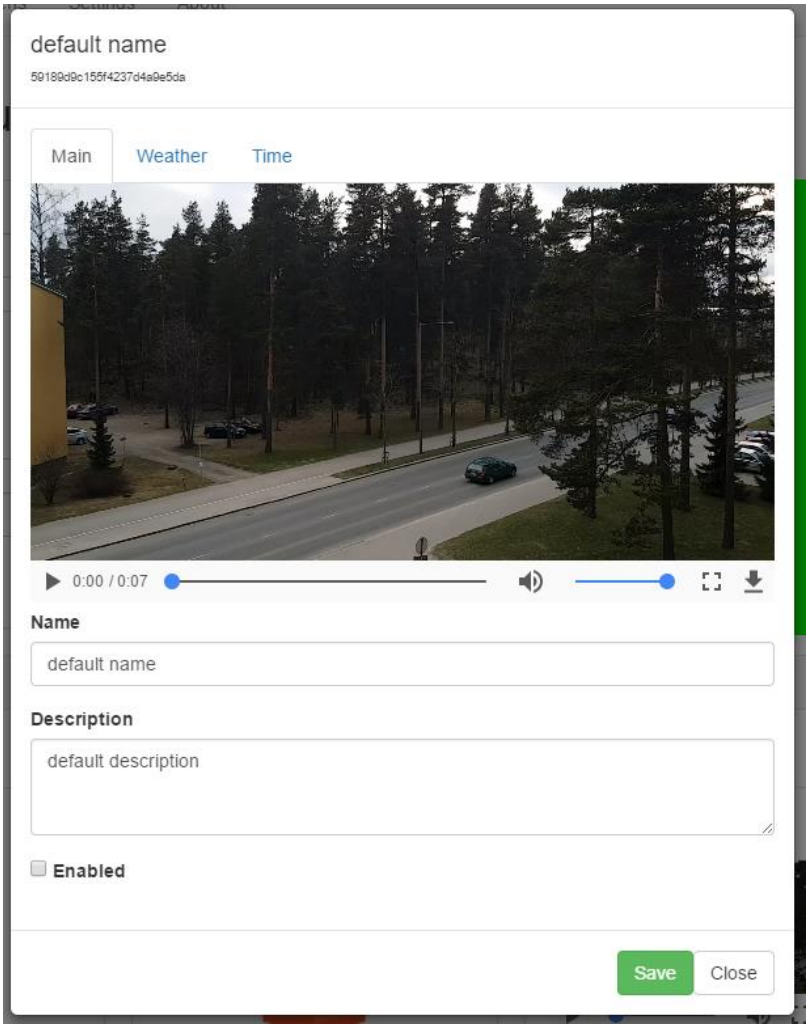
Kuva 16. Uuden sisällön lisäys

Alla koodiesimerkki uuden sisällön lähettämisestä palvelimelle ”uploadContent”-metodilla.

```
uploadContent(e){
  const _this = this;
  console.log("sending content");
  e.preventDefault();

  let xhr = new XMLHttpRequest();
  let formData = new FormData();
  let content = this.state.content;
  this.setState({saving: true});
  Object.keys(content).forEach((key) => {
    if (key !== "file"){
      formData.append(key, content[key]);
    }
  });
  formData.append("screen", this.props.screenId);
  formData.append("file", document.getElementById("newContent-
File").files[0]);
  xhr.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {

      if (this.responseText === "error"){
        alert("error");
        _this.setState({saving: false});
      }
      else {
        console.log("success");
        _this.setState({saved: true, saving: false});
      }
```

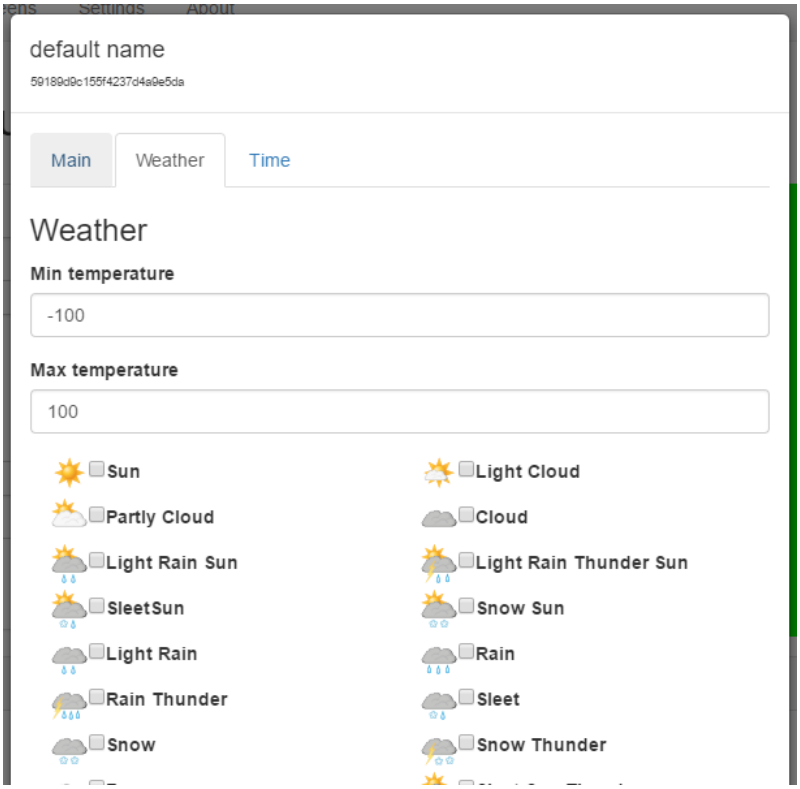
```
    }
  };
  xhr.open("POST", "/contentAdd", true);
  xhr.send(formData);
}
```

Sisällön muokkaus tapahtuu avaamalla sisältö painamalla ”edit”-painiketta, jolloin aukeaa kehys, jossa sisältöä pystyy muokkaamaan tarkemmin kuin sisällönluontivaiheessa. Kehykselle on oma komponentti tiedostossa ”screenModify.js”. Sisällön muokkaus koostuu välilehdistä ”Main”, ”Weather” ja ”Time”. ”Main”-välilehti sisältää sisällön esikatselun, nimen, kuvauksen ja ”enabled”-valintaruudun, jolla määritellään onko sisältö käytössä (Kuva 17).



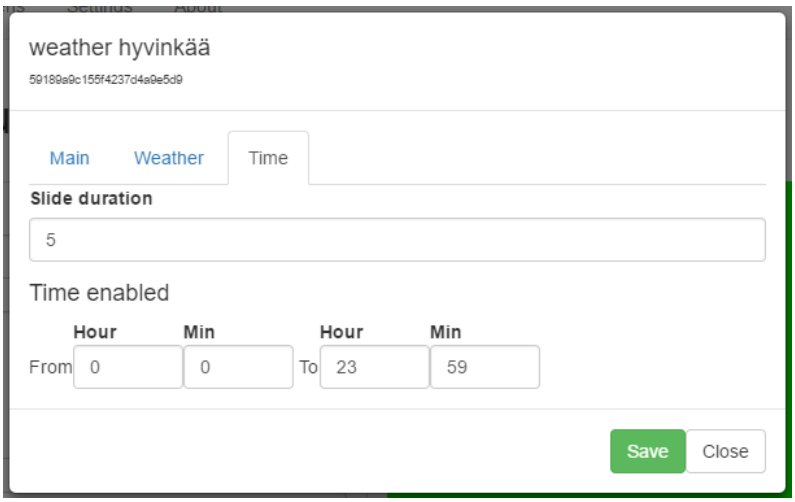
Kuva 17. Sisällön muokkaus ”Main”-välilehti

”Weather”-välilehdeltä voidaan määrittää minkälaisella säällä sisältöä halutaan näyttää. Jättämällä kaikki valintaruudut tyhjäksi, näkyy sisältö joka säällä. Sisällölle voidaan myös määritellä kuinka lämpöisellä ilmalla sitä näytetään (Kuva 18).



Kuva 18. Sisällönmuokkaus ”weather”-välilehti

”Time”-välilehdellä asetetaan sisällölle aika, kuinka kauan sisältöä näytetään kerralla sekunteina ja mihin aikaa päivästä sitä näytetään (Kuva 19).



Kuva 19. Sisällönmuokkaus ”Time” -välilehti

ContentCreate ja ContentModify -komponentit sisältävät molemmat tilan ”saved” ja ”saving”. Kun tietoja tallennetaan, asetetaan tila ”saving=true”, silloin oikeaan alakulmaan tulee merkki ”tallennetaan”. Kun tiedot on saatu tallennettua, asetetaan ”saved=true” ja ”saving=false”, jolloin oikeassa reunassa näytetään ”tallenetaan”-merkin tilalla vihreää ✓-merkkiä

(Kuva 16). Myös virhetilanteelle on tehty oma ”error”-merkki. Tämä tallennusmerkki on oma React-komponenttinsa, jonka tiedostonimi on ”saveIcon.js”. Alla koodiesimerkki ”saveIcon”-komponentistä.

```
// render saving icon
return (
  <div style={style}>
    { this.props.saving ? <span>s</span> : null }
    { this.props.saved ? <span>&#x2713;</span> : null }
    { this.props.error ? <span>error</span> : null }
  </div>
);
```

7.8.2 Palvelimella

Sisällön hakuun palvelimelta palvelimella on oma moduuli ”screenContentProvider.js”. Se sisältää luokan ”ScreenContentProvider”, joka sisältää joukon staattisia metodeita, joita voidaan kutsua ilman, että ”server.js”-tiedostossa luokasta luodaan erikseen oma olio.

Sisällön tiedot tallennetaan MongoDB-tietokantaan ja sisällön tiedosto tallennetaan erikseen kansioon ”/src/content”. Kun uutta sisältöä lisätään palvelimelle, hoitaa ”multer”-kirjasto tiedoston tallentamisen ja ”mongoose”-kirjasto tallentaa sisällön tiedot MongoDB-tietokantaan. Tiedoston nimeen lisätään myös lisäysaika, ettei palvelimelle tule samannimisiä tiedostoja. Alla koodinpätkä mongooselle määritellystä sisällöstä.

```
const schema = new mongoose.Schema({
  _id: mongoose.Schema.Types.ObjectId,
  name: {type: String},
  description: {type: String},
  screen: {type: mongoose.Schema.Types.ObjectId},
  type: {type: String},
  src: {type: String},
  enabled: {type: Boolean, default: true},
  duration: { type: Number, default: 5},
  minuteFrom: {type: Number, default: 0},
  hourFrom: {type: Number, default: 0},
  minuteTo: {type: Number, default: 59},
  hourTo: {type: Number, default: 23},
  weatherSymbol: [{type: Number}],
  weatherMaxTemperature: { type: Number, default: 100 },
  weatherMinTemperature: { type: Number, default: -100 }
});
```

Sisällön poistaminen tapahtuu sisällön id:n mukaan. Sisällön tiedoston poistamisessa käytetään noden ”file system” (fs) moduulia. Alla koodinpätkä sisällön poistamisesta.

```
static delete(contentID, callback) { // delete content
  let id = new mongoose.Types.ObjectId(contentID);
  screenContent.findByIdAndRemove(id, (err, data) => {
    if(!err){
      fs.unlink("src/" + data.src, (error) => { // delete file
        if (error){
          callback(error, data);
        } else {
          callback(null, data);
        }
      });
    } else {
      callback(err);
    }
  });
}
```

7.9 Sään haku ja tallennus palvelimella

Toteutin sään haun palvelimen puolella käyttämällä kirjastoja xml2js ja xmlhttprequest. xml2js-kirjasto kääntää tässä tapauksessa Yr:n sääpalvelusta haettavan XML-tiedoston JavaScript-objektiksi. Xmlhttprequest-kirjastolla saadaan tehtyä AJAX-hakuja samaan tapaan kuin selaimien XMLHttpRequest-objektilla.

Sää-moduulin nimi on weatherProvider.js, joka sisältää luokan WeatherProvider. Sää-moduulia luodessa rakentajaan on määritelty sään hakemisen päivitysväli, sääntarjoajan linkkiteksti ja kaupunkien säät ja linkit erikseen. Linkkiteksti sisältää Yr:n määrittämän tekstin, joka pitää näyttää sisällössä, jos säätietoa haluaa näyttää näytöllä. Kaupunkien sääennusteet sisältävät lämpötilan, tuulen suunnan, sääsymbolin sekä linkin mistä sää haetaan. Alla on esimerkki, kuinka sijainnin sää tallennetaan. Sijainnit joutuu lisäämään vielä tällä hetkellä käsin koodiin.

```
this.weather = [
  {
    name: "hyvinkää",
    temperature: undefined,
    symbol: undefined,
    symbolName: undefined,
    windDir: undefined,
    url: "http://www.yr.no/place/Finland/Southern_Fin-
land/Hyvinkää/"
```

```
  }
];
```

Säähän pääsee käsiksi näytöillä tekemällä palvelimelle AJAX-kyselyn osoitteeseen ”/weather” ja lisäämällä hakuparametriksi ”location=haluttuKaupunki”. Palvelin lähettää silloin vastaukseksi kaupungin sään JSON-muodossa.

7.10 **Sisällön toisto näytöllä**

Sisältöön pääsee käsiksi tietokoneen selaimella tai vaikka nykyaikaisten televisioiden selaimella menemällä osoitteeseen http://esimerkki-osoite.fi/player?id=näytönid . Osoitteeseen on liitetty parametriksi näytön id, jolla ohjelma tietää, mitä sisältöä näytöllä näytetään.

7.10.1 Sisällön toistaja

Sisällön toisto näytöllä on toteutettu myös käyttäen Reactia. Sisällön toistamiseen on HTML-tiedosto, joka sisältää sivun pohjan, linkin JavaScript-tiedostoon ja tyylin bodyn taustavärille. Alla esimerkki HTML-koodista.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>CMS thesis player</title>
    <link rel="stylesheet" href="https://maxcdn.boot-
strapcdn.com/bootstrap/latest/css/bootstrap.min.css">
    <style>
      body {
        background: #000;
      }
    </style>
  </head>
  <body>
    <div id="app"></div>
    <script src="js/player.min.js" charset="utf-8"></script>
  </body>
</html>
```

Kuten HTML-koodista huomaa, myös sisällöntoistajan skriptit on liitetty yhteen tiedostoon Webpackin ja Babelin avulla. ”player.min.js”-tiedosto koostuu kuitenkin vain kahdesta JavaScript-tiedostosta. Toinen niistä on ”player.js” ja toinen ”item.js”.

”player.js”-tiedosto sisältää luokan ”Player” joka sisältää koodin sisällön pyörittämistä varten. Player luokan rakentajassa määritellään komponentille tila, joka sisältää mm. tiedot säästä, näytöllä näytettävästä sisällöstä,

tällä hetkellä näytettävä sisällön indeksistä ja oletusarvo sille, kuinka kauan sisältöä näytetään. Rakentajassa määritellään myös pari tyyliä materiaaleja varten. Alla koodinpätkä rakentajassa asetettavasta tilasta ja tyyleistä.

```
this.state = {
  name : "No name",
  playing: false,
  currentItemIndex : 0,
  content : [],
  weather : {
    name: "",
    temperature: 0,
    symbolName: "",
    symbol: [],
    windDir: "",
    url: ""
  },
  interval : 10,
};

this.styles = {
  active : {
    display : "block"
  },
  default : {
    display : "none"
  }
};
```

Kun komponentti lisätään DOM:iin, kutsuu React automaattisesti ”componentDidMount”-metodia. Tämä metodi kutsuu muita komponentille määriteltyjä metodeja, joilla haetaan palvelimelta näytön tiedot, näytölle määritellyn sisällön ja sään näytön sijainnin mukaan.

```
componentDidMount(){
  this.getScreen(() => {
    this.getWeather();
  });
  this.getContent();
}
```

”getScreen”-metodilla haetaan AJAX:ia käyttäen palvelimelta näytölle näytön tiedot ja lisätään ne komponentin tilaan käyttämällä ”setState”-metodia. Kun tiedot on saatu haettua, kutsutaan callback-funktiota, joka on määritelty sään hauksi. Sää ja näytön sisältö haetaan myös samalla tavalla AJAX:lla ja lisätään komponentin tilaan. Sisällönhaun onnistuessa lisätään myös tilaksi ”playing : true”, että sisältö alkaa pyörimään näytöllä. Alla koodiesimerkki näytön tietojen hausta.

```
getScreen(callback){
  let _this = this;
  let xhr = new XMLHttpRequest();
  let params = {id : location.search.split("=")[1]};

  xhr.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      let screen = JSON.parse(this.responseText);
      _this.setState({
        name : screen.name,
        description: screen.description,
        location : screen.location,
        interval : screen.interval
      });
      callback();
    }
  };
  xhr.open("POST", "/screen.json", true);
  xhr.setRequestHeader("Content-type", "application/json");
  xhr.send(JSON.stringify(params));
}
```

Player-komponentin ”render”-metodissa lisätään DOM:iin kaikki näytölle määritelty sisältö eli Itemit. Jokaiselle Itemille lähetetään myös parametreina kaikki kyseisen sisällön data, ”next”-funktio, sää, ja tyyli sen mukaan onko kyseinen sisältö määritelty aktiiviseksi. Alla koodinpätkä ”render”-metodista.

```
render(){
  return(
    <div>
      {
        this.state.content.map( (item, i) => {
          return(
            <Item key={i} {... item}
currentWeather={this.state.weather}
next={this.nextItem.bind(this)}
style={(this.state.currentItemIndex === i) ?
this.styles.active : this.styles.default } />
          );
        })
      }
    </div>
  );
}
```

7.10.2 Sisältö

Item.js sisältää luokan Item, joka on React-komponentti. Luokan rakentajassa määritellään vain Itemille määritellyt parametrit ja tyylit kuville ja muille sisältötyypeille. Esimerkiksi jokaiselle itemille on annettu ”next”-parametri joka sisältää luokasta ”Player” metodin ”nextItem”. Tätä ”nextItem”-metodia Item käyttää silloin, kun sen määritelty näyttöaika loppuu ja pitää siirtyä seuraavan sisällön näyttämiseen. Alla esimerkki luokan rakentajasta.

```
constructor(props){
  super(props);
  this.imgStyle = {
    maxWidth: "100%",
    maxHeight: "100%",
    top: "50%",
    left: "50%",
    transform: "translateX(-50%) translateY(-50%)",
    position: "absolute"
  };
  this.style = {
    width: "100vw",
    height: "100vh",
    border: 0
  };
  this.timeOut;
}
```

Item sisältää metodin ”play”, jota kutsutaan aina kun komponentti päivittyy. Metodin suorittaminen jatkuu, jos Player on määritellyt parametriksi ”className : active”. Funktiossa tarkastetaan sitten onko sisältö määritelty näytettäväksi. Jos sisältöä ei ole määritelty näytettäväksi sään tai ajan mukaan tai se on poistettu käytöstä, käytetään ”nextItem”-metodia, jolla päästään suoraan seuraavaan sisältöön. Muussa tapauksessa kutsutaan ”nextItem”-metodia silloin, kun aikaa on kulunut määritellyn näytösajan verran. Videoissa näytösaika määräytyy videon pituuden mukaan ja nextItem kutsutaan kun video loppuu. Alla esimerkki “play”-metodista.

```
play(){
  const d = new Date();
  const hour = d.getHours();
  const minute = d.getMinutes();
  if (this.props.className === "active"){
    if (this.props.enabled === false){
      this.props.next();
    }
    else if (this.props.hourFrom > hour ||
(this.props.hourFrom === hour && this.props.minuteFrom > mi-
nute)){
```

```
      this.props.next();
    }
    else if (this.props.hourTo < hour || (this.props.hourTo
=== hour && this.props.minuteTo < minute)){
      this.props.next();
    }
    else if(this.props.currentWeather.temperature >
this.props.weatherMaxTemperature || this.props.cur-
rentWeather.temperature < this.props.weatherMinTemperature ){
      this.props.next();
    }
    else if( !(this.props.weatherSymbol.length === 0 ||
this.props.weatherSymbol.includes(this.props.cur-
rentWeather.symbol ))){
      this.props.next();
    }
    else if (this.props.type === "video"){
      const video = document.getElementById("video" +
this.props._id);
      video.play();
      video.onended = () => {
        console.log("video ended");
        this.props.next();
      };
    }
    else {
      clearTimeout(this.timeout);
      this.timeout = setTimeout(this.props.next,
this.props.duration*1000);
    }
  }
}
```

Itemin ”render”-metodissa määritellään sisällön tyypin mukaan mitä HTML-elementtiä käytetään.

```
render(){
  let item = null;
  if (this.props.type === "image"){
    item = <img src={this.props.src} style={this.imgStyle}/>;
  }
  else if (this.props.type === "html"){
    item = <iframe src={this.props.src} style={this.style}
scrolling="no" />;
  }
  if (this.props.type === "video"){
    item = (
      <video id={"video" +this.props._id} style={this.style}>
        <source src={this.props.src} type="video/mp4" />
```

```
        </video>
        );
    }
    return(
      <div className={this.props.className}
style={this.props.style}>
        {item}
      </div>
    );
}
```

7.11 Jatkokehitys

Sovelluksen jatkokehitysmahdollisuuksia on vielä paljon. Tällä hetkellä työ sopii hyvin esimerkiksi omalla Raspberry Pi:llä käytettäväksi kotiverkossa. Mikäli sovellusta haluaisi käyttää internetin välityksellä esimerkiksi joltain virtuaalipalvelimelta käsin, pitäisi projektiin lisätä vielä kirjautuminen, johon käyttäisin itselle jo tutuksi tullutta passport.js-kirjastoa. Sovellukselle tarvitsisi myös lisätä salattu yhteys käyttäen esimerkiksi Let's Encryptin tarjoamaa ilmaista SSL-sertifikaattia. Sovelluksesta puuttuu myös vielä monesta paikkaa virheiden hallinta ja testit.

Sisällöntoistossa sisältöjen väleille voisi vielä lisätä häivytys- tai siirtymisanimaatiot.

Settings-sivu on vielä tyhjä, mutta sen on tarkoitus toimia sivun teeman muokkaamiseen ja oman profiilin hallintana, kun kirjautuminen on lisätty.

Mahdollisuutena olisi myös lisätä enemmän tuettuja tiedostomuotoja kuvien, videoiden ja HTML-sivujen lisäksi. Myös jonkinlainen web-editori, jolla saisi esimerkiksi lisättyä tekstiä kuvan päälle voisi olla hyvä idea jatkokehitykselle.

Myös callback-funktiot voisi korvata JavaScriptin promiseilla, joilla koodista tulisi hieman luettavampaa, vaikka sovelluksessa ei montaa callback-funktiota olekaan.

Todennäköisesti jatkan sovelluksen kehittämistä vielä opinnäytetyön valmistumisen jälkeenkin, sillä projektista saa hyvää näyttöä osaamisesta CV:tä varten ja tämä projekti on ollut minulle hyvin mielenkiintoinen uusiin JavaScript-kirjastoihin ja työkaluihin tutustuessa.

8 YHTEENVETO

Tässä opinnäytetyössä kävin läpi alussa sisällönhallintajärjestelmiä yleisesti, jossa kerron missä ja miten sisällönhallintajärjestelmiä käytetään ja pohjustan mitä sisällönhallintajärjestelmällä tarkoitetaan.

Kerron kuinka toteutin oman sisällönhallintajärjestelmän infonäytöille, kuinka aloittaa projektin suunnittelu, kuinka valitsin haluamani työkalut ja tekniikat, mistä aloitin projektin ohjelmoinnin ja kuinka projektia voi vielä jatkaa.

Lopputuloksena projektista syntyi vielä hieman jatkokehitystä vaativa sisällönhallintajärjestelmä, mikäli sen haluaisi julkaista. Lopputulos toimii kuitenkin jo hyvin esimerkiksi minun omaan käyttötarkoituksiini sisäverkossa.

Halusin tutustua ja opetella opinnäytetyön ohella joihinkin minulle uusiin työkaluihin ja JavaScript-kirjastoihin joita käytetään nykyään paljon työelämässä. Päällimmäisenä olen erittäin tyytyväinen, että päädyin opettelemaan Reactin käyttöä, josta on varmasti paljon hyötyä tulevaisuudessa. React toi myös paljon uusia tapoja ajatella miten ohjelmoida. Opin web-ohjelmoinnista taas paljon uutta, joka oli opinnäytetyötä tehdessä minulle tarkoituskin.

Sain tehtyä sisällönhallintajärjestelmän pääasiassa ilman, että olisi tullut vastaan mitään suurempia ongelmia.

LÄHTEET

Babeljs.io (n.d.). Haettu 14.5.2017 osoitteesta
http://babeljs.io/

Expressjs.com (n.d.). Haettu 15.5.2017 osoitteesta
https://expressjs.com/

FirstView (n.d.). Haettu 15.4.2017 osoitteesta
http://www.firstview.fi/

Git-scm (n.d.). Haettu 2.5.2017 osoitteesta
https://git-scm.com/

Jake, S. 2013. Bootstrap: responsive web development. Sivu 1

Jimmy, S. 2013. Digital Signage: Software, Networks, Advertising, and Displays: A Primer for Understanding the Business, Sivu 1

Kayla, N. (n.d.). What is Git? Haettu 2.5.2017 osoitteesta
https://www.visualstudio.com/learn/what-is-git/

MongoDB (n.d.). What is MongoDb, Haettu 6.5.2017 osoitteesta
https://www.mongodb.com/what-is-mongodb

Npmjs (n.d.). What is npm? Haettu 1.5.2017 osoitteesta
https://docs.npmjs.com/getting-started/what-is-npm

Peter Bell. 2015. Introducing GitHub: A Non-Technical Guide. Sivu 1

Peter Bright. 2015. Microsoft's new Code editor is built on Google's Chromium. Haettu 22.5.2017 osoitteesta
https://arstechnica.com/information-technology/2015/04/microsofts-new-code-editor-is-built-on-googles-chromium/

React Native (n.d.). Haettu 5.5.2017 osoitteesta
https://facebook.github.io/react-native/

Survivejs (n.d.). React Introduction. Haettu 5.5.2017 osoitteesta
https://survivejs.com/react/introduction/

TrainingDotCom 2016. About Node.js, and why you should add Node.js to your skill set? Haettu 3.5.2017 osoitteesta
http://blog.training.com/2016/09/about-nodejs-and-why-you-should-add.html

W3Techs 2017. Usage of content management systems for websites. Haettu 10.5.2017 osoitteesta
https://w3techs.com/technologies/overview/content_management/all/

Webpack (n.d.). Haettu 30.4.2017 osoitteesta
https://webpack.github.io/
Wikipedia 2017. Content Management System. Haettu 10.5.2017 osoitteesta
https://en.wikipedia.org/wiki/Content_management_system

YR 2016. Information about the free weather data service. Haettu 29.4.2017 osoitteesta
http://om.yr.no/verdata/free-weather-data/