TAMPEREEN
AMMATTIKORKEAKOULU

JAVASCRIPT-POHJAISEN WEB-SOVELLUKSEN TOTEUTUS

Case: Sisällönjulkaisujärjestelmä

Tuomas Hakala

Opinnäytetyö
Toukokuu 2017
Tietotekniikan koulutusohjelma
Sulautetut järjestelmät ja elektroniikka /
Ohjelmistotekniikka

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietotekniikan koulutusohjelma
Sulautetut järjestelmät ja elektroniikka /
Ohjelmistotekniikka

TUOMAS HAKALA:
JavaScript-pohjaisen web-sovelluksen toteutus
Case: Sisällönjulkaisujärjestelmä

Opinnäytetyö 46 sivua, joista liitteitä 0 sivua
Toukokuu 2017

Tässä opinnäytetyössä tarkastellaan JavaScript-pohjaista sisällönhallintasovellusta, jonka palvelinpäässä käytetään Node.js-ympäristöä ja Express-sovelluskehystä, ja jonka käyttöliittymä toteutettiin React-kirjastolla. Sovelluskehityksessa käytettiin ECMAScript 2015-standardin mukaisia JavaScript-ominaisuuksia ja sen tyylitiedostot kehitettiin CSS-kielen Sass-laajennoksella. Lähdetiedostot käännettiin Babel-kääntäjällä ja Sass-parsijalla selainyhteensopivaan muotoon, jonka jälkeen ne paketoitiin Browserify-työkalulla.

Kehitetty sovellus näyttää käyttöliittymässään palvelinkoneella sijaitsevaa web-sisältöä, jonka muutokset päivitetään sovellukseen reaaliajassa. Web-sisältöä hallitaan muokkaamalla palvelinkoneella sijaitsevia sisältökansioita, ja sovellusta konfiguroidaan muokkaamalla sen konfiguraatiotiedostoja.

Opinnäytetyö käsittelee sovelluksen suunnittelu- ja kehitysprosessia, siinä hyödynnettyjä tekniikoita ja työkaluja, sekä lopullisen sovelluksen arkkitehtuuria, käyttöä, ja ylläpitoa. Lopuksi pohditaan mahdollisia jatkotoimenpiteitä, kuten jatkokehitystä ja sovelluksen tuotteistamista.

Asiasanat: web-sovelluskehitys, javascript, node, react

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
ICT Engineering
Embedded Systems and Electronics /
Software Engineering

TUOMAS HAKALA:
JavaScript-based Web Application development
Case: Content Delivery Application

Bachelor's thesis 46 pages, appendices 0 pages
May 2017

This thesis takes a look at a JavaScript-based content delivery application. Its back-end application runs in the Node.js environment and uses the Express framework, while its user interface was implemented using the React user interface library. ECMAScript 2015 compliant JavaScript features were used during the development, and the application's style sheets were written using the Sass syntax extension of the CSS-language. The source files were converted into a browser friendly format using the Babel-compiler and the Sass-parser. Afterwards they were bundled with the Browserify tool.

The resulting application's user interface displays web content, which is located on the server, and the application updates in real time upon content changes. The web content is managed by editing the content folders located on the server, and the application is configured by editing its configuration files.

The thesis discusses the application's planning and development process, and also describes the technologies and the tools that were used. The application's architecture, usage, and content management are also addressed. Finally, the text discusses possible further development and productization of the application.

Key words: web application development, javascript, node, react

SISÄLLYS

1 JOHDANTO 8
2 KÄYTETYT TYÖKALUT JA TEKNIIKAT 9
2.1 JavaScript 9
2.1.1 ECMA-262 9
2.1.2 JavaScript-moottorit ja -runtimet 9
2.2 Node.js 10
2.2.1 Yksisäikeisyys ja event loop 10
2.2.2 NPM 12
2.3 Käytetyt JavaScript-kirjastot ja -sovelluskehykset 12
2.3.1 React 12
2.3.2 Express 14
2.3.3 Muut käytetyt kirjastot 15
2.4 Markdown ja Marked-parsija 17
2.5 Kehitystyökalut 18
2.5.1 Babel 19
2.5.2 Browserify 20
2.5.3 Sass 21
2.5.4 Git 22
3 PROJEKTI 24
3.1 Toimeksianto ja tavoitteiden määrittely 24
3.1.1 Käyttöliittymätoiminnallisuuden määritykset 24
3.1.2 Ylläpitotoiminnallisuuden määritykset 25
3.2 Kehitysprosessi 25
3.3 Lopputulos 25
4 SOVELLUKSEN RAKENNE 27
4.1 Palvelinpään sovellusosa 27
4.1.1 HTTP-rajapinta 27
4.1.2 Sisällönhallinta-, Watcher- ja fileNameParser-moduulit 27
4.1.3 Loggers-moduuli 28
4.2 Selainpään sovellusosa 28
4.2.1 Käyttöliittymäkomponenttirakenne 29
4.2.2 Oleellisimmat käyttöliittymäkomponentit 30
4.2.3 Yleiset Sass-määritystiedostot 30
4.2.4 Yleiskäyttöiset JavaScript-funktiot 31
5 Sovelluksen käyttö 32
5.1 Käyttöönotto ja asennus 32

5.2 Käyttöliittymä 32
5.2.1 Banner ja navigaatiopalkki 33
5.2.2 Päänäkymä 34
5.3 Ylläpito 36
5.3.1 Sovelluksen konfigurointi 36
5.3.2 Web-sisältöjen hallinta 37
6 JATKOTOIMENPITEET 40
6.1 Jatkokehitys 40
6.2 Sovelluksen kaupallinen hyödyntäminen 40
6.2.1 Mehiläishoitajamalli 40
6.2.2 Freemium-malli 41
6.2.3 Mainosten näyttäminen käyttäjille 42
7 YHTEENVETO 43
LÄHTEET 45

LYHENTEET JA TERMIT

Tulkattava ohjelmointikieli	Englanniksi interpreted programming language. Ohjelmakäskyt käännetään ohjelman ajon aikana.
JavaScript runtime	Ajoympäristö, joka tarjoaa suoritettavalle JavaScript-ohjelmakoodille rajapinnan selaimen tarjoamiin toimintoihin.
Ohjelmointikielen kääntäjä	Englanniksi compiler. Kääntää ohjelmointikieliset lauseet tietokoneen ymmärtämiksi konekielisiksi käskyiksi.
Ohjelmointikielen tulkki	Englanniksi interpreter. Tulkitsee ohjelmointikielisiä lauseita käskyiksi kääntämättä niitä konekielisiksi.
API	Application programming interface, eli ohjelmointirajapinta.
Tiedostojärjestelmä	Englanniksi File system. Käyttöjärjestelmän tarjoama palvelu, jonka kautta ohjelmat pääsevät käsiksi laitteen kestomuistiin.
Non-blocking I/O	Syötteiden ja tulosteiden käsittelymalli, jossa niiden käsittely ei keskeytä ohjelmakoodin suorittamista.
Riippuvuus	Englanniksi dependency. Ohjelmakomponentti, jota sovellus tarvitsee toimiakseen.
DOM	Document object model. Selaimen paljastama abstraktio HTML-sisällön selaimen muistiin luomasta rakenteesta.
Middleware	WWW-palvelinohjelmistojen ketjutettavia ohjelmakomponentteja, joiden läpi palvelimelle tulevia pyyntöjä suodatetaan.
HTTP-metodi	HTTP-protokollalle määriteltyjä pyynnön aikeita kuvaavia verbejä, kuten get, post ja delete.
Villikorttioperaattori	Merkki, jota käytetään ohjelmallisessa vertailussa kuvaamaan mitä tahansa merkkiä tai merkkejä.
Regular Expression	Merkkijono, jota käytetään merkkijonovertailuissa ja -hauissa.

Wrapper	Ohjelmakomponentti, joka muokkaa, abstraktoi tai suojaa sisältämänsä ohjelmakomponentin käyttörajapintaa.
Polyfill-paketti	Ohjelmakomponentti, joka implementoi puuttuvan ominaisuuden selainympäristöön.
Repository	Ohjelmakoodipakettejen säilytyspaikka, jota hallitaan esimerkiksi versionhallintajärjestelmän kautta.
Monorepository	Monolithic Repository. Repository, joka sisältää useita toisiinsa liittyviä, mutta erillisiä ohjelmakoodipaketteja.
VCS	Version control system, eli versionhallintajärjestelmä on ohjelmatyökalu, jolla hallitaan sovelluskehitysprojektin eri versioita.
Snapshot	Versionhallintajärjestelmään tallennettava ote hallittavista tiedostoista.
Web-sisältö	Verkkosivuilla esitettyä sisältöä, kuten tekstiä, kuvia, tai videoita.
Sisällönjulkaisujärjestelmä	Englanniksi content delivery system. Sovellus, jonka kautta sen ylläpitäjä jakaa web-sisältöä sovelluksen käyttäjille.
Käyttäjänhallinta	Toiminnallisuus, joka mahdollistaa järjestelmän käyttäjien rekisteröimisen, tunnistautumisen, sekä muokkaamisen.
Sisällönhallintajärjestelmä	Englanniksi content management system. Tietojärjestelmä, johon tyypillisesti sisältyy web-sisällönjulkaisuominaisuuksien lisäksi esimerkiksi käyttäjänhallintaominaisuuksia.
SPA	Single page application, eli internet-selaimessa käytettävä sovellus, jonka käyttö edellyttää vain yhtä sivun latausta.
Access control	Suomeksi kulunvalvonta. Tietotekniikassa termillä viitataan käyttäjän oikeuteen käsitellä resursseja, kuten tiedostojärjestelmän tiedostoja.
REST	Tulee termistä representative state transfer. Palvelinpään rajapinnan arkkitehtuurityyppi.

1 JOHDANTO

Yritykset ja muut näkyvyyttä kaipaavat tahot ovat jo pitkään suosineet verkkosivuja tiedotus- ja julkaisuväylinään, koska ne ovat helposti asiakkaiden saavutettavissa ja lähes täysin alustariippumattomia.

Teknologisesta kehityksestä johtuva asiakaslaitteiden prosessointitehon nousu on sallinut selainympäristöjen kehittymisen täysin varustelluiksi sovellusalustoiksi, ja niille on alettu kehittää kokonaisia selainsovelluksia. Selainsovelluksista alettiin kehittää dynaamisia SPA (single page application) -sovelluksia, joiden käyttökokemus alkaa muistuttaa työpöytäsovellusta. (Brehn 2013.)

Selainsovellusten yleistyminen on tuonut lisää painoarvoa JavaScript-kielelle, joka taas on johtanut selaimessa käytettävien kirjastojen ja sovelluskehysten nousuun (Sotelo 2014). Node.js-pohjaisten palvelinsovellusten myötä selainpainotteiset SPA-arkkitehtuurit on saatu toimimaan tehokkaasti, ja kokonaisia web-sovelluksia on alettu kehittää täysin JavaScript-pohjaisena (Hernandez 2013).

Tässä opinnäytetyössä tarkastellaan JavaScript-pohjaisen sisällönhallintasovelluksen suunnittelu- ja kehitysprosessia, siinä käytettyjä moderneja selain- ja palvelintekniikoita, sekä sovelluskehitystä helpottavia työkaluja. Lisäksi työssä perehdytään valmistuneen sovelluksen arkkitehtuuriratkaisuihin, sekä sen käyttö- ja ylläpitotoiminnallisuuteen. Lopuksi käsitellään mahdollisia jatkotoimenpiteitä, kuten jatkokehitysmahdollisuuksia ja sovelluksen tuotteistamista.

2 KÄYTETYT TYÖKALUT JA TEKNIIKAT

Sovelluskehityksessä hyödynnettiin lukuisia web-sovellustekniikoita ja kehitystyökaluja. Tässä kappaleessa esitellään niistä oleellisimmat.

2.1 JavaScript

JavaScript on kevyt, tulkattava, oliosuuntautunut ohjelmointikieli, joka tunnetaan parhaiten verkkosivuille tarkoitettuna kielenä, mutta sitä käytetään selaimen lisäksi myös muissa ympäristöissä. Sen perussyntaksi muistuttaa tarkoituksellisesti Java- ja C++-ohjelmointikieliä, jotta sen opetteleminen vaatisi vähemmän uusien käsitteiden opettelua. (Mozilla Developer Network 2015.)

2.1.1 ECMA-262

ECMA-262 on Ecma International -standardointiorganisaation ylläpitämä standardi, jonka tavoitteena on määrittää yhteneväinen muoto JavaScript-kielelle. Tätä määritelmää kutsutaan ECMAScriptiksi, ja sen tavoitteena on taata ohjelmakoodin yhtenäinen toiminta eri ympäristöissä.

ECMAScript-määritelmän ensimmäinen versio julkaistiin vuonna 1997, jonka jälkeen kieleen on lisätty standardin kautta lukuisia parannuksia. Nykyaikaiset selaimet tukevat jo hyvin ECMAScript 5 -muotoista JavaScriptiä, joten se on yleisesti käytössä selainohjelmoinnissa. Myöhemmin julkaistut ECMAScript 2015 ja ECMAScript 2016 toivat kieleen paljon uusia ominaisuuksia, kuten luokat, nuolifunktiot ja eksponenttioperaattorin, ja niitä hyödynnetään ohjelmistokehityksessä kääntämällä kirjoitettu ohjelmakoodi ECMAScript 5 -muotoon vanhempia selaimia varten.

2.1.2 JavaScript-moottorit ja -runtimet

JavaScript-ohjelmakoodin suorittaminen vaatii kaksi komponenttia: Javascript-moottorin ja -suoritusympäristön eli runtimen.

JavaScript-moottorin tehtävä on tulkita ohjelmakoodia ja kääntää siitä suoritettavia käskyjä. Esimerkiksi WebKit-selainmoottorin käyttämä JavaScriptCore-moottori tekee tämän parsimalla ohjelmakoodista tavukoodia, jonka se tulkitsee käskyiksi (WebKit 2014). Google Chrome -selaimen käyttämä V8-moottori taas kääntää ohjelmakoodin konekieliseksi ajon aikana käännösnopeutta painottavalla JIT (just-in-time) -kääntäjällä (Laurens 2015). Molemmista moottoreista löytyy myös käännöstulosta tehokkaammin optimoivat JIT-kääntäjät, jotka kääntävät uudelleen ohjelmanosat, joita suoritetaan usein.

JavaScript runtimen tehtävä on tarjota suoritettavalle ohjelmakoodille tarpeellisia rajapintoja (Mather 2014). Tällainen on esimerkiksi selainikkunaa edustava Window-olio, jonka kautta ohjelma pääsee käsiksi esimerkiksi DOM- ja Console-rajapintaan.

JavaScript-moottorin ja -suoritusympäristön käsitteissä esiintyy päällekkäisyyttä, ja niiden rajat ovat eri lähteiden välillä häilyvät. Nämä komponentit kuuluvat usein samaan pakettiin, kuten esimerkiksi V8-moottorin tilanteessa.

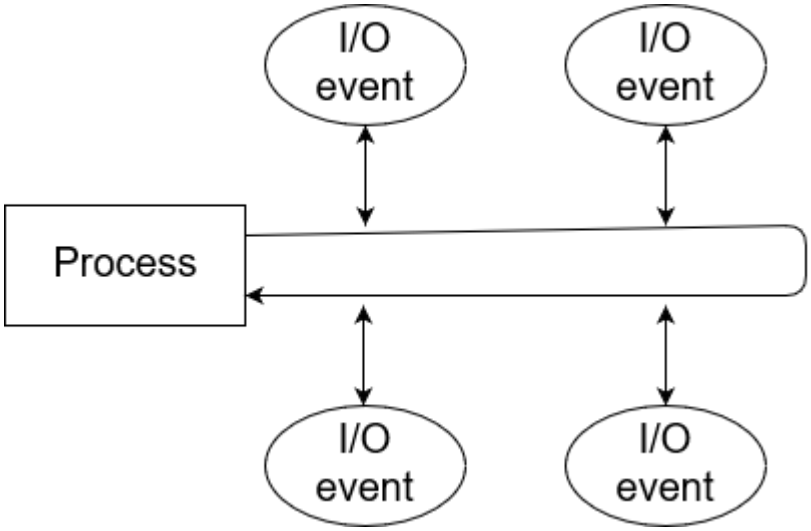
2.2 Node.js

Node.js on avoimen lähdekoodin JavaScript runtime, joka on rakennettu V8-moottorin päälle. Sen tarkoitus on mahdollistaa JavaScript-ohjelmointi muilla alustoilla kuin verkkoselaimissa, ja se tarjoaa lukuisia ohjelmointirajapintoja, kuten esimerkiksi HTTP-API WWW-palvelinohjelmistojen vaatimuksiin, File System-API tiedostojärjestelmän käsittelyyn ja Console-API debuggaustarkoitukseen. Node.js käyttää event-pohjaista non-blocking I/O -mallia, joka tekee siitä kevyen ja tehokkaan (Node.js Foundation 2017).

2.2.1 Yksisäikeisyys ja event loop

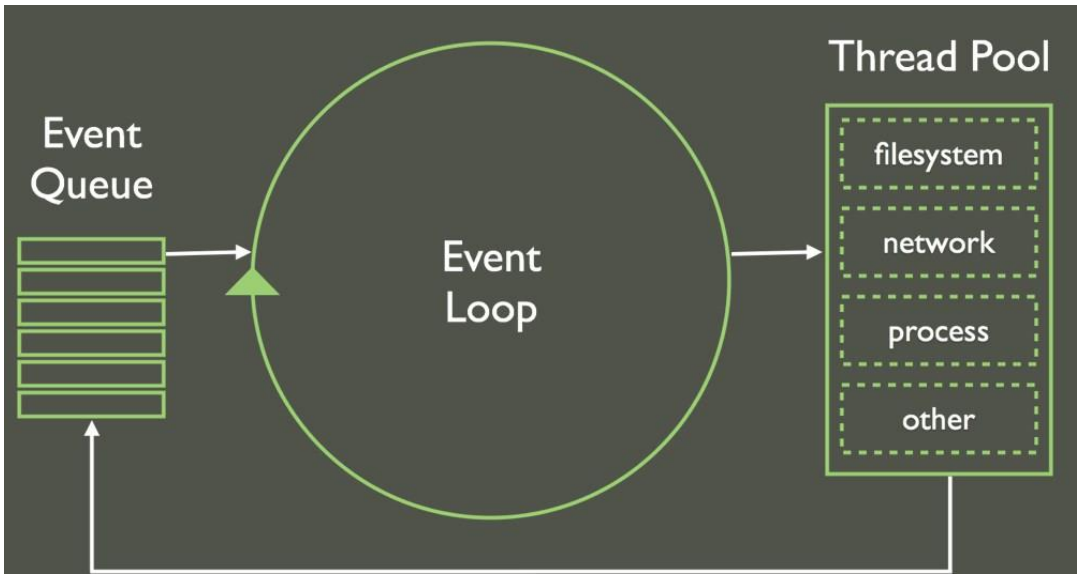
Toisin kuin useimmat WWW-palvelinohjelmat, kuten Apache HTTP Server, Node.js ei luo uutta prosessia tai säiettä jokaiselle saapuvalle kutsulle, vaan kutsu laukaisee eventin, jonka ympäristö asettaa jonoon odottamaan palvelemista. Tämä mahdollistaa Node.js:n yksisäikeisen toiminnan (kuvio 1), jolloin keskusmuistista ei tarvitse allokoida muistia

lukuisille prosesseille. Yksisäikeinen toiminta suoriutuu kutsuista tehokkaasti, kun suurin osa kutsun käsittelyyn kuluvasta ajasta johtuu esimerkiksi tietokantakutsun käsittelemisen odottamisesta. Jos taas kutsun käsitteleminen vaatii palvelinohjelmalta monimutkaista prosessointia, kuten esimerkiksi Fourier-muunnosta, suoriutuu monisäikeinen prosessi tehtävästä tehokkaammin. (Takada 2011.)



KUVIO 1. Node.js:n yksisäikeinen non-blocking I/O –malli

Jonoon asetettuja eventtejä käsittelee runtimessa toimiva event loop (kuvio 2). Event loop on entiteetti, joka käsittelee ja prosessoi ulkoisia eventtejä ja muuntaa niitä callback-funktiokutsuiksi (Takada 2011). Ohjelmakoodin tallennettua pyynnölle luodun callback-kutsun kontrolli palaa runtimelle. Kun tarvittava toimenpide on tehty tai data on haettu, alkaa callback-funktion suoritus.



KUVIO 2. Node.js-ympäristössä toimiva event loop (Meyerson 2015)

2.2.2 NPM

Node.js:n paketinhallintajärjestelmä, NPM (Node Package Manager), on maailman suurin avoimen lähdekoodin kirjastojen ekosysteemi maailmassa (Node.js Foundation 2017). Se tarjoaa mahdollisuuden tallentaa ja julkaista JavaScript-kirjastoja ja -ohjelmakomponentteja ilmaiseksi. Pakettejen ei tarvitse olla Node.js-ympäristön käyttöön tarkoitettuja, vaan NPM on käytettävissä myös selainpään komponenttejen hallintaan. Kaikki sovelluksen riippuvuudet voivat löytyä samasta paketinhallinnasta, mikä helpottaa projektin käyttöönottoa uusissa työympäristöissä.

NPM-työkalua hallitaan tyypillisesti komentoriviltä, ja se tallentaa konfiguraationsa selkokielisenä package.json-nimiseen tiedostoon. Tiedostoon tallennetaan esimerkiksi sovelluksen kuvaus ja sen riippuvuuksien tiedot.

2.3 Käytetyt JavaScript-kirjastot ja -sovelluskehykset

Kehitetty sovellus hyödyntää useita ohjelmakomponentteja, kuten kirjastoja ja sovelluskehyksiä, jotka tuotiin projektiin NPM-paketinhallinnan kautta. Komponentit ratkaisevat projektin suunnittelussa ja sovelluskehityksessä vastaan tulleita ongelmia, kuten DOM-rakenteen tehokas manipulaatio, selainyhteensopivuuden varmistaminen, sekä tapahtumien lokitus lokitiedostoihin. Tässä kappaleessa esitellään projektissa käytetyt sovelluskomponentit.

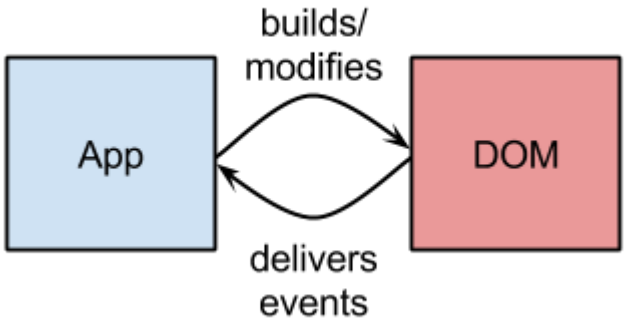
2.3.1 React

React on Facebookin ja Instagramin kehittämä käyttöliittymän näkymäkerrosta käsittelevä komponenttipohjainen käyttöliittymäkirjasto, jota suositaan erityisesti pienen kokonsa ja tehokkaan suorituskykynsä ansiosta.

Reactilla rakennetaan kapseloituja käyttöliittymäkomponentteja, jotka pitävät huolta omasta tilastaan (state) ja laativat sen perusteella monimutkaisia käyttöliittymiä

(Facebook 2017). React-komponenttejen DOM-rakenne määritetään kehitysvaiheessa JavaScript-koodiin upotettuna XML-tyylisellä syntaksilla, joka tunnetaan nimellä JSX. JSX-sapluunoiden kautta data on helppo päivittää komponentin rakenteeseen. Selainympäristö ei kuitenkaan osaa tulkita JSX-sapaluunoita, vaan ne pitää kääntää JavaScript-muotoon erillisellä kääntäjällä ennenkuin lähdekoodi toimitetaan selaimeen. JSX-sapluunat käännetään JavaScriptiksi tyypillisesti Babel-kääntäjällä.

React tunnetaan suorituskykyisestä DOM-manipulaatiostaan, joka on sen tarjoaman virtuaalisen DOM-kerroksen ansiota. Normaalisti JavaScript-sovellus muokkaa DOM-rakennetta aina, kun käyttöliittymässä täytyy näkyä muutoksia (kuvio 3). Tämä on aikavievää ja hankalaa, koska suurissa selainsovelluksissa puumainen DOM-rakenne on valtavan monimutkainen.

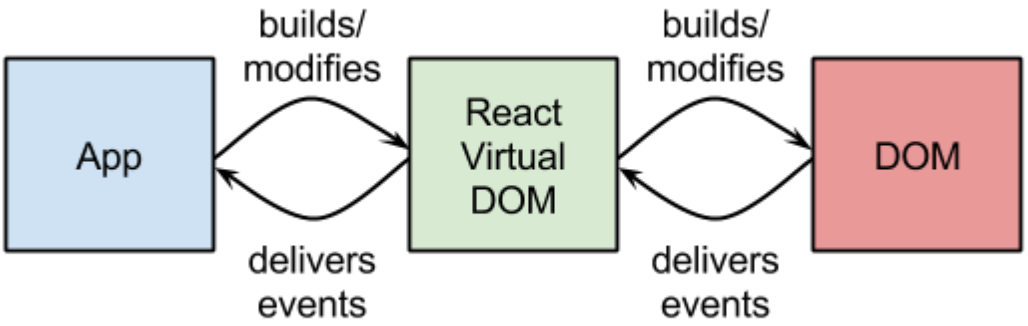


KUVIO 3. Tavallisen JavaScript-sovelluksen ja DOM-rakenteen interaktio (Haberman 2014)

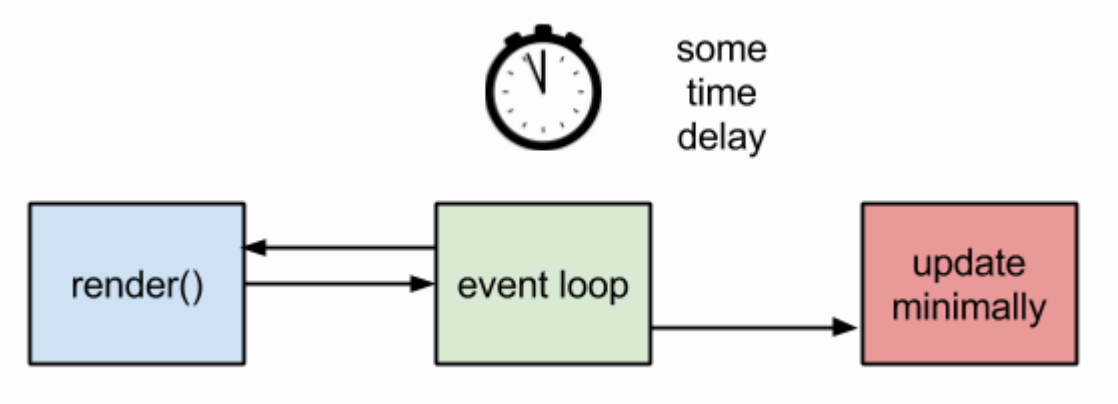
Reactin virtuaalinen DOM on HTML DOM:n abstraktio. Se on kevyt ja irrallaan selainkohtaisen implementaation yksityiskohdista. Koska DOM on jo itsessään abstraktio, Reactin virtuaalinen DOM on itse asiassa abstraktion abstraktio. (Krajka 2015.)

Reactin virtuaalinen DOM toimii välikerroksena sovelluksen ja DOM-mallin välissä (kuvio 4). Käyttöliittymäkomponentin tilan muuttuessa React päivittää sen perusteella kootun DOM-elementin virtuaaliseen DOM-malliinsa. Näitä muutoksia kerätään virtuaaliseen DOM-malliin, kunnes Reactin event loop -kierros on tullut päätökseensä (kuvio 5). Tällöin React vertaa virtuaalisen DOM-mallin ja HTML DOM-mallin eroja ja

päivittää DOM-malliin ainoastaan ne kohdat, joissa on tapahtunut muutosta. Tämä tekee Reactin DOM-manipulaatiosta suorituskykyistä.



KUVIO 4. Reactin virtuaalinen DOM-kerros sovelluksen ja DOM-mallin välissä (Haberman 2014)

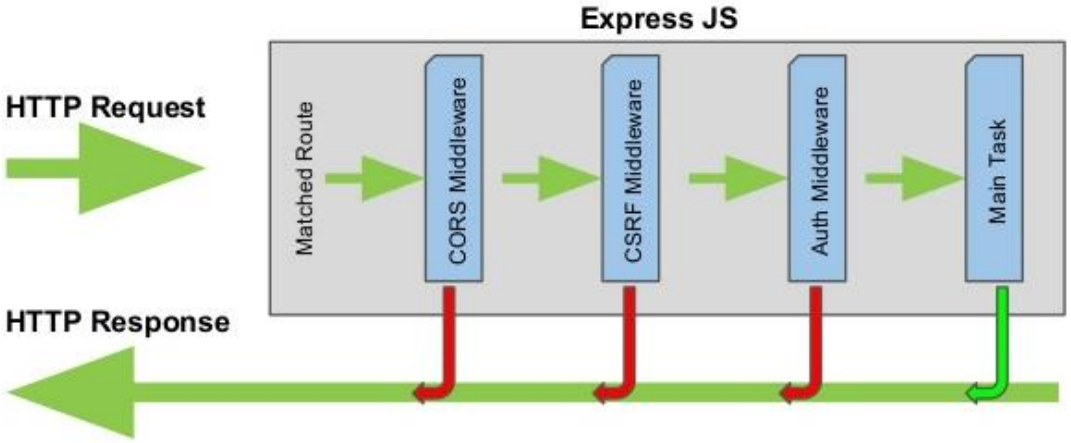


KUVIO 5. React kerää muutokset event loopin aikana ja päivittää mahdollisimman vähän DOM-mallia (Haberman 2014)

2.3.2 Express

Express on Node.js-ympäristössä käytettävä minimalistinen web-sovelluskehys, joka tarjoaa kattavasti ominaisuuksia web-sovellusten kehittämiseen. Express abstraktoi Node.js-ympäristön HTTP-API:n tarjoamia ominaisuuksia ja tarjoaa kehitysprosessia helpottavaa lisätoiminnallisuutta, kuten reititystä ja middleware-funktioiden määrittelyä. (Node.js Foundation 2016.)

Expressin reititys helpottaa palvelimelle saapuvien pyyntöjen käsittelyä jakamalla käsittelijafunktion pienempiin osiin haetun polun ja HTTP-metodin avulla. Polkujen sovitus tukee myös villikorttioperaattoria ja Regular Expression -merkkijonoja. Middleware-funktiot muodostavat putkimaisen sarjan, jonka läpi palvelimelle saapuvat pyynnöt kulkevat (kuvio 6). Niiden jaottelu pieniin kokonaisuuksiin selkeyttää palvelimen rajapinnan hahmottamista. Middleware-funktiot voivat esimerkiksi jaotella pyyntöjä polkuihin pyynnön parametrejen perusteella, muokata pyynnön sisältöä, tai hoitaa virheenkäsittelyä. (Hahn 2014.)



KUVIO 6. Pyynnön suodatus middleware-funktiosarjan läpi (Polychronakis 2014)

Lisäksi Express tarjoaa toiminnallisuuden renderöidä HTML-näkymiä sapluunatiedostoista. Tämä mahdollistaa sapluunoihin upotettujen muuttujien arvojen ratkaisemisen sekä uudelleenkäytettävien näkymäosioiden nopean kehittämisen. Sapluunoita HTML-muotoon konvertoivan moottorin voi valita Expressin tukemista moottoreista, joista suosittuja ovat esimerkiksi Pug ja Mustache, tai sen voi kehittää kokonaan itse. (Node.js Foundation 2016.)

2.3.3 Muut käytetyt kirjastot

Chokidar on Node.js-ympäristön File System-API:n ympärille rakennettu wrapper-olio, joka korjaa sen tiedostomuutoksia käsittelevien watch-, fileWatch- ja fsevents-rajapintojen virheitä ja yhtenäistää niiden toimintaa eri alustoilla. Watch- ja fileWatch-rajapintojen tiedostomuutoksista raportoivat eventit antavat usein väärää tietoa muutoksesta ja saattavat laukaista monta eventtiä yhtä muutosta kohden. Chokidar korjaa

raportit todenmukaisiksi tarkastamalla eventtejen yhteydessä kansioiden sisältöjä ja tiedostojen metadataa. (Miller 2016.)

Directory-tree on olio, joka lukee tiedostojärjestelmästä halutun kansion sisällön ja palauttaa sen rakennetta kuvaavan JavaScript-olion (kuva 1). Olio kertoo kansiosisällöstä tietoja, kuten tiedostonimen, -muodon ja -koon. Palautettua kansiosisältöä voi suodattaa tiedostomuodon perusteella ja tuloksia voi käsitellä itsemääritetyllä callback-funktiolla. (Dobrescu-Balaur 2017.)


```
{
  "path": "photos",
  "name": "photos",
  "size": 600,
  "children": [
    {
      "path": "photos/summer",
      "name": "summer",
      "size": 400,
      "children": [
        {
          "path": "photos/summer/june",
          "name": "june",
          "size": 400,
          "children": [
            {
              "path": "photos/summer/june/windsurf.jpg",
              "name": "windsurf.jpg",
              "size": 400,
              "extension": ".jpg"
            }
          ]
        }
      ]
    },
```


KUVA 1. Directory-tree-olion palauttama kansiosisältö JSON-muodossa (Dobrescu-Balaur 2017)

Winston on monipuolinen Node.js-ympäristöön tarkoitettu lokituskirjasto, jonka kattaa kaikki tavallisen Node.js-sovelluksen lokitustarpeet. Kirjasto mahdollistaa omien lokituksesta vastaavien logger-olioiden luomisen, ja niille on mahdollista lisätä useita

transport-olioita, jotka määrittävät mihin kohteeseen lokitus tallennetaan. Transport-oliot voidaan konfiguroida lokittaamaan esimerkiksi konsoliin, tiedostoon, tietokantaan tai etäkohteeseen HTTP-yhteyden avulla. Lokimerkinnöille on mahdollista määrittää tallennettavaa metadataa tai jopa räätälöityjä lokitusformaatteja. Niille määritetään lokitustaso, joka kuvaa viestin tärkeyttä, ja ne voivat ohjautua eri transport-olioihin lokitustasonsa perusteella. Winston hoitaa myös lokitiedostojen dynaamisen hallinnan, joten sovelluskehittäjän ei tarvitse huolehtia lokikansioiden paisumisesta. (Robbins 2017.)

Sovelluksen selainpäässä hyödynnetään Window-olion sisältämiä Fetch- ja Promise-rajapintoja, joille ei löydy tukea Internet Explorer -selaimista. Fetch-APIa käytetään resurssejen hakemiseen, ja Promise-APIn avulla käsitellään asynkronisesti dataa, jonka saapuminen vie aikaa tai on epävarmaa. Näiden rajapintojen selainyhteensopivuusongelmat ratkaistaan polyfill-paketeilla. (Mozilla Developer Network 2015.)

2.4 Markdown ja Marked-parsija

Markdown on web-sisältöjen kehittämiseen tehty sapluunakieli, jonka tavoitteena olla mahdollisimman selkeästi luettavaa (kuva 2). Sen alkuperäinen tavoite oli olla niin siistin näköistä, että sapluunan voisi julkaista sellaisenaan ilman, että se näyttäisi sisältävän tageja tai muotoilukomentoja. Sen suurin inspiraatio olivat sähköpostiviestit, joita ei oltu muotoiltu HTML-tageilla. Markdown-kielinen sapluuna muunnetaan HTML-muotoon Markdown-parsijaksi kutsutulla ohjelmakomponentilla. (Gruber 2004.)

```
A First Level Header
====================

A Second Level Header
---------------------

Now is the time for all good men to come to
the aid of their country. This is just a
regular paragraph.

The quick brown fox jumped over the lazy
dog's back.

### Header 3

> This is a blockquote.
>
> This is the second paragraph in the blockquote.
>
> ## This is an H2 in a blockquote
```

KUVA 2. Markdown-syntaksi on selkeästi luettavaa sellaisenaan (Gruber 2004)

Github-palvelu on kehittänyt oman laajennoksensa Markdown-syntaksiin. Laajennoksen nimi on Github Flavored Markdown, ja sen käyttö on yleistynyt myös Githubin ulkopuolella. Laajennos lisää syntaksiin monia yleisesti käytännöllisiä ominaisuuksia, kuten ohjelmointikielten syntaksikorostuksia, syntaksi taulukoiden luomiseen ja URL-osoitteiden automaattinen linkkaaminen. Lisäksi se lisää kieleen Githubissa hyödynnettyjä ominaisuuksia, kuten käyttäjäprofiililinkkien ja commit-viittausten upottamisen tekstiin. (Github 2014.)

Marked on JavaScript-pohjainen Markdown-parsija, jonka kehittäjien tavoitteena oli kehittää kevyt ja suorituskykyinen Markdown-parsija, joka ei varastoi käännöstuloksia välimuistiin. Marked-parsijaa voi käyttää sekä Node.js- että selainympäristössä, ja sen voi konfiguroida tukemaan Github Flavored Markdown -syntaksia ja suodattamaan pois sapluunatiedostossa olevat HTML-tagit. Markdown-sapluunoista HTML-syntaksia kääntävistä renderer-olioista voi myös luoda räätälöityjä versioita, mikä mahdollistaa oman Markdown-syntaksilaajennoksien hyödyntämisen. (Jeffrey 2014.)

2.5 Kehitystyökalut

Sovelluksen kehityksessä hyödynnettiin kehitysprosessia helpottavia työkaluja, joiden tehtävä oli kääntää ohjelmakoodia selainympäristöön sopivaan muotoon, jäsentää

kehitettävää ohjelmakoodia, ja paketoida kehitetyt moduulit yhdeksi kokonaisuudeksi. Tässä kappaleessa esitellään oleellisimmat projektissa hyödynnetyt kehitystyökalut.

2.5.1 Babel

Babel on avoimen lähdekoodin JavaScript-kääntäjä, jota käytetään tyypillisesti kääntämään uusien ECMAScript-versioiden mukaista ohjelmakoodia (kuva 3) vanhemman standardin muotoon (kuva 4). Babelin Github-repository on monorepository, joka sisältää kaikki sen tyypillisimpiin käyttötarkoituksiin vaadittavat paketit, kuten kääntäjäytimen, babel-preset-env -paketin, ja plugineja. (Babel 2017.)

```
1  const myConst = 'my string';
2
3  let logString = myString =>
4    console.log(`Value of the string is '${myString}'.`);
5
6  logString(myConst);
```

KUVA 3. ECMAScript 2015 -standardin mukaista JavaScript-koodia

```
1  'use strict';
2
3  var myConst = 'my string';
4
5  var logString = function logString(myString) {
6    console.log('Value of the string is \'' + myString + '\'.');
7  };
8
9  logString(myConst);
```

```
"Value of the string is \'my string\'."
```

KUVA 4. Babelilla ECMAScript 5 -muotoon käännetty JavaScript-koodi

Babel-core -paketti sisältää itse Babel-kääntäjän, joka koostuu Babylon-parsijasta, Babel-traverse -komponentista, ja Babel-generaattorista. Babylon-parsija generoi alkuperäisestä lähdekoodista AST (abstract syntax tree) node -tyyppisiä olioita, joita käytössä olevat plugin-paketit käsittelevät hyödyntäen Babel-traverse -komponenttia. Babel-generaattori muuntaa tuloksena saadut AST nodet takaisin ohjelmakoodimuotoon. (Babel 2017.)

Babel-pluginit ovat liitännäispaketteja, joilla määritetään muunnokset, joita Babel-kääntäjä suorittaa käännettävälle lähdekoodille. Babel-core ei sinänsä sisällä plugineja, vaan liittämällä plugin-paketteja Babel-kääntäjään käyttäjä määrittää minkälaista lähdekoodia kääntäjä tulkitsee ja tuottaa. Babel-presetit ovat plugin-pakettikokoelmia, jotka mahdollistavat suurempien syntaksikokonaisuuksien, kuten esimerkiksi ECMAScript 2015 -syntaksin kääntämisen ilman, että käyttäjän tarvitsee sisällyttää plugin-paketteja jokaista syntaksin ominaisuutta varten. (Babel 2017.)

Babel-kääntäjää on mahdollista konfiguroida komentorivityökalulla tai antamalla konfiguraatiot JSON-muotoisena erillisessä .babelrc-tiedostossa tai Node.js-ympäristön package.json-tiedostossa. Konfiguroitavilla asetuksilla voidaan esimerkiksi määrittää käytettävät plugin- ja preset-paketit, sekä karsia rivinvaihdot ja whitespace-merkit. Asetuksia voidaan myös lohkottaa eri ympäristöjen perusteella. Näin voidaan säätää eri asetukset käytettäviksi esimerkiksi kehitys- ja tuotantoympäristöissä. (Babel 2017.)

2.5.2 Browserify

Browserify on työkalu, joka mahdollistaa Node.js-tyylisen moduulinhallinnan selainpään JavaScript-sovelluksissa. Moduulit määritellään joko Node.js-tyylisillä require- ja exports-avainsanoilla, tai ECMAScript 2015 -syntaksin mukaisilla import- ja export-avainsanoilla. Browserifyta on mahdollista käyttää komentoriviltä tai ohjelmallisesti JavaScript-API:n kautta. (Browserify 2017.)

Browserify paketoi sovelluksen käyttämät moduulit yksittäiseen bundle.js-tiedostoon, jolloin selainpään lähdekooditiedostojen hallinta helpottuu. Käyttäjän on mahdollista paketoida mukaan itse kirjoittamaansa lähdekoodia, sekä selainympäristössä toimivia NPM-paketteja. Browserify tukee myös kolmannen osapuolen liitännäisiä, kuten esimerkiksi SCSS-tiedostojen moduulinhallintaa hoitavaa Bundlify-scss -pluginia. (Browserify 2017.)

2.5.3 Sass

Sass (Syntactically awesome style sheets) on CSS-laajennos, joka tuo kieleen ilmaisukykyä ja selkeyttä. Sen tiedostot parsitaan selaimen ymmärtämään CSS-muotoon Sass-parsijalla. Sassin uuden SCSS (Sassy CSS) -syntaksin (kuva 5) mukaiset tiedostot tunnistaa .scss-päätteestä. (Hampton, C., Weizenbaum, N., Eppstein, C. 2016.)

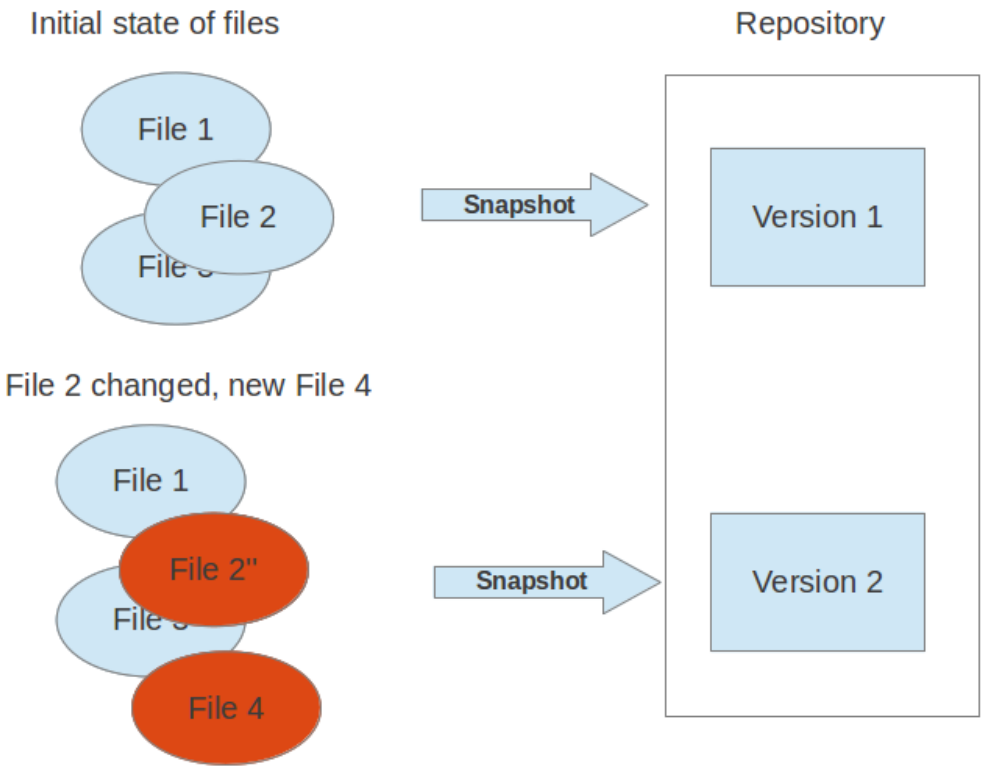
```scss
.blog{
  margin: $h1-margin-top-and-bottom 0;

  .blog-article{
    position: relative;
    background-color: $blog-bg-color;
    padding: 20px 0 10px;
    border-bottom: 1px solid $blog-border-color;

    &:first-child{
      padding-top: 0;
    }

    .blog-article-link{
      position: absolute;
      top: 0;
      right: 0;
      padding: 10px 0 0;
    }
  }
}

```

KUVA 5. SCSS-syntaksin mukaisen tyylitiedoston sisältöä

Sass lisää CSS-syntaksiin lukemista selkeyttäviä ominaisuuksia, kuten tyylimääritysten sisäkkäistämisen (nesting). Tämän lisäksi se tarjoaa tyylitiedostoissa käytettäväksi pienen joukon laajennoksia, jotka tunnetaan SassScriptinä. SassScript mahdollistaa esimerkiksi yksinkertaisten laskutoimitusten suorittamisen, muuttujien määrittämisen, sekä argumentteja vastaanottavien function-direktiivejen hyödyntämisen. SassScript-operaatiot ratkotaan tiedostojen parsimisvaiheessa, joten selaimeen toimitetaan ainoastaan puhtaan CSS-syntaksin mukaisia tyylitiedostoja. (Hampton, C., Weizenbaum, N., Eppstein, C. 2016.)

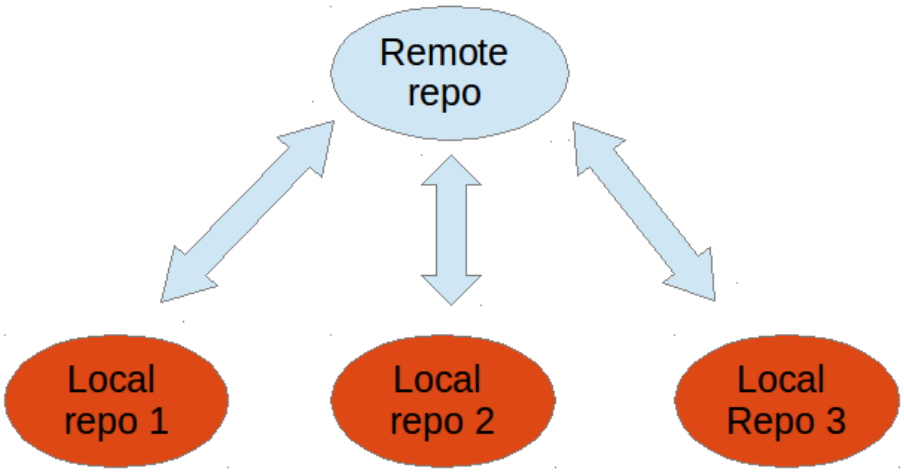
2.5.4 Git

Git on maailman suosituin sovelluskehitykseen tarkoitettu jaoteltu versionhallintajärjestelmä (distributed VCS). Sitä käytetään tyypillisesti tekstisisällön, kuten esimerkiksi lähdekoodi-, HTML-, tai konfiguraatiotiedostojen versionhallintaan. Gitin avulla käyttäjän tiedostoista luodaan snapshot, joka tallennetaan versioiden säilytyspaikkaan eli Git-repositoryyn (kuvio 7). (Vogel 2016.)



KUVIO 7. Snapshottejen tallentaminen Git-repositoryyn (Vogel 2016)

Toisin kuin esimerkiksi Subversion-versionhallintajärjestelmässä, Gitin jaotellun versionhallinta-arkkitehtuurin ansiosta sen versiohistoria ei sijaitse ainoastaan keskitetyssä repositoryssa. Jokaisella projektin sovelluskehittäjällä on paikallinen työkopio sovelluksen repositorysta (kuvio 8), joka päivitetään ajan tasalle viimeistään kun snapshot-versio tallennetaan keskitettyyn repositoryyn. Tällöin keskitetyn repositoryn tuhoutuessa sovelluksen versiohistoria on vielä tallessa sovelluskehittäjien paikallisissa repositoryissa. (Atlassian 2017.)



KUVIO 8. Sovelluskehittäjät luovat paikallisia työkopioita alkuperäisestä Git-repositorysta (Vogel 2016)

Keskitetty Git-repository voi sijaita kehittäjän koneella paikallisesti, tai ulkoisella palvelimella. Avoimen lähdekoodin sovelluskehitysprojekteissa on yleistä käyttää Git repository hosting -palveluita, kuten Githubia tai Atlassianin BitBuckettia. Nämä avoimet palvelut tarjoavat ilmaisen Git-repositoryn ylläpitämisen ja kätevän käyttöliittymän lisäksi muita yhteistyötä ja verkostoitumista helpottavia ominaisuuksia, kuten avointen repositoryjen kloonamista omalle käyttäjälle (forking), sekä muutosehdotusten, eli ”pull request” -ilmoitusten helppoa esittämistä muille käyttäjille. (Finley 2012.)

3 PROJEKTI

Projektissa kehitettiin JavaScript-pohjainen sisällönjulkaisusovellus yhden sovelluskehittäjän voimin. Sovelluskehitysprosessi sijoittui vuoden 2016 syksyn ja vuoden 2017 kevään väliselle ajalle.

3.1 Toimeksianto ja tavoitteiden määrittely

Projektin tavoitteena oli kehittää käyttövalmis web-sovellus, joka toimii yksisuuntaisena web-sisällönjakokanavana toimeksiantajalta sovelluksen käyttäjille. Sovelluksella tulee olla verkkoselaimella käytettävä käyttöliittymä, jossa on mahdollista jakaa kuvia, artikkeleita ja alasivuja. Sisällöntuottamisen ja muun ylläpidon tulee olla nopeaa ja vaivatonta.

Toimeksiantaja ei kuitenkaan halunnut ottaa käyttöön valmista sisällönhallintajärjestelmää tai sisällönjulkaisujärjestelmää, vaan sovelluksesta haluttiin toimeksiantajan tarpeisiin räätälöity kokonaisuus, joka ei sisällä ylimääräisiä ominaisuuksia, kuten käyttäjänhallintaa. Sovelluksen palvelinpää päätettiin toteuttaa Node.js-ympäristöön ja sen käyttöliittymän haluttiin hyödyntävän React-käyttöliittymäkirjastoa.

Sovelluksen määrittelyssä painotettiin yksinkertaista toteutusta, jotta sovellus saadaan käyttöön nopeasti. Määrityksissä ei menty yksityiskohtiin, vaan ne saivat tarkentua kehitysprosessin aikana.

3.1.1 Käyttöliittymätoiminnallisuuden määritykset

Käyttöliittymästä tehdään SPA-tyyppinen verkkosivu, jolla on mahdollista selata kaikkea sovelluksen web-sisältöä. Käyttöliittymän tulee tukea uusia versioita Mozilla Firefox-, Google Chrome-, Opera-, Windows Internet Explorer-, sekä Microsoft Edge-selaimista, koska sovelluksen käyttäjien uskotaan käyttävän päivitettyjä versioita selaimista.

Sovellukselle ei toteuteta käyttäjänhallintaa, joten kaikki web-sisältö näkyy tunnistautumattomalle käyttäjälle. Sisällönmuokkausta ei käyttöliittymän kautta ole mahdollista tehdä.

3.1.2 Ylläpitotoiminnallisuuden määritykset

Sisällöntuottaminen ja muu ylläpitotoiminta ei edellytetä syvää web-tekniikoiden tuntemista. Käyttöliittymässä näytettävää web-sisältöä hallitaan muokkaamalla erillisten sivu-, artikkeli-, ja kuvakansioiden sisältöä. Artikkeleiden ja alasivujen sisältö luodaan sapluunatiedostoista, jotka on kirjoitettu Markdown-merkintäkielellä. Kuvasisältö tulee erillisistä kuvatiedostoista, jotka jaotellaan käyttöliittymässä kuvakokoelmiin kuvakansion sisäisillä alikansioilla. Sapluuna- ja kuvatiedostojen tiedostonimillä määritetään sisällön järjestys selainpäässä.

3.2 Kehitysprosessi

Sovelluksen lähdekoodia hallittiin Git-versionhallintajärjestelmällä ja sitä säilytettiin avoimesti sovelluskehittäjän Github-repositoryssa (https://github.com/tuomasHa/file-read-app). Toimeksiantaja pääsi tarkkailemaan sovelluksen kehittymistä Github-repositoryssa näkyvän lähdekoodin ja commit-päivitysten kautta. Kehitysprosessin aikana toimeksiantajan edustajan kanssa järjestettiin myös katselmointitilanteita, joissa käytiin läpi sovelluksen ominaisuuksia sekä hahmoteltiin jatkossa kehitettävää toiminnallisuutta.

Sovelluskehittäjä testasi sovelluksen muutoksia Windows-alustalla omalla kehityskoneellaan sekä Linux-ympäristössä Raspberry Pi -palvelimella. Toimeksiantaja ei suorittanut testausta sovelluskehityksen aikana.

3.3 Lopputulos

Projektin lopputuloksena saatiin luotettavasti toimiva sisällönjulkaisusovellus, joka täytti toimeksiantajan asettamat määritteet. Toimeksiantaja hyväksyi sovelluksen ja antoi siitä

positiivista palautetta, eli projektissa onnistuttiin hyvin. Toimeksiantaja sai sovelluksen toimimaan ja otti sen testikäyttöön.

Sovelluskehitysprosessin aikana kehittäjälle kehittyi syvempi käsitys React-kirjaston käytöstä, modernista selainohjelmoinnista, sekä Node.js-pohjaisten web-sovellusten suunnittelusta ja kehittämisestä.

4 SOVELLUKSEN RAKENNE

Kehitetty CDA toimii yksisuuntaisena web-sisällönjakokanavana sen ylläpitäjältä sovelluksen käyttäjille. Toisin kuin kokonaisessa sisällönhallintajärjestelmässä, sovelluksessa ei ole ylläpitokäyttöliittymää, eikä käyttäjän- tai ryhmänhallintaa. Sovelluksen ylläpitäjä tunnistautuu kirjautumalla palvelinkoneelle jonka jälkeen hänellä on koneen tiedostojärjestelmän access controllin määrittämät oikeudet sovelluksen ylläpitoon liittyviin kansioihin.

4.1 Palvelinpään sovellusosa

Palvelinpään sovellus toimii Node.js-ympäristössä, ja se koostuu palvelinohjelmasta, sekä kuudesta erillisestä moduulista.

4.1.1 HTTP-rajapinta

Sovellus tarjoaa Express-sovelluskehyksellä toteutetun HTTP-rajapinnan, josta selainpään sovellus voi hakea sovelluksen HTML-, JavaScript-, ja CSS-lähdetiedostot. Lisäksi rajapinta tarjoaa sovelluksen konfiguraatiot ja web-sisältökansioiden tiedostorakenteet JSON-muodossa, sovelluksen käyttämät fontti- ja kuvatiedostot, sekä web-sisältöjen kuva- ja sapluunatiedostot.

Rajapinta reitittää saapuvat pyynnöt haetun polun perusteella. Se hyväksyy ainoastaan GET-metodia käyttäviä HTTP-pyyntöjä, koska selainpään sovellus ei voi vaikuttaa palvelinpään sovelluksen tilaan.

4.1.2 Sisällönhallinta-, Watcher- ja fileNameParser-moduulit

Sovelluksessa on erilliset moduulit alasivu-, artikkeli- ja kuvasisältöjen hallitsemista varten. Näiden moduulien päätarkoitus on valvoa niiden sisältötyypin kansion muutos-

eventtejä, pitää yllä listaa kansioiden tiedostorakenteista, sekä lokittaa tiedostojärjestelmän muutokset lokitiedostoihin.

Sisällönhallintamoduulit valvovat sisältökansioiden muutos-eventtejä luomalla instanssin watcher-moduulista löytyvästä Watcher-luokasta, joka hyödyntää Chokidar-oliota. Moduulit lukevat kansioiden sisällöt käyttäen directory-tree -oliota ja parsivat ne sovelluksen käyttämään muotoon käyttäen fileNameParser-moduulia. Lisäksi sisällönhallintamoduuli noutaa loggers-moduulista oman loggerinsa, joka passataan Watcher-instanssille. Watcher-instanssi lokittaa loggerilla tiedostomuutoksia koskevia tietoja tekstitiedostoihin.

4.1.3 Loggers-moduuli

Loggers-moduuli luo Winston-kirjastolla logger-oliot alasivu-, artikkeli- ja kuvatiedostoja varten. Nämä tiedostomuutokset lokitetaan logs-kansiossa sijaitseviin tekstitiedostoihin, sekä käyttäjän halutessa konsoliin. Lisäksi Loggers-moduulissa luodaan errorLogger-olio, joka lokittaa sovelluksen virheviestit omaan lokitiedostoonsa sekä konsoliin. Winston-kirjasto hoitaa automaattisesti lokitedostojen hallinnan, ja se säilyttää jokaisesta lokitiedostosta viisi viimeisintä alle 1 MB:n kokoista versiota.

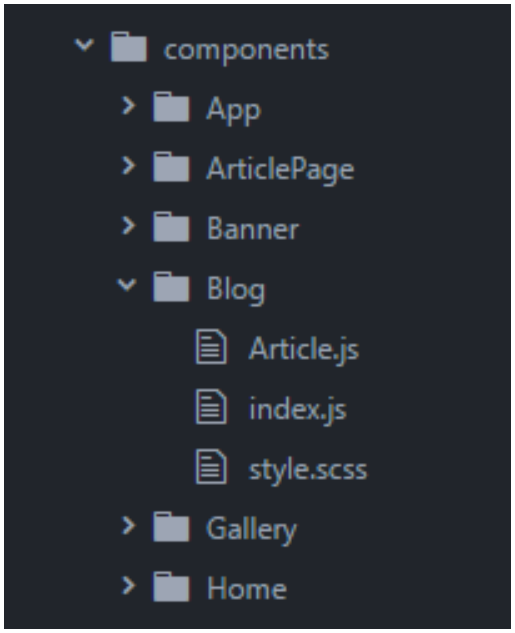
4.2 Selainpään sovellusosa

Selainpäässä toimiva osa sovelluksesta koostuu components-kansion käyttöliittymäkomponenteista, styles-kansion Sass-määritystiedostoista, sekä utility-kansion yleiskäyttöisistä JavaScript-funktioista. Sovelluksen lähdekoodi käännetään Babel-kääntäjällä ECMAScript 5 -syntaksin mukaiseksi ja se pakataan yksittäiseen bundle.js-tiedostoon Browserifylla. Sovelluksen SCSS-tiedostot on paketoitu Browserifyn Bundlify-scss -laajennoksella bundle.scss-tiedostoon, joka taas on parsittu CSS-muotoiseksi bundle.css-tiedostoksi Sass-parsijalla. Tällöin sovellus toimitetaan selaimeen bundle.js- ja bundle.css-tiedostoissa.

Selain noutaa sovelluksen konfiguraatiotiedoston sekä web-sisällöt hyödyntäen Promise- sekä Fetch-rajapintoja. Selaintuki näiden rajapintojen käytölle varmistetaan polyfill-paketeilla.

4.2.1 Käyttöliittymäkomponenttirakenne

Käyttöliittymäkomponenttihierarkia muodostuu React-kirjastoa käyttävistä käyttöliittymäkomponenteista, jotka on jaoteltu omiin moduulikansioihinsa components-kansiossa (kuva 6). Komponentin moduuli rakentuu JavaScript-lähdekoodista index.js-tiedostossa ja sen tyylimäärittelyistä style.scss-tiedostossa. Lisäksi moduulikansio saattaa sisältää sen sisällä käytettyjä alakomponentteja, kuten esimerkiksi Blog-moduulin sisältämä Article.js-tiedostossa määritelty Article-komponentti.



KUVA 6. Components-kansion sisältämien moduulikansioiden rakenne

Index.js-tiedostossa tuodaan käyttöliittymäkomponentin käyttämät moduulit ja tyylitiedosto, sekä määritellään sen toiminnallisuus. Komponentin HTML-rakenne määritellään JSX-muodossa komponenttiluokan render-metodissa. Tyylitiedostossa määritellään komponentin muotoilusäännöt sekä animaatiot hyödyntäen style-kansiossa määriteltyjä Sass-muuttujia.

4.2.2 Oleellisimmat käyttöliittymäkomponentit

App-komponentti vastaa käyttöliittymän päänäkymästä. Se sisältää yläbannerin, navigaatiopalkin, sekä päänäkymän, jossa sovellus esittää web-sisältöä. App-komponentti vastaa myös sovelluksen konfiguraatioiden noutamisesta JSON-muotoisena palvelimelta, sekä asetusten käyttämisestä sovelluksessa.

Navigaatiopalkista vastaava Navigation-komponentti generoi linkit sovelluksen alasivurakenteesta sekä Blog- ja Gallery-näkymistä, mikäli näitä näkymiä ei ole sovelluksessa piilotettu.

Päänäkymässä sovellus esittää web-sisältöä, kuten esimerkiksi tekstisisältöä, kuvia, tai videoita. Page-komponentti näytetään päänäkymässä, kun käyttäjä tarkastelee alasivuja. Artikkeleita näytettäessä käytetään joko Blog-komponenttia, jossa käyttäjä näkee kaikki artikkelit, tai ArticlePage-komponenttia, kun käyttäjä lukee yksittäistä artikkelia. Kuvasisältöjä käyttäjä näkee Gallery-komponentin kautta. Gallery-näkymässä käyttäjä voi selata kuvia suurempikokoisena TopModal-komponentin avulla.

Blog- ja Gallery-komponentit noutavat palvelimelta JSON-muotoisen artikkeli- tai kuvasisältökansioiden tiedostorakenteen, jonka perusteella ne generoivat listat sisällöistä näkymäänsä, sekä ohjaavat komponentteja ArticlePage- ja TopModal-komponenteille. Alasivu- ja artikkelisisällöt noudetaan palvelimelta Markdown-muotoisina, ja ne parsitaan selaimessa HTML-muotoon Marked-parsijalla. Marked-parsija on konfiguroitu suodattamaan pois sapluunatiedostoissa olevat HTML-tagit. Kuvatiedostot voivat olla joko .jpg-, .png-, tai .gif-päätteisiä.

4.2.3 Yleiset Sass-määritystiedostot

Selainpään sovelluksen style-kansio sisältää Sass-määritystiedostoja, jotka vaikuttavat globaalisti sovellukseen. Näitä määritystiedostoja muokkaamalla on helppo muuttaa koko sovelluksen asettelua, värejä, fontteja ja muuta ulkonäköä.

Kansiosta löytyvät colors.scss- ja variables.scss-tiedostot sisältävät väreihin ja asetteluun liittyviä muuttujamäärityksiä. Mixins.scss- ja functions.scss-tiedostot sisältävät

tyylimäärityksien kehittämistä helpottavien mixin- ja function-direktiivien määrityksiä. Animations.scss- ja fonts.scss-tiedostot taas sisältävät sovelluksen animaatioiden ja fonttien määrityksiä.

4.2.4 Yleiskäyttöiset JavaScript-funktiot

Utility-kansio sisältää sovelluksessa yleisesti käytettäviä funktiomäärityksiä, jotka on jaettu omiin tiedostoihinsa. ParsePagePaths-funktiota käytetään parsimaan JSON-muotoinen alasivurakenne sovelluksen käyttämään oliomuotoon.

GenerateVideoIframe-funktio taas generoi alasivu- ja artikkelisisältöihin upotettuja Youtube-videoita itse kehitetyn direktiivin avulla. Direktiivi lisätään Markdown-muotoiseen sapluunatiedostoon, ja sen HTML-muotoon parsimisen yhteydessä sisältöön luodaan iframe-elementti, jossa video esitetään. Direktiivissä määritellään kohdevideo, näytettävä otsikko, ja kehyksen mitat.

5 SOVELLUKSEN KÄYTTÖ

Kehitetty sovellus on sisällönjulkaisujärjestelmä, jonka kautta ylläpitäjä jakaa web-sisältöä sovelluksen käyttäjille. Web-sisällön hallinta ja sovelluksen konfigurointi tapahtuu muokkaamalla palvelinkoneen tiedostojärjestelmän tiedostoja, ja web-sisältöä esitetään sovelluksen käyttäjille internet-selaimella käytettävässä käyttöliittymässä. Sovelluksessa ei sinänsä ole ylläpitokäyttöliittymää eikä käyttäjänhallintaa.

5.1 Käyttöönotto ja asennus

Sovelluksen käyttöönotto vaatii, että palvelimelle on asennettu Node.js 6.9.2 tai tätä uudempi versio Node.js-ympäristöstä, jotta palvelinpään sovelluksen käyttämät ECMAScript 2015 -ominaisuudet ovat käytettävissä. Node.js-ympäristön asennuksen mukana asentuu palvelimelle myös NPM-paketinhallintatyökalu.

Sovelluksen voi noutaa Githubissa sijaitsevasta avoimesta repositorysta joko kloonaamalla sen Git-versionhallintajärjestelmällä tai lataamalla sen ZIP-pakettina. Kun sovellus on ladattu palvelinkoneelle, tulee projektin juurikansiossa suorittaa komentorivikomento ”npm install”. Tämä komento asentaa sovelluksen tarvitsemat riippuvuudet NPM-paketinhallintajärjestelmästä.

Tämän jälkeen tulee samassa sijainnissa suorittaa komentorivikomento ”npm run build”, joka suorittaa projektiin räätälöidyn build-skriptin. Tämä skripti kääntää ja parsii selainpään sovelluksen JavaScript- ja SCSS-tiedostot selainyhteensopivaan muotoon, sekä paketoi ne yksittäisiksi tiedostoiksi.

Sovellus käynnistetään Node.js-ympäristössä suorittamalla projektin juurikansiossa komentorivikomento ”node server.js”.

5.2 Käyttöliittymä

Selaimella käytettävän käyttöliittymän näkymä (kuva 7) koostuu bannerista, navigaatiopalkista ja päänäkymästä, jossa käyttäjälle esitetään web-sisältöä.

Käyttöliittymä on SPA-sovellus, joten sivulla navigoiminen ei laukaise sivun lataamista uudelleen, vaan web-sisältöä ja muuta dataa haetaan tarvittaessa asynkronisesti näkymän vaihtuessa.

KUVA 7. Web-käyttöliittymän näkymän banner, navigaatiopalkki, ja alasivunäkymä

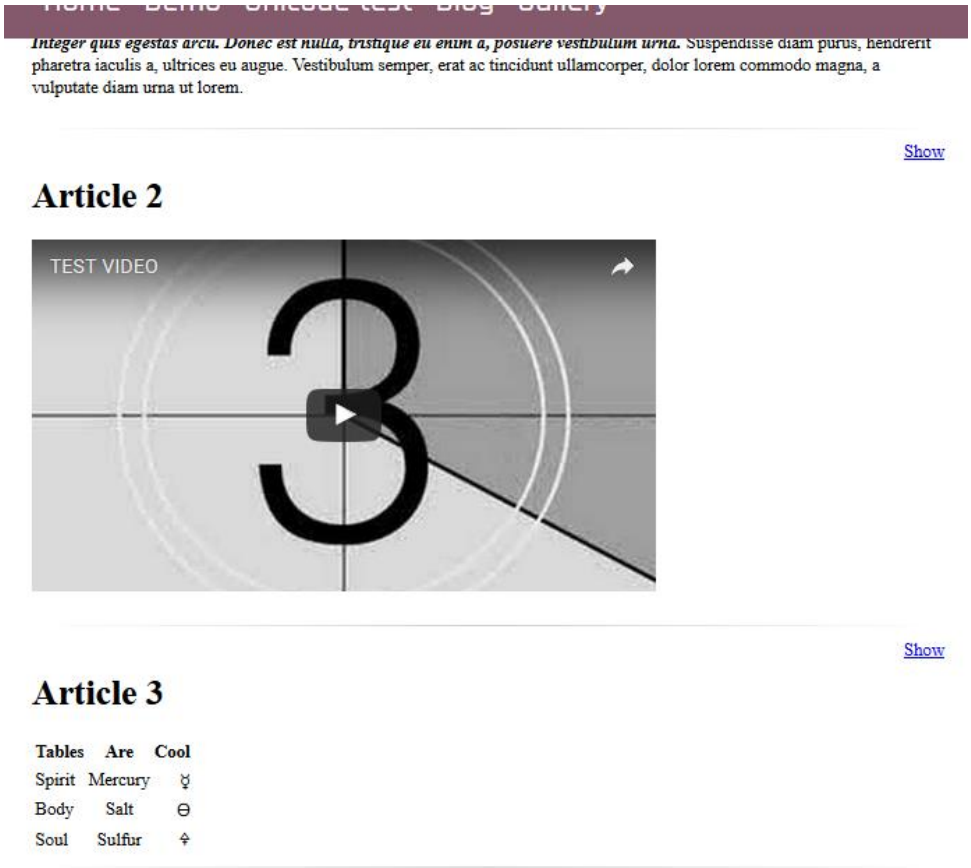
5.2.1 Banner ja navigaatiopalkki

Web-käyttöliittymän yläreunassa esiintyvän banner-elementin logo ja teksti ovat ylläpitäjän muokattavissa. Navigaatiopalkin linkit generoidaan alasivutyyppisten web-sisältöjen pohjalta, ja niihin lisätään linkit Blog- ja Gallery-näkymiin, mikäli näitä näkymiä ei ole piilotettu. Blog- ja Gallery-linkkien tekstit ovat myös säädettävissä ylläpitäjän taholta.

5.2.2 Päänäkymä

Käyttöliittymän päänäkymässä esitetään joko alasivu-, Blog-, Gallery- tai ArticlePage-näkymä. Alasivunäkymässä esitetään yksittäinen alasivu-tyyppinen web-sisältö. Yksittäisen artikkelin lukemiseen tarkoitettu ArticlePage-näkymä esittää yksittäisen artikkeli-tyyppisen web-sisällön samalla tavalla, kuin alasivunäkymä alasivusisällön.

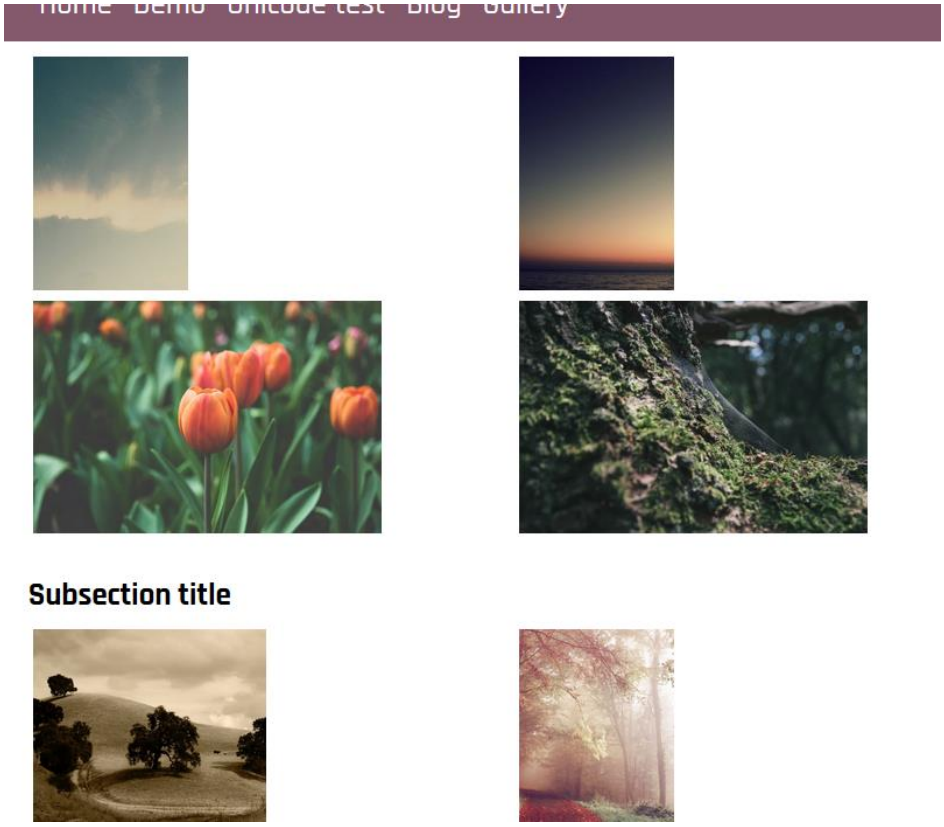
Blog-näkymä (kuva 8) esittää artikkelityyppiset web-sisällöt aikajärjestyksessä uusimmasta vanhimpaan. Bloginäkymässä artikkelielementtien oikeaan yläkulmaan luodaan linkki, joka ohjaa kyseisen artikkelin ArticlePage-näkymään. Linkin teksti on ylläpitäjän muokattavissa.



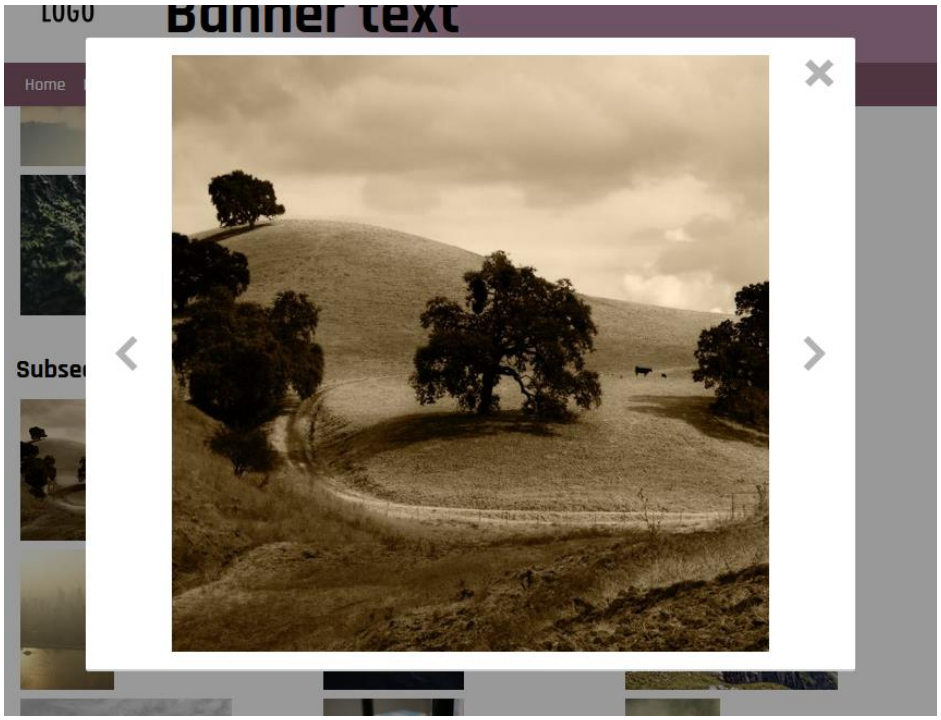
KUVA 8. Käyttöliittymän Blog-näkymä

Gallery-näkymä (kuva 9) esittää kuvatyyppisiä web-sisältöjä aikajärjestyksessä uusimmasta vanhimpaan. Kuvasisältöjä on mahdollista jakaa myös otsikoituihin kuvakokoelmiin. Gallery-näkymän kuvia on mahdollista selata suurikokoisempina galleriamodaali-ikkunassa (kuva 10), jossa ikkunan sivuilla olevat nuolet hakevat

aikajärjestyksessä seuraavan tai edellisen kuvan, ja yläkulman rasti sulkee modaali-ikkunan. Modaali-ikkuna aukeaa, kun käyttäjä klikkaa kuvaa.



KUVA 9. Käyttöliittymän Gallery-näkymä



KUVA 10. Gallery-näkymän modaali-ikkuna

5.3 Ylläpito

Sovelluksen ylläpitäjän tehtäviin kuuluu sovelluksen konfigurointi sekä web-sisällön tuottaminen ja hallinta. Nämä toimenpiteet suoritetaan muokkaamalla sovelluksen files-kansiossa sijaitsevia sisältökansioita ja config-kansiota. Tämän ylläpitäjä voisi tehdä palvelinkoneen ulkopuolelta esimerkiksi SSH- tai SCP-protokollaa käyttävillä asiakasohjelmilla.

Alasivu-, artikkeli-, ja kuvasisällöt, sekä konfiguraatiotiedostot ovat kaikki omissa kansioissaan. Tällöin sovelluksen toimiessa esimerkiksi Linux-palvelimella on ylläpitäjien oikeuksia eri ylläpitotoimenpiteisiin helppo hallita muokkaamalla erillisten sisältökansioiden kirjoitusoikeuksia.

5.3.1 Sovelluksen konfigurointi

Sovellusta konfiguroidaan muokkaamalla files-kansiossa sijaitsevan config-kansion sisältämiä tiedostoja. Kansiosta löytyvää config.json-tiedostoa (kuva 11) muokkaamalla ylläpitäjä voi muokata käyttöliittymässä näkyviä tekstejä, kuten esimerkiksi bannerin tekstiä, sekä piilottaa Blog- ja Gallery-näkymät.

```
{
  "title": "My CDA",
  "banner": "Banner text",
  "blogTitle": "Blog",
  "galleryTitle": "Gallery",
  "blogLinkText": "Show",
  "hideBlog": false,
  "hideGallery": false
}
```

KUVA 11. Config.json-tiedoston esimerkkisisältö

Bannerin logo vaihdetaan korvaamalla kansiosta oletuksena löytyvä logo.png-tiedosto uudella saman nimisellä tiedostolla. Ylläpitäjä voi myös räätälöidä sovelluksen väriteeman muokkaamalla config-kansion colors.scss-tiedostoa (kuva 12). Tiedostossa

määritetään SassScript-muuttujia, jotka korvaavat oletusarvoja sovelluksen tyylitiedostoissa.

KUVA 12. Colors.scss-tiedoston esimerkkisisältö

5.3.2 Web-sisältöjen hallinta

Sovelluksen web-sisältöjä hallitaan käsittelemällä sisältökansioiden tiedostoja. Tiedostomuutokset heijastuvat sisältömuutoksina välittömästi sovelluksen web-käyttöliittymään.

Alasivuja muokataan muuttamalla pages-kansiossa Markdown-merkintäkielisiä sapluunatiedostoja. Uusia alasivuja taas luodaan lisäämällä kansioon uusia sapluunatiedostoja. Alasivujen nimet ja järjestys käyttöliittymässä määritetään sapluunatiedostojen nimillä. Tiedostot on tarkoitus nimetä muodossa ”[järjestysindeksi]-[sivun nimi].md”, jossa ”[järjestysindeksi]”-kohta korvataan alasivun järjestysnumerolla navigaatiopalkissa, ja kohta ”[sivun nimi]” korvataan alasivun nimellä, joka näytetään navigaatiopalkissa. Alasivulinkit järjestetään indeksin perusteella nousevaan järjestykseen. Esimerkiksi tiedosto ”4-yhteystiedot.md:n” alasivusisältö näkyisi palkissa neljäntenä tekstillä ”Yhteystiedot”, jos sapluunatiedostoista kolme on indeksoitu pienemmällä numerolla.

Bloginäkymän artikkeleita hallitaan muokkaamalla articles-kansion tiedostoja. Artikkelisapluunatiedostot toimivat alasivusapluunoiden tavoin, mutta niillä on erilainen nimeämisformaatti. Artikkelisapluunat nimetään muodossa ”vvvv-kk-dd-[järjestysindeksi].md”, missä alku korvataan artikkelin julkaisupäivämäärällä ja kohta ”[järjestysindeksi]” korvataan saman julkaisupäivän artikkeleiden julkaisujärjestysindeksillä. Artikkelit järjestetään käyttöliittymässä uusimmasta

vanhimpaan, ja korkeamman järjestysindeksin omaava artikkeli näkyy ennen matalamman indeksin omaavaa.

Alasivu- ja artikkelisapluunatiedostot tukevat Markdown-merkintäkielen ominaisuuksia, ja niihin on toteutettu myös Youtube-videon upotusominaisuus (kuva 13). Youtube-video upotetaan sisältöön lisäämällä sapluunatiedostoon itse kehitetty direktiivi (kuva 14), jossa videolle asetetaan otsikko, pikselimitat, sekä Youtube-videon ID.

KUVA 13. Web-sisältöön upotettu Youtube-video

```
2
3   !VIDEO(title:"Test video" width:"560" height:"315" id:"C0DPdy98e4c")
4
```

KUVA 14. Sapluunatiedostossa käytettävä videodirektiivi

Gallerianäkymän kuvia hallitaan muokkaamalla img-kansion tiedostoja. Sovellus hyväksyy JPG-, PNG-, ja GIF-tyyppiset kuvatiedostot, ja niiden järjestys käyttöliittymässä määritetään samalla nimeämistavalla kuin artikkelisapluunoiden nimeämisessä.

Gallerianäkymän kuvakokoelma luodaan lisäämällä img-kansioon alakansio, joka sisältää kokoelman kuvat. Kuvakokoelman sijainti käyttöliittymässä määritetään nimeämällä kansio samalla tavalla kuin kuvatiedostot, mutta jättämällä tiedostopääte pois

sen nimestä. Kuvakokoelman otsikko määritetään lisäämällä alakansioon title.json-tiedosto (kuva 15), jossa otsikko määritetään. Alakansion kuvat esitetään käyttöliittymässä laskevassa aakkosjärjestyksessä, joten helpointa on nimetä ne esimerkiksi muodossa ”[järjestysindeksi].[tiedostopääte]”.

```
1  ["Subsection title"]
2
```

KUVA 15. Kuvakokoelmakansion title.json-tiedoston esimerkkisisältö

6 JATKOTOIMENPITEET

Projektin tuloksena syntynyt sovellus täytti sille asetetut vaatimukset, mutta sovelluksessa olisi kuitenkin vielä tilaa jatkokehitykselle. Tässä kappaleessa tarkastellaan mahdollisia jatkotoimenpiteitä, joilla sovelluskehittäjä voisi jatkokehittää ja hyödyntää sovellusta jatkossa.

6.1 Jatkokehitys

Sovelluksen ylläpitäjän käyttökokemusta voisi parantaa internet-selaimella käytettävä käyttöliittymä web-sisällön hallintaan ja sovelluksen konfigurointiin. Tässä käyttöliittymässä ylläpitäjä voisi esimerkiksi kirjoittaa sapluunoita alasivuja ja artikkeleita varten, sekä ladata kuvatiedostoja. Lisäksi käyttöliittymässä voisi luoda kuvakokoelmia gallerianäkymään, poistaa ja muokata olemassa olevaa sisältöä, sekä muuttaa sovelluksen konfiguroitavia tekstejä ja värejä. Sovellus voisi hoitaa tiedostojen nimeämisen, tiedostosyntaksien noudattamisen, sekä muut teknisempää lähestymistä vaativat osat ylläpidosta automaattisesti.

Ylläpitokäyttöliittymä pitäisi kuitenkin sulkea ulkopuolisilta, mikä vaatisi jonkinlaista tunnistautumista sen käyttäjältä. Sovellukseen pitäisi toteuttaa yksinkertainen käyttäjänhallinta, tai käyttöliittymä voisi olla eristetty ulkoverkosta, jolloin ylläpitäjä voisi käyttää sitä VPN (virtual private network) -yhteyden kautta.

6.2 Sovelluksen kaupallinen hyödyntäminen

Sovelluksen lähdekoodi aiotaan pitää avoimena Githubissa, mutta on olemassa tapoja, joilla open source -sovellusta voi hyödyntää rahallisesti.

6.2.1 Mehiläishoitajamalli

Mehiläishoitajamalli (beekeeper model) jakaa sovelluksen käyttäjät kahteen ryhmään: mehiläisiin ja asiakkaisiin. Mehiläisiksi kutsutaan sovelluksen ympärille rakentuneen

yhteisön sovelluskehittäjiä, jotka työskentelevät avoimen lähdekoodin sovelluksen parissa pyrkien korjaamaan sen virheitä ja kehittämään sen ominaisuuksia. Asiakkaat taas ovat yrityksiä, jotka käyttävät sovellusta ilmaiseksi, mutta maksavat sovellukseen liittyvästä asiakastuesta, palveluista ja koulutuksesta. (Dixon 2007.)

Mehiläishoitajamallin kannattava käyttö vaatii sovelluksen haltijan taholta aktiivisuutta yhteisön parissa, jotta sovelluskehittäjät ovat kiinnostuneita kehittämään sovellusta. Lisäksi maksavien yritysasiakkaiden löytäminen vaatii sovellukselta suurta markkinanäkyvyyttä. Mehiläishoitajamallin on kuitenkin saanut toimimaan esimerkiksi avoimen lähdekoodin dokumentinhallintasovelluksia kehittävä Alfresco.

Projektissa kehitetylle sisällönhallintasovellukselle mehiläishoitajamallin kannattava hyödyntäminen voi olla toistaiseksi saavuttamattomissa, koska sovelluksella ei ole näkyvyyttä yritysmaailmassa eikä open source -kehittäjien parissa.

6.2.2 Freemium-malli

Freemium-mallin mukaisessa bisnesmallissa tuotteesta on tarjolla ilmainen versio, sekä maksullinen lisäominaisuuksia tarjoava premium-versio. Sovelluksen ilmaisversion tulee tarjota riittävän kattavasti ominaisuuksia, jotta asiakas kokee sovelluksen käyttökelpoiseksi ja tarpeelliseksi. Tämä motivoi asiakkaita maksamaan lisähyödyistä. (Balaji 2016.)

Tärkeää freemium-mallin hyödyntämisessä on, että tuotetta myyvälle yritykselle ei aiheudu suuria kustannuksia lukuisista ilmaisversiota käyttävistä asiakkaista. Chris Anderson kertoo aiheesteen liittyvässä kirjassaan, että tuotetta käyttävistä asiakkaista noin 5 % maksaa tuotteesta, ja tämän ryhmän tuki kattaa koko käyttäjäkunnan aiheuttamat kulut. Freemium-mallia hyödyntää esimerkiksi pilvipalveluja tarjoava Dropbox. (Balaji 2016.)

Jos freemium-mallia haluttaisiin hyödyntää sisällönjulkaisusovelluksessa, premium-ominaisuutena voisi toimia esimerkiksi jatkokehityksessä tehtävä ylläpitokäyttöliittymä. Premium-ominaisuudet tulisi pitää suljetussa versionhallinnassa, joten ne pitäisi kehittää erillisessä repositoryssa.

6.2.3 Mainosten näyttäminen käyttäjille

Yleinen keino tuottaa tuloja verkkosovellusten kautta on näyttää mainoksia verkkosivuilla. Mainosten näyttäminen ei aiheuta suurta haittaa käyttäjäkokemukselle, eikä se juurikaan edellytä ylläpitotyötä. Hyödyntämällä valmiita mainospalveluita, kuten Googlen AdSensea, on helppoa alkaa tienaamaan rahaa välittömästi, kun sovellus saa käyttäjiä. Google AdSense mahdollistaa hyödyllisiä toimintoja, kuten mainosten kohdentamisen käyttäjäkunnalle ja niiden räätälöimisen sivun ulkonäköön (Google 2017).

Sisällönjulkaisusovelluksen latauslinkin, käyttöohjeen ja dokumentaation voisi laittaa erilliselle verkkosivustolle, jolla näytettäisiin kohdistettuja mainoksia. Tällä keinolla voidaan alkaa tuottaa voittoja heti sovelluksen julkaisemisen alkutaipaleella ja muiden bisnesmallien käyttöä voitaisiin harkita, jos sovellus onnistuu saavuttamaan laajemman käyttäjäkunnan. Mainosten näyttäminen olisi realistisin keino ansaita rahaa tämänhetkisellä sovelluksella.

7 YHTEENVETO

Projektin tavoitteena oli kehittää toimiva ja helppokäyttöinen sisällönjulkaisusovellus, joka on käytettävissä web-käyttöliittymän kautta. Sovellusta käytetään web-sisällön, kuten esimerkiksi kuvien ja artikkeleiden esittämiseen sovelluksen käyttäjille, ja käyttöliittymä on avoin kaikille. Sovelluksesta päätettiin tehdä täysin JavaScript-pohjainen web-sovellus, joka hyödyntää palvelinpäässä Node.js-ympäristöä, ja jonka käyttöliittymä toteutetaan React-käyttöliittymäkirjastolla. Sovelluskehittäjä valitsi nämä tekniikat johtuen halusta tutustua syvemmin Node.js- ja React-sovelluskehitykseen.

Sovelluskehityksessä hyödynnettiin ECMAScript 2015 -standardin mukaisia JavaScript-ominaisuuksia, ja sovelluksen tyylitiedostojen kehitys toteutettiin CSS-kielen Sass-laajennoksella. Lähdetiedostot käännettiin selainyhteensopivaan muotoon Babel-kääntäjällä ja Sass-parsijalla, jonka jälkeen ne paketoitiin Browserify-työkalulla. Sovelluksen lähdekoodin versioita hallittiin Git-järjestelmällä ja ne säilytettiin avoimessa repositoryssa Github-palvelussa.

Sovelluksen palvelinpäähän toteutettiin Express-sovelluskehyksellä HTTP-rajapinta, josta selainpään sovellus hakee web-sisältöjen tiedot ja tiedostot. Sovellus valvoo web-sisältökansioita Chokidar-kirjastolla, ja päivittää sisällöt sovellukseen tiedostomuutosten yhteydessä. Lisäksi sovellus lokittaa tietoja sovelluksesta dynaamisesti hallittuihin lokitedostoihin Winston-kirjastolla. Selainpään sovellusosa toteutettiin dynaamisena SPA-sovelluksena, joka esittää alasivu-, artikkeli-, ja kuvasisältöä omissa näkymissään. Sovellus hyödyntää moderneja selainrajapintoja, joiden selainyhteensopivuus varmistettiin polyfill-paketeilla.

Tuloksena saatiin valmis ja käyttökelpoinen web-sovellus, joka toteutti sille asetetut vaatimukset. Sovelluksen käyttäjät voivat tarkastella käyttöliittymän kautta alasivu-, artikkeli-, ja kuvasisältöä. Sovelluksen ylläpitäjä hallitsee web-sisältöä muokkaamalla palvelinkoneella sijaitsevien sisältökansioien tiedostoja. Alasivu- ja artikkelisisältö kehitetään Markdown-kielisinä sapluunatiedostoina, ja kuvasisällöksi kelpaavat yleisten kuvaformaattien mukaiset kuvatiedostot. Sisällönhallinnan lisäksi ylläpitäjä voi konfiguroida käyttöliittymän ulkonäköä ja tekstisisältöä muokkaamalla sovelluksen konfiguraatiotiedostoja.

Sovelluskehitysprosessin tuloksena sovelluskehittäjä saavutti syvemmän tuntemuksen käytettyihin web-tekniikoihin ja kehitystyökaluihin. Sovellus pidetään toistaiseksi open source -projektina, ja sovelluskehittäjä harkitsee mahdollista jatkokehitystä tai sovelluksen tuotteistamista.

LÄHTEET

Brehn, Spike 2013. Isomorphic JavaScript: The Future of Web Apps. Luettu 31.5.2017. https://medium.com/airbnb-engineering/isomorphic-javascript-the-future-of-web-apps-10882b7a2ebc

Sotelo, Caleb 2014. Evolution of the Single Page Application. Luettu 31.5.2017. http://paislee.io/evolution-of-the-single-page-application/

Hernandez, Alejanhdro 2013. An Introduction To Full Stack JavaScript. Luettu 31.5.2017. https://www.smashingmagazine.com/2013/11/introduction-to-full-stack-javascript/

Mozilla Developer Network 2015. JavaScript. Luettu 9.5.2017. https://developer.mozilla.org/en-US/docs/Web/JavaScript

WebKit 2014. JavaScriptCore. Luettu 9.5.2017. https://trac.webkit.org/wiki/JavaScriptCore

Laurens, Thibalt 2015. How the V8 engine works?. Luettu 9.5.2017. http://thibaultlaurens.github.io/javascript/2013/04/29/how-the-v8-engine-works/

Mather, Chris 2014. What is a Runtime?. Opetusvideo. Katsottu 9.5.2017. https://www.eventedmind.com/classes/the-javascript-runtime/what-is-a-runtime

Node.js Foundation 2017. Node.js. Luettu 10.5.2017. https://nodejs.org/en/

Meyerson, Jeff 2015. JavaScript: Fundamental Answers. Luettu 10.5.2017. https://softwareengineeringdaily.com/2015/08/02/javascript-fundamental-answers/

Takada, Mikito 2011. Understanding the node.js event loop. Luettu 10.5.2017. http://blog.mixu.net/2011/02/01/understanding-the-node-js-event-loop/

Facebook 2017. React. Luettu 11.5.2017. https://facebook.github.io/react/

Haberman, Josh 2014. React Demystified. Luettu 11.5.2017. http://blog.reverberate.org/2014/02/react-demystified.html

Krajka, Bartosz 2015. The difference between Virtual DOM and DOM. Luettu 11.5.2017. http://reactkungfu.com/2015/10/the-difference-between-virtual-dom-and-dom/

Node.js Foundation 2016. Express. Luettu 13.5.2017. https://expressjs.com/

Hahn, Evan 2014. Understanding Express.js. Luettu 13.5.2017. https://evanhahn.com/understanding-express/

Polychronakis, Thanasis 2014. Intro to node.js. Kalvosarja. Luettu 13.5.2017. https://www.slideshare.net/thanpolas/intro-to-nodejs-39066435

Miller, Paul 2016. Chokidar. Github-repository. Luettu 13.5.2017.
https://github.com/paulmillr/chokidar

Dobrescu-Balaur, Mihnea 2017. Directory-tree. Github-repository. Luettu 13.5.2017.
https://github.com/mihneadb/node-directory-tree

Robbins, Charlie 2017. Winston. Github-repository. Luettu 13.5.2017.
https://github.com/winstonjs/winston#logging-with-metadata

Gruber, John 2004. Markdown. Luettu 13.5.2017.
https://daringfireball.net/projects/markdown/

Github 2014. Mastering Markdown. Luettu 13.5.2017.
https://guides.github.com/features/mastering-markdown/

Jeffrey, Christopher 2014. Marked. Github-repository. Luettu 13.5.2017.
https://github.com/chjj/marked

Babel 2017. Babel-dokumentaatio. Luettu 14.5.2017. https://babeljs.io/

Browserify 2017. Github-repository. Luettu 29.5.2017.
https://github.com/substack/node-browserify

Hampton, C., Weizenbaum, N., Eppstein, C. 2016. Sass-dokumentaatio. Luettu 29.5.2017. http://sass-lang.com/documentation/file.SASS_REFERENCE.html

Vogel, Lars 2016. Git - Tutorial. Luettu 30.5.2017.
http://www.vogella.com/tutorials/Git/article.html

Atlassian 2017. What is Git. Luettu 30.5.2017.
https://www.atlassian.com/git/tutorials/what-is-git

Finley, Klint 2012. What Exactly Is GitHub Anyway?. Luettu 30.5.2017.
https://techcrunch.com/2012/07/14/what-exactly-is-github-anyway/

Dixon, James 2007. The Beekeeper Model. Luettu 31.5.2017.
http://wiki.pentaho.com/display/BEEKEEPER/3.+The+Beekeeper+Model

Balaji, Sadhana 2016. What makes Freemium work?. Luettu 31.5.2017.
https://www.chargebee.com/blog/freemium-business-model/

Google 2017. Google AdSense. Luettu 31.5.2017.
https://www.google.co.uk/adsense/start