Haaga-Helia
ammattikorkeakoulu Oy

Moderni web-sovellus hierojalle

Janne Berg

Opinnäytetyö
Tietojenkäsittelyn
koulutusohjelma
2018

Tekijä(t)
Janne Berg

Koulutusohjelma
Tietojenkäsittelyn koulutusohjelma

Raportin/Opinnäytetyön nimi
Moderni web-sovellus hierojalle.

Sivu- ja liitesivumäärä
36 + 2

Projektin tavoitteena oli luoda modernin näköinen web-sovellus hieroja yrittäjälle. Web-sovelluksella pystyy varaamaan hieronta-aikoja ajanvarausjärjestelmällä, sekä se sisältää näkymät hierojan esittelystä, hinnastosta ja yhteystiedoista. Sovellus toimii responsiivisesti kaikilla päätelaitteilla.

Teoriaosuudessa käydään läpi projektissa käytetyt teknologiat, kuten Sketch, Invision, Github, React.js ja Firebase. Luvussa käydään tarkemmin läpi suunnitteluun käytetyt teknologiat, sekä Reactin perusteet, jotka olivat olennainen osa sovelluksen kehitystä.

Toiminnallisessa osuudessa pyritään perustelemaan valitut teknologiat ja kerrotaan, miten niitä käytettiin projektissa. Luvun tarkoitus on antaa kokonaiskuva kehitysprosessista, käymällä läpi vaihe vaiheelta, miten web-sovellus luotiin. Jokainen sovelluksen näkymä käydään läpi.

Viimeisessä Pohdintaosuudessa käydään muun muassa annettuihin tavoitteisiin pääsemisestä, aikataulua, mitä opittiin ja mitkä asiat olivat helppoja, sekä mitkä tuottivat vaikeuksia. Osuuden lopussa käsitellään jatkokehitysideoita.

Asiasanat
Web-sovellus, React.js, Firebase, SPA

Sisällys

1 Johdanto

Opinnäytetyön tarkoituksena ja tavoitteena oli saada aikaan web-sovellus hierojayrittäjälle. Web-sovellus suunniteltiin toimimaan yrittäjän vaatimusten mukaan. Yrittäjälle Web-sovelluksen tarkoitus on saada lisää näkyvyyttä ja sitä kautta lisää asiakkaita, sekä mahdollistaa asiakkaiden hierontahoitojen ajanvaraukset netin kautta. Asiakasvarauksien haluttiin tallentuvan tietokantaan. Sovelluksen näkymissä oli tarkoituksena näkyä yrittäjän kuvaus itsestään, yhteystiedot, ajanvarausjärjestelmä ja hinnasto.

Sovellus suunniteltiin ja luotiin toimimaan kaikilla päätelaitteilla, moderneja teknologioita käyttäen.

Suunnitteluvaiheessa piirrettiin sovelluksen käyttöliittymästä kuvat Sketch piirtoeditorilla. Kuvat piirrettiin mobiili- ja työpöytänäkymistä. Tämän jälkeen kuvista luotiin prototyyppi Invision prototyyppien tekoon tarkoitetulla web-työkalulla. Prototyyppi testattiin käyttäjillä.

Koodausvaiheeseen siirryttiin, kun prototyypillä oli tehty testaukset. Sovelluksen pääteknologiana toimii React.js, joten web-sivusta tuli SPA sovellus. Tietokannaksi valittiin Firebase, minne asiakkaiden varaukset tallentuvat. Sivuston skaalautuvuus tehtiin CSS/SASS teknologioita käyttäen.

Web-sovelluksen luomiseen käytettiin myös seuraavia teknologioita ES6, jQuery, JavaScript, BABEL, Node.js, HTML5, Atom ja GitHub.

1.1 Käsitteet

HTML (Hypertext Markup Language) = Web-sivujen tekemiseen tarkoitettu ohjelmointikieli, jolla voidaan luoda sivuston rakenne luomalla html-elementtejä.

CSS (Cascading Style Sheets) = Koodikieli jota käytetään WWW-dokumenttien tyylittelyyn. Esimerkiksi HTML dokumentit tyylitellään haluttaessa CSS:llä.

SASS (syntactically awesome stylesheets) = CSS:n laajennuskieli, jolla helpotetaan CSS:n käyttöä ja rakennetta.

SPA(Single Page Application) = Yhden sivun sovellus. Verkkosovellus tai verkkosivu joka päivittyy dynaamisesti yhdelle sivulle.

React.js = JavaScript kirjasto jolla voidaan tehdä SPA sovelluksia.

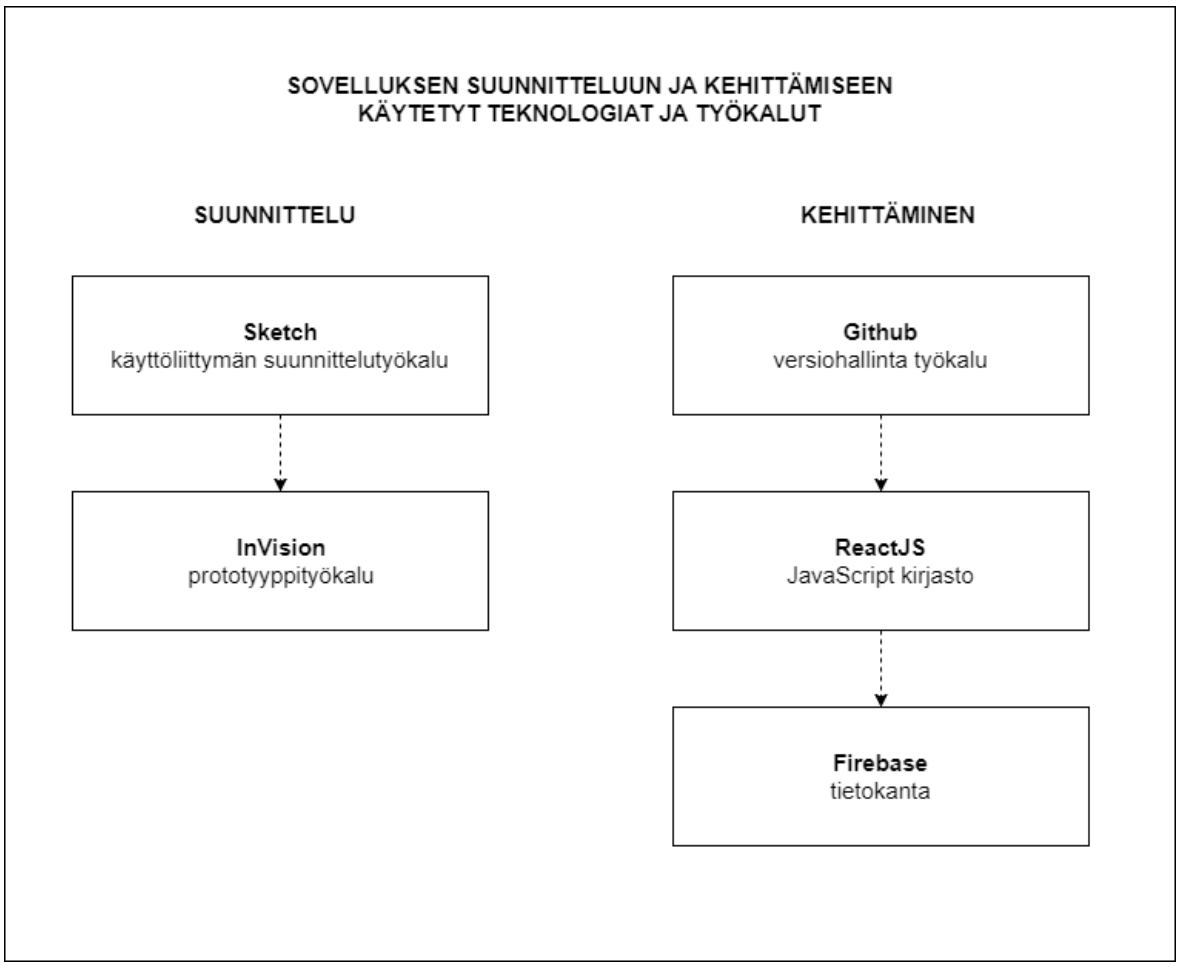
Babel = Eri JavaScript kirjastojen koodien syntaksien kääntäjä. Kääntää koodin lukukelpoiseksi selaimelle.

State/Tila = State on React komponentin tila.

ES6 (ECMAScript 6) = Standardoitu ohjelmointikieli, jota käytetään JavaScriptin kanssa.

2 Teknologiat sovelluksen suunnitteluun ja luomiseen

Tässä luvussa käydään läpi web-sivun suunnitteluun, sekä luomiseen käytettyjä teknologioita ja sovelluksia. Suunnitteluun käytettiin Sketch editoria, sekä Invision prototyyppityökalua. Web-sivun koodaukseen käytettiin Github, ReactJS ja Firebase teknologioita.



Kuva 1. Käytetyt työkalut ja teknologiat

2.1 Sketch

Sketch-ohjelmisto on vektorigrafiikkaeditori, jota käytetään digitaaliseen suunnitteluun. Sen kehitti hollantilainen yritys Bohemian Coding. Sketchin ensimmäinen julkaisu tapahtui seitsemäs päivä syyskuuta 2010. Se voitti Apple Design Awardin vuonna 2012. Sketch ohjelmisto toimii ainoastaan Mac OS käyttöjärjestelmässä. (CNET 2017.)

Sketchin vahvuus tulee juuri siitä, että se on suunniteltu erityisesti käyttöliittymien, websivujen, iconien ja logojen luomiseen. Käyttäjä tarvitsee lisenssin käyttääkseen ohjelmistoa. Lisenssi ostetaan kerran vuodessa. Jos ei halua uusia lisenssiä seuraavana vuonna, niin voi silti käyttää ohjelmistoa, mutta ei saa uusimpia päivityksiä. Eli käytännössä käyttä-

jä maksaa päivityksistä. Muut maksulliset editorit ovat yleensä kuukausivelotteisia ja maksavat niin kauan kuin käyttäjä niitä käyttää, mikä voi tulla yllättävän kalliiksi käyttäjälle. (Sketch 2017a.)

Sen lukuisista hyödyistä merkittävimmät ovat natiivin ohjelmiston hyödyt, vektorigrafiikan tarkkuus, työn exportointi ja katselmointi, sekä työkaluvalikon laajuus. (Sketch 2017b.)

Sketch on rakennettu yksinomaan Mac:lle, täysin natiivina ohjelmistona ja näin se ottaa kaiken hyödyn Applen sovelluskehyksistä. Tämän johdosta Sketch toimii sulavasti ja nopeasti raskaidenkin projektien parissa. Sketch haitoiksi voi laskea sen, ettei se taivu muille käyttöjärjestelmille, kuten Windowsille. (Apple 2017a.)

Vektorigrafiikan ansiosta objekteja pystyy tarkkailemaan pixelin tarkkuudella. Sketch pyöristää objektin reunat lähimpään pikseliin, näin tekemällä objekti pysyy aina sulavareunaisena. Vektorigrafiikka mahdollistaa siis sulavan objektien skaalauksen, ilman että mittasuhteet tai tarkkuus kärsii.
Sketchin exportointi ominaisuudet ovat varsin kattavat ja helppokäyttöiset. Projektit ja yksittäiset piirtoalueet voi tuoda eri tiedostomuodoissa ulos, kuten svg, jpg, png jne. Ne voi myös skaalata haluamansa kokoiseksi exporttia suorittaessa. Vektorigrafiikan ansiosta käyttäjä ei menetä objekteista tarkkuutta. Sketch mahdollistaa myös kuvioiden ja tekstiobjektien CSS määrityksien exportoinnin. (Sketch 2017c.)

Projektin katselmointi ja testaus on myös otettu huomioon. Projektin piirtoalueet voi testata eri näkymissä kuten mobiili, pöytäkone, tabletti jne. Testinäkymiä löytyi myös varsin paljon laitekohtaisilla vaihtoehdoilla. Sketchin ohessa voi käyttää myös Sketch Mirror sovellusta, jolla voi katsella, testata ja jakaa tekemiä projekteja. (Apple 2017b.)

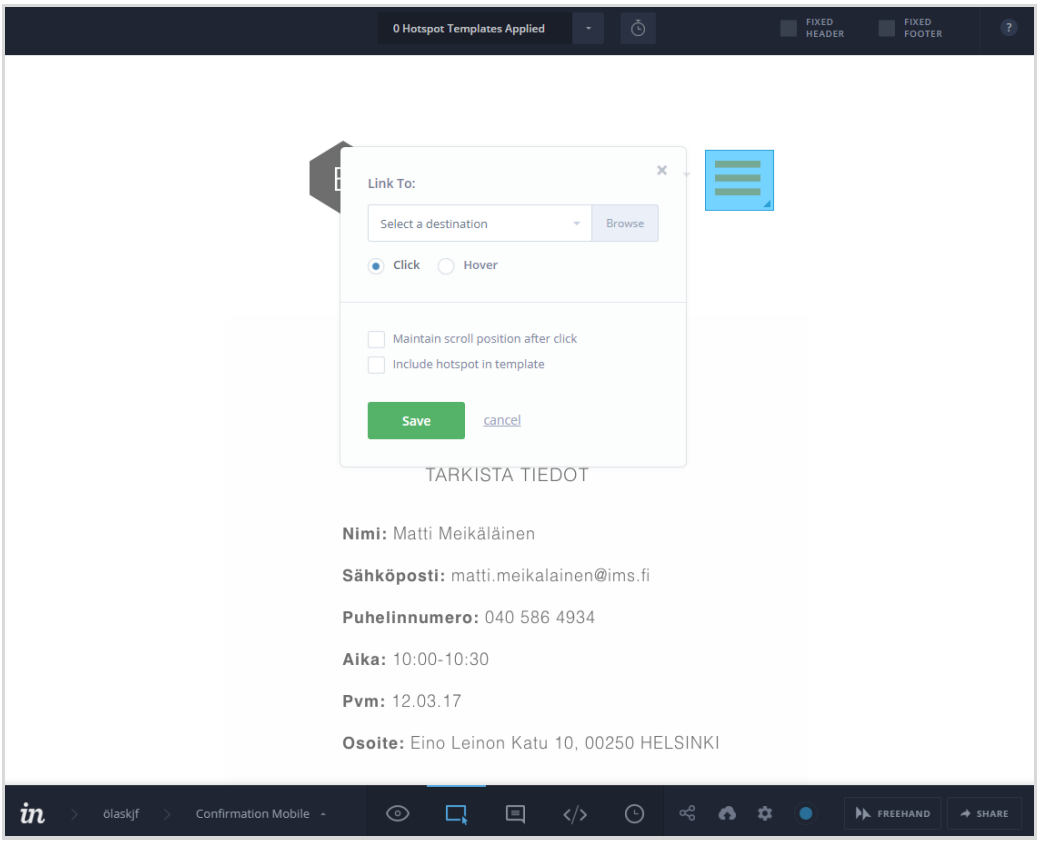
Työkaluvalikko ohjelmassa on suhteellisen laaja ja selkeä. Valikosta löytyy perus grafiikkaeditorin työkaluja kuten valmiit kuviot, teksti luonti, kuvan tuonti, objektin leikkaus jne. Elementtien muokkaamiseen löytyy kaikki perus grafiikkaeditorin ominaisuudet, kuten elementin täyttö ominaisuudet, varjostus, koko ja sijainti, sekä kerrosasettelu. (Sketch 2017d.)

Käyttöliittymäsuunnittelussa käytetään usein samoja elementtejä, kuten nappeja, ikoneja, logoja jne. Sketchin työkaluilla pystyy luomaan helposti kyseisiä elementtejä, joita on myös mahdollista tallentaa omiin elementteihin. Näin käyttäjä pystyy uudelleenkäyttämään valmiita elementtejä muissakin projekteissa. (Sketch 2017e.)

2.2 Invision

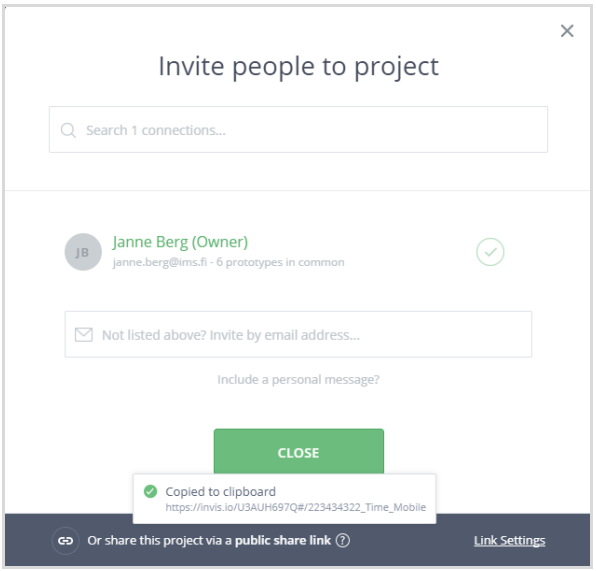
Uuden sovelluksen, projektin luonti on aina hieman hakuammuntaa, usein joudutaan palaamaan askelia takaisinpäin, kun ei ole pystytty suunnittelemaan ja testaamaan projektin toimivuutta tarpeeksi hyvin. Tämä vie yleensä aikaa ja tulee varsin kalliiksi. InVision tarjoaa tämän kaltaisiin ongelmiin erittäin pätevät työkalut, jotta suunnittelu on mahdollisimman hyvin toteutettu. (Invision 2017a.)

InVision sovellus on nykyään yksi maailman suosituimpia prototyyppien kehittämiseen tarkoitettu web-työkalu. Se kehitettiin New Yorkissa vuonna 2011. Sen voi hankkia ilmaiseksi, mutta ilmaisversiolla käyttäjä voi luoda vain yhden aktiivisen prototyypin. Jos käyttäjällä on enemmän käyttöä sovellukselle hän voi ostaa eri pakettivaihtoehtoja. Starter paketti sisältää 3 prototyyppiä ja maksaa 15$/kk. Professional paketissa loputon määrä prototyyppejä ja 25$/kk. Team paketti sisältää loputtoman määrän prototyyppejä ja 5 henkilölle lisenssit, maksaa 99$/kk. Erikseen on vielä yrityksille omat paketit. Useat huippu yritykset, kuten Twitter, Netflix, Adobe, Salesforce ja Airbnb käyttävät InVisionia projektiensa suunnitteluun. (Invision 2017b.)



Kuva 2. InVision prototyypin editointi näkymä.

InVisionin avulla projektin tekijä pystyy luomaan lopputuotteensa näköisen prototyypin. Prototyyppi rakennetaan pelkästään kuvilla, koodia ei siis kirjoiteta riviäkään. Kuviin lisätään työkaluilla eri tapahtumat. Tapahtumat voivat olla esimerkiksi klikkaus, kuvien vaihto, hover, linkitys tai vaikka veto, liukutapahtumat. InVisionilla pystyy myös testaamaan prototyypin varsin kätevästi. Testaukseen InVision tarjoaa käyttäjälle web-linkin, jonka kautta näkee prototyypin ns. livetilassa haluamallaan laitteella. Tämän avulla voidaan tehdä esimerkiksi käyttäjätestausta. Linkin pystyy myös jakamaan esimerkiksi projektitiimin kanssa, joten projektin seuraaminen, kommentointi, testaus onnistuu varsin kätevästi. Linkin aukaisuun ei tarvita InVisionia. (Invision 2017a.)



Kuva 3. Projektin jakaminen linkillä muille tiimin jäsenille.

Sketch ja InVision tuotteet ovat yhteensopivat, joten prototyypin luominen onnistuu vaivatta. Sketchillä pystyy valitsemaan piirtoalueet ja objektit, jotka lähetetään suoraan InVisionille, joka luo automaattisesti prototyyppi-projektin käyttäjälle. (Invision 2017c.)

2.3 Github

GitHub on web-sovellus, jonne voidaan tallentaa Git-versiohallintaohjelmistoa käyttävät sovellusprojektit. Se on tarkoitettu sovellusprojektien versionhallintaan. Git:n käyttö yleensä toimii komentorivisovelluksilla, kuten CMD, GitBash tai Cmder. GitHub tarjoaa graafisen työpöytäversion Gitin hallintaan, mikä helpottaa varsinkin aloittelevaa sovelluskehittäjää ymmärtämään paremmin Git-versiohallintaa. GitHub tarjoaa myös muun muassa käyttäjien projekteihin bugi-seurantaa, tiimi-, ja tehtävienhallintaa.
(GitHub 2017a.)

GitHub kehitettiin vuonna 2008 ja sovellus on kirjoitettu Ruby ohjelmointikielellä. Sillä on nykypäivänä noin 26 miljoonaa käyttäjää. GitHub tilin saa ilmaiseksi, jos haluaa, että projekti näkyy muille käyttäjille julkisena. Jos yksittäinen kehittäjä ei halua projektinsa näkyvän julkisena, pitää maksaa 9$/kk. Yksityinen tiimin projektin hinta on 9$/kk per henkilö. Yritykset 21$/kk per käyttäjä. Muita GitHubin tapaisia versiohallinta ohjelmistoja on muun muassa Bitbucket, GitLab ja Coding. (GitHub 2017b.)

2.4 ReactJS

ReacJS on JavaScript kirjasto, jota käytetään dynaamisten web-käyttöliittymien luomiseen. Toimii erityisen hyvin, kun halutaan tehdä SPA projekteja, eli yhden sivun sovelluksia. Reactin kehitti Facebookin sovelluskehittäjä Jordan Walken. Reactia käytettiin ensimmäisen kerran vuonna 2011 Facebookin uutisvirrassa ja vuonna 2012 Instagram sovelluksessa. Vuonna 2013 se oli saatavilla avoimena lähdekoodina. React Native kirjasto, joka mahdollistaa Android, UWP ja iOS kehityksen natiivina ilmestyi vuonna 2015. Reactia käyttää useat tunnetut sovellukset, kuten Dropbox, Facebook, Netflix, Reddit ja Airbnb. (Techmagic 2017.)

Reactia käytetään käyttöliittymän luomiseen ja hallitsemiseen, ei siis esimerkiksi serveripuolen koodin kehittämiseen. Se ei ole, tietoinen mistä data tulee, mutta se osaa näyttää sen. Reactin vahvuudet ovat yksinkertaisuus, helppokäyttöisyys, nopeus ja skaalautuvuus. (React 2017a.)

2.4.1 Virtual DOM

Reactin käyttää Virtual DOM:n menetelmää, mikä tekee sillä tehdyistä sovelluksista tehokkaita. Ilman virtual DOM menetelmää, DOM manipuloidaan aina kokonaan jokaisella kerralla, kun DOM:iin tulee muutos. Vaikka vain yksi DOM elementti olisi muuttunut. Tämä tekee sovelluksista raskaita, varsinkin jos niiden on tarkoitus olla yhden sivun web-sovelluksia(SPA). Virtual DOM nopeuttaa sivujen latausta huomattavasti, koska se löytää DOM elementin johon muutos on tullut ja manipuloi ainoastaan sen, ei siis päivitä koko DOMia. (Codeacademy 2017.)

2.4.2 Komponentit

React on komponenttipohjainen, mikä tarkoittaa sitä, että komponenttien avulla voit jakaa käyttöliittymän eri osat itsenäisiksi kokonaisuuksiksi. Käyttöliittymä koostuu siis useasta eri komponentista. Komponentit on tarkoitus tehdä itsenäisiksi, jotta niitä voi käyttää useassa eripaikassa, esimerkiksi sama footer-komponentti voidaan asettaa web-sovelluksen jokaiselle sivulle. Itse komponentit koostuvat yleensä, javascript funktioista, html elementeistä ja muista React komponenteista. Tyypillisesti React sovelluksissa käytetään yhtä pääkomponenttia, jossa pidetään kirjaa sovelluksen tilasta. Myös muissa komponenteissa voidaan ylläpitää tiloja. (React 2017b.)

Reactissa komponentit ovat yleensä kirjoitettu JSX:llä, mikä laajentaa JavaScript syntakseja. Sen avulla komponentteihin voidaan kirjoittaa HTML syntakseja. Reactissa ominaisuuksia(property) siirretään komponenttien välillä props(property) syntaksilla.

React komponentin luonti voidaan tehdä tavallisella JavaScript funktiolla mikä ottaa yksinkertaisen ’props’ objektin vastaan. Näitä komponentteja kutsutaan funktionaalisiksi komponenteiksi. Kuvassa 4 on esimerkki funktionaalisesta komponentista, missä käytetään myös JSX:n tarjoamia HTML syntakseja.

```
function uusiAsiakas(props) {
  return <h1>Nimi:, {props.nimi}</h1>;
}
```

Kuva 4. Funktionaalinen komponentti

Useasti kuitenkin komponentit luodaan käyttämällä ES6 luokkaa. Render funktiolla piirretään komponentti näkyviin käyttöliittymässä. Kuvassa 5 näkyy ES6 luokkalla tehty komponentti.

```
class UusiAsiakas extends React.Component {
  render() {
    return <h1>Nimi:, {this.props.name}</h1>;
  }
}
```

Kuva 5. ES6 luokalla tehty komponentti

Komponentteja voidaan asettaa sisäkkäin. Esimerkiksi pääkomponentti koostuu monista muista sovelluksen komponenteista. Komponentti luodaan pääkomponentin sisälle tässä esimerkissä käyttämällä tagia <UusiAsiakas />.

```
function PaaKomponentti() {
  return (
    <div>
         <UusiAsiakas />
         <UusiAsiakas />.
    </div>
  );
}
```

Kuva 6. Pääkomponentti.

2.4.3 Komponentin Tila

React komponentilla voi olla oma State, eli oma tilansa. Komponentin alustusvaiheessa komponentti alustetaan määritellyillä tilatiedoilla. Tilaa voidaan muuttaa komponentissa haluamalla tavalla. Muutos saadaan tehtyä setState() funktiolla. Komponentin tilaa voidaan muuttaa ainoastaan komponentin sisällä. (React 2017c.)

Esimerkiksi jos on nettisivu, missä pitää painaa painiketta, jotta laskuriin saadaan lisättyä arvoa, niin tässä on kyseessä komponentin tilanmuutos. Kuva 7 havainnollistaa esimerkin alkutilasta ja tilanmuutoksesta kooditasolla. Kuva 8 havainnollistaa esimerkin tilanmuutoksesta käyttöliittymätasolla.

```
class Button extends React.Component {
  constructor() {
    super();
    this.state = {
      count: 0,
    };
  }

  updateCount() {
    this.setState((prevState, props) => {
      return { count: prevState.count + 1 }
    });
  }

  render() {
    return (<button
              onClick={() => this.updateCount()}
            >
              Clicked {this.state.count} times
            </button>);
  }
}
```

Kuva 7. Komponentin tilanmuutos kooditasolla.

Kuva 8. Komponentin tilanmuutos käyttöliittymätasolla.

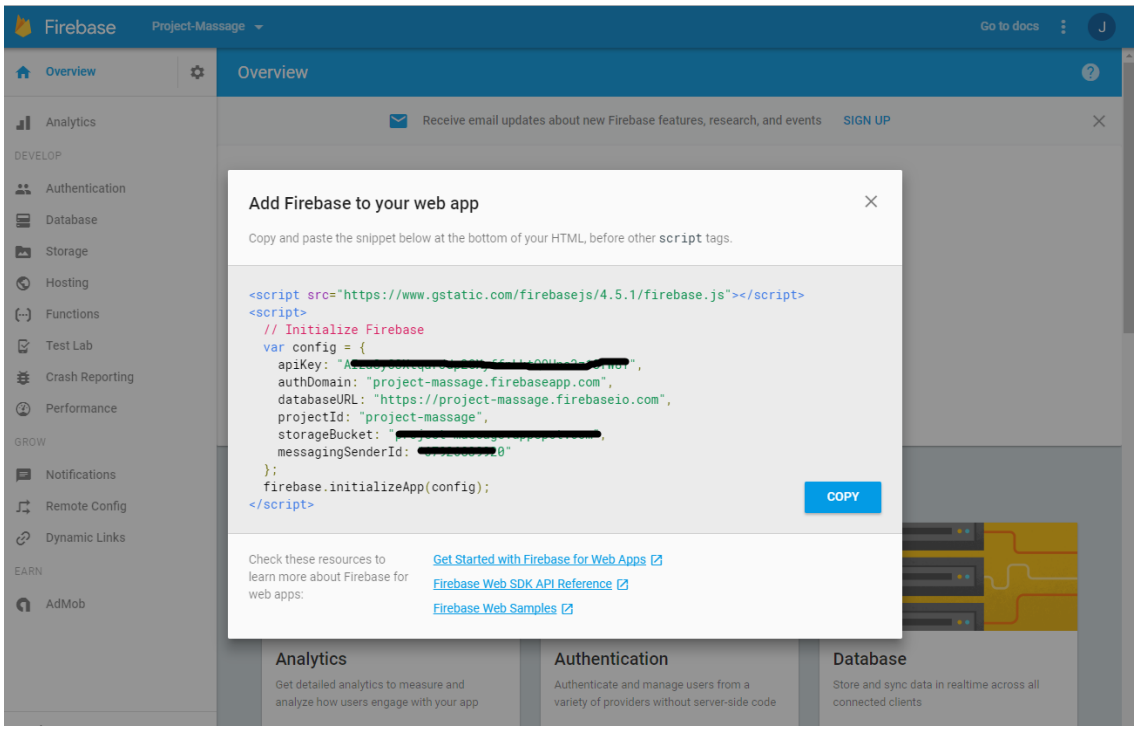
2.5 Firebase

Firebase on mobiili ja web sovelluksille tehty kehitysalusta, joka tarjoaa käyttäjän sovellukseen reaaliaikaisen tietokannan ja sen hallintajärjestelmän.
Se perustettiin vuonna 2011, Firebase siirtyi Googlen omistukseen vuonna 2014. Harrastusmielessä sovellusta pystyy käyttämään ilmaiseksi. Yritysmielessä sovellus maksaa vähintään 25$/kk, riippuen tarpeista. Firebase toimii React sovellusten kanssa erinomaisesti yhteen. (Crunchbase 2017.) (Firebase 2017a.)

Firebasessa käyttäjä tallentaa dataa NoSQL pilvi tietokantaan. Data on synkronisoitu kaikkien sen käyttäjien kanssa reaaliaikaisesta. Data pysyy saatavilla, vaikka sovellus menisi offline-tilaan. Data tallentuu JSON muodossa kantaan. (Firebase 2017b.)

Käyttäjällä on oma konsoli, mistä omaa sovellusta pystyy hallitsemaan. Konsolin kautta pystytään muun muassa vaikuttamaan käyttäjähallintaan, tietokantoihin, käyttäjätilastoihin, laitetestaukseen ja suorituskyky asioihin.

Oman sovelluksen saa integroitua Firebaseen hyvin helposti. Firebase sovellus antaa konfigurointiin koodinpätkä. Konfigurointikoodi liitetään omaan HTML koodin loppuun ennen muita script tageja, ja näin integraatio on valmis. Kuvassa 9 näkyy koodin pätkä, mikä liitetään käyttäjän omiin JavaScript tiedostoon, jotta integraatio saadaan toteutettua. Kuvan taustalla näkyy myös Firebase konsoli. (Firebase 2017c.)



Kuva 9. Konfigurointikoodi

3 Web-sovelluksen suunnittelu ja kehittäminen

Tämän osion tarkoitus on antaa kokonaiskuva kehitysprosessista, käymällä läpi vaihe vaiheelta, miten web-sovellus luotiin alusta loppuun. Jokaisessa vaiheissa käydään läpi haaste, haasteen ratkaisu, teknologiavalinnat, mahdolliset ongelmat, sekä vaiheen lopputulos.

3.1 Suunnittelun alkuvaihe

Tarkoitus oli luoda hierojalle web-sovellus, jolla pystytään varaamaan hieronta-aikoja ja josta löytyy sisältönä hierojan esittely, hoitojen hinnasto, sekä yhteystiedot. Käyttöliittymästä haluttiin modernin näköinen, mahdollisimman selkeä, nopea ja sulava käyttäjälleen.

Suunnittelu aloitettiin useiden muiden hieronta sivustojen, sekä modernien käyttöliittymien omaavien sivustojen tarkkailulla. Hieronta sivustoja en lähde erittelemään, mutta kaksi erittäin hyvän käyttöliittymä innovointiin tarkoitetun sivua nostan esiin, dribbble.com ja thebestdesigns.com. Dribble sivustolla pääsee katsomaan käyttöliittymä ammattilaisten ja harrastajien uusimpia innovointeja. The best designs sivusto arvostelee ja kerää netin hienoimpia sivustoja yhteen. Kyseisiltä sivuilta otettiin vinkkejä tämän projektin ulkoasuun liittyen. Muilta hierontasivustoilta tutkittiin lähinnä itse sisältöä. (Dribbble 2017, Best Designs 2017)

Sovelluksen väreiksi valittiin pääväri valkoinen, tumman ja vaalean harmaat. Tehosteväriksi oranssi. Nämä värit valittiin koska värit tekevät käyttöliittymästä pirteän, selkeästi luettava.

Muiden sivustojen tutkimisen jälkeen web-sivusta tein rautalankamalli, jonka avulla saatiin parempi kokonaiskuva projektista. Rautalankamallin selvensi sivuston rakenteen, mihin tulee logo, navigointi, nappulat, eri sisällöt, sekä ajanvaraus prosessin eri vaiheet.

3.2 Sketch ja käyttökokemus

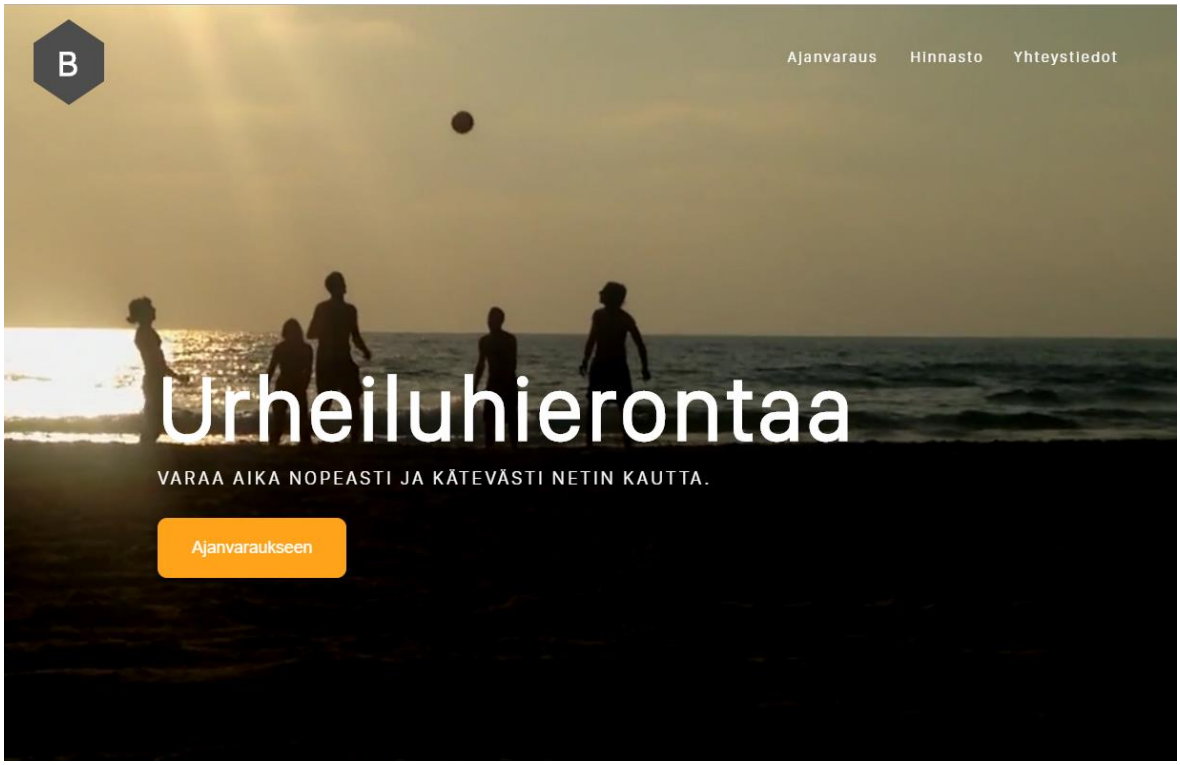
Tässä vaiheessa oli selvää mitä halutaan ja millainen rakenna sovellukselle tulee. Seuraavaksi haluttiin tehdä sovelluksesta kuvat käyttöliittymästä ja miettiä samalla käyttäjän käyttökokemusta.

Käyttöliittymän suunnitteluun valittiin teknologiaksi Sketch. Valinta oli helppo tehdä, koska minulta löytyy aikaisempaa kokemusta sen käytöstä ja olen todennut sen erittäin päteväksi käyttöliittymä suunnitteluun. Sketchin ehdottomia vahvuuksiin kuuluu helppokäyttöisyys, nopeat importit haluttuihin muotoihin, piirtoalueiden laitekohtaiset koot, Insivion kanssa yhteensopiva, sekä se että se on vektorigrafiikka ohjelmisto.

Sketchillä piirrettiin jokainen sovelluksen sivu, vaihe ja tapahtuma kuvana, jotta saadaan mahdollisimman selkeä kuva sovelluksesta. Tein myös kuvat mobiili ja desktop näkymästä. Kuvat piirrettiin oikeisiin mittasuhteisiin. Näiden kuvien avulla voitiin suunnittelun jatkovaiheessa rakentaa Invisionilla prototyyppi. Kuviin määritettiin myös kaikki yksityiskohdat, eli mikä fonttiperhe valitaan, millä fontti koolla tietyt otsikot ja leipätekstit, mitkä elementit saavat mitkäkin värit, mihin tulee mitkäkin kuvat.

Tässä kohtaan, kun kuvia piirrettiin, mietittiin myös tarkemmin sovelluksen käyttökokemusta. Hyvän kokemuksen takaamiseksi otettiin huomioon asetetut tavoitteet eli, helppokäyttöisyys, selkeys ja käyttäjäkokemus.

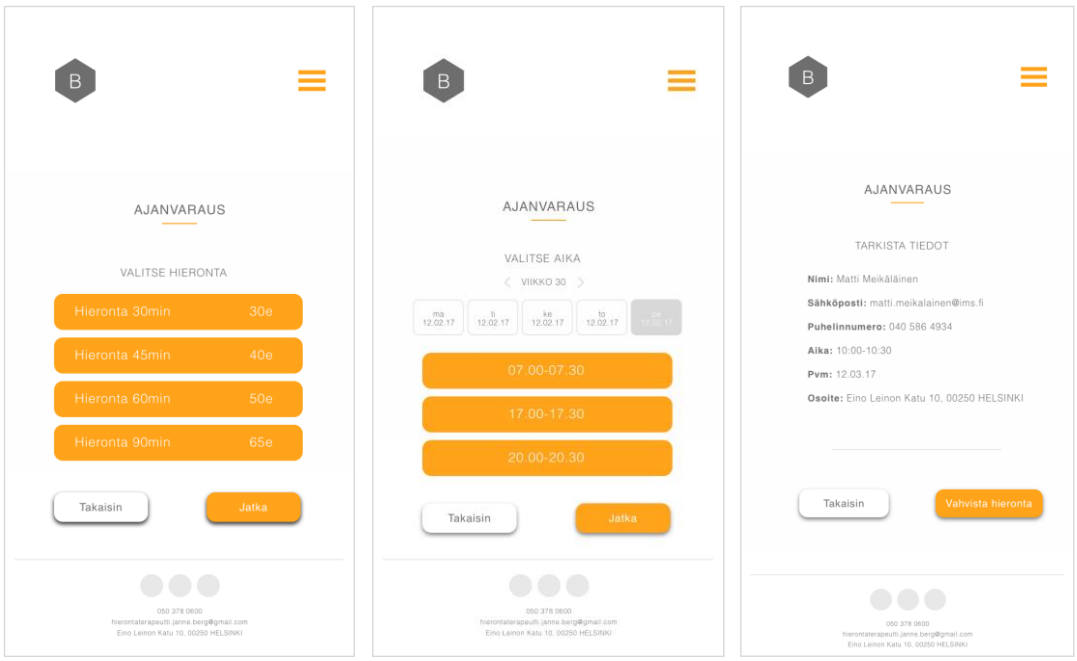
Helppokäyttöisyyteen panostettiin laittamalla kaikki oletetut linkit, tiedot, nappulat oikeisiin paikkoihin. Esimerkiksi oletettiin, että yleisin käyttötapaus mitä sivulla käyttäjä tulee tekemään, on hierontahoitojen varaus. Tämän oletuksen mukaan ajanvaraukseen vievä nappula asetettiin heti näkymään käyttäjälle, kun hän sivulle saapuu. Kuva 10 havainnollistaa tämän.



Kuva 10. Etusivunäkymä

Helppokäyttöisyyteen panostettiin myös määrittelemällä sivustolla tarvittavien klikkauksien määrä minimiin, jotta käyttäjän työmäärä olisi mahdollisimman vähäinen. Esimerkiksi ajanvarauksen läpivientiin tarvitaan 5-7 klikkausta, kun muilla hierontasivustoilla noin 7-10 klikkausta.

Selkeyteen panostettiin laittamalla sovelluksen värimaailma silmälle sopivaksi, jotta käyttäjä näkee, välittömästi miten edetä sivulla. Esimerkiksi ajanvarauksen näkymiin laitettiin valkoinen tausta ja mistä selkeästi erottuu oranssit nappulat, mihin halutaan käyttäjän keskittyvän. Selkeyteen panostettiin myös siinä mielessä, että kaikki ylimääräinen otettiin pois, eli ei mitään ylimääräisiä kuvia tai tekstejä. Mahdollisimman vähän häiriötekijöitä käyttäjän silmille. Etusivulla on kuvia, mutta esimerkiksi ajanvarauksessa ei, koska käyttäjää ei siinä kohtaan kiinnosta mikään muu kuin hoidon varaus.



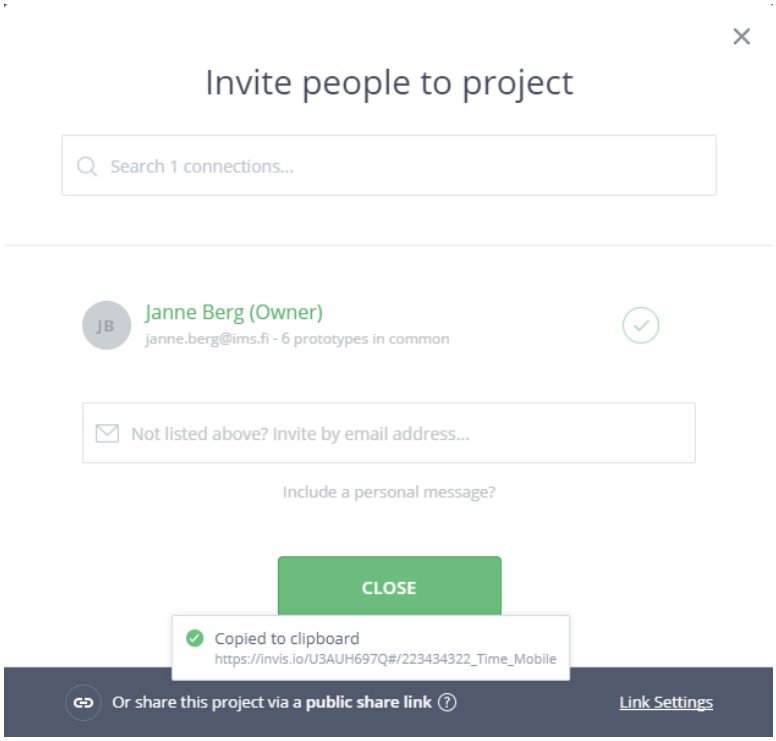
Kuva 11. Suunnittelukuvia ajanvarauksen vaiheista

3.3 Invision, prototyypin luomiseen

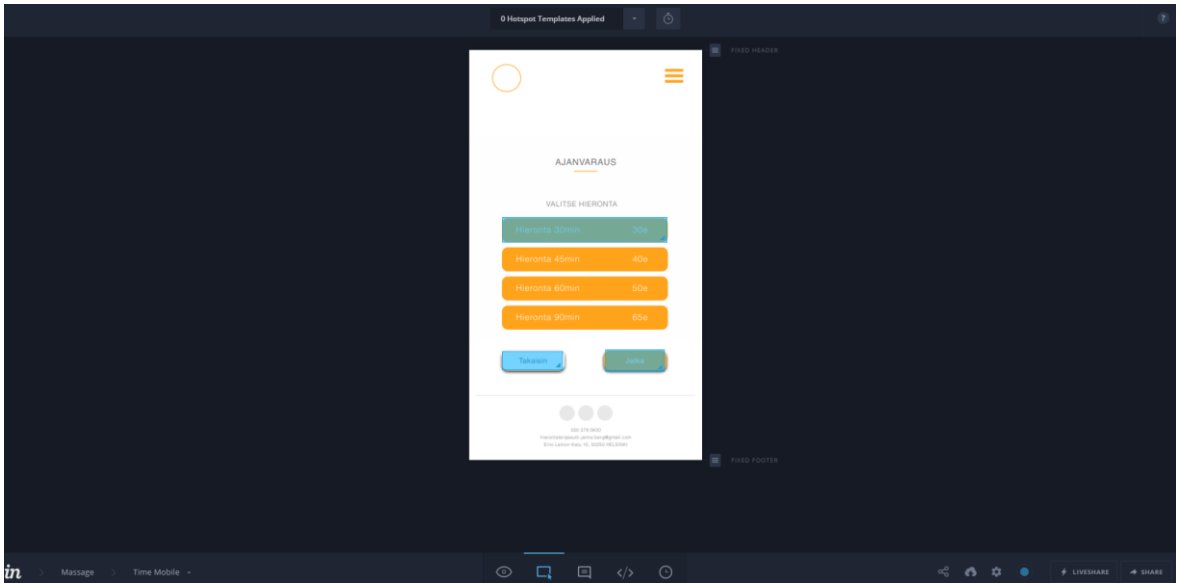
Sovelluksen suunnittelu oli ulkoasun ja vaiheiden osalta saatu kuvattua Sketchillä. Halusin testata sovelluksen monivaiheisen ajanvarauksen, tekemällä siitä prototyypin Sketch kuvilla. Testaukseen oli tarkoitus ottaa muutama käyttäjä testaamaan prototyyppiä.

Prototyypin tekoon valittiin Invision, koska se on yhteensopiva Sketchin kanssa ja sen käyttötarkoitus on kuvaprototyyppien luominen. Lisäksi minulta löytyy jo aikaisempaa kokemusta Invisionin käytöstä. Olen todennut sen erittäin hyväksi prototyyppien tekoon. Helppokäyttöinen ja nopea jakaa testattavien kanssa. Huomioon otettiin myös Invisionin käytön suosio huippuyrityksien keskuudessa.

Invisionilla sain tehtyä onnistuneen prototyypin sovelluksesta. Kaikki halutut toiminnot lisättiin kuviin, esimerkiksi siirtymiset, animaatiot ja klikkaukset. Prototyyppi testautettiin viidellä käyttäjällä. Testaajan tehtävänä oli varata hieronta-aika itselleen. Testit menivät hyvin ja palaute oli hyvää. Prototyyppi jaettiin Invisionin tarjoamalla linkillä, suoraan testaajien mobiililaitteisiin. Linkin voi aukaista ilman Invisio-tiliä. Kuva 12 havainnollistaa linkin jakamisen. Kuvassa 13 Invisionin prototyypin luomisnäkymä.



Kuva 12. Testaus-linkki



Kuva 13. Luomisnäkymä

Testien avulla tiesin, että varsinaisen koodauksen voi aloittaa, mitään suurempia muutoksia ei sovellukseen enää tulisi. Suunnitteluvaiheessa mietittiin kaikki valmiiksi, niin sovelluksen koodaus oli helppo ja selkeää aloittaa.

3.4 Koodausvaiheen teknologiavalinnat

Teknologia valintoja tehdessä. Tavoite oli oppia jokin moderni ja työmarkkinoilla arvokas ohjelmointikieli, mikä olisi myös mahdollisimman sopiva tämän sovelluksen tekemiseen. Tiesin että tämä web-sovellus, mikä olisi tarkoitus tehdä, luodaan pääasiassa vain front-end koodilla. Toinen tavoite oli päättää millä ja miten tehdään tietokannat ajanvarauksille. Tietokannoista ja niiden luomisesta minulla oli todella vähän kokemusta. Tarve oli saada reaaliaikainen tietokanta, jotta ajanvaraukset päivittyvät heti sivulle, kun asiakas on ajan varannut.

Päädyin valitsemaan ohjelmointikieleksi React.js:n, koska huomasin, että työmarkkinoilla react-osaajalla on kysyntää. Valintaa helpotti myös se seikka, että Reactia käytetään pääasiassa front-end koodin tekemiseen. Minulla ei ole aikaisempaa kokemusta Reactin käytöstä eikä muutenkaan yksisivuisten sovellusten tekemisestä. Tämä valinta siis opettaisi minulle uuden ohjelmointikielen ja modernin tavan luoda sovelluksia. Valinta oli helppo tehdä.

React oli valittu ohjelmointikieleksi, se tarkoitti sitä, että node.js valittiin samalla hallinnoimaan projektin back-endiä. Node.js:llä onnistuu helposti JavaScript-pakettien hallinta ja luominen.

Sovelluksen tietokantojen luomiseen ja hallitsemiseen valitsin Firebasen. Firebase oli helppo valinta, koska se vastasi tarpeisiini, reaaliaikainen ja nopea. Firebasen graafinen käyttäjäkonsoli on selkeä ja helppo hallita. Siltä löytyi myös selkeät ohjeet ja tutoriaalit sivuilta, joten se oli helppo oppia nopeasti. Firebasesta minulla ei ollut aikaisempaa kokemusta, eikä muutenkaan hirveästi kokemusta tietokantojen luomisesta tai hallitsemisesta.

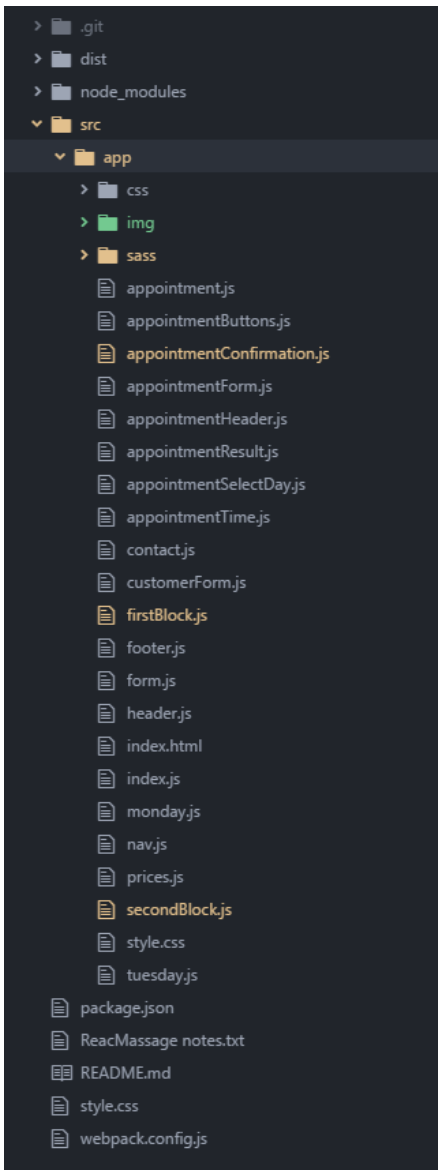
Sovellusprojektit tarvitsevat aina omat kehitysympäristönsä. Tiesin, että tähän projektiin ei tulla tarvitsemaan välttämättä mitään ihmeempiä ympäristöjä. Vaatimuksena oli Node.js tuki, paketinhallinta, automaattinen täydennys, tuttu ohjelmisto ja useiden tiedostojen yhtäaikainen muokkaaminen. Päädyin valitsemaan Atom-editorin, joka vastaa kaikkiin vaatimuksiini tämän projektin kehitysympäristöstä.

Versiohallintaan halusin tutun, hyväksi todetun ja helppokäyttöisen ohjelmiston. Valitsin GitHubin mitä olen käyttänyt useimmissa projekteissani.

3.5 Projektin luominen

Projektin luominen ja ympäristön pystytys, vaati monta eri vaihetta ennen, kuin se on toimintakelpoinen, varsinkin kun on kyse React projektista. Vaiheet olivat seuraavanlaiset, GitHubista luotiin uusi projekti, projektin tiedostojen tekeminen, npm asentaminen, jolla asennetaan kaikki tarvittavat paketit projektiin.

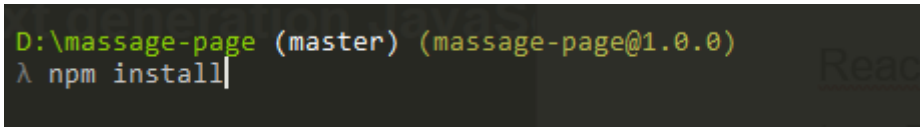
Ensimmäiseksi luotiin projektin repository GitHubiin. Eli luotiin versionhallinta paikka projektille. Tämän jälkeen tein tiedostorakenteen repositoryyn, jonka jälkeen kloonasin projektin omalle koneelleni GitHubista. Kuva 14 näyttää projektin loppuvaiheen tiedostorakenteen.



Kuva 14. Tiedostorakenne

React-koodia kirjoitettiin tässä projektissa JSX ja ES6 kielillä. Ongelmana on se, että selaimet eivät kuitenkaan ymmärrä näitä kieliä. Jotta selaimet saadaan ymmärtämään näitä, tarvitaan tulkiksi Babel. Se on JavaScript koodin kääntäjä. Babel pystyy kääntämään JSX ja ES6 kielet Vanilla JavaScriptiksi jota selaimet ymmärtävät.

Reactin ja Babel:n ja muiden pakettien asennukseen käytin Node.js:n npm paketin hallintaa. Ensimmäinen vaihe pakettien hallintaan oli asentaa itse npm, mikä asennettiin komentoriviltä kuvan 15 komennolla.

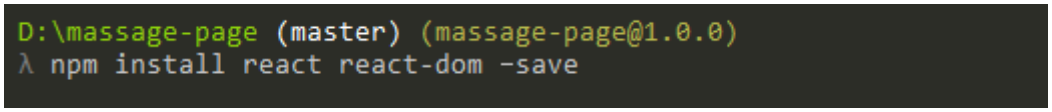


Kuva 15. Npm:n asennuskomento

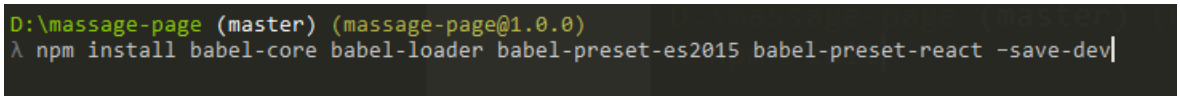
Toinen vaihe pakettien asennukseen oli se että piti luoda packake.json tiedosto mikä seuraa kaikkia asentamiamme riippuvuuksia, kuten esimerkiksi Babel ja React. Tiedosto luotiin antamalla komento npm init.

Kuva 16. Komento npm init.

Kolmas vaihe oli asentaa projektiin React, kirjoittamalla kuvan 17 komento. Tämän jälkeen asennettiin babel ja kaikki sen toimintaan tarvittavat paketit, kirjoittamalla kuvan 18 komento.



Kuva 17. Reactin asennuskomento



Kuva 18. Babel pakettienasennuskomento

Reactin ja Babelin asennuksen jälkeen asensin projektiin Webpackin. Webpack lyhyesti on moduulin kasaaja, mikä tarkoittaa sitä, että webpack prosessoi sovelluksen läpi ja kasaa tarvittavat moduulit mitä sovellus käyttää ja tekee siitä yhden tiedoston selaimelle näytettäväksi. Jos projekti on isompi, voi kasattuja tiedostoja olla enemmänkin kuin yksi. Omassa projektissani luotiin vain yksi tiedosto, bundle.js. Webpack asennettiin kuvan 19 komennolla.

```
D:\massage-page (master) (massage-page@1.0.0)
λ npm install webpack webpack-dev-server -save-dev
```

Kuva 19. Webpackin asennuskomento

```
{
  "name": "massage-page",
  "version": "1.0.0",
  "description": "massage site",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "npm run build",
    "build": "webpack -d && webpack-dev-server --content-base src/ --inline --hot --port 1234 --history-api-fallback"
  },
  "author": "me",
  "license": "ISC",
  "dependencies": {
    "firebase": "^3.9.0",
    "moment": "^2.18.1",
    "react": "^15.4.2",
    "react-dom": "^15.4.2",
    "react-router": "^3.0.2"
  },
  "devDependencies": {
    "babel-core": "^6.23.1",
    "babel-loader": "^6.3.2",
    "babel-preset-es2015": "^6.22.0",
    "babel-preset-react": "^6.23.0",
    "webpack": "^2.2.1",
    "webpack-dev-server": "^2.3.0"
  },
  "repository": {
    "type": "git",
    "url": "git+https://github.com/JalmariB/massage-page.git"
  },
```

Kuva 20 Package.json tiedostonäkymä, kun kaikki projektin paketit on asennettu.

3.6 Web-sovelluksen rakenne ja toiminta

Web-sovellus on yksisivuinen (SPA, single page application) niin kuin kaikki Reactilla tehdyt sovellukset ovat. Tähän html sivuun luodaan kaikki projektin komponentit. Pääkomponenttina toimii tässä projektissa App-komponentti joka luodaan html-sivun DOM:iin. Pääkomponentin kautta koko sovellus toimii. Sen kautta sovelluksen navigointi ja tila(state) hoidetaan.

Kuva 21 kertoo, miten tässä React-projektissa luodaan pääkomponentti html-sivuun. Koodinpätkä sijaitsee App-komponentin tiedostossa. Siinä määritellään mihin html elementtiin komponentti tehdään. Kuvassa 22 nähdään sovelluksen ainoan html sivun sisältö, sekä elementti johon App-komponentti luodaan.

```
ReactDOM.render(<App />, document.getElementById('app-container'));
```

Kuva 21. React komponentin liittäminen html sivuun.

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Urheiluhieroja | Janne Berg</title>
   <link rel="stylesheet" href="sass/style.css">
  <link rel="stylesheet" href="css/font-awesome.css">
  <link rel="stylesheet" href="css/font-awesome.min.css">
</head>
  <body class="image">
    <div id="app-container">
    </div>
    <script src="https://www.gstatic.com/firebasejs/3.9.0/firebase.js"></script>
    <script src="/app/bundle.js"></script>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.1.1/jquery.min.js"></script>
  </body>
</html>

```

Kuva 22. Sovelluksen html-sivu, index.html.

Hieronta-ajan varausprosessiin tehtiin ajanvaraukselle oma pääkomponentti, joka seuraa prosessin aikana ajanvarauksen tilaa. Tämä komponentti sisältää kaikki ajanvaraukseen tarvittavat muut komponentit. Prosessin edetessä käyttäjän tekemät valinnat, kuten hoitotyyppi, asiakastiedot, hoidon ajankohta otetaan ylös ajanvarauksen pääkomponentin tilaan. Asiakkaan vahvistaessa hieronta-ajan, pääkomponentin tilaan kerätyt tiedot lähetetään Firebaseen tehtyyn tietokantaan.

Kuvassa 23 on tämän sovelluksen ajanvarauksen pääkomponentin tila ajanvarauksen alussa. State objektin customerInfo-listaan lisätään aina dataa, kun ajanvaraus etenee, eli ajanvarauksen tila muuttuu. Kuvassa 24 on esimerkki koodista, kun tila vaihtuu asiakkaan antaessa asiakastiedot sovellukselle.

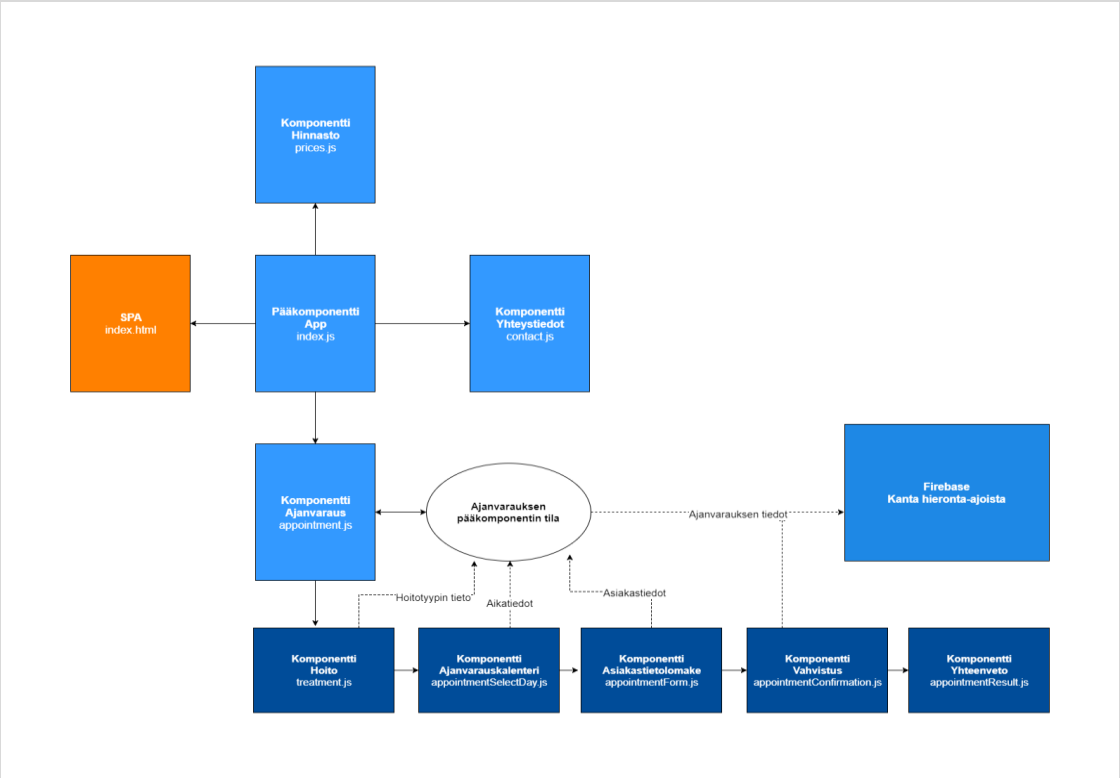
```
constructor(props) {
   super(props);
   this.state = {
   customerInfo: [],
   componentTime: true,
   componentSelectDay: false,
   appointmentForm: false,
   appointmentConfirmation: false,
   appointmentResult: false,};
 }
```

Kuva 23. Tilan alkutilanne

```
formDetails:function(name, email, phone){
  var updatedCustomerInfo = this.state.customerInfo;
  updatedCustomerInfo.push(name, email, phone);
  this.setState({
    customerInfo: updatedCustomerInfo
  })

},
```

Kuva 24. Tilan muutos, kun asiakastietoja päivitetään

Periaatteessa ajanvarauksen pääkomponentin tila oltaisiin voitu hoitaa myös ylempänä sovelluksen pääkomponentissa. Kuvassa 25 havainnollistetaan koko sovelluksen komponenttirakenne ja toiminnan vaiheet.



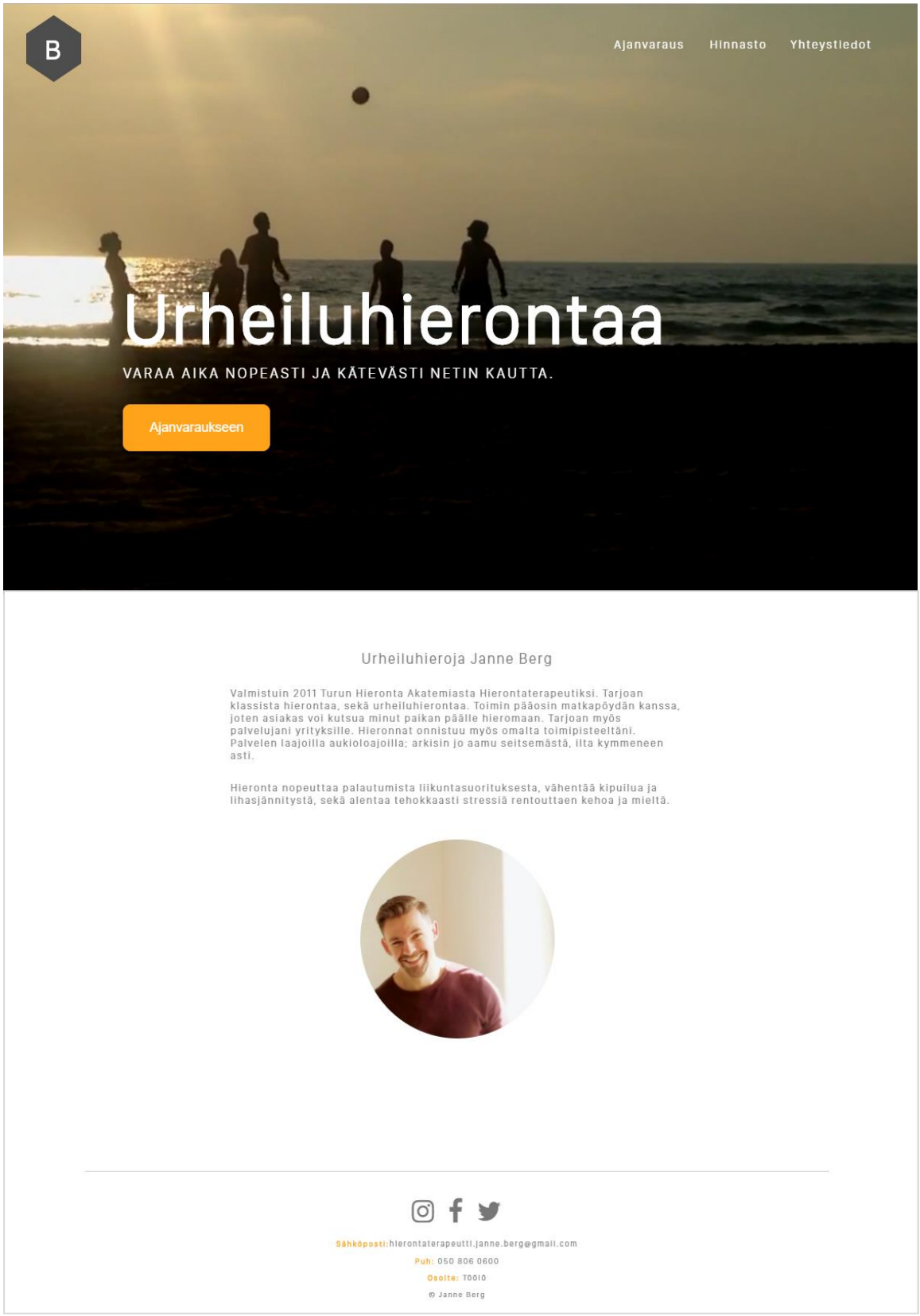
Kuva 25. Komponenttirakenne ja toimintavaiheet

3.7 Etusivunäkymä

Web-sovelluksen etusivunäkymässä pyrittiin näyttämään heti tärkeimmät asiat käyttäjännäkökulmasta. Hieronta-aikoja pääsee varaamaan kätevästi heti näkymässä olevasta ”ajanvaraukseen”-nappulasta. Ajanvaraukseen pääsee myös navigointipalkista. Etusivuun on myös laitettu esittely hierojasta kuvan kanssa, koska se koettiin tärkeäksi sisällöksi, varsinkin jos käyttäjä on uusi asiakas. Etusivulta pääsee navigoimaan ajanvarauksen lisäksi myös hinnastoon ja yhteystietoihin. Kuvassa 26 on kuvattu etusivunäkymän komponentti, joka sisältää neljä muuta komponenttia. Sovelluksen muut näkymät käyttävät samantapaista komponenttirakennetta.

```
class MainComponent extends React.Component {

  render(){
    return(
        <div>
          <NavComponent />
          <FirstBlockComponent/>
          <SecondBlockComponent/>
          <FooterComponent/>
        </div>
      );
  }
```

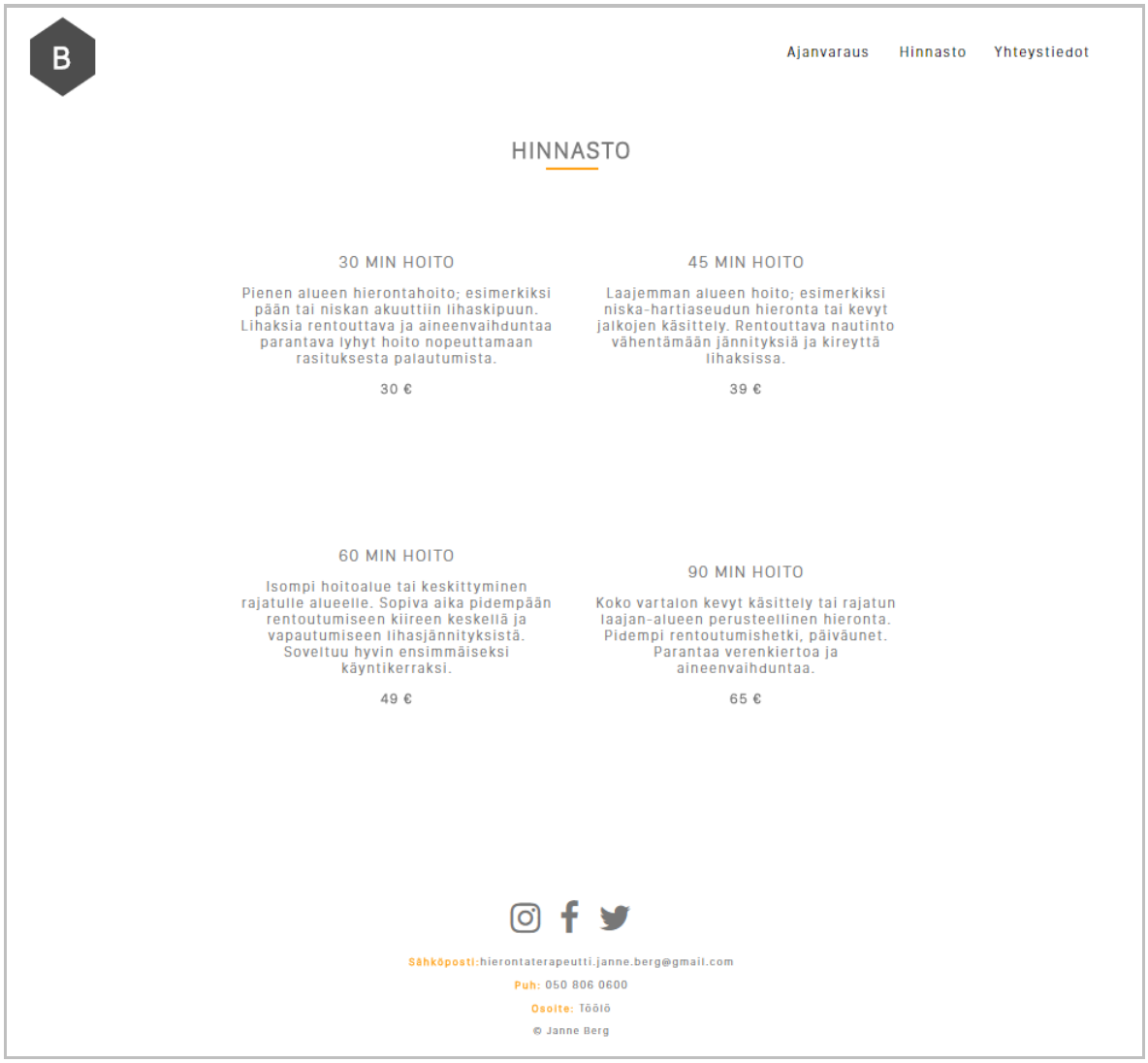
Kuva 26. Etusivunäkymän komponentti ja sen komponenttirakenne



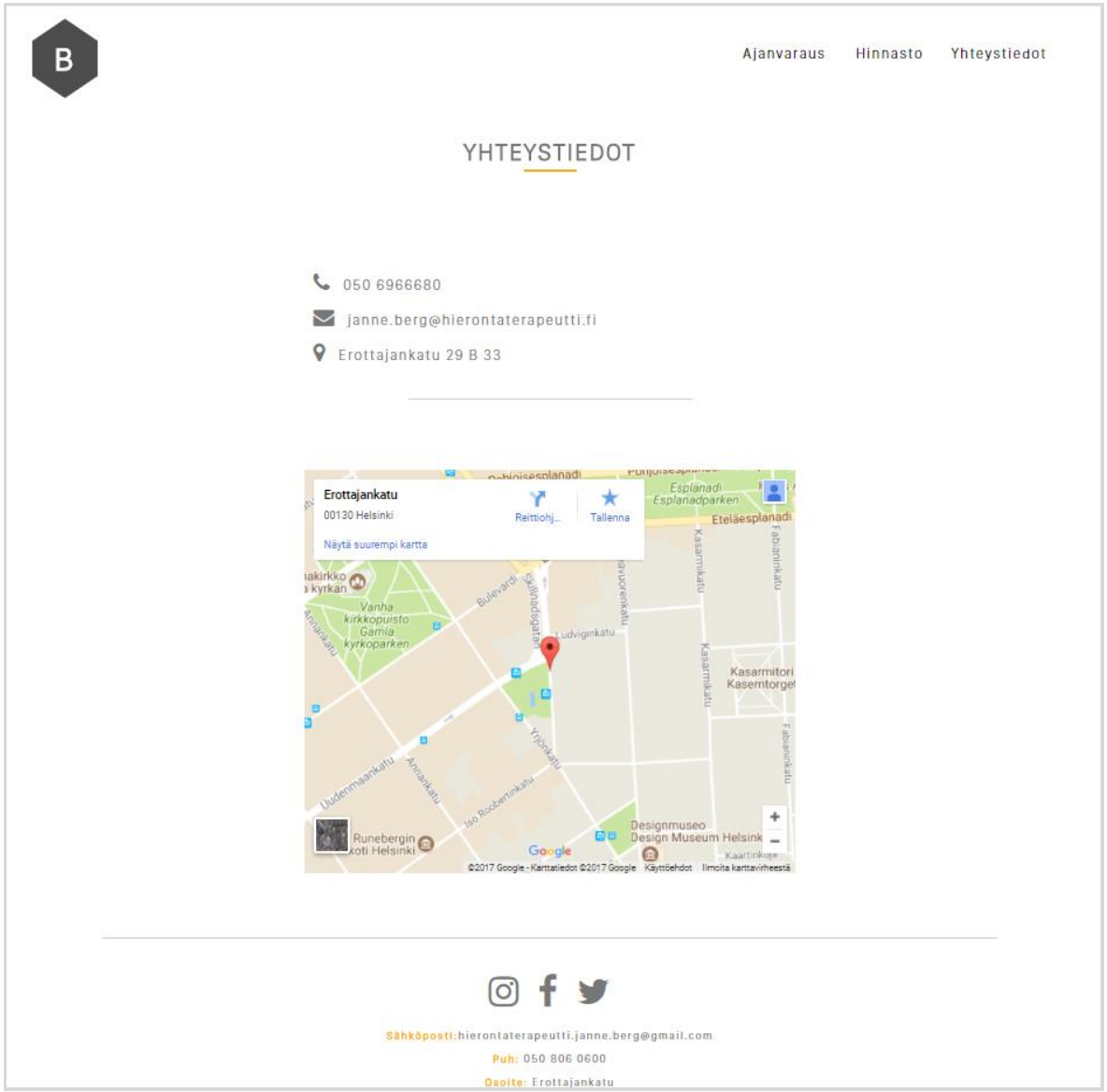
Kuva 27. Etusivunäkymä

3.8 Hinnastonäkymä ja yhteystiedot-näkymä

Hinnastonäkymässä käyttäjälle esitellään erilaisia hierontahoitoja, joista selviää hoidon tyyppi, hinta ja aika. Yhteystiedot-näkymässä käyttäjälle esitetään hierojan yhteystiedot, kuten puhelinnumero, sähköpostiosoite ja toimipaikan osoite. Toimipaikanosoite näytetään myös googlen tarjoamalla karttanäkymällä.



Kuva 28. Hinnastonäkymä

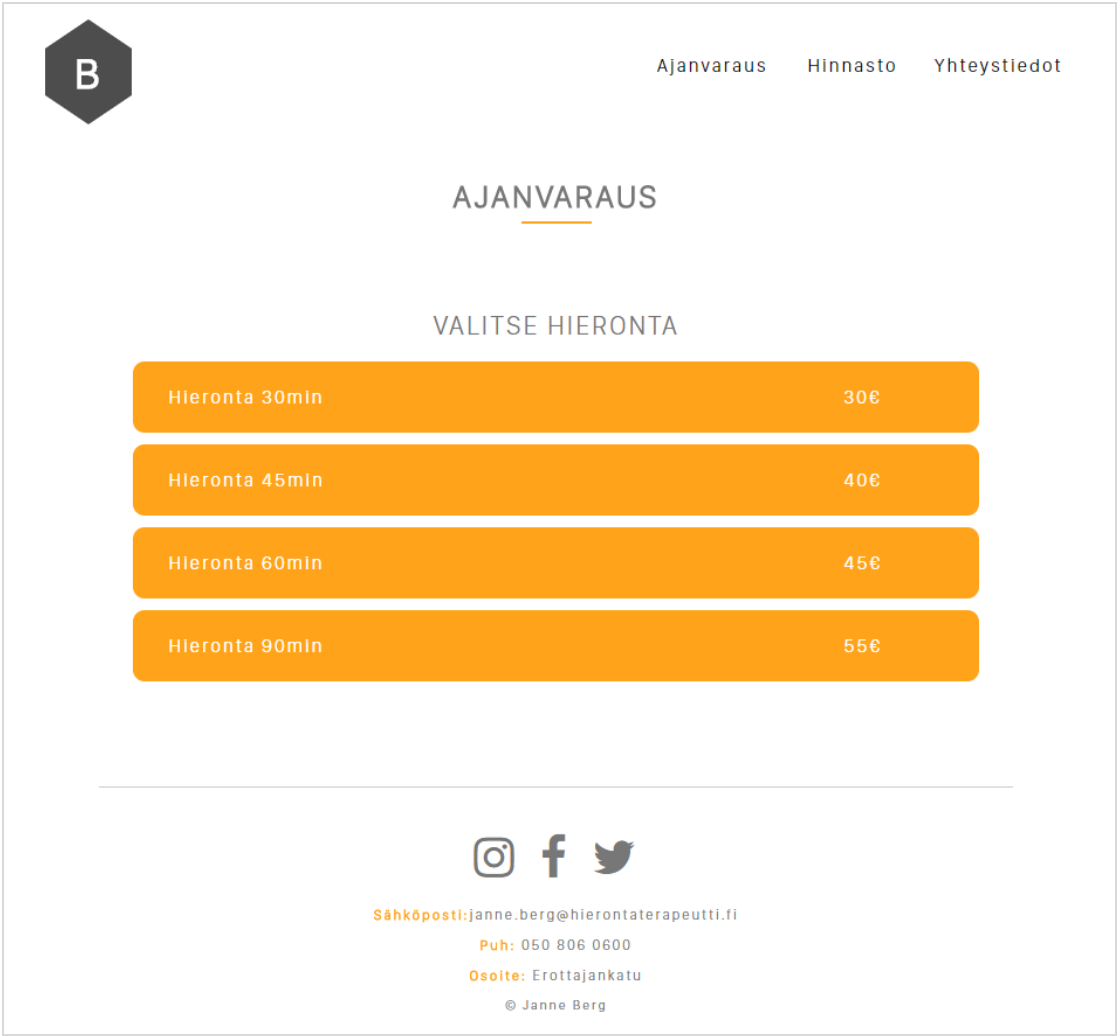


Kuva 29. Yhteystiedot-näkymä

3.9 Ajanvarausnäkymät

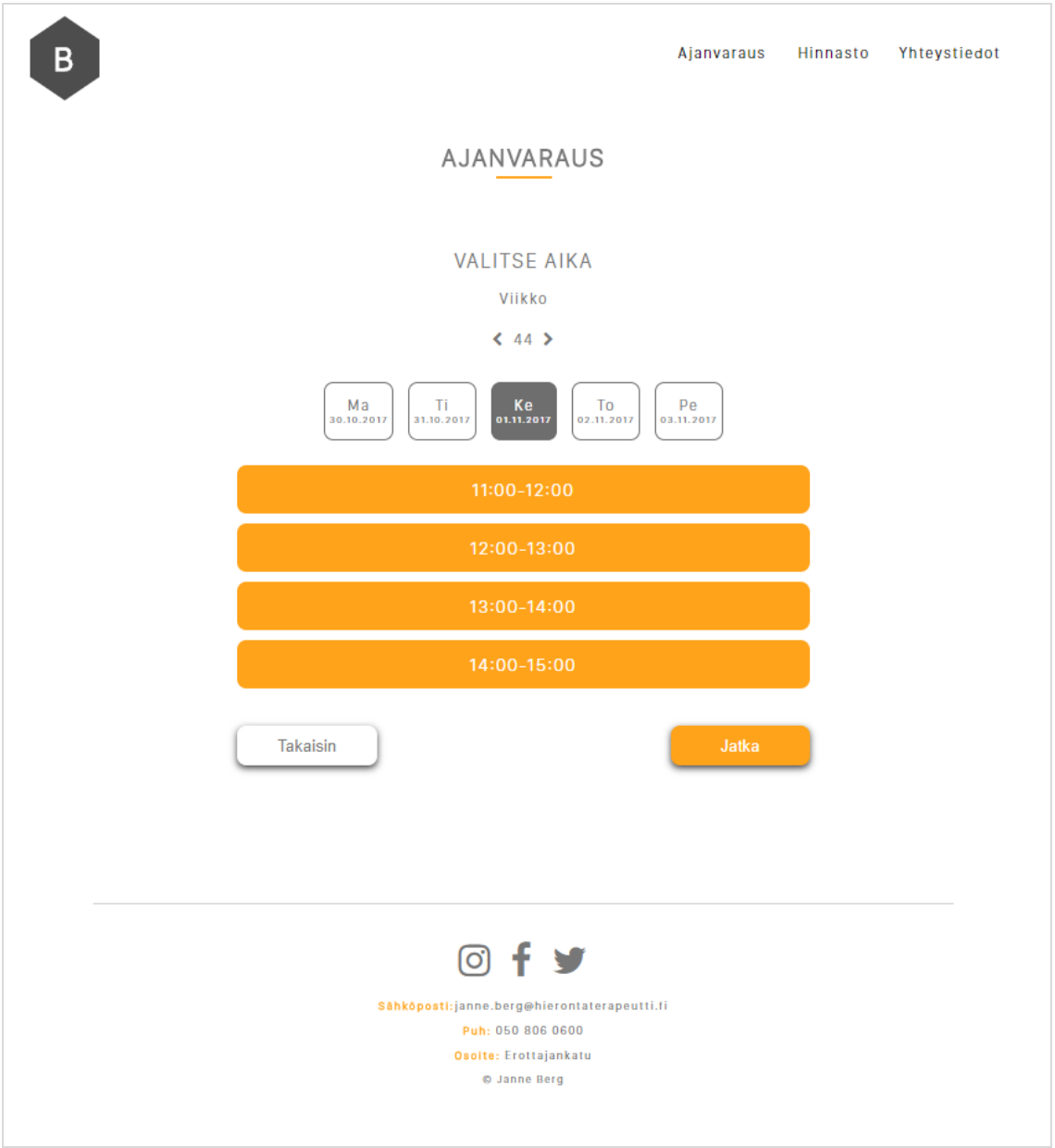
Ajanvarausnäkymät koostuvat hoidot-, ajankohta-, asiakastiedot-, vahvistus- ja yhteenvetonäkymästä. Kaikki näkymät ovat omia komponenttejaan. Jokaisessa ajanvarauksen siirtymisvaiheessa näytetään aina vaiheen seuraava komponentti ja piilotetaan edellinen komponentti. Siirtymävaihetta kontrolloidaan sovelluksen tilassa. Ajanvarauksen edetessä pääkomponentti(Ajanvarauksen) päivittää ajanvarauksen tietoja jokaisen näkymän perusteella, ottamalla tiedot omaan tilaansa. Ajanvarauksen eri näkymissä voi aina palata edelliseen näkymään klikkaamalla takaisin painiketta. Jos näin tehdään, niin ajanvarauksen tilasta poistetaan edellisen näkymän tiedot.

Hoidot-näkymässä käyttäjä näkee listan hierontahoidoista mistä hän voi valita mieleisensä hoidon hiirellä klikkaamalla. Samalla kun käyttäjä klikkaa valitsemaansa hoitoa, lähtee tieto valinnasta ajanvarauksen pääkomponentissa sijaitsevalle tila-objektille, jonne valinta tallennetaan. Samalla klikkauksella tapahtuu myös siirtyminen hoidot-näkymästä, ajankohtanäkymään.



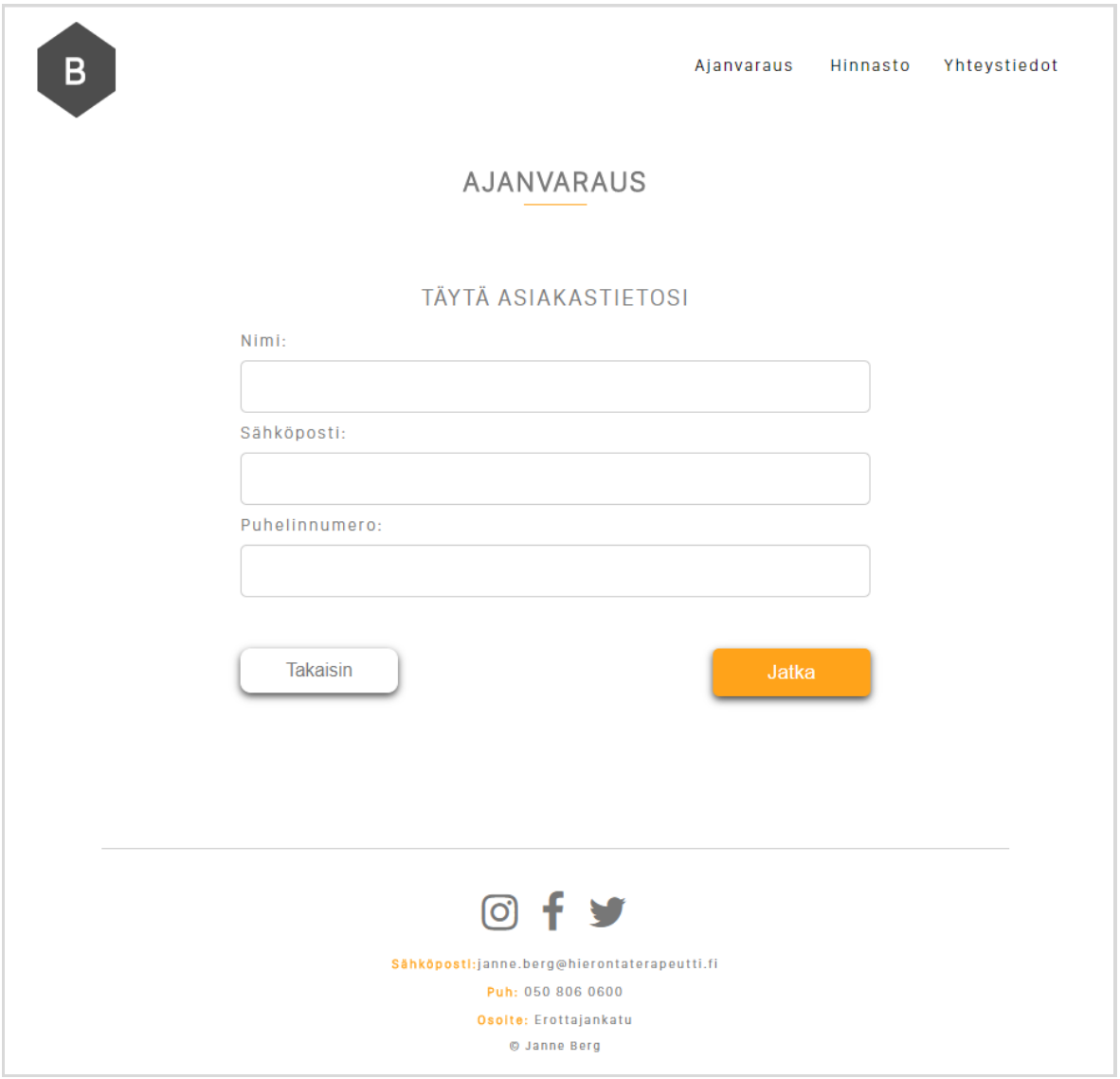
Kuva 30. Hoidot-näkymä

Ajankohta-näkymässä käyttäjä näkee viikkonumeron, kyseisen viikon arkipäivät ja valitun päivän vapaat hieronta-ajat. Käyttäjän klikatessa viikonkohdalta nuoli-ikoneja, vaihtuu viikkonumero seuraavaan tai edelliseen viikkoon, sekä samalla vaihtuu myös valitun viikon päivät näkymään. Käyttäjä ei voi selata menneitä viikkoja tai aikoja. Jotta käyttäjä saa varattua ajan, hänen pitää klikata haluamaansa päivää ja tämän valitsemansa päivän hieronta-aikaa. Hieronta-aikalistassa näkyy ainoastaan vapaana olevat ajat, ei varattuja aikoja. Käyttäjän tehdessä valinnat, lähtee jälleen valintojen tiedot ajanvarauksen pääkomponentille, jonka tila-objektiin tallentuu valinnat. Samalla myös näkymä vaihtuu asiakastiedotnäkymään.



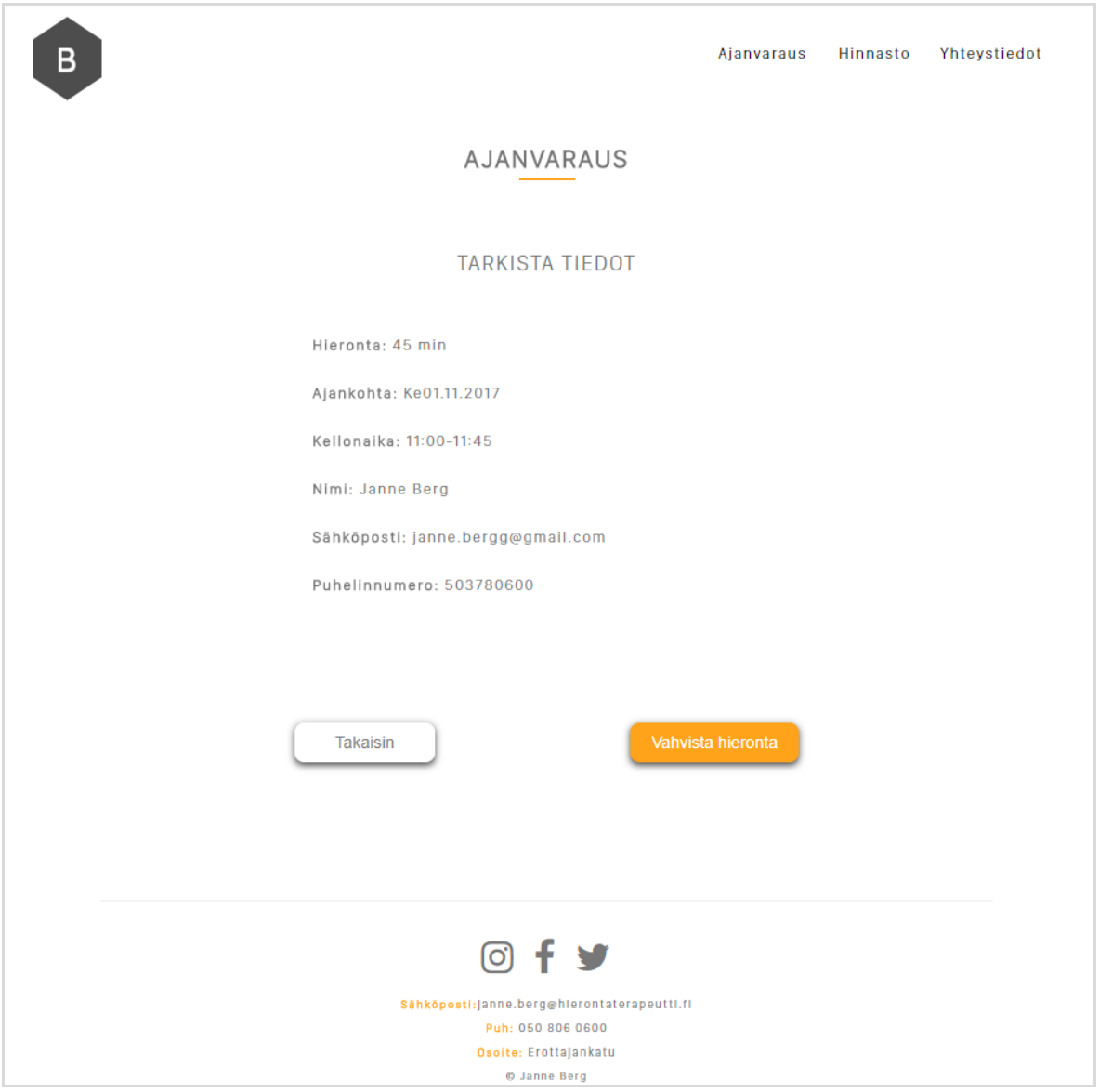
Kuva 31. Ajankohta-näkymä

Asiakastiedot-näkymässä käyttäjä näkee asiakastietolomakkeen. Lomakkeessa pyydetään käyttäjän nimi, sähköposti ja puhelinnumero. Kaikki tiedot ovat pakollisia, jotta käyttäjä pääsee jatkamaan ajanvarausta. Jatka painiketta klikkaamalla käyttäjä pääsee siirtymään ajanvarauksen seuraavaan vaiheeseen ja samalla käyttäjän tiedot otetaan jälleen pääkomponentin tilaan talteen.



Kuva 32. Asikastiedot-näkymä

Vahvistusnäkymä näyttää käyttäjän valitsemat ajanvaraustiedot, sekä pyytää käyttäjää tarkistamaan ja vahvistamaan hieronta-ajan. Jos käyttäjän mielestä tiedot täsmäävät, niin vahvista hieronta-painiketta klikkaamalla käyttäjä siirtyy ajanvarauksen viimeiseen näkymään. Samalla klikkauksella käyttäjän tiedot lähetetään ajanvarauksen pääkomponentin tilasta Firebasen tarjoamaan tietokantaan, jonne tiedot ajanvarauksesta tallennetaan. Tietojen lähetyksen havainnollistaa koodissa kuva 24.

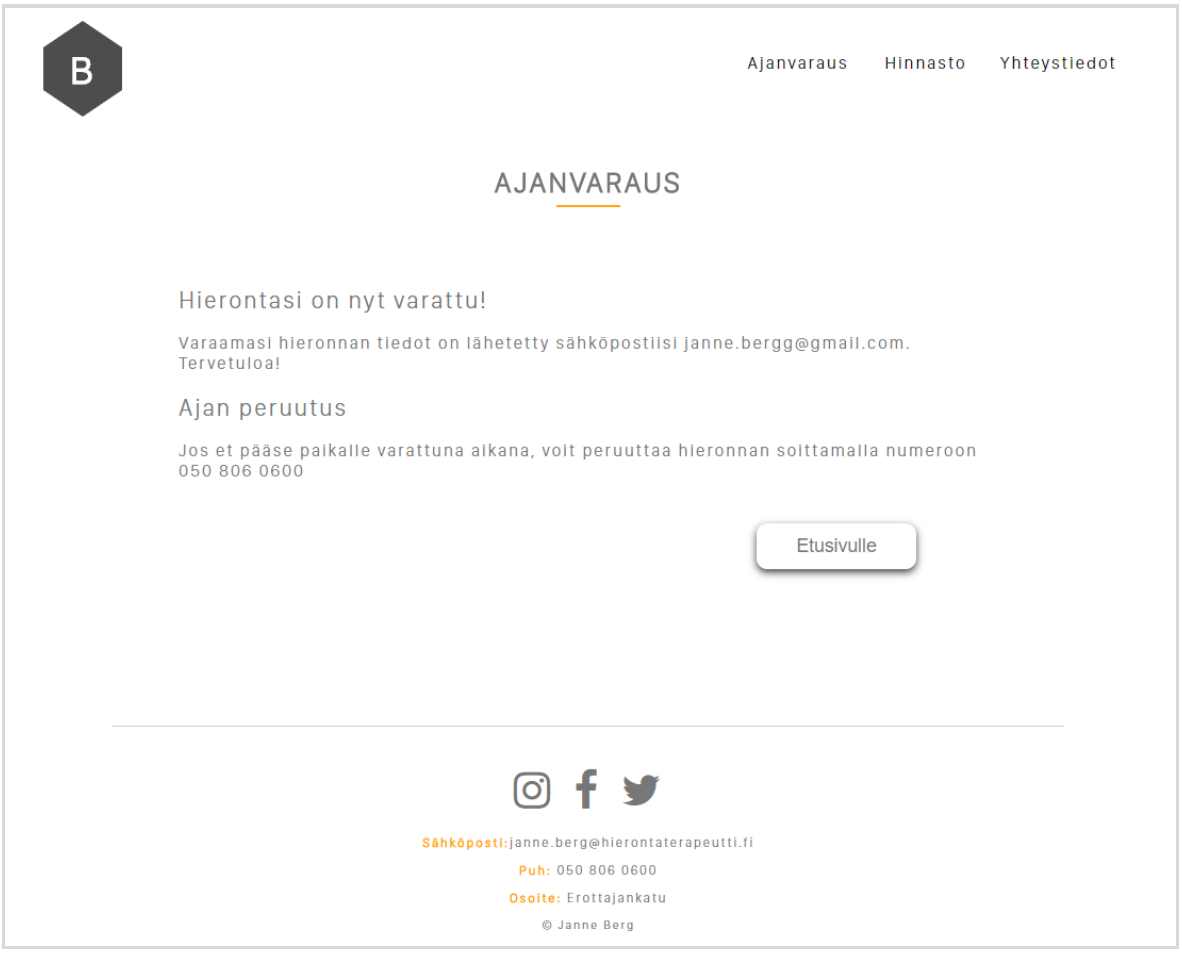


Kuva 33. Vahvistusnäkymä

Yhteenvetonäkymästä käyttäjä näkee vahvistusviestin, jossa kerrotaan ajanvarauksen onnistuneen, varaaman hieronnan tietojen lähteneen käyttäjän sähköpostiin, sekä ajanvarauksen peruutuksen ohjeistuksesta. Sähköpostiosoite tulee näkymään ajanvarauksen tilan objektista, kuva 34 havainnollistaa tämän. Sähköpostin lähetystä ei vielä tähän versioon ehditty tekemään ajanpuutteen takia. Käyttäjä voi tästä näkymästä palata etusivulle, klikkaamalla etusivupainiketta. Tästä sivusta ei enää pääse takaisinpäin ajanvarauksen aikaisempiin vaiheisiin.

```
<p>Varaamasi hieronnan tiedot on lähetetty sähköpostiisi {this.props.customerInfo[4]}. Tervetuloa!</p>
```

Kuva 34. Sähköpostiosoite ajanvarauksen tilasta



Kuva 35. Yhteenvetonäkymä

3.10 Sovelluksen tiedot Firebase tietokantaan

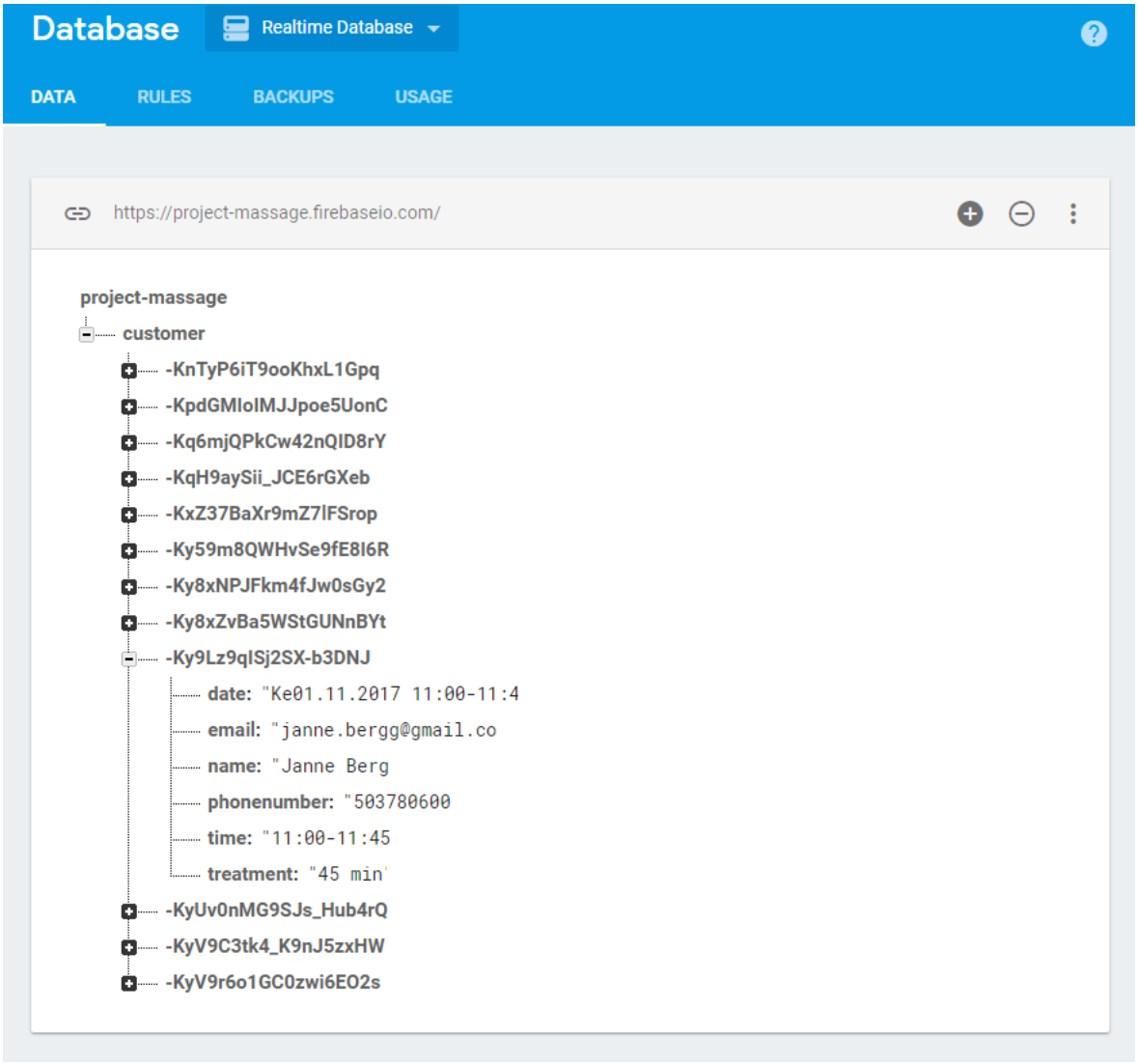
Ajanvarauksen prosessin vahvistusnäkymässä lähetetään käyttäjän hierontahoidosta tiedot Firebaseen tehtyyn tietokantaan. Tietokannan avulla voidaan pitää kirjaa tulevista ja tulleista hoitotilauksista. Tarkoitus olisi tehdä myöhemmin sivuston hallitsijalle oma erillinen web-näkymä mihin kannan tiedot tulisivat kannasta, jotta hoitojen seuraaminen olisi helpompaa.

Ajanvarauksen tilaan kerättyjen hierontahoidon tiedot asetetaan objektille, mikä lähetetään tietokantaan. Muuttuja customerInfoRef kertoo polun, minne tiedot lähetetään kannassa. Käyttäjän kaikki tiedot varauksesta lähetetään customer tauluun. Kuva 36 havainnollistaa koodissa, miten tämä tietojenlähetys tehdään.

Kuvassa 37 nähdään Firebasen graafisessa konsolinäkymässä kantarakenne ja havainnollistetaan, mitkä tiedot ajanvarauksesta otettiin kantaan.

```
sendToFireBase: function(){
  var firebaseRef = firebase.database().ref();
  var customerInfoRef = firebaseRef.child('customer');

    customerInfoRef.push({
      treatment: this.state.customerInfo[0],
      date: this.state.customerInfo[1] + this.state.customerInfo[2],
      time: this.state.customerInfo[2],
      name: this.state.customerInfo[3],
      email: this.state.customerInfo[4],
      phonenumber: this.state.customerInfo[5]
    });

},
```

Kuva 36. Ajanvarauksen tietojen lähetys kantaan



Kuva 37. Sovelluksen kantarakenne, Firebase konsolinäkymässä.

3.11 Sivuston responsiivisuus ja käyttäjäkokemus

Sivuston käyttäjäkokemukseen panostettiin pienten animaatioiden ja videoiden lisäämisellä sivuille, jolla käyttäjäkokemus tehdään mukavaksi ja nautittavaksi. Esimerkiksi ajanvarauksen nappuloihin on lisätty pienet animaatio efektit, etusivulla pyörii video, sekä navigointi palkkiin lisätty pieni animaatio, millä palkki kutistetaan, kun käyttäjä rullaa sivustoa alaspäin.

Tavoitteena oli tehdä sivustosta responsiivinen kaikille laitteille. Nykypäivänä löytyy useita ohjelmointikehyksiä, jolla saadaan sovelluksien tyylit ja responsiivisuus tehtyä, kuten esimerkiksi Bootstrap. Halusin kuitenkin näyttää ja kehittää css taitojani, niin päätin tehdä tyylit ja responsiivisuuden itse, ilman mitään front-end ohjelmistokehystä. Minulla oli ennestään jo hyvät css taidot, niin mitään tyyleihin liittyvää ei varsinaisesti pitänyt opetella

tätä projektia varten. Ainoa apuväline mitä käytin oli SASS, mikä helpottaa css:n kirjoittamista.

Responsiivisuutta lähdettiin tekemään pääasiassa css tyylimäärityksillä. Web-sovellusten responsiivisuudesta kun puhutaan, tulee usein mieleen sanonta ”mobile first”, eli mobiili ensin. Tämän sanonnan johdolla luotiin siis ensimmäiset css-tyylimääritykset mobiilinäkymään ja siitä aina isompiin laitteisiin, kuten tabletti- ja pöytäkonenäkymiin.

Projektissani jokainen komponentti sai omat tyylitiedostot(.scss) ja näin omat tyylimääritykset. Komponenttikohtaiset tyylitiedostot sisälsivät mobiilinäkymän tyylimääritykset, ei siis muiden näkymiä tyylejä, kuten esimerkiksi tablettinäkymän.

Seuraavaksi alettiin kehittämään tyylimäärityksiä tabletti- ja pöytäkonenäkymille. Loin tiedoston responsive.scss, minne asetettiin näkymärajat, joilla määriteltiin milloin komponenttien tyylit vaihtuvat tabletti- tai työpöytänäkymiin. CSS tarjoaa Media Query tekniikan, jolla pystytään määrittelemään näkymärajat. Määritykset kirjoitetaan css tiedostoon. Näkymärajoille asetetaan näytön maksimi pikselimäärä, jolloin mobiilinäkymän tyylimääritykset ajetaan yli ja isomman näkymän tyylimääritykset tulevat voimaan.

Kuvassa 38 on navigointikomponentin css mobiilityylit. Kuvassa 39 on näkymärajan asettamisesta Media Query tekniikalla. Näkymäraja on asetettu 768px. Media Query lohkon sisään on asetettu navigointikomponentin työpöytänäkymän tyylimääritykset.

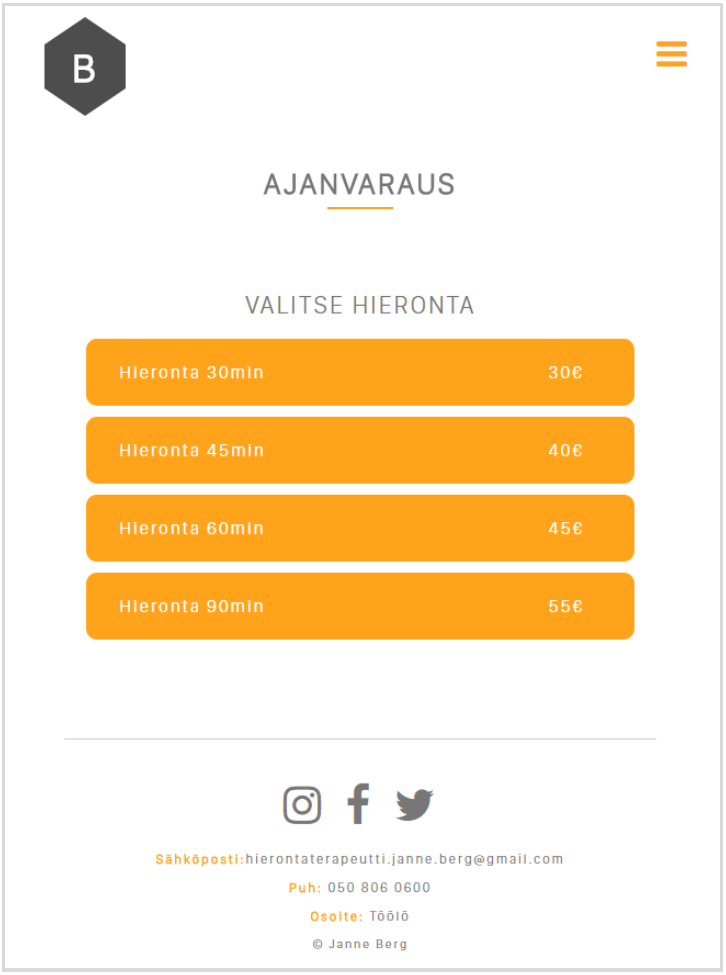
```
/********** NAV MOBILE **********/

#header-container {
  position: fixed;
  text-align: left;
  top:0;
  width: 180px;
  height: 100%;
  left:-180px;
  background-color:$black;
  transition:0.2s all;
  z-index: 1;
  padding-top:130px;
}
.hamburger-icon {
  position: fixed;
  font-size: 2em;
  right:5%;
  top:30px;
  transition:0.2s all;
  cursor:pointer;
  color:$orange;
}
.hamburger-icon:hover {
  color:#222;
}
```

Kuva 38. Mobiilityylit komponenttiin

```
@media screen and (min-width: 768px) {


/*****  NAVIGATION DESKTOP *****/

  #header-container {
    text-align: right;
    top:0px;
    width: 100%;
    height: 60px;
    left:0;

    background-color:transparent;
    padding-top:30px;
    padding-right:30px;


    .hamburger-icon {
      display: none;
    }

    .scrolled-hexagon {
      transition: 1s all;
      opacity: 0;
```

Kuva 39. Työpöytätyylit komponenttiin

Kuva 40. sovelluksen ajanvaraus työpöytänäkymässä



Kuva 41 sovelluksen ajanvaraus mobiilinäkymässä

4 Pohdintaa ja jatkokehitysideoita

Tässä luvussa keskitytään pohtimaan muun muassa annettuihin tavoitteisiin pääsemistä, aikataulua, mitä opittiin, mitkä asiat olivat helppoja, sekä mitkä asiat tuottivat vaikeuksia. Lopussa käsitellään jatkokehitysideoita.

4.1 Pohdinta

Projektin tavoite oli tehdä hierojalle modernin näköinen web-sovellus, jolla käyttäjät pystyvät varaamaan hierontahoitoja.

Tähän tavoitteeseen päästiin minusta tarpeeksi hyvin, koska sovelluksella pystyy varmaan aikoja ja se saatiin näyttämään myös kohtalaisen modernilta. Huomioitavaa on myös se, että sovellus suunniteltiin hyvin ja siitä tehtiin prototyyppi. Lisäksi sovellus skaalautuu kaikille laitteille varsin hyvin.

Aikataulu oli todella tiukka tämän lopputyön suhteen, koska tein samalla täyspäiväisesti töitä. Onnistuin omasta mielestäni kuitenkin hyvin pitämään aikatauluista kiinni olosuhteet mukaan ottaen. En kuitenkaan ole täysin tyytyväinen vielä sovellukseen, koska sovellusta ei ehditty testaamaan tarpeeksi kattavasti, jotta mahdollisilta ongelmilta vältyttäisiin. Suunnitteluvaiheessa sovelluksen prototyyppiä testattiin useamman käyttäjän voimin. Varsinaista sovellusta testattiin vain parilla käyttäjällä. Testaus tullaan tekemään kattavammin tulevaisuudessa.

Toinen asia johon en ole tyytyväinen, on sähköpostivarmistuksen puuttuminen sovelluksesta. Käyttäjän olisi tarkoitus saada varmistusviesti sähköpostiin, kun hieronta-aika on varattu. Tämä jäi tekemättä, koska aikataulu oli liian kiireinen. Tullaan tekemään myöhemmin.

Myös ulkoasun suhteen tullaan tekemään vielä jonkin verran muutoksia, esimerkiksi sivuilla näkyvä logo tullaan muuttamaan vielä erinäköiseksi. Lisäksi otetaan uudet kuvat ja videot, jotta sivusto näyttää enemmän itse hierojan näköisiltä. Ulkoasun suhteen olen muuten tyytyväinen. Sivusto näyttää omaan silmään hyvältä ja se on täysin responsiivinen tyyleiltään.

Otin projektiin paljon uutta opittavaa, tuntui että vähän liikaakin otin, koska aikataulu ei aivan täysin pitänyt. Olen kuitenkin tyytyväinen, että haastoin itseni ottamaan uusia tekniі-

koita opittavaksi. Reac.js:n oppiminen vei paljon aikaa ja oli suhteellisen haastava, koska minulla ei ollut aikaisempaa kokemusta SPA tekemisestä. Opiskelin Reactin nettikurssilla jonka Treehouse.com tarjosi.

Tietokantojen tekeminen Firebasella oli myös minulle uusi asia. Onneksi Firebaseen tutustuminen oli helppoa ja heillä oli omilla sivuillaan hyvät ohjeet, miten toimia. Firebase sivusto tarjosi myös videoita nimenomaan React projekteihin.

Sovellusten suunnittelusta ja prototyyppien tekemisestä minulla oli jo aikaisempaa kokemusta, joten suunnitteluvaihe meni projektissa suhteellisen jouhevasti. Sketch ja Invision olivat tuttuja sovelluksia. Lisäksi minulta löytyi vankkaa CSS osaamista, niin sivuston tyylien ja responsiivisuuteen liittyvät asiat oli helppo tehdä.

4.2 Jatkokehitysideat

Projektin aikana tuli lukuisia jatkokehitysideoita, kuten varausviestin lähetys, asiakasdatan seuranta, automaatiotestit, sekä varatuista ajoista erillinen näkymä sovelluksen omistajalle. Kaikki ideat tullaan toteuttamaan lopulliseen versioon.

Ajanvarauksen varausviestin voisi lähettää tekstiviestinä tai sähköpostilla. Firebase tarjoaa FCM-palvelun (Firebase Cloud Messaging) josta voi lähettää tekstiviestejä tai sähköposteja suoraan asiakkaille/käyttäjille. Varausviesti tullaan tekemään tällä palvelulla tulevaisuudessa.

Erillinen näkymä varatuista hieronta-ajoista tullaan myös toteuttamaan tulevaisuudessa. Hieronta-ajat data tuodaan näkymään sovelluksen Firebase kannasta. Näkymässä luultavasti ajat esitetään listamuodossa. Näkymässä myös mahdollistetaan yksittäisten aikojen poisto. Myös erilaisen käyttäjädatan kerääminen näkymään voisi onnistua. Käyttäjädatanäkymä voisi olla esimerkiksi dashboard tyylinen, mistä näkyisi asiakkaiden dataan liittyviä lukuja, esimerkiksi tulevan viikon hoitojen määrät, hoitojen kokonaismäärä, hoidoista tuleva rahan määrä jne.

Automaatiotestauksen tekeminen sovellukseen luultavasti jää myöhemmäksi. Minulla ei ole aikaisempaa kokemusta näiden tekemisestä, joten voi jopa olla, että tässä projektissa niitä ei koskaan tulla näkemään.

Lähteet

Apple 2017a. Autosave. Luettavissa:
https://support.apple.com/en-us/HT202255
Luettu: 19.8.2017

Apple 2017b. Sketch Mirror. Luettavissa:
https://itunes.apple.com/us/app/sketch-mirror/id677296955?mt=8
Luettu: 20.8.2017

BestDesigns 2017. The Best Design. Luettavissa:
https://www.thebestdesigns.com/
Luettu: 4.7.2017

Codeacademy 2017. React: The Virtual DOM. Luettavissa:
https://www.codecademy.com/articles/react-virtual-dom
Luettu: 2.9.2017

Dribbble 2017. Dribbble. Luettavissa:
https://dribbble.com
Luettu: 4.7.2017

Firebase 2017a. Firebase Pricing. Luettavissa:
https://firebase.google.com/pricing/
Luettu: 3.9.2017

Firebase 2017b. Firebase realtime database. Luettavissa:
https://firebase.google.com/docs/database/
Luettu: 3.9.2017

Firebase 2017c. Add Firebase to your JavaScript Project. Luettavissa:
https://firebase.google.com/docs/web/setup
Luettu: 3.9.2017

GitHub 2017a. GitHub. Luettavissa:
https://github.com/
Luettu: 25.8.2017

GitHub 2017b. GitHub. Luettavissa:
https://github.com/pricing
Luettu: 26.8.2017

Invision 2017a. Prototyping tool. Luettavissa:
https://www.invisionapp.com/tour/website-mobile-prototyping-tool
Luettu: 10.10.2017

Invision 2017b. Pricing. Luettavissa:
https://projects.invisionapp.com/d/main#/upgrade/plans?openedDueTo=plans%20page
Luettu: 19.8.2017

Invision 2017c Sketch prototyping. Luettavissa:
https://www.invisionapp.com/sketch-prototyping
Luettu:20.8.2017

React 2017a. React. Luettavissa:
https://reactjs.org/
Luettu: 20.8.2017

React 2017b. Components and props. Luettavissa:
https://reactjs.org/docs/components-and-props.html
Luettu: 20.8.2017

React 2017c. State and lifecycle. Luettavissa:
https://reactjs.org/docs/state-and-lifecycle.html
Luettu: 20.8.2017

Sketch 2017a. Pricing. Luettavissa:
https://www.sketchapp.com/pricing/
Luettu: 16.8.2017

Sketch 2017b. Documentation. Luettavissa:
https://www.sketchapp.com/docs/
Luettu: 19.8.2017

Sketch 2017c. Exporting. Luettavissa:
https://www.sketchapp.com/docs/exporting/
Luettu: 19.8.2017

Sketch 2017d. Toolbar. Luettavissa:
https://www.sketchapp.com/docs/the-interface/toolbar/
Luettu: 19.8.2017

Sketch 2017e. Symbols. Luettavissa:
https://www.sketchapp.com/docs/symbols/
Luettu: 19.8.2017

Techmagic 2017. Angular 2 vs React. What to chose in 2017? Luettavissa:
https://blog.techmagic.co/angular-2-vs-react-what-to-chose-in-2017/
Luettu:20.8.2017

CNET 2017. Apple Announces 2012 Design Award Winners. Luettavissa:
https://www.cnet.com/news/apple-announces-2012-design-award-winners/
Luettu: 15.8.2017

Crunchbase 2017. Firebase. Luettavissa:
https://www.crunchbase.com/organization/firebase#section-overview
Luettu: 15.8.2017