Haaga-Helia
ammattikorkeakoulu Oy

Henkilöstöportaalin kehitys

Aapo Laakso

Opinnäytetyö
Tietojenkäsittelyn
koulutusohjelma
2018

Tiivistelmä

Haaga-Helia ammattikorkeakoulu Oy

Tekijä(t)
Aapo Laakso

Koulutusohjelma
Tietojenkäsittelyn koulutusohjelma

Raportin/Opinnäytetyön nimi
Henkilöstöportaalin kehitys

Sivu- ja liitesivumäärä
29 + 2

Opinnäytetyön tarkoitus on selostaa Talented Solutions Oy:lle toimeksiantona toteutettu käyttöliittymän kehitys. Tarpeena oli luoda olemassa olevalle järjestelmälle uusi näkymä. Sen tarkoitus olisi parantaa sovelluksen käytettävyyttä. Työn laajuus rajattiin pelkkään front-end -kehittämiseen. Ajallisesti työn suorittaminen sijoittui tammikuulle 2018. Alustavat käyttäjähaastattelut suoritettiin joulukuussa 2017.

Kehitystyö tehtiin hyödyntämällä React Javascript-kirjastoa. Opinnäytetyön tietoperusta esittelee kirjaston ja sen hyödyntämisen käytännössä. Lisäksi käydään läpi muita hyödynnettäviä työkaluja. Esimerkiksi paketointityökalu Webpack on esiteltävä työkalu.

Opinnäytetyön lopputuloksena syntyi uudistettu henkilöstönhallinnan käyttöliittymä, jota toimeksiantaja voi hyödyntää. Tuloksista hyötyvät myös kaikki, jotka haluavat oppia käyttöliittymäkehityksestä ja React-kirjastosta. Huomioitavaa oli lyhyt kehitysaika, jonka puitteissa syntyi tavoiteltu lopputulos.

Asiasanat
React, Redux, käyttöliittymä

Sisällys

1 Johdanto

Talented Solutions oy on henkilöstönvälitysyritys. Yrityksen tarkoitus on tarjota osaaville sovelluskehittäjille parempi kokemus työnhakemisesta. Samalla se tarjoaa kumppaniyrityksilleen osaavan kokoelman sovelluskehittäjiä. Keskeinen käsite yrityksen sanastossa on talentti. Talentit ovat heidän ohjelmistokehittäjäasiakkaitaan, jotka etsivät uusia mahdollisuuksia työurallaan. Yritys on ollut toiminnassa reilun vuoden ja työllistää alle 20 työntekijää. Työnhaun lisäksi yritys tarjoaa verkostonsa jäsenille koulutuksia ja erilaisia tapahtumia.

Toimeksiannon tavoite on kehittää Talented Solutionsin sisäistä, henkilöstön käytössä olevaa portaalia. Palvelun kehitys on jäänyt arjen töiden jalkoihin ja aikaa kehitystyölle ei ole ollut. Yrityksellä on tarve kustomoidulle järjestelmälle, koska kustannustehokkaita vaihtoehtoja se ei ole löytänyt. Tarjolla olevat palvelut ovat kalliita ja niiden ominaisuudet eivät vastaa liiketoiminnan tarpeita. Työn tulokset ovat erittäin merkittäviä, sekä liiketoiminnan kriittisiä. (Tiilikainen 25.4.2018.) Tarkoituksena on työntekijöiden haastattelujen avulla selvittää nykyisen käyttöliittymän ongelmakohdat. Suoritettujen haastattelujen pohjalta tehdään suunnitelma uudesta käyttöliittymästä. Suunnitelman mukaan tuotetaan produktina uusi käyttöliittymä toimeksiantajalle.

Tämän opinnäytetyön tarkoitus on yrityksen käyttöliittymän kehittäminen ja kehitystyöprosessin läpikäynti. Lopussa analysoidaan lopputulosta ja omaa oppimista. Pohdinnan tukena ovat loppukäyttäjien haastattelut valmistuneessta produktista.

Tietoperusta opinnäytetyölle tulee käytössä olleen Javascript-kirjasto Reactin ja sen käsitteiden läpikäynnistä. Kirjaston lisäksi esitellään muita yleisesti Reactin kanssa käytettyjä tekniikoita ja työkaluja. Opinnäytetyö on rajattu käsittelemään sovelluksen käyttöliittymää ja sen kehittämistä. Taustajärjestelmän käsittely jää lähinnä maininnan tasolle. Tuloksista hyötyvät kaikki, jotka haluavat tutustua React-kirjastoon ja sen periaatteisiin. Lisäksi pienen yrityksen sisäisen kehittämisen tavat tulevat esille.

2 React-kirjasto

Tämän luvun tarkoitus on kertoa käyttöliittymien tekoon tarkoitetusta React-kirjastosta. Ensimmäiset alaluvut käsittelevät kirjaston ideologiaa ja pääkäsitteitä. Sen lisäksi tarkastelussa ovat Reactin kanssa hyödynnettäviä työkaluja. Esimerkiksi Webpack nimisellä ohjelmalla voidaan yhdistää erilaisia tiedostoja yhdeksi paketiksi. Luettuaan kappaleet, lukijan tulisi ymmärtää React-sovelluksen rakenne ja tiedonkulku sen osien välillä.

2.1 Reactin ideologia

React on Facebookin insinöörien kehittämä Javascript-kirjasto verkkosovellusten käyttöliittymien toteuttamista varten. Reactin historia on PHP:ssa ja nimenomaan sen Javascript käännös XHP:ssa. XHP:n avulla voidaan XML-syntaksin avulla luoda HTML-elementtejä PHP:lla. XHP:n tarkoitus on vähentää sivujen välisiä hyökkäyksiä XSS (cross site scripting). (Laverdet 2010.) Iso ongelma sen lisäksi olivat dynaamisten sivujen jatkuvat kutsut palvelimelle. Se kuluttaa verkkoa ja vie aikaa käyttäjältä. (Dawson 2014.)

Kirjaston kehittäminen alkoi 2011 ja syynä olivat ongelmat isojen järjestelmien koodin ylläpidon kanssa. Facebook Ads -projekti kasvoi ja sille tarvittiin enemmän kehittäjiä. Ylläpito muuttui hankalaksi ominaisuuksien ja tekijöiden lisääntyessä. Jordan Walke lähti kokeilemaan ja kehittämään parempaa ratkaisua. Sen lopputuloksena on ReactJS-kirjasto. Reactin oli tarkoitus alun perin olla vain Facebookin sisäinen, mutta Instagramin oston jälkeen syntyi tarve laajemmalle käytölle myös Facebookin ulkopuolella. Sen pohjalta koko kirjasto päätettiin muuttaa avoimeksi lähdekoodiksi 2015. (Fisher 2015; Hunt 2015.)

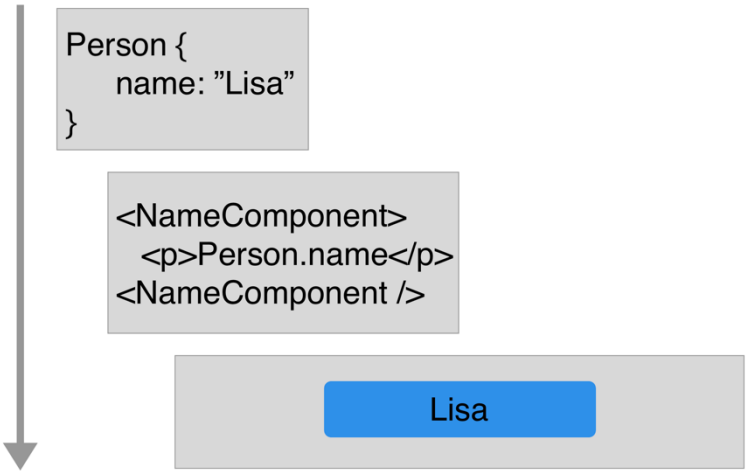
Reactin tärkein oivallus on tehdä verkkosivun sisällön uudelleenlataus vain silloin kun muutosta tapahtuu ja vain sen sisällön osalta, joka muuttuu. Kaikki tapahtuu selaimessa, joten renderöintiin eli uudelleenpiirtoon ei tarvita palvelimella tapahtuvaa ja lähetettävää laskentaa. Kuorman vähentäminen palvelimelta ja siirtäminen osin käyttäjän koneelle mahdollistaa monipuolisemmat verkkosivut, jotka eivät ole täysin riippuvaisia verkko-olosuhteista. Käyttäjä voi silloin hakea ja manipuloida dataa lataamatta sivua uudestaan. (Dawson 2014.) Muutoksia React seuraa käyttämällä virtuaalista kuvausta selaimen DOM-puusta. Siitä kerrotaan enemmän myöhemmässä luvussa. Toinen olennainen osa Reactia on sen komponenttipohjaisuus, jossa sovellus on jaettu pieniin mahdollisimman itsenäisiin kokonaisuuksiin. Komponentit ovat parhaassa tapauksessa uudelleenkäytettäviä kyseisen sovelluksen sisällä ja mahdollisesti täysin muissa sovelluksissa. (Reactjs) React on myös rakennettu skaalautuvaksi, ja sitä voidaan käyttää pienistä SPA (single

page application) -tyyppisistä ratkaisuista Netflixin kokoisen palvelun osien rakentamiseen. (Netflix 2015.)

Reactin etuja on, että se pakottaa miettimään sovelluksen rakennetta. Kun rakenne on mietitty hyvin, tulee DRY:n (don't repeat yourself) mukaista kirjoittamista enemmän. Sille ei aluksi ole välttämättä tarvetta, mutta sovelluksen kasvaessa se lisää luettavuutta. (Reactjs)

React on vain tapa tehdä ja kirjoittaa koodia. Se ei ole tiukka kehikko (framework), jonka mukaan verkkosovellus tulee rakentaa. Näin se tarjoaa mahdollisuuden helposti yhdistää erilaisia kehikoita ja muita ohjelmistokehityksen tapoja. Kirjaston ja kehikon ero on rakenteessa. Kehikko antaa kehittäjälle mallin, jonka mukaan sovellus tulee rakentaa. Kirjasto sen sijaan ei tarjoa valmista rakennetta sovellukselle. Sen mukana on vain rakennuspalasia. Kirjaston mukana tulee paljon vapautta, mutta myös vastuuta. Kehikon kanssa kehittäjä voi olla huolettomampi, koska mahdollisuus tehdä virheitä on pienempi. (Spaho 2015.) Koska React on kirjasto, sitä on mahdollista hyödyntää helposti olemassa olevissa sovelluksissa. Ei ole tarvetta kirjoittaa koko järjestelmää uudestaan sitä varten. Reactin sivuilla ehdotetaan kokeilemaan ensin pienessä osassa sovellusta ja siitä jatkokehittämään. (Reactjs)

React on osa trendiä eritellä käsiteltävä data ja käyttöliittymä. Kuvassa 1 on esimerkinomaisesti kuvattu pseudokoodilla idea. Käyttöliittymä vain kutsuu Javascript objektia, johon on tallennettuna data, tässä tapauksessa nimi. Käyttöliittymän muokkaaminen ei ole näin riippuvainen käsiteltävän datan sisällöstä. Sininen laatikko näyttäisi aina samalta, ainoastaan sen sisällä oleva nimi muuttuisi annetun objektin parametrien perusteella.



Kuva 1. Käyttöliittymä on vain datan funktio

2.2 Komponentti

Pitkän yksittäisen dokumentin sijaan React kostuu komponenteista, yksittäisistä tiedostoista, joita on helppo käyttää uudelleen. Ne ovat Javascript funktioita ja luokkia jotka ottavat parametrejä (props) vastaan ja palauttavat kuvauksen käyttöliittymästä saatujen parametrien perusteella. Koska kaikki kirjoitetaan Javascriptilla, voidaan komponenttien luomiseen käyttää kaikkia kielen ominaisuuksia. Yksinkertainen komponentti on vain Javascript luokka, joka palauttaa HTML-elementin sille annettujen parametrien perusteella.
Alla olevassa esimerkissä React luokka palauttaa H1-otsikon, jossa lukee ”Hello, User”

```
class Greeting extends React.Component {
  render() {
    return <h1>Hello, User<h1>;
  }
}
```

Komponentit ovat vain ideologia tehdä asioita, eikä React ota tarkkaan kantaan mittaan ja malliin. Ainoastaan render()-metodi on pakollinen, koska ilman sitä komponentti ei palauta mitään ja olisi turha.
(Reactjs)

Sovelluksen suunnittelija saa siis tehdä komponentit sellaiseksi kuin kulloinkin tarvetta. Koska HTML elää Reactin sisällä Javascript-objekteina, se on vain palautettava arvo funktioissa. Palvelimen palauttaessa käyttäjän pyynnöstä sivu, kootaan komponentit yhteen ja näytetään kokonainen sivu ja yhtenäinen HTML-dokumentti. (Fischer 2016.)

2.3 Komponentin tila

Komponentilla voi olla erilaista sisäistä dataa. State (komponentin sisäinen, muuttuva tila) ja properties/props (komponentin sisäiset muuttumattomat ominaisuudet). Käytännössä ne ovat vain Javascript-objekteja. Propsit ovat komponentille ylhäältä annettuja, vain luettavaa dataa. Propseja ei muokata komponentin sisällä. Koska ne ovat muuttumattomia, joten yritys muokata niitä ei näkyisi missään. (Reactjs). Alla olevassa esimerkissä renderöitävä elementti saa vanhemmaltaan tiedon, miltä sen kuuluu näyttää ja palauttaa h1-elementin. Funktio Welcome ei pysty muuttamaan saamaansa propsia. Sen pystyy tekemään vain sen vanhempi.

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
const element = <Welcome name="Sara" />;
```

Selaimessa esimerkki palauttaisi h1-otsikon ja siinä lukisi Sara.

State on komponentin sisäinen data, jota voidaan muuttaa. Muodoltaan se voi olla mikä tahansa Javascript-objekti. Sovellusta rakentaessa tuleekin miettiä tarkkaan kumpaa datamuotoa komponenteilla käyttää. Koska propsit eivät muutu komponentin itsensä toimesta niiden käyttö on suositeltavampi. Esimerkiksi ohjelmistovirheiden välttämiseksi. (Fischer 2016.)

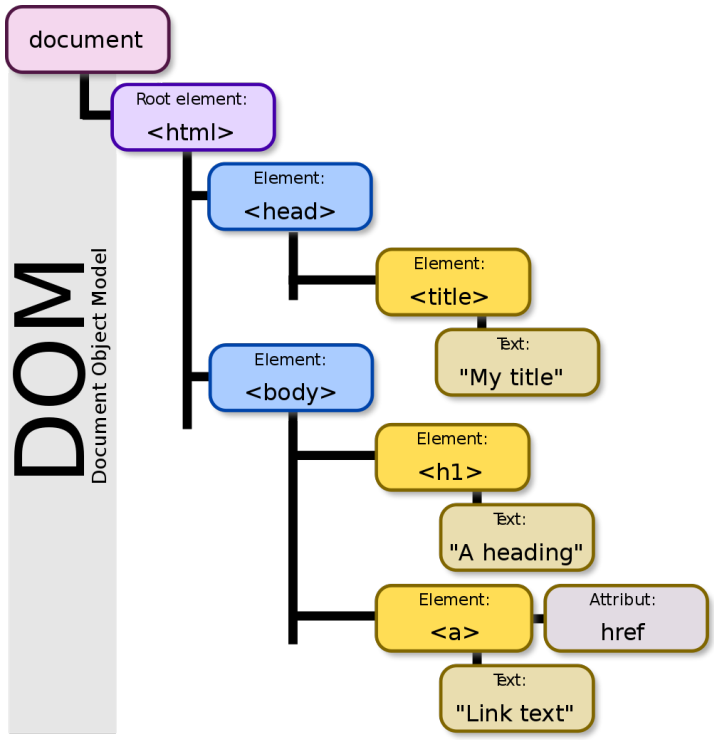
Sovellusta tehdessä tulee huomioida, että Reactin sisällä data kulkee ylhäältä alas, vanhemmalta lapselle. Komponentit tulee suunnitella sen logiikan mukaan. Jos tilamuutoksen tulee näkyä muissa komponenteissa, tulee muutos lähettää propseina alaspäin. Tilan siirtyminen vain alaspäin, tekee sovelluksesta ennustettavan ja helpomman seurata. Komponenttien sisäistä tilaa tulisi olla mahdollisimman vähän. Koska se vähentää ohjelmistovirheitä, kuten aiemmin on mainittu. Sovelluksen arkkitehtuuria suunnitellessa tulisi muuttuvien tilojen määrä tunnistaa. Hyviä kysymyksiä sen tunnistamiseen ovat: muuttuko datan arvo ja onko arvo riippuvainen toisista muuttujista? (Fischer 2016.)

State tallennetaan komponentissaan omaan objektiinsa this.state. Reactin dokumentaatio suosittelee alustamaan staten luokan konstruktorissa. Siinä tulee käyttää super-syntaksia. Super tarkoittaa, että luokkaa laajentaa sen vanhempaa ja saa sen propsit käyttöönsä. Statea voidaan muokata vain this.setState funktiolla (Reactjs). Alla olevassa esimerkissä on tilallinen komponentti, jossa nappulaa painamalla voidaan vaihtaa toisen komponentin väriä

```
class Box extends React.Component {
  constructor(props) {
    super(props);
    this.state = { color: ‘red’ };
  }

  changeColor() {
    this.setState(color: ‘green’);
  }

  render() {
          <button onClick={this.changeColor()} />
          <Block style={{backgroundColor: this.state.color}} />
  }
```

2.4 Virtuaalinen DOM

DOM (Document Object Model) on selaimen muistissa oleva kuvaus HTML-dokumentista. Malli kuvataan puurakenteisena, eli se muistuttaa puun juuria, jotka haaroittuvat pääjuuresta. Kuvan 2 mallin mukaan.



Kuva 2. DOM-mallin kuvallinen esitys. (Eriksson 2012.)

Selaimet ja Javascript tarjoavat rajapinnan DOM-päivityksen tekemiseen, kuten esimerkiksi AppendChild() ja innerHTML(). Funktioilla voidaan muokata jo renderöityä näkymää, ja laittaa tilalle uutta tietoa. DOM:n manipulointi ja sen elementeissä liikkuminen on kehittäjän näkökulmasta helppoa. Nykypäivänä se voi olla kuitenkin hidasta. Suuret, yhdestä DOM-objektista koostuvat yksisivuiset sovellukset ovat suosittuja. Myös puhtaan DOM-rajapinnan käyttäminen ei ole tehokasta, koska sitä käytettäessä, joutuu toistamaan koodia paljon. Siksi on luotu kirjastoja kuten React, helpottamaan DOM-manipulointia. (Krajka 2015.) Reactilla tehty yksinkertainen Hello World sovellus näyttää tältä. Se renderöi <h1> -elementin HTML-tiedostossa sellaisen elementin sisälle, jonka ID on ”root”.

```
ReactDOM.render(
  <h1>Hello, world!</h1>,
  document.getElementById('root')
);
```

Pelkällä Javascriptilla vastaava saataisiin tehtyä näin

```
document.getElementById('root').appendChild(’<h1>Hello, World</h1>).
```

Hello World on huono esimerkki, koska se on niin yksinkertainen, mutta monimutkaisempia sovelluksia tehdessä. Kirjoittaminen nopeutuu huomattavasti.

React auttaa DOM-manipuloinnin suorituskykyyn liittyvissä ongelmissa. Reactissa komponentit kuvaillaan niitä tehdessä, eli ei tarvitse liikkua DOM-puussa. React huolehtii siitä. Kehittäjä vain kertoo komponenteille, miltä niiden tulee näyttää. Se ei itsessään ratkaise

suorituskykyongelmia. Sen sijaan React luo virtuaalisen DOM-puun. Se on yksinkertaistettu versio, jota vain React lukee. Reactin ei tarvitse huolehtia miten kukin selain hallitsee DOM:ia, vaan muutokset siihen tehdään vasta kun React tietää tarkasti mitä pitää tehdä. React erottaa muuttuvat komponentit ja muuttumattomat elementit. Muuttuvat komponentit tehdään elementeiksi. Jonka jälkeen se vertaa mahdolliset muutokset virtuaaliseen DOM:iin ja tekee muutokset oikeaan muistissa olevaan DOM:iin jos eroa on. Se huolehtii suorituskyvystä. (Krajka 2015; Reactjs)

2.5 JSX

JSX on Facebookin kehittämä Javascript-syntaksin laajennus. Sen tarkoitus on helpottaa React-elementtien kirjoittamista vähentämällä koodia. JSX:n ei ole pakollista, mutta suositeltava syntaksi Reactia kirjoitettaessa. Ideana on tehdä HTML:ää muistuttava tapa kirjoittaa elementtejä Javascript tiedoston sisälle (Reactjs). Alla olevassa esimerkissä näkyy, miten JSX:llä kirjoitettu elementti on luettavampi ja helpommin kirjoitettu:

```
<MyButton color="blue" shadowSize={2}>
  Click Me
</MyButton>
```

On puhtaana Javascriptinä kirjoitettuna:

```
React.createElement(
  MyButton,
  {color: 'blue', shadowSize: 2},
  'Click Me'
)
```

Esimerkissä elementin tyylit on määritetty sen sisällä, mutta on suositeltavampaa käyttää erillisiä CSS-tiedostoja. Ne ovat suorituskyvyn kannalta parempi vaihtoehto. Tavallinen JavaScript mahdollistaa kirjoitusvirheet paljon helpommin. JSX:n sääntöjä ovat esimerkiksi, että JSX:ssä isolla kirjaimella alkavat elementit ovat itsetehtyjä tai ladattu toisesta kirjastosta. Pienellä kirjoitetut ovat tavallisia HTML-elementtejä. Jos vahingossa kirjoittaa omat komponenttinsa pienellä kirjaimella, voi aiheuttaa paljon ohjelmistovirheitä. Silloin React kuvittelee komponentin olevan standardin mukainen HTML-elementti. (Reactjs)

JSX propseina saa olla vain suoria Javascript-ilmauksia, (expression) eli Javascriptiä joka palauttaa arvon. If ja For (statements) ovat kiellettyjä. Jos tarvetta konditionaaliseen renderöintiin, se pitää tehdä palautettavan JSX elementin ulkopuolella. On myös mahdollistaa käyttää ternary operator -syntaksia, joka näyttää tältä.

```
The user is <b>{isLoggedIn ? 'currently' : 'not'}</b> logged in.
```

Aaltosulkujen sisällä olevassa koodissa katsotaan muuttujan arvo ja palautetaan jompikumpi kysymysmerkin jälkeen olevista arvoista. Jos muuttujan arvo on tosi, palautetaan

vasemmalla puolella oleva ja muuttujan ollessa epätosi palautetaan oikealla puolella oleva arvo.

Sovellusta rakennettaessa tuotantoon kääntäjänä toimiva Babel muuttaa kaiken JSX:n tavalliseksi Javascriptiksi. Se myös tarkastaa syntaksin oikeellisuuden. (Reactjs)

2.6 ES6/ES7 ja Babel

Tässä luvussa käydään läpi Javascriptin uutta syntaksia, joka tuo lisää ominaisuuksia kieleen. Lopuksi esimerkkejä ominaisuuksista, jotka ovat hyödyllisiä React-kehityksessä. Javascript on kehittyvä kieli, jonka syntaksia päivitetään nykyään vuoden välein. Päivitykset pohjautuvat kielen standardina toimivaan EcmaScriptiin. Versiot nimetään sen mukaan. Esimerkiksi ECMAScript 2015 joka yleensä lyhennetään muotoon ES6, koska se on standardin kuudes versio. (Developer.Mozilla 2016.) ES6 oli iso päivitys kieleen ja toi paljon uusia ominaisuuksia. Uuden syntaksin saaminen käyttöön kesti kuusi vuotta. Edellinen versio ES5 oli julkaistu 2009. (Ecma-International 2017) Sen takia Javascriptia hallinoiva komitea siirtyä vuosittaisiin päivityksiin. Muutoksen takia kielen kehitys on tällä hetkellä iteroivaa. Vuosittainen päivitystahti mahdollistaa myös asioiden siirtämisen vuodella, jos jotkin osat eivät valmistu. Tarkempi aikataulu helpottaa Javascriptin käyttäjiä, koska se luo jatkuvuutta ja ennakoitavuutta. (Neves 2017.)

Koska kielen uudet ominaisuudet vaativat selainten päivityksiä, tarvitaan uusien ominaisuuksien käyttöönottoon Javascript-käännin, jonka avulla voidaan uuden syntaksin mukaista koodia muuttaa sellaiseksi, että vanhempaa syntaksia käyttävät selaimet ymmärtävät sitä. (Sengstacke 2016.)

Babel on käännin sitä varten. Se asennetaan paketinhallinnan kautta ja kerrotaan konfiguraatiotiedostossa, millaista käännöstä sen tulee suorittaa. Erilaiset lisäosat mahdollistavat erilaisten kirjastojen hyödyntämisen kehityksessä, jolloin selaintuesta ei tarvitse huolehtia. (Babeljs) React kehityksessä Babel on todella hyödyllinen. Se kääntää JSX-tiedostot tavalliseksi Javascriptiksi ja mahdollistaa ES6 ominaisuudet.

Reactin kanssa työskennellä hyödyllisiä ES6:n mahdollistavia ominaisuuksia löytyy useampi. Niitä ovat esimerkiksi nuolifunktio ja leksikaalinen *this*. Nuolet ovat lyhenne funktiosta, ja ne käyttävät => syntaksia. Nuolifunktiossa this (tämä) merkitys pysyy samana kuin sen vanhempifunktiolla. Nuolifunktiossa ei tarvitse kirjoittaa function ja return. (ExploringJS 2015.)

Esimerkkinä ryhmän läpikäyminen.

```
const arr = [1, 2, 3];
```

ES6:

```
const squares = arr.map(x => x * x);
```

ES5:

```
const squares = arr.map(function (x) { return x * x });
```

Luokat ovat objekteja, jotka on tehty koodiltaan yksinkertaisemmiksi. Luokan sisällä funktiota ei tarvitse function liitettä eteensä, pelkästään nimi ja sulkujen sisään tulevat argumentit riittävät. Luokkia vois myös johtaa toisista extends komennolla. Jolloin se saa sen ominaisuudet käyttöönsä. Esimerkissä luokka Dog on jatkettu Animal luokasta ja saa sen speak metodin käyttöönsä. (Franklin 2016.)

```
class Animal {
  constructor(name) {
    this.name = name;
  }

  speak() {
    console.log(this.name + ' makes a noise.');
  }
}

class Dog extends Animal {
  speak() {
    console.log(this.name + ' barks.');
  }
}

var d = new Dog('Mitzie');
d.speak(); // Mitzie barks.
```

ES6 tuo mukanaan uusia tapoja määritellä muuttujia. Let ja const korvaavat var muuttujan. Let on muuttuja, jota ei voi käyttää ennen kuin sille on annettu arvo. Toisin kuin var tyyppinen muuttuja, joka palauttaa arvon ‘undefined’, jos sitä ei ole määritelty. Const taas on pysyvä muuttuja, jonka arvoa ei voi muuttaa. Sen käyttöä suositellaan Reactin kanssa, koska suurinta osaa muuttujista ei tarvitse vaihtaa. Jos kehittäjä käyttää tätä tapaa johdonmukaisesti silloin muuttujatyyppi let nousee esiin erikoisena ja tiedetään, että nyt on jotain muuttumassa. (Elliot 2015; JBallin 2017.)

Moduulien avulla voidaan käyttää import ja export komentoja. Se mahdollistaa sovelluksen jakamisen pienempiin osiin. Moduulin koko voi vaihdella kirjastosta vain yhteen muuttujaan. Tiedosto joka kutsuu moduulia, suorittaa sen koodia vasta kun koko moduuli on ladattu. (ExploringJS 2015.)

2.7 NPM

Npm (node package manager) on avoimen lähdekoodin Javascript-pakettien jakeluun tarkoitettu rekisteri. Se on tällä hetkellä maailman suurin. NPM koostuu sen verkkosivusta, komentorivityökalusta ja varsinaisesta rekisteristä. Verkkosivujen kautta käyttäjät voivat etsiä paketteja ja komentorivityökalulla kehittäjät asentavat paketit omalle koneelleen. Pakettienhallinta hoidetaan projektin juuritiedostossa olevan pakettitiedoston package.json avulla. Sinne kehittäjä määrittelee projektin riippuvuudet. Muut projektin kanssa työskentelevät voivat asentaa ne yhdellä komennolla helposti. NPM:n suurin vahvuus on sen koko. Todella moneen kehittäjän kohtaamaan ongelmaan löytyy valmis ratkaisu, jonka voi ottaa käyttöön projektissaan. Sen sijaan, että käyttäisi aikaa ominaisuuden kirjoittamiseen. (Npmjs, jakemmarsh 2015.) Myös React on NPM-paketti.

NPM toimii myös projektin rakennustyökaluna (build tool). Package.json tiedostoon voidaan kirjoittaa ajettavia skriptejä. Yleisimpiä kehittäjien käyttämiä komentoja ovat erilaisten testien ajaminen, sekä kehityspalvelinten käynnistäminen. Yhteen komentoon voidaan sitoa useampi ajettava skripti. Esimerkiksi kehityspalvelimen käynnistäminen voi vaatia useamman eri toimen suorittamista etukäteen ja ne voidaan NPM:n avulla tehdä yhdeksi komennoksi. Alla olevassa esimerkissä käynnistetään Webpackin kehityspalvelin ja määritetään sille konfiguraatiotiedosto. Webpackista kerrotaan enemmän myöhemmässä luvussa.

```
"dev": "webpack-dev-server --config webpack.config.js"
```

Komento ajetaan projektin juuritiedostosta komentorivikomennolla:

```
npm dev
```

(NPM 2018; Gazimada 2017; Madhankumar 2017; Pellegrom 2017.)

2.8 Redux

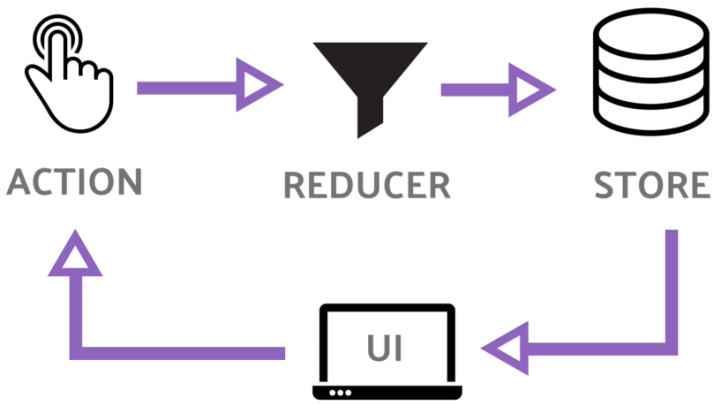
Tässä alaluvussa käsitellään Redux välipalvelua. Se on koko sovelluksen keskitetty, tilan tallentamisen ratkaisu. Sillä ei itsessään ole mitään tekemistä Reactin kanssa. Reduxia voidaan käyttää minkä tahansa Javascript-kirjaston kanssa. Mutta se toimii erityisen hyvin Reactin kanssa, koska se noudattaa sen logiikka. Reduxin tarkoitus on olla mahdollisimman pieni rajapinta tilanhallintaan. Samalla sen käytöksen tulisi olla mahdollisimman ennalta-arvattavaa.

Yksisivuiset verkkosovellukset käsittelevät paljon erilaista tietoa ja sovelluksen sisäistä tilaa (State). Tila voi olla esimerkiksi palvelimelta tuleva vastaus, välimuistissa olevaa tietoa tai paikallista tietoa jota ei ole vielä lähetetty palvelimelle tallennettavaksi. Myös käyttöliittymän pyörittäminen on monimutkaista, liikkuvia palasia on paljon. Näkymä X voi muokata komponentin Y tilaa, joka aiheuttaa näkymän Z muuttumiseen.

Isoissa ja monimutkaisissa sovelluksissa eri palaset päivittävät toisiaan ja syy-seuraussuhteiden selvittäminen voi muuttua mahdottomaksi. Tarvitaan järjestystä. Läpinäkyvään ja ymmärrettävään järjestelmään voi lisätä ominaisuuksia ja virheiden paikallistaminen on helppoa. On täysin mahdollista hävittää ymmärrys kokonaisuudesta, mitä ja missä sovelluksen tila on. Se tekee myös virheiden etsimisen ja korjaamisen lähes mahdottomaksi. (Reduxjs)

Redux on niin sanotusti sovelluksen sisäinen palvelin, jonka läpi kaikki tilan muutokset kulkevat. Kuvassa 3 on kuvaus sen toimintaperiaatteesta. Sen tarkoitus on pakottaa järjestys ja tapa tehdä asioita. Sillä on kolme periaatetta. Yksi ainoa lähde: tila sijaitsee yhdessä paikassa, eikä ole ripoteltuna ympäri sovellusta. Tilan varasto on objekti nimeltä Store. Tila on vain luettavissa, eli tilaa ei voida suoraan muuttaa. Muutokset tilaan voidaan tehdä vain funktioilla, jotka ottavat vanhan tilan ja palauttavat uuden tilan. Eli vanhaa tilaa ei muuteta. Silloin tiedetään tarkkaan mitä ja missä tapahtuu. Näistä funktioista Redux käyttää termiä Reducer.

Redux kerää tiedot omaan muistiinsa. Sovellus ei suoraan muokkaa Storea vaan sitä varten pitää luoda Action-toiminto, joka kertoo, miten muistia tulee käsitellä. Reducer kerää Storen ja Actionit ja sisältää varsinaisen logiikan, kuinka tilan muokkaaminen tapahtuu. Koko sovelluksella on vain yksi Store, jossa kaikki tilat sijaitsevat. Sovelluksen koon ja monimutkaisuuden kasvaessa tehdään uusia Reducereita, eikä uusia Storeja.



Kuva 3. Redux toimintovirtaus (Wikimedia 2018)

Redux otetaan käyttöön sijoittamalla koko React-sovellus Provider tagin sisälle. Propsina sille annetaan luotu Store-objekti. Storen sisällä oleva tila voidaan kuljettaa sovelluksen läpi propseina, tai kutsua tarvittavissa komponenteissa Reduxin apufunktioilla. (Reduxjs)

2.9 Router

Verkkosivun reititystä varten Reactissa on oma kirjastonsa, joka pitää ottaa käyttöön. Suosituin tällä hetkellä oleva on React Router. (Github 2018.) Sitä käytetään myös projektissa, jota itse teen.

React Routerin filosofia eroaa paljon muissa kirjastoissa käytetyistä tavoista. Sen sijaan että reititys määriteltäisiin sovelluksen käynnistyessä, tehdään se komponenttitasolla dynaamisesti. Router pitää huolta myös käyttäjän liikkeistä. Eteen- ja taaksepäin kulkeminen selaimessa toimii normaalisti.

Router on reititin, jonka sisälle eri reitit tulevat. Sen sisällä käytetään <Link> ja <Route> tageja URL:in ja näkymän muokkaamiseen. Link-tageilla ohjataan käyttäjä oikeaan osoitteeseen ja Route-tagi ottaa URL:n vastaan ja näyttää sen sisällä valitun komponentin. Reitit ovat siis ”vain” komponentteja. URL:sta saa sisäkkäisiä tekemällä komponentista toisen lapsi.

```
<Router>
  <Link to="/">Home</Link>
  <Link to="/about">About</Link>

  <Route exact path="/" component={Home} />
  <Route path="/about" component={About} />
</Router>
```

Lisäämällä polkuun tuplapiste: voidaan sille antaa käyttäjän toimesta parametrejä ja toimia niiden mukaan. Esimerkiksi URL-muutoksia seuraava suodatus saadaan aikaan alla olevan esimerkin mukaan.

```
<Route path="/:filter?" component={App} />
```

(React Training 2017.)

2.10 Virheiden käsittely

Reactissa on virheiden käsittelyä varten oma metodinsa componentDidCatch. Se sijoitetaan mahdollisesti virheen kohtaavan komponentin sisälle. Alla olevassa esimerkissä virheen sattuessa, komponentin tilalle näytetään vain virheilmoitus.

```
class ErrorBoundary extends React.Component {
  constructor(props) {
    super(props);
    this.state = { hasError: false };
  }

  componentDidCatch(error, info) {
    // Display fallback UI
    this.setState({ hasError: true });
    // You can also log the error to an error reporting service
    logErrorToMyService(error, info);
  }

  render() {
    if (this.state.hasError) {
      // You can render any custom fallback UI
      return <h1>Something went wrong.</h1>;
    }
    return this.props.children;
  }
}
```

Jokaisen virheen käyttöliittymässä ei tulisi kaataa koko sovellusta ja näin estää sen käyttämistä. ErrorBoundary-komponentti nappaa Javascript virheen kaikilta sen lapsikomponenteilta. Sen jälkeen komponentti kirjoittaa lokiin virheilmoituksen ja näyttää vaihtoehtoisen palan käyttöliittymää, joka sille on määritelty. ErrorBoundary saa virheet kiinni niin ensimmäisen renderöinnin aikana kuin muutosten tapahtuessa.

Jos jokin osa Reactia kohtaa virheen, koko sen alainen komponenttipuu lopettaa toimintansa. Siksi on tärkeää huolehtia virheiden käsittelystä. Järjestelmän kehittäjät päätyivät lopputulokseen, koska on parempi olla näyttämättä mitään kuin mahdollisesti korruptoitunutta tai virheellistä käyttöliittymää. Esimerkkeinä olivat viesti- ja pankkisovellus. Olisi katastrofaalista, jos virheen takia viesti lähtisi väärälle henkilölle tai pankki näyttäisi virheellistä tietoa tilin sisällöstä.
Virheiden eristäminen on myös mahdollista. Koko sovelluksen ei ole aina tarpeellista kaatua, vaikka jokin osa sitä olisikin toimintakyvytön. Facebookin Messenger verkkosivulla viestien lähettämiseen tarkoitettu käyttöliittymä on erotettu sivupalkista. Esimerkiksi sivupalkin kaatuessa, käyttäjä pystyy lähettämään viestejä. Reactin kehittäjät myös suosittelevat vielä erillistä virheiden paikallistajaa ja raportoijaa, jotta tuotannossa olevasta koodista saadaan virheilmoitukset. (Abramov 2017.)

2.11 Spa

Single page application (SPA) eli yhden sivun sovellus, on yleinen tapa tehdä websovelluksia tällä hetkellä.
Ne muistuttavat toiminallisuudeltaan enemmän natiiveja sovelluksia kuin perinteisiä verkkosivuja. SPA eroaa staattisesta sivusta siten, että uudet näkymät haetaan dynaamisesti eikä tarvita koko selainikkunan päivitystä. Sen sijaan käytössä on AJAX-tyyppisiä ratkaisuja. Suurin osa sivun renderöinnistä tapahtuu siis selaimessa, eikä palvelimella. (Codeschool 2017.)

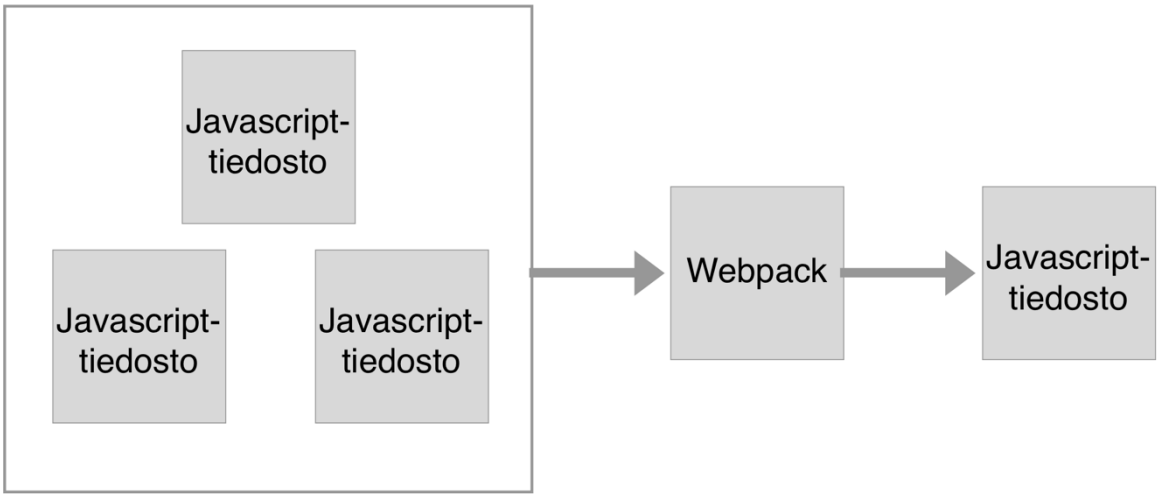
React sopii hyvin SPA-sivujen tekemiseen. Sovelluksella on pääkomponentti ja sillä on alakomponentteja. Jokainen oma sivunsa on siis komponentti. React Routeria hyödyntämällä voidaan päättää, mikä komponenteista on milloinkin esillä. Router huolehtii navigoinnista ja selaimen edes- ja takaisinnappulat toimivat sivujen välillä liikuttaessa.

Muihin kirjastoihin verrattuna Reactin ollessa vain näkymätaso, joutuu itse tekemään paljon päätöksiä ja määrityksiä. Osaavan ihmisen käsissä React on tehokas työkalu. Mutta aloitteleva kehittäjä voi joutua käyttämään enemmän aikaa kuin esimerkiksi Googlen Angular-kehikkoa käytettäessä. Se tarjoaa valmiita käytäntöjä enemmän. Vaihtoehdoista Reactille kerrottaan enemmän luvussa 3.2
(Singhaniya 2016.)

2.12 Webpack

Webpack on Javascript-moduulien paketointiin tarkoitettu työkalu (englanniksi module bundler). Sitä voidaan hyödyntää isommissa ja pienemmissä sovelluksissa yhdistämään useasta eri tiedostosta koostuva projekti yhdeksi paketiksi (bundle). Webpack rakentaa sisäisen riippuvuuskaavion, jossa ovat sovelluksen kaikki moduulit ja Webpackin lisäosat. Webpackia ajettaessa se paketoi tiedostot annettujen ohjeiden mukaan yhdeksi paketiksi. (Kuva 4.) Webpack on todella muokattavissa oleva työkalu. Sitä käytetään muokkaamalla

Webpackin konfigurointitiedostoa. Työkalun ydinkonsepteja ovat: entry, output, loaders ja plugins.



Kuva 4. Webpackin toiminta

Tiedostossa määritetään Entry eli sisääntulo. Siinä kerrotaan, mitä moduuleita Webpackin tulee käyttää sen tehdessä riippuvuuskaaviota. Output eli ulostulo kohdassa määritetään minne Webpack tuottaa paketin ja minkä nimisenä. (Webpack)

Loaderit mahdollistavat tiedostojen kääntämisen paketoinnin aikana. Esimerkiksi Babel on Webpackissä käytettävä loader, jonka avulla uuden syntaksin Javascript käännetään tukemaan myös vanhempia selaimia. Webpack on ensisijaisesti tarkoitettu Javascriptia varten, eikä se ymmärrä muita tiedostotyyppejä. Loaderit mahdollistavat sen prosessoida myös muita tyyppejä, esimerkiksi CSS-tiedostoja. Silloin paketissa on mahdollista olla kaikki sovelluksen tarvitsemat tiedostot. (Webpack) Hyvä esimerkki Webpackin eritellyistä paketeista on erottaa admin-käyttäjät tavallisista käyttäjistä. Tavallinen käyttäjä ei tarvitse kaikkia sovelluksen ominaisuuksia, joten Webpack antaa vain ne mitä tarvitaan. Siinä säästetään kaistaa, ja tehdään samalla sovelluksesta tietoturvallisempi. (Tiilikainen 2018.)

Plugins eli lisäosat mahdollistavat tiedostojen manipuloinnin paketoinnin aikana. Esimerkiksi tiedostojen koon pienentäminen tai valmiin HTML-tiedoston luominen paketin tiedostoista. (Webpack)

3 Työn toteutus

Tässä luvussa esitellään toimeksiantona saatu käyttöliittymän suunnittelu ja toteutus. Ensimmäisissä aliluvuissa käydään läpi työn lähtökohtia. Millaisia käyttötapauksia sovelluksella on ja millaisia ratkaisuja yritys on jo tehnyt. Uusi näkymä on tarkoitus toteuttaa React-kirjaston avulla, mutta sille olevat vaihtoehdot tuodaan esille.

Seuraavissa luvuissa käydään läpi varsinainen työprosessi ja kuinka toimeksiannon suorittaminen sujui. Aliluvun tarkoitus on esitellä työn eri vaiheet esimerkiksi haastattelujen lopputulosten referointi.

Viimeiset luvut esittelevät teknisiä ominaisuuksia. Huomiota saa toteutuksen kannalta olennainen kirjasto React DnD(Drag&Drop). Varsinaisesta järjestelmästä esitellään sen arkkitehtuuri, johon uusi käyttöliittymä tulee osaksi. Sen jälkeen käydään läpi valmistuneen produktin komponentit. Viimeisenä lukuna esitellään käytössä ollutta Agile-prosessia.

3.1 Kehitystyön lähtökohdat

Kehitystyön lähtökohtana toimi asiakasyritys Talented Solutionsin tarve päivitetylle käyttöliittymälle omassa järjestelmässään. Sisäinen portaali on tarkoitettu yrityksen työntekijöiden käytettäväksi työprosessin hallintaan. Kehitettävässä osiossa hallitaan yrityksen asiakkaina toimivien ohjelmistokehityksen ammattilaisten tietoja. Järjestelmässä voidaan selata ja etsiä asiakkaita, sekä muokata heidän tietojaan. Käyttöliittymä oli vain yksinkertainen lista asiakkaista ja tietojen muokkaaminen vaati erilliselle sivulle menemistä. Käyttäjä joutui klikkailemaan hiirellä paljon.

Käyttöliittymä ei myöskään tukenut nykyisiä prosesseja. Työntekijöille olisi tärkeää nähdä helposti prosessin kulku ja vanha käyttöliittymä ei visuaalisesti tuonut sitä esille. Käyttäjähallinta on myös tehty osana vanhaa järjestelmää, josta ollaan siirtymässä pois. Uuden järjestelmän tulisi vastata käytettävyyden ongelmiin ja yhtenäistää järjestelmän kokonaisarkkitehtuuria. Osa portaalista on päivitetty käyttämään Reactia, jota tässäkin uudistuksessa käytetään. (Tiilikainen 23.7.2017. & 29.30.2017.)

Reactin etuja on sen helppokäyttöisyys ja kehittäjäystävällisyys. React on helppo lisätä osaksi olemassa olevaa järjestelmää. Parhaimmillaan on, ettei vanha osa sovelluksesta ole edes tietoinen Reactista. Kirjaston oppiminen ei ole myöskään vaikeaa ohjelmistokehityksen ammattilaiselle. React on kehittäjäystävällinen. (Martin-Colby 2017.) Esimerkiksi

mahdollisuus käyttää valmiina olevia komponentteja sovelluksen sisällä. Suuri etu on aktiivinen yhteisö, joka takaa tukea ja vastauksia kysymyksiin. Käyttäjälle React näyttäytyy nopeina ja responsiivisina sivuina. (DA-14 2017.)

3.2 Vaihtoehdot

Vaihtoehtoisia kehikoita käyttöliittymän kehittämiseen löytyy paljon. React on tällä hetkellä suosituin, eivätkä trendit ennusta asian muuttumista. Kaksi toiseksi suosituinta olisivat Vue.js ja Angular 2. (Elliot 2018.) Työssäni valinta oli työpaikan, joten en voinut siihen vaikutta.

React eroaa kaikkein eniten perinteisestä HTML-syntaksista, koska on puhdasta Javascriptiä. Vue ja Angular käyttävät HTML-syntaksia ja rakentavat sen päälle omat ominaisuutensa. Ne voivat olla parempi vaihtoehto, jos käytettävässä projektissa on paljon HTML-osaavia kehittäjiä.

Vue ja React ovat hyviä yksinkertaisille, pienille komponenteille. Ne saavat sisääntulon (input) ja tuloksena syntyy esitettävä elementti (output). Ne ovat myös hyvin joustavia ja voidaan yhdistää erilaisten arkkitehtuurien kanssa sekaisin. Kuten Reactilla, myös Vuella on paljon jo valmiiksi tehtyjä npm-paketteja. Vapauden takia on myös suurempi vastuu sovelluksen arkkitehtuurin suunnittelun kanssa. Mahdollisuus mennä vikaan on suurempi ja luoda arkkitehtuuri, joka ei mahdollisesti toimi enää jatkossa. Angular on paljon tiukempi sen suhteen ja ongelmia on vaikeampi aiheuttaa väärillä valinnoilla.

Vue on nopeasti kasvava Javascript-kehikko. Se on tehty nopeaksi ja mahdollisimman pieneksi. Vue voidaan ottaa käyttöön vähitellen pala kerrallaan. Sitä kehittää Evan You ja hänen muutaman hengen ryhmänsä. Kehitystiimin kokoon nähden Vue on kasvattanut suosiotaan nopeasti. Vue käyttää yhden tiedoston komponenttirakennetta. Siinä kaikki komponentin tiedot ovat yhdessä tiedostossa ja ne erotellaan omilla tageilla. (Neuhaus 2017.)

Angular 2 on Googlen kehittämä käyttöliittymäalusta. Se käyttää puhtaan Javascriptin sijaan TypeScript-kieltä. Angular 2 on kokonaan uudelleenkirjoitettu, eikä muistuta alkuperäistä AngularJS:ää. Kehikon etuna on sen tarkka rakenne. Uusi työntekijä, joka on jo tuttu sen kanssa, oppii nopeasti, kuinka uudessa paikassa sitä käytetään. React sen sijaan vaatii aina opettelua, koska jokaisessa paikassa voidaan tehdä aivan erilaisia ratkaisuja rakenteen kanssa. Angular käyttää TypeScriptä kuten aiemmin mainittiin. Se tarjoaa

lisää ennustettavuutta, koska Reactissa voidaan käyttää niin ES5 kuin ES6 mukaista syntaksia. TypeScript tarjoaa myös lisää käsitteitä esimerkiksi staattiset tyypit. Ne helpottavat automaattista koodin kanssa työskentelyä, koska tavallinen Javascript on todella kevyt tyyppien suhteen. Vahvan tyypityksen pitäisi myös vähentää ohjelmointivirheitä. TypeScriptin vaarana on, että häviää tai vähenee käytössä koska ei ole virallinen Javascript. Angularin vaikeutena on sen oma kielensä ja syntaksinsa. Toiminnallisesti Angularissa laitetaan tavallisen HTML sekaan Angularin tageja. React puolestaan on vain Javascriptiä, eikä vaadi uutta tapaa kirjoittaa koodia. (Neuhaus 2017.)

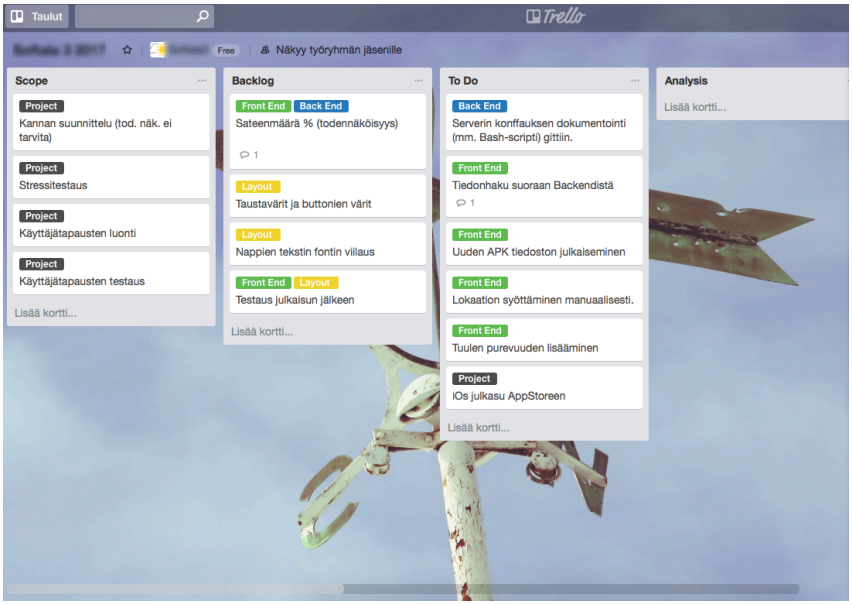
3.3 Työprosessin kulku

Kehitystyö aloitettiin joulukuussa 2017 tehdyillä käyttäjähaastatteluilla (liite 1). Haastattelujen tarkoitus oli muodostaa ymmärrys vanhan käyttöliittymän ongelmakohdista. Samalla selvitettiin sen käyttäjien työprosessi, jota uuden käyttöliittymän tulisi tukea. Haastattelut onnistuivat hyvin, koska työntekijöillä oli selvät mielipiteet suurimmista ongelmista. Ehdotuksia suunnittelutyön lähtökohdiksi tuli myös paljon. Työntekijät olivat sanavalmiita ja sitoutuivat osaksi kehitystyötä alusta asti.

Tammikuussa 2018 aloitettiin varsinainen kehitys. Aikaa tekemiselle oli varattu noin kuukausi, joka on todella lyhyt aika ohjelmistokehitykselle. Menetelmänä käytettiin Lean-mallia, jossa asetettiin viikkokohtaiset tavoitteet. Viikon lopuksi tavoitteet käytiin läpi ja asetettiin seuraavan viikon tavoitteet. Mukana prosessissa olivat tulevat käyttäjät, jotka pääsivät antamaan kommentteja koko prosessin aikana. Kommentit muokkasivat tulevia päätöksiä, koska ilman niitä moni asia olisi jäänyt ilman huomiota.

Uusi näkymä käyttöliittymään tehtiin Reactilla, joten vanhasta HAML-näkymästä ei voinut käyttää mitään uudelleen. Keskustelimme asiasta portaalin kanssa työskentelevien kanssa ja tulimme siihen johtopäätökseen. Ainoastaan portaalinlaajuiset valikkoelementit jäivät ennalleen. Teknisten päätösten jälkeen kävimme yhdessä visuaalisen suunnittelijan kanssa läpi haastattelujen ideoita. (liite 1.) Paras ehdotus oli Trello-verkkopalvelun tapainen kolumninäkymä eli kanban-taulu. Kuten kuvassa 5 näkyy. Perusteluna sille oli käyttäjien tottumus ja osaaminen sen käyttämisessä. Heidän ei tarvitse opetella uutta. Uudessa näkymässä jokainen talentti olisi oma korttinsa kolumnissa ja niitä voisi siirrellä raahaamalla. Jokainen kolumni olisi talenteille annettava tila, jossa he ovat Talentedin prosessissa. Se mahdollistaa visuaalisesti sen esittämisen, jos jossain kohtaa prosessia nousee pullonkauloja. Myös ylimääräiset painallukset tehtävien suorittamista varten vähenevät. Lisätietoa antavan ikkunan kanssa oli pohdintaa käyttäjien kanssa. Mikä on heille työn-

kannalta olennaista ja tärkeää. Esimerkiksi talentin kulkua prosessissa kuvaavan aikajanan merkitystä ei alkuperäisessä suunnitelmassa tiedetty. Käyttäjähaastattelut nostivat sen tärkeyttä, joten siitä tehtiin tärkeämpi osa ikkunan käyttöliittymää.



Kuva 5. Trello-verkkosovellus

Suunnittelun jälkeen aloitettiin kehitystyö. Valmiina olevan konfiguraatio mahdollisti nopean aloituksen. Projekti sijaitsee yrityksen Github-sivulla, josta latasin sen. Uusi sivu piti vain liittää osaksi vanhoja määrityksiä. Työprosessi eteni siten, että yhdessä työpaikan ohjaajan kanssa kävimme läpi tavoitteet, joita tulisi suorittaa. Niiden valmistuttua kävimme läpi valmistuneen koodin ja mahdolliset korjaukset. Arvioiden välissä tulevilta käyttäjiltä kysyttiin jatkuvasti palautetta ja kehitysehdotuksia. Kehitystyön edetessä tarkentui myös, minkälainen lopputulos on mahdollista saada valmiiksi työjakson aikana.

3.4 Sovelluksen rakenne

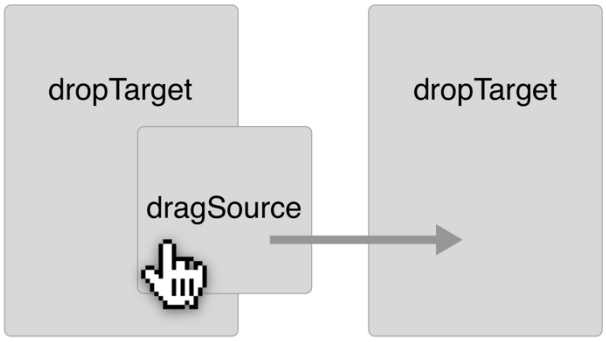
Portaalin taustajärjestelmä on toteutettu Ruby on Rails teknologialla. Sivujen piirtoon käytetään pääasiassa Rubyn HAML-mallia. Siinä verkkosivujen piirto suoritetaan palvelimella ja käyttäjälle palautetaan valmis sivu. Osa kolmansille osapuolille näytettävistä sivuista oli tehty jo Reactilla. Se mahdollisti komponenttien uudelleenkäytön. Kyse oli vain muutamasta osasta käyttöliittymää. Esimerkiksi talentit antavat arvion omasta osaamisestaan numeroin ja sain valmiina sen tiedon näyttävän komponentin, jonka pystyin liittämään sellaisenaan osaksi omaa sivuani. Suurin valmis komponentti oli talenttien suodattamiseen tarkoitettu osio. Sen ulkoasun sain valmiina, minun tehtävä oli ainoastaan hyödyntää sen tarjoamaa logiikka komponenteissa.

Redux konfiguraatio oli tehty valmiiksi. Konfiguraatio ei ollut aivan perinteinen vaan vähän yksinkertaistettu duck-redux. Siinä usean eri tiedoston sijaan Actionit ja Reducerit ovat samalla tiedostolla. Reduxin kanssa tehtäväni oli kirjoittaa komennot duck-redux mallin mukaan.

Ruby on Rails ja React yhdistetään käyttämällä Rubylle kirjoitettua kirjastoa React on Rails. Sen avulla voidaan liittää helposti olemassa olevaan Ruby-sovellukseen React-komponentteja. Kirjasto mahdollistaa esimerkiksi Rubyn muuttujien antamisen propseina Reactille. Minun tehtäväni oli keskittyä käyttöliittymään, joten en osallistunut taustajärjestelmän kehittämiseen.

3.5 React DnD

Tässä luvussa käydään läpi kehitystyössä käytössä ollutta kirjastoa React DnD (Drag and drop eli suomeksi raahaa ja pudota). Sen avulla oli mahdollista helposti tehdä komponenteista raahattavia ja raahauksia vastaanottavia säiliöitä (englanniksi container). Säiliöihin komponentti voidaan pudottaa käyttäjän toimesta. Yksinkertaistaen komponentille määritellään, onko se pudotettava vai pudottamista vastaanottava säiliö. Sen jälkeen kuvan 6 mukainen toiminta on mahdollista. Kirjasto on todella monipuolinen ja sen ominaisuuksista tarvittiin vain osa.

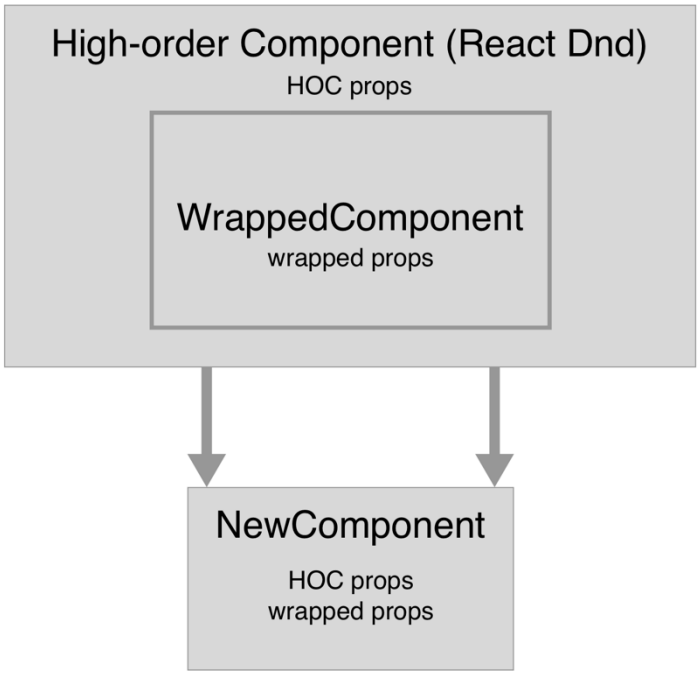


Kuva 6. Käyttäjän suorittama raahaus osoittimella.

React DnD käyttää hyväkseen Reactin ylemmän tason komponentteja (higher-order component, HOC). Ylemmän tason komponentit ovat funktioita, jotka ottavat sisäänsä toisen komponentin, kuten alla olevassa esimerkissä, sekä kuvassa 7. Ne eivät palauta kuvausta käyttöliittymästä vaan luovat uuden komponentin.

```
const EnhancedComponent = higherOrderComponent(WrappedComponent);
```

(Reactjs). Uusi palautettava komponentti saa alkuperäisen lisäksi ylemmän tason komponentin mukaisia lisäyksiä. Esimerkiksi sen propsit tulevat sille. React Dnd:n tapauksessa se mahdollistaa komponenttien raahauksen ja pudottamisen. Sekä muutokseen reagoimisen ohjelmallisesti.



Kuva 7. Ylemmän tason komponentin käyttäminen.

Kirjasto käyttää selainten sisäistä Drag and Drop -rajapintaa. Sen implementoinnit vaihtelevat, joten React DnD tarjoaa Reactin tapaisen abstraktion. Kehittäjän ei tarvitse miettiä selaintukea, vaan voi keskittyä koodin kirjoittamiseen. Selaimen rajapinnan käytössä on kuitenkin etuna sen tuomat ominaisuudet. Esimerkiksi tietokoneella olevien tiedostojen pudottaminen voidaan hoitaa ainoastaan selaimen rajapinnan kautta. Suurimpana haittana on, ettei se tue kosketusnäyttöjä. Kirjasto tarjoaa kuitenkin kehittäjälle mahdollisuuden kirjoittaa itse oman rajapintasa, jos sille on tarvetta.

Kirjaston peruskäsitteitä ovat datatyypit item ja type. Ne ovat Javascript-objekteja komponentin sisällä. Item on yksittäinen raahattava komponentti ja type, on kokoelma itemejä. Kirjasto käyttää datatyyppejä sisäisesti toimintojen suorittamiseksi. React DnD ei välitä komponenttien ulkoasusta, vaan pystyy pelkästään datatyyppien perusteella suorittamaan toiminnot. Eli kirjasto ei siis käsittele varsinaisia DOM-elementtejä. Pelkästään Javascript-objekteja. Ideologia on Reduxin mukainen, jota kirjasto käyttääkin sisäisesti. Idea toimii käytännössä hyvin, koska raahaaminen on binäärinen tila. Komponenttia joko raahataan tai ei raahata. Typet ovat hyödyllisiä koska sovelluksen koon kasvaessa, halutaan mahdollisesti lisätä raahattavia elementtejä. Typejen avulla voidaan rajata, mitkä raahattavat elementit sopivat millekin säiliölle.

Pelkkien datatyyppien määrittäminen ei riitä. React DnD:n aputoiminnoilla monitors ja connectors otetaan kirjaston ominaisuudet käyttöön. Monitor on kirjaston sisäinen ominaisuus, joka seuraa tilan muutoksia. Alla olevassa esimerkissä collect-funktio ottaa vastaansa monitor-objektin. React DnD HOC-ominaisuuden avulla propsit saadaan injektoitua komponenttiin käytettäväksi. Kirjasto huolehtii suorituskyvystä ja funktion kutsumisesta oikeaan aikaan käyttäjän tekemien muutosten mukaan.

```
function collect(connect, monitor) {
  return {
    highlighted: monitor.canDrop(),
    hovered: monitor.isOver()
  };
}
```

Connectoreita käytetään määrittämään komponentti olemaan joko raahattava tai pudotuksia vastaanottava container. Connector-funktio on komponentin palautusarvo, jonka sisälle varsinainen renderöitävä JSX sijoitetaan. Alla olevassa esimerkissä elementistä saadaan tehtyä pudotuksia vastaanottava alue.

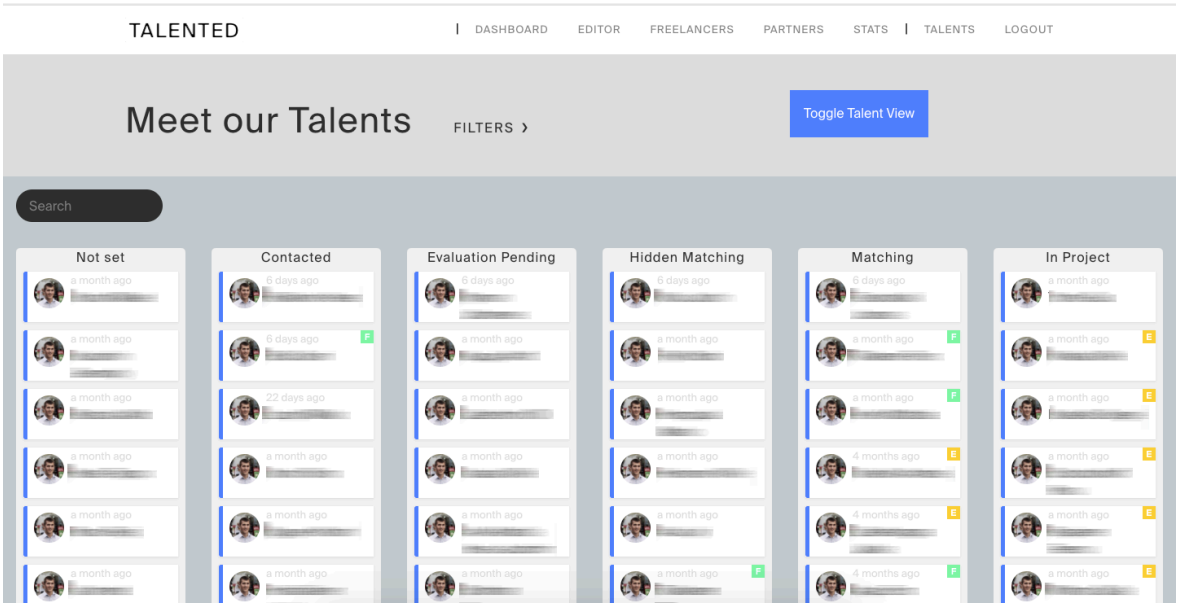
```
function collect(connect, monitor) {
  return {
    connectDropTarget: connect.dropTarget(),
  };
}

render() {
  const { connectDropTarget } = this.props;

  return connectDropTarget(
    <div>
      DropTarget
    </div>
  );
}
export default DropTarget(Vastaanotettavan komponentin tyyppi, col-
lect)(Komponentin tyyppi)
```

Komponentin export-toiminnossa kerrotaan minkätyyppisiä komponentteja dropTarget ottaa vastaan. Collectin mukana menevät komponenttiin injektoitavat propsit ja lopuksi kerrotaan, minkätyyppinen komponentti on itsessään. Raahattava komponentti on hyvin samankaltainen. Siinä DropTarget määritysten sijaan käytetään DragSource-määritystä.

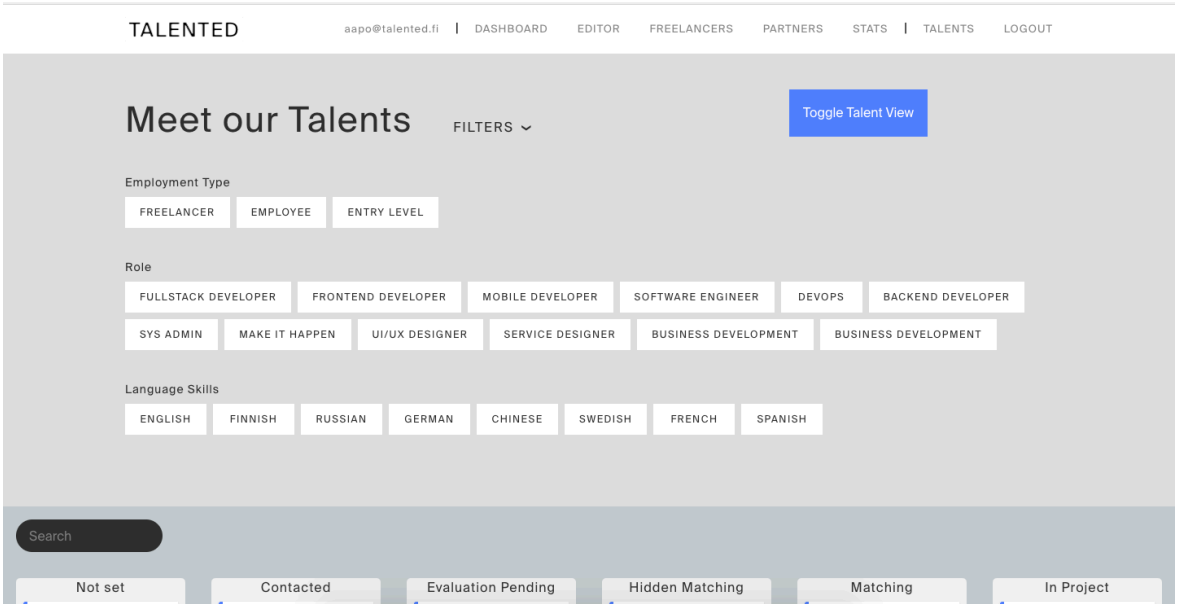
Kirjasto tarjoaa monipuolisen valikoiman elinkaarimetodeja raahauksen ja pudotuksen ajalle. Niissä ohjelmoija pystyy tarkasti muokkaamaan, mitä sovelluksessa tapahtuu raahauksen aikana, ja komponentin pudottamisen jälkeen.

3.6 Komponentit

Tässä luvussa esitellään uusi käyttöliittymää, sekä sitä varten tehdyt komponentit ja niiden toiminallisuudet. Alla olevissa kuvissa 8 ja 9 nähdään valmis käyttöliittymä.

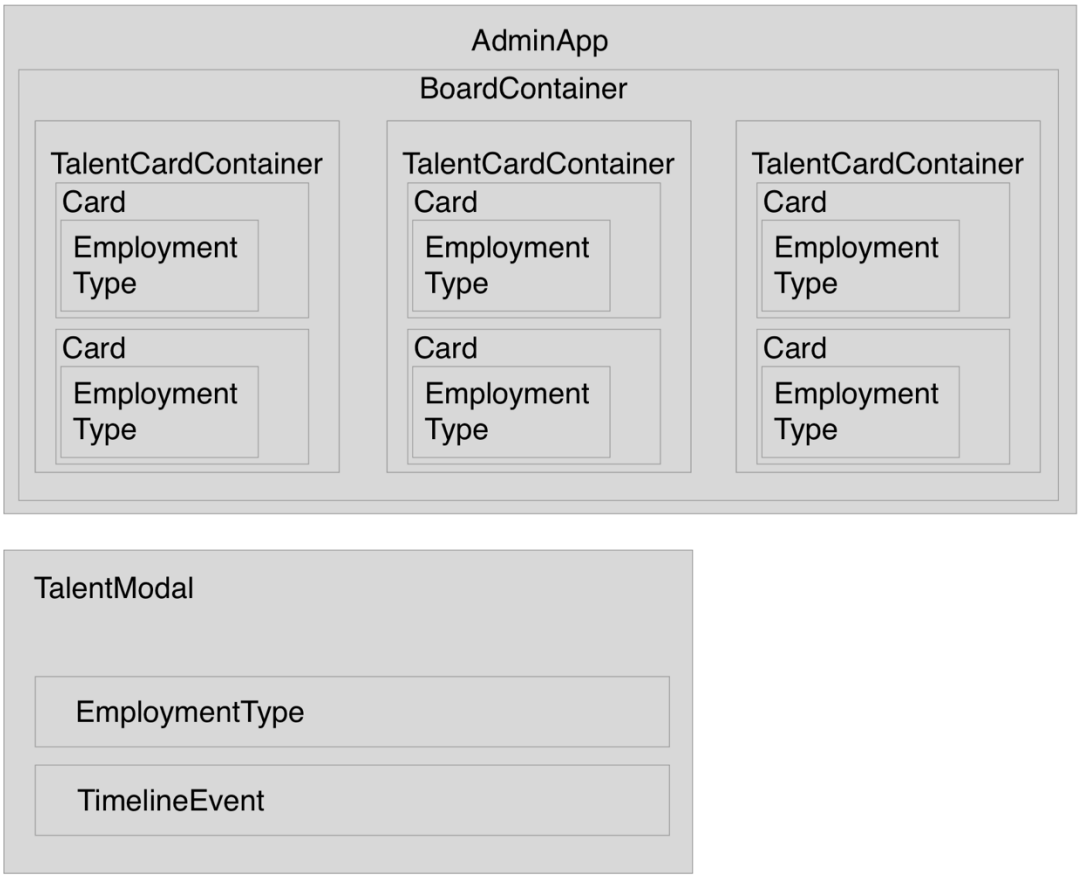


Kuva 8. Kanban-taulu talenteille.



Kuva 9. Suodatinvalikko avattuna.

Sovellus on jaettu kahteen erilaiseen säiliökomponenttiin. BoardContainer ottaa sisäänsä koko sovelluksen ja TalentCardContainer ottaa sisäänsä talenttien tiedot sisältävät kortit. Tarkemmin sovelluksen arkkitehtuuri on kuvattu kuvassa 10. Korttien lisäksi sovelluksessa on TalentModal ikkuna, joka avataan painamalla korttia. Se lataa kyseisen talentin tiedot ikkunaan. (Kuva 11.) Seuraavaksi käydään läpi jokaisen komponentin toiminnallisuudet.



Kuva 10. Sovelluksen rautalankamalli.

Ensimmäinen komponentti on AdminApp, mutta se on tarkoitettu vain React on Rails määrityksiä varten. Boardcontainer on ensimmäinen näytettävä osa sovellusta. Siitä alkaa käyttöliittymää piirtävien komponenttien purku. Ensimmäisenä ovat hakutoiminnot sisältävä palkki ylhäällä. Jossa on ladattu valmiina oleva suodatusominaisuus (Filters). Sen avulla käyttäjä voi valmiina olleiden hakusanojen avulla rajata näytettävät talentit. Sen vieressä ovat tekstihaku ja nappula, jolla tuodaan esiin piilotetut talentit. Tekstihaku toimii valmiilla kirjastolla react-search-input. (enkidevs 2015) Se toimii nimihakuna ja etsii kaikki käyttäjät, joiden nimissä on kirjoitettu tekstikappale.

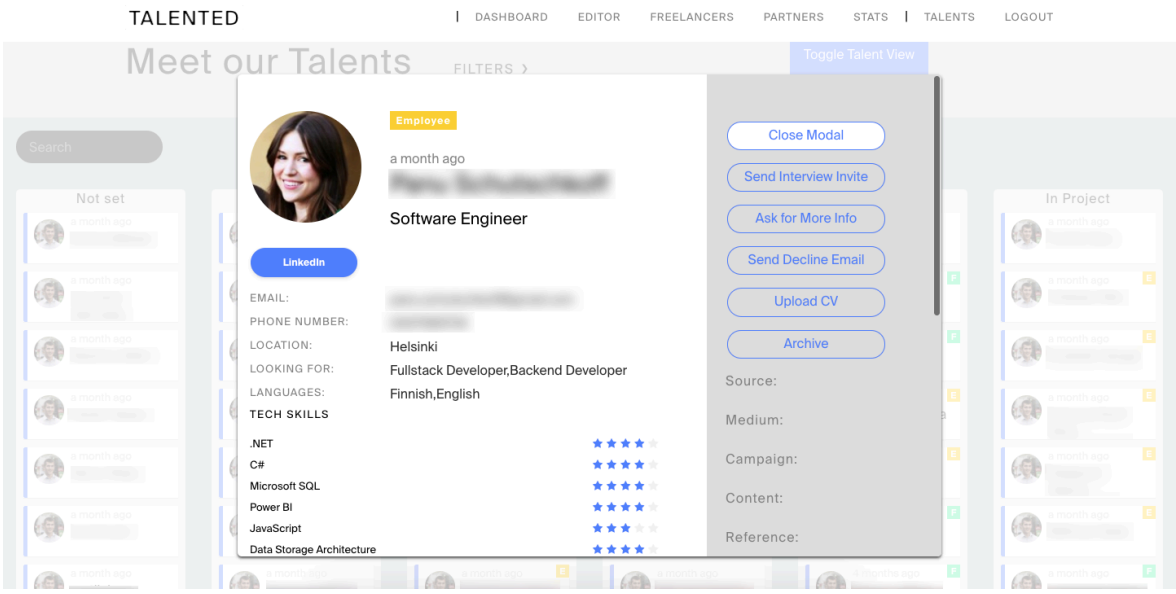
Hakupalkin alapuolella renderöidään kolumnisäiliö ja niiden sisälle kolumnit komponentin apumetodeilla. Jokaiselle kolumnille annetaan kunkin tilan mukaiset talentit aikajärjestyksessä, sen mukaan milloin niitä on muokattu.

TalentCardContainer-komponentti huolehtii käyttäjien näyttämisestä. Siinä on myös React DnD -kirjaston mukaisia metodeja. Ne huolehtivat, että kolumnit ovat pudotettavia alueita joihin raahaus ja pudotus onnistuu. Sekä metodi, joka huolehtii toiminnoista, kun käyttäjä

raahaa kortin kolumnista toiseen. Toiminto on Redux Action, joka kommunikoi tietokannan kanssa ja tekee tarvittavat muutokset. Reducer palauttaa uuden staten, jossa päivitetty tieto korteista oikeissa kolumneissa.

Jokaisella kolumnilla on status eli tila. Sen perusteella jokainen TalentCardContainer saa propseina näytettävät talentit. Käyttäjän mahdolliset suodattamiset poistavat näytettävien talenttien määrää. Komponentin saamat talentit renderöidään jokainen oman Card-komponentin sisälle käyttämällä map-funktiota.

Card-komponentti on kolumnissa sijaitseva talentti. Jokaiselle talentille on oma korttinsa. Sen ainoa state on tarkoitettu modaalin avaamista varten. Propseina se saa oman talenttinsa. Komponentti käyttää apufunktiota, joka sijaitsee omassa polussaan. Se laskee milloin talenttia on päivitetty viimeksi. Tieto on tärkeä, jotta järjestelmän käyttäjät näkevät nopeasti, milloin ketäkin on päivitetty. Komponentti palauttaa kortin, jossa on talentin nimi, kuva, milloin talenttia on viimeksi päivitetty sekä raita joka värillään erittelee talenttien osaamista. Tärkeä funktio on modaalin avaaminen, joka tapahtuu onClick metodilla korttia painettaessa. Se luo uuden modaalin, kuten kuvassa 11 näkyy. Se saa propseina tiedot talentista ja funktion, jolla modaali sulkeutuu.



kuva 11. Lisätietoikkuna

Talentmodal komponentti sisältää kaikista eniten tietoa. Koska kaikkia talentin tietoja ei ladata järjestelmän käynnistyessä ladataan ne vasta kun modaali avataan. Tiedot ladataan Redux storeen, ja tiedot näytetään modaalissa, kun ne ovat latautuneet. Tässä ensimmäisessä versiossa tiedot vain ilmestyvät latauksen valmistuessa tietokannasta. Jatkokehityksessä voisi esimerkiksi olla latausanimaatio indikoimassa, että lisää tietoa on tulossa.

Komponentti on täynnä erilaisia apumetodeja, joilla tiedot näytetään. Näin render-funktio näyttää siistimmältä. Tämä oli työpaikan ohjaajan neuvo. Kun JSX:n sisällä on mahdollisimman vähän Javascriptiä, on se paljon luettavampaa.

Modaalin sisällä näytetään tarkasti erilaisia tietoja talentista. Esimerkiksi yhteystiedot ja tarkka kuvaus osaamisesta. Modaalia rakentaessa, tuli hyvin esiin Reactin hyvät puolet. Pystyin käyttämään olemassa olevaa komponenttia, joka näyttää teknisen osaamisen. Komponenttia ei tarvinnut erikseen muokata sopimaan. Oli mahdollista liittää se sellaisenaan omaan koodiini. Riittävän geneeriset komponentit ovat siis todella hyödyllisiä.

Monimutkaisin osa komponenttia on aikajana, jossa näytetään kaikki talentille tehdyt muutokset ja yrityksen työntekijöiden kirjoittamat kommentit. Tapahtumadata oli todella epäjärjestelmällistä, joten sen parsiminen näytettävään muotoon vei aikaa. Kyseinen tehtävä jäi keskeneräiseksi opinnäytetyön puitteissa.

Aikajanan tueksi täytyi tehdä uusi Redux-komento, jolla käyttäjän kirjoittama viesti tallentuu tietokantaan. Osa aikajanasta on omassa komponentissa, mikä siistii modaalin koodia. Komponentti palauttaa ajan, milloin muutos tehty, muutoksen tehneen käyttäjä nimen, sekä käyttäjän viestin. Jos tapahtuma on tietojen muutos eikä kommentti, näytetään tehdyt muutokset.

Pienin komponentti on käyttöliittymäelementti, jolla näytetään, haluaako talentti olla freelancer vai palkattu työntekijä. Tieto näytetään sekä kolumninäkymässä, että modaalissa. Erona on vain tekstin pituus, joten samaa koodia voi kierrättää. Tämä komponentti saa tiedon, kumpi komponentti sitä kutsuu ja palauttaa sen perusteella pidemmän tai lyhyemmän tekstin. CSS-tyyli toimii molemmilla, eikä sen tarvitse olla erillinen näkymille.

Komponenttien tyylit tehtiin suunnittelijan kanssa tehdyn suunnitelman mukaisesti. React tarjoaa erilaisia ratkaisuja tyylien toteuttamiseen. On mahdollista käyttää erillisiä tyylitiedostoja tai kirjoittaa esimerkiksi tyylit tiedostojen sisään (Reactjs). Yrityksen käytössä oli erilliset CSS-tyylitiedostot. CSS-syntaksia oli jatkettu ottamalla käyttöön SASS. SASS on syntaksijatke, joka tuo tyylien kirjoittamiseen lisää ominaisuuksia ja yksinkertaistaa koodia. Esimerkiksi tarjoamalla mahdollisuuden käyttää muuttujia. Koodi on yksinkertaisempaa, koska aaltosulkuja ei SASS:ia käyttämällä tarvitse (Sass-Lang). Suurimman osan tyylittelyistä jouduin kirjoittamaan itse, mutta yrityksellä oli valmiina esimerkiksi marginaalien kokoja sekä käytössä olevia värejä. Sivuston asettelu on tehty Baseguide-kehikkoa käyttäen. (Baseguide)

Se muistuttaa yleisesti käytössä olevaa Bootstrap-kehikkoa muotoilultaan ja käytöltään. Siitä on karsittu vain ylimääräisiä ominaisuuksia ja tarjoaa lähinnä asetteluun pohjautuvia määrityksiä.

3.7 Agile käytännössä

Tässä luvussa käydään läpi teoriaa Lean-filosofiasta, joka oli käytössä työskentelyn aikana. Filosofialle on useita määritelmiä ja se tarkoittaa usein montaa eri asiaa. Lähimpänä omaa tekemistäni on Lean UX -kirja. (Gothelf & Seiden 2016.) Siinä keskeistä on tehdä käyttäjätutkimus yhteistyönä. Nopeita päätöksiä ja muutoksia tehdessä itse tehty tutkimus on nopeampaa ja tuottaa parempia tuloksia kuin ulkopuolisen firman teettämä tutkimus.

Suoritettava tutkimus on myös jatkuvaa. Optimaalisessa tilanteessa tutkimus ja uudet versiot kulkevat samassa syklissä ja saatava palaute on jatkuvaa. Kerätystä tiedosta tulisi kerätä toistuvia kaavoja. Myös poikkeukset kerätään ja niitä voidaan hyödyntää tulevissa tutkimuksissa. Tulos pitää lopuksi varmistaa useasta lähteestä.

Ensimmäisen tutkimuksen jälkeen kehitystyö tulee aloittaa oletuksista. Oletusten perusteella voidaan kirjoittaa testattavia hypoteeseja. Gothelf ja Seiden jakavat oletukset neljään luokkaan. Liiketoiminnallinen tulos on mitattava ja muuttuva liiketoiminnallinen asia. Oletukset käyttäjistä eli ketkä ovat kehitystyön lopputuloksen käyttäjiä. Sekä oletukset käyttäjän haluamasta lopputuloksesta hänen käyttäessä tuotetta tai palvelua. Lopuksi oletukset ominaisuuksista eli mitä tarvitaan lisää, jotta käyttäjän haluama lopputulos toteutuisi mahdollisimman hyvin.

4 Lopputulos

Tässä luvussa tarkastellaan kehitystyön päätteeksi valmistunutta käyttöliittymää, sekä kehitysehdotuksia tulevan varalle. Lopuksi opinnäytetyön tekijän pohdintaa omasta oppimisesta ja prosessin kulusta.

Valmistunut produkti oli odotetun mukainen. Kuukauden mittainen kehitysaika on todella vähän sovelluskehityksessä. Alkuvaiheessa aikaa kuluu järjestelmään tutustumiseen, että tiedetään, mitä ollaan tekemässä. Vastan sen jälkeen voidaan aloittaa varsinainen työskentely. Uusien asioiden opettelu vie oman aikansa. Vastaan tuli useita uusia kirjastoja. React oli tuttu kirjasto jossain määrin. Sen käyttäminen oikeassa projektissa tarkoitti kuitenkin sen mahdollisuuksien ja rajoitteiden opettelua. Työnantaja oli tyytyväinen suoritettuun työhön. Se mistä päätimme kehitystyön alkaessa, saatiin suoritettua. Käyttöliittymä muistutti suunniteltua ja sen perustoiminnot toimivat.

Kehitystyön päätyttyä suoritettiin loppuhaastattelut, jossa kysyttiin työntekijöiden kokemuksia prosessista ja työn lopputuloksesta. (liite 1.) Kaikki käyttäjät olivat todella tyytyväisiä uuteen käyttöliittymän. He eivät malttaneet odottaa päästä käyttämään sitä vanhan sijasta. Se oli nopea ja ymmärrettävä. Koko prosessi sai kiitosta. Työntekijät pidettiin koko ajan mukana muutoksista ja kysyttiin kun parhaasta mahdollisesta vaihtoehdosta oli epävarmuutta. Palautteessa mainittiin sen tuovan omistajuuden tunteen sovellukseen.

Kaiken kaikkiaan työn suorittaminen sujui jouhevasti. Vaatimusten määrittäminen aivan ensimmäisenä varmisti, että kehitys kohdistuu oikeisiin ongelmiin. Kehitystyössä osattiin jakaa tehtävät sopivan kokoisiksi paloiksi. Aikataulussa pysyminen kertoo siitä.

4.1 Kehitettävää

Keskeisimmät kesken jääneet asiat olivat talenttien tietojen muokkaaminen lisätietoikkunassa, sekä kuvien hakeminen tietokannasta. Tällä hetkellä kaikilla käyttäjillä on sama kuva, jotta käyttöliittymän testaaminen onnistuu. Aloitin myös sovelluksen purkamisen pienempiin komponentteihin. Työ jäi kesken ja osa komponenteista voisi olla pienempiä. Se tekisi koodista luettavampaa. Se olisi myös paremmin Reactin ideologian mukaista ja mahdollistaisi komponenttien mahdollisen uudelleenkäytön tulevaisuudessa.

Tulosten mitattavuutta voisi parantaa tekemällä haastattelut standardoidusti. Kehityksen seuraaminen ilman selviä mittareita ei ole tieteellisesti mahdollista. Opinnäytetyössä ollut

otanta oli myös riittämätön oikeiden johtopäätösten tekemiselle. Toimeksiantajana toimivan yrityksen sisäiseen kehitykseen se toimii, mutta yleisiä johtopäätöksiä ei ole mahdollista tehdä tämän opinnäytetyön pohjalta. (Six & Macefield 2016.)

4.2 Pohdintaa

Opinnäytetyö tarjosi paljon opittavaa ja sain siitä sen takia paljon irti. Pääsin näkemään käytännössä, mitä oikean yrityksen kehitys- ja tuotantoympäristö ovat. Pienessä työyhteisössä ollaan todella läheisiä, mikä piti asioiden kyselyn matalalla kynnyksellä. Se mahdollisti ominaisuuksia tehdessä, että pystyin kysymään helposti tulevilta käyttäjiltä mielipidettä.

Tekninen osaamiseni kehittyi hyvin. On aivan erilaista tehdä, joka viikko samaa asiaa. Toisin kuin koulussa. Koulussa tehdään korkeintaan kahdesti viikossa samaa projektia. Tukenani ollut ohjaaja töissä mahdollisti oppimisen. Sain apua aina pyydettäessä, mikä esti kaiken pysäyttävien ongelmien aiheuttaman hidastelun.

Opinnäytetyö prosessina oli haastava. Viikkokohtainen tekemisen seuraaminen tuotti vaikeuksia. Saatoin olla useamman viikon tekemättä mitään ja jatkaa taas sen jälkeen. Alkuperäinen aikatauluni oli saada työ valmiiksi nopeammin, mutta henkilökohtaiset syyt hidastivat valmistumista. Oman ajankäytön hallinta on selkein kehitettävä elementti omassa tekemisessäni. Ohjaajan kanssa käydyt keskustelut olivat kaikkein hedelmällisimpiä. Niissä valmistui hyvä kehikko, jonka päälle opinnäytetyö pohjautuu. Yhteen aiheeseen paneutuminen pakotti kuitenkin oppimaan. Moni Reactin konsepti, joka on ollut vähän heikommalla tasolla, on nyt paljon ymmärrettävämpi.

Lähteet

AAMINE1965. 2018. Ngrx-redux-pattern-diagram. Luettavissa: https://commons.wikimedia.org/wiki/File:Ngrx-redux-pattern-diagram.png. Luettu: 6.5.2018.

Abramov, D. 2017. Error Handling in React 16. Luettavissa: https://reactjs.org/blog/2017/07/26/error-handling-in-react-16.html. Luettu: 13.1.2018.

Krajka, B. 2015. The difference between virtual DOM and DOM. Luettavissa: http://reactkungfu.com/2015/10/the-difference-between-virtual-dom-and-dom/. Luettu: 14.1.2018.

Baseguide. Luettavissa: https://basegui.de. Luettavissa: 8.5.2018

Birder Eriksson 2012. DOM-model. Luettavissa: https://commons.wikimedia.org/wiki/File:DOM-model.svg. Luettu: 8.4.2018.

CodeSchool. Single-page Applications. Luettavissa: https://www.codeschool.com/beginners-guide-to-web-development/single-page-applications. Luettu: 13.1.2018.

Dawson, C. 2014. Javascript's History and How it Led To ReactJS. Luettavissa: https://thenewstack.io/javascripts-history-and-how-it-led-to-reactjs. Luettu: 12.12.2017.

DA-14. 2017. Top 10 Advantages of Using React. Luettavissa: https://da-14.com/blog/its-high-time-reactjs-ten-reasons-give-it-try. Luettu: 11.5.2018.

Ecma International. Ecma-262. Luettavissa: http://www.ecma-international.org/publications/standards/Ecma-262-arch.htm. Luettavissa: 22.8.2018

Elliot, E. 2017. Top JavaScript Libraries & Tech to Learn in 2018. Luettavissa: https://medium.com/javascript-scene/top-javascript-libraries-tech-to-learn-in-2018-c38028e028e6. Luettavissa: 24.4.2018.

Fischer, L. 2017. React for Real Front-End Code, Untangled. Pragmatic Bookshelf.

Fisher, Bill. 2015. How was the idea to develop React conceived and how many people worked on developing it and implementing it at Facebook? Luettavissa: https://www.quora.com/React-JS-Library/How-was-the-idea-to-develop- React-conceived-

and-how-many-people-worked-on-developing-it-and-implemen- ting-it-at-Facebook/answer/Bill-Fisher-17. Luettu 25.11.2017.

Franklin, J. 2014. An introduction to ES6 classes. Luettavissa: https://javascriptplayground.com/introduction-to-es6-classes-tutorial/. Luettu: 14.1.2018.

Future of Javascript in 2017 and beyond. 2017. Luettavissa: https://multivers3d.fr/constructor/users/51184/users/verteAzur/future-of-javascript-in-2017-and-beyond.pdf. Luettu: 12.12.2017.

Gasimzada, G. 2017. What are NPM, Yarn, Babel and Webpack and how to properly use them. Luettavissa: https://medium.com/front-end-hacking/what-are-npm-yarn-babel-and-webpack-and-how-to-properly-use-them-d835a758f987. Luettu: 11.5.2018.

Gothelf, J. & Seiden, J. 2016. Lean UX. O'Reilly. Sebastopol.

Jakemmarsh. 2015. What is NPM and why do I need it. Luettavissa: https://stackoverflow.com/questions/31930370/what-is-npm-and-why-do-i-need-it. Luettavissa: 2.5.2018

JBallin. 2017. Var, let or const. Luettavissa: https://hackernoon.com/js-var-let-or-const-67e51dbb716f. Luettu: 14.5.2018.

Kirupa 2017. Creating a Single-Page App in React using React Router. Luettavissa: https://www.kirupa.com/react/creating_single_page_app_react_using_react_router.htm. Luettu: 24.1.2018.

Laverdet, M. 2010. XHP: A New Way to Write PHP. Luettavissa: https://www.facebook.com/notes/facebook-engineering/xhp-a-new-way-to-write- php/294003943919. Luettu 25.11.2017.

Ledion Spaho 2015. What's the difference between a library and a framework? Luettavissa: https://www.linkedin.com/pulse/whats-difference-between-library-framework-ledion-spaho/. Luettu: 26.3.2018.

Neuhaus, J. 2017. Angular vs. React vs. Vue: A 2017 comparison. Luettavissa: https://medium.com/unicorn-supplies/angular-vs-react-vs-vue-a-2017-comparison-c5c52d620176. Luettu: 16.3.2018.

Netflix. 2015. Netflix Likes React. Luettavissa: https://medium.com/netflix-techblog/netflix-likes-react-509675426db. Luettu: 6.5.2018.

Neves, A. 2017. What to expect from JavaScript ES2017—The Async Edition. Luettavissa: https://medium.com/komenco/what-to-expect-from-javascript-es2017-the-async-edition-618e28819711. Luettu: 13.1.2018.

NPM. What is NPM. Luettavissa: https://docs.npmjs.com/getting-started/what-is-npm. Luettavissa: 27.4.2018.

Madhankumar. 2017. Using Npm as a build tool. Luettavissa: https://scotch.io/tutorials/using-npm-as-a-build-tool. Luettu: 11.5.2018.

Martin-Colby, A. 2017. What has been your experience using React.js as a developer. Luettavissa: https://www.quora.com/What-has-been-your-experience-using-React-js-as-a-developer. Luettu: 11.5.2018.

Panigrahi, S. 2014. Cursor Icon with Shadow. Luettavissa: https://commons.wikimedia.org/wiki/File:Cursor_icon_with_shadow.png. Luettu: 6.5.2018.

Pellegrom, G. 2017. Using Npm scripts as a build tool. Luettaviss: https://deliciousbrains.com/npm-build-script/. Luettu: 11.5.2018.

Hunt, P. 2015. Luettavissa: https://www.youtube.com/watch?v=A0Kj49z6WdM. Luettu: 8.4.2018.

Rasmussen, E. 2015. Ducks: Redux Reducer Bundles. Luettavissa: https://github.com/erikras/ducks-modular-redux. Luettu: 13.4.2018.

Rauschmayer, A. 2015. Arrow Functions. Luettavissa: http://exploringjs.com/es6/ch_modules.html. Luettu: 14.1.2018.

Rauschmayer, A. Modules. Luettavissa: http://exploringjs.com/es6/ch_modules.html. Luettu: 14.1.2018.

React DnD. Luettavissa: http://react-dnd.github.io/react-dnd/. Luettu: 19.3.2018.

ReactJS. JSX in Depth. Luettavissa: https://reactjs.org/docs/jsx-in-depth.html. Luettavissa: 18.4.2018.

ReactJS. Components and Props. Luettavissa: https://reactjs.org/docs/components-and-props.html. Luettavissa: 18.4.2018.

ReactJS. React Component. Luettavissa: https://reactjs.org/docs/react-component.html. Luettavissa: 18.4.2018.

ReactJS. Thinking in React. Luettavissa: https://reactjs.org/docs/thinking-in-react.html. Luettavissa: 18.4.2018.

React Training. React Router. Luettavissa: https://github.com/ReactTraining/react-router. Luettavissa: 18.4.2018.

ReduxJS. Usage with React Router. Luettavissa: https://redux.js.org/docs/advanced/UsageWithReactRouter.html. Luettu: 13.1.2018.

Redux Motivation, Luettavissa: https://redux.js.org/docs/introduction/Motivation.html. Luettu: 11.12.2017.

Reifman, J. 2016. The Future of Javascript: 2016 and Beyond. https://code.tutsplus.com/tutorials/the-future-of-javascript-2016-and-beyond--cms-26305. Luettu: 25.11.2107.

Sass Basics. Luettavissa: https://sass-lang.com/guide. Luettu: 14.5.2018.

Sengstacke, P. 2016. JavaScript Transpilers: What They Are & Why We Need Them. Luettavissa: https://scotch.io/tutorials/javascript-transpilers-what-they-are-why-we-need-them. Luettu: 13.5.2018.

Singhaniya, A. 2016. Problem with React Single Page App. Luettavissa; https://www.classandobjects.com/tutorial/problems_with_react_single_page_app/. Luettu: 13.1.2018.

Sites Using React. 2017. GitHub, Inc. https://github.com/facebook/react/wiki/sites-using-react. Luettu: 25.11.2017.

Six, J. & Macefield, R. 2016. How to Determine the Right Number of Participants for Usability Studies. Luettavissa: https://www.uxmatters.com/mt/archives/2016/01/how-to-determine-the-right-number-of-participants-for-usability-studies.php. Luettu: 13.5.2018.

Tiilikainen, T. 23.7.2017. & 30.11.2017. & 25.4.2018. Ohjelmistokehittäjä. Talented Solutions Oy. Haastattelu. Helsinki.

Web Frameworks: Pros and Cons Of Using Frameworks. 2015. Luettavissa: http://1stwebdesigner.com/web-frameworks. Luettu: 25.11.2017.

Webpack. Entry-points. Luettavissa: https://webpack.js.org/concepts/entry-points/. Luettavissa: 7.5.2018.

Webpack. Loaders. Luettavissa: https://webpack.js.org/concepts/loaders/. Luettavissa: 7.5.2018.

Webpack. Output. Luettavissa: https://webpack.js.org/concepts/output/. Luettavissa: 7.5.2018.

Webpack. Plugins. Luettavissa: https://webpack.js.org/concepts/#plugins. Luettavissa: 7.5.2018.

Liitteet

Liite 1. Käyttäjähaastattelut

Käyttäjähaastattelut olivat vapaamuotoisia, eivätkä ne noudattaneet tiettyä kaavaa. Tarkoitus oli antaa käyttäjien kertoa, mitä mieltä he olivat vanhasta järjestelmästä, sekä antaa palautetta kehitystyön aikana. Haastattelun aikana haastateltava käytti järjestelmää ja näytti haastattelijalle omia huomioitaan. Kartoituksena tehtäviin haastatteluihin osallistui kolme yrityksen työntekijää.

Alla olevat huomiot tehtiin käyttöliittymän läpikäynnin aikana:

- Tarve nähdä uusimmat talentit, naamat ja tiedot
- Drag & Drop -tyylinen tapa kontrolloida tietoja voisi olla parannus
- Ehdotuksena Trello tyylinen näkymä. Tiedot eivät olisi horisontaalisesti vaan vertikaalisesti
- Nykyisen käyttöliittymän monen askeleen polut hidastavat tekemistä
- voisi muokata tietoja jo listanäkymässä, ei tarvitse avata uutta sivua
- käyttöliittymäelementtien sijoittelu ei tue tekemistä
- scrollattavan tiedon määrä suuri ja hidastaa työtä – kognitiivinen rasite on suuri
- Talentit voisivat olla lajiteltu sen mukaan, milloin viimeksi muokattu
- Filtterien hyödyllisyys kyseenalaistettiin
- Eivät aina toimi, joten eivät ole luotettavia
- Sama hakutoiminnoissa
- relevantin infon puute editorissa – vanhassa käyttöliittymässä vain vähän tietoa
- toiveena edistynyt haku
- ohjelmointikielet näkyviin helpommin talenteille
- muistutukset talenttien "kortteihin"
- käyttäjien merkitseminen tietyille talenteille
- muistiinpanojen tekeminen
- cv liittäminen, muistutus jos ei ole lähetetty
- muistiinpanojen kokoaminen samaan paikkaan
- pakolliset kentät ennen profiilin hyväksyntää

Ennen suunnittelun aloittamista toteutettiin yrityksen suunnittelija Kaisa Mäkipään haastattelu. Sen tuloksena tultiin seuraaviin lopputulemiin. Lähtökohtana oli parantaa olemassa olevaa käyttöliittymää. Tarkoitus on saada sen näköinen käyttöliittymä, jota halutaan käyttää. Halutut toiminnot tulee löytyä ja tarpeellisten asioiden näkyvyyden tulee olla parempi. Käyttöliittymän tulisi olla visuaalinen ja intuitiivinen käyttää. Sen ei tulisi olla vain lista asioista. Trello on tuttu käyttäjille ja nousi sieltä esimerkkinä. Sen käyttö yleisesti tarkoitti, että malli on toimivaksi todettu. Trello myös sopii olemassa olevaan työntekijöiden prosessiin. Uuden näkymän tulisi vähentää klikkailua. Tärkeimpänä esimerkkinä olisi tuoda prosessin pullonkaulat esiin käyttöliittymän kautta. Talenttien profiilien ei tulisi olla sekavavia. Vaan sellaisia että sovelluksen käyttäjät ymmärtävät esitetyt tiedot. Käyttäjien mukana oleminen suunnittelussa tulisi olla isoin lähtökohta. Suunnittelussa on hyvä huomioida,

että sovellus on työkalu työntekijöille. Suunnittelun lähtökohdat ovat silloin erilaiset kuin kuluttajille suunnitellussa tuotteessa.

Kehitystyön päätteeksi haastattelin loppukäyttäjää valmistuneesta produktista, sekä miltä prosessi tuntui. Haastateltava kertoi, että uusi sovellus on nopea käsitellä. Se oli myös helppo ja nopea oppia. Vastaa ja ylittää odotukset. Tapahtumien käsittelyn helppous yllätti. Omistajuus tekemisessä tuntui. Ei odottanut toiveiden kuuntelua. Mielipiteitä kuunneltiin riittävästi ja sai olla mukana prosessia. Haastateltava ei olisi kaivannut lisää osallistumista. Ei olisi tehnyt asioita toisin.

Haastattelin myös ohjaajanani toimineen ohjelmistokehittäjän. Haastattelu keskittyi prosessin onnistumiseen. Haastateltavan mukaan, kun toiminnallisuuksien määrittely tehtiin hyvin, silloin odotukset osuivat kohdalleen. Tärkeää oli saada React käyttöön. Sai olla riittävästi vaikuttamassa. Toimeksiannon suorittajan työskentely oli ripeää, oma-aloitteista ja itsenäistä. Kehityksessä junior-status näkyy, mutta tekeminen ja perustoiminnot ovat hyviä.