Tampereen yliopisto

Rasmus Lempinen

TEKNOLOGIAVALINTOJEN VAIKUTUS WEB-SOVELLUKSEN KEHITYSPROSESSIIN

Diplomityö
Informaatioteknologian ja viestinnän tiedekunta
Tarkastaja: Yliopistonlehtori Terhi Kilamo
Huhtikuu 2020

TIIVISTELMÄ

Rasmus Lempinen: Teknologiavalintojen vaikutus web-sovelluksen kehitysprosessiin
Diplomityö
Tampereen yliopisto
Tietotekniikka
Huhtikuu 2020

Tässä diplomityössä tutkitaan Cybercom Finland Oy:lle toteutetun uusien työntekijöiden tehtävälistasovelluksen eli NewbieMakerin kehitysvaiheita. NewbieMaker on yhden sivun web-sovellus, joka on rakennettu hyödyntäen MERN-pinon teknologioita eli MongoDB:tä, Expressiä, React-kirjastoa sekä Node.js:ää. Tutkimuksen tavoitteena on selvittää, mitä valintoja NewbieMakerin eri kehitysvaiheissa on tehty ja, miten ne ovat vaikuttaneet sovelluksen kehitykseen. NewbieMaker-sovelluksen kehitys on jakautunut kolmeen selkeään eri kehitysvaiheeseen, joita käsitellään tässä tutkimuksessa erillisinä kokonaisuuksina. Sovelluksen ensimmäinen kehitysvaihe koostuu NewbieMakerin alkuperäisestä toteutuksesta osana Tampereen teknillisen yliopiston (nykyinen Tampereen yliopisto) projektityökurssia, jonka mukana Cybercom Finland Oy oli yhtenä yhteistyökumppanina. Toinen kehitysvaihe alkoi, kun alkuperäinen NewbieMakerin toteutus luovutettiin asiakkaan haltuun. Tämän kehitysvaiheen aikana sovelluksen selainpuolen teknologiat ja ulkoasu kokivat merkittäviä uudistuksia. Kolmas kehitysvaihe aloitettiin, kun toisen kehitysvaiheen lopussa aloitettu NewbieMakerin pilottitesti saatiin päätökseen.

Tutkimuksessa huomattiin, että MERN-pinon selain- ja palvelinpuolen yhtenäisen kehityskielen (JavaScript) ansiosta sovelluksen kehittäjien on helpompi ymmärtää sekä selain- että palvelinpuolen toimintaa. Toisessa kehitysvaiheessa tehty selainpuolen kehityskielen vaihto TypeScriptiin paransi koodin laatua vaikeuttamatta koodin ymmärrettävyyttä huomattavasti. Myös CSS-viitekehysten ja React-kirjaston käyttö nopeutti ja selkeytti NewbieMakerin käyttöliittymäkehitystä. CSS-viitekehykset mahdollistavat tyylikkäiden käyttöliittymäkomponenttien luonnin nopeasti ja Reactin komponenttipohjaisuus selkeytti sovelluksen rakennetta ja toiminnan ymmärtämistä. MERN-pinon tietokanta eli MongoDB mahdollisti myös dokumenttien datamallien helpon muuttamisen kehityksen aikana NoSQL-rakenteensa ansiosta.

Selkeä vaikutus kehitykseen oli myös alkuperäisen sovelluksen koon vähättelyllä, jonka seurauksena Redux-kirjastoa ei otettu vielä ensimmäisessä kehitysvaiheessa käyttöön. Tästä seurasi paisuneita React-komponenttien tilamäärittelyjä ja turhia komponenttien välisiä propsien välityksiä, mikä johti huonolaatuiseen ja vaikeasti ylläpidettävään koodiin. Toisen kehitysvaiheen aikana käyttöönotettu Redux-kirjasto ratkaisi tilamäärittelyyn liittyvät ongelmat ja sen vaikutusten ansiosta sovelluksen ylläpidettävyys kasvoi huomattavasti ja siirtyminen tulevien kehitysvaiheiden välillä helpottui.

Työssä saatuja tuloksia on mahdollista hyödyntää tulevaisuudessa uusissa sovelluskehitysprojekteissa. Tutkittujen valintojen vaikutusta kehitykseen voi punnita projektikohtaisesti ja niiden avulla on mahdollista parantaa sovelluksen laatua ja kehitysnopeutta alusta lähtien.

Avainsanat: web-sovellus, React, yhden sivun web-sovellus, sovelluskehitys, JavaScript

ABSTRACT

Rasmus Lempinen: The effect of technological choices on the development process of a web application
Master of Science Thesis
Tampere University
Information Technology
April 2020

In this master's thesis, the development phases of a task list application called NewbieMaker developed for Cybercom Finland Oy, are studied. NewbieMaker is a single page application built using technologies from the MERN stack which include MongoDB, Express, React and Node.js. The goal of this study is to find out what kind of decisions have been made in the different development phases of NewbieMaker and how they've affected the development of the application. The development of NewbieMaker is divided into three clear development phases which are handled as individual entities in this study. The first development phase of the application consists of the original implementation of NewbieMaker as a part of a project work course in Tampere university of technology (now Tampere university) in which Cybercom Finland Oy was a partner. The second development phase began when the original implementation of NewbieMaker was delivered to the customer. During this development phase the frontend technologies and the user interface of the application were renewed. The third development phase was launched when a pilot test which was started at the end of the second development phase, ended.

In this study, it was noticed that due to MERN stack's uniform development language for both, frontend and backend, it was easier for the developers to understand the functionality of both, the frontend and the backend. The switch to TypeScript as the development language for the frontend improved the code quality without considerably affecting the readability of the code. Also, the usage of CSS frameworks and React library sped up and clarified the user interface development of NewbieMaker. CSS frameworks enabled the creation of stylish user interface components quickly and React's component-based structure clarified the structure and functionality of the application. MERN stack's database, MongoDB, also made it possible to change the data models of the documents during development due to it being a NoSQL database.

Underestimating the size of the original application had a clear effect on development since it resulted in not using Redux in the first development phase. This resulted in bloated state definitions and unnecessary passing of props for the React components which led to bad quality and hard-to-maintain code. During the second development phase the problems with state definitions were resolved by implementing the usage of Redux which improved the maintainability of the application and made the transition to future development phases easier.

The results of this thesis can be made use of in new software development projects in the future. The effects of the studied decisions can be weighed depending on the project. With the help of the results it's possible to improve the quality and development speed of the application from the beginning.

Keywords: web application, React, single page application, software development, JavaScript

ALKUSANAT

Tämä diplomityö toteutettiin yhteistyössä Cybercom Finland Oy:n kanssa. Työssä tutkittu sovellus oli aluksi vain osa Tampereen teknillisen yliopiston projektikurssia, mutta muovautui kehityksen aikana sovellukseksi, joka pääsi oikeaan käyttöön Cybercomilla.

Haluan kiittää Cybercomia ja erityisesti Vesa Leppälää mahdollisuudesta olla osana tutkitun sovelluksen jokaista kehitysvaihetta. Kiitokset ansaitsee myös diplomityöni ohjaaja Terhi Kilamo, jonka jatkuvan palautteen ansiosta pystyin parantamaan työtäni kaikin puolin sen teon aikana. Erityiskiitos myös Tampereen yliopiston Hervannan kampuksen Päätalon tilalle PB 018, jonka uumenissa väsynytkin diplomityöntekijä sai voimaa jatkaa eteenpäin. Kunniamaininnan haluan antaa myös edesmenneelle TTY:lle, joka tarjosi unohtumattomia muistoja opiskelujeni aikana.

Tampereella, 29. huhtikuuta 2020

Rasmus Lempinen

SISÄLLYSLUETTELO

KUVALUETTELO

LYHENTEET JA MERKINNÄT

AWS	Pilvipalvelualusta (engl. Amazon Web Services)
CI/CD	Jatkuva integraatio ja jatkuva kehitys (engl. Continuous Integration / Continuous Development)
CSS	HTML-sivun tyylit määrittävä kieli (engl. Cascading Style Sheets)
DOM	HTML-tiedoston rakennetta kuvaava puu (engl. Document Object Model)
HTML	Web-sivun rakenteen muodostava merkintäkieli (engl. HyperText Markup Language)
JavaScript	Skriptauskieli, joka mahdollistaa toiminnallisuuksien luonnin web-sivuille
JSON	Tiedostomuoto (engl. JavaScript Object Notation)
Palvelinpuoli	Web-sovelluksen palvelimella toimiva osa (engl. Backend)
QA	Laadunvarmistus (engl. Quality Assurance)
React	Käyttöliittymäkehityksessä käytettävä selainpuolen JavaScript-kirjasto
Redux	Web-sovellusten tilanhallintaan käytettävä kirjasto
REST	Arkkitehtuurityyli web-sovellusten backend-rajapintojen suunnitteluun (engl. Representational State Transfer)
Selainpuoli	Web-sovelluksen käyttäjälle näkyvä osa (engl. Frontend)
SPA	Yhden sivun web-sovellus (engl. Single Page Application)
TTY	Tampereen Teknillinen Yliopisto
UI/UX	Käyttöliittymä ja käyttäjäkokemus (engl. User Interface / User Experience)

1 JOHDANTO

Nykyaikaisen sovelluksen toteuttaminen web-pohjaisena pitää sisällään lukuisia hyötyjä verrattuna sovelluksen toteuttamiseen työpöytäversiona. Käyttöalustaltaan web-sovellukset ovat joustavia, sillä ne käyttävät suoritusympäristönään web-selaimia, mikä mahdollistaa sovelluksen käytön niin mobiililaitteella, tabletilla kuin tietokoneella. Web-sovellusten käyttö ei siis vaadi loppukäyttäjää asentamaan sovellusta käyttäjän laitteelle, koska ne sijaitsevat palvelimella, joka mahdollistaa myös sovellusten päivittämisen ilman loppukäyttäjän toimenpiteitä. [1][2]

Web-sovellusten rakenteen voi yleisesti jakaa kolmeen osaan: selain- ja palvelinpuoleen sekä tietokantaan 3. Selainpuoli viittaa siihen sovelluksen osaan, joka on näkyvillä käyttäjälle ja, jonka kautta käyttäjä voi suorittaa toimintoja ja näin olla vuorovaikutuksessa sovelluksen kanssa. Palvelinpuoli on sovelluksen palvelimella toimiva osa eli se sovelluksen osa joka ei näy käyttäjälle. Palvelinpuoli kommunikoi sovelluksen selainpuolen ja tietokannan kanssa vastaanottamalla ja lähettämällä käyttäjän pyyntöjen mukaista dataa. [3] Tietokantaan voidaan säilöä sovelluksen tietoja ja palauttaa niitä tarvittaessa sovelluksen palvelinpuolelle [4].

Web-sovellukset voidaan toteuttaa niin sanottuna yhden sivun web-sovelluksina, jolla pyritään saavuttamaan natiivin työpöytäsovelluksen ulkonäkö ja käytettävyys. Yhden sivun web-sovellusten toiminta perustuu yhden näkymän dynaamiseen muokkaamiseen selaimessa sen sijaan, että jokaista näkymää kohden haettaisiin palvelimelta uusi web-sivu. Tämän ansiosta yhden sivun web-sovellusten toiminnot tapahtuvat käyttäjän näkökulmasta samassa näkymässä ilman ylimääräisiä sivulatauksia, mikä luo käyttäjälle paremman käyttäjäkokemuksen. Yhden sivun web-sovellusten kehittämisen apuvälineiksi on olemassa useita eri viitekehyksiä ja kirjastoja. Yksi suosituimmista kirjastoista tällaisten sovellusten kehittämiseen on tässä diplomityössä esitelty React-kirjasto. [5]

Tässä diplomityössä esitellään Cybercom Finland Oy:lle toteutetun web-sovelluksen kehitysprosessin vaiheet. Sovelluksen toteutus oli alun perin osa Tampereen teknillisen yliopiston (nykyinen Tampereen yliopisto) projektityökurssia, jonka tavoitteena oli toimittaa oikealle yritysasiakkaalle toimiva tuote opiskelijoista koostuvan kehittäjätiimin voimin. Tarkasteltava sovellus on uusien työntekijöiden perehdyttämiseen ja etenemisen seuraamiseen käytettävä tehtävälistasovellus. Kyseinen sovellus toteutettiin yhden sivun web-sovelluksena ja se sai viralliseksi nimekseen NewbieMaker. Tämän diplomityön tavoitteena on tutkia, millaisia valintoja NewbieMaker-sovelluksen eri kehitysvaiheissa on tehty ja, miten kyseiset valinnat ovat vaikuttaneet sovelluksen kehitykseen.

NewbieMaker-sovellus on käynyt läpi kolme selkeää kehitysvaihetta. Ensimmäinen kehitysvaihe sisältää sovelluksen alkuperäisen toteutuksen Cybercom Finland Oy:lle osana TTY:n projektityökurssia. Toinen vaihe koostuu sovelluksen yleisilmeen ja teknologioiden parantamisesta pilottitestiä varten, joka alkoi kun alkuperäinen sovellus luovutettiin virallisesti Cybercom Finland Oy:n haltuun. Kolmanteen vaiheeseen siirryttiin sovelluksen pilottitestauksen jälkeen, jolloin sovellus sai Cybercom Finland Oy:n sisällä virallisen sisäisen kehitysprojektin tittelin. Kolmas kehitysvaihe koostuu sovelluksen ylläpidettävyyden, käyttäjähallinnan ja tuotantoonviemisen paranteluista. Jokainen kehitysvaihe käydään läpi järjestyksessä diplomityön luvuissa 4, 5 ja 6.

Lopuksi diplomityön luvussa 7 pohditaan aiempien kehitysvaiheiden aikana tehtyjen valintojen vaikutuksia kehitykseen. Miten uusien tekniikoiden mukaan ottaminen ja vanhojen tekniikoiden pois jättäminen vaikutti kehitykseen? Työssä pohditaan myös ei-teknologisten valintojen vaikutusta prosessiin esimerkiksi kehitystiimin muuttuessa. Luvun 7 lopussa pohditaan myös hieman sovelluksen tämänhetkisen toteutuksen ylläpidettävyyden tasoa ja mahdollisia jatkokehityskohtia.

2 WEB-SOVELLUKSET

Perinteisten tietylle käyttöjärjestelmälle luotujen sovellusten sijaan on noussut trendi toteuttaa sovellukset web-pohjaisina. Web-sovellukset eivät ole riippuvaisia käyttöjärjestelmistä tai laitteista, vaan niitä voi käyttää yhtä lailla esimerkiksi Windowsilla, macOS:llä, Linuxilla ja mobiililaitteilla. [1] Niitä käytetään web-selaimella sekä niiden päivittäminen ja ylläpito tapahtuu etänä, minkä ansiosta käyttäjien ei tarvitse ladata tai uudelleenasentaa sovelluksia erikseen [2].

Luvussa 2 käydään läpi perinteisen web-sovelluksen ja yhden sivun web-sovelluksen rakenne, jotta tässä työssä käsiteltävä kysymys olisi lukijalle mahdollisimman selkeä. On tärkeää ymmärtää mistä osista web-sovellukset muodostuvat ja miten ne toimivat keskenään. Kohdassa 2.1 ja sen alakohdissa käydään läpi web-sovellusten peruspalasia, kuten asiakas-palvelin -malli, selainpuoli, palvelinpuoli ja tietokanta sekä nopea katsaus pilvipalveluihin. Kohdassa 2.3 esitellään tämän työn kannalta tärkeä JavaScript-kirjasto React, joka oli suuressa roolissa tarkasteltavan sovelluksen selainpuolen kehityksessä.

2.1 Web-sovellusten toiminta ja rakenne

Web-sovellukset käyttävät suoritusympäristönään web-selaimia (Google Chrome, Mozilla Firefox, Safari jne.) [2] ja hyödyntävät toimiakseen selainten komponentteja joihin lukeutuu muun muassa HTML (HyperText Markup Language), CSS (Cascading Style Sheets), JavaScript ja DOM (Document Object Model) [1]. Web-sovellukset eroavat web-sivuista siten, että niiden avulla käyttäjä pysyy suorittamaan tiettyjä tehtäviä, kuten sähköpostin lähetys, hotellivaraus tai laskujen maksaminen. Perinteisiä web-sivuja käytetään sen sijaan pääosin staattisen sisällön selaamiseen. [2]

Kuva 2.1. *Asiakas-palvelin -malli*

Web-sovellusten yhtenä kehittämisen nopeuttajana toimii verkon arkkitehtuuri eli asiakas-palvelin -malli (engl. Client-Server model). Arkkitehtuurin mukaan sovellus jaetaan kah-

teen osaan: asiakkaaseen ja palvelimeen. Asiakas (esim. sovelluksen käyttäjä) tekee pyyntöjä palvelimelle, joka puolestaan käsittelee pyynnöt, kutsuu tarvittaessa tietokantaa ja palauttaa vastauksen asiakkaalle. [6] Asiakas-palvelin -malli on esitetty kuvassa 2.1. Kuvassa esiintyvät osat voidaan jakaa web-kehityksen termien alle: asiakaspää on selainpuoli, josta kerrotaan alakohdassa 2.1.1, palvelinpää on palvelinpuoli, josta kerrotaan alakohdassa 2.1.2 ja tietokannasta kerrotaan alakohdassa 2.1.3.

2.1.1 Selainpuoli

Web-sovelluksista puhuttaessa selainpuolella tarkoitetaan sovelluksen käyttäjälle näkyvää osaa. Sovelluksen selainpuoli reagoi käyttäjän toimintoihin ja suorittaa toimintojen mukaista koodia lokaalisti käyttäen suoritusympäristönään web-selainta. Selainpuolisovelluksia toteutetaan käyttäen kohdassa 2.1 mainittuja HTML, CSS ja JavaScript teknologioita. [3]

HTML on "merkintäkieli" (engl. Markup language), jonka avulla muodostetaan luotavan dokumentin perusrakenne. HTML voi määritellä myös tyylittelyjä sekä sisältää skriptejä, joiden avulla voidaan muokata luodun dokumentin toimintoja web-selaimessa. [1]

CSS mahdollistaa esim. HTML:llä luodun dokumentin tyylien määrittelyn erilliseen tiedostoon, jolloin tyylejä ei tarvitse määritellä HTML tiedostoon. Tyylimäärittelyillä voidaan vaikuttaa HTML-dokumentin ulkonäköön esim. määrittelemällä värejä, fontteja tai dokumentin asettelu. [1]

JavaScript on ECMAScript-standardiin pohjautuva maailman suosituin verkossa käytettävä skriptauskieli, joka mahdollistaa suoritettavan koodin luomisen web-dokumenttiin. JavaScriptin avulla web-sivuille voidaan luoda toiminnallisuuksia ja tällä tavalla niistä voidaan luoda dynaamisempia. [1] Toiminnallisia web-sivuja kutsutaankin usein web-sovelluksiksi [7].

DOM toimii rajapintana verkkoselaimen ja skriptauskielen (kuten JavaScript) välissä. Se määrittelee dokumenttien loogisen rakenteen ja mahdollistaa dokumenttien tutkimisen ja muokkaamisen. [8]

Modernien web-sovellusten selainpuolet sisältävät kuitenkin niin paljon toiminnallisuuksia, että niiden kehittäminen ja ylläpito pelkästään yllä mainittujen tekniikoiden avulla on todella työlästä. Tämän takia selainpuolen kehityksen apuvälineeksi on luotu useita eri viitekehyksiä (engl. framework), joiden avulla web-sovellusten kehitys on yhtenäisempää ja yksinkertaisempaa. Selainpuolen suosituimpiin viitekehyksiin lukeutuvat mm. React (kirjasto), Vue.js ja Angular. [9] Kohdassa 2.3 kerrotaan enemmän React-kirjastosta, jota on hyödynnetty tässä työssä käsiteltävässä web-sovelluksessa.

2.1.2 Palvelinpuoli ja REST

Palvelinpuolella tarkoitetaan web-sovelluksen palvelimella toimivaa osaa, joka ei näy käyttäjälle ja on yleensä synonyymi palvelimelle. Tämä voi sisältää palvelinalustan, jonka yleisimpiä toteutuskieliä ovat muun muassa PHP, Java ja Node.js sekä tietokannan. [3] Palvelinpuolen tehtäviin kuuluu muun muassa käyttäjän pyyntöjen käsittely, sisällön tuottaminen käyttöliittymään ja kommunikointi tietokannan kanssa [10]. Yleisin tapa selainpuolen ja palvelinpuolen väliseen kommunikaatioon ja datan siirtoon on käyttää RESTful-periaatteita [3].

REST (Representational State Transfer) on Roy Fieldingin vuonna 2000 esittelemä arkkitehtuurityyli, jota käytetään apuna web-sovelluksen palvelinpuolen rajapinnan suunnittelussa. Kaikkea tietoa, joka voidaan nimetä, kutsutaan REST-arkkitehtuurin mukaan resurssiksi. Resurssi voi olla lähes mitä tahansa, kuten dokumentti, kuva, lyhytaikainen palvelu, henkilö tms. REST-arkkitehtuurille on asetettu kuusi ohjaavaa rajoitetta:

1. Yhtenäinen rajapinta (engl. Uniform interface)
2. Asiakas ja palvelin (engl. Client-server)
3. Tilattomuus (engl. Stateless)
4. Välimuistin käyttö (engl. Cacheable)
5. Kerroksittainen järjestelmä (engl. Layered system)
6. Ladattava koodi (engl. Code on demand) (vapaaehtoinen) [11][12]

Yhtenäinen rajapinta yksinkertaistaa arkkitehtuuria ja tekee toiminnoista näkyvämpiä. Tällä rajoitteella on itsessään neljä omaa rajoitetta:

1. Resurssi on oltava tunnistettavissa pyynnössä esim. URIn avulla.
2. Resurssin manipulointi esitysmuotojen kautta. Jos käyttäjällä on jokin esitysmuoto resurssista, on hänellä aina tarpeeksi tietoa joko poistaa tai muokata resurssia.
3. Itsekuvaavat viestit. Jokaisessa viestissä kuuluu olla tarpeeksi tietoa siitä, miten kyseinen viesti voidaan käsitellä.
4. Hypermedia sovelluksen tilamoottorina. Käyttäjä pystyy dynaamisesti käyttämään palvelimelta saatuja linkkejä kaikkien saatavilla olevien toimintojen ja resurssien löytämiseen.

Asiakas ja palvelin -rajoite tarkoittaa yksinkertaisesti käyttöliittymän erottelua palvelimen logiikasta, jolloin molempia voidaan kehittää itsenäisesti toisistaan riippumatta. Tällöin asiakaspään ei tarvitse tietää kuin palvelimella olevien resurssien URIt kommunikaatiota varten.[11][12]

Tilattomuus-rajoitteella tarkoitetaan, että palvelin käsittelee jokaisen saamansa pyynnön uutena eikä täten säilö mitään tietoa vanhoista pyynnöistä. Tämä tarkoittaa sitä, että jokaisen asiakkaalta palvelimelle tulevan pyynnön tulee sisältää kaikki tarpeellinen tieto pyynnön käsittelyä varten. [11][12]

Välimuistin käyttö -periaattella mahdollistetaan datan uudelleenkäyttö. Tämän periaatteen mukaan jokaisen resurssin tulee määrittää, onko sitä sallittua tallentaa välimuistiin vai ei. Välimuistiin tallentaminen parantaa asiakaspään tehokkuutta eliminoidessaan tarpeen tehdä uusia pyyntöjä palvelimelle. [11][12]

Kerroksittainen järjestelmä -periaate nimensä mukaan mahdollistaa järjestelmän luomisen kerroksittain esim. eri palvelimille. Tällöin asiakaspää ei näe muuta kuin sen kerroksen (palvelimen), jonka kanssa se on vuorovaikutuksessa, vaikka kyseinen kerros keskustelisikin pyynnön jälkeen muiden kerrosten (palvelinten) kanssa. [11][12]

Vaihtoehtoinen rajoite **ladattava koodi** sallii asiakaspään toimintojen laajentamisen mahdollistamalla suoritettavan koodin hakemisen palvelimelta normaalin staattisen XML/JSON-datan lisäksi. Tämä poistaa tarpeen toteuttaa joitakin asiakaspään toiminnallisuuksia, sillä ne saadaan suoraan palvelimelta. [11][12]

HTTP metodi	Selitys	Esimerkki
GET	Hae lista kaikista käyttäjistä.	http://example.com/api/users
GET	Hae tietty käyttäjä id:n (123) perusteella.	http://example.com/api/users/123
POST	Luo uusi käyttäjä pyynnön sisältämien tietojen perusteella.	http://example.com/api/users
PUT	Päivitä tietyn käyttäjän tiedot id:n (123) perusteella.	http://example.com/api/users/123
DELETE	Poista tietty käyttäjä id:n (123) perusteella.	http://example.com/api/users/123

Kuva 2.2. *REST-apin CRUD-operaatiot. Muokattu lähteestä [13]*

Resursseihin pääse käsiksi HTTP-kutsujen avulla. Resurssien lukeminen, luominen, päivitys ja poisto tapahtuu niin sanottujen CRUD-operaatioiden (engl. Create, Read, Update, Delete) avulla. Kuvassa 2.2 on esitelty viisi REST-apille tehtävää kutsua. GET-pyynnöt keskittyvät tiedon hakemiseen, POST-pyynnöt tiedon luomiseen, PUT-pyynnöt tiedon päivittämiseen ja DELETE-pyynnöt tiedon poistamiseen. [13]

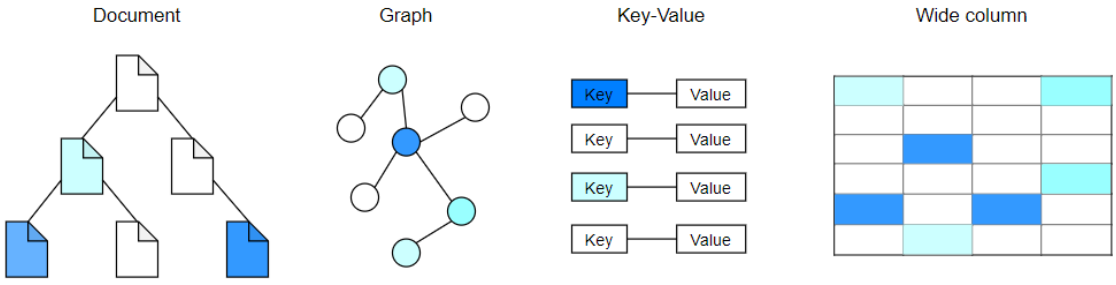
2.1.3 Tietokanta

Web-sovelluksien toimintoihin yleensä lukeutuu tiedon tallessapito, josta huolehtii tietokanta. Tietokannalla tarkoitetaan organisoitua tietorakennetta, joka sisältää usein useita eri tauluja. Tauluilla tarkoitetaan tietokannan sisäistä rakennetta liittyen tiettyyn tietoon esim. yksi taulu voi sisältää sovelluksen käyttäjien tietoja, kuten käyttäjänimi, salasana, puhelinnumero, sähköposti jne. Jos kyseessä on verkkokauppa niin jokin muu taulu voi sisältää tietoja kyseisen kaupan tuotteista ja niiden hinnoista, määristä jne. Jokaisen taulun kentät riippuvat siitä, mitä tietoa kyseisessä taulussa halutaan säilyttää. Web-sovelluksen kontekstissa tietokannan kanssa keskustelu tapahtuu palvelimen toimesta, joka on esitetty kuvassa 2.1. Käyttäjän tekemät kutsut "valuvat" palvelimelle, jolloin palvelin kutsuu tarvittaessa tietokantaa, joka palauttaa halutun tiedon palvelimelle. [4]

Relaatiosymboli R	Ominaisuus A1	Ominaisuus A2	Ominaisuus A3
Monikko t1	arvo	arvo	arvo
Monikko t2	arvo	arvo	arvo

Kuva 2.3. *Relaatiotietokanta. Muokattu lähteestä [14]*

Tietokannat jaetaan yleensä kahteen eri tyyppiin: **relaatiotietokantoihin** ja **relaatiomallista poikkeaviin tietokantoihin**. Relaatiotietokannan perusrakenne esitetään kuvassa 2.3 ja sillä tarkoitetaan tietokantaa, joka perustuu tiedon säilömiseen riveiksi taulujen sisään. Jokainen rivi edustaa yhtä kokonaisuutta ja jokainen rivin sarake yhtä ominaisuutta. Tarkemmin kuvattuna jokaisella taululla ja sarakkeella on ennaltamääritelty nimi. Taulun nimeä kutsutaan relaatiosymboliksi (kuvassa "R") ja jokainen sarake edustaa yhtä taulun ominaisuutta (kuvassa "A"). Taulun rivit, joita kutsutaan monikoiksi (kuvassa "t") muodostetaan arvoista, jotka asetetaan kutakin arvoa vastaavan ominaisuus-sarakkeen alle. Tällöin tietyn ominaisuuden arvot löytyvät vain ja ainoastaan kyseisen sarakkeen alta. Relaatiosymbolille määriteltyjä ominaisuusnimiä kutsutaan relaatiokaavioksi. Tietokantakaavio muodostuu, kun kaikkien relaatiosymboleiden relaatiokaaviot yhdistetään. [14] Relaatiotietokannan lukeminen ja muuttaminen on helppoa ja se onnistuu niin sanottujen kyselyiden avulla. Ylivoimaisesti suosituin kyselykieli on SQL (engl. Structured Query Language). [4]



Kuva 2.4. *NoSQL tietokantatyypit. Muokattu lähteestä [15]*

Toinen mainittu tietokantatyyppi on ns. relaatiomallista poikkeava tietokanta eli NoSQL (engl. Not only SQL). NoSQL mahdollistaa joustavamman mallin datan tallentamiseen ja lukemiseen verrattuna relaatiotietokantojen taulurakenteeseen. NoSQL on myös helppokäyttöisempi ja tarjoaa enemmän skaalautuvuutta ja tehokkuutta kuin relaatiotietokannat. NoSQL tietokannat voidaan jakaa neljään eri päätyyppiin:

1. ***Key-value store*** perustuu nimensä mukaisesti avaimiin ja niiden osoittamiin arvoihin. Avain on yleensä merkkijono ja data on jotakin yleistä datatyyppiä, kuten merkkijono, kokonaisluku tai olio. Key-value store -malli mahdollistaa kenttien ja olioiden rakenteen helpon muokkaamisen sovelluksen kehittyessä.
2. ***Document-based*** -tietokannoissa dokumenttiavaimet ovat yhdistetty johonkin monimutkaisempaan tietorakenteeseen, jota kutsutaan dokumentiksi. Document-based -tietokannat mahdollistavat myös helpon muokkauksen, sillä ne eivät pakota

käyttämään tiettyä mallia.

3. ***Wide column*** -malli perustuu riveihin ja mahdollistaa nopean pääsyn dataan riviavaimen, sarakenimen ja solun aikaleiman avulla. Tietokannan rivien sarakkeet voivat erota toisistaan, jolloin kaikkien rivien ei tarvitse sisältää samoja tietoja.
4. ***Graph-based*** on graafipohjainen ratkaisu, jota käytetään kun halutaan tutkia datan välisiä suhteita ja riippuvuuksia. Graph-based ratkaisu helpottaa myös datan visualisointia ja analysointia. [16]

Kaikki yllämainitut NoSQL tietokantatyypit ovat kuvattu ylempänä kuvassa 2.4. Yleisesti ottaen NoSQL on hyvä tietokantavalinta, jos työskentelee muuttuvan datan kanssa, jolloin NoSQL:n joustavuus tuo erityistä lisäarvoa.

2.2 Pilvipalvelut

Yleisesti pilvipalveluilla tarkoitetaan virtuaalisten resurssien, kuten palvelinten tarjontaa internetin välityksellä. Pilvipalvelun avulla voidaan säilöä ja hallita dataa vuokratulla etäpalvelimella, jolloin ei ole tarvetta ylläpitää omia paikallisia ja yksityisiä palvelimia. [17] Web-sovellusten hostaus (engl. hosting) jouduttiin toteuttamaan itse ostetuilla ja ylläpitämillä fyysisillä palvelimilla (ns. on-site) ennen pilvipalveluiden olemassaoloa. Palvelintilaa jouduttiin ostamaan runsaasti, jotta pystyttiin takaamaan tarpeellinen suorituskyky ruuhka-aikoina, joka myös johti siihen, että suuri osa palvelintilasta oli käyttämättömänä ruuhka-ajan ulkopuolella. Nykyään kokonaisia sovelluksia, testiympäristöjä ja web-sivuja voidaan hostata ja ylläpitää helposti pilvipalveluiden avulla, jolloin oman fyysisen datakeskuksen hankkiminen on turhaa. Pilvipalveluiden suurimpiin etuihin lukeutuu mm. niiden skaalautuvuus, luotettavuus, kustannustehokkuus ja helppous, minkä takia niiden suosio on kasvanut todella suureksi. [18]

Pilviä on olemassa kolme eri tyyppiä:

1. Julkinen pilvi
2. Yksityinen pilvi
3. Hybridipilvi

Julkisen pilven palvelutarjoajat tarjoavat resurssinsa julkiseen käyttöön verkon kautta. Näihin resursseihin voi lukeutua tiedon varastointi, sovellukset tai virtuaalikoneet. Julkinen pilvi tarjoaa tietoturvaa, skaalautuvuutta, resurssien jakamista ja niistä maksamista tarpeen mukaan. [17][18]

Yksityinen pilvi muistuttaa eniten vanhanaikaista on-site -ratkaisua, jossa yksi organisaatio tarjoaa pilvipalveluita sisäisille käyttäjille sisäisen verkon kautta. Yksityisen pilven datakeskus voi sijaita fyysisesti organisaation omissa tiloissa tai se voi olla jonkin kolmannen osapuolen ylläpitämä off-site -toteutus. Arkkitehtuurin kustomointi ja tietoturvakäytännöt ovat yksityisen pilven etuja. [17][18]

Hybridipilvi yhdistää julkisen ja yksityisen pilven, jolloin yksityisen pilven ratkaisua voi-

daan laajentaa julkisen pilven ratkaisuilla resurssitarpeiden kasvaessa. Hybridipilvi mahdollistaa tietojen ja toimintojen jakamisen loogisesti. Esim. yksityiset tiedot voidaan säilöä yksityisessä pilvessä ja laskentatehoa vaativat toiminnot voidaan suorittaa julkisessa pilvessä. [17][18]

Pilvipalveluita on olemassa kolme eri tyyppiä:

1. Software as a Service (SaaS)
2. Infrastructure as a Service (IaaS)
3. Platform as a Service (PaaS)

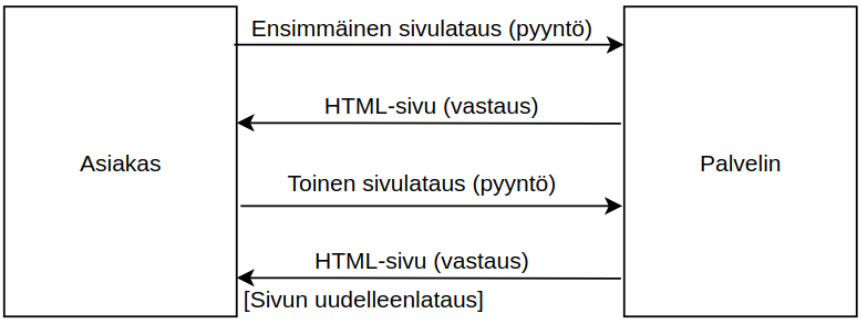
Software as a service tarkoittaa ohjelmiston hankkimista palveluna, jolloin käyttäjän ei tarvitse ladata, asentaa ja ylläpitää ohjelmistoa, vaan se tarjotaan suoraan pilvestä. SaaSin käytöllä voidaan vähentää ylläpito- ja palvelukuluja. [17][18]

Infrastructure as a service -tyyppi tarkoittaa datakeskuksen siirtoa kokonaisuudessaan pilveen on-site -ratkaisun sijaan. Tällöin organisaatio vuokraa tarvitsemansa palvelimet ja laitteet pilvipalveluntarjoajalta, joka huolehtii niiden ylläpidosta. [17][18]

Platform as a service mahdollistaa sovellusten nopean kehittämisen tarjoamalla valmiin ympäristön sovelluksen kehittämistä, testausta, toimittamista ja hallintaa varten. Tällöin kehittäjän ei tarvitse itse huolehtia sovelluksen "alla" pyörivästä infrastruktuurista, vaan voi keskittyä kehittämiseen. [17]

2.3 Yhden sivun web-sovellukset

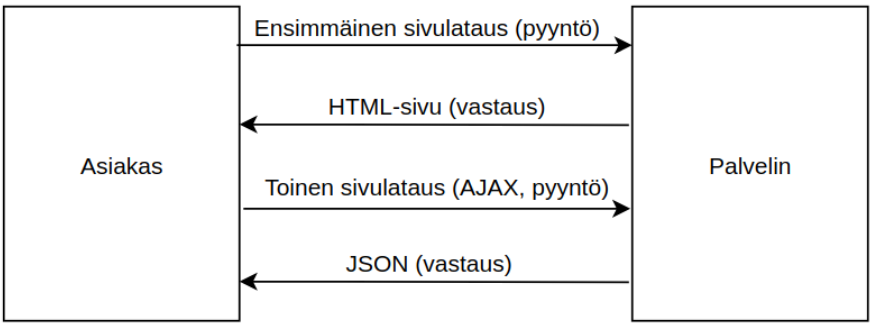
Web-sovelluksia luodessa usein halutaan, että sovellus tuntuu ja näyttää yhtä hyvältä kuin perinteinen työpöytäsovellus. Tätä varten on luotu yhden sivun web-sovelluksen (engl. Single Page Application, SPA) konsepti. SPA-sovelluksella voidaan saavuttaa juurikin edellä mainittu käyttäjäkokemus ja sovellus on samalla alustarippumaton, koska suoritusympäristönä toimii web-selain. SPA-sovellusten perustana toimii se, että käyttöliittymän hallinta tapahtuu palvelimen sijaan selaimessa, jolloin koko sovellus toimii yksittäisellä verkkosivulla. [5]



Kuva 2.5. *Perinteinen web-sovellus. Muokattu lähteestä [19]*

Perinteisessä web-sovelluksessa uuden näkymän pyyntöön reagoidaan lähettämällä se

palvelimelle. Palvelimella pyynnön nappaa käsittelijä, joka yhdessä malli-kerroksen kanssa käsittelevät pyynnön ja muokkaavat näkymää sen mukaisesti. Tämän jälkeen palvelin vastaa lähettämällä uuden muokatun HTML-sivun näkymälle, jonka käyttäjä saa näkyviin web-selaimen päivityksen jälkeen. Tämä prosessi on kuvattu yksinkertaistettuna kuvassa 2.5. Käyttäjä (asiakas) tekee ensimmäisen latauspyynnön, johon palvelin vastaa HTML-sivulla ja ensimmäinen näkymä latautuu. Käyttäjä navigoi web-sovelluksessa toiseen näkymään, josta seuraa uusi latauspyyntö palvelimelle. Palvelin vastaa uudella HTML-sivulla, joka laukaisee web-sovelluksen uudelleenlatauksen. [5]



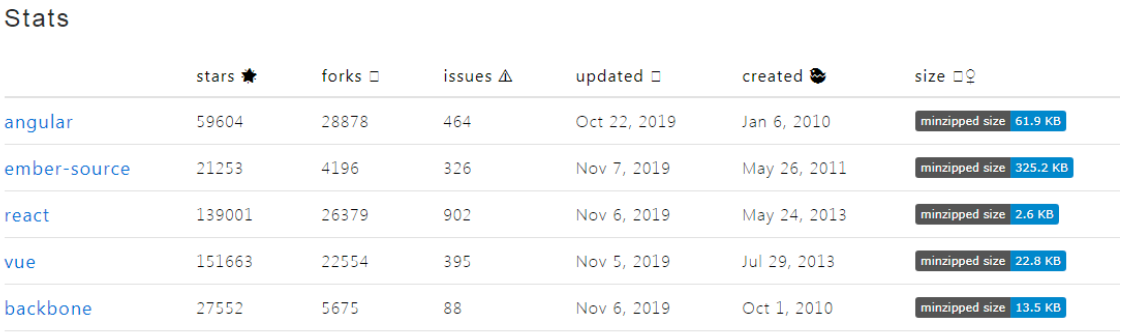
Kuva 2.6. *Yhden sivun web-sovellus. Muokattu lähteestä [19]*

SPA-sovelluksissa näkymä koostuu DOMin osista eikä kokonaisista HTML-sivuista. Kaikki näkymän luontia ja muokkausta varten tarvittavat työkalut ladataan ensimmäisellä sivulatauksella. Tämän jälkeen näkymien muokkaus tapahtuu selaimessa DOMin ja JavaScriptin yhteistyöllä, jonka ansiosta web-selainta ei tarvitse erikseen päivittää, jos web-sovelluksen näkymää halutaan muuttaa. Tämä toimintaperiaate tarkoittaa samalla sitä, että SPA-sovellukset ovat raskaampia käyttäjäpäässä, koska HTML:n ja datan yhdistely on siirretty palvelimelta selaimeen. Selaimen tekemää työmäärää voi kuitenkin helpottaa ottamalla selainpuolella käyttöön JavaScript viitekehyksen/kirjaston kuten Angular tai React, joista jälkimmäinen esitellään tarkemmin kohdassa 2.3. SPA-sovelluksen toiminta on esitetty yksinkertaistettuna kuvassa 2.6. Käyttäjä (asiakas) lataa web-sovelluksen näkymän ensimmäistä kertaa ja saa vastauksena palvelimelta HTML-sivun ja samalla kaikki näkymän muokkaukseen käytettävät työkalut. Seuraavilla näkymän muokkauspyynnöillä palvelin vastaa JSON-muotoisella datalla, jonka mukaan näkymää muokataan työkaluilla, jotka latautuivat käyttäjäpäätyyn jo ensimmäisellä latauskerralla. JSON-muotoisen palvelinvastauksen ansiosta web-sovelluksen näkymää ei tarvitse uudelleenladata, vaan näkymän muokkaus tapahtuu web-selaimessa. [5]

2.4 React-kirjasto

React (React JS, React.js) on Facebookin kehittämä Javascript-kirjasto interaktiivisten ja ajan myötä muuttuvien käyttöliittymien kehittämistä varten. Se on yksi maailman suosituimmista Javascript selainpuoli -kirjastoista ja sillä on todella suuri kehittäjä- ja käyttäjäkanta. [20] Kuvassa 2.7 on listattu suosituimpia GitHub repositorioita, joista Reactilla

on toisiksi eniten tähtiä. Reactin GitHub repositorio on myös (12.02.2019) neljänneksi suosituin koko maailmassa [21].


Stats

	stars	forks	issues	updated	created	size
angular	59604	28878	464	Oct 22, 2019	Jan 6, 2010	minzipped size 61.9 KB
ember-source	21253	4196	326	Nov 7, 2019	May 26, 2011	minzipped size 325.2 KB
react	139001	26379	902	Nov 6, 2019	May 24, 2013	minzipped size 2.6 KB
vue	151663	22554	395	Nov 5, 2019	Jul 29, 2013	minzipped size 22.8 KB
backbone	27552	5675	88	Nov 6, 2019	Oct 1, 2010	minzipped size 13.5 KB

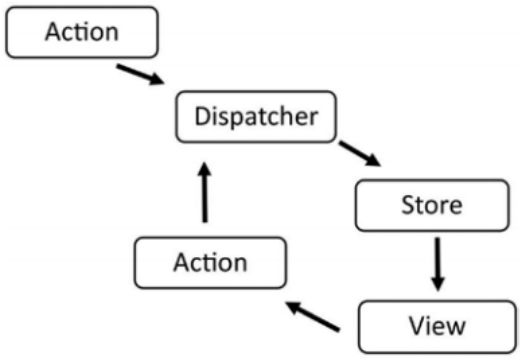
Kuva 2.7. *Githubin suosituimpia selainpuoli-viitekehyksiä. Lähde [22]*

React toimii MVC-mallin (Model-View-Controller) näkymänä (View) ja sen avulla pyritään luomaan uudelleenkäytettäviä ja modulaarisia käyttöliittymäkomponentteja. Sillä pystytään toteuttamaan suuria ja monimutkaisia kohdassa 2.2 mainittuja SPA-sovelluksia. [20] React-sovellukset ovat komponenttipohjaisia, joka tekee niistä helposti muokattavia, koska komponentteja voi vapaasti yhdistellä keskenään skaalautuvuuden ansiosta. Usein Reactilla luodut käyttöliittymät ovatkin yhdistelmiä useista eri komponenteista. [23] React on myös todella tehokas Javascript-kirjasto, koska se eroaa muista kilpailijoistaan kriittisellä tavalla: se käyttää apunaan ns. **Virtuaalista DOM**ia "normaalin" DOMin sijaan. React säilöö virtuaalisen DOMin muistiin ja tekee siihen muokkauksia saadessaan käyttäjältä pyyntöjä näkymän muuttamiseen. [20] Näkymän päivittämisen prosessin vaiheet ovat seuraavat:

1. Sovelluksen tila muuttuu, jolloin DOM ja React hahmontavat (engl. Render) uuden näkymän virtuaaliseen DOM-esitysmuotoon.
2. Näkymän tarvitsemat muutokset lasketaan vertailemalla kohdassa 1 muodostettua virtuaalista DOMia edelliseen, ennen tilan muuttumista luotuun virtuaaliseen DOMiin. Näin saadaan selville vain ne DOMin osat, jotka kuuluu päivittää.
3. React päivittää oikean DOMin vain niiden osien kohdalta, jotka saatiin laskettua kohdassa 2. [24]

Virtuaalisen DOMin käyttö nopeuttaa näkymän muuttamisen prosessia huomattavasti, koska sillä vältytään koko DOMin uudelleenhahmontamiselta joka kerta, kun sovelluksen tila muuttuu [24].

React-sovellusten datan hallintaa voi pitää helppona verrattuna perinteiseen MVC-arkkitehtuurin omaavaan sovellukseen. Datavirta on Reactissa yksisuuntaista, joka tarkoittaa sitä, että data voi kulkea vain yhteen suuntaan: parent komponentilta lapsi komponentille. Komponenttien sisäisen tilan hallinta tapahtuu siis "valuttamalla" dataa korkeamman tason komponenteilta alemman tason komponenteille tai muuttamalla komponentin tilaa sen itsensä sisällä. Tämä periaate kuitenkin voi hankaloittaa tilan hallintaa sovellusten kasvaessa. Facebook onkin luonut suurempien sovellusten tilanhallinnan



Kuva 2.8. *Flux-arkkitehtuurin datavirta. Lähde [23]*

helpotukseksi Flux-arkkitehtuurin, jonka mukaan data kulkee käyttäjän toimintojen (action) mukaan niin sanotun *dispatcher*in ja *store*n kautta näkymälle. Dispatcherin tehtäviin kuuluu varmistaa, että toiminnot suoritetaan yksi kerrallaan ja samassa järjestyksessä kuin ne saapuivat storeen. Sovelluksen tila ja logiikkaa on säilöttynä storessa, jonka muutokset muuttavat sovelluksen näkymää. Kuvassa 2.8 on kuvattu Flux-arkkitehtuurin datavirran periaate. [23]

2.4.1 Syntaksi ja komponentit

React pyrkii pitämään hahmontamis- ja käyttöliittymälogiikan mahdollisimman tiiviisti yhdessä. Tämän takia logiikka- ja käyttöliittymäkoodeja (esim. JavaScript ja HTML) ei tarvitse sijoittaa erillisiin tiedostoihin, vaan ne voivat sijaita samassa tiedostossa, jota kutsutaan komponentiksi. Koodien yhdistämisen apuna Reactissa on mahdollista käyttää JSX-syntaksia (JavaScript XML). JSX on JavaScript-syntaksin jatke, jota käytetään sovelluksen käyttöliittymän ulkonäön määrittelyyn. Sen käyttö ei kuitenkaan ole millään tavalla pakollista Reactissa, vaan se toimii lähinnä koodin visuaalisena apuna kehityksessä. [25] Kuvassa 2.9 on esitelty kaksi Reactilla luotua komponenttia, joiden määrittelyyn on käytetty apuna JSX-syntaksia.

```
// Luokkakomponentti
class Tervehdys extends React.Component {
  render() {
    return (
      <div>Moi {this.props.nimi}!</div> // JSX
    );
  }
}

// Funktiokomponentti
function Tervehdys(props) {
  return <div>Moi {props.nimi}!</div>; // JSX
}

ReactDOM.render(<Tervehdys nimi="Rasmus" />, document.getElementById("root"));
```

Kuva 2.9. *Saman luokka- ja funktiokomponentin määrittely JSX:n avulla.*

Reactilla on kaksi tapaa luoda komponentteja. Komponentit voivat olla joko luokka- tai

funktiopohjaisia. Kuvassa 2.9 on määritelty kaksi täysin samaa komponenttia, joista toinen on luotu käyttämällä JavaScriptin (ES6) luokkasyntaksia ja toinen JavaScriptin funktiota. Reactilla luotujen komponenttien mukana kulkeutuu olio, joka sisältää JSX-ominaisuuksia. Tätä oliota kutsutaan nimellä *props* ja sen avulla voidaan välittää tietoa parametrien tavoin komponenttien välillä. [25] Kuvassa 2.9 komponenteille annetaan niitä kutsuessa parametri "nimi", jota kutsutaan komponenttien sisällä käyttämällä *props*-oliota.

2.4.2 Komponentin tila ja elinkaarimetodit

React-komponenteilla on olemassa niiden sisäinen tila (engl. State), jota voidaan muuttaa esimerkiksi sovelluksen käyttäjän toimintojen seurauksena. Tilamuutokset voivat esimerkiksi aiheuttaa muutoksia sovelluksen käyttöliittymässä ja tekevät siitä vuorovaikutteisen. Kuvassa 2.9 olevat komponentit hyödyntävät *props*-oliota, jonka sisältämiä arvoja ei voi muuttaa. Tilan tarkoitus on sisältää dataa, jota voi muuttaa. Jokainen komponentin sisäinen tilamuutos aiheuttaa komponentin uudelleenhahmontamisen, jolloin sovelluksen näkymä pysyy ajan tasalla. [13] Kuvassa 2.10 edellistä luokkakomponenttia on muokattu niin, että sen hahmontaminen on riippuvainen sisäisestä tilasta eikä *props*ista.

```
class Tervehdys extends React.Component {
  constructor(props) {
    super(props);
    this.state = { nimi: "" };
  }

  componentDidMount() {
    this.setState({
      nimi: "Rasmus"
    });
  }

  render() {
    return <div>Moi {this.state.nimi}!</div>;
  }
}

ReactDOM.render(<Tervehdys />, document.getElementById("root"));
```

Kuva 2.10. *Tilallinen luokkakomponentti ja yksi elinkaarimetodi.*

Kuvassa 2.10 on käytetty myös yhtä Reactin tarjoamista elinkaarimetodeista (engl. Lifecycle method). Elinkaarimetodien tarkoitus on mahdollistaa tiettyjen toimintojen suorittaminen riippuen komponentin elinkaaren tilasta. *componentDidMount*-metodin avulla voidaan suorittaa sen sisään määritettyjä toimintoja heti sen jälkeen, kun komponentti on hahmonnettu DOMiin. [25] Kuvassa 2.10 komponentin tilan arvo "nimi" asetetaan aluksi tyhjäksi, mutta sille annetaan componentDidMount-metodin avulla oikea arvo heti, kun komponentti on hahmonnettu DOMiin.

3 TARKASTELTAVA SOVELLUSKEHITYSPROJEKTI

Tässä luvussa käydään läpi tämän diplomityön tausta ja tarkoitus. Kohdassa 3.1 kerrotaan miksi työssä käsiteltävä sovelluskehitysprojekti on tehty. Kohta 3.2 keskittyy projektikurssiin, jonka osana käsiteltävä web-sovellus on alun perin tehty ja ensimmäisen vaiheen kehitystiimiin. Toteutetun sovelluksen kehitysprosessi, tutkimus ja kirjoittajan rooli sovelluksen kehityksessä esitellään kohdassa 3.3

3.1 Asiakkaan tarve

Tässä diplomityössä tarkasteltava web-sovelluksen kehitysprojekti on toteutettu Cybercom Finland Oy:lle alun perin osana Tampereen teknillisen yliopiston (TTY) projektityökurssia. Asiakkaan alkuperäinen tavoite oli korvata uusien työntekijöiden perehdyttämiseen käytettävä paperinen tehtävälista -ratkaisu digitaalisella versiolla. Tämän ratkaisun toteutusta varten asiakas ehdotti projektia mahdolliseksi vaihtoehdoksi TTY:n projektityökurssille.

Ehdotettu sovellus oli nimeltään NewbieMaker. NewbieMakerin avulla uusi työntekijä (newbie) pystyy katselemaan ensimmäisten viikkojen perehdytykseen tarkoitettuja tehtäviä sekä merkitsemään niitä suoritetuiksi. Uudet työntekijät pystyvät myös antamaan joko yleistä tai tehtävään liittyvää palautetta. Sovellus toimii myös esimiesten työkaluna, koska sillä on mahdollista seurata alaisten edistymistä, luoda uusia tehtäviä sekä käyttäjiä ja lukea saatua palautetta. Asiakkaan toiveena oli, että kyseinen sovellus toteutettaisiin yhden sivun web-sovelluksena.

3.2 Projektikurssi

Kyseinen projektikurssi on osa Tampereen teknillisen yliopiston (nykyinen Tampereen yliopisto) ohjelmistotuotannon pääaineen kursseja. Kurssin tarkoituksena on, että opiskelijat pääsevät työskentelemään osana oikeaa työelämän projektia muistuttavaa projektia. Kurssilla on yhteistyöyrityksiä, joilla on ideoita erilaisista sovelluskehitysprojekteista, joihin he etsivät toteutustiimiä. Tässä työssä käsiteltävä sovellus on toteutunut osana vuoden 2018 projektityökurssia, jolloin projektitiimi, jossa tämän tutkimuksen tekijä oli osana, valitsi yhteistyöyrityksekseen Cybercom Finland Oy:n ja heidän sovellusprojektinsa NewbieMakerin.

Projektitiimi koostui kuudesta tietotekniikan maisterivaiheen opiskelijasta, joilla oli kaikilla

erilainen tausta sovelluskehitykseen. Eri taustat helpottivat tiimin sisäistä työnjakoa, sillä jokaisen tiimiläisen osaamista pystyttiin hyödyntämään tehokkaasti. Projektitiimin tavoitteena oli toteuttaa toimiva, käyttäjäystävällinen ja ylläpidettävä sekä helposti laajennettava sovellus. Tavoitteiden toteutumisen tutkiminen on osa tämän työn tutkimuskysymystä.

3.3 Tutkimus

Tässä diplomityössä pyritään selvittämään, miten NewbieMaker-sovelluksen kehityksen aikana tehdyt valinnat ovat vaikuttaneet sen kehitykseen. Kysymykseen pyritään saamaan vastaus tutkimalla toteutetun sovelluksen kolmea selkeää eri kehitysvaihetta. Ensimmäiseksi vaiheeksi lukeutuu alkuperäinen toteutus asiakkaalle TTY:n projektityökurssina. Toinen vaihe alkoi ensimmäisen vaiheen toteutuksen asiakkaalle luovuttamisen jälkeen, jolloin sovelluksesta toteutettiin uusi versio parannetuilla tekniikoilla. Tällöin sovelluksen kehitys tapahtui Cybercomin alla sisäisenä niin sanottuna Innovation Zone -projektina. Toisen vaiheen loputtua sen aikainen versio otettiin asiakkaalla pilottikäyttöön uusilla työntekijöillä ja heidän esimiehillään, jotka käyttivät sovellusta sen tarkoituksen mukaisesti. Kolmas eli (tällä hetkellä) viimeinen vaihe alkoi, kun projektin tila muutettiin viralliseksi sisäiseksi projektiksi. Kolmannessa vaiheessa pyrittiin korjaamaan toisen vaiheen toteutuksen puutteita sekä tekemään korjauksia pilottivaiheen käyttäjiltä saadun palautteen perusteella.

Tämän diplomityön tekijä on ollut mukana selkeissä eri rooleissa koko NewbieMaker-sovelluksen kehityksen ajan. Ensimmäisen kehitysvaiheen aikana hän oli mukana alkuperäisten teknologiavalintojen tekemisessä ja oli yksi sovelluksen selainpuolen kehittäjistä TTY:n kurssin puolesta. Toisessa kehitysvaiheessa työn tekijä siirtyi täysipäiväiseksi NewbieMakerin kehittäjäksi Cybercom Finland Oy:lle. Tässä vaiheessa hän oli yksin vastuussa sovelluksen teknisestä toteutuksesta eli uusien toiminnallisuuksien toteuttamisesta. Kolmannen kehitysvaiheen ajan tutkimuksen tekijä on toiminut teknisenä apuna sovelluksen uusille kehittäjille. Hänen toimenkuvaansa on kuulunut muun muassa sovelluksen arkkitehtuurin ja toiminnallisuuden selittäminen uusille kehittäjille sekä mahdollisten teknologisten uudistusten ja refaktorointien ideointi. Kirjoittaja on siis ollut tiiviisti mukana NewbieMakerin kehityksessä koko sen elinkaaren ajan.

Jokainen vaihe ja niissä tehdyt valinnat on kuvattu erikseen tämän työn luvuissa 4, 5 ja 6. Luvussa 7 keskitytään diplomityön tutkimuskysymykseen eli eri vaiheissa tehtyjen valintojen vaikutukseen kehitysprosessissa. Sovellusta tullaan jatkokehittämään ja käyttämään myös tulevaisuudessa, joten on myös tärkeää ottaa kantaa nykyisen toteutuksen ylläpidettävyyteen. Työn avulla pyritään tunnistamaan miten tietyt teknologiset valinnat vaikuttavat sovelluskehitykseen ja millaisia valintoja tulevaisuudessa kannattaa tehdä, jotta sovelluksen kehitysprosessi olisi tehokkaampi.

4 ENSIMMÄINEN KEHITYSVAIHE

Tässä luvussa käydään läpi tarkasteltavan sovelluskehitysprojektin ensimmäisen vaiheen aikana tehdyt teknologiset sekä projektihallinnalliset valinnat. Sovelluksen teknologiapinoksi valikoitui niin sanottu MERN-pino eli MongoDB, Express, React ja Node.js. React-kirjastosta on kerrottu aiemmin luvussa 2, mutta uudet MERN-pinon teknologiat esitellään tässä luvussa. Kohta 4.1 keskittyy projektinhallinnallisiin seikkoihin, selainpuolen valitut teknologiat esitellään kohdassa 4.2 ja palvelinpuolen teknologiat kohdassa 4.3. Ensimmäisen kehitysvaiheen aikana käytetyistä testaus- ja laadunvarmistusmenetelmistä kerrotaan kohdassa 4.4. Lopuksi kohdassa 4.5 esitellään aikaansaatu web-sovellus.

4.1 Tavoitteet ja kehityksen työkalut

Projektin ensimmäisen vaiheen korkean tason tavoitteiksi asetettiin seuraavat asiat:

1. Toimiva sovellus.
2. Käyttäjäystävällinen sovellus.
3. Helposti jatkokehitettävä ja ylläpidettävä sovellus.
4. Helppo palautteen anto ja edistyksen seuraaminen.

Toimivalla sovelluksella tarkoitetaan julkiseen verkkoon tuotantoon vietyä tuotetta, jonka toiminnallisuudet ovat tavoitteiden mukaiset tai vähimmäisvaatimuksena riittävät. Sovelluksen toimintavaatimuksiin lukeutui muun muassa seuraavat:

1. Esimies pystyy luomaan ja poistamaan käyttäjiä ja tehtäviä.
2. Esimies pystyy seuraamaan alaistensa tehtävien etenemistä.
3. Esimies pystyy lukemaan saatua palautetta.
4. Newbie pystyy lukemaan ja suorittamaan tehtäviä.
5. Newbie pystyy lähettämään palautetta esimiehelle.

Käyttäjäystävällisyyden saavuttamiseksi kehitystiimin UI/UX-vastaavat pyrkivät luomaan käyttöliittymäkuvia jo aikaisessa vaiheessa. Myös CSS-viitekehyksen käyttöönoton avulla helpotettiin hienomman ja selkeämmän käyttöliittymän luontia. Sovelluksen teknologiavalinnoilla pyrittiin edesauttamaan sen jatkokehityksen ja ylläpidon helppoutta. Palautteen anto ja edistyksen seuraaminen lukeutui alemman prioriteetin toiminnallisuusvaatimuksiin.

NewbieMakerin ensimmäisen vaiheen tavoitteiden saavuttamisen apuna kehitystiimi noudatti ketterää projektinhallinnan viitekehystä scrumia. Scrum on yleisesti ohjelmistokehityksessä käytetty joustava projektinhallintamalli, joka sopeutuu erityisen hyvin ennaltaarvaamattomiin projekteihin. Scrum perustuu iteraatioihin ja sen tavoitteisiin kuuluu korkealaatuisten ohjelmistotuotteiden toimittaminen ennaltamääritettyjen ajanjaksojen kulumisen jälkeen. Näitä ajanjaksoja kutsutaan sprinteiksi, joiden pituus vaihtelee yleensä kahdesta viikosta kuukauteen. [26] NewbieMaker-projektissa noudatettiin kehitystiimin valitsemaa kolmen viikon sprint-pituutta.

Scrum-tiimissä on kolme pääroolia: tuoteomistaja (engl. Product owner), scrummaster (engl. Scrum master) ja kehitystiimi (engl. Development team). Tuoteomistaja huolehtii kehitystiimin tekemän työn arvon maksimoinnista. Tuotteen kehitysjonosta (engl Product backlog) huolehtiminen kuuluu myös tuoteomistajan tehtäviin. Kehitysjonolla tarkoitetaan järjestettyä listaa kaikista tuotteen tarvitsemista ominaisuuksista. [26]

Scrummasterin tehtäviin kuuluu varmistaa, että scrum-tiimi ja scrum-tiimin ulkopuoliset henkilöt ovat tietoisia scrum-periaatteista. Scrummaster auttaa tuoteomistajaa muun muassa varmistamalla, että kehitystiimi on tietoinen projektiin liittyvistä tavoitteista ja tuotteen alasta. [26]

Kehitystiimi vastaa siitä, että jokaisen sprintin jälkeen voidaan toimittaa potentiaalisesti toimiva tuote, joka on lähempänä valmista. Kehitystiimi vastaa itse omasta järjestäytymisestään ja työn hallinnasta. [26]

Versionhallinnan työkaluna projektin ensimmäisessä vaiheessa käytettiin Tampereen teknillisen yliopiston tarjoamaa GitLab-palvelua. GitLab on palvelu, joka tarjoaa muun muassa Git-repositorion hallintatyökalun ja sisäänrakennetun jatkuvan integraation (engl. CI, Continuous integration) ja jatkuvan julkaisun (engl. CD, Continuous Delivery) ratkaisun, jota hyödynnettiin NewbieMakerin kehitysprosessissa, mistä kerrotaan enemmän kohdassa 4.4.

4.2 Valitut selainpuolen teknologiat

Projektitiimi oli melko yhtenäinen teknologioiden valitsemisen suhteen. Koska tiedettiin, että toteutettava tuote tulee olemaan web-sovellus, osattiin teknologiavaihtoehdot rajata kahteen ryhmän jäsenille ennalta tuttuun vaihtoehtoon: JavaScript (React) ja Python (Django). Kaikilla ryhmän jäsenillä oli molemmista ohjelmointikielistä jonkinlaista kokemusta, mutta suuren suosionsa takia JavaScript ja React-kirjasto (esitelty luvussa 2) valittiin selainpuolen teknologioiksi.

Vaikka tiimin tehtävänä olikin tuottaa sovellus SPA-toteutuksena, täytyy sen reititys silti hoitaa jollain tapaa. Valinta oli helppo, koska selainpuoli valittiin toteutettavaksi Reactin avulla pystyttiin reitityksen apuna käyttämään helposti React Router nimistä kirjastoa.

4.2.1 React

Ensimmäisen vaiheen tavoitteena oli luoda kokonainen, toimiva ja käytettävä web-sovellus yrityskäyttöön. Tällaisen sovelluksen toteuttaminen ilman helpottavia tekijöitä, kuten viitekehyksiä tai kirjastoja olisi tehotonta. Sovelluksen arkkitehtuuria suunnitellessa kehitystiimillä oli selkeä kiinnostus suosittua ja koko ajan kasvavaa JavaScript selainpuoli -kirjastoa, Reactia kohtaan, joka päätyi selainpuolen kehityksen pääteknologiaksi.

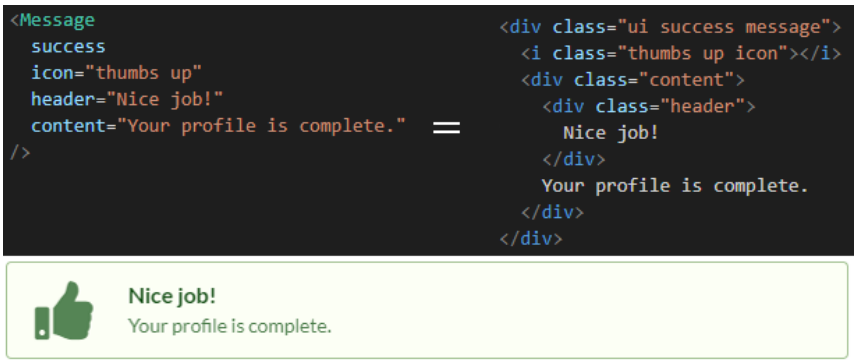
Projektin alussa uusin React-versio oli 16.5.2, jonka toiminnallisuuksilla ja rajoitteilla sovelluksen selainpuoli toteutettiin. React-versiossa 16.5.2 sovelluksen komponenttien sisäisen tilan hallinta ja elinkaarimetodien käyttö oli mahdollista vain luokkasyntaksilla luoduilla komponenteilla. Tämän takia NewbieMaker-sovelluksen ensimmäinen versio toteutettiin lähes kokonaisuudessaan pelkillä luokkakomponenteilla.

4.2.2 Semantic UI React -viitekehys

Käyttöliittymän toteutuksen helpottamista varten ryhmä sai vinkin asiakkaan teknisen puolen tukihenkilöltä, joka suositteli käyttöönotettavaksi Semantic UI React -nimisen CSS-viitekehyksen. CSS-viitekehykset helpottavat ja nopeuttavat käyttöliittymien luontia tarjoamalla valmiita komponentteja, jotka sisältävät tarvittavat HTML- ja CSS-dokumentit interaktiivisten ja responsiivisten käyttöliittymien rakentamiseen [27]. Sovelluskehitysprosessi hyötyy CSS-viitekehysten käytöstä usealla eri tavalla:

1. Samojen komponenttien käyttö yhtenäistää koodia ja sen rakennetta.
2. Käyttöliittymistä tulee yhteensopivia eri selainten välillä.
3. Sovellusten ylläpito, muokkaus ja laajentaminen helpottuu.
4. Ruudukko-pohjainen (engl. Grid-based) suunnittelumalli helpottaa käyttöliittymän suunnittelua ja asettelua. [27]

CSS-viitekehysten avulla web-sovelluskehittäjät välttävät niin sanotun pyörän uudelleen keksimisen, koska kaikkia sovelluksen osia ei tarvitse toteuttaa alusta alkaen. Suosittuja CSS-viitekehyksiä on olemassa useita, kuten Bootstrap, Foundation [27] ja tässä työssä käsiteltävässä projektissa käytetty Semantic UI React.



Kuva 4.1. *Semantic UI Reactilla luotu komponentti ja sitä vastaava HTML-koodi.*

Kuvassa 4.1 on esitelty yksinkertaisen *Message*-komponentin luonti Semantic UI React -viitekehyksen avulla. Kuvasta huomaa, että Semantic UI Reactilla luodun komponentin (kuvassa vasemmalla) syntaksi on huomattavasti yksinkertaisempaa ja helpommin ymmärrettävää verrattuna saman komponentin luomiseen pelkän HTML:n avulla (kuvassa oikealla). Huomattavaa on myös, että kyseisen CSS-viitekehyksen avulla on yksinkertaista luoda tyylikkäitä käyttöliittymäkomponentteja vähällä koodimäärällä.

4.2.3 React Router

Vaikka SPA-sovellukset koostuvat nimensä mukaan vain yhdestä sivusta, on niissä silti usein monia eri loogisia näkymiä. Sovelluksen reititys on olennainen osa toteutusta, jotta käyttäjien navigointi näiden näkymien välillä olisi mahdollisimman helppoa. Ilman reititystä koko sovelluksella ja sen jokaisella näkymällä olisi vain yksittäinen URL. Tällöin navigointi näkymien välillä tapahtuisi aina käyttöliittymän kautta tiettyjen toimintojen seurauksena. Käyttäjällä ei olisi mahdollisuutta päästä tiettyyn näkymään esimerkiksi suoraan linkin kautta. [13]

SPA-sovelluksissa reititys tapahtuu linkittämällä sovelluksen tila selaimessa näkyvään URLiin. Hyödyt reitityksen linkittämisestä URLiin ovat huomattavat:

1. Navigointi näkymien välillä onnistuu käyttämällä selaimen takaisin- ja eteenpäin-painikkeita.
2. Tiettyjen näkymien jakaminen muille käyttäjille helpottuu, koska muut käyttäjät pääsevät näkymään käsiksi suoraan URLin avulla. [13]

Näkymän ja selaimen välillä on oltava jokin yhdistävä tekijä, jotta reititys olisi mahdollista. Yhdistävän tekijän luomiseen on yleisesti ottaen SPA-toteutuksissa kaksi toteutustapaa: hash-linkitys ja selainhistoria. **Hash-linkityksessä** näkymän sijainti ilmoitetaan URLin ankkuriosuudessa #-merkin jälkeen. Koska SPA-toteutuksissa on vain yksi sivu, on hash-linkityksellä helppo viitata mihin tahansa osaan sivua, joka on tällä hetkellä näkyvissä. **Selainhistoria** hyödyntää HTML5 APIa, joka luovuttaa näkymien välisen navigoinnin JavaScriptille, joka huolehtii myös ettei sivun uudelleenlatauksia tapahdu URLin vaihtuessa. [13] NewbieMaker-sovelluksessa reitityksen käytettiin selainhistoria-lähestymistapaa.

4.3 Valitut palvelinpuolen teknologiat

Palvelinpuolen teknologiavalinnat vaativat osittain hieman suurempaa pohdintaa. Palvelimen toteutuksessa oli helppo päätyä valitsemaan Node.js ja Express-viitekehys, koska web-sovelluksen selainpuoli toteutettiin myös JavaScriptillä. Täten selain- ja palvelinpuoli toteutettiin samalla ohjelmointikielellä, joka helpotti kehittäjien työskentelyä.

Tietokantavalintaan liittyen oli kaksi eri vaihtoehtoa: jokin relaatiotietokanta tai NoSQL-tietokanta. Päätös vaati enemmän pohdintaa, sillä tiimillä ei ollut vielä täyttä varmuutta tulevasta tietokantarakenteesta. Epävarmuus tietokantarakenteesta sai loppujen lopuksi

päätöksen kallistumaan NoSQL-tietokantaan ja tarkemmin MongoDB:hen sen joustavuuden helpomman lähestyttävyyden takia.

Kehittäjät päättivät käyttää NewbieMaker-sovelluksen tietokannan hostaukseen mLab-pilvipalvelua. mLab tarjoaa niin sanottua database as a service (DaaS) -palvelua. mLabin avulla kehittäjät pystyivät hyödyntämään sovelluksen tietokantana pilvessä hostattua MongoDB-tietokantaa sen sijaan, että jokainen kehittäjä olisi hostannut omaa tietokantaansa lokaalisti. Tämä mahdollisti sen, että pilvessä oli mahdollista ylläpitää aina uusinta toimivaa versiota tietokannasta, jota kehittäjät pystyivät käyttämään referenssinä kehityksessä.

4.3.1 Node.js ja Express

Node.js on niin sanottu JavaScript runtime-ympäristö, jonka avulla mahdollistetaan JavaScript-koodin suorittaminen selaimen ulkopuolella eli palvelimella. Node.js on alustariippumaton ja se pohjautuu Chromen V8 JavaScript-moottoriin. Selainpuolella HTML-sivu yhdistää JavaScript-tiedostot, mutta palvelimella ei tällaista HTML-sivua ole. JavaScript-tiedostojen kutsuminen ja liittäminen toisiinsa palvelimella perustuu moduuleihin. Node.js perustuu asynkroniseen ja tapahtumapohjaiseen malliin yhtäaikaisen toiminnan saavuttamiseksi säikeiden käytön sijaan. Takaisinkutsufunktiot (engl. Callback function) mahdollistavat asynkronisen toiminnan. Esimerkiksi funktiolle, joka avaa tietyn tiedoston voidaan antaa parametrina takaisinkutsuntafunktio, jota kutsutaan vasta tiedoston avauduttua. Koodin suorittamista, joka ei riipu avatusta tiedostosta voidaan jatkaa vapaasti, jolloin luodaan mielikuva rinnakkaisuudesta. Tapahtumasilmukka (engl. Event loop) on jono prosessoitavia tapahtumia ja niiden takaisinkutsuntafunktioita ja se huolehtii moniajosta (engl. Multitasking) Node.js ympäristössä. [13]

Node.js tuo mukanaan kolmannen osapuolen kirjastojen hallintaan käytettävän työkalun nimeltä npm (engl. Node Package Manager). Npm oli alun perin Node.js:n moduuleille tarkoitettu repositorio, mutta nykyään sitä käytetään useiden JavaScript-pohjaisten moduulien toimittamiseen. [13] Tässä projektissa npm:ää käytettiin muun muassa selainpuolen kirjastojen (esim. React) ja apukirjastojen asentamiseen ja hallintaan. [13]

```
const http = require("http");
const port = 3000;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader("Content-Type", "text/plain");
  if (req.url === "/") {
    res.write("Hello World!");
    res.end();
  }
  res.end();
});

server.listen(port, () =>
  console.log(`Example app listening on port ${port}!`)
);
```

```
const express = require("express");
const app = express();
const port = 3000;

app.get("/", (req, res) => res.send("Hello World!"));

app.listen(port, () =>
  console.log(`Example app listening on port ${port}!`));
```

Kuva 4.2. *Palvelimen ja yhden reitin luontiin vaadittava koodimäärä puhtaalla Node.js:llä ja käyttämällä Express-viitekehystä.*

Palvelinlogiikan toteuttamisen nopeuttamiseksi projektissa otettiin käyttöön Node.js:n rinnalle Express-niminen web-viitekehys. Expressin avulla voidaan määrittää toimintoja, jotka suoritetaan, kun ennalta määritellyssä muodossa oleva HTTP-pyyntö saapuu palvelimelle. Express helpottaa palvelinpuolen koodin kirjoittamista myös tarjoamalla toimintoja, kuten pyyntöjen URLin, ylätunnisteen (engl. Header) ja parametrien parsimisen ja väliohjelmistojen (engl. Middleware) käytön esimerkiksi virheiden hallinnassa. [13] Kuvassa 4.2 on vasemmalla esitelty palvelimen ja yhden reitin määrittelyyn vaadittava koodi pelkän Node.js:n avulla ja oikealla sama käyttämällä apuna Express-viitekehystä.

4.3.2 MongoDB ja Mongoose

Sovellukseen valittu tietokanta MongoDB on dokumenttipohjainen (engl. Document-based, Kuva 2.4) NoSQL-tietokanta. MongoDB:n dokumentit vertautuvat relaatiotietokannan riveihin ja koostuvat kenttä-arvo -pareista (engl. field-value pair) ja muistuttavat pitkälti JSON-olioita. Dokumentit tukevat useita eri datatyyppejä, kuten totuusarvomuuttujia (engl. Boolean), merkkijonoja (engl. String), päivämääriä (engl. Date), aikaleimoja (engl. Timestamp) jne. [13] Kuvassa 4.3 on esitelty yksi NewbieMakerin käyttäjän dokumentti tietokannasta. Huomattavaa on se, että käyttäjän salasanaa ei tallennettu tietokantaan selkotekstinä, vaan hashina joka saatiin käyttämällä bcrypt-kirjastoa.


```
{
  "_id": { "$oid": "5cd2a7174e7b0b317c92c18e" },
  "isSupervisor": false,
  "name": "Rasmus Lempinen",
  "firstDay": { "$date": { "$numberLong": "1559911650022" } },
  "username": "rasmusl",
  "password": "$2b$10$NIfzTze3WOalzkmgIbLMaOFio7Xl.Q2WWgfLFQSNeXi47K6/dpKAy",
  "mySupervisor": null,
  "tasks": [],
  "nearbyColleagues": []
}
```


Kuva 4.3. *Yksittäisen käyttäjän JSON-muotoinen dokumentti NewbieMaker-sovelluksen tietokannasta.*

Tietokannan dokumentit sijaitsevat kokoelmissa (engl. Collection), joita voi verrata relaatiotietokannan tauluihin. MongoDB ei pakota käyttämään tiettyä tietty skeemaa (engl. Schema), jolloin kaikkien kokoelman dokumenttien ei tarvitse sisältää keskenään samoja kenttiä. Tämä helpottaa tietokannan kehittämistä varsinkin alkuvaiheissa, kun tietokannan rakenne saattaa muuttua usein. Skeeman määrittely kuitenkin auttaa tietokannan järkevänä pitämisessä ja helpottaa isompien projektien kehitystä. Skeeman avulla voidaan määritellä esimerkiksi dokumentin kenttiä pakollisiksi tai sisältämään tietyn minimimäärän merkkejä. Kuvassa 4.4 on määritelty user-dokumentin kenttä "username" pakolliseksi sekä uniikiksi. [13] NewbieMaker-projektissa kehitystiimi valitsi kokoelmien skeeman luonnin ja hallinnan apuvälineeksi kirjaston nimeltä Mongoose. Kuvassa 4.4 on esitelty NewbieMaker-sovelluksessa Mongoosen avulla määritelty user-skeema.

```
const userSchema = mongoose.Schema({
  name: String,
  title: String,
  username: { type: String, required: true, unique: true },
  password: String,
  isSupervisor: { type: Boolean, default: false },
  tasks: { type: [userTaskSchema], default: [] },
  firstDay: { type: Date, default: Date.now },
  mySupervisor: { type: mongoose.Schema.Types.ObjectId, ref: 'User' },
  nearbyColleagues: [
    { type: mongoose.Schema.Types.ObjectId, ref: 'User', default: [] }
  ]
});
```

Kuva 4.4. *Mongoosella määritelty user-skeema NewbieMaker-sovelluksen käyttäjille.*

MongoDB oli suhteellisen helppo valinta NewbieMakerin tietokannaksi, koska sen kyselykielenä toimii relaatiotietokantojen SQL:n sijaan JSON-pohjainen kyselykieli. Kaikki MongoDB:n toiminnot (luku, luonti, poisto) tehdään määrittelemällä toiminto JSON-olioon. Tietokannasta saatu tieto on myös JSON-muodossa [13]. Tämä tarkoitti sitä, että koko sovelluksen teknologiapino oli ohjelmointikieleltään yhtenäinen. Selainpuolen, palvelinpuolen ja tietokannan kehitys tapahtui kaikki JavaScriptillä, joka yksinkertaisti kehittämistä ja yhtenäisti koodia.

4.4 Testaus ja laadunvarmistus

NewbieMaker-projektin testaus keskittyi pääosin sovelluksen käyttöliittymä- (selainpuoli), yksikkö- ja rajapintatestaukseen (REST API). Rajapintatestauksella pyrittiin varmistamaan sovelluksella tehtäväien tietokantalisäysten, -muokkausten ja -poistojen toiminnallisuus. Testaus toteutettiin jokaisen kehittäjän käyttämää omaa tietokantaa vastaan sen sijaan, että olisi käytetty yhteistä testikantaa, koska yhteisen pilvessä hostatun tietokannan käyttö yhteiskehityksessä olisi tuottanut ongelmia.

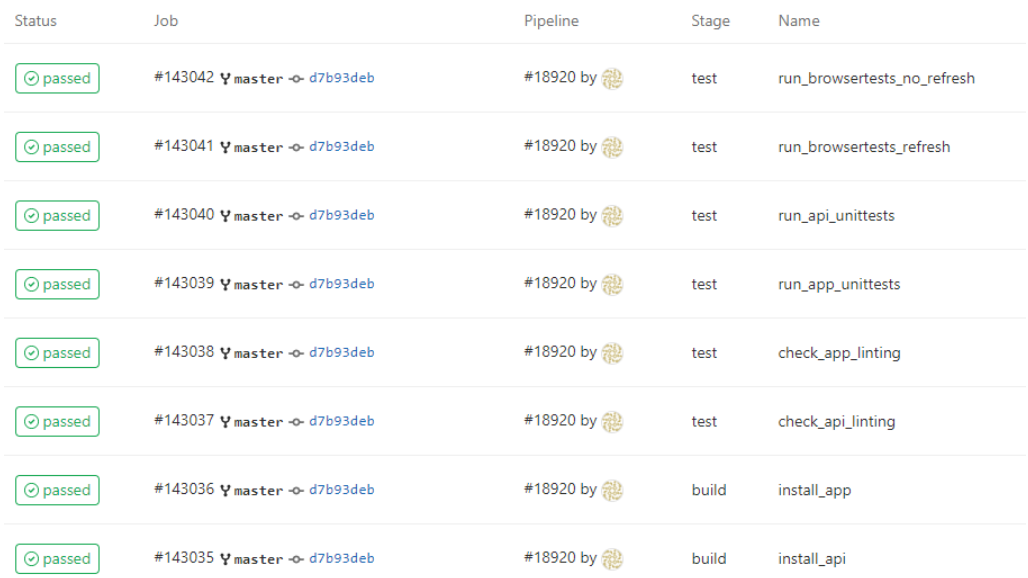
Molemmat käyttöliittymä- ja rajapintatestaus suoritettiin hyödyntämällä GitLabin tarjoamaa jatkuvan integraation *pipeline*a *Jenkis*-työkalun avulla. Pipelinen avulla testit pystyttiin ajamaan automaattisesti joka kerta, kun kehittäjä puski uutta koodia versionhallintaan. Jokaisella puskulla suoritettiin myös niin sanottu "linttaus" (engl. Linting) eli koodin syntaksi- ja tyylitarkistus. Tämä auttoi pitämään koodin tyylin yhtenäisenä riippumatta kehittäjien käyttämästä koodieditorista. Automaattitestit helpottivat koodin toiminnallisuuden ja luettavuuden ylläpitoa huomattavasti. Rajapintatestaus toteutettiin käyttämällä Jest- ja Supertest-kirjastoja, käyttöliittymätestaus Robot Frameworkilla ja syntaksin sekä tyylin tarkistus ESLint-säännöillä.

Jest on Facebookin ylläpitämä JavaScript testaus viitekehys, jota käytetään JavaScript-koodin oikeellisuuden varmistamiseen. Jest priorisoi edellisillä testikerroilla epäonnistuneet testit ajamalla ne ensimmäisenä ja järjestää testit niiden ajallisen pituuden mukaan.

[28]

Robot Framework on avoimen lähdekoodin python-pohjainen syntaksiltaan helppo työkalu, jota käytetään avainsanoihin perustuvassa automaattitestauksessa. Se on riippuvuusvapaa käyttöjärjestelmistä ja sovelluksista. Robot Framework valikoitui NewbieMakerin käyttöliittymätestauksen työkaluksi yhden kehittäjätiimin kokemuksen perusteella. [29]

ESLint on Node.js:llä toteutettu staattisen analyysin työkalu, minkä avulla koodista voidaan löytää tyyli- ja koodivirheitä. JavaScript on dynaaminen ja löysästi tyypitetty kieli, jonka syntaksivirheet ilmenevät vasta koodia suoritettaessa. ESLintin avulla kehittäjät voivat löytää virheitä koodistaan suorittamatta sitä. Kehittäjät voivat myös ESLintin avulla luoda täysin omat koodin tyylisäännöt, joita kehitysprojektissa noudatetaan. [30]

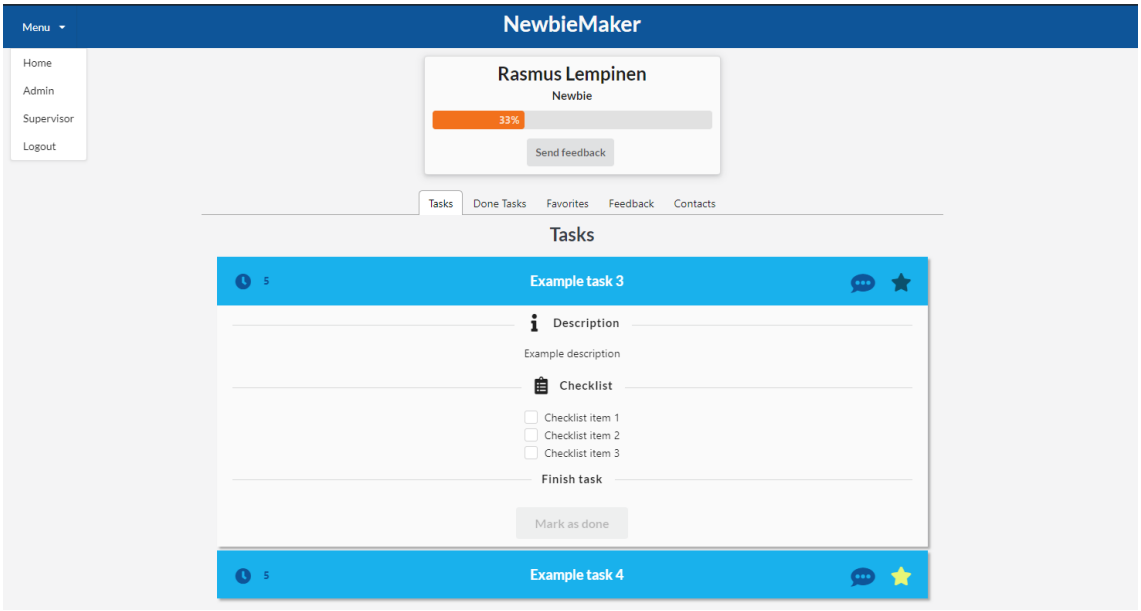


Kuva 4.5. *NewbieMaker-sovelluksen jatkuvan integraatioputken työt.*

Jokainen projektin Jenkinsillä luotu pipeline sisälsi kahdeksan erillistä työtä (engl. Job), mitkä on esitelty kuvassa 4.5. Pipeline jakautui kahteen eri vaiheeseen (kuvassa *stage*: *build* ja *test*. Build-vaiheessa asennettiin aina ensiksi sovelluksen selain- ja palvelinpuolen riippuvuudet, jonka jälkeen siirryttiin test-vaiheeseen. Jokainen test-vaiheen työ suoritettiin niin ikään selain- ja palvelinpuolelle lukuun ottamatta käyttöliittymätestejä. Test-vaiheen alussa tehtiin linttauksen tarkistus, jonka jälkeen suoritettiin yksikkötestit, jotka sisälsivät myös rajapintatestauksen. Viimeiset työt sisälsivät käyttöliittymätestauksen kahdella eri tavalla: päivittämättä ja päivittämällä sivu erikseen määritettyjen toimintojen jälkeen. Build ja test -vaiheiden läpimeneminen oli vaatimus koko pipelinen onnistumista varten.

4.5 Toteutunut sovellus

Projektitiimi luovutti ensimmäisessä vaiheessa toteutetun sovelluksen asiakkaalle 1.2.2019. Kohdassa 4.1 luetellut ensimmäisen vaiheen tavoitteet toteutuivat melko hyvin. Sovellus julkaistiin tuotantoon virtuaalikoneelle Microsoftin Azure -pilvipalveluun. Kaikki korkeimmalla prioriteetilla olleet toiminnallisuudet, kuten myös helppo palautteen anto ja edistyksen seuraaminen saatiin toteutettua ja toteuttamatta jäi vain yksi kehitysjonossa ollut toiminnallisuus.



Kuva 4.6. *NewbieMaker-sovelluksen ensimmäisen kehitysvaiheen jälkeinen näkymä.*

Pikaisen käytettävyystestauksen jälkeen sovelluksen käyttäjäystävällisyys todettiin asiakkaan puolesta myös riittäväksi. Käyttäjäystävällisyyden todellista tasoa oli kuitenkin hankala arvioida, koska sovelluksella ei ollut virallisia testikäyttäjiä kehitystiimin ulkopuolelta. Asiakas ei ollut myöskään vielä tässä vaiheessa nähnyt sovelluksen lähdekoodia, joten sitä he eivät pystyneet arvioimaan.

Kuvassa 4.6 on esitelty yksi NewbieMaker-sovelluksen ensimmäisen kehitysvaiheen jälkeisistä näkymistä. Sovelluksen käyttöliittymän ulkonäkö luotiin käytännössä kokonaan ensimmäisen kehitysvaiheen viimeisen kolmen viikon sprintin aikana. Tämän takia käyttöliittymästä huomaa ammattimaisen UI/UX-henkilön näkemyksien puutteen.

Suurimmaksi kysymysmerkiksi toteutuksen tavoitteista jäi sovelluksen ylläpidon ja jatkokehityksen helppous. Sovelluksen selainpuoli toteutettiin Reactilla ilman tilanhallintaan käytettävää kirjastoa, josta seuraa todennäköisesti ongelmia sovelluksen koon kasvaessa. Myös JavaScriptin valinta kehityskielenä on altis kehittäjien virheille sen korkean dynaamisuuden ja löysän tyypityksen takia. Ensimmäisen kehitysvaiheen teknologiavalintojen vaikutuksia ja niiden seurauksia käydään läpi tarkemmin luvussa 7.

5 TOINEN KEHITYSVAIHE

Tässä luvussa käydään läpi NewbieMaker-sovelluksen toisen kehitysvaiheen teknologiset ja projektinhallinnalliset valinnat. Toinen kehitysvaihe alkoi, kun ensimmäinen versio sovelluksesta luovutettiin Cybercom Finlandin haltuun. Kohta 5.1 pyrkii kertomaan sovelluksen puutteista, joita jäi ensimmäisen kehitysvaiheen jäljiltä. Kohdassa 5.2 kerrotaan sovelluksen uudesta kehitysprosessista ja -tiimistä. Kohta 5.3 keskittyy uusiin teknologiavalintoihin, joilla pyrittiin parantamaan sovelluksen ylläpidettävyyttä ja ulkonäköä. Toisen kehitysvaiheen refaktoroitu sovellus esitellään kohdassa 5.4.

5.1 Ensimmäisen toteutuksen puutteet

Johtuen pitkälti ensimmäisen vaiheen kehitystiimin kokemattomuudesta, sovelluksen lähdekoodi sisälsi todella paljon turhaa ja tyyliltään huonoa koodia. Toisen kehitysvaiheen alussa Cybercomin kokeneempi sovelluskehittäjä kävi läpi NewbieMakerin lähdekoodin ja teki listan tarvittavista muutoksista itse koodiin sekä teknologioihin liittyen.

Lähdekoodi sisälsi muun muassa pienempiä tilattomia komponentteja, jotka oli toteutettu luokka-pohjaisina. Näiden refaktorointi funktiokomponentteihin olisi oleellista, jotta ylimääräisestä koodista pääsisi eroon. Myöhemmin toisen kehitysvaiheen aikana myös tilallisia luokkakomponentteja refaktoroitiin funktiokomponenteiksi alakohdassa 5.3.2 esiteltävien React Hookien avulla. Jotkin sovelluksen komponentit sisälsivät myös todella paisuneita tilamäärittelyjä, joiden ylläpito ja muokkaus olisi jatkossa todella haastavaa. Näille paisuneille komponenteille oli myös tyypillistä sisältää paljon funktionaalista koodia. Reactin periaatteiden mukaan on hyvä pyrkiä pitämään komponentit mahdollisimman kuvaavina ilman ylimääräistä funktionaalista koodia. Sovelluksen tyylittelyt sisälsivät myös puutteita, sillä ne oli määritelty yksinomaan yhteen CSS-tiedostoon. Tämän takia kyseinen CSS-tiedosto sisälsi tarpeettoman paljon ylimääräistä sekä huonolaatuista koodia.

5.2 Uusi kehitysprosessi ja tavoitteet

NewbieMaker-sovelluksen toinen kehitysvaihe alkoi sen jälkeen, kun ensimmäisen vaiheen kehitystiimi luovutti sovelluksen lähdekoodin Cybercom Finlandille 1.2.2019. NewbieMakerin kehitys jatkui tällöin Cybercomin sisäisenä Innovation Zone -projektina. Koska Cybercom on konsulttitalo, toimii sisäisten projektien kehittäjinä usein työtekijät, jotka

ovat niin sanotusti projektien välissä ilman sen hetkistä asiakastyötä. Toinen kehitysvaihe sisälsikin sovelluskehittäjien vaihdoksia muutamaan kertaan.

Toisen vaiheen kehitysprosessin aloitti Cybercomin sovelluskehittäjäkonsultti, joka aloitti NewbieMaker-sovelluksen selainpuolen refaktoroinnin uusien teknologiavalintojen perusteella. Ensimmäisen refaktoroinnin tuloksena oli toiminnallisuuksiltaan todella alkeellinen runko. Runko oli kuitenkin toteutettu niin, että sen jatkokehittäminen ja sovelluksen toiminnallisuuksien lisääminen oli melko suoraviivaista.

Ensimmäisen refaktoroinnin jälkeen NewbieMaker-sovelluksen kehittämistä jatkettiin Cybercomilla vasta-aloittaneen sovelluskehittäjäharjoittelijan toimesta yhtäjaksoisesti noin neljän kuukauden ajan. Kehitysprosessissa jatkettiin ensimmäisessäkin vaiheessa käytetyn scrum-viitekehyksen noudattamista. Prosessin erot ensimmäiseen vaiheeseen olivat kuitenkin melko suuret, sillä varsinaisia kehittäjiä sovelluksella oli tässä vaiheessa vain yksi kuuden sijaan. Kehittäjä kuitenkin toimi kokopäiväisenä, joten käytetty viikoittainen kehitysaika ei pudonnut merkittävästi. Projektissa noudatettiin kahden viikon sprinttejä ja tuoteomistajana sekä scrummasterina toimi Cybercomin UI/UX-suunnittelija, joka suunnitteli myös sovelluksen uuden käyttöliittymän.

Toisen kehitysvaiheen selkeänä päätavoitteena oli tarpeeksi toimivan tuotteen luominen, jotta se voitaisiin ottaa pilottikäyttöön. Tarkoituksena oli viedä sovellus tuotantoon elokuun 2019 loppuun mennessä, jolloin Cybercomilla aloittaisi uusia työntekijöitä, joilla tuotetta voitaisiin testata. Sovelluksen käyttäjille näkyvimpänä tavoitteena oli myös käyttöliittymän uusiminen Cybercomin UI/UX-ammattilaisen näkemysten perusteella.

5.3 Uudet teknologiat

NewbieMaker-sovelluksen toisen kehitysvaiheen uudet teknologiavalinnat tehtiin ensimmäisen vaiheen toteutuksen suurimpien puutteiden ja toisen kehitysvaiheen tavoitteiden pohjalta. Uudet teknologiavalinnat painottuvat lähes yksinomaan selainpuoleen, koska ensimmäisen toteutuksen palvelinpuolen toiminnallisuus ja laatu oli tämän vaiheen tavoitteita varten riittävä. Kaikista tämän vaiheen uusista teknologioista kerrotaan enemmän tämän kohdan alakohdissa.

Mahdollisten kehittäjän tekemien koodivirheiden minimoimiseksi ja koodikannan ylläpidon helpottamiseksi selainpuolen toteutuskieli vaihdettiin JavaScriptistä TypeScriptiin. NewbieMakerin ensimmäisen version selainpuoli oli toteutettu React versiolla 16.5.2, joka pohjautui vielä vahvasti luokkakomponentteihin. React esitteli kuitenkin uudessa React 16.8 -versiossaan React Hookit, joita hyödynnettiin toisessa kehitysvaiheessa suuresti. Tilanhallinta oli yksi ensimmäisen toteutuksen suurimmista ongelmista/kysymysmerkeistä sovelluksen jatkokehitystä ajatellen. Tämän yksi toisen vaiheen suurimmista muutoksista oli Redux-kirjaston käyttöönotto sovelluksen tilanhallinnan helpottamiseksi. Sovelluksen käyttöliittymästä ja käyttäjäkokemuksesta vastaava henkilö päätti myös vaihtaa ensimmäisessä versiossa käytetyn Semantic UI React -viitekehyksen uuteen, Material UI -viitekehykseen.

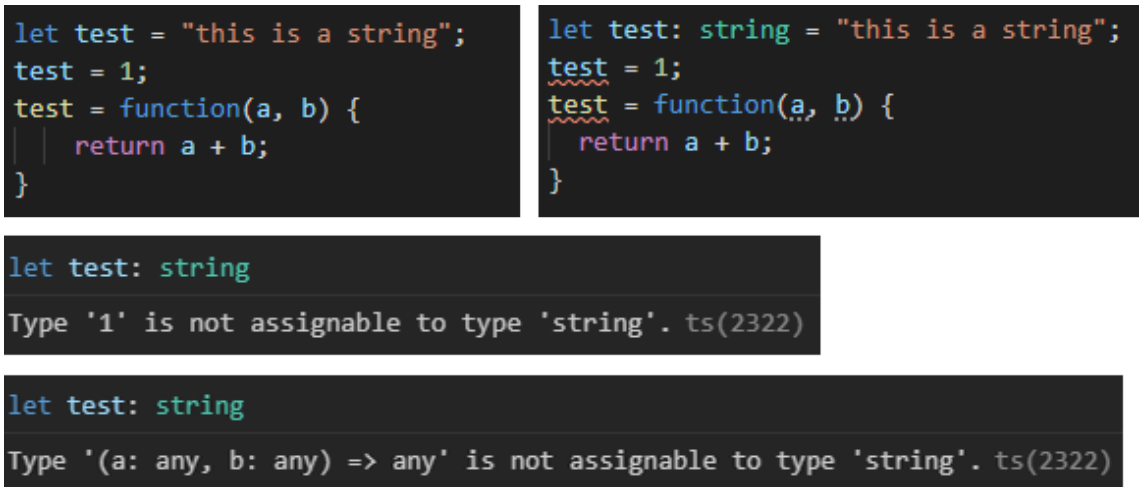
5.3.1 TypeScript

TypeScriptin syntaksi on EcmaScript 2015:a (ES6) ylijoukko ja sisältää kaikki ES6:n ominaisuudet. Se on kieli ja työkalu JavaScript-koodin luomiseen. TypeScriptin alkuperäinen tarkoitus oli helpottaa suurien JavaScript-ohjelmien kehitystä ja ylläpitoa mahdollistamalla esimerkiksi rajapintojen luomisen ohjelman komponenttien välille. [31] TypeScriptin suurimpiin hyötyihin lukeutuu muun muassa:

1. Koodin kääntäminen (engl. Compilation).
2. Staattinen ja vahva tyypitys.
3. Oliorajapintojen tyyppimäärittelyt. [32]

JavaScript-koodi on suoritettava, jotta sen sisältämät virheet saadaan selville, koska se on niin sanottu tulkattu kieli. TypeScript löytää koodista syntaksivirheet **kääntämällä koodin**, jonka avulla virheet löydetään ennen koodin suorittamista. [32]

JavaScriptin löysän tyypityksen takia muuttujien ominaisuudet ja käyttäytyminen voi muuttua dynaamisesti. TypeScriptin tarjoama **staattinen ja vahva tyypitys** eliminoi JavaScriptin dynaamisesta ja heikosta tyypityksestä johtuvan arvaamattoman käytöksen, minkä ansiosta suurenkin koodikannan ylläpito helpottuu huomattavasti. TypeScript mahdollistaa myös tyyppihuomautuksien (engl. Type annotations) lisäyksen, jonka avulla muuttujien ja olioiden tyypitykset voidaan tarkistaa jo koodin käännösvaiheessa. [32]



Kuva 5.1. *JavaScript ja TypeScript tyypitys. Muokattu lähteestä [32]*

Kuvassa 5.1 on esitelty vierekkäin sama koodinpätkä JavaScriptillä ja TypeScriptillä. JavaScript-koodissa (vasemmalla) muuttuja *test* on määritelty aluksi merkkijonoksi, jonka jälkeen sen arvoa pystytään vapaasti muuttamaan numeroksi tai funktioksi ilman virheilmoituksia. TypeScript-koodissa tämä ei ole mahdollista, sillä muuttuja *test* määritellään aluksi merkkijonoksi tyyppihuomautuksen avulla. Tämä tarkoittaa, että *test* on vahvasti tyypitetty merkkijonoksi eikä sen arvo voi olla mitään muuta tyyppiä. Koodieditori ilmoittaa tyyppivirheistä suoraan TypeScriptin koodin käännösvaiheen ansiosta kuvan 5.1 mukaisesti. [32]

```
interface User {
  id: string;
  name: string;
  title: string;
  office: string;
  username: string;
  isSupervisor: boolean;
  tasks: Task[];
  firstDay: Date;
  mySupervisor?: string;
}
```

```
export const saveUser = async (user: Api.User) => {
  const res = await axios.put(apiUrl + user.id, user);
  return res.data;
};
```

Kuva 5.2. *NewbieMaker-sovelluksen User-olion rajapinta ja funktio, joka saa kyseisen rajapinnan tyyppisen olion parametrina.*

Tyyppimäärittelyjä voi myös laajentaa kattamaan kokonaisia olioita, jonka avulla voidaan varmistaa, että esimerkiksi funktion parametri tai paluuarvo on tiettyä määriteltyä tyyppiä [31]. Kuvassa 5.2 on NewbieMaker-sovelluksen käyttäjä-oliolle määritelty rajapinta, jossa luetellaan kaikki käyttäjä-olion sisältämät ominaisuudet ja niiden tyypit sekä funktio, joka saa parametrikseen kyseisen User-rajapinnan mukaisen olion. TypeScriptillä on mahdollista määritellä rajapinnalle myös valinnaisia ominaisuuksia *?*-notaatiolla, jolloin olion ei tarvitse sisältää kyseistä ominaisuutta. Kuvassa 5.2 on määritelty ominaisuus *mySupervisor*-ominaisuus valinnaiseksi.

5.3.2 React Hooks

Sovelluksen ensimmäisen kehitysvaiheen aikana kaikki Reactilla tehtävät tilalliset komponentit täytyi toteuttaa luokkakomponentteina, koska muuta teknistä vaihtoehtoa ei ollut. Helmikuussa 2019 julkaistiin React versio 16.8 ja sen mukana **React Hooks**, joiden avulla tilanhallintaa, elinkaarimetodien tapaisia funktioita ja muita Reactin ominaisuuksia pystyi käyttämään myös funktiokomponenteissa. Hookien avulla luodut funktiokomponentit toimivat rinnakkain luokkapohjaisten komponenttien kanssa, jonka ansiosta kehittäijien ei ole tarpeellista refaktoroida vanhoja luokkakomponentteja funktiokomponenteiksi. [25]

Luokkakomponenttien kasvaessa niiden ylläpidon helppous ja ymmärrettävyys hankaloituu huomattavasti. Jos komponentti käyttää useita elinkaarimetodeja, saatetaan kahdessa metodissa tehdä toisiinsa liittyviä toimintoja, kuten datan hakua, mutta samalla vain toisessa tehdään datan hakuun liittymättömiä toimintoja. Toisiinsa liittyvä koodi päätyy näin eri metodeihin, jotka sisältävät myös toisistaan riippumattomia toimintoja, mikä voi aiheuttaa bugeja ja johtaa koodin epäjohdonmukaisuuteen. Hookit ratkaisevat tämän ongelman antamalla kehittäjälle mahdollisuuden jakaa komponentti pienempiin funktioihin. Tämä mahdollistaa toimintojen eriyttämisen niin, että jokainen funktio huolehtii vain sille määritetystä toiminnallisuudesta ilman sivuvaikutuksia.

Yleisimmät hookit ovat tilanhallintaan käytettävä *useState* ja elinkaarimetodeja korvaava *useEffect*. useState korvaa luokkakomponenttien *this.state*-syntaksin mahdollistamalla funktiokomponentin tilan asettamisen ja muokkaamisen. useState-hookia käytettäes-

```
function MontaTilaa() {
  const [numero, asetaNumero] = React.useState(1);
  const [teksti, asetaTeksti] = React.useState("tekstiä");
  const [totuusarvo, asetaTotuusarvo] = React.useState(false);
  // ...
}
```

Kuva 5.3. *Funktiokomponentti usealla useState-hookilla.*

sä määritellään ensin tila ja tilaa päivittävä funktio sekä kyseisen tilan alustava arvo. Tilan päivittyessä komponentti hahmonnetaan uudelleen aivan kuten luokkakomponenttien tapauksessa. Yhdelle funktiokomponentille voidaan määritellä useita useState-hookeja, kuten kuvassa 5.3 on tehty.

```
function Laskuri() {
  const [laskuri, asetaLaskuri] = React.useState(0);

  React.useEffect(() => {
    document.title = `Laskurin arvo on ${laskuri}`;
  }, [laskuri]);
}
```

Kuva 5.4. *Funktiokomponentti useEffect-hookilla.*

useEffect-hook yhdistää kolme luokkakomponenttien elinkaarimetodia yhteen: *componentDidMount, componentDidUpdate ja componentWillUnmount*. useEffectin määrittelemät toiminnot suoritetaan oletuksena jokaisella hahmonnuskerralla. Hookin voi kuitenkin määritellä suoritettavaksi vain silloin, kun jokin tietty arvo funktiokomponentin sisällä muuttuu, jolla mahdollistetaan tehokkaamman koodin luonti. Kuten useState-hookeja, funktiokomponentille voi määritellä useita eri useEffect-hookeja. Kuvassa 5.4 on määritelty useEffect-hook, joka suoritetaan vain silloin, kun tila "laskuri" muuttuu.

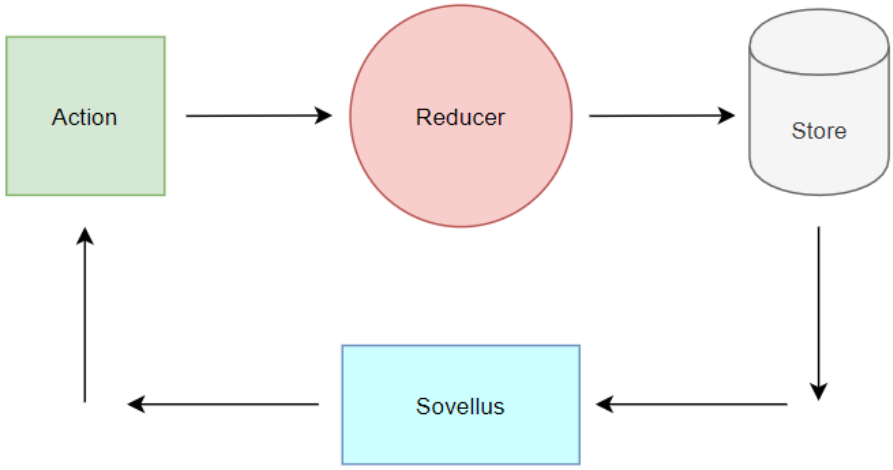
5.3.3 Redux ja Redux-Saga

Redux on skaalautuva web-sovellusten tilanhallintaan käytettävä kirjasto. Sen tarkoituksena on helpottaa niin yksinkertaisten kuin monimutkaisten sovellusten tilanhallintaa. Reduxin luojat Dan Abramov ja Andrew Clark ovat määritelleet sovelluksen tilan ylläpidon Reduxin avulla ennalta-arvattavaksi. [33] Ennalta-arvattavuudella tarkoitetaan seuraavaa:

1. Koko sovelluksen tila sijaitsee yhdessä paikassa.
2. Tila on kirjoitussuojattu ja sen muokkaaminen tapahtuu vain actionien kautta.
3. Lopullinen tila on kehittäjän itse määriteltävissä. [33]

Yksittäisen tilan päivitys on helppoa, kun koko sovelluksen tila sijaitsee yhdessä paikassa. Kehittäjän on myös varmistettava, ettei storen tilan muokkaaminen onnistu mitään

muuta kautta kuin actionien avulla. Tilaa ei myöskään koskaan muuteta tai mutatoida suoraan, vaan uusi tila luodaan joka kerta reducerien avulla. [33]



Kuva 5.5. *Reduxin toiminnan peruspalaset.*

Kuvassa 5.5 on esitelty Reduxin toimintaan liittyvät olennaisimmat kokonaisuudet. Redux on riippumaton sovelluksen toteutustavasta, joka mahdollistaa sen käytön millä tahansa web-viitekehyksellä/-kirjastolla toteutetulla sovelluksella. Ainut vaatimus sovellukselle on, että sillä on jokin tapa hallita ja säilöä sovelluksen tilaa. Koko sovelluksen tila säilötään Reduxissa sijaintiin, jota kutsutaan *Storeksi*. Tiedon lukeminen storesta on yksinkertaista, mutta monimutkaisuus ilmenee, kun uutta tilaa yritetään lisätä tai vanhaa tilaa päivittää. Redux storen tilan päivitys tapahtuu *actionien* ja *reducerien* yhteiskäytöllä. Actionin tehtävä on kuvata mitä päivitetään ja reducerin tehtävä on määrittää päivitetty tila sitä kutsuneen actionin perusteella. [33]

Reduxissa ei ole tukea asynkronisten operaatioiden suorittamiseen ilman väliohjelmistojen käyttöä [34]. NewbieMaker-sovellus hyödyntää toimintalogiikassaan asynkronisia operaatioita, joihin kuuluu muun muassa datan hakeminen sovelluksen selainpuolelle sovelluksen tietokannasta. NewbieMaker-sovelluksen käyttäjätiedot haetaan sovelluksen palvelinpuolen kautta tietokannasta ja asetetaan Redux storeen. Jotta tämä olisi Reduxia käyttämällä mahdollista, otettiin NewbieMakerissa käyttöön Redux-väliohjelmisto nimeltä Redux-Saga.

Redux-Sagan tarkoituksena on helpottaa sovelluksen sivuvaikutuksien, kuten asynkronisten operaatioiden hallintaa. Redux-Sagalla on pääsy sovelluksen Redux storeen ja voi suorittaa Redux actioneita. Koodin helppolukuisuus on yksi Redux-Sagan tavoitteista, johon se pyrkii ES6:n *generator*-funktioiden avulla, joiden avulla asynkroninenkin koodi näyttää enemmän tavalliselta synkroniselta koodilta. Generator-funktioiden tarkoituksena on toimia kuin async/await-syntaksi muutamilla lisäominaisuuksilla. Redux-Sagan avulla vältytään myös niin sanotulta takaisinkutsuhelvetiltä, joka kuuluu toisen Redux väliohjelmiston, Redux Thunkin ongelmiin. [35]

5.3.4 Material Design

NewbieMaker-sovelluksen käyttöliittymän, käytettävyyden ja käyttäjäkokemuksen suunnitteluun otettiin toisessa kehitysvaiheessa täysin uudenlainen lähestymistapa ottamalla käyttöön *Material Design* -periaatteet. Material Design on Googlen vuonna 2014 julkaisema muotokieli mobiili- ja web-sovellusten käyttöliittymien käytettävyyden parantamiseen. Material Design tarjoaa selkeän joukon suosituksia suunnittelun apuvälineeksi laajan käyttäjävuorovaikutustutkimuksen pohjalta. Näiden periaatteiden avulla pyritään välttämään huolimatonta käyttöliittymäkomponenttien suunnittelua. [36]

Material Designin perusperiaatteena on käsitellä mobiili- ja web-sovellusten käyttöliittymäkomponentteja kuten oikeita fyysisiä kappaleita. Peruselementtejä, joiden kanssa käyttäjät vuorovaikuttavat verrataan yhden pikselin paksuiseen paperiin, joihin pätee tietyt Material Designin mukaiset säännöt, jolloin elementit saavat 2D-ominaisuuksien lisäksi syvyyden. Syvyyden avulla elementit voidaan helposti erottaa yksittäisiksi olioiksi. Kun sovelluksen elementit luodaan mahdollisimman todenmukaisiksi, on käyttäjän vuorovaikutus käyttöliittymän kanssa huomattavasti intuitiivisempaa. [36]

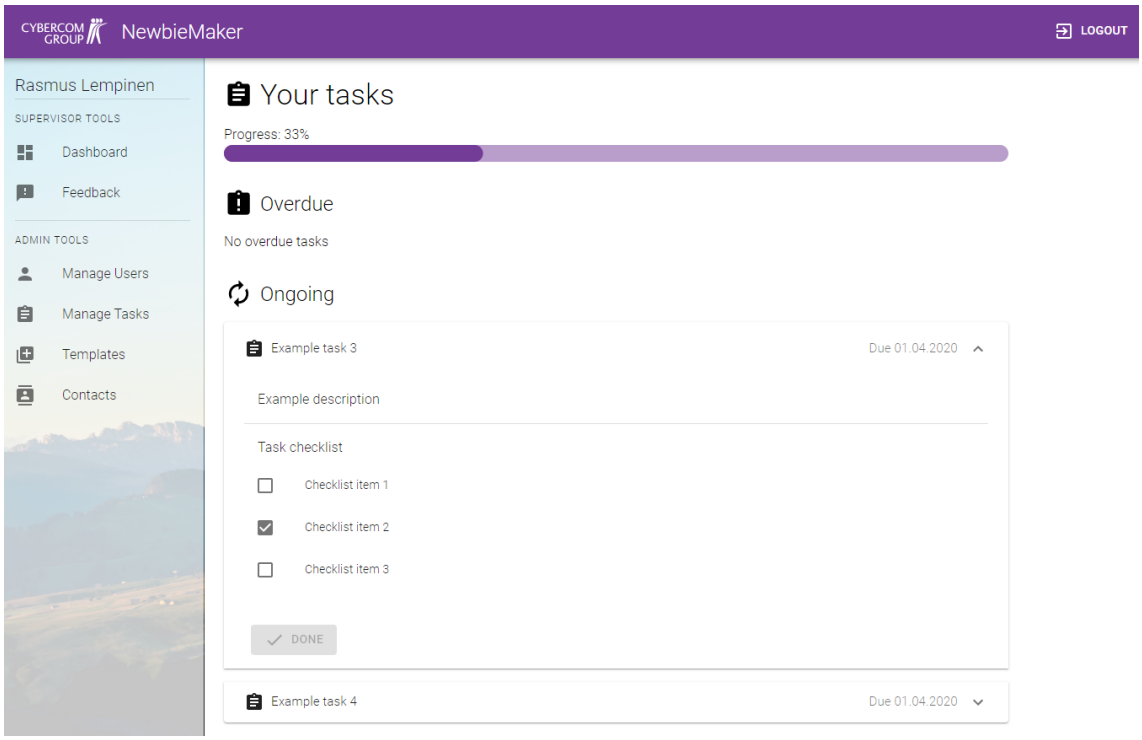
Yhtenäisen, selkeän ja intuitiivisen käyttäjäkokemuksen luominen oli yksi tavoite, johon NewbieMaker-sovelluksen toisessa kehitysvaiheessa pyrittiin. Tavoitteen saavuttamiseksi sovelluksessa aiemmin käytetty Semantic UI korvattiin Googlen Material Design -periaatteita noudattavalla käyttöliittymäviitekehyksellä, Material UI:lla. Material UI on nimenomaan Reactilla luoduille sovelluksille toteutettu viitekehys, joka tarjoaa valmiita interaktiivisia komponentteja sovelluksen käyttöön [37].

5.4 Toteutunut sovellus

Toisen kehitysvaiheen lopussa oli saavutettu tämän vaiheen alussa asetetut tavoitteet kiitettävästi. NewbieMaker-sovelluksen selainpuoli saatiin uusittua täysin uusien tekniikoiden ja käytettävyyden osalta. Lähes kaikki alkuperäiset toiminnallisuudet saatiin myös säilytettyä. Sovelluksen uusi versio todettiin tämän vaiheen lopussa kelpoiseksi pilottitestiä varten.

Kuvassa 5.6 on esitelty yksi NewbieMaker-sovelluksen toisen kehitysvaiheen jälkeisistä näkymistä. Ero ensimmäisen kehitysvaiheen jälkeiseen näkymään on todella huomattava. Toisessa kehitysvaiheessa sovelluksen kehitys tapahtui tiiviissä yhteistyössä Cybercomin UI/UX-asiantuntijan kanssa, joka suunnitteli sovelluksen jokaisen uuden näkymän ulkonäön. Sovelluksen taso parani myös käytettävyyden osalta verrattuna ensimmäiseen versioon. Suuren eron käyttöliittymän ulkonäön muutokseen oli myös CSS-viitekehyksen vaihto Material Design -pohjaiseen suunnittelumalliin.

Sovelluksen tuotantoversio julkaistiin virtuaalikoneella Cybercomin omistamassa ja ylläpitämässä Cybercom Cloud -pilvipalvelussa. Tuotantoympäristön ja sovelluksen tuotantoversion luonti sekä päivitys tapahtui vielä tässä vaiheessa täysin manuaalisesti. Päivitys tapahtui ottamalla sovelluksen selain- ja palvelinpuolen uusimmat versiot versionhallin-



Kuva 5.6. *NewbieMaker-sovelluksen toisen kehitysvaiheen jälkeinen näkymä.*

nasta kehittäjän tietokoneelle, jonka jälkeen ne vietiin manuaalisesti SSH-yhteydellä palvelimelle. Tämän jälkeen palvelinprosessit täytyi vielä käynnistää uudelleen, jotta sovelluksen uusin versio saatiin julkaistua.

6 KOLMAS KEHITYSVAIHE

Kolmas kehitysvaihe tiivistyy lähinnä tuotantoympäristön siirrosta, koodin ylläpidettävyyden ja tyylin sekä käyttäjähallinnan parantamisesta. Kohdassa 6.1 kerrotaan kolmannen kehitysvaiheen tavoitteista ja sen vaihtuvasta kehitystiimistä. Kohta 6.2 keskittyy koodin ylläpidettävyyden parantamiseksi tehtyyn Redux-refaktorointiin. Lopuksi kohdissa 6.3 ja 6.4 kerrotaan NewbieMakerin siirrosta AWS:ään ja käyttäjähallinnan uudistamisesta.

6.1 Kehitystiimi ja tavoitteet

NewbieMaker-sovelluksen kolmannen kehitysvaiheen alkaessa sovellus oli saavuttanut Cybercomilla virallisen sisäisen projektin statuksen. Virallisen projektin statuksella tarkoitetaan sovelluksen kehityksen jatkumista niin, että sen ylläpidettävyys ja käytettävyys ovat selkeitä päätavoitteita. Virallisen sisäisen projektin tarkoituksena on myös saada kehitettävä sovellus käyttöön virallisesti Cybercomin sisäisenä työntekijöiden työvälineenä.

Koska Cybercom on konsulttiyritys, menee mahdolliset asiakastyöt aina muiden projektien edelle. Tämä tarkoittaa sitä, että vaikka NewbieMaker olikin virallinen sisäinen projekti, oli sen kehitykseen hankala omistaa kehittäjiä pitkäaikaisesti. Kolmannen kehitysvaiheen aikana kehitys olikin osittain hidasta ja pätkittäistä, koska NewbieMakerin kehittäjät koostuivat lähinnä asiakastyön välissä olevista kehittäjistä. Jos kehittäjälle saatiin uutta asiakastyötä, siirtyi hän kyseiseen projektiin ja jätti NewbieMakerin kehittämisen kesken. Projekti nähtiin kuitenkin sopivana pätevyyden kehityksen alustana, jonka takia kolmannessa kehitysvaiheessa pidempään mukana olleet kehittäjät koostuivat lähinnä uusista työntekijöistä.

Tavoitteiden saavuttamiseksi kolmannessa kehitysvaiheessa tehtiin jälleen uusia teknisiä ratkaisuja. Sovelluksen käyttäjäkannan kasvaessa sen hostaaminen Cybercom Cloudissa ei ole enää järkevää. Kapasiteetin nostamisen vuoksi sovelluksen tuotantoympäristö siirrettiin AWS:n pilvipalveluun. Myöskään sovelluksen käyttäjänhallintaa ei ole virallisella sovelluksella viisasta pitää sovelluskohtaisena, joten sen siirtäminen AWS:n Amazon Cognito -palveluun oli olennaista. Itse koodin ylläpidettävyyden kannalta oli tärkeää myös muuttaa sovelluksen Redux-logiikka Redux-tyylioppaan (engl. Redux Style Guide) mukaiseksi, minkä vuoksi *slice*t otettiin käyttöön.

6.2 Redux refaktorointi

Yksi kolmannen kehitysvaiheen tavoitteista oli koodin ylläpidettävyyden parantaminen. Koska suuri osa NewbieMaker-sovelluksen lähdekoodista sisälsi Redux-logiikkaa, koettiin järkeväksi myös toteuttaa kyseinen Redux-logiikka mahdollisimman järkevällä ja ylläpidettävällä tyylillä. Redux-syntaksin parantamisen ohjekirjana käytettiin virallista Redux-tyyliopasta [38]. On monia eri tapoja ja tyylejä toteuttaa Redux-sovelluksia, joista mikään ei ole absoluuttisen oikea. Redux-tyyliopas toimii apuna kehittäjille ja sisältä **suositeltuja** lähestymistapoja Redux-sovellusten syntaksiin, rakenteeseen ja logiikkaan liittyen.

NewbieMaker-sovelluksessa noudatettiin toisessa kehitysvaiheessa jo pitkälti Redux-tyylioppaan ohjeita, mutta ylläpidettävyyttä ajatellen päätettiin joitain toteutustapoja parantaa. Redux-tyylioppaasta valittiin kaksi selkeää kehityskohtaa: *Redux Toolkit* -kirjaston käyttöönotto ja *action creator* -funktioiden rakenteen muuttaminen. Näistä Redux Toolkitin käyttöönotto on oppaan mukaan vahva suositus ja action creator -funktioiden rakennemuutos suositus.

Redux Toolkit sisältää nimensä mukaan työkaluja parhaiden käytäntöjen mukaisen Redux-logiikan luomiseen. Kyseisen kirjaston mukana tulee *createSlice*-niminen funktio, joka mahdollistaa action creator -funktioiden rakenteen muutoksen. Redux-tyylioppaan mukaan on suositeltua käyttää alkuperäisen Flux-arkkitehtuurin mukaisia action creator -funktioita sovelluksen Redux-toimintojen määrittelyyn. Action creator -funktioiden käyttö tuo selkeyttä ja johdonmukaisuutta koodiin.

Toisen kehitysvaiheen jäljiltä NewbieMaker-sovelluksessa käytettiin jo action creator -funktioita Flux-arkkitehtuurin mukaisesti. Redux-tyylioppaan suositteleman createSlice-funktion avulla päästään kuitenkin eroon action creator -funktioiden erillisestä määrittelystä. Tämä tarkoittaa, että action creator -funktiot sekä sovelluksen tilan muokkaamiseen käytetyt reducerit voidaan yhdistää automaattisesti.

	Tiedostot (kpl)	Koodi (riviä)	Kommentit (riviä)	Tyhjät (riviä)	Yhteensä (riviä)
Ennen refaktorointia	106	6813	465	1120	8398
Refaktoroinnin jälkeen	99	6661	461	1038	8160
Erotus	7	152	4	82	238

Kuva 6.1. *NewbieMaker-sovelluksen lähdekoodin määrä Redux refaktorointia ennen ja jälkeen.*

Kuvassa 6.1 esitellään NewbieMaker-sovelluksen lähdekoodirivien (engl. Source Lines of Code, SLOC) määrä ennen Redux-logiikan refaktorointia ja sen jälkeen. Huomattavaa on, että createSlice-funktion avulla päästiin eroon seitsemästä erillisestä tiedostosta, jotka sisälsivät action creator -funktioiden määrittelyjä. Refaktoroinnin tuloksena sovelluksen koodirivien määrä väheni myös huomattavasti 152 rivillä. Koodirivien vähenemisen lisäksi saatiin parannettua koodin ymmärrettävyyttä ja ylläpidettävyyttä.

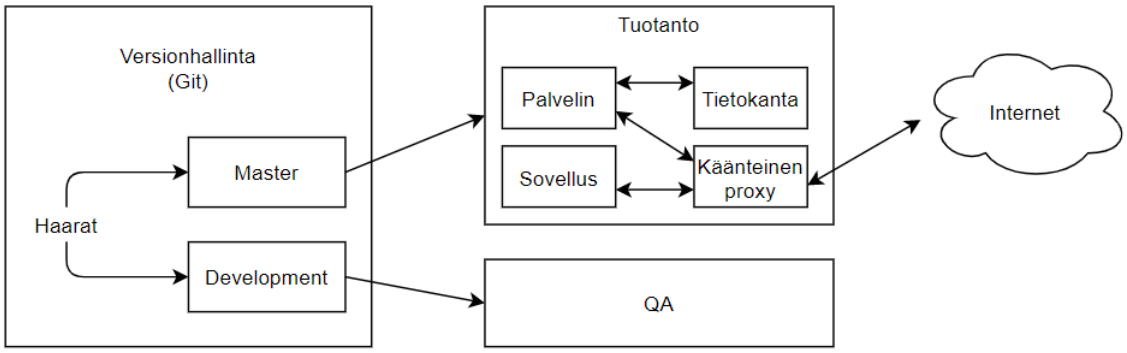
6.3 AWS

Toisessa kehitysvaiheessa NewbieMakerin tuotantoympäristö luotiin täysin manuaalisesti Cybercom Cloudiin. Tämä ei ole viisas toimintatapa silloin, kun sovelluksen kehitys on jatkuvaa ja uusia toiminnallisuuksia sekä bugikorjauksia halutaan viedä lyhyin aikavälein tuotantoon. Jotta sovelluksen tuotantoympäristön päivitys olisi mahdollisimman yksinkertaista, siirrettiin tuotantoympäristö Cybercom Cloudista AWS:ään ja tuotantoympäristön luontia varten luotiin AWS CloudFormation -malli.

AWS eli Amazon Web Services on yksi maailman suurimmista pilvipalvelualustoista. AWS:n palvelutarjontaan kuuluu yli 175 palvelua, joihin kuuluu muun muassa laskentatehon, tiedon varastointitilan, tietokantojen ja koneoppimisen palvelut. [39] AWS valittiin tuotantoympäristöksi sen suuren palvelutarjonnan sekä muutoksen tehneen kehittäjän johdosta. NewbieMaker-projektia käytettiin tässä vaiheessa myös osaamisen kehittämisen kohteena niiden konsulttien toimesta, joilla ei sillä hetkellä ollut asiakastyötä. Tällöin nähtiin, että AWS-osaamisen harjoittelu onnistuisi hyvin soveltamalla AWS:n tekniikoita NewbieMakeriin.

NewbieMaker-sovelluksessa hyödynnettiin AWS:n Amazon EC2 (engl. Amazon Elastic Compute Cloud), AWS CloudFormation ja Amazon S3 (engl. Amazon Simple Storage Service) -palveluita. **Amazon EC2**:n avulla voi hyödyntää AWS:n skaalautuvaa laskentatehoa pilvessä. Amazon EC2 mahdollistaa esimerkiksi virtuaalipalvelinten luonnin ja turvallisuus- ja verkkokonfiguraation määrittämisen. Skaalautuvuudella tarkoitetaan esimerkiksi laskentatehon muuttamista tarpeen mukaan. [40]

AWS CloudFormation -palvelun avulla on mahdollista mallintaa ja luoda käytettävät AWS-resurssit. Kehittäjä luo ensin mallin, johon kuvataan tarvittavat resurssit ja niiden ominaisuudet ja AWS CloudFormation hoitaa niiden konfiguroinnin kehittäjän puolesta. AWS CloudFormationin ansiosta kehityksessä voidaan keskittyä itse sovelluksiin, jotka pyörivät AWS:ssä resurssien käsittelyn sijaan. [41]



Kuva 6.2. *NewbieMakerin AWS-ympäristöt.*

Kolmannessa kehitysvaiheessa AWS:ään luotiin kaksi erillistä ympäristöä NewbieMaker-sovellukselle: tuotanto- ja kehitysympäristöt. Näistä kahdesta kehitysympäristön tarkoitus on toimia laadunvarmistuksen (QA, engl. Quality assurance) työkaluna. Molempia ympä-

ristöjä varten luodaan erilliset Amazon EC2 -instanssit eli virtuaalikoneet. Ympäristöt luodaan erikseen määritellyn AWS CloudFormation -mallin avulla, johon on määritelty minkä versionhallinnan haaran pohjalta ympäristö luodaan. NewbieMaker-sovelluksen versionhallinnassa on kaksi haaraa: *master* ja *development*. Master-haaraa käytetään tuotantoympäristön luontiin ja development-haaraa kehitysympäristön luontiin. Kuvassa 6.2 on havainnoitu AWS-ympäristöjen rakennetta. QA-ympäristö on rakenteeltaan samanlainen kuin tuotanto, mutta on toiston vähentämiseksi kuvattu yksinkertaistettuna.

CloudFormation -mallin avulla palvelimelle luodaan kaksi sovelluksen päivitykseen käytettävää skriptiä, jotka kehittäjä voi halutessaan suorittaa ottamalla palvelimelle SSH-yhteyden. Skriptien avulla haetaan sovelluksen uusin versio versionhallinnasta, siitä luodaan koontiversio (engl. Build), asennetaan riippuvuudet ja palvelinprosessit käynnistetään uudelleen. Tämän ansiosta sovelluksen uusin tuotantoversio on todella yksinkertaista saada tuotantoon.

6.4 Käyttäjähallinta

Alusta alkaen yksi suurimmista NewbieMaker-sovelluksen kehityskohdista on ollut käyttäjähallinnan toteutus. Ensimmäisessä kehitysvaiheessa käyttäjähallinta toteutettiin säilömällä jokainen käyttäjä sovelluksen tietokantaan mukaan lukien käyttäjän käyttäjänimi ja salasana. Salasanaa ei tallennettu selkokielisenä, vaan siitä luotiin tiiviste (engl. Hash) bcrypt-nimisen kirjaston avulla. Koska NewbieMaker on tarkoitettu yrityksen työntekijöiden sisäiseen käyttöön, ovat kaikki erilliset käyttäjätunnukset ja salasanat epäkäytännöllisiä. Oletetaan, että jokaisella yrityksen työntekijällä on käytössään henkilökohtainen tunnus, jolla on mahdollista kirjautua jokaiseen kyseisen yrityksen tarjoamiin sovelluksiin ja resursseihin. Tällöin ylimääräisten ja täysin erillisten tunnusten luominen yhtä yrityksen sovellusta varten olisi turhaa ja epäkäytännöllistä. NewbieMakeriin ei ollut toteutettu myöskään mahdollisuutta palauttaa vanhaa salasanaa, jos käyttäjä sen unohtaa. Tämän takia myös NewbieMakerin kirjautumislogiikka päätettiin eriyttää sovelluksen omasta tietokannasta täysin erilliseen palveluun.

Alun perin tarkoituksena oli siirtää käyttäjähallinta kokonaan Cybercomin käyttämään Azure AD (engl. Active Directory) -palveluun. Tälle toteutukselle ei kuitenkaan saatu konsernitason hyväksyntää. AD-integraation saatua kielteinen päätös oli käyttäjähallinnan työkalun valinta helppo, sillä jo tuotantoympäristössä hyödynnettävä AWS tarjoaa palvelua nimeltä *Amazon Cognito*. Amazon Cognito on AWS:n palvelu web- ja mobiilisovellusten käyttäjien autentikoinnin ja autorisoinnin hallintaan. Sovelluksen käyttäjät voivat käyttää Amazon Cogniton käyttäjähakemistoa eli käyttäjävarantoa (engl. User pool) kirjautumiseen. Amazon Cogniton avulla voi suoran kirjautumisen lisäksi käyttää kirjautumiseen esimerkiksi kolmannen osapuolen tunnistustietojen tarjoajaa (Google, Facebook jne.). Käyttäjävaranto tarjoaa myös "sisäänrakennetun" web-käyttöliittymän kirjautumista, uuden käyttäjän luomista ja salasanan palauttamista varten. [42]

NewbieMaker-sovelluksessa Amazon Cognito -palvelua hyödynnettiin kuvan 6.3 esittä-

Kuva 6.3. *Amazon Cogniton toimintalogiikka NewbieMaker-sovelluksessa. Muokattu lähteestä [42].*

mällä tavalla. Uuden kirjautumislogiikan voi jakaa pääosin neljään eri vaiheeseen:

1. Käyttäjän autentikointi.
2. ID Tokenin palautus sovellukselle.
3. Käyttäjätietojen haku.
4. Käyttäjätietojen palautus sovellukselle.

Sovelluksessa hyödynnettiin Amazon Cogniton tarjoamaa käyttöliittymää kirjautumiseen, jonne sovellus ohjaa käyttäjän silloin, kun aktiivista kirjautumista ei ole havaittu. Käyttäjän syöttäessä oikeat tunnukset kirjautumislomakkeeseen, palauttaa Amazon Cognito sovelluksen selainpuolelle *token*in merkkinä onnistuneesta autentikoinnista. NewbieMakerissa hyödynnettiin Amazon Cogniton ID Tokenia, joka sisältää kirjautuneen käyttäjän identifioivia tietoja käyttäjävarannosta JSON-muodossa. Tokenin sisältämän tiedon perusteella pystytään hakemaan käyttäjän sovellusspesifit tiedot NewbieMakerin tietokannasta ja palauttamaan ne sovelluksen selainpuolelle, jolloin kirjautumisprosessi saadaan päätökseen ja käyttäjä on kirjautunut sovellukseen.

7 TEHTYJEN VALINTOJEN VAIKUTUS KEHITYKSEEN

Tässä luvussa käydään läpi yksitellen jokainen NewbieMaker-sovelluksen kehitysvaihe ja niiden aikana tehtyjen teknologisten ja projektinhallinnallisten valintojen vaikutus toisiinsa. Kohta 7.1 keskittyy sovelluksen alku- ja luomisvaiheeseen, jolloin suurimmat valinnat liittyen sovelluksen kehitykseen tehtiin. Miten alun valinnat vaikuttivat alun kehitykseen ja mitä vaikutuksia niillä oli ajatellen toista kehitysvaihetta. Toisen kehitysvaiheen valinnat ja edellisen vaiheen valintojen vaikutukset käydään läpi kohdassa 7.2. Lopuksi kohta 7.3 keskittyy sovelluksen kolmannen kehitysvaiheen valintoihin ja niiden vaikutukseen ajatellen sovelluksen tulevaisuutta liittyen sen jatkokehitykseen ja ylläpitoon.

7.1 Vaihe 1

Ensimmäisen kehitysvaiheen valinnat olivat luonnollisesti ne, joilla oli sovelluksen kehitysprosessia ajatellen suurimmat vaikutukset. Ensimmäisillä valinnoilla luotiin perusta sovelluksen toteutuksen tyylille, arkkitehtuurille sekä rakenteelle.

Tämän luvun alakohdissa käydään läpi jokainen ensimmäisen kehitysvaiheen kokonaisuutena merkittävä valinta. Valinnan vaikutusta pohditaan erityisesti kehitysvaiheen aikaiseen kehitykseen ja aikaansaannosten vaikutuksen siirtymistä seuraavaan kehitysvaiheeseen. Merkittävimpiin valintoihin kuuluu muun muassa selain- ja palvelinpuolen teknologiat, käyttöliittymäsuunnittelu, tietokanta, laadunvarmistus sekä projektinhallinnan tekniikat.

7.1.1 Projektinhallinta

Scrum-viitekehyksen käytöllä ensimmäisessä kehitysvaiheessa oli useita positiivisia vaikutuksia. Koska scrum oli kaikille kehitystiimin jäsenille entuudestaan tuttu, oli sen mukaanotto projektiin helppoa ja yksinkertaista. Scrumin avulla pystyttiin määrittelemään tiimin sisällä selkeät vastuualueet, kuten tuoteomistaja, joka huolehti NewbieMakerin kehitysjonosta sekä asiakkaan puolelle kommunikoinnista. Scrumin iteraatioihin perustuva malli auttoi myös huomattavasti sovelluksen jatkuvassa parantamisessa, koska sprintin aikana saadut kehitykset käytiin asiakkaan kanssa läpi jokaisen sprintin lopussa. Tällöin tehdyistä muutoksista saatiin suoraa palautetta ja parannusehdotuksia. Tämä oli hyödyllistä erityisesti käyttöliittymäsuunnittelua ajatellen.

Negatiivisiin puoliin scrumissa kuului projektin alussa ollut vaikeus määritellä oikea määrä sprinttikohtaisia tehtäviä NewbieMakerin kehitysjonosta. Projekti ja monet valituista tekniikoista olivat kehitystiimille uusia, minkä takia sprintin aikana aikaansaatavan työmäärän arviointi oli aluksi hankalaa. Työmäärän arviointi kuitenkin luonnollisesti helpottui mitä pidemmälle projekti eteni. Myös kehitystiimin valitsema kolmen viikon sprint-pituus todettiin liian pitkäksi, sillä ensimmäisen viikon aikana turvauduttiin ajatukseen, että tämän jälkeen on vielä kaksi viikkoa aikaa kehittää. Tämän takia työmäärä kasaantui aina sprintin loppupäähän aiheuttaen turhaa kiirettä.

Scrumin käyttö jatkui myös toiseen kehitysvaiheeseen, joskin hieman muunneltuna versiona, sillä kehitystiimiin kuului vain kaksi henkilöä. Näistä henkilöistä toinen oli samaan aikaan sovelluksen käyttöliittymäsuunnittelija, tuoteomistaja sekä scrummaster. Pienestä kehitystiimistä huolimatta scrumin toimintatapojen noudattaminen sujui toisessa kehitysvaiheessa erinomaisesti. Sprintin pituudeksi valittiin kaksi viikkoa, joka oli huomattavasti tehokkaampi valinta kolmen viikon sijaan. Kahden viikon sprinttien ansiosta myös palautetta saatiin tiheämpään tahtiin.

Voidaan todeta, että scrum-viitekehys oli valintana erittäin onnistunut projektinhallinnan työkalu, sillä sen käyttö jatkui onnistuneesti NewbieMakerin kahden eri kehitysvaiheen läpi. Kolmannessa kehitysvaiheessa scrumia ei enää hyödynnetty tiheään vaihtuvien kehittäjien/tiimien takia. Scrumin noudattamisen aikana erityisesti iteratiivisuuden mahdollistama jatkuva parantaminen todettiin hyödylliseksi kehitettävän tuotteen laadun kannalta.

7.1.2 JavaScript ja React

JavaScriptin valinta NewbieMakerin selainpuolen toteutukseen vaikutti kokonaisvaltaisimmin sovelluksen kehitysprosessiin. Se auttoi Node.js:n valinnassa palvelinpuolen toteutukseen sekä vaikutti sovelluksessa käytettävien viitekehysten ja kirjastojen valintaan.

JavaScriptin maailmanlaajuinen yleisyys helpotti varsinkin ensimmäisen kehitysvaiheen kehitystä, sillä jokaisella kehitystiimin jäsenellä oli siitä entuudestaan kokemusta. Suuren suosionsa takia JavaScriptin dokumentaatio on todella kattava sekä internet on pullollaan vastauksia kehityksen aikana esiintyviin kysymyksiin ja ongelmiin. Myös jatkokehitystä ajatellen voidaan olettaa, että sovellukselle on helppo löytää kehittäjiä, joilla on kokemusta JavaScriptistä verrattuna harvinaisempaan web-ohjelmointikieleen.

Koska selainpuolen kehityskieleksi valittiin JavaScript, antoi se mahdollisuuden hyödyntää mitä tahansa lukuisista JavaScript-viitekehyksistä ja -kirjastoista, joihin kuuluu muun muassa ensimmäisessä kehitysvaiheessa selainpuolen kehityksen avuksi käyttöönotettu React-kirjasto. Suurimmalla osaa kehitystiimiä ei ollut lainkaan kokemusta Reactin käytöstä ennaltaan, mutta sen kasvava suosio ja osaajien tarve työmarkkinoilla houkutteli kehittäjiä. Tämän takia ensimmäisen kehitysvaiheen alusta kului suuri osa aikaa Reactin toiminnan ja periaatteiden opetteluun. Kokemattomuus Reactin käytöstä aiheutti myös huonojen ratkaisujen tekemistä kehityksen aikana liittyen tilanhallintaan ja komponent-

tien toimintaan. Huonojen React-käytäntöjen seurauksena ensimmäisen kehitysvaiheen aikana jouduttiin käyttämään yksi kokonainen sprintti pelkkään refaktorointiin.

Vaikka Reactin käytössä oli projektin alussa hankaluuksia, osoittautui se kehitysprosessin aikana todella hyödylliseksi työkaluksi NewbieMakerin selainpuolen kehityksessä. Kehityksen aloittaminen ja sovelluksen luominen oli todella yksinkertaista *Create React App*in (CRA) ansiosta. CRA on virallisesti tuettu tapa luoda uusia yhden sivun React-sovelluksia ilman konfigurointia. CRA luo sovellukselle automaattisesti perustarpeisiin sopivan koontiversion kehitykseen ja tuotantoon sekä valmiin kansiorakenteen, minkä ansiosta kehittäjän ei itse tarvitse huolehtia sovelluksen konfiguroinnista ja kehityksen voi aloittaa heti. [43]

Loppujen lopuksi NewbieMakerin selainpuolen luominen JavaScriptin ja Reactin avulla oli todella hyvä valinta. Koska React perustuu komponentteihin, oli NewbieMakerin rakenteen jakaminen selkeisiin osiin yksinkertaista. React toimi myös erinomaisesti toisessa kehitysvaiheessa käyttöön otetun Redux-kirjaston kanssa, jonka avulla suurienkin React-komponenttien tilanhallinta saatiin yksinkertaistettua. Suuren käyttäjä- ja kehittäjäkannan ansiosta myös kehittämisen aikana vastaan tulleisiin ongelmiin oli helppo saada vastauksia. Jatkuvasti kehittyvän luonteensa takia Reactiin ilmestyi NewbieMakerin kehitysprosessin aikana useita uusia ominaisuuksia, joiden avulla koodimäärää pystyttiin vähentämään toiminnallisuuden pysyessä samana.

7.1.3 Sovelluksen tilanhallinta

Sovelluksen koon vähättely ja sen perusteella tehty valinta jättää tilanhallinta pelkästään Reactin omien komponenttien sisälle oli yksi ensimmäisen kehitysvaiheen seurauksiltaan suurimmista päätöksistä. Ensimmäisen toteutusvaiheen jäljiltä sovelluksen koodi sisälsi komponentteja, joilla oli todella paisuneita tilamäärittelyjä. Tällaisten komponenttien ylläpito ja laajentaminen on todella hankalaa pelkän Reactin avulla ilman tilanhallintaan tarkoitettua kirjastoa.

```
<EditUser
  templates={this.state.templates}
  updateUsers={this.updateUsers}
  key="newUser"
  allTasks={this.state.tasks}
  allTemplates={this.state.templates}
  allUsers={this.state.users}
  user={emptyUser}
  ref={ref => (this.addNewUser = ref)}
/>
```

```
class EditUser extends Component {
  constructor(props) {
    super(props);

    this.state = {
      userId: null,
      tasks: [],
      mySupervisor: null,
      nearbyColleagues: [],
      name: '',
      username: '',
      password: '',
      firstDay: '',
      isSupervisor: false,
      taskOptions: [],
      newUser: true,
      collapsed: true,
      expandState: 'editUser editItem editItem-collapsed'
    };
```

Kuva 7.1. *Yksi NewbieMaker-sovelluksen paisuneista komponenteista.*

Esimerkki komponentin paisuneesta tilamäärittelystä näkyy kuvassa 7.1. Komponentin jättimäistä tilaa hallittiin kokonaisuudessaan kyseisen komponentin sisällä ja sille annettiin kutsuessa lisäksi kasa propseja kyseisen komponentin renderöivältä komponentilta. Näin suuren tilan hallitseminen komponentin itsensä sisällä ei ole järkevää koodin ymmärtämistä tai jatkokehitystä ajatellen. Myös se, että komponentille täytyy erikseen välittää useita eri propseja, jotka riippuvat kyseistä komponenttia kutsuvan komponentin sisäisestä tilasta, tekee komponentin käytöstä kömpelöä.

7.1.4 Käyttöliittymä

NewbieMaker-sovellusta kehitettiin pitkään toiminnallisuus ensin -periaatteella. Tämä tarkoitti sitä, että sovelluksen käyttöliittymä oli pitkään todella karun näköinen, koska se oli toteutettu niin sanotusti "puhtaalla" HTML:llä ja CSS:llä. Tarkoituksena oli toteuttaa sovelluksen toiminnallinen puoli ensimmäisenä minimalistisella käyttöliittymäkehityksellä. Tätä periaatetta noudatettiin kuitenkin projektissa toisiksi viimeiseen sprinttiin saakka, jonka takia käyttöliittymän kunnolliselle toteutukselle jäi rajallinen määrä aikaa.

Semantic UI Reactin käyttöönotolla NewbieMakerin CSS-viitekehykseksi oli todella suuri vaikutus sovelluksen käyttöliittymäkehitykseen. Semantic UI Reactin valmiiksi tyyliteltyjen HTML/CSS-komponenttien avulla sovelluksesta sai kehityksen kolmen viimeisen viikon aikana edes jollain tapaa ammattimaisen näköisen. Kuten aiemmin kuvassa 4.1 esiteltiin, on tyylikkäiden komponenttien luonti Semantic UI Reactin avulla todella yksinkertaista. CSS-viitekehyksen käytöstä hyödyttiin myös NewbieMakerin responsiivisuutta ajatellen, sillä valmiit käyttöliittymäkomponentit skaalautuivat muuttuvilla näyttöko'oilla automaattisesti. Tämän ansiosta kehittäjien ei tarvinnut käyttää liikaa ylimääräistä aikaa sovelluksen ulkonäön kehittämiseen pienemmille näytöille, kuten älypuhelimille.

Koska CSS-viitekehyksen käyttöönotto jäi viimeiseen sprinttiin, jäi sen syvempi opettelu ja dokumentaatioon tutustuminen todella vähälle. Tämän takia NewbieMakerissa ei hyödynnetty esimerkiksi Semantic UI:n tarjoamaa teeman asettelua, mikä avulla muun muassa sovelluksen väri- ja fonttimaailman hallintaa olisi voitu helpottaa ja yhtenäistää. Ensimmäisen kehitysvaiheen lopussa sovelluksen lähdekoodiin jäi yli 700 riviä pitkä CSS-tiedosto, jonka pituudesta olisi voinut karsia huomattavasti tutustumalla huolellisemmin Semantic UI Reactin käyttöön ja toimintaan.

7.1.5 Palvelinpuoli ja tuotantoympäristö

NewbieMaker-sovelluksen palvelinpuolen teknologiat säilyivät kaikkien kehitysvaiheiden läpi muuttumattomina. Sovelluksen palvelinpuoli käyttää nykyiselläänkin Node.js:ää ja Expressiä palvelinlogiikan ohjelmointiin sekä tietokantanaan MongoDB:tä. Vaikka palvelinpuolen teknologioita ei ole kehitysprosessin aikana muutettu, on palvelinpuolen koodin logiikan muokkaus ollut kehityksen aikana tarpeellista. Palvelinpuolen muokkausta helpotti ohjelmointikielen yhtenäisyys selainpuolen kanssa, minkä ansiosta esimerkiksi

selainpuolen-kehittäjän oli helpompi ymmärtää palvelinpuolen toimintalogiikkaa ja tehdä siihen tarvittaessa muutoksia.

NewbieMakerin kehityksen edetessä muutokset tietokannan dokumenttien skeemoihin olivat myös välttämättömiä. MongoDB:n valinta sovelluksen tietokannaksi todistautuikin tämän ansiosta erinomaiseksi valinnaksi sen joustavuuden ansiosta, minkä avulla aiemmin määriteltyjen dokumenttien datamallien muuttaminen helpottui. Suuri positiivinen vaikutus oli myös MongoDB:n käyttämällä kyselykielellä, jonka todettiin olevan paljon yksinkertaisempi ja ymmärrettävämpi kuin esimerkiksi SQL. Helposti ymmärrettävän kyselykielen ansiosta tietokannan kehitys ja muokkaaminen nopeutui huomattavasti.

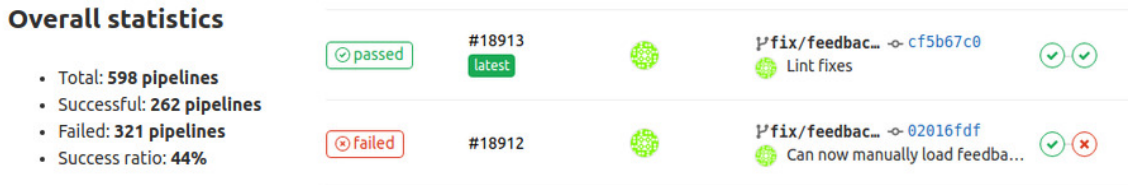
Azuren virtuaalikoneen käyttö tuotantoympäristönä ei missään vaiheessa ollut tarkoitettu lopulliseksi valinnaksi. NewbieMaker-sovelluksen ensimmäisen kehitysvaiheen lopullinen versio vietiin Azuren virtuaalikoneelle vain demotarkoituksessa ja tuotantoympäristön muutos tulevassa kehitysvaiheessa oli tässä vaiheessa tiedossa.

Kokonaisuutena ensimmäisessä kehitysvaiheessa tehdyt palvelinpuolen teknologiavalinnat voidaan todeta onnistuneiksi. Valinnoilla oli huomattavasti suuremmat positiiviset kuin negatiiviset vaikutukset eivätkä ne aiheuttaneet tarvetta muutoksille seuraavissa kehitysvaiheissa.

7.1.6 Laadunvarmistus

Ensimmäisen kehitysvaiheen lopussa NewbieMakerin kehitystiimi totesi olleensa projektissa eniten tyytyväinen käytettyyn jatkuvan integraation ja kehityksen putkeen (CI/CD-putki). CI/CD-putki oli mukana projektin kehityksen tukena lähes alusta saakka, minkä ansiosta siitä hyödyttiin ensimmäisen kehitysvaiheen aikana huomattavasti.

CI/CD-putken avulla varmistettiin, että versionhallinan master-haaraan päätyi aina toimivaa versio sovelluksesta. Koska sovellukseen lisättiin uusia toiminnallisuuksia melko tiehään tahtiin, oli käyttöliittymätestaus tärkeässä roolissa, koska sillä varmistettiin ettei uudet toiminnallisuudet riko vanhoja toiminnallisuuksia. CI/CD-putken avulla käyttöliittymätestaus pystyttiin suorittamaan automaattisesti, minkä ansiosta kehittäjän ei tarvinnut itse testata jokaista toimintoa manuaalisesti.



Kuva 7.2. *NewbieMaker-projektin ensimmäisen vaiheen jatkuvan integraatioputken dataa.*

Kuvassa 7.2 on NewbieMakerin ensimmäisen kehitysvaiheen CI/CD-putken dataa. Kuvasta nähdään, että CI/CD-putki oli todella kovassa käytössä, sillä ensimmäisen kehitysvaiheen aikana ajettiin yhteensä 598 testiputkea. Datasta nähdään myös, että testien

onnistumisprosentti oli 44%, johon on kaksi selkeää syytä: linttaus ja käyttöliittymämuutokset. Kehitystiimillä oli käytössään yhteiset ESLint-säännöt, joiden avulla varmistettiin koodityylin yhtenäisyys. Kehittäjän täytyi kuitenkin ajaa linttauskorjaukset manuaalisesti ennen muutosten puskemista versionhallintaan. Tämä tarkoitti sitä, että linttaukset unohdettiin suorittaa ennen, minkä seurauksena testiputki epäonnistui. Tässä tapauksessa testiputken epäonnistuminen ei siis johtunut varsinaisesti toimimattomasta koodista, vaan sen tyylistä. Linttausvirheet olisi voitu välttää esimerkiksi käyttämällä useimpiin tekstieditoreihin saatavilla olevaa Prettier-lisäosaa, jonka avulla linttaukset voidaan suorittaa esimerkiksi samalla, kun tallentaa muutokset muokattuun tiedostoon.

Toinen syy alhaiseen testien onnistumisprosenttiin oli käyttöliittymämuutokset. Käyttöliittymä muuttui todella aktiivisesti, kun uusia toiminnallisuuksia luotiin, koska ensimmäisen kehitysvaiheen aikana ei ollut selkeää kuvaa toteutettavasta käyttöliittymästä. Tämä tarkoitti, että toteutetut käyttöliittymätestit eivät pysyneet mukana käyttöliittymäkehityksessä ja olivat usein jäljessä verrattuna sen hetkiseen kehitykseen. Tällöin, jos kehittäjä puski uudet käyttöliittymämuutokset versionhallintaan ennen päivitettyjä käyttöliittymätestejä, seurasi siitä todennäköisesti epäonnistunut testiputki.

7.2 Vaihe 2

Toinen kehitysvaihe koostui lähinnä koko NewbieMaker-sovelluksen selainpuolen refaktoroinnista. Löysän tyypityksen omaava virheherkkä JavaScript vaihdettiin vahvan tyypityksen TypeScriptiin. Komponenttien tilanhallintaa helpotettiin siirtämällä koko sovelluksen tila yhteen Redux storeen. Tuotantoympäristö vaihtui Azuren virtuaalikoneesta Cybercomin sisäiseen Cybercom Cloud -pilvipalveluun. Sovelluksen käyttöliittymä suunniteltiin kokonaan uusiksi Material Design -periaatteiden pohjalta, joka tarkoitti myös CSS-viitekehyksen vaihtoa Semantic UI:sta Material UI:hin. Tämän seurauksena sovelluksen ulkonäkö muuttui täysin alkuperäiseen verrattuna, mutta toimintalogiikka pysyi suunnilleen samana.

Yksi suurista ei-teknologisista toisen kehitysvaiheen muutoksista oli kehitystiimin muuttuminen kokonaisesta kuuden hengen tiimistä yhteen kehittäjään ja yhteen UI/UX-suunnittelijaan. Tästä seurasi melko hidas kehitystahti, jolla ei loppupeleissä ollut vaikutusta projektin aikatauluun. Kehitystiimin pienentyminen tarkoitti myös aiemmin olleiden kehittäjäroolien, kuten testaajien poisjäämistä.

7.2.1 TypeScript

Selainpuolen kehityskielen vaihto JavaScriptistä TypeScriptiin toi mukanaan todella suuria hyötyjä, mutta myös hidasteita. TypeScriptin syntaksin opetteluun kului toisen vaiheen kehittäjältä jonkin verran aikaa, mikä hidasti NewbieMakerin kehitystä osittain. TypeScriptin mahdollistaman staattisen tyypityksen takia NewbieMakerin lähdekoodirivien määrä kasvoi runsaasti muun muassa tyyppirajapintojen määrittelyn seurauksena. Staattisen

tyypityksen takia myös joidenkin funktioiden toteutus ei onnistunut enää samalla tapaa kuin JavaScriptillä, minkä takia funktioiden logiikkaa jouduttiin ajattelemaan uudestaan. Tällä oli tietysti positiivinen vaikutus koodin laatuun nähden ja sovelluksen tulevaisuutta ajatellen, mutta se aiheutti lisätöitä toisen kehitysvaiheen aikana.

Jos JavaScript-sovelluksessa joudutaan muuttamaan esimerkiksi jonkin funktion toimintaa, ei ole aina selkeää mitä parametrejä funktioille annetaan. Jos esimerkiksi funktio saa parametrinaan olion, miten kyseisen olion kentät ja kenttien tyypit määritellään? TypeScriptin ansiosta funktioiden muuttaminen yksinkertaistui, sillä erikseen määriteltyjen tyyppien avulla nähtiin suoraan missä muodossa kyseisen funktion saamat parametrit täytyy olla. Jos funktiolle yritettiin antaa väärän tyyppistä parametria, huomautti koodieditori siitä heti TypeScriptin koodin kääntämisvaiheen ansiosta.

TypeScriptin käyttöönotto helpotti myös siirtymistä toisesta kehitysvaiheesta kolmanteen. NewbieMakerin uusien kehittäjien oli helpompi ymmärtää sovelluksen toimintaa. Staattisen tyypityksen avulla kehittäjien on myös helpompi ymmärtää koodia ja koodimuutosten vaikutusta sen toimintaan.

7.2.2 Redux-kirjasto tilanhallintaan

Yksi toisen kehitysvaiheen merkittävimmistä uusista teknologiavalinnoista oli Redux-kirjaston käyttöönotto sovelluksen tilanhallintaan. Reduxin avulla koko sovelluksen tila sijaitsi yhdessä paikassa, joka mahdollisti komponenttien yksinkertaisemman rakenteen ja käytön. Datan kuljettaminen korkean tason komponenteista alemman tason komponentteihin helpottui huomattavasti, kun dataa ei tarvinnut enää liikuttaa komponenttien välillä propsien kautta.

```
class Feedback extends Component {
  constructor(props) {
    super(props);
    this.state = {
      feedback: this.props.feedback,
      feedbackType: this.props.feedbackType,
      buttonStyle: this.props.buttonStyle,
      text: this.props.text,
      showModal: false,
    };
  }
```

```
const FeedbackDialog: React.FC<ConnectedProps> = props => {
  const { open, toggleDialog } = props;
  const activeUser = useSelector<State.Root, Api.User>(
    state => state.main.activeUser!
  );
```

Kuva 7.3. *Komponentin tilamäärittely ilman Reduxia ja sen kanssa.*

Kuvassa 7.3 on sama *Feedback*-komponentti määriteltynä vasemmalla ilman Reduxia ja oikealla Reduxin kanssa. Kuvasta huomataan selkeästi, miten komponentin määrittely yksinkertaistui Reduxin ansiosta. Alkuperäiselle komponentille välitettiin esimerkiksi kaikki palautetta antavaan käyttäjään liittyvät tiedot propsien välityksellä feedback-propsin kautta. Reduxia käyttävä komponentti voi hakea kirjautuneen käyttäjän tiedot suoraan Redux-storesta kutsumalla *useSelector*-funktiota.

NewbieMaker-sovellus kuitenkin sisälsi myös asynkronista logiikkaa, jonka hallitseminen Redux-kirjastolla itsessään ei ole mahdollista. Asynkronisten toimintojen hallintaan pro-

jektiin otettiin Reduxin rinnalle Redux-Saga-väliohjelmisto. Toisessa kehitysvaiheessa olleelle NewbieMakerin kehittäjälle niin Redux kuin Redux-Saga olivat tuntemattomia kirjastoja. Tämän takia NewbieMakerin kehitys hidastui huomattavasti toisen kehitysvaiheen alussa, koska kehittäjän täytyi opetella kahden täysin uuden kirjaston toiminta, joiden rooli sovelluksen toiminnassa oli merkittävä.

7.2.3 Selainpuolen syntaksin refaktorointi

Toisessa kehitysvaiheessa tehdyn selainpuolen syntaksin refaktoroinnin tavoitteena oli siistiä ja selkeyttää sen hetkistä React-komponenttikoodin syntaksia. Ensimmäisen kehitysvaiheen jäljiltä NewbieMakerin lähdekoodiin oli jäänyt syntaksiltaan paisunutta ja huonokäytäntöistä koodia. Muun muassa komponentit, joilla ei ollut käytössä elinkaarimetodeja tai sisäistä tilaa, oli toteutettu luokkakomponentteina. Tällaisten komponenttien määrittelyt saatiin helposti muutettua funktiokomponenteiksi, mikä vähensi huomattavasti turhan koodin määrää ja selkeytti komponenttien toimintaa.

Suurin syntaksin refaktorointi kuitenkin tapahtui, kun projektissa otettiin React Hookit käyttöön. Sen sijaan, että vain tilattomat ja elinkaarimetodeja hyödyntämättömät komponentit olisi määritelty funktiokomponentteina, saatiin React Hookien avulla jokainen sovelluksen komponenteista määriteltyä funktiokomponenttina. Tällä saavutettiin huomattava ylimääräisten koodirivien vähentyminen ja komponenttien rakenteen parempi jäsentely esimerkiksi elinkaarimetodien osalta käyttämällä useEffect-hookia. Komponenttien sisäinen tilamäärittely pystyttiin hoitamaan selkeästi useState-hookin avulla, jolla vältettiin muun muassa ruman *constructor*-metodin käyttö komponentin määrittelyssä.

React Hookien avulla päästiin käsiksi myös NewbieMakerissä hyödynnettävien apukirjastojen hookeihin. Muun muassa käytetyn React Redux -kirjaston *useSelector*-hookin avulla päästiin komponenteissa käsiksi sovelluksen Redux storeen yksinkertaisen syntaksin avulla. useSelector-hookin ansiosta vältyttiin myös korkeamman tason komponenttien (engl. Higher-Order Components, HOC) käytöltä, jotka kietovat alkuperäisen komponentin niin sanottuun säiliökomponenttiin (engl. Container component), mikä luo sovelluksen DOMiin ylimääräisen elementin [44].

7.2.4 Material Design ja käyttäjäkokemussuunnittelu

NewbieMaker-sovelluksen ulkonäköön liittyvistä muutoksista suurin vaikutus oli ehdottomasti käyttöliittymä- ja käyttäjäkokemusammattilaisen (UI/UX) mukaan ottamisella kehitysprosessiin. Ammattilaisen mielipiteiden perusteella sovelluksen käyttöliittymä uusittiin täysin Material Design -periaatteiden pohjalta. Kehitys muuttui samalla yksinkertaisemmaksi, sillä kehittäjä sai kuvat uusista näkymistä suoraan käyttöliittymäsuunnittelijalta, jolloin hänen ei tarvinnut itse käyttää aikaansa käyttöliittymien suunnitteluun. Tekemällä selkeä jako sovelluksen kehittäjän ja käyttöliittymäsuunnittelijan vastuiden välille, saavutettiin samalla käytettävyydeltään parempilaatuinen tuote.

Toisen kehitysvaiheen lopussa NewbieMaker laitettiin pilottitestiin uusilla työntekijöillä. Työntekijät käyttivät sovellusta noin kuukauden ajan aloituksestaan Cybercomilla. Kuukauden jälkeen toisessa kehitysvaiheessa mukana ollut UI/UX-ammattilainen suoritti kyselyn, joka suunnattiin pilottitestissä mukana olleille uusille työntekijöille sekä heidän esimiehelleen. Kyselyn tavoitteena oli selvittää sovelluksen sen hetkisen tilan käyttäjäkokemusta.

Oliko NewbieMaker miellyttävä käyttää?	Mitä mieltä olet NewbieMakerin visuaalisesta ilmeestä?	Tiesitkö aina mitä olit tekemässä NewbieMakeria käytettäessä?	Koitko tehtävien seuraamisen helpoksi?	Koitko palautteen antamisen helpoksi?
Hirveän selkeä käyttää.	Visuaalisesti miellyttävä.	Kyllä suurin piirtein, ei ollut epäselvää.	Tehdyt ja tekemättömät tehtävät olivat selkeästi esillä.	Kyllä, oli intuitiivista.
Hyvin suunniteltu.	Hyvä ja selkeä.	Ei tarvinnut asioita etsiä. Vasemmalta löytyi kaikki.	Oli aika helppo seurata.	Simppeli ja selkeä.
Ei ollut ongelmia.	Toimii tuollaisenaan.	En aina.		"Onko siellä sellainenkin nappula?"

Kuva 7.4. *NewbieMakerin pilottitestin jälkeisen kyselyn kysymyksiä ja vastauksia.*

Kuvaan 7.4 on valittu suoritetusta kyselystä viisi olennaista kysymystä muutama niihin annettu vastaus liittyen NewbieMakerin ulkonäköön ja käytettävyyteen. Vastauksista huomataan, että käyttäjät ovat kuvanneet sovelluksen käyttöliittymää muun muassa hyvin suunnitelluksi, selkeäksi ja toimivaksi. Käytettävyyden osalta sovellus tuntuu olleen myös selkeäkäyttöinen, intuitiivinen ja toimiva. Joitain korjauskohtia kuitenkin vielä on, sillä kaikki käyttäjät eivät olleet täysin tyytyväisiä.

7.2.5 Laadunvarmistuksen puuttuminen

Yksittäinen isoin muutos toisen vaiheen kehitysprosessiin oli ensimmäisessä kehitysvaiheessa olleen automaatiotestauksen puuttuminen, jonka myötä sovelluksen laadunvarmistus heikkeni. Testauksen pois jättäminen oli tietoinen valinta toisen kehitysvaiheen alkaessa, sillä aika eikä resurssit antaneet periksi testauksen mukaan ottoon. Aiemmin jokaisen master-haaraan liitettävän muutoksen tuli ensin läpäistä GitLabin automaattitestit, jonka jälkeen muutokset piti hyväksyttää kehitystiimin koodikatselmoinnilla. Vasta tämän prosessin jälkeen muutokset voitiin liittää sovelluksen master-haaraan.

Toisessa kehitysvaiheessa NewbieMaker-projekti oli vielä Innovation Zone projektistatuksen alaisuudessa eli se toimi niin sanottuna harjoitteluprojektina. Tämän takia Cybercom ei pystynyt varsinaisesti omistamaan projektille muita työntekijöitä harjoittelijan lisäksi esim. testauksen avuksi. Uudet koodimuutokset kävivät edelleen koodikatselmoinnin läpi ennen niiden yhdistämistä master-haaraan, mutta automaattitestausta (käyttöliittymä- ja yksikkötestaus) ei hyödynnetty.

Kehitysprosessissa luotettiin siihen, että kehittäjä testaa itse manuaalisesti toteuttamansa toiminnallisuudet. Manuaalitestaus toimi toisessa kehitysvaiheessa yllättävän hyvin, koska sovelluksella oli vain yksi varsinainen kehittäjä. Tämän takia kaikki toiminnallisuusmuutokset olivat saman henkilön tekemiä, jolloin kehittäjä oli aina tietoinen sovelluksen

kehitystilasta eikä versionhallintaan päässyt täten ilmestymään esimerkiksi toisen kehittäjän koodia, joka voisi olla ristiriidassa jonkin nykyisen toiminnallisuuden kanssa. Kattavan automaattitestauksen puuttumisen takia sovellukseen jäi kuitenkin muutamia bugeja, jotka ilmenivät vasta sovelluksen mennessä pilottitestiin toisen kehitysvaiheen lopussa. On hankala arvioida, oliko ensimmäisen kehitysvaiheen (CI/CD-putki käytössä) jälkeisessä NewbieMaker-toteutuksessa vähemmän bugeja kuin toisen kehitysvaiheen (ei CI/CD-putkea) jälkeisessä versiossa, koska ensimmäistä versiota ei koskaan pilottitestattu.

7.3 Vaihe 3

Kolmannen kehitysvaiheen muutokset olivat selkeästi jo pienempiä verrattuna aiempiin kehitysvaiheisiin. Tässä vaiheessa sovelluksen kokonaisvaltainen laatu oli jo hyvä, jolloin tarvetta suurelle määrälle muutoksia ei ollut.

Koska kolmas kehitysvaihe oli vielä tämän työn kirjoitushetkellä käynnissä, ei kolmannessa kehitysvaiheessa tehtyjen valintojen vaikutusta seuraavaan kehitysvaiheeseen voi vielä tietää varmasti. Tämän kohdan alakohdissa keskitytäänkin sovelluksen nykyiseen tilaan, sen ylläpidettävyyteen ja toisen kehitysvaiheen jälkeen tehdyn pilottitestin palautteiden pohjalta tehtäviin jatkokehitysmahdollisuuksiin.

7.3.1 Nykyisen sovelluksen ylläpidettävyys

Nykyisellään NewbieMaker-sovelluksen ylläpidettävyys on hyvällä tasolla. MongoDB:n käyttö sovelluksen tietokantana takaa sen skaalautuvuuden tilanteessa, jossa käyttäjäkanta kasvaa huomattavasti. Myös kolmannessa kehitysvaiheessa tehty sovelluksen siirto AWS:ään takaa laskentatehon skaalautuvuuden, jos sovelluksen käyttäjäkantaa kasvaa. Myös sovelluksen tuotantoversion päivittäminen ja mahdollisten vikojen korjaaminen on helpompaa AWS CloudFormation skriptin sekä palvelimella olevien päivitysskriptien ansiosta.

Myös kolmannessa kehitysvaiheessa tehty Redux-logiikan refaktorointi selkeytti koodia samalla parantaen sen ylläpidettävyyttä ja vähentämällä turhaa koodia. NewbieMaker-sovelluksen kehitykselle oli tyypillistä kehittäjien ja kehitystiimin vaihtuminen vähintään kehitysvaiheiden välillä. Tällaiselle projektille on elintärkeää, että sovelluksen koodi on ymmärrettävää ja rakenteeltaan järkevä, johon Redux refaktoroinnilla pyrittiin. Selkeällä koodilla varmistetaan, että tulevaisuudessa uusien kehittäjien mahdollisimman helppo jatkaa siitä mihin edelliset kehittäjät ovat jääneet.

7.3.2 Jatkokehitysmahdollisuudet ja ongelmat

NewbieMaker-sovellus kehittyy jatkuvasti ja vanhojen ongelmien korjaaminen sekä uusien ominaisuuksien toteutus on tulevaisuudessa olennaista. Onkin hyvä miettiä sovelluksen mahdollisia jatkokehityskohteita ja mahdollisia puutteita nykyisessä toteutuk-

sessa jo etukäteen. TypeScriptin käyttöönotto todettiin hyväksi ratkaisuksi selainpuolella. Koska sovelluksen palvelinpuoli on toteutettu JavaScriptillä, on myös sen refaktorointi TypeScriptille mahdollista. Vaikka palvelinpuolen toteutuksen laatu todettiin riittäväksi joka kehitysvaiheessa, toisi TypeScript entistä enemmän varmuutta palvelinpuolen toiminnan takaamiseksi.

Käyttäjähallinnan siirtäminen NewbieMaker-sovelluksesta Amazon Cognitoon oli askel oikeaan suuntaan. Vaikka autentikoinnin vastuu siirrettiin sovelluksen ulkopuolelle, on nykyisessä ratkaisussa silti puutteita, koska käyttäjät joutuvat edelleen käyttämään täysin erillisiä tunnuksia autentikointiin. AD-integraation toteuttaminen tulevaisuudessa tulee olemaan olennaista, sillä se eliminoi tarpeen NewbieMaker-sovelluksen erillisille käyttäjätunnuksille, jolloin kirjautuminen tapahtuu yrityksen käyttämiin sovelluksiin samoilla AD-tunnuksilla.

Toisen kehitysvaiheen jälkeen tehdyn pilottitestin perusteella käyttäjiltä nousi yksi selkeä toive uudeksi ominaisuudeksi: notifikaatiot. Notifikaatioiden avulla olisi mahdollista huomauttaa käyttäjää esimerkiksi tehtävän aikarajan lähestymisestä tai esimiestä siitä, että käyttäjä on lähettänyt palautetta. Tämänhetkisessä toteutuksessa kaikki käyttäjien tekemät toimet, kuten saadut palautteet täytyy tarkistaa proaktiivisesti. Notifikaatioiden toteuttaminen olisi yksi selkeä ja hyödyllinen jatkokehityskohta sovelluksen tulevaisuutta ajatellen.

8 YHTEENVETO

Tässä diplomityössä tutkittiin Cybercom Finland Oy:lle toteutetun yhden sivun web-sovelluksen (NewbieMaker) kehitysvaiheita. NewbieMaker on uusien työntekijöiden perehdyttämisen apuna käytettävä tehtävälistasovellus niin uusille työtekijöille kuin esimiehillekin. Sovellus on rakennettu pääosin MERN-pinon palasia hyödyntäen, mikä sisältää seuraavat teknologiat: MongoDB, Express, React ja Node.js. Työn tarkoituksena oli selvittää, miten NewbieMakerin kehitysvaiheiden aikana tehdyt valinnat vaikuttivat kehitysprosessiin ja sovelluksen laatuun. Työssä keskitytään pääosin sovelluksen teknologisiin valintoihin, mutta myös projektinhallinnallisten valintojen, kuten scrumin käyttöönoton vaikutusta on pohdittu.

NewbieMakerin kehitysprosessi jakaantui kolmeen erilliseen kehitysvaiheeseen, jotka esiteltiin tässä työssä erillisinä kokonaisuuksina. Jokaisessa kehitysvaiheessa esiteltiin kyseisessä vaiheessa tehdyt merkittävimmät valinnat ja syyt valinnan tekoon. Ensimmäisen kehitysvaiheen valinnat rakensivat NewbieMakerin perusarkkitehtuurin ja teknologiapinon. Toisessa kehitysvaiheessa tehtiin suuria muutoksia NewbieMakerin selainpuolen teknologioihin ottamalla käyttöön muun muassa TypeScript, Redux ja Material Design -periaatteet. Kolmannen eli viimeisen kehitysvaiheen tavoitteena oli parantaa NewbieMakerin sen hetkisen toteutuksen ylläpidettävyyttä, käyttäjähallintaa ja tuotantoon viemistä. Tavoitteiden saavuttamiseksi suoritettiin sovelluksen Redux-logiikan refaktorointi sekä alettiin hyödyntää AWS:n palveluja autentikointiin ja tuotantoympäristön luomiseen.

Tässä työssä pohdittiin kehitysvaiheiden aikana tehtyjen valintojen vaikutusta kehitysprosessiin vaihekohtaisesti. Käytännössä siis tutkittiin, miten aiemman kehitysvaiheen aikana tehdyt valinnat vaikuttivat seuraavaan kehitysvaiheeseen. Ei-teknologisista valinnoista scrum-viitekehyksen käytön hyödyt olivat huomattavat. Scrumin iteratiivinen malli mahdollisti NewbieMakerin jatkuvan parantamisen, sillä toteutetut toiminnallisuudet käytiin läpi jokaisen sprintin lopussa. Tämä mahdollisti korjausten ja parannusehdotusten tekemisen lyhyin väliajoin, minkä seurauksena turhan työn tai vääränlaisen toteutuksen tekemisen riski aleni ja ajankäyttö oli tehokkaampaa.

Tutkimuksessa huomattiin myös MERN-pinon todella selkeä hyöty kehitykseen. MERN-pinon ansiosta sovelluksen selain- ja palvelinpuolen kehityskielenä oli JavaScript, joka mahdollisti kehittäjien paremman ymmärtämisen sekä selain- että palvelinpuolen toiminnasta. Yhtenäisen kehityskielen ansiosta kehittäjien oli myös helppo siirtyä selainpuolen kehityksestä palvelinpuolen kehitykseen tai toisin päin. TypeScriptin käyttöönotto selain-

puolella toisessa kehitysvaiheessa ei juurikaan vaikeuttanut koodin ymmärtämistä, jonka ansiosta sen vaikutus projektiin oli vahvasti positiivinen. MERN-pinon tietokanta eli MongoDB mahdollisti myös dokumenttien datamallien helpon muuttamisen kehityksen aikana NoSQL-rakenteensa ansiosta.

NewbieMakerin käyttöliittymäkehitystä nopeutti ja selkeytti CSS-viitekehysten sekä React-kirjaston käyttö. CSS-viitekehysten avulla pystyttiin luomaan tyylikkäitä ja responsiivisia käyttöliittymäkomponentteja nopeasti, mikä edisti sovelluksen käytettävyyttä ja ulkonäköä. React mahdollisti SPA-sovelluksen helpon luonnin ja sen komponenttipohjaisen mallin ansiosta sovellus pystyttiin rakentamaan selkeistä erillisistä komponenteista, joka helpotti sovelluksen rakenteen ymmärtämistä. Toteutetun sovelluksen koon aliarvioimisen takia tilanhallinta jätettiin ensimmäisessä kehitysvaiheessa React-komponenttien sisälle, mistä seurasi tarpeettoman suuria tilamäärittelyjä sekä propsien liikuttelua komponenttien välillä. Tilanhallinnan ongelma korjattiin toisessa kehitysvaiheessa ottamalla käyttöön Redux-kirjasto, jonka avulla jokaisella sovelluksen komponentilla oli käytössään yhteinen tila. Reduxin käyttöönotto selkeytti NewbieMakerin komponenttien tilamäärittelyjä ja sen ansiosta säästyttiin myös turhalta komponenttien väliseltä propsien välitykseltä.

Tässä diplomityössä tutkitun NewbieMaker-sovelluksen kehitysprosessin perusteella voidaan luoda yhdenlainen ohjenuora tulevia sovelluskehitysprojekteja varten. Työssä käsiteltyjen teknologiavalintojen ja niiden vaikutusten perusteella voidaan helpottaa uusien projektien kehitysprosessia välttämällä mahdolliset virheet ja hidasteet ohjaamalla kehittäjät tekemään parempia valintoja alusta alkaen. Tässä työssä käsitellyt teknologiavalinnat liittyvät kuitenkin vain yhden sovelluksen kehitysprosessiin ja valintojen vaikutukset saattavat vaihdella riippuen toteutettavan projektin rakenteesta ja tavoitteista. Tehty tutkimus voidaan todeta onnistuneeksi, sillä tutkitun sovelluskehitysprojektin aikana tehtyjen valintojen vaikutukset kehitysprosessiin on analysoitu selkeästi ja kattavasti. Työssä saadut vastaukset vastaavat myös tämän diplomityön tutkimuskysymykseen.

LÄHTEET

[1] Mikkonen, T. ja Taivalsaari, A. *Web Applications-Spaghetti Code for the 21st Century*. Tekninen raportti. 2007.

[2] Vora, P. Web Application Design Patterns. *Web Application Design Patterns* (2009), 1–14. DOI: 10.1016/b978-0-12-374265-0.00001-3.

[3] Lehdonvirta, P. ja Korpela, J. K. *HTML5 sovellusalustana*. Helsinki: RPS-yhtiöt, 2013.

[4] LINFO. *Database Definition*. 2006. URL: http://www.linfo.org/database.html (viitattu 19. 02. 2020).

[5] Emmit A. Scott, J. *About this Book - SPA Design and Architecture: Understanding single-page web applications*. URL: https://livebook.manning.com/book/spa-design-and-architecture/about-this-book/ (viitattu 06. 02. 2020).

[6] Oluwatosin, H. S. Client-Server Model. *IOSR Journal of Computer Engineering* 16.1 (2014), 67–71. ISSN: 22788727. DOI: 10.9790/0661-16195771. URL: http://www.iosrjournals.org/iosr-jce/papers/Vol16-issue1/Version-9/J016195771.pdf.

[7] Hazaël-Massieux, D. *Javascript Web Apis*. 2016. URL: https://www.w3.org/standards/webdesign/script (viitattu 06. 02. 2020).

[8] Robie, J., Le Hégaret, P., Wood, L., Robie, J., Le Hégaret, P., Wood, L. ja Robie, J. *What is the Document Object Model?* 2000. URL: http://www.w3.org/TR/WD-DOM/introduction.html (viitattu 06. 02. 2020).

[9] Kaluža, M. ja Vukelić, B. Comparison of front-end frameworks for web applications developmentUsporedba front end frameworka za izradu web-aplikacija. *Zbornik Veleučilišta u Rijeci* 6.1 (2018), 261–282. ISSN: 18481299. DOI: 10.31784/zvr.6.1.19.

[10] Christensson, P. *Backend Definition*. URL: https://techterms.com/definition/backend (viitattu 16. 04. 2020).

[11] Restfulapi. *What is REST – Learn to create timeless REST APIs*. 2019. URL: https://restfulapi.net/ (viitattu 20. 02. 2020).

[12] Roy Thomas Fielding. *Fielding Dissertation: CHAPTER 5: Representational State Transfer (REST)*. 2000. URL: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm (viitattu 20. 02. 2020).

[13] Subramanian, V. *Pro MERN Stack*. Apress, 2019. ISBN: 9781484226520. DOI: 10.1007/978-1-4842-4391-6.

[14] Wiese, L. *Advanced data management: For SQL, NoSQL, cloud and distributed databases*. Berlin: De Gruyter, 2015, 1–352. ISBN: 9783110441413. DOI: 10.1515/9783110441413. URL: http://search.ebscohost.com/login.aspx?direct=true%

7B%5C&%7DAuthType=cookie,ip,uid%7B%5C&%7Ddb=nlebk%7B%5C&%7DAN=1107018%7B%5C&%7Dsite=ehost-live%7B%5C&%7Dscope=site.

[15] *NoSQL Databases | Technology solutions | DB Best*. URL: https://www.dbbest.com/technologies/nosql-databases/ (viitattu 20. 02. 2020).

[16] MongoDB. *NoSQL Databases Explained | MongoDB*. 2016. URL: https://www.mongodb.com/nosql-explained (viitattu 20. 02. 2020).

[17] Microsoft Corporate Author. *What is cloud computing? A beginner's guide | Microsoft Azure*. 2016. URL: https://azure.microsoft.com/en-gb/overview/what-is-cloud-computing/ (viitattu 21. 02. 2020).

[18] Karim, S. ja Soomro, T. R. *What Is Cloud Computing?* 2020. DOI: 10.4018/978-1-7998-1294-4.ch001. URL: https://www.citrix.fi/glossary/what-is-cloud-computing.html (viitattu 21. 02. 2020).

[19] Gore, A. *How to Not Screw Up UX in a Single-Page Application - DZone Web Dev*. 2020. URL: https://dzone.com/articles/how-to-not-screw-up-ux-in-a-single-page-applicatio (viitattu 14. 02. 2020).

[20] Aggarwal, S. *Modern Web-Development using ReactJS*. Tekninen raportti. 2018, 133–137. URL: http://ijrra.net/Vol5issue1/IJRRA-05-01-27.pdf.

[21] *Github.com*. URL: https://github.com/search?q=stars%5C%3A%5C%3E100&s=stars&type=Repositories (viitattu 12. 02. 2020).

[22] Shifa Martin. *List of Top JavaScript Frameworks 2020 For Front-End Developers*. 2019. URL: https://www.freecodecamp.org/news/complete-guide-for-front-end-developers-javascript-frameworks-2019/ (viitattu 12. 02. 2020).

[23] Gackenheimer, C. *Introduction to React*. 2015. ISBN: 9781484212462. DOI: 10.1007/978-1-4842-1245-5.

[24] Hayward, J. et al. *React: Building Modern Web Applications*. 1. painos. Packt Publishing, 2016, 910.

[25] *React – A JavaScript library for building user interfaces*. URL: https://reactjs.org/ (viitattu 18. 03. 2020).

[26] Schwaber, K. ja Sutherland, J. *The Scrum Guide™ The Definitive Guide to Scrum: The Rules of the Game*. Tekninen raportti. 2017.

[27] Shenoy, A. ja Prabhu, A. *CSS framework alternatives : explore five lightweight alternatives to Bootstrap and Foundation with project examples*. Apress, 2018. ISBN: 1484233980.

[28] *Jest · Delightful JavaScript Testing*. URL: https://jestjs.io/ (viitattu 02. 03. 2020).

[29] *Robot Framework*. URL: https://robotframework.org/ (viitattu 02. 03. 2020).

[30] *About - ESLint - Pluggable JavaScript linter*. URL: https://eslint.org/docs/about/ (viitattu 02. 03. 2020).

[31] *TypeScript Language Specification*. URL: https://github.com/Microsoft/TypeScript/blob/master/doc/spec.md (viitattu 11. 03. 2020).

[32] Rozentals, N. *Mastering TypeScript*. 2017, 552. ISBN: 9781784399665.

[33] Chinnathambi, K. *Learning React : a hands-on guide to building web applications using React and Redux*. Addison-Wesley Professional, 2018, 304. ISBN: 9780134843582. URL: https://www.safaribooksonline.com/library/view/learning-react-a/9780134843582/.

[34] *Async Flow | Redux*. URL: https://redux.js.org/advanced/async-flow#async-flow (viitattu 12. 03. 2020).

[35] *Read Me · Redux-Saga*. URL: https://redux-saga.js.org/ (viitattu 12. 03. 2020).

[36] Fox, R. The etymology of user experience. *Digital Library Perspectives* 33.2 (2017), 82–87. ISSN: 20595816. DOI: 10.1108/DLP-02-2017-0006. URL: www.emeraldinsight.com/2059-5816.htm.

[37] *Material-UI: A popular React UI framework*. URL: https://material-ui.com/ (viitattu 27. 03. 2020).

[38] *Style Guide | Redux*. URL: https://redux.js.org/style-guide/style-guide/ (viitattu 02. 04. 2020).

[39] *What is AWS*. URL: https://aws.amazon.com/what-is-aws/ (viitattu 20. 03. 2020).

[40] *What Is Amazon EC2? - Amazon Elastic Compute Cloud*. URL: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html (viitattu 20. 03. 2020).

[41] *What is AWS CloudFormation? - AWS CloudFormation*. URL: https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/Welcome.html (viitattu 20. 03. 2020).

[42] *Amazon Cognito User Pools - Amazon Cognito*. URL: https://docs.aws.amazon.com/cognito/latest/developerguide/cognito-user-identity-pools.html (viitattu 25. 03. 2020).

[43] *Create React App - Getting Started*. URL: https://create-react-app.dev/docs/getting-started/ (viitattu 13. 04. 2020).

[44] *Higher-Order Components – React*. URL: https://reactjs.org/docs/higher-order-components.html (viitattu 17. 04. 2020).