Verkkosovelluksen kehitys Scrum-projektinhal-lintamenetelmällä

Pyry Lamberg

2021 Laurea

Laurea-ammattikorkeakoulu

Verkkosovelluksen kehitys Scrum-projektinhallintamenetelmällä

Pyry Lamberg

Tietojenkäsittely

Opinnäytetyö

Syyskuu, 2021

Laurea-ammattikorkeakoulu **Tiivistelmä**
Tietojenkäsittely
Tradenomi (AMK)

Pyry Lamberg

Verkkosovelluksen kehitys Scrum-projektinhallintamenetelmällä

Vuosi 2021 Sivumäärä 37

Tämä opinnäytetyö käsittelee toimeksiantajalle tehtyä uutta verkkosovellusta. Opinnäytetyössä käsitellään kyseiseen verkkosovellukseen käytettyjä työkaluja ja teknologia valintoja. Opinnäytetyön tavoitteena on luoda hyvä käsitys siitä, miten sovelluskehitys toteutuu käytännön työelämässä ja mitä teknologia vaihtoehtoja kannattaa harkita verkkosovellusta kehittäessä.

Kehitystyö toteutettiin käyttämällä Scrum-projektinhallintamenetelmää, JavaScript-pohjaisia frontend-teknologioita sekä Java-ohjelmointikieltä tuottamaan sovellukselle RESTful-palvelu.

Opinnäytetyön tuloksena syntyi uusi verkkosovellus, joka jäi toimeksiantajan ylläpidettäväksi. Kehitystyön aikana projektin sovelluskehittäjät kehittyivät IT-alan ammattilaisina sekä saivat erittäin hyvän perehdytyksen toimeksiantajan toimintatapoihin. Kehitystyö onnistui kaikkien osapuolien mielestä hyvin ja toimeksiantaja pohtii jatkossakin palkkaavan useampia harjoittelijoita yhdellä kerralla.

Asiasanat: Sovelluskehitys, Scrum, JavaScript, Java

Laurea University of Applied Sciences **Abstract**
Degree Programme in Bachelor of Business Information Technology

Pyry Lamberg

Web application development with Scrum project management tool

Year 2021 Pages 37

The purpose of this Bachelor's thesis was to develop brand new web application. This thesis addresses the tools and technologies that were used in making of this web application. The goal of this thesis is to create good understanding about how software development is managed in real working environment and what technology choices should be considered when developing new web applications.

The development of this project was made with Scrum project management, JavaScript based frontend technologies and Java programming language to develop RESTful service for the application.

Result of this thesis was a new web application which was left in the employer's administration. During the development the project's software developers gained lots of experience as IT industry professionals and they received very upstanding introduction to the employer's procedure. Overall, every party was very satisfied by the results of this software development project and the employer is considering hiring multiple interns again.

Keywords: Web application development, Scrum, JavaScript, Java

Sisällys

1 Johdanto

Opinnäytetyön aiheena on moderni verkkosovelluskehitys Scrum-projektinhallintamenetelmällä, jonka konkreettista toteutumista rinnastetaan omaan kehitystyöhön. Opinnäytetyön toimeksiantaja haluaa jäädä nimettömäksi. Aihevalinta on luonnollinen, se tuo esille omaa osaamistani sovelluskehityksestä samalla antaen hyvän kehyksen tulevaisuuden verkkosovelluskehitykselle.

Toimeksiantajallani on nykyisellään vahvasti räätälöity sovellus, joka on osa laajempaa verkkosovellusten kokonaisuutta. Tästä halutaan eroon, kun käyttöliittymään ja muihin sovelluksen toimintoihin ei saa päivityksiä ja niiden käyttäminen sitoo toimeksiantajaa vanhoihin ohjelmistoversioihin. Uusi kehitetty verkkosovellus tulee toiminnallaan olemaan täysin vastaava edelliseen verrattuna, mutta se tullaan nostamaan modernille tasolle käyttämällä tuoreimpia JavaScript-kirjastoja sen kehityksessä. Tuoreimpia teknologioita käyttämällä voidaan varmistaa sovellukselle pitkää ikää ja hyvää mahdollisuutta ylläpitoon sekä päivityksiin.

Tässä opinnäytetyössä tullaan paneutumaan moderniin verkkosovellus kehitykseen teoriassa ja siihen, miten tuo tapahtuu todellisuudessa. Sovelluksen kehittämiseen käytettiin React ja Node.Js JavaScript-kirjastoja luomaan moderni ja responsiivinen ulkoasu. Sovellus tallentaa tietonsa IBM:n DB2-tietokantaan Java EE RESTful-palvelun avulla. Lisäksi sovellus on suojattu KeyCloak identiteetti- ja pääsynhallinnalla.

Opinnäytetyön tavoitteena on tuoda esille omaa osaamistani sovelluskehityksestä ja siihen liittyvistä muista tekijöistä, sekä luoda myös eräänlaista dokumentaatiota opinnäytetyössä kehitettävälle sovellukselle.

2 Lähtökohdat

2.1 Kehittämistavoite

Opinnäytetyön kehitystavoitteena on kehittää kokonaan uusi verkkosovellus, joka korvaa vanhan vastaavan sovelluksen. Nykyiseen sovellukseen päivitysten saaminen on hankalaa ja se sitoo toimeksiantajaa käyttämään vanhoja ohjelmistoversioita. Sovellus tulee helpottamaan usean IT-asiantuntijan päivittäistä työtä. Toimeksiantaja haluaa myös saada tuon sovelluksen omaan hallintaan ja ylläpitoonsa.

2.2 Aihealueen rajaus

Tässä opinnäytetyössä keskitytään verkkosovelluskehitykseen ja siinä käytettäviin teknologioihin sekä ketterään kehittämiseen. Itse kehitetty sovellus ei sinällään ole opinnäytetyön keskiössä, muuta kuin verkkosovelluksen kehyksen luomisessa. Opinnäytetyössä ei oteta kantaa sovelluksen tuotantoon viemiseen, testaamiseen, suunnitteluun tai sen määrittelyyn.

2.3 Keskeiset käsitteet

JSON	JavaScript Object Notation on tekstiformaattinen tapa esittää strukturoitua dataa.
XML	Extensible Markup Language on merkintäkielinen standardi, joka on ihmisen ja koneen luettavissa.
HTML	HyperText Markup Language on merkintäkieli, jolla kuvataan verkkosivuja.
React	React on JavaScript-kirjasto.
Npm	Npm on Noden mukana tuleva pakkaustenhallinta työkalu.
Node.Js	Avoimen lähdekoodin, järjestelmäriippumaton palvelinpuolen JavaScript suoritusympäristö.
CSS	Cascading Style Sheets määrittää verkkosivun ulkoasun tyylin.
BootStrap	Suosituin CSS-viitekehys.
REST	Representation State Transfer on arkkitehtuurimalli ohjelmointirajapintojen toteuttamiseen.

Git	Versionhallintatyökalu.
BitBucket	Lähdekoodin versionhallinta.
Express	Express on Node.Js verkkosovelluksille tehty viitekehys.
DB2	IBM:n kehittämä relaatiotietokanta.
KeyCloak	Avoimen lähdekoodin identiteetti- ja pääsyhallinta ratkaisu.
Jira	Tehtävienhallintaohjelmisto.
Http	HyperText Transfer Protocol, tiedonsiirto protokolla.
SPA	Single Page Application, Yhden sivun applikaatio
SQL	Structured Query Language, rakenteellinen kyselykieli
JVM	Java Virtual Machine, Javan suorittamiseen edellytetty virtuaalikone.

3 Moderni verkkosovelluskehitys

Vielä kymmenen vuotta sitten oli internetissä paljon staattisia verkkosivuja, eli sivuja, joiden lähdekoodi löytyy palvelimelta HTML- ja CSS-muotoisena ja se tarjoillaan käyttäjälleen juuri sellaisena, kuin se on tehty.

Nykyisin lähes kaikki verkkosivut ovat dynaamisia, jolloin verkkosivu luodaan vasta silloin, kun verkkoselain sitä pyytää. Dynaamisuus tulee erityisemmin huomioon, kun sivun tulee reagoida käyttäjän syötteisiin tai käyttäytymiseen. Lomakkeilla ja muilla erilaisilla syötteillä voidaan määritellä sitä, mitä palvelin lähettää takaisin käyttäjälleen. Myös esimerkiksi hyvin dynaamisella ohjelmointikielellä kuten JavaScriptillä voidaan luoda verkkosivuille responsiivisuutta ja dynaamisuutta. JavaScriptistä on tarjolla paljon kirjastoja, joista tämä kehitystyö nostaa esille etenkin React-kirjaston (Jörg, 2016).

SPA (Single Page Application) eli verkkosovellus, joka toimii vain yhdellä sivulla. Perinteisesti selain lataa verkkosivun joka kerta kun käyttäjä siirtyy samalla sivulla. Toisin taas SPA lataa sivun kerran ja luo jo ladatulle sivulle sisällön dynaamisesti, eli kun käyttäjä antaa syötteitä selain pyytää palvelimelta uutta dataa renderöitäväksi jo ladatulle sivulle. SPA ei pelkästään nopeuta sivuston selaamista, vaan helpottaa myös verkkosivujen ylläpitoa ja niiden tuotantoon julkaisua. Tässä kehitystyössä tullaan myös hyödyntämään SPA:n tarjoamia etuja (Angular University 2020).

3.1 Kehitysmenetelmät

3.2 Scrum-projektinhallinta

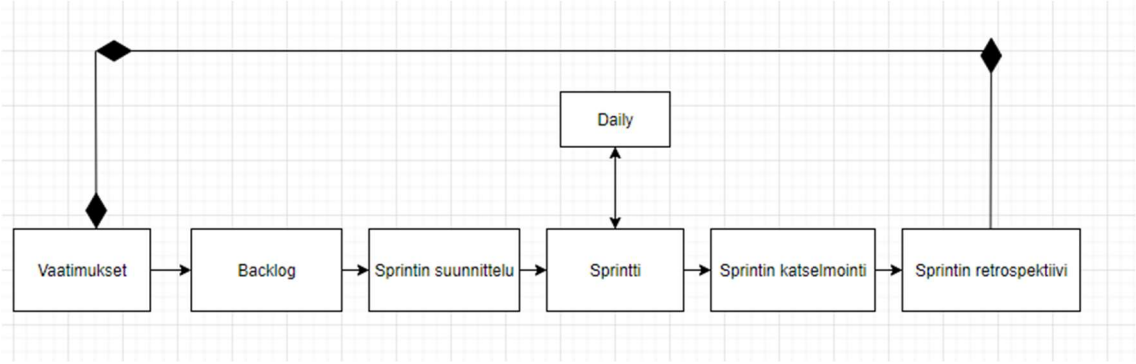
Scrum on työkalu, josta on oikein mainio lähteä liikkeelle missä tahansa projektissa, oli se sitten uuden ison verkkosovelluksen kehittäminen tai mökin rakentaminen, on Scrum hyvä työkalu kumpaan tahansa. Scrumin tarkoituksena on luoda kehykset toteutukselle, jotta voidaan parantaa tehokkuutta ja luoda nopeammin tuloksia. Scrumissa on keskeistä maalaisjärjen käyttö, eli keskitytään siihen mitä voidaan tänään tehdä, samalla kuitenkin tulevaa mielessä pitäen (Layton & Morrow 2018).

Scrum prosessi on ketterä ja sallii täten nopeat adaptoitumiset mahdollisiin muutoksiin, oli ne liiketoiminnallisia, teknologisia, uusia regulaatioita, innovaatioita tai mitä tahansa. Scrum noudattaa myös agilea manifestia, joka sanoo ketterästä kehittämisestä seuraavasti.

- **Individuals and interactions** over processes and tools
- **Working software** over comprehensive documentation
- **Customer collaboration** over contract negotiation
- **Responding to change** over following a plan

“That is, while there is value in the items on the right, we value the items on the left more.” (Kent, B ym. 2001).

Scrum prosessi koostuu jatkuvasta iteraatioista. Jokainen iteraatio pitää sisällään vaatimukset, tehtävälistan, sprintin suunnittelun, sprintin, dailyt, sprintin katselmoinnin ja retrospektiivin (Layton & Morrow 2018).



Kuvio 1: Scrum iteraatio

Scrum-tiimit koostuvat yleensä 5–9 hengen kehitystiimistä, jonka tukena on scrum master ja tuoteomistaja. On mahdollista, että projektitiimissä on useita Scrum-tiimejä, kuitenkin niin, ettei yhden Scrum-tiimin koko nousisi turhan suureksi. Tiimin ytimenä on kehitystiimi eli ne, jotka ovat luomassa kehitettävää tuotetta. Kehitystiimi toimii suoraan scrum masterin ja

tuoteomistajan kanssa, jotka yhdistävät liiketoiminnan ja kehityksen prioriteetit (Layton & Morrow 2018).

Scrum pitää sisällään selkeän tehtävälistan listan, jota yleisimmin kutsutaan backlogiksi. Tehtävälista pitää sisällään kaikki lähitulevaisuuden tehtävät sekä vanhat suoritetut tehtävät. Yleensä tehtävälistan ylläpitoon käytetään jotain työkalua, esimerkiksi tässä projektissa käytetään Atlassianin Jira projektinhallintatyökalua. Tehtävälistalle on tarkoitus kerätä kaikki tulevat tehtävät ja antaa niille prioriteetti, jonka perustella tehtävät voidaan asettaa tärkeysjärjestykseen. Tärkeysjärjestyksen lisäksi jokaiselle tehtävälle annetaan estimaatti, eli tarinapisteet. Yksi tarinapiste vastaa yhtä henkilötyöpäivää ja on tavanmukaista, että tarinapisteet annetaan Fibonacci:n lukujonon (1, 2, 3, 5, 8, 13, 21 jne.) mukaisesti. Joskus estimaattien antaminen tehtäville voi olla vaikeaa, jolloin tiimi voi pelata estimaatiopokeria. Estimaatiopokerista voi myös selvitä lisätietoa tehtävään liittyen, jos osa tiimistä ajattelee tehtävän kestävän viisi henkilötyöpäivää ja muutama 13 henkilötyöpäivää ja yksi ajattelee tehtävän kestävän kaksi henkilötyöpäivää. Vastaavassa tilanteessa tulisi näiden ääripäiden avata enemmän, miksi ajattelevat tehtävän kestävän heidän arvioiman ajan. Tämä voi selventää tehtävää, tai avata tärkeää keskustelua tehtävään liittyen. Tehtävälistaa ylläpitää tuoteomistaja Scrum Masterin avulla, mutta yleensä myös kehittäjät voi lisätä tehtävälistalle tehtäviä sprintin aikana (Layton & Morrow 2018).

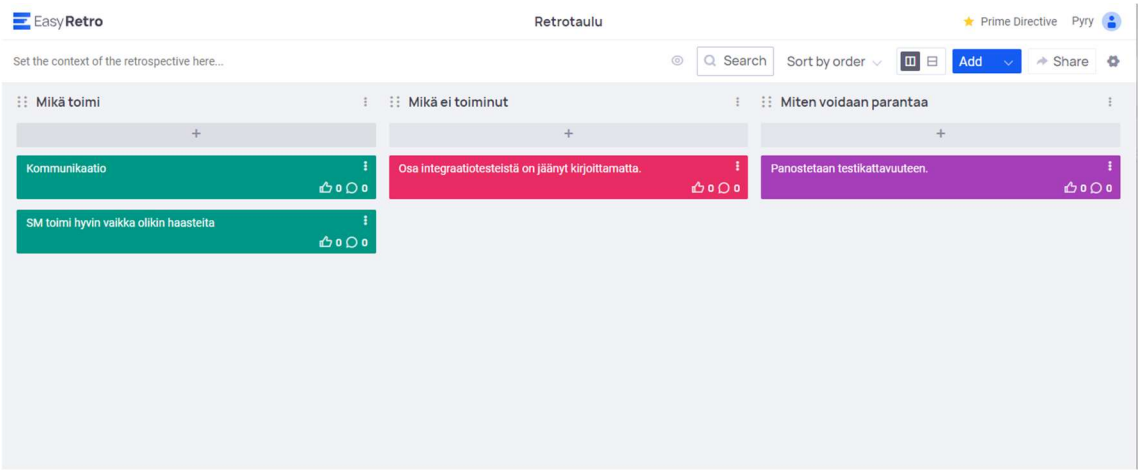
Sprintin suunnittelu on ennen sprintin aloitusta pidettävä tilaisuus, joka on suunnattu koko Scrum-tiimille. Sprintin suunnittelun tarkoituksena on kartoittaa mitä ehditään ja mitä voidaan tehdä seuraavan sprintin aikana. Sprintin suunnittelussa päätetään myös miten pitkä seuraava sprintti tulee olemaan ja mitkä ovat sprintin tavoitteet (Atlassian, 2021).

Sprintti on lyhyt aikajakso, yleensä kahdesta neljään viikkoon, mutta voi olla lyhyempikin, jolloin Scrum-tiimi tekee määritetyn määrän työtä. Sprintit pitää sisällään dailyt, eli päivittäiset lyhyet palaverit. Päivittäisissä palavereissa informoidaan muuta tiimiä siitä, mitä on saatu aikaan eilen ja mitä tehdään tänään, sekä otetaan esille mahdollisia ongelmatilanteita. Scrumin ketteryys ja kyky adaptoitua Scrum-tiimin mukaisesti pätee myös dailyihin ja tiimi voikin sopia, että pitää dailyt vain muutamana päivänä viikossa (Atlassian, 2021).

Sprintin katselmointi on yleensä hyvin kasuaali tilaisuus, joka suoritetaan sprintin lopussa. Katselmoinnissa tarkastellaan tiimin tekemää työtä, jota on sprintin aikana on tehty. Katselmoinnissa voidaan myös esitellä tiimin tekemisiä muille sidosryhmille (Atlassian, 2021).

Viimeinen tapahtuma Scrumin iteraatiossa on retrospektiivi. Retrospektiivissä reflektoidaan tiimin toimintaa kuluneen sprintin aikana. Retrospektiiviin on tarkoitus luoda rento ja mukava ilmapiiri, jossa tiimin on mahdollista oppia ja parantaa toimintaansa. Lyhykäisyydessään retrospektiiviin luodaan retrotaulu, jossa on kolme saraketta ”mikä toimi”, ”mikä ei toiminut” ja

”miten voidaan parantaa”. Kaikki tiimin jäsenet voivat anonyymisti antaa palautetta retrotaulun avulla (Easyretro, 2021).



Kuvio 2: Retrotaulu (Easyretro, 2021)

4 Käytettäviä teknologioita

Tässä luvussa esitellään muutamia teknologioita, joita kannattaa harkita verkkosovellusten kehittämiseen. Esitellyt teknologiat ovat kyseisen verkkosovelluksen kehityksessä käytettyjä teknologioita. Tarkoituksena on tuoda esille teknologioiden perustaa ja ydintoimintaa.

4.1 Sovelluksen käyttöliittymän teknologiat

Perinteisesti sovellus jaetaan kahteen osaan; frontend ja backend. Näistä frontend keskittyy tiedon esittämiseen käyttäjälle, kun taas backend toimii datan tallentajana käyttöliittymän ja tietokannan välissä. Backend tyypillisesti manipuoli dataa ja pitää sisällään liiketoimintalogiikkaa, kun taas logiikan puolesta frontend korkeintaan estää käyttäjää syöttämästä vääriä päivämääriä.

4.1.1 JavaScript

Sen lisäksi, että internetsivuilla on HTML-merkistökieltä ja CSS-tyylejä, tarvitaan lähes aina myös toiminnallisuutta. Tämän vuoksi lähes jokaisella verkkosivulla on jonkin muotoista JavaScript-ohjelmointikieltä. JavaScript on järjestelmäriippumaton, olio-orientoitunut tai funktionaalinen, script-kieli, jolla saadaan toiminnallisuutta verkkosivuille. JavaScriptillä voidaan esimerkiksi tehdä animaatioita ja painettavia nappeja. JavaScriptistä on tarjolla myös palvelin puolen versioita kuten Node.Js, jota käsitellään opinnäytetyön myöhemmissä osioissa (MDN Web Docs, 2021).

Toiminnallisuuden tarve tuli hyvin nopeasti nettisivuille heti kun internetin käyttö alkoi yleistä, tällöin NetScape kehitti JavaScriptin. NetScape lähetti JavaScript-kielen standardoitavaksi ECMAlle, The European Computer Manufacturer's Association, mutta siitä kuitenkin aiheutui patenttikiistoja ja täten standardisoitu kieli sai nimekseen ECMAScript, mutta todellisuudessa kaikki kutsuvat kieltä JavaScriptiksi (Flanagan, 2020).

Alkuperäinen suoritusympäristö JavaScriptille oli verkkoselain ja vieläkin se on kaikista yleisin paikka suorittaa JavaScriptiä. Myöhemmin kuitenkin pelkästään verkkoselain ei enää riittänyt, vaan tarve myös palvelinpuolen JavaScriptille alkoi nousta ja näin kehittyi Node.Js. Node.Js antaa JavaScriptille pääsyn itse käyttöjärjestelmään, jolloin sillä voidaan tallentaa tiedostoja, lähettää ja vastaanottaa dataa sekä tarjota http-kutsuja (Flanagan, 2020).

JavaScriptin toiminnallisuudesta on rakennettuna käyttäjäpuolelle ohjelmointirajapinta, niin sanottu API, eli Application Programming Interface. Ohjelmointirajapinnalla tarjotaan kehittäjille tapa implementoida ns. ydinkoodiblokkeja, jotka olisivat muuten erittäin hankalia tai mahdottomia ottaa käyttöön. Verkkoselaimiin on valmiiksi rakennettuna selaimen ohjelmointirajapinta, joka antaa koodille dataa käytettäväksi verkkosivulta. Yksi tunnetuimmista ohjelmointirajapinnoista on DOM, eli Document Object Model, jolla voidaan muuttaa verkkosivun HTML- ja CSS-rakennetta. Tämä mahdollistaa verkkosivun asun dynaamisen muuttamisen. (MDN Web Docs, 2021).

JavaScript sekoittuu helposti toiseen ohjelmointikieleen nimeltä Java, mutta ne ovat todellisuudessa hyvin erilaisia kieliä eivätkä liity toisiinsa oikein mitenkään. Vaikka molemmat kielet ovat syntaksiltaan jokseenkin saman näköisiä niillä on kuitenkin muutoin valtavia eroja. JavaScriptillä ei ole Javan staattista tyypitystä, eikä sen vahvasti tyypitettyjä muuttujia (kuvio 3). Javassa kaikki muuttujat, luokat ja metodit tulee esitellä ja nimetä, toisin kuin JavaScriptissä. JavaScript ei myöskään tarvitse erikseen näkyvyysmääreitä eikä sille tarvitse implementoida rajapintoja (MDN Web Docs, 2021).

```
/** AUTHOR: PYRY LAMBERG **/

/* Var esittelee muuttujan, jolle voidaan asettaa arvo tai olla asettamatta arvoa. Var muuttujan arvoa voidaan muuttaa */
var muuttuja;
var tervehdys = "Huomio!";

/* let esittelee muuttujan yhden koodiblokin käytettäväski, let muuttujaa arvoa voidaan muuttaa */
let luku = 5;

/* Koodiblokki on aaltusulkeiden sisään kirjoitettu koodi */
/* Let tervehdys näkyy vain koodiblokin sisällä ja on vain sen koodiblokin käytettävissä */
function tervehdiNimella(name) {
    let tervehdys = "Hei ,";
    alert(tervehdys + name);
}

/* Const muuttujat näkyvät vain oman koodiblokkinsa sisällä, mutta  niiden arvoa ei voida muuntaa */
const portti = 3003;
```

Kuvio 3 JavaScript-muuttujat

4.1.2 React

React on JavaScript-kirjasto, jonka on kehittänyt sovellusinsinööri Jordan Walke vuonna 2011 työskennellessään Facebookilla. Reactin tarkoitus on helpottaa ja parantaa verkkokäyttöliittymien kehittämistä. Reactista tehtiin avoimen lähdekoodin projekti JavaScript konferenssissa vuonna 2013, jolloin se liittyi käyttöliittymäkirjastojen kategoriaan, jossa oli jo ennestään mm. jQuery, Angular, Dojo ja muita. Reactin suosio jatkoi kehittymistä ja vuonna 2012 Netflix ilmoitti, että he aikovat käyttää Reactia heidän käyttöliittymänsä kehittämisessä. React adaptoitui myös nopeaan mobiilikehitykseen ja React Native julkaistiin mobiilisovellusten kehitystä varten. Nopeasti Reactin suosion kasvaessa myös erilaisia työkaluja tuli tarjolle. Viimeisimpiä mullistavia asioita Reactissa oli vuonna 2019 esitellyt React Hookit, jotka antoivat uuden tavan siirtää logiikkaa ja tietoa eri komponenttien välillä (Porello & Banks, 2020).

Mikä React sitten on? React on deklaratiivinen, tehokas ja joustava JavaScript-kirjasto käyttöliittymien rakentamista varten. Reactissa erityistä on sen mahdollisuus kirjoittaa HTML-merkkejä suoraan JavaScript-koodin sisään. Reactilla voidaan jakaa laajojakin ja kompleksisia käyttöliittymiä pieniin uudelleen käytettäviin komponentteihin (ReactJs Docs, 2021).

Kuvion 4. React-komponentti on hyvin yksinkertainen React-komponentti, joka saa ”message”-muuttujan ja palauttaa sen kutsujalleen HTML-formaatissa.

```
const Notification = ({ message }) => {
    return (
      <div className="error">
        {message}
      </div>
    )
  }

export default Notification
```

Kuvio 4 React-komponentti

React-komponentteja voidaan kutsua toisten komponenttien sisältä. Kuviossa 5. ”Notification”-komponentti otetaan käyttöön antaen sille propsi ”message”, jonka arvo on ”virhe viesti”.

```
const App = () => {
  return (
    <div>
      <Notification message={'Virhe viesti'} />
    </div>
  )
}

export default App
```

Kuvio 5 React-komponentin kutsu

Reactilla voi myös tehdä luokkapohjaisia komponentteja (kuvio 6).

```
class HeiMaailma extends React.component {
    render() {
        return (<h5>Huomenta, {this.props.etunimi}</h5>)
    }
}
```

Kuvio 6 React-luokkakomponentti

Reactin komponentit pääsevät parhaaseen hyötyyn, kun niihin lisätään tilanhallinta. Tilanhallinnalla voidaan muuttaa verkkosivun ulkoasua sen mukaan mitä käyttäjä tekee, tai vaihtoehtoisesti vaikka lähettää käyttöliittymän palvelimelle dataa tietyn toiminnon jälkeen. Reactin yksi tunnetuimmista hookeista on useState-hook, joka voidaan importoida React-sovellukseen react-paketista.

```
1   /* AUTHOR: PYRY LAMBERG */
2   import React, { useState } from 'react'
3
4   const [count, setCount] = useState(0);
5   const App = () => {
6   return (
7       <div>
8           <p>Nappia on painettu {count} kertaa</p>
9           <button onClick={() => setCount(count +1)}></button>
10      </div>
11      )
12  }
13
14  export default App
```

Kuvio 7 React useState

Kuviossa 7. importoidaan useState-hook react-paketista, jonka jälkeen komponentti saa tilan, joka saa arvokseen 0. UseState-hook palauttaa taulukon, jossa on kaksi arvoa ”count” ja ”setCount”. Muuttujaan ”count” on tallennettu tilan nykyinen arvo, eli tässä tapauksessa nolla. Toinen muuttuja ”setCount” on viittaus funktioon, jolla voidaan asettaa uusi arvo ”count”-muuttujalle. Nyt sovellus luo yksinkertaisen nappulan, jota painamalla ”setCount(count +1)”-funktio nostaa ”count”-muuttujan arvoa yhdellä jokaisen painalluksen jälkeen.

Asioiden uudelleen renderöinti on React-sovelluksen perimä, kun jotain tapahtuu sovelluksessa komponentteja uudelleen renderöidään ja muutetaan juuri siihen tilaan mihin se on haluttu. Hyvänä esimerkkinä jatkuvasta uudelleen renderöinnistä ja uudelleen renderöinnin suorittamisesta tilan perusteella on react-paketissa oleva useEffect-paketti. Käyttämällä useEffect-pakettia voidaan tarkasti määritellä, milloin sivun tulee latautua uudestaan.

```
1   /* AUTHOR: PYRY LAMBERG */
2   import React, { useState, useEffect } from 'react'
3
4   const [count, setCount] = useState(0);
5
6   useEffect(() => {
7       console.log("Nappia on painettu!");
8   }, [count]);
9
10  const App = () => {
11  return (
12      <div>
13          <p>Nappia on painettu {count} kertaa</p>
14          <button onClick={() => setCount(count +1)}></button>
15      </div>
16      )
17  }
18
19  export default App
```

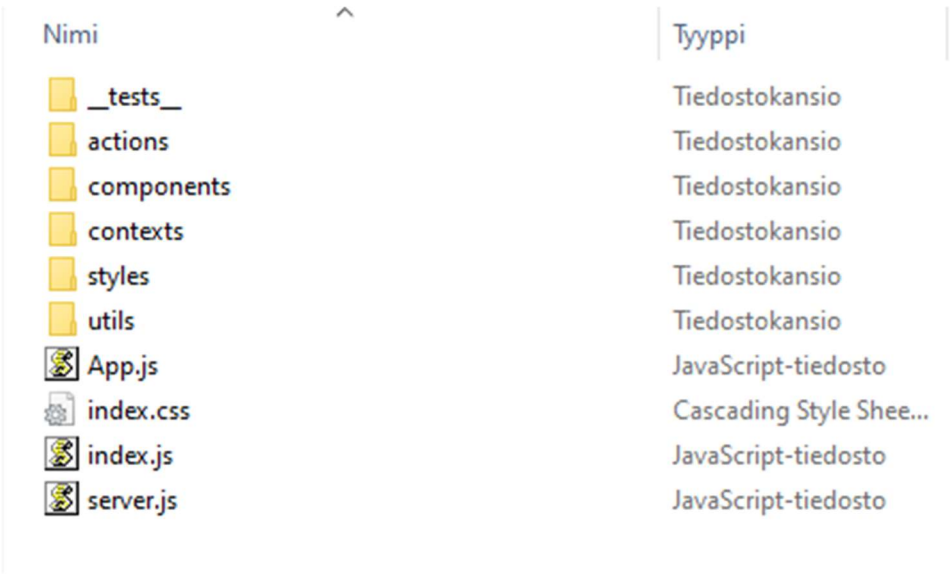
Kuvio 8 React useEffect-paketin käyttö

Kuviossa 8. käytetään useEffect-hookkia siten, että joka kerralla, kun muuttujan ”count” arvo muuttuu useEffect-hook herää ja renderöi sivun uudelleen, sekä kirjoittaa lokiin ”Nappia on painettu”.

Reactin hookit auttavat myös tiedonvälityksessä ohjelman sisällä. UseContext-hookkin on hyvä vaihtoehto Redux-kirjaston käyttämiselle. Redux on avoimen lähdekoodin JavaScript-kirjasto, jolla voidaan muuttaa verkkosovelluksen tilaa. Etenkin pienemmissä ohjelmissa useContext-hookin käyttäminen on osoittautunut hyväksi tavaksi hallita sovelluksen tilaa. UseContextia voidaan esimerkiksi käyttää tallentamaan useita eri useState-hookkeja ja viedä ne sovelluksen

juuren käytettäväksi. Kun sovelluksen komponentti on sisällytetty contextiin, voidaan minkä vaan komponentin sisällä ottaa contextiin tallennettuja hookkeja käyttöön samalla pitäen contextiin tallennetut tiedot tallessa, sekä muiden komponenttien käytettävinä.

Kun React-sovelluksia tehdään, on tavanomista, että komponentit, contextit, hyödykkeet ym. refaktoroidaan omiin kansioihinsa. Refaktoroimalla tiedostot omiin kansioihin selkeytetään projektin rakennetta ja yksinkertaisestaan koodia, sekä tehdään siitä huomattavasti helpommin luettavaa. Tyypillinen React-sovelluksen src-kansion kansiorakenne voisi näyttää kuvion 9. mukaiselta.



Kuvio 9 Tyypillinen React-sovelluksen rakenne

4.1.3 Node.Js

Node.js on asynkroninen avoimen lähdekoodin erityisesti palvelimilla pyörivä JavaScript-ajoympäristö, jonka kehitti Ryan Dahl vuonna 2009. Node.Js pohjautuu Google Chromessa ajettavaan JavaScript-tulkkiin nimeltä V8. Node.Js:n asynkronisuudella tarkoitetaan sitä, että Node.Js:llä voidaan tehdä yhtäaikaisesti useita ohjelmointirajapinta kutsuja, sillä se ei odota yhden ohjelmointirajapinta kutsun valmistumista, vaan suoritta pyydetyn kutsun ja palaa siihen, kun se on saanut vastauksen. Tämä mahdollistaa hyvin skaalautuvien, kevyiden ja tehokkaiden verkkosovellusten luomisen.

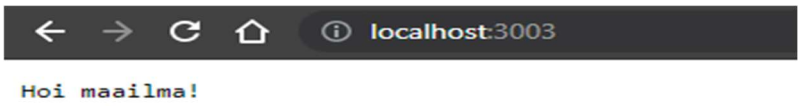
Kuuden kuukauden välein uusi Node.js versio menee ”current”-julkaisustatukseen, joka antaa JavaScript-kirjastojen kehittäjille aikaa adaptoitua tukemaan niitä. Kuuden kuukauden jälkeen ”current”-tilassa ollut versio julkaistaan aktiiviseen LTS-tilaan ja samalla vanhoista Node.Js:n parittomista versioita loppuu tuki. LTS-julkaisut saavat pitkän 30 kuukauden tuen

kriittisille bugeille. Tuotannossa olevien sovellusten tulisi käyttää vain aktiivisia tai ylläpidettyjä LTS-versioita (Node.Js Doc, 2021).

```
JS server.js > ...
1   //* AUTHOR: Pyry Lamberg **//
2
3   const http = require('http')
4
5   const app = http.createServer((request, response) => {
6     response.writeHead(200, { 'Content-Type': 'text/plain' })
7     response.end('Hoi maailma!')
8   })
9
10  const port = 3003
11  app.listen(port)
12  console.log(`Palvelin on käynnistetty porttiin: ${port}`)
```

Kuvio 10 Node.Js ”Hoi maailma”-esimerkki

Kuvio 10. on esimerkki Node.Js-sovelluksesta, joka vastaa ”Hoi maailma!”, kun sitä kutsutaan. Ensin rivillä 3. Node.Js ottaa käyttöön sisäänrakennettunsa http-moduulin ja sen jälkeen riveillä 5–8 luodaan http-palvelin, joka saa tapahtumankäsittelijän. Tapahtumankäsittelijä kutsutaan joka kerta, kun verkkoselaimella mennään ”localhost:3003”-osoitteeseen. Portin mitä Node.Js kuuntelee, voi valita antamalla konstantti muuttuja ”3003” rivillä 10, jonka jälkeen muuttujaan ”app” sijoitettu http-palvelin aloittaa kyseisen portin kuuntelun. Lopuksi annetaan konsolille vastauksena mihin porttiin palvelin on käynnistetty. Jos nyt käynnistää Node.Js-palvelimen ja menee selaimellaan osoitteeseen ”http://localhost:3003” aukeaa kuviota 11. vastaava sivu.



Kuvio 11 Node.Js ”Hoi maailma”-esimerkki

Tällä tavoin Node.Js-palvelimen rakentaminen on mahdollista, mutta turhan työlästä etenkin, jos kyseessä on yhtään laajempi toteutus. Tätä varten on olemassa npm-paketteja, joilla voidaan helpottaa ohjelmointia ja tehdä siitä huomattavasti yksinkertaisempaa.

Npm on maailman suurin sovellusrekisteri, jota käyttää sovelluskehittäjät ympäri maata jakaakseen, lainatakseen ja käyttääkseen npm-paketteja. Myös monet yksityiset kehittäjät käyttävät npm:n tarjoamia paketteja (npm Docs, 2021). Npm-pakettien käyttöönotto on hyvin

suoraviivaista ja esimerkiksi express-paketin asentaminen tapahtuisi antamalla konsolille komento ”npm install express”.

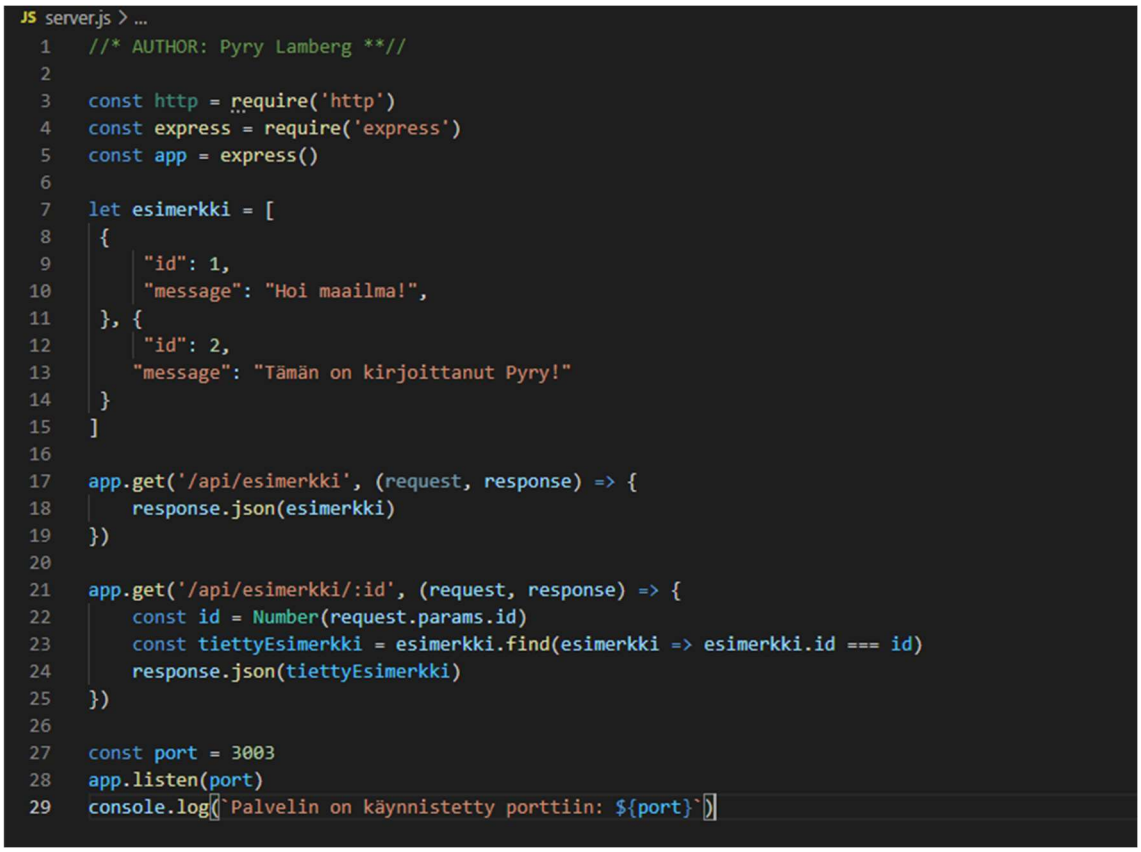
Kaikki npm-paketit asennetaan automaattisesti samaan kansioon nimeltä node_modules. Sovellukseen jo asennettuja paketteja voi tarkastella npm:n luomasta package.json-tiedostosta. Kuvion 12. package.json-tiedosto näyttää seuraavalta, kun siihen on asennettu express-paketti.

Kuvio 12: package.json

Package.json-tiedostosta voidaan tarkastella ”dependencies”-osaa, jossa näkyy asennettujen pakettien nimet ja niiden versiot. Sen lisäksi, että esimerkin package.json-tiedostossa kerrotaan npm:lle mitä paketteja sovellukseen on asennettu. Npm myös listaa Node.Js-sovelluksen nimen, version, kuvauksen, main tiedoston ja testi scriptin. On tavallista, että näiden lisäksi package.json-tiedostoon määritellään proxy, npm engine versiot, eslintConfig ja devDependencies. Nyt esimerkin tiedostoon ei ole luotu kuin yksi automaattisesti generoitu scripti.

Tällä hetkellä, jos Node.Js-palvelimeen tekee muutoksen, se tulee ensin sammuttaa ja sitten käynnistää, jotta muutokset tulevat voimaan. Tätä varten on kehityksen tueksi useimmiten otettu käyttöön nodemon-kirjasto, jonka asennus tapahtuu ”npm install nodemon –save-dev nodemon”. Komennon lisämääre ”—save-dev” tallentaa nodemonin versiotiedot devDependencyjen alle package.json-tiedostossa. Nodemon käynnistää palvelimen heti uudelleen, kun muutoksia tehdään palvelimen tiedostoihin. Nodemon otetaan käyttöön lisäämällä package.json-tiedoston ”Scripts” alapuolelle esimerkiksi skripti ”dev”: ”nodemon server.js”,

jolloin antamalla komentoriville komennon ”npm start dev” nodemon käynnistyy ja tarjoaa server.js-tiedostoa.



```
JS server.js > ...
1   //* AUTHOR: Pyry Lamberg **//
2
3   const http = require('http')
4   const express = require('express')
5   const app = express()
6
7   let esimerkki = [
8    {
9        "id": 1,
10       "message": "Hoi maailma!",
11   }, {
12       "id": 2,
13      "message": "Tämän on kirjoittanut Pyry!"
14   }
15  ]
16
17  app.get('/api/esimerkki', (request, response) => {
18      response.json(esimerkki)
19  })
20
21  app.get('/api/esimerkki/:id', (request, response) => {
22      const id = Number(request.params.id)
23      const tiettyEsimerkki = esimerkki.find(esimerkki => esimerkki.id === id)
24      response.json(tiettyEsimerkki)
25  })
26
27  const port = 3003
28  app.listen(port)
29  console.log(`Palvelin on käynnistetty porttiin: ${port}`)
```

Kuvio 13 Express esimerkki

Kuviossa 13. käytetään express-pakettia, se ensin alustetaan antamalla konstantti ”express” ja kertomalla sille mitä se tarvitsee toimiakseen, eli ”require(’express’)”. Seuraavaksi määritetään esimerkki json, joka pitää sisällään kaksi elementtiä, joilla on erilaiset ”message”-arvot ja joilla on oma ”id”-arvo. Rivillä 17. kerrotaan expressille, että kun kutsu tulee polkuun ’/api/esimerkki’ palautetaan vastauksena koko json. Nyt kun käyttäjä menee selaimellaan

osoitteeseen ”http:/localhost:3003/api/esimerkki”, niin verkkoselain saa vastaukseksi koko esimerkki json:in, joka näkyy kuviossa 14.

Kuvio 14: Node.Js palaute selaimessa

Expressille voidaan myös antaa lisämääre polkuun esimerkiksi ”/:id”, jonka Node.Js osaa poimia osoitteen polusta. Kuviossa 6. rivillä 22. kerrotaan Node.Js:lle, että id on löydettävissä request.params alapuolelta id-kentästä. Koska polussa oleva muuttuja on tyyppiä String, se pitää ensin muuntaa numeroksi käyttämällä JavaScriptin funktiota Number. Tämän jälkeen konstantti ”tietty esimerkki” hakee esimerkki json:ista id:tä vastaavan json-elementin, jonka jälkeen se palautetaan verkkoselaimelle. Expressillä voidaan tehdä GET, POST, PUT ja DELETE kutsuja.

Hieman monimutkaisempi kutsu voisi esimerkiksi näyttää seuraavalta

```
app.put('/api/muokkaus', (req, res) => {

    const body = req.body
    const muokkattava = {

        kuvaus: String(body.kuvaus),
        id: body.id,
        muokkaaja: String(req.kauth.grant.id_token.content.given_name)
    }
    axios.put(`${baseURL}muokkaus`, muokkattava, {
        agent: httpAgent
    })
        .then(data => {
            res.status(201).json({
                status: 'onnistui',
                data: data.config.data
            })
            console.log('vastaus: ', data.data)
        }).catch(error => {
            console.log('Tapahtui virhe', error)
            res.status(500).end()
        })
})
```

Kuvio 15: Node.Js PUT

Kuviossa 15. nähdään miten Node.Js käyttää PUT-operaatiota. Kun Node.Js saa PUT-kutsun se saa mukanaan requestin, jossa on lähetettävä data, tässä se on lyhennetty req. Sen jälkeen konstanttiin ”muokattava” puretaan bodyn mukana tulleet tiedot ja lähetetään ne axios-paketin avulla palvelimelle. Jos kutsu onnistuu, niin selain saa vastaukseksi responsen 201, statuksen ja datan, sekä kirjoittaa lokiin onnistuneen vastauksen tiedot. Kun taas epäonnistuessa saa vastaukseksi http 500-statuksen ja kirjoittaa lokiin virheen ja sen sisältämät tiedot.

Sen lisäksi, että Node.Js toimii palvelimena, joka tarjoaa dataa, sillä voidaan myös tarjota käyttöliittymä. Kun React-sovellus rakennetaan, eli buildataan, se rakentuu sovelluksen juurikansioon nimeltä ”build”. Kun Node.Js julkaistaan esimerkiksi pilvipalveluun toimintaan, niin se tarjoaa Reactin käyttöliittymän lähettämällä selaimelle build-kansioon tallentuneen index.html-tiedoston (kuvio 16).

```
app.use("/", [
    keycloak.protect(),
    express.static(path.join(__dirname, 'build'), {
        setHeaders: (res, path) => {
            if (!path.endsWith('.html')) {
                res.setHeader('Cache-Control', 'public, max-age=604800, immutable');
            }
        }
    }
    )])

app.get('/*', (req, res) => {
    res.sendFile(__dirname + '/build/index.html');
})
```

Kuvio 16: Node.Js build

4.1.4 Keycloak identiteetti ja pääsynhallinta

Keycloak on avoimen lähdekoodin identiteetti ja pääsynhallinnan työkalu, joka on tarkoitettu suojaamaan moderneja verkkosovelluksia, REST-palveluita ja mobiilisovelluksia. Keycloak antaa mahdollisuuden määrittää itse käyttäjät sisältävän tietokannan tai vaihtoehtoisesti käyttäjät voidaan hakea Windowsin toimialueen käyttäjätietokannasta ja hakemistopalvelusta tai LDAP-palvelimilta (Thorgersen & Silva, 2021).

Keycloakin konfigurointi Node.Js-palvelimella on hyvin yksinkertaista, sillä Keycloakin tavoitteena on ollut kehittää palvelu, joka olisi mahdollisimman hyvin suojattu ja mahdollisimman yksinkertainen käyttää (Thorgersen & Silva, 2021).

Keycloakin käyttöönotto käyttöliittymässä vaatii ensin Keycloak-palvelimen konfiguroinnin, sen jälkeen Node.Js-palvelimelle annetaan Keycloakin tiedot json muodossa, jonne on määritelty autentikaatio-palvelin ja realm mistä Keycloak hakee tiedot. Tämän jälkeen Keycloakkia voidaan käyttää express-session npm-paketin avulla. Jotta keycloak tallentaisi istunnon selaimeen ja jottei autentikaatiota tarvitsisi tehdä jatkuvasti keycloak hyötyy valtavasti express-session-paketin mukana tulevasta memoryStoresta. Memorystoreen voidaan tallentaa nykyinen istunto ja Keycloakin antama token. Keycloakin käyttö Node.Js:ssä tapahtuu kutsumalla ”keycloak.protect()”-funktiota, jolloin osoitteen polkuun mentäessä keycloak tarkistaa käyttäjän tokenin. Tokenin puuttuessa käyttäjää pyydetään kirjautumaan sisään. Keycloakille voidaan määritellä kuka voi kirjautua sisään antamalle argumentteja esimerkiksi ”keycloak.protect(’admin’)” päästäisi vain admin-tasoiset käyttäjät kirjautumaan. Kuviossa 17. on esitelty Keycloakin toimitaan vaadittu koodin osuus kokonaisuudessaan. Kuviossa 17.”Keycloak.protect” suojaa sovelluksen kirjautumisella kaikilta, mutta määrittää erikseen, ettei GET, POST, PUT tai DELETE-kutsuja voi tehdä kuin tietyllä määreellä olevat käyttäjät.

”process.env.ROLE_VIEW ” ja ”process.env.ROLE_EDIT” on eroteltu kahteen eri ympäristö-muuttujaan sallien vain EDIT oikeudella olevien tehdä muokkaavia tai poistavia https-kutsuja.

```
const express = require("express");
const app = express();
const session = require("express-session");
const Keycloak = require("keycloak-connect");

let memoryStore = new session.MemoryStore();

app.use(
  session({
    secret: "some secret",
    resave: false,
    saveUninitialized: true,
    store: memoryStore,
  })
);

const keycloak = new Keycloak({
  store: memoryStore,
});

app.use(keycloak.middleware());

app.use("/", [
  keycloak.protect(),
  express.static(path.join(__dirname, "build"), {
    setHeaders: (res, path) => {
      if (!path.endsWith(".html")) {
        res.setHeader("Cache-Control", "public, max-age=604800, immutable");
      }
    },
  }),
]);

app.get("/api/*", [keycloak.protect(process.env.ROLE_VIEW)]);
app.post("/api/*", [keycloak.protect(process.env.ROLE_VIEW)]);
app.put("/api/*", [keycloak.protect(process.env.ROLE_EDIT)]);
app.delete("/api/*", [keycloak.protect(process.env.ROLE_EDIT)]);
```

Kuvio 17 Keycloak Node.Js

4.2 RESTful-palvelut

REST on akronyymi sanoista **RE**presentational **S**tate **T**ransfer. Se on arkkitehtuurillinen tyyli hajautetulle hypermedia systeemille (RestfulApi.net, 2021). REST sai alkunsa Roy Fieldining väitöskirjasta Architectural Styles and the Design of Network-based Software Architectures, jonka hän esitteli vuonna 2000. Fielding jakaa RESTin kuuteen eri perusteeseen. (Fielding 200)

- Client-server
- Stateless
- Cache

- Uniform Interface
- Layered System
- Code-On-Demand

Client-server, eli asiakas-palvelin-malli, perusteella tarkoitetaan käyttöliittymän erottamista tiedon tallentamisesta vastaavasta palvelusta, eli käyttöliittymän ja palvelun tulee olla riippumattomia toisistaan. Tällä mahdollistetaan usean käyttöliittymän yhdistäminen yhteen palveluun. (Fielding 2000)

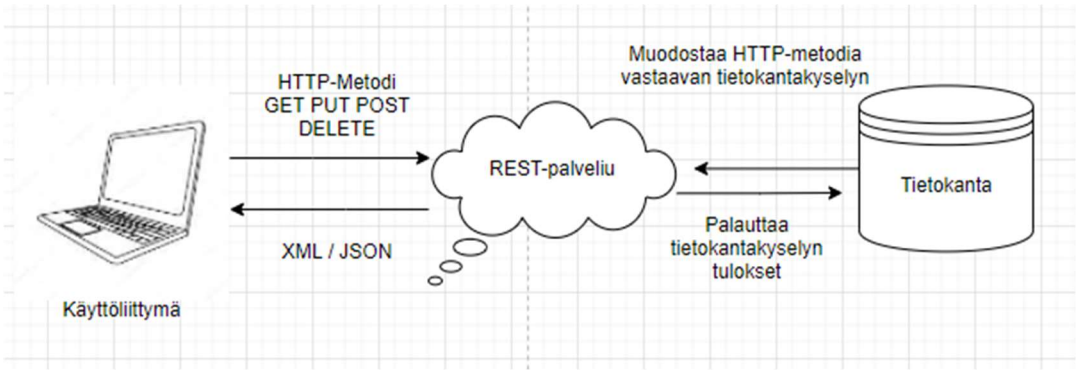
Stateless, eli tilattomuudella tarkoitetaan, että palvelun ja käyttöliittymän välinen tiedonjako pitää olla tilatonta siten, että jokainen käyttöliittymän tekemä pyyntö palvelulle pitää sisällään kaiken tarvittavan tiedon palvelun kannalta, eikä se saa käyttää mitään palvelimelle tallennettua tietoa hyväkseen. Näin tila jää täysin käyttöliittymän vastuulle. (Fielding 2000)

Cache, eli välimuistilla, edellytetään pyyntöjä pitämään tieto siitä, voidaanko vastaus tallentaa välimuistiin. Välimuistilla voidaan vähentää toistuvien pyyntöjen tekemistä palvelulle, jos tieto löytyy välimuistista ei sitä tarvitse kysyä palvelimelta uudestaan, joka puolestaan parantaa suorituskykyä. (Fielding 2000)

Uniform Interface tarkoittaa yhdenmukaista rajapintaa, eli palvelimen vastaama data on aina samassa formaatissa kuin sinne lähetetty data. (Fielding 2000)

Layered System, eli kerroksittainen järjestelmä, jonka arkkitehtuurillinen hierarkia sallii kerrosten kommunikoinnin vain välittömän seuraavan kerroksen välillä. (Fielding 2000)

Viimeinen peruste Code-on-Demand on koodin lataamista pyynnöstä, eli REST sallii käyttöliittymän toiminnallisuuden laajentamisen lataamalla palvelimelta koodia, joka suoritetaan käyttöliittymässä, usein miten ne ovat scriptejä. (Fielding 2000)

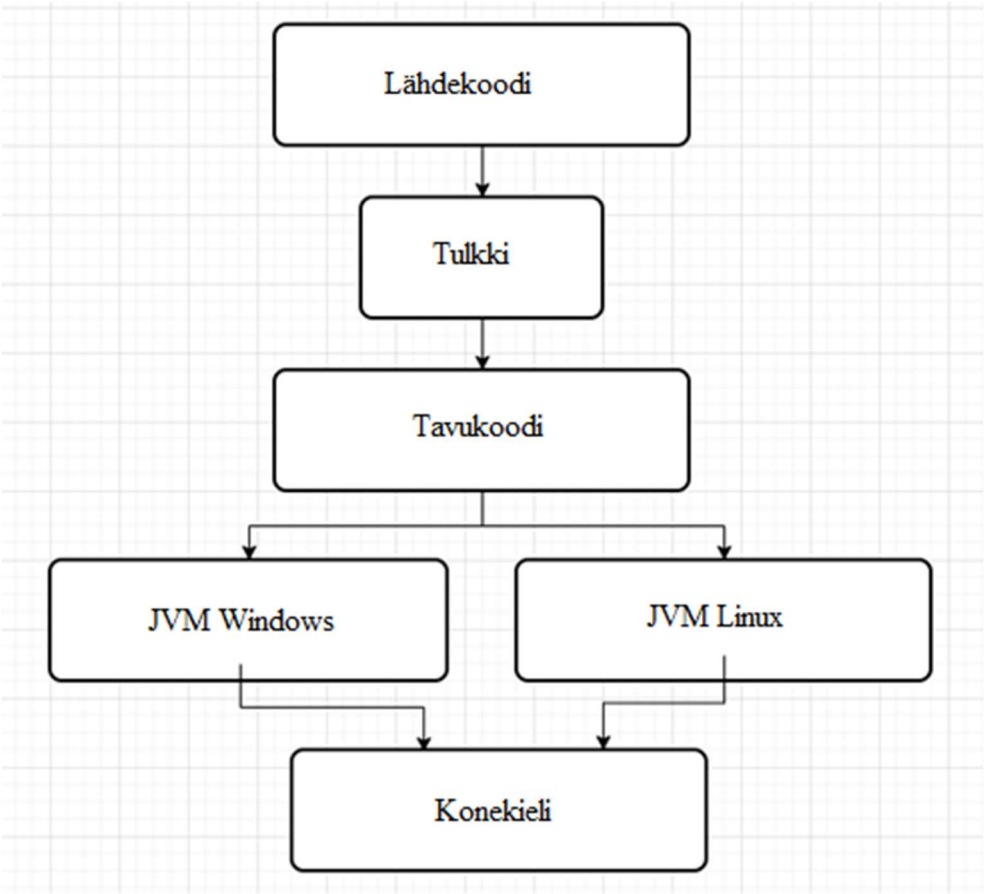


Kuvio 18 REST-ohjelmointirajapinnan toimintakaavio

4.3 Java

Java on suhteellisen vanha ohjelmointikieli ja se on kehitetty vuonna 1991 Sun Microsystems yrityksessä. Java oli alun perin nimeltään ”Oak”, mutta se muutettiin Javaksi vuonna 1995. Javalla on paljon ominaisuuksia, mutta niistä tärkeimmät ovat Javan alustariippumattomuus, yksinkertaisuus, olio-orientoineisuus, dynaamisuus ja tehokkuus. Javasta yritettiin tehdä mahdollisimman yksinkertaista, jotta ohjelmoijat voisivat mahdollisimman helposti ottaa Javan käyttöön ja käyttää sitä tehokkaasti. Voidaan sanoa, että jos on jo ennestään ohjelmointi kokemusta niin Javan oppiminen on helppoa. Javasta piti tehdä olio-ohjelmointi kieli, jonka lähdekoodin ei tarvitse olla yhteensopiva minkään muun ohjelmointikielen kanssa. Tämä mahdollisti Javan kehittämisen aloittamisen niin sanotulta puhtaalta pöydältä (Schildt, 2019).

Java ratkaisee kyberturvallisuuteen ja siirrettävyyteen liittyvät ongelmat Java Virtual Machinella. Tämä tarkoittaa sitä, ettei itse Javan lähdekoodi ole suoritettavaa koodia, vaan se pitää ensin kääntäjällä muuntaa tavukoodiksi. Kun kääntäjä on muuntanut koodin tavukoodiksi, voidaan se tämän jälkeen ajaa Java Virtual Machinella, joka tekee tavukoodista tietokoneen luettavaa konekieltä (Kuvio 19). Tästä tuleekin mahdollisuus suorittaa Javaa millä tahansa alustalla, jolle voidaan asentaa Java Virtual Machine (Schildt, 2019).



Kuvio 19 Java-ohjelmointikielen suoritus

Aiemmin käsittelimme JavaScriptiä, jossa tyypitys on hyvin löyhää, toisin kuin Javassa. Java on vahvasti tyypitetty kieli, eli kaikilla Javan muuttujilla ja määritelmillä on tyyppi ja tyypit ovat vahvasti määriteltyjä. Sen lisäksi myös kaikki metodit on tyypitettävä (Schildt, 2019).

Java ei ole pelkästään ohjelmointikieli vaan myös alusta. Java alusta, eli platform, on tietynlainen ympäristö missä Java-ohjelmia suoritetaan. Javalle on olemassa neljä erilaista alustaa (Oracle, 2012).

- Java Standard Edition, eli Java SE pitää sisällään Javan ydintoiminnallisuuden (Oracle, 2012).
- Java Enterprise Edition, eli Java EE on rakennettu Java SE-alustan päälle, mutta se on tarkoitettu suorittamaan laajasti skaalautuvia, luotettavia ja tietoturvallisia verkkosovelluksia (Oracle, 2012).
- Java Micro Edition, Java ME on hieman karsitumpi versio Java SE:stä, sekä sillä on omia valmiita kirjastoja, jotka auttavat tukemaan kehitystä pienille alustoille, kuten puhelimille. Java ME-sovellukset ovat usein Java EE-sovelluksen asiakkaita (Oracle, 2012).
- Viimeinen Java alusta on JavaFX, joka on tarkoitettu luomaan rikkaita internet-sovelluksia, eli Rich Internet Applications (RIA). JavaFX-alustalle rakennetut sovellukset ovat yleensä perinteisen tietokone sovelluksen näköisiä (Oracle, 2012).

Pelkällä Javalla kuitenkin esimerkiksi RESTful-rajapintojen tekeminen olisi kovin työlästä ja hankalaa. Tätä varten kehitetty työkaluja, kuten Spring-viitekehyksen Spring Boot -työkalu ja Apache Maven. Maven on työkalu, jolla helpotetaan Java-projektienhallintaa ja niiden rakentamista. Maven rakentaa projektin käyttäen Project Object Modelia, eli lyhennettynä POM. Maven luo informaatiota projektista hyödyntäen POM-tiedostoa sekä projektin lähdekoodia. Ehkäpä tärkein Mavenin tuottama informaatio on tieto siitä mitä riippuvuuksia, eng. Dependency, projektissa on käytetty (Apache Maven, 2021).

POM on XML-tiedosto, joka on Mavenin toiminnan kannalta pakollinen. POM-tiedosto sisältää tietoa projektista ja konfiguraatiotiedot mitä Maven käyttää projektin rakentamiseen. Kun Maven suorittaa tehtävää se lukee POM-tiedostosta tarvittavat konfiguraatiot ja tiedot suorituksen tekemiseksi (Apache Maven, 2021). Hyvin yksinkertainen POM.xml-tiedosto (kuvio 20).

```xml
<project>
  <modelVersion>4.0.0</modelVersion>

  <groupId>fi.yritys.projekti</groupId>
  <artifactId>module</artifactId>
  <version>1.0.2</version>
</project>
```

Kuvio 20 POM.xml

POM-tiedostoon on yleensä myös aina listattu riippuvuuksia, joiden avulla Maven-projektille voidaan antaa huomattava määrä lisäominaisuuksia. Esimerkiksi JavaEE-API-riippuvuuden listaaminen POM-tiedostolle kävisi kuvion 21. mukaisesti.

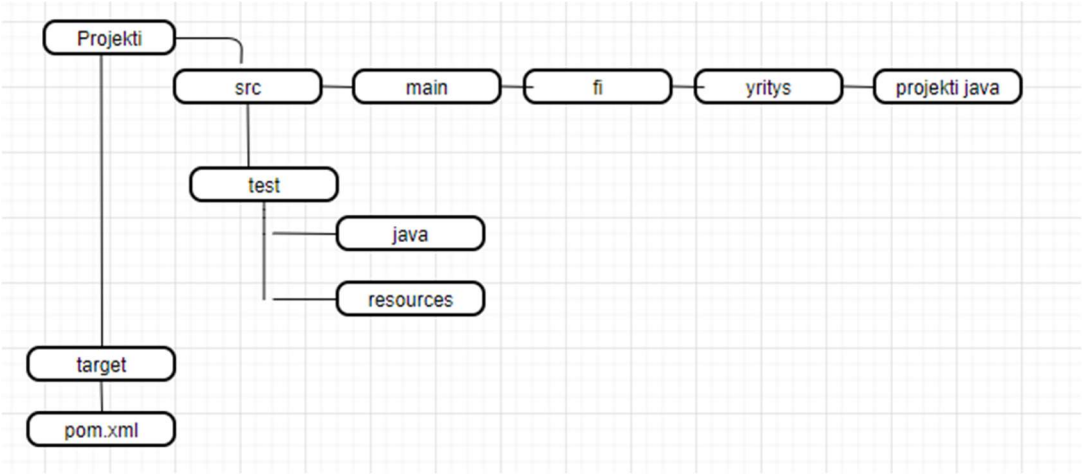
```xml
<project>
  <modelVersion>4.0.0</modelVersion>

  <groupId>fi.yritys.projekti</groupId>
  <artifactId>module</artifactId>
  <version>1.0.2</version>

<!-- JavaEE API -->
    <dependencies>
        <dependency>
            <groupId>javax</groupId>
            <artifactId>javaee-api</artifactId>
            <version>7.0</version>
        </dependency>
    </dependencies>
</project>
```

Kuvio 21 POM.xml JavaEE-API

Maven-projektin tiedostorakenne on hieman erilainen kuin perinteisen Java-sovelluksen rakenne (kuvio 22).



Kuvio 22 Maven-projektin rakenne (Apache Maven, 2021)

4.4 Java RESTful-rajapintana

Java EE -alustalle RESTful-rajapinnan toteuttaminen voidaan tehdä luomalla Maven-projekti ja antamalla sille riippuvuuden JAX-RS. JAX-RS-ohjelmointirajapinta käyttää Java ohjelmointikielen annotaatioita helpottaakseen ja yksinkertaistaakseen REST-rajapinnan luomista (Oracle, 2013). JAX-RS:n sisältämistä annotaatioista hyödyllisimpiä ovat seuraavat.

- @Path-annotaatio, jolle voidaan antaa parametrinä URI
- @GET, @POST, @PUT ja @DELETE-annotaatioilla kuvataan http-metodeita.
- @PathParam-annotaatio on tapa ottaa parametreja käytettäväksi URI:sta.
- @Produces-annotaatiota voidaan esimerkiksi käyttää metodilta palautuvan mediatyypin määrittelemiseen

```
/** AUTHOR: PYRY LAMBERG **/

// Luodaan polku kyseiselle Java luokalle.
// Todellinen URI voisi näyttää tältä http://localhost:8080/api/talo/45
@Path("/talo/taloId")
// HTTP GET metodi
@GET
// Tuottaa Json tyyppistä palautetta
@Produces(MediaType.APPLICATION_JSON)
// @PathParam ottaa URIsta taloId ja sijoittaa sen Integer muuttujaan taloId
public Response haeTiettyTalo(@PathParam("taloId") Integer taloId) {
    return talonhakuService.haeTiettyTalo(taloId);
}
```

Kuvio 23 Java EE RESTful-luokan esimerkki

Kuviosta 23. huomaa, että itse tietokantakutsut ja liiketoimintalogiikka on delegoitu talonhakuService-luokan vastuulle. Tämä on tyypillistä, sillä näin saadaan refaktoroitua lähdekoodia, joka yksinkertaistaa koodin ymmärtämistä ja helpottaa sen lukemista. Service-luokka huolehtii tietokantayhteyden luomisesta ja sql-lauseiden tekemisestä, jotka välitetään tietokannalle suoritettavaksi.

4.5 Versionhallinta

Versionhallinta, englanniksi version control tai source control, on tapa seurata ja hallinnoida lähdekoodiin tulevia muutoksia. Versionhallinta tallentaa kaikki lähdekoodiin tehdyt muutokset ja voi tarpeen tullen siirtää sovellusta ajassa taaksepäin. Jos lähdekoodi on tallennettuna versionhallintaan – mikä on erittäin tavanomaista – silloin kaikki kehittäjät pääsevät käsiksi siihen ja voivat kehittää sovellusta saman aikaisesti. Versionhallinta myös suojaa projekteja täysiltä katastrofeilta ja ihmisen tekemiltä virheiltä, jotka voisivat pahimmassa tapauksessa tuhota koko sovellusprojektin. Versionhallinta osaa myös yhdistää yhden kehittäjän tekemät muutokset toisen kehittäjän tekemiin muutoksiin, mutta joskus ongelmat ovat suurempia ja ne vaativat lähdekoodin yhdistämistä manuaalisesti (Atlassian, 2021).

Versionhallintaan käytetään yleensä työkaluja ja niistä oma suosikki on komentorivillä toimiva Git. Git on avoimen lähdekoodin projekti, jonka on kehittänyt Linus Torvalds vuonna 2005 (Atlassian, 2021). Git vaatii versionhallinnan kaverikseen esimerkiksi BitBucket tai GitHub ovat tähän mainioita vaitoehtoja. Gitin käyttö on hyvin suoraviivaista ja sen käyttö on helposti kenen tahansa opeteltavissa. Lyhyimmillään Git-komentoriviä voi käyttää muutamilla komennoilla.

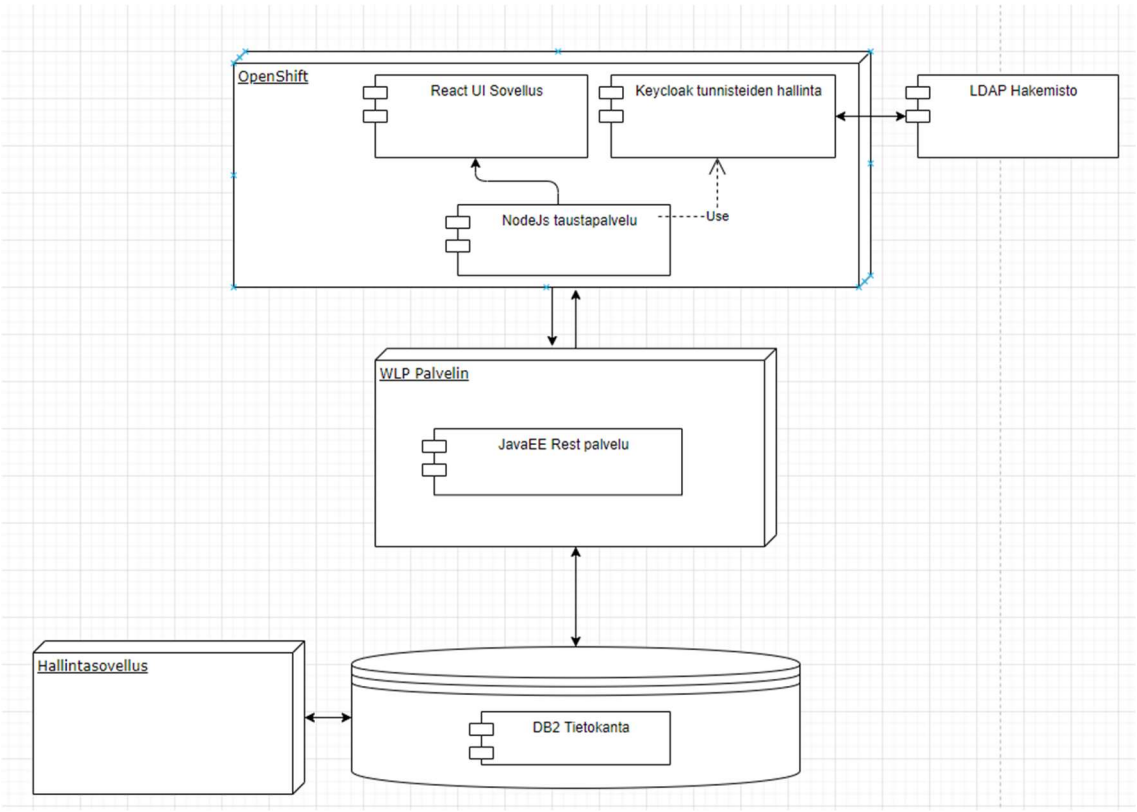
- Git Clone kloonaa projektin versionhallinnasta lokaalille tietokoneelle.
- Git Branch tekee uuden haaran versionhallintaan.
- Git pull hakee haaran tuoreimmat muutokset versionhallinnasta.
- Git Status näyttää lokaalinversion nykyisen statuksen, esimerkiksi kloonattuun lähdekoodiin tehty muutos näkyy Git Statuksen listaamana.
- Git add lisää tietyn tiedoston Gitin ns. lavastusalueelle eng. staging area.
- Git commit -m ”Tämä on ensimmäisen commitin nimi” lisää kaikki lavastusalueella olevat tiedostot yhteen committiin ja tyhjentää lavastusalueen. On myös tapana antaa commitille sitä kuvaava nimi.
- Git Push työntää kaikki commitit sille määritettyyn haaraan versionhallinnassa.

Versionhallintaan on myös tehty käyttöliittymällä varustettuja työkaluja kuten Git GUI, GitHub Desktop, SourceTree ja Eclipse IDEn lisäosa Git.

5 Verkkosovelluksen kehitys

Verkkosovelluksen kehittämiseen valittiin kolme kehittäjää, jotka olivat kaikki työharjoittelijoita. Työharjoittelijat saivat tuekseen Scrum Masterin, jotta talon tavat ja kehitysmenetelmät tulisivat tutummaksi. Jotta kehitys vastaisi mahdollisimman paljon normaalia työympäristöä päätettiin työtä tehdä Scrum-toimintamallia noudattaen. Tiimille sovittiin dailyt joka aamulle, sekä sovittiin ensimmäisestä suunnittelupäivästä.

Ensimmäinen suunnittelupäivä aloitettiin käymällä lävitse asiakkaan tarpeita ja perehtymällä vaatimusmäärittelyihin. Vaatimusmäärittelyiden perusteella pystyttiin tehdä teknologia valintoja, sekä aloittaa sovellusarkkitehtuurin pohtiminen. Teknologioiksi valikoitui React luomaan responsiivisen ja modernin ulkoasun, sekä toimeksiantajan oma bootstrap CSS-viitekehys tukemaan sovelluksen tyylejä. React-käyttöliittymä piti julkaista pilvipalveluun tuotantoon, joten valittiin Node.Js-taustapalvelu tarjoamaan React-käyttöliittymää ja samalla luomaan suojauksen frontendille hyödyntäen KeyCloak identiteetin- ja pääsynhallintasovellusta. Verkkosovelluksen palvelurajapinnan toteutukseen valittiin Java EE -sovellus, joka toimii WLP-palvelimella. Saatiin kuvion 24. mukainen sovellusarkkitehtuuri valmiiksi.



Kuvio 24 Verkkosovelluksen arkkitehtuuri

Kehitystyö aloitettiin tekemällä Proof of Concept ensimmäisen sprintin aikana, jolla luotiin lisää luottamusta sovellusarkkitehtuurin toiminnalle. Ensimmäiseen demoon, joka oli

ensimmäisen sprintin viimeisellä viikolla, oli tavoitteena saada kovakoodattua dataa esittävä käyttöliittymä valmiiksi. Ensimmäinen sprintti meni nopeasti, jonka aikana kehitysympäristöt oli saatu pystytettyä sekä käyttöliittymänrunko rakennettua.

Käyttöliittymänrungon toteutuksen jälkeen katseet kohdistuivat JavaEE RESTful-palvelun luomiseen. Aloitetiin miettimällä mitä Maven-riippuvuuksia projekti tulisi tarvitsemaan, sekä minkälaisia polkuja olisi hyvä saada heti toimintaan, että käyttöliittymä voitaisiin muuntaa pois kovakoodatusta datasta. Näin saimme toisen suunnittelupäivän pidettyä ja kuviot olivat selkeät.

Toiseen demoon olimme saaneet käyttöliittymän rakennettua, joka esitti tietokannasta haettua dataa. Kehitys oli päässyt kovaan vauhtiin ja kun oli kyse suhteellisen pienestä sovelluksesta, niin demot muutetiin pidettäväksi joka perjantai, josta muodostui nopeasti viikon kohokohta. Demoissa pyrittiin vahvistamaan kehittäjien ja käyttäjien yhteisiä näkemyksiä, sekä validoimaan vaatimusmäärittelyiden toteutumista.

Kehitys oli saatu täyteen vauhtiin ja pitikin hetkeksi pysähtyä miettimään sovelluksen suojausta, koska oli aika julkaista ensimmäinen versio sovelluksesta testiympäristöön. Suojaukseksi valittiin Keycloak, jolla pystyi suojaamaan käyttöliittymän, sekä välittämään Keycloakin tokenin myös REST-palvelulle, jonka jälkeen sovelluksen tietoturva oli taattu.

Muutama sprintti meni tosi nopeasti, kun tekemistä oli yllin kyllin ja kaikilla homma luisti mainiosti. Tiimi kommunikoi erittäin hyvin ja dailyissä käytiin eilisen ja tulevan päivän tapahtumat hyvin kattavasti lävitse.

Muutama kuukausi aloituksen jälkeen sovellus oli saavuttanut tuotantovalmiuden, se oli siis hyvin lähellä valmista. Muutamia ominaisuuksia oli vielä millä ei ollut kiire, eivätkä ne olleet kriittisiä sovelluksen toiminnan kannalta, joten päätettiin siirtää sovellus tuotantoon. Hyvin nopeasti tuotantoon päästyään sovelluksen kaikki ominaisuudet saatiin toimimaan ja suuremmilta bugeiltakin oli säästytty, projekti oli valmis ja jäi meidän kolmen ylläpidettäväksi. Sovellukseen tulee silloin tällöin lisättävää, mutta muuten se on toiminut erittäin luotettavasti ja on saanut paljon kiitosta ja kehuja uusilta käyttäjiltään.

6 Sovelluksen kehittämisen tulokset

Kehittämisen suurin tulos oli toteutettu uusi sovellus, joka jäi toimeksiantajan ylläpidettäväksi ja on edelleen allekirjoittaneen ylläpidettävänä. Sovellus on parantanut monen järjestelmäasiantuntijan päivittäistä työtä ja tehnyt siitä tehokkaampaa. Toimeksiantaja on päässyt myös eroon vanhasta sovelluksesta, johon ei enää saanut päivityksiä ja nyt päivitykset ovat vain muutamien sähköpostiviesti keskusteluiden takana, eli allekirjoittanut on sovelluksen

ylläpitäjä. Sovellus on täyttänyt kaikki vaatimuksensa ja hieman enemmänkin. Se tekee juuri sitä mitä tilattiin juuri siten, miten haluttiin ja siihen on lisätty myös ominaisuuksia, joita vanhassa sovelluksessa ei ollut.

Kehityksen tuloksena löytyi myös muutakin, kuin pelkästään kehitetty sovellus. Toimeksiantaja näki miten hyvin kolmen samaan aikaan aloittaneen harjoittelijan tiimi toimi, unohtamatta tietenkään mahtavaa Scrum Masteriamme. Toimeksiantaja aikookin jatkossa panostaa tähän. Harjoittelijoita on kehitystyömme perusteella paljon helpompi heittää suoraan niin sanottuun syvään päätyyn, jos heitä on useampi kuin yksi. Kaikki kolme harjoittelijaa työllistyivät toimeksiantajalle ja kaikkien työsuhde on jatkunut edelleen. Kolmen harjoittelijan samanaikainen aloittaminen ja sen onnistuminen tietenkin pitää oletuksen, että kolme henkilöä tulee hyvin toimeen ja osaa kommunikoida sujuvasti työssään. Kommunikointiin tiimi käytti sähköpostia, pikaviestipalvelua ja skypeä. Mielestäni kehitystyössä käytetty pikaviestipalvelu, jossa oli tiimille oma kanava, auttoi paljon hyvän kommunikoinnin toteutumisessa.

7 Yhteenveto

Kehitystyön päämääräinen tavoite oli luoda uusi verkkosovellus vanhan sovelluksen tilalle. Sovelluskehitys tehtiin käyttäen Scrum-menetelmää ja sen apuna Jira-tehtävienhallintaa. Kehitystyö yritettiin saada mahdollisimman paljon aitoa asiakasprojektia vastaavaksi työksi, vaikka se tehtiin yrityksen sisäiseen tarpeeseen.

Kehitystyön teknologiavalinnat osoittautuivat etenkin käyttöliittymän osalta erinomaisiksi valinnoiksi. Käyttöliittymän kehittämiseen käytetty JavaScriptin React-kirjasto sai huomattavaa suosiota tiimimme kehittäjien keskuudessa, myös Node.Js:n osaaminen kehittyi tiimissä projektin aikana. Backendin osalta tiimi tekisi nyt hieman toisin ja käyttäisi Javan Spring Boot-viitekehystä hyödykseen sen yksinkertaisuuden vuoksi. Spring Boot-viitekehys mahdollistaisi myös sovelluksen RESTful-palvelun suorittamisen pilvipalvelussa ilman erillistä WLP-palvelinta.

Kehitystyön aikana löydettiin myös muita positiivisia asioita, kuten se miten harjoittelijoista ja Scrum Masterista koostunut tiimi pystyi loistamaan ilman suurempia ongelma ja sai tuotettua toimivan sovelluksen aikatavoitteiden puitteissa. Suurena vaikuttavana tekijänä tiimin suoriutumiseen oli sen hyvä kommunikointi ja hyvä yhteishenki. Toimeksiantaja näki tilanteen ja aikoo jatkossakin panostaa siihen, että ottaa muutaman harjoittelijan yhden sijasta. Muutama samaan aikaan aloittanut harjoittelija pystyy antamaan toiselleen vertaistukea, joka parantaa ongelmanratkaisua ja vähentää harjoittelijan tuomaa rasitusta muulle tiimille.

8 Jatkokehitysehdotukset

8.1 Javan Spring-viitekehys

Nykyisen sovelluksen RESTful-palvelu toimii WebSphere Liberty Profile, eli WLP-palvelimella ja on riippuvainen siitä. Toimeksiantajalla on tavoitteena luopua WLP-palvelimien käytöstä lähivuosien aikana, sekä siirtää kaikki palvelut ajettavaksi OpenShift-ajoympäristöön. OpenShift on Red Hatin kehittämä hybridipilvipalvelu, joka toimii Kubernetes-sovellusohjelmistolla. Se miksi sovellus ylipäätään rakennettiin WLP-palvelimesta riippuvaiseksi, johtui toimeksiantajan sen aikaisista arkkitehtuurilinjauksista. Sen lisäksi että Spring Boot -viitekehys on hyvin yhteensopiva Kuberneteksen ja OpenShiftin kanssa sen itsenäisen toimintatavan vuoksi, se tarjoaa myös ohjelmoijalle helpotuksia.

Javan Spring-viitekehys on suosittu avoimen lähdekoodin yritystason viitekehys, jolla on tarkoituksena luoda itsenäisiä sovelluksia, jotka toimivat Java Virtual Machinella. Java Spring Boot on taas työkalu, joka helpottaa verkkosovellusten luontia Spring-viitekehyksellä (IBM, 2021). Spring Bootin dokumentaatiossa sanotaan:” Spring Boot makes it easy to create stand-alone, production-grade Spring based Applications that you can "just run.”” (Spring, 2021).

Spring Bootin keskeisimpiä ominaisuuksia on

- Mahdollisuus tehdä itsenäisiä Spring-sovelluksia.
- Sulautettu Tomcat, Jetty tai Undertow, eli sovellus ei tarvitse WAR-tiedostoja.
- Antaa aloitus riippuvuudet (eng. Dependency) yksinkertaistaakseen build konfiguraatioita.
- Automaattinen Spring ja kolmannen osapuolen kirjastojen konfigurointi.
- Tuotantovalmiita ominaisuuksia kuten metriikka ja sovelluksen ”health checks”.
- Ei koodin generointia tai tarvetta XML-konfiguraatioille (Spring, 2021).

Jatkokehityksenä pelkästään Springin tuoma mahdollisuus tehdä sovelluksen palvelusta itsenäinen WLP-palvelimesta riippumaton on jo riittävä syy pohtia uuden palvelun kehittämistä. Näiden lisäksi Spring Boot tekee Java koodista myös turvallisempaa ja yksinkertaisempaa (Spring, 2021).

9 Oman oppimisen arviointi

Projekti oli minun ensimmäinen sovellusprojektini, johon osallistui muitakin henkilöitä kuin minä itse. Projektin alussa teoria sovelluskehityksestä oli tuttua, mutta käytännön tekemistä oli vain harjoiteltu koulussa, eikä aitoa työelämän tuottamaa osaamista ja varmuutta vielä ollut. Projektin alussa myös ajattelin, että sovelluksen kehittämien täysin etänä olisi

haasteellista. Teknologioista React, Node.Js ja Java – tuo lempilapseni - oli ennestään itselle tuttuja, mutta täysin uutena haasteena oli OpenShift-ajoympäristö, WLP-palvelimet ja KeyCloak-suojaus.

Projektin aikana teoreettinen osaamiseni ketterästä työskentelystä sai tukea käytännön tekemisestä ja ketterät toimintatavat tulivat hyvin nopeasti tutuksi ja ymmärrys myös siitä, miksi niin halutaan tehdä nousi nopeasti esille. Teknologinen osaamiseni kehittyi mielestäni valtavasti, kun aiemmin omat sovellusprojektit ovat kuitenkin olleet omassa ”kellarissa” kehitettyjä viikonloppu projekteja. Eniten vahvistin osaamistani React ja Node.Js puolella, kun Java kuitenkin oli se enemmän tuttu ohjelmointikieli. Javastakin uutena haasteena oli Apache Maven ja muutenkin Java EE -ajoalusta.

Tämän sovellusprojektin ja harjoittelujakson jälkeen työllistyin toimeksiantajalle määräaikaiseen työsuhteeseen, joka on jo tätä kirjoittaessa on muuttunut vakituiseen työsuhteeseen. Olen mielestäni täydessä kiidossa kehittymään alan osaajaksi.

Lähteet

Painetut

Flanagan, D. 2020. JavaScript – The Definitive Guide.

Layton, M. & Morrow, D. 2018. Scrum for Dummies.

Porcello, E. & Banks, A. 2020. Learning React.

Schildt, H. 2019. Java The Complete Reference Eleventh Edition

Thorgersen, S. & Silva, P. 2021. Keycloak – Identity and Access Management for Modern Applications

Sähköiset

Angular University 2020. Angular Single Page Applications (SPA): What are the Benefits, Viitattu 25.8.2021 https://blog.angular-university.io/why-a-single-page-application-what-are-the-benefits-what-is-a-spa/.

Apache Maven Project 2021. Introduction, Viitattu 5.9.2021 https://maven.apache.org/what-is-maven.html

Atlassian 2021. What is scrum, Viitattu 22.08.2021 https://www.atlassian.com/agile/scrum

Atlassian 2021. What is Version control, Viitattu 6.9.2021 https://www.atlassian.com/git/tutorials/what-is-version-control

Easyretro 2021. What is Fun Retrospective, Viitattu 22.08.2021 https://Easyretro.io

Fielding, R. 2000. Architectural Styles and the Design of network-based Software Architectures. Viitattu 4.9.2021 https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

IBM, 2021. What is Java Spring Boot? Viitattu 18.9.2021
https://www.ibm.com/cloud/learn/java-spring-boot

Jörg, K. 2016. Introducing Web Development, Viitattu 15.8.2021
www.apress.com/gp/book/9781484224984

Kent, B. Mike, B. Arie, V. Alistair, C. Ward, C. Martin, F. James, M. Jim, H. Andrew, H. Ron, J. Jon, K. Brian, M. Robert, C. MellorKen, S. Jeff, S. Dave, T. 2001. Manifesto for Agile Software Development, Viitattu 22.08.2021 https://agilemanifesto.org/

MDN Web Docs. JavaScript, Viitattu 1.9.2021 https://developer.mozilla.org/en-US/docs/Web/JavaScript

Node.js 2021. About documentation, Viitattu 1.9.2021 https://nodejs.org/en/docs/

Npm Docs 2021. Npm Docs, Viitattu 1.9.2021 https://docs.npmjs.com/

Oracle, 2012. Your First Cup, Viitattu 5.9.2012 https://docs.oracle.com/javaee/6/firstcup/doc/gkhoy.html

Oracle, 2013. Java EE 6 Tutorial, Viitattu 5.9.2021 https://docs.oracle.com/javaee/6/tutorial/doc/gilik.html#gilqb

ReactJs Docs 2021. Getting Started, Viitattu 4.9.2021 https://reactjs.org/docs/getting-started.html

Restfulapi.net 2020. What is Rest, Viitattu 4.9.2021 https://restfulapi.net/

Spring, 2021. Spring Boot 2.5.4, Viitattu 18.9.2021 https://spring.io/projects/spring-boot#overview

Spring, 2021. Why Spring, Viitattu 18.9.2021 https://spring.io/why-spring

Kuviot