Tampereen ammattikorkeakoulu

Ohjelmiston päivittäminen nykytekniikoille

React ja Spring Boot

Taavi Tuomela

OPINNÄYTETYÖ
Toukokuu 2020

Tietojenkäsittelyn tutkinto-ohjelma
Ohjelmistotuotanto

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietojenkäsittelyn tutkinto-ohjelma
Ohjelmistotuotanto

TUOMELA, TAAVI:
Ohjelmiston päivittäminen nykytekniikoille
React ja Spring Boot

Opinnäytetyö 23 sivua, joista liitteitä 0 sivua
Toukokuu 2020

Opinnäytetyössä kartoitettiin, mitä hyötyjä voidaan saavuttaa rakentamalla vanha ohjelmisto uudelleen nykyaikaisin tekniikoin. Työssä uusittiin 2000-luvun alussa PHP:llä toteutetun sovelluksen käyttöliittymä Reactilla ja palvelinohjelmisto Spring Bootilla. Toimeksiantajan toiveena oli saada ohjelmiston toiminnallisuus ja ulkoasu pysymään samanlaisina kuin ennenkin.

Alkuperäinen sovellus tuntui nykystandardein tarkasteltuna kankealta ja ulkoasultaan vanhanaikaiselta. Opinnäytetyössä selvitettiin ohjelmistokehityksessä ja -suunnittelussa tapahtuneita tekniikoiden ja filosofioiden muutoksia, joiden avulla sovelluksen käyttökokemusta saatiin parannettua. Tekniikoiden kehittyminen oli huomattavissa varsinkin käyttöliittymän rakenteen muuttuessa vanhanaikaisesta HTML-verkkosivusta nykyaikaiseksi yhden sivun sovellukseksi (single-page application).

Lopputuotoksena saatiin edellistä versiota pirteämpi ja responsiivisempi käyttöliittymä sekä tehokkaammin suuria määriä dataa välittävä palvelinohjelmisto. Opinnäytetyö koettiin onnistuneeksi ja tuotosten pohjalta päätettiin luoda valmis pohja vastaavaa rakennetta käyttävien projektien kehityksen helpottamiseksi. Jatkokehitykseen suunnitellut ominaisuudet voidaan toteuttaa helpommin kuin aikaisemmilla tekniikoilla.

Asiasanat: käyttöliittymä, palvelinohjelmisto, react, spring boot, modernisointi

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Business Information Systems
Software Production

TUOMELA, TAAVI:
Upgrading software to modern technologies
React and Spring Boot

Bachelor's thesis 23 pages, appendices 0 pages
May 2020

The purpose of the thesis was to discover possible benefits of rebuilding old software with modern techniques and technologies. The goal was to renew a software built in the 2000's using PHP. It was decided to build the new frontend using React and the backend using Spring Boot. The wishes of the client were to update the technology stack while keeping the workflow of the software the same.

The original software felt clumsy to use and the overall look was old-fashioned. The thesis looks at the evolution of technologies and philosophy in designing and developing software in the last ten years, which factor in the improvement of the product. The evolution of frontend techniques was especially notable when the structure of the frontend changed from the traditional HTML-pages to single-page application.

The resulting frontend was livelier and more responsive. The new backend proved to handle large data loads more effectively. The thesis was deemed successful and the end results were used to create a base to build similar software on. Features planned for future development proved to be easier to achieve using the new technology stack.

Key words: frontend, backend, react, spring boot, modernisation

SISÄLLYS

ERITYISSANASTO

SQL	Standardoitu tietokantojen käsittelykieli.
Frontend	Ohjelmiston käyttöliittymä.
Backend	Ohjelmiston palvelinkerros, kuljettaa datan tietokannalta käyttöliittymälle.
Full stack	Full stack tarkoittaa sovelluskokonaisuutta, joka koostuu käyttöliittymästä sekä palvelinohjelmistosta.
PHP	Varsinkin 2000-luvulla paljon käytetty ohjelmointikieli.
CRUD	Kuvastaa yleisimpiä tietokantatoimintoja. Tulee sanoista Create, Read, Update ja Delete.
Bean	Spring-kehyksessä käytetty termi toiminnallisesta komponentista.
Maven	Ohjelmistoprojektin hallintaan käytetty työkalu.
Thymeleaf	Usein Springin kanssa käytettävä yksinkertainen verkkokäyttöliittymä.

1 JOHDANTO

Työpaikallani toteutettiin projekti, jossa päivitettiin vanha PHP:llä toteutettu kulujenseurantasovellus nykyaikaisemmille teknologioille. Uudessa versiossa käytettäviksi teknologioiksi valikoituivat React käyttöliittymään ja Spring palvelinpuolelle. Vanha toteutus oli tehty 2000-luvulla ja se oli teetetty ulkoisella toimijalla, eikä siitä ei ollut olemassa varsinaista dokumentaatiota. Sovelluksen haluttiin näyttävän mahdollisimman pitkälti samalta kuin ennenkin päivityksestä huolimatta, joten projektin aikana keskityttiin paljon suorituskyvyn optimointiin.

Sovelluksen varsinainen logiikka on toteutettu Microsoft SQL -palvelimella toimivaan tietokantaan, jota omakin toteutuksemme käyttää. Sovelluksen PHP:llä toteutettu käyttöliittymä ja palvelinohjelmisto haluttiin päivittää, koska firman sisällä ei ollut enää henkilöstöä, joka osaisi ilman koulutusta jatkaa vanhan ylläpitämistä. Samalla sovelluksesta haluttiin modernimpi ja sen jatkokehitystä haluttiin helpottaa. Varsinkin React mahdollistaa paljon uusia ominaisuuksia käyttöliittymään.

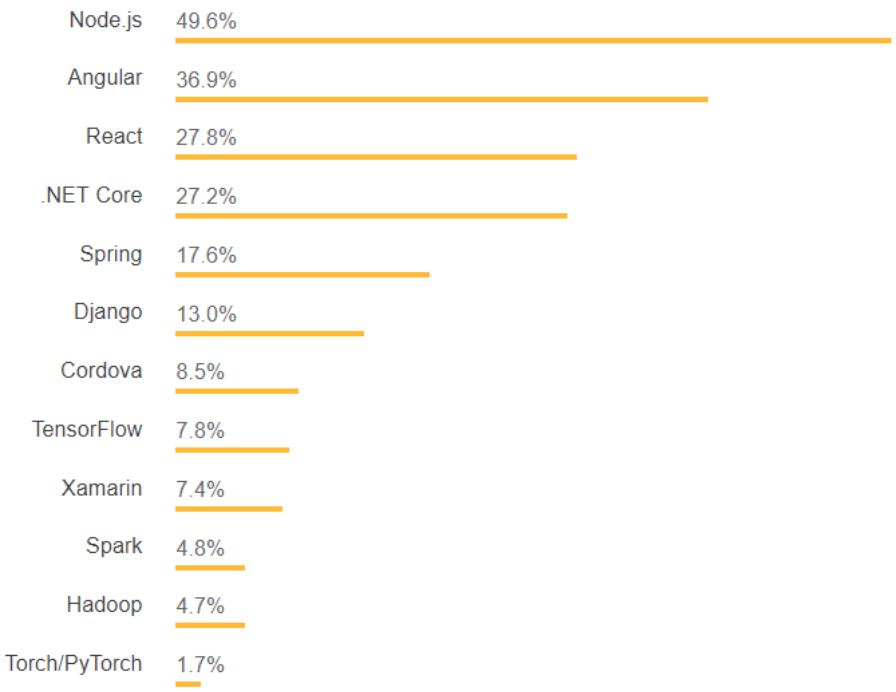
Pääasiallisesti toteutin projektissa palvelinpuolta, mutta avustin käyttöliittymän logiikan kanssa koko projektin ajan. Myöhemmin olen jatkanut sovelluksen jatkokehitysprojektissa, jossa on työstetty lisää sekä front- että backendiä. Tehtävänäni oli myös toimia tavallaan asiakasrajapinnassa, koska tietokanta oli toteutettu firman eri osaston toimesta, joka toimi myös projektin tilaajana ja luonnollisesti palvelinpuolemme on suuresti tekemisissä tietokannan kanssa.

Opinnäytetyö tulee selvittämään, miten ohjelmistokehitys on muuttunut viimeisen vajaan kymmenen vuoden aikana. Opinnäytetyössä myös tarkastellaan, kuinka toiminnallisuus muuttuu suuntaan tai toiseen päivityksen myötä. Kerron myös, mitä valituilla kirjastoilla on tehty ja mitä kolmannen osapuolen komponentteja on käytetty tuomaan ohjelmaan toiminnallisuuksia. Varsinainen tavoite on saada kuvaa siitä, onko vanhojen sovellusten uudelleenrakentaminen uusilla teknologioilla kannattavaa suhteessa vanhojen teknologioiden ylläpitämiseen. Tarkoituksena on rakentaa nykyaikaisemmalta tuntuva käyttöliittymä sekä rakenne vanhalle, edelleen tarpeelliselle sovellukselle.

2 OPINNÄYTETYÖSSÄ KÄYTETTYJEN VANHOJEN TEKNOLOGIOIDEN ESITTELY SEKÄ UUSIEN TEKNOLOGIOIDEN VALINTA

2.1 Käyttöliittymän teknologiat

Käyttöliittymässä käytettävien teknologioiden valitsemiseen vaikuttavia tekijöitä ovat muun muassa eri teknologioiden suosio, firman sisäinen osaaminen sekä työkalujen tuki niin virallisilta julkaisijoilta kuin yhteisöiltäkin. Suosittu ohjelmointifoorumi Stack Overflow kysyi vuonna 2018 käyttäjiltään, mitä kirjastoja he käyttävät eniten töissään. Kuviosta 1 voidaan nähdä Reactin olevan toiseksi käytetyin käyttöliittymäkirjasto Angularin jälkeen vuonna 2018, ainakin Stack Overflow´n käyttäjien keskuudessa. (Stack Overflow 2018.)



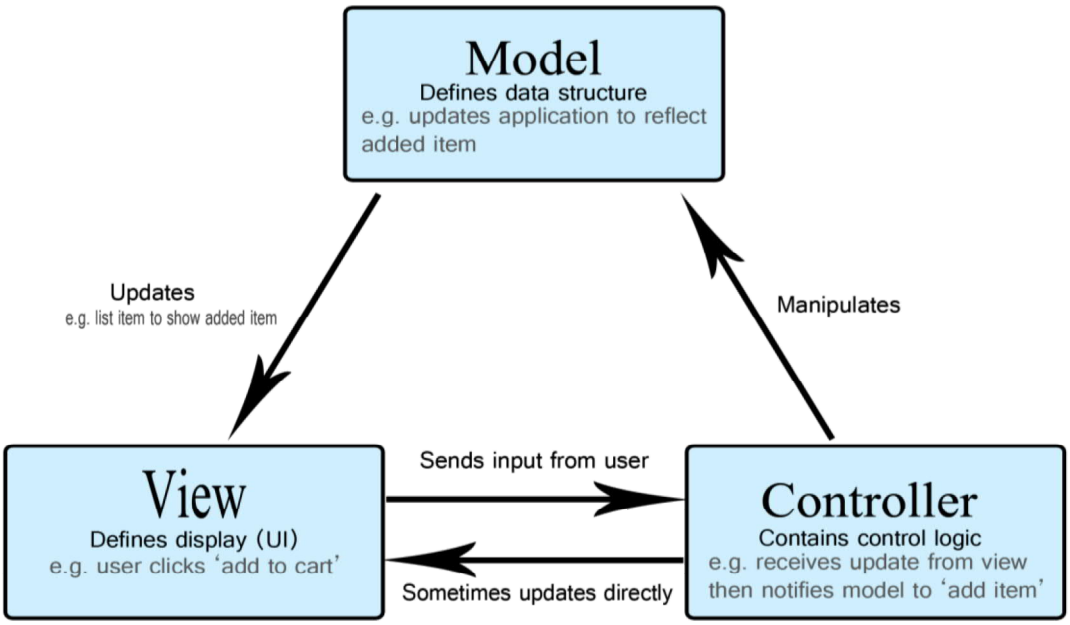
KUVIO 1. Suosituimmat ohjelmistokehykset ja -kirjastot vuonna 2018 Stack Overflown kyselyn mukaan. (Stack Overflow 2018)

Kyselyssä käytetyimmäksi teknologiaksi todettu Node.js on JavaScript-variaatio, jota käytetään lähinnä palvelinohjelmistojen toteuttamiseen. Tässä projektissa palvelinohjelmisto toteutettiin Springillä.

2.1.1 Vanha toteutus

Vanhassa toteutuksessa käyttöliittymään oli käytetty PHP-ohjelmointikieltä CodeIgniter-kehyksen kera. PHP oli varsinkin 2000-luvulla suosittu ohjelmointikieli, koska sillä voidaan toteuttaa sekä käyttöliittymä että tietokantayhteyksiä hallitseva palvelinohjelmisto. CodeIgniter on kehys, joka helpottaa fullstack-sovellusten kehittämistä. CodeIgniter tarjoaa käyttäjälleen valmiiksi useita kirjastoja yleisimpien tehtävien hoitamiseksi. Tämä nopeuttaa ohjelmiston kehitystä sekä auttaa koodin pitämistä siistinä ja selkeänä. CodeIgniter myös luo käyttäjälle valmiiksi ohjelmistoon loogisen rakenteen, joka helpottaa eri palasien toiminnan ymmärtämistä ja löytämistä jälkikäteen. (CodeIgniter 2019.)

CodeIgniter käyttää sisäisessä rakenteessaan niin sanottua MVC-rakennetta, kuvattuna kuvassa 1. MVC tulee sanoista model view controller ja tarkoittaa käytännössä sitä, että ohjelmiston rakenne jaotellaan kolmeen eri komponenttiin: malliin, näkymään ja ohjaimeen. Malli kuvaa datakerrosta sekä palvelinohjelmistoa. Se siis hallitsee ohjelmiston käyttämää dataa. Mallin tulisi yleisesti olla riippumaton näkymästä ja ohjaimesta. Käytännössä siis saman mallin päälle tulisi voida rakentaa esimerkiksi täysin erilainen käyttöliittymä. Näkymä kuvaa käyttöliittymää, joka tarjoilee mallin sisältämän datan käyttäjälle esitettäväksi tai muokattavaksi. Ohjain taas tarjoilee mallin datan näkymälle ja välittää näkymässä tehdyt muutokset takaisin mallille. Joissain tapauksissa näkymä ja ohjain voidaan yhdistää samaksi objektiksi. (Dalling 2019.)



KUVA 1. MVC-arkkitehtuuri mallinnettuna. (Mozilla 2019)

Se, että vanha toteutus noudatti MVC-rakennetta, auttoi huomattavasti uuden version rakenteen suunnittelua, sillä myös uudessa toteutuksessa käytetään tätä rakennetta. CodeIgniter kuitenkin yhdistelee esimerkiksi ohjaimet samaan suureen tiedostoon, kun taas Spring Bootin kanssa ohjaimet jaetaan yleensä omiin tiedostoihinsa sen mukaan, missä ohjainta käytetään. Spring Bootia kuvataan tarkemmin luvussa 3.2.1.

Sovelluksen varsinainen logiikka sijaitsee tietokannassa, joka on toteutettu Microsoftin SQL Server -alustalle sen natiivia MSSQL-kieltä käyttäen. Käyttöliittymästä käytännössä käynnistettiin tallennettuja funktioita tietokannassa, jotka toteuttivat muutokset tietokantaan.

2.1.2 Käytettävien teknologioiden valinta

Projektin käyttöliittymässä käytettäväksi kirjastoksi valittiin React. Valintaan vaikutti pääosin se, että Reactia käytettiin yrityksessä yleisesti, ja projektitiimi oli käyttänyt sitä ennenkin. Tutun kirjaston käyttö oli tärkeää senkin vuoksi, että projekti toteutettiin pääosin opiskelijoiden voimin.

Reactin koettiin myös tukevan helposti sovellukselta vaadittuja ominaisuuksia. Myös jatkokehitykseen suunniteltuja ominaisuuksia vastaavia sovellutuksia oli jo tehty yrityksen sisällä.

2.2 Palvelinohjelmiston teknologiat

Palvelinohjelmistoon käytettävien teknologioiden valinnassa käytettiin samoja perusteita kuin käyttöliittymän kanssa. Yrityksessä on käytetty yleisesti seuraavia kehyksiä: Spring, Spring Boot sekä Node.js. Projektitiimissä oli eniten kokemusta Spring Bootista, joten se otettiin käyttöön projektissa. Spring Boot on myös erittäin hyvin tuettu ja laajasti käytössä oleva, joten sen tiedettiin tukevan Microsoftin SQL Server -pohjaisia tietokantoja. Tämä oli valinnassa tärkeää, sillä sovelluskerros rakennetaan jo olemassa olevan tietokannan päälle.

2.2.1 Vanha palvelinohjelmisto

Vanhassa versiossa CodeIgniter hallitsi myös palvelinpuolen. Toteutus sinänsä oli siisti, joskin vanhentuneen näköinen ja tuntuinen. Tähän oli syynä varsinkin ohjelmiston kansiorakenne, jossa eri osiot oli jaettu omiin kansioihinsa, mutta varsinaiset tiedostot olivat hyvinkin massiivisia siihen nähden, mihin nykyään pyritään. Usean eri näkymän ohjaimet oli kasattu samoihin tiedostoihin, mikä vaikeutti haluttujen toimintojen etsimistä koodista. Koska eri komponentteja oli useita samoissa tiedostoissa, olivat tiedostojen nimetkin ajoittain harhaanjohtavia. Eri näkymistä ja ohjelmiston osista oli myös käytetty ohjelmistoa kehitettäessä eri nimiä, kuin mitä nykyään käytettäisiin hyvien tapojen mukaisesti. Esimerkiksi sovelluksemme Discover-välilehden sisältö oli nimellä Treeview, kun taas projektitiimissä päätettiin kutsua komponenttia tableksi, sillä sen rakentamiseen käytettiin HTML tablea. Hämmentävää oli myös nähdä tietokantakutsuja sekä käyttöliittymän komponentteja saman tiedoston sisällä.

2.2.2 Käytettävien teknologioiden valinta

Projektin palvelinohjelmiston kehitykseen valittiin kieleksi Java ja kehykseksi Spring Boot. Valintaan vaikuttivat pitkälti samat kriteerit kuin käyttöliittymänkin osalta. Spring Bootin tiedettiin myös tukevan MSSQL-kutsuja, joten tietokantayhteyden luomisen tiedettiin onnistuvan melko kivuttomasti.

Ohjelman rakenteen osattiin myös odottaa vaativan enemmän kuin yleinen CRUD-rakenne antaa myöden. Springin tiedettiin mahdollistavan näiden toiminnallisuuksien toteuttaminen vaivattomasti. Sovellus muun muassa suorittaa tietokannassa laskutoimituksia sekä yhdistelee tietokannan tauluja dynaamisiksi tauluiksi käyttäen tietokantaan talletettuja stored procedureja.

2.2.3 SQL Server -esittely

Sovelluksen tietokanta on rakennettu Microsoftin SQL Server -palvelimelle Microsoftin omaa SQL-variaatiota, MSSQL:ää, käyttäen. Microsoft SQL oli itselleni täysin uusi tuttavuus, vaikka SQL-kantojen parissa olenkin työskennellyt. Käytössämme on SQL Server 2016 -versio.

Microsoft on julkaissut SQL Server -versioita vuodesta 2002 lähtien. Viimeisin versio on 2019, joka julkaistiin marraskuussa 2019. (Microsoft 2019). Nykyisin Microsoft pyrkii korvaamaan MSSQL-ratkaisuja Azure SQL -palvelullaan. Projektissamme käyttämämme tietokanta koettiin kuitenkin liian laajaksi siirtää tällä erää uudelle alustalle.

Suurin osa toiminnallisuudesta luotiin käyttäen tietokannan stored procedureja. Stored proceduret muistuttavat toiminnallisuudeltaan funktioita mutta suurin erottava tekijä on se, että tietokannassa funktio vaatii return-arvon, eli sen on pakko palauttaa dataa kutsujalleen. Stored proceduren return-arvo taas on optionaalinen. Tätä toiminnallista eroavaisuutta vaaditaan ohjelmaltamme, sillä proceduren kautta halutaan suorittaa laskelmia kannassa, jotka eivät välttämättä palauta suoraan tuloksia käyttäjälle, vaan muokkaa suoraan tauluja.

3 FULLSTACK -SOVELLUKSEN SUUNNITTELU

3.1 Käyttöliittymän suunnittelu

Käyttöliittymää suunnitellessa otettiin ensinnä huomioon asiakkaan toive siitä, että perusilme tulisi olemaan mahdollisimman pitkälle samanlainen kuin vanhassa versiossa. Tässä vaiheessa esiteltiin ehdolle kaksi erilaista tapaa toteuttaa sovelluksen Discover-välilehti. Ensimmäinen idea oli luoda siitä moderni listaa muistuttava ratkaisu mutta tämä teilattiin ja päädyttiin käyttämään saman tyylisiä table-elementtejä kuin vanhassakin versiossa.

Vastaavia tilanteita, joissa asiakas haluaa pitää ohjelmiston saman näköisenä, vaikka tarkoitus on päivittää ilmettä, ilmenee ohjelmistokehityksessä usein. Asiakkaiden loppukäyttäjät ovat kuitenkin saattaneet käyttää ohjelmistoa jo vuosia, joten uuteen ilmeeseen totutteluun saattaisi kulua työtunteja. Radikaalimpien muutosten vuoksi saatettaisiin jopa joutua kouluttamaan työntekijöitä uudelleen uuden ulkoasun tai toiminnallisuuden vuoksi.

Yleiseen ilmeeseen tehtiin kuitenkin pieniä muutoksia, jotka saivat sovelluksen tuntumaan nykyaikaisemmalta. Sivun yläosassa joka välilehdellä näkyvät linkit käyttäjän asetuksiin, apua-osioon sekä tukilinkkeihin muutettiin tekstimuotoisista ikoneihin ja sivuston taustalla ollut matalaresoluutioinen kuva muutettiin abstraktimmaksi, jotta sen toistuvuus sivun venyessä saatiin siistimmän näköiseksi.

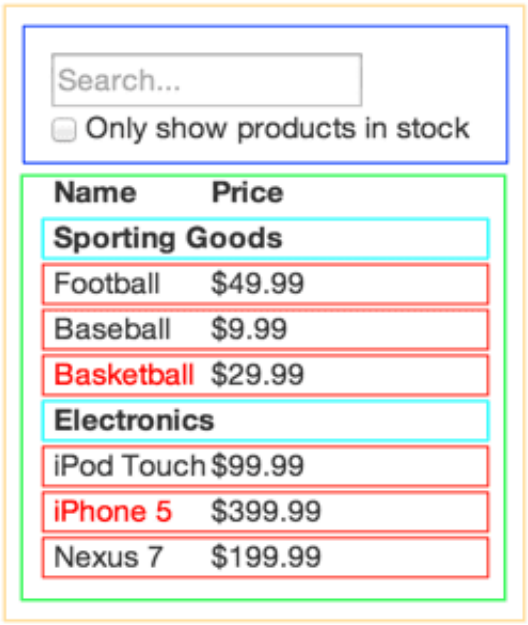
3.1.1 Reactin esittely

React on Facebookin kehittämä Javascript-pohjainen käyttöliittymäkirjasto. React-sovellukset koostuvat niin kutsutuista komponenteista, jotka hallitsevat omaa tilaansa. React-pohjainen käyttöliittymä rakentuu useimmiten useista komponenteista. Reactia tulisi yleisesti ottaen käyttää siten, että käyttöliittymää suunnitellessa se jaetaan jo komponentteihin. Hyvä sääntö komponentteihin jaotellessa on se, että jokaisen komponentin tulisi ideaalisesti toteuttaa vain yhtä tehtävää. Mikäli komponentin huomataan tekevän useampia tehtäviä, tulee tutkia, voitaisiinko se sitten jakaa pienempiin osiin. (React 2019.)

Reactia tehdessä käytetään varsinaisten käyttöliittymäelementtien luomiseen yleensä JSX-nimistä syntaksia. JSX on tunnistepohjainen JavaScript syntaksi. Ulkomuodoltaan JSX muistuttaa hyvinkin paljon HTML:ää. (Banks & Porcello 2019, 4.)

3.1.2 Käyttöliittymän suunnittelu Reactin avulla

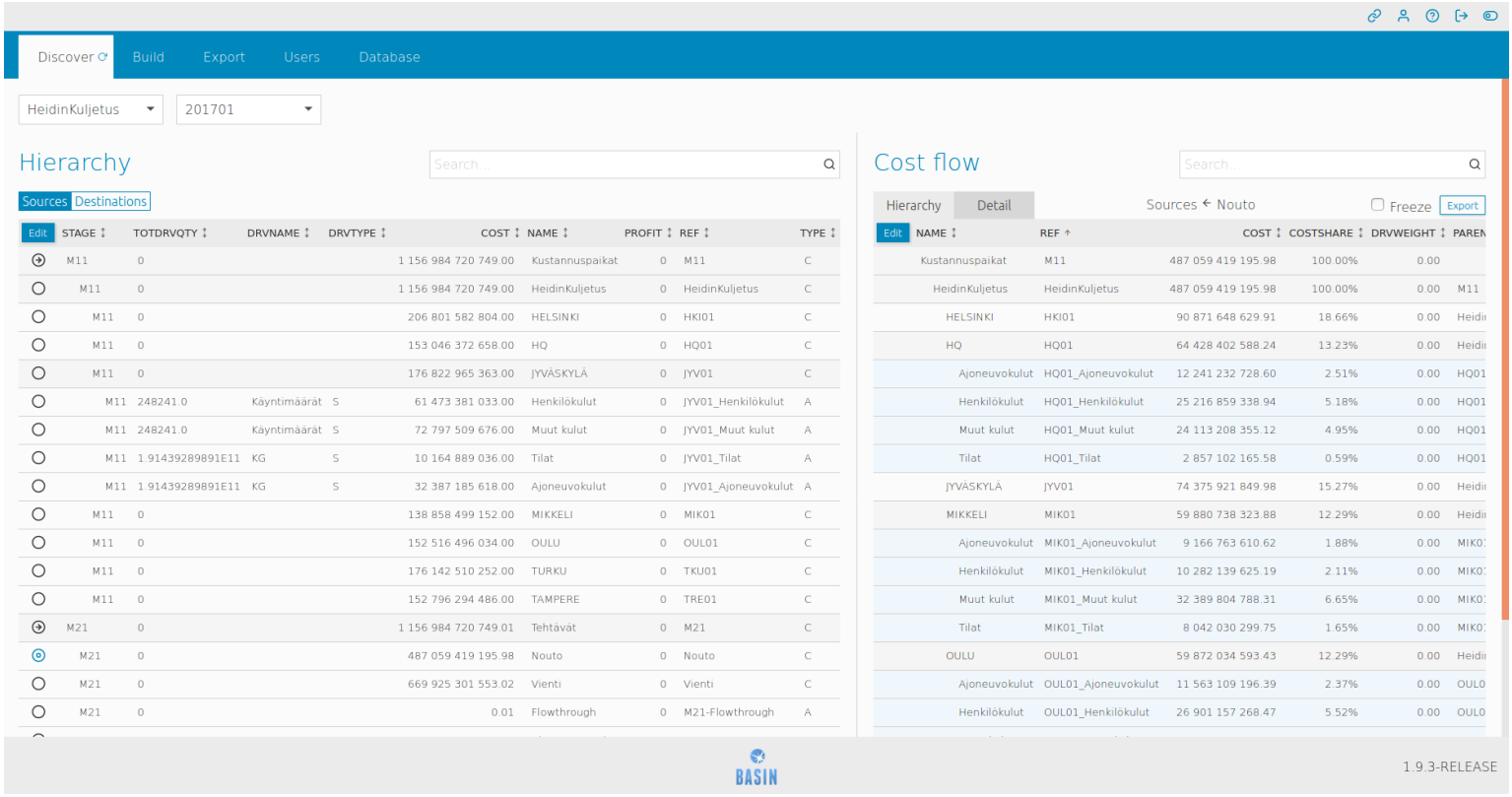
Reactin hyvässä tavassa toteuttaa sovelluksia suositellaan suunnitelman osiot pilkottavaksi omiksi osikseen. Kuvassa 2 on esimerkkinä kaupan varaston tarkistamiseen tarkoitettu komponentti, joka on pilkottu pienempiin osiin. Kun komponentti jaetaan pienempiin osiin, eri osia voidaan helpommin käyttää uudelleen eri puolilla sovellusta. Samalla sovelluksen jatkokehitys helpottuu, sillä pienempiin komponentteihin voidaan tehdä helposti muutoksia ilman, että ne vaikuttavat muokattavien komponenttien ympärillä olevaan rakenteeseen.



KUVA 2. Esimerkki komponentin pilkkomisesta pienempiin, helpommin hallittaviin osiin. (React, 2019.)

Sovelluksen Discover-välilehti, esitetty kuvassa 3, on helppoa jaotella eri komponentteihin. Sivulla on ylä- sekä alalaidassa koko sovelluksen yhteiset komponentit. Ylälaidasta löytyy työkalupalkiksi kutsuttu komponentti, josta löytyvät hyödylliset linkit, käyttäjän muokkaaminen, sovelluksen ohjeet, uloskirjautuminen sekä teeman vaihtamiseen käytettävä kytkin.

Sivun varsinainen sisältö koostuu näytettävän datan valintaan käytettävistä alasvetovalikoista sekä kahdesta taulusta. Taulut on vielä jaoteltu muun muassa otsikkoon, hakupalkkiin, sekä erinäisiin näytettävän datan säätämiseen käytettäviin valintoihin, kuten radionappeihin. Jotkin komponentit on vaikeampi huomata. Taulujen rivit ovat itse luomiamme komponentteja, jotta niihin on saatu upotettua valintapainikkeet sekä tarvittavat muokkaustoiminnot.



KUVA 3. Discover-välilehti. (Kuvakaappaus projektin sovelluksesta.)

3.2 Palvelinohjelmiston suunnittelu

Kun palvelinohjelmistossa käytettävä kieli ja kehys on valittu, tulee valita tarpeellisia kirjastoja helpottamaan esimerkiksi tietokantayhteyden sekä kirjautumisen hallintaa. Kaikkia tarpeellisia kirjastoja on kuitenkin hankalaa arvioida heti suunnitteluvaiheessa, joten useimmiten ohjelmistoa kehitettäessä kirjastoja tuodaan lisää. Tässä projektissa valittiin käytettäväksi Hibernate tietokantayhteyden hallintaan. Hibernate tekee mallien tallettamisen tietokantaan helpoksi. Autentikaatiota päätettiin hallita JWT-tekniikalla. Lyhenne JWT tulee sanoista JSON Web Token. Käytännössä kun käyttöliittymästä kirjaudutaan palveluun,

käyttöliittymä lähettää kirjautumistiedot palvelinohjelmistolle, joka vertaa niitä kannasta löytyviin arvoihin. Mikäli tunnukset ovat validit, lähetetään vastauksessa käyttöliittymälle takaisin niin sanottu tokeni, joka on käytännössä kryptattu merkkijono, jonka luomiseen käytetään salassa pidettäviä merkkijonoja.

3.2.1 Spring Bootin esittely

Spring on hyvin suosittu sovelluskehys Java-pohjaisten palvelinohjelmistojen luomiseen. Spring-ohjelmistot koostuvat beaneista, jotka ovat käytännössä komponentteja, joista ohjelmisto itsessään on riippuvainen. Beaneja voidaan luoda XML:ää, annotaatioita tai puhdasta Javaa käyttäen. Spring-sovelluksen kehittämisen aloitus koettiin hankalaksi, joten tätä varten luotiin Spring Boot. Spring Boot auttaa hallitsemaan ohjelmiston eri komponentteja käyttäjän puolesta luomalla konfiguraatiotiedoston käyttäjän valitsemien kirjastojen pohjalta. (Reddy 2017.)

Spring Boot -sovelluksen kehitys voidaan aloittaa käyttäen verkossa toimivaa Spring Boot Initializeria. Käytännössä verkkosivulla saa valita haluamansa version Springistä sekä Javasta, halutun paketointimuodon, halutaanko projektissa käyttää Mavenia vai Gradlea sekä halutut riippuvuudet, eli ulkoiset kirjastot. Spring Boot tukee nykyään myös Kotlin- sekä Groovy-kieliä, joskin nämä ovat molemmat Java-pohjaisia. Käytetyimmät koodieditorit tukevat Spring Boot Initializeria natiivisti ja tarjoavat projektia aloitettaessa Initializerin ominaisuudet ilman erillisen verkkosivun käyttöä.

3.2.2 Ohjelmiston suunnittelu Spring Boot -kehyksen kanssa

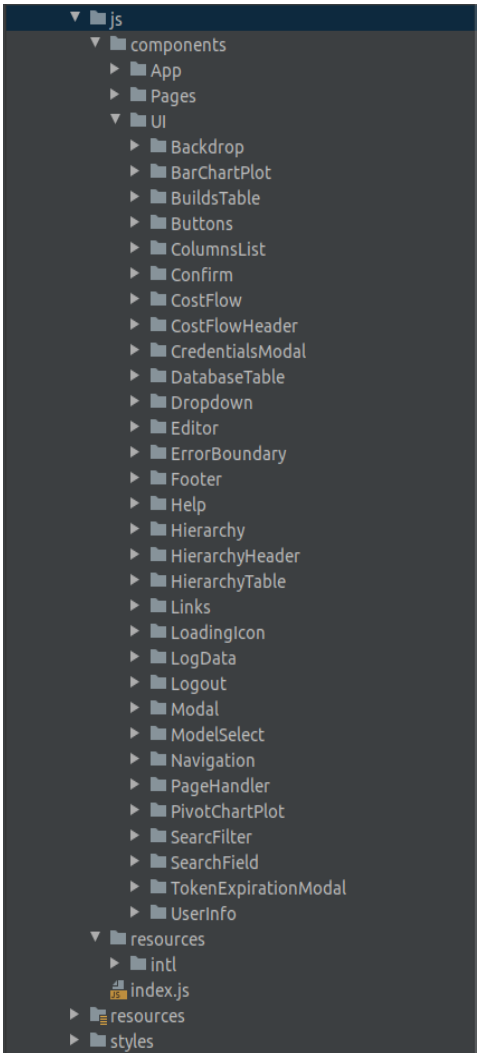
Spring Bootilla toteutettavaa ohjelmistoa suunnitellessa ensimmäinen harkinnan aihe ovat valmiiksi valittavat ulkoiset kirjastot. Projektissa tuotetun sovelluksen tärkein toiminnallisuus on tietokannasta datan noutaminen, joten luonnollisesti mukaan otetaan Hibernate sekä Spring data -paketit. Projektissa päätettiin käyttää myös Lombok-lisäosaa, joka nopeuttaa koodin kirjoittamista hoitamalla luokkien constructor-, set- sekä get-funktiot käyttäjän puolesta.

Ohjelma vaatii myös käyttäjänhallintaa, joten käyttöön otettiin myös Spring Security -paketti. Spring Security tarjoaa valmiiksi käyttäjänhallintaan vaaditut kirjastot palvelinohjelmiston osalta. Se ohjaa ohjelmaa käynnistettäessä käyttäjän suoraan Thymeleafilla toteutetulle kirjautumissivulle, jolta löytyy vakiona valmiiksi myös salasanan palautuslinkin. Tämä sivu tosin toimii vain palvelinohjelmiston kirjautumiseen, joten mikäli käyttöliittymälle tarvitaan kirjautuminen, tulee paketin valmis kirjautumissivu ottaa pois käytöstä ja luoda omat konfiguraatiot sille, mihin käyttäjä ohjataan sovellusta käynnistettäessä.

4 SOVELLUKSEN KEHITYS

4.1 Käyttöliittymän kehitys

Käyttöliittymän varsinainen kehitys käynnistettiin luomalla ensin sovelluksen kansiorakenne Reactin hyviä tapoja mukaillen. Kuvasta 4 voidaan nähdä, miten eri komponentteja on jaoteltu kansoihin, jotta niitä on helpompi selata tarvittaessa.



KUVA 4. Ohjelmiston käyttöliittymän kansiorakenne. (Kuvakaappaus)

Kehitys aloitettiin Discover-välilehdestä, sillä sen toiminnallisuuden koettiin olevan hankalin toteuttaa. Monia sivun komponentteja, kuten geneeristä alasvetovalikkoa sekä footeria, tiedettiin myös tarvittavan muilla sivuilla. Sivun pääasiallisen toiminnallisuuden, eli mallien dataa esittävien taulujen, aikaan saaminen

todettiinkin monimutkaiseksi. Taulut kuvaavat käytännössä hierarkioita, joiden eri tasot piti saada indikoitua käyttäjälle helposti. Mallit sisältävät myös sisäisiä tasoja, joilla on eri määriä alatasoja. Mallien eroavaisuuksien vuoksi kaikkien komponenttien tulee olla dynaamisia, eli komponenttien otsikoista lähtien kaiken näytettävän datan tuli tulla suoraan kannasta. Tämä tekee kehittämisestä hankalaa, sillä olisi paljon helpompi tyylitellä komponentteja, joiden sisällön tietää etukäteen. Onneksemme React on suunniteltu dynaamisten sivujen luomiseen, joten emme joutuneet tämän takia painimaan CSS:n parissa muuten kuin tyylejä luodessamme.

4.1.1 React komponenttien luominen

Käyttöliittymän komponentit koottiin käyttäen JSX-elementtejä. Se on yleisesti suositeltu tapa luoda varsinaiset käyttöliittymän elementit Reactia käytettäessä. JSX muistuttaa ulkoasultaan HTML-syntaksia, joka on perinteinen tapa tehdä verkkosivuja. Kuvassa 5 esimerkki JSX-elementistä.

```
<Modal show={showNoChanges} modalHeight="175px">
  <h1 style={{ textAlign: 'center' }}>
    No changes made into the table.
    <br />
    Check inserted fields!
  </h1>
</Modal>
```

KUVA 5 Esimerkki JSX-elementistä. (Kuvakaappaus)

Komponentteja luodessa saattaa helposti alkaa toteuttamaan turhan paljon toiminnallisuutta yhden komponentin sisään. Mikäli Reactilla ei ole vielä toteuttanut kokonaisia projekteja, voi olla hankalaa hahmottaa, mitä toiminnallisuutta kannattaa toteuttaa erillisissä funktioissa, jotta komponentit itsessään pysyisivät mahdollisimman abstrakteina ja uudelleen käytettävinä. Kun projektiin on tuotettu ensimmäiset toimivat komponentit, on ne hyvä tarkistuttaa projektitiimin sisällä tai mahdollisuuksien mukaan projektitiimin ulkopuolisella taholla. Komponenttien uusiokäytön helppous tulee auttamaan projektin edetessä sekä mahdollisissa muissa projekteissa, joissa vastaavaa toiminnallisuutta tarvitaan.

4.1.2 Palvelinyhteyden toteuttaminen

Sovellukselle luotiin oma SercureFetch-luokka käsittelemään hakukutsut käyttöliittymältä palvelinohjelmistolle. Käytännössä luokka käytti hyväkseen JavaScriptin omaa Fetch-funktiota, mutta projektin sovelluksessa kutsuihin piti liittää käyttäjänhallintaan liittyvää dataa, jota vakio funktio ei tukenut.

Muokattuun luokkaan saatiin myös luotua omat lokikäytännöt, joilla voitiin määrittää, kuinka paljon ohjelmisto kirjoittaa välivaiheita lokeihin. Lokien määrä laitettiin myös säätymään automaattisesti niin, että mikäli palvelinohjelmistossa tapahtui virhetilanne, saatiin siitä täsmällinen tieto myös käyttöliittymään. Tämä paransi ongelmien korjaamisen tehokkuutta, koska vioista saatiin nopeasti tarkkaa tietoa riippumatta siitä, missä osassa ohjelmistoa vika ilmeni.

4.2 Palvelinohjelmiston toteuttaminen

Koska tietokanta oli jo olemassa, palvelinohjelmiston kehitys alkoi tutkimalla, millaisia malleja tietokannasta saadaan tuotua. Meidän tapauksessamme suuri osa malleista luotiin erinäisissä tietokannan prosesseissa, joten mallien rakennetta ei voitu suoraan katsoa kannan tauluista, vaan prosesseja piti ajaa manuaalisesti ja tutkia, mitä ne palauttavat. Osa malleista oli kuitenkin perinteisempiä, eli tietokannan taulu vastaa suoraan Java-ohjelmiston luokkaa. Parhaana esimerkkinä tästä käyttäjät. Käyttäjillä on aina tallessa muun muassa sähköpostiosoite, etu- sekä sukunimi, salasana sekä erillinen käyttäjätunnus. Näistä saadaan suoraan muuttujat Java-oliolle.

4.2.1 RESTful-rajapinnan toteutus

Lähtökohtaisesti ohjelmiston tietokantarajapinta toteuttaa CRUD-toiminnallisuudet ja täyttää RESTful-periaatteen kriteerit. Ohjelmisto siis käyttää rajapinnan ja käyttöliittymän välisessä viestinnässä JSON-formaattia ja HTTP-pyyntöjen ohessa lähetetään tieto, mitä välitettävällä datalla halutaan tehdä.

Restin periaatteena on, että kutsut ovat niin sanotusti tilattomia (stateless). Kutsut siis ohjautuvat rajapinnan läpi ilman, että niitä talletetaan mihinkään. Tämän vuoksi on tärkeää, että kutsuissa itsessään on kaikki data, mitä tarvitaan kutsun tavoitteiden saavuttamiseksi. Jos esimerkiksi halutaan muokata objektia tietokannassa, tulee kutsu rajapinnalle käyttäen PUT-metodia ja se sisältää korvaavat tiedot kaikkiin tietokannan kenttiin, jotta palvelinohjelmisto osaa hakea ja korvata objektin datan tietokannasta.

4.2.2 Tietokantayhteyden luominen

Tietokantayhteyttä hallitaan Hibernate-kirjastoa käyttäen. Hibernate lukee projektin Application.yml-tiedostoa, josta löytyy konfiguraatio tietokantayhteydelle. Konfiguraatiotiedosto esiteltynä kuvassa 6. Kuvasta voidaan nähdä, että samasta tiedostosta löytyy konfiguraatiot sekä tietokannalle, että ohjelmiston käyttämälle sähköpostipalvelimelle. Sähköpostipalvelinta käytetään käyttäjän salasanan palauttamiseen.

```yaml
spring:
  profiles:
    active: prod
  mail:
    host: smtp.office365.com
    port: 25
    username: login@email.fi
    password: 3JA9ZEAuVGSZ
    properties:
      mail:
        smtp:
          auth: true
          starttls:
            enable: true
            required: false
  datasource:
    driverClassName: com.microsoft.sqlserver.jdbc.SQLServerDriver
    url: jdbc:sqlserver://testikanta;databaseName=Testing
    username: testaaja
    password: c6GQUYd3ARz7aAqu
    platform: mssql
    hikari:
      maximum-pool-size: 30
```

KUVA 6. Application.yml-tiedosto. Kriittiset tiedot muokattu. (Kuvakaappaus)

Konfiguroitaessa tietokantayhteydelle (datasource) määritetään käytettävä ajuri, tietokannan url-osoite, sisäänkirjautumistiedot, alusta sekä mahdollisia lisämäärityksiä. Jotta tietokanta tai palvelinohjelmisto ei menisi jumiin liian monesta samanaikaisesta yhteydestä, on konfiguraatiossa rajoitettu samanaikaisten yhteyksien määrää.

5 POHDINTA

Opinnäytetyöhön liittyvä projekti käynnistettiin, jotta vanha ohjelmisto saataisiin uudelleen kehitettäväksi uusien tekijöiden voimin ja käyttäjien käyttökokemusta parannettua. Opinnäytetyön tarkoituksena oli uudistaa vanhan ohjelmiston rakenne nykytekniikoin, jotta sen käyttömukavuus kasvaisi ja jatkokehitys helpottuisi. Tavoitteena oli kartoittaa vastaavien projektien tarvetta, hyötyjä sekä haittoja.

Lopputuotoksen katsottiin olevan onnistunut, sillä ohjelmiston uusi käyttöliittymä on responsiivisempi ja ohjelmisto kokonaisuutena käsittelee suuria määriä dataa entistä tehokkaammin. Ohjelmistossa käyttöön otetut teknologiat ovat yleisesti laajemmassa käytössä kuin entiset ja niitä opetetaan monissa oppilaitoksissa ympäri maailman. Tämä helpottaa uusien työntekijöiden sijoittamista ohjelmiston jatkokehitysprojektiin, sillä heitä ei tarvitse kouluttaa käyttämään jo vanhentuneita tekniikoita.

Jatkokehitykseen suunniteltiin jo projektin loppuvaiheessa toimintoja, jotka helpottaisivat käyttäjää löytämään yleisimmin ohjelmistolta haluttua dataa. Ohjelmistoon lähdetäänkin jatkossa luomaan uutta etusivua, jolta nähdään nopeasti eri graafeja, jotka antavat ikään kuin yleiskatsauksen Discover-välilehden datasta. Tällöin käyttäjän ei tarvitse seuloa ajoittain hyvin monimutkaistenkin taulujen sisältä tärkeitä datapisteitä.

Jälkikäteen tutkittiin, mitkä olivat haasteellisimmat vaiheet projektin aikana ja totesimme, että alkuun kehitys oli huomattavan hidasta. Tämä johtui paljolti projektiryhmästä, sillä kaikki ryhmän jäsenet olivat junioreita. Kuitenkin todettiin, että vastaava rakenne on yleinen yrityksen sisällä, joten tuotoksen pohjalta haluttiin kehittää pohja, joka nopeuttaisi projektin käynnistämistä, kun käytössä on samat kehitystyökalut.

LÄHTEET

Banks, A. Porcello, E. 2019. Learning React, 2nd Edition. Sebastopol: O'Reilly Media, Inc.

CodeIgniter 2019.
https://codeigniter.com/user_guide/overview/at_a_glance.html

Dalling, Tom 2019.
https://www.tomdalling.com/blog/software-design/model-view-controller-explained/

Microsoft, 2019.
https://support.microsoft.com/fi-fi/help/321185/how-to-determine-the-version-edition-and-update-level-of-sql-server-an

Mozilla, 2019.
https://developer.mozilla.org/en-US/docs/Glossary/MVC

React, 2019.
https://reactjs.org/

React, 2019.
https://reactjs.org/docs/thinking-in-react.html

Reddy, S. 2017. Beginning Spring Boot 2: Applications and Microservices with the Spring Framework. New York: Apress

Stack Overflow 2018.
https://insights.stackoverflow.com/survey/2018/