SAVONIA

OPINNÄYTETYÖ - **AMMATTIKORKEAKOULUTUTKINTO**
TEKNIIKAN JA LIIKENTEEN ALA

LUENNOITSIJAPORTAALI-SOVELLUKSEN TOTEUTUS

TEKIJÄ: Oskari Harjunen

SAVONIA-AMMATTIKORKEAKOULU

OPINNÄYTETYÖ
Tiivistelmä

Koulutusala Tekniikan ja liikenteen ala	
Koulutusohjelma/Tutkinto-ohjelma Sähkötekniikan koulutusohjelma	
Työn tekijä(t) Oskari Harjunen	
Työn nimi Luennoitsijaportaali-sovelluksen toteutus	
Päiväys 30.05.2019	Sivumäärä/Liitteet 33
Ohjaaja(t) Lehtori Jussi Koistinen, lehtori Keijo Kuosmanen	
Toimeksiantaja/Yhteistyökumppani(t) Data Prisma Oy	

Tiivistelmä

Opinnäytetyö toteutettiin yhteistyössä Data Prisma Oy:n kanssa. Opinnäytetyön tavoitteina olivat uuden käyttöliittymäkoodin tekniikan valinta sekä tuottaa luennoitsijaportaalin eri ominaisuudet uudella tekniikalla yrityksen tekemään Kongressi-ohjelmistojärjestelmään.

Ensimmäisenä tavoitteena oli suorittaa front-end tekniikoiden valinta. Tämä ratkaistiin Angular:in ja React:in välillä. Kummastakin tekniikasta muodostettiin PoC:it ja vertailtiin näiden tuloksia. Tuloksien vertailun lisäksi tärkeänä osana oli oppimiskynnys, sillä tekniikan mahdollinen käyttö tuli huomioida jatkokehityksessä muun henkilöstön osalta. Valinta kohdistui React:iin, sillä sen oppimiskynnys oli pienempi ja henkilöstöstä löytyi jo kokemusta tekniikan osalta. Tämän jälkeen toteutusta aloitettiin tekemään React-projektin arkkitehtuurin suunnittelulla, josta jatkettiin sivukokonaisuuksien toteutukseen React:lla. Kun React:lla tehdyt sivut olivat valmiita, ne liitettiin vanhan MVC-projektin Razor:lla generoituihin sivuihin.

Lopputuloksena luennoitsijaportaalin tärkeimmät osat saatiin integroitua Kongressiin. Osa ominaisuuksista jäi toteuttamatta, mutta puuttuvat ominaisuudet toteutetaan jatkokehitysvaiheessa.

Avainsanat
React, .NET Framework, ASP.NET MVC, C#

SAVONIA UNIVERSITY OF APPLIED SCIENCES

THESIS
Abstract

Field of Study Technology, Communication and Transport			
Degree Programme Degree Programme in Electrical Engineering			
Author(s) Oskari Harjunen			
Title of Thesis Lecturer portal -application implementation			
Date	30 May 2019	Pages/Appendices	33
Supervisor(s) Mr Jussi Koistinen, Senior Lecturer, Mr Keijo Kuosmanen, Senior Lecturer			
Client Organisation /Partners Data Prisma Oy			

Abstract

The thesis was executed in cooperation with Data Prisma Ltd. The goals of the thesis were the choosing of a new front-end development technique and the implementation of Lecturer Portal features into the company's Kongressi software system with a new technique.

The first goal was the choice of front-end techniques. This was resolved between Angular and React. With both technologies a proof of concept was produced and the results were compared. One of the keys of the comparison was learning speed because it would be taken in production with the rest of the employees. The choice was made in favor of React, because it had a faster learning speed and some of the employees were already familiar with it. After that the implementation was started by designing the architecture of the React project which was continued by developing the pages with React. When the pages were ready, they were attached into MVC-project's Razor-generated pages.

As a result, the features of Lecturer Portal were successfully integrated into Kongressi. Some features were not produced but the missing features are going to be produced in the upcoming development.

Keywords
React, .NET Framework, ASP.NET MVC, C#

ESIPUHE

Haluan kiittää työnantajaani Data Prisma Oy:tä työn tekemisen mahdollistamisesta. Tämän lisäksi haluan kiittää myös työpaikan ohjaajaani Jukka-Pekka Kurkea sekä kollegoitani Jere Martiskaista, Niko Kauppista ja Sami Gråstenia sekä lehtori Jussi Koistista hienosti toteutetusta ohjauksesta.

Kuopiossa 30.5.2019
Oskari Harjunen

SISÄLTÖ

1 JOHDANTO

Opinnäytetyön aiheena oli kehittää luennoitsijoille webpohjainen portaaliohjelma eli luennoitsijaportaali. Luennoitsijaportaali on selainohjelma, jossa luennoitsijat voivat ylläpitää tietojaan ja luentosarjoja, luoda luentoehdotuksia/abstrakteja, luentoja ja luentomateriaaleja sekä käsitellä järjestäjän tiedotteita.

Projektin ensimmäisenä tavoitteena oli tutkia front-endin käytettävä tekniikka. Tämä suoritettiin tekemällä vertailu nykyaikaisten selainkäyttöliittymä-tekniikoiden välillä, joista yrityksen tarpeisiin nähden paras valittiin. Toisena tavoitteena oli kehittää luennoitsijaportaalin osat uudella tekniikalla olemassa olevaan Portaali-projektiin, joka on osa Kongressi-ohjelmistojärjestelmää. Toteutukseen kuuluivat käyttöliittymäohjelmointi ja palvelinohjelmointi.

Kappaleessa 2 kuvataan yritystä ja työhön liittyviä yrityksen projekteja. Tämän jälkeen kappaleessa 3 kerrotaan työssä käytetyistä kehitystyökaluista. Sitten kappaleessa 4 käsitellään eri selainkäyttöliittymätekniikoita ja niiden vertailua. Kappaleessa 5 selostetaan työn toteutusta niin käyttöliittymäkuin palvelinohjelmoinnin osalta. Seuraavassa kappaleessa nähdään työn tuloksia sekä niihin tehtyjen ominaisuuksien kertomista. Viimeisessä kappaleessa referoidaan työn tulokset ja jatkokehitystoimenpiteet.

1.1 Työn toteutus

Projekti toteutettiin Data Prisma Oy:lle. Työn kehitysosuuden teimme yhdessä Jere Martiskaisen kanssa projektin laajuuden takia. Tekniikoiden tutkinta suoritettiin luomalla projektiin pieni osa ohjelmaa molemmilla tekniikoilla, joista parempi valittiin. Valintaan vaikuttavat päätekijät olivat mobiiliresponsiivisuus, suorituskyky, helppokäyttöisyys ja teknologian tuottavuus.

Työn tavoitteena oli tutkia käytettävien tekniikoiden mahdollisuuksia ja valita näistä paras mahdollinen vaihtoehto ohjelmiston/yrityksen jatkokehityksen tarpeisiin nähden. Tekniikan tuli sopeutua responsiivisuuteen, eli ohjelmiston tuli toimia niin pöytäkoneella kuin mobiilipäätelaitteilla.

1.2 Lyhenteet ja määritelmät

SQL = Structured Query Language, käytetään tietokantojen käsittelyyn.
TFS = Team Foundation Server, versionhallintajärjestelmä.
C# = Olioperustainen ohjelmointikieli.
JSON = JavaScript Object Notation. Datansiirron tiedostomuoto
IDE = Integrated Development Environment, ohjelmointiympäristö.
JavaScript = Ohjelmointikieli, jota käytetään pääasiassa HTML:n ja CSS:n hallinnoimiseen.
Framework = Ohjelmistokehys, joka tarjoaa jo valmiiksi rakennettuja osia ohjelmoijalle
let = Koodiblokin sisäinen muuttuja, ei ole saatavilla funktion ulkopuolelta
const = Vakioitu muuttuja, jonka arvoa ei voida muuttaa
Nuolifunktio = Toimii kuten tavallinen koodissa määritetty funktio, mutta lyhyemmällä kirjoituskaavalla.
PoC = Proof of Concept. Käytännön demonstraatio potentiaaliselle asialle tai teorialle.
OOP = Object Oriented Programming
BL = Business Logic
MVC = Arkkitehtuuri-malli, tulee sanoista Model-View-Controller.
WCF = Windows Communication Framework, käytetään palvelinten väliseen viestintään
Client-palvelin = MVC-arkkitehtuurilla toteutettu palvelin
BL-palvelin = .NET Framework-projekti, joka on yhteydessä tietokantaan
Fluent API = ohjelmointitapa, jossa luokka palauttaa itseään metodin paluuarvona
Repository-pattern = ohjelmointitapa, jolla pidetään halutun tietokantaobjektin kantaoperaatiot yhdessä luokassa
Unit of Work = luokka, joka hallinnoi Repository:n sisällä tapahtuvat kantaan kohdistuneiden operaatioiden tallennuksen tai peruuttamisen

2 DATA PRISMA OY

”Data Prisma on kotimainen ohjelmistotalo, jonka vahvuutena ovat kokemus, osaava henkilöstö ja tuoreiden teknologioiden hyödyntäminen. Teemme läheistä yhteistyötä asiakkaidemme kanssa, lähtökohtana asiakkaidemme toiveet ja tarpeet. Suunnittelemme ja toteutamme järjestelmiä yritysten ja yhteisöjen toiminnan tueksi. Järjestelmiemme avulla asiakkaamme hallinnoivat omien asiakkaidensa, heidän tilaustensa sekä tuotteiden tietoja ja näin tehostavat myynnin, laskutuksen ja reskontran hoitoa. Laajaan asiakaskuntaamme kuuluu operaattoreita, tapahtumien ja koulutusten järjestäjiä, liittoja ja yhdistyksiä sekä matkailualan yrityksiä.” (Data Prisma Oy, 2019)

Yksi Data Prisman projekteista on Kongressi-ohjelmisto. Sen käyttötarkoitus on toimia erilaisten tapahtumien osanottajien, luennoitsijoiden, tilojen ja talouden hallintatyökaluna (Data Prisma Oy, 2019). Kongressin kokonaisuuteen kuuluu pienempi projektikokonaisuus Portaali, jonka tarkoituksena on toimia asiakkaan tietojen ylläpitotyökaluna. Opinnäytetyössä on tarkoitus integroida Portaali-projektiin halutut ominaisuudet, jotka käsitetään Luennoitsijaportaalina.

Luennoitsijaportaali on aikaisemmin käsitetty omana projektinaan, jonka tehtävänä on ollut toimia mm. ohjelmaehdotuksien, tiedostojen, luentojen ja luentosarjojen käyttötyökaluna. Uuden toteutuksen pyrkimyksenä on toteuttaa uudella tekniikalla tarvittavat osat Portaali-projektiin.

3 KÄYTETYT KEHITYSTYÖKALUT

3.1 Visual Studio 2017

Visual Studio on ohjelmiston kehitykseen käytettävä työkalu, toisin sanoen ohjelmointiympäristö (IDE). Sitä käytetään mm. erilaisten Web-sivujen, ohjelmien ja sovellusten tekoon. Sillä pystytään myös helposti hallinnoimaan projektikokonaisuuksia. Visual Studio tukee myös monia ohjelmointikieliä. (Microsoft Visual Studio, 2018) Kyseinen IDE valikoitui palvelinsovelluksen generointiin, sillä se on tälläkin hetkellä yrityksen päätoiminen kehitystyökalu. Tässä projektissa käytettäviin ohjelmointikieliin kuuluivat HTML, JavaScript ja C#.

3.2 Visual Code

Visual Code on toinen projektissa kehitykseen käytettävä työkalu/ohjelmointiympäristö. Kyseinen ohjelmisto toimii samalla tavalla kuten Visual Studio eli sillä voidaan kehittää ja hallinnoida projekteja. (Visual Code, 2018)
Kyseinen työkalu otettiin projektin aikana käyttöön, sillä React:in kehittäminen oli sillä mutkattomampaa. Esimerkiksi npm -pakettien lisääminen ja käyttäminen sekä React-tuki olivat päätekijöitä työkalun valintaan.

3.3 Microsoft SQL Server Management Studio

Microsoft SQL Server Management Studio:lla hallinnoidaan SQL tietokantoja. (SQL Server Management Studio, 2018) Valinta työkalulle oli selkeä, sillä pohjalla oleva projekti käytti tietokantana Microsoft SQL-palvelinta ja on ollut käytössä yrityksessä aikaisemminkin.

3.4 Versionhallinta

Versionhallintaa käytettiin hallinnoimaan lähdekoodi-projektia. Sillä pystytään seuraamaan, mitä ja miksi kukin projektin kanssa työskentelevä on tehnyt muutoksia/poistoja/lisäyksiä tiedostoihin/projekteihin. Versionhallinta toimii yleensä omalla palvelimella tai pilvipalvelimella, jonka käyttöoikeudet jaetaan kaikille sitä tarvitseville ohjelmoijille.

Projektissa käytettiin Microsoftin TFS:ää sekä Git:iä. TFS huolehti pääosakseen palvelinkoodeista ja Git taas käyttöliittymäkoodeista. Tämä järjestely toteutettiin näin, koska Visual Code toimi huomattavasti helpommin Git:in kanssa, kun taas vanha, olemassa olevan projektin palvelinkoodia ylläpidettiin TFS:ssä.

4 TEKNIIKOIDEN VERTAILU

Tekniikan tuli sopeutua responsiivisuuteen eli ohjelmiston tulee toimia niin pöytäkoneella kuin mobiililipäätelaitteilla. Tämän tuli lisäksi integroitua olemassa olevaan MVC-projektiin, jonka myötä sen tuli toimia selaimessa. Kun uusia käytettäviä tekniikoita tutkittiin, päädyimme vertailemaan kolmea sen hetken käytetyintä UI-kehitykseen käytettävää tekniikkaa. Nämä olivat Vue.js, Angular ja React.

4.1 Vue.js

Vue.js on avoimen lähdekoodin progressiivinen framework käyttöliittymien rakentamiseen. Sen on kehittänyt Evan You (entinen Googlen ja AngularJS:än työntekijä). Vue.js sivuilla mainostetaan sen kykyä olla erittäin helposti lähestyttävät ja sen integroimiskynnys toisiin projekteihin tai kirjastoihin on erittäin matala. (Evan You, 2018) Sanotaankin, että Vue on ottanut itselleen React:n ja Angular:n parhaimmat puolet.

4.2 Angular

Angular on avoimen lähdekoodin alusta ja framework HTML- ja JavaScript- käyttöliittymäsovellusten rakentamiseen. Sitä kirjoitetaan TypeScript:llä. Sen mukana tulevat pakolliset sekä valinnaiset TypeScript-toiminnallisuudet. Angular sisältää siis paljon valmista toiminnallisuutta. Sen kehittäjänä toimii Google. (Google, 2018)

4.3 React

React on avoimen lähdekoodin JavaScript-kirjasto, jonka on kehittänyt Facebook (Facebook, 2018). React toimii pohjana käyttöliittymäkoodille, jonka ajatuksena on luoda helposti uudelleenkäytettäviä käyttöliittymäkomponentteja, joihin voidaan sisällyttää helposti kustomoitavia ominaisuuksia. Tämä mahdollistaa erittäin helpon uudelleenkäytettävyyden komponenteista kokonaisiin sivurakenteisiin. Koodin generointi ja lukeminen on täten helppoa. (Eric Simons, 2018)

React sisältää erittäin vähän valmiita osia tai kirjastoja. Siihen on ladattavissa yhteisön tekemiä kirjastolaajennuksia, joita voidaan käyttää projektissa. Tämä mahdollistaa valinnanvapauden joko valitsemalla valmiita käytettäviä kokonaisuuksia/komponentteja tai luomalla sellaisia itse.

4.3.1 ECMAScript

ECMAScript on standardi, jota JavaScript implementoi. Se määrittää esimerkiksi syntaksin, jolla JavaScriptiä kirjoitetaan. Projektia aloittaessa päädyttiin käyttämään ECMAScript 6 -versiota, sillä sen oppiminen oli helppoa ja syntaksi oli jo lähestulkoon tuttua C#:n puolelta sekä se oli suosittu standardi. Kyseiseen versioon kuuluvat esimerkiksi let, const ja nuolifunktio**.** (W3Schools, 2018)

Nuolifunktion syntaksia on kuvattu kuvassa 1.

```
// ES5
var x = function(x, y) {
   return x * y;
}

// ES6
const x = (x, y) => x * y;
```

Kuva 1 Esimerkki nuolifunktion käytöstä (W3Schools, 2018)

4.3.2 JSX

JSX on koodisyntaksi, joka toimii XML-formaatin mukaisesti, mutta kääntäjä kuitenkin muokkaa sillä kirjoitetun koodin standardoiduksi ECMAScript:ksi. (Facebook, 2018)
Esimerkiksi React:ssa omien komponenttien käyttö toimii kehittäjän näkökulmasta kuvan 2 mukaisesti.

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

Kuva 2 JSX-syntaksi ennen kääntöä (Facebook, 2018)

Kääntäjän työskentelyn jälkeen lopputulemaksi saadaan kuvan 3 mukainen tulos, joka on nähtävissä selaimessa.

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

Kuva 3 JSX-syntaksi käännön jälkeen (Facebook, 2018)

4.4 Vertailun suunnittelu

Vue pudotettiin vertailujoukosta jo alkuvaiheessa, kun saatiin selville, että sen parissa työskentelee vain muutama kehittäjä, jolloin kehityksen edistyminen on luonnollisesti hitaampaa sekä riski tekniikan kehityksen kuolettumiselle oli huomattavasti suurempi, kun taas toisten tekniikoiden takana toimivat suuret yritykset (Google ja Facebook). Näin käyttöliittymätekniikoiden vaihtoehdoiksi päädyttiin rajaamaan React ja Angular.

Valinnalle vaadittiin muitakin perusteluita kuin blogeissa ja foorumeilla käytyjä pohdintoja, joten vertailua päätettiin toteuttaa PoC:ien avulla. Molempien tekijöiden tuli toteuttaa yhdellä tekniikalla Omat Tiedot-näkymä, joka sisälsi yksinkertaisen lomakkeen henkilötietojen ylläpitämiseen. Varsinainen valinta perustuisi aikaisemmin määritettyjen kriteereiden myötä myös oppimiskynnykseen.

Vertailua varten listattiin nykyisen kehityksen ongelmakohtia, joihin tekniikan tuli tarjota ratkaisuja. Nykyisessä kehityksessä elementtien yleiskäyttöisyys oli tehty hankalaksi, debuggaaminen näkymällä oli vaivalloista, ulkoisten JavaScript-kirjastojen hyödyntäminen vaati paljon JavaScript-koodilla alustamista ja asettelua, live-validointia ei juurikaan ollut sekä lomakkeen hallinta koodissa tapahtui pitkälti palvelimen koodissa.

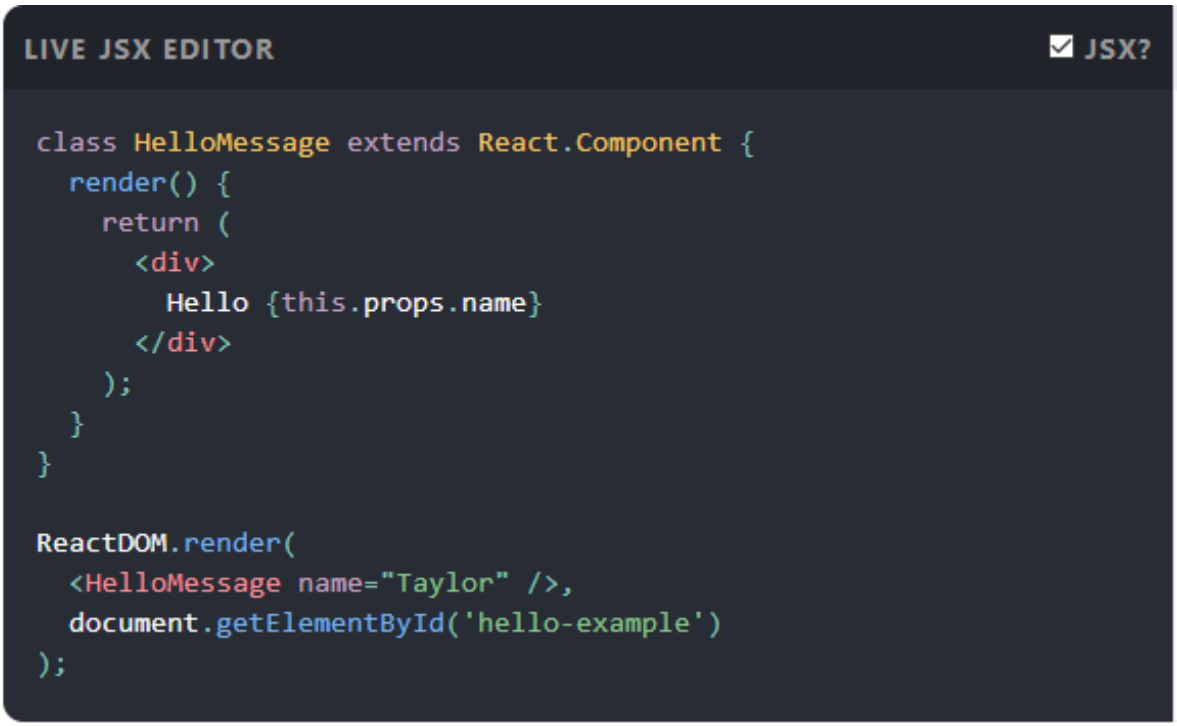
4.5 Vertailun vaadittavan datan generointi

Ennen PoC:in aloittamista oli luonnollisesti opeteltava uutta tekniikkaa eli React:ia. Tämä toteutettiin tutkimalla ja suorittamalla eri opetusmateriaaleja ja kursseja verkosta. Kollegaltani saamani vinkin mukaan lähdin suorittamaan Codecademy-sivuston React:iin liittyvät kurssit 1-3, joissa käytiin läpi teoriaa sekä käytännön työtä kädestä pitäen, joka helpotti React -maailmaan sisäistämistä huomattavasti.

Kurssien suorittamisen jälkeen toteutin Omat Tiedot-näkymän. Itse käyttöliittymäprojektin luonti ja komponenttien tekeminen oli mutkatonta, mutta integrointi vanhaan järjestelmään ja datan siirtäminen palvelimelle eivät vielä tulleet selviksi kurssien avulla. Aloitin toteuttamaan vanhan järjestelmään liittämistä käyttämällä axios-kirjastoa, jolla käsiteltiin kaikki http-pyynnöt käyttöliittymästä palvelimelle. Näiden pyyntöjen avulla dataa tuotiin käyttöliittymään ja siirrettiin lomakkeelta takaisin palvelimelle. Tämä em. kirjasto jäikin käyttöön myös Luennoitsijaportaali-projektiin.

4.5.1 Elementtien yleiskäyttöisyys

Nykyisessä MVC-tekniikan kehityksessä modalien ja vastaavien toistuvasti käytettävien elementtikokonaisuuksien teko oli toteutettu haastavasti. Yhteistä linjanvetoa ei ollut, joten uudelta käytettävältä tekniikalta odotettiin tähän korjausta. Esimerkkinä tästä on toiminut Bootstrap-Modal, joka on muodostanut selaimen näkymälle dialog-laatikon, joka ilmestyy Razor-näkymän päälle. React sekä Angular mahdollistivat tämän helposti arkkitehtuurin ja komponenttiajattelun myötä, jolloin ko. komponentille pystyttiin syöttämään halutut parametrit helposti ja selkeästi, kuten kuvan 4 esimerkissä nähdään.



Kuva 4 Komponenttiesimerkki React:ssa (Facebook, 2018)

4.5.2 Debuggaus

Nykyisen kehityksen käyttöliittymän debuggaus oli vaivalloista Razor-tekniikan ja niissä käytettävän JavaScriptin takia, sillä Visual Studio ei pysty käsittelemään näitä halutusti. Visual Studio Code tarjosi molemmilla tekniikoilla näihin ratkaisun selainkohtaisilla laajennuksilla. Käytännössä tämä vaati selaimeen asennettavaksi käytettävän tekniikan debuggaus-laajennuksen sekä Visual Studio Code:ssa tuli asettaa tarvittavat parametrit launch.json tiedostoon.

4.5.3 Ulkoisten kirjastojen hyödyntäminen

Ulkoisten ”plugielementtien”, kuten datepickerin, checkboxien tai radiobuttoneiden käyttö oli helppo ottaa käyttöön React:ssa. Valinnan jälkeen voitiin vain asentaa haluttu node-paketti projektiin ja ottaa se käyttöön halutussa kohtaa koodia. Angular:ssa tulee itsessään peruskäyttöiset komponentit ko. tapauksissa, jotka pienellä muokkaamisella saadaan halutunlaiseksi.

4.5.4 Live-validaatio

Live-validaation tarkoituksena oli kertoa käyttäjälle heti syötteen antamisen yhteydessä syötteen validiteetti. Tämä toteutuisi täysin käyttöliittymän päässä eli selaimessa (front-end), jolloin palvelimelle lähtevät tiedot olisivat jo alustavasti oikeanlaisessa muodossa. Tämä vähentäisi palvelimen kuormaa, sillä vääränlaista dataa sisältäviä pyyntöjä tulisi huomattavasti vähemmän käsiteltäväksi.

4.5.5 Kielisyys

Kielisyys on tärkeä osa Kongressia. Tämän tuli siis toimia erittäin jouhevasti valittavassa tekniikassa. React:ssa tämä oli helppo toteuttaa ylimääräisen kirjaston avulla, joka hallinnoi näytettävää kieltä ja noukki JSON-tiedostosta avaimen avulla halutun kielisyyden alta oikean tekstin. Angular sen sijaan generoi jokaisesta kielisyydestä oman komponentin sivukokonaisuudesta, mikä tuntui taas hankalalta integroinnin ja käytettävyyden kannalta.

4.5.6 Integrointi muihin projekteihin

Uuden tekniikan osien tuli soveltua toimimaan myös muissa projekteissa, jotka ovat toteutettu ASP.NET:in MVC-mallilla. Tämän myötä voitaisiin päivittää projekteihin vanhoja toteutuksia tehtynä uudella tekniikalla sekä lisätä uusia komponenttikokonaisuuksia toimimaan vanhan toteutuksen seassa. Molemmissa tekniikoissa tämä on helppo mahdollistaa ohjelmoimalla JavaScript:llä osan näkyvyyden niin sanottuun globaaliin kontekstiin, jolloin eri projektikokonaisuudet ovat samassa ympäristössä ja voivat näin tunnistaa toisessa projektissa määritettyjä funktioita. Toinen tapa on käyttää MVC:ssä Razor:lla generoitujen div-elementtien id-attribuuttia React-projektissa. Näin osa pystyttiin asettamaan haluttuun paikkaa JavaScriptin getElementById-funktion avulla, jonka parametriksi syötetään div-elementin id.

4.5.7 Oppimiskynnys

Vertailussa tutkittiin, kuinka paljon kunkin tekniikan oppimiseen oli käytetty aikaa. Oppimiskynnyksen kriteereinä toimivat teoreettiseen ja käytännölliseen oppimiseen käytetty aika. Tekniikoiden välillä ei kuitenkaan ollut mainittavaa eroa kummankaan kriteerin osalta. Lopputulemana Angular oli nopeampi ottaa käyttöön, mutta kärsi siitä kustomoitavuuden puitteissa. React vaati käyttöönotossa paljon suunnittelua ja pohdintaa siitä, otettaisiinko jokin ulkoinen kirjasto käyttöön vai toteutettaisiinko ratkaisu itse.

4.6 Vertailun lopputulos

Kun taustatutkimukset sekä PoC:it saatiin valmiiksi vertailua varten, tuloksia esiteltiin palaverissa. Vertailussa käytiin läpi kuvaan 5 kirjatut vertailukohdat. Palaverin lopputuloksena päädyttiin valitsemaan käytettäväksi tekniikaksi React. Tähän päätökseen johtivat kustomoitavuus ja kielisyyden käyttö. Tämän lisäksi yrityksen henkilöstöstä löytyi sitä kohtaan jo kokemusta harrastuksen kautta. Lisäksi Angularin kielisyys toteutti useamman ilmentymän joka näkymästä, riippuen tuettujen kielien määrästä, joka teki integroinnista MVC-projekteihin haasteellisen.

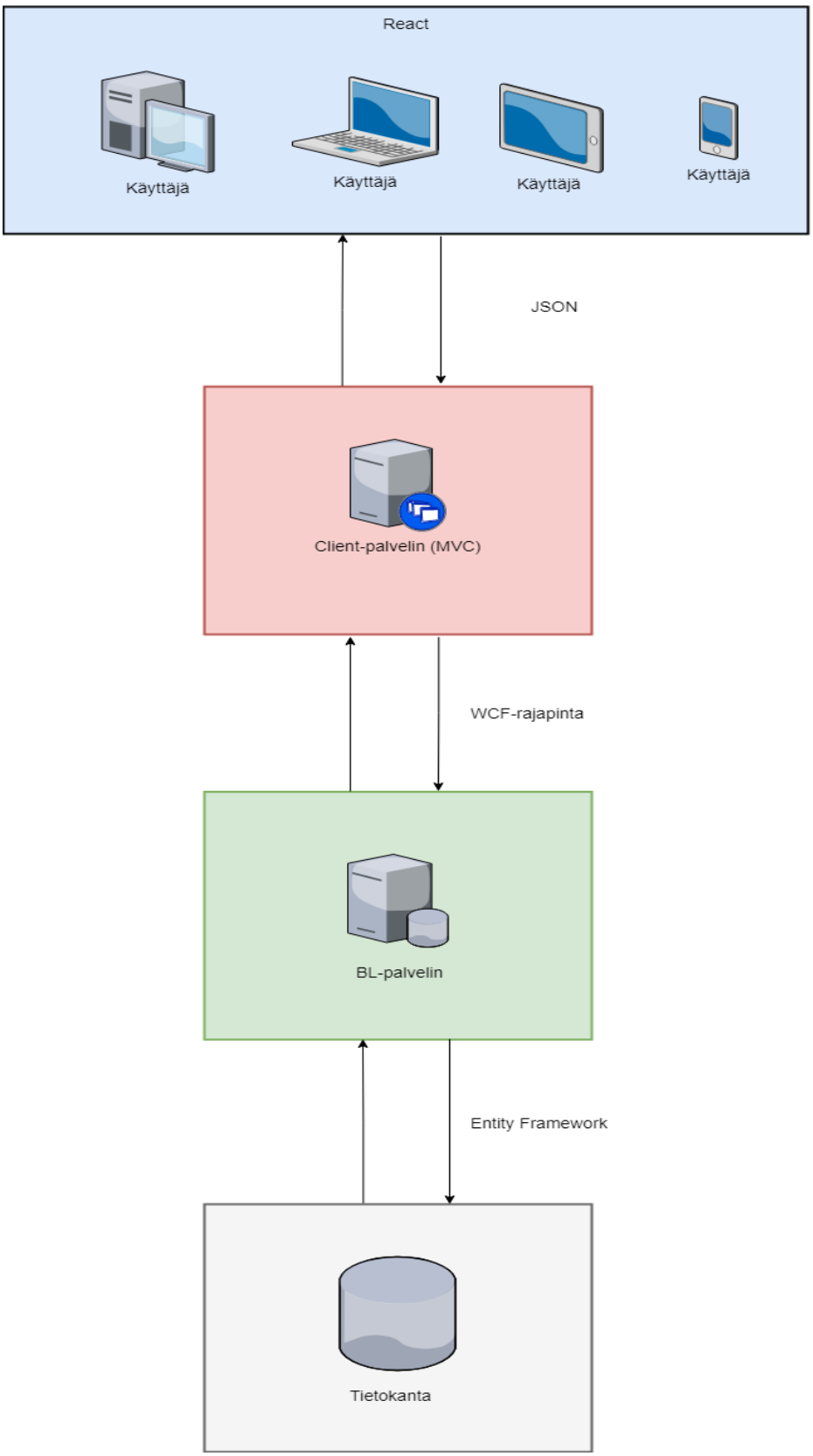
PoC vertailu	Angular	React	MOLEMMAT
1) Modaalien hallinta on hankalaa => aukominen, sulkeminen, populoiminen datalla ja uudelleenkäyttö ei ole selkeää	Voidaan luoda modal-komponentti, jonka sisälle voidaan renderöidä toinen komponentti	Voidaan luoda modal-komponentti, jonka sisälle voidaan renderöidä toinen komponentti	Voidaan luoda molempiin omanlainen modal, vaatii lähinnä yhteiset pelisäännöt käytön osalta.
Tavoite: helposti hallittavat modaalit, joiden tilaa voi seurata ja muokata ohjelmallisesti.			
2) Tarkoitukseltaan ja ulkoasultaan identtisiin elementteihin löytyy useita rakenteellisia toteutuksia(!)	Voidaaan luoda komponentteja elementtirakenteista	Voidaaan luoda komponentteja elementtirakenteista	
Tavoite: elementtirakenteet attribuutteineen voi paketoida uusiokäyttöisiksi paketeiksi.			
3) Debuggaus käyttöliittymässä monen mutkan takana			
Tavoite: käyttöliittymää voi testata suorasukaisesti eri käyttäjätyypeillä (peruskäyttäjä/admin) ja validoinneilla (valid/invalid)	Chrome debugger laajennuksella	Chrome debugger laajennuksella	
4) Plugielementtien käyttö ohjelmallisesti vaatii ylimääräistä js-koodia (datepickerit, checkboxit, radiobuttonit)	Voidaan käyttää Angularin omia datepickereitä yms komponentteja	Voidaan luoda omat datepicker yms komponentit tai importata npm:n kautta	
Tavoite: plugielementtien hallinta toimii ilman js-kontrollikoodia tai keskitetyllä paketilla.			
5) Käyttäjävahvistuksen toteutukselle ei ole yleistä ratkaisua (muu kuin confirm()-metodi)			
Tavoite: yksinkertainen tapa pyytää käyttäjältä vahvistus toiminnon "x" suorittamiselle.	Onnistuu luoda vahvistusikkunakomponentti	Onnistuu luoda vahvistusikkunakomponentti	Molempiin voidaan luoda komponentit
6) Implementaatio tai kehys live-validaatiolle			Molempiin voidaan luoda
Tavoite: kenttiä voi validoida jo ennen lomakkeen lähetystä, jotta käyttäjä ja serveri säästyvät turhalta työltä.	Voidaan rakentaa itse tai käyttää Angularin omaa Validators luokkaa	Voidaan rakentaa itse (vaatii aika paljon työtä) tai käyttää jotain valmista kirjastoa	
7) Lomake-toimintojen hallinta ohjelmallisesti			
Tavoite: sisäinen hallinta painikkeiden disabled/enabled tilalle, mukaanlukien tyylien hallinta, ilman erillistä scriptausta.	Onnistuu	Onnistuu	
8) Lomake-elementtien hallinta ilman show/hide scripteja			
Tavoite: sisäinen tapa säätää kenttien/elementtien näkyvyys sivulla (ei saa häiritä lähetystä).	Voidaan tehdä ehdollistamalla komponentin dataa vasten	Voidaan käyttää statea hyväksi, sijoitetaan sinne arvo ja lähetetään sen arvo eteenpäin	
9) Integroitavissa olemassa olevaan ympäristöön			
Tavoite: uusi käyttöliittymäkoodi pystyy elämään aiemman koodin kontekstissa ja sen metodeja voi kutsua frameworkin ulkopuolelta.	Näkymät voidaan upottaa Razorin sekaan ja sen metodeita voidaan kutsua myös ulkoa päin	Näkymät voidaan upottaa Razorin sekaan ja sen metodeita voidaan kutsua myös ulkoa päin	Henkilötiedot portaaliin
10) kielisyys	Useammalla kielellä toteutetun näkymän upottaminen vanhaan toteutukseen voi olla hankalaa	Näkymien kielisyyksien toteuttaminen näkymäkomponenttiin onnistuu	

Kuva 5 Vertailukohteet taulukossa

5 TYÖN TOTEUTUS

Tekniikan valinnan jälkeen aloitettiin suunnittelemaan React-projektin ja palvelinohjelman toteutusta. Ensimmäisenä tuli ratkaista arkkitehtuuri React-projektissa, jotta saataisiin selkeät jaot erinäisten vastuualueiden kesken. Tämän jälkeen mietittiin selaimelta tulevan pyynnön käsittely client-palvelimen päässä. Käsittelyn jälkeen jatkettiin yhä edelleen WCF-rajapinnan ylitse BL-palvelimelle

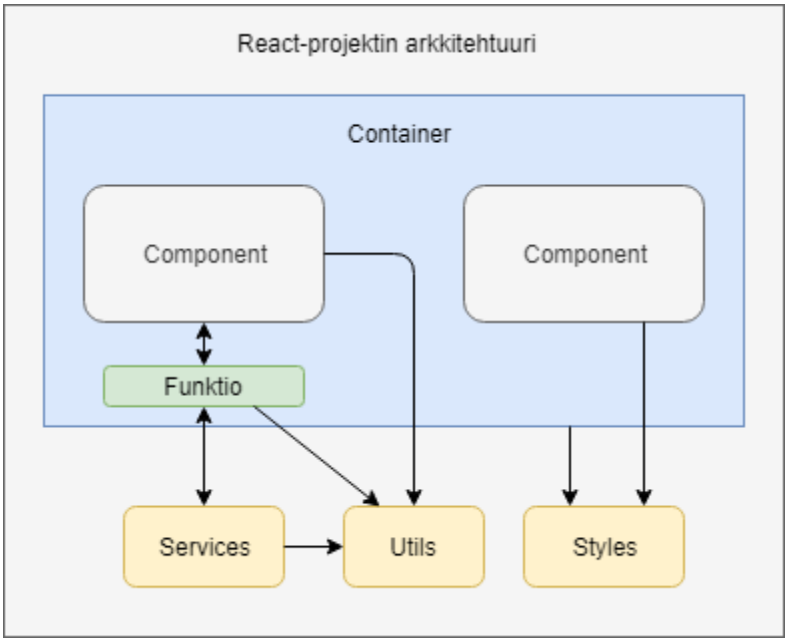
Töiden jako projektissa olleiden kesken jaettiin käyttöliittymäkoodin ja palvelinkoodin toteutuksiin. Suunnitteluvaiheessa päätettiin, kuka toteuttaa minkäkin osan yhdestä kokonaisuudesta. Tämä testasi samalla myös koodien riippumattomuutta keskenään, jotta käyttöliittymäkoodi olisi puhtaana liiketason logiikasta. Kuvassa 6 nähdään toiminnan kulku, kun käyttäjä laukaisee jonkin palvelinta kutsuvan tapahtuman.



Kuva 6 Alustava sovelluksen toimintakaavio

5.1 React-projekti

React-projektin tuli saada mahdollisimman selkeä arkkitehtuuri ja säännöt ominaisuuksien toteuttamisesta. Tämän myötä tiedettäisiin jatkossa mistä löydettäisiin tietynlaisen vastuun sisältävän tiedoston sijainti. Alustava projektin arkkitehtuurin rakenne toteutettiin kuvan 7 mukaisesti.



Kuva 7 React-projektin alustava arkkitehtuurimalli

5.1.1 React-projektin osat

Container käsittää pohjan komponenteista muodostetulle kokonaisuudelle. Tähän pohjaan kerättiin eri komponentit, jotka hallitsevat pienempiä vastuita, esimerkiksi napin painalluksen toimintaa.Component käsitetään yksittäisenä komponenttina. Komponenteista esimerkkinä toimivat syötekentät, otsikot ja modalit. Jos komponenttia tarvitaan useammassa paikkaa, se siirrettäisiin CommonComponents-osioon, josta useampi eri container voi halutessaan käyttää sitä.

```
10  interface IProps {
11      files: IFileRow[];
12      selectAllClick(): void;
13  }
14
15  const TableHeading: React.SFC<IProps> = (props) => {
16      const allSelectableSelected = !props.files.some((file) => file.deletable && !file.selected);
17      const hasDeletableFile = props.files.some((file) => file.deletable);
18      const hasFiles = props.files.length > 0;
19
20      return (
21          <React.Fragment>
22              <section className="table--heading">
23                  <div className="cell-2 centered">
24                      <ConditionalCheckbox
25                          showCondition={hasFiles && hasDeletableFile}
26                          onChange={props.selectAllClick}
27                          id="cb_select_all"
28                          checked={allSelectableSelected}
29                          disabled={!props.files.some((file) => file.editable)}
30                      />
31                  </div>
32                  <div className="cell-2 icon-cell centered">{Icons.file}</div>
33                  <div className="cell-auto hideOverflow">
34                      <strong>{resources.fileName}</strong>
35                  </div>
36                  <div className="cell-8">
37                      <strong>{resources.fileState}</strong>
38                  </div>
39                  <div className="cell-4" />
40              </section>
41          </React.Fragment>
42      );
43  };
44
45  export default TableHeading;
46
```

Kuva 8 Komponentti-esimerkki

Styles eli tyyliluokat sisältävät käsittävät kaikki CSS-tyylit. Nämä kertovat selaimelle, millaisena ja missä kohtaa sivua jokin elementti näkyy.

Services eli palvelut ovat vastuussa selaimen ja palvelimen välillä tapahtuvasta datan käsittelystä ja siirrosta. Nämä operaatiot tapahtuvat REST-arkkitehtuurimallin mukaisesti. Operaatiot toimivat axios-kirjaston avulla, joka hoitaa pyyntökäsittelyt asynkronisesti.

```
TS CommonService.ts ×

    nikaupp, 2 months ago | 2 authors (nikaupp and others)
1   import axios, { AxiosError, AxiosResponse } from 'axios';      jemartyy, 6 months ago • tapahtuman nimen haku
2   import { CommonService as routes } from '../Constants/Routes';
3
    nikaupp, 2 months ago | 2 authors (nikaupp and others)
4   class CommonService {
5       static getLanguageId = () =>
6           axios
7               .post(routes.languageRoute)
8               .then((response) => response.data)
9               .catch((error: AxiosError) => console.error(error));
10
11      static getNetworkName = () =>
12          axios
13              .post(routes.getNetworkNameRoute)
14              .then((response) => JSON.parse(response.data))
15              .catch((error: AxiosError) => console.error(error));
16  }
17  export default CommonService;
18
```

Kuva 9 CommonService-luokan sisältöä

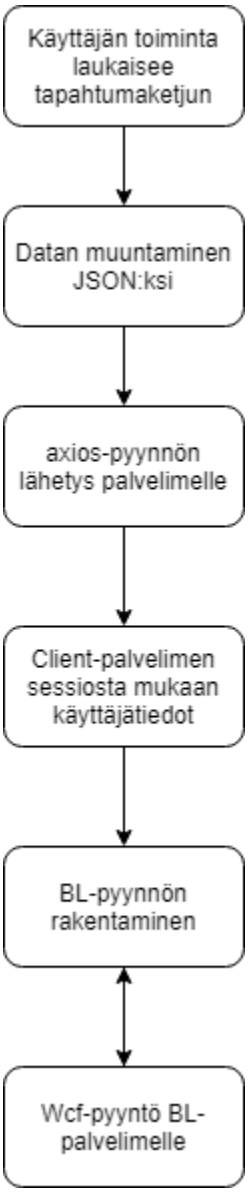
Utils eli työkalut sisältävät kaikki yleiskäyttöiset koodit, joita komponentit voivat kutsua, kuten apu-funktiot ja toiminnallisuutta sisältävät funktiot. Esimerkiksi olion kopiointi ja datakonversiot.

```
// Interface language
static getCurrentLanguage() {
    return parseInt(localStorage.getItem(KEY.CURRENT_LANGUAGE) as string, 10);
}

static setCurrentLanguage(key: number) {
    return localStorage.setItem(KEY.CURRENT_LANGUAGE, key.toString());
}

//interface locale
static getCurrentLocale() {
    const locale = localStorage.getItem(KEY.CURRENT_LOCALE);
    if (!locale || locale !== window.navigator.language) {
        WebStorage.setCurrentLocale(window.navigator.language);
    }
    // tällä functiolla hallitaan applikaation localea,
    return 'fi-FI';
}       nikaupp, 3 months ago • webstorage update, locale ja language erotettu toisistaan

static setCurrentLocale(key: string) {
    return localStorage.setItem(KEY.CURRENT_LOCALE, key.toString());
}
```

Kuva 10 Esimerkki Utils-kansion sisältävästä luokasta

5.2 Tiedonsiirto selaimen ja client-palvelimen välillä

Selaimen ja client-palvelimen välinen tiedonsiirto pyrittiin toteuttamaan RESTful API:n kaltaisesti sen selkeyden ja helppokäyttöisyyden takia. Jokin komponentti pyytää tiedon lähetystä Service-luokalta, joka päättelee funktion myötä millainen URL rakennetaan. Tämän jälkeen data paketoidaan FormData-luokkaan. Tarkoituksena oli ensin suorittaa datansiirtoa pelkästään JSON-muodossa, mutta se paljastui haasteelliseksi tiedostojen siirrossa, jonka myötä siirryttiin käyttämään FormData-luokkaa. Pyyntö suoritettiin Services-luokasta axios-pyynnöllä HTTP-POST:na client-palvelimelle, jossa data siirtyi yhä eteenpäin BL-palvelimelle WCF-rajapinnan kautta lukuun ottamatta tiedostoja, sillä tiedostojen siirto BL-palvelimelle asti olisi tuonut mukanaan huomattavaa suoritussakkoa varsinkin isoilla tiedostoilla.



Kuva 11 Selaimen ja client-palvelimen toimintakaavio

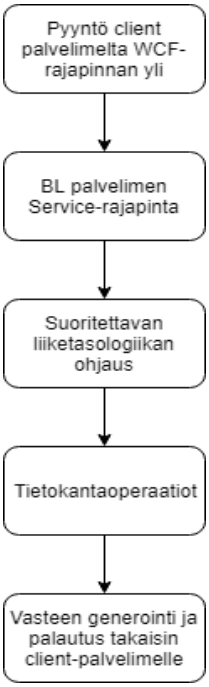
5.3 Tiedonsiirto client-palvelimen ja BL-palvelimen välillä

Lähtökohtaisesti haluttiin, että client-palvelimen koodiin ei tarvitsisi tehdä React-pyyntöjen osalta loogista päättelyä suorituskyvyn takia, poikkeuksena tiedostonkäsittelyn koodaaminen. Muut toiminnot toteutettaisiin pelkästään RESTful-implementaation ohjaukselle. Näin ylläpidettävän koodin määrä pysyisi mahdollisimman keskitetysti React-projektissa ja BL-palvelimella, jolloin hallinnointi ja ylläpito säilyisi yksinkertaisena. Client-palvelimella rakennettiin pyyntö WCF-rajapintaa varten, joka lähetettiin sitten BL-palvelimelle. Pyynnön rakentamisen logiikassa käytettiin client-palvelimelle tulleen pyynnön URL:ia, josta saatiin pyynnön tyyppi sekä haluttu operaatio. Tämä tieto siirrettiin WCF-rajapinnan yli BL-palvelimelle, joka päätteli pyynnön tyypin ja operaation perusteella halutun koodilohkon suorittamisen.

5.4 BL-palvelimen toteutus

BL-palvelimella purettiin client-palvelimelta lähetetty pyyntö, josta ensin poimittiin Route:n avain ja operaatio, joista voitiin päätellä, mitä toimenpiteitä tulisi suorittaa. Tämän jälkeen purettiin pyynnön muut parametrit.

BL-palvelimella koodausta toteutettiin OOP-mallin mukaisesti, jolla vastuut saatiin jaettua järkevästi omiin osiinsa. Tässä hyödynnettiin lisäksi Fluent API:a, Repository-patternia sekä Unit of Work:ia, mitä ei aikaisemmin ole ollut käytössä projekteissa. Koodista saatiin helpommin ymmärrettävää ja tiedettiin mitä missäkin koodilohkossa tapahtuu. Lisäksi ylläpidettävyys helpottui. Koodin suorituksen jälkeen palautettiin vaste client-palvelimelle, joka käytännössä toimi vain vasteen siirtäjänä yhä edelleen selaimella pyörivälle React-projektille asti.



Kuva 12 BL-palvelimen toimintakaavio

5.4.1 Logiikan toteutus BL-palvelimella

Logiikkaosuuden toteutuksessa pidettiin tärkeänä saada aikaiseksi mahdollisimman helposti lähestyttävää ja ylläpidettävää koodia, jotta seuraavan koodia tutkivan ohjelmoijan olisi helppo ymmärtää ja nähdä koodien loogiset vastuualueet selkeästi. Tähän tutkittiin eri variaatioita ja päädyttiin lopulta tukemaan OOP-pohjaista Fluent API:n kaltaista ratkaisua. Lisäksi tietokantaoperaatioiden vastuut toteutettiin Repository ja Unit of Work-mallien mukaisesti.

Fluent API:n (toiselta kutsumanimeltään Fluent Interface) tarkoituksena on tehdä koodista ihmiselle luettava. Yleisenä periaatteena toimii yhden vastuualueen sitominen luokaksi, johon luodaan ketjutettavia metodeja. Tällöin koodin ulkoasu näyttää kuvan 13 mukaiselta.

```
[HttpPost]
0 references
public class MaterialFileGetListRoute : RouteAction {
    24 references
    public override string RouteName => "file/getlist";

    24 references
    public override ProxyResponse Execute(ProxyRequest request) {
        using(var context = request.GetContext()) {
            var response = context.GetFileLister()
                                  .ValidateFileListRequest()
                                  .SetVisibleFileIds()
                                  .SetFileInfos()
                                  .SetFileStatuses().GetResponse();
            if(!response.Result.IsNullOrEmpty()) {
                Content = JsonConvert.SerializeObject(response.Result);
                Result = true;
            }
            else {
                Result = false;
            }
        }
        return GetResponse();
    }
}
```

Kuva 13 Fluent API:n käyttö BL-palvelimen koodissa

Luokkien sisällä toteutettiin tietokantaan kohdistuvia operaatioita, joka toteutettiin Entity Framework:llä. Nämä käsittelyt toteutettiin repository-mallin mukaisesti. Repository:n vastuulla on sisältää kantakuvaan kohdistuvat kyselyt eli tietueiden haut, lisäykset, muokkaukset ja poistot. Se ei kuitenkaan ole vastuussa itse tietokantaan kohdistuvasta lopullisesta tallentamisesta tai peruuttamisesta. Tämä vastuu annettiin Unit of Work-malliselle luokalle. Unit of Work:in tarkoituksena on syöttää repositoryille tietokantakuva halutusta oliokokonaisuudesta ja hallita kantaan kohdistuneiden operaatioiden tallennus eli commit tai peruuttaminen eli rollback.

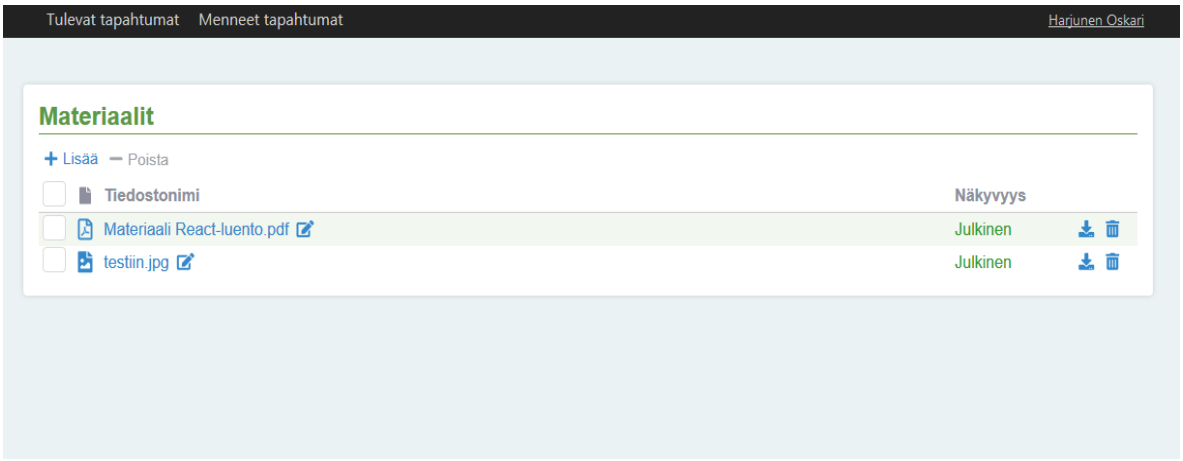
6 LUENNOITSIJAPORTAALI

Luennoitsijaportaalilla käsitetään Portaaliin tehdyt lisäosat, jotka ovat erillisiä sivuja. Näitä varten vanhaan projektiin luotiin linkit, jotka lähettivät pyynnön client-palvelimelle, joka taas palautti Razor-näkymän sisältäen React-projektin käännetyn, minifioidun JavaScript-tiedoston. Luennoitsijaportaalin sivujen näkyvyyttä käyttäjälle hallinnoidaan Kongressin ilmoittautumislomakkeiden hallinnan kautta.

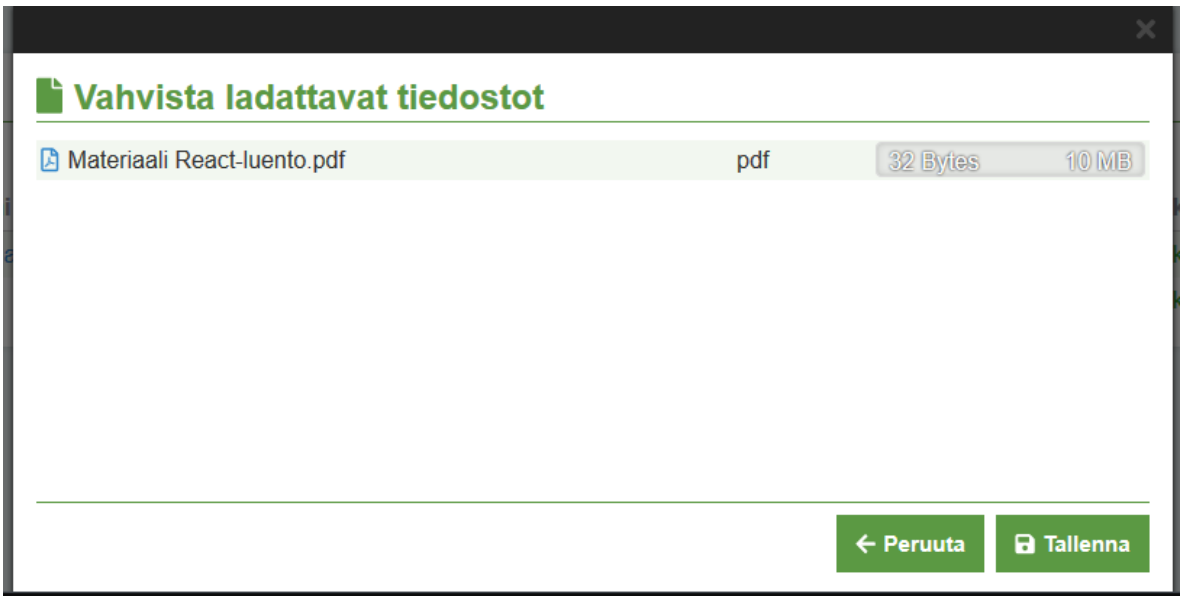
6.1 Materiaalit

Kirjoitusvaiheessa ohjelman toiminnot olivat muuttuneet niin, että oikeuksia pystyttiin muokkaamaan vain uuden ominaisuuden, Kongressin tiedostonhallinnan kautta. Kuvareferenssit tiedoston näkyvyyden muokkaamisesta tulevat Jere Martiskaisen opinnäytetyöstä, jolloin ominaisuudet olivat vielä opinnäytetyön aikaan määriteltyjen ominaisuuksien mukaiset. Materiaalit-sivu on tarkoitettu tiedostojen lataamiseen, jakamiseen ja ylläpitoon. Käyttäjän oikeuksien mukaan sivulla voidaan ladata, lisätä, muokata tai poistaa tiedostoja. Kuvassa 14 nähdään Materiaalit-sivu kokonaisuudessaan. Sen sisällä hallinnoidaan omia tiedostoja ja nähdään myös muiden jakamia, julkisia tai käyttäjälle kohdennettuja tiedostoja. Näistä ominaisuuksista toteutin tiedoston tallennuksen, latauksen ja jaetun tiedoston ominaisuudet.

Kuvassa 15 on modal, joka sisältää käyttäjän järjestelmään lähettämän tiedoston.

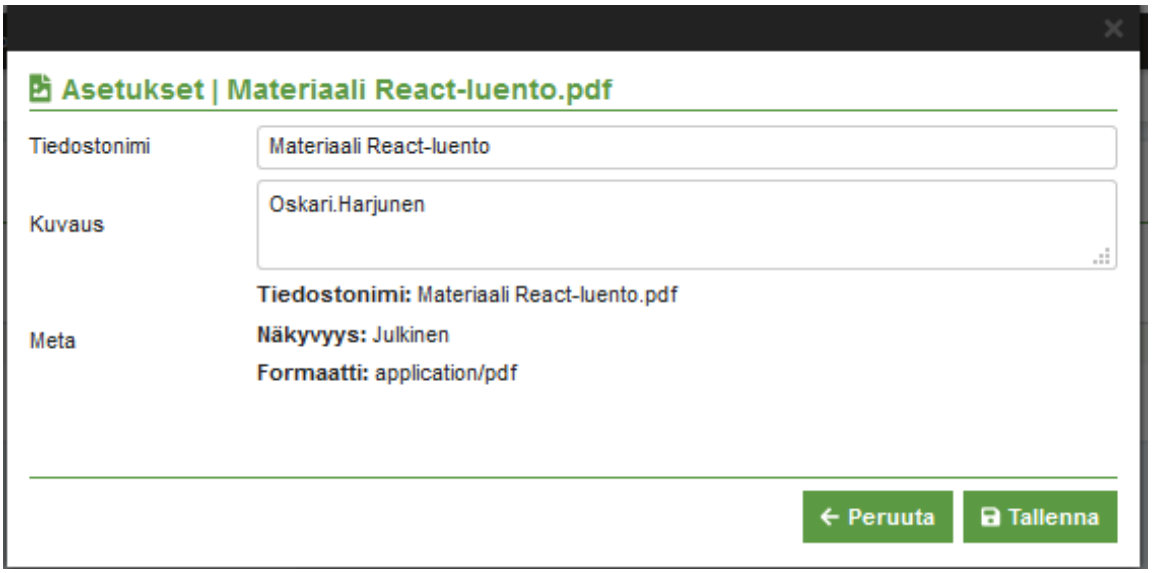


Kuva 14 Materiaalit Portaalissa



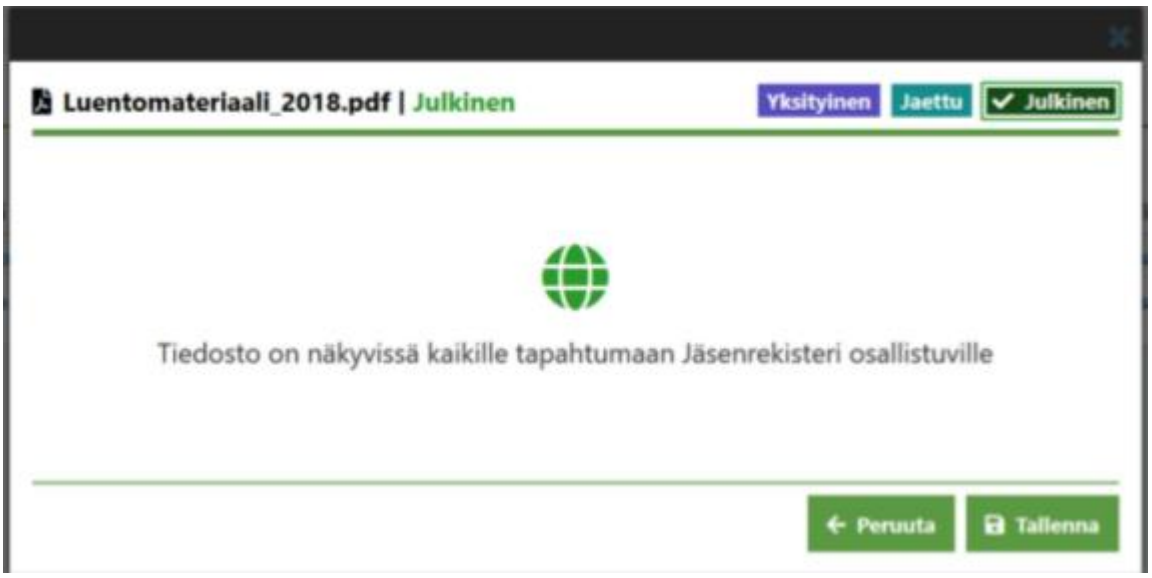
Kuva 15 Tiedoston lisäys-modal

Käyttäjä voi muokata oman tiedoston nimeä, kuvausta ja näkyvyyttä. Jokaiselle kentälle on omat validointisäännöt niin React-projektissa kuin BL-palvelimen päässä. Tällä pyritään estämään virheellisen datan käsittely, jos käyttäjä pyrkii hakkeroimaan sovellusta. Kuvassa 16 on tiedoston tietojen muokkausnäkymä.



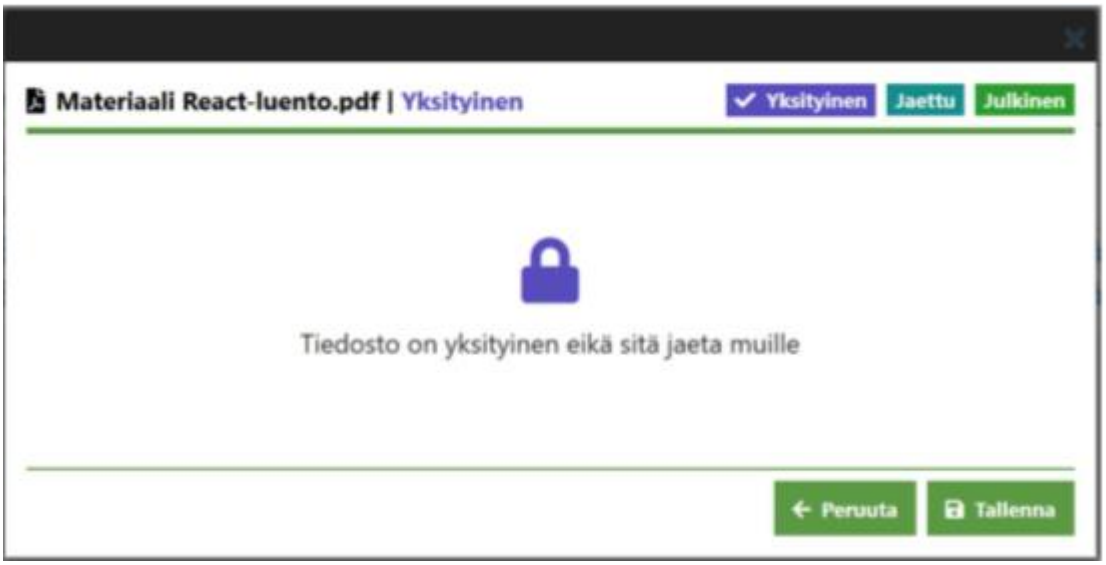
Kuva 16 Tiedoston muokkaus

Kuvassa 17 on julkiseksi määritetyn tiedoston modal. Julkinen näkyvyys asettaa tiedoston näkyviin kaikille käyttäjille tapahtuman sisällä.



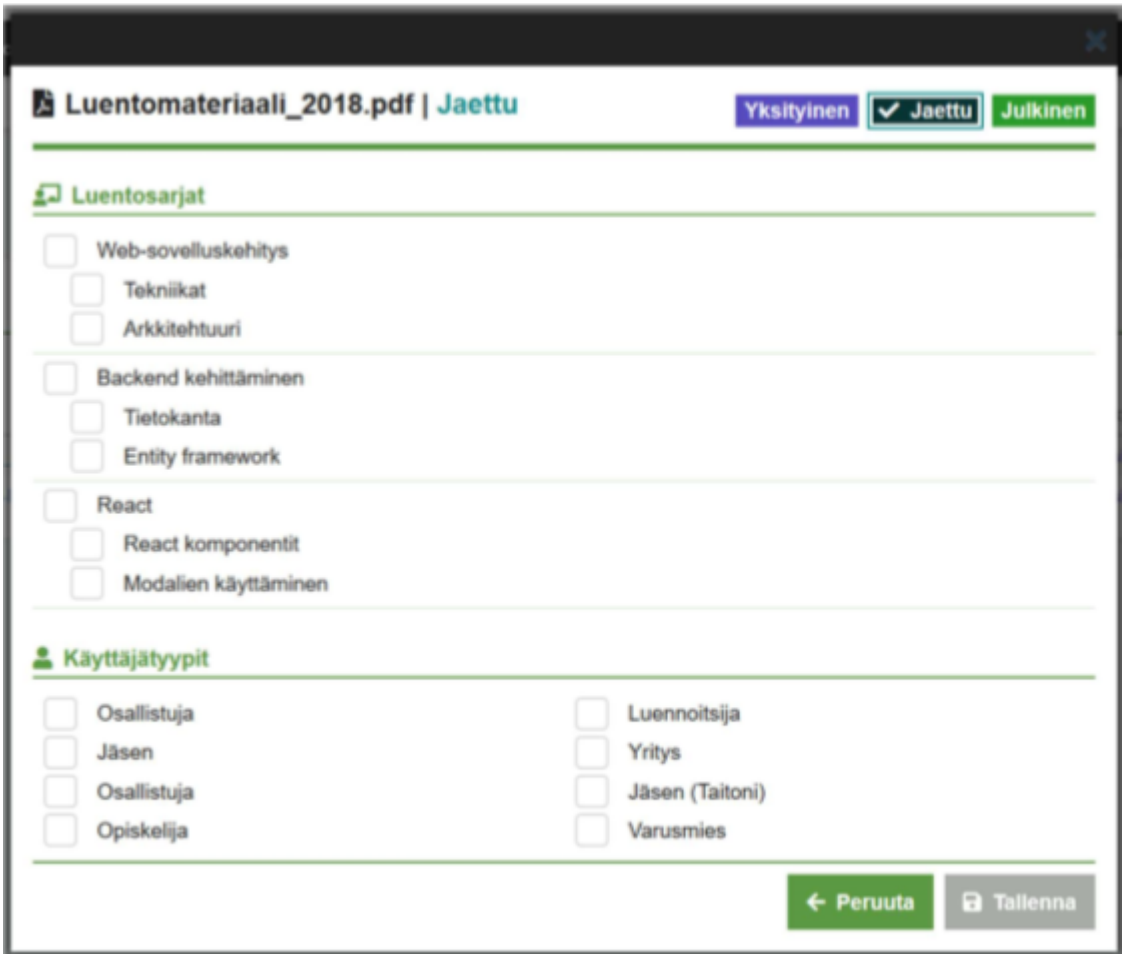
Kuva 17 Tiedoston näkyvyys julkisena (Martiskainen, Jere. 2019)

Kuvassa 18 on yksityisen tiedoston modal. Yksityinen tiedosto näkyy vain käyttäjälle itselleen.



Kuva 18 Tiedoston näkyvyys yksityisenä (Martiskainen, Jere. 2019)

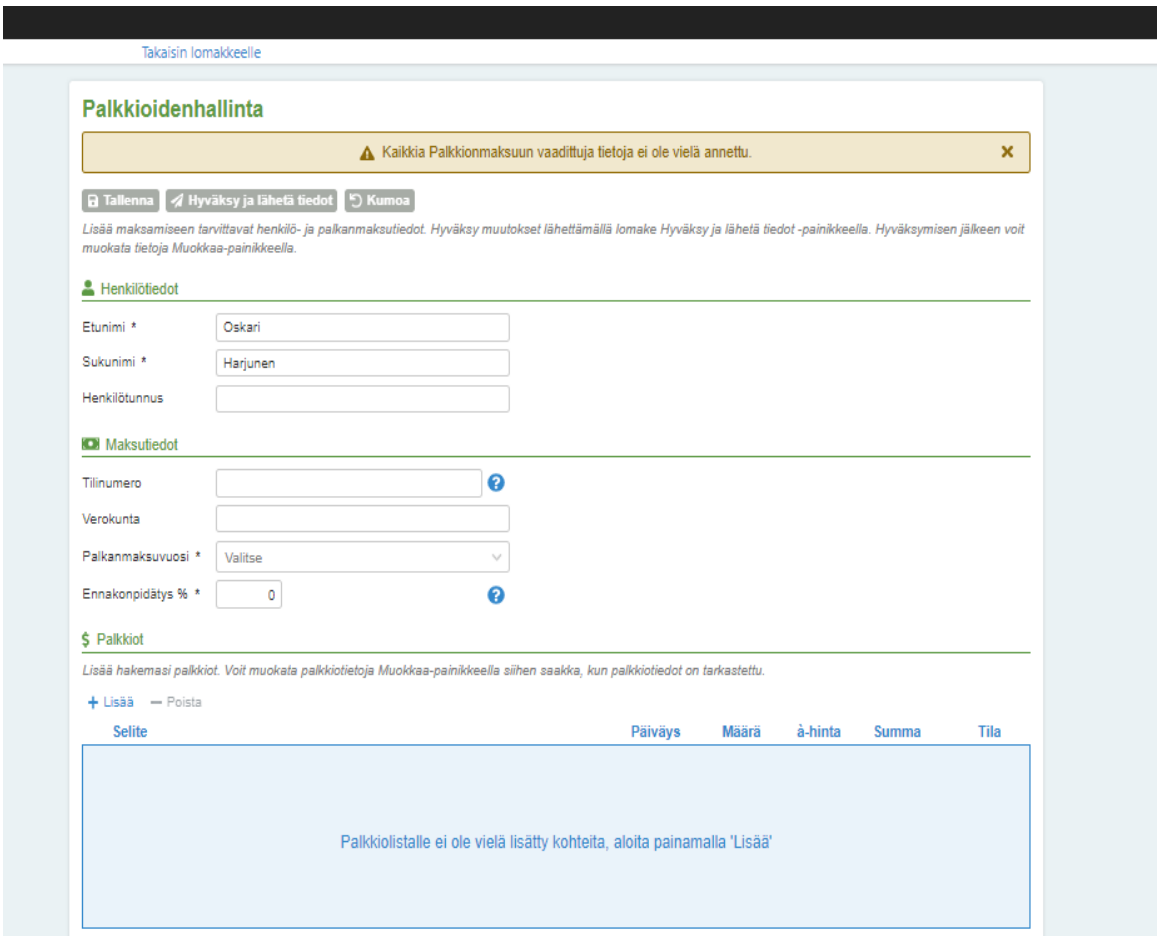
Kun tiedoston näkyvyys määritellään jaetuksi, voi käyttäjä kohdentaa tiedoston näkyvyyden halutuille rooleille, luentosarjoille, luennoille tai näiden kombinaatioille. Kuvassa 19 on jaetun tiedoston modal.



Kuva 19 Tiedoston näkyvyys jaettuna (Martiskainen, Jere. 2019)

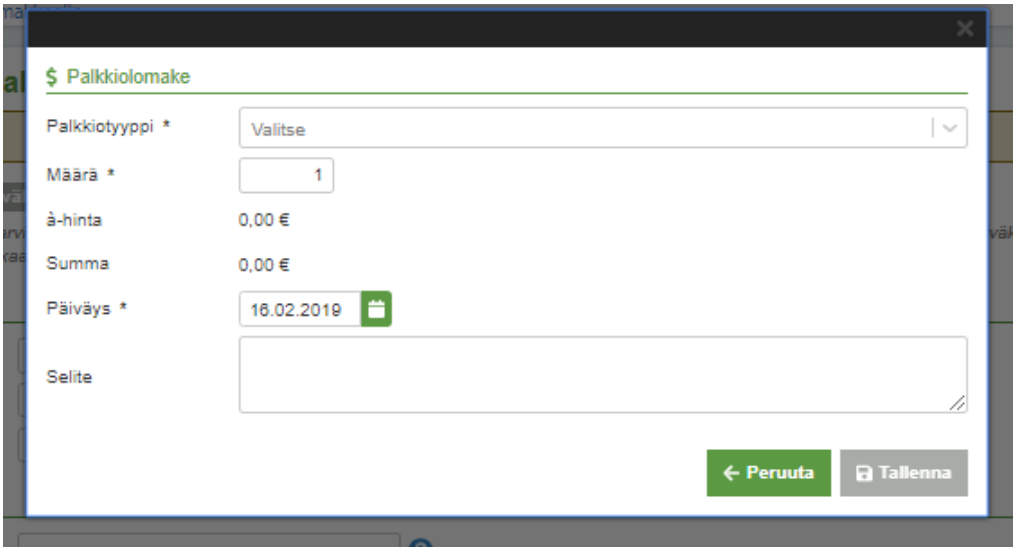
6.2 Palkkioidenhallinta

Palkkioidenhallinnassa käyttäjä pystyy hallinnoimaan omia maksutietojaan ja lisäämään palkkiohakemuksia. Vaaditut kentät on merkitty lisäämällä labelin yhteyteen * -merkintä. Palkkioidenhallinnasta toteutin henkilötiedot ja maksutiedot. Palkkoidenhallintaan siirryttäessä nähdään kuvan 20 mukainen sivu.



Kuva 20 Palkkioidenhallinta-sivu

Palkkioiden lisäyspyynnöt tapahtuvat Lisää-painiketta painamalla. Tällöin modal aukeaa kuvan 20 mukaisesti.



Kuva 21 Palkkiolomake-modal

7 YHTEENVETO

7.1 Tulokset

Projekti kehitti osaamista uusien tekniikoiden ja toimintatapojen sekä suunnittelumallien myötä erittäin paljon. React ja Angular vertailu antoi hyvän tuntemuksen nykyaikaisten käyttöliittymien koodauksesta. Kokemusta karttui paljon käyttöliittymä- ja palvelinkoodaamisen muodossa. Lisäksi tiimityöskentelytaidot kehittyivät kokonaisvaltaisen projektityöskentelyn ohessa.

Backend-koodaamisen suunnittelumallien käyttöönotto oli mielestäni haastavin toteutettava, mutta sitä opetellessa ja ohjelmoidessa huomattiin, että käytäntö on huomattavasti helpommin ylläpidettävämpi ja luettavampi ensimmäisen ohjelmointikerran jälkeen. Lopputuloksena React:lla luodut osat saatiin onnistuneesti integroitua vanhaan projektiin.

7.2 Jatkokehitys

TDD eli Test Driven Development ja siihen liittyvät yksikkötestit sekä SpecFlow-testit jäivät toteuttamatta, mutta näitä pyritään ottamaan käyttöön jatkossa. Itse luennoitsijaportaalia on tarkoitus vielä kehittää ja tuoda siihen sivukokonaisuuksia, kuten järjestäjän tiedotteet, luento- ja ohjelmaehdotukset ja henkilötietojen ylläpito. React:in integrointi MVC-projektiin oli osittain haasteellista, mutta sen tuoma hyöty on ollut kuitenkin osoittautunut niin käteväksi, että sillä on kehitetty Data Prisman muihinkin projekteihin osia.

LÄHTEET

Microsoft Visual Studio. [viitattu 2018-12-03]. Saatavissa:

https://docs.microsoft.com/en-us/visualstudio/opbuildpdf/toc.pdf?view=vs-2017&branch=live

Microsoft SQL Management Studio. [viitattu 2018-12-03]. Saatavissa:

https://docs.microsoft.com/en-us/sql/opbuildpdf/temp2016a/ssms/toc.pdf?view=sql-server-2017&branch=live

Visual Studio Code. [viitattu 2018-12-03]. Saatavissa:

https://code.visualstudio.com/

Git. [viitattu 2018-12-03]. Saatavissa:
https://git-scm.com/

W3Schools. What is SQL. [viitattu 2018-12-04]. Saatavissa:
https://www.w3schools.com/whatis/whatis_sql.asp

W3Schools. JSON. [viitattu 2018-12-04]. Saatavissa:
https://www.w3schools.com/js/js_json_intro.asp

Microsoft. Introduction to the CSharp language and the NET framework. [viitattu 2018-12-04]. Saatavissa:
https://docs.microsoft.com/en-us/dotnet/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework

JSON. [viitattu 2018-12-04]. Saatavissa:
https://www.json.org/

W3Schools. What is JavaScript. [viitattu 2018-12-04]. Saatavissa:
https://www.w3schools.com/whatis/whatis_js.asp

npm. [viitattu 2018-12-04]. Saatavissa:
https://docs.npmjs.com/about-npm/

Google. Angular. [viitattu 2018-12-04]. Saatavissa:
https://angular.io/docs
https://angular.io/guide/architecture

Facebook. React. [viitattu 2018-12-04]. Saatavissa:
https://reactjs.org/tutorial/tutorial.html
https://reactjs.org/
https://github.com/facebook/react

SIMONS, E. What exactly is React. [viitattu 2018-12-04]. Saatavissa:
https://thinkster.io/tutorials/what-exactly-is-react

NEUMANN, J. ECMAScript. [viitattu 2018-12-07]. Saatavissa:
https://www.ecma-international.org/activities/General/presentingecma.pdf

W3Schools. JavaScript Versions. [viitattu 2018-12-07]. Saatavissa:
https://www.w3schools.com/js/js_versions.asp

Facebook. JSX. [viitattu 2018-12-07]. Saatavissa:
https://facebook.github.io/jsx/

Evan You. Meet the Team [viitattu 2018-12-10]. Saatavissa:
https://vuejs.org/

Data Prisma Oy. Data Prisma Oy. [viitattu 2019-02-09].
http://www.dataprisma.fi/fin/info.html

Martiskainen, Jere. Kongressin luennoitsijaportaalin kehitys. [viitattu 2019-02-16] Saatavissa: https://www.theseus.fi/bitstream/handle/10024/159049/Martiskainen_Jere.pdf?sequence=1&isAllowed=y