SAVONIA

OPINNÄYTETYÖ - **AMMATTIKORKEAKOULUTUTKINTO**
TEKNIIKAN JA LIIKENTEEN ALA

KONGRESSIN LUENNOITSIJAPORTAALIN KEHITYS

TEKIJÄ: Jere Martiskainen

SAVONIA-AMMATTIKORKEAKOULU

OPINNÄYTETYÖ
Tiivistelmä

Koulutusala Tekniikan ja liikenteen ala	
Koulutusohjelma/Tutkinto-ohjelma Sähkötekniikan tutkinto-ohjelma	
Työn tekijä(t) Jere Martiskainen	
Työn nimi Kongressin luennoitsijaportaalin kehitys	
Päiväys 17.12.2018	Sivumäärä/Liitteet 39
Ohjaaja(t) Lehtori Keijo Kuosmanen, lehtori Jussi Koistinen	
Toimeksiantaja/Yhteistyökumppani(t) Data Prisma Oy	

Tiivistelmä

Opinnäytetyö tehtiin yhteistyössä Data Prisma Oy:n kanssa. Opinnäytetyössä oli kaksi tavoitetta. Ensimmäisenä tavoitteena oli tutkia ja vertailla uusimpia front-end-kehitystekniikoita. Toinen tavoite oli tuottaa Luennoitsijaportaalin ominaisuudet uudelleen uudella käyttöliittymätekniikalla ja integroida ne osaksi Kongressi-järjestelmää.

Tutkittaviksi tekniikoiksi valikoituivat Angular ja React. Molemmilla tekniikoilla tehtiin POC:t ja selvitettiin, kumpi tekniikoista soveltuu paremmin Luennoitsijaportaalin ominaisuuksien tuottamiseen. Vertailussa kävi ilmi, että tekniikat ovat hyvin samankaltaisia ja kumpikin täytti valittavalle tekniikalle asetetut vaatimukset. React valittiin käytettäväksi tekniikaksi, koska sen oppisi nopeammin ja yhdellä tiimimme jäsenistä oli jo entuudestaan huomattavaa kokemusta siitä. Näkymien uudelleenkehittämisprosessi aloitettiin suunnittelulla. Kun näkymien layoutit ja toiminnallisuudet oli suunniteltu, näkymät tuotettiin Reactilla. Valmiit näkymäkomponentit integroitiin ASP.NET MVC projektin Razor-näkymiin. Sovelluskehityksessä käytettiin Visual Studio 2017, Visual Studio Codea, Reactia, ASP.NET MVC ja .NET Frameworkia.

Lopputuloksena saatiin tärkeimmät Luennoitsijaportaalin ominaisuudet tuotettua ja integroitua ne osaksi Kongressi-järjestelmää onnistuneesti. Osa ominaisuuksista on vielä suunnitteilla ja ne tullaan toteuttamaan jatkokehitysvaiheessa.

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

SAVONIA UNIVERSITY OF APPLIED SCIENCES

THESIS
Abstract

Field of Study Technology, Communication and Transport			
Degree Programme Degree Programme in Electrical Engineering			
Author(s) Jere Martiskainen			
Title of Thesis Development of Lecturer Portal of Kongressi			
Date	17 December 2018	Pages/Appendices	39
Supervisor(s) Mr Keijo Kuosmanen, Senior Lecturer, Mr Jussi Koistinen, Senior Lecturer			
Client Organisation /Partners Data Prisma Ltd			

Abstract

The thesis was carried out in cooperation with Data Prisma Ltd. The thesis had two goals. The first goal was to study and compare latest front-end development technologies. The second goal was to remake Lecturer Portal features with new user interface technology and integrate them into the Kongressi system.

Angular and React were selected to be investigated. Proofs of concept were made with both technologies and it was studied which one fits better for producing the features of Lecturer Portal. In comparison it turned out that both technologies are quite similar and both met the requirements. React was voted in because it would be easier to learn, and one member of the team had significant experience from it. The remaking process of the views was started by planning. After the layouts and functionalities of views were planned, they were produced with React. Complete user interface components were integrated into the Razor views of the ASP.NET MVC project. Features were developed by using Visual Studio 2017, Visual Studio Code, React, ASP.NET MVC and .NET Framework.

As a result, the most important features of Lecturer Portal were remade, and they were integrated into the Kongressi system successfully. Some of the features are still in the planning stage and they will be implemented in the upcoming development.

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

ESIPUHE

Haluan kiittää työnantajaani Data Prisma Oy:tä siitä, että he tarjosivat ohjelmistoprojektin, josta minun oli mahdollista tehdä opinnäytetyöni. Haluan osoittaa erityiskiitokset ohjaajalleni Jukka-Pekka Kurjelle Data Prismalta tuesta ja ohjauksesta opinnäytetyössä. Lisäksi haluan kiittää ohjaajaani Keijo Kuosmasta ja Jussi Koistista, Savonia Ammattikorkeakoululta, ohjeista ja opastuksesta opinnäytetyön eri vaiheissa.

Kuopiossa 17.12.2018
Jere Martiskainen

SISÄLTÖ

1 JOHDANTO 7
1.1 Työn tarkoitus 7
1.2 Määritelmät ja lyhenteet 8
2 DATA PRISMA OY 9
2.1 Kongressi 9
3 OHJELMISTOT JA TEKNIIKAT 10
3.1 Kehitysympäristöt 10
3.1.1 Visual Studio 2017 10
3.1.2 Visual Studio Code 10
3.1.3 Microsoft SQL Server Management Studio 11
3.2 Tekniikat 11
3.2.1 React lyhyesti 11
3.2.2 Angular lyhyesti 12
3.2.3 C# 14
3.3 Tietokanta 14
3.3.1 Microsoft SQL Server 14
3.4 Versionhallinta 14
3.4.1 Team Foundation Server 14
3.4.2 Git 15
4 REACT, ANGULAR VERTAILU 16
4.1 Modalit 17
4.2 Uusiokäyttöiset komponentit 17
4.3 Live-validaatio 18
4.4 Debuggaus 18
4.5 Lomaketoimintojen hallinta (disabled/enabled) 18
4.6 Kielisyydet 18
4.7 Integrointi 19
4.8 Yhteenveto 20
5 SOVELLUS 22
5.1 React-projekti 22
5.2 React-näkymien tuottaminen 23

5.3 Kongressi 23
5.4 React-näkymien integrointi 24
5.5 Client-palvelin 24
5.6 BL-palvelin 26
6 LUENNOITSIJAPORTAALI 28
6.1 Materiaalit 28
6.2 Palkkiot 32
6.3 Tapahtuman tiedostojen hallintanäkymä 33
7 YHTEENVETO 36
7.1 Jatkokehitys 36
7.2 Työn tulokset 36
LÄHTEET JA VIITTAUKSET 38

1 JOHDANTO

Opinnäytetyössä käsitellään web-sovelluskehityksen tekniikoita sekä kerrotaan Luennoitsijaportaalin uudelleen kehityksen vaiheista. Luennoitsijaportaalin kehittämisen valinta opinnäytetyön aiheeksi tuntui luonnolliselta, koska oli kiinnostavaa päästä tutustumaan uusiin web-sovellusten kehitystekniikoihin ja sitä kautta syventää osaamista sovelluskehittäjänä. Projekti päätettiin tehdä tiimissä, johon kuului lisäkseni mm. luokkakaverini ja kollegani Oskari Harjunen. Hän myös teki opinnäytetyönsä samasta projektista.

Opinnäytetyön alussa kerrotaan kehitysympäristöistä ja tekniikoista, joita projektissa käytettiin. Seuraavaksi kerrotaan tekniikkavertailun tuloksista ja selitetään, miksi React valittiin käytettäväksi tekniikaksi. Esitellään sovellusta ja sen arkkitehtuuria. Esitellään näkymät, joita projektin aikana tuotettiin ja kerrotaan miten ne toimivat. Lopuksi kerrotaan projektin jatkokehityssuunnitelmista ja tuloksista.

1.1 Työn tarkoitus

Tämän opinnäytetyön tarkoituksena on tutkia uusimpia web-sovellusten käyttöliittymien kehittämiseen tarkoitettuja tekniikoita sekä kehittää Kongressi-ohjelmiston yhteyteen Luennoitsijaportaalin toiminnallisuus uudella tekniikalla. Luennoitsijaportaalista on olemassa aiempi versio, joka ei kuitenkaan enää vastannut tämän päivän tarpeisiin. Toiminnallisuudet haluttiin tuottaa uudelleen uudella tekniikalla niin, että näkymät olisivat responsiivisia ja helppokäyttöisiä mobiililaitteilla selattaessa. Luennoitsijaportaali haluttiin osaksi uutta Kongressia. Luennoitsijaportaaliin kuuluu näkymiä, joissa luennoitsijat voivat mm. hallinnoida luentoja, luoda luentoehdotuksia, ylläpitää tietojaan, hakea palkkioita ja ladata luentomateriaaleja. Opinnäytetyön varsinaiseen työosuuteen kuului siis näkymien konkreettinen tuottaminen ja teoriaosuuteen tekniikoiden tutkiminen ja vertaileminen.

Luennoitsijaportaalin käyttöliittymäosien tuottamiseen haluttiin käyttää jotain uutta, kehittyvää ja nykyaikaista tekniikkaa. Mahdollisia vaihtoehtoja tutkittiin ja kaksi tekniikkaa, jotka nousivat ylitse muiden, olivat Javascript tekniikat React ja Angular. Päätettiin tehdä POC:it, eli proof of conceptit, molemmilla tekniikoilla ja selvittää kumpi tekniikoista soveltuisi paremmin käytettäväksi näkymien tuottamiseen. Valittavalle tekniikalle asetettiin tiettyjä vaatimuksia. Ajatuksena oli, että jos valittu tekniikka osoittautuu tehokkaaksi ja pystyy korvaamaan aiemmin käytettyä Razor-tekniikkaa, sitä käytettäisiin jatkossa muidenkin näkymien tuottamiseen.

1.2 Määritelmät ja lyhenteet

POC (Proof of concept)

- Soveltuvuusselvitys

TFS (Team Foundation Server)

- Microsoftin kehittämä versionhallintajärjestelmä

C# (C Sharp)

- Tyypitetty olio-ohjelmointikieli

JSX (JavaScript XML)

- JavaScriptin syntaksilaajennus

BL (Business Logic)

- Business-logiikka

SPA (Single page application)

- Yhden sivun sovellus

REST (Representational State Transfer)

- HTTP-protokollaan perustuva arkkitehtuurimalli ohjelmointirajapintojen toteuttamiseen

JSON (JavaScript Object Notation)

- Yksinkertainen avoimen standardin tiedostomuoto tiedonvälitykseen

MVC (Model-View-Controller)

- Ohjelmistoarkkitehtuuri, jonka tarkoituksena on käyttöliittymän erottaminen sovelluslogiikasta

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.”
(Data Prisma Oy, 2017)

KUVA 1. Data Prisma Oy:n logo (DP Logo)

Data Prisman konttori sijaitsee Kuopion keskustassa. Yritys työllistää noin pari kymmentä henkeä, joista noin puolet työskentelevät sovelluskehityksen parissa. Data Prisman tuotteita ovat mm. Kongressi ja Visio. Kongressi on tapahtumien järjestämiseen ja ilmoittautumisten vastaanottamiseen tarkoitettu sovellus. Sillä voidaan hoitaa myös laskutus sekä asiakashallinta. Visio on palvelutelevision hallinta-, markkinointi- ja laskutusohjelmisto.

2.1 Kongressi

”Kongressi on järjestelmä erilaisten tapahtumien osanottajien, luennoitsijoiden, tilojen ja talouden hallintaan.”
(Kongressi, 2017)

Kongressilla luodaan tapahtuma, johon osallistujat voivat ilmoittautua ilmoittautumislomakkeiden kautta. Tapahtumat voivat sisältää luentosarjoja, jotka pitävät sisällään yksittäisiä luentoja. Kun henkilö on ilmoittautunut tapahtumaan, hän voi käydä tarkistamassa ja muokkaamassa omia henkilötietojaan Portaalissa, johon hänen on kirjauduttava saamillaan tunnuksilla. Opinnäytetyön aiheena oleva luennoitsijaportaali on joukko näkymiä Portaalissa, joiden avulla luennoitsijaroolinen henkilö voi hakea palkkioita, ilmoittaa palkkiomaksutietonsa ja lisätä tapahtumaan luentomateriaaleja.

3 OHJELMISTOT JA TEKNIIKAT

3.1 Kehitysympäristöt

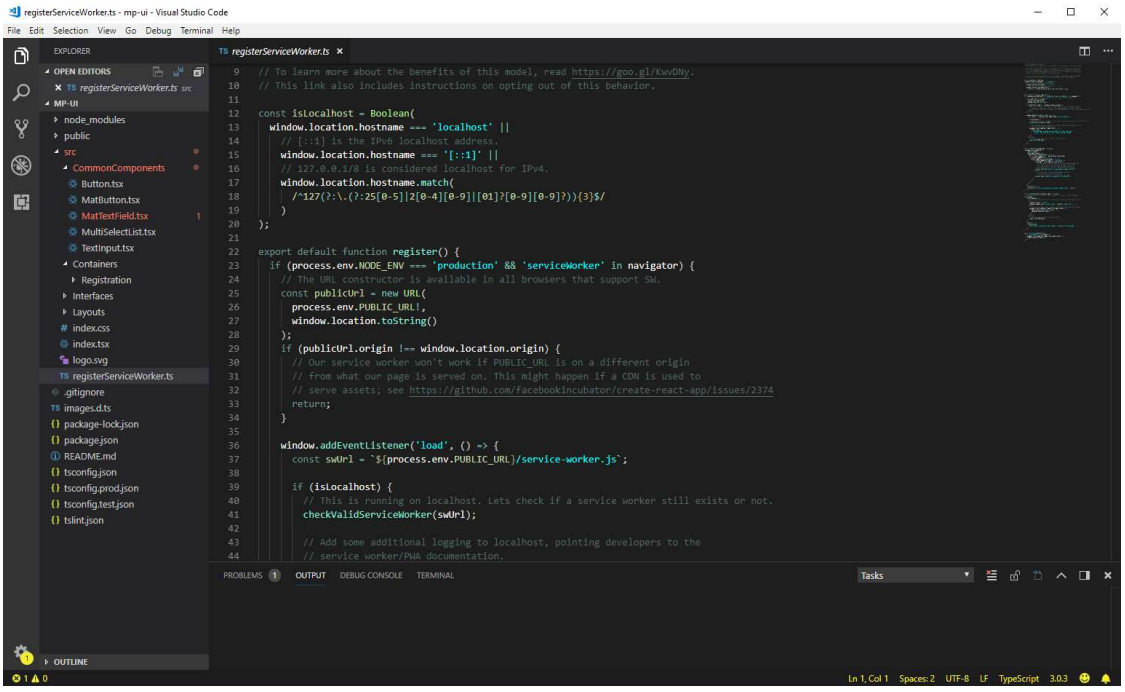
3.1.1 Visual Studio 2017

Pääasiallisena kehitysympäristönä toimi Microsoftin Visual Studio 2017, jolla kirjoitettiin sovelluksen logiikkakoodi. Visual Studio oli jo entuudestaan tuttu kehitysympäristö aiemmista työ-, harrastus- sekä kouluprojekteista. Se on helppokäyttöinen, vakaa ja monipuolinen kehitysympäristö hyvin monenlaiseen sovelluskehitykseen.

Microsoft Visual Studio on Microsoftin kehittämä ohjelmistokehitysympäristö. Sitä käytetään tietokoneohjelmien, webbisivujen, web-palveluiden sekä mobiilisovellusten kehittämiseen. Sen tärkeimpiä ominaisuuksia ovat monipuolinen koodieditori, IntelliSense (ennakoiva tekstinsyöttö), sekä hyvät debuggaus-työkalut. IntelliSense on älykäs koodin täydentäjä, joka helpottaa ja nopeuttaa ohjelmointia vähentämällä kirjoitusvirheiden määrää sekä tarjoamalla ennakoivaa tekstin syöttöä. (Microsoft Visual Studio, 2018)

3.1.2 Visual Studio Code

Visual Studio Code on Microsoftin kehittämä ilmainen lähdekoodieditori, jota voi käyttää Windows:lla, Linuxilla sekä MacOS:llä. Se on kevyempi ja muokattavampi kuin perinteinen Visual Studio. (Visual Studio Code, 2018)



KUVA 2. Visual Studio Code kuvankaappaus

Visual Studio Codea käytettiin käyttöliittymäkomponenttien kehitykseen. Code valittiin React-koodin kirjoittamiseen, koska siinä on hyvä tuki Reactille sekä Node.js:lle. React sovelluksen ajaminen Node.js ympäristössä mahdollistaa käyttöliittymäkomponenttien muutosten välittömän tarkastelun selainnäkymässä. Kun koodia muutetaan editorissa, riittää, että tiedosto tallennetaan niin kääntäjä kääntää koodista automaattisesti uuden version, joka päivittyy selainnäkymään. Lisäksi Code on helppokäyttöinen ja siihen on ladattavissa paljon koodaamista helpottavia laajennuksia, kuten mm. Linter ja Prettier.

3.1.3 Microsoft SQL Server Management Studio

Microsoft SQL Server Management Studio käytettiin tietokantakomentojen ajamiseen ja tietokannan muokkaamiseen.

SQL Server Management Studio on Microsoftin kehittämä sovellus SQL serverin konfigurointiin ja hallintaan. Sillä voi suorittaa kantahakuja SQL palvelimille sekä muokata niiden tietokantoja, tauluja sekä dataa. Se sisältää skriptieditorit sekä graafiset työkalut objektien ja serverin ominaisuuksien työstämiseen.
(SQL Server Management Studio, 2018)

3.2 Tekniikat

Tekniikat-osiossa kerrotaan pintapuolisesti tekniikoista, joita käytettiin projektin eri osa-alueiden kehittämisessä. Avataan myös hieman Angularia, joka oli mukana front-end-tekniikka vertailussa. Angularista ja Reactista enemmän niiden vertailuosiossa.

3.2.1 React lyhyesti

React on JavaScript-kirjasto käyttöliittymäkomponenttien tuottamiseen. Se on tarkoitettu pääasiassa SPA- (single page application) sekä mobiilisovellusten kehittämiseen. Reactin loi Facebookin kehittäjä Jordan Walke. Se otettiin ensimmäisenä käyttöön Facebookin uutissyötteessä vuonna 2011 ja myöhemmin Instagramissa 2012. Reactia ylläpitää ja kehittää Facebook sekä laaja yhteisö, joka koostuu niin yrityksistä kuin itsenäisistä sovelluskehittäjistä.
(React, 2018)

```
function getGreeting(user) {
  if (user) {
    return <h1>Hello, {formatName(user)}!</h1>;
  }
  return <h1>Hello, Stranger.</h1>;
}
```

KUVA 3. Esimerkki JSX-koodista (Introducing JSX, 2018)

Reactia voidaan kirjoittaa JSX:llä, joka on JavaScriptin syntaksilaajennus. Se sisältää Javascriptin ominaisuudet ja sillä voidaan kuvailla miltä React sovelluksen käyttöliittymäkomponentit näyttävät ja miten ne toimivat. JSX on eräänlainen sekoitus HTML ja JavaScriptiä. Sivuun liittyvä logiikka voidaan kirjoittaa JavaScriptillä tai esimerkiksi tyypitetyllä TypeScriptillä. React itsessään ei sisällä työkaluja suorittamaan tiettyjä web-sovellukselta vaadittuja tehtäviä, joten sovellukseen joudutaan tuomaan kirjastoja mm. tilanhallintaa ja API:en kanssa kommunikointia varten. Komponenttien käyttämät API kutsut on järkevää kirjoittaa Service luokkiin, jotka sitten tarvittaessa importataan komponenteille.

React-projektissa ei ole varsinaista arkkitehtuuria, joten on ohjelmoijan vastuulla osata järjestää sovelluksen sisältö järkevästi. Reactilla tuotetaan oikeastaan vain näkymäkomponentteja. Kaikki niiden tarvitsema toiminnallisuus toteutetaan sovellukseen lisättävillä kirjastoilla.

3.2.2 Angular lyhyesti

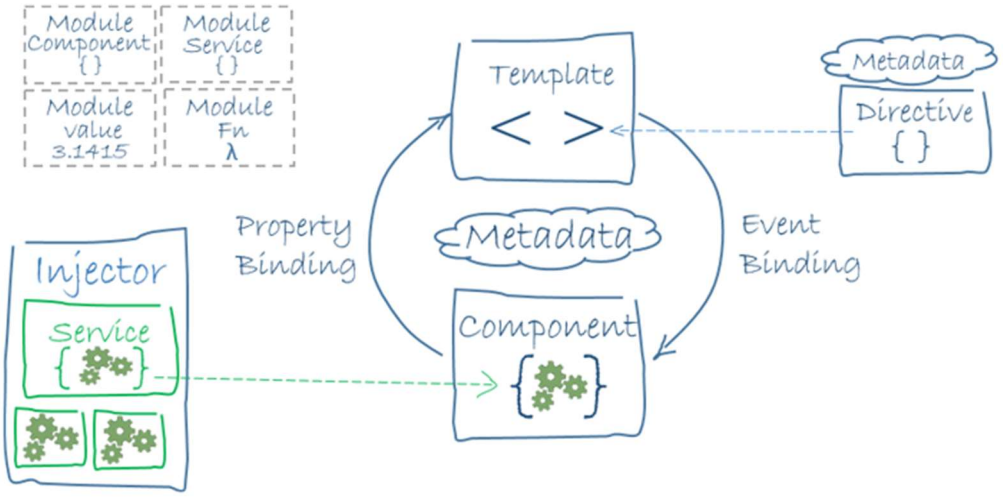
Angular on Googlen kehittämä TypeScriptiä käyttävä avoimen lähdekoodin web-sovelluskehitysalusta. Angular on uudelleen kirjoitettu sitä edeltäneestä AngularJS:stä.
(Angular, 2018)

```
import { Component, OnInit, Input } from '@angular/core';
import { Hero } from '../hero';
import { ActivatedRoute } from '@angular/router';
import { Location } from '@angular/common';
import { HeroService } from '../hero.service';

@Component({
  selector: 'app-hero-detail',
  templateUrl: './hero-detail.component.html',
  styleUrls: ['./hero-detail.component.css']
})

export class HeroDetailComponent implements OnInit {
  @Input() hero: Hero;

  constructor(private route: ActivatedRoute,
              private heroService: HeroService,
              private location: Location) { }
  ngOnInit() {
    this.getHero();
  }
  getHero(): void {
    const id = +this.route.snapshot.paramMap.get('id');
    this.heroService.getHero(id)
      .subscribe(hero => this.hero = hero);
  }
  goBack(): void {
    this.location.back();
  }
}

```

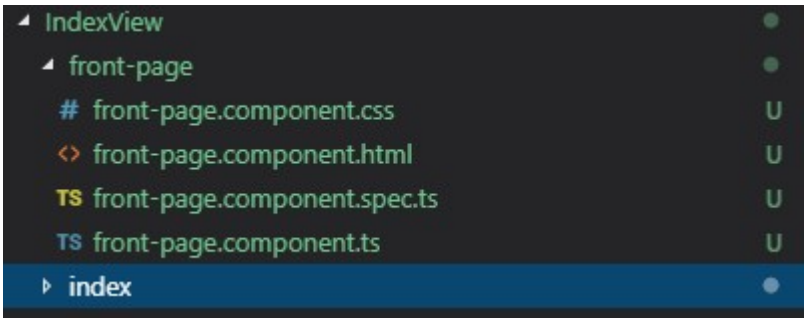
KUVA 4. Esimerkki Angular komponentin luokasta

Angularissa sovellus koostuu moduuleista ja komponenteista. Sovellus pitää sisällään ainakin yhden moduulin, joka pitää sisällään komponentit, palvelut ja muut koodit. Komponentti määrittelee näkymän. Komponentin sovelluslogiikka kirjoitetaan luokkaan, joka toimii yhdessä html-templaten, komponentin näkyvän osan, kanssa. Komponentit käyttävät palveluita, jotka eivät ole suoraan kytköksissä näkymiin. Palvelut injektoidaan komponentteihin riippuvuuksina, jotta koodi olisi modulaarista

sekä uudelleenkäytettävää. Komponentit ja palvelut ovat luokkia, joille annettaan tarvittava metadata ja tyypitys. Metadata yhdistää komponenttiluokan templateen. Templatesta muodostetaan näkymä käyttäen HTML koodia ja Angular muotoiluja.
(Angular Architecture, ei pvm)



KUVA 5. Angular sovelluksen osat. (Angular Architecture, ei pvm)



KUVA 6. Angular komponentin tiedostot

```
1  <button (click)="goBack()">go back</button>
2  <div *ngIf="hero">
3    <h2>{{ hero.name | uppercase }} Details</h2>
4    <div><span>id: </span>{{ hero.id }}</div>
5    <div>
6      <label>name:
7        <input [(ngModel)]="hero.name" placeholder="name">
8      </label>
9    </div>
10 </div>
```

KUVA 7. Esimerkki Angular komponentin template-koodista

Angularin arkkitehtuuri ja ominaisuudet ovat hyvin kuvattu keskitetyssä dokumentaatiossa. Keskitetty dokumentaatio helpotti suuresti Angularin parissa työskentelyä. Erillisen templaten ja komponentin luokan ansiosta logiikka oli helppo pitää erillään näkymästä. Komponentilla oli näin selkeä

rakenne. API kutsut saatiin omaan kerrokseen Service luokkien avulla. Service tasolta pystyi kommunikoimaan esimerkiksi REST-pyynnöillä palvelimen kanssa. Arkkitehtuurissa oli siis eroteltu näkymät (template), logiikat (luokka), modelit ja servicet.

3.2.3 C#

C# (C Sharp) on Microsoftin kehittämä ohjelmointikieli, jonka ominaispiirteitä ovat vahva tyypitys, oliomaisuus ja luokkarakenteet. C#:ssa yhdistyy ominaisuuksia C:stä, C++:sta ja Javasta.
(C Sharp, 2015)

Suurin osa Luennoitsijaportaalin komponenttien logiikkakoodista kirjoitettiin C# -ohjelmointi kielellä. Itse henkilökohtaisesti pidän C# ohjelmoinnista juuri sen selkeän syntaksin takia. C# opetteleminen on helppoa, mutta myös haastavaa siinä mielessä, että siitä löytää jatkuvasti uutta opittavaa. Työskentely C#:lla on hyvin palkitsevaa, kun jatkuvasti huomaa kehittyvänsä ja oppii parempia tapoja tuottaa parempaa koodia.

3.3 Tietokanta

3.3.1 Microsoft SQL Server

Sovelluksen tarvitsema tietokanta on toteutettu Microsoft SQL Serverillä.

Microsoft SQL Server on Microsoftin kehittämä relaatiotietokannan hallintajärjestelmä. Se ylläpitää tietokantaa, josta voidaan toisilla sovelluksilla pyytää tietoa tai tallentaa sinne tietoa. Nämä sovellukset voivat sijaita joko samalla tai eri koneella kuin itse tietokanta.
(Microsoft SQL Server, 2018)

3.4 Versionhallinta

Versionhallintajärjestelmät auttavat kehittäjiä tallentamaan muutokset tiedostoissa ja projektissa ja jakamaan ne niin, että jokaisella kehittäjällä on tarvittaessa uusimmat versiot niistä. Kehittäjien on myös mahdollista tarvittaessa ottaa tiedostoista vanhempi versio takaisin. Myös koko projekti on versionhallintajärjestelmällä mahdollista palauttaa takaisin vanhaan tilaansa, jos jotain menee pieleen. Järjestelmän avulla on mahdollista seurata, kuka teki minkäkin muutoksen ja milloin. Versionhallinta mahdollistaa sotkettujen tai menetettyjen tiedostojen palauttamisen helposti takaisin.
(Versionhallinnasta, ei pvm)

3.4.1 Team Foundation Server

ASP.NET ja .NET Framework projektien sekä taskien hallinnoimiseen käytettiin Microsoftin Team Foundation Serveriä yhdessä Team Foundation Version Controllin kanssa.

Team Foundation Server on Microsoftin kehittämä lähdekoodien hallintatyökalu, jota voi käyttää joko Gitin tai Team Foundation Version Controllin kanssa. TFS tarjoaa työkalut mm. myös projektin ja julkaisun hallintaan, automaattikäännöksiin, testaukseen.
(Team Foundation Server, 2005)

3.4.2 Git

Git:iä käytettiin versionhallintaan React-komponenttien osalta. Syy tähän oli Git:n helppokäyttöisyys ja käyttöönoton vaivattomuus Visual Studio Codessa, jota käytettiin React-komponenttien kehittämiseen.

Git on hajautettu versionhallintajärjestelmä (SCM), jota käytetään ohjelmistokehityksessä lähdekoodien hallintaan. Gitin kehitti Linus Torvalds vuonna 2005 helpottamaan Linuxin kehitystä. Git eroaa muista versionhallintajärjestelmistä siinä, että jokaisella käyttäjällä on käytössään paikallinen kopio versionhallinnasta. Tämä tarkoittaa sitä, että lähdekoodin lisäykset ja muokkaukset tehdään lokaalisti omaan kopioon ja liitetään tarvittaessa pääasialliseen versionhallintajärjestelmään.
(Git, 2018)

Eräs Gitin ominaisuus on sen kehityshaarojen (branch) käyttämisen helppous. Ajatuksena kehityshaarassa on, että isoimmista muutoksista on mahdollista tehdä oma kehityshaaransa. Tätä voidaan kehittää erillään toisista kehityshaaroista ja vasta kun kehityshaara on vakaa ja siinä on halutut ominaisuudet, voidaan se liittää varsinaiseen päähaaraan mergellä (merge).
(Git, 2018)

4 REACT, ANGULAR VERTAILU

Projektiin valittiin kaksi sillä hetkellä suosituinta front-end-kehitystekniikka. Haluttiin tutustua molempiin tekniikoihin ja selvittää kumpi tekniikoista soveltuisi projektiin parhaiten. Molemmilla tekniikoilla päätettiin tehdä POC:t, proof of conceptit, jotka sisälsivät yksinkertaisen henkilötietolomakkeen. Näin saatiin selvitettyä, onko tekniikoilla vaadittavia ominaisuuksia ja miten ne soveltuisivat Luennoitsijaportaalin näkymien tuottamiseen. Samalla havainnoitiin, miten helppo on tekniikkaan päästä sisälle ja tutkia tekniikan ominaisuuksia. Ensivaikutelmat Angularista olivat positiiviset. Toimintaperiaatteet ja arkkitehtuuri olivat hyvin dokumentoidut. Sovelluksen kannalta hyvin olennaisiin asioihin kuten reititykseen sekä API kutsuihin oli valmiit työkalut.

Tekniikoilla on muutama hyvin selkeä eroavaisuus. React on JavaScript-kirjasto, kun taas Angular on framework. Näiden eroa voidaan selittää sillä, että Angular sisältää valmiiksi paljon työkaluja millä saadaan tuotettua web-sovelluksessa tarvittavia toimintoja, kuten REST-rajapintakutsuja, reititystä ja lomakkeiden hallintaa. Reactin mukana ei tule samoja työkaluja vaan kehittäjä joutuu etsimään ja liittämään projektiin tarvittavat kirjastot. Tämä on kehittäjälle vapaus, mutta myös rasite.

Angular ja React ovat hyvin samankaltaisia. Molemmat tekniikat ovat ns. component-based, eli komponenttipohjaisia. Tämä tarkoittaa sitä, että sovelluksen näkymät koostuvat komponenteista. Komponenttien sisälle on mahdollista renderöidä toisia komponentteja. Molemmat tekniikat tuottavat käännöksessä minifioidun JavaScript-tiedoston. Tämä JavaScript-tiedosto on mahdollista liittää johonkin olemassa olevaan HTML-sivuun tai ajaa se omalla HTML-sivullaan.

Technology	Angular	React
Technology type	Component-Based Framework using Typescript	User Interface Library with a component-based architecture using Javascript
Data binding	2-way data binding	1-way data binding
Size	Quite large and since it needs to be shipped to the client side, it increases the initial load time	Quite small in size, especially when compared with Angular
Learning Curve	Quite steep, given the number of features and options you have in Angular	It's easy to pick up and learn
Performance	Comparable to React, Angular 2 and 4 are some	Faster than Angular thanks to the Virtual DOM
Simplicity	Quite complex	Fairly simple but takes some time to set up a project and configure everything
Scalability	Easy to scale thanks to the power CLI and generation tools, It's also used by many large companies	Fairly easy to scale and is quite testable which facilitates the scaling procedure

KUVA 8. Reactin ja Angularin ominaisuuksia (Angular vs React, 2018)

Ennen kuin tekniikoiden vertailu aloitettiin, listattiin nykyisellä tekniikalla vastaan tulevia ongelmakohtia ja niihin haettiin ratkaisuja vertailussa olevilta tekniikoilta. Uudelta käyttöönotettavalta tekniikalta vaadittiin tiettyjä ominaisuuksia ja kykyä tuottaa tiettyjä toiminnallisuuksia.

4.1 Modalit

Eräs vaadittava ominaisuus oli, että tekniikalla pitäisi pystyä tuottamaan modaaleja, jotka ovat muokattavissa ilman turhaa koodin monistamista. Tällä tavoin päästäisiin eroon sekalaisista käytännöistä käyttää erilaisia modaleita, joita populoidaan eritavoin datalla. Projektissa haluttiin toteuttaa helppokäyttöinen modal-komponentti, jota voidaan käyttää missä vain tarvittaessa ja joka on hyvin parametrisoitavissa.

Modalien luonti onnistui kummallakin tekniikalla. Käytännössä luotiin modal-komponentti, jolle voitiin parametrina antaa modalissa näytettävä komponentti. Näin modal-komponentti on uusiokäyttöinen, eli samaa modal-komponenttia voidaan käyttää missä vain sovelluksen sisällä. Käyttäjälle voidaan näyttää dataa tai käyttäjä voi täyttää ja lähettää lomakkeen modalin kautta palvelimelle. Tämä ominaisuus pystyttiin tuottamaan molemmilla tekniikoilla, tosin Angularilla modal-komponentti vaati hieman enemmän koodia kuin Reactilla tehtäessä.

4.2 Uusiokäyttöiset komponentit

Tekniikalla haluttiin pystyä tuottamaan elementtirakenteet attribuutteineen helposti uusiokäyttöisiksi komponenteiksi ja tällä tavoin vähentää koodin monistusta. Koska molemmat tekniikat käyttävät komponentteja rakennuspalikoina, oli ilmiselvää, että tämä on sisäänrakennettu ominaisuus molemmissa tekniikoissa. Myös plugielementit (datepickerit, checkboxit, radiobuttonit) saatiin näin paketoitua helposti uusiokäyttöisiksi komponenteiksi, ja voitiin välttää monien toisiaan vastaavien toteutusten synty koodissa.

```
13  function Button(props) {
14    return (
15      <button className="fancy-button"
16                onClick={props.onClick}>
17        {props.text}
18      </button>
19    );
20  }
```

KUVA 9. Esimerkki Reactin stateless ("tyhmä") komponentista

Reactin vahvuudeksi nousi komponenttiajattelu, jossa komponentit ovat jaettu tyhmiin komponentteihin (stateless component) ja älykkäisiin komponentteihin (stateful component). Tyhmät komponentit eivät omista tilaa. Ne saavat tarvitsemansa parametrit attribuutteina, jotka näkyvät komponentin sisällä ns. props-objektina. Nämä ovat read-only arvoja, joita komponentti voi ainoastaan

lukea, mutta ei muokata. Esimerkiksi kuvan komponentti palauttaa painikkeen, johon sisältyy propseissa annettu teksti ja jota painamalla suoritetaan onClick-funktio.

Älykkäät komponenteilla on tila (state), jossa komponentti säilyttää sille kuuluvaa dataa. Kun komponentti alustetaan, voidaan sen tila alustaa ja komponentti voi muuttaa tilaansa sen elinkaaren aikana. Älykkäät komponentit ovat luokkia, jotka voivat käyttää lifecycle-metodeita, kuten ComponentDidMount, joka ajetaan joka kerta kun komponentti renderöidään.

4.3 Live-validaatio

Tekniikalla haluttiin pystyä tuottamaan yksinkertainen ja nopea live-validaatio. Live-validaatiolla tarkoitan, että lomakkeiden kentät alustavasti validoitaisiin näkymällä jo ennen kuin lomakkeen tiedot lähetettäisiin palvelimelle. Tällä saataisiin säästetty palvelimen ja käyttäjän aikaa. Angularissa lomakkeen validoinnin pystyi toteuttamaan Angularin oman validointiluokan avulla, josta löytyy validaatiot mm. min- ja max-arvoille, vaadituille kentille, email-osoitteelle, tekstien min- ja max-pituuksille ja patterneille. Validaatiot toimivat hyvin ja päivittyvät nopeasti käyttöliittymään.

Reactissa validaatio oli mahdollista toteuttaa monilla eri ulkoisilla kirjastoilla. Huomattiin, ettei validaation kanssa kumpikaan tekniikoista suoriutunut toista paremmin. Reactilla validaation toteuttaminen vaati ehkä hieman enemmän koodia, mutta toteutus oli paremmin kustomoitavissa.

4.4 Debuggaus

Molempia tekniikoiden koodia onnistuu debugata Chrome-selaimen laajennuksen avulla. Tämä tekee virheiden paikallistamisen koodista helpoksi. Käytännössä tämä onnistuu niin, että Chrome-selaimeen asennetaan laajennus Chrome Debugger. Tämän jälkeen debugger konfiguroidaan Visual Studio Codessa tekemällä tarvittavat muutokset launch.json tiedostoon. Debugger käynnistetään ja Visual Studio Code ajaa sovellusta debugger-tilassa, jolloin se pysähtyy breakpointtien kohdalla. Tämä ominaisuus löytyi samalla tavalla kummastakin tekniikasta.

4.5 Lomaketoimintojen hallinta (disabled/enabled)

Tekniikalta vaadittiin, että sillä tulisi olla mahdollista hallita helposti lomakkeiden ja sivujen sisältöä. Esimerkiksi jos halutaan piilottaa tiettyjä kenttiä tai valikoita. Tämä onnistui molemmissa tekniikoissa ehdollistamalla haluttujen osien renderöinti attribuuttien ja parametrien avulla. Tämä ominaisuus toteutui kummallakin tekniikalla yhtä hyvin.

4.6 Kielisyydet

Kielisyyksien hallinnan toimivuus on koko Kongressi sovelluksessa tärkeää. Sovellus tukee kolmea kieltä, suomea, ruotsia ja englantia. Tästä syystä Luennoitsijaportaalin näkymät täytyi ohjelmoida toimimaan näillä kolmella kielellä. Kielisyyden toteutusta kokeiltiin Angular POC:ssa sen omalla i18n

työkaluilla. Kielisyyksien toteuttaminen tapahtui niin, että erillisiin kielitiedostoihin lisättiin käännökset vastaamaan niiden avaimia. Näkymäkoodissa tekstikenttiin, jotka haluttiin olevan käännösten piirissä, lisättiin avain, josta kääntäjä tunnisti mikä käännös tekstille tulee. Kun sovellus käännettiin, kääntäjä teki sovelluksesta jokaiselle kielisyydelle omat käännökset. Kun sovelluksessa vaihdettiin sivun kieltä, koko sovellus ladattiin uudelleen.

Reactissa kielisyydet onnistui toteuttaa niin, ettei koko sivua tarvinnut ladata uudelleen, kun kielisyyttä vaihdettiin. Sovelluksesta ei myöskään tarvinnut olla jokaiselle tuetulle kielelle omaa käännöstä. React osaa Virtual DOM ominaisuuden ansioista päivittää sivulta ainoastaan muuttuneet kohdat. Eli kun sivun kieltä vaihdettiin, React päivitti sivulta vain kohdat, joihin kielisyys vaikuttaa.

4.7 Integrointi

Uudet käyttöliittymäkomponentit piti pystyä liittämään olemassa oleviin ASP.NET MVC-projektin Razor-näkymiin. Tämä oli toteutettavissa kummallakin tekniikalla samalla tavalla. Käännöksessä syntyvät JavaScript-tiedostot liitettäisiin olemassa olevalle Razor-näkymälle haluttuihin diveihin. Myös komponenttien sisäisten funktioiden paljastaminen Razor-näkymälle onnistuu liittämällä funktio näkymän ulommaiseen scopeen.

4.8 Yhteenveto

PoC vertailu	Angular	React
1) Modaalien hallinta on hankalaa => aukominen, sulkeminen, populoiminen datalla ja uudelleenkäyttö ei ole selkeää Tavoite: helposti hallittavat modaalit, joiden tilaa voi seurata ja muokata ohjelmallisesti.	Voidaan luoda modal-komponentti, jonka sisälle voidaan renderöidä toinen komponentti	Voidaan luoda modal-komponentti, jonka sisälle voidaan renderöidä toinen komponentti
2) Tarkoitukseltaan ja ulkoasultaan identtisiin elementteihin löytyy useita rakenteellisia toteutuksia(!) Tavoite: elementtirakenteet attribuutteineen voi paketoida uusiokäyttöisiksi paketeiksi.	Voidaaan luoda komponentteja elementtirakenteista	Voidaaan luoda komponentteja elementtirakenteista
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) Tavoite: plugielementtien hallinta toimii ilman js-kontrollikoodia tai keskitetyllä paketilla.	Voidaan käyttää Angularin omia datepickereitä yms komponentteja	Voidaan luoda omat datepicker yms komponentit tai importata npm:n kautta
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
6) Implementaatio tai kehys live-validaatiolle 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 etiäpäin
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
10) kielisyys	Useammalla kielellä toteutetun näkymän upottaminen vanhaan toteutukseen voi olla hankalaa	Näkymien kielisyyksien toteuttaminen näkymäkomponenttiin onnistuu

KUVA 10. Taulukko React, Angular vertailukohdista

Kun POC:t oltiin saatu valmiiksi, käytiin läpi niiden kehityksessä ilmenneitä vaikeuksia ja onnistumisia. Yhteenvetona vertailusta voi sanoa, että molemmat tekniikat täyttivät niille asetetut vaatimukset. Kumpikin tekniikoista olisi soveltunut Luennoitsijaportaalin näkymien tuottamiseen. Kummallakin tekniikalla pystyi tuottamaan nopeita, modulaarisia näkymäkomponentteja. POC:n kehitys Angularilla

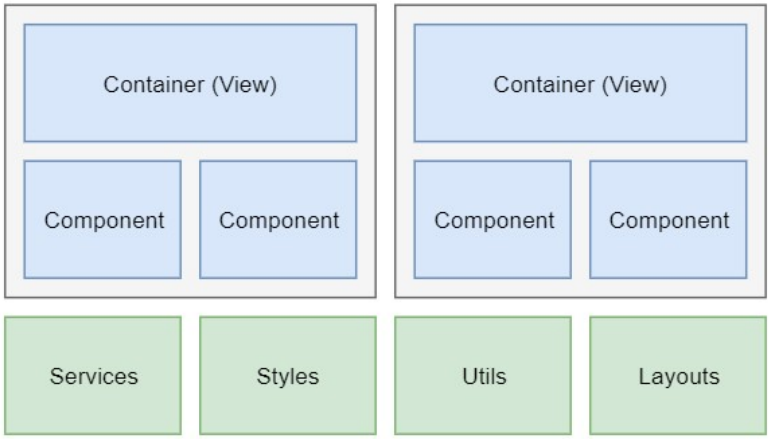
sujui aluksi nopeammin, koska Angularin mukana tuli paljon työkaluja erilaisten toimintojen suorittamiseen, kun taas Reactilla kehittäessä joutui käyttämään paljon aikaa tarvittavien kirjastojen etsimiseen ja tutkimiseen. Angular toteuttaa omalla tavallaan MVC-mallia, eli näkymät ovat erotettu logiikkakoodista. Tästä syystä Angularissa on selkeä arkkitehtuurillinen rakenne, mikä tavallaan puuttuu Reactista.

React vaikutti kuitenkin rakenteeltaan yksinkertaisemmalta ja nopeammin opittavalta. Reactilla kehittäessä kehittäjällä on vapaus valita ne laajennukset, mitä hän sovellukseen haluaa, eikä mitään ylimääräisiä liitännäisiä tule kehitysympäristön mukana. Tiimin sisällä koettiin, että Reactin opettelusta olisi pitemmän päälle enemmän hyötyä. Angularin hylättiin toistaiseksi ja päätös lähteä kokeilemaan uusien komponenttien tekoa Reactilla syntyi. Päätökseen vaikutti osaltaan myös se, että yhdellä tiimin jäsenistä oli jo huomattavaa osaamista Reactista.

5 SOVELLUS

5.1 React-projekti

Kun tekniikka oli päätetty, React-projektin suunnittelu aloitettiin. React-projektille kehitettiin selkeä rakenne. Komponentit ja näkymät jäsenneltiin loogisesti. Sovittiin, että logiikkaa sisältävät luokat esim. REST-kutsuja lähettävät servicet eriytetään omiin tiedostoihinsa. Näkymät jaettiin Containereihin ja niiden alle niihin liittyvät komponentit. Containereiden yhteiset, yleiskäyttöiset komponentit sijoitettiin omaan, CommonComponents kansioon.



KUVA 11. React-projektin osia

Ylläolevassa kuvassa on esitettynä joitain React-projektin osia. Näkymäkomponentteja nimitettiin containereiksi, ne koostuvat pienemmistä komponenteista. Palvelimen REST-rajanpintaan pyyntöjä lähettävät ja vastaanottavat toiminnot kirjoitettiin Service-luokkiin. Näkymäkomponenttien käyttämät CSS-tyylit kirjoitettiin tyyliluokkiin. Komponenttien käyttämät apufunktiot ja muu toiminnallisuus kirjoitettiin Utils-luokkiin. Komponentit ja containerit kirjoitettiin käyttäen TypeScriptiä sekä JSX. Muut luokat kirjoitettiin TypeScriptillä.

```
57  static async uploadFiles(files: FileList) {
58    const formData = new FormData();
59    for (let i = 0; i < files.length; i++) {
60      formData.append('file_' + files.item(i)!.name, files.item(i)!);
61    }
62    return axios
63      .post(uploadFileRoute, formData, { headers: { 'content-type': 'multipart/form-data' } })
64      .then((response) => response)
65      .catch((error: AxiosError) => console.error(error));
66  }
```

KUVA 12. Tiedostot lähettävä service pyyntö

Kun React-komponentti lähetti pyynnön palvelimelle, pyyntö kasattiin service-luokassa FormData-muotoiseksi. Tiedostojen lähettämiseen käytettiin FormData-luokkaa, koska siihen oli mahdollista paketoida lähetettävät tiedostot sekä muu mahdollinen JSON-data. Pyyntöön lisättiin myös Routen

tunniste, josta BL-palvelimen Route-käsittelijä tunnisti pyynnön ja osasi ohjata sen oikealle käsittelijälle. Service-luokka lähetti pyynnön React-komponentilta kontrollerille käyttäen POST-metodia. Jokainen luennoitsijaportaalin näkymä lähetti pyynnöt samalle kontrollerille.

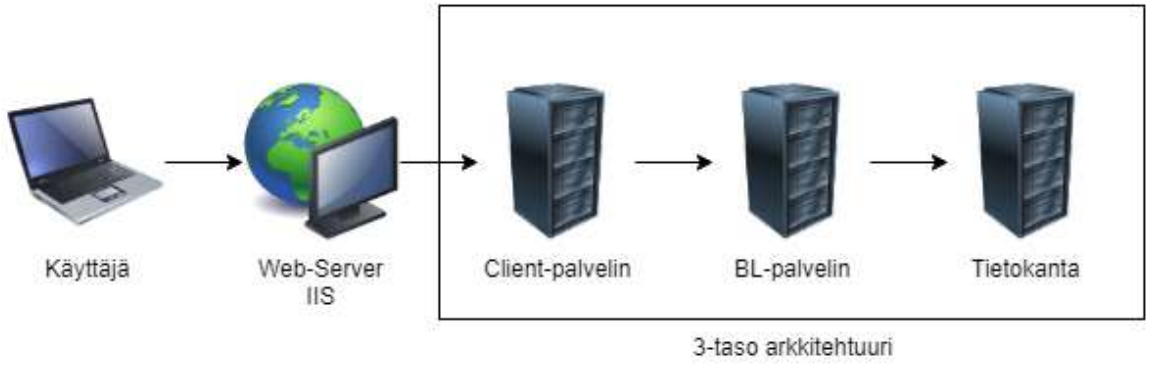
5.2 React-näkymien tuottaminen

Luennoitsijaportaalin näkymiä tuotettiin tiimissä. Ennen kuin näkymiä alettiin tuottaa, ne suunniteltiin uudelleen ja niiden tarvitsema logiikka käytiin läpi. Sovittiin, kuka tekee näkymän minkäkin osan. Komponentin tuottaminen suunniteltiin niin, että yksi henkilö teki komponentin käyttöliittymä osan ja toinen henkilö komponentille tarvittavan palvelinlogiikan. Näin saatiin tuotettua osat tehokkaasti. Kommunikointi tiimin jäsenten kesken sujui saumattomasti.

Aika ajoin pidettiin statuspalavereja koskien projektin kokonaiskuvaa. Niissä käytiin läpi missä vaiheessa projektin minkäkin näkymän kehitys on ja onko kehityksessä jotain esteitä, joita pitäisi tiimin kesken ratkoa. Suunniteltiin myös ennakkoon vaadittavia toimenpiteitä, joilla pystyttiin poistamaan esteitä projektin etenemisen tieltä. Kun kaikki toiminnallisuuteen liittyvät osat, näkymät ja logiikka, oli suunniteltu, niiden koodaaminen aloitettiin. Kun näkymä saatiin valmiiksi, se testattiin vaatimusmäärittelyssä määriteltyjen testitapausten mukaan. Näkymiä päivitettiin asiakkaan testiympäristöön sitä mukaa kun niitä oltiin saatu valmiiksi ja testattua.

5.3 Kongressi

Kongressi-sovellus toteuttaa 3-taso arkkitehtuuria. Kolme eri palvelinta hoitavat Client-tason (näkymät), BL-tason (Business-logiikka) ja tietokantatoiminnot.



KUVA 13. Sovellusarkkitehtuuri

Client-palvelin on ohjelmoitu ASP.NET tekniikalla toteuttaen MVC-suunnittelumallia. React-komponentit liitettiin Client-palvelimen Razor-näkymiin. BL-palvelin on ohjelmoitu .Net Frameworkillä ja tietokanta on toteutettu Microsoft SQL Serverillä. Client-palvelin pitää sisällään näkymät, sessiotiedot sekä näkymien käyttämiseen liittyvän logiikan. BL-palvelin sisältää varsinaisen business-logiikan sekä tietokantakäsittelyt. Tämän etuna on se, että sovelluslogiikka on selkeästi erotettu tietokannasta ja käyttöliittymästä. Tämä mahdollistaa käyttöliittymäosien nopean tuottamisen, koska ne voivat käyttää samoja sovelluslogiikan osia.

5.4 React-näkymien integrointi

React-komponentit liitettiin Razor-näkymille div elementeillä ja importoimalla React käännöksessä syntyvä JavaScript-tiedosto sivulle. Käytännössä HTML-sivulle tulee rivi

** **

Kun JavaScript-tiedosto importataan ja ajetaan, se tunnistaa id:stä lohkon, johon renderöi halutun komponentin. Javascript-tiedosto importataan rivillä

<script src="@Url.ContentVersioned(~/Content/js/reactComponents.js)">

```
@using DataPrisma.Common.UI.Helpers
@{
    Layout = null;
}
<section class="content--xl">
    <section class="content--insides">
        <div id="root5"></div>
    </section>
</section>

<link href="@Url.ContentVersioned("~/Content/css/reactComponents.min.css")"
      rel="stylesheet" />
<link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.2.0/css/all.css"
      integrity="sha384-hWVjflwFxL6sNzntih27bfxkr27PmbbK/iSvJ+a4+0owXq79v+lsFkW54bOGbiDQ"
      crossorigin="anonymous" />
<script src="@Url.ContentVersioned("~/Content/js/reactComponents.js")"></script>
```

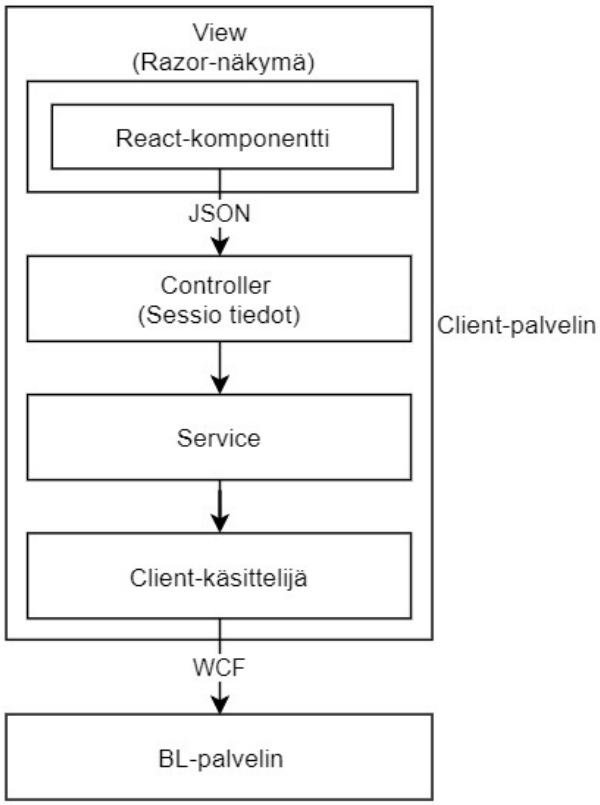
KUVA 14. React-komponentti Razor-näkymällä Id:llä root5

React-projektin käännöksessä syntyvät Javascript- ja CSS-tiedostot liitetään ASP.NET projektin Content kansioon, jossa ne ovat tarvittaessa päivitettävissä myös Release buildissä. Näin ollen voidaan päivittää asiakkaan asennuksesta pelkät React-komponentit korvaamalla vanha JavaScript-tiedosto uudella.

React-komponentit keskustelevat Client-palvelimen kontrollereiden kanssa, joista pyynnöt virtaavat eteenpäin BL-palvelimelle. Näkymien suunnitteluvaiheessa pohdittiin, olisiko React-komponentit voineet lähettää pyynnöt suoraan BL-palvelimelle. Mutta koska tarvittiin mm. sessioon tallennettuja käyttäjätietoja Client-palvelimelta, putken tekeminen Client-palvelimen läpi oli välttämättömyys.

5.5 Client-palvelin

Razor-näkymällä oleva React-komponentti lähettää FormData-muotoisen pyynnön Client-palvelimen kontrollerille REST-rajapintaan. Kontrolleri välittää pyynnön Service-tason läpi BL-palvelimelle WCF-rajapinnan kautta, joka ottaa pyynnön vastaan, käsittelee sen ja tekee tarvittavat toimenpiteet tietokantaa vasten. Client-palvelimen kontrolleri vastaa React-komponentille JSON-muotoisena.



KUVA 15. Client-palvelimen kerrokset

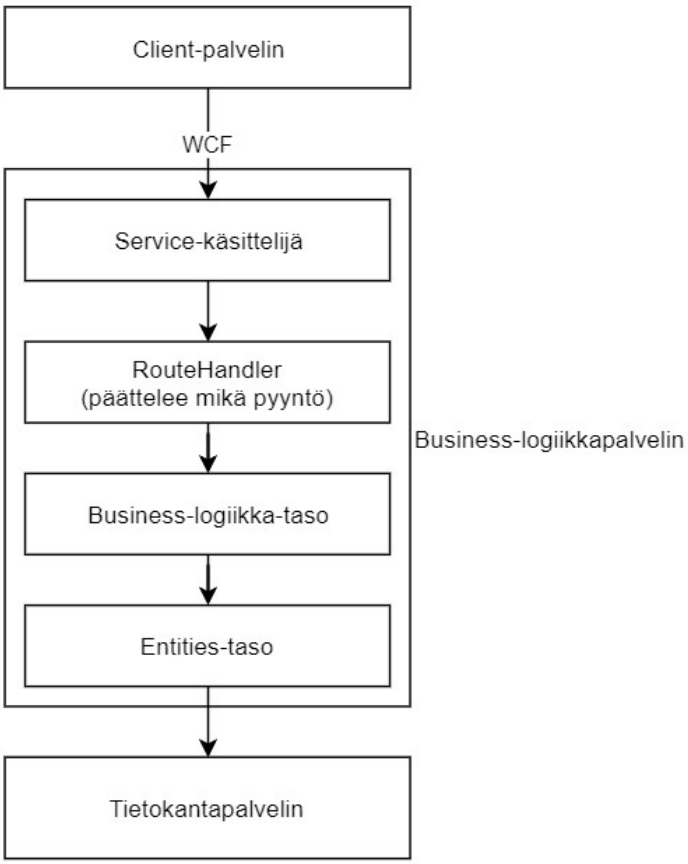
Putki Client-palvelimen läpi toteutettiin niin, että kun tarvittiin lisätä logiikkaa BL-palvelimelle uutta näkymää varten, Client-palvelimelle ei tarvinnut lisätä uutta koodia. Ainoastaan lisättiin React-näkymään kutsu ja BL-palvelimelle Route, jonka tunnistaa pyynnön avaimesta ja siten ohjaa pyynnön oikealle käsittelijälle.

Tämä tapahtuu käytännössä niin, että kun React-komponentilta lähetetty pyyntö saapuu kontrollerille, kontrollerilla pyynnöstä kerätään parametrit ja mahdolliset tiedostot. Sessiosta poimitaan käyttötilanteeseen liittyvät tiedot. Näitä ovat mm.

- CustomerId, asiakkaan id
- NetworkId, tapahtuman id
- LanguageId, näkymän käyttökieli

Client-palvelimen Service-tasolla parametreista ja sessiotiedoista kasataan Request-objekti. Service-tasolla myös tutkitaan, sisälsikö pyyntö tiedostoja. Request-objekti lähetetään edelleen Client-käsittelijälle, josta se ohjataan WCF-rajapinnan kautta BL-palvelimelle. Jos alkuperäinen pyyntö sisälsi tiedostoja, niin Service-tason käsittely eroaa siinä määrin muista käsittelyistä, että kun pyyntö palaa BL-palvelimelta, ja jos validaatio on onnistunut, niin tiedostot käydään tallentamassa Client-palvelimen levylle.

5.6 BL-palvelin



KUVA 16. BL-palvelimen kerrokset

BL-palvelimella pyyntö otetaan vastaan WCF-käsittelijällä, mistä se ohjataan Route-käsittelijälle. Tämä osaa tunnistaa pyynnön parametristä ja ohjata sen oikealle Routelle. Route sisältää tiedon siitä mikä luokka hoitaa pyynnön käsittelyn Business-tasolla.

```
## API kutsut - lista

#### Common/

- **/getnetworkname** (hae tapahtuman nimi)
- **/getlanguage** (hae UI:n aktiivinen kieli)

#### Files/

- **/getlist** (haetaan tiedostolista)
- **/right/{ID}** (hae tiedoston näkyvyystiedot)
- **/editfileinfo** (päivitä tiedoston perustiedot)
- **/editfileright** (päivitä tiedoston näkyvyystiedot)
- **/upload** (lisää tiedosto/t)
- **/delete** (poista tiedosto)
- **/getaddfileright** (tarkista onko käyttäjällä oikeus lisätä tiedostoja)

#### Rewards/

- **/getRewardList** (hae palkkiolista)
- **/getPayDetails** (hae maksutiedot)
- **/getRewardTypeList** (hae palkkiotyyppilista)
- **/updateReward/{ID}** (lisää/muokkaa palkkio)
- **/updatePayDetails** (muokkaa maksutiedot)
```

KUVA 17. Kutsut, joita varten BL-palvelimella route ja oma käsittelijä

Erilaisia pyyntöjä on esimerkiksi ”getrewardlist”, jolla pyydetään asiakkaan hakemat palkkiot. Pyyntö menee Business-tasolle, jossa tehdään pyyntöön liittyvä logiikka, esimerkiksi haetaan tai tallennetaan tietoa. Vaste kasataan Entities-tasolta pyydetyillä tiedoilla. Vaste muutetaan objektiksi ja lähetetään takaisin BL-palvelimen läpi Client-palvelimelle ja edelleen JSON-muodossa React-komponentille. React-komponentti muodostaa näkymän ja näyttää JSON:n muodossa saamansa tiedot käyttäjälle.

6 LUENNOITSIJAPORTAALI

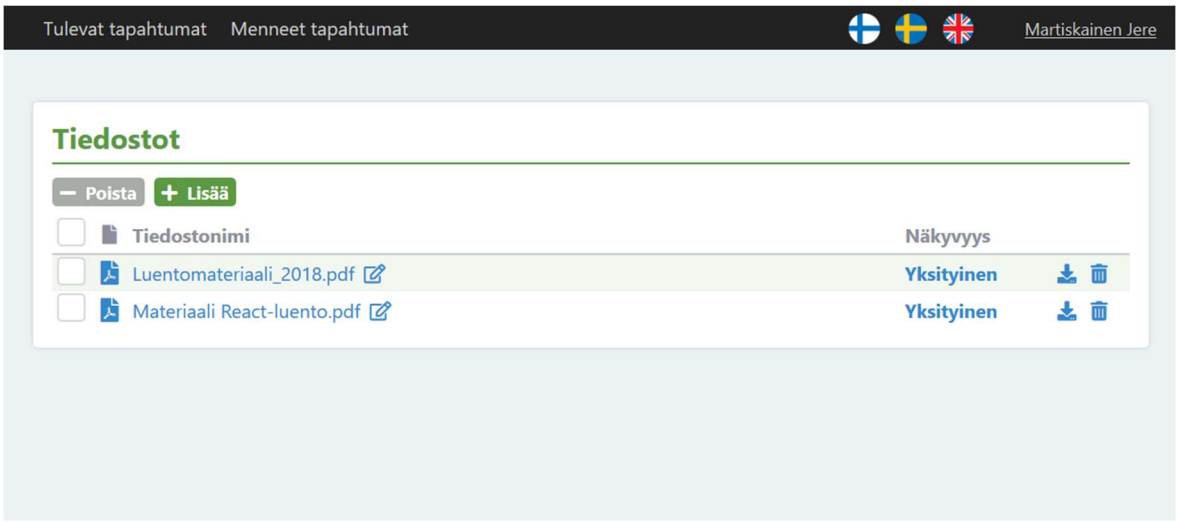
Luennoitsijaportaali on ryhmä näkymiä, joita sisään kirjautuneet käyttäjät voivat käyttää Portaalissa. Portaali on osa Kongressi sovellusta. Henkilöt ilmoittautuvat Kongressin ilmoittautumislomakkeen kautta tapahtumaan. Ilmoittautuneet käyttäjät voivat kirjautua Portaaliin ilmoittautumisen yhteydessä saamillaan tunnuksilla. Jos heillä on luennoitsijarooli, he pääsevät käsiksi Luennoitsijaportaalin näkymiin.

KUVA 18. Käyttötapauksen kuvaus

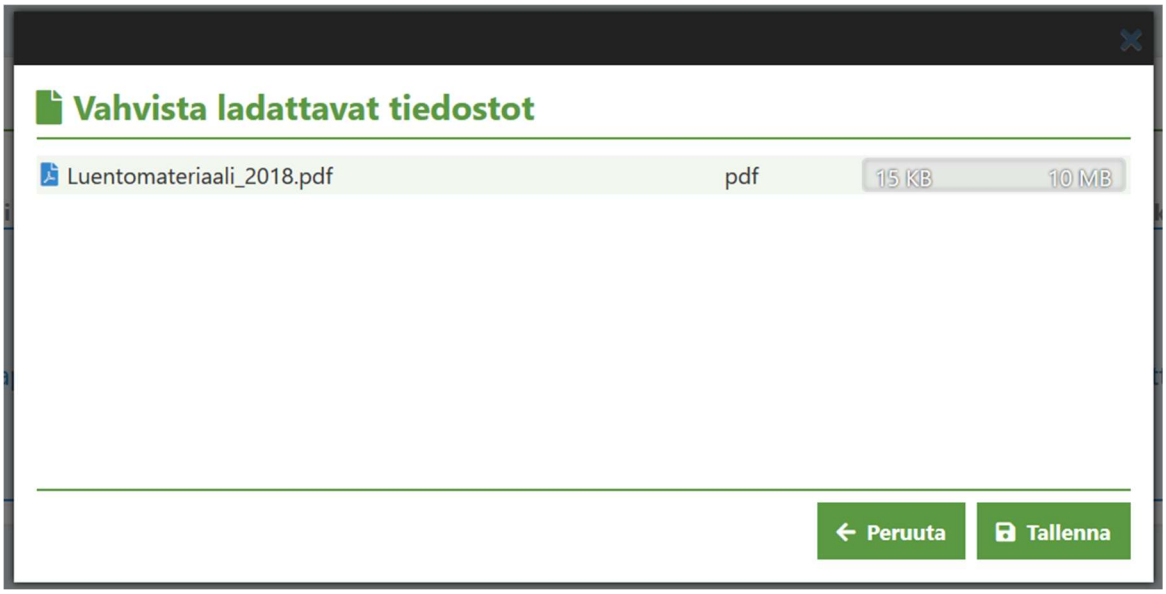
Luennoitsijaportaalin käyttötapaus on seuraavanlainen. Käyttäjän on kirjauduttava Portaali-palveluun. Kirjauduttuaan hän näkee linkit Materiaalit- ja Palkkiot-näkymille jos näkymät on asetettu näkyviin ilmoittautumislomakkeille. Materiaalit- ja palkkiotiedot-näkymä on siis asetettava päälle sille ilmoittautumislomakkeelle millä ne halutaan näkyvän, ilmoittautumislomakkeen ylläpidosta. Käyttäjä voi lisätä luentomateriaaleja Materiaalit-näkymällä. Hän voi hakea palkkioita sekä muuttaa palkkionmaksutietojaan palkkionäkymällä. Henkilötietonäkymällä hän voi muokata omia henkilötietojaan.

6.1 Materiaalit

Materiaalit näkymässä käyttäjä voi lisätä tapahtumaan tiedostoja, esimerkiksi luentoihin liittyvää materiaalia. Hän voi asettaa tiedoston näkyviin eri henkilöille esimerkiksi roolien, luentosarjojen tai luentojen mukaan. Materiaalit-näkymässä tapahtumaan osallistuja (ei luennoitsija) voi ladata luennoitsijoiden lisäämiä materiaaleja.



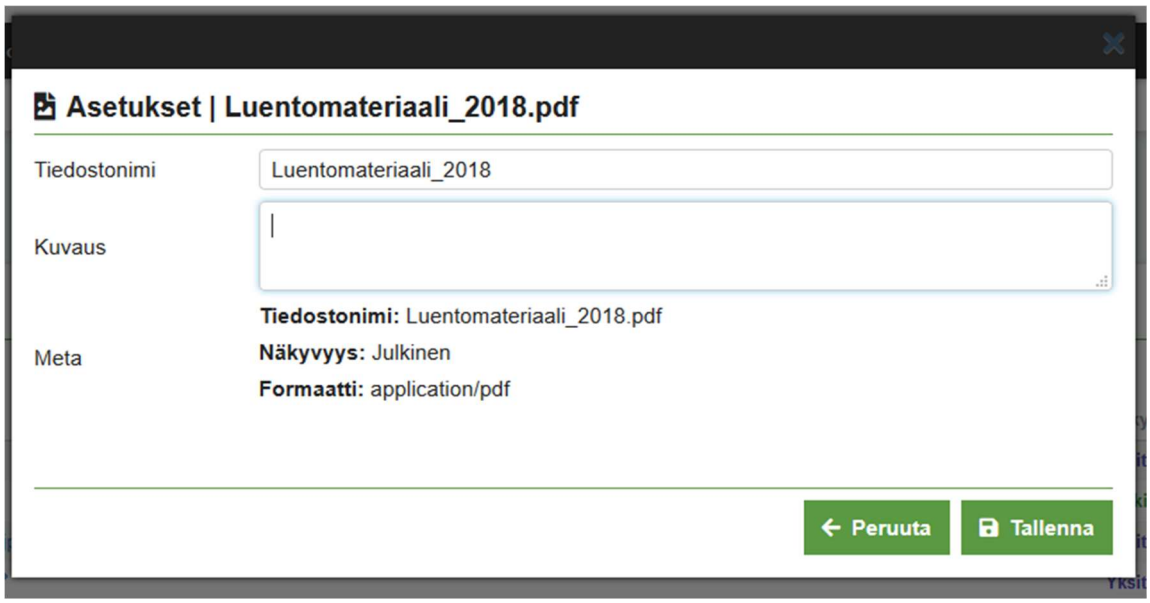
KUVA 19. Materiaalit-näkymä



KUVA 20. Tiedoston lataaminen

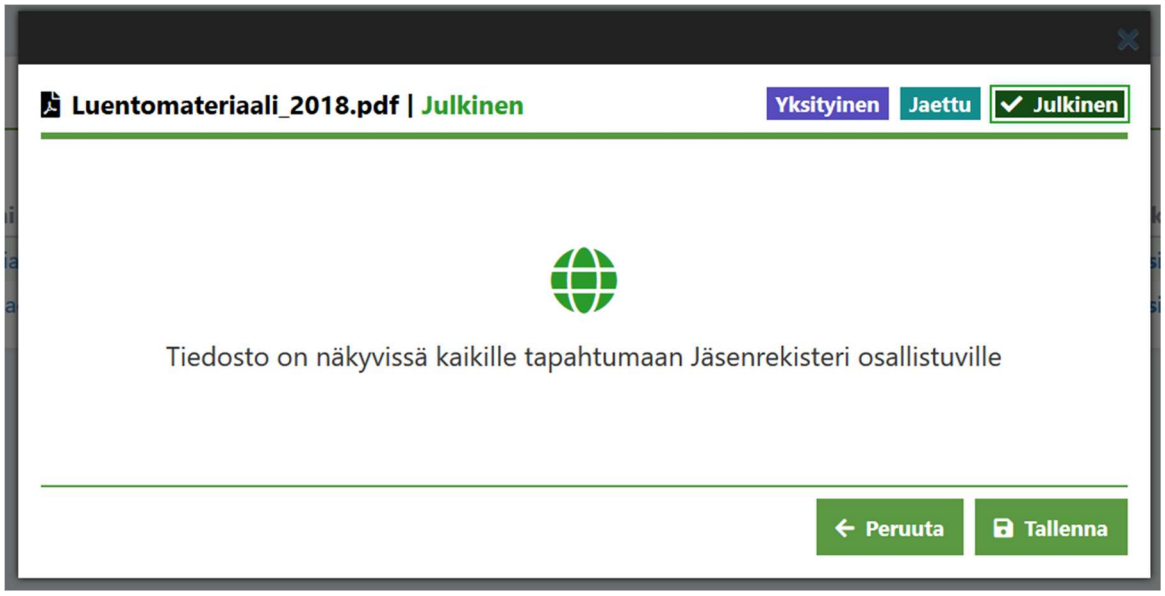
Tiedoston lähettämiselle on asetettu tiettyjä validaatio-sääntöjä. Vain tietyn tyyppiset tiedostot ovat sallittuja. Näkymä sallii lähettää vain dokumentti- (esimerkiksi pdf ja excel) ja kuvatiedostoja (esimerkiksi .jpg ja .png). Tiedoston koko on rajoitettu 10 megatavuun. Valitun tiedoston kokoa indikoidaan näytöllä olevalla palkilla. Kun käyttäjä on valinnut tiedoston, näkymän live-validaatio tarkastaa tiedoston tyypin ja koon. Jos validaatio menee läpi, tiedosto lähetään Client-palvelimelle. Tiedoston tiedot käydään vielä kertaalleen validoimassa BL-palvelimella. Jos BL-palvelimen validaatio menee läpi, tiedosto tallennetaan Client-palvelimen kovalevylle.

Tiedoston lataaja voi muokata tiedoston tietoja. Hän voi muuttaa tiedoston nimeä ja tiedoston kuvausta.

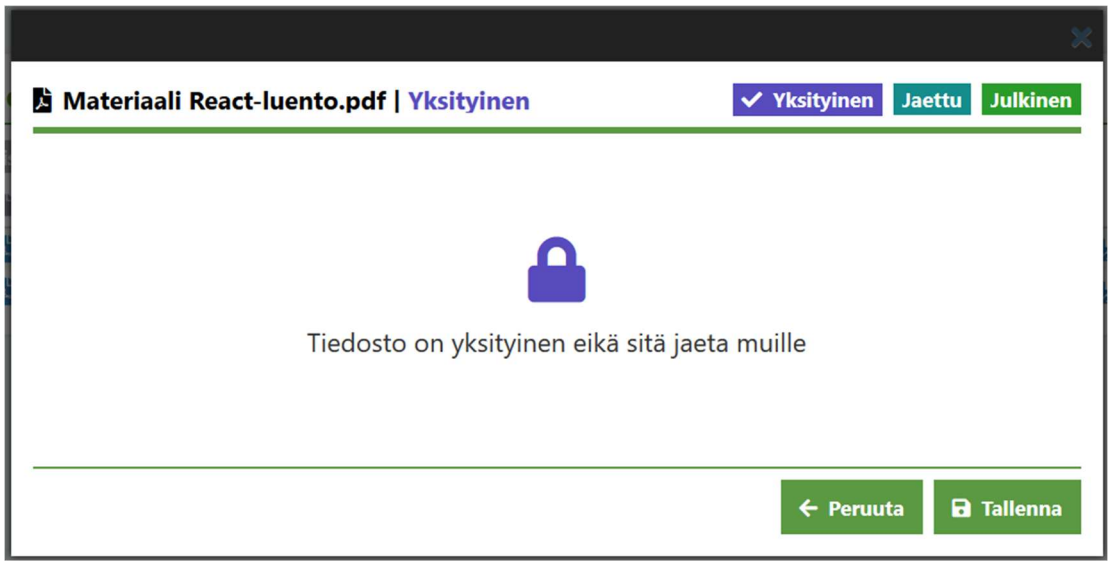


KUVA 21. Tiedoston tietojen muokkaus

Nimi- ja kuvauskentät validoidaan käyttöliittymässä sekä BL-palvelimella. Niissä ei voi käyttää tiettyjä merkkejä sekä niiden merkkien määrää on rajoitettu. Live-validaatio validoi lomakkeen ennen lähetystä. Se tarkistaa tiedoston nimen, joka on pakollinen kenttä. Käyttäjä voi jättää kuvauksen tyhjäksi.

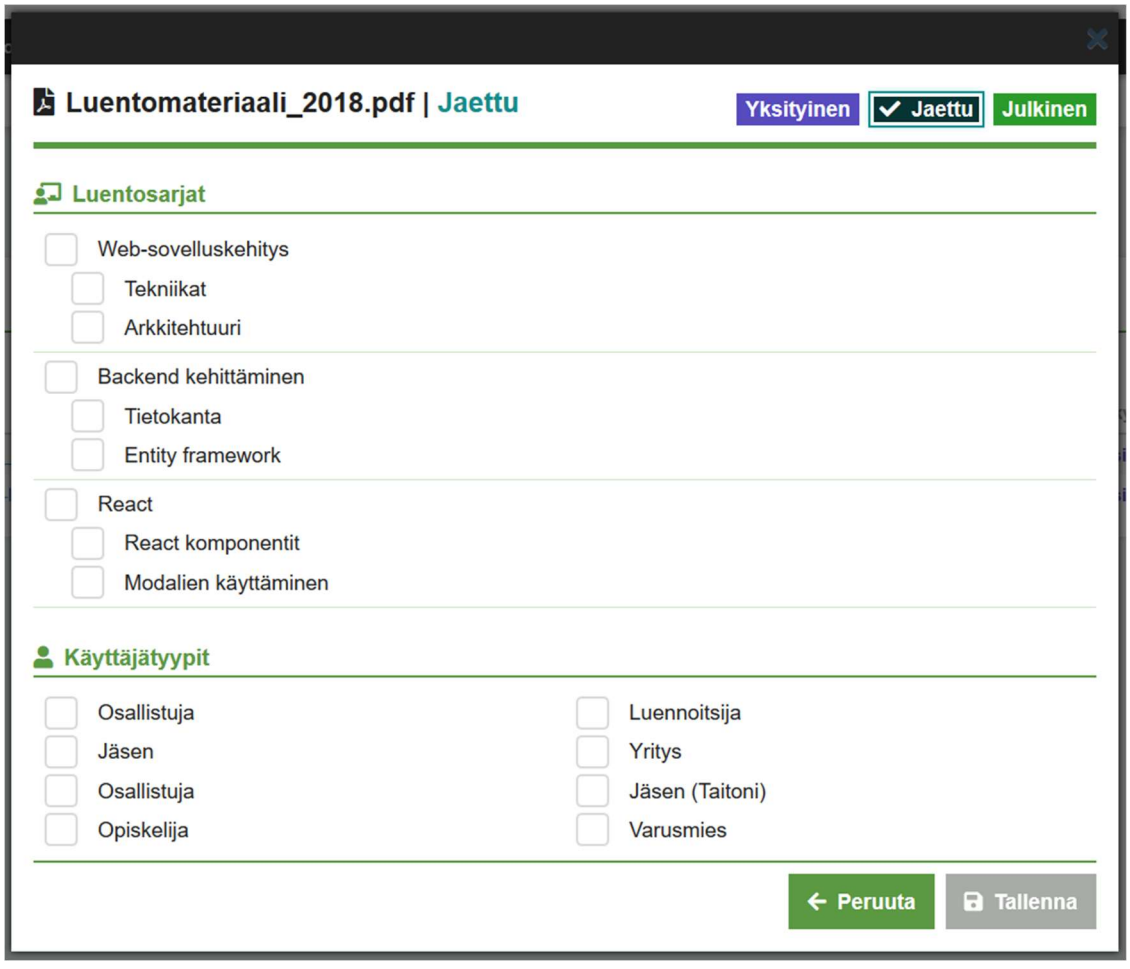


KUVA 22. Materiaalin asettaminen julkiseksi



KUVA 23. Materiaalin asettaminen yksityiseksi

Käyttäjä voi muuttaa lataamansa tiedoston oikeuksia. Hän voi asettaa tiedoston näkyväksi roolin, luentosarjan tai luennon perusteella. Hän voi valita esimerkiksi ”Luennoitsija”-roolin, sekä jonkin tietyn luentosarjan ja/tai luennon. Hän voi ainoastaan valita luentosarjoja ja luentoja, joissa hän itse on luennoitsijana. Tällöin tiedosto näkyy vain henkilöille, jotka täyttävät nämä kriteerit. Hän voi asettaa materiaalin myös julkiseksi, jolloin se näkyy kaikille tapahtuman henkilöille.



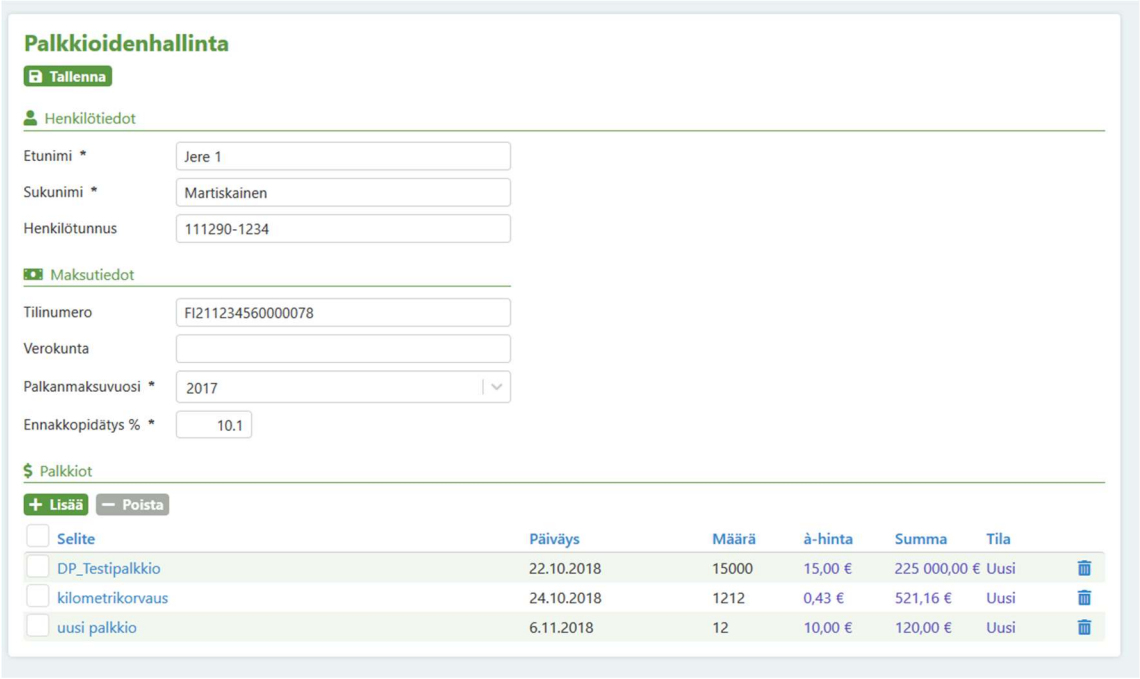
KUVA 24. Materiaalin jakaminen roolin tai luentosarjan mukaan

Kun hän painaa tallenna painiketta, näkymä lähettää tiedot valituista oikeuksista eteenpäin. BL-palvelimen validointi tarkistaa onko henkilöllä oikeasti riittävät oikeudet lisätä materiaaleja. Lisäksi tarkistetaan, onko hän luennoitsija niillä luennoilla mille hän yrittää asettaa materiaaleja näkyviin. Kun validaatio on mennyt läpi, tiedot oikeuksista tallennetaan ja käyttöliittymään palautetaan tieto onnistuneesta tallennuksesta.

Kun osallistuja näkee Materiaalit-näkymällä tiedostoja ja hän painaa tiedoston lataa-painiketta, lähtee näkymältä latauspyyntö palvelimelle. Palvelimella tarkistetaan, onko käyttäjällä oikeasti oikeus ladata tiedosto. Jos tarkistus menee läpi, lähetetään tiedosto käyttäjälle. Jos ei, näytetään virheviesti.

6.2 Palkkiot

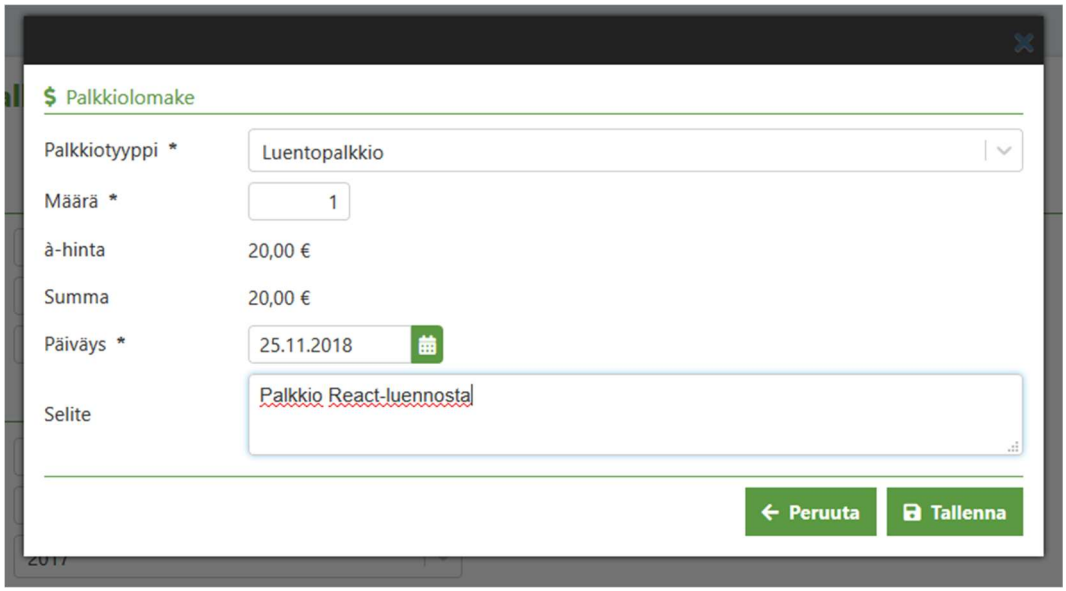
Palkkionäkymällä portaaliin kirjautunut henkilö voi muokata palkkiomaksutietojaan ja kirjata palkkiohakemuksia. Palkkiomaksutietoja ovat henkilön tilinumero, veroprosentti sekä palkanmaksuvuosi. Tietoja ylläpitävät kentät on validoitu, eli tilinumeroon ei voi syöttää muuta kuin IBAN muotoisen tilinumeron ja veroprosenttiin kelpaa vain 2 numeroinen luku.



KUVA 25. Palkkioidenhallinta-näkymä

Jos käyttäjä jättää täyttämättä pakolliset, tähdellä merkityt kentät ja painaa tallenna-nappia, näkymä ilmoittaa käyttäjälle, ettei voi tallentaa, koska vaaditut kentät puuttuvat. Tämä on toteutettu ns. live-validaatiolla eli käyttöliittymä tarkistaa vaaditut kentät jo ennen kuin pyyntöä lähetetään käyttöliittymästä eteenpäin. Live-validaatio tarkistaa myös, että palkanmaksu vuosi on kuluva vuosi tai yksi vuosi taaksepäin. Ennakonpidätysprosentti validoidaan niin, että sen on oltava pienempi kuin 100% ja suurempi tai yhtä suuri kuin 0%. Kun live-validaatio on onnistunut ja lomake on validi,

käyttöliittymä lähettää tiedot lomakkeelta Client-palvelimelle ja liittää pyyntöön parametrin ”updatecustomerpaymentdetails”. Tällä parametrillä BL-palvelin osaa ohjata pyynnön oikealle Routelle ja näin oikeanlaiseen käsittelyyn.



KUVA 26. Palkkion lisäys

Palkkiota hakiessa hakijan täytyy valita palkkiotyyppi ja kappalemäärä. Näkymä laskee maksettavan palkkion summan syötetyn määrän ja palkkiotyypin yksikköhinnan mukaan. Hän voi myös antaa lisäselvitystä haetusta palkkioista. Palkkiotyyppejä ylläpidetään Kongressi-sovelluksessa. Lomakkeen live-validaatio tarkastaa lomakkeesta pakolliset kentät, joita ovat palkkiotyyppi, määrä ja päiväys. Kun nämä kentät ovat täytetty, antaa näkymä lähettää lomakkeen eteenpäin. Syötetyn palkkion tiedot tarkistetaan vielä BL-palvelimella.

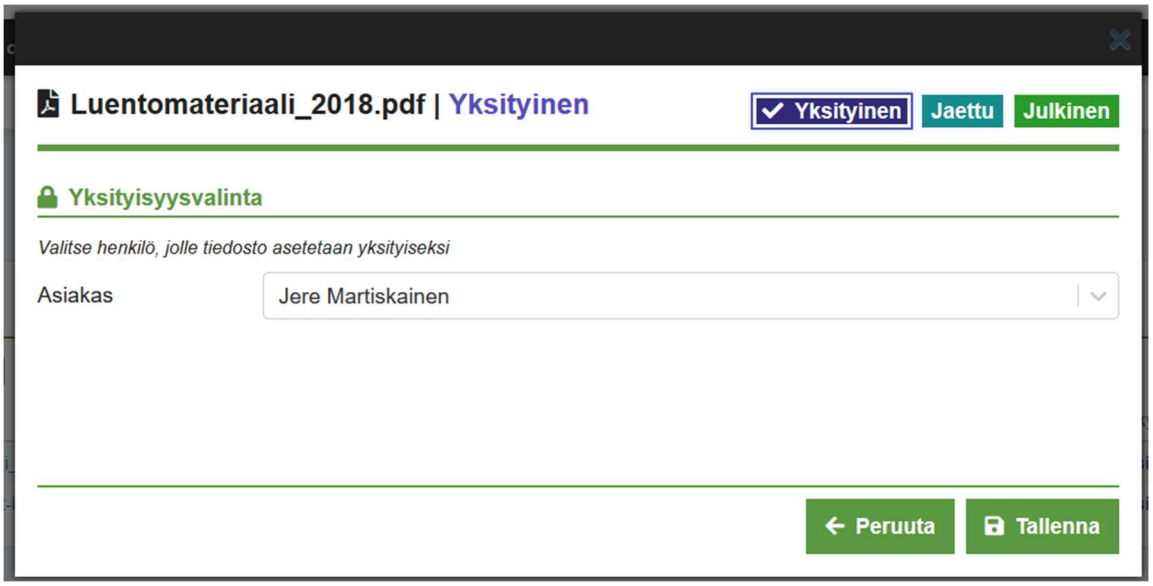
6.3 Tapahtuman tiedostojen hallintanäkymä

Kongressin puolella olevassa tiedostojen hallintanäkymässä admin-tason käyttäjä voi tarkastella, lisätä ja poistaa tapahtuman tiedostoja. Käyttäjä voi myös asettaa tiedostoja näkyviin eri henkilöille roolien, luentosarjojen tai luentojen mukaan. Kyseinen näkymä tarvittiin, jotta luennoitsijoiden lataamia tiedostoja voitaisiin hallinnoida keskitetysti Kongressin puolelta.

KUVA 27. Tiedostojen hallinta näkymä Kongressissa

Tiedostot-näkymällä tiedostot ovat listattuna samaan tapaan kuin Materiaalit-näytöllä. Itseasiassa kyseessä on sama käyttöliittymäkomponentti, laajennetuilla ominaisuuksilla. Käyttäjä voi asettaa tiedoston yksityiseksi valitsemalleen henkilölle. Tällöin kukaan muu ei näe tiedostoa eikä pysty lataamaan sitä. Käyttäjä voi asettaa tiedoston näkyvyyden myös roolin ja luentosarjan mukaan, aivan kuten luennoitsijan näkymässä, tosin sillä erotuksella, että käyttäjä voi valita vapaasti minkä tahansa luentosarjan tai luennon.

Tätä näkymää on tarkoitus laajentaa niin, että henkilöille voi antaa 3 tasoisia oikeuksia tiedostoon, eli luku-, muokkaus- ja poisto-oikeudet. Näkymää on tarkoitus laajentaa käsittämään myös tapahtuman muita tiedostoja kuten lomakkeiden ja tapahtuman kuvia, palkkiomaksuliitteitä ja osallistujien lataamia liitteitä.



KUVA 28. Kuvan asettaminen yksityiseksi valitulle henkilölle

Erona Materiaalit-näytön tiedostonoikeusmodaaliin, tässä modalissa voi asettaa tiedoston näkyviin haluamalleen henkilölle. Tiedosto on tällöin yksityinen hänelle, eikä kukaan muu pysty katsomaan tai

lataamaan tiedostoa. Tiedoston voi Materiaalit-näytön tapaan myös asettaa jaetuksi halutulle roolille ja/tai luentosarjalle ja/tai luennolle, tai kokonaan julkiseksi.

7 YHTEENVETO

7.1 Jatkokehitys

Luennoitsijaportaalin näkymien uudelleen kehitys jatkuu. Myös muita näkymiä Kongressiin on tehty Reactilla ja sitä tullaan käyttämään jatkossakin uusien näkymien tuottamiseen. Päätökseen Reactin valinnasta voi olla tyytyväinen. Vaikka React on osoittautunut varsin hyväksi tekniikaksi näkymien tuottamiseen, sillä kehittäminen on tuonut myös omat ongelmansa, joihin pyritään löytämään ratkaisuja tulevaisuudessa.

Luennoitsijaportaaliin kuuluu tässä esiteltyjen näkymien lisäksi:

- Järjestäjän tiedotteiden hallinta
- Luento- ja ohjelmaehdotusten lisäys- ja arviointinäkymät
- Henkilötiedot-näkymä, joka pitää sisällään nimi- ja osoitetiedot, rajoitetiedot, tietosuojat sekä työosoitetiedot.

Näiden näkymien suunnittelua ja kehitystä jatketaan. Suunnitteilla on ottaa uutta tekniikkaa myös backendin kehitykseen. Suunnitelmissa on alkaa vähitellen siirtymään .NET Frameworkistä .NET Coren käyttöön.

7.2 Työn tulokset

Projekti on ollut erittäin opettavainen. Sen aikana opittiin paljon käyttöliittymäosien sekä logiikkakoodin ohjelmoinnista. Yksi syy miksi projekti valikoitui opinnäytetyön aiheeksi, oli se, että tutustuminen käyttöliittymätekniikoihin auttaisi eteenpäin oppiessa web-sovelluskehitystä ns. Full-stackinä, eli pystyä kehittämään sekä front-endiä että back-endiä. Se tavoite täyttyi ja projekti toi arvokasta kokemusta molempien suunnittelusta ja kehittämisestä, sekä tietämystä siitä mitä kaikkea tulee huomioida, jotta käyttöliittymäkomponentit sekä palvelinkomponentit toimivat mahdollisimman hyvin keskenään.

Tekniikoita tutkiessa tuli yllätyksenä, kuinka paljon löytyy valinnan varaa tekniikoista, joilla tuottaa web-sivun UI-komponentteja ja siitä, miten nopeasti tekniikat kehittyvät. Vaatii jatkuvaa seuraamista ja opiskelua, jos haluaa pysyä kehityksen vauhdissa mukana. Tekniikkavertailuun valitut Angular ja React osoittautuvat molemmat päteviksi tekniikoiksi UI-komponenttien tuottamiseen. Lisähaasteen tekniikoiden opiskeluun toi puutteelliset JavaScript-taidot, molempien tekniikoiden ollessa JavaScriptiin pohjautuvia tekniikoita. JavaScriptiä ja TypeScriptiä oppi kuitenkin nopeasti tekemällä tutoriaaleja ja POC:ia Angularilla. Vaikka tekniikkavertailussa päätettiin, että React on tekniikka, jota käytetään näkymien tuottamiseen, ei Angularin opettelu ollut mennyt hukkaan. Angular tekemisen jäljiltä oli jo valmiiksi osaamista JavaScriptistä sekä TypeScriptistä, mikä auttoi suuresti JSX:n opettelussa ja React-komponenttien tuottamisessa.

Kun päätöstä tehtiin tekniikan valinnasta, eräs päätökseen vaikuttava tekijä oli se, että yhdellä tiimin jäsenistä oli jo entuudestaan kokemusta Reactilla tekemisestä. Tämä edesauttoi suuresti muita tiimin jäseniä etenemään nopeasti Reactilla tuottamisen opettelemisessa, koska oli joku keneltä kysyä ratkaisua yksinkertaisiinkin ongelmiin eikä tarvinnut käyttää aikaa googlettamiseen ja ratkaisun etsimiseen netistä. Myös näkymien suunnitteluvaiheessa auttoi suuresti, kun oli joku, joka jakoi tietämystään siitä, miten asioita kannattaa lähteä toteuttamaan, ettei tarvinnut ensin erehtyä ja sitten todeta, ettei lähestymistapa ollutkaan hyvä. Näin säästyi paljon aikaa niin suunnittelussa kuin kehityksessä.

React on osoittautunut hyväksi front-end-tekniikaksi. Se on täyttänyt kaikki sille asetetut vaatimukset. Reactilla tuotetut näkymät ovat nopeita ja niiden integroiminen Razor-näkymiin kävi oletettua helpommin. Näkymien tuottaminen on nopeaa ja ne ovat hyvin modulaarisia ja uusiokäyttöisiä. Reactilla tullaan varmasti tuottamaan jatkossa muitakin uusia näkymiä Kongressiin.

Projektin aikana saatiin tuotettua tärkeimmät Luennoitsijaportaalin näkymät uudelleen Reactilla ja integroitua ne onnistuneesti osaksi Kongressia. Projekti on onnistunut oikein hyvin. Se on opettanut tekijäänsä niin sovelluskehityksessä kuin projektityöskentelyssä. Yksikkö- ja integraatiotestien tekemiseen olisi voinut keskittyä enemmän ja tehdä koodia enemmän testivetoisesti. Tähän tullaan kiinnittämään enemmän huomiota jatkokehitysvaiheissa.

LÄHTEET JA VIITTAUKSET

Data Prisma Oy. [Viitattu 2018-10-26]. Saatavissa:
http://www.dataprisma.fi/fin/index.html

Kongressi. [Viitattu 2018-10-28]. Saatavissa:
http://www.dataprisma.fi/fin/kongressi.html

Visual Studio Code. [Viitattu 2018-11-2]. Saatavissa:
https://en.wikipedia.org/wiki/Visual_Studio_Code

SQL Server Management Studio. [Viitattu 2018-11-2]. Saatavissa:
https://en.wikipedia.org/wiki/SQL_Server_Management_Studio

React. [Viitattu 2018-11-2]. Saatavissa:
https://en.wikipedia.org/wiki/React_(JavaScript_library)

Versionhallinnasta. [Viitattu 2018-11-10]. Saatavissa:
https://git-scm.com/book/fi/v1/Alkusanat-Versionhallinnasta

Microsoft Visual Studio. [Viitattu 2018-11-10]. Saatavissa:
https://en.wikipedia.org/wiki/Microsoft_Visual_Studio

Microsoft SQL Server. [Viitattu 2018-11-11]. Saatavissa:
https://en.wikipedia.org/wiki/Microsoft_SQL_Server

Git. [Viitattu 2018-11-11]. Saatavissa:
https://www.linux.fi/wiki/Git

Angular Architecture. [Viitattu 2018-11-11]. Saatavissa:
https://angular.io/guide/architecture

Team Foundation Server. [Viitattu 2018-11-25]. Saatavissa:
https://docs.microsoft.com/en-us/previous-versions/visualstudio/visual-studio-2013/fda2bad5(v=vs.120)

C Sharp. [Viitattu 2018-12-2]. Saatavissa:
https://docs.microsoft.com/en-us/dotnet/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework

Angular. [Viitattu 2018-12-2]. Saatavissa:
https://en.wikipedia.org/wiki/Angular_(application_platform)

DP Logo. [Viitattu 2018-12-9]. Saatavissa:
http://www.dataprisma.fi/images/Data_Prisma.png

Angular vs React. [Viitattu 2018-12-16]. Saatavissa:
https://hackr.io/blog/angular-vs-react-2018

Introducing JSX. [Viitattu 2018-12-18]. Saatavissa:
https://reactjs.org/docs/introducing-jsx.html