Frontend koodin päivitys Angulariin

Juulia Jokinen

Haaga-Helia ammattikorkeakoulu

Tradenomi

Opinnäytetyö

2023

Tiivistelmä

Tekijä(t)
Juulia Jokinen

Tutkinto
Tradenomi

Raportin/Opinnäytetyön nimi
Frontend koodin päivitys Angulariin

Sivu- ja liitesivumäärä
28 + 3

Tämän toiminnallisen opinnäytetyön tavoitteena on aloittaa SmartPerso-Office ohjelmiston frontend koodin päivitys AngularJS:tä Angulariin. Tämänhetkisen AngularJS ohjelmistokehyksen tuki on lopetettu, eikä se siten ole enää luotettava käyttää. Myös jatkokehityksen kannalta on parempi siirtyä käyttämään uudempaa ja toimivampaa ohjelmistokehystä.

Opinnäytetyö on tehty Evry Card Services Oy:lle toimeksiantona. Evry Card Services on maksukorttien yksilöintiin erikoistunut yritys ja SmartPerso-Office on korttien yksilöintiprosessiin liittyvä hallinnollinen työkalu. Valikoituja kehityskohteita tälle opinnäytetyölle on autentikointi, toiminnallinen sivu, käyttäjien luonti sekä hallinnointi ja tyylien päivitys. SmartPerso-Office sisältää paljon muitakin toimintoja, mutta tässä opinnäytetyössä keskitytään vain edellä mainittuihin aiheisiin.

Uuden projektin aloitus alkoi taustaselvityksellä parhaista toteutustavoista sekä syvemmällä tutustumisella projektissa käytettäviin teknologioihin. Seuraavaksi oli vuorossa toiminnallisten osuuksien toteutus, yksi aihe kerrallaan edeten. Työn edistymisestä kirjoitettiin ylös muistiinpanoja myöhemmin toteutettavaa raportointivaihetta varten.

Projektin toteutus onnistui tavoitteiden mukaisesti ja lopputuloksesta saatiin tavoiteltu tuotos. Uusi Angular versio on jatkokehitettävämpi, kuin edeltäjänsä. Ulkoasua päivitettiin, mutta yleisilme pysyi samana, sillä tarkoituksena ei ollut uudistaa sitä täysin.

Projektia voi jatkaa opinnäytetyön valmistumisen jälkeen siirtämällä kaikki muutkin komponentit uuteen versioon yksitellen. Ennen lopullisen Angular version valmistumista olisi kuitenkin tarkoituksena yhdistää uusi sekä vanha versio toimimaan yhdessä yhtenä ohjelmistona väliaikaisesti.

Asiasanat
AngularJS, Angular, SmartPerso-Office

Sisällys

1 Johdanto

Opinnäytetyöni tulen toteuttamaan Evry Card Services Oy:n toimeksiannolla. Evry Card Services on maksukorttien yksilöintiin erikoistunut yritys, jolla on toimipisteet myös Latviassa ja Norjassa. Evry Card Services tarjoaa maksukorttien yksilöinnin lisäksi myös erilaisia korttipalveluita asiakkailleen. Evry Card Servicesillä on omia sisäisiä ohjelmistoja, joista yksi on SmartPerso-Office, josta tämä opinnäytetyö on tehty.

SmartPerso-Officen frontend on toteutettu jo kauan aikaa sitten ja AngularJS ohjelmistokehyksen ylläpitokin on jo lopetettu. AngularJS on päivitetty täysin uuteen Angular ohjelmistokehykseen, ja tässä opinnäytetyössä aloitan vanhan SmartPerso-Office koodin päivittämisen uuteen. Päivitystä tarvitaan, sillä SmartPerso-Officen jatkokehitys ei ole kannattavaa nykyisellään ja liian vanha AngularJS versio ei ole enää luotettava ohjelmistokehys. Projektin tavoitteena on saada toimivampi frontend SmartPerso-Officelle sekä parempi alusta jatkokehitystä varten. SmartPerso-Officen yleisilmettä haluttiin myös saada yhtenäisemmäksi muiden Evry Card Servicesin ohjelmistojen kanssa.

Opinnäytetyöni tarkoituksena oli aloittaa SmartPerso-Officen AngularJS frontend version päivittäminen uudempaan Angular versioon, yksi ohjelmistokomponentti kerrallaan. Tavoitteena oli saada luotua hyvä pohja SmartPerso-Officen uudesta frontend koodista, jotta myöhemmin loputkin ohjelmistokomponentit voidaan siirtää uuteen versioon ja vanhasta AngularJS koodista päästään kokonaan eroon. Tässä projektissa oli tavoitteena luoda autentikointi, käyttäjien hallinta, sivu- ja yläpalkki sekä yksi toiminnallinen sivu uuteen versioon. Myös ulkoasua uudistettiin projektin yhteydessä, mm. Angularin uuden komponenttikirjaston PrimeNg:n avulla. Yleisilme projektissa pysyi kuitenkin samankaltaisena edelliseen verrattuna. Projekti päätettiin toteuttaa luomalla täysin uusi Angular projekti, sillä AngularJS:n tarjoamat päivityskomponentit eivät sopineet yhteen vanhan koodin kanssa. Projektin tavoitteet toteutettiin yksi komponentti kerrallaan edeten.

Projektissa tulen oppimaan Angularia syvällisemmin, luomaan autentikoinnin sekä ulkoasun suunnittelua. Tulen myös syventymään hakemistorakenteiden luomiseen enemmän. Moni projektissa toteutettava asia on minulle entuudestaan tuttu, mutta nyt pääsen luomaan kaiken aivan alusta saakka. Opinnäytetyö on suunnattu muille kehitystyössä työskenteleville.

1.1 Käsitteet

Alasvetovalikko	Valikko, joka avautuu sitä klikkaamalla.
API	Ohjelmointirajapinta, jolla järjestelmät kommunikoivat keskenään.

Backend	Ohjelmiston palvelinpuoli, jota käyttäjät eivät näe.
CLI	Command Line Interface, eli komentoriviliittymä.
Checkbox	Valintaruutu.
CRUD	Toiminnot tiedon luomiseen, lukemiseen, muokkaamiseen ja poistamiseen.
Frontend	Ohjelmiston käyttöliittymäpuoli.
HTTP-tilakoodi	Kolminumeroisia lukuja, joilla seurataan HTTP-pyyntöjen tilaa.
ID	Identifier, tunniste.
JSON	Tiedostomuoto.
Modaali	Sivun päälle avautuva toinen ikkuna.
Routing	Reititysmäärittely.
SEO URL	Verkko-osoite, jonka linkistä selviää sen sisältöä.
Token	Objekti, joka antaa oikeuksia suorittaa operaatioita.
Widget	Käyttöliittymäkomponentti, yksittäinen elementti graafisessa käyttöliittymässä.

2 SmartPerso-Office

SmartPerso-Office on sisäiseen maksukorttien yksilöinnin sekä laskutuksen hallintaan käytetty ohjelmisto. Sitä käyttävät monet erilaisissa työtehtävissä työskentelevät ihmiset, mm. asiakaspalvelijat, operaattorit, tuotannon valvojat, tuotannon tuki sekä kehittäjät. SmartPerso-Officessa on monia eri toimintoja liittyen esimerkiksi laskutukseen sekä raportointiin, mutta ei toimintoja mitkä liittyisivät suoraan maksukorttien yksilöintivaiheeseen. Se on kuitenkin erittäin tärkeä työväline korttien yksilöintiprosessin hallinnassa.

Projekti alkoi syvemmällä tutustumisella käytettäviin teknologioihin, sillä oli päätettävä projektin toteutustapa. Lähes kaikki käytettävät teknologiat ovat minulle entuudestaan tuttuja, muutamaa lukuun ottamatta. Silti projektin aikana pääsen syventymään paremmin jo minulle tuttuihinkin teknologioihin. Ainoastaan AngularJS on minulle täysin uusi verkkokehys. Vaikkakin se on vanha ja nykyään harvemmin käytössä oleva verkkokehys, niin uskon, että siihen tutustuminen voi jatkossa helpottaa Angularin kanssa työskentelyä.

2.1 AngularJS

AngularJS on avoimen lähdekoodin käyttöliittymä, jolla rakennetaan lähinnä yksisivuisia verkkosovelluksia. Se käyttää MVC (model-view-controller eli malli-näkymä-käsittelijä) arkkitehtuuria. AngularJS on täysin laajennettavissa ja toimii hyvin muiden kirjastojen kanssa sekä sen jokaista ominaisuutta voidaan muokata yksilöllisen kehityksen ja tarpeiden mukaisesti. AngularJS on kirjoitettu JavaScriptillä, sillä TypeScript lisättiin vasta sen seuraavaan versioon, Angulariin. AngularJS kehitys on lopetettu ja sen tuki päättyi tammikuussa 2022. (AngularJS 2021, AngularJS 2022, GeeksForGeeks 2022a)

2.1.1 NgUpgrade ja LazyLoading

NgUpgrade on suunniteltu vanhemman AngularJS koodin päivittämiseen. Sen avulla voidaan sekoittaa Angular ja AngularJS komponentteja keskenään samassa sovelluksessa saumattomasti. Yksi sen tärkeimpiä ominaisuuksia on UpgradeModule, joka sisältää apuohjelmia hybridisovellusten hallintaan, jotka tukevat Angular ja AngularJS koodia. Kaikki koodi suoritetaan omissa kehyksissään, joten molempien kehysten ominaisuudet toimivat normaaliin tapaan. (Angular s.a)

LazyLoading on tapa, joka lykkää vaadittujen resurssien lataamista, ennen kuin niitä todella tarvitaan. Tämä lisää tehokkuutta, erityisesti kun käytetään montaa eri kehystä samassa sovelluksessa. Kun siirretään sovelluksia AngularJS:stä Angulariin käyttämällä hybridilähestymistapaa, halutaan siirtää ensin käytetyimpiä ominaisuuksia ja käyttää harvemmin tarvittuja ominaisuuksia ainoastaan tarvittaessa. (Angular s.a a)

2.2 Angular

Angular on komponenttipohjainen ohjelmistokehys, jolla voi rakentaa skaalautuvia verkkosovelluksia. Sillä on laaja kokoelma hyvin integroituja kirjastoja, jotka kattavat suuren valikoiman erilaisia ominaisuuksia, kuten lomakkeiden hallinnan ja reitityksen. Angularilla voi luoda sovelluksia aina yhden kehittäjän projekteista suurempiin yritystason projekteihin. Angular on luotu AngularJS:n pohjalta, mutta se on kuitenkin aivan erillinen verkkokehyksensä, eikä siksi AngularJS applikaatioita voi suoraan päivittää Angular versioon. (Angular s.a b)

Päivitetty Angular on myös paljon tehokkaampi, kuin edeltäjänsä. Yhtensä syynä tähän on ydintoimintojen siirtäminen moduuleihin sekä erillinen uusi CLI komentorivikäyttöliittymä. CLI:n avulla voi asentaa uusia paketteja, jotka helpottavat uusien komponenttien luomista. Se myös tekee monimutkaisesta koodista modulaarisen, jota on helpompi hallita. (GeeksForGeeks 2022b)

2.3 Mahdollisten päivitystapojen läpikäynti

Aluksi ennen projektin varsinaista aloitusta täytyi miettiä, mikä olisi paras tapa päivittää vanha koodi. Ensin kokeiltiin sopisiko ngUpgrade kirjasto, jonka avulla AngularJS ohjelman voi komponentti kerrallaan päivittää Angulariin. Molempia projekteja ajetaan samaan aikaan, mutta valitaan mitä komponentteja ajetaan AngularJS:llä ja mitä uudemmalla Angularilla. Tämän tavan käyttöönotto osoittautui kuitenkin lähes mahdottomaksi, sillä ngUpgrade on tarkoitettu lähinnä yksinkertaisemmille ohjelmille, jotka noudattavat AngularJS:n best practices malleja.

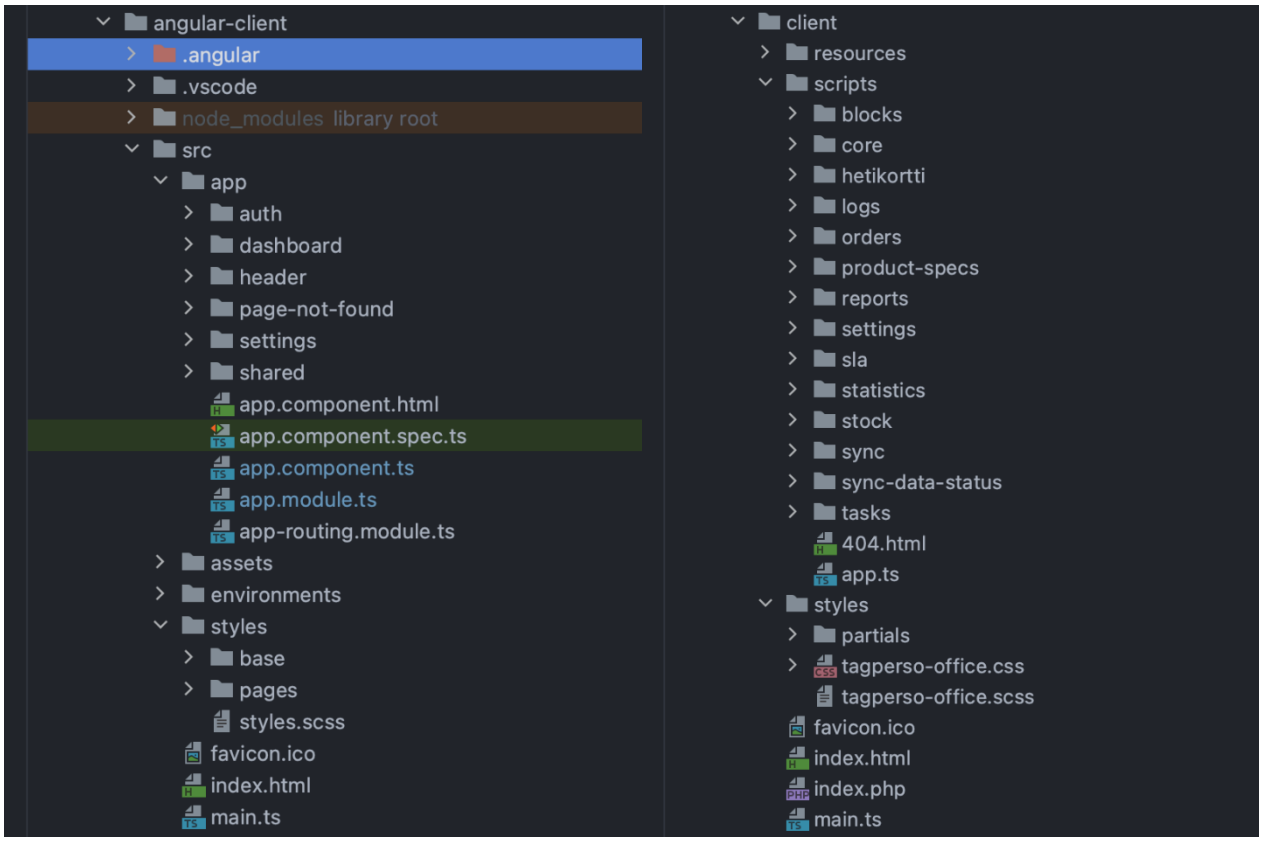
Kokeilussa oli myös LazyLoading, mutta sekään ei valikoitunut projektin toteutustavaksi samojen ongelmien takia, sillä molempien lisäosien käyttöönotto osoittautui aivan liian hankalaksi. Molemmat toteutustavat olisivat joka tapauksessa olleet myös vain väliaikaisia ja lopullinen uusi Angular projekti olisi pitänyt tehdä kuitenkin jossain vaiheessa. Projektia lähdettiin siis toteuttamaan luomalla aivan uusi Angular projekti, johon uutta frontendiä alettiin rakentamaan komponentti kerrallaan.

AngularJS:n koodi on lähes täysin erilaista, kuin uudemmassa Angularissa, jonka takia vanhaa koodia ei voinut suoraan käyttää uudemmassa versiossa hyödyksi. Monia osia oli kuitenkin mahdollista saada muista Evry Card Servicesin Angular ohjelmistoista.

2.4 Uusi Angular projekti

Loin uuden Angular projektin SmartPerso-Officen public kansioon Angular CLI:n avulla ja tyyliformaatiksi valitsin SCSS:n. Uusi angular-client niminen projekti sisältää kaikki komponentit app kansiossa, jossa kaikilla ominaisuuksilla on omat kansionsa, jotta tiedostorakenne pysyy selkeänä.

Vanha AngularJS versio säilyy samassa kansiossa uuden version kanssa omassa client kansiossaan. Lisäsin myös app kansion alle shared kansion, jonne luon mm. services kansion. Services kansio tulee sisältämään API-palveluita aina yksi service, eli palvelu kerrallaan. Palvelut ovat yleensä komponentti tai sivukohtaisia. Vielä lopuksi lisäsin environments kansion, joka sisältää projektiympäristöihin liittyviä määrittelyitä.



Kuva 1 ja 2. Ensimmäisessä kuvassa valmis uusi Angular kansiorakenne. Tässä kohtaa projektia kaikkia kuvassa näkyviä tiedostoja sekä kansioita ei ole vielä lisätty. Toisessa kuvassa vanhempi AngularJS kansiorakenne.

2.5 Uudenlainen hakemistorakenne

Kaikissa kohdissa tämän projektin aikana oli tarkoituksena seurata samankaltaista kansiorakennetta. Projekti sisältää pääkansion src, jossa määritellään projektin pääkomponentit app.component.ts, app.component.html, app.module.ts sekä app-routing.module.ts. Se sisältää myös kansion app, jossa sijaitsevat muut komponentit kansioittain. Lähes jokainen komponenttikansio sisältää HTML, module- sekä component tiedostot. Normaalisti mukana olisi vielä scss tyylitiedosto, mutta uuden tiedostorakenteen ansiosta niitä ei tarvita. HTML-tiedostossa määritellään sivujen asettelu sekä ulkoasu ja module tiedosto sisältää tarvittavia tuonteja sekä palvelun tarjoajia.

Component tiedosto sisältää moduulin toiminnallisuuden sekä funktiot. Joissain tiedostoissa on vielä erikseen routes tiedosto, joka sisältää reitityksen, jos kyseinen kansio sisältää useampia sivuja.

3 Autentikointi

Uuden projektin luomisen jälkeen päätin aloittaa projektin toiminnallisten tavoitteiden toteuttamisen autentikoinnista, jotta vaikein osuus olisi tehtynä ensimmäisenä. Autentikoinnin tavoitteena oli saada luotua toimiva todentamisjärjestelmä, jonka avulla käyttäjät pääsevät kirjautumaan sisälle SmartPerso-Officeen. Käyttäjätunnukset ovat henkilökohtaisia, paitsi kehitysympäristössä on olemassa vielä lisäksi yleisiä ylläpitäjän tunnuksia, joilla on oikeudet ohjelmiston kaikkiin asetuksiin ja sivuihin. Myöhemmin projektissa toteutettiin vielä käyttäjäoikeuksien lisäys, joiden avulla voidaan rajoittaa normaalien käyttäjien oikeuksia sivustoilla. SmartPerso-Officen uudessa versiossa käytetään samoja todentamismenetelmiä, kuin vanhemmassakin versiossa.

3.1 LDAP

LDAP eli Lightweight Directory Access Protocol on ohjelmistoprotokolla, jota käytetään tietojen tai laitteiden etsimiseen verkossa. LDAP:ia käytetään laajalti todennuspalvelimien rakentamiseen. Nämä palvelimet sisältävät käyttäjätunnukset sekä salasanat kaikille verkon käyttäjille. Sovellus muodostaa yhteyden LDAP-palvelimeen käyttäjien todentamiseksi ja valtuuttamiseksi. LDAP-hakemistot sisältävät yleensä tietoja, joita muutetaan harvoin. Siksi sen READ-suorituskyky suuremmillekin tietojoukoille on suunniteltu poikkeuksellisen nopeaksi. (Onelogin s.a)

LDAP-todennus on prosessi, jolla tarkistetaan hakemistopalveluun tallennetut käyttäjätunnukset sekä salasanat. Järjestelmänvalvoja roolin saaneet käyttäjät voivat luoda uusia käyttäjiä hakemistoon sekä myöntää muille käyttöoikeuksia. Resursseja käyttäessä lähetetään pyyntö LDAP-todennuspalvelimelle, joka tarkistaa syötetyn käyttäjätunnuksen sekä salasanan hakemistossa olevien tietojen perusteella. Jos vastaavuus löytyy, tarkistetaan seuraavaksi, onko käyttäjällä lupa pyytää resurssia. (Onelogin s.a)

3.2 JSON Web token

Autentikoinnissa on käytössä JSON Web token (JWT). Se on avoin standardi, joka määrittelee kompaktin ja itsenäisen tavan siirtää tietoja turvallisesti osapuolten välillä JSON-objektina. Näihin tietoihin voi luottaa, sillä ne ovat allekirjoitettu digitaalisesti. Jwt:t voidaan allekirjoittaa julkisen avaimen menetelmällä tai HMAC algoritmilla salaisuutta käyttäen. Jwt-tokenit sisältävät tietoja, kuten sen hetkisen istunnon käyttäjänimen. (JWT s.a)

Allekirjoitetut tunnukset, eli tokenit, voivat varmistaa sen sisältämien vaatimusten eheyden, kun taas salatut tunnukset piilottavat nämä vaatimukset muilta, kuin osapuolilta. Kun tunnuksia allekirjoitetaan käyttämällä julkisia tai yksityisiä avainpareja, allekirjoitus varmistaa, että vain yksityisen avaimen hallussa oleva osapuoli on sen kirjoittanut. (JWT s.a)

3.3 LocalStorage

LocalStorage on tallennustila, johon verkkosovellukset voivat tallentaa tietoja paikallisesti käyttäjän selaimeen. Aikaisemmin sovellustiedot piti tallentaa evästeisiin, jotka sisältyivät jokaiseen palvelinpyyntöön. Verkkotallennus on turvallisempi tapa ja suuriakin määriä tietoja voidaan tallentaa paikallisesti vaikuttamatta verkkosivun suorituskykyyn. Tietoja ei myöskään koskaan siirretä palvelimelle. (W3Schools s.a)

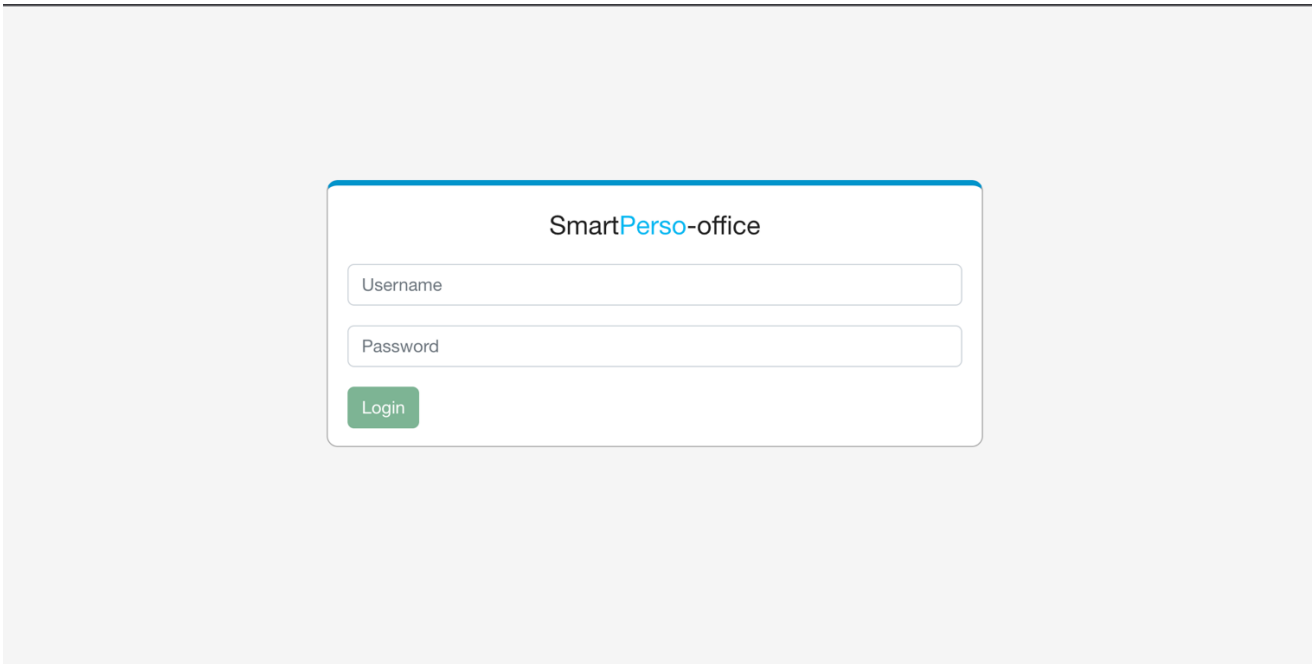
3.4 Autentikoinnin aloitus

Kaikki uuden frontendin komponentit sijaitsevat app kansiossa, jossa jokaisella toiminnallisuudella on oma kansionsa. Autentikointia varten sain kopioitua monia osia ja komponentteja muista Evry Card Servicesin Angular ohjelmistoista autentikoinnin sisältävään auth kansioon, mutta kaikki piti muokata vielä yhteensopiviksi. Lisäsin vielä services kansioon autentikoinnissa tarvittavan authservicen. Se on todennusjärjestelmä, jonka avulla haetaan tietoja projektin ohjelmointirajapinnasta. Uudessa SmartPerso-Office versiossa on käytössä vain LDAP tunnistautumismenetelmä. Joissain muissa Evry Card Servicesin ohjelmistoissa on saattanut olla käytössä myös muita tunnistautumismenetelmiä, mutta LDAP valikoitui, koska se on helppo käyttää.

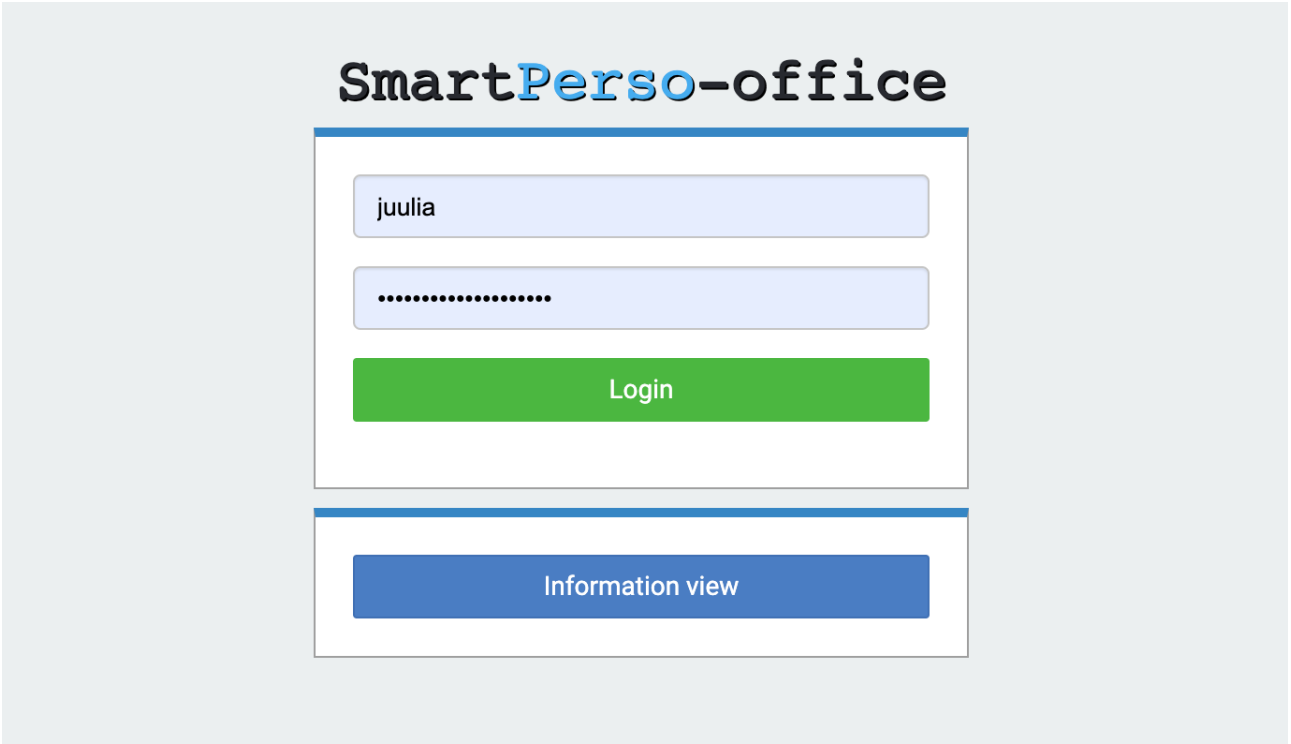
Auth kansiossa oleva auth.component.ts tiedosto sisältää mm. login() funktion, joka tarkastaa authservicen avulla onko käyttäjän syöttämät käyttäjänimi ja salasana oikein. Tietojen ollessa oikein, käyttäjä kirjautuu sisään ja funktio avaa, vielä tässä vaiheessa tyhjän, asetukset sivun. Onnistuneen sisäänkirjautumisen vastauksena login() funktio palauttaa jwt-tokeniin tarvittavat tiedot, jonka jälkeen loginSuccess() parsii Angularin JwtHelperServicen avulla tokenin ja tallettaa sen localStorageen.

Autentikoinnin koodaaminen osoittautui luultua helpommaksi, sillä montaa tiedostoa ei tarvinnut muokata lähes ollenkaan, koska autentikoinnin logiikka toimii samalla tavalla muissakin projekteissa. Piti vain olla tarkkana ja muistaa kopioida kaikki tarvittavat tiedostot, jotta kokonaisuus toimii yhdessä.

Kirjautumisen logiikan valmistuttua, siirryin kirjautumissivun toteutukseen. Käytin kirjautumissivun komponentteihin samoja tyylejä, kuin aikaisemmassakin AngularJS versiossa. Kirjautumissivun tyylit sijaitsevat pages kansion login.scss tiedostossa, jossa määrittelin mm. kirjautumiskomponentin koon ja värejä.



Kuva 3. Valmiin autentikoinnin kirjautumiskomponentti.



Kuva 4. Vanhan AngularJS version kirjautumiskomponentti.

Valmiista lopputuloksesta saatiin tavoiteltu päivitetty versio ja todentamismenetelmä toimii. Nykyisestä versiosta puuttuu vielä ’information view’ nappi, josta näkee toimiston infotauluilla olevaa infoa. Tämä ei kuitenkaan kuulunut opinnäytetyön rajaukseen, joten kyseistä toimintoa ei tässä

projektissa toteutettu. On kuitenkin mahdollista, että jatkossa sekin lisätään vielä kirjautumisnäkymään.

4 Navigointi ja tyylien päivitys

Projektin aloituksen ja autentikoinnin valmistumisen jälkeen täytyi alkaa pohtia perusteellisemmin SmartPerso-Officen uutta ulkoasua, kun vuorossa oli ylä- ja sivupalkkien toteutus. Ylä- ja sivupalkit ovat käyttäjällä näkyvissä jokaisella sivulla sisäänkirjautumisen jälkeen. Sivupalkin kautta navigoidaan SmartPerso-Officessa sivulta toiselle. Yläpalkki ei sisällä muuta toiminnallisuutta, kuin uloskirjautumisen vaihtoehdon.

Tarkoituksena oli pitää ulkoasu hyvin samankaltaisena, kuin aiemmassakin versiossa, mutta tarpeen tullen päivittää ja selkeyttää sitä. Otin monia tyylejä muista Evry Card Servicesin ohjelmistoista, jotta yleisilme kaikissa Angular ohjelmissa olisi samankaltainen. AngularJS versiosta sain haettua myös muutamia CSS tyylejä, mutta koodin eroavaisuuksien takia, montaa kohtaa ei voinut uuteen versioon hyödyntää. Aikaisemmassa SmartPerso-Office versiossa tyylitiedostot olivat hyvin hajautettuja, eikä niille ollut selkeää tiedostorakennetta. Tähän halusin tehdä muutoksen uudemmassa Angular versiossa kokoamalla kaikki tyylit saman kansion alle, jotta tyylitiedostoja ei löydy monesta eri paikkaa. Uusi tiedostorakenne selkeyttää jatkokehitystä tyylien osalta, kun kaikki löytyy samasta paikkaa ja samoja tyylejä on mahdollista käyttää monissa eri komponenteissa.

4.1 CSS

CSS (cascading style sheets) on verkkosivuille kehitetty tyylisivu, jota käytetään kuvaamaan HTML- tai XML-kielellä kirjoitettujen asiakirjojen esitystapaa. CSS kuvaa, kuinka elementit tulee renderöidä erilaisilla näytöillä. Se on yksi verkkokehityksen ydinkielistä ja se on standardoitu W3C-spesifikaatioiden mukaisesti kaikissa selaimissa. Sitä käytetään laajalti HTML:n ja JavaScriptin ohella. (MDN Web Docs 2022)

4.1.1 Flexbox

CSS Flexible Box Layout, eli Flexbox on yksiulotteinen asettelumalli ja menetelmä, jonka avulla voidaan tarjota tilanjaon käyttöliittymä elementtien välille. Yksiulotteisella tarkoitetaan, että flexbox käsittelee asettelua yhdessä ulottuvuudessa kerrallaan, joko rivinä tai sarakkeena. Elementit taipuvat ja täyttävät tilan tai tarvittaessa kutistuvat. Flexboxin kaksi akselia ovat pääakseli, jonka määrittää flex-direction ominaisuus sekä poikkiakseli, joka on kohtisuorassa pääakselia vastaan. (MDN Web Docs)

Kauan ainoat luotettavat sekä yhteensopivat työkalut CSS asettelujen luomiseen olivat vain ominaisuuksia, kuten elementtien paikannus. Tällaiset ominaisuudet olivat hyvin rajoittavia erilaisten asettelujen luomisessa. Flexbox luotiin helpottamaan asettelujen luomista ja tekemään siitä yksinkertaisempaa. (MDN Web Docs)

4.2 SCSS

SCSS (syntactically awesome style sheet) on taas CSS:n edistyneempi versio, jota kutsutaan myös Sassy CSS:ksi sen edistyneiden ominaisuuksien takia. Siinä on kaikki samat ominaisuudet, kuin CSS:sä, mutta SCSS tarjoaa mm. muuttujia, joilla voi lyhentää koodia. Nämä lisäominaisuudet tekevät SCSS:n kirjoittamisesta nopeampaa ja yksinkertaisempaa, kuin tavallisen CSS:n kirjoittamisesta. (GeeksForGeeks 2022c)

4.3 Bootstrap

Bootstrap on avoimen lähdekoodin työkalukokoelma responsiivisten verkkosivujen ja verkkosovellusten luomiseen. Se on suosituin HTML-, CSS- ja JavaScript-kehys verkkosivujen kehittämiseen ja Bootstrapillä luodut sivustot sopivat yhteen kaikkien selaimien kanssa. Bootstrapin avulla on helppo muokata mitä tahansa tyylejä, kuten tekstin väriä tai taustaväriä. (GeeksForGeeks 2022d)

4.4 Tyylien tiedostorakenne

SCSS:n ansiosta pystyin luomaan haluamani tiedostorakenteen tyyleille ja tuoda ne aina päätiedostoon saakka. Lisäsin styles kansion src kansion alle, jonne aloin kokoamaan muita tiedostoja. Styles kansio sisältää päätiedoston styles.scss sekä kaksi muuta kansiota, pages ja base. Base kansiossa on eri sivuilla toistuvia tyylejä ja pages kansiossa on sivukohtaisia tyylejä. Kaikki nämä tiedostot on tuotu päätiedostoon styles.scss, jonka kautta moduulit pääsevät niihin käsiksi. Päätiedosto styles.scss sisältää SmartPerso-Officen komponenttien asettelun, joka toistuu jokaisella sivulla.

Yleensä kaikissa komponentteja sisältävissä kansioissa on mukana myös niiden omat tyylit sisältävä tiedosto. Projektin edetessä poistin kaikki nämä tiedostot, sillä ne olivat tyhjiä. Uuden kansiorakenteen ansiosta projektissa ei tarvittu enää kansiokohtaisia tyylejä.

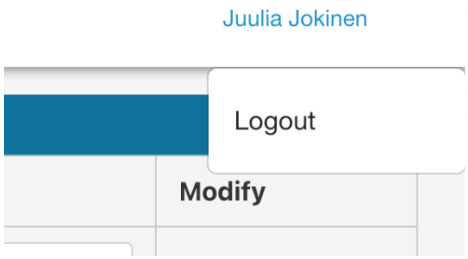
4.5 Yläpalkki

Ylä- ja sivupalkit sijaitsevat projektin juuressa app.component.html tiedostossa, jotta ne näkyvät jokaisessa näkymässä kirjautumisen jälkeen. Tämä tarkistetaan authServicen isAuthenticated funktiolla, joka tarkistaa löytyykö käyttäjältä jwt-tokenia. Ennen kirjautumista jwt-tokenia ei vielä ole, joten ylä- ja sivupalkit eivät näy kirjautumisnäkymässä.

Ylä- ja sivupalkkien luomisen aloitin lisäämällä header kansion app kansion alle. Header kansio sisältää kansiot user sekä sidebar. Header kansiossa on myös header.component.ts sekä header.component.html tiedostot, joista löytyy yläpalkin koodi. Molemmat tiedostot ovat lähes tyhjiä sillä yläpalkin toiminnallisuus löytyy user kansiosta ja sen tyylit ovat header.scss tiedostossa

base kansiossa. User kansion user.component.ts tiedostossa oleva ngOnInit funktio hakee jwt-tokenista käyttäjän nimen. NgOnInit funktio suoritetaan aina kyseisen sivun latautuessa heti ensimmäisenä, joten tässä kohtaa käyttäjän nimi asetetaan automaattisesti yläpalkkiin uudelle sivulle siirryttäessä, sillä yläpalkki on aina näkyvissä sisään kirjautuneena. Nimeä klikkaamalla avautuu alasvetovalikko, josta löytyy 'kirjaudu ulos' painike. Alasvetovalikko sulkeutuu painamalla käyttäjän nimeä uudestaan tai klikkaamalla mihin tahansa sen ulkopuolelle.

Aikaisemmassa AngularJS versiossa alasvetovalikosta löytyi myös asetukset kohta, jonka kautta pystyi muokkaamaan muutamaa käyttäjäasetusta, mm. käyttäjän aloitussivun valintaa. Tätä ominaisuutta en toteuttanut, sillä se ei kuulunut opinnäytetyöni rajaukseen.

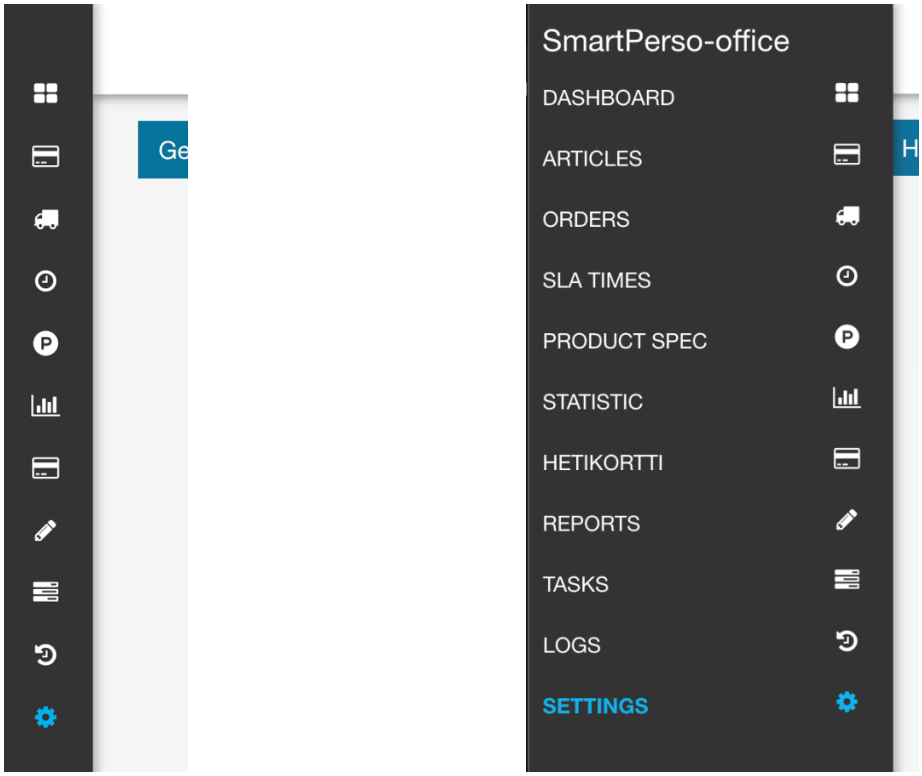


Kuva 5. Yläpalkki, jossa näkyy käyttäjän nimi. Sitä klikkaamalla käyttäjälle aukeaa uloskirjautumismahdollisuus.

4.6 Sivupalkki

Sivupalkin toiminnallisuudet sisältävästä sidebar kansiosta löytyy myös sidebar-closed.directive.ts tiedosto, joka kuuntelee sivua Angularin hostListenerin avulla. Tiedoston funktiot seuraavat, milloin osoitin osuu sivupalkin päälle, jolloin se avautuu automaattisesti. Muuten sivupalkki on suljettuna, kun käyttäjän osoitin on muualla. Sidebar.component.ts tiedostoon lisäsin listaan kaikki tarvitut linkit, jotka sidebar.component.html listaa sivupalkille. Component tiedostosta löytyy myös hasRole funktio, joka tarkistaa onko käyttäjällä oikeuksia kyseiselle sivulle. Tässä kohtaa käyttäjät pääsevät käsiksi kaikille sivuille, sillä en ole vielä tässä vaiheessa projektia luonut käyttäjien ja roolien hallintaa. Myöhemmässä vaiheessa lisään asetukset sivulle tarvittavan 'ROLE_ADMIN' oikeuden, jolloin hasRole estää kaikkien käyttäjien pääsyn asetukset sivulle, joilla kyseistä roolia ei ole.

Sivupalkista tuli lähes täysin samankaltainen, kuin edellisessäkin versiossa. Pidin linkkien järjestyksen sekä ikonit samoina, kuin aiemmin, jotta navigointi uudessa versiossa olisi yhtä helppoa. Erona aikaisempaa on synkronointipaketin aikaleima, joka on siirretty dashboardille. Synkronointipaketilla haetaan uudet tiedot tuotannosta tunnin välein SmartPerso-Officeen, sillä kaikki sen data ei päivity automaattisesti. Uudessa versiossa sivupalkki on myös osittain piilotettuna, kun taas vanhassa AngularJS versiossa sivupalkki oli koko ajan näkyvissä sellaisenaan.



Kuva 6. ja 7. Sivupalkit avattuna ja suljettuna.

Kuvassa 6. sivupalkki on kiinni, mutta siitä kuitenkin näkyy valkoisella ikonit. Sininen väri merkitsee senhetkistä sivua, jolla käyttäjä on. Tällä hetkellä kaikki sivut, ’settings’ kohtaa lukuunottamatta, vievät ’page-not-found’ virhesivulle, sillä ne eivät sisällä toiminnallisuutta. Page-not-found kansion tiedostot ovat tyhjiä ja vain HTML tiedostossa on yksi rivi, jolloin sivulle ilmestyy teksti ’page-not-found’. Siirtyminen virhesivulle on määritelty app-routing.module.ts juuritiedostossa, jolloin kaikki sivut, joita ei ole määritelty, vievät ’page-not-found’ komponentille. Tällä estetään projektin kaatuminen.

5 Käyttäjienhallinnan asetukset

Sovelluksen pohjan luomisen jälkeen oli aika siirtyä jatkamaan muita tavoitteessa määritettyjä toiminnallisuuksia. Päätin jatkaa käyttäjien hallinnasta, sillä en ollut vielä varma minkä muun toiminnallisen sivun haluan toteuttaa. Käyttäjienhallinta löytyy asetukset sivulta, joka sisältää myös kaikki muut SmartPerso-Officeen liittyvät säädöt. Sivulla on navigointipalkki, jonka välilehdiltä asetukset löytyvät kategorioittain. Käyttäjienhallinta sijaitsee omalla ’users’ välilehdellään. Sivulle siirryttäessä eteen avautuu taulukko, joka sisältää kaikki olemassa olevat käyttäjät. Käyttäjien tietoja voidaan muokata eteen aukeavan modaalin kautta. Uutta käyttäjää lisätessä aukeaa sama modaali, ilman ennalta täytettyjä kenttiä. Käyttäjiä voi suodattaa taulukkoon valittujen tietojen perusteella niiden yläpuolelta löytyvien hakupalkkien avulla. Kaikilla käyttäjillä ei kuitenkaan ole oikeuksia hallinnointi sivuille, vaan sitä varten tarvitaan ylläpitäjän oikeudet. Näitä oikeuksia voi muuttaa käyttäjien hallinnassa käyttäjää muokatessa.

5.1 PrimeNg

PrimeNg tarjoaa runsaasti erilaisia käyttöliittymäkomponentteja Angularille. Kaikilla widgeteillä on avoin lähdekoodi ja ne ovat ilmaisia käyttää MIT-lisenssillä. Komponentit ovat suunniteltu käyttökokemusta ajatellen ja niitä voi muokata vielä omiin tarpeisiin sopiviksi. (PrimeNg s.a.)

5.2 HTTP-interceptor

HTTP-interceptor käsittelee HTTP pyyntöjä sekä vastauksia. Interceptorit muuntavat lähtevän pyynnön ennen sen välittämistä ketjun seuraavalle interceprorille. Interceptorit välittävät tokenin jokaisessa HTTP-kutsussa backendille. (Angular s.a c)

5.3 Microsoft Active Directory

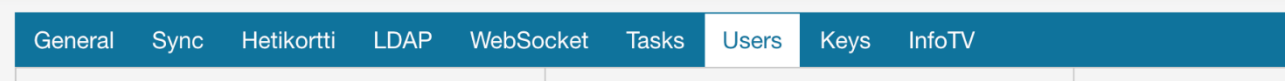
Microsoftin Active Directory Domain Services (AD DS) on hakemistopalvelu, joka tarjoaa menetelmiä tietojen tallentamiseen. Hakemisto on hierarkkinen rakenne, joka tallentaa tietoja verkon objekteista. Hakemistopalveluun voi tallentaa esimerkiksi käyttäjätietoja, kuten salasanoja, käyttäjänimiä, puhelinnumeroita tai muita tietoja. Nämä tiedot ovat saatavilla verkon käyttäjille ja järjestelmävalvojille ja se mahdollistaa muiden valtuutettujen käyttäjien pääsyn näihin tietoihin samassa verkossa. (Microsoft 2022)

5.4 Asetukset sivu

Asetukset sivun rakentaminen täytyi aloittaa luomalla uusi settings kansio app kansion alle, jonne kaikki asetukset sivulle tarvittavat toiminnallisuudet lisätään. Aloitin luomalla asetukset sivun, jossa myös käyttäjien hallinta sijaitsee. Sivulla on myös muita asetuksia, jotka on luokiteltu eri

välilehdille. Sama ominaisuus löytyy myös vanhasta AngularJS versiosta, joten loin asetukset sivulle erillisen navigointipalkin, jonka kautta valittua välilehteä voi vaihtaa.

Sivupalkista alinta ’settings’ kohtaa klikkaamalla avautuu ensin asetukset sivun ’general’ näkymä. Tällä hetkellä general sivulta ei löydy toiminnallisuutta sekä muutkin asetukset sivun välilehdet vievät ’page-not-found’ sivulle. Tässä vaiheessa ainoastaan ’users’ välilehdeltä löytyy toiminnallisuutta. Navigointipalkissa on samat toiminnallisuudet täysin samassa järjestyksessä, kuin aikaisemmassakin versiossa. Navigointipalkin tyylit sijaitsevat base kansion base.scss tiedostossa, jotta se on helppo lisätä tulevaisuudessa myös muille sivuille. Rakensin navigointipalkin käyttämällä flexboxia. Sinisellä näkyvä taustapaneeli, eli container, on taustalla ja sen sisään on asetettu välilehdet, eli itemit. Välilehdet on asetettu settings.component.html tiedostossa ja sivu vaihtuu välilehteä painamalla router-outletin avulla, jolloin navigointipalkki jää näkyviin. Router-outlet täyttää dynaamisesti sille valitun paikan reitittimen sen hetkisen tilan perusteella.



Kuva 8. Asetukset sivun välilehdet. Valkoisella rajattu välilehti merkitsee valittua sivua, jolla käyttäjä on sillä hetkellä. Viemällä osoittimen navigointipalkin päälle, välilehdet muuttavat väriä :hover efektin avulla.

5.5 Käyttäjien hallinnointi

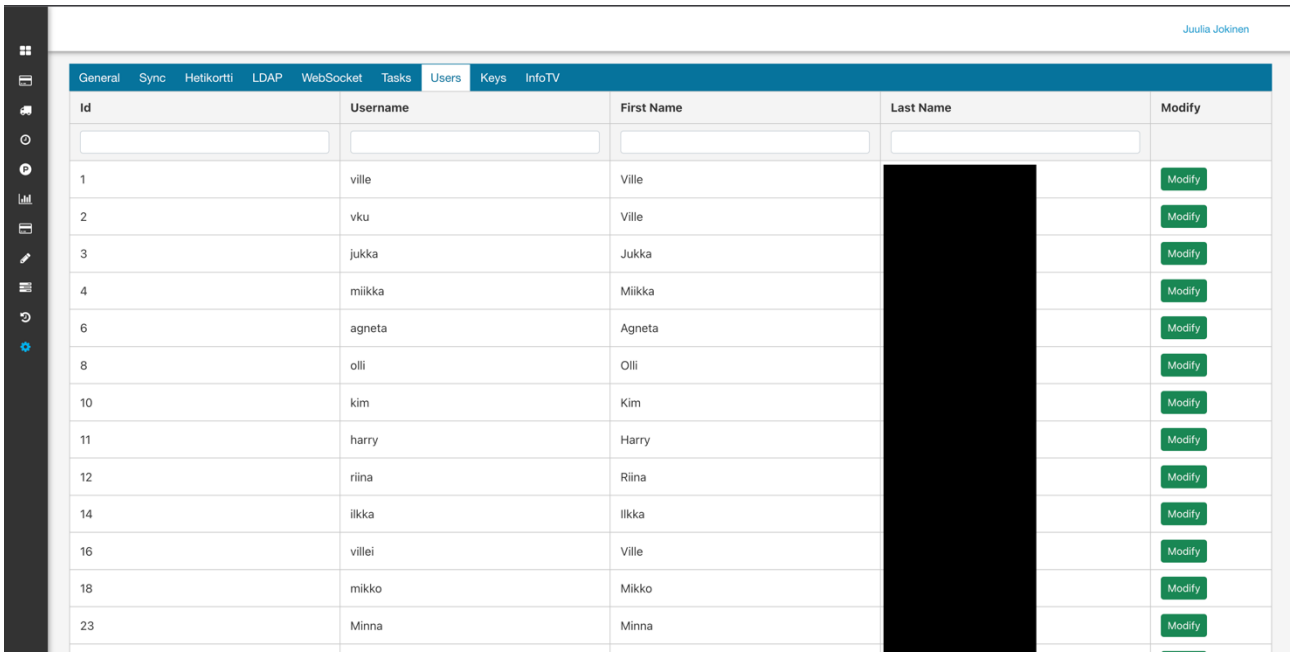
Käyttäjien luomisen aloitus sujui hyvin. Monia osia sain taas haettua muista ohjelmistoista ja ne piti vain saada muokattua yhteensopiviksi uuden SmartPerso-Office version kanssa. Loin users kansion settings kansion alle, mutta users kansioon tulisi myöhemmin vielä omat kansionsa käyttäjien lisäykselle ja muokkaamiselle. Tässä vaiheessa keskityin kuitenkin vain käyttäjätaulukon luomiseen.

Pian aloittamisen jälkeen alkoi esiintyä ensimmäisiä ongelmia. Ensin en saanut käyttäjiä haettua API-endpointista ja vastauksena tuli vain 403 forbidden HTTP-tilakoodi. Tässä kohtaa autentikointi kuitenkin toimi ja jwt-tokenin toteutus oli myös tehty oikein, joten en ollut varma mistä ongelma johtui. Selvisi kuitenkin, että minulla ei ollut HTTP-interceptoria projektissa vielä ollenkaan. Interceptor luokka tarkistaa onko tokenia tallennettu LocalStorageen, jonka jälkeen HTTP-interceptor lisää tokenin HTTP-kutsuun ja välittää sen serverille. Lisäsin shared kansioon interceptors kansion jonne lisäsin http.interceptor.ts tiedoston. Sain onneksi haettua sen tarvitseman sisällön toisesta projektista, jolloin piti enää vain tuoda se vielä app.module.ts tiedostoon. Tämän jälkeen sain haettua käyttäjät ja pääsin aloittamaan users välilehden käyttäjätaulukkoa.

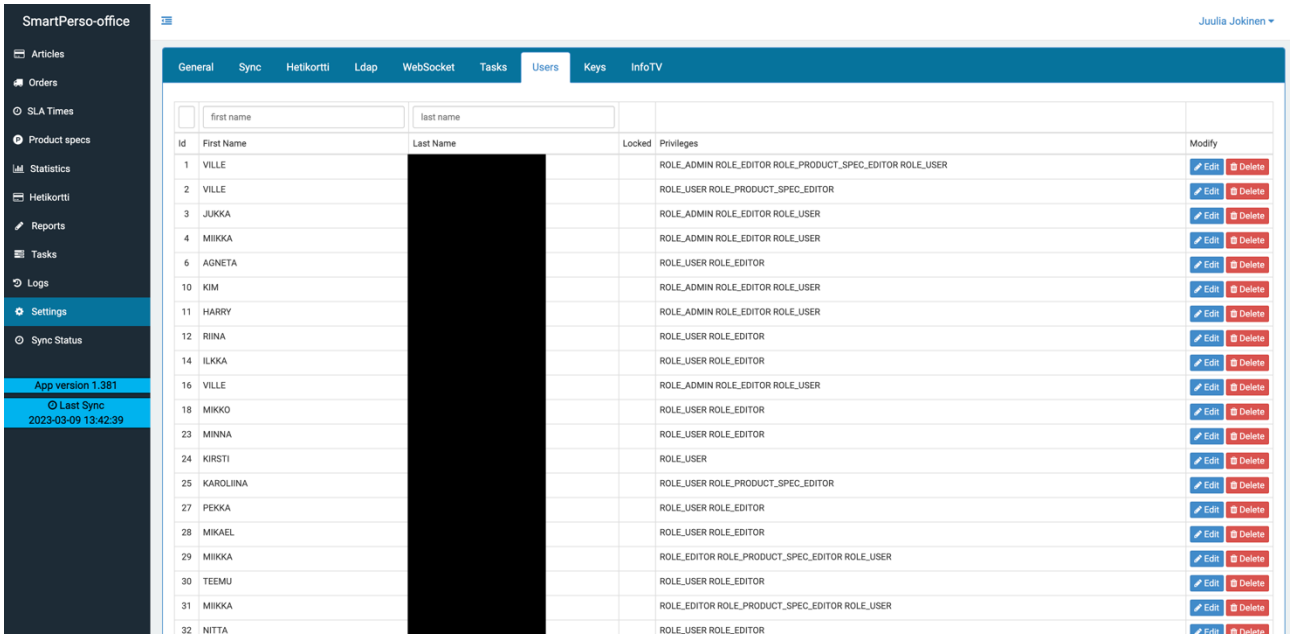
Käyttäjätaulukon pohja oli minulla jo valmiina toisesta projektista, koska PrimeNg:n ansiosta pystyin käyttämään täysin samoja komponentteja. Users.component.ts tiedostossa ohjasin ngOnInit funktiossa datan oikeisiin kohtiin. Taulukkoon valitut tiedot tallentuvat cols array taulukko tietorakenteeseen, jonka kautta data asetetaan automaattisesti käyttäjätaulukkoon. Käyttäjän tietoja varten loin shared kansion alle models kansioon user.ts tiedoston. Se sisältää kolme eri TypeScript tyyppiä Privilege, EmailEvents ja User. Näitä apuna käyttämällä on helpompi sitoa domain puolen logiikka API-puolen JSON-dataan. Moduulin avulla tietojen käyttäminen on helppoa ja se on erotettu palvelinpuolen arkkitehtuurista. Asetin käyttäjän tiedot samalla user.ts tiedoston User tyyppiin, käyttäjäoikeudet Privileges tyyppiin ja sähköposti ilmoitukset saman luokan EmailEvents tyyppiin. Näitä kolmea en vielä tässä vaiheessa tarvinnut, sillä käyttäjäoikeuksia ja sähköposti ilmoituksia ei listata käyttäjätaulukossa. Niitä kuitenkin tarvitaan myöhemmin, joten oli järkevää luoda ne jo tässä vaiheessa.

Seuraavat ongelmat ilmenivät pian, sillä PrimeNg teemat eivät asettuneet käyttäjätaulukkoon. Ongelmaa selvitettäessä ymmärsin, että kaikki muu toimi kuin pitikin, mutta en ollut tuonut kaikkia tarvittavia komponentteja app.component.ts tiedostoon, eikä tyylit siksi asettuneet kunnolla. Piti siis vielä lisätä PrimeNg:n sekä teeman URL-osoitteet tuonteihin, jotta teemat saadaan käyttöön taulukkoon.

Lopuksi lisäsin vielä käyttäjien suodatus mahdollisuuden taulukkoon valittuihin tietoihin, jossa jokaisella sarakkeella on oma suodattimensa. Tietojen suodatus tapahtuu syöttämällä hakusana valittuun sarakkeeseen, jonka jälkeen PrimeNg listaa automaattisesti vain kyseisen tiedon sisältävät käyttäjät. Tämä onnistuu PrimeNg:n valmiin ominaisuuden avulla. Vanhassa AngularJS versiossa käyttäjätaulukon suodatus oli rakennettu alusta alkaen itse, sillä AngularJS ei tue PrimeNg:tä. Normaalisti käyttäjät ovat listattuna ID:n mukaan.



Kuva 9. Asetukset sivun käyttäjienhallinta.



Kuva 10. Vanha AngularJS version käyttäjienhallinta välilehti.

Vertailussa uusi ja vanha käyttäjienhallinta. Pieniä muutoksia on havaittavissa esim. uudessa versiossa ’delete’ nappula on siirretty ’modify’ nappulan taakse. Listassa näkyy myös listattuna hieman eri tietoja. Yleisilme ja värimaailma on kuitenkin hyvin samankaltainen, pienistä muutoksista huolimatta.

5.6 Käyttäjä modaali

Aloitin lisäämällä user-new ja user-edit kansiot users kansion alle. Ne sisältävät tarvittavat toiminnallisuudet käyttäjien lisäämiseen sekä niiden muokkaamiseen. Molemmat kansiot sain haettua muista projekteista ja niiden tiedostoista piti vielä muokata SmartPerso-Officeen yhteensopivat. Aloitin käyttäjän muokkausmodaalista, sillä sen valmistuttua on helpompi kopioida sieltä tarvittava koodi uuden käyttäjän luomista varten, ilman käyttäjätietojen hakua.

Kaikki API-endpointit olivat jo valmiita, mutta reititys piti saada vielä oikein. Modify nappulaa painamalla pitäisi avautua valitun käyttäjän tiedot. Aikaisemmin User tyyppiin asetetut tiedot siirtyvät modaalin kenttiin users.component.html tiedostossa olevan PrimeNg:n p-dialogin avulla. Kun sain datan haettua modaalille, pystyin aloittamaan sen sisällön muokkaamisen. HTML-tiedostossa oli jo tarvitsemiani tekstikenttiä, joten lisäsin niihin oikeat tiedot.

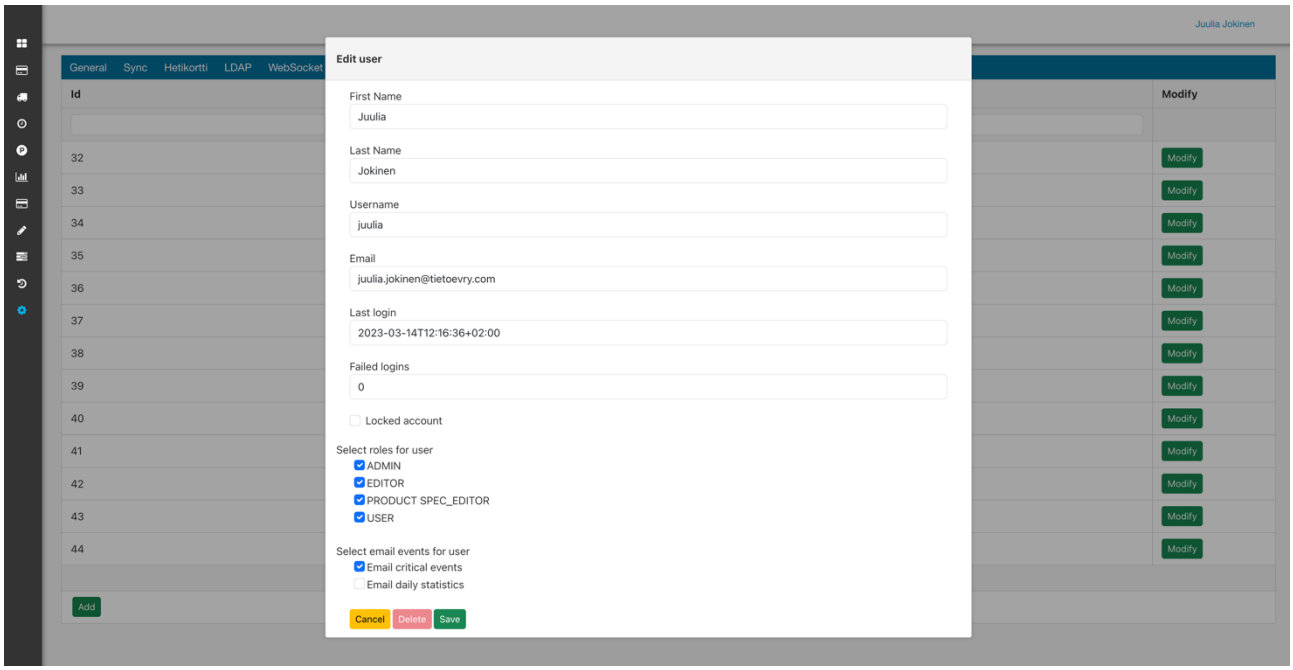
Viimeiset ongelmat olivat käyttäjien roolien ja sähköposti ilmoitusten kanssa. Myös muissa projekteissa käyttäjien hallinnoinnissa oli hyvin samankaltaisia kohtia, mutta SmartPerso-Officen backend oli rakennettu eri tavoin ja siksi täytyi keksiä täysin uusi tapa, kuinka roolit ja sähköposti ilmoitukset saadaan toimimaan.

Käyttäjämodaalin auetessa initForm() funktio asettaa User tyypin tiedot oikeisiin kohtiin userEditFormille, joka modaalilla näytetään. Sähköposti ilmoitukset ja käyttäjäoikeudet täytyy käydä läpi, jotta tiedetään mitä kohtia käyttäjällä on jo valittuina. Ensin käydään läpi Privileges tyypistä tulevat tiedot, jossa sijaitsee kaikki mahdolliset roolit. Seuraavaksi käydään läpi User tyypin mukana tulleet käyttäjän roolit ja jos samoja käyttäjäoikeuksia löytyy, valitaan sen roolin arvoksi true. Tällöin modaalissa roolien kohdalla, joiden arvo on true, näkyvät käyttäjällä jo valittuina checkboxissa. Molemmissa käytetään samaa logiikkaa valintojen tarkastamiseen, joten sähköposti ilmoitukset tarkastetaan samalla tavalla. Lisäksi rooleja läpikäydessä niistä poistetaan ’ROLE_’ etuliite, jotta yleisilme pysyisi siistimpänä ja käyttäjäoikeuksien nimiä olisi helpompi ymmärtää.

Nyt ongelmaksi kuitenkin muodostui käyttäjäoikeuksien ja sähköposti ilmoitusten muoto, eivätkä ne tällä hetkellä käy sellaisenaan vietäväksi backendin koodiin. Kun käyttäjä tallennetaan, käyttäjäoikeudet ja sähköposti ilmoitukset käydään samalla tavalla läpi ja valitut arvot listataan uuteen listaan. Lopuksi uusi lista tallennetaan käyttäjän tietojen paikalle, jotta data olisi oikeanlaista ja backend pystyy tallettamaan oikeat arvot tietokantaan.

```
1 usage  Juulia Jokinen
checkPrivileges (usr) {
    const roleList = [];
    const keys = Object.keys(usr.privileges);
    keys.forEach((key : string, index : number ) => {
        if (usr.privileges[key] === true) {
            this.roles.forEach(role => {
                if (role.name === key) {
                    roleList.push(role)
                }
            })
        }
    });
    usr.privileges = roleList;
}
```

Kuva 11. Esimerkkinä käyttäjäoikeuksien läpikäynti.



Kuva 12. Käyttäjämodaali. Kuvassa käyttäjänmuokkaus.

Lopullinen toteutus käyttäjämodaalille näyttää yllä olevalta. Kuvassa näkyy jo olemassa olevan käyttäjän muokkaus, mutta myös uutta käyttäjää lisätessä modaali on identtinen. Ainoana erona uutta käyttäjää lisätessä on salasana kenttä. Salasanakenttä oli vanhassa AngularJS versiossa näkyvillä myös jo olemassa olevaa käyttäjää muokatessa. Uuteen Angular versioon käyttäjän salasanan vaihtaminen päätettiin ottaa pois, sillä SmartPerso-Officen käyttäjien salasanat tallennetaan Microsoftin Active Directoryyn. Käyttäjillä on AD DS:n ansiosta samat salasanat käytössä

muissakin Evry Card Servicesin ohjelmistossa, mutta käyttäjillä ei ole mahdollisuutta vaihtaa salasanaa SmartPerso-Officen kautta. Kun salasana vaihdetaan muualta, AD DS vaihtaa kaikkien ohjelmistojen salasanat automaattisesti uuteen salasanaan. Samaa pohjaa eri modaaleille on helppo hyödyntää tulevaisuudessa projektia jatkaessa, sillä se on täysin muokattavissa erilaisiin tarpeisiin, mutta yleisilme pysyy samana. Modaalin valmistuttua pystyin myös lisätä sivupalkin sidebar.component.ts tiedostoon käyttäjäoikeuksien tarkistuksen asetukset sivun linkin kohdalle, jotta vain ’ROLE_ADMIN’ käyttäjät pääsevät käsiksi asetukset sivuun.

Käyttäjiin liittyviä CRUD-toimintoja ei tarvinnut tässä projektissa pohtia. CRUD, eli Create, Read, Update ja Delete ovat tiedon tallennuksen ja muokkauksen eri toimintoja. Projektissa on käytössä sama koodi backendissä, kuin aikaisemmassakin AngularJS versiossa, joten tallennuksen toiminnoista ei tarvinnut huolehtia. Data täytyy vain lähettää samanmuotoisena samoihin API-endpointteihin, kuin AngularJS versiossakin ja backendin koodi hoitaa yhteyden tietokantaan.

6 Toiminnallinen sivu

Viimeisenä toteutuksena oli jäljellä yksi toiminnallinen sivu. Tähän valikoitui täysin uusi sivu, dashboard, eli työpöytä. Tarkoituksena oli luoda näkymä, joka aukeaa käyttäjälle heti hänen kirjautuessaan sisään. Työpöydällä on valitut widgetit, jotka sisältävät yleisimpiä, mahdollisimman monia käyttäjiä kiinnostavia tietoja. Tätä kyseistä sivua ei siis ollut vielä aikaisemmassa AngularJS versiossa ollenkaan. Dashboard on kuitenkin hyvin yleinen sivu web sovelluksissa ja ajattelin sen sopivan SmartPerso-Officeenkin, sillä se on lisätty myös muutamaan muuhun Evry Card Servicesin ohjelmistoon. Tässä kohtaa projektia päädyttiin tekemään myös poikkeus ja päivitettiin myös backendin koodia.

6.1 Symfony

Symfony avoimen lähdekoodin hajautettu PHP-kehys. Se on innovatiivinen ja helppokäyttöinen sekä myös kansainvälisesti tunnustettu vakaa kehitysympäristö. Symfonyn avulla PHP:n kanssa työskentely on helpompaa, sillä se helpottaa pitkän aikavälin ylläpitoa ja skaalautuvuutta noudattamalla standardikehityssääntöjä. Kehitysstandardien noudattaminen yksinkertaistaa sovelluksen integrointia ja sen liittämistä muuhun tietojärjestelmään. (Symfony s.a)

Symfony routing vastaa ohjelmiston reititysmäärittelyistä. Reititysmääritys määrittää, mikä toimintoja suoritetaan kullekin saapuvalle URL-osoitteelle. Se tarjoaa myös muita hyödyllisiä ominaisuuksia, kuten SEO URL-osoitteiden luomisen. (Symfony s.a)

Symfony caching, eli välimuistikomponentti, tarjoaa ominaisuuksia, jotka kattavat yksinkertaisista edistyneempiin välimuistitarpeisiin. Se on suunniteltu suorituskykyä ja joustavuutta varten ja sen mukana toimitetaan käyttövalmiit sovittimet yleisimpiin välimuisteihin. (Symfony s.a)

6.2 Dashboard

Aluksi aloin pohtia dashboardin sisältöä ja asettelua. Dashboardin tietoihin valikoitui lopuksi synkronointipaketin viimeisin ajankohta, kahden viimeisimmän kuun korttien yksilöintitilastot, aktiiviset palvelutasosopimukset ja vialliset kortit prosentteina sekä tuotantojono. Kaikki muut edellä mainitut tiedot haetaan välimuistista, paitsi synkronointipaketin aikaleima tallentuu käyttäjän jwt-tokeniin, josta se haetaan erikseen dashboard näkymään siirtyessä.

Dashboardin data voi muuttua vain tietyin väliajoin ja jotta API-kutsujen kuormitus pysyisi maltillisena, sen tiedot tallennetaan Symfony cache komponentin avulla yleiseen MemCache välimuistiin. Välimuisti on sama kaikille käyttäjille ja tiedot haetaan aina ensisijaisesti sieltä. Synkronointipaketin uusiutuessa välimuisti tyhjennetään automaattisesti, jonka jälkeen seuraava käyttäjä, joka avaa

dashboard sivun, hakee uuden päivitetyn datan API-endpointista ja tallettaa sen samalla MemCacheen.

Dashboardin muita tietoja varten täytyi luoda uusi API-endpoint backendin puolelle. Projektissa ei ollut tarkoitus tehdä mitään muutoksia backendin puolelle, mutta dashboardille valittu data oli hyvin hajallaan. Siksi tästä ideasta poikettiin ja päädyttiin luomaan backendin puolelle uusi API-endpoint vain dashboardia varten, jotta kaikki tarvittava data saadaan haettua kerralla.

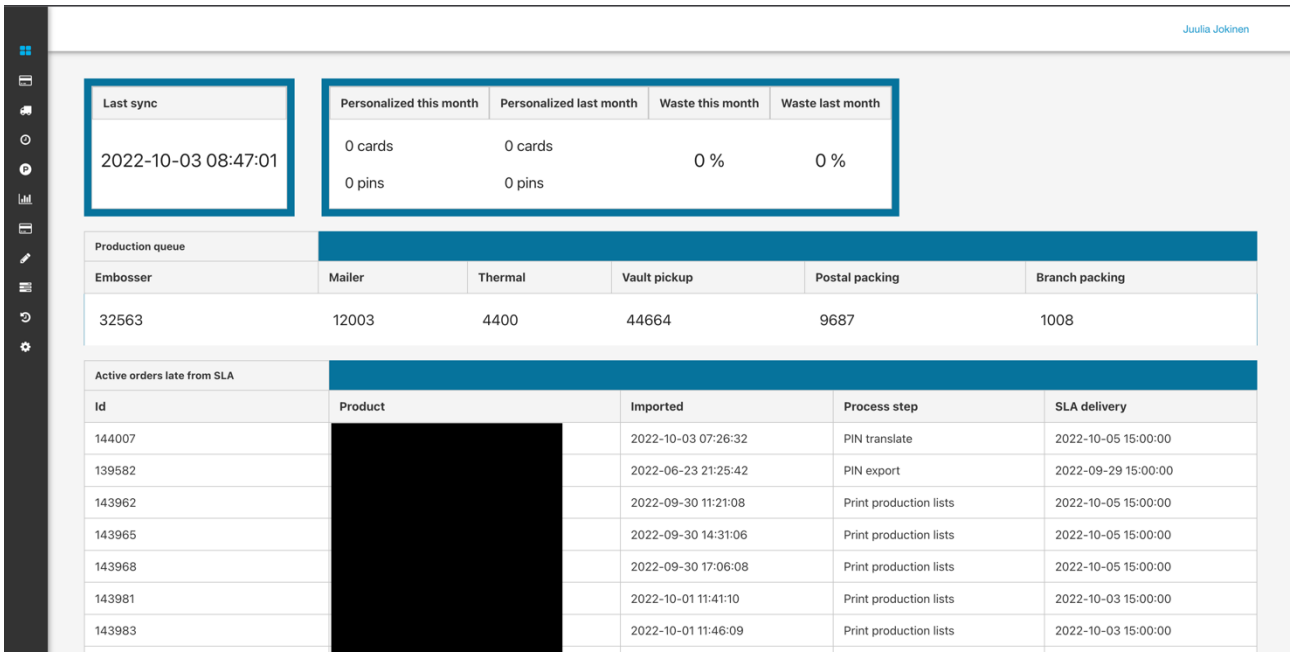
6.3 Toteutus

Päätin aloittaa toteutuksen API-endpointista, jotta ei tarvitsisi jälkikäteen muokata enää rajapintahakuja. Loin uuden DashboardController.php tiedoston Symfonyn Controller kansioon, missä myös muut controllerit, eli ohjaimet, sijaitsevat. Controller tiedostojen PHP-funktiot lukevat Request-objektien tietoja ja palauttavat Response-objektin, joka on tässä tapauksessa JSON dataa. DashboardController.php ei sisällä muuta, kuin yhden funktion, jossa dashboardin data haetaan. Funktio tarkistaa ensin löytyykö välimuistista dashboard.info kohdan alta jo dataa. Jos osumia ei löydy, haetaan tarvittava data tietokannasta, tallennetaan se välimuistiin ja lähetetään se takaisin Angularille. Jos dashboard.info kuitenkin jo sisältää dataa, tiedot haetaan suoraan sen alta välimuistista.

Seuraavaksi oli vuorossa sivun widgettien asettelu. Loin dashboard sivulle pohjan flexboxilla, jotta widgettien sijoittaminen sivulle olisi helpompaa. Palvelutasosopimuksille loin yhdessä bootsrapin ja PrimeNg:n p-table komponentin avulla taulukon, jossa on listattuna tärkeimpiä tietoja. Tiedot on haettu samalla tavalla, kuin käyttäjäsivun taulokossa asettamalla ne cols array taulukkoon ngOnInit() funktiossa. Loppujen widgettien tiedot haettu ja asetettu muuttujiin samassa funktiossa. Nämä tiedot ovat listattu manuaalisesti HTML-tiedostossa.

Widgettien tyyleissä käytin jo aikaisemmin luomiani tyylejä hyödyksi, mm. otsikkojen sininen tausta on peräisin navigointipalkin pohjasta. Kaikissa dashboardin komponenteissa esiintyy joitain samoja tyylejä. Loin dashboardia varten kuitenkin erilaisia tyylejä dashboard.scss tiedostoon. Esimerkiksi synkronoinnin ajankohtaan sekä korttien yksilöintitilastoihin käytin samoja tyylejä, sillä molemmat widgetit ovat hyvin samankaltaiset. Samoista tyyleistä loin eri versioita taulukoita varten, sillä ne sisältävät paljon enemmän dataa ja siksi widgettienkin täytyi olla isompia.

Dashboardin valmistuttua vaihdoin vielä sen aloitussivuksi kirjautumisen jälkeen app-routing.module.ts tiedostoon. Tähän mennessä aloitussivuna toimi asetukset sivun general välilehti, mutta toiminnallisen osuuden valmistuttua oli aika vaihtaa oikea sivu aloitussivuksi.



Kuva 13. Valmis dashboard näkymä.

Lopullisesta dashboardista tuli sen kaltainen, kuin olin ajatellutkin. Valitut tiedot ovat hyvin yleisiä ja siksi moni käyttäjä hyötyy niistä. Widgettien tyylit sopivat toisiinsa ja niissä on selvästi yhtenäinen ilme, mutta eroja silti löytyy asettelussa, jotta sivu ei olisi liian yksitoikkoinen.

7 Pohdinta

Koko projekti onnistui mielestäni hyvin. Pysyin omissa tavoitteissani ja aikataulukin piti koko projektin ajan. Suurempia ongelmia tai vastoinkäymisiä ei ilmennyt alun taustaselvityksien jälkeen ja projekti eteni koko ajan tasaiseen tahtiin. Projektin toteutus oli mielekästä ja opin uusia asioita ohjelmistokehityksestä sen aikana.

Toiminnallisen työn toteutus onnistui mielestäni erittäin hyvin. Tuloksesta tuli sellainen, kuin projektia suunniteltaessa haettiinkin ja se on erittäin hyvä lähtökohta SmartPerso-Officen uuden frontend koodin jatkokehitykselle. Tässä opinnäytetyössä oli tarkoituksena toteuttaa vain aikaisemmin mainitut komponentit ja tavoite toteutui. Projektia on tarkoitus jatkaa siirtämällä kaikki loputkin komponentit yksi kerrallaan uuteen Angular versioon. Projektissa saatiin uudelle frontendille toteutettua selkeämpi tiedostorakenne tyyleille, joka helpottaa jatkokehitystä huomattavasti. Ulkonäöllisesti uudesta toteutuksesta tuli hyvin samankaltainen, kuin aikaisempikin, mutta selvästi päivitetty version. Käytin toteutuksessa samoja värejä sekä tyylejä mitä aikaisemmassa AngularJS versiossa oli, jotta tuotos pysyisi mahdollisimman samankaltaisena, sillä tarkoituksena ei ollut uudistaa SmartPerso-Officen ulkoasua.

Projekti oli kokonaisuudessaan hyvin mielenkiintoinen, sillä siinä oli paljon uutta opittavaa, mutta asiat eivät olleet kuitenkaan ylitsepääsemättömän vaikeita. Siinä oli hyvinkin erilaisia kokonaisuuksia, niin ei tarvinnut koko opinnäytetyön ajan keskittyä vain yhteen asiaan. Halusin kuitenkin edetä aina yksi kohta kerrallaan, jotta minulla on aikaa syventyä sen hetkiseen aihepiiriin paremmin, eikä aikaa kulunut monen eri asian pyörittelemiseen samanaikaisesti.

Ennen projektia en tiennyt mitään AngularJS koodin päivityksestä Angulariin. Projektin aikana kuitenkin yllätyin, kuinka vanhasta koodista ei pystynyt käyttämään oikeastaan mitään uuteen koodiin. Oli helpompaa hyödyntää muita jo valmiita Angular pohjaisia ohjelmistoja, kuin hyödyntää vanhaa koodia. Jos joskus uudestaan alkaisin päivittämään vanhempaa AngularJS koodia uudempaan, aloittaisin varmaan suoraan toteutuksen täysin uuteen Angular projektiin. Tämän projektin alussa läpikäydyt AngularJS:n tarjoamat komponentit koodin päivitykselle eivät vaikuttaneet kovin lupaavilta. Ne ovat myös vain väliaikainen ratkaisu ja lopullinen päivitys Angulariin pitäisi kuitenkin tehdä, jossain kohtaa, jos ohjelmistolle on käyttöä vielä pitkällä tulevaisuudessakin.

Ennen projektia luulin, että autentikointi tulisi mahdollisesti olemaan projektin haastavin osa, mutta toisin kävi. Selvästi eniten haasteita oli käyttäjien hallinnoimisen luomisessa. Ongelmat eivät kuitenkaan olleet suuria, mutta siihen vaiheeseen projektia sattui monta erilaista pulmaa, joihin kului aikaa.

Itse opin projektin aikana Angularin kehitystä paljon syvällisemmin. Olen aikaisemminkin työskennellyt Angularin parissa, mutta tämän opinnäytetyön tekeminen selvensi kokonaiskuvaa Angularista, sillä en ole koskaan aikaisemmin mm. luonut koko Angular projektia alusta lähtien. Myös CSS sekä Bootstrap tuli tutummiksi projektin edetessä, kun aloin keskittymään enemmän ulkoasun muutoksiin. Näidenkin parissa olen ennenkin työskennellyt, mutta nyt minulla oli hyvä mahdollisuus syventyä enemmän mm. flexboxin käyttöön.

Jatkokehitys uudelle frontendille ei ole vielä alkanut, mutta voisin uskoa sen olevan minun työtäni myös tulevaisuudessa. Seuraavaksi olisi tarkoituksena saada yhdistettyä vanha ja uusi versio, jotta ne toimisivat yhdessä. Vanhasta AngularJS versiosta korjattaisiin reititys uusiin valmiisiin Angular komponentteihin, mutta puuttuvat komponentit toimisivat edelleen AngularJS:n puolella. Tämän opinnäytetyön tekeminen on kasvattanut ymmärrystäni SmartPerso-Officesta ja näkisin itseni tulevaisuudessa jatkamassa tätä projektia, nyt kun hyvä pohja on jo luotu.

Projektin aikana nousi myös ajatus dashboardin muokattavuudesta, jotta jokaisella käyttäjällä olisi mahdollisuus valita haluamansa widgetit työpöydälle. Olisin mahdollisesti halunnut toteuttaa tämän ominaisuuden jo tämän opinnäytetyöprojektin aikana, mutta tulin siihen lopputulokseen, että se olisi liian työläs lisä kaikkien muiden tavoitteiden lisäksi. Tämänkaltainen toiminnallisuus vaatisi todella paljon työtä ja taustatutkimusta, sillä ymmärtääkseni se ei ole kovin yksinkertainen lisä toteuttaa. Tämä on kuitenkin hyvä idea jatkokehityksen kannalta, johon voisi tulevaisuudessa palata, kun uusi frontend versio on saatu vietyä pidemmälle, sillä se toisi dashboardille aivan uudenlaista arvoa.

Lähteet

Angular s.a a. Upgrading from AngularJS to Angular. Luettavissa: https://angular.io/guide/upgrade#lazy-loading-angularjs. Luettu: 19.4.2023.

Angular s.a b. What is Angular?. Luettavissa. https://angular.io/guide/what-is-angular. Luettu: 20.2.2023.

Angular s.a c. HttpInterceptor. Luettavissa: https://angular.io/api/common/http/HttpInterceptor#description. Luettu: 21.4.2023.

AngularJS 2021. AngularJS. Luettavissa: https://angularjs.org. Luettu: 17.2.2023.

AngularJS 2022. Version support status. Luettavissa: https://docs.angularjs.org/misc/version-support-status. Luettu: 17.2.2023.

GeeksForGeeks 2022a. AngularJS. Luettavissa: https://www.geeksforgeeks.org/angularjs/. Luettu: 20.2.2023.

GeeksForGeeks 2022b. Difference between Angular and AngularJS. Luettavissa: https://www.geeksforgeeks.org/difference-between-angular-and-angularjs/. Luettu: 8.3.2023.

GeeksForGeeks 2022c. What is the difference between CSS and SCSS?. Luettavissa: https://www.geeksforgeeks.org/what-is-the-difference-between-css-and-scss/. Luettu: 20.2.2023.

GeeksForGeeks 2022d. Bootstrap. Luettavissa: https://www.geeksforgeeks.org/bootstrap/. Luettu: 21.2.2023.

JWT s.a. Introduction to JSON Web Tokens. Luettavissa: https://jwt.io/introduction. Luettu: 14.3.2023.

MDN Web Docs s.a. Basic concepts of flexbox. Luettavissa. https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Flexible_Box_Layout/Basic_Concepts_of_Flexbox. Luettu: 13.3.2023.

MDN Web Docs 2022. CSS: Cascading Style Sheets. Luettavissa: https://developer.mozilla.org/en-US/docs/Web/CSS. Luettu: 20.2.2023.

Microsoft 17.8.2022. Active Directory Domain Services Overview. Luettavissa: https://learn.microsoft.com/en-us/windows-server/identity/ad-ds/get-started/virtual-dc/active-directory-domain-services-overview. Luettu: 11.4.2023.

Onelogin s.a. What is LDAP?. Luettavissa: https://www.onelogin.com/learn/what-is-ldap. Luettu: 13.3.2023.

PrimeNg s.a. PrimeNg. Luettavissa: https://www.primefaces.org/primeng-v8-lts/#/. Luettu: 21.2.2023.

Symfony s.a. Routing. Luettavissa: https://symfony.com/doc/current/routing.html. Luettu: 28.3.2023.

Symfony s.a. Symfony explained to a Developer. Luettavissa: https://symfony.com/explained-to-a-developer. Luettu: 28.3.2023.

Symfony s.a. The Cache Component. Luettavissa: https://symfony.com/doc/current/components/cache.html. Luettu: 30.3.2023.

W3Schools s.a. HTML Web Storage API. Luettavissa: https://www.w3schools.com/html/html5_webstorage.asp. Luettu: 17.4.2023.