Asianhallintajärjestelmän prototyypin kehittäminen

Ville Jaatinen

2021 Laurea

Laurea-ammattikorkeakoulu

Asianhallintajärjestelmän prototyypin kehittäminen

Ville Jaatinen

Tietojenkäsittely

Opinnäytetyö

Lokakuu, 2021

Laurea-ammattikorkeakoulu **Tiivistelmä**

Tietojenkäsittely

Tradenomi (AMK)

Ville Jaatinen

Asianhallintajärjestelmän prototyypin kehittäminen

Vuosi 2021 Sivumäärä 37

Tämä toiminnallinen opinnäytetyö kuvaa asianhallintajärjestelmän prototyypin kehittämisprosessin Oikeat Oliot Oy:lle suunnittelusta käytännön toteutukseen. Opinnäytetyön tavoitteena oli kehittää toimeksiantajalle lomakkeiden täyttö- ja käsittelyprosessit sähköistävän sovelluksen prototyyppi järjestelmän jatkokehityksen pohjaksi. Osana opinnäytetyötä toteutettiin järjestelmän vaatimusmäärittely, käyttötapauskuvaukset ja prosessikaaviot keskeisimmistä toiminnoista sekä asianhallintajärjestelmän prototyyppi.

Opinnäytetyön tietoperusta pohjautuu prototyypin kehittämiseen hyödynnettyihin kehittämismenetelmiin sekä teknologioihin. Tietoperusta koottiin hyödyntämällä ohjelmistokehitykseen ja käytettyihin teknologioihin syventyvää kirjallisuutta, teknologioiden tarjoamaa sähköistä dokumentaatiota sekä muita tietoperustaa tukevia sähköisiä lähteitä. Lisäksi kehittämistyössä hyödynnettiin yrityksessä aiemmin syntynyttä osaamista asianhallinnan ohjelmistojen suunnittelusta.

Opinnäytetyön tuloksena kehitettiin web-käyttöliittymän sekä palvelinsovelluksen tarjoava asianhallintajärjestelmän prototyyppi. Kehitetty prototyyppi täytti projektin alussa sille asetetut tavoitteet tarjoten toiminnot dynaamisten lomakepohjien määrittämiseen ja hallintaan, lomakkeiden täyttöön, tallennukseen, muokkaukseen ja käsittelyyn sekä käsittelyprosessin aloittamiseen, läpivientiin ja päättämiseen. Lisäksi prototyyppi mahdollistaa lomakkeiden sähköisen allekirjoittamisen sekä täytetyn lomakkeen pohjalta generoidun PDF-tiedoston tulostamisen. Toimeksiantaja hyödyntää kehitettyä prototyyppiä järjestelmän jatkokehityksen pohjana.

Asiasanat: ohjelmistokehitys, prototyyppi, asianhallintajärjestelmä, sähköinen lomake

Laurea University of Applied Sciences **Abstract**

Degree Programme in Business Information Technology

Bachelor's degree

Ville Jaatinen

Development of a case management system prototype

Year 2021 Pages 37

This functional Bachelor's thesis describes the development process of a prototype of a case management system from design to practical implementation for Oikeat Oliot Ltd. The aim of the thesis was to develop a prototype of an application that digitizes the forms filling and processing processes for the client as a basis for further development of the system. As part of the thesis, the system requirements specification, use case descriptions and process diagrams of the most important functionalities were made, and a prototype of the case management system was developed.

The theoretical background of the thesis is based on the development methods and technologies utilized in the development of the prototype. The theoretical background was gathered by utilizing in-depth literature on software development and the technologies used, digital documentation provided by the technologies and other digital sources supporting the theoretical background. In addition, the development work utilized the company's previously acquired know-how in the design of case management software.

As a result of the thesis, a prototype of a case management system offering a web user interface and a server application was developed. The developed prototype met the objectives set for it at the beginning of the project, offering functionalities for defining and managing dynamic form templates, filling in forms, saving, editing and processing forms, and starting, executing and ending the processing process of the form. In addition, the prototype enables the digital signing of forms and the printing of a PDF file generated on the basis of a filled form. The client utilizes the developed prototype as a basis for further development of the system.

Keywords: software development, prototype, case management system, digital form

Sisällys

1 Johdanto 6
2 Kehitystyön lähtökohdat 6
2.1 Kohdeorganisaatio 7
2.2 Työn tarkoitus ja tavoitteet 7
2.3 Aiheen rajaus 7
3 Tietoperusta 8
3.1 Teknologiat 8
3.1.1 Vue.js 8
3.1.2 Quasar 9
3.1.3 TypeScript 9
3.1.4 Kotlin 9
3.1.5 PostgreSQL 10
3.1.6 Docker 10
3.1.7 Gradle 11
3.1.8 Kanto 11
3.2 Kehittämismenetelmät 11
3.2.1 Ketterä kehittäminen 11
3.2.2 Prototypointi 12
4 Prototyypin toteutus 13
4.1 Määrittely ja suunnittelu 13
4.1.1 Vaatimusmäärittely 14
4.1.2 Käyttötapauskuvaukset 15
4.1.3 Prosessikaaviot 16
4.2 Arkkitehtuuri 17
4.3 Toteutus 19
4.3.1 Sovelluskonfiguraation ja tietokantarakenteen määrittäminen 19
4.3.2 Käyttöliittymän pohja - sisäänkirjautuminen ja kotinäkymä 20
4.3.3 Lomakepohjaeditori 23
4.3.4 Lomakkeen täyttö- ja muokkausnäkymät 25
4.3.5 PDF-tiedoston generointi ja sähköinen allekirjoitus 26
4.3.6 Lomakkeen käsittelyprosessi 31
5 Yhteenveto ja johtopäätökset 32
5.1 Jatkokehitys 33
Kuviot 37

1 Johdanto

Tämä toiminnallisena opinnäytetyönä tehty asianhallintajärjestelmän prototyypin kehittämistyö kuvaa lomakkeiden täytön, hallinnan ja asianhallintaprosessien läpiviennin sähköistävän web-sovelluksen ohjelmointiprojektin ensimmäisen iteraation keskeiset vaiheet suunnittelusta toteutukseen. Opinnäytetyön tietoperusta pohjautuu työssä käytettyjen teknologioiden ja kehitysmenetelmien teoriaan.

Opinnäytetyön toimeksiantajana toimii Oikeat Oliot Oy, joka haluaa saada aikaan ohjelmistorungon tai -työkalun, jolla voidaan yksinkertaistaa lomakkeiden täyttö- ja käsittelyprosessien tekeminen. Työn tavoitteena on luoda jatkokehityksen pohjana toimiva asianhallintajärjestelmän tärkeimmät perustoiminnallisuudet kattava ensimmäinen kehitysversio, jonka avulla sovellusta voidaan myös tarvittaessa esitellä asiakkaille.

Työssä käydään läpi kehitysprojektin toteutuksen kannalta keskeiset teknologiat ja kehitysmenetelmät, sekä prototyypin kehittämisprosessin toteutuksen vaiheet läpi ensimmäisen iteraation. Työn lopputuloksena syntyi ensimmäinen versio asianhallintajärjestelmän ohjelmistorungosta, jota jatkokehitetään sen hyödyntämiseksi tulevaisuudessa prototypoinnin lisäksi järjestelmien pohjana.

2 Kehitystyön lähtökohdat

Opinnäytetyön aiheen taustalla on aito työelämän tarve sähköistää lomakkeiden käsittely- ja asianhallintaprosessit. Opinnäytetyön toimeksiantaja Oikeat Oliot Oy on tietotekniikkakonsultointia harjoittava yritys, joka muiden tietotekniikkapalvelujen ohella kehittää asiakaskohtaisia tietojärjestelmiä useille eri julkisen- ja yksityisen sektorin toimijoille. Monet toimeksiantajan asiakkaista käsittelevät päivittäisessä toiminnassaan lomakkeita, joiden täyttö- ja käsittelyprosessit ovat usein hitaita ja saattavat vaatia fyysisten lomakkeiden tulostamista ja täyttämistä. Oikeat Oliot haluaa ratkaista ongelman kehittämällä asianhallintajärjestelmän sähköistämään lomakkeiden täyttö-, lähetys- ja päätösprosessit generoimalla lomakkeita ennalta määritellyn säännöstön mukaan ja hallitsemalla käsittelyprosessia sen eri vaiheiden läpi. Asiakkaat ovat ilmaisseet jo aikaisemmin mielenkiintoa vastaavia ominaisuuksia kohtaan, joten ratkaisulle on myös jo valmista kysyntää asiakaskunnan keskuudessa. Opinnäytetyön rooli tämän järjestelmän kehittämisessä on saada kehitysprosessi aluilleen kehittämällä järjestelmän prototyyppi.

2.1 Kohdeorganisaatio

Opinnäytetyön kohdeorganisaationa toimii Oikeat Oliot Oy, joka on tietotekniikkakonsultointia harjoittava ohjelmistotalo Helsingistä. Oikeat Oliot Oy suunnittelee ja toteuttaa tietojärjestelmiä, asiakirjan- ja asianhallinnan sovelluksia sekä tarjoaa monipuolisia palveluita ohjelmistotestauksen, palvelumuotoilun ja pilvipalveluiden saralla. Oikeilla Olioilla on yli 25 vuoden kokemus vaativien asiakaskohtaisten tietojärjestelmien kehittämisestä niin suurille julkisen sektorin toimijoille kuin pienemmille yksityisille yrityksillekin. Yritysmuodoltaan osakeyhtiönä toimivan Oikeat Oliot Oy:n liikevaihto vuonna 2020 oli 2,2 miljoonaa euroa (Asiakastieto 2021.) ja yritys työllistää 20 työntekijää.

2.2 Työn tarkoitus ja tavoitteet

Opinnäytetyö on muodoltaan kehittämistyö, jonka tavoitteena on kehittää Oikeat Oliot Oy:lle asianhallintajärjestelmän prototyyppi osana kehittämisprosessin ensimmäistä iteraatiota. Prototyypin tavoitteena on luoda asianhallintajärjestelmän ensimmäinen kehitysversio, joka hallitsee ainakin yhden asianhallintatapauksen läpiviennin lomakkeen täytöstä asianhallintaprosessin päättämiseen asti. Prototyypin on tarkoitus myös tarjota yksinkertainen web-käyttöliittymä lomakkeiden täyttämistä, tallentamista ja tulostamista, lomakepohjien luomista ja hallintaa sekä käsittelyprosessin hallintaa varten. Työn tuloksena syntyneen järjestelmän avulla järjestelmää, sen toiminnallisuutta ja ominaisuuksia on tarkoitus pystyä myös esittelemään lopulta mahdollisille asiakkaille.

2.3 Aiheen rajaus

Opinnäytetyönä toteutettavan asianhallintajärjestelmän prototyypin kehitys rajataan ensimmäisen iteraation toteuttamiseen, jonka osana toteutetaan prototyypin vaatimusmäärittely, käyttötapauskuvaukset, prosessikaaviot prototyypin ydinprosesseista sekä ensimmäinen kehitysversio asianhallintajärjestelmästä. Prototyypin tukemat asianhallintatapaukset rajataan ensimmäisen iteraation aikana erään toimeksiantajan asiakkaan yhteen asianhallintatapauksen käsittelyn läpivientiin. Asianhallintaprosessin vaiheisiin sisällytetään lomakkeen/hakemuksen täytön aloitus, allekirjoitus, käsittely, käsittelyn päättäminen joko hyväksyntään, hylkäykseen tai käsittelemättä jättämiseen, sekä lomakkeen säilytys. Asianhallintajärjestelmän lopulliseen käyttöliittymään ei oteta sen tarkemmin kantaa ensimmäisen iteraation aikana, vaan käyttöliittymä toteutetaan vielä tässä kehitysvaiheessa toiminnallisuus edellä MVP, eli Minimum Viable Product periaatteella.

3 Tietoperusta

Opinnäytetyön tietoperusta pohjautuu prototyypin kehittämisessä hyödynnettyihin teknologioihin ja kehitysmenetelmiin. Prototyyppi kehitettiin hyödyntäen avoimen lähdekoodin teknologioita kuten Vue.js, Typescript, SCSS, Quasar, Kotlin, PostgreSQL ja Docker sekä Oikeiden Olioiden kehittämää talon sisäistä teknologiaa. Kehittämistyön toteutuksessa seurattiin prototypoinnin sekä ketterän kehittämisen periaatteita yleisellä tasolla. Opinnäytetyön tietoperusta on kerätty käyttämällä teknologioihin ja kehitysmenetelmiin syventyvää kirjallisuutta sekä sähköisiä lähteitä. Teknologioiden kehittyessä kasvavalla vauhdilla ja niiden rakenteen muuttuessa jatkuvasti, on etenkin teknologioiden osalta hyödynnetty laajasti myös niiden kehittäjien kokoamaa sähköistä dokumentaatiota.

3.1 Teknologiat

Prototyypin kehittämisessä hyödynnetyt teknologiat valittiin pitkälti Oikeiden Olioiden jo aikaisemmin hyödyntämien teknologioiden mukaan, jotka tarjoavat hyvän pohjan yhteensopivuudelle aiemmin kehitettyjen ohjelmistojen ja talon sisäisen teknologian kanssa. Prototyyppi kehitettiin omana sovelluksenaan, mutta kehitystyössä otettiin huomioon samanaikaisesti osittainen yhteensopivuus Oikeiden Olioiden muun tuoteperheen kanssa, jotta prototyyppi voidaan tarvittaessa myöhemmin tarjota osana isompaa ohjelmistoratkaisua tai lisäpalveluna.

3.1.1 Vue.js

Vue.js on moderni JavaScript-kehys web-sovellusten käyttöliittymien rakentamiseen. Progressiivinen sovelluskehys soveltuu hyvin asteittain käyttöönotettavaksi, mutta se skaalautuu hyvin myös laajempien web-sovellusten kehitykseen kokonaisena sovelluskehyksenä. Vue-sovellus koostuu komponenteista, voidaan hyödyntää vapaasti sovelluksen eri osissa ja kerran kirjoitettua komponenttia voi käyttää uudelleen useaan otteeseen. Komponenttirakenteen koostavat HTML-, JavaScript/TypeScript- ja CSS/SCSS-määritykset on mahdollista määritellä yhteen komponenttitiedostoon eli Vue-templateen, mutta ne tai osia niistä voi tarvittaessa myös pilkkoa omiin tiedostoihinsa komponentin ulkopuolelle. Vue on reaktiivinen, eli se osaa reagoida itsenäisesti datan muutoksiin ja päivittää käyttöliittymän koostavat komponentit ajan tasalle datan muutosten mukaan. (Vue.js 2021.) Vue.js on alun perin Evan Youn kehittämä avoimen lähdekoodin JavaScript-sovelluskehys, jonka ensimmäinen versio 0.6.0 ilmestyi GitHubiin vuonna 2013 (GitHub 2013.). Vue toimii asianhallintajärjestelmän prototyypin käyttöliittymän pohjana, ja mahdollistaa muun muassa reaktiivisten käyttöliittymäkomponenttien sekä ylläpidettävän ja modulaarisen sovellusrakenteen toteuttamisen modernien websovellusten asettamien standardien mukaan.

3.1.2 Quasar

Quasar on Vue.js-kehykseen pohjautuva käyttöliittymäkehys, joka tarjoaa valmiita käyttöliittymäratkaisuja ja komponentteja web-sovellusten rakentamiseen kaikille alustoille. Quasarin avulla kehitetty sovellus voidaan kääntää samasta koodipohjasta useisiin eri muotoihin ja useille eri alustoille, mukaan lukien SPA- (Single Page Application), SSR- (Server Side Rendered Application), mobiili- ja työpöytäsovellukseksi. Quasar-sovellusten alustamiseen ja hallintaan kehitetty QuasarCLI komentorivityökalu kokoaa Quasar-sovelluksen juurikansion, johon sisällytetään käyttäjän komentojen mukaan tuet niille alustoille, joille sovellus halutaan kehittää. (Quasar Framework 2021.) Quasarin avulla nopeutetaan asianhallintajärjestelmän prototyypin Vue-käyttöliittymän rakentamista hyödyntämällä sen laajaa käyttöliittymäkomponenttien valikoimaa sekä valmiita SCSS-tyylejä.

3.1.3 TypeScript

TypeScript on staattisesti tyypitetty web-sovelluskehityksessä käytettävä JavaScriptiin pohjautuva ohjelmointikieli. TypeScript on kielenä JavaScriptin ylijoukko (engl. superset) (Fenton 2017, xix), eli se on syntaktisesti yhtenäistä JavaScriptin kanssa ja JavaScript-koodi on itsessään rakenteeltaan laillista TypeScriptiä (Fenton 2017, 2). Vaikka TypeScriptistä puhutaan omana kielenään, kaikki TypeScriptillä kirjoitettu koodi käännetään ennen sovelluksen ajoa kuitenkin JavaScriptiksi. (TypeScript 2021.) TypeScript tarjoaa JavaScript-kehitykseen lisäominaisuutena staattisen virheiden tarkistuksen ennen koodin suorittamista ja vähentää ajoympäristössä tyypityksen puutteesta nousevia virheitä, parantaen laajojen JavaScript sovellusten hallittavuutta ja helpottaen koodipohjan ylläpitoa (Fenton 2017, xxv). TypeScript tuo asianhallintajärjestelmän prototyypin client-kerrokseen vahvan tyypityksen, jonka avulla vältytään suurimmalta osalta tyyppikonfliktien aiheuttamista virheistä ja pystytään kirjoittamaan kaikilta osin luotettavampaa koodia.

3.1.4 Kotlin

Kotlin on alun perin Tšekkiläisen JetBrainsin kehittämä staattisesti tyypitetty ohjelmointikieli, joka ensimmäinen versio ilmestyi vuonna 2012 (JetBrains 2021.). Kotlin on korkean tason ohjelmointikieli, joka tarjoaa ytimekkään tavan kirjoittaa Java-yhteensopivaa koodia vähentäen karkeasti noin 40 % tarvittavia koodirivejä suhteessa Java-koodiin. Kotlin kääntyy JVM-tavukoodiksi, joten se voidaan helposti ottaa osaksi jo olemassa olevaa Java-koodipohjaa. Kotlinilla tuotettu koodi on hyvin uudelleenkäytettävää eri alustoiden välillä sen tarjotessa työkalut Kotlin-sovellusten kehittämiseen aina mobiililaitteista, alustariippumattomiin multiplatform-sovelluksiin sekä palvelin- ja web-sovelluksiin. Tämän lisäksi se voidaan kääntää muun muassa myös JavaScriptiksi. Kotlin sisältää olio- ja funktionaalisen ohjelmoinnin rakenteita, ja mahdollistaa näin ollen molempien ohjelmointityylien yhdistelyn sovellusrakenteessa. (JetBrains 2021.) Yksi Kotlinin isoimpia vahvuuksia on sen täysi yhteensopivuus Javan ja Androidin

kanssa (Karanpuria 2018.). Android-käyttöjärjestelmän kehittäjänä tunnettu Google nimesi vuonna 2019 Kotlinin virallisesti tuetuksi kieleksi Android-kehittämiseen, tuhannesta suosituimmasta Android-sovelluksesta noin 60-prosentin sisältäessä Kotlin-koodia (Winer 2019.), mikä on vakiinnuttanut Kotlinin asemaa etenkin Android-sovelluskehityksen suurena tulevaisuuden trendinä. Monikäyttöisyytensä ja idiomaattisen kielirakenteensa ansiosta Kotlin on vahvassa nousussa oleva ohjelmointikieli, joka on varteenotettava vaihtoehto ohjelmistokehittämiseen alustasta riippumatta. Kotlin tarjoaa vankan ja skaalautuvan pohjan asianhallintajärjestelmän prototyypin palvelinkerroksen operaatioiden suorittamiseen sekä toimii luotettavana rajapintana web-käyttöliittymän ja PostgreSQL-tietokannan välillä.

3.1.5 PostgreSQL

PostgreSQL on relaatiotietokantojen hallintaan kehitetty avoimen lähdekoodin oliotietokannanhallintajärjestelmä (engl. object-relational database management system, tai ORDMS). PostgreSQL skaalautuu hyvin tukemaan myös isojen tietojärjestelmien tarpeita ja tarjoaa yritystasoisen suorituskyvyn laajojen relaatiotietokantojen hallintaan. Sen lisäksi PostgreSQL on alustariippumaton sen toimiessa tunnetuimmilla moderneilla käyttöjärjestelmillä kuten Windows, Linux ja MacOS (Salahaldin, Vannahme & Volkov 2015, 31). PostgreSQL mahdollistaa kommunikoinnin tietokanta-clientin ja palvelimen välillä client/server-mallin mukaisesti, tavallisimmin TCP/IP-protokollan tai Linux-pistokkeiden (engl. socket) välityksellä (Salahaldin, Vannahme & Volkov 2015, 36). PostgreSQL on tunnettu datan eheydestä ja sen datan validointiin tarjoamista tehokkaista työkaluista. PostgreSQL tarjoaa vakaan ja luotettavan tietokannan asianhallintajärjestelmän hyödyntämän datan säilöntään ja käsittelyyn.

3.1.6 Docker

Vuonna 2013 avoimen lähdekoodin ohjelmistona julkaistu Docker on ohjelmistojen lähetykseen ja käyttöönottoon kehitetty työkalu, jonka avulla ohjelmistoja voidaan ajaa ajoympäristöstä riippumattomissa eristetyissä konteissa. dotCloudin yrityksen sisäisenä työkaluna alkunsa saaneella Dockerilla voidaan luoda virtuaalinen käyttöjärjestelmä eli Docker-kontti, joka on itsenäinen ja erillään käyttöjärjestelmästä, jolla sitä ajetaan. Docker-kontti paketoi ohjelmiston kaikkine riippuvuuksineen yhteen standardimuotoiseen yksikköön, joka sisältää kaiken ohjelmiston ajamiseen tarvittavan lähdekoodista ajoympäristöön ja tiedostojärjestelmään. Kontitettu ohjelmisto on ajoympäristöstä riippumaton ja se voidaan ajaa samalla tavalla kaikilla järjestelmillä, jotka tukevat Dockeria ilman tarvetta asentaa sovelluksen vaatimia riippuvuuksia erikseen jokaiselle konttia ajavalle alustalle. Ajoympäristöjen perustaminen sekä poistaminen on Dockerin avulla helppoa ja nopeaa kontin sisältäessä kaikki sovelluksen ajamiseen tarvittavat riippuvuudet. Riippuvuudet myös siivotaan pois kontin mukana automaattisesti poistettaessa kontti, joka mahdollistaa Dockerilla paketoitujen ohjelmistojen vaivattoman ylläpidon myös järjestelmän näkökulmasta. (Krochmalski 2016, 6-10.)

3.1.7 Gradle

Gradle on suosittu avoimen lähdekoodin koontiautomaatiotyökalu sovellusten kokoamiseen ja käyttöönottoon sekä niiden riippuvuuksien hallintaan (Varanasi 2015, 1). Koontiautomaatiotyökalut ovat kehityksen tukena hyödynnettäviä ohjelmistoja, jotka automatisoivat sovelluksen kehittämiseen ja käyttöönottoon liittyviä toistuvia manuaalisia tehtäviä, kuten sovelluksen kokoaminen, sen riippuvuuksien hallinta sekä testien ajaminen, säästäen tehtävien toteuttamiseen kuluvaa aikaa ja kuluja (Mitra 2015, 2). Gradle on deklaratiivinen koontityökalu, eli se osaa koota ja hakea projektille määritetyt riippuvuudet niiden määrittelyjen perusteella ilman että kehittäjän on määritettävä kuinka riippuvuudet tulisi noutaa (Varanasi 2015, 1). Gradle määrittää koontia varten joukon tehtäviä, jonka jälkeen se määrittää tehtävien suoritusjärjestyksen niiden riippuvuuksien mukaan. Gradle koontiin hyödynnettävät koontiskriptit voidaan määrittää joko Groovylla tai Kotlin DSL:lä (Gradle 2021). Prototyypin kehittämistyössä hyödynnettiin Gradlea Java- ja Kotlin-riippuvuuksien hallintaan sekä projektin kokoamiseen kehitys- ja tuotantoympäristöissä.

3.1.8 Kanto

Kanto on Oikeat Oliot Oy:n kehittämä Vueen, Kotliniin ja Javaan perustuva sovellukehys selainkäyttöisten rekisteri- ja tietokantasovellusten kehittämiseen. Kehyksen tarkoituksena on mahdollistaa Java-palvelinympäristössä toimivien sovellusten kustannustehokas toteuttaminen vaihtuviin asiakasympäristöihin hyödyntämällä monille sovelluksille yhteisiä piirteitä uudelleen. Kanto-sovellukset ovat arkkitehtuuriltaan matalia ja sen vuoksi myös yleisesti suorituskykyisiä. Kanto tarjoaa monia valmiita ominaisuuksia rekisteri- ja tietokantasovellusten kehittämiseen, kuten CRUD-toiminnot, tietotyyppikohtaisen validoinnin, monikielisyyden tuen, deklaratiivisen pääsynvalvonnan ja käyttäjänhallinnan sekä lokipalveluita. (Oikeat Oliot Oy 2021.)

3.2 Kehittämismenetelmät

Asianhallintajärjestelmän prototyypin kehittämisessä hyödynnettiin ketterän kehittämisen sekä protoilumallin periaatteita. Kehittäminen pyrittiin pitämään tavoitteellisena mutta yksinkertaisena niin, että määritetyistä vaatimuksista huolimatta muutoksiin mukautuminen on joustavaa ja prototyypillä on tilaa muovautua projektin edetessä.

3.2.1 Ketterä kehittäminen

Ketterä kehittäminen on jäykkien sovelluskehitysmenetelmien, kuten vesiputousmallin vastineeksi syntynyt kehitysfilosofia, jonka tavoitteena on yksinkertaistaa kehitysprosessia sekä auttaa keskittymään kehityksen kannalta olennaisiin asioihin karsimalla ylimääräiset rakenteet pois ja lisäämällä joustavuutta kehitysprosessiin. Teknologian ja liiketoiminnan

kehittyessä ja ollessa jatkuvan muutoksen alla, myös ohjelmistokehitysprosessien on pystyttävä muuttumaan ja mukautumaan vaatimuksiin sekä projektin laajuuteen kohdistuviin nopeisiin muutoksiin ketterästi (Holocombe 2008, 1-2). Vuonna 2001 julkaistu ja ketterän ohjelmistokehityksen pohjaksi muodostunut Agile Manifesto, eli ketterän ohjelmistokehityksen julistus määrittää ketterän filosofian pääarvot sekä perusperiaatteita ketterän kehittämisen toteuttamiselle. Julistus listaa ketterän kehittämisen pääarvoissa ketterän filosofian arvostavan ”yksilöitä ja kanssakäymistä enemmän kuin menetelmiä ja työkaluja, toimivaa ohjelmistoa enemmän kuin kattavaa dokumentaatiota, asiakasyhteistyötä enemmän kuin sopimusneuvotteluja” sekä ”vastaamista muutokseen enemmän kuin pitäytymistä suunnitelmassa”. Julistuksen mukaan molemmat osat ovat tärkeitä, mutta edellä mainittuja arvostetaan enemmän ketterän filosofian näkökulmasta. Agile Manifeston noudattamat periaatteet korostavat yksinkertaisia, ihmislähtöisiä ja johdonmukaisia toimintamalleja, joiden keskiössä on toimivan ohjelmiston tuottaminen mukauttamalla toimintaa toteutuksen tehostamiseksi. (Agile Manifesto 2001)

Ketterän kehittämisen filosofia kulki mukana prototyypin toteutuksessa suunnittelusta toteutukseen. Prototyypin suunnitteluvaiheen oli tarkoitus luoda pohja järjestelmän kehittämiselle, mutta vaatimusten ja muun alustavan dokumentaation toteutuksessa vältettiin tarkoituksella liian tarkkaa määrittelyä ja yksityiskohtiin syventymistä, jotta toteutukselle jäi tilaa muovautua vapaasti projektin edetessä sekä tilanteiden muuttuessa parhaan lopputuloksen saavuttamiseksi. Laatua tarkkailtiin muun muassa vanhempien kollegojen suorittamilla koodikatselmoinneilla, joiden tarkoituksena oli saada useampi näkökulma ja laadullista palautetta tuotettuun tekniseen toteutukseen. Tuotetuille muutoksille suoritettiin vakiotestit ja muutokset päivitettiin automaattisesti CD/CI-putkien kautta testiympäristöön puskettaessa uusia osia projektin versionhallintaan. Vaikkei toteutuksessa seurattu tiukkaa aikasykliä, toteutus eteni sprinttiluontoisesti tilanteen läpikäyntiin keskittyvien palaverien välillä, joissa käytiin läpi mitä on toteutettu, mitä seuraavaksi tullaan toteuttamaan, onko kohdattu ongelmia sekä vaihdettiin ideoita ja ajatuksia prototyypin edistymisestä. Toimeksiantajalta pyydettiin palautetta suunnitteluvaiheen ja toteutuksen tuotoksiin, ja toteutusta muokattiin saadun palautteen perusteella. Kokonaisuudessaan prototyypin kehittäminen pyrittiin toteuttamaan mahdollisimman kevyesti keskittyen tärkeimpään eli tuotettavaan prototyyppiin.

3.2.2 Prototypointi

Prototyypillä tarkoitetaan alkuperäistä, usein toimivaa mallia uudesta tuotteesta tai tuotteen versiosta, joka toimii perustana tai standardina kehitysprosessin myöhemmille vaiheille ja lopulliselle tuotteelle. Prototyypin tarkoituksena on pyrkiä todentamaan suunnitteluvaiheessa kehitettyjä ideoita ja konsepteja sekä mallintaa tuotteen ominaisuuksia ja toiminnallisuutta. Prototyyppi visualisoi ohjelmistovaatimusten kuvaukset niiden selittämisen ja kuvailun sijaan, ja jättää tilaa uusien ideoiden kokeiluun ja optimaalisten ratkaisujen etsintään luodun mallin puitteissa. (Arnowitz, Arent & Berger 2007, 3-4.) Prototypointi (engl. prototyping) on

kehitysprosessi, jonka tuloksena tuotetaan yksi tai useampi prototyyppi kehitettävästä tuotteesta. Prototypointi on hyödyllinen työkalu kehitettyjen ideoiden jalostamiseen sekä sidosryhmien kanssa kommunikointiin. Prototypoinnin avulla voidaan vastata uuden tuotteen kehittämisestä syntyviin kysymyksiin, kuten toimiiko suunniteltu malli oletetusti, kuinka käyttäjät suhtautuvat malliin, voidaanko malli toteuttaa taloudelliselta kannalta tehokkaasti sekä mikä lähestymistapa soveltuu parhaiten mallin realisoimiseksi valmiiksi tuotteeksi. (Arnowitz, Arent & Berger 2007, 9-10.) Asianhallintajärjestelmän prototyyppi toteutettiin evoluutioprototyyppinä, eli sen on tarkoitus lopulta kehittyä kehitysiteraatioiden seurauksena varsinaiseksi tuotteeksi. Evoluutioprototyypin ideana on kehittää alkuperäistä prototyyppiä testaamisen sekä sidosryhmiltä ja käyttäjiltä vastaanotetun palautteen tuottaman tiedon pohjalta. Evoluutioprototyypin rakentamista jatketaan olemassa olevan prototyypin luomalle pohjalle kehittäen tuotetta ajan myötä vastaamaan käyttäjien sekä yritysten asettamia tavoitteita. Prototypoinnin muotona evoluutioprototyyppi antaa kehittäjille mahdollisuuden keskittyä rakentamaan käyttäjälle merkityksellisiä sovelluksen osia ilman, että kehittäjien täytyy seurata tiukasti projektin alussa ennalta määritettyä suunnitelmaa. (Knight 2018, 132-133)

4 Prototyypin toteutus

Asianhallintajärjestelmän prototyypin toteutus koostui kahdesta päävaiheesta; suunnitteluvaiheesta sekä käytännön toteutuksesta. Prototyypin toteutuksen osana tuotettiin kehitettävän sovelluksen alustava vaatimusmäärittely sekä käyttötapauskuvaukset ja prosessikaaviot järjestelmän ydinprosesseista. Käytännön toteutus eli järjestelmän prototyypin koodaus suoritettiin itsenäisesti vanhempien kollegoiden ohjauksessa hyödyntäen ketterien kehitysmenetelmien ideologiaa vuorovaikutuksessa toimeksiantajan kanssa. Toteutus aloitettiin asianhallintajärjestelmän alustavalla määrittelyllä ja suunnittelulla, jonka pohjalta lähdettiin työstämään käytännön toteutusta eli asianhallintajärjestelmän prototyyppiä.

4.1 Määrittely ja suunnittelu

Prototyypin määrittelyssä ja suunnittelussa lähdettiin liikkeelle vaatimusmäärittelyllä, jonka avulla pyrittiin saamaan alustava käsitys järjestelmän tarpeista ja kartoittamaan sovelluksen vaatimia toiminnallisuuksia. Vaatimusmäärittelyssä keskityttiin ensimmäisen iteraation aikana toteutettavien ominaisuuksien esiin nostamiin toiminnallisiin ja ei-toiminnallisiin tarpeisiin, mutta määriteltiin myös prototyypin laajuuden ylittäviä tulevien kehitysvaiheiden vaatimuksia järjestelmän jatkokehitystä varten. Vaatimusmäärittelyyn pyydettiin palautetta toimeksiantajalta ja siihen tehtiin muutoksia ja lisäyksiä saadun palautteen pohjalta. Havaittujen vaatimusten pohjalta koottiin käyttötapauskuvaukset sekä prosessikaaviot järjestelmän keskeisistä toiminnoista tukemaan järjestelmän tarpeiden ymmärrystä.

4.1.1 Vaatimusmäärittely

Prototyypin vaatimusmäärittelyssä vaatimukset jaettiin kahteen pääluokkaan, toiminnallisiin ja ei-toiminnallisiin vaatimuksiin. Toiminnallisilla vaatimuksilla tarkoitetaan yksityiskohtaisia kuvauksia ohjelmiston toivotuista toiminnoista. Toiminnalliset vaatimukset kuvaavat ohjelmiston toimintaa ja sitä, kuinka käyttäjä tai järjestelmä suorittaa vaaditut toiminnot eri ehtojen toteutuessa. Näillä vaatimuksilla saadaan selkeä kuva ohjelmiston eri toiminnallisuuksien työnkulusta ja toivotusta lopputulemasta. Ei-toiminnalliset vaatimukset sen sijaan kuvaavat ohjelmiston toiminnan ja käyttäytymisen laatuun liittyviä vaatimuksia, sekä asettavat rajoitteita sille, kuinka toiminnallisten vaatimusten lopputulema saavutetaan. Ei-toiminnallisten vaatimusten avulla pyritään havainnollistamaan ohjelmiston toiminnallisten vaatimusten toteutuksen kannalta tärkeitä, muun muassa laatuun, suorituskykyyn, luotettavuuteen sekä tietoturvaan liittyviä vaatimuksia, ehtoja ja rajoitteita. (Stephens 2015, 63)

Asianhallintajärjestelmän toiminnallisiin vaatimuksiin koottiin järjestelmän ydintoimintojen ja yleisen toiminnan kannalta keskeisiä toiminnallisia tarpeita, kuten lomakepohjien hallintaan, lomakkeiden täyttöön ja käsittelyyn, käyttäjäroolikohtaisiin toimintoihin sekä asianhallintaprosessin läpivientiin liittyviä vaatimuksia. Vaatimusmäärittelyssä linjattiin, että asianhallintajärjestelmä toteutetaan sisäänkirjautumista edellyttävänä roolipohjaisena järjestelmänä. Järjestelmään voidaan määritellä ja tallentaa lomakepohjia, joiden pohjalta voidaan täyttää lomakkeita ja suorittaa niiden käsittelyprosessit. Lomakepohjien rakenteet ovat vapaasti muokattavissa ja niitä voi määritellä asianhallintajärjestelmän käyttöliittymään toteutettavan yksinkertaisen web-editorin avulla. Täytetyt lomakkeet voidaan allekirjoittaa sähköisesti ja niistä voidaan tulostaa myös allekirjoitettu PDF-tiedosto.

Järjestelmässä on kolme eri käyttäjäroolia - täyttäjä, käsittelijä ja pääkäyttäjä - joiden perusteella eri näkymien ja toimintojen saatavuutta hallitaan. Pääkäyttäjä toimii järjestelmän ylläpitäjänä, jolla on laajimmat käyttöoikeudet järjestelmässä. Pääkäyttäjä pystyy luomaan ja poistamaan käyttäjiä järjestelmässä, sekä määrittämään käyttäjille käyttöoikeudet roolin mukaan. Pääkäyttäjä pystyy myös määrittämään uusia lomakepohjia sekä muokkaamaan ja poistamaan aiemmin tallennettuja lomakepohjia. Täyttäjät ovat asianhallintajärjestelmän loppukäyttäjiä, jotka voivat täyttää ja lähettää järjestelmään tallennettujen lomakepohjien mukaisia lomakkeita käsiteltäväksi. Täyttäjä voi halutessaan tallentaa lomakkeen keskeneräisenä ennen sen lähettämistä käsittelyyn, sekä poistaa tallennettuja lomakkeita, joita ei halua säilyttää. Käsittelijä huolehtii täyttäjien asianhallintaprosessien käsittelystä sekä käsittelyprosessin etenemisestä. Käsittelijä voi ottaa täyttäjien lähettämiä lomakkeita käsittelyyn ja muuttaa lomakkeiden tilaa käsittelyprosessin vaiheen mukaan, sekä päättää asianhallintaprosessin käsittelyn päätyttyä. Kaikkien käyttäjien on mahdollista täyttää asianhallintaprosessin kannalta olennaisia henkilötietoja käyttäjätililleen, joita voidaan tarvittaessa hyödyntää myöhemmin asianhallintatapausten läpiviennissä.

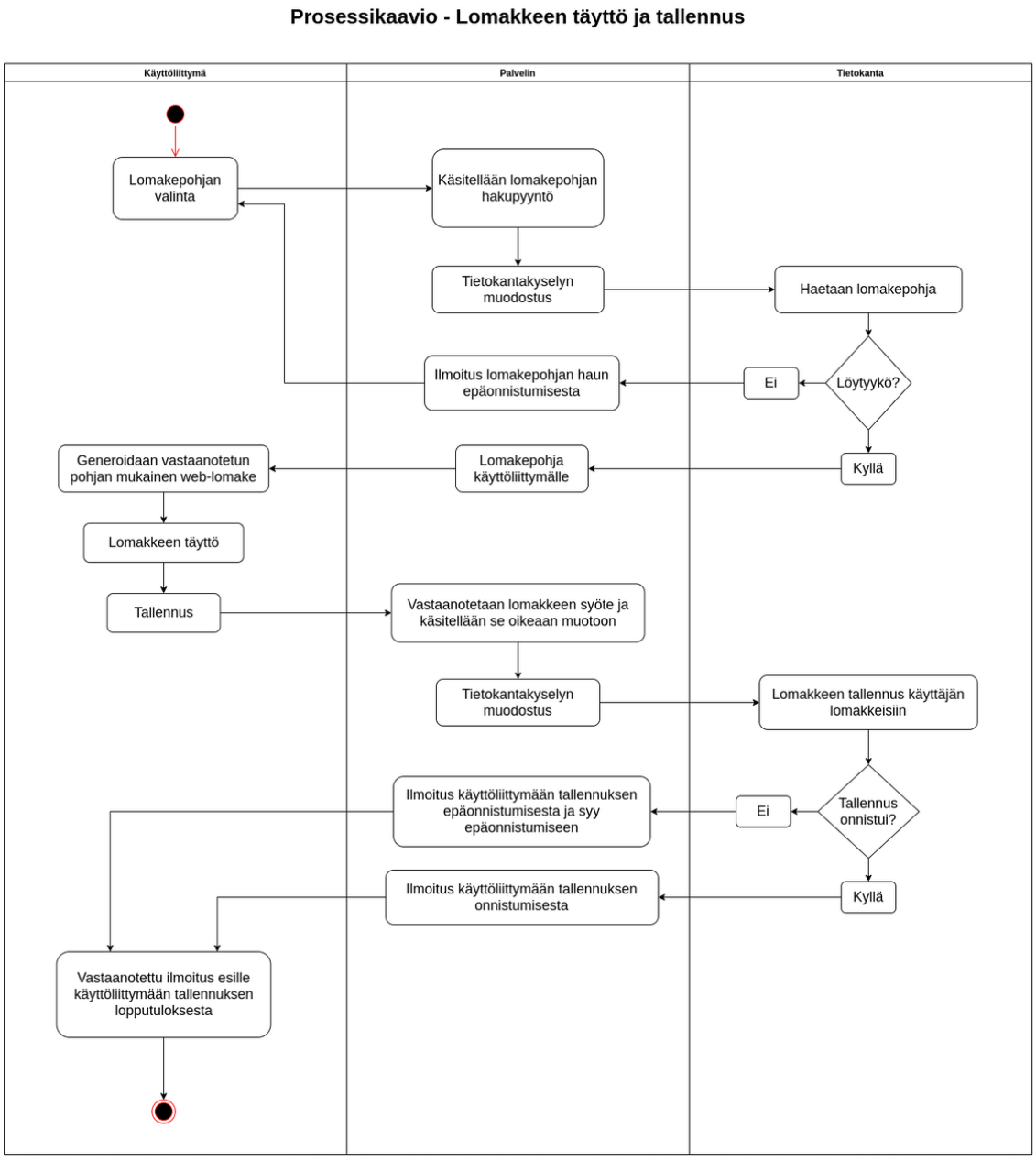
Asianhallintajärjestelmän ei-toiminnallisiin vaatimuksiin määriteltiin muun muassa järjestelmän laatuun, tietoturvaan sekä suorituskykyyn liittyviä ehtoja ja vaatimuksia, jotka järjestelmän toiminnallisten vaatimusten tulisi täyttää ydintoimintonsa suorittamisen lisäksi. Asianhallintajärjestelmän ei-toiminnallisten vaatimusten yläkäsitteiksi määriteltiin tietoturva, tehokkuus, helppokäyttöisyys, saavutettavuus sekä skaalautuvuus. Tietoturvan ollessa nykypäivänä kasvavissa määrin ratkaisevassa asemassa järjestelmän käytön ja käyttäjien turvallisuuden kannalta, järjestelmän on taattava sen käsittelemien tietojen eheys ja suojaaminen leviämiseltä haitallisille toimijoille lain asettamien määräysten mukaisesti. Järjestelmän on toimittava riittävän tehokkaasti, jotta sen käyttö tuottaa lisäarvoa asianhallintaprosessin käsittelyyn ja käyttäjän käyttökokemus sekä vasteajat ovat miellyttävät. Toimiakseen tehokkaasti myös isojen työkuormien alla, järjestelmän on oltava skaalautuva sekä kevyt ajaa, jotta se pystyy tarvittaessa palvelemaan suurempia samanaikaisia käyttäjämääriä. Helppokäyttöisyyden osalta määriteltiin, että järjestelmän on oltava yksiselitteinen ja helposti ymmärrettävä, jotta sen käyttö onnistuu pienellä vaivalla. Järjestelmän tarjoaman käyttöliittymän on oltava responsiivinen niin, että sen käyttö onnistuu tarvittaessa myös mobiililaitteilla. Vuonna 2019 EU-saavutettavuusdirektiivin seurauksena voimaan astunut laki digitaalisten palvelujen tarjoamisesta velvoittaa viranomaiset ja julkishallinnon tekemään digitaaliset palvelunsa saavutettaviksi (Valtiovarainministeriö 2021.). Näin ollen myös asianhallintajärjestelmän on oltava saavutettava ja täytettävä saavutettavuuden kriteeristö lain ja saavutettavuusdirektiivin asettamien vaatimusten mukaisesti WCAG, eli Web Content Accessibility Guidelines, verkkosisällön saavutettavuusohjeiden AA-tasolla. Saavutettavuuden yhteydessä huomioidaan myös järjestelmän esteettömyys. WCAG on kansainvälisiä WWW-standardeja kehittävän ja ylläpitävän W3C:n, eli The World Wide Web Consortiumin, määrittämä kokoelma ohjeita ja standardeja verkkosisällön saavutettavuuden parantamiseksi (W3C 2021.)

4.1.2 Käyttötapauskuvaukset

Käyttötapauskuvaukset ovat kuvauksia järjestelmän toimintojen käyttötapauksista, joissa avataan järjestelmän eri osien ja toimijoiden välisiä vuorovaikutussarjoja (Stephens 2015, 78). Kuvattavia vuorovaikutussarjoja voivat olla esimerkiksi käyttöliittymässä suoritettujen toimintojen vaikutus palvelin- ja tietokantakerroksessa suoritettaviin operaatioihin, sekä näistä seuraaviin käyttöliittymän näkymien muutoksiin. Käyttötapauskuvaukset sisältävät kuvauksen ihannetilannetta vastaavasta toimintaketjusta, eli niin sanotusta Happy Day- / Happy Path-skenaariosta, sekä kuvaukset vaihtoehtoisista tapahtumankuluista. Vaihtoehtoiset tapahtumienkulkujen tarkoituksena on usein kuvata järjestelmän toimintaa virhetilanteissa, sekä mallintaa ydintoiminnosta poikkeavia vaihtoehtoisesti suoritettavia jatkotoimia. Asianhallintajärjestelmän prototyypin kehitystyön ensimmäisen iteraation osana toteutettiin käyttötapauskuvaukset lomakkeen täytöstä täyttäjän näkökulmasta, lomakepohjan luonnista, lomakepohjan päivityksestä, lomakkeen käsittelyn läpiviennistä käsittelijän ja täyttäjän näkökulmista sekä käyttäjäprofiilin tietojenhallinnasta.

4.1.3 Prosessikaaviot

Keskeisimmistä käyttötapauskuvauksista piirrettiin myös prosessikaaviot, jotka kuvaavat järjestelmän toimintaa sovelluksen eri kerroksissa. Prosessikaaviot toteutettiin UML-kaavioina, joissa vuorovaikutuksessa oleviksi toimijoiksi oli määritelty client eli selaimessa ajettava sovelluksen käyttöliittymäkerros, palvelin eli asianhallintajärjestelmän perustana toimiva Kotlin pohjainen palvelin sekä tietokanta eli järjestelmän PostgreSQL-relaatiotietokantakerros. Prosessikaavioiden tavoitteena oli mallintaa sovelluskerrosten välistä vuorovaikutusta toistensa kanssa pseudokoodin tapaan, sekä nostaa esiin järjestelmän toiminnallisuuksien vaatimia prosesseja.



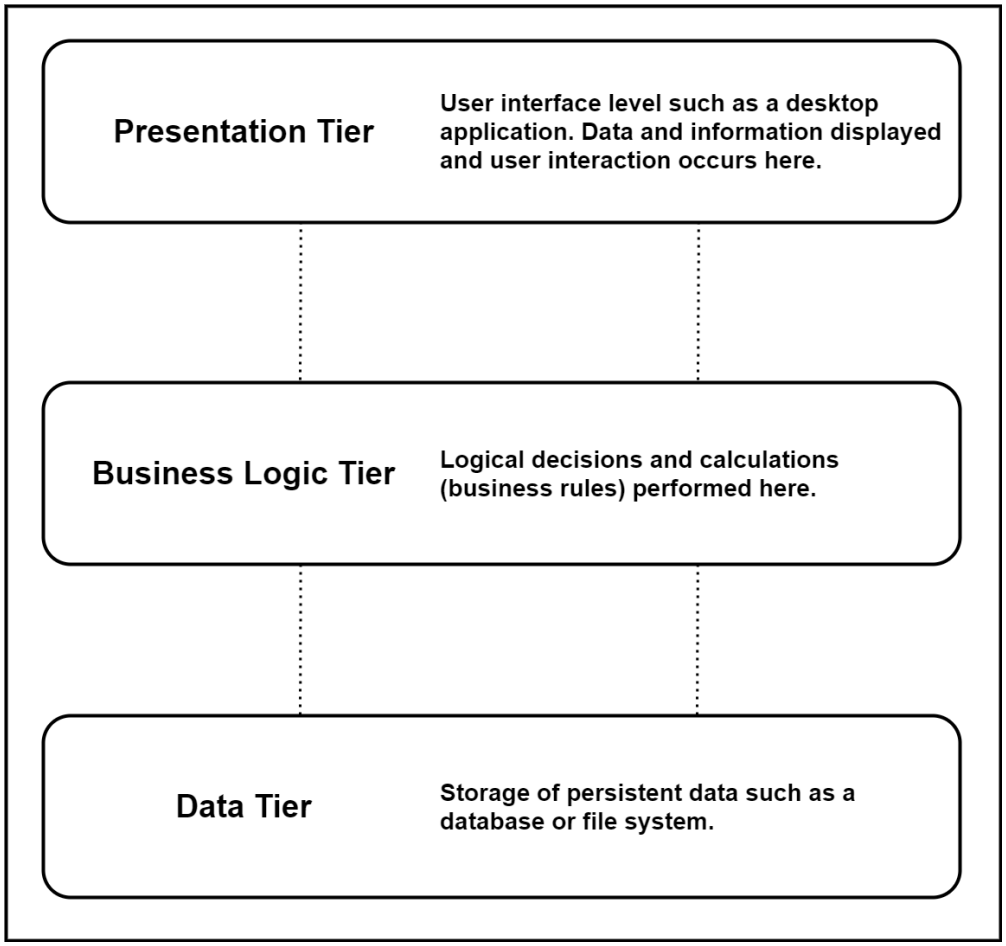
Kuvio 1: Prosessikaavio - Lomakkeen täyttö ja tallennus

4.2 Arkkitehtuuri

Sovellusarkkitehtuuri on abstrakti malli sovellusjärjestelmän rakenteesta, joka kuvaa järjestelmän koostavien osien ja elementtien rakenteellisia suhteita toisiinsa ohjelmistosuunnittelun tarkoitusten saavuttamiseksi ja määritettyjen suunnitteluominaisuuksien ilmentämiseksi (Zhu 2005, 105). Asianhallintasovelluksen arkkitehtuuri koostuu kolmesta päätasosta: Vue.js web-käyttöliittymästä, Kotliniin pohjautuvasta Ktor-palvelimesta sekä PostgreSQL-relaatiotietokannasta. Vue-käyttöliittymä edustaa sovelluksen esitystasoa, joka huolehtii datan esittämisestä ja vuorovaikutuksesta käyttäjän kanssa.

Liiketoiminnan logiikasta (engl. business logic) huolehtivana tasona sovelluksessa toimii pääasiassa Ktor-palvelin, joka suorittaa datan käsittelyyn ja validointiin, web-clientin kautta lähetettyihin pyyntöihin sekä käyttäjäkohtaisen tietokantapääsyn rooli- ja käyttöoikeuskohtaiseen hallintaan liittyvät loogiset päätökset ja operaatiot. Ktor on Kotliniin pohjautuva asynkroninen sovelluskehys mikropalveluiden ja web-sovellusten luontiin, jonka avulla voidaan kehittää esimerkiksi toisiinsa client-server mallin mukaisesti yhteydessä olevia HTTP-sovelluksia (JetBrains 2021). Joitain lomakedatan tallennukseen ja validointiin liittyviä toimintoja suoritetaan myös jo käyttöliittymätasolla ennen datan lähettämistä palvelimelle, mutta pääasiassa palvelin vastaa täysin liiketoiminnan logiikasta huolehtimisesta.

Sovelluksen kolmas taso, datataso, muodostuu PostgreSQL-relaatiotietokannasta, joka huolehtii datan säilömisestä ja eheydestä, sekä sen tallentamiseen, hakuun, muokkaukseen, päivittämiseen, poistamiseen liittyvistä CRUD-operaatioista. PostgreSQL-tietokanta kommunikoi Ktor-palvelimen kanssa ja suorittaa datan hallintaan liittyviä operaatioita palvelimen muodostamien tietokantakyselyiden perusteella. Sovelluksen palvelin on yhteydessä sovelluksen muihin tasoihin client-server arkkitehtuurin mukaisesti molemmissa rooleissa, toimiessaan Vue-clientille palvelimena ja muodostaessaan tietokanta-clientin kommunikoidessaan PostgreSQL-tietokannan kanssa. Näiden lisäksi sovellus hyödyntää tietokannan ulkopuolista Java KeyStore-arkistoa sähköisen allekirjoituksen varmentamiseen käytettävien sertifikaattien säilömiseen, sekä tallentaa lomakkeiden pohjalta generoidut allekirjoitetut PDF-tiedostot levylle sovelluksen tiedostojärjestelmään.



Kuvio 2: Classic 3-tier architecture (Crookshanks 2014, 117.)

Käyttöliittymäkomponentit ja palvelin on toteutettu seuraamalla olio-ohjelmoinnin periaatteita. Ktor-palvelimen sovellusrakenne on pilkottu luokkiin, jotka sisältävät johonkin tiettyyn palveluun keskeisiä toimintoja. Vue-käyttöliittymän komponentit on toteutettu luokkakomponentteina, joiden toiminnallisuuden määrittävä script-osuus itsessään seuraa TypeScriptin luokkarakennetta. Luokkakomponentit mahdollistavat komponenttien täyden toiminnallisuuden sekä Vue-templaten HTML-rakenteen rajatun periyttämisen toiseen komponenttiin olio-ohjelmoinnista tutun luokkarakenteen mukaisesti. Olio-ohjelmointi on suunnittelu- ja ohjelmointisuuntautunut ohjelmistokehitysparadigma, joka määrittelee kehitettävän järjestelmän luokkien kokoelmana, joista voidaan luoda itsenäisesti muiden sovelluksen osien kanssa kommunikoivia olioita eli ilmentymiä (Oussalah 2014, 3–9). Ohjelmistokehitysparadigma määrittää kuinka tietotekninen ratkaisu on muotoiltava selkeästi määriteltyjen käsitteiden ja mekanismien mukaisesti sekä miten ongelma pitäisi käsitellä, muodostaen oman tapansa ratkaisujen analysointiin, suunnitteluun ja kehittämiseen. Paradigma ohjaa kehitystä tiettyyn suuntaan, muttei ota kantaa toimintokohtaisiin ongelmiin, vaan jättää niiden ratkaisemisen kehittäjän vastuulle. (Oussalah 2014, 2.)

Tuotantoympäristössä sovelluksen osat kootaan Docker-konttiin ja ajetaan kontitettuna, jolla varmistetaan, että ajoympäristöstä huolimatta sovellus voidaan koota, käynnistää ja ajaa aina samaan tapaan. Sovelluksen tietokantana hyödynnettävää PostgreSQL-relaatiotietokantapalvelinta ajetaan kehitysympäristössä paikallisessa Docker-kontissa, joka perustetaan sovelluksen ajoympäristöön varsinaisen asianhallintajärjestelmän rinnalle.

4.3 Toteutus

Prototyypin toteutus alkoi kehitysympäristön perustamisella, jonka tavoitteena oli alustaa Kanto-sovelluksen pohja kehittämisen perustaksi ja luoda alustavat konfiguraatiot sovelluksen ajoon ja kokoamiseen. Kehitysympäristön perustamisen aikana alustettiin Kanto-sovelluskehys kehitystyön pohjaksi ja määritettiin sovelluksen ajokonfiguraatiot niin kehitys- kuin tuotantoympäristössä. Sovelluspohjan rakentaminen aloitettiin ajamalla alustusskripti, joka alusti sovelluksen juurihakemistoon Kanto-kehyksen tarvitsemat riippuvuudet ja konfiguroi Kannon sovelluksen git-alimoduuliksi. Projektin alustamisen jälkeen rakennettiin sovelluksen pohjan perusrakenne ja määritettiin Gradle-kokoamiseen sekä Docker-kontitukseen tarvittavat konfiguraatiotiedostot. Järjestelmälle määriteltiin erikseen Docker-konttiin perustettava PostgreSQL-tietokanta lokaaliin kehitysympäristöön sekä CI-putken kautta testi-/tuotantoympäristöön, joita varten luotiin ympäristökohtaiset Docker-kontin perustamiseen hyödynnettävät docker-compose.yml konfiguraatiotiedostot.

4.3.1 Sovelluskonfiguraation ja tietokantarakenteen määrittäminen

Kanto-sovelluskehys tarjoaa valmiita elementtejä tietojärjestelmän palvelimen ja web-käyttöliittymän luontiin, mutta itse sovellus ja sen perustan koostavat osat täytyy kehittäjän luoda ja rakentaa osa kerrallaan itse. Kanto-kehys nojaa vahvasti ennalta määritettyyn sovelluskonfiguraatioon, joka pohjautuu pääosin PostgreSQL relaatiotietokannan taulurakenteeseen. Sovelluskonfiguraatio määrittää sovelluksen käyttöliittymärakenteen lisäksi sovelluskonfiguraation elementtirakenteen mukaisille osille muun muassa oikeus- ja näkyvyysmääreet käyttäjäroolikohtaisesti taulurakenteen attribuuttitasolle asti.

Kehityksen alkuvaiheessa määriteltiin ensimmäisenä alustava tietokantarakenne, johon koottiin taulumäritykset käyttäjiin, käyttöoikeuksiin sekä lomakkeisiin ja lomakepohjiin liittyvän datan säilömiseen. Tietokantarakenteen määrittely osoittautui odotettua haastavammaksi sovelluksen käyttötarkoituksen ja tietorakenteiden dynaamisen muokkaamisen tarpeen vuoksi. Järjestelmän tavoitteena oli tarjota alusta lomakkeiden ja lomakepohjien hallintaan, jonka kautta on mahdollista luoda uusia lomakepohjia sekä muokata ja poistaa dynaamisesti järjestelmään aiemmin tallennettuja lomakepohjia. Määriteltyjen lomakepohjien pohjalta täyttäjien on mahdollista täyttää ja lähettää lomakkeita myös käsittelyyn asianhallintaprosessin käynnistämiseksi.

Relaatiotietokanta on rakenteeltaan hyvin staattinen sen tietorakenteiden perustuessa ennalta määritettyyn skeemaan, jossa määritellään taulukohtaisesti ennalta tarvittavat sarakkeet ja niiden datatyypit. Asianhallintajärjestelmän tapauksessa ei voitu määrittää valmiiksi lomakepohjan tietorakenteen määrittävää tietokantataulua, joka sisältäisi kaikki lomakepohjien yleiset tiedot sisältävät sarakkeet lomakkeelle määritettyine kenttineen, koska yksittäisen lomakepohjan kenttärakennetta ei voida tietää ennestään, eikä tietokantataulujen skeemojen dynaaminen muodostaminen ja muokkaus ole mahdollista. Yhtenä vaihtoehtoisena ratkaisuna olisi voitu määrittää keskitetty yleispätevä lomakepohja-taulu, joka sisältää kaikki mahdolliset lomakepohjien kentät kaikkien lomakepohjien kesken. Liian laaja määrittely olisi kuitenkin aiheuttanut tarpeetonta sarakkeiden sisällyttämistä ja mukana kuljettelua jokaisen lomakepohjan kantaessa kaikkia mahdollisia lomakepohjille määritettäviä sarakkeita, suurimmalta osalta tyhjänä, vaikuttaen heikentävästi datan normalisointiin. Myös tietokantakyselyiden muodostaminen olisi monimutkaistunut turhaan, joten ratkaisu ei ollut tyydyttävä järjestelmän tarpeisiin.

Lopulta ratkaisuna päädyttiin määrittämään tietokantarakenne, joka pohjautuu lomakepohjien, lomakkeiden sekä niiden kenttien ja kenttien sisällön metarakenteisiin. Tietokanta tuntee lomakkeiden ja lomakepohjien sekä niiden kenttien ja kenttien sisällön kuvaavat ominaisuudet, kuten yksilöivän tunnuksen, nimen, luomis- ja päivityspäivämäärät, tilan sekä lomakkeen tapauksessa täyttäjän ja käsittelijän, muttei ota kantaa yksittäisen lomakkeen tai lomakepohjan rakenteeseen. Lomakkeen rakenne määritellään kenttäkohtaisesti luomalla relaatio kentän ja lomakepohjan välille lomakepohjan yksilöivän tunnuksen avulla. Täytettäessä uutta lomaketta, haetaan valitun lomakepohjan mukaiset kentät saman tunnuksen avulla tietokannasta ja rakennetaan niistä lomake täytettävään muotoon käyttöliittymässä. Vastaavasti lomakkeen kenttäkohtainen sisältö yhdistetään oikeaan kenttään ja lomakkeeseen kentän ja tallennetun lomakkeen yksilöivien tunnusten perusteella. Yksittäisen lomakepohjan ja lomakkeen rakenteen tunnistamisen sekä kokoamisen vastuu on käyttöliittymä- ja palvelinkerroksissa, jotka sisältävät tarvittavan logiikan rakentamiseen tarvittavien operaatioiden sekä tietokantakyselyiden suorittamiseen.

4.3.2 Käyttöliittymän pohja - sisäänkirjautuminen ja kotinäkymä

Sovelluskonfiguraation ja tietokantarakenteen määrittelyn jälkeen seuraava luonnollinen vaihe prototyypin kehittämistyössä oli rakentaa pohja Vue-käyttöliittymälle, joka toimisi käyttäjän ja palvelimen välisenä rajapintana selaimessa mahdollistaen asianhallintaprosessiin liittyvien toimintojen suorittamisen. Käyttöliittymä tukee useita eri kieliversioita ja tekstisisällöt on mahdollista kääntää suomeksi, ruotsiksi ja englanniksi missä vain sovelluksen osassa ilman sivun uudelleenlataamista. Asianhallintajärjestelmän web-käyttöliittymä toteutettiin Single-Page Applicationina, joka koostuu toimintojen suorittamiseen hyödynnettävistä näkymistä, joiden näkyvyys vaihtelee eri roolien välillä. Single-Page App on web-sovellus, joka

lataa kaikki sovelluksen tarvitsemat käyttöliittymärakenteet sekä niiden toiminnallisuuden sisältävät skriptit ensimmäisellä sivulatauksella. Navigoitaessa sovelluksessa uusien sivujen rakennetta ei tarvitse välttämättä ladata uudelleen joka kerta siirryttäessä sovelluksen osasta toiseen, vaan käyttöliittymän näkymiä ja toimintoja ohjataan datan muutosten perusteella. Sovellus voi olla yhteydessä palvelimeen noutaessaan tai lähettäessään dataa, mutta kaikki Single-Page Appit eivät välttämättä keskustele palvelimen kanssa ensimmäisen sivulatauksen jälkeen enää ollenkaan. (Flanagan 2006, 497.) Asianhallintajärjestelmän prototyyppi on kuitenkin aktiivisesti yhteydessä palvelimen kanssa vielä rakenteen lataamisen jälkeen sen noutaessa näkymistä ja suoritettavista toiminnoista riippuen tarvittavaa dataa.

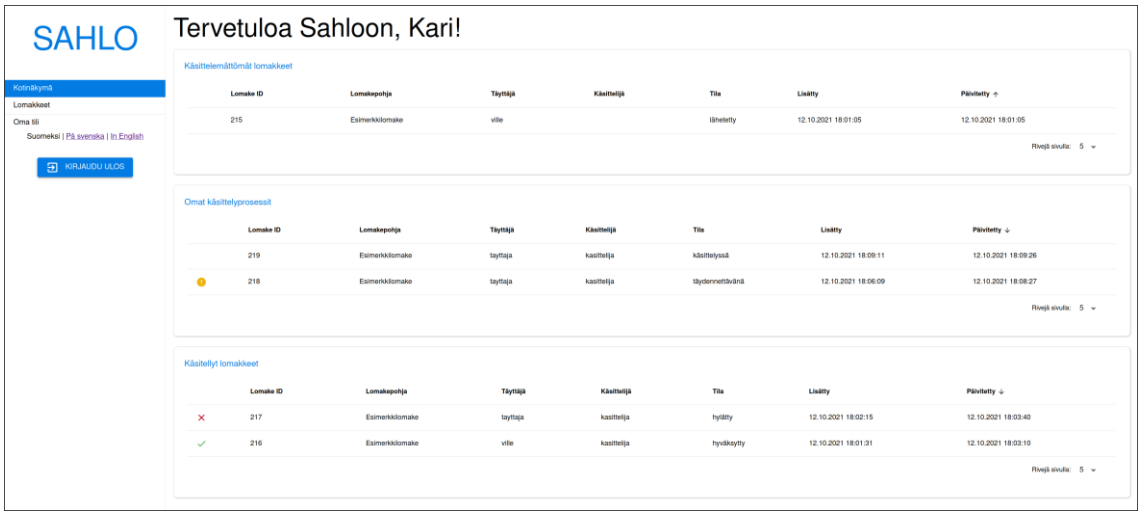
Ensimmäisessä kehitysvaiheessa järjestelmään toteutettavat näkymät rajattiin koti-, lomakkeen täyttö- ja käsittely-, lomakepohjan luonti- ja käyttäjäprofiilinäkymiin, joiden kautta käyttäjät pystyvät suorittamaan järjestelmän keskeisimmät toiminnot rooliensa määrittämien oikeuksien puitteissa. Näkymät toteutettiin pääsääntöisesti omina Vue-luokkakomponentteinaan, joiden sisältämiä käyttöliittymärakenteita pilkottiin vielä omiin komponentteihinsa komponenttien ja sovellusrakenteen ylläpidettävyyden ja uudelleenkäytettävyyden edistämiseksi. Näkymien välinen reititys hoidettiin käyttöliittymätasolla Vue-ekosysteemiin kuuluvalla Vue Routerilla. Vue Router mahdollistaa Single Page Appien koostavien komponenttien kartoittamisen sovelluksen eri reiteille, sekä huolehtii komponenttien renderöinnistä siirryttäessä komponentille rekisteröityyn polkuun. Asianhallintajärjestelmän prototyypin tapauksessa sovelluksen juurikomponenttiin sijoitettiin Vue Routerin renderöintielementti, joka heijastaa aktiivisen näkymän käyttäjän näkyville sovelluksen aktiivisen reitin mukaan.

Käyttöliittymään toteutettiin ensimmäisenä sovelluksen käytön autentikoinnista vastaava sovelluksen latuduttua oletuksena aukeava sisäänkirjautumisnäkymä, joka huolehtii käyttäjän sisäänkirjautumisen käsittelystä ja syötettyjen tunnusten välittämisestä palvelimelle. Sisäänkirjautumisen onnistuessa palvelin palauttaa käyttöliittymään istuntokeksin (engl. session cookie), joka sisältää käyttäjän istuntoon ja roolitukseen oleellisia tietoja, ja ohjaa käyttäjän rooliaan vastaavaan kotinäkymään. Kotinäkymä toimii kirjautuneen käyttäjän keskusnäkymänä, johon kootaan roolille keskeistä tietoa ja toimintoja, ja on kaikille käyttäjärooleille erilainen.

Täyttäjän kotinäkymässä on listaus täyttäjän omista täytetyistä, käsittelyssä olevista ja käsitellyistä lomakkeista, sekä navigointimahdollisuus uuden lomakkeen täyttämiseen. Lomakelistauksen kautta täyttäjä voi myös siirtyä lomakkeen tarkastelunäkymään.

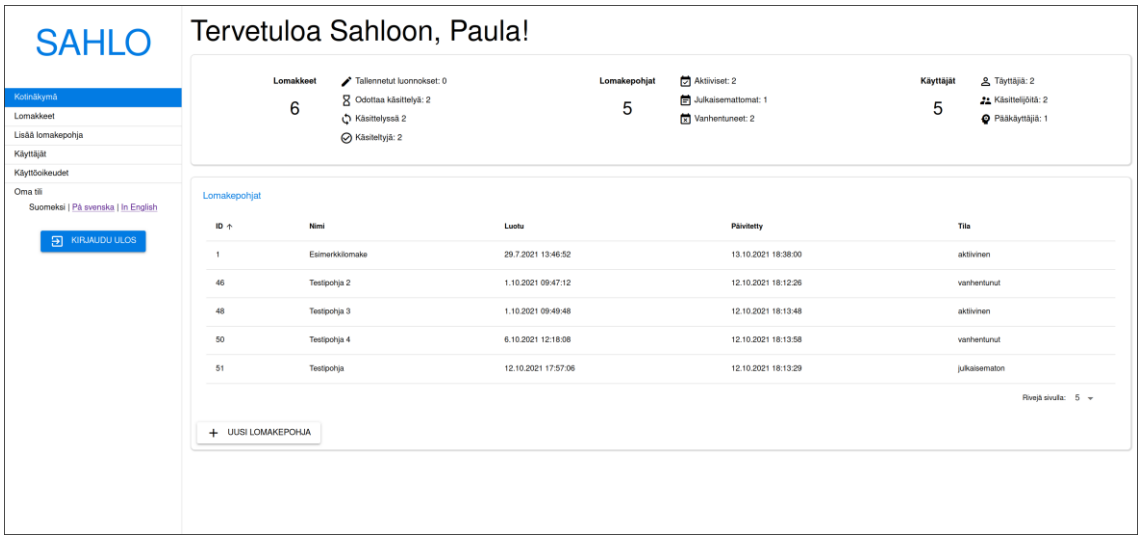
Kuvio 3: Täyttäjän kotinäkymä

Käsittelijällä on koonti käsittelyyn lähetetyistä lomakkeista, käsittelijän omassa käsittelyssä olevista lomakkeista sekä käsittelijän käsittelemistä lomakkeista. Käsittelijä pystyy siirtämään lähetettyjen lomakkeiden listauksen kautta lomakkeita omiin käsiteltäviin lomakkeisiinsa sekä siirtymään käsittelynäkymään käsittelyssä olevien lomakkeiden listauksesta.



Kuvio 4: Käsittelijän kotinäkymä

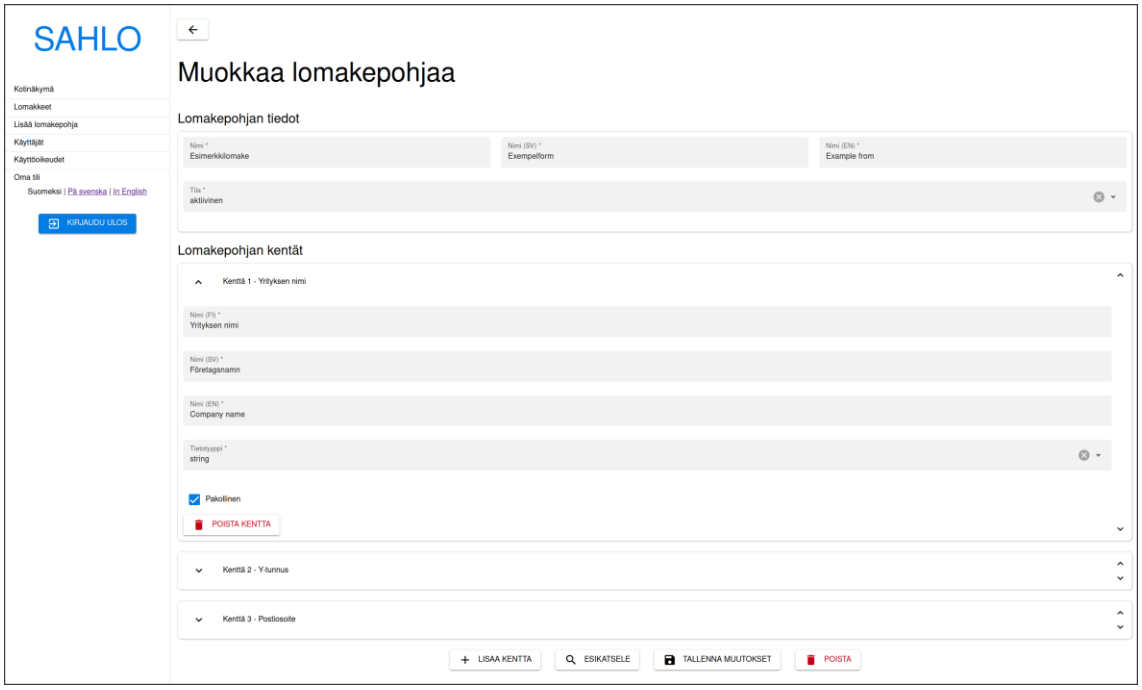
Pääkäyttäjän kotinäkymässä on listaus järjestelmään tallennetuista lomakepohjista sekä statistiikkapaneeli, johon kootaan järjestelmän ylläpidon kannalta olennaista dataa järjestelmän lomakepohjista, lomakkeista ja käyttäjistä. Lomakepohjalistauksesta pääkäyttäjä voi siirtyä suoraan tarkastelemaan sekä muokkaamaan yksittäistä lomakepohjaa.



Kuvio 5: Pääkäyttäjän kotinäkymä

4.3.3 Lomakepohjaeditori

Lomakkeiden täyttämisen mahdollistamiseksi järjestelmään täytyi pystyä tallentamaan lomakepohjia säännöstöinä, joiden mukaan täytettävä lomake voitaisiin generoida käyttöliittymään täytettäväksi. Lomakepohjien säännöstöjen koodaus ennalta käsin kävisi ennen pitkää työlääksi ja olisi järjestelmän käytettävyyden kannalta kankeaa, joten asianhallintajärjestelmän prototyyppiin päätettiin toteuttaa yksinkertainen web-editori työkaluksi lomakepohjien dynaamisen luomisen ja muokkaamisen tueksi. Lomakepohjaeditorin kautta järjestelmän pääkäyttäjä pystyy luomaan uusia tai muokkaamaan jo aikaisemmin järjestelmään tallennettuja lomakepohjia, jotka koostuvat pääkäyttäjän määrittelemistä kentistä. Kenttiä voidaan lisätä tai poistaa yksi kerrallaan, ja niitä voidaan järjestellä eri järjestykseen pystysuunnassa. Kenttäjärjestys heijastuu suoraan lomakkeen täyttönäkymään sekä lomakkeen pohjalta generoitavaan PDF-tiedostoon.



Kuvio 6: Lomakepohjaeditori muokattaessa tallennettua lomakepohjaa

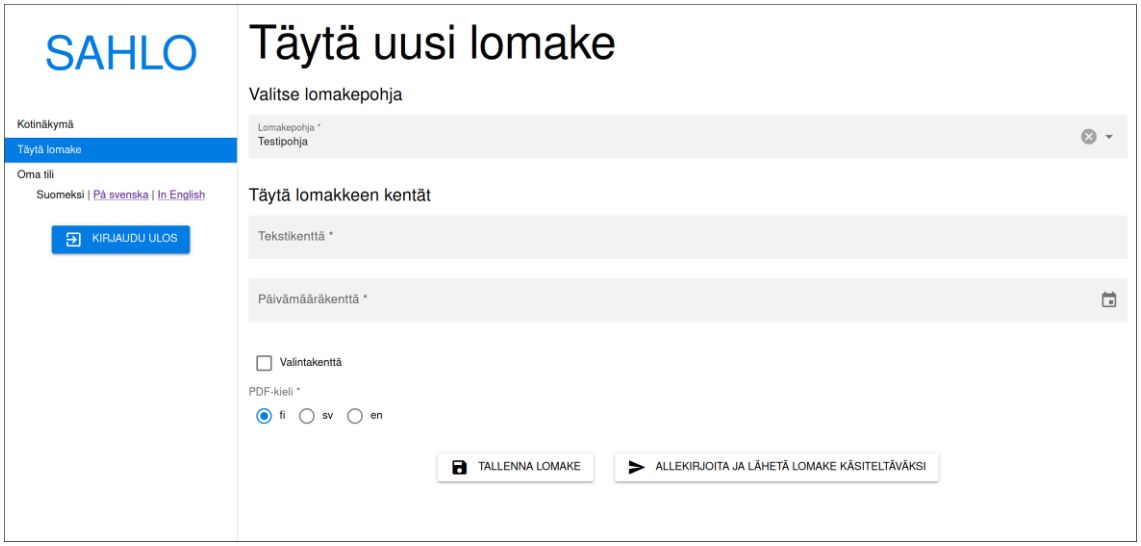
Lomakepohja muodostuu lomakepohjan yleisistä tiedoista sekä kenttämäärittelyistä. Yleiset tiedot sisältävät lomakkeen nimen eri kieliversioilla ja lomakepohjan tilan, kun taas kenttämäärittelyt sisältävät kentän nimen suomeksi, ruotsiksi ja englanniksi, kentän pakollisuuden sekä kentän tietotyypin. Kenttien pakollisuus ja tietotyyppi vaikuttavat tallennettavan lomakkeen kenttien validointiin, jonka lisäksi kentän rakenne käyttöliittymässä muuttuu syötteelle soveltuvaksi kentäksi sen tietotyypin mukaan. Lomakepohjaeditori huolehtii lomakepohjan ja sen kenttämäärittelyiden relaatioiden muodostamisesta lisäämällä tarvittavat tunnisteet viitteiksi tallennettaviin lomakepohjan sekä kenttämäärittelyiden metatietoihin. Lomakepohjista tallennetaan automaattisesti metatietoja, joita käyttäjä ei pysty itse muokkaamaan, kuten lomakepohjan luomispäivämäärä, päivityspäivämäärä, luoja sekä lomakepohjan ja sen kenttien yksilöivät tunnukset. Lomakepohjaeditori mahdollistaa myös muokattavana olevan lomakepohjan esikatselun ennen muutosten tai uuden lomakepohjan tallennusta siinä muodossa, kuin lopullinen lomakepohja näyttäytyisi lomakkeen täyttäjälle käyttöliittymässä tämän täyttäessä lomakepohjan mukaista lomaketta.

Tallennettaessa uutta lomakepohjaa tai muutoksia jo olemassa olevaan lomakepohjaan, pohjalle täytetyt yleistiedot ja kenttämäärittelyt validoidaan ennen tallennuksen suorittamista. Muokattaessa olemassa olevaa pohjaa, käyttäjällä on mahdollisuus tallentaa muutokset vanhaan pohjaan tai luoda niistä uusi lomakepohjaversio, joka korvaa vanhan lomakepohjan. Tallennettaessa uusi lomakepohjaversio, vanhan lomakepohjan tila muutetaan vanhentuneeksi ja järjestelmään tallennetaan muokattu kopio vanhasta lomakepohjaversiosta kenttärakenteineen. Uusi lomakepohjaversio julkaistaan ja siihen asetetaan vanhan pohjan yksilöivä tunniste

viitteeksi, jonka avulla on mahdollista koota versiohistoria lomakepohjaversioiden välillä. Vanhentuneet lomakepohjat eivät näy enää täyttäjille näiden valitessa lomakepohjaa uuden lomakkeen pohjaksi, mutta vanhan version pohjalta aiemmin tallennetut ja käsittelyyn lähetetyt lomakkeet säilyvät ja niiden käsittelyprosessi voidaan viedä tarvittaessa loppuun. Vastuu muutoksista aiheutuvien konfliktien arvioinnista on jätetty lomakepohjan laatijalle. Järjestelmä suosittelee tallentamaan vanhaan pohjaan vain pieniä muutoksia, kuten kirjoitusvirheiden korjauksia tai kenttien nimien ja kieliversioiden päivityksiä. Isommat kenttärakennetta muuttavat sekä muut potentiaalisesti konflikteja aiheuttavat muutokset suositellaan tallentamaan kokonaan uutena lomakepohjaversiona.

4.3.4 Lomakkeen täyttö- ja muokkausnäkymät

Lomakkeen täyttö- ja muokkausnäkymät ovat keskiössä uusien lomakkeiden luonnissa ja tallentamisessa asianhallintaprosessien käynnistämiseksi. Täyttönäkymä on uuden lomakkeen luomiseen ja tallennukseen tarkoitettu näkymä, kun taas lomakkeen muokkausnäkymä mahdollistaa jo aiemmin tallennetun lomakkeen muokkaamisen vielä ennen sen lähettämistä käsittelyyn. Eri komponentteihin jaetut näkymät ovat lähes identtiset niiden hyödyntäessä samaa lomakkeen kenttärakennekomponenttia, joka sisältää lomakkeen kenttärakenteen kokoavan logiikan. Lomakkeen kenttärakenne keskitettiin ylläpidettävyyden parantamiseksi ja samojen rakenteiden toistuvan koodaamisen välttämiseksi omaan komponenttiinsa, josta se on helppo heijastaa kaikkiin rakennetta tarvitseviin sovelluksen osiin. Keskitetyn rakennekomponentin avulla kenttärakenteen muutokset heijastuvat automaattisesti lomakerakennetta hyödyntäviin näkymiin, kuten lomakkeen täyttö- ja muokkausnäkymiin, sekä lomakepohjan esikatselunäkymään.



Kuvio 7: Uuden lomakkeen täyttönäkymä

Muokkausnäkymästä eroten täyttönäkymässä on lomakepohjan valintaa varten oma valintarakenne, johon haetaan kaikki pääkäyttäjän julkaisemat aktiiviset lomakepohjat. Täyttäjän valitessa lomakepohjan valikosta, tietokannasta noudetaan lomakepohjan yksilöivän tunnuksen mukaiset kenttämäärittelyt, joiden pohjalta rakennetaan täytettävä lomake käyttöliittymään kenttärakennekomponentissa. Koottu lomakerakenne tukee kaikkia sovelluksen tukemia kielivaihtoehtoja ja kääntää kenttärakenteen tekstisisällöt käyttäjän kielivalintaa vastaavalle kielelle muutettaessa sovelluksen kieltä ilman erillistä sivulatausta. Lomakkeen kenttien lisäksi näkymät sisältävät kenttämäärittelyistä erillisen PDF-kielen valintakentän, jolla täyttäjän on mahdollista valita lomakkeesta tulostettavan PDF-tiedoston kieli erikseen riippumatta käyttöliittymän kielestä lomakkeen täytön aikana.

Molemmat näkymät mahdollistavat lomakkeen ja siihen tehtyjen muutosten tallentamisen. Täytettäessä uutta lomaketta täyttönäkymässä lomakepohja on mahdollista lähettää suoraan käsittelyyn, jolloin täytetty lomake tallennetaan sähköisen allekirjoituksen suorittamisen yhteydessä järjestelmään. Muokkausnäkymään siirrytään sisällön listaavan tarkastelunäkymän kautta, jonka kautta suoritetaan kaikki tallennetulle lomakkeelle suoritettavat jatkotoimet. Jos muokattavana oleva tallennettu lomake halutaan lähettää käsittelyyn, sen lähettäminen on mahdollista täyttäjän palattua muokkausnäkymästä tallennuksen jälkeen tarkastelunäkymään, joka sisältää toiminnon lomakkeen lähettämiselle. Ennen lomakkeen lähettämistä käsittelyyn lomakkeen kenttien sisältö validoidaan ja järjestelmä tarkistaa onko kaikki pakolliseksi merkatut kentät täytetty. Validoinnin ehdot määräytyvät kenttätyypin ja suoritettavan tallennuksen kontekstin mukaan. Lomake voidaan tallentaa keskeneräisenä järjestelmään, jolloin tallennuksen yhteydessä ei suoriteta kenttien validointia. Lähetettäessä lomake käsittelyyn kaikki kentät validoidaan aina ennen lähetyksen käynnistämistä. Virhetilanteissa käyttöliittymä antaa käyttäjälle selvennyksen sisältävän ilmoituksen ja keskeyttää lomakkeen lähettämisen.

4.3.5 PDF-tiedoston generointi ja sähköinen allekirjoitus

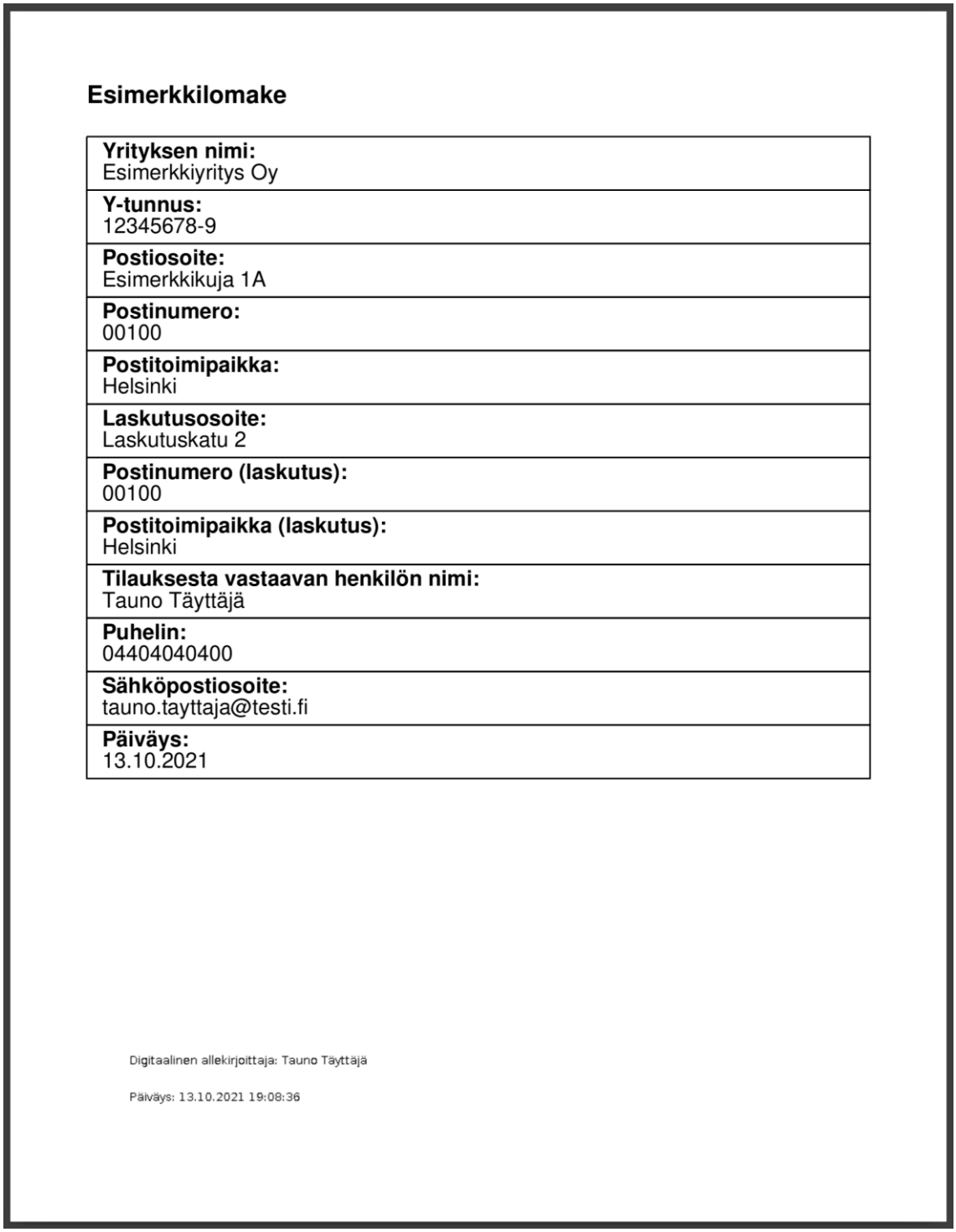
Aloitettaessa lomakkeen käsittelyyn lähettäminen, lomakkeesta generoidaan PDF-tiedosto, joka allekirjoitetaan ennen lomakkeen välittämistä käsittelijöille. PDF-tiedosto sisältää täytetyn lomakkeen sisällön kenttineen, lomakkeen täyttäjän näkyvän allekirjoituksen sekä tiedostoon upotetun digitaalisen allekirjoituksen. PDF-tiedoston generointi ja sähköinen allekirjoitus suoritetaan lomittain yhtenäisenä prosessina, joka käynnistyy täyttäjän lähettäessä tallennetun lomakkeen käsiteltäväksi. Prosessi päättyy järjestelmän asettaessa kortinlukijaohjelmistolta palautuneen allekirjoituksen generoituun PDF-lomakkeeseen ja tallentaessa allekirjoitetun tiedoston levylle.

Sähköisen allekirjoituksen suorittamiseksi käyttäjällä täytyy olla voimassa oleva henkilökortti, aktivoitu kansalaisvarmenne, tietokoneeseen integroitu tai siihen liitettävä älykortinlukija sekä Fujitsu mPollux DigiSign Client-kortinlukijaohjelmisto. Allekirjoitus suoritetaan sähköisesti Digi- ja väestötietoviraston, eli DVV:n myöntämällä kansalaisvarmenteella, joka löytyy henkilökortista. Kansalaisvarmenne täytyy aktivoida henkilökortin vastaanottamisen jälkeen erillisessä kirjeessä saapuvilla aktivointitunnuksilla luettaessa kortti ensimmäistä kertaa kortinlukijaohjelmistolla. DigiSign-kortinlukijaohjelmisto on Fujitsun kehittämä virallinen kortinlukijaohjelmisto suomalaisen henkilökortin sähköiseen käyttöön allekirjoitus-, tunnistautumis- ja varmennetarkoituksiin. DigiSignilla voi aktivoida henkilökortin kansalaisvarmenteen, sekä hallita kortin allekirjoituksen ja tunnistautumisen varmentamiseen tarvittavia tunnuslukuja. (DVV 2021.) DigiSign-ohjelmistoa hyödynnetään asianhallintajärjestelmän prototyypissä sähköisen allekirjoituksen luontiin, joka asetetaan tallennetusta lomakkeesta generoitavaan PDF-tiedostoon. DigiSign ohjelmiston kanssa kommunikointi suoritetaan järjestelmää varten räätälöidyllä versiolla DVV:n kehittämästä Signature Creation Service JavaScript-moduulista. SCS-moduuli on web-sovelluksiin sisällytettävä JavaScript-moduuli, joka sisältää metodeja allekirjoituspyynnön ja sähköisen allekirjoituksen muodostukseen sekä kommunikointiin DigiSign-kortinlukijaohjelmiston ja Signature Creation Servicen kanssa. SCS-moduuli mahdollistaa allekirjoitettavan datan lähettämisen DigiSign-kortinlukijaohjelmiston Signature Creation Servicelle web-selaimessa HTML5-sovelluksesta HTTP- tai HTTPS-protokollan välityksellä. Sovelluksen ja Signature Creation Servicen välillä lähetettävä ja vastaanotettava data välitetään JSON, eli JavaScript Object Notation-formaatissa. (DVV 2017.)

Sähköisessä allekirjoituksessa hyödynnetään DVV:n myöntämiä Valtion kansalaisvarmenne -sertifikaatteja suoritetun allekirjoituksen validointiin. Asianhallintajärjestelmän hyödyntämät sertifikaatit säilötään sovelluksen ajoympäristöön konfiguroitavaan Java KeyStoreen, joka on kryptografisten avainten ja sertifikaattien säilömiseen tarkoitettu salattu arkisto. Jokaiselle KeyStoreen tallennetulle merkinnälle määritetään merkkijonomuotoinen alias, jonka avulla avain tai sertifikaatti voidaan yksilöidä ja hakea arkistosta. Avainten ja sertifikaattien lataamiseen käyttäjä tarvitsee aliaksen lisäksi arkistolle määritetyn salasanan. (Oracle 2021.) Sovelluksen KeyStore konfiguroitiin hyödyntäen kehitysympäristön Java-asennuksen sisältämän keytool komentorivityökalun avulla. Keytool mahdollistaa uuden KeyStoren luonnin sekä ladattujen sertifikaattien asettamisen salattuun arkistoon.

PDF-tiedoston generointi käynnistyy täyttäjän yrittäessä lähettää tallennetun lomakkeen käsittelyyn, jolloin lomakkeen kenttien validoinnin jälkeen palvelimella aloitetaan uuden PDF-tiedoston piirtäminen. Lomakkeen piirtämiseen hyödynnetään Apachen kehittämää Java-pohjaista Apache PDFBox-kirjastoa, joka on avoimen lähdekoodin työkalu PDF-dokumenttien luontiin, manipulointiin sekä sisällön käsittelyyn (The Apache Software Foundation 2021). Palvelin alustaa PDFBoxilla uuden PDF-tiedoston, johon lisätään ensin täyttäjän määrittämää PDF-tulosteen kieltä vastaavat lomakkeen otsikko ja kentät sisältöineen, sekä piirretään

kenttien laatikkorakenne. Lomakkeen sisällön piirtämisen jälkeen luodaan näkyvä ilmentymä sähköisestä allekirjoituksesta, johon merkitään allekirjoittajan nimi käyttäjäprofiiliin tallennetuista tiedoista sekä allekirjoituksen aikaleima. Vaikkei allekirjoitusta ole suoritettu vielä lomakkeen generointivaiheessa, on näkyvä allekirjoitus ja kaikki muut lomakkeen osat generoitava jo ennen allekirjoitusta, sillä allekirjoitettua tiedostoa ei saa enää muuttaa allekirjoituksen jälkeen. Tiedosto välitallennetaan ja ladataan uudestaan tavumatriisiin (engl. bytearray), jonka jälkeen sille määritetään asetettavan allekirjoituksen sijainti valmiiksi. Sijainti määrittää tavualueen (engl. byterange), johon kortinlukijaohjelmistolta palautuva digitaalinen allekirjoitus asetetaan tiedostossa. Sähköisen allekirjoituksen alustuksen jälkeen PDF-tiedosto välitallennetaan uudestaan väliaikaistiedostoksi järjestelmään ja järjestelmä tallentaa PDF-tiedoston metatiedot tietokantaan.



Esimerkkilomake

Yrityksen nimi: Esimerkkiyritys Oy
Y-tunnus: 12345678-9
Postiosoite: Esimerkkikuja 1A
Postinumero: 00100
Postitoimipaikka: Helsinki
Laskutusosoite: Laskutuskatu 2
Postinumero (laskutus): 00100
Postitoimipaikka (laskutus): Helsinki
Tilauksesta vastaavan henkilön nimi: Tauno Täyttäjä
Puhelin: 04404040400
Sähköpostiosoite: tauno.tayttaja@testi.fi
Päiväys: 13.10.2021

Digitaalinen allekirjoittaja: Tauno Täyttäjä

Päiväys: 13.10.2021 19:08:36

Kuvio 8: Esimerkki generoidusta PDF-tiedostosta – Esimerkkilomake

PDF-tiedostosta luodaan SHA-256-tiiviste, joka välitetään käyttöliittymään allekirjoituspyynnön muodostamista varten. Käyttöliittymän vastaanottaessa tiivisteen se muodostaa SCS-moduulin avulla allekirjoituspyynnön, johon lisätään palvelimelta vastaanotettu tiiviste. Käyttöliittymä lähettää allekirjoituspyynnön paikallisesti ajettavalle kortinlukijaohjelmistolle porttiin 53951 tai 53952 riippuen lähetetäänkö pyyntö HTTP- vai HTTPS-protokollalla. DigiSign-

kortinlukijaohjelmiston Signature Creation Service avaa täyttäjälle varmenteen valintaikkunan, jonka valinnan jälkeen täyttäjän tulee syöttää henkilökortille määrittämänsä sähköisen allekirjoituksen PIN-koodi. Allekirjoituksen onnistuessa Signature Creation Service palauttaa sähköisen allekirjoituksen sekä kolmiosaisen varmenneketjun sisältävän JSON-vastauksen käyttöliittymään, josta se välitetään takaisin palvelimelle asetettavaksi generoituun PDF-tiedostoon. Allekirjoituksen epäonnistuessa tai täyttäjän peruuttaessa allekirjoitusoperaation aiemmin välitallennettu PDF-tiedosto sekä sen metatiedot poistetaan järjestelmästä ja lomakkeen käsittelyyn lähettäminen keskeytyy.

Allekirjoituksen asetus tiedostoon toteutettiin PDF-tiedoston generoinnista erillisenä prosessina, joka ajetaan sähköisen allekirjoituksen onnistuessa ja kortinlukijaohjelmiston palauttaessa digitaalisen allekirjoituksen sovellukselle. Palvelimen vastaanottaessa sähköisen allekirjoituksen sisältävän JSON-objektin käyttöliittymältä, se hakee aiemmin välitallennetun PDF-tiedoston välitallennussijainnista ja lataa sen tavumatriisina allekirjoituksen asettamista varten. Tämän jälkeen JSON-objektista poimitaan sähköinen allekirjoitus ja kolmiosainen varmenneketju, jotka alustetaan allekirjoituselementin luontia varten. Signature Creation Service palauttaa sähköisen allekirjoituksen Base64-enkoodatussa muodossa, joten allekirjoitus täytyy vielä dekoodata merkkijonoksi ennen kuin sitä voidaan hyödyntää PDF-tiedostoon asetettavan PKCS7-allekirjoituselementin muodostukseen. PKCS7 eli Public Key Cryptography Standard 7 on alun perin RSA:n kehittämä monikäyttöinen kryptografiaa sisältävän datan, kuten sähköisten allekirjoitusten säilömisen tietoformaatti, joka mahdollistaa useiden sertifikaattien niputtamisen yhteen tiedostoon (Kaliski 1998). JSON-objektin mukana vastaanotettu kolmiosainen varmenneketju poimitaan osissa sertifikaattilistaan, jonka jälkeen siitä ja merkkijonomuotoisesta allekirjoituksesta luodaan aikaleimattu PKCS7-elementti. Allekirjoituselementti asetetaan aiemmin tavumatriisiin ladattuun PDF-tiedostoon sille määritettyyn kohtaan oikealle tavualueelle, joka tallennettiin muistiin istuntomuuttujaan generoitaessa PDF-tiedostoa. Osana PKCS7-elementtiä tiedostoon asetettu varmenneketju validoidaan vertaamalla allekirjoituksessa käytettyä sertifikaattiavainta sovelluksen Java-keystoresta ladattuihin sertifikaatteihin. Allekirjoituksen varmennuksen jälkeen järjestelmä tallentaa allekirjoitetun PDF-tiedoston lopulliseen tiedostosijaintiin, josta se voidaan ladata käyttäjän koneelle käyttöliittymän kautta.

4.3.6 Lomakkeen käsittelyprosessi

Lomakkeiden ja lomakepohjien luonnin ja hallinnan ominaisuuksien lisäksi järjestelmään kehitettiin yksinkertaisen asianhallintatapauksen läpivientiin tarvittavat käsittelijän toiminnot. Ensimmäisen iteraation aikana järjestelmään ei kehitetty lomakepohjakohtaisia toimintoja käsittelyprosessiin tai lomakkeiden sisällön tarkempaan käsittelyyn, vaan toteutetut toiminnot kattavat yleisesti erityyppisten lomakkeiden käsittelyyn tarvittavat perustoiminnot, joiden päälle voidaan myöhemmin rakentaa hienostuneempaa käsittelylogiikkaa. Perustoimintojen avulla täyttäjien lähettämät täytetyt lomakkeet voidaan siirtää käsittelyyn, lomakkeelle voidaan pyytää täydennystä ja aloitettu käsittelyprosessi voidaan viedä päätökseen tai keskeyttää.

Lomakkeen käsittelyprosessi käynnistyy täyttäjän lähettäessä sähköisesti allekirjoitetun lomakkeen käsittelyyn. Lomakkeen lähettämisen jälkeen sen tietoja ei pysty enää muokkaamaan erikseen, ellei lomaketta palauteta täydennettäväksi. Asianhallintajärjestelmän prototyypissä lomakkeiden käsittelystä vastaavat Käsittelijä-roolin käyttäjät, jotka järjestelmän pääkäyttäjä valtuuttaa käsittelijöiksi. Lähetetty lomake siirtyy ”lähetetty”-tilaan odottamaan, että käsittelijä ottaa lomakkeen käsittelyyn.

Käsittelijän kotinäkymä koostuu lomaketaulukoista, joihin on koottu lomakkeita eri kriteerien mukaan. Ensimmäinen taulukko koostuu täyttäjien lähettämistä lomakkeista, joita ei ole vielä siirretty käsittelyyn. Taulukossa listatut lomakkeet näkyvät kaikille järjestelmän käsittelijöille ja ne järjestetään päivityspäivämäärän mukaan vanhimmasta uusimpaan, jotta aikaisemmin lähetetyt lomakkeet otettaisiin käsittelyyn ensimmäisenä. Käsittelijä pääsee taulukon kautta tarkastelunäkymään tarkastelemaan lähetettyä lomaketta ennen sen siirtämistä käsittelyyn. Käsittelijän tarkastelunäkymä hyödyntää samaa lomakkeen sisällön esittävää komponenttia, kuin täyttäjän tarkastelunäkymä sekä muut lomakkeen sisällön listaavat näkymät. Komponentin näkyviä osia renderöidään ehdollisesti kontekstin ja kirjautuneen käyttäjän roolin mukaan niin, että tarvittavat osat ovat hyödynnettävissä oikeissa tilanteissa. Käsittelijä pystyy lataamaan lomakkeesta generoidun sähköisesti allekirjoitetun PDF-tiedoston, sekä ottamaan lomakkeen käsittelyyn. Käsittelijän siirtäessä lomakkeen käsittelyyn, se siirtyy käsittelijän omien käsiteltävien listalle, joka on koottu toiseen taulukkoelementtiin käsittelijän kotinäkymään.

Omien käsiteltävien taulukkoon kootaan vain käsittelijän omat käsittelyprosessit, eivätkä käsittelyssä olevat lomakkeet näy enää järjestelmän muille käsittelijöille. Omien käsittelyprosessien kautta käsittelijä pääsee siirtymään käsittelynäkymään, jossa pystytään suorittamaan käsittelyprosessin olennaisimmat toiminnot. Käsittelynäkymässä käsittelijä pystyy näkemään lomakkeen sisällön sekä lataamaan lomakkeesta generoidun PDF-tiedoston kuten tarkastelunäkymässä, palauttamaan lomakkeen odottamaan käsittelyä, pyytämään lomakkeelle

täydennystä sekä päättämään käsittelyn hyväksymiseen tai hylkäykseen. Jos käsittelijä havaitsee puutteita lomakkeen täytössä, käsittelijä voi lähettää täydennyspyynnön kommentin kera täyttäjälle, jolloin lomake palautetaan täydennettäväksi takaisin täyttäjälle. Käsittelijän kommentit näytetään lomakkeen tarkastelunäkymässä omassa osiossaan ja ne järjestetään uusimmasta vanhimpaan. Täydennettäväksi palautettu lomake pysyy käsittelijän omissa käsittelyprosesseissa ja on heti käsiteltävänä täyttäjän palauttaessa täydennetyn lomakkeen uudelleen käsittelyyn. Lomakkeiden käsittelyprosessin ja päätösten tiloja korostetaan käyttöliittymän listauksissa tilalle sopivalla värillä sekä ikonilla, jonka lisäksi lomakkeen tila on luettavissa lomakkeen tiedoista sekä taulukkoelementtien tilasarakkeesta. Lomakkeen palautuessa täyttäjälle tämä pystyy muokkaamaan käsittelijän täydennyspyynnön kommenttien mukaiset täydennykset lomakkeelle ja lähettämään lomakkeen uudelleen käsittelyyn. Lomake on allekirjoitettava sähköisesti uudelleen ja lomakkeesta generoidaan uusi PDF-tiedosto aina kun se lähetetään käsittelyyn. Täydennetyn lomakkeen palautuessa käsittelijälle se voidaan lähettää uudestaan täydennettäväksi tai käsittelyprosessi voidaan päättää. Käsittelijä voi myös milloin tahansa käsittelyn aikana palauttaa lomakkeen omista käsittelyprosesseista takaisin käsittelyä odottavien lomakkeiden joukkoon, jolloin muut järjestelmän käsittelijät voi ottaa lomakkeen vapaasti omaan käsittelyyn.

Käsittelyprosessi päätetään hyväksymiseen tai hylkäykseen, jolloin käsitelty lomake siirtyy käsiteltyjen lomakkeiden joukkoon ja asianhallintatapaus päättyy. Käsittelijän kotinäkymässä kolmas taulukkoelementti sisältää kaikki käsittelijän aikaisemmin käsittelemät lomakkeet järjestettynä tuoreimmasta vanhimpaan. Täyttäjän puolella käsitelty lomake siirretään myös arkistoon käsiteltyjen lomakkeiden taulukkoon erilleen tallennetuista tai käsittelyssä olevista lomakkeista.

5 Yhteenveto ja johtopäätökset

Opinnäytetyön tuloksena toteutettiin asianhallintajärjestelmän prototyyppi Oikeat Oliot Oy:lle, jonka tarkoituksena oli sähköistää lomakkeiden täyttö- ja käsittelyprosessit. Tuotettu prototyyppi tarjoaa asianhallintajärjestelmän web-pohjaisen sovelluksen, jonka pohjalta järjestelmää voidaan jatkokehittää kohti valmista tuotetta, sekä jonka avulla sovelluksen toiminnallisuutta ja eri ominaisuuksia voidaan esitellä palvelusta kiinnostuneille asiakkaille. Prototyypin kehittämisen tuloksena tunnistettiin uusia jatkokehitystarpeita järjestelmän kehityksen seuraavaan vaiheeseen ja luotiin pohja asianhallintajärjestelmän kehitystyölle käynnistämällä järjestelmän kehitysprosessi. Toimeksiantaja oli tyytyväinen ensimmäisen iteraation tuloksena tuotettuun prototyyppiin ja koki järjestelmän tarjoavan ratkaisuja toivotuille ominaisuuksille, joita yrityksen asiakkaat ovat aiemmin tuoneet esille tarjouspyynnöissään.

5.1 Jatkokehitys

Tulevia kehitysvaiheita varten tunnistettuihin jatkokehitystarpeisiin kuului muun muassa järjestelmän kommunikoinnin sekä järjestelmän lomakerakenteen kattavuuden kehittäminen. Toteutettu prototyyppi kommunikoi käyttäjän kanssa tämän suorittamien toimintojen yhteydessä käyttöliittymärakenteen sekä popup-ilmoitusten avulla yleisellä tasolla. Tulevaisuudessa järjestelmän kommunikointia voitaisiin parantaa kehittämällä esimerkiksi lomakkeen täytön tai lomakepohjan määrittelyn yhteydessä osio-/kenttäkohtaisesti käyttäjää ohjastava aputoiminto käytön tueksi. Myös järjestelmän ilmoituksia voitaisiin tarkentaa yksityiskohtaisemmiksi kuvaamaan kattavammin suoritettua toimintoa tai poikkeustilanteesta syntynyttä varoitusta. Esimerkiksi järjestelmän validoidessa täytetyn lomakkeen kenttiä, järjestelmä voisi koostaa kenttäkohtaisesti yksilöidyt täyttövirheet käyttäjän näkyville yleisen ”Virheellinen täyttö, tarkista lomakkeen kentät.”-tyyppisen varoituksen sijaan. Prototyypin laajuudella toteutettu lomakerakenne sisälsi lomakerakenteen mallintamisen ja generoinnin perustarpeet täyttävät yleiset ominaisuudet, mutta todelliset liike-elämän käyttöskenaariot vaativat mitä todennäköisemmin laajempaa rakenteellista muokkausmahdollisuutta, kuten kenttien kehittyneempää ryhmittelyä lopullisella generoidulla PDF-lomakkeella sekä lomakkeen täyttöä tukevan ohjeistuksen tai vakiotekstin sisällyttämistä.

Myös lomakkeen käsittelyprosessin käsittelijän toimintoihin voitaisiin toteuttaa mahdollisuus lisätä käsittelijän omia huomioita, jotka olisivat näkyvissä vain käsittelijöille ja siirtyisivät lähetetyn lomakkeen mukana, mikäli lomake palautetaan odottamaan käsittelyä ja se siirtyy toisen käsittelijän käsiteltäväksi. Tallennetut huomiot auttaisivat useita lomakkeita samanaikaisesti käsittelevää käsittelijää yksittäisiin asianhallintatapauksiin liittyvien asioiden muistamisessa sekä mahdollistaisivat keskeneräisten käsittelyprosessien siirtämisen käsittelijältä toiselle niin, että uusi käsittelijä saa tietoonsa heti aiemmin tehdyt käsittelyn kannalta olennaiset huomiot.

Web-sovelluksen käyttöliittymä toteutettiin prototyypin kehittämisen aikana toiminnallisuus edellä ottamatta kantaa pohjarakennetta lukuun ottamatta sen tarkemmin lopulliseen käyttöliittymärakenteeseen tai sovelluksen brändäykseen. Valmista tuotetta varten sovelluksen käyttöliittymää tullaan jalostamaan eteenpäin myöhempien kehitysvaiheiden aikana käyttökokemuksen ja käytettävyyden parantamiseksi.

Lähteet

Painetut

Arnowitz, J, Arent, M, & Berger, N. 2007. Effective Prototyping for Software Makers. Elsevier Science & Technology, Burlington.

Crookshanks, E. 2014. Practical Software Development Techniques : Tools and Techniques for Building Enterprise Software. Apress L. P., Berkeley, CA.

Fenton, S. 2017. Pro TypeScript: Application-Scale JavaScript Development. Apress L. P., Berkeley, CA.

Flanagan, D. 2006. JavaScript – The Definitive Guide, 5th ed. O'Reilly, Sebastopol, CA.

Holcombe, M. 2008. Running an Agile Software Development Project. John Wiley & Sons, Incorporated, Hoboken.

Karanpuria, R, & Roy, AS. 2018. Kotlin Programming Cookbook: Explore More Than 100 Recipes That Show How to Build Robust Mobile and Web Applications with Kotlin, Spring Boot, and Android. Packt Publishing Limited, Birmingham.

Kelly, A. 2008. Changing Software Development : Learning to Become Agile. John Wiley & Sons, Incorporated, New York.

Knight, W. 2018. UX for Developers : How to Integrate User-Centered Design Principles into Your Day-To-Day Development Work. Apress L. P., Berkeley, CA.

Krochmalski, J. 2016. Developing with Docker. Packt Publishing, Limited, Birmingham.

Mitra, M. 2015. Mastering Gradle. Packt Publishing, Limited, Birmingham.

Oussalah, MC (ed.). 2014. Software Architecture 1. John Wiley & Sons, Incorporated, Somerset.

Salahaldin, JAVAV. 2015. Learning PostgreSQL. Packt Publishing Limited, Birmingham.

Stephens, R. 2015. Beginning Software Engineering. John Wiley & Sons, Incorporated, Somerset.

Varanasi, B. 2015. Introducing Gradle. Apress L. P., Berkeley, CA.

Zhu, H. 2005. Software Design Methodology : From Principles to Architectural Styles. Elsevier Science & Technology, Oxford.

Sähköiset

The Agile Alliance. 2001. Agile Manifesto. Viitattu 11.10.2021.
https://agilemanifesto.org/iso/fi/manifesto.html

Asiakastieto. 2021. Oikeat Oliot Oy. Viitattu 24.8.2021.
https://www.asiakastieto.fi/yritykset/fi/oikeat-oliot-oy/10307495/yleiskuva

DVV. 2021. Fujitsu mPollux DigiSign Client / Asennus- ja käyttöohje Linux. Viitattu 5.10.2021.
https://dvv.fi/documents/16079645/17581618/Fujitsu+mPollux+DigiSign+asennus-+ja+k%C3%A4ytt%C3%B6ohje+-+Linux.PDF/877a7691-4c63-e92e-35c7-6d0e5e177549/Fujitsu+mPollux+DigiSign+asennus-+ja+k%C3%A4ytt%C3%B6ohje+-+Linux.PDF?version=1.0&t=1595587791336

DVV. 2017. HTML5 and Digital Signatures. Viitattu 5.10.2021.
https://dvv.fi/documents/2634109/2858578/SCS-signatures_v1.1.PDF/76a3a7a2-f1ff-4dfb-8aa4-337c9413695f/SCS-signatures_v1.1.PDF

Gradle. 2021. Gradle User Manual. Viitattu 10.10.2021.
https://docs.gradle.org/current/userguide/userguide.html

Gradle. 2021. What is Gradle?. Viitattu 10.10.2021.
https://docs.gradle.org/current/userguide/what_is_gradle.html#what_is_gradle

JetBrains. 2021. A Ktor application. Viitattu 24.10.2021.
https://ktor.io/docs/a-ktor-application.html

JetBrains. 2021. FAQ. Viitattu 21.8.2021.
https://kotlinlang.org/docs/faq.html

JetBrains. 2021. Get started with Kotlin. Viitattu 29.7.2021.
https://kotlinlang.org/docs/getting-started.html#create-your-powerful-application-with-kotlin

JetBrains. 2021. Our history. Viitattu 29.7.2021.
https://www.jetbrains.com/company/

Kaliski, B. 1998. PKCS #7: Cryptographic Message Syntax. Viitattu 6.10.2021.
https://datatracker.ietf.org/doc/html/rfc2315

Oracle. 2021. Class KeyStore. Viitattu 6.10.2021.
https://docs.oracle.com/javase/7/docs/api/java/security/KeyStore.html

Quasar Framework. 2021. Preparation for Electron. Viitattu 23.8.2021.
https://quasar.dev/quasar-cli/developing-electron-apps/preparation

Quasar Framework. 2021. Why Quasar?. Viitattu 23.8.2021.
https://quasar.dev/introduction-to-quasar

Signature Creation Service. 2021. HTML5 and Digital Signatures: Signature Creation Service. Viitattu 5.10.2021.
http://developer.fineid.fi/scs/

Typescript. 2021. TypeScript for the New Programmer. Viitattu 12.7.2021.
https://www.typescriptlang.org/docs/handbook/typescript-from-scratch.html

Valtiovarainministeriö. 2021. Saavutettavuus. Viitattu 31.8.2021.
https://vm.fi/saavutettavuusdirektiivi

Vue.js. 2021. What is Vue.js?. Viitattu 12.7.2021
https://vuejs.org/v2/guide/

W3C. 2021. Facts about W3C. Viitattu 31.8.2021.
https://www.w3.org/Consortium/facts

W3C. 2021. Verkkosisällön saavutettavuusohjeet (WCAG) 2.0. Viitattu 31.8.2021.
https://www.w3.org/Translations/WCAG20-fi/

Winer, D. Android's commitment to Kotlin. Viitattu 21.8.2021.
https://android-developers.googleblog.com/2019/12/androids-commitment-to-kotlin.html

You, E. 2013. 0.6.0: VueJS. Viitattu 12.7.2021.
https://github.com/vuejs/vue/releases/tag/0.6.0

Julkaisemattomat

Oikeat Oliot Oy. 2021. Kanto dokumentaatio. 6.10.2021. Oikeat Oliot Oy. Helsinki.

Kuviot

Kuvio 1: Prosessikaavio - Lomakkeen täyttö ja tallennus 16
Kuvio 2: Classic 3-tier architecture (Crookshanks 2014, 117.) 18
Kuvio 3: Täyttäjän kotinäkymä 22
Kuvio 4: Käsittelijän kotinäkymä 22
Kuvio 5: Pääkäyttäjän kotinäkymä 23
Kuvio 6: Lomakepohjaeditori muokattaessa tallennettua lomakepohjaa..... 24
Kuvio 7: Uuden lomakkeen täyttönäkymä..... 25
Kuvio 8: Esimerkki generoidusta PDF-tiedostosta – Esimerkkilomake 29