LAUREA
AMMATTIKORKEAKOULU
Yhdessä enemmän

Web-sovelluksen kehittäminen Amazon-pilvipalvelua hyödyntäen

Nieminen, Oskari

2018 Laurea

Laurea-ammattikorkeakoulu

Web-sovelluksen kehittäminen Amazon-pilvipalvelua hyödyntäen

Nieminen, Oskari
Liiketalous
Opinnäytetyö
Toukokuu, 2018

Laurea-ammattikorkeakoulu **Tiivistelmä**
Otaniemi
Liiketalous (AMK)

Nieminen, Oskari

Web-sovelluksen kehittäminen Amazon-pilvipalvelua hyödyntäen

2018 Sivumäärä 52

Tämän opinnäytetyön tarkoituksena oli selvittää toteutetun sovelluksen kehitysmenetelmiä ja työvaiheita. Opinnäytetyön tavoitteena on esittää web-sovelluskehityksessä käytettäviä menetelmiä, työvälineitä ja teknologioita sekä web-sovelluksille tyypillisiä ominaisuuksia ja arkkitehtuurimallia. Toimeksiantaja oli startup-yritys, jolle toteutettu sovellus toimii ensimmäisenä versiona yrityksen tuotteesta ja mahdollisena rahoituksen haun perusteena.

Opinnäytetyössä kuvataan sovelluksen suunnittelua ja toteutusta, tekijän kehitysympäristöä, sovelluksen toteuttamiseen vaadittua osaamista sekä sovelluksen arkkitehtuurista rakennetta ja sen osia. Opinnäytetyössä myös esitellään kirjallisuudesta ohjelmistokehityksen yleisiä toimintatapoja ja vaiheita.

Sovellusta kehitettiin käyttäen ketteriä ohjelmistokehitysmenetelmiä, joten opinnäytetyö perehtyy myös niiden periaatteisiin. Lisäksi selvitetään sovelluksen kehittämisessä käytettyjä tärkeimpiä ohjelmointikieliä, kirjastoja ja viitekehyksiä kuten JavaScript, ReactJS ja ExpressJS. Sovellus kehitettiin Amazonin pilvipalveluita hyödyntäen, mistä syystä opinnäytetyö esittelee pilvipalveluratkaisujen käyttämistä verkkosovellusten toteuttamisessa sekä sovelluksen toteuttamisessa käytettyjä Amazonin pilvipalvelujen tuotteita.

Lopputuloksena syntyi toimeksiantajan toiveiden mukainen, pilvipalveluun rakennettu, selaimessa toimiva web-sovellus. Sovelluksen taustajärjestelmä ja tietokanta saatiin onnistuneesti toimimaan Amazonin pilvipalveluun ja sovelluksen käyttöliittymästä tuli visuaalisesti näyttävä ja eri kokoisille näytöille skaalautuva. Valmis sovellus oli suunnitelmien mukainen ja se sisälsi kaikki toimeksiantajan määrittelemät toiminnallisuudet.

Asiasanat: sovelluskehitys, ohjelmistokehitys, JavaScript, pilvipalvelut, AWS

Laurea University of Applied Sciences
Otaniemi
Business management

Nieminen, Oskari

Abstract

Web application development with Amazon cloud computing

2018 Pages 52

The aim of this Bachelor's thesis was to clarify the methods used to create the application and stages in which it was carried out. The goal of the thesis was to show methods, tools and technologies applicable to web development as well as typical properties and architectural model of web applications. The client was a startup, to which the application functions as a first version prototype of their product and is the possible basis for their future fundraising.

The thesis represents design and implementation of the application, development environment, qualifications required to implement the application as well as the application's architectural structure and its sections. The thesis also introduces common procedures and phases of software development based on literature.

The application was developed using agile software development, therefore, the thesis also studies its principles. The thesis examines essential programming languages, libraries and frameworks utilized in the application development like JavaScript, ReactJS and ExpressJS. The application was developed using Amazon cloud computing services and the thesis deals with the use of cloud service solutions in web application development and Amazon Web Services' products used by the application.

The final outcome was a web application with browser interface and cloud service backend which was consistent with the client's wishes. The application's backend was successfully installed in Amazon cloud service and the interface of the application became visually impressive and scalable on different screen sizes. The ready application met its requirements and it contained all the functionalities required by the client.

Keywords: application development, software development, JavaScript, cloud computing, AWS

Sisällys

1 Johdanto

Opinnäytetyössä esitellään demoversion kehittämistä sovelluksesta, jonka tarkoituksena oli toimia taloyhtiöiden asukkaiden, osakkaiden, hallituksen jäsenten ja isännöitsijöiden yhteistyöalustana. Toteutettavan sovelluksen oli tarkoitus kehittää ja ylläpitää tietoa taloyhtiön sidosryhmistä ja helpottaa heidän välistä kommunikaatiotaan sekä kerätä ja esittää analytiikkaa asuntoihin liittyvistä tiedoista. Ohjelmistoon kehitettiin web-käyttöliittymä, jonka avulla taloyhtiöiden eri sidosryhmät voivat www-selaimessa käyttää sovellusta. Sovelluksen käyttäjillä on erilaisia rooleja sovelluksessa, jotka vaikuttavat käyttäjän toimintavaltuuksiin yksittäisen taloyhtiö-yhteisön sisällä. Käyttäjät voivat omien rooliensa suomin toimintavaltuuksin tarkastella, lisätä tai muokata tietoa sovelluksessa.

Opinnäytetyö tehtiin toimeksiantona startup-yritykselle. Yrityksen tavoitteena on hakea rahoitusta laajemman ohjelmiston kehittämiseen demoversion avulla. Yrityksen visiona on tuottaa palvelu, joka siirtää taloyhtiön viestinnän ja hallinnoinnin digitaaliseen muotoon tarjoamalla ohjelmiston taloyhtiön asukkaille, isännöitsijöille ja huoltoyhtiöille. Sovellus ja siihen liittyvä aineisto on kehitetty opiskelijatyönä toimeksiantajien toiveiden mukaisesti.

Opinnäytetyön tavoitteena oli saada sovelluksen kehittämisen ensimmäisen vaiheen ominaisuuksista toimiva kokonaisuus, johon kuuluu web-käyttöliittymä, järjestelmään kirjautuminen ja käyttäjän todentaminen, käyttäjien ja heidän toimintavaltuuksiensa hallinnointi, taloyhtiö-yhteisöjen hallinnointi sekä tietomallin ja tietokannan suunnittelu ja toteuttaminen. Kokonaisuus oli tarkoitus saada toimimaan Amazon Web Services -pilvipalveluun perustettaville virtuaalipalvelimille.

Opinnäytetyössä oppimistavoitteena oli sovelluskehitys valittuja teknologioita hyödyntäen sekä näiden teknologioiden oppiminen. Erityisesti Amazonin pilvipalveluiden käyttöönotto sekä sovelluksen toteutukseen valittavien ohjelmointikielten ja -viitekehysten käyttäminen. Lisäksi ohjelmistoprojektin suunnittelun hallitseminen on yksi oppimistavoitteista.

2 Työn lähtökohdat

Lähtökohtana oli toteuttaa toimeksiantajien toiveiden pohjalta sovellus, jonka käyttäjiksi on tarkoitettu taloyhtiöt ja sen asukkaat, osakkaat, hallituksen jäsenet sekä isännöitsijätoimistot ja huoltoyritykset. Sovelluksen tavoitteena oli helpottaa taloyhtiön sisällä toimivien henkilöiden sisäistä viestintää, kerätä ja analysoida taloyhtiö- ja asuntokohtaisia tietoja sekä selkeyttää isännöitsijä- ja huoltopalveluntarjoajien ja taloyhtiöiden välisiä yhteyksiä.

Ohjelmistoprojektin alussa toimeksiantajilla oli selkeä kuva toteutettavan sovelluksen kokonaisuudesta. Sovellukseen toteuttavia ominaisuuksia oli projektin alussa paljon ja niiden tärkeys ja tarpeellisuus ei ollut toimeksiantajille täysin selvää. Projektin edetessä sovellukseen suunniteltiin lisää ominaisuuksia ja osasta suunnitelluista ominaisuuksista luovuttiin kokonaan. Sovelluksen suunnittelua, kehittämistä sekä ominaisuuksien valitsemista ja rajaamista seurattiin viikoittaisissa palavereissa yhteistyökumppaneiden kanssa sekä Slack-pikaviestintäsovelluksen viestiryhmissä.

Oli todennäköistä, että sovellukseen kohdistuvat vaatimukset tarkentuvat projektin edetessä. Tästä syystä sovellus toteutukseen valittiin ketterän ohjelmistokehityksen mukainen iteratiivinen kehitysmenetelmä.

2.1 Aihealueen rajaukset

Opinnäytetyön aiheeksi rajattiin ohjelmistoprojektin suunnittelu ja toteutus tuotettavan sovelluksen ensimmäisen vaiheen osalta eli opinnäytetyö käsittelee läpileikkausta kyseisen sovelluksen kehittämisestä. Sovelluksen ensimmäisen vaiheen suunnittelun ja toteutuksen lisäksi opinnäytetyö selvittää toteutuksessa käytettävien teknologioiden valitsemisen.

Sovelluksen kehittämisen ensimmäinen vaihe sisälsi muun muassa tärkeimpien ominaisuuksien suunnittelun ja toteutuksen. Sovelluksen tärkeimpiä ominaisuuksia olivat taloyhtiö-yhteisöjen hallinnointi sekä yhteisön sisällä olevien käyttäjien ja erilaisten käyttäjä-roolien oikeuksien hallinnointi.

2.2 Keskeiset käsitteet

UML (Unified Modeling Language) – kuvaustekniikka, joka määrittelee joukon kaaviotyyppejä käytettäväksi ohjelmiston kuvaamiseen.

HTML (HyperText Markup Language) – Muu muassa internetsivuilla käytettävä kuvauskieli, jolla kuvataan hypertekstiä.

CSS (Cascading Style Sheet) – tyylitiedostokieli, jolla kuvataan, miten kuvauskielellä kirjoitetun dokumentin sisältö tulee esittää.

Javascript – Korkeatasoinen, dynaaminen, tulkattava oliopohjainen komentosarjakieli, jolla voidaan muun muassa lisätä toiminnallisuutta web-sivuille. Lisäksi JavaScriptiä voidaan käyttää taustajärjestelmissä esimerkiksi Node.js alustassa.

Node.js – avoimeen lähdekoodiin perustuva, järjestelmäriippumaton JavaScript ajonaikainen ympäristö, joka mahdollistaa JavaScript-komentosarjakielen käyttämisen taustajärjestelmässä.

WWW (World Wide Web) – Hypertekstijärjestelmä, jonka avulla voidaan selaimella avata web-dokumentteja internetin välityksellä.

HTTP (HyperText Tranfer Protocol) – Hypertekstin siirtoprotokolla, joka määrittelee joukon sääntöjä tiedonsiirtoon asiakasohjelman ja palvelimen välillä.

URI (Uniform Resource Identifier) – Tietoresurssin tunniste internetissä.

AJAX (Asynchronous Javascript and XML) – Yhdistelmä teknologioita, jotka mahdollistavat selaimen ja palvelimen välisen tiedonsiirron ilman, että verkkosivua tarvitsee ladata kokonaan uudestaan.

JSON (Javascript Object Notation) – JavaScript-komentosarjakielen syntaksin mukainen, mutta siitä riippumaton tiedostomuoto tiedon esittämiseen, siirtämiseen ja varastointiin.

mLab – Pilvessä toimiva tietokantapalvelu MongoDB:lle.

MongoDB – Järjestelmäriippumaton dokumentti-orientoitunut tietokanta ohjelma.

ReactJS – Javascript-kirjasto interaktiivisten graafisten käyttöliittymien rakentamiseen.

GIT – Versionhallintajärjestelmä, joka mahdollistaa ohjelmistoprojektin tuotosten hallintaa.

GitLab – Verkkoselainpohjainen Git-versionhallintamanageri.

Taloyhtiö-yhteisö – Asiakas-taloyhtiön suljettu ryhmä sovelluksessa, joka sisältää tietoa taloyhtiöstä. Tänne pääsevät kirjautumaan taloyhtiön asukkaat, osakkaat ja isännöitsijä.

Rooli – Käyttäjillä on sovelluksessa tietty rooli, joka määrittää heidän toimintavaltuutensa sovelluksen taloyhtiö-yhteisössä. Rooleja ovat käyttäjä, ylläpitäjä, isännöitsijä ja osakas.

Pääkäyttäjä – Koko järjestelmän ylläpitäjä, jolla on oikeudet uusien taloyhtiö-yhteisöjen luomiseen ja hallinnointiin. Pääkäyttäjä on lisäksi jokaisen taloyhtiö-yhteisön ylläpitäjä.

Ylläpitäjä – Taloyhtiö-yhteisön sisällä vaikuttava rooli, jonka omaava käyttäjä pystyy esimerkiksi luomaan uusia käyttäjiä kyseiseen taloyhtiö-yhteisöön ja hallinnoimaan muita yhteisön tietoja.

Käyttäjä – Taloyhtiö-yhteisön sisällä vaikuttava rooli, joka pystyy toimimaan sovelluksessa taloyhtiö-yhteisön sisällä. Usein taloyhtiön asukas.

Isännöitsijä – Taloyhtiö-yhteisön sisällä vaikuttava rooli, jolla on omia toiminnallisuuksia taloyhtiö-yhteisön sisällä. Isännöitsijä voi kirjautua omaan näkymään, josta on mahdollista hallinnoida taloyhtiö-yhteisöjä, joihin isännöitsijä kuuluu.

Osakas - Taloyhtiö-yhteisön sisällä vaikuttava rooli, joka pystyy toimimaan sovelluksessa taloyhtiön-yhteisön sisällä. On osakkaana taloyhtiössä.

3 Ohjelmistokehitys

Ohjelmistokehitys kattaa ohjelmiston toteuttamisen ideasta valmiiseen tuotteeseen. Kehitysmenetelmiä on monia ja kaikilla on omat etunsa ja heikkoutensa, mutta kaikki käyvät läpi samat vaiheet. Kehitysmenetelmästä ja -tyylistä riippumatta ohjelmisto käy läpi ainakin vaiheet: määrittely, suunnittelu ja toteutus. Määrittelyssä selvitetään, millainen järjestelmä täyttää ongelman vaatimukset. Suunnittelu selvittää, miten järjestelmä toteutetaan. Toteutus kattaa järjestelmän ohjelmoinnin ja muun toteutuksen. (Haikala, 2000) Vaiheiden suoritusjärjestys, toteuttajat ja suoritustavat vaihtelevat ohjelmistoprojektin ja kehitysmenetelmän mukaan. (Luukkainen, 2017) Opinnäytetyö ei kuvaile tarkasti mitään tiettyä ohjelmistokehitystyyliä vaan esittelee tietyn ohjelmiston kehittämisessä käytettyjä kehittämistekniikoita.

3.1 Ohjelmiston määrittely

Ohjelmistokehityksessä ohjelmiston määrittelyyn liittyvät tehtävät ovat projektin tarpeellisuuden ja toteuttamiskelpoisuuden selvittäminen, tavoitteiden ja vaatimusten asettaminen sekä ratkaisumallin laatiminen. Määrittely jaetaan usein kahteen osaan: asiakasvaatimusten kartoittamiseen ja toteutettavan ohjelmiston vaatimusten määrittelyyn. Asiakasvaatimusten kartoittamista kutsutaan usein esitutkimukseksi. Vaatimuksia kartoitettaessa asetetaan usein järjestelmälle yleisiä tavoitteita, kuten esimerkiksi *isännöitsijän ja asukkaan välisen viestinnän parantaminen taloyhtiöissä*. Varsinaisessa määrittelyssä spesifioidaan nämä vaatimukset toteuttava järjestelmä. (Haikala, 2000)

Vaatimusmäärittelyllä tarkoitetaan toteutettavan ohjelmiston vaatimusten selvittämistä, dokumentointia ja hallinnointia (Luukkainen, 2017). Vaatimusmäärittelyssä on tarkoitus yhdessä

asiakkaan ja ohjelmistontuottajan kanssa selvittää, mitä ominaisuuksia toteutettava ohjelmisto tulee sisältämään sekä saavuttaa ja ylläpitää yhteinen ymmärrys toteutettavaan ohjelmistoon kohdistuvista vaatimuksista. Lisäksi tavoitteena on antaa kehittäjälle tarpeeksi selkeä kuva ohjelmiston edellytyksistä ja tarjota perusta projektin toteutuksen hallintaan. (Kruchten 2003, 157–158.)

Vaatimusmäärittelyä voi toteuttaa kuvitelluilla käyttöskenaarioilla tai käyttöliittymäluonnoksilla, jotka perustuvat haastatteluihin asiakkaan kanssa. Vaatimusmäärittelyä voi sitten asiakkaan toiveiden mukaisesti ryhtyä tarkentamaan, jos tarpeellista. Pääasia on, että asiakkaan näkemykset ohjelmistolle asetettavista vaatimuksista saadaan dokumentoitua, jotta kehittäjä voi niiden pohjalta ryhtyä suunnittelemaan ohjelmistoa. (Klimczak 2013, 73.)

3.2 Ohjelmiston suunnittelu

Ohjelmiston suunnittelu jaetaan usein kahteen osaan: arkkitehtuurisuunnitteluun ja moduulisuunnitteluun. Arkkitehtuurisuunnittelussa määritellään järjestelmän yleinen rakenne ja sen osat. Moduulisuunnittelussa suunnitellaan järjestelmän osien sisäinen rakenne. (Haikala, 2000.)

Ohjelmistoa suunniteltaessa kehitetään suunnitelma, joka tiivistää, mitä tullaan tekemään ohjelmiston toteuttamiseksi. Se selvittää ohjelmistoon liittyvää kehittämisprosessia, riskejä, resursseja, työvälineitä, arvioita, vaatimusten hallinnointia ja toteuttamista. Suunnitelman ei tarvitse sisältää kaikkia yksityiskohtia projektin elinkaaresta, mutta kaikkien projektin jäsenten täytyy ymmärtää se ja hyväksyä sen sisältämät asiat. (Dooley 2011, 3.)

Ohjelmiston suunnittelu jatkuu koko projektin eliniän ajan ja suunnitelmaa muokataan projektin tarpeiden mukaan. Ohjelmistoprojekti harvoin tulee etenemään täysin ensimmäisen kirjoitetun suunnitelman mukaisesti vaan kehittäjien tulee mukautua muuttuviin tilanteisiin ja muokata suunnitelmaa kesken projektin. (Dooley 2011, 28.)

3.3 Ketterät menetelmät

Toteutetun ohjelmiston kehittämiseen ei katsottu tarpeelliseksi valita mitään tiettyä projektinhallinnan menetelmää, sillä ohjelmiston toteuttamiseen ei toimeksiantajan ja yhden ohjelmistokehittäjän lisäksi kuulunut muita henkilöitä, joten ohjelmisto toteutettiin soveltaen yleisiä ketterien menetelmien käytäntöjä. Ketterässä ohjelmistokehittämisessä järjestelmä suunnitellaan ja toteutetaan pienissä osissa eli sprinteissä. Ketterissä menetelmissä ideana on, että valitaan vain osa vaadittavista ominaisuuksista, jotka suunnitellaan ja toteutetaan kokonaan ennen kuin siirrytään seuraaviin ominaisuuksiin. Näin ohjelmistokehitys pysyy toimivana jatkuvasti muuttuvista vaatimuksista huolimatta. (Luukkainen 2018.)

Ketterillä menetelmillä tarkoitetaan ohjelmistokehityksessä järjestelmän kehittämistä iteratiivisesti eli pienissä osissa. Sen sijaan, että alussa määriteltäisiin ja suunniteltaisiin järjestelmän toiminnallisuus kokonaisuudessaan, suunnitellaan ja toteutetaan kerrallaan vain pieni osa järjestelmän toiminnallisuutta. Näin saadaan jo ensimmäisen iteraation jälkeen valmiiksi ohjelmisto, jossa on mukana osa koko järjestelmän toiminnallisuutta. Seuraavassa iteraatiossa toteutetaan taas hieman uutta toiminnallisuutta asiakkaan toiveiden mukaisesti. (Luukkainen & Laine 2012, 4-5.)

Ketterää kehittämistä ei käytetä pelkästään ohjelmistotuotannossa, mutta siihen se soveltuu erityisen hyvin ja on yleisesti käytössä. Ketterään kehittämiseen perustuvia projektinhallinnan menetelmiä on useita, kuten esimerkiksi Scrum ja Extreme Programming. Niiden periaatteina ovat muun muassa toimittaa asiakkaalle toimivia versiota projektin aikaisessa vaiheessa ja säännöllisesti koko kehitystyön ajan sekä ottaa vastaan muuttuvat vaatimukset riippumatta kehityksen vaiheesta. Ketterissä menetelmissä liiketoiminnan edustajat ja toteuttajat työskentelevät yhdessä koko projektin ajan ja projektitiimi tarkastelee säännöllisesti omaa tehokkuuttaan mukauttaen toimintaansa sen mukaisesti. (Koski.)

Ketterässä kehityksessä ei ole tapana tehdä koko järjestelmän kattavaa vaatimusmäärittelyä etukäteen, vaan vaatimuksia tarkennetaan projektin kuluessa. Ketterässä kehityksessä käytetään usein käyttäjätarinoita, joilla selvitetään yksittäisiä toiminnallisia vaatimuksia. Käyttäjätarinoita luodaan mahdollisimman suuri määrä, ottamatta kantaa siihen, mitä niistä lopulta toteutetaan. Tavoitteena käyttäjätarinoissa on selvittää kuka pystyy tekemään mitä ja miksi. (Sininen meteoriitti, 2013.)

Toinen tapa ohjelman toiminnallisten vaatimusten ilmaisemiseen on käyttötapausmalli. Käyttötapausmallin tarkoituksena on selvittää yksittäisen toiminnallisuuden kulku sekä toiminnallisuuden mahdolliset poikkeukset ja vaihtoehtoiset lopputulokset. Lisäksi selvitetään, kuka on toiminnallisuuden käyttäjä, mitä toiminnallisuudella tavoitellaan ja mikä laukaisee tarpeen käyttää toiminnallisuutta. Käyttötapauksessa on esiehto eli tila, joka vallitsee ennen toiminnallisuutta sekä jälkiehto eli tila, joka vallitsee toiminnallisuuden käyttämisen jälkeen. (Elliott 2014, 87.)

Ketteriä menetelmiä käyttäen saatiin toimeksiantajilta välitöntä palautetta toteutetusta toiminnallisuudesta, jolloin sovelluksen vaatimuksia voitiin tarkentaa ja muuttaa projektin edetessä. Ominaisuuksien määrittely, suunnittelu, toteutus ja testaus eivät välttämättä edenneet peräkkäin vaan kaikkia kehittämisen osia voitiin tehdä samanaikaisesti.

4 Suunnittelun vaiheet

Sovelluksen suunnittelun tarkoitus oli laatia dokumentaatiota, jonka pohjalta sovelluksen toteutus tehtäisiin. Suunnitteluvaiheessa toimeksiantajilta pyrittiin saamaan selvitystä toteutettavista ominaisuuksista ja niiden tarpeellisuudesta. Ennen kuin voitiin aloittaa ohjelmiston toteuttaminen, tehtiin siitä vaatimusmäärittely, käyttötapaushahmotelmat, käyttötapausmallit, sekvenssikaaviot sekä tietomallit. Nämä dokumentaatiot toimivat apuna sovelluksen suunnittelussa ja toteutuksessa.

4.1 Vaatimusmäärittely

Sovelluksen suunnittelu aloitettiin selvityksellä ohjelmistolta vaadittavista ominaisuuksista. Vaatimusmäärittelyssä (liite 1) listattiin ensimmäisenä toiminnallisuuksia ja ohjelmiston sanastoa, jotka perustuvat toimeksiantajien haastatteluille. Toiminnallisuuksista saatiin myöhemmin perusta toteutettaville ominaisuuksille ja sanasto auttoi hahmottamaan sovelluksen osien yhteyksiä toisiinsa ja järjestelmän kokonaisuutta. Vaatimusmäärittelyistä tehtiin useita versioita, joita kehitettiin toimeksiantajien toiveiden mukaisiksi. Kun ohjelmiston kannalta oleellisimmat ominaisuudet ja toiminnallisuudet saatiin listattua, valittiin yhdessä toimeksiantajien kanssa ensimmäiseen iteraatioon eli ensimmäisenä toteutukseen vietävät toiminnallisuudet (taulukko 1). Nämä toiminnallisuudet oli tarkoitus suunnitella, toteuttaa ja testata kokonaan ennen seuraaviin toiminnallisuuksiin siirtymistä.

Sovelluksen vaatimusmäärittelyn mukaan valittiin ympäristöksi Amazonin pilvipohjaiset verkkopalvelut, jonka tarjoamille virtuaalipalvelimille asennettiin sovelluksen kehittämiseen tarvittavat ohjelmistot.

Taloyhtiöllä on oma suljettu yhteisö järjestelmässä.
Pääkäyttäjä on kaikkien taloyhtiöryhmien ylläpitäjänä.
Pääkäyttäjä voi luoda uuden ryhmän järjestelmään.
Pääkäyttäjä voi poistaa ryhmän järjestelmästä.
Uuden ryhmän luomisen yhteydessä järjestelmä lisää pääkäyttäjille oikeudet ryhmään.
Ylläpitäjät ja käyttäjät luodaan järjestelmään ylläpitäjän toimesta (vain ryhmään, jonka ylläpitäjänä ylläpitäjä toimii).
Käyttäjille luodaan yksilölliset käyttäjätunnukset ja salasanat järjestelmään.
Järjestelmä luo käyttäjälle salasanan käyttäjän luonnin yhteydessä.
Ryhmää poistettaessa järjestelmä poistaa jokaiselta käyttäjältä oikeudet ryhmään.
Käyttäjällä tulee olemaan tietty rooli ryhmän sisällä (käyttäjä, ylläpitäjä, isännöitsijä, osakas).
Ylläpitäjät voivat toimia yhteisön käyttäjinä.
Ylläpitäjät voivat hallinnoida yhteisön tietoja.
Ylläpitäjät voivat lisätä ja poistaa käyttäjiä omassa taloyhtiö-yhteisössään.
Ylläpitäjät voivat hallinnoida yhteisöön kuuluvien käyttäjien rooleja.
Käyttäjät voivat kirjautua järjestelmään.
Käyttäjät voivat valita mihin kuulumistaan taloyhtiö-yhteisöistään haluaa kirjautua.
Käyttäjät voivat kuulua useampaan yhteisöön samoilla tunnuksilla.

Taulukko 1: Ensimmäisen vaiheen vaatimusmäärittely.

4.2 Käyttötapaushahmotelmat

Ohjelmiston kehittämisessä käytettiin käyttäjätarinoiden sijasta käyttötapauskuvauksia, jotka ensin hahmoteltiin ja toimeksiantajan hyväksynnän jälkeen mallinnettiin. Käyttötapauskuvaukset valittiin käyttäjätarinoiden sijasta ohjelmiston kehittämiseen, koska niiden määrä

pystyttiin pitämään vaivattomammin pienenä ja näin helpommin käsiteltävinä yhdelle kehittäjälle.

Vaatimusmäärittelystä ensimmäiseen iteraatioon valituista ominaisuuksista listattiin käyttötapaushahmotelmat (liite 2). Käyttötapaushahmotelmat käännettiin vaatimusmäärittelyn listaamista ominaisuuksista mahdollisiksi järjestelmän toiminnallisuuksiksi. Järjestelmän toiminnallisuudeksi kääntämisellä tarkoitetaan, että toiminnolla kuvataan jokin yksittäinen prosessi, jonka käyttäjä suorittaa ohjelmistolla. Esimerkiksi vaatimusmäärittelyssä mainittu ominaisuus ”*Pääkäyttäjä voi luoda uuden ryhmän järjestelmään*” lisättiin käyttötapaushahmotelmaksi ”*Pääkäyttäjä lisää taloyhtiö-yhteisön järjestelmään*”.

4.3 Käyttötapausmallit

Käyttötapaushahmotelmat laajennettiin jokainen omaksi käyttötapausmallikseen (liite 3). Käyttötapausmalleja käytettiin, jotta voitiin yksityiskohtaisesti käydä läpi ohjelmiston toiminnallisuuksia toimeksiantajien kanssa, joilla ei ollut aikaisempaa kokemusta ohjelmistoprojekteista tai ohjelmiston teknisestä toteuttamisesta.

Käyttötapausmalli toimi apuna toimeksiantajille selkeyttämään ominaisuuksien hahmottamista ja suunnittelua (taulukko 2). Sovelluksen toteuttajalle käyttötapausmallit toimivat apuna sovelluksen toteutuksessa käytettävien työvälineiden valitsemisessa sekä selkeyttämään kehitettävän järjestelmän kokonaiskuvaa ja ominaisuuksien suhdetta toisiinsa. Lisäksi käyttötapausmallit ovat myöhemmin käännettävissä järjestelmätestauksessa käytettäviksi testitapauksiksi.

Käyttäjät	Pääkäyttäjä
Tavoite	Luoda uusi suljettu yhteisö taloyhtiölle
Laukaisija	Uusi asiakkuus
Esiehto	Taloyhtiöllä ei ole ennestään ryhmää järjestelmässä
Jälkiehto	Taloyhtiöllä on oma instanssi järjestelmässä
Käyttötapauksen kulku	1. Pääkäyttäjä kirjautuu järjestelmään 2. Pääkäyttäjä aloittaa lisäämistoiminnon 3. Pääkäyttäjä syöttää uuden taloyhtiön tiedot järjestelmään
Poikkeuksellinen toiminta	Taloyhtiölle on jo luotuna asiakkuus

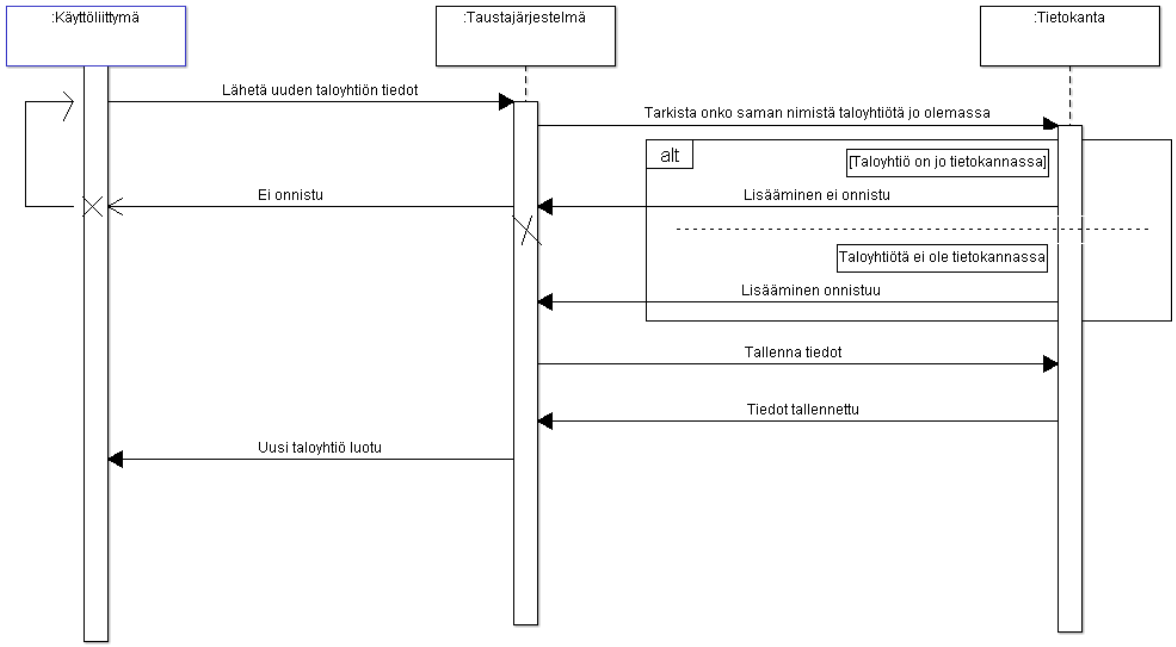
Taulukko 2: *Taloyhtiön-yhteisön lisääminen järjestelmään* – Käyttötapausmalli.

4.4 Sekvenssikaaviot

Käyttötapausmalleista tehtiin Unified Modeling Language (UML) -kuvaustekniikan mukaiset sekvenssikaaviot. Sekvenssikaaviossa kuvataan yksittäisen toiminnon suoritusjärjestys ja -logiikka. Kaavion tarkoitus oli selventää käyttöliittymän, taustajärjestelmän ja tietokannan välisiä yhteyksiä ja toimenkuvia.

Ohjelmistoprojektissa käytetyt kaaviot eivät ole tarkkoja ohjelmalogiikkaa mallintavia sekvenssikaavioita, vaan niissä on tarkoituksena kuvata myös esimerkiksi liittymien välistä kommunikaatiota, kuten kaaviossa, jossa selvitetään *taloyhtiö-yhteisön lisääminen järjestelmään* toiminnon etenemistä (kuvio 1). Pääkäyttäjä lähettää taloyhtiön tiedot käyttöliittymässä olevan lomakkeen kautta taustajärjestelmälle, joka käsittelee pyynnön. Taustajärjestelmä tarkistaa, onko tietokannassa jo valmiiksi luotuna taloyhtiö kyseisillä tiedoilla. Jos tietokannassa on jo luotuna kyseinen taloyhtiö-yhteisö, ilmoitetaan pääkäyttäjälle käyttöliittymän kautta, että taloyhtiön lisääminen järjestelmään ei onnistu. Jos tietokannasta ei löydy taloyhtiötä samoilla tiedoilla, tallennetaan pääkäyttäjän lähettämät tiedot tietokantaan. Järjestelmä ilmoittaa pääkäyttäjälle, kun lisääminen on onnistunut.

Todellisuudessa toimintoon liittyy muitakin alitoimintoja, mutta kaikkea ei ole tarpeellista esittää sekvenssikaaviossa. Sekvenssikaavioon kannattaa aina merkitä vain sen verran tietoa, mikä on luettavuuden ja ymmärrettävyyden kannalta tarpeen (Luukkainen & Laine 2012, 63). Tärkeintä on esittää toiminnon suoritus mahdollisimman yksinkertaisesti. Esimerkiksi pääkäyttäjän lähettämien tietojen validointi voidaan jättää merkitsemättä sekvenssikaavioon käyttötapauksessa *taloyhtiö-yhteisön lisääminen järjestelmään* (kuvio 1).
Ohjelmistoprojektissa käytetyt



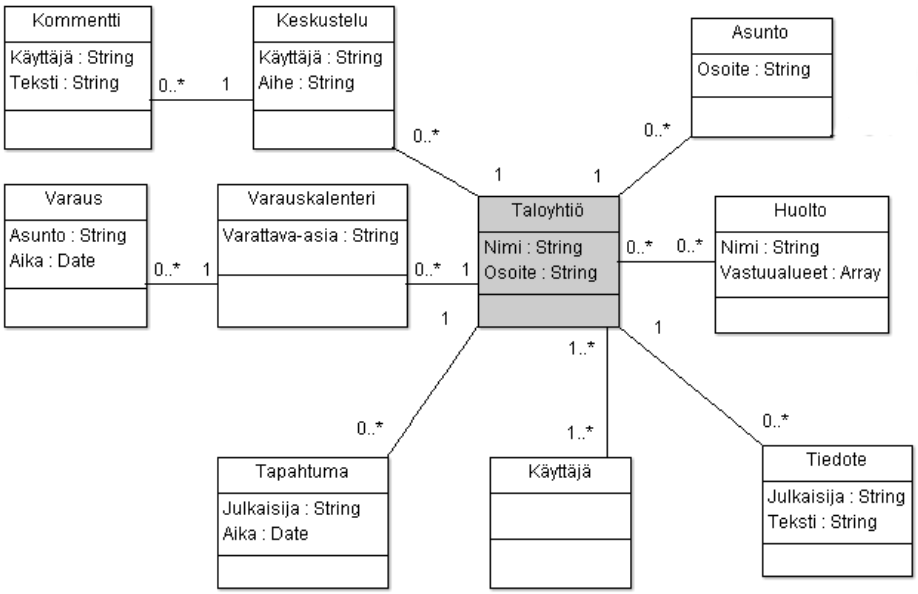
Kuvio 1: Sekvenssikaavio toiminnosta taloyhtiö-yhteisön lisääminen järjestelmään.

4.5 Tietomalli

Järjestelmän kokonaisuutta selventämään ja järjestelmän osien yhteyttä hahmottamaan tehtiin alustavat tietomallit. Tietomallit selvittävät järjestelmän keskeisiä termejä ja jaottelevat järjestelmän abstraktien-osien suhteita. Alustavat tietomallit toimivat pohjana myös myöhemmässä vaiheessa toteutettavan tietokannan tietomallille. Tietokanta on paikka, jonne sovellus tallentaa tarvitsemansa tiedon ja sen tietomalli on mallinnus sinne tallennettavan tiedon rakenteesta ja ominaisuuksista.

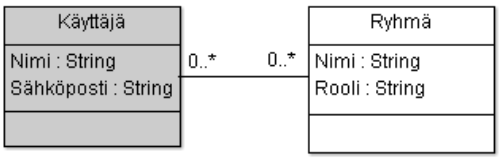
Tietomallit jaettiin projektin alkuvaiheessa käyttäjä-tietomalliin ja taloyhtiö-tietomalliin. Käyttäjä-tietomalli sisältää käyttäjän perustiedot ja kirjautumistiedot (kuvio 3). Lisäksi käyttäjä-tietomalli sisältää taloyhtiö-yhteisöt, joihin kyseinen käyttäjä kuuluu sekä roolin, joka

käyttäjällä on sen yhteisön sisällä. Taloyhtiö-tietomalli sisältää taloyhtiön perustietojen lisäksi taloudet, jotka taloyhtiöön kuuluvat sekä niiden tiedot ja asukkaat (kuvio 2). Lisäksi taloyhtiö-tietomalliin kuuluu muun muassa taloyhtiön asiakirjat, keskustelut, tapahtumat ja huolto-palvelut.



Kuvio 2: Alustava Taloyhtiö-tietomalli.

Järjestelmän abstrakteja-osia, jotka sisältävät joukon loogisesti yhteenkuuluvaa toiminnallisuutta ja tietoa, kuten henkilöt tai taloyhtiöt, kutsutaan olioiksi. Alustava tietomalli ei ota tarkasti kantaa tietomallin olioiden ominaisuuksiin, vaan pyrkii lähinnä osoittamaan olioiden väliset suhteet. UML-kuvaustekniikan avulla voidaan tarkastella tietomallin rakennetta. Tietomallista nähdään, mistä tiedoista tietty kokonaisuus koostuu. Esimerkiksi taloyhtiö sisältää asuntoja ja asunto sisältää käyttäjiä. Lisäksi nähdään ominaisuuksien välinen suhde toisiinsa numeroina ominaisuuksia yhdistävien viivojen molemmissa päissä. Esimerkiksi taloyhtiöllä voi olla nollasta ylöspäin mielivaltainen määrä asuntoja, mutta asunnolla voi olla vain yksi taloyhtiö.



Kuvio 3: Alustava Käyttäjä-tietomalli.

5 Kehitysympäristö

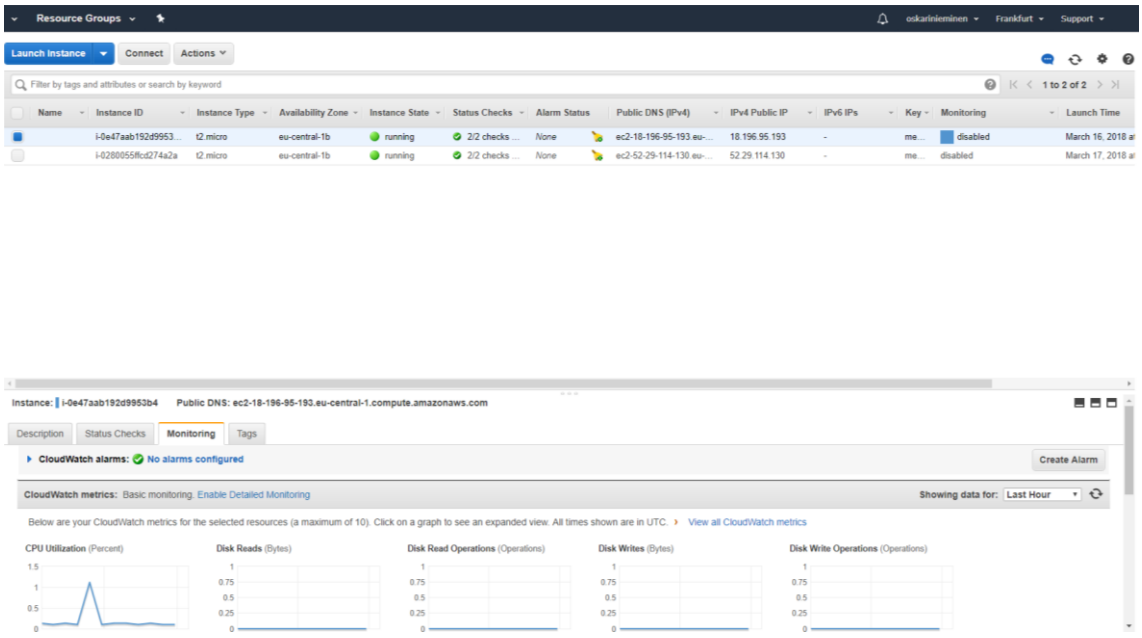
Järjestelmän kehittämisen eri vaiheissa käytettiin erinäisiä sovelluksia ja työvälineitä toteuttamisen helpottamiseksi. Ohjelmointiin käytettiin SublimeText -tekstieditoria, versionhallintaan käytettiin GitLab -versionhallintamanageria, testaamiseen käytettiin Post Man -sovellusta, tietokannan kehittämiseen käytettiin mLab -tietokantapalvelua ja sovelluksen taustajärjestelmä asetettiin toimimaan Amazon Web Services -pilvipalvelualustalle.

5.1 Amazon Web Services

Amazon Web Services (AWS) tarkoittaa Amazonin pilvipohjaisia verkkopalveluita, joka tarjoaa suoritintehoa, tiedon varastointia ja sisällönhallintaa. AWS:n palvelut skaalautuvat käyttäjän tarpeiden mukaisesti. Esimerkiksi kehitettävän järjestelmän käyttäjämäärien kasvaessa ja järjestelmän tarvitessa enemmän toimintatehoa, antaa AWS automaattisesti lisää suorittimia järjestelmän käyttöön, jotta ohjelman suoritusteho ei laskisi ja sovellus pysyisi toimivana käyttäjille. Kun järjestelmällä on vähemmän käyttäjiä ja se ei tarvitse suurta suoritintehoa, antaa AWS automaattisesti vähemmän toimintatehoa järjestelmälle vähentäen kustannuksia.

AWS:n palveluiden käyttöönotto ei vaatinut alkuinvestointeja ja palvelun maksut kertyivät käytön mukaan. AWS:n verkkopalveluiden automatisoidessa palvelimien toiminnan, oli sovelluksen kehittäminen mahdollista ilman suurempaa perehtymistä fyysisiin laitteisiin. (Amazon Web Services -pilvipalvelut)

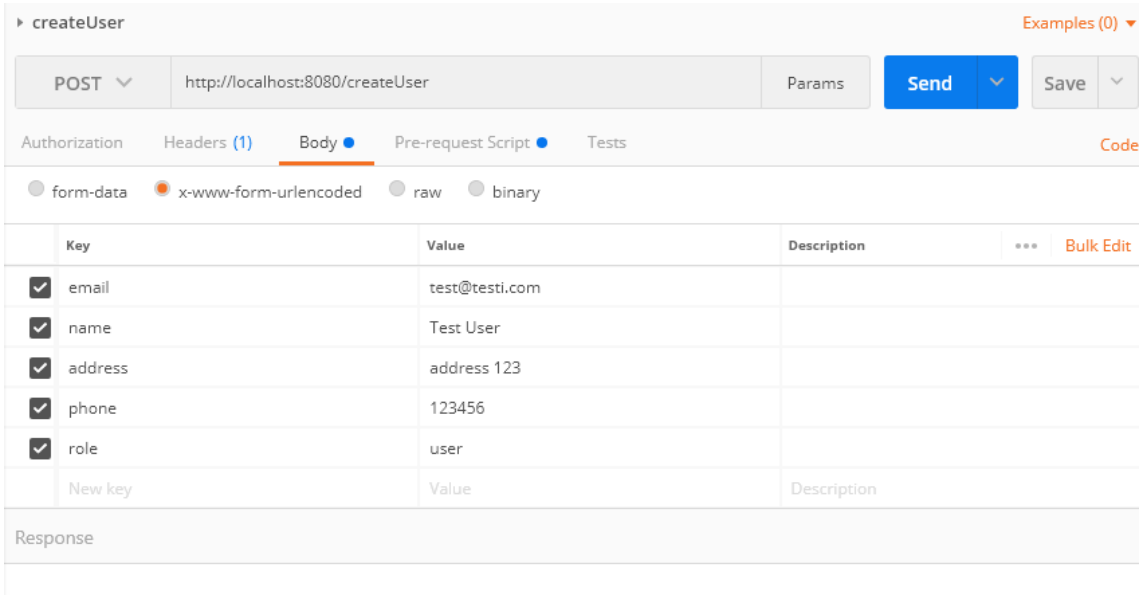
AWS:n verkkopalveluista sovelluksen kehittämisen ensimmäisessä vaiheessa otettiin käyttöön Amazon Elastic Cloud Computing -virtuaalipalvelimet (EC2), joita luotiin turvallisuussyistä kaksi (kuvio 4). Toiseen asennettiin sovelluspalvelin, joka pyörittää sovelluksen taustajärjestelmää. Toiselle palvelimelle asennettiin sovelluksen tietovarasto, jonne tallennetaan sovelluksen käyttämä tieto.



Kuvio 4: AWS:n EC2 -palvelussa käynnissä olevat virtuaalipalvelimet.

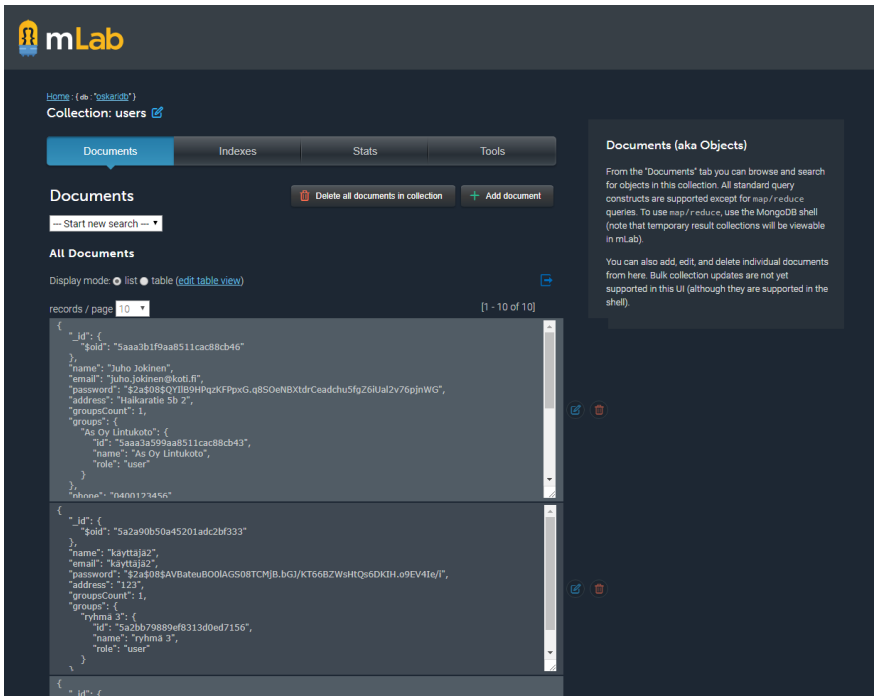
5.2 Testaaminen

Järjestelmää testattiin kehittämistyön ohessa jatkuvasti yksikkötestauksella ja järjestelmätason testeillä. Yksikkötestauksessa kokeiltiin yksittäisten funktioiden ja toimintojen toimimista ja järjestelmätason testauksessa kokeiltiin järjestelmän osien yhdessä toimimista. Automatisoituja testejä sovellusta kehitettäessä ei kirjoitettu vaan järjestelmän toimintaa testattiin vain kokeilemalla.



Kuvio 5: POST-kutsun lähettäminen Post-man sovelluksella.

Taustajärjestelmän Hypertext Transfer Protocol eli hypertekstin siirtoprotokolla (HTTP) -kutsujen testaamiseen käytettiin Post-man sovellusta, jolla pystyttiin tekemään pyyntöjä palvelimelle (kuvio 5) (getpostman). Pyyntöjä tekemällä selvitettiin myös tietokantakutsujen toimivuus. Sovelluksen testaamisessa tietokantana käytettiin mLab -alustaa, jossa sinne tallennettavat tietokannat ovat graafisesti nähtävissä (kuvio 6). Tietokantoihin tallennetut dokumentit ovat mLab -alustalla tarkasteltavissa ja muokattavissa (mLab). Näin voitiin kehittämisvaiheessa esimerkiksi ensin tallentaa tarvittava testidokumentti tietokantaan ja testata järjestelmän toimivuutta noutaa dokumentti. Vastaavasti voitiin Post-man -sovelluksella viedä HTTP-kutsun avulla tietokantaan dokumentti ja tarkastaa mLab -sovelluksessa sen oikeellisuus. Post-manin pyynnöillä testattiin järjestelmän tarjoamaa rajapintaa, jonne sovelluksen käyttöliittymä tekee HTTP-pyyntöjä.



Kuvio 6: mLab -sovelluksessa käyttäjätiedot -kokoelma.

Kehitettäessä esimerkiksi käyttäjän luomista järjestelmään, palvelimelle kirjoitettiin ensin reitti POST-kutsulle tiettyyn URI-osoitteeseen, joka vastaanottaa saapuvan kutsun ja sen sisältämät käyttäjätiedot (kuvio 7). Kutsu voitiin helposti lähettää monta kertaa eri arvoilla ja tulostaa näkyviin palvelimella, jolloin voitiin todeta tiedon saapuvan perille halutulla tavalla (kuvio 8). Tämän vaiheen toimiessa, voitiin toteuttaa toiminnallisuus, joka tarkastaa palvelimelle saapuvan tiedon ja käsittelee sen.

```
app.post('/createUser', function(req, res){
    // Vastaanottaa /createUser uri-osoitteeseen saapuvan POST-kutsun.
    // Kutsun mukana lähetetyt käyttäjä tiedot ovat tarkasteltavissa ja muokattavissa.
    console.log( req.body.email, req.body.name, req.body.phone);
    // Tarvittavien toimenpiteiden jälkeen käyttäjälle lähetetään palautetta
    // toiminnon onnistumisesta tai epäonnistumisesta sekä mahdolliset ohjeet jatkoon.
    res.json({
        userCreatedMessage : messages.userCreatedMessage,
        errorMessage : messages.errorMessage,
        existingUserMessage : messages.existingUserMessage,
        existingUserId : messages.existingUserId
    })

});
```

Kuvio 7: POST-kutsun vastaanottaminen palvelimella.

Tämän jälkeen pystyttiin taas post-man -sovelluksella testaamaan, että tiedot käsitellään ja luodaan halutulla tavalla. Sitten kirjoitettiin toiminnallisuus, joka lähettää tarkastetut käyttäjän tiedot tietokantaan. MLab -sovelluksen avulla, voitiin tarkastaa, että lähetettävät tiedot tallentuvat oikein; salasanat ovat salattuja, järjestelmään ei voi luoda useita tunnuksia samoilla tiedoilla ja dokumentin arvot tallentuvat oikeaan kohtaan.

```
{ email: 'test@test.com' } { name: 'Test User' } { phone: '123456' }
POST /createUser 200 70ms - 110b
```

Kuvio 8: Käyttäjätietojen tulostuminen komentoriville.

5.3 Versionhallinta

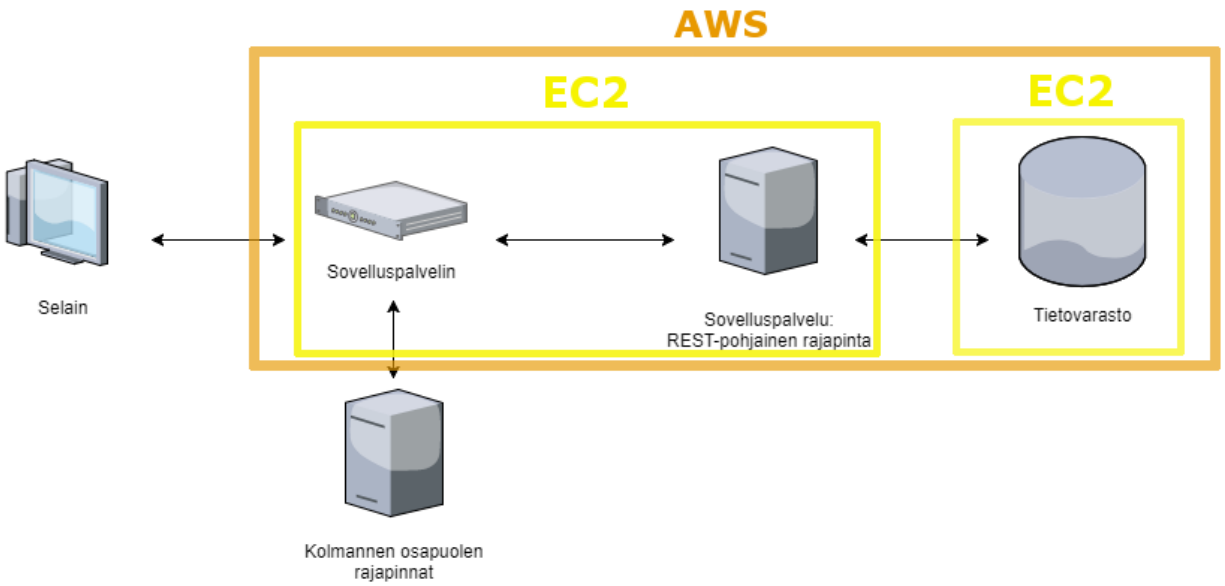
Järjestelmän kehittämisessä käytettiin GIT-versionhallintajärjestelmää. GIT-versionhallintajärjestelmän avulla voidaan tallentaa ohjelman lähdekoodin eri versiot niin, että jälkeenpäin on nähtävissä ja palautettavissa tehdyt muutokset (git-scm).

Versionhallinnan avulla ohjelman lähdekoodi tallennettiin aina ulkoiselle palvelimelle GitLab -alustalle luodulle yksityiselle tilille. Sovellusta kehitettäessä aina uuden ominaisuuden lisäämisen jälkeen tallennettiin muutokset ja kirjattiin kommenttiin, mitä muutoksia lähdekoodiin tehtiin. Näin voitiin ylläpitää selkeää dokumentaatiota ohjelman kehityksestä.

6 Sovellusarkkitehtuuri

Sovelluksen arkkitehtuuri koostuu tietovarastosta, sovelluspalvelimesta, www-sovelluspalvelusta, kolmannen osapuolen rajapinnoista ja asiakasohjelmasta (kuvio 9). Tietovarastona toimii Mongodb -tietokanta, joka on asennettu omalle virtuaalipalvelimelleen AWS:n EC2 -palveluun. Myös Node.js -sovelluspalvelin on asennettu omalle virtuaalipalvelimelleen. Node.js -virtuaalipalvelimella ajettava ohjelma eli www-sovelluspalvelu sisältää järjestelmän REST -

pohjaisen rajapinnan. Kolmannen osapuolen rajapinnat kuuluvat sovelluksen myöhemmin toteutettaviin osiin, mutta suunnittelun osalta ne ovat oleellista hahmotella arkkitehtuuri-malliin. Asiakasohjelmana toimii käyttäjän selain, joka pyörittää käyttöliittymää ja lähettää http-pyyntöjä taustajärjestelmälle.



Kuvio 9: Sovellusarkkitehtuuri.

6.1 Taustajärjestelmä

Taustajärjestelmä käsittelee käyttäjän selaimelta tulevia http-pyyntöjä sekä hakee, päivittää ja luo dokumentteja tietokantaan. Se myös huolehtii tarvittavien tietojen salaamisesta, käyttäjän autentikoinnista eli todennuksesta ja hallinnoi sessioita. Taustajärjestelmä vastaa muun muassa siitä, ettei tietokannan tietoja pääse tarkastelemaan tai muokkaamaan, kuin siihen oikeuttavan roolin omaava järjestelmään kirjautunut käyttäjä. Suurin osa järjestelmän toimintalogiikasta tapahtuu taustajärjestelmässä.

6.1.1 REST-rajapinta

Taustajärjestelmä sisältää sovelluksen rajapinnan, jonka tehtävänä on käsitellä käyttöliittymältä tulevia kutsuja. Rajapinta voi vastaanottaa käyttöliittymästä pyynnön mukana tulevaa tietoa, manipuloida sitä sekä lähettää vastauksena takaisin tietoa käyttöliittymälle.

Sovelluksen rajapinta on REST-pohjainen rajapinta. REST eli *Representational State Transfer* on asiakasohjelman ja palvelimen välinen kommunikaatio-arkkitehtuuri, joka erottelee tietolähteen ja käyttöliittymän toisistaan. Palvelin hallinnoi muistilähteitä eikä ole riippuvainen tietyn muotoisesta käyttöliittymästä vaan käyttöliittymä voi olla toteutettu millä ohjelmointi-

kielellä ja tyylillä tahansa. REST -verkkopalvelu vain ylläpitää tilaa asiakasohjelman ja palvelimen välillä. REST -verkkopalvelut käyttävät HTTP -verbejä ilmoittaakseen palvelimelle, mitä toimintoja asiakasohjelma haluaa tehtävän. (Elliott 2014, 6-8.)

HTTP -verbejä ovat POST, GET, PUT ja DELETE. HTTP POST -verbiä käytetään, kun luodaan uusi kirjaus tietoresurssin kokoelmaan. HTTP GET -verbillä noudetaan tietoresurssin tiedot. HTTP PUT -verbiä käytetään päivittämään jo luotu tietoresurssi-tiedosto. (kuvio 10) HTTP DELETE -verbiä käytetään poistettaessa tietoresurssi-tiedosto. HTTP -verbit lähetetään asiakasohjelmasta palvelimelle lähde URI:lla (uniform resource indicator). Jokaisella lähteellä on palvelimella oma uniikki URI. (Elliott 2014, 6-8.) Esimerkiksi luodessa uuden käyttäjän järjestelmään lähetetään taustajärjestelmälle URI */user/* POST-pyynnöllä, jossa on mukana uuden käyttäjän tiedot. Näin järjestelmä tietää, mikä toiminto tulee suorittaa.

Sovelluksen käyttöliittymän ja taustajärjestelmän toimiessa JavaScript-ohjelmointikielellä, on järkevää säilyttää ja siirtää tietoa JSON-muodossa (JavaScript Object Notation) sen ollessa osa JavaScriptin alkuperäistä olio-pohjaista syntaksia. JSON:a käytetään tiedon esittämiseen, siirtämiseen ja tallentamiseen. (Flanagan 2011, 138.). Lisäksi järjestelmässä tiedon tallentamiseen käytettävä Mongodb -tietokanta on JSON-pohjainen, joten järjestelmässä liikkuva informaatio voidaan pitää samassa muodossa riippumatta järjestelmän tasosta.

REST -tapahtumaketju käyttäjätietojen muokkaamiselle:

1. Käyttäjä pyytää tietoa palvelimelta HTTP GET -kutsulla. Kutsun URI-osoite on */user/fg24yh2y*, jossa ensimmäinen osa viittaa haettavaan tietolähteeseen ja jälkimmäinen osa on haettavan käyttäjän uniikki identifikaatio-koodi eli id. (kuvio 10)

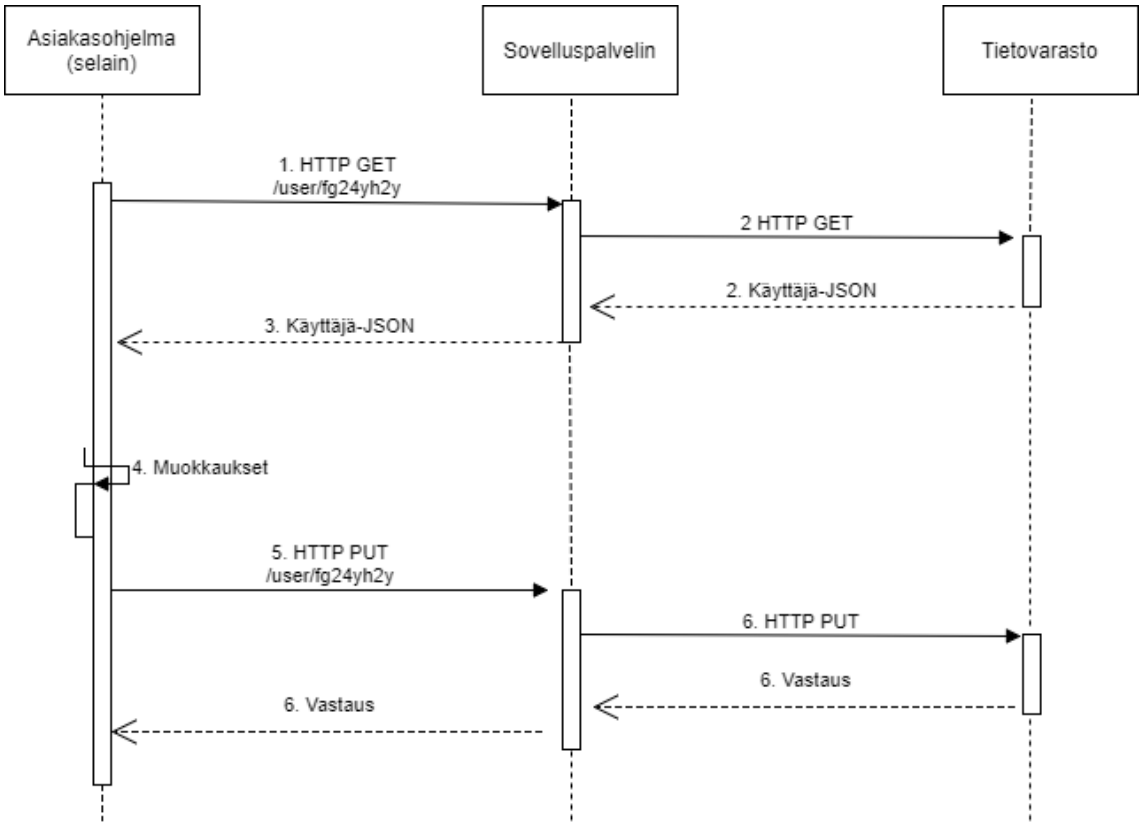
2. Sovelluspalvelin noutaa käyttäjän tiedot tietokannasta käyttäen URI-osoitteen mukana tullutta id-indikaattoria ja pakkaa tiedot asiakasohjelmalle sopivaan JSON-muotoon. (kuvio 10)

3. Käyttäjän tiedot palautetaan asiakasohjelmalle JSON-muodossa. (kuvio 10)

4. Asiakasohjelma käsittelee käyttäjän tietoja. (kuvio 10)

5. Asiakasohjelma kutsuu samaa URI-osoitetta PUT-pyynnöllä lähettäen sovelluspalvelimelle takaisin muokatut käyttäjän tiedot. (kuvio 10)

6. URI-osoitteen id-indikaattoria vastaavan käyttäjän tiedot korvataan PUT-pyynnön tiedoilla. Asiakasohjelmalle lähetetään vastaus onnistuneesta PUT-pyynnöstä. (kuvio 10)



Kuvio 10: REST tapahtumaketju.

6.1.2 Työvälineet ja teknologiat

Sovelluspalvelimeksi perustettuun Amazon Elastic Cloud Computing -instanssiin asennettiin Node.js -viitekehys, jonka avulla voitiin käyttää samaa JavaScript-ohjelmointikieltä taustajärjestelmässä, kuin mitä käyttöliittymän kehittämisessä käytettiin. Node.js on skaalautuva, nopea ja asynkroninen. Asynkronisuudella tarkoitetaan, että ohjelman suorittaminen ei välttämättä etene rivi kerrallaan vaan aikaa vievä operaatio, kuten tiedoston lataaminen, saattaa jäädä suorittamaan toimintoaan samalla, kun muu ohjelma jatkaa suorittamistaan järjestyksessä ja palaa tähän toimintoon, kun se on valmis. Tämän ansiosta Node.js -ohjelmat eivät lukitu paikalleen, vaikka jokin toiminto veisikin paljon aikaa. (Allam, Pollack)

Järjestelmän käyttämien lisäosien ja laajennosten hallinnointii käytetään npm -pakетinhallintajärjestelmää. Paketinhallintajärjestelmän tarkoitus on automatisoida järjestelmän käyttämien ohjelmien asentaminen, päivittäminen ja poistaminen (Opensuse. 2017). Npm pitää sisällään maailman suurimman tietokoneohjelmistoarkiston, jossa JavaScript -kehittäjät voivat jakaa avoimenlähdekoodin paketteja (what-is-npm. 2018).

6.1.3 Lisäosat ja laajennokset

Taustajärjestelmän käyttämät lisäosat ladattiin npm -paketinhallintajärjestelmää käyttäen. Järjestelmän käyttämät lisäosat ja laajennokset sisältävien pakettien tarkoituksena oli helpottaa tiettyjen toiminnallisuuksien toteuttamista ja näin vähentää ja selkeyttää kehittäjän itse kirjoittamaa koodia. Lisäosat ja laajennokset tarjoavat usein ratkaisua, jonkin yksittäisen ongelman ratkaisemiseen tai helpottavat kehittäjien usein kohtaamien monivaiheisten prosessien tekemistä. Toteutetun järjestelmän toimimisen kannalta oleellisimpia lisäosia ja laajennoksia olivat Express.js, Passport, Bodyparser ja MongoConnect.

Express.js on joustava Node.js -web-sovellus-viitekehys, joka tarjoaa laajan joukon ominaisuuksia käytettäväksi web- ja mobiilisovelluksille (Expressjs. 2017). Toteutettava järjestelmä käyttää Express.js:n tarjoamaa route -osiota, static -väliohjelmistoa ja erillisesti asennettavaa express-session -pakettia. Route -osio tarjoaa toiminnallisuuden sovelluksen reitittämiselle eli sille, miten taustajärjestelmä vastaa asiakasohjelmistolta (selaimelta) tuleviin pyyntöihin. Static -väliohjelmisto jakaa tarvittavat staattiset tiedostot kuten kuvat ja tyylitiedostot niin, että asiakasaohjelmisto löytää ne tietämättä täydellistä tiedostosijaintia. Express-session tarjoaa toiminnallisuuden istuntoon liittyvien tietojen tallentamista varten. Tieto pidetään taustajärjestelmässä ja asiakasohjelma saa tekemiensä pyyntöjen yhteydessä vain identifikaatiotunnuksen, joka yhdistää sen tiettyyn istuntoon. Asiakasjärjestelmän tehdessä toiminnon, lähetetään identifikaatiotunnus pyynnön yhteydessä taustajärjestelmälle. Taustajärjestelmä tarkastaa, onko kyseiselle tunnukselle voimassa olevaa istuntoa. Näin taustajärjestelmä tietää esimerkiksi, onko käyttäjä kirjautunut sisään vai ei.

Passport on Express-sovelluksissa käytettävä väliohjelmisto, joka käsittelee todennuksia (Passportjs). Passportin avulla sovelluksen käyttäjä saadaan todennettua istuntoon käyttäjätunnuksen ja salasanan avulla.

Bodyparser on Node.js:n väliohjelmisto, joka muuttaa taustajärjestelmälle saapuvan tiedon helpommin käsiteltäväksi. Tiedon saapuessa taustajärjestelmälle, se tulee pienissä osissa, Bodyparser kokoaa tiedon kokonaisuudeksi ja jäsentelee sen.

MongoConnect tarjoaa toiminnallisuuden yhteyden luomiselle ja ylläpitämiselle taustajärjestelmän ja MongoDB-tietokannan välillä. MongoConnect tarjoaa myös toiminnallisuuden käyttäjän istuntojen säilömiselle.

6.2 Tietokanta

Tietokanta toimii järjestelmässä käsiteltävän tiedon tallennuspaikkana. Sovelluksen tarvitsemat dokumentit, kuten käyttäjätiedot, on tallennettu tietokantaan, josta tiedot voidaan hakea ja jonne tietoa voidaan tallentaa.

Tietokantaa suunniteltaessa oli otettava huomioon järjestelmän tarpeet tiedon käyttämiseen. Ensimmäisenä on valittava tietokantatyyppi, jota järjestelmä käyttää.

Perinteisemmässä relaatiotietokannassa on määriteltävä tietomalli ennen, kuin tietoa viedään tietokantaan. Relaatiotietokannan tietomalli on rakenne, joka kuvataan virallisella tietokannan tukemalla kielellä ja tarjoaa suunnitelman tietokannan taulukoille sekä taulukoiden ja tiedon välisille suhteille. Relaatiotietokannoissa taulukkoihin on määriteltävä valmiiksi rivit ja nimetyt sarakkeet sekä tiedon muoto, jossa se säilytetään yksittäisessä sarakkeessa.

Relaatiotietokannasta poikkeavia tietokantoja kutsutaan NoSQL -tietokannoiksi. Yksi NoSQL-tietokantatyyppi on dokumenttitietokanta, jossa tietokannan rakenne koostuu dokumenteista. Dokumentit voivat olla määrittelemättömän monitahoisia sisältäen erilaisia tietomuotoja ja ala-kategorioita. (Compare Docs vs Relational 2018) Dokumenttitietokanta yleensä säilöö tiedon JSON-muotoisena, mikä sopii erityisen hyvin web-sovelluksien kehittämiseen, sillä JSON on muoto, jolla natiivisti kommunikoidaan JavaScript -kielellä (Elliott 2014. 6).

6.2.1 Työvälineet ja teknologiat

Sovelluksen vaatimusten mukaan järjestelmään valittiin dokumenttitietokanta perinteisemmän relaatiotietokannan sijaan. Tietokantakieleksi valittiin MongoDB sen skaalautuvuuden ja selkeän syntaksin vuoksi. Järjestelmään oli tarkoituksena pystyä lisäämään ja muokkaamaan tietomalleja jälkeenpäin, jolloin olio-pohjainen tietokantarakenne on joustavuutensa ansiosta sopivampi valinta. Lisäksi on eduksi, että tietokanta tallentaa tiedot JSON-muodossa, jolloin järjestelmässä käytettävä data pysyy samassa muodossa järjestelmän osasta riippumatta.

MongoDB on joustavuutensa lisäksi sopiva kyseiseen järjestelmään, sillä tietokantadokumenttien on tärkeää pystyä sisältämään monia erilaisia arvoja. Esimerkiksi päivämäärien, listojen ja sisäkkäisten olioiden tallentaminen onnistuu MongoDB:ssä.

Kehitysvaiheessa järjestelmä osoitettiin käyttämään mLab -sovelluksen tarjoamaa tietokantaa, jonne järjestelmän käyttämä tieto tallennettiin. MLab -sovelluksessa tietokantaan tallennetut tiedot ovat graafisesti nähtävillä ja muokattavissa, mikä tekee ohjelman kehittämisestä ja testaamisesta helpompaa.

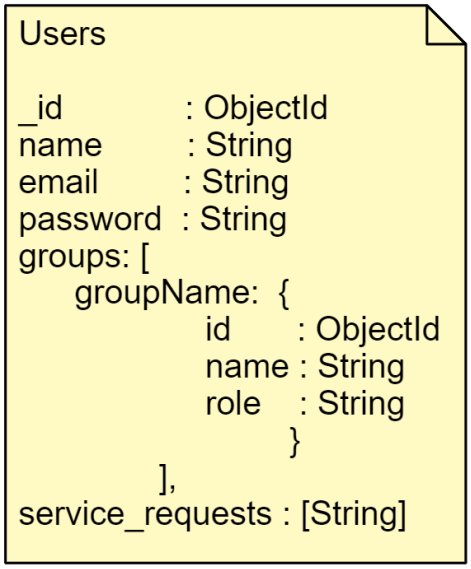
Kehittämisvaiheesta siirrettäessä sovellus tuotantoon, otetaan käyttöön oma salattu MongoDB -tietokanta. Tuotannossa tietokanta on asennettu omalle Amazon Elastic Compute Cloud (Amazon EC2) -instanssilleen. Tietokantaa pääsee käyttämään asetetun käyttäjätunnuksen ja salasanan avulla, jotka ovat sovelluksen taustajärjestelmällä hallussa. Järjestelmän taustajärjestelmä ja tietokanta siis ovat käynnissä erillisillä palvelimilla, jotka ovat yhteydessä toisiinsa.

6.2.2 Tietomallit

Tietomallit käsittävät sen viitekehyksen, missä muodossa tiedot ovat tietokannassa ja mikä on niiden yhteys toisiinsa. Malli sisältää dokumentin attribuutit ja niiden tietomuodon.

Järjestelmän tietomallit suunniteltiin niin, että tarvittava tieto on tehokkaasti haettavissa tietokannasta. Tietomalleja suunniteltaessa otettiin huomioon, miten paljon tietoa yksi dokumentin kenttä sisältää sekä miten usein ja mitä tietoa dokumentista haetaan. Jos esimerkiksi yksi dokumentin kenttä sisältää listallisen tietoa ja lista mahdollisesti kasvaa suureksi, voi olla tarpeellista tallentaa se omaan dokumenttiinsa.

Projektin alkuvaiheessa suunnitellut tietomallit toimivat tietokantaan toteutettavien tietomallien pohjana. Alussa tietokannassa oli kaksi kokoelmaa. Toinen käyttäjän tiedoille, joka sisälsi sähköpostin, salatun salasanan, listan käyttäjän tekemien palvelupyyntöjen identifiointi-tunnuksista ja listan ryhmistä, joihin käyttäjä kuuluu, sekä käyttäjän roolin kunkin ryhmän sisällä (kuvio 11). Toiseen kokoelmaan kuului taloyhtiön tiedot joita ovat alustavasti: taloyhtiön nimi, y-tunnus, pinta-ala, taloudet, isännöitsijä, huoltoyhtiöt, keskustelut ja palvelupyynnöt. Näistä ainakin keskustelut ja palvelupyynnöt tullaan eriyttämään myöhemmissä vaiheissa omiksi kokoelmikseen, jotka sitten linkitetään taloyhtiö -kokoelmaan identifiointi -tunnuksella.



Kuvio 11: Tietokannan kokoelmaksi suunniteltu tietomalli käyttäjän tiedoille.

6.3 Graafinen käyttöliittymä

Käyttöliittymä on näkymä, jossa sovelluksen käyttäjä toimii. Käyttöliittymän kautta käyttäjä voi pyytää ja lähettää tietoa sovelluksen taustajärjestelmälle käsiteltäväksi. Graafinen käyttöliittymä koostuu painikkeista, tekstikentistä ja linkeistä, joiden avulla käyttäjän on helppoa käyttää sovellusta internet-selaimen välityksellä. Selain ajaa verkkosivun päälle ohjelmoitua ohjelmaa, joka reagoi käyttäjän toimintoihin ja lähettää pyyntöjä rajapintaan, joihin taustajärjestelmä vastaa toimintoon kuuluvalla tavalla.

Käyttöliittymän suunnittelussa otettiin huomioon kaikkien eri käyttäjä-ryhmien tarpeet. Esimerkiksi eri roolien tuli päästä käsiksi eri toimintoihin taloyhtiö-yhteisön sisällä eli käyttöliittymän näkymän tuli vaihdella riippuen käyttäjän roolista. Taloyhtiö-yhteisön -käyttöliittymän lisäksi isännöitsijöillä ja pääkäyttäjillä tuli olla omat käyttöliittymät.

Graafisen käyttöliittymän suunnittelussa toteutettavan sovelluksen osalta otettiin huomioon myös responsiivisuus eli ohjelman käytettävyys erilaisilla päätelaitteilla. Sovelluksen tuli olla helppokäyttöinen niin kosketusnäyttöisillä laitteilla kuin muillakin. Käyttöliittymän näkymä tuli myös pystyä esittämään selkeästi riippumatta käyttäjän päätelaitteen näytön leveydestä.

6.3.1 Työvälineet

Graafisen käyttöliittymän toteuttaminen selaimelle tehtiin käyttämällä HTML-kieltä, CSS -tyyliohjeita ja JavaScript -ohjelmointikieltä. HTML -kielellä määritellään selaimessa ajettavaa ohjelman sisältö. CSS -tyyliohjeiden avulla selain muuttaa sisällön ulkonäköä. Javascript -komentosarjakielellä ohjelmasta saadaan dynaaminen, eli sisältöjä ja tyylejä voidaan muokata käyttäjän toimintojen mukaan.

Käyttöliittymän suunnittelu aloitettiin hahmottelemalla piirtämällä eri näkymiä usealle eri kokoiselle näytölle. Seuraavaksi tehtiin rautalankamallit käyttöliittymän näkymistä Axure RP 8 -ohjelmalla (kuvio 12 ja 13). Axure RP 8 on ohjelmisto, joka mahdollistaa käyttöliittymien suunnittelun. Sillä voidaan esimerkiksi piirtää tavallisia web-komponentteja, kuten painikkeita ja tekstikenttiä sekä siirrellä niitä mielivaltaisesti. Ohjelmalla voidaan myös kokeilla, miltä eri toiminnallisuudet näyttävät. Tässä vaiheessa ei vielä otettu kantaa käyttöliittymän väreihin, tekstifontteihin tai muihin ulkonäköseikkoihin vaan tavoitteena oli suunnitella sisällön asettelemista käyttöliittymään.

Kuvio 12: Rautalankamalli sovelluksen etusivusta.

Käyttöliittymän muotoiluun ja tarkempaan ulkonäön suunnitteluun käytettiin GIMP-kuvankäsittelyohjelmaa. Ohjelmalla suunniteltiin käyttöliittymän värimaailmaa ja fontteja piirtämällä käyttöliittymän näkymiä. Käyttöliittymän ulkonäön tarkempi suunnittelu ei kuulu opinnäytetyön sisältöön.

Kuvio 13: Rautalankamalli sovelluksen etusivusta, mobiiliversio.

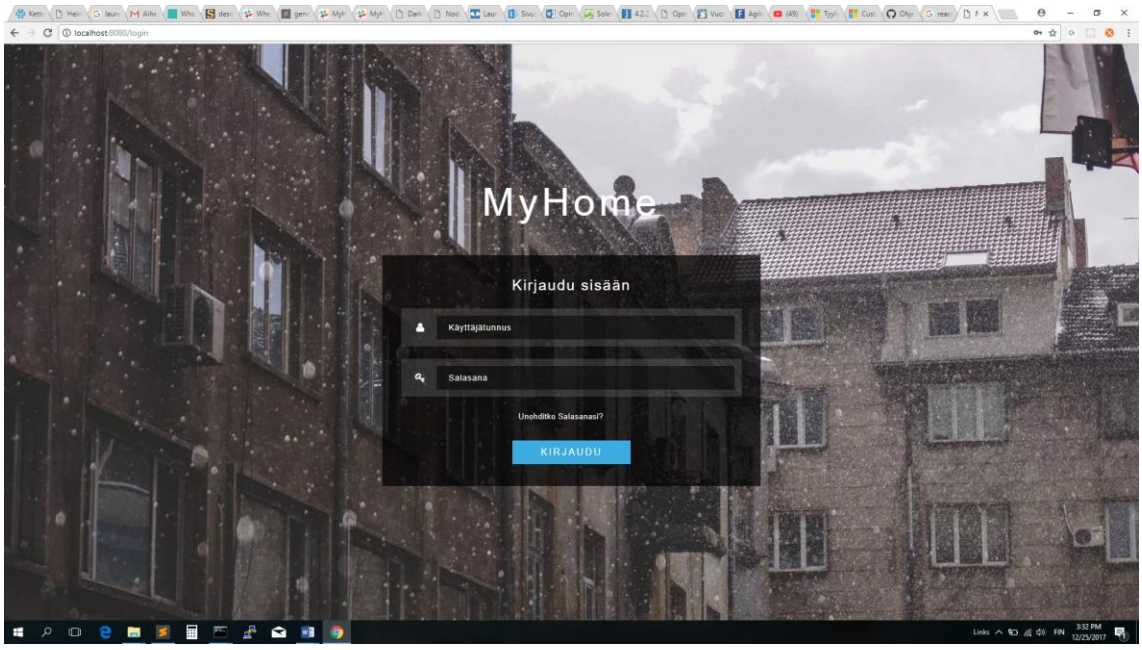
6.3.2 React

Graafisen käyttöliittymän toteuttamiseen käytettiin React -kirjastoa. React on JavaScript -kirjasto käyttöliittymien toteuttamiseen. React:n toiminta perustuu sisäkkäisiin komponentteihin, joiden hierarkiapuu luo kokonaisuuden, joka voidaan ajaa selaimessa. Komponenteilla on oma tila, joka sisältää komponentin itsensä hallinnoimaa tietoa. Komponentin tilaa voidaan muuttaa esimerkiksi käyttäjän toimintojen mukaan, jolloin saadaan luotua interaktiivinen käyttöliittymä. (reactjs, 2018)

React:in tehokkuus perustuu sen tapaan päivittää tietoa. Kun käyttäjä tekee jonkin toiminnon käyttöliittymässä ja yhden tai usean komponentin tila muuttuu, päivittää React vain komponentin, jonka tila on muuttunut sen sijaan, että selaimeen jouduttaisiin lataamaan uudestaan koko käyttöliittymän lähdekoodi. (reactjs, 2018) React:in tapa muokata selaimessa pyörivän ohjelman sisältöä mahdollisimman vähän tekee siitä todella nopean, mikä parantaa käyttöliittymän käyttökokemusta. React:in tyyli jakaa kokonaisuudet komponentteihin pitää React-sovelluksen koodin siististi jäsenneltynä, jolloin sen kehittäminen on mielekästä ja helppoa.

6.3.3 Sisäänkirjautuminen

Sisäänkirjautumis-näkymä on projektin ensimmäisessä vaiheessa sovelluksen graafisen käyttöliittymän ensimmäinen näkymä (kuvio 14). Tästä näkymästä käyttäjä pääsee jatkamaan eteenpäin vasta syötettyään oikean käyttäjätunnuksensa ja salasanansa. Näkymässä on sovelluksen nimi, tekstikentät käyttäjätunnukselle ja salasanalle, kirjaudu-painike sekä linkki, jonka kautta käyttäjä voi pyytää uuden salasanan. Näkymä on yksinkertainen, mutta sisältää silti pientä toiminnallisuutta. Kun käyttäjä painaa, kirjaudu-painiketta, tarkastaa selain, että tekstikentät ovat täytettyinä. Jos molemmat kentät ovat täytetty vaatimusten mukaisesti, lähettää selain käyttäjätunnuksen ja salasanan taustajärjestelmälle tarkistettavaksi. Mikäli kirjautuminen ei onnistu saa käyttöliittymä taustajärjestelmältä tiedon, mikä toiminnossa ei onnistunut. Käyttöliittymä osaa näin ilmoittaa käyttäjälle, miksi kirjautuminen ei onnistu näkymään ilmestyvällä tekstillä.



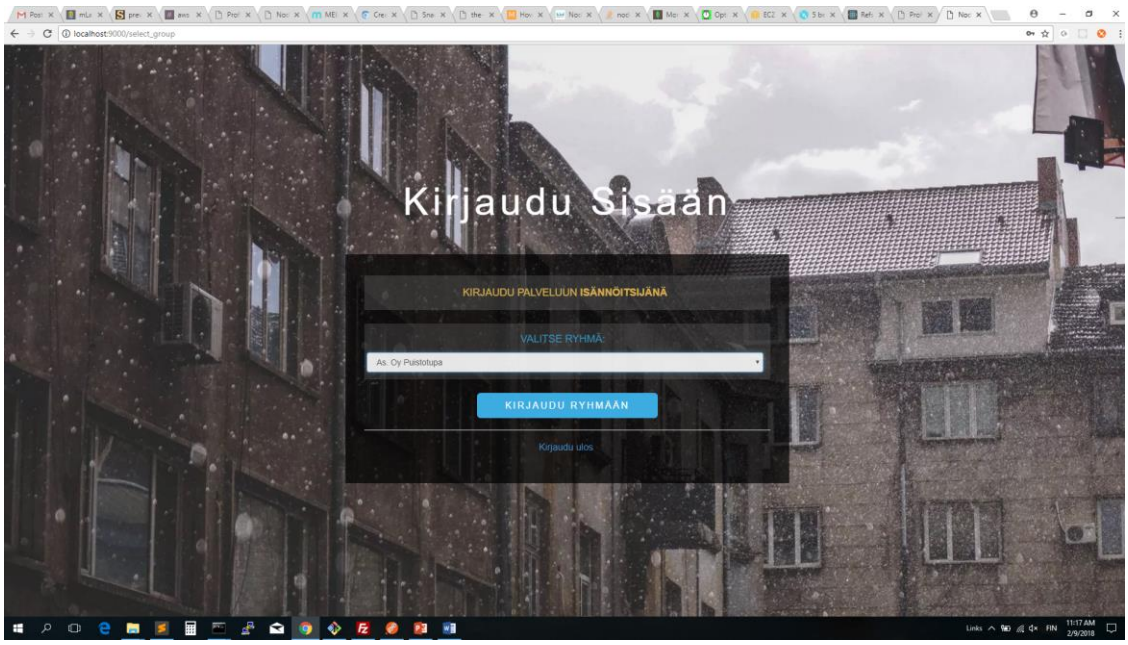
Kuvio 14: Kirjautumis-näkymä selaimessa tietokoneruudulta.

6.3.4 Toiminnan valinta

Käyttäjän syöttäessä oikean käyttäjätunnuksen ja salasanan, siirtää taustajärjestelmä käyttäjän seuraavaan näkymään. Jos käyttäjä kuuluu vain yhteen taloyhtiö-yhteisöön eikä ole isännöitsijä tai pääkäyttäjä, taustajärjestelmä siirtää hänet oman taloyhtiö-yhteisön sivulle. Käyttäjän kuuluessa useampaan taloyhtiö-yhteisöön tai käyttäjän ollessa isännöitsijä, huolto-yhtiö tai pääkäyttäjä, siirretään käyttäjä näkymään, jossa hän saa valita mihin taloyhtiö-yhteisöön

tai näkymään hän haluaa kirjautua (kuvio 15). Tästä näkymästä isännöitsijät pääsevät siirtymään omaan käyttöliittymäänsä, pääkäyttäjä omaansa ja käyttäjät, jotka kuuluvat useaan taloyhtiö-yhteisöön, voivat valita minkä taloyhtiön tilille haluavat kirjautua.

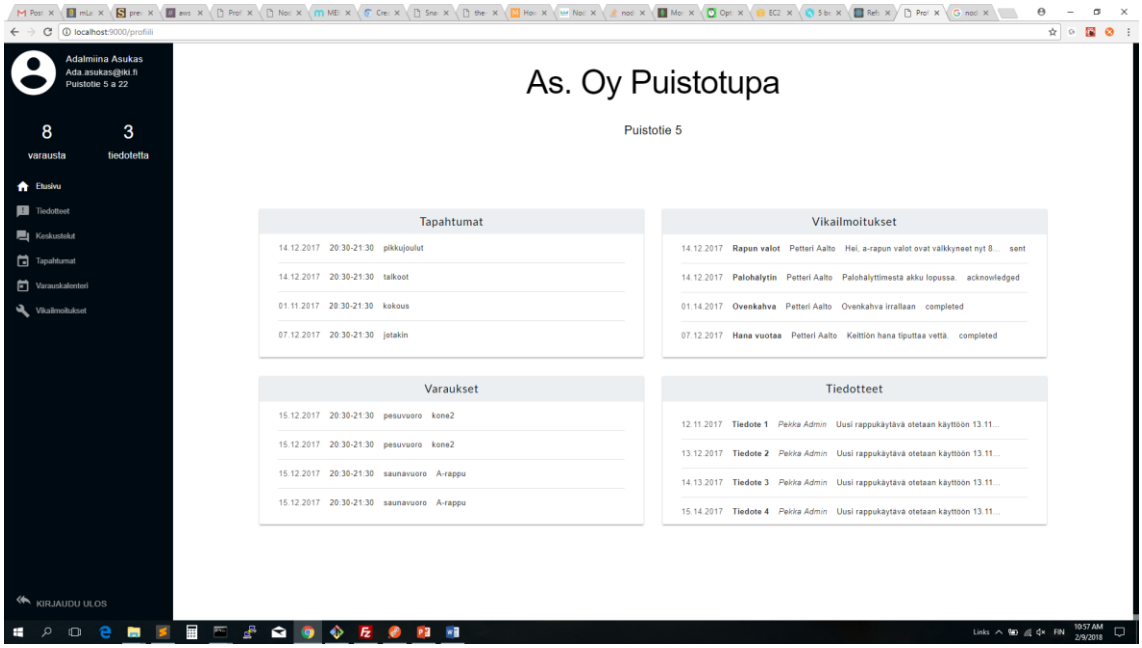
Isännöitsijällä, huolto-yhtiöllä ja pääkäyttäjällä ilmestyy näkymään oman roolin oikeuttama kirjautumis-linkki, joka vie käyttäjän oikeaan käyttöliittymään. Lisäksi valitse-toiminto -näkymään ilmestyy valikko, josta käyttäjä voi valita, mihin taloyhtiö-yhteisöistään kirjaudutaan, mikäli käyttäjä päättää kirjautua yksittäisen taloyhtiöyhteisön sivulle.



Kuvio 15: Toiminnon valinta.

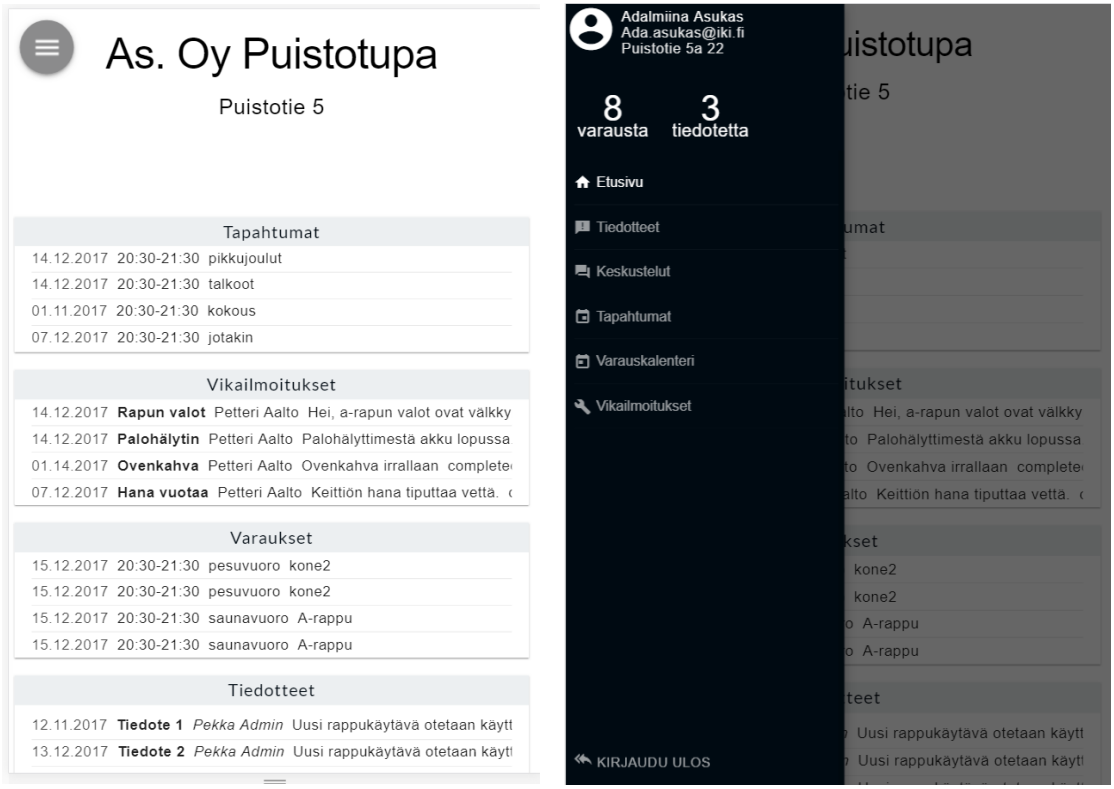
6.3.5 Käyttäjän näkymä

Kun käyttäjä siirretään tietyn taloyhtiö-yhteisön sivulle, ilmestyy hänelle tämän taloyhtiö-yhteisön etusivu-näkymä (kuvio 16 ja 17). Taloyhtiö-näkymä toimii omana React-sovelluksenaan, joka antaa käyttäjälle tämän roolin oikeuttamat toiminnallisuudet näkyville.



Kuvio 16: Sovelluksen etusivu.

Sivustolla on navigaatio-osio, josta käyttäjä voi valita mitä haluaa sivustolla tehdä. Navigaatiosta pääsee siis sovelluksen eri toiminnallisuuksiin sekä mahdollisesti takaisin toiminnon-valinta-osioon tai käyttäjä voi kirjautua kokonaan ulos sovelluksesta, jolloin tausta-järjestelmä lopettaa hänen istuntonsa ja hänen tulee kirjautua uudestaan sisään, päästäkseen käsiksi sovelluksen ominaisuuksiin.



Kuvio 17: Sovelluksen etusivu mobiilissa. Navigaation toiminta.

7 Yhteenveto ja johtopäätökset

Opinnäytetyössä kuvatun ohjelmiston kehittäminen onnistui laadullisesti ja ajallisesti odotusten mukaisesti. Huolimatta kokemattomuudestani ohjelmistokehitysmenetelmiin, opinnäytetyöprojekti eteni ripeästi ja toimeksiantajat olivat tyytyväisiä koko projektin sujuvuuteen. Opinnäytetyön osalta rajattu toiminnallisuus sovellukselle saatiin toteutettua kokonaisuudessaan ja sovellus saatiin onnistuneesti toimimaan verkkoon. Aiemmin itselle tuntemattomat työkalut aiheuttivat pienen viivästyksen alkuperäiseen aikatauluun.

Järjestelmän taustajärjestelmä ja tietokanta saatiin odotusten mukaisesti onnistuneesti asennettua Amazonin pilvipalveluun. Järjestelmää pystyy käyttämään web-selaimessa toimivan käyttöliittymän kautta.

Järjestelmän toimivuutta testattiin erilaisia työvälineitä ja toimintatapoja käyttäen, mutta järjestelmän osien toimivuutta automaattisesti testaavia testejä ei projektin aikana kirjoitettu. Testien puutteellisuus saattaa vaikuttaa sovelluksen jatkokehittämiseen. Jos testejä ei kirjoiteta jälkeenpäin, voi jatkokehittämisen aikana ilmenevien virheiden jäljittäminen olla vaikeampaa.

Käyttöliittymästä tuli riittävän selkeä ja visuaalisesti näyttävä. Käyttöliittymän koodi on myös järjestelty niin, että se on helposti muokattavissa, kun sovelluksen käyttöliittymälle suunnitellaan sopiva ulkoasu. Koko järjestelmän lähdekoodi on jäsennelty selkeästi järjestelmän

osan ja yksittäisten toiminnallisuuksien mukaan, mikä helpottaa jatkokehittämistä kehittäjästä riippumatta.

Yhteistyö toimeksiantajien kanssa sujui hyvin ja viikoittaiset palaverit pitivät kaikki projektiin osallistuvat ajan tasalla. Toimeksiantajien tärkein osuus sovelluksen kehittämisessä oli alkuvaiheen haastattelut, joissa selvitettiin toteutettavan sovelluksen tarkoitus ja ominaisuudet, muuten sain tehdä sovelluksen toteuttamista ja ominaisuuksien suunnittelua itsenäisesti.

Työn aloittamisvaiheessa osa järjestelmään valituista teknologioista oli minulle ennestään tuntemattomia ja osasta minulla oli hyvin vähän kokemusta. Työn toteuttamisen aikana teknologiat tulivat erittäin tutuiksi ja osaamiseni laajeni monella eri ohjelmistokehittämisen tasolla.

Osaamista kertyi myös projektin aikana tehdyistä virheistä, jotka nyt tekisin eri tavalla. Kaikkia järjestelmän osia ei olisi ollut tarpeellista toteuttaa alusta asti itse, vaan niissä olisi voinut hyödyntää valmiita työkaluja. Vaikka alusta asti itse tiettyjen toimintojen toteuttaminen oli oman oppimisen kannalta erittäin hyödyllistä, oli se yksittäisen projektin kannalta erittäin aikaa vievää. Toinen asia, jonka tekisin toisin vastaavanlaisen ohjelman kehittämisessä, on automaattisten testien kirjoittaminen. Ohjelman eri osien testaaminen kokeilemalla vei joissain tilanteissa paljon aikaa, sillä testit oli toistettava aina, kun ohjelmaan tehtiin muutoksia. Automatisoimalla testit, voidaan kaikki testit ajaa kerralla ja selvittää heti, jos jokin toiminto ei toimi halutulla tavalla.

8 Jatkokehitys

Järjestelmän jatkokehittäminen jatkuu kehittäjän itsensä toimesta, opinnäytetyön kuvaaman kehittämismenetelmän mukaisesti. Järjestelmään tullaan lisäämään yksi kerrallaan lisää ominaisuuksia ja lopulta sovelluksen testiversio tullaan ottamaan käyttöön kahden asiakas-taloyhtiön kanssa.

Käyttöliittymän ulkonäön tarkempaan suunnitteluun on tarkoitus palkata graafinen suunnittelija. Graafisen suunnittelijan tavoitteena on saada käyttöliittymästä mahdollisimman käyttäjäystävällinen sekä visuaalisesti näyttävä sovellus.

Lähteet

Painetut

Dooley, J. 2013. Software Development and Professional Practice. United States of America: Apress L. P.

Elliott, E. 2014. Programming JavaScript Applications. Sebastopol, California: O'Reilly Media.

Flanagan, D. 2011. JavaScript: The Definitive Guide. 6. painos. Sebastopol, California: O'Reilly Media.

Haikala, I. 2000. Ohjelmistotuotanto. 7. painos. Helsinki: Talentum.

Klimczak, E. 2013. Design For Software. Chichester, West Sussex: John Wiley & Sons Ltd.

Kruchten, P. 2003. The Rational Unified Process an introduction. Boston: Pearson education.

Sähköiset

About aws. 2018. Viitattu 9.2.2018. https://aws.amazon.com/about-aws/

Allam E., Pollack G. Codeschool Real-time-web with Node. Viitattu 10.2.2018. http://courseware.codeschool.com/real-time-web-with-node-js/all-levels.pdf

Amazon Web Service -pilvipalvelut. Viitattu 9.2.2018. https://solinor.fi/amazon-web-services-pilvipalvelut/

Compare Docs vs Relational. 2018. Viitattu 5.3.2018. https://developer.couchbase.com/documentation/server/3.x/developer/dev-guide-3.0/compare-docs-vs-relational.html

Expressjs. 2017. Viitattu 6.3.2018. https://expressjs.com/

Getpostman. Viitattu 9.5.2018. https://www.getpostman.com/

Git-scm. Viitattu 9.5.2018. https://git-scm.com/about

Koski, J. Ketterät menetelmät, agile LEAN ja scrum. Viitatttu 9.6.2018. https://www.itewiki.fi/opas/ketterat-menetelmat-agile-lean-ja-scrum/

Luukkainen M. & Laine H. 2012. Helsingin yliopisto. Luentomoniste kurssille Ohjelmistotekniikan menetelmät. Viitattu 22.12.2017. https://www.cs.helsinki.fi/u/mluukkai/otm2012/otm.pdf

Luukkainen M. 2017. Helsingin yliopisto. Luentomoniste kurssille Ohjelmistotuotanto. Viitattu 11.5.2018. https://github.com/mluukkai/ohjelmistotuotanto2017/blob/master/kalvot/luento1.pdf?raw=true

Luukkainen M. 2018. Github. Ohjelmistotekniikan menetelmät kevät 2018. Viitattu 11.5.2018. https://github.com/mluukkai/otm-2018/blob/master/web/materiaali.md

Mlab. Viitattu 9.5.2018. https://mlab.com/

Package management. 2017. Viitattu 5.3.2018. https://en.opensuse.org/Package_management

Passportjs. Viitattu 6.3.2018. http://www.passportjs.org/

Reactjs. 2018. Viitattu 24.4.2018. https://reactjs.org/

Sininen meteoriitti. Ketteryys haltuun: Yleisimmät ketterät käytännöt.2013. Viitattu 9.6.2018. https://www.meteoriitti.com/2013/06/06/ketteryys-haltuun-yleisimmat-ketterat-kaytannot/

What is npm. 31.01.2018. Viitattu 5.3.2018. https://docs.npmjs.com/getting-started/what-is-npm

Taulukot

Kuviot

Liitteet

Liite 1: Vaatimusmäärittely

Termistö

Roolit:

Pääkäyttäjä = me

Ylläpitäjä = isännöitsijä tai joku hänen valtuuttamansa assistenssi tms

Käyttäjä = huoneiston asukas

Osakas = huoneiston omistaja

Hallituksen jäsen = taloyhtiön hallituksen jäsen (on myös osakas)

Palvelutuottaja = huoltoliike tms.

Samalla *käyttäjätunnuksella* voi olla useita rooleja.

Objektit:

Tapahtuma = pihatalkoot, yhtiökokous,

Tiedote = isännöitsijän lähettämä "virallinen" tiedote

Keskustelu = käyttäjien välinen keskustelualueella tapahtuva keskustelu?

Varaus = saunavuoro, pyykkitupa, autopaikka, yhteiset tilat

Tehtävä = suunniteltu toimenpide

Dokumentti = taloyhtiön pöytäkirja, vastuunjakotaulukko, pelastussuunnitelma jne. Mitä tahansa mitä voisi olla taloyhtiön ilmoitustaululla.

Vastuunjakotaulukko = osakkaan ja taloyhtiön kunnossapitovastuiden määrittävä

Toiminnalliset vaatimukset:

- Taloyhtiöllä on oma suljettu yhteisö järjestelmässä.
- Pääkäyttäjä on kaikkien taloyhtiöryhmien ylläpitäjänä.
- Pääkäyttäjä voi luoda uuden ryhmän järjestelmään.
- Uuden ryhmän luomisen yhteydessä järjestelmä lisää pääkäyttäjille oikeudet ryhmään.
- Ylläpitäjät ja käyttäjät luodaan järjestelmään ylläpitäjän toimesta (vain ryhmään, jonka ylläpitäjänä ylläpitäjä toimii).
- Käyttäjille luodaan yksilölliset käyttäjätunnukset ja salasanat (note: kt *oletuksena sähköposti. Jos käyttäjällä ei ole sähköpostia, tulee ylläpitäjän päästä käsiksi käyttäjän salasanaan*)
- Järjestelmä luo käyttäjälle salasanan käyttäjän luonnin yhteydessä.
- Ryhmää poistettaessa järjestelmä poistaa jokaiselta käyttäjältä oikeudet ryhmään.

- Ylläpitäjät voivat:
    - Toimia ryhmän käyttäjänä
    - Hallinnoida ryhmän tietoja.
    - Lisätä palvelutuottajan taloyhtiölle.
    - Lisätä isännöitsijän taloyhtiölle.
    - Lisätä ja poistaa käyttäjiä (kaikkia rooleja) omassa taloyhtiö-yhteisössään.
    - Tarkastella asukkaiden varauksia.

- tehdä kausivarauksia.
- Hallinnoida tapahtumia.
- Hallinnoida tehtäviä.
- Käyttäjien tuottaman sisällön hallinnointi.
- Ilmiantojen vastaanottaminen.
- Lähettää tiedotteita (talous- tai henkilökohtaisia sekä kaikille jäsenille).
- Tarkastella ketkä ovat nähneet tiedotteen.
- Seurata ryhmän vikailmoitusten tilannetta.

- Käyttäjät voivat:
  - Selata ja lisätä tapahtumia kalenterissa.
  - Poistaa omia varauksiaan kalenterista.
  - Lisätä keskustelualueita.
  - Kommentoida keskustelualueita.
  - Ilmiantaa asiattomia kommentteja tai keskustelualueita.
  - Tehdä vikailmoituksen.
  - Seurata omien vikailmoitusten tilannetta.
  - Vastaanottaa tiedotteita.
  - Vastata kohdennettuihin tiedotteisiin.
  - Arvostella keskustelun, kommentin tai kyselyn.
  - kuulua useampaan ryhmään kerralla (kirjautuessa valittava, kumpaa ryhmää tahtoo tarkastella).
  - Muuttaa oman salasanansa.
  - Muuttaa puhelinnumeronsa.
  - Antaa palautetta fiilismittarilla.
  - Arvostella suoritetun vikailmoituksen.
  - Tarkastella dokumentteja.

- Osakkaat voivat:

  - Tarkastella tiedotteita.
  - Tarkastella tapahtumia.
  - Tarkastella tehtäviä.
  - Tarkastella dokumentteja.
  - Osallistua keskusteluihin.
  - Tarkastella omista huoneistoista jätettyjä vikailmoituksia?

- Hallituksen jäsenet voivat tehdä kaikkea mitä osakkaat voivat, sekä lisäksi:

  - Hallinnoida tapahtumia.
  - Hallinnoida tehtäviä?
  - Hallinnoida dokumentteja.
  - Hallinnoida tiedotteita.
  - Tarkastella palvelutuottajista annettua palautetta.
  - ~~Seurata/hallinnoida vikailmoitusten tilannetta.~~

- Isännöitsijä (ylläpitäjän oikeudet):
  - Tarkastella yleiskatsausta taloyhtiöistä, joissa toimii isännöitsijänä.
  - Tarkastella tapahtumia.
  - Hallinnoida tehtäviä.
  - Hallinnoida vikailmoituksia.
  - Hallinnoida dokumentteja.

Isännöitsijä ei pääse keskustelualueelle.

- Palvelutuottaja voi:
    - Vastaanottaa vikailmoituksia asiakastaloyhtiöiltään.
    - Vastaanottaa ilmoituksen saapuneesta vikailmoituksesta sähköpostiin.
    - Ilmoittaa vikailmoituksen statuksen.
    - Välittää palvelupyynnön.
    - Tehdä palvelupyynnön/vikailmoituksen.
    - Saada raportin kaikista kohteen palvelupyynnöistä.
    - Saada raportin kaikista taloyhtiöistä, joissa huolto on osallisena.
    - Saada käyttäjältä arvostelun suoritetusta huollosta.
    - Vastaanottaa muistutuksen lähestyvästä tehtävästä.
    - Kuitata tehtävän suoritetuksi.
    - Saada käyttäjältä tai isännöitsijältä arvostelun suoritetusta tehtävästä.

Ei-toiminnalliset vaatimukset:

- Tapahtumat ja tehtävät esitetään sekä luettelona että kalenterinäkymässä.
- Kalenterinäkymän sisältö riippuu roolista.
- Pääkäyttäjä, Ylläpitäjä ja Osakas -rooleilla voi tarkastella kalenterinäkymässä yhden tai useiden taloyhtiöiden sisältöä.

Liite 2: Käyttötapaushahmotelmat

· Pääkäyttäjä lisää taloyhtiön järjestelmään

· Pääkäyttäjä luo ylläpitäjän taloyhtiölle

· Pääkäyttäjä poistaa taloyhtiön järjestelmästä

· Pääkäyttäjä poistaa ylläpitäjän taloyhtiöstä

· Ylläpitäjä luo uuden käyttäjän taloyhtiöryhmäänsä.

· Ylläpitäjä poistaa käyttäjän ryhmästä

· Käyttäjä luo tunnukset järjestelmään

· Käyttäjä muuttaa salasanansa

· Käyttäjä tarkastaa uudet tiedotteet

· Käyttäjä selaa kalenteria

· Käyttäjä lisää tapahtuman kalenteriin

· Käyttäjä luo uuden keskustelun

· Käyttäjä poistaa oman keskustelunsa

· Ylläpitäjä poistaa/piilottaa keskustelun

· Käyttäjä luo kommentin keskusteluun

· ääjä poistaa/piilottaa kommentin

· Käyttäjä julkaisee pöytäkirjan

· Käyttäjä tarkastelee pöytäkirjaa

· Käyttäjä luo vikailmoituksen

· Ylläpitäjä/isännöitsijä/huolto kuittaa vikailmoituksen

· Käyttäjä tarkastelee varauskalenteria

· Käyttäjä tarkastaa oman taloutensa varaukset

· Käyttäjä poistaa oman varauksensa

Liite 3: Käyttötapausmallit

Taloyhtiön lisääminen järjestelmään

· Käyttäjät: Pääkäyttäjä

· Tavoite: Luoda uusi suljettu ryhmä taloyhtiölle

· Laukaisija: Uusi asiakkuus

· Esiehto: Taloyhtiöllä ei ole ennestään ryhmää järjestelmässä

· Jälkiehto: Taloyhtiöllä on oma instanssi järjestelmässä

· Käyttötapauksen kulku:

1. Pääkäyttäjä kirjautuu järjestelmään *
2. Pääkäyttäjä aloittaa lisäämistoiminnon
3. Pääkäyttäjä syöttää uuden taloyhtiön tiedot järjestelmään

· Poikkeuksellinen toiminta: Taloyhtiölle on jo luotuna asiakkuus

Ylläpitäjän lisääminen taloyhtiölle

· Käyitäjät: Pääkäyttäjä

· Tavoite: Luoda ryhmälle ylläpitäjä(t)

· Laukaisija: Luodaan uusi taloyhtiöryhmä, jolle tarvitaan ylläpitäjä, ylläpitäjä vaihtuu tai muuttuu.

· Esiehto: Luodaan uutta taloyhtiöryhmää tai muokataan nykyistä. Ylläpitäjä on valittuna taloyhtiön puolesta.

· Jälkiehto: Taloyhtiöllä on vähintään yksi ylläpitäjä.

· Käyttötapauksen kulku:

1. Pääkäyttäjä luo ylläpitäjän-oikeudet henkilölle taloyhtiöryhmään

· Poikkeuksellinen toiminta: Taloyhtiöryhmää ei ole olemassa. Ylläpitäjän tiedot eivät ole oikein.

Taloyhtiön poistaminen järjestelmästä

· Käyttäjät: Pääkäyttäjä

· Tavoite: Poistaa ryhmä-instanssi ja siihen kuuluvat jäsenet järjestelmästä

· Laukaisija: Asiakkuuden päättyminen / Viallinen asiakkuus.

· Esiehto: Taloyhtiö on järjestelmässä

· Jälkiehto: Taloyhtiö ei ole järjestelmässä

· Käyttötapauksen kulku:

1. Pääkäyttäjä etsii kyseisen asiakkuuden
2. Pääkäyttäjä poistaa asiakkuuden

· Poikkeuksellinen toiminta: Taloyhtiö ei ole järjestelmässä.

Ylläpitäjän poistaminen ryhmästä

· Käyttäjät: Pääkäyttäjä

· Tavoite: Poistaa ylläpitäjä ryhmästä tai muuttaa ylläpitäjä käyttäjäksi

· Laukaisija: Ryhmän ylläpitäjä poistuu tai vaihtuu

· Esiehto: Ryhmällä on ylläpitäjä

· Jälkiehto: Ryhmän ylläpitäjällä ei ole enää ylläpitäjän oikeuksia tai hän ei ole enää ryhmässä ollenkaan

· Käyttötapauksen kulku:

1. Pääkäyttäjä etsii ryhmän ylläpitäjä(n/t)
2. Pääkäyttäjä poistaa ylläpitäjän (tai muuttaa oikeuksia)

· Poikkeuksellinen toiminta: Ryhmällä ei ole ylläpitäjää. Ryhmällä ei ole ylläpitää toiminnon suorittamisen jälkeen.

Uuden käyttäjän luominen

· Käyttäjät: Ylläpitäjä, pääkäyttäjä

· Tavoite: Uuden käyttäjän lisääminen ryhmään.

· Laukaisija: Taloyhtiössä asukas, joka ei ole käyttäjänä ryhmässä

· Esiehto: Käyttäjää ei ole luotu kyseiseen ryhmään.

· Jälkiehto: Käyttäjä on luotu järjestelmään (ja liitetty ainakin yhteen ryhmään)

· Käyttötapauksen kulku:

1. Ylläpitäjä tai pääkäyttäjä valitsee kyseisen ryhmän, johon käyttäjä lisätään
2. Syötetään käyttäjän oot
3. Lisätään käyttäjä

· Poikkeuksellinen toiminta: Käyttäjä on jo luotu.

Käyttäjän poistaminen

· Käyttäjät: Ylläpitäjä, pääkäyttäjä

· Tavoite: Käyttäjän poistaminen ryhmästä

· Laukaisija: Käyttäjä ei ole enää taloyhtiön jäsen

· Esiehto: Käyttäjä on luotu järjestelmään ja on kyseisen taloyhtiöryhmän jäsen

· Jälkiehto: Käyttäjä ei enää kuulu kyseiseen ryhmään/ei ole järjestelmässä

· Käyttötapauksen kulku:

1. Ylläpitäjä tai pääkäyttäjä valitsee kyseisen ryhmän johon käyttäjä kuuluu
2. Etsitään kyseinen käyttäjä
3. Poistetaan käyttäjä

· Poikkeuksellinen toiminta: Käyttäjä ei kuulu kyseiseen ryhmään. Käyttäjää ei ole luotuna järjestelmään.

Tunnusten luominen järjestelmään

· Käyttäjät: Käyttäjä

· Tavoite: Päästä jäseneksi oman taloyhtiönsä ryhmään

· Laukaisija: Muutto taloyhtiöön. Taloyhtiön ottaessa järjestelmä käyttöön.

· Esiehto: Taloyhtiöllä on luotuna ryhmä järjestelmässä. Käyttäjä on saanut kutsun ryhmään sähköpostilla.

· Jälkiehto: Käyttäjällä on kirjautumistunnukset ja käyttöoikeudet järjestelmään

· Käyttötapauksen kulku:

1. Käyttäjä saa sähköpostilla tunnukset (tai linkin, josta voi luoda tunnukset).
2. Käyttäjä täyttää omat tietonsa ja kirjautuu järjestelmään.

· Poikkeuksellinen toiminta: Käyttäjä syöttää tietonsa väärin.

Salasanan vaihtaminen

· Käyttäjät: Käyttäjä

· Tavoite: Muuttaa oman käyttäjätunnustensa salasana

· Laukaisija: Tarve muuttaa salasana / salasanan unohtuminen

· Esiehto: Käyttäjällä on voimassa olevat käyttäjätunnukset

· Jälkiehto: Käyttäjä asettaa uuden salasanan käyttäjätililleen

· Käyttötapauksen kulku:

1. Käyttäjä kirjautuu tililleen
2. Menee kohtaan omat tiedot
3. Asettaa vanhan salasanan

4. Asettaa uuden salasanan

tai

1. Käyttäjä syöttää käyttäjätunnuksensa **
2. Ilmoittaa unohtaneensa salasanansa
3. Saa sähköpostiinsa uuden salasanan / ohjeet vaihtamiseen

· Poikkeuksellinen toiminta: Käyttäjänimeä ei ole olemassa.

Uusien tiedotteiden tarkastaminen

· Käyttäjät: Käyttäjä, ylläpitäjä

· Tavoite: Tarkastaa ryhmässä tapahtuneet muutokset, joita käyttäjä ei ole vielä nähnyt

· Laukaisija: Käyttäjän tarve

· Esiehto: Ryhmässä on tapahtunut muutoksia, jotka johtavat tiedotteeseen.

· Jälkiehto: Käyttäjä on nähnyt tiedotteet eikä niistä enää ilmoiteta käyttäjälle.

· Käyttötapauksen kulku:

1. Käyttäjä siirtyy tiedotteet -kohtaan, jossa näkyy lukemattomien tiedotteiden lukumäärä
2. Käyttäjä selaa tiedotteita

· Poikkeuksellinen toiminta: Ei ole uusia tiedotteita tai ei ole tiedotteita

Kalenterin selaus

· Käyttäjät: Käyttäjä, ylläpitäjä

· Tavoite: Tarkastaa kalenterista tulevat tapahtumat

· Laukaisija: Käyttäjän tarve

· Esiehto:

· Jälkiehto:

· Käyttötapauksen kulku:

1. Käyttäjä avaa kalenterin
2. Valitseee oikean kuukauden
3. Tarkastelee tapahtumia

· Poikkeuksellinen toiminta:

Tapahtuman lisäys

· Käyttäjät: Käyttäjä, ylläpitäjä

· Tavoite: Lisätä kalenteriin tapahtuma

· Laukaisija: Tuleva yleinen tapahtuma

· Esiehto: Tapahtuma ajankohta ei ole vielä mennyt, eikä ole liian pitkällä tulevaisuudessa

· Jälkiehto: Ryhmään on lisätty yleinen tapahtuma, jonka kaikki ryhmän jäsenet pystyvät näkemään. (tiedotteet)

· Käyttötapauksen kulku:

1. Käyttäjä avaa kalenterin
2. Valitsee oikean kuukauden, päivän ja ajan
3. Täyttää tapahtuman tiedot
4. Vahvistaa tapahtuman

· Poikkeuksellinen toiminta: Vapaita aikoja ei ole kalenterissa haluttuun aikaan. (päällekkäisyydet)

Keskustelun luominen

· Käyttäjät: Käyttäjä, ylläpitäjä

· Tavoite: Luoda uusi ryhmän sisäinen keskusteluaihe

· Laukaisija: Tarve ilmoittaa, kysyä tai keskustella jostakin yleisellä tasolla.

· Esiehto:

· Jälkiehto: Uusi ryhmälle avoin keskustelualue on luotu johon ryhmän jäsenet voivat kommentoida ja äänestää

· Käyttötapauksen kulku:

1. Käyttäjä siirtyy keskustelut -osioon
2. Siirtyy kohtaan ”lisää keskustelu”
3. Määrittelee keskustelulle aiheen
4. Julkaisee uuden keskustelun

· Poikkeuksellinen toiminta: Käyttäjällä ei ole oikeutta luoda keskustelua. Keskusteluaihe on asiaton ja se on ilmoitettu.

Keskustelun poisto

· Käyttäjät: Käyttäjä, ylläpitäjä

· Tavoite: Poistaa keskustelu mahdollisine kommentteineen

· Laukaisija: Keskustelu ei ole enää ajankohtainen tai paikkaansa pitävä. Keskustelu on asiatonta.

· Esiehto: Jos poistaja on tavallinen käyttäjä, on keskustelun oltava hänen oma luomansa. Muuten hän voi vain ilmiantaa asiattoman keskustelun ylläpitäjälle.

· Jälkiehto: Keskustelu kommenteineen ei enää näy ryhmän jäsenille

· Käyttötapauksen kulku:

1. Käyttäjä siirtyy keskustelut -osioon
2. Valitsee poistettavan keskustelun
3. Poistaa/ilmiantaa keskustelun

Poikkeuksellinen toiminta: Käyttäjällä ei ole oikeutta poistaa keskustelua.

Kommentin luominen

· Käyttäjät: Käyttäjä, ylläpitäjä

· Tavoite: Jättää keskusteluun (tai muuhun) muille käyttäjille näkyvä viesti

· Laukaisija: Oman mielipiteen tai uuden näkökulman ilmoittaminen muille käyttäjille

· Esiehto: Keskustelualuetta ei ole suljettu. Objektia on mahdollista kommentoida.

· Jälkiehto: Haluttuun keskusteluun (tai muuhun) on jätetty muille ryhmän jäsenille julkinen kommentti.

· Käyttötapauksen kulku:

1. Käyttäjä valitsee kommentoitavan alueen
2. Kirjoittaa kommenttinsa
3. Julkaisee kommentin

· Poikkeuksellinen toiminta: Käyttäjä yrittää syöttää liian pitkää tai tyhjää kommenttia.

Kommentin poistaminen

· Käyttäjät: Ylläpitäjä

· Tavoite: Poistaa tai piilottaa asiaton kommentti

· Laukaisija: Käyttäjän luoma asiaton kommentti

· Esiehto: Kommentti on luotuna ja näkyvänä kaikille ryhmän jäsenille

· Jälkiehto: Kommentti on poistettu tai piilotettu muulta ryhmältä. Kommentin luojalle lähtee tiedote.

· Käyttötapauksen kulku:

1. Ylläpitäjä saa tiedon asiattomasta kommentista tai huomaa sen itse
2. Poistaa kommentin kokonaan tai piilottaa sen sisällön (ja kirjoittajan)

· Poikkeuksellinen toiminta: Kommentti on viestiketjun keskellä ja kerännyt jatkokommentteja. (piilottaminen)

Pöytäkirjan luominen

· Käyttäjät: Käyttäjä, Ylläpitäjä

· Tavoite: Saada pöytäkirja ryhmän jäsenille julkiseksi

· Laukaisija: Uusi pöytäkirja

· Esiehto: Saman nimistä pöytäkirjaa ei ole vielä julkaistu ryhmään.

· Jälkiehto: Pöytäkirja on ryhmän sisällä julkisesti selattavissa.

· Käyttötapauksen kulku:

1. Käyttäjä siirtyy kohtaan ”Pöytäkirjat”
2. Lisää uusi pöytäkirja
3. Antaa tarvittavat tiedot
4. Lataa tiedoston tai kirjoittaa suoraan formiin

· Poikkeuksellinen toiminta: Yritetään ladata tiedostoa, joka ei ole tuettu. Yritetään luoda pöytäkirjaa otsikolla, joka on jo olemassa ryhmän sisällä.

Pöytäkirjojen tarkasteleminen

· Käyttäjät: Käyttäjä, Ylläpitäjä

· Tavoite: Tarkastella uusia ja vanhoja pöytäkirjoja

· Laukaisija: Käyttäjän tarve

· Käyttötapauksen kulku:

1. Käyttäjä siirtyy kohtaan ”pöytäkirjat”
2. Valitsee haluamansa pöytäkirjan
3. Lataa tiedoston tai esikatselee selaimessa

· Poikkeuksellinen toiminta:

Vikailmoituksen luominen

· Käyttäjät: Käyttäjä, Ylläpitäjä

· Tavoite: Ilmoittaa yleisestä tai talouskohtaisesta viasta

· Laukaisija: Vika

· Esiehto: Taloyhtiöllä on käytössä vikailmoitus

· Jälkiehto: Huollolle/ylläpitäjälle/isännöintsijälle on ilmoitettu viasta.

· Käyttötapauksen kulku:

1. Käyttäjä siirtyy kohtaan vikailmoitukset
2. Täyttää vikailmoituksen
3. Lähettää vikailmoituksen
4. Saa vahvistuksen

· Poikkeuksellinen toiminta: Vikailmoitukset eivät ole käytössä

Vikailmoituksen kuittaaminen

· Käyttäjät: Ylläpitäjä, isännöitsijä

· Tavoite: Reagoida huollon tarpeeseen ja kuitata ilmoittajalle

· Laukaisija: Saapunut vikailmoitus

· Esiehto: Vikailmoitus on oikein täytetty

· Jälkiehto: Vikailmoitus on kuitattu ja ilmoittajalle on kerrottu miten toimitaan

· Käyttötapauksen kulku:

1. Ylläpitäjälle/isännöitsijälle on saapunut uusi vikailmoitus
2. Tarkastaa vikailmoituksen
3. Kuittaa asian ilmoittajalle

· Poikkeuksellinen toiminta:

Varauskalenterin selaaminen

· Käyttäjät: Käyttäjä, ylläpitäjä

· Tavoite: Varaustilanteen tarkastaminen

· Laukaisija: Tarve varata, peruuttaa tai tarkastaa oma sauna-/pyykkivuoro

· Käyttötapauksen kulku:

1. Käyttäjä avaa varauskalenterin
2. Valitsee haluamansa päivän kalenterista

Oman talouden varauksien tarkastaminen

· Käyttäjät: Käyttäjä, ylläpitäjä

· Tavoite: Omien sauna-/pyykkivuorojen tarkastaminen

· Laukaisija: Ilmoitus lähestyvästä omasta vuorosta, tarve tarkastaa

· Esiehto: Käyttäjällä tai hänen kanssaan samassa taloudessa asuvalla on tulevia varauksia varauskalenterissa

· Jälkiehto: Käyttäjä tietää varaustilanteensa

· Käyttötapauksen kulku:

1. Käyttäjä avaa varauskalenterin
2. Tarkastele omia varauksia

· Poikkeuksellinen toiminta: Käyttäjällä ei ole talouskohtaisia varauksia voimassa

Käyttäjä poistaa varauksensa

· Käyttäjät: Käyttäjä, Ylläpitäjä

· Tavoite: Perua oman sauna/pyykkivuoro

· Laukaisija: Tarve

· Esiehto: Käyttäjällä on varauksia varauskalenterissa

· Jälkiehto: Käyttäjän varaus on peruttu ja vapautettu muille käyttäjille varattavaksi

· Käyttötapauksen kulku:

1. Käyttäjä avaa varauskalenterin
2. Tarkastelee omia varauksiaan ”omat varaukset” -kohdasta tai varauskalenterista
3. Poista varaus

· Poikkeuksellinen toiminta: Käyttäjä yrittää peruuttaa vuoroaan liian myöhään

*Kaikissa tapauksissa, jossa käyttäjä on pääkäyttäjä, ylläpitäjä tai käyttäjä, käyttötapauksen kulun ensimmäinen vaihe on käyttäjän kirjautuminen palveluun.

** Paitsi tässä