Tampereen ammattikorkeakoulu

# Käyttäjähallinnan toteuttaminen full stack -verkkosovellukseen

Joona Torvinen

OPINNÄYTETYÖ
Toukokuu 2021

Tieto- ja viestintätekniikka
Ohjelmistotekniikka

**TIIVISTELMÄ**

Tampereen ammattikorkeakoulu
Tieto- ja viestintätekniikka
Ohjelmistotekniikka

TORVINEN, JOONA
Käyttäjähallinnan toteuttaminen full stack -verkkosovellukseen

Opinnäytetyö 61 sivua
Toukokuu 2021

---

Opinnäytetyössä suunniteltiin ja toteutettiin käyttäjähallinta opiskelijan omaan verkkosovellukseen. Sovellus sai alkunsa opiskelijan aiemmin suorittamalla projektikurssilla ja opiskelija on jatkanut sen kehittämistä itsenäisesti kurssin jälkeen. Sovelluksen kehityksessä on käytetty yleisiä ja hyviksi havaittuja teknologioita, joita on verkkosovellusten kehityksessä käytetty kansainvälisesti paljonkin.

Työssä toteutettiin sovellukseen käyttäjähallinta, joka kattaa oleellisimmat ja tyypillisimmät käyttäjähallintaan liittyvät toiminnot. Käyttäjä pystyy rekisteröimään käyttäjätilin sovellukseen käyttäen sähköpostiosoitettaan. Tämän jälkeen käyttäjä pystyy kirjautumaan sisään, kirjautumaan ulos, vaihtamaan sähköpostiosoitteensa, vaihtamaan salasanansa, palauttamaan käyttäjätilinsä unohtuneen salasanan ja poistamaan käyttäjätilinsä palvelusta.

Käyttäjähallinnassa huomioitiin sekä teknisiä vaatimuksia että pohdittiin vaatimuksia myös lain näkökulmasta. Opinnäytetyössä tunnistettiin myös parannuskohteita, kuten oman sähköpostipalvelimen käyttöönotto sovellusta varten. Opinnäytetyössä kerrytettiin arvokasta tietoa ja osaamista, jota tarvitaaqn sovelluskehityksessä.

---

Avainsanat: käyttäjähallinta, verkkosovellus, full stack

## ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in ICT Engineering
Software Engineering

TORVINEN, JOONA
Implementing User Management to a Full Stack Web Application

Bachelor's thesis 61 pages
May 2021

---

In this bachelor’s thesis a user management system was planned and implemented for the author’s own web application. The work on the application began as part of a course the author completed earlier and has continued working privately on the project since then. The application has been developed using technologies that have been tried and found to be feasible, which have also been used extensively around the world to develop web applications.

In this study a user management system was implemented, which comprises most relevant functionalities and features. User can register an account using their email address. After that the user can log in, log out, change their email address, change their password and delete their user account.

Technical requirements and requirements mandated by national and international laws were taken into consideration in this study. Some needs for improvement were recognized as well in this study, such as utilizing an own email server. Relevant and valuable information and skills were developed and acquired while conducting this study.

---

Key words: user management, web application, full stack

**SISÄLLYS**

**LYHENTEET**

| | |
|---|---|
| TAMK | Tampereen ammattikorkeakoulu |
| Framework | Ohjelmointikieltä varten kehitetty jatke tai lisäosa |
| MEVN | MongoDB, Express.js, Vue.js, Node.js |
| Vue.js | Frontend-kehitykseen suunniteltu Javascript-framework |
| MongoDB | Tietokantasovellus |
| SPA | Single Page Application |
| API | Application Programming Interface |
| SQL | Structered Query Language |
| NoSQL | Tietokantasovellusten perhe, jotka eivät pohjaudu SQL-kieleen tai periydy siitä |
| MVC | Model-View-Controller |
| GDPR | General Data Protection Regulation |
| npm | Node Package Manager |
| CORS | Cross-Origin Resource Sharing |
| XSS | Cross-Site Scripting |

# 1 JOHDANTO

Tässä opinnäytetyössä otettiin jatkokehitettäväksi opiskelijan oma harrastusprojekti, joka sai alkunsa opiskelijan aiemmin suorittamalla projektikurssilla. Tämän opinnäytetyön tarkoituksena oli käyttäjähallintajärjestelmän suunnittelu ja toteuttaminen tähän projektiin, sekä samalla yleisesti perehtyminen sovelluskehityksen eri vaiheisiin. Itse kohdeprojekti esitellään seuraavassa luvussa.

Nykyaikaiset internetistä löytyvät verkkosovellukset tarjoavat käyttäjille moninaisia palveluja, joita ihmiset käyttävät päivittäisessä arkielämässään. Tässä opinnäytetyössä käytettäessä termiä verkkosovellus, tarkoitetaan yleisesti ottaen mitä tahansa internetin yli käytettävää ohjelmistokokonaisuutta, joka mahdollistaa käyttäjän ja järjestelmän välisen molemmin suuntaisen vuorovaikutuksen. Erilaisia palveluita käyttäessään ihmiset tulevat tuottaneeksi tietoa, jota ei haluta jaettavan kenen tahansa kanssa. Tämän takia verkkosovellukset on suunniteltava ja toteutettava siten, että sen käyttäjät pystyvät toimimaan turvallisesti ja varmennetusti sovelluksessa ilman, että ulkopuoliset ja asiattomat henkilöt pääsevät lukemaan, muokkaamaan tai poistamaan näitä tietoja. Yleinen tapa varmentaa käyttäjien identiteetti ja rajata heidän oikeutensa tarkasti, on toteuttaa jonkinlainen rekisteröitymiseen ja kirjautumiseen perustuva järjestelmä.

Rekisteröidyttäessä johonkin verkkopalveluun tulee väistämättä tuotettua ihmisten henkilökohtaisia tietoja sisältävää dataa, jonka keräämistä, säilyttämistä, käsittelyä ja luovuttamista säätelevät erilaiset kansalliset ja kansainväliset lait. Tämän vuoksi opinnäytetyössä pohdittiin myös käyttäjähallintajärjestelmän lainmukaisuuteen liittyviä aspekteja, jotka olisi otettava huomioon, jos tämän harrastusprojektin kaltaista verkkosovellusta lähdettäisiin julkaisemaan yleiseen käyttöön.

# 2 PROJEKTI

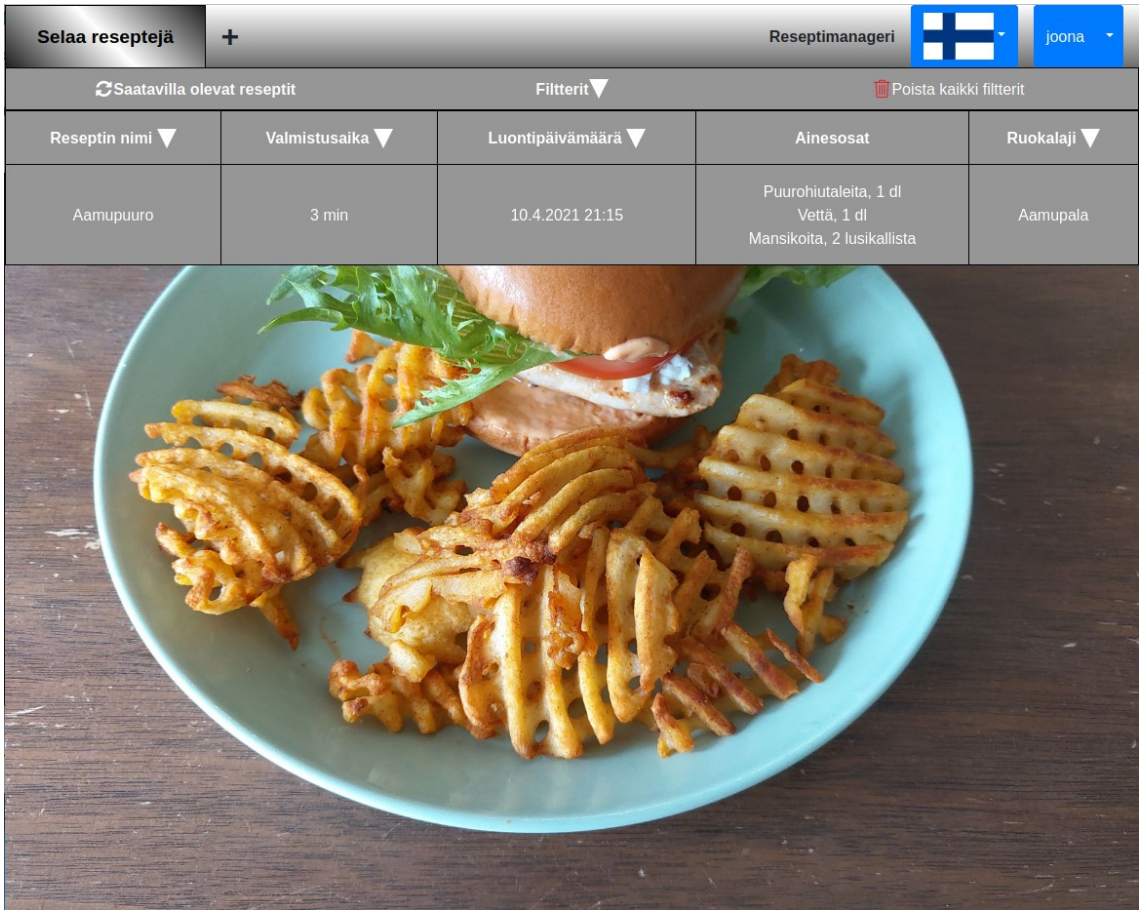
Harrastusprojekti on opiskelijan aiemmin suorittamalla projektikurssilla alkunsa saanut verkkosovellus, jota opiskelija on ollut alusta alkaen mukana toteuttamassa. Tämän sovelluksen kehittämistä jatkettiin itsenäisesti kurssin jälkeen omana harrastusprojektina ja sovelluksesta käytetään nimeä Reseptimanageri. Sovellus kehitettiin projektikurssin puitteissa toimivaksi kokonaisuudeksi, josta löytyi paljon ominaisuuksia, mutta josta kuitenkin puuttui kunnollinen käyttäjähallintajärjestelmä.

Sovelluksen jatkokehitystä ja käyttäjähallinnan ratkaisuja ohjasivat samat teknologiavalinnat, joiden avulla sovellusta on alusta kehitetty. Tässä luvussa kerrotaan ensin sovelluksen perusideasta, jonka jälkeen kerrotaan sovelluksen rakenteesta. Lopuksi esitellään lyhyesti projektissa käytettyjä teknologioita.

## 2.1 Projektin idea

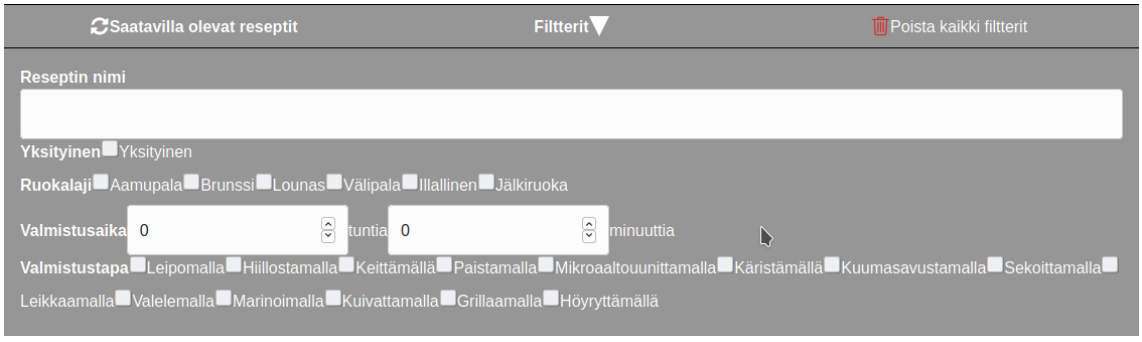
Reseptimanageri on ruokaohjeiden eli ruokareseptien luomiseen, järjestelemiseen, katselemiseen ja muokkaamiseen suunniteltu verkkosovellus, jonka tarkoituksena on toimia apuvälineenä ruoanlaiton yhteydessä. Ajatuksena on, että käyttäjä voi itse kirjoittaa ylös omia reseptejä sovelluksessa ja tallettaa niitä tietokantaan. Käyttäjä voi sitten tarvitessaan hakea niitä katseltaviksi ja käyttää niitä ohjeena ruoanlaitossa.

Reseptimanagerin oletusnäkymä on alla olevassa kuvassa 1 näkyvä Selaa reseptejä -näkymä, johon listataan taulukkonäkymään kyseisen käyttäjän omat reseptit. Taulukkonäkymän eri sarakkeet kertovat muutamia olennaisimpia tietoja resepteistä ja niitä on mahdollista järjestää tässä näkymässä nimen, valmistusajan, luontipäivämäärän ja ruoan tyypin mukaan.




KUVA 1. Reseptimanagerin Selaa reseptejä -näkymä

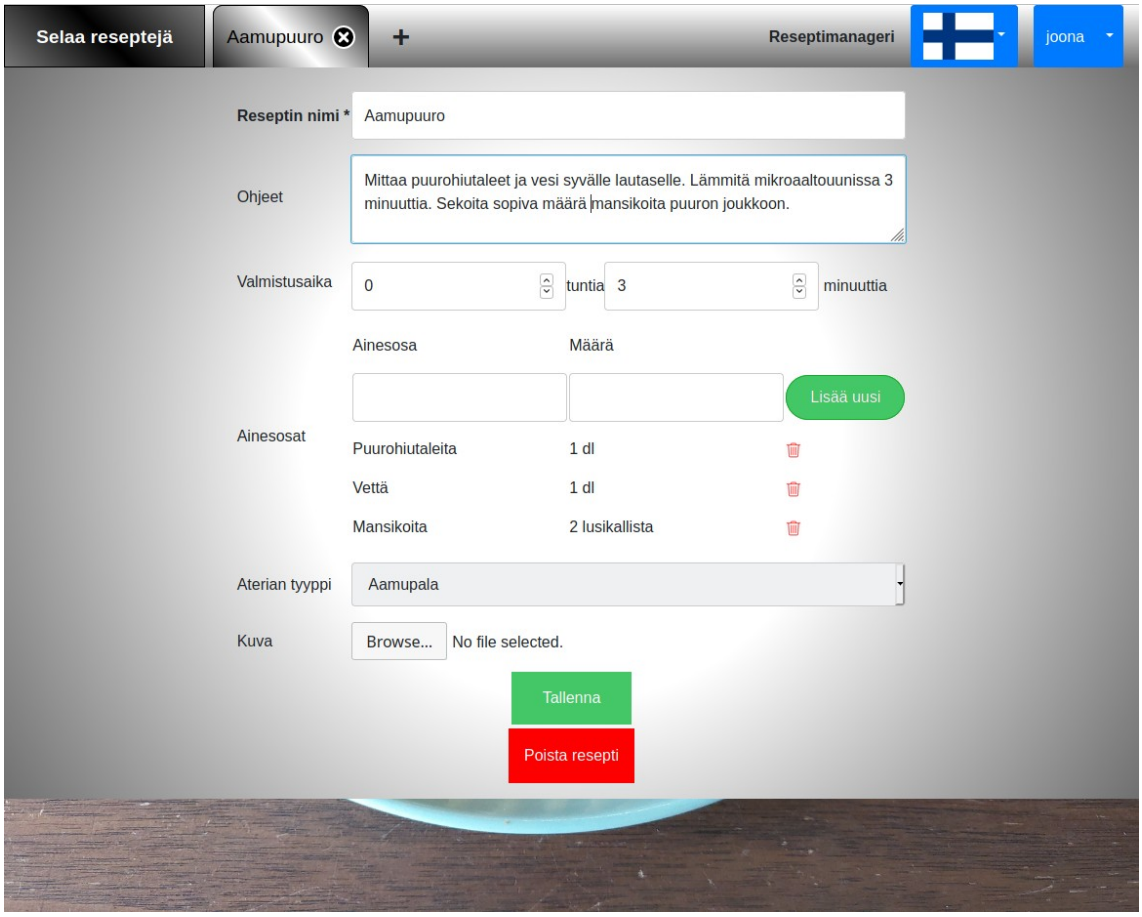
Oletuksena tässä näkymässä näytetään kaikki käyttäjän itseluomat ruokareseptit, mutta reseptejä on myös mahdollista suodattaa erilaisilla arvoilla. Kuvassa 2 on avattu suodatinvalikko ja siitä nähdään, että reseptejä on mahdollista suodattaa nimen, ruokalajin, valmistusajan sekä valmistustavan mukaan.




KUVA 2. Suodatinvalikko

Reseptin luominen tapahtuu painamalla käyttöliittymän ylärivistä löytyvää plus-merkkiä, jolloin uusi resepti aukeaa omaan välilehteensä. Kuvan 3 näkymässä käyttäjän tulee täyttää reseptin tiedot. Reseptin nimi on ainoa pakollinen tieto ja

muiden tietojen antaminen on vapaaehtoista. Muiden tietojen täyttäminen on kuitenkin hyödyllistä, sillä niiden perusteella on mahdollista hakea ja suodattaa reseptejä päänäkymässä.




KUVA 3. Yksittäinen ruokaresepti

Reseptiin on mahdollista liittää valokuva valmiista tuotoksesta, jonka voi myös selaimessa avata omaan välilehteensä tarkempaa tarkastelua varten. Valokuvia ei tallenneta tietokantaan vaan ne tallennetaan erikseen palvelimen levyjärjestelmään.

## 2.2 Kohdeprojektin rakenne

Reseptimanageri on niin kutsuttu full stack -verkkosovellus, sillä siinä on selkeästi eroteltavia kerroksia, jotka toimivat itsenäisesti ja joilla on omat tehtävänsä kokonaisuudessa. Termille full stack -sovellus ei ole yleisesti hyväksyttyä

määritelmää mutta vähimmillään sen yleisesti katsotaan koostuvan kahdesta sovelluskerroksesta, front endistä ja back endistä (Doshi 2019).

Front endillä tarkoitetaan sitä kerrosta, joka pitää sisällään sovelluksen käyttöliittymän ja jonka kanssa loppukäyttäjä on suorassa vuorovaikutuksessa. Front endiin katsotaan yleisesti kuuluvan ne toiminnallisuudet, jotka tuottavat käyttäjälle näkyvää grafiikkaa ja toiminnallisuuksia, ja joiden kautta käyttäjä pystyy käyttämään sovellusta. (Doshi 2019)

Back end tarkoittaa sovelluksen kerrosta, joka vastaa taustalla käyttäjältä näkymättömissä suoritettavista toiminnoista eli tyypillisesti käsittelee dataa. Se kommunikoi front endin kanssa, kuuntelee tulevia yhteyksiä ja vastaa pyyntöihin. Hyvin usein kuitenkin full stack -sovelluksesta puhuttaessa back end jaetaan vielä kahteen erilliseen kerrokseen, varsinaiseen back endiin ja tietokantaan. Tietokanta on sovelluskerros, johon sovelluksen ajonaikana tuotettu tieto talletetaan pysyvästi ja josta sitä luetaan. Tietokannassa tietojen tulee säilyä silloinkin, kun sovellus ajetaan alas syystä tai toisesta.

Sovelluksen eri kerrokset voivat olla tai voivat olla olematta samalla fyysisellä tai virtuaalisella laitteella. Kerroksia voi olla myös näitä enemmän eivätkä niiden rajavedot ole aina välttämättä näin selkeitä. Monesti front end ja back end -termien ohella käytetään termejä client ja server. Näiden termien erot ovat häilyvät ja enimmäkseen niillä tarkoitetaankin samoja asioita. Client ja server ovat perusteltuja nimityksiä silloin, kun halutaan erottaa paikallisesti käyttäjän omalla koneella suoritettava koodi sekä palveluntarjoajan päässä erillisellä tietokoneella eli palvelimella suoritettava koodi. (Software Engineering Stack Exchange 2013) Reseptimanageri toimii kolmella edellä mainitulla selkeästi eroteltavissa olevalla sovelluskerroksella, joista käytetään tässä opinnäytetyöstä nimitystä client, palvelin ja tietokanta.

Reseptimanagerin client noudattaa single-page application (SPA) -filosofiaa eli se muodostaa käyttäjälle tarvittavat näkymät ja dynaamisesti muokkaa niitä käyttäjän toimintojen mukaisesti. Client ladataan kokonaisuudessaan kerralla, kun käyttäjä avaa SPA-sovelluksen, eikä näkymää tämän jälkeen käytännössä enää päivitetä. Tällöin clientin ja palvelimen välinen tietoliikenne voidaan typis-

tää olennaisen datan välittämiseen, jolloin käyttäjän kokee sovelluksen käyttämisen jouheammaksi. Tällöin myös palvelimen kuormitus pienenee ja se voi keskittyä clientilta saatavien kutsujen suorittamiseen, jotka käytännössä useimmiten ovat tietokantaan tehtävien muutosten toteuttamista.

## 2.3 Kohdeprojektin teknologiat

Reseptimanagerin toteutukseen valittiin MEVN-stackinä tunnettu ohjelmointikielistä ja frameworkeistä koostuva sovelluspino, joka on maailmalla yleisesti käytetty sovelluspino verkkopalveluiden kehityksessä (educative.io n.d.). MEVN on akronyymi käytettävien teknologioiden nimistä, jotka ovat MongoDB, Express.js, Vue.js ja Node.js. Näiden pääteknologioiden lisäksi esille on nostettu vielä erikseen Noden kirjasto nimeltään Mongoose, joka toimii eräänlaisena ajurina MongoDB:tä varten. Tämä on erityisen tärkeä kirjasto, sillä sen avulla luodaan käyttäjähallinnan rakenne tietokannassa. Alla käydään kukin teknologia hyvin lyhyesti läpi. Tarkemmin niihin ei tämän opinnäytetyön raameissa ole tarpeen mennä.

Nämä teknologiat valittiin muun muassa sen vuoksi, että tämän pinon jokaisella tasolla käytetään JavaScriptiä jossakin muodossa, mikä helpottaa MVC-mallin (model-view-controller) laajentamisen front endistä koko pinoon. Tämä tuo kirjoitettuun koodiin selkeyttä, yksinkertaisuutta ja tehokkuutta, koska tietoa ei juurikaan tarvitse järjestää uudelleen sen matkatessa pinon lävitse kumpaan suuntaan tahansa. MVC-mallilla tarkoitetaan sovelluksen mallia, jossa käyttöliittymän toiminta jaetaan kolmeen selkeästi eroteltavaan osaan. Saket Kumarin (2018) mukaan mallin etuina on muun muassa se, että loogiset toiminnot voidaan ryhmittää keskenään samoihin tiedostoihin.

### 2.3.1 Vue

Vue.js, lyhyemmin Vue, on sovelluksen graafisen käyttöliittymän kehittämiseen tarkoitettu JavaScript-ohjelmointikielen framework, jota käytetään sovelluksen käyttöliittymän kehittämiseen. Vue valittiin käyttöliittymän suunnitteluun mm. sik-

si, että se on frameworkin kehittäjien mukaan helppo oppia. Se on heidän mukaansa myös kevyempi kuin esimerkiksi Angular, joka on toinen suosittu framework. (Vue n.d.) Vuen kehittäjän itsensä mukaan Vue eroaa muista monoliittisista frameworkeistä siten, että se on inkrementaalisesti adoptoitavissa olemassa olevaan ohjelmakoodiin ilman massiivisia refaktorointeja. (Vue n.d.)

### 2.3.2 Node

Node.js, lyhyemmin Node, on javascript-ohjelmointikieltä varten kirjoitettu runtime environment. Perinteisesti JavaScript-ohjelmakoodia on suoritettu selaimessa client-puolen sovelluksissa ja palvelinpuolen sovelluksia on tehty muilla ohjelmointikielillä. Node on palvelinsovellus, joka käyttää nimenomaan JavaScriptiä kielenään. Node valittiin, koska se integroituu hyvin yhteen front endin puolen JavaScript-koodiin. Se on myös erittäin suosittu ja siten siihen löytyy paljon dokumentaatiota. Geoffrey Mungai (2021) kirjoittaa section.io-sivustolla, että Node on erityisen hyvä hallitsemaan useita samanaikaisia yhteyksiä ja skaalautuu hyvin vastaamaan muuttuviin tarpeisiin.

### 2.3.3 Express

Express.js, lyhyemmin Express, on Nodea varten kehitetty framework, jonka tarkoituksena on tarjota kehittäjän palvelinsovellukselle toimiva rakenne. Mozillan mukaan se mahdollistaa myös front endin puolelta tutun MVC-mallin mukaisen arkkitehtuurin laajentamisen back endin puolelle sekä mahdollistaa integroitumiseen front endin renderoimistä suorittavien osien kanssa. Expressistä käsin on mahdollista syöttää tietoa suoraan niin kutsuttuihin templateihin, jotka client voi sellaisenaan esittää käyttäjälle. Express on Mozillan mukaan yleisin ja käytetyin framework Nodelle, johon löytyy siksi paljon dokumentaatiota ja laajennuskirjastoja. (Mozilla 2021)

### 2.3.4 MongoDB

MongoDB on tietokantasovellus, joka käyttää niin sanottua dokumenttiorientoitua tietomallia. Sen katsotaan kuuluvan niin kutsuttuun NoSQL-tietokantasovellusten perheeseen, joka ei pohjaudu perinteisempään relaatiotietokantamalliin. MongoDB:ssä tieto syötetään sovellukselle JSONin kaltaisina dokumentteihin ja sitä haetaan tekemällä JavaScript-ohjelmointikielen syntaksia muistuttavia kyselyjä. Tämä tekee kommunikaation MongoDB-tietokannan ja Node-palvelimen välillä erittäin helpoksi ja joustavaksi, koska kuljetettavan tiedon muotoa ei juurikaan tarvitse muuttaa. Se tuo sovellusten kehittämiseen joustavuutta ja kasvukyvykkyyttä, kun tietomallien ei tarvitse olla kiveen hakattuja etukäteen vaan ne voivat mukautua tarpeisiin paljon myöhemminkin. Reseptimanagerin tapauksessa tästä katsottiin olevan suurta hyötyä, koska ruokareseptien rakenteen ja tietomallin odotettiin voivan muuttua useaankin otteeseen kehityksen aloittamisen jälkeen. Tällöin kuvatun kaltainen joustavuus olisi suuri etu.

MongoDB:ssä yksittäinen tietokanta pitää sisällään collection-nimisiä tietorakenteita, jotka vastaavasti pitävät sisällään document-tietorakenteita. Tässä opinnäytetyössä, käytettäessä englannin kielen sanoja document ja collection, viitataan nimenomaan edellä mainittuihin MongoDB:n tietorakenteisiin eikä yleisesti dokumentteihin tai kokoelmiin.

### 2.3.5 Mongoose

MongoDB itsessään ei pakota sovelluskehittäjää käyttämään mitään erityistä rakennetta vaan ottaa sisäänsä lähes minkä muotoista ja rakenteista tietoa tahansa. Tämä on ennen kaikkea MongoDB:n vahvuus, mikä tekee siitä joustavan tietokantasovelluksen. Vastaan voi silti tulla tilanteita, joissa halutaan käytettävän tarkasti määriteltyjä tietorakenteita.

Tätä tarvetta varten Nodelle on kehitetty Mongoose-niminen kirjasto, joka on MongoDB:hen talletettavien tietojen rakenteiden mallintamista (modeling) varten tehty työkalu. Se suorittaa MongoDB:hen tehtävät kyselyt ja vastaa tiedon

kääntämisestä molempiin suuntiin oikeaan muotoon sekä talletettaessa validoi annetun tiedon.

Aiemmin kuvatut User- ja Session-tyyppiset oliot ovat nimenomaan Mongoosen Model-konstruktorin avulla rakennettuja olioita, jotka vastaavasti rakentuvat Mongoosen Schema-luokan instanssien pohjalta. Esimerkiksi User-tyyppiset oliot tallettuvat tietokannassa users-nimiseen collectioniin ja vastaavasti Session-tyyppiset oliot tallentuvat tietokannassa sessions-nimiseen collectioniin.

# 3 VAATIMUKSET

Käyttäjähallinnalla säädellään ja rajoitetaan verkkosovelluksessa sitä, kuka sovellusta pääsee käyttämään ja millä ehdoilla. Käyttäjähallinnan ensisijainen tarkoitus on turvata verkkopalvelun käyttö sen loppukäyttäjille mutta myöskin itse palveluntarjoajalle. Käyttäjähallinnan voidaan mieltää jakautuvan esimerkiksi seuraavaan neljään osaan: auktorisoitumiseen, autentikoitumiseen, administrointiin ja valvontaan.

Auktorisoitumisella tarkoitetaan sitä, kun käyttäjä rekisteröityy sovellukseen eli hänelle luodaan siihen oma identiteetti. Hänelle määrätään tietyt oikeudet käyttää sovellusta ja mahdollistetaan pääsy tiettyihin rajattuihin resursseihin.

Autentikoimisella tarkoitetaan sitä, että käyttäjä kirjautuu sisään sovellukseen ja hänen todennetaan olevan juuri se käyttäjä, kuka hän väittää olevansa. Sovelluksen varsinainen käyttäminen alkaa käytännössä aina autentikoitumisella mutta sen voidaan ajatella myös käsittävän jokaisen clientilta palvelimelle tapahtuvan pyynnön eli kutsun alussa tapahtuvaa istunnon todentamista.

Administroimisella eli hallinnoimisella tarkoitetaan sovelluksen ylläpitäjän toimesta tehtäviä hallinnollisia toimenpiteitä. Sovelluksen ylläpitäjä voi esimerkiksi sulkea käyttäjätilin ja evätä pääsyn sovellukseen käyttäjältä, joka menettelee toiminnallaan ehtojen vastaisesti. Ylläpitäjä voi myös ongelmatilanteissa auttaa käyttäjää esimerkiksi palauttamaan tämän käyttäjätunnuksen.

Valvonnalla tarkoitetaan ennen kaikkea automatisoidusti ohjelmallisesti tapahtuvia tarkkailutoimenpiteitä, joilla pyritään huomioimaan ja tuomaan ylläpitäjän tietoisuuteen selkeästi oletetusta poikkeava toiminta, jonka voidaan olettaa vaativan ylläpitäjältä toimenpiteitä. Tyypillisesti tällaista valvontaa suoritetaan tallentamalla erillisiin lokitiedostoihin vaikkapa rekisteröitymis- ja sisäänkirjautumistapahtumia ja niiden yrityksiä. Jos esimerkiksi jostakin tietystä ip-osoitteesta tulisi yhtäkkiä hyvin lyhyen ajan sisällä niin monta epäonnistunutta sisäänkirjautumisyritystä, ettei niiden voida ajatella olevan todellisen ihmisen tekemiä, voidaan ajatella kyseessä olevan automaattisesti suoritettava brute force -hyökkäys.

Brute force -hyökkäys tarkoittaa menetelmää, jossa hyökkääjä pyrkii murtamaan käyttäjätunnuksen ja salasanan käymällä hyvin nopeassa tahdissa lävitse valtavan määrän käyttäjätunnuksia ja salasanoja, kunnes lopulta arvaamalla osuu oikeaan yhdistelmään. Tällöin esimerkiksi valvonnallinen toimenpide voisi olla estää kyseinen IP-osoite palvelusta ja ilmoittaa tapahtuneesta ylläpitäjälle, joka voi ryhtyä asian tiimoilta jatkotoimenpiteisiin.

### 3.1 Toiminnalliset vaatimukset

Sovelluksessa on tiettyjä toimintoja, joita sen käyttäjien tulee voida tehdä. Heidän tulee voida käyttää palvelua ja luottaa, että heidän sekä suorasti luomansa että epäsuorasti tuottamansa tieto on turvassa eikä sitä jaeta muille kuin niille, keille sitä on tarkoituksenmukaista jakaa. Suoraan luodulla tiedolla tarkoitetaan Reseptimanagerin tapauksessa esimerkiksi käyttäjien luomia ruokareseptejä ja epäsuorasti tuotetuilla tiedoilla esimerkiksi sitä, koska ja mistä käyttäjä on kirjautunut sisään palveluun. Tiedolta edellytetään eheyttä, luottamuksellisuutta ja oikeellisuutta, mikä tarkoittaa Reseptimanagerin tapauksessa sitä, että käyttäjän luomia ruokareseptejä ei pysty lukemaan, muokkaamaan tai poistamaan kukaan muu kuin käyttäjä itse. Tarkoituksenmukaisena poikkeuksena tähän on se, että käyttäjä on tietoisesti tarkoittanut jakaa ruokareseptejään muiden käyttäjien kanssa. Käyttäjän tulee voida nähdä, mitä tietoja hänestä on tallennettu, kuten sähköpostiosoite, käyttäjätunnus ja rekisteröitymisaika. Käyttäjän tulee voida myös poistaa käyttäjätilinsä palvelusta ja täten tulla unohdetuksi.

Palveluntarjoajan eli verkkosovelluksen ylläpitäjän näkökulmasta käyttäjähallinnan tarkoitus on suojata palvelua ja taata sen saatavuus ja käytettävyys käyttäjille mahdollisimman pienin keskeytyksin ja haitoin. Palveluntarjoajat tyypillisesti toivovat verkkosovelluksiensa käyttäjiltä, että nämä eivät rekisteröisi useamman kuin yhden käyttäjätilin heidän palveluihinsa. Tätä toivotaan siksi, että jokainen käyttäjätili syö lisää resursseja palvelimelta ja käyttäjämäärien kasvaessa tämä voi muuttua hyvinkin merkittäväksi resurssikuluksi. Mikäli käyttäjien suorittamia rekisteröitymisiä ei rajoitettaisi mitenkään, voisi pahansuopa taho luoda uusia käyttäjätilejä sovellukseen automaattisesti niin kauan ja niin nopeasti, että sovellus hidastuisi merkittävästi tai jopa kaatuisi. Tällöin sovelluksen käyttö varsi-

naisilta käyttäjiltä estyisi. Tällöin kyseessä olisi yhdellä tavalla suoritettu palvelunestohyökkäys. Tyypillinen tapa vastata tähän uhkaan on sitoa yksi sähköpostiosoite yhtä käyttäjätiliä varten, jolloin useampien käyttäjätilien luominen hidastuu merkittävästi.

Ylläpitäjän näkökulmasta käyttäjähallinta mahdollistaa selkeän rajavedon eri käyttäjien välille. Se pitää yhden käyttäjän luoman ja tuottaman tiedon vain tämän itsensä käytettävissä ja nähtävissä ja eristää sen muiden käyttäjien ulottuvilta. Käyttäjähallinta mahdollistaa myös helpon ja tehokkaan tavan sulkea esimerkiksi ehtojen vastaisesti menettelevän käyttäjän käyttäjätili ja täten estää häntä käyttämästä sovellusta.

### 3.2 Lain näkökulma

Sekä Suomen laeissa että kansainvälisissä laeissa ja asetuksissa määritellään tarkasti ja yksiselitteisesti, mitä tietoja ihmisestä voi ja mitä ei voi kerätä, miten niitä voi kerätä ja miten niitä tulee säilyttää. Tietosuojavaltuutetun toimiston mukaan henkilötietojen käsittely vaatii aina laista löytyvää käsittelyperustetta. Lähtökohtaisesti henkilötietojen käsittelyyn tulisi saada suostumus henkilöltä itseltään, jonka henkilötietoja käsitellään. Laki määrittelee, mitä oikeuksia niin rekisteröidyllä kuin rekisterin ylläpitäjälläkin on. Rekisteröidyllä tarkoitetaan henkilöä, jonka henkilötietoja kerätään ja käsitellään, ja rekisterin ylläpitäjällä henkilöä tai tahoa, joka näitä henkilötietoja kerää ja käsittelee. Lähtökohtaisesti rekisterin ylläpitäjän tulee kerätä ja käsitellä vain niitä tietoja, joita on välttämätöntä kerätä, sekä ilmoittaa näistä toimenpiteistä rekisteröidylle selkeästi ja yksiselitteisesti. (Tietosuojavaltuutetun verkkosivut n.d.)

Tietosuojavaltuutetun toimisto määrittelee henkilötiedot seuraavalla tavalla: ”Henkilötietoja ovat kaikki tiedot, jotka liittyvät tunnistettuun tai tunnistettavissa olevaan henkilöön. Henkilötietoja ovat esimerkiksi nimi, puhelinnumero, sijaintitiedot ja isovanhempien perinnöllisiä sairauksia koskevat tiedot.” (Tietosuojavaltuutetun verkkosivut n.d.)

Keväällä 2018 otettiin Euroopan Unionissa käyttöön henkilötietojen käsittelyä sääntelevä laki, joka tunnetaan nimellä GDPR (General Data Protection Regulation). Se vahvistaa entisestään yksityishenkilöiden lain tuomaa suojaa ja antaa ihmisille enemmän oikeuksia puuttua henkilöstä itsestään kerättyjen tietojen käsittelyyn, säilyttämiseen ja poistamiseen. GDPR on kokonaisuudessaan nähtävissä omilla verkkosivuillaan ja siellä on saatavilla myös selkeät ohjeet tietosuojaselosteen laatimista varten. (GDPR 2019)

Reseptimanageri vaatii rekisteröitymisen yhteydessä käyttäjää antamaan voimassaolevan sähköpostiosoitteen. Sähköpostiosoitteen kerääminen katsottiin välttämättömäksi ja tarpeelliseksi henkilötiedoksi kerätä, koska se on yleisin ja mahdollisten käyttäjien kannalta tutuin tapa yhdistää luotava käyttäjätili johonkin olemassa olevaan tunnisteeseen. Sähköpostiosoite on siksi hyvä, että käyttäjät eivät tyypillisesti rekisteröi niitä kovinkaan monta ja käyttäjän on kohtalaisen hankala muuttaa tai vaihtaa se. Tämän tarkoituksena on suojella verkkopalvelun ylläpitäjää hallitsemattomilta rekisteröitymisiltä. Sähköpostiosoitteen keräämisen tarkoitus on myös tarjota käyttäjälle mahdollisuus palauttaa pääsy tämän käyttäjätilille, mikäli tämä unohtaisi esimerkiksi käyttäjätunnuksensa ja/tai salasanansa.

Reseptimanageri tallettaa myös käyttäjän IP-osoitteen sekä tämän rekisteröityessä palveluun että sisään kirjautuessa. IP-osoitteen tallentaminen katsottiin välttämättömäksi ja tarpeettomaksi, koska käyttäjän kirjautuessa sisään täytyy luoda istunto. Istunto tulee linkittää johonkin käyttäjän yksilöivään tunnisteeseen ja IP-osoite on otollinen tunniste tätä tarkoitusta varten. Vilpillisen aikein liikkeellä oleva taho voisi varastaa jonkin istunnon tiedot ja niitä käyttäen yrittää ottaa yhteys palveluun. Tällöin kuitenkin tämän yhteydenoton voi hylätä ja estää, mikäli yhteyden tunnistetaan tulevan toisesta IP-osoitteesta.

Näistä edellä mainituista syistä johtuen täytyy Reseptimanagerin tulkita keräävän ihmisistä yksilöiviä henkilötietoja. Tietosuoja-asetuksia ja lakeja noudattaakseen tässä opinnäytetyössä kirjoitettiin erillinen käyttöehdot ja tietosuoja -sivu, joka on avattavissa sovelluksesta. Lain mukaisesti henkilötietoperusteiden kerääminen vaatii lähtökohtaisesti rekisteröitävältä suostumuksen, mistä syystä

Reseptimanageriin rekisteröityminen edellyttää tämän sivun lukemista ja ehtojen hyväksymistä.

### 3.3 Työn rajaaminen

Tässä opinnäytetyössä keskityttiin sovelluksen käyttäjähallintajärjestelmän tekemiseen. Opinnäytetyön ulkopuolelle rajattiin ne korjaus- ja parantelutarpeet, jotka eivät oleellisesti liittyneet käyttäjähallintaan. Näistä asioista korjattiin ainoastaan ne, jotka koettiin selkeästi häiritseviksi tai toiminnallisuuksia rikkoviksi.

# 4 TOTEUTTAMINEN

Käyttäjähallintajärjestelmän toteuttamisessa lähdettiin painottamaan mahdollisimman suurta itse kirjoitetun koodin ja toiminnallisuuksien määrää ja osuutta koko työstä. Ehdoiksi päätettiin, että rekisteröityvien käyttäjien tietoja ei luovutettaisi kolmansille osapuolille, ja että tuotettujen tietojen hallinnointi ja säilyttäminen olisivat kehittäjän itsensä vastuulla. Tässä opinnäytetyössä toteutettiin joitakin ominaisuuksia, jotka ovat oleellisia vain julkisesti verkossa saatavilla oleville sovelluksille. Reseptimanageri ei tätä ole mutta näin tehtiin silti, jotta opittaisiin ja ymmärrettäisiin erilaisia sovelluskehitykseen kuuluvia vaiheita ja toimenpiteitä.

Henkilön liittyminen jonkin palvelun käyttäjäksi alkaa yleensä aina rekisteröitymisellä. Jotta käyttäjien rekisteröityminen johonkin verkkosovellukseen olisi palveluntarjoajan kannalta luotettava ja turvattu, täytyy jotenkin estää käyttäjiä hallitsemattomasti rekisteröimästä käyttäjätilejä. Mikäli tätä ei rajoitettaisi millään tavalla, olisi rekisteröitymistä mahdollista käyttää eräänlaisena palvelunestohyökkäyksenä, jossa tietokanta täytetään ja tukitaan käyttäjätileillä. Hyvin yleinen tapa varmentaa käyttäjän tunnistautuminen on yhdistää luotava käyttäjätili olemassa olevaan sähköpostiosoitteeseen, jolloin rekisteröityjä ei voisi luoda useita tilejä palveluun ainakaan samalla sähköpostiosoitteella. Mahdollisen hyökkäyksen kannalta tämä prosessi hidastaa ja hankaloittaa rekisteröitymisprosessia sen verran, ettei palvelunestohyökkäys sen kautta ole yhtä varteenotettavaa. Tämä mahdollistaa myös sen, että mikäli käyttäjän pääsy sovellukseen jouduttaisiin estämään esimerkiksi ehtojen vastaisen menettelyn vuoksi, ei tämä voisi ainakaan tällä sähköpostiosoitteella luoda uusia tunnuksia.

Yleisesti käytetty tapa on, että rekisteröitymisen yhteydessä käyttäjä antaa sähköpostiosoitteensa, johon palvelun toimesta lähetetään vahvistuskoodi. Käyttäjän täytyy käydä lukemassa tämä koodi ja palauttaa se sitten takaisin palvelulle jotakin reittiä, joko avaamalla sähköpostiin lähetetty linkki tai itse syöttämällä koodi käyttöliittymään. Näin saadaan varmistuttua, että kyseinen sähköpostiosoite on todella olemassa ja että rekisteröityjä hallinnoi sitä, jolloin se voidaan sitoa luotuun käyttäjätiliin. Samalla se toimii myös yhteyskanavana, jonka avulla

pääsy käyttäjätilille on mahdollista palauttaa, mikäli rekisteröityjä vaikkapa unohtaisi käyttäjätunnuksensa tai salasanansa. Satunnaisesti generoidun varmistuskoodin luominen sinänsä on helppo toimenpide mutta se tulee voida toimittaa rekisteröityjälle luotettavasti. Jotta rekisteröityjän sähköpostiin voitaisiin lähettää jonkinlainen vahvistuskoodi, tulisi sovelluksesta käsin voida lähettää sähköpostia, tai ainakin pistää alulle sen lähettäminen.

Ensimmäiseksi mietittiin oman sähköpostipalvelimen perustamista ja konfiguroimista vahvistuskoodien lähettämistä varten, mutta tämä vaihtoehto suljettiin nopeasti pois lyhyen selvitystyön jälkeen. Kävi ilmeiseksi, että se olisi laajuudessaan ja haastavuudessa oman opinnäytetyönsä veroinen ja vaatisi myös oman domainin eli verkkotunnuksen hankkimisen.

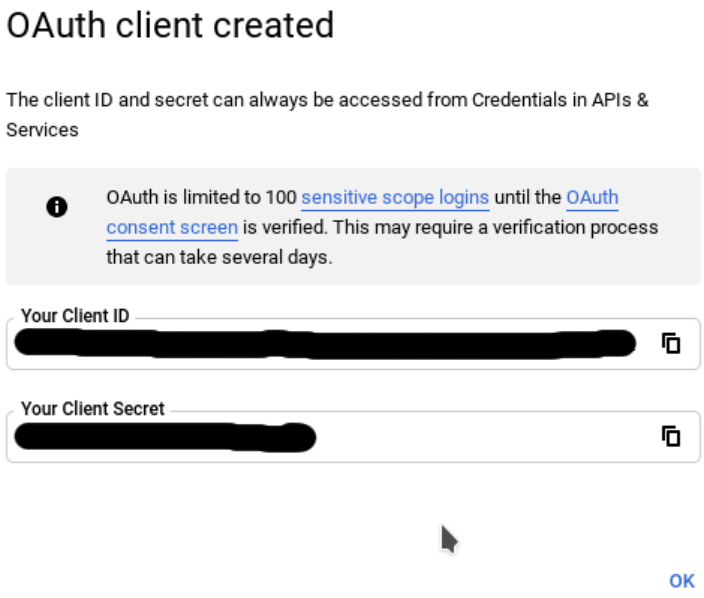
Asian tiimoilta tutkittiin kolmannen osapuolen ratkaisuja. Monet suuret yhtiöt, kuten Google, Microsoft ja Apple, tarjoavat erilaisia rajapintoja omiin sähköpostipalveluihinsa, joiden avulla sovelluskehittäjä voi ohjelmallisesti luoda yhteyden palveluun ja käyttää sitä. Tähän toteutukseen valittiin Google ja heidän Gmailiin tarjoamansa rajapinta, koska heidän rajapintansa käyttämisestä on olemassa paljon dokumentaatiota ja ohjeita internetissä.

### 4.1 Googlen rajapinta

Googlella on oma tuoteperheensä Google Cloud Platform, joka on erilaisista palveluista koostuva ympäristö, jossa sovelluskehittäjät voivat hyödyntää erilaisia Googlen tarjoamia resursseja, kuten esimerkiksi pilvessä tapahtuvaa laskentaa. Yksi sieltä löytyvistä palveluista on API & Services, joka tarjoaa erilaisia rajapintoja Googlen omiin palveluihin. Tässä opinnäytetyössä tarvittiin rajapintaa Gmailiin sähköpostiviestin lähettämistä varten.

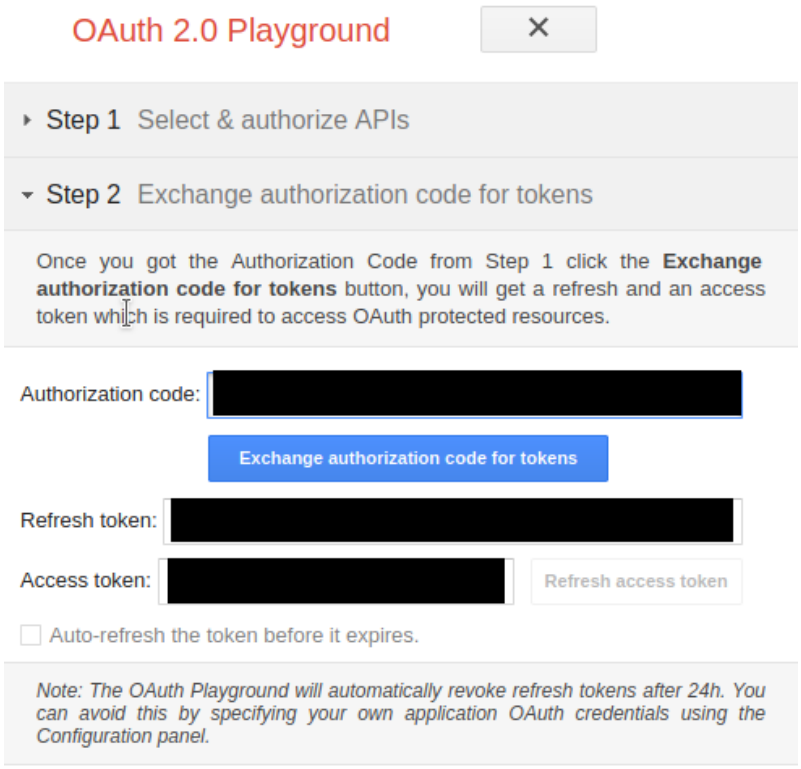
Googlen rajapinnan kokeileminen aloitettiin toteuttamalla lyhyt koeprojekti internetistä löydetyn esimerkin avulla (Chhetri 2020). Ensimmäiseksi rekisteröitiin uusi Google-tili koetta varten, mikä tarvittiin sähköpostiviestien lähettämiseen. Seuraavaksi sisäänkirjautuneena uudella Google-tilillä navigoitiin Google Cloud Platformille. Palvelun hallintapaneelissa luotiin uusi koeprojekti Reseptimanage-

ria varten. Seuraavaksi käytiin lävitse OAuth consent screen -vaihe, jossa annettiin Googlelle yhteyttä pyytävän sovelluksen nimi ja kehittäjän yhteystiedot tukea varten sekä lisättiin yksi testikäyttäjä käyttäen aiemmin rekisteröityä Gmail-osoitetta. Sitten luotiin uusi 0Auth client ID, jonka tyypiksi asetettiin Web Application. Lopuksi saatiin alla olevassa kuvassa 4 näkyvät yksilölliset tiedot, joita tultiin myöhemmin tarvitsemaan palvelimen puolella konfiguraatiossa.




KUVA 4. OAuth client kredentiaalit

Seuraavaksi suoritettiin Googlen OAuth Playgroundin käyttöönotto. Se on Googlen kehittäjille tarkoitettu palvelu, joka generoi käyttäjälle auktorisoitumista varten tokeneja. Näitä tokeneja käyttämällä kehittäjän sovellus kykenee lopulta kommunikoimaan Googlen rajapintojen kanssa ja suorittamaan haluttuja toimenpiteitä. Tässä kohtaa tarvittiin aiemmin luodut client kredentiaalit. Kun ne oli luovutettu OAuth Playgroundille ja valittu haluttu rajapinta, saatiin alla olevassa kuvassa 5 näkyvät refresh ja access tokenit.




KUVA 5. Refresh ja access tokenien luovutus

Access token on se tieto, jonka sovellus lopulta lähettää rajapinnalle halutessaan pääsyn johonkin resurssiin tai suorittaa jonkin toimenpiteen. Kun access token vanhenee, sovellus käyttää refresh tokenia hakemaan rajapinnasta uuden access tokenin. Access tokenit ovat huomattavasti lyhytikäisempiä kuin refresh tokenit. Tämä on OAuth 2.0 -standardin mukainen malli, jota myös OAuth Playground käyttää. (Google Developers, 2021)

Google Cloud Platformin hallintapaneeleista olisi ollut mahdollista tehdä tarkempia rajauksia niihin domaineihin, joista on luvallista käyttää Googlen rajapintoja. Tämä olisi tietoturvasyistä hyödyllistä ja tarpeellista tehdä tuotantoon siirretyn sovelluksen kohdalla. Tässä opinnäytetyössä näitä rajauksia ei katsottu tarpeelliseksi tehdä, kosk asovellus ei ole yleisesti saatavilla.

## 4.2 Nodemailer

Sähköpostiviestien lähettämistä varten asennettiin Nodemailer-niminen kirjasto, joka mahdollistaa sähköpostiviestien lähettämisen kätevästi Node-ohjelmasta

käsin. Se on ilmainen ja avoimen lähdekoodin kirjasto, jonka tehtävänä on käsitellä ja huolehtia tarvittavien lähetysprotokollien käyttäminen ja konfiguroiminen. Sen käyttäminen vähentää huomattavasti sen koodin määrä, jota kehittäjän täytyisi itse kirjoittaa hyödyntääkseen Googlen rajapintaa. Tässä projektissa Nodemaileria käytettiin luomaan SMTP-protokollalle oma Transport-olio, jolla sähköpostiviesti saadaan lähtemään eteenpäin Googlelle.

### 4.3 Sähköpostiviestien lähettäminen

Valittuja kirjastoja ja lähestymistapaa kokeiltiin luomalla testiprojekti sähköpostin lähettämisen toimivuuden testaamiseksi. Seuraamalla verkosta löydettyä Chandra Panta Chhetrin (2020) ohjeartikkelia luotiin alla olevassa kuvassa 6 näkyvä JavaScript-tiedosto index.js.

```
require("dotenv").config();
const nodemailer = require("nodemailer");
const { google } = require("googleapis");
const OAuth2 = google.auth.OAuth2;

const createTransporter = async () => {
    const oauth2Client = new OAuth2(
        process.env.CLIENT_ID,
        process.env.CLIENT_SECRET,
        "https://developers.google.com/oauthplayground"
    );

    oauth2Client.setCredentials({
        refresh_token: process.env.REFRESH_TOKEN
    });

    console.log(oauth2Client);

    const accessToken = await new Promise((resolve, reject) => {
        oauth2Client.getAccessToken((err, token) => {
            if (err) {
                reject("Failed to create access token :(");
            }
            resolve(token);
        });
    });

    const transporter = nodemailer.createTransport({
        service: "gmail",
        auth: {
            type: "OAuth2",
            user: process.env.EMAIL,
            accessToken,
            clientId: process.env.CLIENT_ID,
            clientSecret: process.env.CLIENT_SECRET,
            refreshToken: process.env.REFRESH_TOKEN
        }
    });

    return transporter;
};

const sendEmail = async (emailOptions) => {
    let emailTransporter = await createTransporter();
    await emailTransporter.sendMail(emailOptions);
};

sendEmail({
    subject: "Test",
    text: "This is a test message.",
    to: "target@email.com",
    from: process.env.EMAIL
});
```

KUVA 6. Sähköpostiviestin lähettämisen testauksessa käytetty koodi (Chhetri 2020)

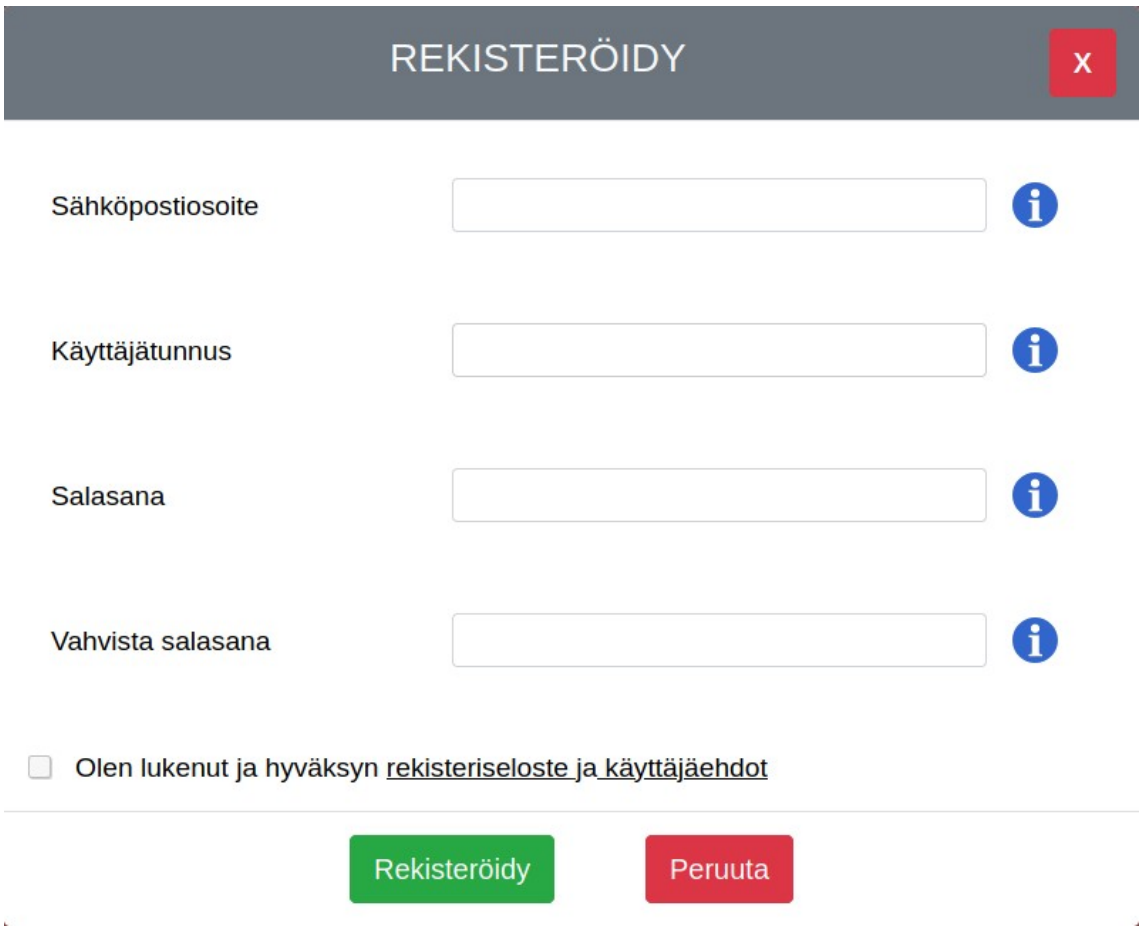
Esimerkissä näkyvän sendEmail-funktion target@email.com korvattiin kehittäjän omalla sähköpostiosoitteella, jonka jälkeen ohjelman toimintaa kokeiltiin. Tämä tapahtui navigoimalla komentoriviltä käsin kansioon, jossa testitiedosto oli, ja suorittamalla komento node index.js. Näin saatiin esimerkissä nähtävä viesti tulemaan kehittäjän omaan sähköpostiin ja voitiin todeta konfigurointien toimivan.

## 4.4 Käyttäjähallinnan toiminnot

### 4.4.1 Rekisteröityminen

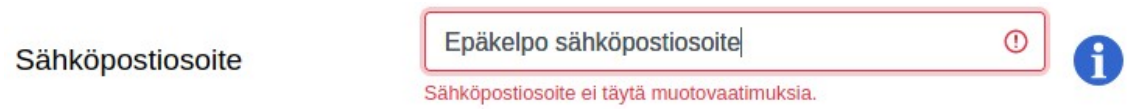
Sovelluksessa oli entuudestaan yksinkertainen rekisteröitymisnäkymä, jossa oli syötekentät vain käyttäjätunnukselle ja salasanalle. Käyttäjän näihin kenttiin antamia syötteitä validoitiin vain pituuksiensa puolesta. Sähköpostiosoitteita ei käytetty sovelluksessa millään tavalla. Tämä käytännössä takasi käyttäjälle mahdollisuuden luoda rajattomasti käyttäjätilejä sekä mahdollisti haitallisen sisällön syöttämisen.

Rekisteröitymisnäkymään lisättiin oma syötekenttä sähköpostiosoitteelle, salasanan vahvistukselle sekä valintaruutu, jonka rastittamalla käyttäjä vakuuttaa lukeneensa käyttöehdot ja tietosuojaselosteen. Käyttöehdoista ja tietosuojaselosteesta kerrotaan lisää myöhemmässä luvussa. Rekisteröitymisnäkymä muokattiin myös visuaalisesti esteettisemmäksi ja lopputulos tästä löytyy alla olevasta kuvasta 7.




KUVA 7. Rekisteröitymisikkuna

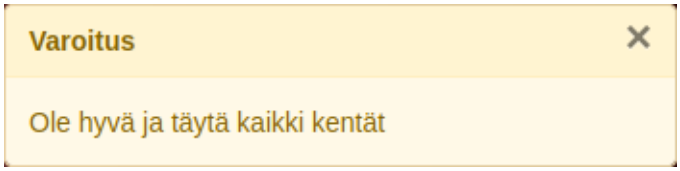
Kaikki käyttäjän antamat syötteet tarkastetaan clientin päässä ennen kuin mitään lähetetään palvelimelle. Yksittäisten syötekenttien arvot tarkastetaan reaaliaikaisesti käyttäjän kirjoittaessa ja ne värjäytyvät reunoiltaan punaiseksi indikoimaan virheellistä syötettä tai vastaavasti vihreäksi indikoimaan kelvollista syötettä. Jokaisen syötekentän arvoista tarkistetaan, etteivät ne ole tyhjiä, liian lyhyitä, liian pitkiä tai sisällä laittomia merkkejä. Lisäksi sähköpostiosoitteesta tarkistetaan vielä erikseen sen vaadittu muoto. Alla olevassa kuvassa 8 on esimerkki syötekentästä , joka on värjäytynyt punaiseksi epäkelvollisen syötteen johdosta.




KUVA 8. Epäkelvollisen syötteen virheilmoitus

Sovellus estää käyttäjää lähettämästä rekisteröitymispyynnön palvelimelle, mikäli kaikkien vaadittujen syötekenttien arvot eivät ole vaatimusta mukaisia. Täl-

löin käyttöliittymä tuo esille alla olevassa kuvassa 9 näkyvän Toast-tyyppisen ponnahdusikkunan ilmoittamaan havaituista puutteista.




KUVA 9. Toast-tyyppinen ilmoitus puuttuvista syötteistä

Hyväksytyt rekisteröitymistiedot paketoidaan avain-arvo-pareina JavaScript-olioksi, joka liitetään myöhemmin lähetettävän HTTP POST -kutsun bodyyn, josta esimerkki alla olevassa kuvassa 10. Kun POST-kutsussa lähetetään JSON-muotoista sisältöä, tulee se muuttaa string-muotoon JSON.stringify()-komennolla. Tällöin kutsun Content-Type-headerille tulee antaa arvo ”application/json”, jotta palvelin ymmärtää sisällön olevan string-muotoon muutettu JavaScript-olio. Samaa periaatetta noudatettiin sovelluksessa jokaisessa palvelimelle lähetettävässä kutsussa silloin, kun sen bodyyn tarvitsee laittaa sisältöä.

```
const body = {
    email: this.emailAddress,
    username: this.username,
    password: this.password,
    locale: this.$i18n.locale,
    agreementToTermsAndConditions: new Date(),
};
```

KUVA 10: Rekisteröitymisessä käytettävän HTTP POST-kutsun bodyksi asetettava olio

### 4.4.2 Rekisteröitymispyynnön vastaanottaminen

Palvelimen puolella kuunnellaan tulevia HTTP POST -kutsuja /api/auth/register-polun eli endpointin takana, josta rekisteröitymistä varten tehty funktio esimerkkinä alla olevassa kuva. Tämän jälkeen kutsu ohjataan vastaavalle controller-funktiolle, joka suorittaa halutut toimenpiteet.

```
// Register
server.post("/api/auth/register", rateLimiters.registrationLimiter, controller.register);
```

Kuva 11. Rekisteröitymistä kuunteleva

Lähetettävien kutsujen sisällön tarkastaminen clientin puolella ei poista sisällön tarkastamisen tärkeyttä ja tarpeellisuutta palvelimen puolella. Software Engineering StackExchange -sivustolla nimimerkillä JacquesB (2015) kirjoittavan henkilön mukaan client-puolen validaatio on lähinnä käyttökokemuksen parantamista, kun taas palvelinpuolen validaatio on ehdottoman tärkeätä. Rekisteröitymisen tapauksessa tarkastetaan ensimmäiseksi, löytyykö POST-kutsun bodysta tarvittavat avaimet username, email, password ja locale. Näiden avainten arvojen odotetaan olevan string-tyyppisiä, joten palvelin tarkastaa, etteivät ne ole null tai undefined ja että ne ovat string-tyyppisiä. Lopuksi tarkastetaan, etteivät ne sisällä laittomia merkkejä ja että niiden pituudet vastaavat määriteltyjä rajoja. Sähköpostiosoitteelle suoritetaan vielä lisäksi aiemmin kuvatun mukainen tarkempi muodon tarkastus.

Mikäli rekisteröitymispyynnön sisältö läpäisee tarkastuksen, niin ensimmäisenä varmistetaan, että annetulla käyttäjätunnuksella tai sähköpostiosoitteella ei ole jo olemassa vahvistettuja rekisteröitymisiä tai odottavia rekisteröitymispyyntöjä. Tämä tapahtuu suorittamalla kyselyt tietokannan pendings- ja users-collectioneihin. Mikäli yksikin osuma löydetään, rekisteröitymispyyntö hylätään ja käyttäjälle lähetetään asiasta ilmoitus. Jos mitään ei löydy, voidaan rekisteröitymistä jatkaa eteenpäin.

Tämän jälkeen generoidaan satunnainen 24 merkkiä pitkä string, jota tullaan käyttämään sähköpostiosoitteen vahvistuskoodina. Tässä sovelluksessa käytetään kaikissa salaukseen liittyvissä toiminnoissa Noden omaa Crypto-moduulia, joka sisältää työkaluja mm. tiivisteiden luomiseen ja halutun tiedon enkryptaamiseen. Salasanaa varten generoidaan ensin oma suola, jota käytetään tiivisteen eli hashin muodostamiseen salasanan pohjalta. Suola on vahvistuskoodin tapaan automaattisesti generoitu sattumanvarainen string, jonka oletuspituus on 24 merkkiä. Varsinaista selkokielistä salasanaa ei koskaan tallenneta mihinkään vaan tietokantaan tallennetaan tiiviste yhdessä suolan kanssa. Suola tarvitaan myöhemmin, jotta käyttäjän esimerkiksi sisäänkirjautumisen yhteydessä anta-

masta salasanasta luotua tiivistettä voidaan vertailla tietokantaan tallennettuun tiivisteeseen. Uusi tiiviste pitää luoda käyttäen vanhaa tiivistettä, jotta vertailu voidaan tehdä.

Näiden tietojen pohjalta luodaan uusi Pending-olio, joka tallennetaan tietokannan pendings-collectioniin, jossa se odottaa sähköpostiosoitteen vahvistamista. Nämä odottavat dokumentit haluttiin pitää omassa collectionissaan erillään vahvistetut käyttäjät sisältävästä users-collectionista, jotta esimerkiksi teknisten ongelmien tapauksissa olisi selkeämpää, missä vahvistamattomat rekisteröitymispyynnöt ovat. Alla olevassa kuvassa 12 on kuvakaappaus lähdekoodista, jossa näkyy uuden Pending-olion luominen.

```
// Create new registration request
const pending = new Pending({
    agreementToTermsAndConditions: new Date(),
    email: req.body.email,
    username: req.body.username,
    hash: hash,
    salt: salt,
    locale: req.body.locale,
    randomString: randomString,
});
```

KUVA 12. Pending-tyyppinen olio

Kun rekisteröitymispyyntö on tallennettu tietokantaan, ollaan valmiita lähettämään sähköpostia rekisteröityjälle. Tässä kohtaa tarvitaan avuksi aiemmin implementoitua sähköpostiviestien lähettämisen toimintoa. Aiemmin toteutetun koesovelluksen sisältämän koodin pystyi melko pienin muutoksin sisällyttämään varsinaiseen projektiin. Tärkeätä on, että Googlen APIn kanssa tarvittavia kredentiaaleja ei lisätä suoraan lähdekoodiin vaan ne luetaan esimerkiksi käyttöjärjestelmän ympäristömuuttujista.

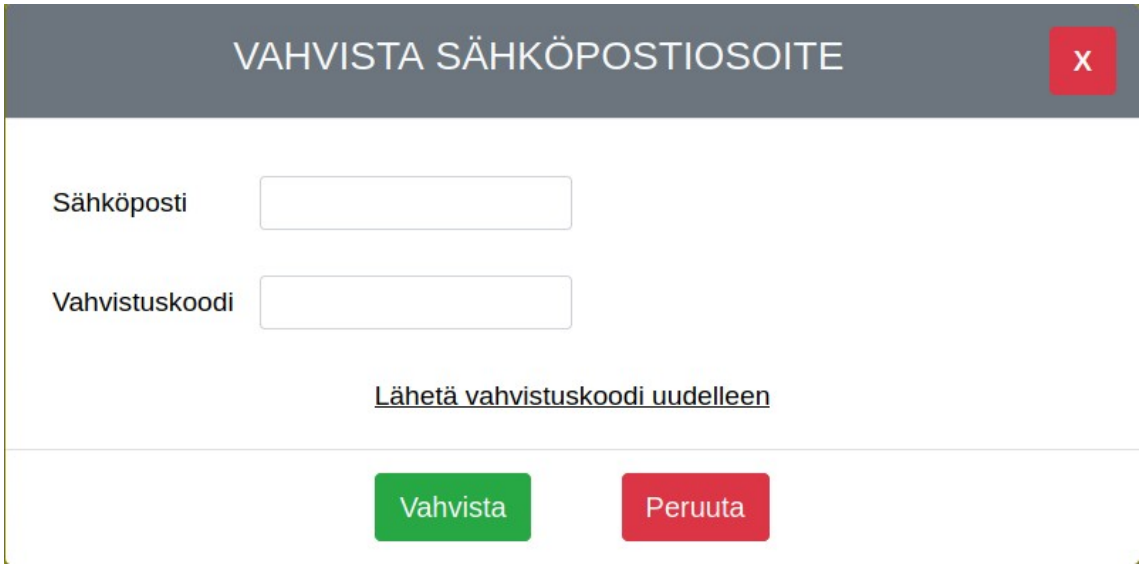
Sähköpostiviesti lähetetään, kun lähdekoodissa kutsutaan aiemmin tehtyä sähköpostiviestin lähetysfunktiota. Tästä sovelluksesta lähetetyt sähköpostiviestit eivät sisällä raakaa tekstiä vaan ne ovat HTML:ää, jota useimmat sähköpostipalveluntarjoajat kykenevät näyttämään sähköpostipalveluissaan. HTML-sähköpostiviestejä varten tehtiin oma registration-request-created.ejs-niminen tiedos-

to, jonka sisältö renderöidään käyttäen kolmannen osapuolen kirjastoa nimeltä ejs ja annetaan sisältönä lähetettävälle sähköpostiviestille.

### 4.4.3 Sähköpostiosoitteen vahvistaminen

Rekisteröityjän sähköpostiin saapuva viesti sisältää lyhyen ohjeen rekisteröitymisen loppuunsaattamiseksi, sekä vahvistamiseen tarvittavan erillisen vahvistuskoodin. Jotta sähköpostiosoite voidaan katsoa vahvistetuksi, tulee käyttäjän palauttaa vahvistuskoodi palvelimelle jollakin tavalla. Tähän tehtiin kaksi erilaista tapaa.

Ensimmäisessä tavassa käyttäjä kopioi saamansa vahvistuskoodin käyttöjärjestelmänsä leikepöydälle ja siirtyy takaisin sovellukseen, johon tehtiin tätä toimenpidettä varten alla näkyvän kuvan 13 mukainen näkymä. Käyttäjän tulee antaa sekä rekisteröitymisessä käyttämänsä sähköpostiosoite että leikepöydälle kopioimansa vahvistuskoodi. Annetut syötteet tarkastetaan aiemmin rekisteröitymisen yhteydessä kuvatuin tavoin. Kun käyttäjä painaa lähetyspainiketta, lähetetään vahvistuspyyntö palvelimelle.

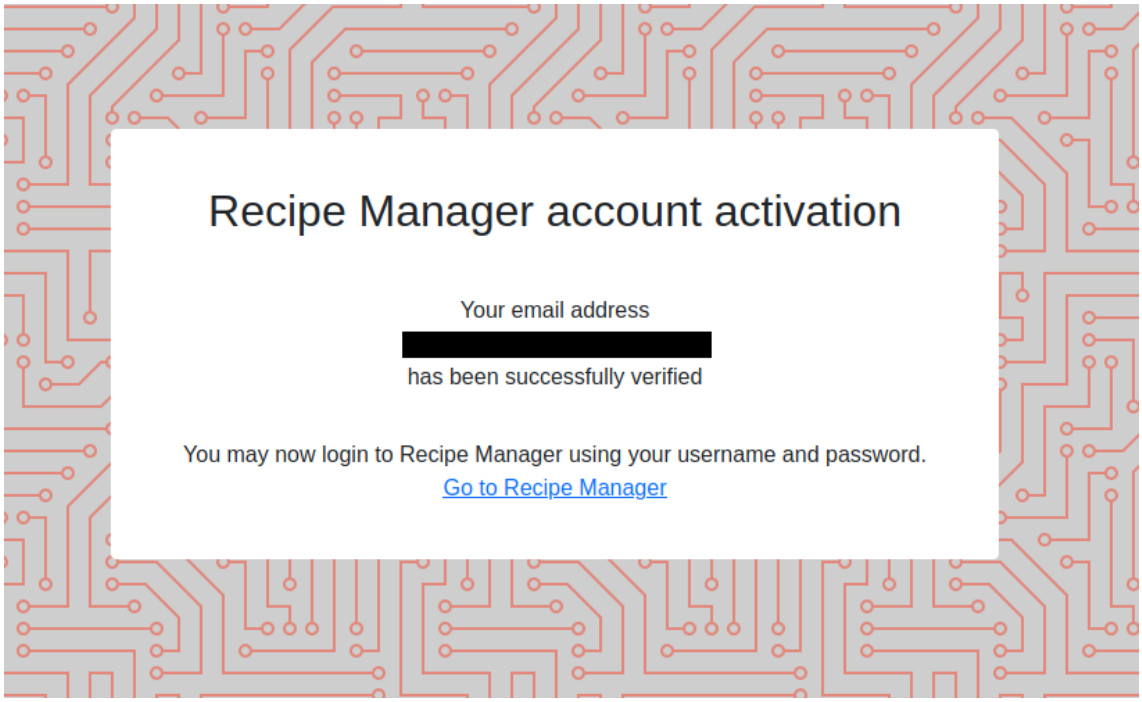



KUVA 13. Sähköpostiosoitteen vahvistusikkuna

Paketin saavuttua palvelimelle, suoritetaan tietokannan pendings-collectioniin kysely käyttäen annettua sähköpostiosoitetta. Mikäli osuma löytyy, vertaillaan vielä annettua vahvistuskoodia löydetyn rekisteröitymispyynnön vahvistuskoo-

diin. Mikäli koodit vastaavat toisiaan, suoritetaan vielä viimeinen kysely tietokannan users-collectioniin, jolla halutaan varmistua, ettei annettua sähköpostiosoitetta ole jostakin syystä ehditty vahvistamaan. Mikäli mitään ei löydy, luodaan uusi käyttäjä lopulta users-collectioniin. Clientille palautetaan vastauksena yksinkertainen viesti, jossa kerrotaan vahvistuksen onnistuneen, ja käyttäjä voisi nyt kirjautua sisään käyttäen käyttäjätunnustaan ja salasanaansa.

Toinen tapa vahvistaa sähköpostiosoite on avata sähköpostiin saapunut suora linkki, joka sisältää URLin polkuosiossa käyttäjän ID:n sekä vahvistuskoodin. Linkin avaamalla palvelimelle lähtee HTTP GET -kutsu ja palvelin lukee edellä mainitut arvot suoraan kutsun polusta. Tämän jälkeen tiedot vahvistetaan samalla tavalla kuin aiemminkin, mutta palvelin palauttaakin rekisteröityjälle vastauksena staattisen web-sivun. Tästä alla havainnollistava kuva 14.




KUVA 14. Onnistunut rekisteröityminen suoraa linkkiä käyttämällä

### 4.4.4 Sisäänkirjautuminen

Sisäänkirjautuminen tapahtuu perinteisesti käyttäjätunnuksen ja salasanan avulla. Käyttäjä aloittaa syöttämällä näkymään edellä mainitut tiedot ja painaa kirjatumispainketta. Syötekenttien sisällöt tarkistetaan alustavasti clientin pääs-

sä tyhjien arvojen varalta eikä pyyntöä lähetetä palvelimelle ollenkaan, mikäli jommasta kummasta puuttuu sisältö. Tällöin käyttäjälle ilmoitetaan toast-tyyppisellä ilmoituksella puuttuvista arvoista. Mikäli syötteet hyväksytään ja kutsu voidaan lähettää, paketoidaan ne HTTP POST -kutsun bodyyn ja lähetetään palvelimelle. Alla olevassa kuvassa 15 näkyy sisäänkirjautumisen näkymä.


KUVA 15. Sisäänkirjautumisen näkymä

Käyttäjän tarjoama syöte tarkastetaan myös palvelimen päässä vastaavalla tavalla kuin edellä clientin tapauksessa ja tarvittaessa hylätään, mikäli syötteessä on jotakin vialla. Mikäli tarkastusseula läpäistään, palvelin suorittaa tietokantaan kyselyn, jossa etsitään kyseisellä käyttäjänimellä olevaa tunnusta. Mikäli vastine annetulle käyttäjätunnukselle löytyy tietokannasta, luetaan seuraavaksi sen salasana ja vertaillaan sitä kirjautumispyynnössä tarjoiltuun salasanaan.

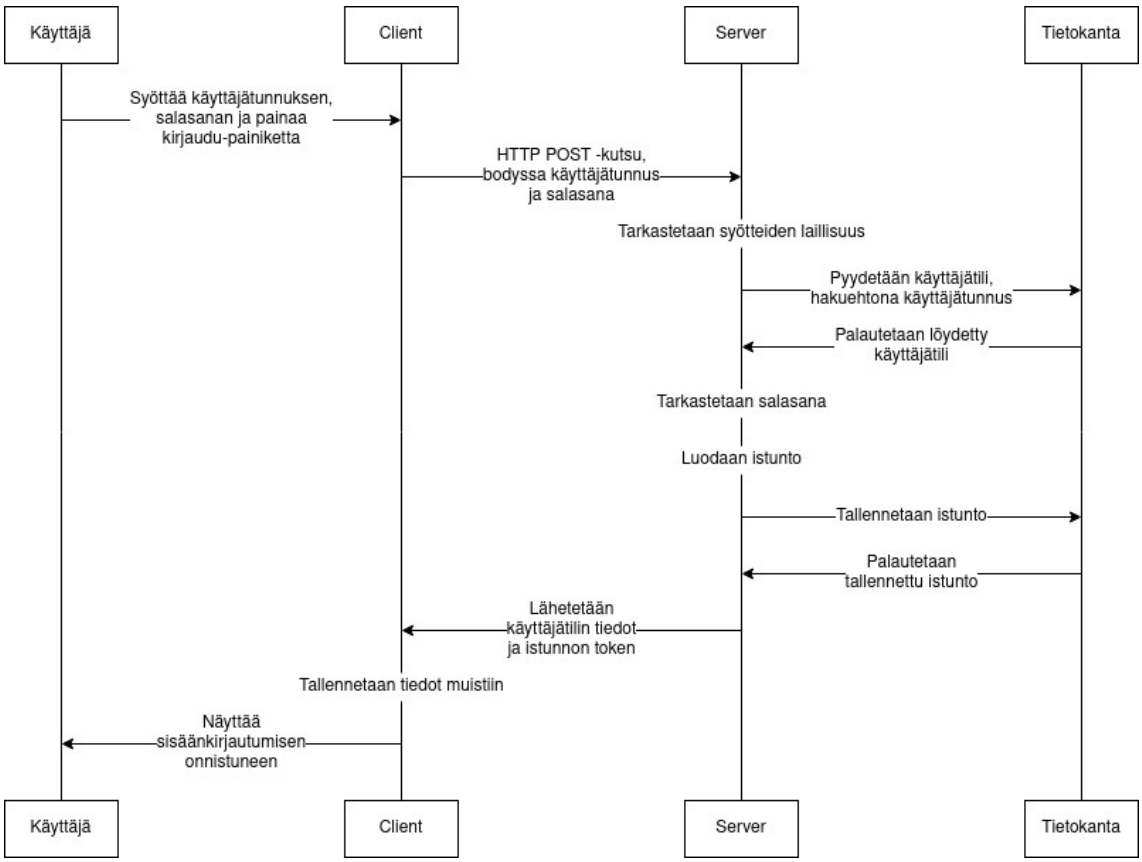
Mikäli käyttäjätunnus ja salasana täsmäävät, kirjautuminen hyväksytään ja käyttäjä katsotaan sisäänkirjautuneeksi. Tämä tarkoittaa sitä, että luodaan sessions-collectioniin uusi Session-tyyppinen olio, johon talletetaan käyttäjältä muun muassa ID, käyttäjänimi, IP-osoite, josta kutsu on saapunut, sekä kellonaika, jolloin sisäänkirjautuminen on tapahtunut. Näiden lisäksi luodaan kryptografisesti avainpari käyttäen Noden omaa Crypto-kirjastoa ja tämän avaimen toinen osapuoli talletetaan istunto-olioon. Istunto talletetaan tietokantaan ja tietokanta palauttaa palvelimelle takaisin tallennetun istunnon, mikäli tallennus onnistui. Tämän jälkeen istunto paketoidaan mukaan käyttäjälle lähetettävään vastaukseen. Vastaukseen tulee myös käyttäjätilin tiedoista mukaan käyttäjän id, käyttäjätunnus, sähköpostiosoite, lokaali sekä MongoDB:n sisäisesti luodut tiedot createdAt sekä updatedAt. Istunnosta otetaan mukaan istunnon id, createdAt,

viimeisimmän aktiviteetin ajankohta sekä aiemmin mainitun avainparin toinen osapuoli.

Mikäli jokin näistä vaiheista epäonnistuu, lähetetään käyttäjälle virheilmoitus. Tässä kohtaa on kuitenkin tärkeätä, että käyttäjälle ei kerrota liian spesifisesti sitä, mikä sisäänkirjautumisessa meni pieleen. Antoipa käyttäjä sitten käyttäjätunnuksen, joka ei ole käytössä, tai oikean käyttäjätunnuksen, mutta sille väärän salasanan, saa hän kummassakin tapauksessa saman, hyvin geneerisen virheilmoituksen, jossa kerrotaan jonkin menneen pieleen sisään kirjautumisessa. Mikäli esimerkiksi pahansuopa taho yrittää brute force -hyökkäyksellä päästä kirjautumaan sisään, on hyvä, ettei hänelle tässä tapauksessa kerrota hänen löytäneen oikean käyttäjätunnuksen, jolloin hän voisi siirtyä kokeilemaan eri salasanoja.

Clientin puolella otetaan palvelimen lähettämä vastaus ja tarkastetaan sen bodyn sisältämät aiemmin mainitut tiedot, jonka jälkeen sen tiedot siirretään sovelluksen muistiin. Ohjelma tulkitsee tämän jälkeen sisäänkirjautumisen tapahtuneen onnistuneesti ja muuttaa näkymän vastaamaan sisäänkirjautunutta. Yläriville tulee näkyviin käyttäjäprofiilisivun avaava painike sekä sisäänkirjautumispainike vaihtuu uloskirjautumispainikkeeksi. Tämän jälkeen client hakee palvelimelta kaikki käyttäjän ruokareseptit ja tuo ne etusivun taulukkoon näkyviin.

Alla olevassa kuvassa 16 on Reseptimanagerin sisäänkirjautumista havainnollistava authentication flow -kuva, joka näyttää sisäänkirjautumiseen liittyvät vaiheet.




KUVA 16. Reseptimanagerin sisäänkirjautumisen authentication flow

### 4.4.5 Istunnon validointi

Useat palvelimen suorittamat toimenpiteet vaativat autentikoitumisen todentamisen eli toisin sanoa voimassa olevan istunnon todentamisen. Esimerkiksi seuraavat ominaisuudet vaativat käyttäjän sisäänkirjautumista:

- sähköpostiosoitteen vaihtamispyynnön lähettäminen
- salasanan vaihtaminen
- uloskirjautuminen
- käyttäjätunnuksen poistaminen

Näiden lisäksi kaikki varsinaiset reseptien käsittelyyn kuuluvat toimenpiteet suoritetaan sisäänkirjautuneena ja vaativat siten istunnon validointia.

Kun palvelin saa clientiltä jonkin pyynnön, joka vaatii voimassa olevan istunnon, varmentaa palvelin ensin itse istunnon, ennen kuin siirtyy varsinaisen pyynnön suorittamiseen. Istunnon oikeellisuuden ja voimassaolon toteaminen tapahtuu tutkimalla tietokannasta löytyvän sessions-collectionia, josta etsitään kyseisen

istunnon oma olio. Mikäli sellainen löytyy, sen sisältöä verrataan käyttäjän HTTP-kutsujen headereissa välittämien avain-arvo-parien sisältöön. Mikäli kaikki tiedot ovat kohdallaan, todetaan käyttäjän istunto validiksi ja palvelin lähtee suorittamaan varsinaisia haluttuja toimenpiteitä.

Koska HTTP-kutsun validointi ja istunnon tarkastaminen tapahtuu aina täsmälleen samalla tavalla, ei jokaista endpointtia varten ole kannattavaa kirjoittaa samaa koodia uudelleen useita kertoja, vaan on järkevämpää käyttää uudelleen samaa funktiota. Express mahdollistaa tämän käyttämällä niin kutsuttuja middleware-funktioita. Ne ovat funktioita, joita Express kykenee hyödyntämään joko yhteisesti kaikkia saapuvia kutsuja varten tai sitten vain yksittäisiä kutsuja varten. Middleware-funktioilla on pääsy niin kutsuttuihin request- ja response-olioihin, joita muokkaamalla päästään vaikuttamaan koodin suoritukseen. Niitä hyödynnetään tyypillisesti juuri tämän kaltaisissa tilanteissa, joissa halutaan suorittaa jonkinlainen toimenpide useammalle kuin yhdelle endpointille tulevalle kutsulle.

Tässä projektissa tehtiin istunnon validiteetin tarkistamista varten oma middleware-funktionsa, joka suorittaa yllä mainitut tarkastukset. SessionVerification.js-tiedostosta löytyy verifySession-metodi, joka tarkastaa, että kaikki tarpeelliset HTTP Headerit löytyvät olemassa olevan istunnon vaativaan endpointtiin tulevasta kutsusta. Esimerkiksi recipe-manager-session-access-token sisältää avaimen, joka luodaan sisäänkirjauduttaessa. Tämä avain tarkastetaan kryptografisesti, että se vastaa haluttua arvoa, ja hyväksytään, mikäli näin on. Myös käyttäjän IP-osoitteen tulee olla sama kuin istunnon oliolta saatavan IP-osoitteen. Tämä perustuu olettamukseen, että käyttäjän IP-osoite ei vaihdu istunnon voimassaolon aikana. Sovellusta mobiililaitteella käyttävälle käyttäjälle tämä voisi aiheuttaa istunnon menetyksen.

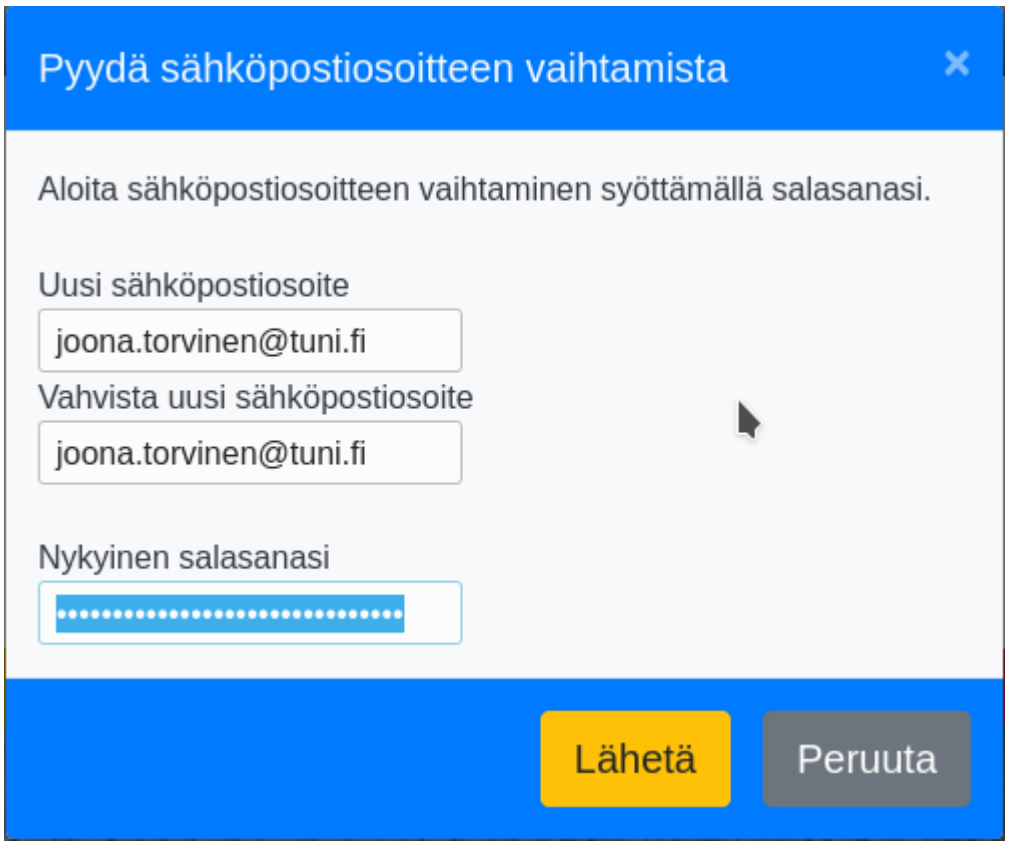
Viimeisenä funktiossa tarkastellaan istunnon voimassaoloa. Session-olion yksi avaimista on lastActivity, joka on Date-tyyppinen Javascript-olio, joka kertoo sen, koska käyttäjä on kyseisellä istunnolla viimeksi suorittanut onnistuneen kutsun palvelimelle. Mikäli tuosta ajasta on kulunut vähemmän kuin yksi viikko aikaa, katsotaan istunnon olevan voimassa. Tällöin tämä arvo nollataan asettamalle se tähän hetkeen alustamalla se uudelleen new Date() -komennolla ja

käyttäjä siirretään eteenpäin kontrollerille, joka suorittaa varsinaiset kutsussa pyydetyt toimenpiteet. Mikäli tuosta ajasta on kulunut kauemmin, katsotaan istunto vanhentuneeksi ja istunto poistetaan. Tässä kohtaa on myös otettava huomioon, että on periaatteessa mahdollista, että roskienkeruuohjelma, josta kerrotaan luvussa 4.5, on mahdollisesti käynyt poistamassa automaattisesti kyseisen istunnon. Näin ollen sitä ei löydy vaikka käyttäjä ei itse olisi kirjautunut ulos.

### 4.4.6 Sähköpostiosoitteen vaihtaminen

Käyttäjälle saattaa tulla eteen tilanne, että hän haluaa vaihtaa sovelluksen käyttäjätiliinsä linkitetyn sähköpostiosoitteen. Käyttäjä voi esimerkiksi menettää pääsyn sähköpostiinsa syystä tai toisesta ja tarvitsee uuden sähköpostiosoitteen linkitetyksi Reseptimanagerissa.

Sähköpostiosoitteen vaihtamista varten tehtiin käyttöliittymän profiilisivulle oma painikkeensa, joka avaa alla olevassa kuvassa 17 näkyvän ponnahdusikkunan, josta käyttäjän on mahdollista lähettää palvelimelle sähköpostiosoitteen vaihtamispyyntö. Käyttäjää pyydetään syöttämään uusi sähköpostiosoite kahteen kertaan. Tällä pyritään ehkäisemään se, että käyttäjä ei vahingossa kirjoita sähköpostiosoitetta väärin, jolloin pyyntö ei myöskään lähtisi väärään sähköpostiosoitteeseen. Käyttäjän tulee myös syöttää salasanansa, jotta saadaan lisävarmuutta siihen, että käyttäjä todella on itse vaihtamassa oman sähköpostiosoitteensa.




KUVA 17. Sähköpostiosoitteen vaihtamisen ponnahdusikkuna

Ennen varsinaisen HTTP-kutsun lähettämistä palvelimelle, tehdään annetuille syötteille clientin tasolla muutama tarkastus. Molempiin syötekenttiin annettujen sähköpostien pitää olla kelvollisia sähköpostiosoitteita ja niiden pitää myös keskenään olla samat. Käyttäjä ei myöskään saa antaa uudeksi sähköpostiosoitteeksi nykyistä sähköpostiosoitettaan. Lopuksi salasanalle tehdään samat tarkistukset kuin rekisteröitymisvaiheessa. Mikäli syötteet eivät läpäise jokaista tarkistusta, annetaan käyttäjälle toast-muotoinen virheilmoitus, eikä palvelimelle lähetetä kutsua. Jos taasen kaikki tarkistuksen suoritetaan onnistuneesti, lähetään palvelimelle HTTP POST-kutsu.

Palvelimen päässä kutsu otetaan vastaan omassa /api/request-email-change -endpointissaan. Ensimmäiseksi tehdään aiemmin mainittu istunnon tarkastus. Kun kutsu on läpäissyt tämän middlewaren, siirretään sen käsittely vastaavalle kontrollerille. Ensimmäiseksi tarkistetaan POST-kutsun body, jotta sieltä löytyy kaikki tarvittava ja että annetut syötteet ovat oikean muotoisia. Tämän jälkeen suoritetaan tietokantaan kysely, jossa haetaan käyttäjän User-olio. Ensimmäiseksi käyttäjätilin löydyttyä tarkastetaan, että salasana on oikein. Seuraavaksi tarkistetaan, että uusi haluttu sähköpostiosoite ei ole käyttäjän nykyinen, jo voi-

massa oleva sähköpostiosoite. Mikäli näissä tarkastuksissa epäonnistutaan, pyynnön suorittaminen keskeytetään, ja lähetään takaisin clienttiin virheilmoitus.

Mikäli näistä tarkistuksista päästään lävitse onnistuneesti, tallennetaan käyttäjän User-olioon uusi sähköpostiosoitteen vaihtamispyyntö. User-oliolla on attribuutti nimeltä emailChangeRequest, joka pitää sisällään kolme alla olevan kuvan 18 mukaista tietoa. Sähköpostiosoitteen vaihtamispyyntöä varten generoidaan satunnainen string, joka lähetetään sähköpostilla käyttäjälle.

```
// Generate a new code for email change and save it
user.emailChangeRequest = {
    ip: req.ip,
    newEmail: req.body.newEmail,
    verificationCode: generateRandomHexString(),
};
```

KUVA 18. Sähköpostiosoitteen vaihtamispyynnön alustaminen

Kun sähköpostiviesti on saapunut uuteen osoitteeseen, tulee käyttäjän avata sen mukana saapunut linkki. Tämän linkin avattua palvelimen /api/changeemail/:recipemanageruserid/:emailchangecode -endpointtiin lähtee HTTP GET-kutsu. Palvelin välittää kutsun suoraan kontrollerille, joka tarkastaa sen parametreinä saamansa käyttäjän ID:n ja vahvistuskoodin. Tämän jälkeen palvelin suorittaa tietokantaan kyselyn käyttäen käyttäjän ID:tä ja mikäli osuma löytyy, tarkastaa se käyttäjäolion requestEmailAddressChange-avaimen arvon. Mikäli arvo on null tai undefined, palvelin keskeyttää pyynnön suorittamisen. Mikäli tämä arvo sisältää EmailAddressChangeRequest-tyyppisen olion, luetaan seuraavaksi sen verificationCode-avaimen arvo, jota verrataan kutsun parametrinä saatuun arvoon. Mikäli nämä vastaavat toisiaan, asetaan käyttäjäolion sähköpostiosoitteeksi käyttäjän haluama uusi sähköpostiosoite ja muutokset tallennetaan tietokantaan. Onnistuneesti suoritetusta vaihtamisesta lähetään vastaus käyttäjälle, jonka käyttäjä näkee omana sivunaan omalla välilehdellä selaimessaan.

Mikäli käyttäjä esimerkiksi huomaisi lähettäneensä vaihtamispyynnön vahvistuksen väärään sähköpostiosoitteeseen, on sisäänkirjautuneen käyttäjän mahdollista perua vaihtopyyntö ennen kuin se on vahvistettu. Käyttöliittymän profiilisi-

vulle tulee näkyviin odotustilassa oleva vaihtopyyntö ja yksinkertainen nappula, jota painamalla käyttäjän on mahdollista välittömästi peruuttaa vaihtopyyntö. Palvelimen päässä tälle pyynnölle tehdään vain normaali istunnontarkastus, jonka jälkeen käyttäjäolio haetaan tietokannasta, vaihtopyyntö asetetaan nulliksi, muutokset tallennetaan ja käyttäjän clientille lähetetään viesti onnistuneesta vaihdosta.

### 4.4.7 Salasanan vaihtaminen

Käyttäjätilin salasana tulee voida vaihtaa monestakin eri syystä. Yksi esimerkiksi voisi olla, että käyttäjä epäilee salasanansa vaarantuneen ja levinneen asiattomalle taholle.

Tätä toiminnallisuutta varten luotiin käyttöliittymään profiilisivulle oma painike, jota painamalla avautuu alla olevassa kuvassa 19 näkyvä ponnahdusikkuna, jota käyttämällä käyttäjä voi vaihtaa salasanansa. Ensiksi käyttäjältä pyydetään hänen nykyinen salasanansa ja tämän jälkeen uusi salasana kahteen kertaan. Vanhan salasanan osalta tarkistetaan vain, ettei se ole tyhjä. Uuden salasanan syötteet tarkistetaan, että ne ovat sovelluksen salasanan vaatimusten mukaisia. Niiden pitää myös olla keskenään samat, ja että uusi salasana ja sen varmistus ovat keskenään täysin samat, mutta että uusi salasana ei kuitenkaan ole sama kuin vanha salasana. Mikäli jokin näistä tarkastuksista epäonnistuu, käyttäjälle ilmoitetaan virheestä, eikä palvelimelle lähetetä pyyntöä. Jos tarkastukset menevät kaikki lävitse, lähetetään palvelimelle kutsu.


KUVA 19. Salanan vaihtamisen ponnahdusikkuna

Palvelin ottaa pyynnön vastaan omassa /api/change-password -endpointissa, jossa sille suoritetaan ensiksi aiemmin kuvatun kaltainen istunnontarkastus, jonka jälkeen se siirretään varsinaisen kontrollerin käsiteltäväksi. Kontrolleri suorittaa tietokantaan haun users-collectioniin käyttäen käyttäjän ID:tä. Jos käyttäjä löydetään tietokannasta, tarkastetaan ensimmäiseksi käyttäjän tarjoama nykyinen salasana, että se on oikein. Tämän jälkeen tarkastetaan haluttu uusi salasana, että se on muotovaatimusten mukainen ja että se ei ole sama, kuin nykyinen salasana.

Jos näistä tarkastuksista päästään lävitse, käyttäjän uusi salasana talletetaan tietokantaan, ja käyttäjälle lähetetään viesti onnistuneesti vaihdetusta salasanasta. Tämän jälkeen palvelin vielä poistaa kaikki kyseisen käyttäjän istunnot sessions-collectionista, jotta kaikki käyttäjän istunnot saadaan suljettua. Tällä halutaan varmistaa se, että mikäli käyttäjän vanha salasana olisi vuotanut ulkopuoliselle taholle ja käyttäjä haluaa vaihtaa salasanansa, ei aiemmin luotuja voimassa olevia istuntoja voisi käyttää vahingontekoon.

Käyttöliittymän tasolla käyttäjä kirjataan istunnosta ulos ja pakotetaan käyttäjä kirjautumaan uudelleen sisään uudella salasanallaan, mikäli palvelin kertoo sa-

lasanan vaihdon onnistuneen. Tällä pyritään vähentämään mahdollisten väärinkäytösten riskiä.

### 4.4.8 Uloskirjautuminen

Uloskirjautuminen tapahtuu aiemmin kuvailtujen toimintojen kaltaisesti. Sisäänkirjautunut käyttäjä painaa käyttöliittymässä kirjaudu ulos -painiketta, joka lähettää välittömästi palvelimelle kutsun.

Ensimmäisenä palvelin tarkastaa, onko käyttäjän istunto voimassa. Mikäli se on voimassa ja käyttäjän kredentiaalit ovat kohdallaan, siirretään kutsu eteenpäin sen varsinaiselle kontrollerille. Tässä tapauksessa ei tarvitse tehdä sen kummempia kyselyjä tietokantaan, vaan poistetaan ainoastaan löytynyt istunto. Mikäli istunto saadaan poistettua onnistuneesti, lähetetään clientille takaisin vastaus, jossa kerrotaan poiston onnistuneen. Mikäli tässä kohtaa tapahtuu jokin virhe, lähetetään clientille virheilmoitus.

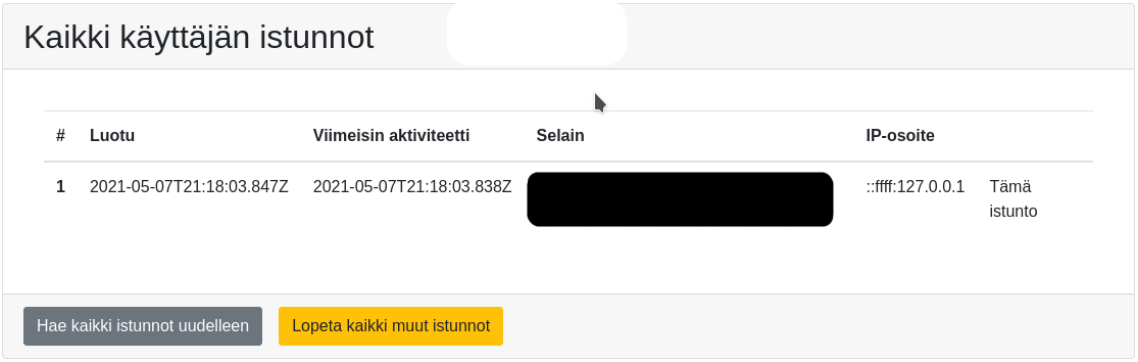
Kun client saa vastauksena viestin onnistuneesti poistetusta istunnosta, kirjataan käyttäjä näennäisesti ulos poistamalla Vuex storesta löytyvät oliot, jotka pitävät sisällään kaikki käyttäjän istuntoon viittaavat tiedot. Lisäksi myös käyttäjän omat ruokareseptejä pitävät taulukot ja oliot tyhjennetään. Näiden tyhjennyttyä ohjelma katsoo, että käyttäjä on nyt kirjautunut ulos ja tyhjentää aloitusnäkymän taulukot, poistaa näkyvistä käyttäjäprofiili-sivulle vievän painikkeen sekä vaihtaa takaisin kirjaudu sisään -painikkeen.

### 4.4.9 Istuntojen poistaminen

Tietoturvan kannalta olisi hyvä, että käyttäjällä ei olisi verkkopalveluun tarpeettomasti auki istuntoja, sillä istuntoja on mahdollista hyödyntää erilaisissa hyökkäyksissä. Esimerkiksi niin kutsutuissa session hijacking -hyökkäyksissä hyökkääjä pyrkii sovelluksen haavoittuvuuksia käyttäen saamaan käyttöönsä istunnon sisältämät tiedot. Tämä voisi tapahtua esimerkiksi siten, että käyttäjien antamia syötteitä ei varmennettaisi mitenkään, jolloin käyttäjä pääsisi syöttämään

sovellukseen haitallista koodia. Saamalla nämä tiedot käsiinsä hyökkääjä pystyy esiintymään uhrina ja tekemään mitä tahansa, mihin uhrilla on oikeudet.

Tämän kaltaisten uhkien ja muiden tunnistamattomien uhkien lieventämiseksi sovellukseen tehtiin näkymä ja toiminnallisuus, joiden avulla käyttäjä pystyy tarkkailemaan hänen käyttäjätilinsä alla auki olevia istuntoja ja tarvittaessa poistamaan niitä. Näkymä on saatavissa käyttäjän omalla profiilisivulla. Alla olevassa kuvassa 20 nähtävään näkymään haetaan taulukkomuodossa kaikki käyttäjällä avoinna olevat istunnot. Tässä näkymässä näytetään jokaiselle istunnolle erikseen ajanhetki, jona käyttäjä on kirjautunut sisään, viimeisin aktiviteetti, käytetty selain sekä IP-osoite, josta istunto on luotu. Lisäksi näkymässä ilmoitetaan käyttäjälle vielä erikseen, jos kyseinen istunto on käyttäjän tämänhetkinen istunto. Muita istuntoja varten taulukkoon generoidaan jokaiselle oma painikkeensa, jota painamalla käyttäjä voi tuhota juuri kyseisen istunnon. Käyttäjällä on myös mahdollisuus tuhota kaikki avoinna olevat istuntonsa, lukuun ottamatta senhetkistä istuntoa. Sen tuhoaminen tapahtuu kirjautumalla ulos.



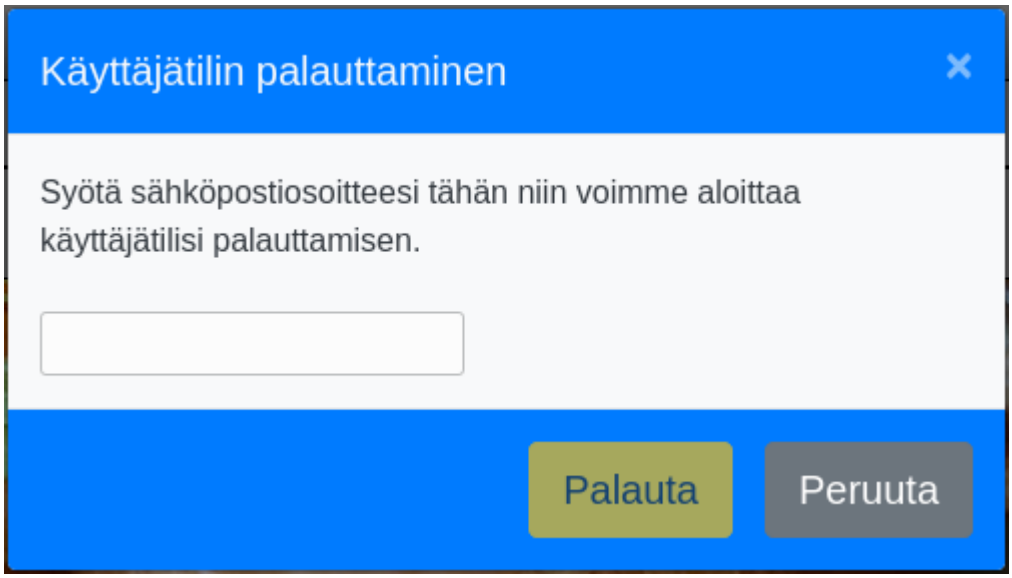

KUVA 20. Istuntojen hallintanäkymä

Näillä toiminnoilla pyritään ehkäisemään käyttäjän mahdollisesti unohtuneita sisäänkirjautumisia sekä auttamaan havaitsemaan mahdollisia vilpillisiä sisäänkirjautumisia.

### 4.4.10 Käyttäjätilin palauttaminen

Käyttäjätilin palauttaminen tapahtuu siten, että käyttäjä avaa sisäänkirjautumisikkunan ja painaa ponnahdusikkunan alareunassa olevaa ”Unohditko käyttäjä-

tunnuksesi tai salasanasi?” -painiketta. Tällöin avautuu uusi ikkuna, johon käyttäjä pyydetään syöttämään tämän sähköpostiosoitteensa. Lopuksi käyttäjä painaa painiketta ja client lähettää palvelimelle pyynnön. Alla olevassa kuvassa 21 näkyy käyttäjätilin palauttamisen näkymä.




KUVA 21. Käyttäjätilin palauttamisen ponnahdusikkuna

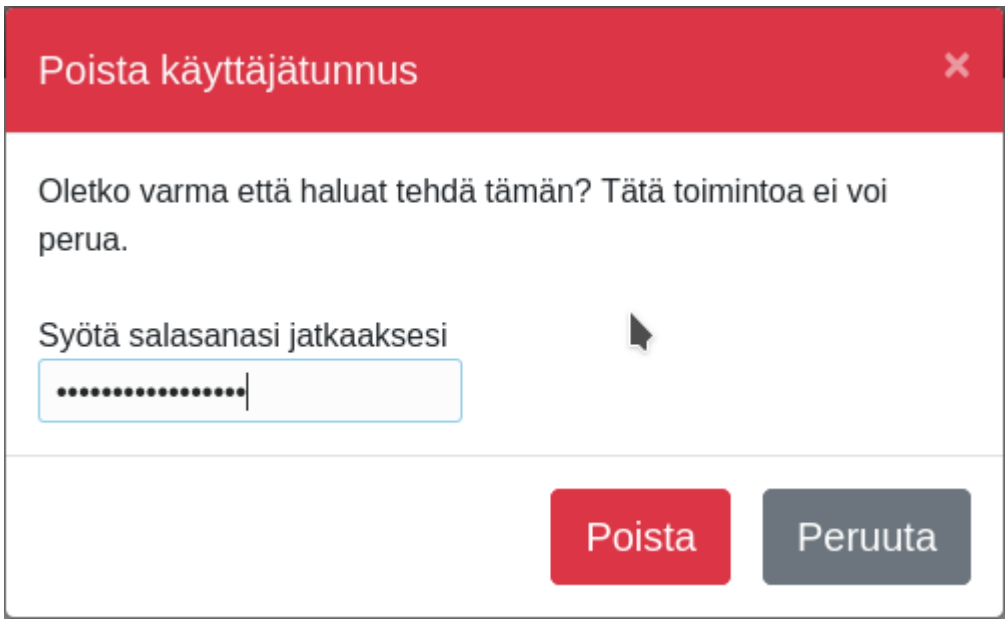
Ensimmäiseksi saapuneelle kutsulle suoritetaan validointi, jossa tarkastetaan aiemmin kuvattujen menetelmien mukaisesti, onko mukana saatu sähköpostiosoite kelvollinen. Mikäli sähköpostiosoite todetaan kelvolliseksi, lähetetään clientille välittömästi takaisin vastaus ennen minkäänlaista tietokantaan suoritettavaa kyselyä. Vastauksessa kerrotaan, että mikäli kyseisellä sähköpostiosoitteella on rekisteröity käyttäjätili, lähetetään sinne ohjeet palauttamista varten. Vasta tämän jälkeen haetaan kyseisellä sähköpostiosoitteella tietokannasta käyttäjän User-olio.

Mikäli käyttäjä löytyy, luodaan uusi AccountRecoveryRequest-tyyppinen olio, joka saa attribuuteikseen sekä palautuspyynnön luoneen tahon IP-osoitteen että Crypto-moduulin metodien avulla generoidun satunnaisen stringin. Uusi olio syötetään käyttäjäolion attribuutiksi ja käyttäjäolion muutokset tallennetaan tietokantaan. Mikäli tallennus tietokantaan onnistuu, ilmoitetaan asiasta käyttäjälle sähköpostitse. Lähetettävään sähköpostiviestiin liitetään mukaan palauttamista varten generoitu palautuskoodi sekä IP-osoite, josta pyyntö on luotu. Viesti sisältää linkin palvelimen endpointtiin, joka suorittaa salasanan nollaamisen.

Käyttäjän avattua linkin, lähtee palvelimelle HTTP GET -kutsu, jonka parametreihin on syötetty käyttäjän yksilöllinen ID sekä palautukseen tarvittava koodi. Käyttäjän ID:n tulee olla validi MongoDB ID ja tämä tarkastetaan, ennen kuin kutsun käsittelyä jatketaan pidemmälle. Tämän jälkeen palvelin hakee käyttäjän Users-olion tietokannasta tämän ID:llä. Mikäli käyttäjäoliolta löytyy voimassa oleva palautuspyyntö, vertaillaan sitä kutsussa saatuun palautuskoodin ja katsotaan, ovatko ne yhteneväiset. Mikäli kaikki on kunnossa, asetetaan käyttäjälle uusi salasana, joka on satunnaisista merkeistä koostuva pitkä string. Uusi salasana talletetaan kryptattuna tietokantaan uuden suolan kanssa. Lopuksi käyttäjälle lähetetään erillinen HTML-sivu, jossa kerrotaan sähköpostiosoitteen vahvistamisen onnistuneen, ja lähetetään sähköpostilla juuri luotu uusi salasana. Tämän jälkeen käyttäjä on valmis sisäänkirjautumaan uudella salasanallaan.

### 4.4.11 Käyttäjätilin poistaminen

Käyttäjän tulee voida poistaa oma käyttäjätilinsä palvelusta ja tulla unohdetuksi, eikä hänen tarvitse tätä millään tavalla perustella palvelun ylläpitäjälle. Tämä tapahtuu siten, että käyttäjä navigoi omalle profiilisivulleen, josta löytyy ”Poista käyttäjätunnus” -painike. Painiketta painettaessa käyttäjän eteen avautuu alla olevan kuvan 22 mukainen ikkuna, johon käyttäjää pyydetään vahvistamaan salasanansa. Kun käyttäjä on syöttänyt syötekenttään salasanansa ja painanut poistamispainiketta, esitetään käyttäjälle vielä viimeinen varmistusviesti. Mikäli käyttäjä tähänkin vastaa kyllä, lähtee palvelimelle HTTP DELETE -kutsu, jonka bodyssa käyttäjän salasana on.




KUVA 22. Käyttäjätunnuksen poistaminen

Palvelimen puolella kutsulle suoritetaan samat alustavat validoinnit kuin kaikille muillekin istunnon todentamisen vaativille kutsuille. Kun istunto on todettu validiksi, suoritetaan tietokantaan kysely, jolla haetaan käyttäjään User-olio. Tämän jälkeen käyttäjätunnus ja kaikki kyseisen käyttäjätilin istunnot poistetaan, mikäli käyttäjän antama salasana oli oikea. Käytännössä tämä tapahtuu niin, että käyttäjätilin sisällään pitävästä users-collectionista poistetaan kyseisen tilin oma document, ja sessions-collectionista poistetaan kaikki kyseiselle käyttäjä-ID:lle kuuluvat documentit. Kun edellä mainitut toimenpiteet on suoritettu, käyttäjälle palautetaan yksinkertainen vastaus, jossa kerrotaan käyttäjätilin poistamisen onnistuneen. Tällöin client näennäisesti kirjaa käyttäjän ulos ja näyttää pienen sievän hyvästelyviestin.

Käyttäjän luomat ruokaohjeet asetetaan poistettaviksi lyhyen varoajan jälkeen. Tällä pyritään siihen, että ylläpitäjän toimesta ne voitaisiin vielä pelastaa, mikäli käyttäjä jostakin syystä tulisi katumapäälle ja haluaisi ne vielä palautettavan.

### 4.5 Tietokannan roskienkeruu

Tietokantaan tallennetut istunnot tulisi säännöllisin väliajoin tarkastaa ja liian kauan avoinna mutta käyttämättöminä olleet istunnot poistaa. Myöskin liian kauan roikkuneet rekisteröitymispyynnöt, eli pyynnöt, joita ei ole vahvistettu, tuli-

si hävittää. Tällä pyritään ennen kaikkea ehkäisemään tietokannan roskautumista mutta myös ehkäisemään mahdollisia tietoturvariskejä, joista osaa opiskelija ei välttämättä edes osaa tässä kohtaa kehitystyötä tunnistaa. Näiden tarpeiden pohjalta sovellukseen tehtiin toiminnallisuus, jota tässä opinnäytetyössä nimitetään roskienkeruuksi. Sillä ei tarkoiteta roskienkeruuta perinteisessä mielessä, eli ohjelman varaaman mutta tarpeettomaksi jääneen muistin vapauttamista.

Roskienkeruu harkittiin alunperin toteutettavaksi osana varsinaista palvelinsovellusta. Halutun toiminnallisuuden olisi voinut toteuttaa omana asynkronisesti suoritettavana koodin osiona, joka tasaisin väliajoin kävisi läpi koko tietokannan ja poistaisi halutut documentit. Tästä tavasta kuitenkin luovuttiin, sillä JavaScript ei tue monisäikeistystä (multi-threading) eikä siten myöskään Node suoranaisesti. Tämän johdosta Noden heikkouksiin katsotaan kuuluvan prosessori-intensiiviset toimet eli toiminnot, jotka vaativat suuria määriä vaativia laskutoimituksia (Martinez 2020). Tämä muuttuisi merkittäväksi asiaksi silloin, kun sovelluksella olisi suuri määrä käyttäjiä.

Nodea varten on olemassa oma Worker-moduuli, jonka avulla on mahdollista luoda oma Worker-olio, jonka suorittaminen tapahtuu erillään pääsäikeestä. Workerit ovat tuore ominaisuus Nodessa eikä niiden toimintaan ole mielekästä mennä syvällisesti tämän opinnäytetyön raameissa, mutta ne tarjoavat eräänlaisen kiertoratkaisun monisäiesuorittamisen puutteeseen. (Sudasinghe 2021) Niiden käyttäminen käytännössä luo uuden instanssin Nodesta omalla, mikä tekee niiden käyttämisestä melko resurssisyöpön. Tästä johtuen myös Workerien käytöstä roskienkeruuta varten luovuttiin.

Ratkaisuksi valittiin oman Node-sovelluksen tekeminen pelkästään roskienkeruuta varten, joka on täysin erillään Reseptimanagerin palvelimesta. Etuna tässä ratkaisussa on se, että näiden kahden ohjelman suorittamiset eivät suoranaisesti riipu toisistaan eikä palvelimen tarvitse olla suoranaisesti tietoinen roskienkerääjän suorittamista toimista. Palvelimen kuormitus ei estä eikä hidasta suoraan roskienkerääjän toimintaa tai päinvastoin. Vaikka erillinen Node-sovellus ei resurssienkäytöltään ole yhtään kevyempi kuin Workerin käyttäminen, niin sitä ei tarvitse millään tavalla integroida palvelimen toimintaan. Ratkaisun huonona

puolena voidaan pitää sitä, että kumpaakin sovellukseen pitää jonkin verran kopioida samaa koodia ja kehittäjän on itse pidettävä huolta siitä, että tämä on täsmälleen samalla tavalla molemmissa sovelluksissa. Tämä haitta katsottiin kuitenkin tässä tapauksessa siedettäväksi, koska molemmissa sovelluksissa tarvitun koodin määrä roskienkeruuohjelmassa on marginaalisen pieni, ja tämä koodi voidaan käytännössä suoraan kopioida ja liittää palvelimelta roskienkerääjään.

Roskienkerääjä varten ainoat tarvittavat Node-kirjastot olivat dotenv sekä mongoose. Ohjelma ottaa käynnistyessään yhteyden tietokantaan käyttäen omia kredentiaalejaan. Kun tietokantaan on muodostettu onnistuneesti yhteys, käynnistetään JavaScriptin setInterval-metodia suorittamaan koodia tietyin kiintein aikavälein. Suoritettava koodi löytyy alla olevasta kuvasta 23.

```
// Login Sessions
let cursor = Session.find().cursor();
for (let doc = await cursor.next(); doc != null; doc = await cursor.next()) {
    const currentMoment = new Date();
    if (currentMoment - doc.lastActivity > ONE_WEEK) {
        doc.remove();

        const str = "\tDeleted expired user login session with\n" +
            "\t\tID " + doc.id + "\n" +
            "\t\tusername " + doc.username;

        logger.info(str);
    }
}
```

KUVA 23. Roskien keräystä suorittava funktio.

Aivan ensimmäiseksi tarkastetaan, suoritetaanko funktiota jo parhaillaan, ja poistutaan funktiosta, jos asia on näin. Näin tehdään sen vuoksi, että asynkronisia funktioita suoritettaessa on teoriassa mahdollista, että edellinen suoritus on vielä kesken, kun seuraavan suoritusta jo aloitetaan. Tällä halutaan ehkäistä mahdollisia virhetilanteita, jotka saattavat syntyä, kun useampi funktio käsittelisi samaa tietokannasta saatavaa resurssia. Lisäksi tällä pyritään välttämään käyttöjärjestelmän tarpeetonta kuormittamista. Tämä voisi tulla eteen ja muodostua ongelmaksi, mikäli Reseptimanagerilla olisi paljon käyttäjä ja suuri määrä istuntoja. Tällöin istunnot sisältävän sessions-collectionin läpikäyminen voisi kestää huomattavankin kauan.

Ohjelma käy sessions-collectionin documentteja lävitse yksi kerrallaan ja tarkastelee jokaisesta documentista löytyvän lastActivity-avaimen arvoa, joka kertoo sen, koska käyttäjä on viimeksi sovellusta käyttänyt. Sovellus laskee, että nykyhetken ja tuon hetken välillä kuluneen ajan, ja poistaa documentin, mikäli erotus on suurempi kuin erikseen määritelty istunnon vanhenemisaika.

### 4.6 Rate limiting

Yksi yleinen uhka, joka verkossa tarjoiltavaa palvelua saattaa kohdata, on palvelunestohyökkäys. Palvelunestohyökkäys toimii yksinkertaistetusti siten, että hyökkääjä lähettää lyhyessä aikavälissä huomattavan suuren määrän kutsuja palvelimelle tarkoituksenaan hidastaa tai jopa kaataa palvelin tai sen tarjoilema sovellus. Tällöin palvelun tosiasialliset käyttäjät eivät pysty käyttämään palvelua. Palvelunestohyökkäyksiltä on hyvin hankalaa, ellei mahdotonta välttyä täysin, mutta niitä vastaan on mahdollista suojautua yksinkertaisilla mutta kohtalaisen toimivilla menetelmillä.

Yksi tällainen tapa on ohjelmakoodilliset laskea palvelimelle tietystä IP-osoitteesta saapuvia kutsuja ja mikäli niitä tulee tietyssä aikaikkunassa enemmän kuin mitä on määritelty, hylätä ne. Tämän strategian nimi on englannin kielessä rate limiting, ja tässä opinnäytetyössä käytetään myös samaa nimeä hyvän ja yleiseen käyttöön päätyneen käännöksen puuttuessa. Rate limiting ei itsessään ole kaikenkattava ja autuaaksi tekevä puolustusstrategia, mutta siitä on tässä yhteydessä hyötyä.

Reseptimanageria varten ladattiin ja asennettiin Node-kirjasto nimeltä express-rate-limit, joka on suunniteltu juuri tätä tarkoitusta varten. Tässä projektissa asetettiin jokaista reittiä eli endpointtia varten koskeva rate limiter alla olevan kuvan 24 mukaisella tavalla juuresta löytyvään server.js-tiedostoon. Tässä rajoitus on melko höllä, minkä tarkoituksena on kattaa kaikki endpointit, joita palvelin tarjoaa.

```
const globalLimiter = rateLimit({
    windowMs: TEN_MINUTES,
    max: 100,
    message: {
        message: "Too many requests. Please try again later."
    },
});
```

Kuva 24. Rate limiter yleiseen käyttöön

Tämän lisäksi yksittäisille endpointeille, kuten rekisteröitymiselle ja sisäänkirjautumiselle, asetettiin tätä huomattavasti tiukempi rajoituksia. Esimerkiksi sisäänkirjautumisia sallitaan yksi yritys kahdessa sekunnissa. Näiden katsottiin antavan lisäturvaa esimerkiksi aiemmin kuvattuja brute force -hyökkäyksiä vastaan.

### 4.7 Lokittaminen

Lokikirjausten tekeminen eli lokittaminen on tärkeä osa sekä sovelluksen kehittämistä että myös sovelluksen ylläpitämistä ja hallintaa. Kehitysvaiheessa lokittamalla on mahdollista debugata eli tutkia ja korjata koodatessa ilmenneitä virheitä. Tuotantovaiheessa lokittaminen on myös tärkeää. Sen tarkoitus on edelleen auttaa kehittäjää ja ylläpitäjää löytämään virheitä, joita kehitysvaiheessa ei ole löydetty. Sitä käytetään myös tuomaan ilmi sovellusta vastaan tapahtuneita hyökkäyksiä. Tietoturvaan liittyen lokittamisesta on syytä nostaa esille muutama huomionarvoinen seikka.

Ensinnäkään salaukseen oleellisesti liittyviä asioita ei tule koskaan lokittaa. Ilmeinen esimerkki tällaisesti ovat käyttäjien salasanat mutta myös esimerkiksi erinäisten tietojen salauksessa ja salauksen purkamisessa käytetyt tiedot. Näiden tietojen vuotaessa ulkopuolisten tietoon vaarantuisivat käyttäjien yksityisyys ja tietoturva.

Toisekseen käyttäjien yksilöiviä henkilötietoja ei lähtökohtaisesti tulisi lokittaa, ellei tälle ole erityistä perustetta ja tarvetta. Reseptimanagerissa ei käyttäjien välinen viestintä ole mahdollista, mutta mikäli sovelluksessa käyttäjät viestisivät keskenään yksityisesti, tulisi tämäkin ottaa huomioon. Suomen perustuslain 10. pykälässä sanotaan, että ”Kirjeen, puhelun ja muun luottamuksellisen viestin

salaisuus on loukkaamaton” (Suomen perustuslaki 731/1999) Tämä tarkoittaa sitä, että esimerkiksi palvelun ylläpitäjällä ei ole oikeutta saattaa omaan tietoonsa näitä viestejä.

### 4.7.1 Winston

Node tarjoaa valmiiksi console-moduulin, jota yleisesti käytetään sovelluksen kehittämisvaiheessa debugattaessa. Tämän moduulin avulla lokittaminen ei ole kuitenkaan resurssitehokkain vaihtoehto, koska se hidastaa Noden yksisäikeisen event loopin suorittamisten, ja sen myös katsotaan voivan herkästi vuotaa yksityistä tietoa. Muun muassa näiden syiden vuoksi console-moduulia ei suositella käytettäväksi tuotanto-ohjelmistossa. (Thor 2019)

Tuotannossa tapahtuvaa lokittamista varten otettiinkin käyttöön winston-niminen kirjasto. Winston on tuotantoympäristössä tapahtuvaan lokittamiseen erikoistunut kirjasto, jonka avulla voi lokittaa useaan eri kohteeseen, kuten tiedostoon tai tietokantaan. Winston jakaa tapahtumia myös erilaisiin vakavuusasteisiin, joita ovat järjestyksessä tärkeimmässä vähiten tärkeimpään: Error, warn, info, http, verbose, debug ja silly. Näiden vakavuustasojen mukaan on mahdollista määrittää, mihin kohteeseen lokitapahtuma tallennetaan tai tallennetaanko ylipäätään minnekään.

Tässä sovelluksessa luotiin ensin winston.js-niminen tiedosto, josta on kuvakaappaus alla olevassa kuvassa 25. Sille luotiin myös File-tyyppinen Transport, mikä tarkoittaa, että tämän funktion avulla luodut lokioliot tallettavat lokitapahtumansa erilliseen tiedostoon server.log-tiedostoon.

```
const { createLogger, format, transports } = require('winston');
require('winston-mongodb');

module.exports = createLogger({
    transports: [

        // File transport
        new transports.File({
            level: "info",
            filename: 'logs/server.log',
            format: format.combine(
                format.timestamp({ format: 'MMM-DD-YYYY HH:mm:ss' }),
                format.align(),
                format.printf(info => `${info.level}: ${[info.timestamp]}: ${info.message}`),
            )
        }),
```

Kuva 25. Kuvakaappaus winston.js-tiedostosta

Tämän jälkeen palvelimelle alustettiin oma logger-olion lisäämällä alla olevan kuvan 26 mukainen lause server.js-tiedostoon.

```
// Winston logging
const logger = require("./utilities/winston");
```

Kuva 26. Lokiolion deklaraatio

Koska yllä tärkeystasoksi määritettiin info, niin ainoastaan info-tason ja sitä korkeamman prioriteetin lokitapahtumat tallennetaan lokitiedostoon. Esimerkiksi komento logger.log(”jotain”) ei tallenna lokitiedostoon mitään kun taas logger.info(”muuta”) tallettaa. Alla olevassa kuvassa 27 näkyy lokitiedostoon tallennettu tieto.

```
info: Apr-24-2021 15:13:27:     Connected to MongoDB
```

Kuva 27. Lokitiedostoon kirjattu info-tason tapahtuma

### 4.8 Tietosuojaseloste ja käyttöehdot

Tämä sovellus ei ole julkisesti saatavilla oleva verkkopalvelu mutta akateemisen kiinnostuksen ja osaamisen laajentamisen vuoksi tässä opinnäytetyössä tehtiin tälle sovellukselle silti erilliset tietosuojaseloste ja käyttöehdot -dokumentit. Nämä dokumentit tarvittaisiin siinä tapauksessa, että tämä sovellus julkaistaisiin verkkoon yleiseen käyttöön. Nämä dokumentit tehtiin erillisinä staattisina HTML-sivuina, jotka palvelin tarjoilee omista erillisistä endpointeistaan. Nämä sivut ei-

vät siten ole suoranaisesti osa Reseptimanagerin SPA-filosofiaa noudattavaa clienttia.

GDPR:n 12., 13. ja 14. artikloissa määritellään, että mikäli palvelu kerää käyttäjältään henkilötietoja, tulee palveluntarjoajan tuoda esille erillinen tietosuojaseloste (englanniksi privacy policy). GDPR:ssä määritellään myös, mitä asioita tässä dokumentissa on tuotava ilmi käyttäjälle, ja tarjotaan toimiva pohja oman tietosuojaselosteen rakentamiselle.

Alla olevassa kuvassa 28 on nähtävissä kuvakaappaus tehdyn tietosuojaselosteen sisällysluettelosta. Tämän tarkoituksena on havainnollistaa niitä asioita, jotka otettiin huomioon tietosuojaselostetta tehtäessä.


Kuva 28. Tietosuojaselosteen sisällysluettelo

# 5 JATKOKEHITYSIDEAT

Tässä opinnäytetyössä toteutetun kokonaisuuden ulkopuolelle rajattiin muutamia teknisiä asioita, jotka olisi hyvä ottaa huomioon ja toteuttaa, mikäli tätä sovellusta lähdettäisiin julkaisemaan yleiseen käyttöön. Tässä luvussa esitellään niistä opiskelijan näkemyksen mukaan oleellisimmat.

## 5.1 Yksikkötestit

Tuotantoon menevään ohjelmakoodia varten tulisi ehdottomasti kirjoittaa yksikkötestejä kattamaan edes sovelluksen tärkeimmät toiminnot. Yksikkötestit ovat ohjelmistokehityksen kannalta erittäin tärkeitä, sillä niiden avulla voidaan havaita ohjelman toimintaan tahattomasti syntyneitä virheitä kehitettäessä uusia ominaisuuksia. Nämä virheet saattavat monesti mennä ohitse ihmisiltä mutta eivät testejä automaattisesti suorittavalta ohjelmalta.

## 5.2 Oma domain ja sähköpostipalvelin

Palveluntarjoajan omavaraisuuden näkökulmasta olisi hyvä, että ulkoisia riippuvuuksia saataisiin karsittua pois. Tältä kannalta ajateltuna Googleen tukeutuminen sähköpostiviestien lähettämiseen voi olla riskialtis sidos, josta olisi hyvä päästä eroon. On täysin mahdollista, että Google tulee muuttamaan käyttöehtojaan, jolloin heidän sähköpostipalvelujensa käyttö tämän kaltaiseen toimintaan muuttuisi mahdottomaksi. Tämä saattaisi pahimmassa tapauksessa estää palvelun käytön kokonaisuudessaan.

Ilmeinen tapa lähteä ratkomaan tätä ongelmaa olisi oman verkkotunnuksen eli domainin hankkiminen ja sen jälkeen oman sähköpostipalvelimen pystyttäminen. Sähköpostipalvelimen pystyttämiseen löytyy verkosta paljon hyödyllistä dokumentaatiota ja monia hyviä esimerkkejä, joiden avulla toteuttaminen on varmasti harrastelijallekin mahdollista. Tämä jouduttiin kuitenkin rajaamaan pois tästä opinnäytetyöstä käytettävissä olevien resurssien säästämiseksi.

### 5.3 Käyttäjien poistaminen ja estäminen

Sääntöjen vastaisesti menettelevä tai muuten rikkomuksiin syyllistynyt käyttäjä tulisi voida pysyvästä poistaa palvelusta. Nykyisellään ainoa tapa ylläpitäjän toimesta estää käyttäjää käyttämästä sovellusta on käydä tietokannasta tuhoamassa käyttäjän User-document. Tämä ei kuitenkaan estä käyttäjää luomasta uutta käyttäjätiliä samalla käyttäjätunnuksella tai sähköpostiosoitteella.

Tätä varten sovellukseen tulisi luoda toiminto, jolla palvelusta poistetun käyttäjän sähköpostiosoite ja käyttäjätunnus voitaisiin ottaa talteen johonkin tietorakenteeseen, jota vasten uusia rekisteröitymispyyntöjä aina vertailtaisiin. Näin saataisiin rekisteröitymisvaiheessa huomattua, mikäli tällä sähköpostiosoitteella tai käyttäjätunnuksella ei ole oikeutta rekisteröidä käyttäjätiliä. Yksi vaihtoehto tähän olisi, että palvelusta poistettua käyttäjää ei poistettaisikaan tietokannasta kokonaan vaan se asetettaisiin deaktivoituun tilaan. Näin poistetun käyttäjän sähköpostiosoitteella tai käyttäjätunnuksella ei voitaisi rekisteröidä uutta tunnusta.

### 5.4 IP-osoitteiden estäminen

Nykyisellään aiemmin mainitut rate limiterit estävät lyhyeksi ajaksi tietystä IP-osoitteesta tulevat kutsut, mikäli niitä johonkin endpointtiin tulee liikaa tietyn ajan sisällä. Tässä kohtaa olisi kuitenkin vielä tarpeellista tutkia, tulisiko IP-osoitteita voida asettaa pysyvästi estolistalle, mikäli jostakin tulee niin paljon kutsuja, ettei voi olla kyse inhimillisestä virheestä vaan voidaan tulkita kyseessä olevan esimerkiksi brute force -hyökkäys. Tässä tulisi kartoittaa erilaisia tapoja toteuttaa IP-osoitteiden estämiseen.

### 5.5 Kiellettyjen käyttäjätunnusten lista

Olisi hyvä luoda jonkinlainen lista käyttäjätunnuksista, joita ei saa rekisteröidä. Nämä voisivat olla esimerkiksi suomen ja englannin kielten kirosanoja. Nodelle on saatavilla erillisiä npm:n kautta asennettavia kirjastoja, joihin sovelluskehittäjien toimesta on kerätty massiivisia listoja sanoista, jotka syystä tai toisesta on luokiteltu huonoiksi. Näiden kirjastojen käyttämistä olisi hyvä jatkokehitysideana tutkia.

### 5.6 Ylläpitäjän käyttöliittymä

Ylläpitäjää varten olisi hyvä tehdä oma käyttöliittymänsä, jonka kautta tämä voisi tehdä ylläpitoon ja hallinnointiin liittyviä toimintoja, jotta tämän ei tarvitsisi tehdä muutoksia lähdekoodiin tai käydä tietokantaa muokkaamassa suoraan. Tämä helpottaisi esimerkiksi yllä mainitun käyttäjien estämisen suorittamista. Tämä tulisi kuitenkin olemaan kohtalaisen työläs ominaisuus, sillä se vaatisi sekä muutoksia palvelimen päähän että käytännössä täysin oman käyttöliittymänsä eli clienttinsa.

### 5.7 Salasanan vaihtamisen pakottaminen

Kun käyttäjä joutuu jostakin syystä palauttamaan käyttäjätilinsä, hänelle lähetetään tällöin sähköpostiviestinä uusi palvelimen toimesta automaattisesti generoitu sähköpostiviesti. Mahdollisten vahinkojen ja väärinkäytösten vuoksi käyttäjä tulisi pakottaa vaihtamaan salasanansa heti ensimmäisellä sisäänkirjautumiskerrallaan onnistuneen palauttamisen jälkeen. Tässä on se hyöty, että käyttäjä voi muistaa paremmin uuden salasanansa, mikäli hän ei käytä salasanojen hallinnointityökalua. Uusi salasana on myös turvattu, mikäli jostakin syystä automaattisesti generoitu salasana päätyisi ulkopuolisten käsiin.

# 6 POHDINTA

Tässä opinnäytetyössä luotiin käyttäjähallinta olemassa olevaan verkkosovellukseen, josta sellainen puuttui. Toteutusta ohjasivat samat ohjelmointikielet ja teknologiat, joita on käytetty tämän projektin parissa jo alusta alkaen.

Kokonaisuudesta saatiin aikaan opiskelijan mielestä toimiva ratkaisu, joka kattaa tärkeimmät ja oleellisimmat käyttäjähallintaan kuuluvat toiminnot ja ominaisuudet. Käyttäjähallinnan rakenteessa ja mallissa pyrittiin noudattamaan alalla vallitsevia hyväksi havaittuja käytäntöjä. Opiskelijan näkemyksen mukaan tätä mallia voisi soveltaa muihinkin vastaavankaltaisiin projekteihin melko pienin muutoksin, mikä voi olla etu tulevaisuudessa. Yksi vaihtoehto voisi olla, että käyttäjähallinnan pohjalta luotaisiin täysin oma järjestelmänsä, joka on erikoistunut tarjoamaan auktorisointi- ja autentikointipalveluja muillekin sovelluksille.

Opinnäytetyössä tunnistettiin myös selkeitä parannuksen ja jatkokehityksen kohteita, joita esiteltiin tarkemmin luvussa 6. Erityistä painoarvoa tulisi antaa salaukseen ja istuntojen ylläpitämiseen liittyville ratkaisuille. Mikäli tätä projektia vietäisiin eteenpäin, tulisi projektissa käytettyjä teknisiä ratkaisuja auditoida huolellisesti ulkopuolisen tahon toimesta. Yleinen tapa sovelluskehityksessä on suorittaa sovelluksiin haavoittuvuus- eli penetraatiotestaus. Sellaisen suorittaminen tähän sovellukseen olisi ensiarvoisen tärkeätä, mikäli tätä haluttaisiin lähteä julkaisemaan yleiseen käyttöön.

Opinnäytetyössä pyrittiin myös ottamaan huomioon käyttäjähallinnan eettisiä näkökulmia perehtymällä kansallisiin ja kansainvälisiin lakeihin ja asetuksiin. Tämä perehtyminen oli tärkeätä, sillä lait ja asetukset määrittävät sekä verkkopalvelujen käyttäjille että palveluntarjoajille selkeät oikeudet ja velvollisuudet. Näiden noudattamatta jättäminen voi johtaa laillisiin seuraamuksiin, mikäli kyseessä olisi yleinen julkisesti saatavilla oleva verkkopalvelu. Tietosuojaselosteessa on opiskelijan mielestä selkeästi määritelty, mitä henkilötietoja käyttäjästä kerätään ja keräämisen syyt on perusteltu hyväksyttävästi. Näiden asioiden läpikäyminen oli myös hyödyllistä, sillä se auttoi ottamaan huomioon tietoturvas-

sa asioita ja luomaan toimintoja, joita ei välttämä olisi ymmärretty ottaa huomioon tai ymmärretty tehdä ilman tätä perehtymistä.

Opinnäytetyössä opittiin myös oman työn ohjausta ja tässä havaittiin myös oppimisen paikkoja. Opiskelija havaitsi hyväksi tavaksi kirjoittaa ensin ohjelmakoodia siihen asti, että sai tuotettua jonkin hyvä osakokonaisuuden aikaiseksi, ja tämän jälkeen kirjoittaa tehdystä toiminnallisuudesta tähän raporttiin. Tästä periaatteesta opiskelija lipsui kuitenkin helposti, kun koodaaminen niin sanotusti vei mennessään. Monesti opiskelija kirjoitti ohjelmakoodia huomattavankin paljon ennen kuin sai asiasta vietyä mitään raportin puolelle. Ohjelmakoodia kirjoitettaessa on helppoa takertua niin sanotusti lillukanvarsiin ja jäädä viilailemaan yksityiskohtia, mikäli ei pidä mielessään sovelluksen kokonaiskuvaa ja käytettävissä olevaa aikaa. On tärkeätä arvioida tarvittavan työn määrää ja tehdä selkeitä suunnitelmia ja rajavetoja niin työtä aloitettaessa kuin myös sitä tehtäessä. Näin voidaan pienentää riskiä myöhemmin tuleville yllätyksille osakokonaisuuksista, jonka kokoa ei alussa osattu arvioida. Sovelluksen monimutkaisuuden ja koon kasvaessa tarvittavan työn määrä kasvaa dramaattisesti ja korjattavaa ja parannettavaa tuntuu opiskelijan kokemuksen mukaan riittävän vaikka kuinka paljon, joten on tärkeätä osata keskittyä olennaiseen.

**LÄHTEET**

Chhetri, Chandra Panta. 17.12.2020. Sending Emails Securely Using Node.js, Nodemailer, SMTP, Gmail, and OAuth2. Luettu 3.3.2021. https://dev.to/chandrapantachhetri/sending-emails-securely-using-node-js-nodemailer-smtp-gmail-and-oauth2-g3a

Doshi, Abhi. 27.6.2019. What is full stack development? Verkkosivu. Luettu 13.3.2021. https://www.geeksforgeeks.org/what-is-full-stack-development/

Educative n.d. What is MEVN stack? Verkkosivu. Luettu 13.3.2021. https://www.educative.io/edpresso/what-is-mevn-stack

GDPR. n.d. Article 12: Transparent information, communication and modalities for the exercise of the rights of the data subject. Verkkosivu. Luettu 24.4.2021. https://gdpr.eu/article-12-how-controllers-should-provide-personal-data-to-the-subject/

GDPR. n.d. Writing a GDPR-compliant privacy notice. Verkkosivu. Luettu 24.4.2021. https://gdpr.eu/privacy-notice/

Google Developers. 28.4.2021. OAuth Playground. Verkkosivu. Luettu 23.5.2021. https://developers.google.com/google-ads/api/docs/oauth/playground

Kumar, Saket. 8.2.2018. MVC Design Pattern. Verkkosivu. Luettu 18.3.2021. https://www.geeksforgeeks.org/mvc-design-pattern/

Liedtke, Christopher. 24.6.2020. How to verify your users' email addresses | Node.js/Express. Verkkosivu. Luettu 17.3.2021. https://dev.to/christopherliedtke/how-to-verify-your-users-email-addresses-node-js-express-dg0

Martinez, Juan Cruz. 27.5.2020. When You Should and Shouldn’t Use Node.js for Your Project. Verkkosivu. Luettu 6.4.2021. https://livecodestream.dev/post/when-you-should-and-should-not-use-nodejs-for-your-project/

Mozilla. 27.4.2021. Express/Node introduction. Verkkosivu. Luettu 13.3.2021. https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction

Mungai, Geoffrey. 13.8.2020. Why is Node.js wildly popular among developers? Luettu 26.3.2021. https://www.section.io/engineering-education/why-node-js-is-popular/

Software Engineering StackExchange. 26.2.2013. Is the term 'Front-End' synonymous with 'Client-Side'? If so, is this always the case? Verkkosivu. Luettu 19.3.2021.

https://softwareengineering.stackexchange.com/questions/188521/is-the-term-front-end-synonymous-with-client-side-if-so-is-this-always-the

Software Engineering StackExchange. 18.11.2015. Is client-side validation really all that important, always? Verkkosivu. Luettu 19.3.2021. https://softwareengineering.stackexchange.com/questions/302892/is-client-side-validation-really-all-that-important-always

Sudasinghe, Anjalee. 6.4.2021. How to work with worker threads in NodeJS. Verkkosivu. Luettu 28.4.2021. https://livecodestream.dev/post/how-to-work-with-worker-threads-in-nodejs/

Suomen perustuslaki. 11.6.1999. Luettu 14.4.2021. https://finlex.fi/fi/laki/ajantasa/1999/19990731

Tietosuojavaltuutetun toimisto. n.d. Henkilötietojen käsittely. Verkkosivu. Luettu 14.4.2021. https://tietosuoja.fi/henkilotietojen-kasittely

Tietosuojavaltuutetun toimisto. n.d. Kerro käsittelystä rekisteröidylle. Verkkosivu. Luettu 14.4.2021. https://tietosuoja.fi/rekisteroidyn-informointi

Tietosuojavaltuutetun toimisto. n.d. Rekisteröidyn oikeudet. Verkkosivu. Luettu 14.4.2021. https://tietosuoja.fi/rekisteroidyn-oikeudet

Tietosuojavaltuutetun toimisto. n.d. Tietosuojaperiaatteet. Verkkosivu. Luettu 14.4.2021. https://tietosuoja.fi/tietosuojaperiaatteet

Thor, Wei-Ming. 21.1.2019. Best practices for logging in production for a Node.js application. Verkkosivu. Luettu 29.4.2021. https://twm.me/best-practices-for-logging-in-a-node-js-application/

Vivah, Linda. 28.7.2017. THE BEGINNER'S GUIDE: Understanding Node.js & Express.js fundamentals. Verkkosivu. Luettu 13.3.2021. https://medium.com/@LindaVivah/the-beginners-guide-understanding-node-js-express-js-fundamentals-e15493462be1

Vue. n.d. Comparison with Other Frameworks. Verkkosivu. Luettu 19.3.2021. https://vuejs.org/v2/guide/comparison.html