Talviuintisovelluksen suunnittelu ja toteutus

Maria Haapamäki

Opinnäytetyö, AMK
Huhtikuu 2025
Insinööri (AMK), Tieto- ja viestintätekniikan tutkinto-ohjelma

jamk | Jyväskylän ammattikorkeakoulu
University of Applied Sciences

jamk

Kuvailulehti

Haapamäki, Maria

Talviuintisovelluksen suunnittelu ja toteutus

Jyväskylä: Jyväskylän ammattikorkeakoulu. Huhtikuu 2025, 36 sivua

Tieto- ja viestintätekniikan tutkinto-ohjelma. Opinnäytetyö AMK.

Julkaisun kieli: suomi

Julkaisulupa avoimessa verkossa: kyllä

Tiivistelmä

Avantosilakka-sovelluksen tarkoitus on lisätä talviuimareiden mahdollisuuksia saada tietoa talviuintipaikoista. Etenkin vieraalla paikkakunnalla on usein hankalaa löytää sopivaa uintipaikkaa, johon myös esimerkiksi paikalliseen avantoseuraan kuulumaton on tervetullut. Idea talviuintisovelluksesta sai alkunsa saunan lauteilla talviuimareiden keskustellessa tästä ongelmasta. Toinen ongelma, joka Avantosilakalla haluttiin ratkaista, on omien uintikertojen seuranta. Monen talviuimareiden mielestä on mielekästä seurata omia uintikertoja ja -paikkoja.

Avantosilakka toteutettiin mobiilisovelluksena, koska yksinkertaisin tapa nähdä lähellä olevat uintipaikat on katsoa ne puhelimesta yhdellä vilkaisulla. Sovellus päädyttiin tekemään MERN-teknologioita hyödyntäen. Toteutus tehtiin React Native- ja Node.js-kehyksiä käyttäen. Tietokantana sovelluksessa on MongoDB. Ohjelmointikielenä käytettiin tietotyyppimuunnoksista johtuvien virheiden välttämiseksi TypeScriptiä. Uintipaikkojen ja omien uintikertojen tallentaminen toimivat sovelluksessa hyvin. Käyttäjä pystyy tarkastelemaan omia ja yleisiä uintipaikkoja karttanäkymästä. Seuraavaan versioon tehdään mahdollisuus nähdä raportti omista uintikerroista. Pohdinta-asteelle jäi vielä palveluntarjoaja, jonka palvelinta käytetään sovelluksen julkaisuun. Alustava vertailu tästä on kuitenkin jo tehty.

MERN-pino osoittautui erittäin toimivaksi työkalukokoelmaksi tällaisen mobiilisovelluksen toteutukseen. Sovellus on nopea, ja tulevien käyttäjien mielestä helppokäyttöinen ja ulkoasultaan siisti. Jos sovellus saa tarpeeksi käyttäjiä, se helpottaa tiedon saamista lähellä olevista talviuintipaikoista. Kun käyttäjä näkee tiedot nopeasti, hänen on helppo valita itselleen sopiva uintipaikka ja löytää myös reitti sinne.

Avainsanat (asiasanat)

MERN, JavaScript, MongoDB, Node.js, React Native, talviuinti, mobiilisovellus

Muut tiedot (salassa pidettävät liitteet)

-

jamk

Description

Haapamäki, Maria

Design and Implementation of the Winter Swimming Application

Jyväskylä: JAMK University of Applied Sciences, April 2025, 36 pages

Degree Programme in Information and Communication Technology. Bachelor´s thesis.

Permission for open access publication: Yes

Language of publication: Finnish

Abstract

The purpose of the Avantosilakka application is to increase winter swimmers' opportunities to get information about winter swimming places. Especially in a foreign area, it is often difficult to find a suitable swimming place where even those who do not belong to the local winter swimming club are welcome. The idea for the winter swimming application originated on the sauna benches as winter swimmers discussed this problem. Another problem that Avantosilakka wanted to solve is tracking one's own swimming sessions. Many winter swimmers find it meaningful to track their own swimming sessions and locations.

Avantosilakka was implemented as a mobile application because the simplest way to see nearby swimming places is to check them on the phone with a single glance. The application was decided to be made using MERN technologies. The implementation was done using React Native and Node.js frameworks. The database used in the application is MongoDB. TypeScript was used as the programming language. Saving swimming places and one's own swimming sessions works well in the application. The user can view their own and general swimming places on the map view. The next version will include the ability to see a report of one's own swimming sessions. The service provider for hosting Avantosilakka to a larger audience is still being decided. However, preliminary comparisons have been made.

The MERN stack proved to be a very functional toolset for implementing such a mobile application. The application is fast, easy to use and visually neat. If the application gets enough users, it will facilitate obtaining information about nearby winter swimming places. When the user sees the information quickly, it is easy for them to choose a suitable swimming place and also find the route there.

Keywords/tags (subjects)

MERN, JavaScript, MongoDB, Node.js, React Native, winter swimming, mobile application

Miscellaneous (Confidential information)

-

Sisältö

Kuviot

Taulukot

1 Johdanto

Talviuinnin suosio on viime vuosien aikana kasvanut Suomessa. Vuonna 2015 talviuintia harrasti säännöllisesti noin 100 000 suomalaista ja sen jälkeen määrä on lisääntynyt ollen nykyään jo noin 150 000 talviuimaria. Suomen Ladun (2024) teettämän tutkimuksen mukaan avantouinnista kiinnostuneita henkilöitä on vielä moninkertaisesti enemmän eli jopa yli 600 000. Talviuinnin suosiota saattavat selittää ainakin uudet tutkimukset sen terveyshyödyistä ja kylmäaltistuksen stressiä lievittävät vaikutukset. Harrastuksen suosion lisääntyminen on johtanut tarpeeseen saada lisää uusia uintipaikkoja Suomeen. (Talviuintihanke, opas talviuintipaikan kehittämiseen ja uuden talviuintipaikan perustamiseen 2024.)

Suomessa on paljon talviuintipaikkoja, mutta niistä on saatavilla tietoa hyvin vaihtelevasti. Varsinkin vieraassa kaupungissa on yleensä työlästä etsiä tarvittavat tiedot lähellä olevista uintipaikoista. Kesällä luonnon vesissä uiminen on helppoa, koska se onnistuu melkein minkä tahansa vesistön äärellä. Talviuintipaikka sitä vastoin vaatii ylläpitoa, eivätkä kaikki talviuintipaikat ole siksi edes avoinna kaikille halukkaille. Koska avantouinnin terveysvaikutukset syntyvät vain säännöllisellä kylmäaltistuksella, monella talviuimarilla on tarve myös seurata omia uintikertoja. Näitä tietojen puutteisiin ja ylläpitoon liittyviä asioita pyrittiin ratkaisemaan tällä projektilla.

Tämän työn tavoitteena oli luoda sovellus, josta talviuimarit saavat tarvitsemiansa tietoja etenkin vierailla paikkakunnilla. Avantosilakka-sovelluksen avulla uimarit jakavat tietoa avantouintipaikoista ja niiden varustelusta. Lisäksi käyttäjät pystyvät seuraamaan omia uintikertojaan sovelluksen avulla. Sovelluksesta haluttiin mobiilisovellus, koska matkapuhelimella on helpointa ja nopeinta etsiä tietoa avantouintipaikoista. Sovellukseen haluttiin myös paikannus, jotta käyttäjä näkee nopeasti häntä lähimpänä olevat uintipaikat ja saa tallennettua uintikertansa helposti. Näin ollen tämän työn isoin kysymys oli, että millä työkaluilla ja tavoilla avantouinnista saadaan vieläkin mukavampaa ja helpompaa.

Idea Avantosilakka-sovelluksesta syntyi ystäväporukan ideoinnin tuloksena vastaamaan yllä oleviin ongelmiin. Sen ensisijainen tarkoitus on helpottaa etenkin paljon matkustavien avantouimareiden harrastusta. Avantosilakan myöhempiä kaupallistamismahdollisuuksia on kuitenkin myös ideoitu. Ainakin pienimuotoisesti sille voisi löytyä yhteistyökumppaneita esimerkiksi avantouintiseuroista ja sauna- ja avantouintipalveluita tarjoavista yrityksistä ja tapahtumista.

2 Tutkimusongelma ja -menetelmä

Tämän työn tavoitteena on ratkaista talviuintipaikkojen saavutettavuusongelmaa teknologian keinoin. Tavoitteena on ratkaista ongelmaa tekemällä helppokäyttöinen ja yksinkertainen sovellus, joka toimii sekä Android- että iOS-puhelimilla. Projektin onnistumisen ja jatkokehityksen kannalta oleellista on tehdä käytettävien työkalujen ja teknologioiden valinta huolellisesti. Tärkeää on myös kerätä vaatimusmäärittelyyn ne asiat, jotka ovat sovelluksen kannalta oleellisia. Tavoitteena tässä työssä on tehdä toteutus välttämättömien ominaisuuksien osalta ja pohtia myös sovelluksen jatkokehitysmahdollisuuksia.

Tässä työssä käytetään tutkimusmenetelmänä soveltavaa tutkimusta. Soveltavalla tutkimuksella on tarkoitus ratkaista jokin käytännön ongelma hyödyntäen tutkittua teoriatietoa. Soveltavassa tutkimuksessa määritellään ensin tutkimusongelmat eli asiat, jotka tutkimuksella halutaan ratkaista. Ratkaisun etsimiseksi luodaan kehittämissuunnitelma, jonka pohjalta tehtyjä ratkaisuja testataan. Lopuksi raportoidaan tulokset ja johtopäätökset. (Rantala 2023; Soveltava tutkimus 2024.)

Yllä kuvatut avantouintipaikkoihin liittyvät ongelma rajataan seuraaviin tutkimuskysymyksiin:

-Mitkä teknologiat sopivat parhaiten tämän mobiilisovelluksen toteuttamiseen?
-Miten sovelluksen suunnittelussa huomioidaan käytettävyys?
-Millaisia jatkokehitysmahdollisuuksia on ja miten ne huomioidaan suunnittelu- ja kehitysvaiheessa?
-Voidaanko sovelluksen avulla lisätä avantouintipaikkojen saavutettavuutta?

3 Valitut teknologiat

3.1 Valintaperusteet

Ohjelmistokehityksessä on tehtävä valintoja käytettävien teknologioiden välillä. Tietoa ja mielipiteitä eri lähteistä on saatavilla paljon, mutta kaiken yksityiskohtainen opiskelu on mahdotonta. Vaikka tässä työssä on päätetty käyttää MERN-teknologioita, valintoja on pitänyt lisäksi tehdä käytettävien kirjastojen ja muiden työkalujen suhteen. Tässä työssä käytettiin muutamia pääasiallisia lähteitä, joiden perusteella tärkeimmät työkalut valittiin. Työkalujen valinnassa ja asennuksessa käytettiin apuna Biswas Nabendun kirjaa Ultimate full-stack development with Mern. Lisäksi

työssä käytettiin apuna Udemyn kurssia MERN Stack E-Commerce Mobile App with React Native. Myös React Nativen ja Expon dokumentaatiot olivat hyödyllisiä työkaluvalintoja tehdessä.

Ohjelmisto päätettiin toteuttaa mobiilisovelluksena, johon käyttäjä voi helposti merkitä avantouintipaikan sijaintinsa perusteella. Tärkein kriteeri käytettävien teknologioiden suhteen oli, että niiden pitää soveltua mobiilikehitykseen. Koska React Nativella on mahdollista kehittää alustariippumaton sovellus ja siihen on saatavilla runsaasti ohjeita ja tietoa, sovelluksen frontend -kehitykseen valikoitui React Native. Toinen tärkeä kriteeri oli, että sovelluksen kehitykseen haluttiin käyttää teknologioita, jotka olivat opinnäytetyön tekijälle jo ennestään tuttuja ja hyväksi havaittuja. Tämän vuoksi päätettiin hyödyntää MERN-pinoa. MERN pitää sisällään kaiken tarvittavan, kun halutaan luoda nykyaikainen ja tehokas JavaScript-pohjainen ohjelmisto.

3.2 MERN-pino

Ohjelmistokehityksessä pinolla tarkoitetaan tiettyjä työkaluja ja teknologioita, joita käytetään yhdessä (Bakker & Sermon 2023). MERN-pino tarkoittaa sovelluskehitykseen käytettyä teknologiakokonaisuutta, joka muodostuu JavaScript-kirjastoista ja sisältää MongoDB-tietokannan, Expressin, Reactin ja Node.js:n. Näillä teknologioilla työhön saatiin tehtyä frontend- ja backend-osiot, tietokanta sekä yhteydet näiden välillä. MERN-pinoa käytettäessä ainakin lähes kaikki ohjelmakoodi kirjoitetaan JavaScriptillä (tai TypeScriptillä), mikä vähentää projektin monimutkaisuutta ja sitä kautta virheitä. Ohjelmakoodin kirjoittaminen vain yhdellä kielellä lisäsi ainakin tässä projektissa suoraviivaisuutta ja helppoutta. (Biswas 2023, 1.)

MongoDB on NoSQL-tietokanta, jonka data tallennetaan BSON-muodossa. MongoDB:n hyviin puoliin kuuluu nopeus ja kyky selviytyä useista yhtäaikaisista kyselyistä. Express kuuluu backend-osioon ja sitä käytetään API-kutsujen luomiseen. Node.js:ää käytetään backend-kehitykseen. Node.js:n avulla myös backend-koodi saadaan kirjoitettua JavaScriptillä. MERN:ssä frontend-kehitys tehdään React-kehyksellä. Tässä työssä käytettiin Reactista React Native -versiota, joka soveltuu alustariippumattomaan mobiilikehitykseen. Reactissa ohjelmistokoodi kirjoitetaan komponenteiksi, joita ladataan halutulla tavalla näytettäväksi käyttöliittymässä. Näin saadaan toteutettua nopea ja suorituskykyinen käyttöliittymä. Työssä käytettiin JavaScriptin sijaan TypeScriptiä, koska

tietotyypitetty ohjelmistokieli aiheuttaa vähemmän virheitä kuin JavaScriptin automaattiset tyyppimuunnokset (Biswas 2023, 1-5; Oraskari 2023.)

MERN-pinolle on olemassa muitakin samantapaisia vaihtoehtoja. MEAN- ja MEVN-pinoissa käytetään muuten samoja teknologioita mutta MEAN:ssa React on korvattu Angularilla ja MEVN:ssä Vue.js:llä (Biswas 2023, 6). Tässä työssä MEAN tai MEVN eivät olleet vaihtoehtoja sen takia, että kyseessä on mobiilisovellus, jonka tekemiseen React Native on ylivoimainen aktiivisen ylläpidon, laajojen kirjastojen ja saatavilla olevan tiedon takia. PERN-pino on hyvä vaihtoehto, jos halutaan käyttää MongoDB:n sijaan sql-tietokantaa. PERN-pinoa käytettäessä tietokantana on PostgreSQL. Avantosilakka ei tarvitse monimutkaisia tietokantakyselyjä, joten nopea ja joustava MongoDB oli paras tietokantaratkaisu tähän projektiin. Avantosilakka-projektia aloitettaessa harkittiin yhdistelmää, jossa backend olisi tehty C#:lla ja frontend React Nativella kuten nytkin. MERN-teknologioiden yhteensopivuus ja joustavuus olivat kuitenkin lopulta ratkaiseva tekijä. (Biswas 2023, 10-11; MERN Stack Alternatives: Top Choices of 2025 2025.)

3.3 RESTful

RESTful tarkoittaa arkkitehtuuria, jolla suurin osa nykyaikaisista web-sovelluksista rakennetaan. REST-arkkitehtuurissa tieto haetaan tietokannasta mutta nämä tietokantakyselyt eivät tapahdu suoraan ohjelmasta. Kyselyt muodostetaan http:n kautta hyödyntäen API-kutsuja. Nämä kutsut voivat yleisimmin olla GET-, POST-, PUT- ja DELETE-tyyppisiä. API-kutsuilla tietoa siis haetaan kannasta, tallennetaan sinne takaisin sekä muokataan ja poistetaan. (Biswas 2023, 6.)

API-kutsuja voidaan kuvata CRUD-toimintoina. CRUD muodostuu sanoista create, read, update ja delete. POST-tyyppinen API-kutsu vastaa CRUD:n create-osuudesta. POST-kutsua käytetään sovelluksessa esimerkiksi heti siihen rekisteröidyttyä. Tällöin käyttäjä antaa ohjelmalle tietoja, jotka välitetään post-kutsua käyttäen tietokantaan asti. GET-kutsu on read-tyyppinen eli se hakee tietokannasta tietoa. Avantosilakka-sovelluksessa GET-kutsua käytetään esimerkiksi uintipaikkojen näyttämiseen. PUT-tyyppistä kutsua käytetään jo olemassa olevien tietojen muokkaamiseen. DELETE-kutsulla poistetaan tietokannasta tietoa. (Biswas 2023, 6-10.)

4 Sovelluksen suunnittelu- ja valmistelutyöt

4.1 Vaatimusmäärittely

Ohjelmiston vaatimusmäärittelyssä kartoitetaan eri sidosryhmien toiminnalliset ja ei-toiminnalliset vaatimukset ohjelmistoa kohtaan. Vaatimusmäärittelyn tarkoituksena on onnistua tekemään laadukasta ohjelmistoa ja ratkaisemaan juuri ne asiat, jotka ohjelmistolla halutaan ratkaista. Alfamen Vaatimusmäärittelyoppaan (2021) mukaan erityisen tärkeää on sidosryhmien kartoittaminen ja ymmärtäminen, koska heitä varten toteutusta ollaan tekemässä. (Paakki 2011; Vaatimusmäärittelyopas 2021.)

Vaatimusmäärittely valmistuu usein iterointiprosessina, jossa toistuu esimerkiksi vaatimusten kartoittaminen, analysointi, validointi ja dokumentointi (Luukkainen 2022). Avantosilakka-sovelluksen vaatimusmäärittely lähti liikkeelle saunan lauteilta, jossa pohdittiin yhdessä, millaisia ominaisuuksia sovelluksessa tulisi olla. Näitä ominaisuuksia kehitettiin, palloteltiin ja lopulta dokumentoitiin vaatimuksiksi. Vaatimusten päätyminen vaatimusmäärittelydokumenttiin asti vaatii analysointia. Jotkut vaatimukset saattavat olla mahdottomia toteuttaa käytössä olevilla resursseilla. Jotkut vaatimukset saattavat olla myös ristiriidassa keskenään. Vaatimusten validointi tarkoittaa, että varmistetaan tilaajalta, että ominaisuudet ovat juuri sellaisia kuin tilaaja on halunnut. Valmiiseenkin vaatimusmäärittelyyn saattaa tulla vielä ohjelmistoprojektin myöhemmissä vaiheissa muutoksia ja nämä muutokset dokumentoidaan vielä uudestaan. Avantosilakka-projektissa vaatimusten validointi tapahtui vaatimusten dokumentoinnin jälkeen tarkempia malleja paperille piirtäen. Projektin edetessä vaatimusmäärittely muuttui hieman. Muutokset eivät olleet suuria vaan lähinnä esimerkiksi priorisointijärjestykseen liittyviä. (Luukkainen 2022.)

4.1.1 Sidosryhmät

Avantosilakka-sovelluksen sidosryhmiä ovat talviuimarit sekä myös avantouinnista kiinnostuneet henkilöt, jotka kartoittavat avantouintipaikkoja ja kokemuksia niistä. Talviuimareissa on monenlaisia ihmisiä. Osa haluaa päästä saunaan, ja joku toinen ei koskaan käy uinnin yhteydessä saunassa. Joku avantouimari haluaa mahdollisimman suuren avannon, jossa pystyy uimaan kunnolla ja toiselle riittää mahdollisuus kastautua kylmässä. Koska talviuimarit ovat niin heterogeeninen ryhmä,

uintipaikoista tarvitaan etukäteen tietoa. Myös talviuintipaikkojen ylläpitäjät, kuten avantouintiseurat ja kaupalliset yritykset, ovat Avantosilakan sidosryhmiä. Talviuintipaikat saavat sovelluksen kautta näkyvyyttä ja palautetta. Alla olevassa taulukossa on listattuna sidosryhmät ja niiden tarpeet sovelluksen suhteen.

Taulukko 1. Avantosilakan sidosryhmät

Käyttäjätarina	Käyttäjä
Käyttäjänä haluan saada tietoa lähellä olevien uintipaikkojen aukioloista, hinnoista, saunomismahdollisuudesta, pukuhuoneista jne.	Talviuimari
Käyttäjänä haluan pystyä käyttämään sovellusta helposti ja siten että tiedän ilman ohjeidenkin lukemista, miten sovellus toimii.	Talviuimari
Käyttäjänä haluan, etteivät muut käyttäjät näe minun tietojani.	Talviuimari
Käyttäjänä haluan nähdä tietoja omista uintikerroistani.	Talviuimari
Avantouintipaikan ylläpitäjänä haluan, että uimarit saavat helposti tietoa paikasta.	Avantouintipaikan ylläpitäjä
Avantouintipaikan ylläpitäjänä haluan saada palautetta siitä mikä toimii ja mikä ei.	Avantouintipaikan ylläpitäjä
Sovelluksen ylläpitäjänä haluan, että sovellus on tietoturvallinen ja että sisältö pysyy asiallisena	Sovelluksen ylläpitäjä

4.1.2 Toiminnalliset vaatimukset ja ei-toiminnalliset vaatimukset

Vaatimusten dokumentointiin on olemassa erilaisia tapoja. Yleistä on esimerkiksi tehdä vaatimuksista listoja, UML-kaavioita ja/tai käyttäjätarinoita (Luukkainen 2022). Toiminnallisilla vaatimuksilla tarkoitetaan niitä vaatimuksia, jotka liittyvät siihen, mitä järjestelmä tekee ja miten se vaikuttaa ympäristöönsä. Toiminnalliset vaatimukset ovat järjestelmän käyttäjälle näkyviä ominaisuuksia. Ei-toiminnalliset vaatimukset eivät välttämättä ole ohjelmiston näkyviä ominaisuuksia vaan liittyvät ennemminkin siihen, miten edellä mainittuja toiminnallisia vaatimuksia toteutetaan. Ei-toiminnalliset vaatimukset saattavat liittyä esimerkiksi tietoturvaan tai suorituskykyyn. (Paakki 2011; Vaatimusmäärittely 2021.)

Avantosilakka-sovelluksen vaatimukset on dokumentoitu toiminnallisten ja ei-toiminnallisten vaatimusten listoina. Vaatimusten prioriteetti on merkitty värikoodeilla, joista tummin väri tarkoittaa,

että nämä vaatimukset on ehdottomasti saatava mukaan ensimmäiseen versioon. Seuraavaksi tummin väri tarkoittaa, että nämä vaatimukset olisi hyvä saada mukaan ensimmäiseen versioon, mutta se ei ole välttämätöntä ohjelmiston toiminnan kannalta. Vaalein väri tarkoittaa, että kyseinen vaatimus voi odottaa seuraavaan versioon, eikä se estä ohjelmiston julkaisua esimerkiksi pilottikäyttöä varten.

Taulukko 2. Avantosilakan toiminnalliset vaatimukset

Vaatimus	Lisätiedot
Mobiilisovellus	
Paikannus	Käyttäjän täytyy pystyä näkemään lähellä olevat uintipaikat helposti.
Mahdollisuus tallentaa uintipaikka julkisena tai yksityisenä	Esimerkiksi omassa rannassa tehtyjä uinteja ei haluta julkisesti näkyville mutta uintikertojen seuraamiseksi tieto pitää pystyä tallentamaan.
Yleisten uintipaikkojen kommentit	Käyttäjät pystyvät tallentamaan kommentteja julkisille uintipaikoille. Muut käyttäjät pystyvät katsomaan näitä kommentteja.
Omien tietojen tallennus ja muokkaus	Käyttäjä pystyy esimerkiksi vaihtamaan oman salasanansa tai sähköpostiosoitteensa.
Omien tietojen näkyvyys	Käyttäjän täytyy pystyä seuraamaan omia uintikertojaan ja -paikkojaan.

Taulukko 3. Avantosilakan ei-toiminnalliset vaatimukset

Vaatimus	Lisätiedot
Tietoturva	Tietokannan ja käyttäjien tietojen suojaus täytyy olla kunnossa.
Jatkokehitysmahdollisuudet	Ohjelmiston suunnittelussa ja toteutuksessa on huomioitava skaalautuvuus.
Helppokäyttöisyys ja yksinkertaisuus	Käyttäjien tulee ymmärtää heti mistä on kyse ja osata käyttää sovellusta ilman ohjevihkosia.
Autentikointi	Käyttäjän tulee pystyä kirjautumaan omalla tunnuksellaan. Myös uloskirjautumisen tulee onnistua.
Autorisointi	Tavallinen käyttäjä saa nähdä vain omat tietonsa ja yleiset uintipaikat. Admin-käyttäjällä on oikeudet hallinnoida sovellusta.
Sanitointi	Input-kentissä tulee olla sanitointi viimeistään siinä vaiheessa, kun sovellus laitetaan palvelimelle.
Admin-valikko	Tarvitaan Admin-taso käyttäjätunnuksiin. Admin käyttäjä voi esimerkiksi moderoida sovellusta.
Automatisoidut moderointitoimenpiteet	Sovelluksessa tulee olla ainakin ’huonojen sanojen’ filtteröinti

4.2 Hyvät ohjelmointikäytännöt

Hyvien ohjelmointikäytäntöjen huomioiminen tässäkin projektissa on tärkeää, jotta ohjelmiston tehokkuus- ja käytettävyysvaatimukset täyttyvät. Nykyaikaisessa web-ohjelmointiprojektissa tarvitaan useita työkaluja ja kirjastoja, mikä lisää ohjelmointityön monimutkaisuutta. Rafalski (2025) on tehnyt käyttäjälähtöisen listauksen asioista, joihin web-ohjelmointiprojektissa tulee kiinnittää huomiota, jotta varmistetaan onnistunut ja laadukas lopputulos. Rafalskin (2025) mukaan projektin suunnittelu ja käyttäjälähtöisyys on tärkeää. Helppokäyttöinen ohjelmisto toimii nopeasti, on tietoturvallinen ja sen navigointi on loogista. Käyttäjän tulisi pystyä löytämään helposti tarvitsemansa näkymät ja linkit muihin näkymiin. Hyvälaatuisella ohjelmointikoodilla saadaan vähennettyä virheitä ja lisättyä ylläpidettävyyttä. Koodin luettavuutta lisäävät esimerkiksi loogiset nimeämiskäytännöt ja kommenttien lisääminen koodiin. Laadunvarmistuksen kannalta tärkeää ohjelmistoprojektissa on testaaminen, jota tulisi tehdä koko ajan, kun muutoksia tehdään. Testausta tulisi tehdä

eri selaimilla ja eri laitteilla. McKee (2023) käy läpi samoja teemoja kertoessaan hyvistä ohjelmointikäytännöistä. Lisäksi hän neuvoo käytäntöjä datan tehokkaaseen käsittelyyn. Turhia silmukoita tulisi välttää hyvän suorituskyvyn varmistamiseksi. Samasta syystä taulukoiden ja suurten aineistojen käsittelyyn ja tulee kiinnittää huomiota. Ohjelmointikielissä ja työkaluissa on usein valmiita tehokkaita metodeja datan käsittelyn optimoimiseksi. (McKee 2023; Rafalski 2025.)

Avantosilakka-projektin laadunvarmistuksessa pyrittiin tekemään jatkuvaa testausta pientenkin muutosten yhteydessä. Tämä helpotti myös projektin etenemistä, koska jatkuvalla testaamisella oli helppo päästä virheen jäljille niissä tilanteissa, kun jokin ei enää toiminutkaan. Avantosilakan ohjelmointikoodi on pyritty kirjoittamaan mahdollisimman yksinkertaiseksi, ja siinä on pyritty välttämään turhia silmukoita ja monimutkaisia if-rakennelmia. Tällä tavoin toimimalla koodi pysyy helppolukuisena eli paremmin ylläpidettävänä. Myös virheiden todennäköisyys pienenee.

4.3 Valmistelu

4.3.1 Ohjelmointiin liittyvät työkalut

Backend-kehitystä varten tämän projektin valmistelu aloitettiin Node.js:n ja npm:n (node package manager) asentamisella. Samalla ladattiin backend-projektiin myös tietokantayhteyttä ja API-kutsuja varten Express- ja Mongoose-kirjastot. MongoDB-tietokanta oli helpointa perustaa suoraan Mongo Atlas -pilvipalveluun. Koodieditorina tässä projektissa käytettiin Visual Studio Codea. (Bakker & Sermon 2023.)

React Nativea käyttöön otettaessa on valittava, halutaanko käyttää React Native CLI:tä vai Expoa. Tässä työssä valinta tehtiin sen perusteella, että Expo on aloittelijaystävällisempi ja siinä on muutamia projektin kannalta hyödyllisiä ominaisuuksia. Expo-projektin aloitus oli helppoa, eikä asetuksia tarvinnut määritellä kovin paljon. Expossa on olemassa esimerkiksi helppo build-työkalu ja mahdollisuus testata kehitettävää ohjelmistoa mobiililaitteella. Tällaiset ominaisuudet helpottavat ohjelmistokehitystä. Toisaalta, jos tarvitaan työkalua, jonka hallinnointia ja asetuksia kehittäjä pääsee itse tekemään enemmän, React Native CLI on parempi ratkaisu. Tässä työssä kyse on sen verran pienen kokoluokan projektista, että Expo sopi hyvin. (Bakker & Sermon 2023; Expo vs React Native CLI: Key Differences Explained 2024.)

4.3.2 Testaustyökalut ja versionhallinta

Usein ohjelmistoprojekti aloitetaan backendin ja tietokannan sekä niiden välisten kutsujen luomisella. Näitä yhteyksiä ja kutsuja täytyy päästä kuitenkin testaamaan ennen kuin käyttöliittymä on olemassa. Siksi koneelle asennettiin Postman-sovellus, jolla API-kutsuja voi testata. (Bakker & Sermon 2023; Biswas 2023, 12.)

Mobiilikehityksessä on tärkeää päästä testaamaan sovellusta fyysisellä laitteella tai ohjelmalla, joka jäljittelee fyysistä laitetta. Bakker ja Selmon (2023) suosittelevat asentamaan kehitysprojektia varten Android Studion ja iOS Simulatorin, joilla päästään testaamaan sovellusta jäljitellen oikeaa laitetta. Käytännössä Android studiota tarvittiin tässä projektissa lähinnä eri kokoisten näyttöjen testausta varten. Koska kyseessä on Expo-projekti, kehitettävän sovelluksen testaaminen fyysisillä laitteilla on hyvin helppoa. Projektia varten käytössä oli testausta varten sekä Android- että iOS-laitteita.

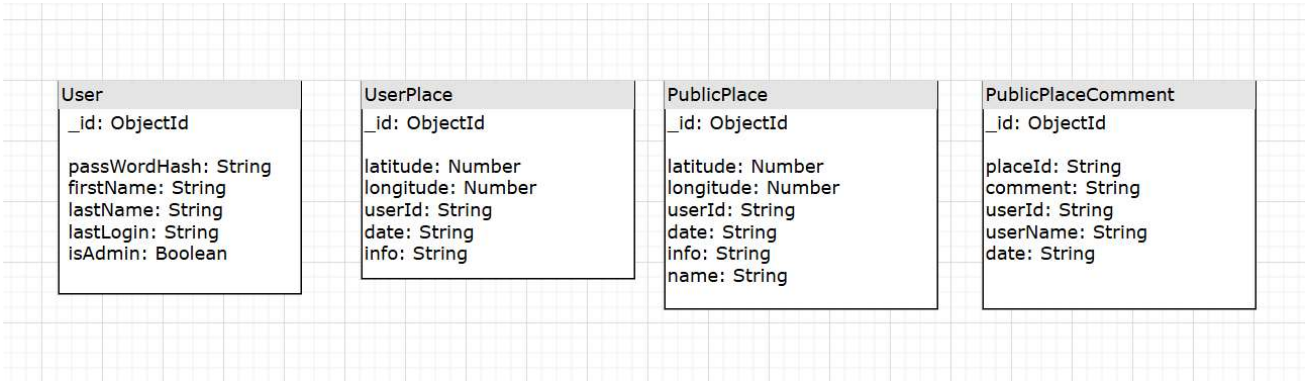
Heti projektin alussa kannattaa myös perustaa projekti versionhallintatyökaluun. Versionhallintatyökalu tarvitaan ainakin, jos projektissa on mukana useampia kehittäjiä. Tällöin kaikki ohjelmakoodi pysyy hallitusti yhdessä paikassa. Versionhallinta toimii myös varmuuskopiona. Tässä työssä versionhallintatyökaluksi valikoitui GitHub sen helppouden vuoksi. Vaikka kehittäjiä oli vain yksi, työn lopputulos oli tärkeää jakaa esimerkiksi tähän raporttiin. GitHubiin on perustettu projekti Talviuintisovellus, joka tällä hetkellä näkyy julkisena. (Versionhallinta n.d.)

4.3.3 Tietokannan suunnittelu

NoSQL-tietokanta on joustavampi kuin perinteinen sql-tietokanta. Jos tietokantarakenne jossain vaiheessa muuttuu, MongoDB-tietokantaa on helpompaa muuttaa kuin perinteistä sql-tietokantaa. Suunnittelu kuitenkin on tärkeää, jotta osataan rakentaa oikeanlaiset API-kyselyt ja tiedostot backendiin. Tässä projektissa on alkuvaiheessa neljä kokoelmaa, joita sql-tietokannassa sanottaisiin tauluiksi. Käyttäjätietoja varten perustetaan User-kokoelma, johon lisätään myös isAdmin-tieto. Tätä tietoa hyödynnetään, kun sovellukseen tehdään ylläpitoa varten järjestelmänhallinta-valikko. UserPlace-kokoelmaan tallennetaan käyttäjän uintikerrat paikkatietoineen. PublicPlace-kokoelma on yleisiä uintipaikkoja ja sen tietoja varten. Yleiset uintipaikat näkyvät kartalla kaikille.

Yleisiin uintipaikkoihin liittyvät käyttäjien kommentit tallennetaan PublicPlaceComment-kokoelmaan.

Tietokannan kokoelmia ei varsinaisesti liitetä yhteen. UserId kuitenkin tallennetaan kaikkiin kokoelmiin, jotta tiedetään, kuka on tehnyt toimenpiteen. PublicPlaceComment-kokoelmaan tallennetaan myös placeId, jotta sovelluksessa saadaan haettua kaikki tietyn yleisen uintipaikan kommentit. Myöhemmin sovellukseen lisätään tarvittaessa kokoelmia esimerkiksi järjestelmän ylläpitoa ja tilastointia varten. Kokoelmiin on myös helppo lisätä kenttiä projektin edetessä.



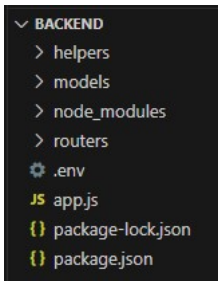
Kuvio 1. Tietokannan kokoelmat

5 Sovelluksen toteutus

5.1 Backendin ohjelmointityö

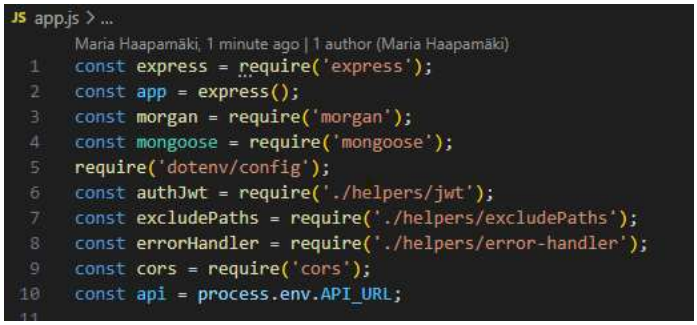
Avantosilakka-sovelluksen backend-hakemiston kansiorakenne on alla olevan kuvan mukainen. App.js tiedostossa on tärkeimmät määrittelyt ohjelmiston käynnistykseen, reitteihin ja eräisiin tietoturva-asioihin liittyen. Muut tiedostot on jaettu omiin kansioihinsa siten, että esimerkiksi API-

kutsuihin liittyvät reitit ja mallit ovat omissa kansioissaan routers ja models. Helpers-kansiossa on esimerkiksi virheenkäsittelyyn, tietoturvaan ja kirjautumiseen liittyviä tiedostoja.



Kuvio 2. Backendin kansiorakenne

Backendin ohjelmointi aloitettiin perustamalla app.js-tiedosto, johon rekisteröitiin tarvittavat kirjastot. Tässä projektissa välttämätön osa ohjelmiston toimintaa on Express, jota käytetään esimerkiksi reittien luomiseen ja ohjelmiston käynnistämiseen oikeassa portissa. Alla olevassa kuvassa näkyy, että Express on rekisteröity app.js-tiedostoon muuttujana const express. Samalla on perustettu muuttuja const app, joka kutsuu express-funktiota. Kun Express on tällä tavalla rekisteröity käyttöön, voidaan käynnistää serveri komennolla app.listen(). Tämän projektin kehitysvaiheessa backend-palvelimena toimii siis kotikone ja siihen saa yhteyden samalla koneella osoitteessa http://localhost:3000.



Kuvio 3. App.js-tiedostoon määritellyt kirjastot

```
53  app.listen(3000, () => {
54      console.log(api);
55      console.log('Server is running at http://localhost:3000');
56  });
```

Kuvio 4. Palvelimen käynnistys app.js-tiedostossa

Tietokantayhteyttä varten asennettu Mongoose otettiin käyttöön samaan tapaan kuin Express. App.js-tiedostoon perustettiin muuttuja const mongoose, joka sisältää Mongoose-objektin. Sen jälkeen määriteltiin tietokantayhteys samaan app.js-tiedostoon. Alla olevassa kuvassakin näkyvä tietokantayhteyden connection string sisältää arkaluontoista tietoa, kuten tietokannan yhteystiedot ja salasanan. Siksi tämä tieto tuodaan env-tiedostosta erikseen.

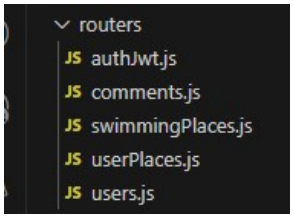
```
40  // Database connection
41  mongoose.connect(process.env.CONNECTION_STRING, {      Maria Haapamäki
42      dbName: 'Avantosilakka'
43  })
44  .then(() => {
45      console.log('Database Connection is ready');
46  })
47  .catch((err) => {
48      console.log(err);
49  });
```

Kuvio 5. Tietokantayhteyden luonti app.js-tiedostossa

Projektin backendin app.js-tiedostossa on edellä mainittujen kohtien lisäksi määrittelyt reiteille ja reitteihin liittyvät middleware-funktiot. Middleware-funktiot pystyvät käsittelemään sekä request- että response objekteja ja myös muokkaamaan niitä. Nämä funktiot toimivat tässä tapauksessa käyttäjän ja tietokannan välissä. Alla olevassa kuvassa näkyy, miten reitit ja middleware-funktiot on määritelty app.js-tiedostossa. (Using Middleware n.d.)

Kuvio 6. Reittien ja middleware-funktioiden määrittelyt

Reitit ovat niitä päätepisteitä, joihin käyttäjän tekemät kyselyt kulkeutuvat. Reitityksen avulla määritellään siis mikä toiminto halutaan suorittaa eli mitä tietoa halutaan hakea, tallentaa, poistaa tai muokata. Alla olevassa kuvassa näkyy Avantosilakan viisi reititykseen liittyvää tiedostoa. Jokaista neljää tietokantakokoelmaa kohti on oma reittitiedostonsa ja lisäksi autentikointiin liittyvä authJwt.js-tiedosto. (Routing n.d.)



Kuvio 7. Routers-hakemiston tiedostot

Alla olevassa kuvassa on näkyvissä esimerkki users.js-tiedostosta, jossa käytetään Expressin Router-ominaisuutta. Router-ominaisuutta käytetään kaikissa Avantosilakka-projektin backend-reiteissä. Router.get-metodin avulla voidaan hakea listaus kaikista käyttäjistä. Router.post-metodeja käytetään esimerkiksi kirjautumiseen ja uuden käyttäjän rekisteröitymiseen. Reitille annetaan parametrina ’/login’, jos kyseessä on kirjautuminen ja ’/signup’, jos kyseessä on rekisteröityminen. Kuvassa näkyy signup-metodi, jolla siis rekisteröidään uusi käyttäjä. Metodi saa käyttöliittymältä tarvittavat käyttäjätiedot json-muodossa osana http post -pyyntöä. Tästä post-pyynnöstä esimer-

kiksi käyttäjänimi saadaan purettua kirjoittamalla: ”userName: req.body.userName”. Signup-metodi käyttää myös käyttäjän antamaa salasanaa, mutta tekee siihen hash-suojauksen ennen tallennusta tietokantaan. Tämä suojaus tehdään käyttämällä bcrypt-kirjastoa. Metodin kohta user.save tallentaa uuden käyttäjäprofiilin tietokantaan.

```
const User = require('../models/user')
const express = require('express')
const router = express.Router()
const bcrypt = require('bcryptjs')
const jwt = require('jsonwebtoken')

router.get(`/`, async (req, res) => {
    const userList = await User.find()

    if(!userList) {
        res.status(500).json({success:false})
    }
    res.send(userList)
})

router.post(`/signup`, (req, res) => {
    const user = new User({
        userName: req.body.userName,
        passwordHash: bcrypt.hashSync(req.body.password, 10),
        firstName: req.body.firstName,
        lastName: req.body.lastName,
        lastLogin: req.body.lastLogin,
        isAdmin: req.body.isAdmin
    })
    user.save().then((createdUser=> {
        res.status(201).json(createdUser)
    })).catch((err) => {
        res.status(500).json({
            error:err,
            success: false
        })
    })
})
module.exports = router      You, 5 days ago • Uncommitted change
```

Kuvio 8. Esimerkki reittitiedostosta

Reitille haetaan aina myös kyseistä reittiä vastaava malli. Users-tiedostoon on haettu User-malli yllä olevassa kuvassa näkyvällä tavalla eli määrittelemällä const User = require(’../models/user’). Malli sisältää kaikki ne tiedot, joita halutaan hakea tietokannasta tai tallentaa tietokantaan. Lisäksi mallissa on funktio userSchema.virtual(’id’).get, jolla muunnetaan MongoDb objectId string-tyyppiseksi. String-tyyppistä id-tietoa on helpompi käsitellä. Alla olevassa kuvassa näkyy userSchema-malli, joka exportataan nimellä User ja sitä voidaan käyttää esimerkiksi reititystiedostoissa.

```
const mongoose = require('mongoose');

const userSchema = mongoose.Schema({
    userName: {
        type: String,
        required: true
    },
    passwordHash: {
        type: String,
        required: true
    },
    firstName: {
        type: String,
        required: true
    },
    lastName: {
        type: String,
        required: true
    },
    lastLogin: {
        type: String,
        default: (Date.now).toString()
    },
    isAdmin: {
        type: Boolean,
        required: true
    }
});

userSchema.virtual('id').get(function () {
    return this._id.toHexString()
})

userSchema.set('toJSON', {
    virtuals:true,
})
```

Kuvio 9. User-malli

5.2 Tietoturva backendissä

Tietoturva Avantosilakan backendissä liittyy esimerkiksi autentikointiin ja tietokantayhteyteen. Avantosilakan backendissä käytetään ympäristömuuttujia, joiden avulla saadaan sensitiivinen tieto ylläpidettyä asianmukaisesti varsinaisen koodin ulkopuolella. Tämän ominaisuuden käyttöönotto on yksinkertaista. Tässä projektissa ladattiin dotenv-kirjasto ja tehtiin .env-tiedosto, johon tarvittavat muuttujat luotiin. Tällaisia muuttujia olivat esimerkiksi tietokantayhteyteen ja tokeniin liittyvät asiat.

Avantosilakassa käytetään tokenia, jonka tarkoitus on estää asiaton pääsy ohjelmaan. Vain kirjautunut käyttäjä, jolla on rekisteröity tunnus, voi päästä sovellukseen. Token otettiin käyttöön asen-

tamalla tarvittava kirjasto komennolla npm install jsonwebtoken. Asennuksen jälkeen tokenia varten perustettiin oma tiedosto jwt.js, jossa määriteltiin seuraavasti: const jwt = require('jsonwebtoken'); Tämä tiedosto sisältää myös tokenin tarkistuksen sovelluksen reiteille.

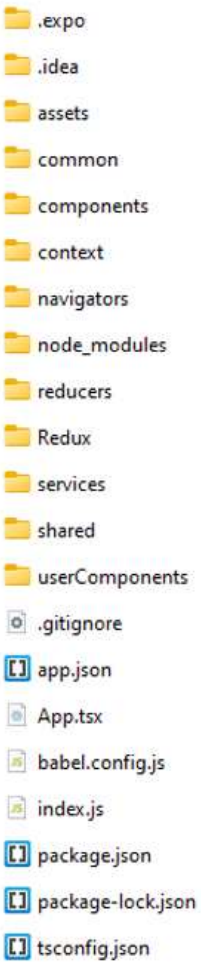
Alla olevassa kuvassa näkyy tokenin käyttö app.js-tiedostossa. ExcludePaths-muuttuja määrittelee ne reitit, joilla tokenia ei tarkisteta. Token luodaan sisäänkirjautumisen yhteydessä, eikä sitä tarkisteta sisäänkirjautumis- tai rekisteröitymissivuilla, koska siinä vaiheessa sitä ei vielä kuulukaan olla olemassa.

Kuvio 10. Tokenin käyttöönotto app.js-tiedostossa

Avantosilakka käyttää salasanojen suojaukseen bcrypt-kirjastoa, jonka avulla salasanoihin tehdään hash-salaus. Näin ollen salasanoja ei koskaan tallenneta sellaisenaan, vaan ne pysyvät salassa, vaikka joku onnistuisikin saamaan ne käsiinsä.

5.3 Frontendin ohjelmointityö

Ohjelmiston käyttöliittymän tiedostot ovat omassa hakemistossaan, joka on jaettu alla olevan kuvan mukaisiin kansioihin. Navigointiin eli näkymästä toiseen siirtymiseen liittyvät tiedostot ovat navigators-kansiossa. Autentikointiin ja käyttäjätietojen näkymiseen liittyvä koodi on userComponents-nimisessä kansiossa. Suurin osa tiedostoista on erilaisia ohjelmiston käyttämiä komponentteja, kuten eri näkymiä ja niiden osia. Nämä tiedostot löytyvät components-kansiosta. Services-kansio sisältää kaikki komponenttien käyttämät axios-kutsut, joiden avulla otetaan yhteys backendiin ja haetaan, tallennetaan tai muokataan tietoa.

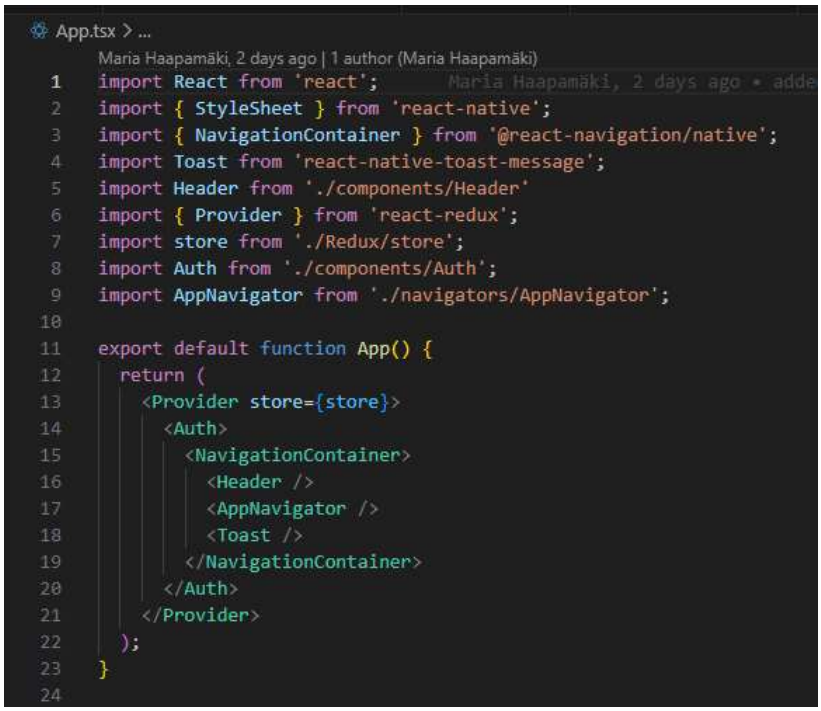


Kuvio 11. Frontendin kansiorakenne

Tyypillisessä React native -tiedostossa on yläreunassa tarvittavien komponenttien ja kirjastojen rekisteröinti, joka toteutetaan erimerkiksi seuraavanlaisella tavalla: *import { StyleSheet } from 'react-native'.* Tässä esimerkissä tiedostoon tuodaan käytettäväksi tyyliluokkien määrittelyyn tarvittava stylesheet-ominaisuus React Nativesta. Seuraavaksi tiedostossa määritellään funktio, joka voidaan viedä komponentiksi muihin haluttuihin paikkoihin sovelluksessa. Tämän funktion sisälle kirjoitetaan haluttu JavaScript- tai TypeScript-koodi. Lopuksi määritellään funktion return-osuus, jossa kerrotaan, mitä käyttäjälle näytetään. Tässä osiossa voidaan käyttää html-tageja, kuten <Text> tai <Button>. Näiden jälkeen tiedoston loppuun voidaan halutessa vielä määritellä tyyliluokkia.

Avantosilakan frontend-ohjelmointi aloitettiin muokkaamalla App.txs-päätiedostoa. Tämän tiedoston return-osiossa kerrotaan, mitä käyttäjän ruudulla näytetään, kun hän avaa sovelluksen. Tässä

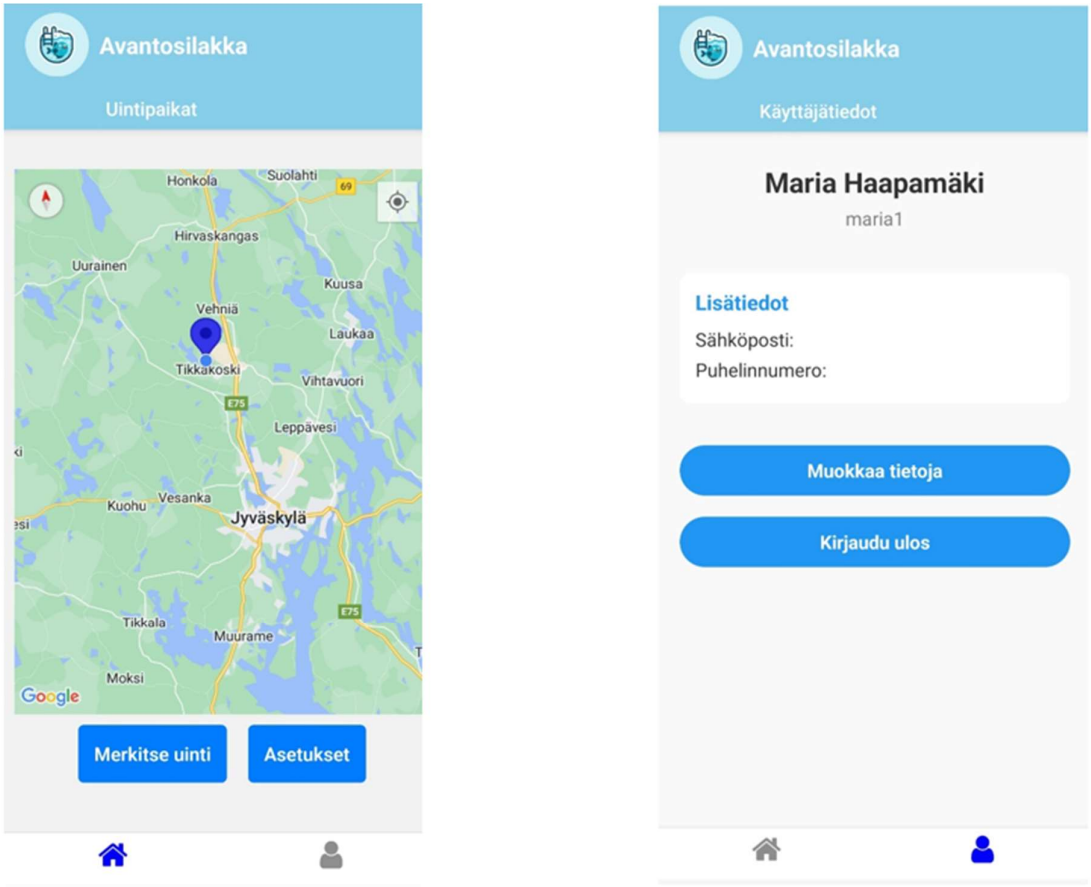
projektissa App-tiedoston return-osio toimii siten, että siinä määritellään aluksi Auth-komponentti, jonka sisällä on navigointi. Tämä tarkoittaa, että kun käyttäjä avaa sovelluksen, tarkistetaan aina, onko käyttäjä kirjautunut sisään. Sen perusteella suoritetaan navigointi. Jos sovellus ei löydä käyttäjää, avataan kirjautumisnäkymä. Jos kirjautunut käyttäjä löydetään, sovellus avautuu pääsivulle. Header-komponentti App-tiedostossa tarkoittaa, että jokaisella sivulla näytetään samalla tavalla määritelty otsikko, joka sisältää sovelluksen nimen ja logon. Toast-komponentti on erillinen kirjasto, jossa saadaan ohjelmiston ilmoitukset näkyviin. Tällaisia ilmoituksia ovat erilaiset virheilmoitukset ja ilmoitukset onnistuneesta toimenpiteestä.



Kuvio 12. Projektin App.tsx-tiedosto

Seuraavassa kuvassa nähdään sovelluksen käyttöliittymän ulkoasu. Jokaisella sivulla on yläreunassa Header-komponentti ja sen alla sivun nimi. Ruudun alaosassa on navigointivalikko, johon toteutettiin tähän ensimmäiseen versioon kaksi painiketta: päänäkymä eli karttasivu ja käyttäjän omat tiedot. Karttanäkymän kautta tallennetaan uintikertoja ja lisätään yleisiä uintipaikkoja. Tämä tapahtuu Merkitse uintikerta -painikkeesta. Tähän versioon uintipaikkojen ja -kertojen näkyminen toteutettiin niin, että omat uintikerrat näkyvät sinisellä merkillä kartassa ja yleiset uintipaikat näkyvät vihreällä merkillä. Vihreää merkkiä painamalla käyttäjä näkee paikan lisätiedot ja muiden

käyttäjien kommentit. Karttanäkymän Asetukset-painikkeesta käyttäjä voi valita, näkyykö kartassa omat uintipaikat, yleiset uintipaikat vai kaikki uintipaikat. Oletuksena näkyvissä on kaikki uintipaikat.



Kuvio 13. Avantosilakan käyttöliittymän päänäkymät

5.3.1 Location-komponentti

Sovelluksen tärkein komponentti on Location, joka sijaitsee samannimisessä tiedostossa. Location-komponentti määrittelee karttanäkymän, paikannuksen ja sisältää painikkeet, joilla päästään vaihtamaan näkymän asetuksia ja tallentamaan uintipaikkoja ja -kertoja. Kuvaruudulta on mahdollista myös katsella muiden käyttäjien jättämiä kommentteja yleisistä uintipaikoista. Näitä karttanäkymän painikkeita ja muita toimintoja varten sovelluksessa on omat komponenttinsa, jotka Location-komponentti hakee käyttöönsä. Ne ovat siis Location-komponentin lapsikomponentteja.

Karttanäkymä toteutettiin sovellukseen React Native Maps -kirjaston avulla. Paikannus toteutettiin Expo Location -kirjaston avulla. Karttanäkymä ja paikannus olivat vaatimusmäärittelyssä sovelluksen toteutuksen kannalta tärkeimpien asioiden joukossa. Edellä mainittujen kirjastojen avulla toteutus oli todella helppo tehdä. Näistä kirjastoista löytyy valmiit metodit ja joustavia ominaisuuksia käyttöliittymän ja sen toimintojen muokkaamiseksi omaan käyttöön sopivaksi.

Käyttäjän tullessa ruudulle, metodi getCurrenPositionAsync hakee nykyisen sijainnin latitudeDeltan ja longitudeDeltan määrittämällä zoomauksella. Sovelluksen pysyessä auki watchPositionAsync päivittää sijaintia ja tallentaa sen hetkisen sijainnin location-muuttujaan. Location-muuttujan perusteella koko ajan on tiedossa sen hetkiset koordinaatit, jotka tallennetaan, kun käyttäjä tallentaa uintipaikan. Jotta käyttäjä pystyy käyttämään näitä paikannukseen liittyviä ominaisuuksia, hänen tulee antaa sovellukselle lupa käyttää sijaintia. Sovellus kysyy lupaa sijainnin käyttämiseen automaattisesti Expo Location -kirjaston valmiilla Location.requestForegroundPermissionsAsync-metodilla. Alla olevassa kuvassa näkyy kommentoituina nämä edellä mainitut metodit tarkemmin.

```
useEffect(() => {
  // Kysytään tarvittaessa lupa sijainnin käyttöön
  (async () => {
    const { status } = await Location.requestForegroundPermissionsAsync();
    if (status !== 'granted') {
          Toast.show({
            type: 'error',
            text1: 'Lupaa sijaintitietojen käyttöön ei myönnetty',
          });
      return;
    }
  // Haetaan käyttäjän nykyiset sijaintitiedot
    const currentLocation = await Location.getCurrentPositionAsync({});
    setLocation(currentLocation);
    setMapRegion({
      latitude: currentLocation.coords.latitude,
      longitude: currentLocation.coords.longitude,
      latitudeDelta: 0.005,
      longitudeDelta: 0.005,
    });
    // Seurataan käyttäjän sijaintia
    Location.watchPositionAsync(
      {
        accuracy: Location.Accuracy.High,
        timeInterval: 1000,
        distanceInterval: 1,
      },
      (location) => {
        setLocation(location);
      }
    );
  })();
}, []);
```

Kuvio 14. Location-komponentin paikannuksen koodi

Location-komponentti käyttää erillisiä komponentteja, joilla käyttäjä voi säätää ruudulla näytettäviä tietoja ja tallentaa uintikertoja. Käyttäjä voi tallentaa uintikerran ja halutessaan lisätä sen samalla yleiseksi uintipaikaksi. Jos samoilla tai lähes samoilla koordinaateilla on jo merkitty yleinen uintipaikka, ohjelma kertoo, ettei sitä voi merkitä enää toiseen kertaan mutta halutessaan käyttäjä voi lisätä sille kommentin. Käyttäjä merkitsee uuden uintikerran painamalla karttanäkymän Merkitse uinti -painiketta, joka on näkyvissä kuviossa 13. Merkitse uinti -painike on oma komponenttinsa SavePlaceButton, jolle välitetään propsina sen hetkiset koordinaatit ja muita tarvittavia tietoja. Komponentille välitetyt propsit näkyvät alla olevassa kuvassa. Kun painiketta painetaan,

kyseisessä komponentissa oleva boolean-tyyppinen muuttuja showModal muuttuu arvoon true. Tällöin ruudulle avautuu modaali, josta käyttäjä voi tallentaa uintikerran tiedot.

```
<SavePlaceButton
    placeData={placeData}
    existingPublicPlaces={swimmingPlaces}
    onSave={handleSavePlace}
/>
```

Kuvio 15. Location-komponentin lapsikomponentti SavePlaceButton

5.3.2 Käyttäjätiedot ja autentikointi

UserComponents-kansiossa on kolme tiedostoa: Login.tsx, SignUp.tsx ja UserProfile.tsx. Sovellukseen pääsee sisälle vain rekisteröitynyt käyttäjä. Login-komponentti on sovelluksessa nimensä mukaisesti kirjautumista varten. SignUp-komponentti hoitaa rekisteröitymisen. UserProfile-komponentti näyttää käyttäjän omat tiedot ja sen kautta onnistuu myös uloskirjautuminen.

Alla olevassa kuvassa näkyy Login-komponentin sisäänkirjautumistoiminto, jossa tarkistetaan käyttäjän tunnus ja salasana. Jos tunnus ja salasana täsmäävät, asetetaan isAuthenticated-muuttuja arvoon true, ja ohjataan käyttäjä sovelluksen pääsivulle. IsAuthenticated-muuttujaa käytetään läpi ohjelman määrittämään, mille kuvaruuduille käyttäjällä on pääsy.

```
const Login = (props: LoginProps) => {
  const { navigation } = props;
  const context = useContext(AuthGlobal);
  const [userName, setUserName] = useState('');
  const [password, setPassword] = useState('');
  const [error, setError] = useState('');

  const handleSubmit = async () => {
    const user = {
      userName,
      password,
    };
    if (userName === '' || password === '') {
      setError('Tarkista käyttäjätunnus ja salasana');
    } else {
      const success = await loginUser(user, context.dispatch);
      if (success) {
        await AsyncStorage.setItem('isAuthenticated', JSON.stringify(true));
        navigation.dispatch(
          CommonActions.reset({
            index: 0,
            routes: [{ name: 'Main' }],
          })
        );
      } else {
        setError('Kirjautuminen epäonnistui');
      }
    }
  };
```

Kuvio 16. Login-komponentin koodi

Autentikointiin liittyy neljä tiedostoa: Auth.tsx, AuthReducer.ts, Auth.actions.ts ja AuthGlobal.ts. Koko sovellus on Auth-komponentin sisällä, kuten kuviossa 12 oli nähtävissä. AuthReducer-tiedostossa ylläpidetään esimerkiksi isAuthenticate-muuttujan tilaa. Metodit, joilla näiden yllä mainittujen tietojen muutokset käsitellään, ovat Auth.actions-tiedostossa. AuthGlobal puolestaan huolehtii siitä, että ohjelmiston kaikilla komponenteilla on pääsy esimerkiksi isAuthenticate-tietoon.

Auth-komponentissa haetaan ja asetetaan käyttäjän tiedot. Auth:lla on siis merkittävä osa esimerkiksi siinä, että sovellus muistaa käyttäjän tiedot, eikä jokaisella kerralla tarvitse kirjautua sisään, jos käyttäjä ei ole edellisellä käyttökerralla kirjautunut ulos. Seuraavalla sivulla on kuva Auth-komponentin kohdasta, jossa käyttäjän tiedot asetetaan.

```
const Auth = (props: any) => {
    isAuthenticate: initialState.isAuthenticate,
    user: {},
    userProfile: null,
  });

  const [showChild, setShowChild] = useState(false);

  useEffect(() => {
    const loadUser = async () => {
      setShowChild(true);
      try {
        const token = await AsyncStorage.getItem('jwt');
        if (token) {
          const decoded: DecodedToken = jwtDecode(token) as DecodedToken;      Maria Haapamäki, 4 weeks ago • pa
          const userString = await AsyncStorage.getItem('user');
          const user = userString ? JSON.parse(userString) : null;
          const userWithId = user ? { ...user, userId: decoded.userId } : null;

          if (userWithId) {
            dispatch(setCurrentUser(decoded, userWithId));
          } else {
            dispatch(setCurrentUser(decoded, { userName: decoded.userName, password: '', userId: decoded.userId,
          }
        }
      } catch (error) {
        console.error("Error loading user:", error);
      }
    };
    loadUser();
    return () => setShowChild(false);
  }, []);
```

Kuvio 17. Auth-komponentin koodi

5.3.3 Navigointi ja reititys

Frontend:n navigators-hakemistosta löytyy kolme navigaatiotiedostoa: AppNavigator.tsx, Main.tsx ja UserNavigator.tsx. Koko sovellus on App.tsx-tiedostossa AppNavigator-komponentin sisällä. AppNavigator-komponentissa määritellään aloitussivu eli initialRouteName. Tämä aloitussivu on joko pääsivu eli Location-komponentti tai sisäänkirjautumissivu eli Login-komponentti. Tämä riippuu siitä, onko käyttäjä kirjautunut ohjelmaan vai ei. Alla on kuva tästä AppNavigator-komponentista, jossa ensimmäisenä haetaan tieto siitä, onko käyttäjä kirjautunut. Tämä tieto tallennetaan isAuthenticated-muuttujaan, jonka tilaa hallinnoidaan useState-hookin avulla.

```
const Stack = createStackNavigator();

const AppNavigator = () => {
  const [isLoading, setIsLoading] = useState(true);
  const [isAuthenticated, setIsAuthenticated] = useState(false);

  useEffect(() => {
    const checkAuthState = async () => {
      try {
        const authState = await AsyncStorage.getItem('isAuthenticated');
        if (authState !== null) {
          setIsAuthenticated(JSON.parse(authState));
        }
      } catch (error) {
        console.error('Failed to load auth state:', error);
      } finally {
        setIsLoading(false);
      }
    };

    checkAuthState();
  }, []);

  if (isLoading) {
    return null;      Maria Haapamäki, 2 weeks ago • added code
  }

  return (
    <Stack.Navigator initialRouteName={isAuthenticated ? "Main" : "Login"}>
      <Stack.Screen name="Login" component={Login} options={{ headerShown: false }} />
      <Stack.Screen name="Main" component={Main} options={{ headerShown: false }} />
    </Stack.Navigator>
  );
};
```

Kuvio 18. AppNavigator-komponentti

Main-komponentissa määritellään sovelluksen alapalkissa näkyvät kuvakkeet ja niiden reitit. Kuvakkeita on tällä hetkellä vain kaksi mutta seuraavassa versiossa palkkiin on tulossa yksi kuvake lisää. Sinisellä värillä näkyy se kuvake, jonka näkymässä käyttäjä parhaillaan on.

Kuvio 19. Main-komponentin näkyminen käyttöliittymässä

UserNavigator-komponentti tarvitaan käyttäjä- ja kirjautumistietoja varten. Käyttäjä pääsee kirjautumisnäkymästä rekisteröitymisnäkymään, jos hänellä ei vielä ole käyttäjätunnusta. Jos käyttäjä on

kirjautunut, hän pääsee katsomaan omia tietojaan painamalla kuvion 19 mukaista oikeanpuoleista kuvaketta.

5.3.4 Service-tiedostot

Frontend-tiedostot haluttiin pitää projektissa mahdollisimman selkeinä. Tiedostoista tulee helposti pitkiä ja vaikealukuisia. Tämän vuoksi kaikki backendiin tehtävät axios-kutsut koottiin Sevices-kansiossa oleviin tiedostoihin. Services-kansiossa on kolme tiedostoa. Api.js-tiedostossa on kerrottu baseUrl, joka voidaan hakea muiden tiedostojen axios-kutsuihin käyttämällä getBaseUrl-metodia. Lisäksi Services-kansiossa on swimmingplace.ts- ja user.ts-tiedostot, joissa on uintipaikkoihin ja käyttäjiin liittyvät axios-kutsut. Näitä axios-kutsuja voidaan käyttää mistä tahansa ohjelmasta.

Alla oleva kuva on swimmingplace.ts-tiedostosta. Kuvassa olevalla metodilla haetaan kaikki yhden käyttäjän omat uintikerrat. Tätä metodia käytetään esimerkiksi Location-komponentissa, jotta kartalla saadaan näytettyä käyttäjän uintipaikat. Metodissa axios-kutsu muodostaan lisäämällä await axios.get -komennon parametriksi baseUrl, userPlaces-endpoint ja käyttäjän userId. Jos kaikki menee, kuten pitää, response-muuttujaan palautuu backendiltä ne tiedot, joita frontend tarvitsee.

```
export const fetchUserSwimmingPlaces = async (userId) => {
  try {
    const response = await axios.get(`${getBaseUrl()}userPlaces/${userId}`);
    return response.data;
  } catch (error) {
    console.error('Error fetching user swimming places:', error);
    throw error;
  }
};
```

Kuvio 20. Metodi Services-hakemiston tiedostossa

6 Tulokset ja pohdinta

6.1 Valmis sovellus

Tämän projektin tuloksena syntyi toimiva mobiilisovellus. Sovellus on ulkoasultaan siisti ja käytettävyydeltään helppo. Ohjelmaan toteutetut ominaisuudet toimivat ilman virheitä. Ohjelmiston toteutuksessa on otettu myös tietoturva huomioon. Sovelluksessa on lisäksi toimiva autentikointi. Ohjelmiston tiedostot ovat järjestyksessä ja lajiteltuina kansioihin. Ohjelmointityössä on pyritty ottamaan huomioon, ettei teknistä velkaa pääsisi syntymään heti alkumetreillä. Ohjelman koodi on pidetty mahdollisimman yksinkertaisena virheiden välttämiseksi.

Paikannus ja karttaominaisuudet olivat sovelluskehityksen helpoin osuus. Näihin löytyneet valmiit kirjastot olivat ominaisuuksiltaan toimivia ja kattavia. Yllättäen haasteellisin osuus sovelluksen toteutuksessa oli navigointi ja reititys. Näiden tulisi olla sellaisia, että myöhemmin sovelluksen navigointiin ja reititykseen olisi helppoa lisätä uusia osia. Nykyinen toteutus sovelluksessa on hieman turhan monimutkainen komponenttien määrään nähden. Toisaalta navigoinnin pohja on kuitenkin monimutkaisuudestaan huolimatta niin toimiva, että siihen on helpohko lisätä uusia osioita, kunhan niiden logiikka esimerkiksi kirjautumisvaatimusten suhteen ei poikkea kovin paljon nykyisistä komponenteista.

Suurin osa projektin vaatimusmäärittelyn toiminnallisista vaatimuksista saatiin toimiviksi. Avantosilakkaan pystytään tällä hetkellä kirjaamaan uintikertoja ja yleisiä uintipaikkoja. Käyttäjä näkee kartasta molemmat ja pystyy halutessaan valitsemaan näkyviin vain omat uintipaikkansa, vain yleiset uintipaikat tai molemmat. Yleisille uintipaikoille voidaan lisätä kommentteja ja katsella muiden käyttäjien kommentteja. Toiminnallisista vaatimuksista seuraavan version toteutukseen jäi raporttisivu käyttäjän uintikerroista sekä omien tietojen muokkaaminen. Alla on kuva vaatimustaulukosta, jossa vihreällä näkyy sovelluksessa nyt toimivat ominaisuudet ja oranssilla ne ominaisuudet, jotka odottavat seuraavaa versiota.

Taulukko 4. Toteutuneet toiminnalliset vaatimukset

Vaatimus	Lisätiedot
Mobiilisovellus	Tiedot uintipaikasta tarvitaan nopeasti ja kätevästi.
Paikannus	Käyttäjän täytyy pystyä näkemään lähellä olevat uintipaikat helposti.
Mahdollisuus tallentaa uintipaikka julkisena tai yksityisenä	Esimerkiksi omassa rannassa tehtyjä uinteja ei haluta julkisesti näkyville mutta uintikertojen seuraamiseksi tieto pitää kuitenkin pystyä tallentamaan.
Yleisten uintipaikkojen kommentit	Käyttäjät pystyvät tallentamaan kommentteja julkisille uintipaikoille. Muut käyttäjät pystyvät katsomaan näitä kommentteja.
Omien tietojen tallennus ja muokkaus	Käyttäjä pystyy esimerkiksi vaihtamaan oman salasanansa tai sähköpostiosoitteensa.
Omien tietojen näkyvyys	Käyttäjän täytyy pystyä seuraamaan omia uintikertojaan ja -paikkojaan.

Ei-toiminnallisista vaatimuksista toteutettiin kaikki sovelluksen käytön aloittamisen kannalta kriittiset kohdat. Autentikointi on toimiva, eikä käyttäjä pääse näkemään muuta kuin omat tietonsa ja yleiset uintipaikat. Sovellus on tehty mahdollisimman yksinkertaiseksi ja helppokäyttöiseksi. Helppokäyttöisyyttä on vaikea arvioida objektiivisesti, mutta sitä testattiin projektin aikana kysymällä eri ihmisiltä ja pyytämällä heitä testaamaan ohjelmiston käyttöä. Näistä testeistä saatiin arvokasta palautetta. Tietoturva on huomioitu siten, että sovelluksen input-kentissä on sanitointi ja tiedostojen sisältö on sellainen, että salaisimmat tiedot on sijoitettu erilleen muusta koodista. Sovellukseen on toteutettu myös tokenin luonti. Kun käyttäjä kirjautuu sisään, ohjelma luo hänelle salaisen tokenin. Tokenin tarkoitus on, että ohjelmaan ei pääse sisälle ilman sitä. Jos käyttäjältä puuttuu token, hän pääsee näkemään vain kirjautumis- ja rekisteröitymissivun mutta ei pääse käsiksi varsinaisiin tiedostoihin. Token ei kuitenkaan toimi ohjelmassa vielä tällä hetkellä täysin ja sen korjaaminen jää seuraavaan vaiheeseen. Tokenin tulee olla toiminnassa ennen ohjelmiston julkaisua.

Taulukko 5. Toteutuneet ei-toiminnalliset vaatimukset

Vaatimus	Lisätiedot
Tietoturva	Tietokannan ja käyttäjien tietojen suojaus täytyy olla kunnossa.
Jatkokehitysmahdollisuudet	Ohjelmiston suunnittelussa ja toteutuksessa on huomioitava skaalautuvuus.
Helppokäyttöisyys ja yksinkertaisuus	Käyttäjien tulee ymmärtää heti mistä on kyse ja osata käyttää sovellusta ilman ohjevihkosia.
Autentikointi	Käyttäjän tulee pystyä kirjautumaan omalla tunnuksellaan. Myös uloskirjautumisen tulee onnistua.
Autorisointi	Tavallinen käyttäjä saa nähdä vain omat tietonsa ja yleiset uintipaikat. Admin-käyttäjällä on oikeudet hallinnoida sovellusta.
Sanitointi	Input-kentissä tulee olla sanitointi viimeistään siinä vaiheessa, kun sovellus laitetaan palvelimelle
Admin-valikko	Tarvitaan Admin-taso käyttäjätunnuksiin. Admin käyttäjä voi esimerkiksi moderoida sovellusta.
Automatisoidut moderointitoimenpiteet	Sovelluksessa tulee olla ainakin ’huonojen sanojen’ filtteröinti

6.2 Tutkimuskysymysten vastaukset ja pohdinta

Tällä projektilla haettiin vastauksia seuraaviin tutkimuskysymyksiin:

-Mitkä teknologiat sopivat parhaiten tämän mobiilisovelluksen toteuttamiseen?
-Miten sovelluksen suunnittelussa huomioidaan käytettävyys?
-Millaisia jatkokehitysmahdollisuuksia on ja miten ne huomioidaan suunnittelu- ja kehitysvaiheessa?
-Voidaanko sovelluksen avulla lisätä avantouintipaikkojen saavutettavuutta?

Suunnitteluvaiheessa käytettiin aikaa työkalujen vertailuun ja löydettiin tähän projektiin sopivin kokonaisuus. MERN-pino sopii hyvin tämän tyyppisten sovellusten kehittämiseen. MERN-pinon työkalut ovat niin yleisesti käytössä, että tietoa kaikenlaisiin ongelmatilanteisiin on saatavilla todella hyvin. Ladattavia lisätyökaluja ja kirjastoja on myös saatavilla todella laajasti. MERN-pinon avulla on helppoa tehdä nykyaikaista web-ohjelmointia. React Native antaa mahdollisuuden tehdä alustariippumatonta mobiilisovellusta. React Native on tähän tarkoitukseen niin ylivoimainen, että muita vaihtoehtoja on vaikea edes harkita. Vaikka kehitystiimi olisi aiemmin tehnyt web-kehitystä esimerkiksi Vuella, mobiilikehityksessä React Native on laajojen kirjastojen, ylläpidon ja saatavilla

olevan tiedon puolesta ylivoimainen. Esimerkiksi Vue Nativen ylläpito on lopetettu mutta on vaikea kuvitella, että React Nativen kohdalla kävisi niin. React Native on myös helppo oppia, jos on jo aiempaa kokemusta web-kehityksestä jollain JavaScript-kirjastolla. Myös muut MERN-työkalut olivat sopivia tähän projektiin. MongoDb -tietokanta on joustava, nopea ja helppokäyttöinen. Node.js-backend on helppo tehdä, kun JavaScript on tuttu. Tämä projekti osoitti MERN:n toimivuuden ja opetti lisää tämän teknologiakokonaisuuden mahdollisuuksista. Seuraava projekti samoilla työkaluilla on jo suunnitteilla.

Johdanto-luvussa pohdittiin avantouintipaikkojen saavutettavuuden ongelmaa, jota Avantosilakka-sovelluksella halutaan ratkaista. Kunhan riittävä määrä käyttäjiä löytää sovelluksen, talviuintipaikkojen saavutettavuus voi sen avulla parantua. Sovelluksen toteutus on suunnitelman mukainen mutta koska sovellus on eräänlainen sosiaalinen media, sen toimivuus on lopulta kiinni siitä, löytyykö sille tarpeeksi käyttäjiä. Sovelluksen käytettävyyttä testattiin koko ensimmäisen kehitysvaiheen ajan eri laitteilla heti, kun muutoksia ja ominaisuuksia oli tehty lisää. Kehitysvaiheen edetessä mielipiteitä kysyttiin sovelluksen tulevilta ja potentiaalisilta käyttäjiltä. Ongelmana tämän tyyppisessä sovelluksessa on tiedon nopea vanheneminen ja sovelluksen ylläpitäminen. Jatkoa ajatellen pohdittavaksi jää, miten huolehditaan tiedon ajantasaisuudesta. Kysymystä uintipaikkojen saavutettavuuden parantamisesta ei siis vielä ole päästy käytännössä testaamaan, joten tähän kysymykseen pystytään vastaamaan vain sen osalta, että tekninen ratkaisu on toimiva ja se mahdollistaa saavutettavuuden paranemisen.

Jatkokehitysmahdollisuuksia ja sovelluksen käytettävyyttä mietittiin heti alusta asti jo suunnitteluvaiheessa. Käytettävyyttä pohdittiin tulevien käyttäjien kanssa piirtäen paperille malleja sovelluksen toiminnoista. Käytettävyyttä testattiin projektin edetessä testaamalla ja keskustelemalla eri vaihtoehdoista. Myös jatkokehitysmahdollisuuksien jatkuva pohtiminen mahdollisti sen, ettei työtä tehdessä tullut tehtyä liian ’kiveen hakattuja’ ratkaisuja. Projektin edetessä esiin nousi jatkokehitysideoita sekä vertailua tulevista palvelinratkaisuista.

Jatkokehityksen ensimmäinen vaihe on korjata tokenin toiminta sekä luoda raporttisivu käyttäjän uintikerroista. Tätä raportointiominaisuutta olisi mahdollista laajentaa siten, että käyttäjä voisi esimerkiksi asettaa omia tavoitteita uintikertojen määrälle viikossa tai kuukaudessa. Sovelluksen moderointia helpottamaan tarvitaan kirosanafiltteri, jota ei vielä tähän versioon saatu toteutettua.

Myös uintikertojen ja -paikkojen tallennukseen tullaan tekemään kehitystä. Jatkossa käyttäjän tallentaessa uintikertaansa, sovellus tallentaa uintikerran tiedot kantaan mutta ei tee samasta uintipaikasta kuin yhden karttamerkinnän. Karttamerkintää painamalla, käyttäjä voi nopeasti nähdä kuinka monta kertaa hän on kyseisessä paikassa uinut. Karttamerkinnät tullaan myös tekemään jatkossa muilla kuin React Nativen omilla Marker-komponenteilla. Karttamerkintöjä varten luodaan itse tehdyt kuvat, joita kartassa käytetään näyttämään merkintöjä.

Sovellus aiotaan ottaa käyttöön pienin askelin. Aluksi sovellus tulee olemaan käytössä muutaman ihmisen pilotointikokeilussa ja myöhemmin se voidaan ottaa laajempaan käyttöön. Sovellusta varten on etsinnässä sopiva ja ainakin aluksi ilmainen pilvipalvelinratkaisu. Vertailussa on ainakin Netlify- ja Vercel -nimiset palveluntarjoajat. Heroku-nimisellä palveluntarjoajalla ei enää ollut saatavilla ilmaista palvelintilaa mutta eco- tai student-paketti on edullinen vaihtoehto. Palvelinratkaisu on kuitenkin vielä vertailun alla. Sovelluksen lähdekoodi löytyy toistaiseksi julkisesta GitHub-repositoriosta osoitteesta https://github.com/mariahaapamaki/Talviuinti.

Lähteet

Bakker, A. Sermon, L. 2023. MERN Stack E-Commerce Mobile App with React Native. Udemy. Online-kurssi.

Biswas, N. 2023. Ultimate Full-Stack Web Development With Mern. Viitattu 15.1.2025. https://www.google.fi/books/edition/Ultimate_Full_Stack_Web_Development_with/c4PnEAAAQBAJ?hl=fi&gbpv=1&dq=ultimate+mern&printsec=frontcover

Expo vs React Native CLI: Key Differences Explained. 2024. Flatirons Development. Viitattu 3.3.2025. https://flatirons.com/blog/expo-vs-react-native/

Luukkainen, M. 2022. Ohjelmistojen vaatimusmäärittely, tuotteen ja sprintin hallinta. Ohjelmistotuotanto avoin yliopisto 2022. Viitattu 7.4.2025. https://ohjelmistotuotanto-hy-avoin.github.io/osa2/

McKee, A. 2023. Coding Best Practices and Guidelines for Better Code. Viitattu 28.3.2025. Coding Best Practices and Guidelines for Better Code | DataCamp

Mern Stack Alternatives: Top Choices of 2025. 2025. Viitattu 4.4.2025. https://www.simplilearn.com/mern-stack-alternatives-article

Mern Stack Explained. N.d. MongoDb:n oma opas. Viitattu 2.2.2025. https://www.mongodb.com/resources/languages/mern-stack

Oraskari, J. 2022. Typescript nousi koodareiden suosikkien joukkoon – auttaa välttämään virheitä. Tivi-lehden artikkeli. Viitattu 3.1.2025. https://www.tivi.fi/uutiset/typescript-nousi-koodareiden-suosikkien-joukkoon-auttaa-valttamaan-virheita/75f290eb-022d-461a-b49d-8bb1ad803193

Paakki, J. 2011. Ohjelmistojen vaatimusmäärittely. Viitattu 20.1.2025. https://www.cs.helsinki.fi/u/paakki/Vaatimus-11-Luentokalvot-1.pdf

Rafalski, K. 2025. Essential Web Development Best Practices for 2025. Viitattu 27.3.2025. https://www.netguru.com/blog/web-development-best-practices

Rantala, A. 2023. Tutkimusasetelman ja -kysymysten esittäminen. Viitattu 4.4.2025. JAMK - Opinnäytetyö

Soveltava tutkimus. 2024. Lapin amk. Viitattu 3.1.2025. https://lapinamk.fi/tutkimus-ja-kehitys/soveltava-tutkimus/

Talviuintihanke, opas talviuintipaikan kehittämiseen ja uuden talviuintipaikan perustamiseen. 2024. Suomen Latu. Viitattu 21.3.2025. https://www.suomenlatu.fi/media/talviuintihanke/opas-talviuintipaikan-kehittamiseen-ja-uuden-talviuintipaikan-perustamiseen.pdf

Using Middleware. N.d. Expressjs:n oma opas. Viitattu 28.2.2025. https://expressjs.com/en/guide/using-middleware.html

Routing. N.d. Expressjs:n oma opas. Viitattu 28.2.2025. https://expressjs.com/en/guide/routing.html

Vaatimusmäärittelyopas. 2021. Alfame. Viitattu 15.2.2025. https://www.alfame.com/hubfs/files/Vaatimusma%CC%88a%CC%88rittely%20kettera%CC%88ssa%CC%88%20ohjelmistokehityksessa%CC%88%20-opas.pdf

Versionhallinta. N.d. Helsingin Yliopisto. Tietokone työvälineenä -kurssi. Viitattu 4.4.2025. https://courses.mooc.fi/org/uh-cs/courses/tietokone-tyovalineena/chapter-2