Niko Ala-aho

Ohjelmistoratkaisu Pyöräilijän tiepalvelulle

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

1.11.2024

Tiivistelmä

Tekijä:	Niko Ala-aho
Otsikko:	Ohjelmistoratkaisu Pyöräilijän tiepalvelulle
Sivumäärä:	31 sivua
Aika:	1.11.2024
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikka
Ammatillinen pääaine:	Ohjelmistotuotanto
Ohjaaja:	Lehtori Vesa Ollikainen

Insinöörityön tavoitteena oli suunnitella ja toteuttaa Pyöräilijän tiepalvelu -sovellus mobiiliympäristöön. Työssä esitellään sovelluksen toteutuksen vaiheet suunnittelusta ja rakentamisesta sekä testauksesta beta-version julkaisemiseen asti. Palveluun liittyneet pyöräilijät pystyvät lähettämään avunpyyntöjä muille käyttäjille maastossa. Sovelluksen käyttö on pääosin passiivista, joten tieto avuntarpeesta täytyy saada käyttäjille ilman sovelluksen aktiivista käyttöä.

Työn toisena tavoitteena oli selvittää, miten progressiivinen verkkosovellus eli PWA soveltuu tämänkaltaisen sovelluksen kehittämiseen. Työssä toteutettiin PWA:ta hyödyntävä verkkosovellus, jolloin sovelluksen kehitystyö pystyttiin toteuttamaan samanaikaisesti kaikille alustoille ilman erillisiä natiiviprojekteja. Sovelluksen keskeisin ominaisuus on push-ilmoitukset, joiden avulla tieto avuntarpeesta saadaan käyttäjille nopeasti ja sovellus palvelee käyttäjiä halutulla tavalla.

Sovellus kehitettiin yhden kehittäjän projektina Scrum-menetelmän sprinttejä mukauttaen. Sovellukselle toteutettiin käyttöliittymä (frontend), palvelin (backend) sekä tietokanta, johon käyttäjät, heidän lähettämänsä pyynnöt sekä push-ilmoitusten tilaukset tallennetaan. Frontend sekä backend kirjoitettiin TypeScriptillä ja sovelluksen käyttöliittymä toteutettiin Reactilla.

Insinöörityön tuloksena saatiin toimiva beta-versio Pyöräilijän tiepalvelu -sovelluksesta. Johtopäätöksenä todettiin, että PWA soveltui projektin toteuttamiseen hyvin. Sovelluksen kehitys jatkuu insinöörityön valmistumisen jälkeen. Testausvaiheen jälkeen tavoitteena on julkaista sovelluksen versio 1.0.

Avainsanat: PWA, React, TypeScript, Azure, Full Stack

Tämän opinnäytetyön alkuperä on tarkastettu Turnitin Originality Check -ohjelmalla.

Abstract

Author:	Niko Ala-aho
Title:	Software Solution for Cyclists' Road Service
Number of Pages:	31 Pages
Date:	1 November 2024
Degree:	Bachelor of Engineering
Degree Programme:	Information and Communication Technology
Professional Major:	Software Engineering
Supervisor:	Vesa Ollikainen, Senior Lecturer

The goal of the study was to design and implement a Road Service for cyclists to a mobile environment. The thesis presents the development stages of the application, from planning and building to testing and the release of the beta version. Cyclists who have joined the service are able to send requests for assistance to other users on the move. Since the application is primarily in passive use, it is essential that the help requests reach users without requiring active engagement with the app.

The second goal of the project was to study how Progressive Web App (PWA) suits the development of this type of an application. A web application utilizing PWA was implemented, allowing the app to be developed simultaneously for all platforms without the need for separate native projects. The key feature of the app is push notifications, which ensure that the help requests reach users quickly and that the application functions as intended.

The application was developed as a solo project, adapting the Scrum method with sprints. The project involved the creation of a user interface (frontend), a server (backend), and a database, which stores user data, their requests, and push notification subscriptions. Both the frontend and backend were written in TypeScript, and the user interface was built with React.

The result is a functional beta version of the Road Service application. The conclusion was that PWA was well-suited for this project. The development of the application will continue with the goal of releasing version 1.0 following the testing phase.

Keywords: PWA, React, TypeScript, Azure, Full Stack

Sisällys

Lyhenteet ja termit

1 Johdanto 1

2 Työn lähtökohdat 2

3 Teknologiat ja kehitystyökalut 3

3.1 Ohjelmointikielet ja -ympäristöt 3
3.2 Käyttöliittymä (frontend) 4
3.3 Palvelinpuoli (backend) 5
3.4 Versionhallinta ja jatkuva integraatio 7
3.5 Tekoälyn hyödyntäminen 8

4 Sovelluksen arkkitehtuuri 8

4.1 Sovelluksen rakenne ja komponentit 8
4.2 Palvelinympäristö ja tietoturva 11

5 PWA:n hyödyntäminen 12

5.1 Progressiivinen verkkosovellus 12
5.2 Push-ilmoitusten toteuttaminen 14

6 Tiepalvelusovelluksen rakentaminen 19

6.1 Alustava käyttöliittymän toteutus 20
6.2 Tietokannan alustus ja käyttäjän autentikaatio 20
6.3 Dokumentaation alustus ja testien kirjoittaminen 20
6.4 Pyyntöjen toteutus ja ensimmäinen prototyyppi 21
6.5 Tietoturva, rajoitukset sekä käyttäjäkokemus 21
6.6 Beta-version käyttöliittymän viimeistely 22
6.7 Sovelluksen käyttöönotto ja loput toiminnallisuudet 25
6.8 Sovelluksen toiminnallisuudet ja testaaminen 25
6.9 Jatkokehitys 26

7 Yhteenveto 27

Lähteet 29

Lyhenteet ja termit

API:	*Application Programming Interface*. Ohjelmointirajapinta, joka määrittelee, miten eri ohjelmistojen komponentit tai palvelut voivat kommunikoida keskenään.
App Service:	Azuren palvelu verkkosivujen isännöintiin pilvessä.
Azure:	Microsoftin julkinen pilvipalvelu.
Backend:	Sovelluksen palvelinpuoli.
CI/CD:	*Continuous Integration and Continuous Delivery*. Jatkuva integraatio ja jatkuva toimitus.
Firebase:	Googlen pilvipalveluna tarjoama sovellusalusta.
Frontend:	Sovelluksen selain- eli käyttöliittymäpuoli.
GitHub Actions:	GitHubin tarjoama CI/CD-alusta.
HTTP:	*Hypertext Transfer Protocol*. Protokolla, jolla selaimet ja www-palvelimet siirtävät tietoa.
JWT:	*JSON Web Token.* JSON-pohjainen avoimen standardin menetelmä käyttöoikeustietueiden hallinnoimiseen eri ohjelmistojen välillä.
PWA:	*Progressive Web Application*. Progressiivinen verkkosovellus eli mobiililaitteeseen asentuva web-sovellus, joka voi vastaanottaa push-ilmoituksia.
React:	JavaScript-kirjasto käyttöliittymien kehittämiseen.
REST API:	Käytetään HTTP- tai HTTPS-pyyntöjen lähettämiseen palvelimille.
SSE:	*Server-Sent Events.* Palvelimen push-tekniikka, jonka avulla asiakas voi vastaanottaa automaattisia päivityksiä palvelimelta HTTP-yhteyden kautta.
SQL:	*Structured Query Language*. Kyselykieli, jolla relaatiotietokantaan voi tehdä hakuja, muutoksia ja lisäyksiä.

1 Johdanto

Työn tavoitteena oli toteuttaa pyöräilijöille suunnattu tiepalvelusovellus. Tavoitteena oli, että sovellukseen liittyneet pyöräilijät pystyisivät lähettämään toisilleen esimerkiksi avunpyyntöjä ongelmatilanteissa maastossa, jolloin lähellä olevat pyöräilijät voisivat tulla auttamaan. Sovellukselle on maastopyöräilypiireissä kysyntää, ja tällaista palvelua on toivottu jo pitkään. Nyt toteutettavaan Pyöräilijän tiepalvelu -sovellukseen rekisteröityneet käyttäjät voivat hyödyntää maastopyöräily-yhteisön osaamista ja saada viipymättä apua ongelmatilanteissa. Kehitystyötä jatketaan edelleen insinöörityön valmistuttua, jotta sovellus saadaan tuotantokäyttöön.

Suurin osa sovelluksen käytöstä tapahtuu mobiiliympäristössä, joten sovelluksen täytyy olla saatavilla sekä Androidille että iOS:lle. Työn toteuttamiseksi valittiin web-pohjainen PWA (Progressive Web App) -sovellus. PWA:n avulla sovellus saadaan toteutettua kaikille alustoille soveltuvaksi. Insinöörityön aikana toteutettiin beta-versio Pyöräilijän tiepalvelu -verkkosovelluksesta.

Työn toisena tavoitteena oli tutkia PWA:n soveltumista tämän tyylisen mobiilisovelluksen toteuttamiseen ja käyttämiseen. Kehitetyssä sovelluksessa keskeisin PWA:n ominaisuus on push-ilmoitusten lähettäminen (push notifications).

Työssä tuli toteuttaa sovellusta varten käyttöliittymä (frontend), palvelin (backend) sekä tietokantaratkaisu, johon rekisteröityneet käyttäjät ja heidän lähettämänsä pyynnöt tallennetaan.

Säännöllisissä käyttäjäpalavereissa yhteistyökumppaneiden kanssa käydään läpi toteutetut sovelluksen ominaisuudet ja mahdolliset jatkokehitystarpeet. Testausvaiheessa hyödynnetään yhteistyökumppaneista koostuvaa testiryhmää, jotta sovelluksesta saadaan mahdollisimman ehyt ja käyttäjäystävällinen.

2 Työn lähtökohdat

Pyöräilijän tiepalvelusovellukselle oli havaittu tarve ja sille etsittiin maastopyöräilypiireissä kehittäjää. Alkukartoituksessa todettiin, ettei valmista mallia sovellukselle ollut olemassa. Sovelluksen esiasteena toimi työn yhteistyökumppaneiden ylläpitämä Pyöräilijän tiepalvelu -Facebook-ryhmä, joka ei kuitenkaan palvellut käyttäjiä halutulla tavalla.

Kuulin tämän kaltaisen sovelluksen kehittämistarpeesta ja selvitin, oliko kukaan vielä ryhtynyt toteuttamaan kyseistä projektia. Kiinnostuin aiheesta ja näin mahdollisuuden toteuttaa toivotun sovelluksen insinöörityönä.

Uuden sovelluksen kehittäminen aloitettiin yhteistyökumppaneiden kanssa pidetyllä aloituspalaverilla, jossa he esittivät toiveitaan sovelluksen sisältämistä ominaisuuksista. Yhteistyökumppaneilla oli selkeät näkemykset sovelluksen toivotuista ominaisuuksista, joten projektin aloittaminen oli suoraviivaista.

Tiepalvelun idea on, että ongelmia kohdatessaan pyöräilijä voi lähettää palvelun kautta avunpyynnön, josta ilmenee hänen sijaintinsa ja avun tarve sekä puhelinnumero, johon avun tarjoaja ottaa yhteyttä. Ongelman luokittelussa tulisi olla valintaruutuja ja lisäksi vapaa tekstikenttä. Mahdollisia ongelmia voivat olla esimerkiksi huoltoapu pyörän korjauksessa, jonkin varaosan pyytäminen tai vaikka vesipullon tarve. Muut palvelun käyttäjät saisivat määrittämänsä säteen sisällä tehdyn ilmoituksen pyynnön tekijän avun tarpeesta. Käyttäjän valitsema säde määrittää kilometreinä etäisyyden omasta sijainnistaan, jonka sisällä tehdyt pyynnöt tulevat ensisijaisesti käyttäjälle näkyviin. Käyttäjä voisi muuttaa valitsemaansa sädettä asetuksista milloin tahansa.

Pyynnön lähettämisen jälkeen pyynnön lähettäjä voisi saada tiedon, kuinka moni käyttäjä on nähnyt avunpyynnön. Se käyttäjä, joka ottaa tehtävän hoitaakseen, kuittaa pyynnön "työn alle", jottei muiden käyttäjien tarvitse enää reagoida kyseiseen pyyntöön. Kun apu on toimitettu perille, pyynnön tekijä muuttaa pyynnön tilan valmiiksi.

Jatkoin yhteistyökumppaneiden kanssa pidetyn aloituspalaverin jälkeen toteutuksen suunnittelua ja valitsin toteutustavaksi PWA-verkkosivuston, jonka pystyy asentamaan mobiililaitteelle kuten natiivisovelluksen. Tällä toteutustavalla pystytään toteuttamaan toimiva mobiilisovellus kaikille alustoille yhdellä projektilla. PWA tarjoaa ominaisuuksia, jotka ovat tärkeässä osassa kehitettävää sovellusta. Sovelluksessa käytettävistä PWA:n ominaisuuksista keskeisin on push-ilmoitusten lähettäminen käyttäjän laitteeseen. PWA:ta ja sen ominaisuuksia on esitelty tarkemmin luvussa 5.

3 Teknologiat ja kehitystyökalut

Sovelluksen kehitys toteutettiin useita erilaisia teknologioita ja kehitystyökaluja hyödyntäen. Seuraavissa alaluvuissa esitellään teknologiat, työkalut ja alustat, joita sovelluksen kehitystyössä on käytetty.

3.1 Ohjelmointikielet ja -ympäristöt

TypeScript

Microsoftin kehittämä TypeScript [1] lisää tyypityksen tavalliseen JavaScriptiin. TypeScriptin avulla virheet huomataan nopeammin jo koodin kirjoitusvaiheessa. Typescript-koodi muunnetaan JavaScript-koodiksi ajovaiheessa, joten se toimii kaikkialla, missä JavaScript toimii. Mikä tahansa JavaScript-koodi on kelvollinen TypeScript-koodi. JavaScript on dynaamisesti kirjoitettu, eli se on erittäin joustava ja helppokäyttöinen kieli, mutta suuremmissa projekteissa tämä saattaa hankaloittaa kehitysprosessia. TypeScript puolestaan on staattisesti kirjoitettu, joka tarkoittaa sitä, että sille tulee määritellä mitä parametreja funktiot ja objektit tarvitsevat, joten mahdolliset virheet huomataan jo kirjoitusvaiheessa. [1; 2.]

Valitsin projektiini TypeScriptin juuri tästä syystä, että mahdolliset virheet voidaan huomata jo varhaisessa vaiheessa ja koodin rakenne pysyy selkeämpänä. Minulla on eniten kokemusta staattisesti kirjoitetuista kielistä, kuten Javasta,

joten TypeScriptin valitseminen tuntui luonnolliselta. Frontend sekä backend on molemmat kirjoitettu TypeScriptillä projektissani.

Node.js ja npm

Node.js [3] on kaikilla alustoilla toimiva avoimen lähdekoodin ajoympäristö, jolla JavaScript-koodi voidaan suorittaa palvelimella [4]. Node.js on käytetyin JavaScript-kirjasto kehittäjien keskuudessa [5]. Node.js oli minulle selkeä valinta, sillä se on tullut tutuksi aiemmista koulu- sekä vapaa-ajanprojekteistani, joten sen käyttöönotto oli suoraviivaista eikä vaatinut opettelua.

Npm [6] on paketinhallintatyökalu Node.js:lle. Se on maailman suurin ohjelmistorekisteri, jossa on yli 800 000 koodipakettia [7]. Valitsin npm:n paketinhallintatyökaluksi, koska Node.js:än rinnalla sekin oli minulle ennestään tuttu aikaisemmista projekteista. Projektissani on useita npm-paketteja käytössä, joista yksi esimerkki on Express [8], joka on kaikista suosituin npm-paketti kehittäjien keskuudessa [9]. Expressistä kerrotaan tarkemmin alaluvussa 3.3.

3.2 Käyttöliittymä (frontend)

React

React [10] on JavaScript-kirjasto, jota käytetään käyttöliittymien kehittämiseen. Facebookin kehittämä React on tilastojen mukaan viime vuosina ollut suosituin käyttöliittymäkirjasto kehittäjien keskuudessa [11]. Päätin tehdä projektin Reactilla, koska itselläni on siitä eniten kokemusta opiskeluista sekä vapaa-ajan projekteista. Osatekijänä Reactin valitsemiseen oli se, että React tarjoaa myös valmiin pohjan PWA-sovelluksen alustamiseen, josta on tässä projektissa hyötyä.

Google Cloud – Maps API

Google Maps API [12] on yksi Google-teknologian osista, jonka avulla pystyy hyödyntämään Google Mapsin ominaisuuksia ja sijoittamaan sen suoraan

omalle nettisivustolleen. Google Maps API:a käyttää yli 150 000 eri sivustoa. [13.] Googlen pilvipalvelussa (Google Cloud) oleva Google Maps Platform tarjoaa useita karttoihin liittyviä palveluja, joista sovelluksessani on käytössä ainoastaan Maps API. Muita mahdollisia palveluita tämän oheen ovat muun muassa navigointipalvelu tai esimerkiksi Air Quality API, jonka avulla karttaan voi lisätä tiedon ilman laadusta määritetyllä alueella.

Valitsin Googlen kartan sovellukseeni, koska sen käyttöönotto vaikutti suoraviivaiselta ja siihen löytyivät hyvät ohjeet ja dokumentaatio. Sovelluksessani käyttäjän sekä pyynnön sijainnin näyttäminen kartalla on keskeisessä osassa sovelluksen toiminnallisuuksia ja tähän tarkoitukseen Maps API on riittävä.

3.3 Palvelinpuoli (backend)

Express.js

Express [9] on joustava Node.js-verkkosovelluskehys. Nodella tapahtuvaa websovellusten ohjelmointia helpottamaan on kehitetty useita ohjelmointirajapinnan tarjoavia kirjastoja. Näistä kaikista suosituin on Express [14]. Express.js on nopea ja joustava verkkokehys Node.js:lle. Sitä käytetään sovellusten REST API -rajapintojen toteuttamiseen. [15.] Sovellukseni frontendin ja backendin tiedon siirto tapahtuu REST API:a (tai RESTful API) käyttäen. Myös Express valikoitui projektiini siitä syystä, että se on ollut käytössä useassa projektissa aikaisemmin ja sen käyttöä ei tarvinnut opetella, mikä nopeutti projektin kehitystä.

Azure App Service

Azure App Service [16] tarjoaa puitteet verkkosovellusten isännöintiin Microsoftin Azure-pilvipalvelussa. Sovellukset toimivat ja skaalautuvat helposti sekä Windows- että Linux-pohjaisissa ympäristöissä. App Servicen avulla saa myös esimerkiksi parannellun tietoturvan sovellukselle Azuren kautta. Lisäksi App Serviceen on helppo yhdistää GitHubin jatkuvan integraation alusta GitHub Actions. [17.]

Azure App Service tarjoaa monia eri tasovaihtoehtoja vaihtelevilla suorituskyvyillä sovelluksen ylläpitoon. Ilmainen ”F1”-taso toimi hyvin projektin kehitysvaiheessa ja tasoa on helppo päivittää ylöspäin, kun sovellus siirretään tuotantokäyttöön ja suorituskykyä tarvitsee nostaa. Päätin sijoittaa sovellukseni Azureen, koska minulla on siitä aikaisempaa kokemusta ja sen sijoittaminen samalle alustalle tietokannan kanssa helpottaa näiden keskustelua keskenään.

Azure SQL Database

Tietokannaksi valitsin SQL-pohjaisen tietokannan, johon rekisteröityneet käyttäjät, heidän lähettämänsä pyynnöt sekä käyttäjien push-ilmoitusten tilaukset (subscriptions) tallennetaan. Azure SQL Database [18] on aina käynnissä SQL Server -tietokantamoottorin uusimmalla vakaalla versiolla [19]. Tietokanta sijoitettiin App Servicen tapaan Azuren pilvipalveluun, jotta sitä on helppo käyttää Azuren sovelluksen isännöinnin rinnalla. Myös tietokannan hintatasoa ja suorituskykyä sekä tallennustilaa on helppo nostaa myöhemmin kehitysympäristöstä tuotantoympäristöön.

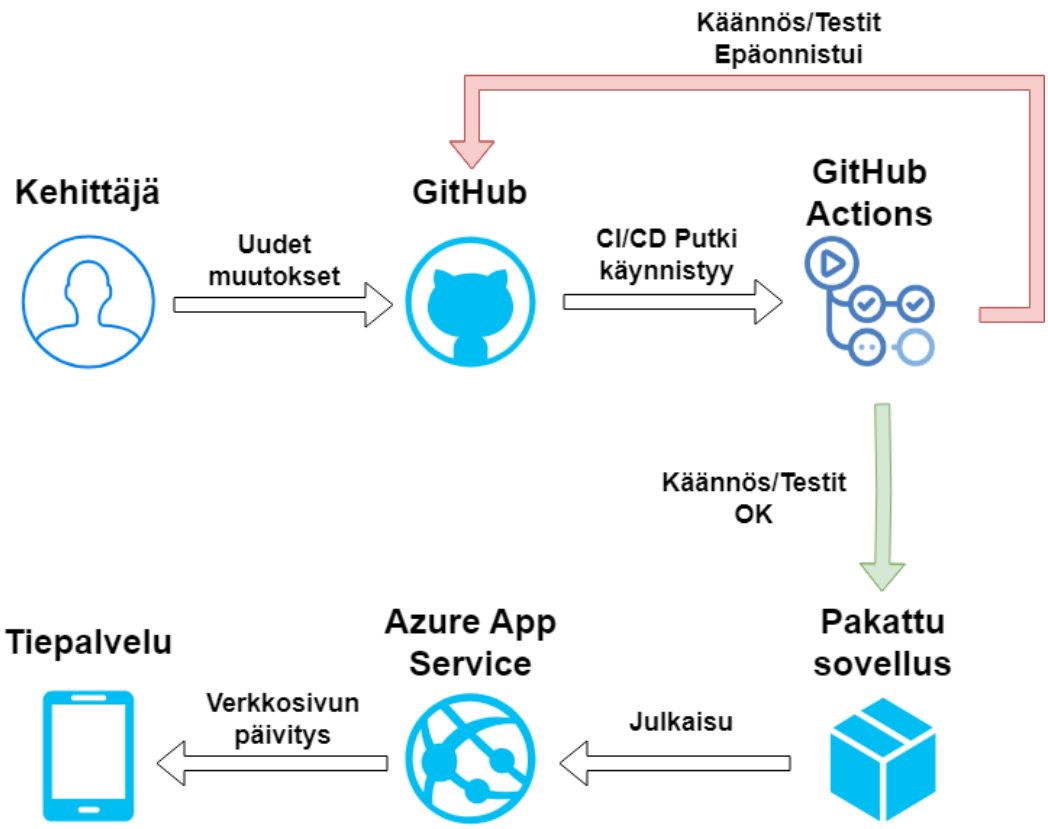
Firebase Authentication

Puhelinnumeron varmistaminen tapahtuu Googlen pilvipalveluna tarjoaman sovellusalusta Firebasen [20] kautta projektissani. Firebase lähettää kertakäyttöisen koodin tekstiviestinä käyttäjälle ja hoitaa itse myös koodin oikeellisuuden tarkastamisen. Projektissani itse tietokanta ja käyttäjien tiedot on sijoitettu Azureen, kuten aiemmin mainittu, joten käytän Firebasea ainoastaan tekstiviestin lähettämiseen ja koodin oikeellisuuden tarkastamiseen. Jo varhaisessa vaiheessa päätettiin yhteistyökumppaneiden kanssa, että sovellukseen ei tarvitse liittää sähköpostia käyttäjätietoihin lainkaan, joten käyttäjien tunnistautuminen tapahtuu puhelinnumeron avulla. Sovelluksessa myös avunpyynnön lähettäjän ja vastaajan välinen yhteydenpito tapahtuu puhelimitse, joten on tärkeää, että puhelinnumerot ovat vahvistettuja kirjoitusvirheiden sekä väärinkäytösten ehkäisemiseksi.

3.4 Versionhallinta ja jatkuva integraatio

Versionhallintana käytän projektissani GitHubia. Se on yksi suosituimmista versionhallinta-alustoista maailmassa, ja sitä käyttää arviolta noin 100 miljoonaa kehittäjää [21].

Jatkuvaan integraatioon käytössäni on GitHubin tarjoama GitHub Actions. Niin kutsuttu CI/CD eli jatkuva integraatio (continuous integration) ja jatkuva toimitus (continuos delivery) on yleinen ja suositeltava tapa huolehtia, ettei sovelluksen tuotantoon päädy viallisia versioita. CI/CD automatisoi sovelluksen koodin kääntämisen tai rakentamisen, testaamisen sekä julkaisun. [22.] Kun kehittäjä, eli tässä tapauksessa minä, pusken uudet koodin muutokset GitHubiin, käynnistää se automaattisesti CI/CD-putken, jossa olen määritellyt kaikki vaiheet projektin rakentamista varten. Kuva 1 havainnollistaa uusien muutoksien käyttöönoton jatkuvaa integraatiota hyödyntäen.



Kuva 1. Jatkuvan integraation näkyminen projektissa. CI/CD-putki käynnistyy uusien muutosten seurauksena.

Edellä oleva kuva havainnollistaa, miten uusien muutoksien puskeminen GitHubiin käynnistää automaattisesti itse määrittelemäni CI/CD-putken. Mikäli koodin kääntäminen tai rakentaminen ei mene onnistuneesti läpi, sovellusta ei pakata, vaan kehittäjälle tulee ilmoitus epäonnistuneesta prosessista. Tällä tavalla minimoidaan virheellisten versioiden julkaisu tuotantoympäristöön. Mikäli kaikki meni onnistuneesti läpi, GitHub Actions pakkaa sovelluksen ja julkaisee sen Azureen, jonka jälkeen uusin versio on heti saatavilla käyttäjille automaattisesti tiepalvelun verkkosivulla.

3.5 Tekoälyn hyödyntäminen

Tekoäly oli käytössä projektin kehitysprosessissa. Sovellusta kehittäessä minulla tuli vastaan uusia kokonaisuuksia, joiden nopeassa omaksumisessa tekoälystä oli apua. Käytin OpenAI:n kehittämää keskustelubotti ja virtuaaliavustaja ChatGPT:tä tiedon hakuun sekä ideointiin. Koodaamisen tukena käytin GitHubin tekoälyyn pohjautuvaa ohjelmointityökalu Copilottia. Tekoälystä oli apua esimerkiksi Googlen kartan toteuttamisessa sovellukseen Googlen oman dokumentaation hyödyntämisen rinnalla. Vaihtoehtoisia toteutustapoja suunnitellessa tekoälyn kanssa pystyi keskustelemaan vaihtoehtojen mahdollisista hyödyistä ja haitoista, mikä auttoi omassa pohdinnassa. Kaikissa tapauksissa tekoälyä ei kuitenkaan ollut tarpeen hyödyntää, vaan perinteinen tiedonhaku toimi luotettavasti.

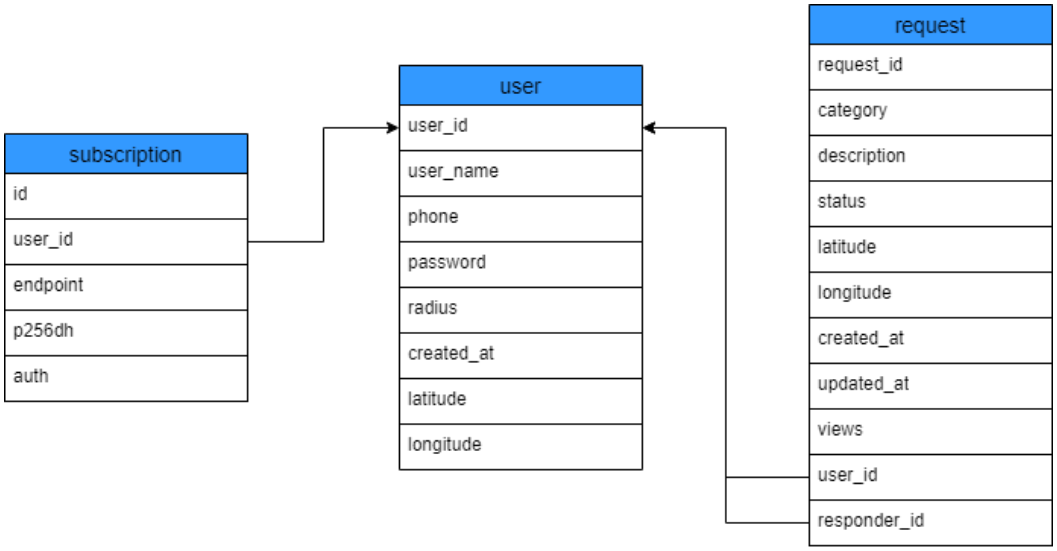
4 Sovelluksen arkkitehtuuri

4.1 Sovelluksen rakenne ja komponentit

Projektissani frontend sekä backend ovat saman koodipohjan (code base) alla. Halusin pitää sovelluksen kehityksen suoraviivaisena ja kaiken koodin samassa paikassa. Isommissa projekteissa se ei välttämättä ole optimaalisin vaihtoehto, mutta tämän kokoisessa sovelluksessa se oli mielestäni toimiva lähestymistapa. Frontendin ja backendin kommunikointi tapahtuu REST API:n sekä Server-Sent Eventsin (SSE) avulla.

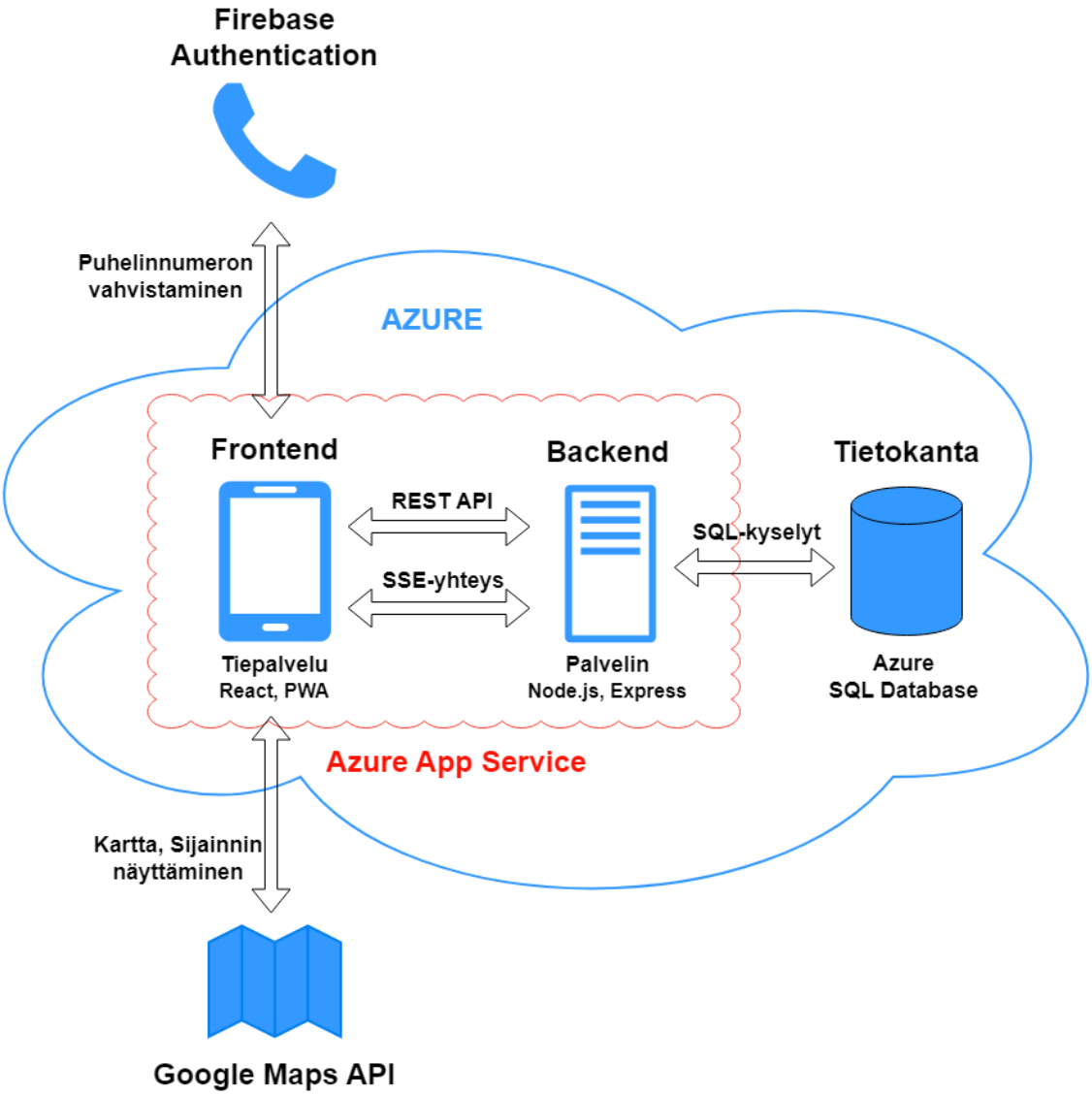
SSE:n avulla hoidetaan sovelluksen frontendin päivittäminen live-ajassa, mikäli useampi käyttäjä käyttää sovellusta samaan aikaan. Nämä livepäivitykset tulevat hyödyllisiksi silloin, kun käyttäjä esimerkiksi muokkaa oman pyyntönsä tilaa ja toisella käyttäjällä on oma sovelluksensa samanaikaisesti auki. SSE:n avulla heidän näkymänsä päivittyy heti, kun tietokannassa olevaan pyyntöön tulee muutoksia.

Kuvassa 2 on esitetty sovelluksen tietokannan taulut ja niiden kentät. Tietokannassa on kolme taulua, joita ovat aiemmin mainitut käyttäjät (user), pyynnöt (request) sekä tilaukset (subscription). Käyttäjien taulussa on pääavaimen (user_id) lisäksi käyttäjänimi, puhelinnumero, kryptattu salasana, säde, luontiaika sekä sijainti leveys- ja pituusasteineen. Pyynnöillä on pääavaimen (request_id) lisäksi kategoria, kuvaus, tila, pituus- ja leveysaste, luonti- ja muokkausaika, katselukerrat, pyynnön luojan id sekä mahdollinen pyyntöön vastaajan id, jotka viittaavat käyttäjätaulun käyttäjän id:hen. Tilauksilla on pääavain id, käyttäjän id, joka viittaa samoin käyttäjätaulun käyttäjä id:hen sekä selaimen endpoint, p256dh ja auth, jotka yksilöivät käyttäjän selaimen.



Kuva 2. Sovelluksen tietokannan rakenne.

Kuvassa 3 on havainnollistettu sovelluksen arkkitehtuuria tarkemmin. Frontend ja backend ovat saman Azure App Servicen alla, ja se on sijoitettu samaan paikkaan Azuren pilvipalveluun kuin tietokantakin. Backend hoitaa kaikki sovelluksen tietokantakyselyt, eli tiedon hakemisen ja päivittämisen. Frontend ja backend keskustelevat REST API -kutsuilla sekä SSE-yhteydellä, kuten aiemmin mainittu. Azuren ulkopuolisina palveluina näkyy Firebasen puhelinnumeron vahvistus sekä Google Maps API, joita frontend käyttää.



Kuva 3. Sovelluksen arkkitehtuurikaaviossa on esitetty frontend, backend, tietokanta, Azuren pilvipalvelu sekä Azure App Service ja ulkopuolisina palveluina Firebase sekä Google Maps API.

4.2 Palvelinympäristö ja tietoturva

Frontendin ja backendin ollessa saman projektin alla, ovat ne myös Azuressa samassa Azure App Servicessä. Tämä tarkoittaa sitä, että ne käyttävät samaa porttia ja jakavat samat osoitteet. Esimerkiksi ”tiepalvelu/login” on frontendin käyttämä osoite ja ”tiepalvelu/api/user/login” on backendin käyttämä osoite.

Tietoturva on olennainen osa sovelluksen arkkitehtuuria. Käyttäjän tunnistukseen on käytössä JWT-tokenit, joiden avulla todennus on turvallista. JWT-token eli JSON Web Token on JSON-pohjainen avoimen standardin menetelmä käyttöoikeustietueiden hallinnoimiseen eri ohjelmistojen välillä. Yleisimpiä käyttökohteita on esimerkiksi käyttäjän varmentaminen, eli onko käyttäjälle myönnetty oikeus haluttuun kohteeseen [23]. Näiden avulla varmistetaan, että vain valtuutetut käyttäjät pääsevät käsiksi suojattuihin resursseihin.

Sovelluksen frontend-komponentit on jaettu julkisiin ja yksityisiin reitteihin. Julkisiin (public) reitteihin käyttäjällä on pääsy milloin tahansa ilman tunnistautumista, mutta yksityisiin (private) reitteihin käyttäjän täytyy olla kirjautunut sisään. Jokainen reitti saa parametrina tai ”proppina” tiedon siitä, onko käyttäjä kirjattu sisään. Yksityiset reitit ohjaavat käyttäjän automaattisesti takaisin kirjautumiseen, mikäli käyttäjä yrittää päästä niihin ilman sisäänkirjautumista. Sisäänkirjautumisen tarkistus tapahtuu erillisellä tokenin validointifunktiolla, joka tarkistaa, onko JWT-token validi.

Esimerkkikoodi 1 havainnollistaa yksityisten ja julkisten reittien käyttöä Reactissa. Esimerkkikoodissa on asennussivu ”/install”, josta PWA-sovelluksen voi ladata laitteelleen. Tämä on sovelluksen etusivu, mikäli ollaan selainnäkymässä. Asennussivu on julkinen reitti, eli sovelluksen voi asentaa laitteelleen ilman sisäänkirjautumista. Seuraavana on kirjautumissivu ”/login”, joka on loogisesti myös julkinen reitti. Viimeisenä on sovelluksen päänäkymä, jonka osoite on juuressa ”/”. Tämä on yksityinen reitti, johon pääsee vain tunnistautumalla.

```
<Route
   path="/install"
   element={
      <PublicRoute
          isLoggedIn={isLoggedIn}
          element={<Install />}
      />
   }
/>
<Route
   path="/login"
   element={
      <PublicRoute
          isLoggedIn={isLoggedIn}
          element={<Login setIsLoggedIn={setIsLoggedIn} />}
      />
   }
/>
<Route
   path="/"
   element={
      <PrivateRoute
          isLoggedIn={isLoggedIn}
          element={<Main />}
      />
   }
/>
```

Esimerkkikoodi 1: Julkiset ja yksityiset reitit Reactissa.

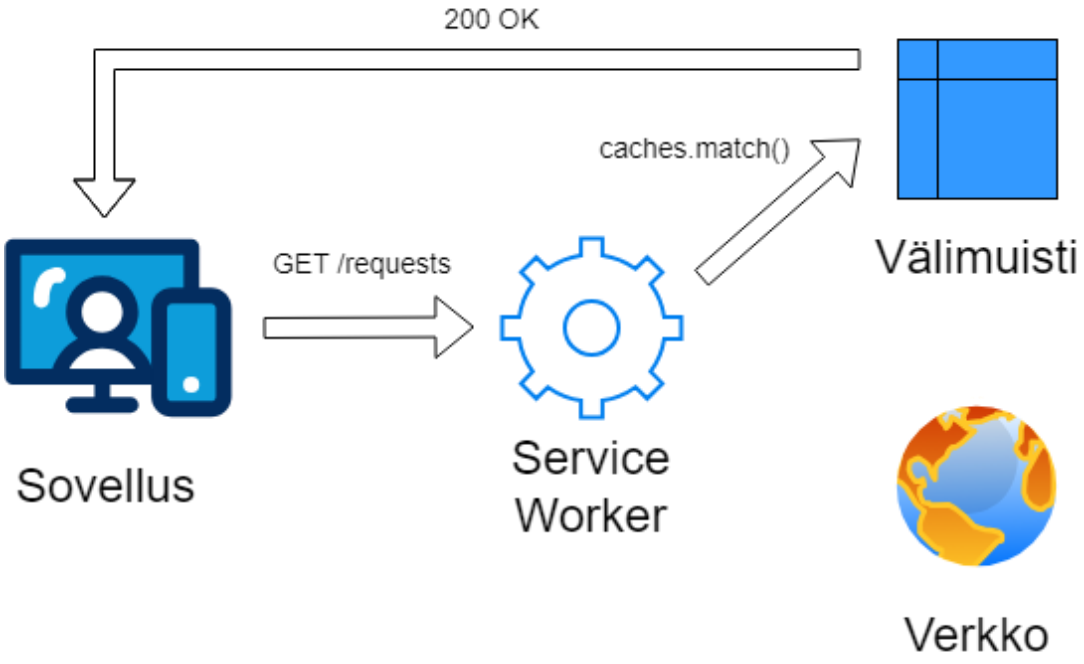
5 PWA:n hyödyntäminen

5.1 Progressiivinen verkkosovellus

Progressive web app (PWA) on verkkosovellus, jonka käyttäjä voi asentaa laitteelleen. Tällöin sovellus käyttäytyy kuten natiivi mobiilisovellus. Sovellusta voi käyttää joko normaalisti selaimen kautta, tai asennettuna versiona oman laitteen kotivalikon kautta. PWA:n avulla verkkosovelluksesta voidaan tehdä enemmän natiivisovelluksen kaltainen, ja progressiivinen verkkosovellus on tehty parantamaan verkkosovelluksen käyttäjäkokemusta mobiililaitteella [24]. Tavoitteena oli tutkia, miten tämän tyylinen ratkaisu soveltuu nyt kehitettävän projektin tarpeisiin.

PWA tarjoaa useita ominaisuuksia, joita myös natiivisovellukset tarjoavat. Näihin ominaisuuksiin kuuluu esimerkiksi ilmoitusten (push notifications) lähettäminen laitteeseen sovelluksen ollessa kiinni, mikä on keskeisessä osassa toteuttamassani sovelluksessa. PWA:n avulla sovellus voi toimia myös offline-tilassa ilman internetyhteyttä, mikä ei ole mahdollista tavallisesti selaimessa.

Kaiken edellä kuvatun mahdollistaa palvelunvälittäjä eli service worker, joka toimii välikappaleena sovelluksen sekä palvelimen välissä. Se tallentaa välimuistiin kaiken tarvittavan tiedon, jotta verkkosovellus voi toimia offline-tilassa ja käyttää viimeisintä saatavilla olevaa dataa. Service worker on muusta sovelluksesta riippumaton, ja se toimii täysin asynkronisesti. Sovelluksessani offline-toiminnallisuudesta ei ole hyötyä, mutta sen avulla sovelluksen voi kuitenkin avata ilman internetyhteyttä, toisin kuin tavallisen nettisivun. Service worker myös vastaanottaa palvelimelta tulevat push-ilmoitukset ja välittää ne käyttäjän laitteeseen. Kuvassa 4 on havainnollistettu service workerin toimintaa sovelluksen ja verkon sekä välimuistin välissä.



Kuva 4. Service worker käyttää välimuistissa olevaa dataa offline-tilassa. [25.]

React tarjoaa valmiin komennon PWA-projektin alustamiseen, jota myös projektissani käytin (esimerkkikoodi 2).

```
npx create-react-app road-service --template cra-template-pwa-typescript
```

Esimerkkikoodi 2: Komento Reactin valmiin PWA-projektin alustamiseen, joka luo automaattisesti service workerin projektiin.

Edellä esitetty komento luo automaattisesti service-worker.ts- sekä serviceWorkerRegistration.ts-tiedostot. Nämä tiedostot sisältävät kaiken tarvittavan koodin valmiiksi, jotta PWA-sovellus on ladattavissa käyttäjän laitteelle. Service worker ei automaattisesti kuitenkaan osaa käsitellä push-ilmoituksia, joten niiden käsittely ja vastaanotto täytyi toteuttaa ja lisätä itse.

5.2 Push-ilmoitusten toteuttaminen

Push-ilmoitukset ovat yksi tärkeimmistä, ellei jopa tärkein ominaisuus koko projektissani. Tiepalvelun käyttö tuotantokäytössä on pääosin passiivista, mikä tarkoittaa sitä, että käyttäjille pitää saada tieto akuutisti avuntarpeessa olevasta pyöräilijästä ilman sovelluksen aktiivista käyttöä. Projektia suunniteltaessa esiteltiin muutamia vaihtoehtoja tästä tiedon välittämisestä käyttäjälle, mutta yhteistyökumppaneiden kanssa päätimme, että push-ilmoitukset puhelimeen olisivat optimaalisin ja käyttäjäystävällisin tapa.

Push-ilmoitukset otetaan käyttöön selainkohtaisella tilauksella (subscription). Jotta käyttäjälle voidaan lähettää push-ilmoituksia selaimen kautta, hänen täytyy ensin antaa lupa ilmoituksille, minkä jälkeen selain tekee tilauksen palvelimen ilmoitusjärjestelmään. Tehty tilaus sisältää olennaisia tietoja, kuten selaimen tunnisteen ja ilmoitusten reititykseen tarvittavat tiedot, jotka ovat yksilöllisiä kyseiselle laitteelle ja selaimelle. Ilman tätä tilausta palvelin ei pysty lähettämään ilmoituksia oikeaan selaimeen. Tilausten tiedot ja rakenne on esitelty kuvan 2 tietokantakaaviossa alaluvussa 4.1. Seuraavina esitetyissä koodiesimerkeissä on havainnollistettu push-ilmoitusten lisääminen progressiiviseen verkkosovellukseen.

Ensimmäisenä määritellään ”payload”-muuttuja, joka pitää sisällään ilmoituksen otsikon sekä viestin (esimerkkikoodi 3). Tämä muuttuja alustetaan tapauskohtaisesti projektissani riippuen ilmoituksen tyypistä. Seuraavassa on esitetty esimerkki ilmoituksen sisällöstä, joka lähetetään kaikille käyttäjille, joiden asettaman säteen sisältä uusi ilmoitus on lähetetty.

```
const payload = JSON.stringify({
      title: 'Uusi pyyntö',
      body: 'Pyöräilijä tarvitsee apua lähelläsi!',
});
```

Esimerkkikoodi 3: ”Payload”-muuttujan otsikoksi asetetaan ”Uusi pyyntö” ja viestiksi ”Pyöräilijä tarvitsee apua lähelläsi!”.

Tämän jälkeen kutsutaan ilmoituksen lähetysfunktiota tilanteen mukaan. Se voi olla esimerkiksi ilmoituksen lähettäminen jokaiselle käyttäjälle, jos ilmoituksessa ei ole sijaintia jaettuna, tai lähettäminen kaikille säteen sisällä oleville käyttäjille. Uuden pyynnön luomisen yhteydessä tarkistetaan, onko sijaintitiedot jaettu. Esimerkkikoodissa 4 kutsutaan ”notifyUsersWithinRadius”-funktiota, joka lähettää ilmoituksen säteen sisällä oleville käyttäjille. Tämä funktio ottaa parametreinä äsken luodun ”payload”-muuttujan, käyttäjän sijaintitiedot sekä käyttäjän ID:n, jonka avulla ilmoitusta ei lähetetä pyynnön luojalle itselleen.

```
await notifyUsersWithinRadius(userId, latitude, longitude, payload);
```

Esimerkkikoodi 4: ”notifyUsersWithinRadius”-funktion kutsuminen ja sen parametrit.

Edellä esitetty funktio hakee ensin kaikki ilmoitusten tilaukset tietokannasta, minkä jälkeen se tarkistaa silmukassa, ovatko tilauksen käyttäjän sijaintitiedot saatavilla. Jos sijaintitiedot ovat saatavilla, funktio laskee uuden pyynnön sekä käyttäjän sijainnin etäisyyden toisistaan.

Käyttäjän osuessa asetetun säteen sisälle hänen tilauksensa (subscription) lisätään tilauslistaan, joille push-ilmoitus tullaan lähettämään (esimerkkikoodi 5). ”Record”-muuttuja pitää sisällään kaikki tarvittavat tiedot käyttäjästä, eli sijainnin, säteen sekä ilmoitusten tilauksen.

```
if (distance <= record.radius) {
    const subscription: PushSubscription = {
        endpoint: record.endpoint,
        keys: {
            p256dh: record.p256dh,
            auth: record.auth,
        },
     };
     usersInRadius.push(subscription);
}
```

Esimerkkikoodi 5: Jos uusi ilmoitus osuu käyttäjän säteen sisälle, hänen push-ilmoituksiensa tilaus lisätään niiden käyttäjien listaan, joille ilmoitus tullaan lähettämään.

Tämän jälkeen ”usersInRadius”-lista käydään silmukassa läpi, ja push-ilmoitus lähetetään kaikille tässä listassa oleville tilauksille. Esimerkkikoodin 6 ”sendNotification”-funktiota kutsutaan silmukassa jokaisen käyttäjän kohdalla, joille ilmoitus lähetetään. Parametreina se ottaa käyttäjän tilauksen tiedot, sekä edelleen mukana kulkevan ”payload” -muuttujan, joka pitää sisällään ilmoituksen viestin.

```
await webPush.sendNotification(subscription, payload);
```

Esimerkkikoodi 6: Funktio, joka lähettää ”webPush”-ilmoituksen service workerille.

WebPush on kirjasto, joka mahdollistaa selaimen ja palvelimen välisten ilmoitusten lähettämisen. Backend lähettää web push -ilmoituksen tarvittavilla parametreilla, kuten edellä esitetyssä koodissa, ja selaimen service worker vastaanottaa kyseisen ilmoituksen. Ilmoitus lähetetään selaimen push-palvelimen kautta ja service worker vastaanottaa ilmoituksen taustalla ja näyttää sen käyttäjän laitteessa push-ilmoituksena, vaikka selain tai sovellus ei olisikaan aktiivinen. [26.]

Service workerille on lisätty tapahtumakuuntelija, joka suoritetaan, kun push-ilmoitus saapuu (esimerkkikoodi 7). Tämä tapahtumakuuntelija muuttaa saapuvan datan JSON-muotoon. ”Options”-muuttujalla määritellään ilmoituksen ominaisuudet sekä ulkoasu. Ilmoituksen sisältö tulee backendistä lähetetyn viestin mukana, ja se asetetaan ”body”:n arvoksi. Ilmoituksen kuvake, värinä

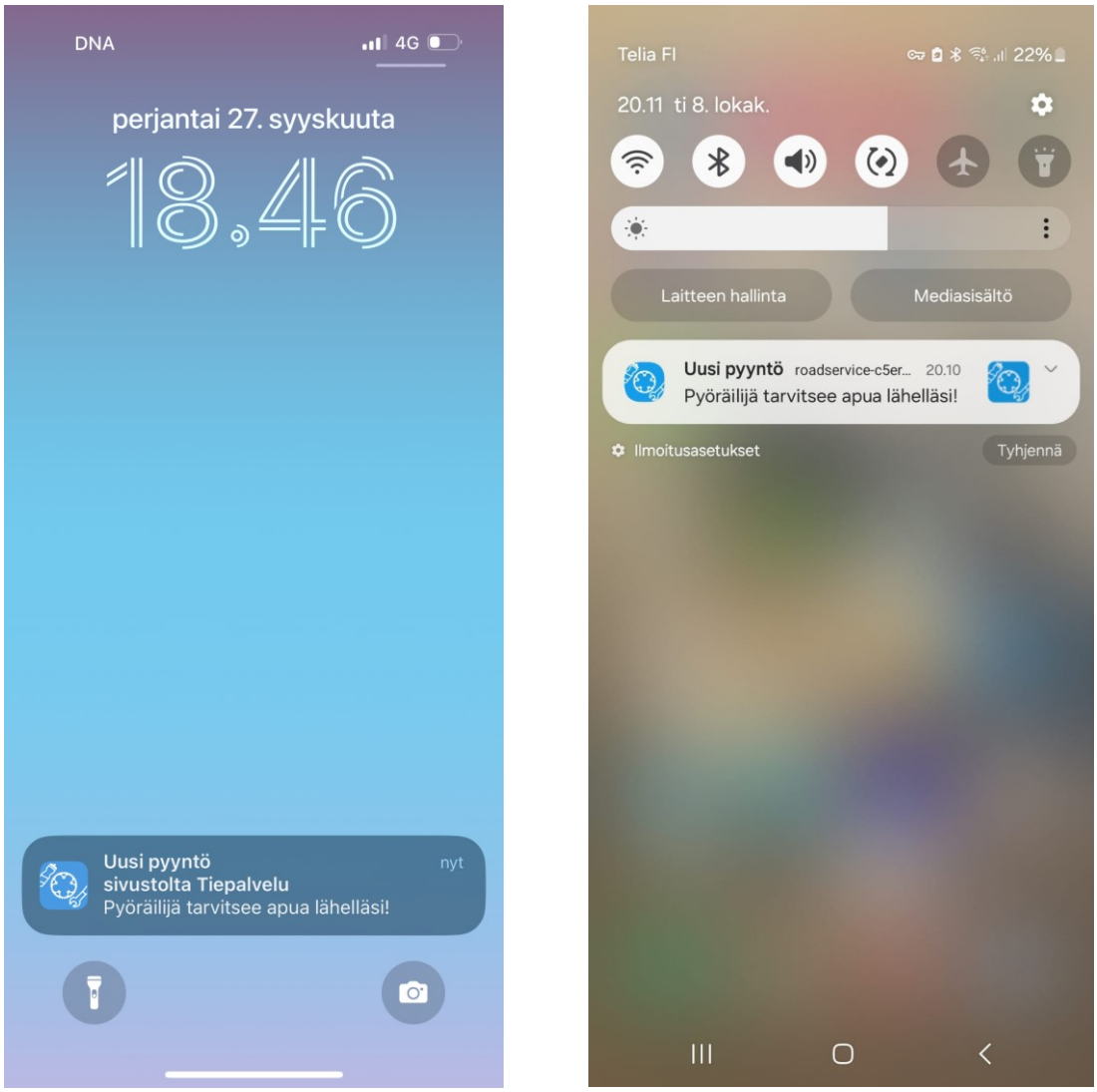
sekä mahdolliset lisätiedot kuten aikaleima asetetaan tämän jälkeen. Lopuksi service worker lähettää käyttäjän laitteeseen backendistä saatavan otsikon ilmoituksena, joka sisältää myös juuri määritellyt asetukset.

```
self.addEventListener('push', (event) => {

  const data = event.data.json();

  const options = {
    body: data.body,
    icon: '/favicon.ico',
    vibrate: [100, 50, 100],
    data: {
      dateOfArrival: Date.now(),
      primaryKey: 1,
      url: '/'
    },
  }
  event.waitUntil(
    self.registration.showNotification(data.title, options))
});
```

Esimerkkikoodi 7: Service workerin ”push”-tapahtumakuuntelijan määritys.

Näiden vaiheiden jälkeen ilmoitus tulee näkyviin käyttäjän laitteelle, vaikka sovellus olisi kiinni.

Kuvassa 5 on esitetty kaksi tilannetta äskeisten vaiheiden jälkeen lähetetystä ilmoituksesta, joka ilmoittaa uudesta pyynnöstä käyttäjän lähellä. Ensimmäinen kuvakaappaus on iOS- ja toinen Android-laitteella näkyvästä push-ilmoituksesta.



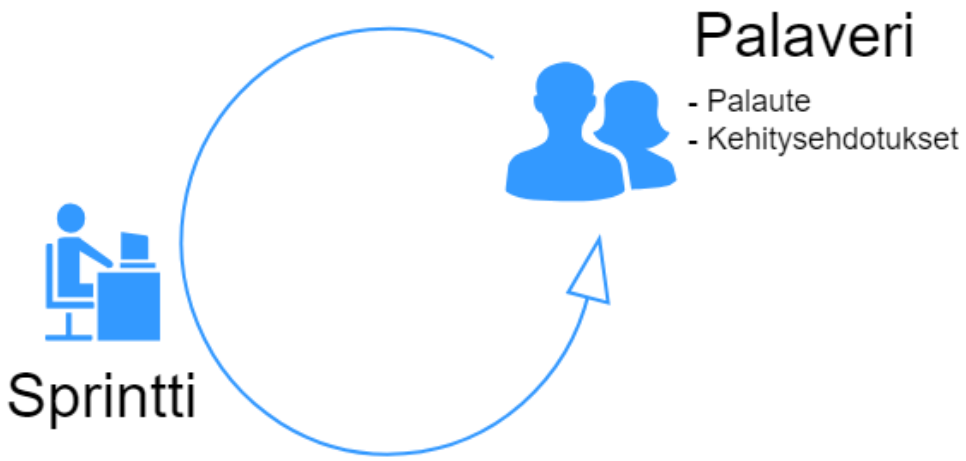
Kuva 5. Esimerkki-ilmoitus uudesta pyynnöstä sovelluksen ollessa kiinni iOS:llä sekä Androidilla.

Ilmoituksissa näkyy sovelluksen logo sekä ilmoituksen otsikko ja teksti. Sovellus aukeaa natiivisovelluksen tavoin push-ilmoitusta napsauttamalla. Android-laitteella aktiivinen ilmoitus näkyy myös sovelluksen kuvakkeessa kotinäytöllä numerona kuvakkeen yläreunassa.

6 Tiepalvelusovelluksen rakentaminen

Toteutin projektin Scrum-menetelmää mukauttaen. Scrum-menetelmän avulla voidaan tehdä lyhyen ja keskipitkän aikavälin suunnittelu sekä sisällön ja työtapojen säännöllinen palautesilmukka [27]. Toteutin sovelluksen yksin, joten menetelmän täysiä periaatteita ei ollut mahdollista hyödyntää. Totesin kuitenkin, että Scrum-menetelmän sprintit sopivat työtapoihini hyvin. Sprinteillä pystyin rytmittämään ja aikatauluttamaan kehitysprosessin, sillä mitään varsinaisia määräaikoja ei yhteistyökumppaneiden toimesta ollut asetettu. Jaoin suunnitteluvaiheessa työprosessin seitsemään eri sprinttiin. Sprinttien jälkeen pidettiin palaverit yhteistyökumppaneiden kanssa tuoreimmista uudistuksista sekä toteutetuista ominaisuuksista ja niiden käytettävyydestä (kuva 6).

Palavereissa yhteistyökumppanit esittivät kehitysehdotuksia projektin edetessä. Projektissa kehitettiin täysin uusi sovellus, joten yhteistyökumppaneiden toiveet selkiytyivät ja muuttuivat työn edetessä.



Kuva 6. Scrum-menetelmän soveltaminen yksin kehitetyssä sovelluksessa, jossa yhteistyökumppanit toimivat palautteen antajina ja kehitysehdotusten tekijöinä.

Sovelluksen aktiivinen jatkuva testaus aloitettiin testausryhmän toimesta heti, kun julkaisin sovelluksen Azureen. Sovellusta testattiin aktiivisesti sekä Androidilla että iOS:llä.

6.1 Alustava käyttöliittymän toteutus

Ensimmäisessä kaksi viikkoa kestäneessä sprintissä lähdin hahmottelemaan sovellukselle mahdollista käyttöliittymää. Vaikka tyyleihin ei vielä tässä vaiheessa käytetty sen enempää aikaa, oli tärkeä saada kaikki olennaiset komponentit käyttöliittymään paikalleen. Näin sain selkeän kuvan siitä, mitä sovelluksessa tulee olla, minkä jälkeen toiminnallisuuksia oli helpompi lähteä kehittämään.

Pyrin rakentamaan sovellukseni käyttöliittymän mahdollisimman kevyeksi käyttää. Päädyinkin käyttämään Reactin modaaleja eri sivujen tekemiseen. Sovelluksessa tulee olemaan yksi päänäkymä, joka on listanäkymä kaikista aktiivisista pyynnöistä, jonka päälle aukeavat eri modaalit. Näitä modaaleja ovat yksittäisen pyynnön sivu, uuden pyynnön lähettämissivu sekä asetussivu. Alaluvussa 6.6 esiteltävissä käyttöliittymäkuvissa näkee nämä modaalit käytännössä, jolloin taakse jää sovelluksen päänäkymä, ja modaali aukeaa sen päälle. Painamalla rastia oikeasta yläkulmasta modaalin saa suljettua.

6.2 Tietokannan alustus ja käyttäjän autentikaatio

Toisessa kaksi viikkoa kestäneessä sprintissä toteutin Azureen sijoitetun tietokannan alustuksen. Loin ensimmäisenä alaluvussa 4.1 esitellyn kuvan 2 mukaisen käyttäjätaulun tietokantaan. Sprintin tarkoituksena oli saada käyttäjän kirjautuminen sekä rekisteröityminen toimimaan. Myös kirjautumisen sekä rekisteröitymisen käyttöliittymät saivat hieman päivitystä. Lisäsin backendiin reitit kirjautumisen sekä rekisteröitymisen lisäksi myös salasanan vaihtamiselle.

6.3 Dokumentaation alustus ja testien kirjoittaminen

Kolmas sprintti oli lyhempi, vain viikon mittainen, jonka aikana alustin sovelluksen dokumentaatiota sekä testejä. Dokumentaation alustukseen kuului GitHub-repositorion README-tiedoston kirjoitus, sovelluksen asennus- ja käyttöohjeiden kirjoitus sekä api-dokumentaation kirjoitus. Kirjoitin myös ensimmäiset

yksikkötestit frontendille sekä backendille, mutta niiden viimeistely jäi myöhempään vaiheeseen. Sovelluksessani käytännön testaus yhteistyökumppaneiden kanssa oli mielestäni tarpeeksi kattavaa, joten päätin pitää tämän sprintin lyhyenä ja lähteä seuraavaksi kehittämään sovelluksen muita toiminnallisuuksia.

6.4 Pyyntöjen toteutus ja ensimmäinen prototyyppi

Neljäs sprintti oli kolmen viikon mittainen, ja siihen kuului paljon tehtäviä. Isoimpina kokonaisuuksia tein ensimmäisen toimivan version pyyntöjen lähettämisestä sekä käyttäjätietojen hallinnasta ja muokkaamisesta. Tämän sprintin aikana tavoitteena oli saada valmiiksi ensimmäinen prototyyppi sovelluksesta, jonka esittelin yhteistyökumppaneille.

Ensimmäisenä lisäsin tietokantaan alaluvussa 4.1 esitellyn kuvan 2 mukaisen taulun pyynnöille. Tämän jälkeen toteutin pyyntöjen käsittelyt, eli niiden lisäämisen, päivittämisen sekä poistamisen.

Käyttöliittymä sai myös päivitystä tässä vaiheessa. Loin ensimmäisen etusivun listanäkymän kaikista pyynnöistä. Asetussivu sai myös päivitystä, kun lisäsin käyttäjän tietojen muokkaamisen sekä käyttäjätilin poistamisen. Lisäsin myös pyyntöihin vastaamisen tässä vaiheessa, jotta kaikki tärkeimmät ominaisuudet olisivat mukana ensimmäisessä prototyypissä. Google Maps API otettiin käyttöön tässä sprintissä, jotta pyynnön sijainti saatiin kartassa näkyviin. Viimeisenä työtehtävänä ennen prototyypin valmistumista lisäsin päänäkymän listaan toiminnallisuuden, joka näyttää käyttäjälle ensisijaisesti hänen asettaman säteen sisältä tulleet pyynnöt. Kaikki aktiiviset pyynnöt saa näkyviin halutessaan ”Näytä kaikki” -painiketta painamalla.

6.5 Tietoturva, rajoitukset sekä käyttäjäkokemus

Viides sprintti oli myös kolmen viikon mittainen, ja se sisälsi paljon erityyppisiä työtehtäviä. Käyttäjäkokemuksen parantaminen oli yksi teemoista, johon

kuuluivat muun muassa vahvistusnäytön lisääminen pyynnön sekä käyttäjän poistamiselle sekä parempien ja selkeämpien virheviestien näyttäminen.

Sovelluksen tietoturvaa paransin tässä sprintissä. Lisäsin kaikkiin käyttäjän syötteisiin validoinnin niin käyttäjätiedoissa kuin myös pyynnöissä. Toteutin frontendin puolella alaluvun 4.2 mukaiset yksityiset (private) ja julkiset (public) reitit. Jokainen käyttöliittymän komponentti käärittiin joko yksityiseen tai julkiseen reittiin. Kirjautuminen, rekisteröityminen sekä salasanan nollaus ovat julkisia reittejä, ja itse sovelluksen kaikki näkymät ovat yksityisiä reittejä.

Backendin puolelle lisäsin tokenin tarkistuksen (middleware token authentication) jokaiseen reittiin, jossa tarvitaan käyttäjätietoja. Tällä viimeistään huomataan ja pystytään estämään tilanne, jossa käyttäjä on päässyt tekemään API-kutsuja ilman sisäänkirjautumista.

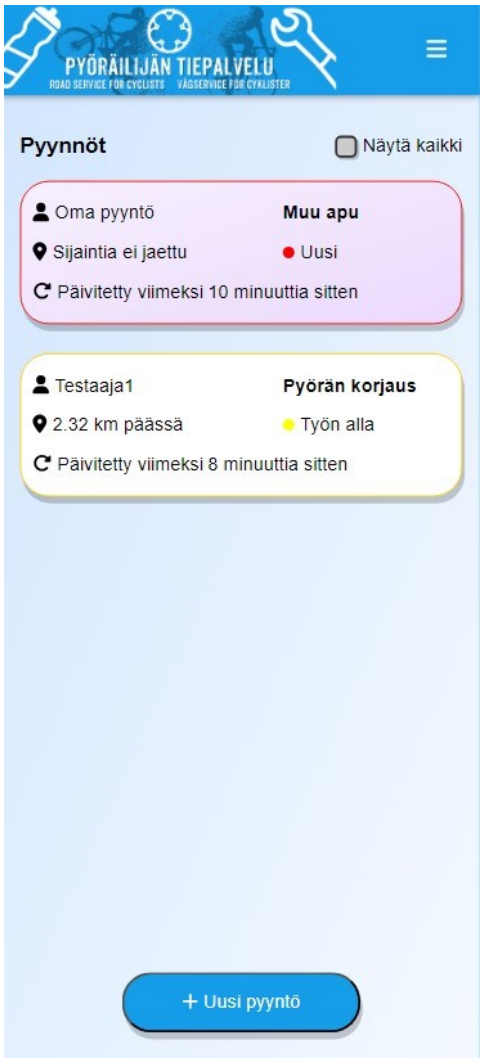
Tässä sprintissä lisäsin myös rajoituksia, joilla estetään mahdollista häiriökäyttöä. Lisäsin esimerkiksi rajoitukset, joiden mukaan käyttäjällä ei voi olla kuin kaksi aktiivista pyyntöä samanaikaisesti, sekä käyttäjäkohtaisen päivittäisen pyyntörajan, joka on viisi pyyntöä. Pyyntöön vastaamiseen lisäsin myös rajoituksen, joka antaa yhden käyttäjän vastata vain kerran yhteen tiettyyn pyyntöön. Näillä rajoituksilla estetään mahdollista häiriökäyttöä (spamming).

6.6 Beta-version käyttöliittymän viimeistely

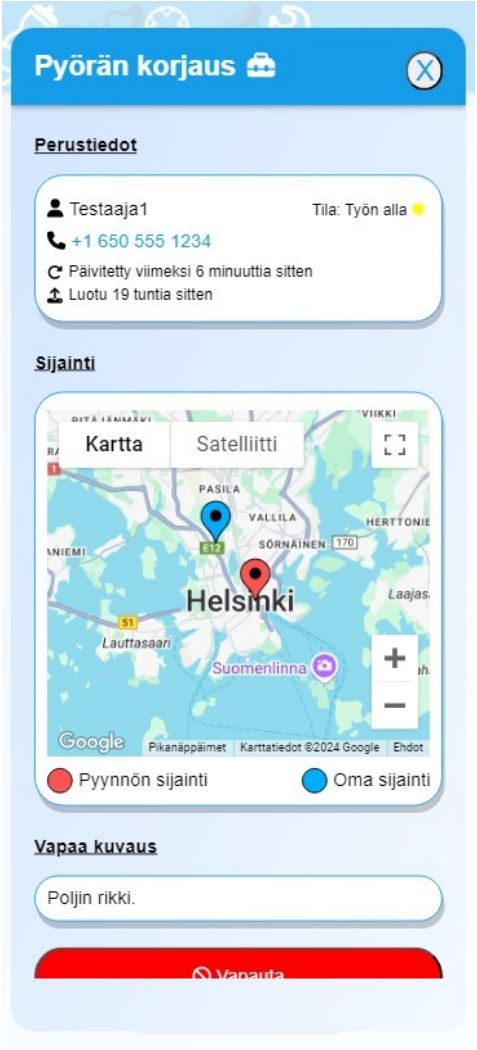
Kuudes sprintti oli viikon mittainen. Tässä sprintissä viimeistelin käyttöliittymän siihen tilaan, jossa se tulee olemaan valmiissa beta-versiossa. Nykyisiä käyttöliittymätrendejä mukaillen lisäsin sovellukseni käyttöliittymäkomponentteihin pyöreitä ja pehmeitä reunoja muun muassa painikkeisiin. Sovelluksen logo sekä väriteemat tulivat valmiina yhteistyökumppaneilta.

Kuvissa 7 ja 8 näkyy valmiin beta-version kaksi eri näkymää. Kuvassa 7 on sovelluksen päänäkymä, jossa näkyy kaksi aktiivista pyyntöä. Ensimmäinen pyynnöistä on käyttäjän itse lähettämä ja toinen on käyttäjältä ”Testaaja1”. Käyttäjän oma pyyntö on tilassa ”uusi”, eli kaikilla rekisteröityneillä käyttäjillä on

mahdollisuus vastata tähän pyyntöön. Testaaja1:n pyyntö on puolestaan tilassa ”työn alla”, eli yksi rekisteröityneistä käyttäjistä on vastannut pyyntöön ja ottanut sen itselleen työn alle. Kuvassa 8 näkyy Testaaja1:n pyyntö avattuna, missä näkyy pyyntöön liittyviä tietoja, kuten käyttäjän puhelinnumero, sijainti sekä vapaa kuvaus.



Kuva 7. Sovelluksen päänäkymä.

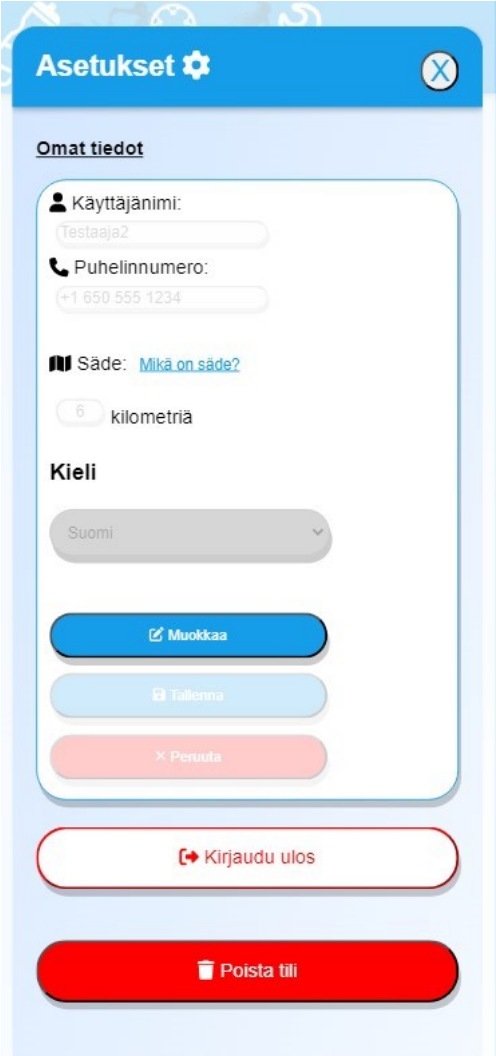


Kuva 8. Pyynnön näkymä.

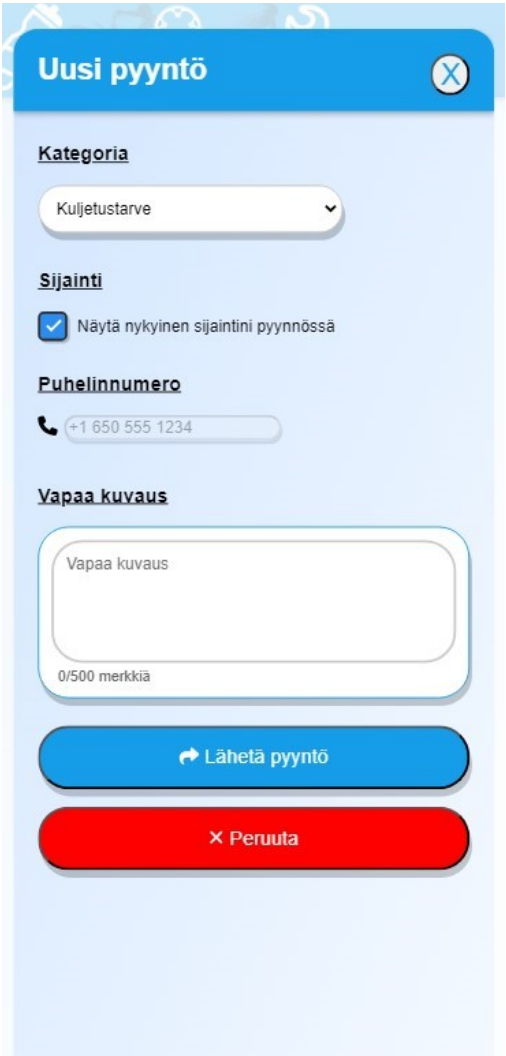
Kuvissa 9 ja 10 on esitetty toiset kaksi näkymää, asetussivu sekä uuden pyynnön lähettämissivu. Asetussivulta löytyy käyttäjän käyttäjänimi, puhelinnumero, valitsemansa säde sekä kieli. Asetussivulta voi myös kirjautua ulos tai poistaa tilin kokonaan. Käyttäjä pystyy muokkaamaan kaikkia muita tietojaan jälkikäteen paitsi puhelinnumeroaan. Sen muokkaaminen on yksi jatkokehityksen kohteista.

Säteen vieressä on painike ”Mikä on säde?”, josta käyttäjä pääsee näkemään tarkemman kuvauksen säteen tarkoituksesta.

Uusi pyyntö -sivulta käyttäjä pääsee luomaan uuden pyynnön mahdollisesta avun tarpeesta. Käyttäjä valitsee kategorian pudotusvalikosta sekä valitsee, haluaako jakaa sijaintinsa pyynnössä. Mikäli sijainti on jaettu, ilmoitus lähetetään vain lähellä oleville käyttäjille, joiden säteen sisälle tämä uusi pyyntö osuu. Mikäli sijaintia ei jaeta, ilmoitus lähetetään kaikille käyttäjille. Puhelinnumeroa ei voi muokata, mutta se on kuitenkin ilmoitettu uusi pyyntö -sivulla selvyyden vuoksi. Viimeisenä käyttäjä syöttää vapaaseen kuvauskenttään tarkemman tiedon avuntarpeesta. Vapaa kuvaus voi olla 1–500 merkkiä pitkä.



Kuva 9. Sovelluksen asetussivu.



Kuva 10. Uuden pyynnön luomissivu.

6.7 Sovelluksen käyttöönotto ja loput toiminnallisuudet

Seitsemäs ja viimeinen sprintti oli kolmen viikon mittainen. Siihen sisältyi itse sovelluksen julkaiseminen (deployment) Azureen sekä kaikkien viimeisten puuttuvien ja edellisiltä sprinteiltä rästiin jääneiden toiminnallisuuksien toteuttaminen. Isoimpina kokonaisuuksina tässä sprintissä oli sovelluksen Azureen julkaisun yhteydessä toimintaan laitettu GitHub Actionsin CI/CD-putki, push-ilmoitukset, SSE-livepäivitykset sekä Firebasen puhelinnumeron varmistus.

Sovelluksen aktiivinen käyttäjätestaus aloitettiin heti, kun sovellus oli julkaistu Azureen ja linkki sovellukseen jaettiin yhteistyökumppaneille sekä muille valituille testaajille. Testaamisesta kerrotaan tarkemmin alaluvussa 6.9.

6.8 Sovelluksen toiminnallisuudet ja testaaminen

Viimeisen sprintin jälkeen sovellus oli valmiissa beta-versiossa, jossa oli valmiina kaikki tarvittavat toiminnallisuudet. Käyttäjä rekisteröityy palveluun käyttäjänimellä, puhelinnumerolla sekä salasanalla. Puhelinnumero vahvistetaan Firebasen kautta. Käyttäjä pystyy myös vaihtamaan salasanan puhelinnumeron vahvistuksen avulla. Lisäksi käyttäjät voivat muokata omaa käyttäjänimeään sekä valitsemaansa sädettä.

Käyttäjä voi lähettää pyyntöjä joko sijaintitiedoillaan tai ilman niitä riippuen pyynnön tyypistä. Toinen käyttäjä voi vastata pyyntöön ottamalla sen työn alle. Mikäli vastaaja pystyi tarjoamaan tarvittavan avun, niin pyynnön lähettäjä voi kuitata pyynnön valmiiksi. Jos vastaaja ei pystynyt tarjoamaan apua, voi kumpi tahansa osapuoli poistaa aktiivisen vastauksen pyynnöstä, jolloin pyyntö siirtyy takaisin tilaan ”uusi”. Pyynnön tilan muuttuminen lähettää uuden ilmoituksen kaikille käyttäjille, joille pyyntö on kohdistettu. Ilmoituksesta näkee, että pyyntö on uudelleen aktiivinen.

Beta-version valmistuttua sovellusta lähdettiin testaamaan isommalla ryhmällä yhteistyökumppaneiden kanssa.

6.9 Jatkokehitys

Sovelluksen kehitys jatkuu insinöörityön valmistumisen jälkeen, ja tarkoituksena on käynnissä olevan testivaiheen päätyttyä siirtyä tuotantoon ja julkaista sovelluksesta versio 1.0. Ennen tuotantovaiheeseen siirtymistä laaditaan tietosuojaseloste ja jatketaan tietoturvan kehittämistä. Jo olemassa olevien suomen- ja englanninkielisten palvelujen lisäksi tullaan tarjoamaan myös ruotsinkielinen palvelu.

Meneillään oleva testivaihe koostuu noin 30 testaajan ryhmästä, jotka tekevät testiluonteisia pyyntöjä, ottavat niitä työn alle ja antavat palautetta sovelluksen toiminnasta. Tämä vaihe on tärkeä ennen tuotantoon siirtymistä, jotta mahdolliset bugit löydetään ja korjataan jo testausvaiheessa. Testaajille on lähetetty kyselylomake, jossa kartoitetaan sovelluksen lataamisen, käytön ja toiminnallisuuksien toimivuutta ja käyttäjäystävällisyyttä. Testaamisen päätyttyä julkaistaan sovelluksen ensimmäinen täysi versio kaikkien käyttäjien saataville.

Yhteistyö jatkuu aktiivisesti yhteistyökumppaneiden kanssa sovelluksen julkaisemisen jälkeen. Insinöörityö kattoi sovelluksen saattamisen beta-versioon ja jatkokehitystä varten tullaan tekemään erillinen sopimus yhteistyökumppaneiden kanssa.

7 Yhteenveto

Insinöörityön tavoitteena oli toteuttaa beta-versio Pyöräilijän tiepalvelu -sovelluksesta. Toteutetussa sovelluksessa palveluun liittyneet pyöräilijät pystyvät lähettämään avunpyyntöjä muille käyttäjille maastossa ollessaan. Sovelluksen käyttö on pääosin passiivista, joten tieto avuntarpeesta täytyy saada käyttäjille ilman sovelluksen aktiivista käyttöä.

Työn toisena tavoitteena oli selvittää, miten PWA soveltuu tämänkaltaisen sovelluksen kehittämiseen. Sovelluksen käyttö tapahtuu pääosin mobiiliympäristössä. Toteutin verkkosovelluksen PWA:na, jolloin sovelluksen kehitystyö pystyttiin toteuttamaan samanaikaisesti Androidille sekä iOS:lle ilman erillisiä natiiviprojekteja. Sovelluksen tärkein ominaisuus oli saada push-ilmoitukset toimimaan, jotta sovellus palvelee käyttäjiä halutulla tavalla ja tieto avuntarpeesta saadaan käyttäjille nopeasti.

Toteutin sovellukselle käyttöliittymän (frontend), palvelimen (backend) sekä SQL-tietokannan, johon rekisteröityneet käyttäjät, heidän lähettämänsä pyynnöt sekä push-ilmoitusten tilaukset (subscriptions) tallennetaan. Frontendin sekä backendin kirjoitin TypeScriptiä käyttäen. Sovelluksen käyttöliittymän toteutin Reactilla ja käytin Reactin valmista komentoa PWA-projektin pohjustamiseen. Backendissä on käytössä Node.js ja Express. Sovelluksen sekä tietokannan sijoitin Azuren pilvipalveluun. Azuren ulkopuolisina palveluina käytin Googlen tarjoamaa Google Maps API:a sekä Firebasen puhelinnumeron varmistusta.

Toteutin sovelluksen Scrum-menetelmän sprinttejä hyödyntäen. Sprinteillä pystyin rytmittämään ja aikatauluttamaan kehitystyön hyvin, sillä yhteistyökumppanit eivät olleet asettaneet mitään määräaikoja. Sprinttien päätteeksi pidimme säännölliset palaverit yhteistyökumppaneiden kanssa. Palavereissa he antoivat palautetta ja uusia kehitysideoita.

Insinöörityön tuloksena saatiin toimiva beta-versio Pyöräilijän tiepalvelu -sovelluksesta. Insinöörityön valmistumisvaiheessa sovellus on laajemman käyttäjäkunnan testattavana. Jatkan kehitystyötä yhteistyökumppaneiden kanssa

insinöörityön valmistumisen jälkeen, jotta sovellus saadaan tuotantokäyttöön ja siitä tulee mahdollisimman ehyt ja käyttäjäystävällinen.

Työn toisen tavoitteen toteutumisen osalta voidaan todeta, että PWA soveltui projektin toteuttamiseen hyvin. Sovelluksessa käytettävät PWA:n tarjoamat ominaisuudet, joista keskeisin on push-ilmoitusten lähettäminen käyttäjän laitteeseen, mahdollistavat palvelun reaaliaikaisen käytön passiivitilassa.

Lähteet

1 TypeScript is JavaScript with syntax for types. Verkkoaineisto. TypeScript. <https://www.typescriptlang.org/>. Luettu 10.9.2024.

2 Mikä on TypeScript, ja pitäisikö sinun käyttää sitä Vanilla JS:n sijasta? Verkkoaineisto. Linux-Console.net. <https://fi.linux-console.net/?p=7088#gsc.tab=0>. Luettu 8.10.2024.

3 Run JavaScript Everywhere. Verkkoaineisto. Node.js. <https://nodejs.org/en> Luettu 15.10.2024.

4 Node.js Introduction. Verkkoaineisto. W3Schools. <https://www.w3schools.com/nodejs/nodejs_intro.asp>. Luettu 13.9.2024.

5 Developer Survey Results 2019. Verkkoaineisto. Stack Overflow. <https://survey.stackoverflow.co/2019#technology-_-other-frameworks-libraries-and-tools>. Luettu 13.9.2024.

6 Build amazing things. Verkkoaineisto. npm. <https://www.npmjs.com/>. Luettu 15.10.2024.

7 What is npm? Verkkoaineisto. W3Schools. <https://www.w3schools.com/whatis/whatis_npm.asp>. Luettu 29.9.2024.

8 Node.js web application framework. Verkkoaineisto. Express. <https://expressjs.com/>. Luettu 15.10.2024.

9 Sahu, Ankit. 2024. 30 Most Popular NPM Packages for Node JS Developers. Verkkoaineisto. Turing. <https://www.turing.com/blog/top-npm-packages-for-node-js-developers>. 20.2.2024. Luettu 9.10.2024.

10 The library for web and native user interfaces. Verkkoaineisto. React. <https://react.dev/>. Luettu 15.10.2024.

11 Bhatt, Tuhin. 2024. 8 Most Popular Front End Frameworks to Use in 2024. Verkkoaineisto. London App Development. <https://londonappdevelopment.co.uk/blog/best-front-end-frameworks>. Päivitetty 25.4.2024. Luettu 10.9.2024.

12 Build awesome apps with Google’s knowledge of the real world. Verkkoaineisto. Google. <https://developers.google.com/maps>. Luettu 15.10.2024.

13 Google Maps API. Verkkoaineisto. Google. <https://static.googleusercontent.com/media/maps.google.com/fi//help/maps/casestudies/maps-api-web.pdf>. Luettu 9.10.2024.

14 Node.js ja Express. Verkkoaineisto. Full stack open. <https://fullstackopen.com/osa3/node_js_ja_express#express>. Luettu 8.10.2024.

15 Express.js Tutorial. 2024. Verkkoaineisto. GeeksforGeeks. <https://www.geeksforgeeks.org/express-js/>. Päivitetty 9.10.2024. Luettu 15.10.2024.

16 Azure App Service. Verkkoaineisto. Microsoft. <https://azure.microsoft.com/en-us/products/app-service>. Luettu 15.10.2024.

17 App Service overview. 2024. Verkkoaineisto. Microsoft. <https://learn.microsoft.com/en-us/azure/app-service/overview>. Päivitetty 13.9.2024. Luettu 9.10.2024.

18 Azure SQL Database. Verkkoaineisto. Microsoft. <https://azure.microsoft.com/en-us/products/azure-sql/database>. Luettu 15.10.2024.

19 What is Azure SQL Database? 2024. Verkkoaineisto. Microsoft. <https://learn.microsoft.com/en-us/azure/azure-sql/database/sql-database-paas-overview?view=azuresql>. Päivitetty 12.9.2024. Luettu 10.10.2024.

20 Make your app the best it can be with Firebase and generative AI. Verkkoaineisto. Firebase. <https://firebase.google.com/>. Luettu 15.10.2024.

21 Tyndall, Shaun. 2024. GitHub Statistics & Trends. Verkkoaineisto. Inclind. <https://www.inclind.com/news/github-statistics-trends>. 20.2.2024. Luettu 8.10.2024.

22 Continuous delivery and continuous integration. Verkkoaineisto. Amazon. <https://docs.aws.amazon.com/codepipeline/latest/userguide/concepts-continuous-delivery-integration.html>. Luettu 8.10.2024.

23 JSON Web Token. 2023. Verkkoaineisto. Wikipedia. <https://fi.wikipedia.org/wiki/JSON_Web_Token>. Päivitetty 16.12.2023. Luettu 10.10.2024.

24 Progressive web application (PWA)/Progressiivinen verkkosovellus. Verkkoaineisto. ite wiki. <https://www.itewiki.fi/opas/progressive-web-application-pwa-progressiivinen-verkkosovellus/>. Luettu 10.10.2024.

25 Posnick, Jeff. 2018. Beyond SPAs - alternative architectures for your PWA. Verkkoaineisto. Chrome for Developers. <https://developer.chrome.com/blog/beyond-spa>. Päivitetty 23.5.2018. Luettu 27.9.2024.

26 Web Push Notifications Explained. Verkkoaineisto. Airship. <https://www.airship.com/resources/explainer/web-push-notifications-explained/>. Luettu 7.10.2024.

27 Hietaniemi, Jari. 2019. Scrum pähkinänkuoressa. Verkkoaineisto. Gofore. <https://gofore.com/scrum-pahkinankuoressa/>. 17.1.2019. Luettu 10.10.2024.