OULUN AMMATTIKORKEAKOULU

Antti Kleemola

VARAJÄRJESTELMÄ TAKSIEN KYYDINVÄLITYKSEEN

VARAJÄRJESTELMÄ TAKSIEN KYYDINVÄLITYKSEEN

Antti Kleemola
Opinnäytetyö
Kevät 2017
Tietotekniikan tutkinto-ohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan tutkinto-ohjelma, ohjelmistokehitys

Tekijä: Antti Kleemola
Opinnäytetyön nimi: Varajärjestelmä taksien kyydinvälitykseen.
Työn ohjaaja: Pekka Alaluukas
Työn valmistumislukukausi ja -vuosi: Kevät 2017
Sivumäärä: 39 + 5 liitettä

Opinnäytetyön aiheena oli varajärjestelmä taksien kyydinvälitykseen. Tällä hetkellä pääjärjestelmän kaatuessa takseille välitetään kyytejä käyttämällä WhatsApp-nimistä sovellusta. Työn toimeksiantajana toimi OTaksi, vanhalta nimeltään Oulun aluetaksi. OTaksin yhteyshenkilönä toimi OTaksin toimitusjohtaja Vesa Tuunainen.

Työn tavoitteena oli luoda toimiva prototyyppi varajärjestelmästä, ja ladata se pilvipalveluun. Opinnäytetyön tekijälle annettiin seuraavat vaatimukset järjestelmää koskien: järjestelmään täytyy pystyä syöttämään kyytejä ja kyydeissä täytyy näkyä asiakkaiden tiedot. Asiakkaiden tietoihin kuuluu mm. osoite, nimi, kellonaika sekä muu informaatio kuten puhelinnumero. Taksikuskien täytyy pystyä joko valitsemaan kyyti järjestelmästä tai kyyti välitetään suoraan lähimmälle kuskille. Järjestelmään täytyy kirjautua ennen käyttöä ja asiakkaiden tietojen täytyy olla turvassa. Opinnäytetyön tekijälle annettiin teknisen puolen toteutuksessa todella vapaat kädet, eikä hänellä ollut muuta tietoa kuin että OTaksin omistamat palvelimet ovat Windows-palvelimia.

Työn tuloksena saatiin luotua toimiva varajärjestelmä, mutta ilman henkilöllisyyden todentamista. Tästä syystä johtuen työtä ei laitettu pilvipalveluun ollenkaan, eikä aika siihen olisi riittänytkään. Työhön oli jo tässä vaiheessa kulunut yli 400 tuntia.

Opinnäytetyöhön tehtiin kaksi asiakassovellusta, serverirajapinta, tietokanta sekä taustapalvelu, joka päivittää tietokantaa noin minuutin välein. Yksi asiakassovellus toteutettiin Androidilla ja toinen WPF-sovelluksena. Rajapinta tehtiin käyttämällä ASP.NET Corea ja tietokanta käyttämällä MongoDB:tä. Taustapalvelu tehtiin .NET Core -konsolisovelluksena.

Asiasanat: ASP.NET Core, .Net Core, WPF, Android, MongoDB, järjestelmäsuunnittelu

ABSTRACT

Oulu University of Applied Sciences
Information technology, software development

Author(s): Antti Kleemola
Title of thesis: Taxi dispatch backup system
Supervisor(s): Pekka Alaluukas
Term and year when the thesis was submitted: Spring 2017
Number of pages: 39 + 5 appendices

The subject of this thesis was a backup system for the distribution of fares for taxis. The orderer of this thesis subject was OTaxi, a taxi company in Oulu, Finland.

The goal of this thesis was to create a working back up system in case the main system ever went down. The last time this happened the jobs got distributed by a mobile texting app called WhatsApp. The requirements for this project were that jobs can be inserted into the system and the system either finds the closest car or the drivers can pick the jobs themselves. Other requirements were that there needs to be an authentication system in place and the customer information needs to be secure. Job has three to four attributes; time the job was created, address, customer name and any extra info that needs to be delivered to the driver.

The developer was given a free reign over pretty much all the technical aspects of the project. Only thing that was known, is that OTaxi has some Windows-servers and this project could possibly be hosted there. As a result of this project a working prototype was created but without authentication services. Because of this the project cannot be deployed just yet.

The project has two clients, an API, database and a background service that calls the database every minute and 15 seconds. One of the clients was made with Android and the other as a WPF-application, meaning a desktop application for Windows devices. The API was created using ASP.NET Core. MongoDB was used as a database. Background service got created as a simple .NET core console application.

Keywords: ASP.NET Core, .NET Core, Android, WPF, MongoDB

SISÄLLYS

SANASTO

API	Rajapinta. Rajapinnan tarkoitus on tarjota käyttömahdollisuudet rajapinnan takana oleviin resursseihin. Rajapinta päättää myös sen, mitä resursseilla tehdään.
Client	Asiakassovellus tai asiakasohjelma.
Cross-platform	Järjestelmäriippumaton.
Dependency injection	Riippuvuusinjektio.
Endpoint	Yksi yhteyden loppupää rajapinnassa. Esim. www.kaleva.fi/uutiset/oulu tai www.kaleva.fi/uutiset/pohjois-suomi.
ID	Tunniste.
Service	Taustapalvelu Androidissa joka pysyy päällä.

1 JOHDANTO

Opinnäytetyön aiheena on tehdä toimiva varakyydinvälitysjärjestelmä takseille. Tällä hetkellä pääjärjestelmän kaatuessa kyydinvälitykseen ei löydy järkevää vaihtoehtoa. Vielä noin kymmenen vuotta sitten takseissa käytettiin LA-puhelimia ja pääjärjestelmän kaatuessa kyytejä pystyttiin vielä välittämään tällä varajärjestelmällä. Nykyään LA-puhelimet ovat poistuneet melkein kaikista autoista ja kyydinvälitys on ulkoistettu keskitetylle yhtiölle nimeltä Eniro Sentraali. Eniro Sentraali välittää kyytejä mm. myös Oulun takseille. Taksit ja Eniro Sentraali käyttävät palvelua nimeltä TaxiBook-ajovälitysjärjestelmä, jonka on kehittänyt yritys nimeltä Mobisoft Oy.

Nykyään varajärjestelmänä käytetään WhatsApp-nimistä mobiilisovellusta. Toisin sanoen kyydit välitetään ryhmäkeskusteluun ja kuitataan ”kuka kerkeää” -periaatteella. Oulussa takseja on töissä normaalissa päivävuorossa noin 18 ryhmää, yhdessä ryhmässä on noin 8–9 taksia. Yövuorossa toimii noin 6 ryhmää. Oulussa on yhteensä noin 230 taksia tämän opinnäytetyön kirjoittamisen hetkellä. Kaikki taksit ovat töissä yhtä aikaa ainoastaan erikoistilanteissa. Esimerkkinä erikoistilanteesta voidaan käyttää esimerkiksi Qstock-musiikkifestivaalia.

Tavoitteena tässä opinnäytetyössä on kehittää mahdollisimman valmis prototyyppi varajärjestelmäksi ja laittaa se pilvipalveluun, kuten Amazon Web Services. Järjestelmän täytyy toimia ainakin Windows-palvelimella. Toisin sanoen opinnäytetyön tekijällä oli todella paljon vapauksia työtä suunnitellessa.

2 VAATIMUSMÄÄRITTELY

Projektin lähtötietomuistion mukaan järjestelmän täytyy pystyä seuraaviin asioihin:

- Järjestelmään syötetään kyytejä.
- Kyydeissä näkyy asiakkaiden tiedot kuten
    - tilauksen osoite
    - asiakkaan nimi
    - tilauksen aika
    - mahdolliset lisätiedot kuten puhelinnumero.
- Taksikuskit pystyvät valitsemaan kyydit, tai kyydit välitetään suoraan lähimmälle taksikuskille.
- Asiakkaiden tiedot ovat turvassa.
- Järjestelmään kirjaudutaan ensin sisälle.

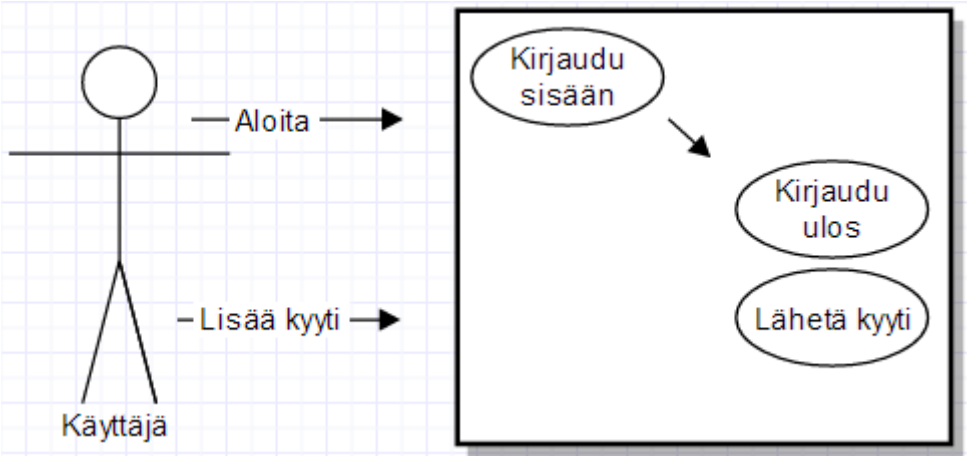
Työn tavoitteena oli saada toimiva prototyyppi 6.6.2017 mennessä.

3 SUUNNITTELU

Ennen kuin järjestelmä voidaan tehdä, täytyy se ensin suunnitella. Järjestelmä voidaan jakaa pääpiirteissään viiteen osaan, jotka ovat kaksi asiakassovellusta (engl. client), palvelimen rajapinta, tietokanta sekä taustalla pyörivä palvelu tai prosessi, joka tarkistaa tietokannan tietyin väliajoin. Tämän lisäksi tarvitaan jonkinlainen käyttäjän todentamispalvelu, jotta järjestelmä ei ole avoin kenelle tahansa.

3.1 Kyydinvälitys

Kyydinvälityssovelluksen ainoa tehtävä on tuottaa sisältöä järjestelmään. Kyydinvälityssovellus tekee käyttäjän näkökulmasta siis kaksi asiaa: todentaa käyttäjän henkilöllisyyden ja antaa hänen tämän jälkeen syöttää kyytejä järjestelmään. Kuvassa 1 näkyy kyydinvälitys käyttäjän näkökulmasta.

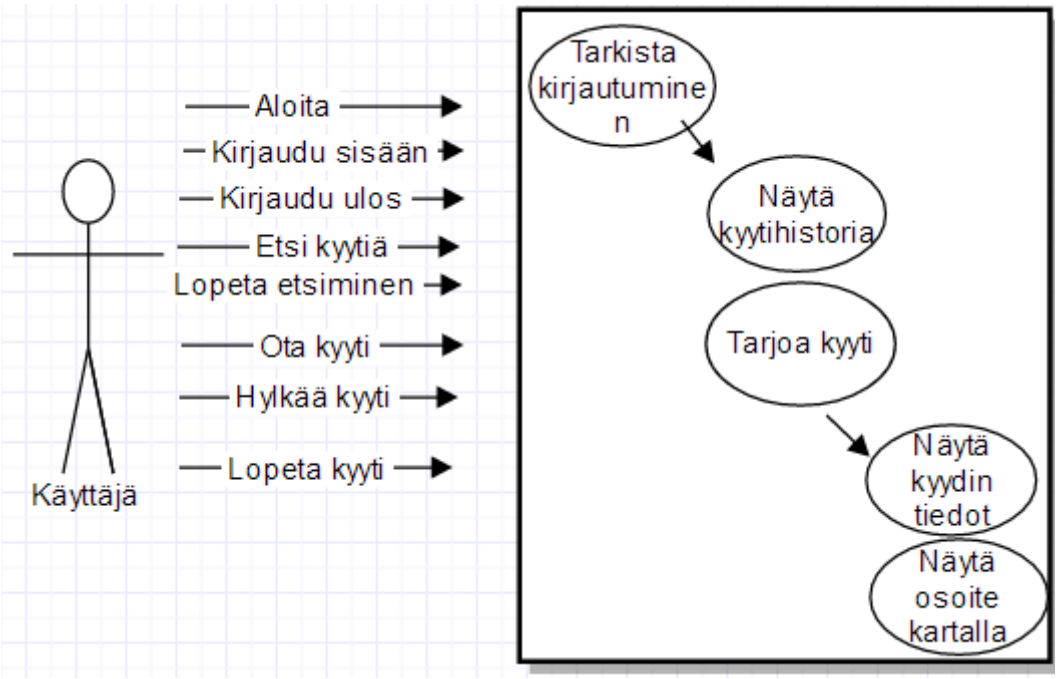


KUVA 1. Kyydinvälitys käyttäjän näkökulmasta

Kun käyttäjä lisää kyydin järjestelmään, vaaditaan kyydille ainakin kolme pakollista attribuuttia. Nämä attribuutit ovat osoite, kaupunki sekä asiakkaan nimi. Lisäksi kyytiin voidaan lisätä lisätietoja tarvittaessa. Ennen kuin kyyti lähetetään rajapinnalle ja sitä kautta tietokantaan, haetaan kyydin osoitteelle vielä koordinaatit, jotta kyyti voidaan välittää lähimmälle autolle. Osoite tulee näyttää myös kartalla. Jos osoitteelle ei löydy koordinaatteja, lisätään koordinaatteihin Oulun kaupungin keskusta. Kaupunkia ei lähetetä eteenpäin, vaan se tarvitaan ainoastaan koordinaattien hakemiseen.

3.2 Kyydin kuittaaminen

Järjestelmän toinen asiakassovellus on jollakin mobiilikäyttöjärjestelmällä toimiva sovellus. Toisen asiakassovelluksen täytyy pystyä myös todentamaan käyttäjän henkilöllisyys. Tämän lisäksi asiakassovelluksen täytyy pystyä etsimään sekä tarjoamaan kyyti käyttäjälle, jos sellainen on saatavilla. Sovelluksen käyttöliittymässä käyttäjän täytyy nähdä kyydin tiedot, kyydin paikka kartalla sekä loki tehdyistä kyydeistä. Loki pitää pystyä tyhjentämään tarvittaessa. Kuvassa 2 on kyydin kuittaamisen kaikki vaatimukset käyttäjän näkökulmasta.

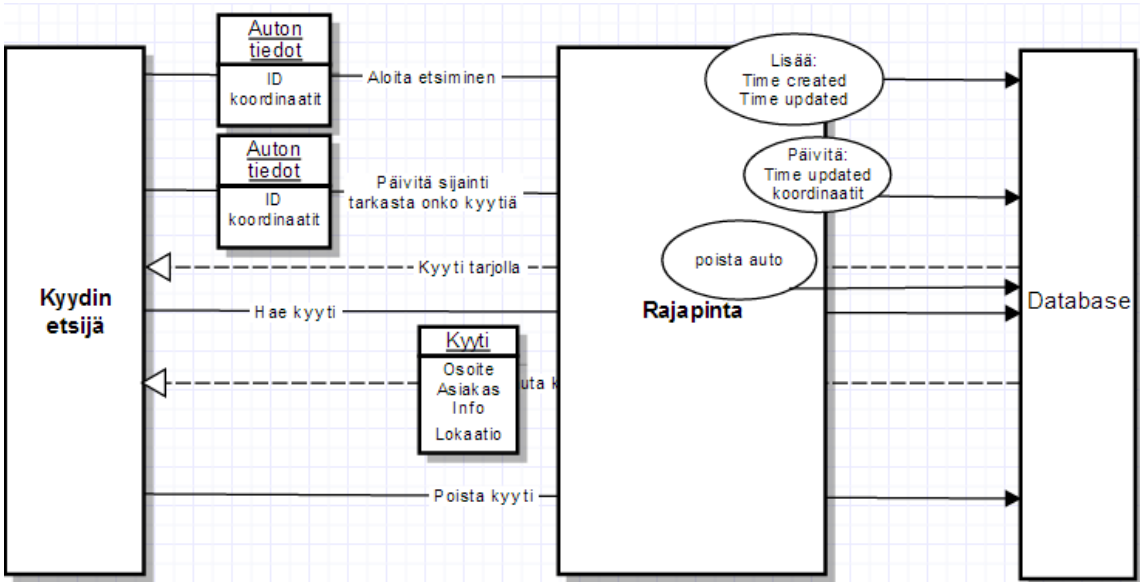


KUVA 2. Kyydin kuittaaminen käyttäjän näkökulmasta

Kun käyttäjä on todennettu ja hän päättää ruveta etsimään kyytiä tai asiakasta, täytyy auto ensin rekisteröidä rajapinnan kautta tietokantaan. Tämän jälkeen auto päivittää minuutin väliajoin sijaintinsa ja kysyy samalla töitä. Jos kysely palauttaa positiivisen vastauksen, kysyy sovellus käyttäjältä, onko hän valmis vastaanottamaan kyydin. Käyttäjälle annetaan 15 sekuntia aikaa vastata. Jos käyttäjä haluaa kyydin, kyyti haetaan tietokannasta. Kun kyyti on vastaanotettu ja todennettu, asiakassovellus lähettää tiedon rajapinnalle, että kyyti on turvallista poistaa. Jos käyttäjä kieltäytyy kyydistä tai ei vastaa 15 sekunnin kuluessa kyytitarjouksen saapumisesta, kyydin etsintä lopetetaan. Kyydin etsijä lähettää siis aina rekisteröidessään ja sijaintiansa päivittäessään yksilöllisen tunnisteen (ID) sekä oman sijaintinsa. Vaihtoehtoisesti jos kyydin etsintä lopetetaan, lähetetään rajapinnalle ilmoitus siitä, että auto täytyy poistaa vapaiden autojen listalta.

3.3 Rajapinta

Asiakassovellukset haluavat tietoa tietokannasta. Jotta järjestelmässä voidaan määritellä, mitä tietoa annetaan ja kenelle, tarvitaan rajapinta. Rajapinta toimii poliisina tietokannan ja asiakassovellusten välissä. Jos järjestelmä tehdään oikein, rajapinta ei päästä tuntemattomia pyyntöjä palveluun käsiksi. Rajapinnan tehtävä on joka ikisen pyynnön saatuaan tarkastaa, onko pyytäjällä valtuuksia tehdä järjestelmässä mitään. Rajapinta myös määrää, mitä palveluun annetulla tiedolla tehdään. Aina kun asiakassovellus tai kyydin etsijä rekisteröi itsensä palveluun, lisätään kyydin etsijän tietoihin aika, milloin etsijä on rekisteröitynyt ja milloin auto on viimeksi päivittänyt tietonsa. Näitä aikatietoja tarvitaan, jotta tietokanta voidaan pitää puhtaana. Tietokantaa on hankala ylläpitää, jos lisätyissä dokumenteissa ei ole lisäyksen tai muokkauksen ajankohtaa. Jos auto esimerkiksi menettää jostakin syystä yhteytensä, ilman ajan lisäystä järjestelmällä ei ole mitään tapaa poistaa turhia dokumentteja tietokannasta. Järjestelmän toimintaperiaate näkyy kuvassa 3. Samat aikakentät lisätään myös kyydeille. Kyseiset kentät tarvitaan, jotta kyyti voidaan tarvittaessa siirtää seuraavalle autolle. Ennen tietokantaan lisäämistä kyydin tietoihin lisätään vielä yksi kenttä, jossa säilytetään lähimmän auton indeksi. Jos autoja ei ole vapaana, jätetään kentän arvo tyhjäksi.



KUVA 3. Järjestelmän yleiskuvaus kyytiä etsiessä

3.4 Taustaprosessi

Taustaprosessi lisätään palvelimen palveluihin, jotka käynnistetään aina, kun palvelin käynnistyy. Kyytejä etsivät asiakassovellukset kyselevät joka minuutti rajapinnalta, onko asiakassovelluksen

tunnisteella varustettua kyytiä tietokannassa. Jos kyyti löytyy, annetaan käyttäjälle vielä 15 sekuntia aikaa vastata kyytitarjoukseen. Taustaprosessille asetetaan ajastin, joka ajaa sovelluksen aina, kun minuutti ja 15 sekuntia on kulunut. Taustaprosessi hakee tietokannasta kaikki kyydit, joita ei ole päivitetty minuuttiin ja 15 sekuntiin, ja etsii kyydeille uuden tekijän.

Täytyy muistaa, että kyydin hakijat ja taustaprosessi eivät ikinä ole synkronoituja, vaan tapahtuvat aina eri aikoihin riippuen siitä, milloin kyytiä on ruvettu hakemaan tai milloin pääjärjestelmä on käynnistetty. Taustaprosessin ajastimen aikaa täytyy luultavasti vielä nostaa, jotta esimerkiksi verkkovirheet voidaan ottaa huomioon. Teoriassa suurin viive kyydin syöttämisestä ja kyydin joutumisesta taksikuskille on kaksi minuuttia ja 30 sekuntia. Suurimmassa osassa tapauksista tämä ei kuitenkaan tapahdu olettaen, että ensimmäinen auto kuittaa heti kyydin. Näin saadaan välitettyä kyydit seuraaville autoille, jos ensimmäinen on kieltäytynyt kyydistä. Autot päivittävät itseään minuutin välein, joten sovelluksessa voidaan määritellä, että autot poistetaan, jos päivitetty aika ei ole muuttunut esimerkiksi yli kahteen minuuttiin.

3.5 Tietokanta

Tietokanta tulee olemaan tässä projektissa dokumenttitietokanta. Tietokantaan lisätään kaksi kokoelmaa: yksi autoille sekä töille. Dokumenttitietokannassa on se hyvä puoli, että eri kokoelmien välille ei tarvitse miettiä yhteyksiä niin kuin relaatiotietokannassa.

3.6 Todentamispalvelu

Koska kyseessä ei ole julkinen rajapinta, tarvitaan palvelu, missä asiakassovellusten käyttäjien henkilöllisyys voidaan ensin todentaa. Asiakassovelluksen täytyy siis todentaa käyttäjä, jos todennus on jo vanhentunut tai jos rajapinta ilmoittaa, että todennus täytyy tehdä uudestaan. Koska todentaminen on erittäin tärkeä asia missä tahansa projektissa, jossa käsitellään ihmisten tietoja, kannattaa palvelu hankkia joltakin kolmannelta osapuolelta. Todentaminen on erittäin hankala tehdä oikein. Tästä syystä on olemassa yrityksiä, jotka erikoistuvat pelkästään kyseiseen asiaan.

4 TOTEUTUKSESSA KÄYTETTÄVÄT TEKNOLOGIAT

Koska työssä annettiin työn toteuttajalle suhteellisen vapaat kädet, täytyi erilaisten toteutustapojen tutkimiseen käyttää runsaasti aikaa. Alla on listattu lyhyesti, mitä teknologioita opinnäytetyössä päädyttiin käyttämään ja miksi.

4.1 Kyydinvälitys

Kyydinvälitys toteutettiin normaalina WPF (Windows Presentation Foundation) -sovelluksena. WPF-sovellus on natiivi Windows-sovellus, eli sovellus toimii kaikilla koneilla, jotka käyttävät Windows-käyttöjärjestelmää. Kyydin lisääminen on mekaaninen ja toistuva prosessi, jossa olisi hyvä pystyä käyttämään hyödyksi hiirtä ja näppäimistöä. Lisäksi WPF-sovellusten toteuttaminen on suhteellisen helppoa ja nopeaa. Projektille varattu aika oli myös rajallinen.

4.2 Kyydin kuittaaminen

Kyydin kuittaus -asiakassovellus päätettiin toteuttaa Androidilla. Android-puhelin löytyy melkein kaikista Oulun takseista. Syynä tähän on se, että vielä noin vuosi sitten Oulussa käytettiin Taxify-nimistä sovellusta. Taxify tuki puhelinta nimeltä Samsung Galaxy S3, joka on Android-puhelin.

Tämän lisäksi Androidille on suosionsa takia iso määrä ohjeita ja julkisia kirjastoja. Myös Googlen omat dokumentoinnit kyseisestä sovellusalustasta ovat todella hyvät.

4.3 Rajapinta

Rajapinta toteutettiin käyttämällä ASP.NET Corea. ASP.NET Core 1.0 julkaistiin kesäkuussa 2016. ASP.NET Core 2.0 tullaan julkaisemaan vuoden 2017 vuosikolmanneksen puolella. ASP.NET Core on siis suhteellisen uusi. (1.) ASP.NET Core on järjestelmäriippumaton (engl. cross-platform), mikä tarkoittaa sitä, että kyseisellä teknologialla toteutettu rajapinta voidaan julkaista niin Windows- kuin Linux-palvelimelle. ASP.NET Coressa moduulit on liitetty toisiinsa löyhästi (engl. loose coupling). Karkeasti sanottuna löyhä liitos ohjelmoinnissa tarkoittaa sitä, että sovelluksessa toimivat eri mo-

duulit eivät ole riippuvaisia toisistaan. Tämä johtaa ainakin ASP.NET Coressa todella hyvään skaalautuvuuteen ja siihen, että sovelluksen eri toimintoja voidaan korvata helposti rikkomatta koko palvelua. (2.) Tästä syystä Core on ihanteellinen esimerkiksi pilvipalveluissa. ASP.NET Corella tehdyn palvelun saa myös Amazon Web Servicessä käytettävän AWS Lambdan päälle. AWS Lambda on niin sanottu ”palvelimeton palvelu”. Lyhyesti sanottuna AWS Lambdassa tarjottu palvelu ei maksa palveluntarjoajalle euroja per tunti, vaan se maksaa tasan sen verran kuin sitä on käytetty. AWS Lambdassa tarjoiltua palvelua ei tarvitse ylläpitää ja se skaalaa itsensä ylös tai alas tarvittaessa. (3.) Yllämainituista syistä ASP.NET Core on tutustumisen arvoinen ohjelmistokomponenttikirjasto.

4.4 Taustaprosessi

Taustaprosessi toteutetaan .NET Core -konsolisovelluksena. Taustaprosessilta ei vaadita mitään muuta, kuin että se lähettää verkkopyynnön aina minuutin ja 15 sekunnin välein rajapintaan.

4.5 Tietokanta

Tietokanta toteutetaan käyttäen MongoDB-nimistä dokumenttitietokantaa. MongoDB, kuten ASP.net Core:lla tehty sovelluskin, skaalaa todella hyvin. MongoDB tukee yli 100 000 luku- ja kirjoituspyyntöä sekunnissa. Määrä on paljon enemmän kuin mitä tämä projekti tulee koskaan tarvitsemaan (4). MongoDB:n C#-ajuri myös huolehtii itse yhteyksistä tietokantaan, joten ohjelmoijalla ei tarvitse niistä välittää (13). Paras asia MongoDB:ssä on kuitenkin se, että tietokanta tukee dokumenttien etsimistä koordinaattien perusteella (5).

4.6 Todentamispalvelu

Järjestelmän todentamispalveluna voidaan käyttää esimerkiksi Auth0-palvelua. Auth0 on palvelu, joka tarjoaa monta eri tapaa todentaa kehitettävän järjestelmän asiakassovellus. Auth0 antaa kehittäjälle oikeuden määritellä rajapintansa erillisenä asiakassovelluksena palvelussa (14). Tämän palvelun avulla voidaan myös määritellä taustaprosessille oma käyttötunnus järjestelmässä (15).

Auth0 on siis palvelu, jossa voidaan käyttää mm. OAuth 2.0 -protokollaa. OAuth 2.0 on protokolla jota käytetään lupien tai valtuuksien antamiseen. Se antaa oikeuden sovellus A:lle käyttää sovellus

B:n resursseja. Esimerksi projektin Android-asiakassovellus haluaa luvan käyttää palvelimelta löytyvän rajapinnan kolmea eri endpointtia. Endpoint on yksi yhteyden loppupää rajapinnassa.

OpenID Connect rakentuu OAuth 2.0:n päälle. Se antaa käyttäjälle mahdollisuuden tunnistaa itsensä, ilman että käyttäjä paljastaa omia kirjautumistietojaan. Käyttäjä ei siis käytä omia tietojaan joka kerta, kun hän ottaa yhteyden rajapintaan, vaan ainoastaan kun hän kirjautuu palveluun.

Esimerkkinä voidaan käyttää sovellusta, joka antaa käyttäjän kirjautua sisään Google-tilille ja puskea tehtäviä asioita puhelimen kalenteriin. Tässä tapauksesssa se osa, jossa käyttäjä yksilöityy palvelulle, on tehty OpenID Connectilla. Se osa, mikä antaa sovelluksen muokata kalenterilisäyksiä, on toteutettu OAuth 2.0:lla. Toisin sanoen OpenID vastaa kysymykseen ”kuka jokin on”. OAuth 2.0 vastaa kysymykseen ”mitä he saavat tehdä”. (6.)

5 TOTEUTUS

Tässä luvussa käydään läpi, miten projektin eri osa-alueet on toteutettu. Projektin aikataulun takia todentamispalvelua ei ehditty implementoida ollenkaan. Tämä tarkoittaa sitä, että taustaprosessista, niin kuin kahdesta muustakin asiakassovelluksesta, puuttuu käyttäjän todentaminen. Tästä syystä yksilöivänä tunnisteena on käytetty vain samaa tekstijonoa testausvaiheessa.

5.1 Toteutusvaiheen työkalut

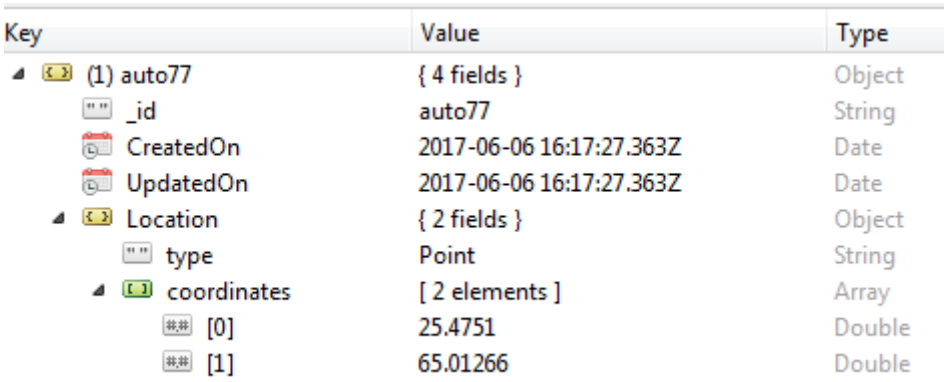
Toteutusvaiheessa käytettiin erinäisiä työkaluja toteutuksen auttamisessa sekä nopeuttamisessa. Ennen toteutuksen läpikäymistä on syytä esitellä kaikki projektissa käytetyt ohjelmat.

5.1.1 Kehitysympäristöt

Mobiilisovellus kyydin kuittaukseen tehtiin Android Studiolla. Kaikki muut projektin osa-alueet tietokantaa lukuun ottamatta tehtiin Visual Studio 2017:llä.

5.1.2 Robomongo

Robomongo on ilmainen, avoimen lähdekoodin graafinen käyttöliittymä MongoDB:n hallitsemiseen. Robomongon voi ladata suoraan kehittäjän sivuilta. Robomongolla voi mm. luoda tietokantaan kokoelmia ja JavaScript-funktioita sekä tarkastella tietokannassa olevia dokumentteja. Graafinen käyttöliittymä auttaa huomattavasti kehitystyössä (kuva 4).



KUVA 4. Robomongon graafinen käyttöliittymä

5.1.3 Node.js

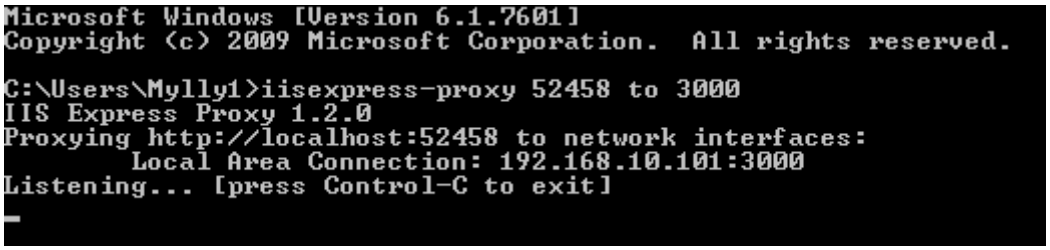
Node.js on avoimen lähdekoodin järjestelmäriippumaton JavaScript Runtime Enviroment. Node.js tarvitaan, jotta voidaan esimerkiksi käynnistää iisexpress-proxyn Windows-komentorivillä.

5.1.4 Iisexpress-proxy

IIS (Internet Information Services) on Microsoftin kehittämä palvelinohjelmistokokonaisuus. Lyhyesti sanottuna IIS ohjaa palvelimelle tulevat kutsut alaspäin niille palveluille, joille kutsut on tarkoitettu. IIS Express on kevyt, omavarainen versio IIS:stä ja se on myös tarkoitettu ohjelmistokehitykseen. IIS Express ei perusasetuksia käyttäessä hyväksy kutsuja ulkoisilta laitteilta. Ulkoisiksi laitteiksi tässä tapauksessa lasketaan esimerkiksi Android Studion emulaattori tai lähiverkossa sijaitseva puhelin. Tämä voidaan kiertää esimerkiksi käyttämällä näppärää työkalua nimeltä iisexpress-proxy(7). Jos Node.js on asennettu Windows-tietokoneelle, iisexpress-proxy asentuu kirjoittamalla ”npm install –g iisexpress-proxy” Windowsin komentoriville.
Tämän jälkeen iisexpress-proxy voidaan käynnistää kirjoittamalla

- iisexpress-proxy localport to proxyport

Kuvassa 5 nähtävä ”Localport” on portti, josta IIS Expressin alaisena pyörivä rajapinta kuuntelee pyyntöjä. ”Proxyport” on portti mihin liikenteen halutaan ohjautuvan. Työkalun on tehnyt GitHubin käyttäjä nimeltä icflorescu. Iisexpress-proxy on myös lisensoitu ISC-lisenssin alla. Lisenssin alla lisensoituja tuotteita voi käyttää ilmaiseksi ja lähdekoodi on avoin(8).



KUVA 5. Iisexpress-proxyn käyttö komentorivillä.

5.1.5 Postman

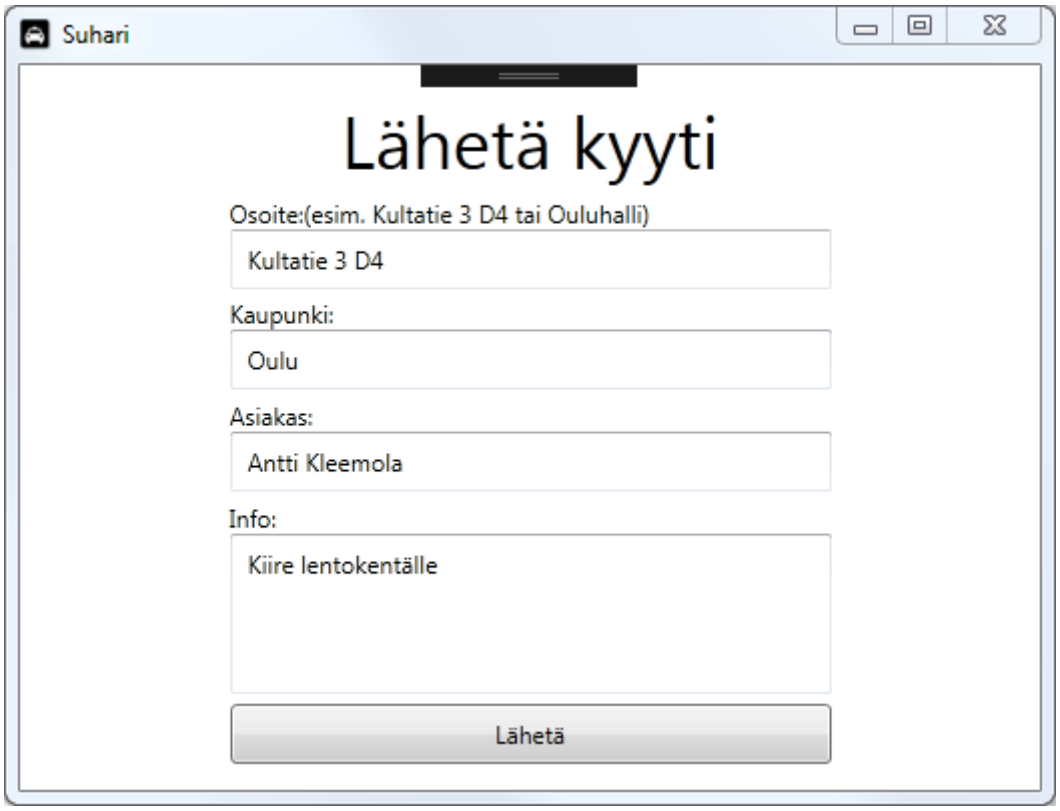
Postman on Googlen omistama graafinen käyttöliittymä, jolla pystyy mm. tekemään HTTP-kutsuja rajapintaan. Työkalu on erittäin kätevä rajapintoja kehitettäessä(9) ja nopeuttaa kehitettävän rajapinnan testausta huomattavasti.

5.2 Kyydinvälitys

Kyydinvälityssovellus toteutettiin WPF-sovelluksena (Windows Presentation Foundation). Toisin sanoen sovellus on natiivi Windows-sovellus, joka käynnistyy kaikilla tietokoneilla, jotka kyseistä käyttöjärjestelmää käyttävät. Kyydinvälityssovelluksesta tullaan käyttämään tästä lähtien nimeä Suhari Desktop.

5.2.1 Käyttöliittymä

Sovelluksessa voidaan lisätä kyyti järjestelmään painamalla Lähetä-painiketta. Sovellus tarkastaa että kentät Osoite, Kaupunki sekä Asiakas ovat täytettyinä, ennen kuin kyydin annetaan lähteä liikkeelle. Sovelluksen käyttöliittymä näkyy kuvassa 6. Kyydin lähetyksen jälkeen käyttäjälle näytetään sanoma, josta käy ilmi, onnistuiko kyydin lähetys vai ei (kuva 7).



KUVA 6. Suhari Desktop -sovellus.

KUVA 7. Suhari Desktop -palautelaatikko.

5.2.2 Ohjelmointi

Kun Lähetä-painiketta on painettu, kerätään tiedot kentistä ja lähetetään ne metodille nimeltä Send-NewJob. SendNewJob sijaitsee luokassa nimeltä ApiOperations. ApiOperations-luokassa on kaksi metodia: SendNewJob ja GetLatLng. SendNewJob-metodi kutsuu ensimmäisenä GetLatLng-metodia. GetLatLng nimensä mukaisesti hakee osoitteelle koordinaatit(kuva 8).

```
private Coordinates GetLatLng(String address, String city)
{
    try
    {
        //TODO remember to implement map
        AddressData data = new AddressData();
        data.Address = address;
        data.City = city;
        data.Country = "Finland";
        var locationService = new GoogleLocationService();
        var point = locationService.GetLatLongFromAddress(data);

        Coordinates cor = new Coordinates(point.Latitude,
            point.Longitude);
        return cor;
    }
    catch(Exception)
    {
        return null;
    }

}
```

KUVA 8. GetLatLng-luokka

Coordinates-luokassa käytetään kirjastoa nimeltä GoogleLocationService. Kirjasto löytyy mm. NuGet-pakettina Visual Studion kautta. Kirjastoa apuna käyttäen sovellus tekee yksinkertaisen HTTP-pyynnön Google Mapsin geocode-rajapintaan. Rajapintaa voi kutsua esimerkiksi kirjoittamalla selaimeen

- https://maps.googleapis.com/maps/api/geocode/json?address=Kultatie,+Oulu,+Finland&sensor=true

Pyyntö palauttaa osoitteen tiedot JSON-tiedostomuodossa, joista käy ilmi mm. osoitteen koordinaatit.

Kun koordinaatit on haettu, ne palautetaan takaisin SendNewJob-metodille, joka tarkastaa, että koordinaatit eivät ole tyhjät. Jos koordinaatit ovat tyhjät, osoitteelle annetaan Oulun keskustan koordinaatit. Näin varmistetaan, että kyydillä on edes jokin sijainti ja että kyyti varmasti tulee hoidettua. Tämän jälkeen kyyti pakataan JSON-tiedostomuotoon ja lähetetään rajapinnan kautta tietokantaan(kuva 9).

```
public String SendNewJob(String address, String city, String customer,
    String info)
{
    Coordinates cords = GetLatLng(address, city);
    if(cords == null)
    {
        cords = new Coordinates(ouluLat, ouluLng);
    }

    string endPoint = this.baseURL + "/postjob";
    string method = "POST";
    string json = JsonConvert.SerializeObject(new
    {
        Address = address,
        Customer = customer,
        Info = info,
        lng = cords.Lng,
        lat = cords.Lat
    });
    WebClient wc = new WebClient();
    wc.Headers["Content-Type"] = "application/json";
    //TODO add headers
    try
    {
        string response = wc.UploadString(endPoint, method, json);
        return "kyyti välitetty";
    }
    catch(Exception e)
    {
        return e.Message;
    }
}
```

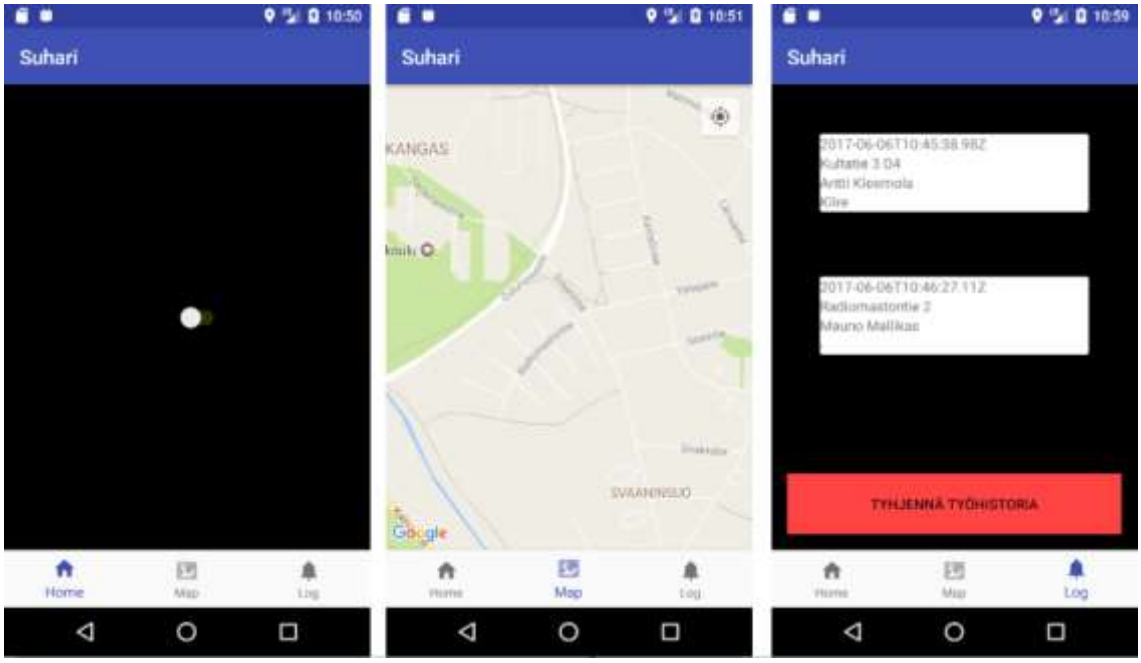
KUVA 9. SendNewJob-luokka

5.3 Kyydin kuittaaminen

Kyydin kuittaus -asiakassovellus toteutettiin Androidilla, ja se tunnetaan tästä lähtien nimellä Suhari. Suharissa on kaksi activitya sekä neljä fragmenttia. Androidissa activity on jokin toiminto, minkä käyttäjä voi toteuttaa. Fragment edustaa jotakin toimintoa käyttöliittymässä. Tässä sovelluksessa fragmenteissa on kaikki, mikä näkyy lopulta käyttäjälle. Ensimmäinen toiminto on nimeltään MainActivity. MainActivity vastaa neljästä fragmentista ja niiden oikeaoppisesta näyttämisestä. Toinen toiminto on nimeltään RideRequestActivity. Tämän lisäksi sovelluksesta löytyy vielä kaksi service-luokkaa. Service-luokat pysyvät päällä ja toimivat taustalla, vaikka näyttö sammutettaisiin.

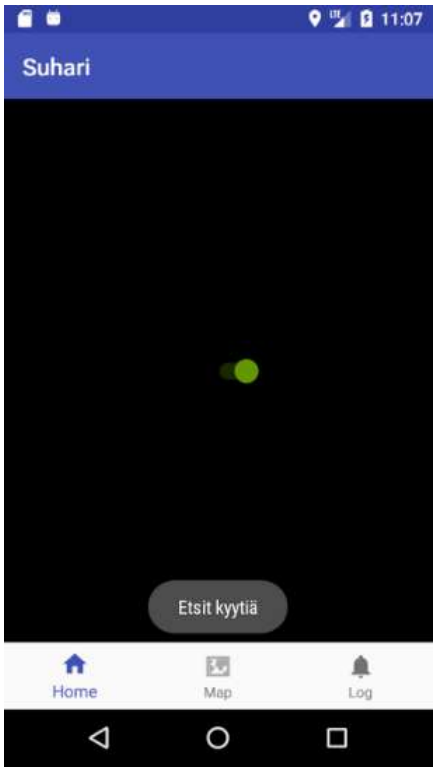
5.3.1 Käyttöliittymä

Käyttöliittymässä on siis kolme sivua; Home, Map ja Log. Kolmannella sivulla eli Logissa on kaikki tällä laitteella ajetut kyydit. Kyydit voi poistaa painiketta painamalla ja ne pysyvät puhelimen muistissa, vaikka sovellus suljettaisiin. Keskimmäisestä sivusta eli Mapista löytyy kartta. Kartta on Googlen oma kartta ja sitä voi liikutella tai tarkentaa. Kartta käyttää hyväksi Googlen Android-laitteille tarkoitettua Google Maps Android -rajapintaa. Sovellus on rekisteröity Googlen rajapintaan ja Google on luovuttanut sovellusta varten rajapinta-avaimen, joka on lisätty sovellukseen. Oikeassa yläkulmassa olevasta napista kartta liikuttaa itsensä aina Oulun kaupungin keskustan ylle. Ensimmäisessä ruudussa ei ole mitään muuta kuin kytkin, jota painamalla kyydin etsiminen alkaa tai loppuu. Kyydin käyttöliittymä kyytiä etsiessä on näytetty kokonaisuudessaan kuvassa 10.



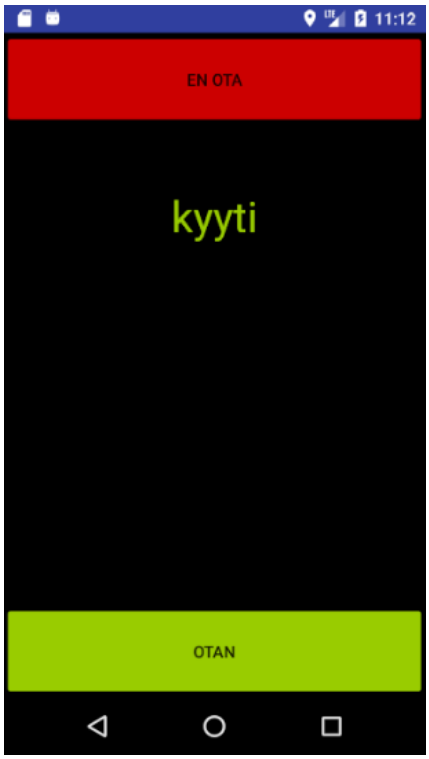
KUVA 10. Vapaa-tilan käyttöliittymä

Jos Home-sivun kytkimen asettaa hakuasentoon, laite rekisteröi itsensä palveluun ja kyselee aina minuutin välein rajapinnasta, onko laitteelle kyytiä (kuva 11).



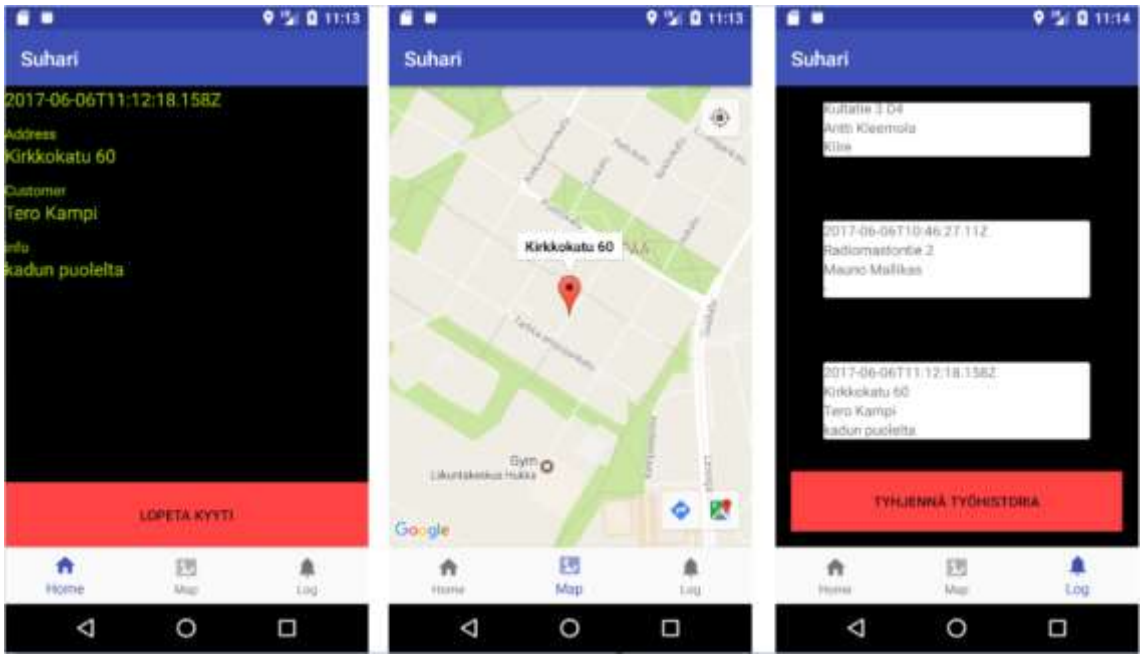
KUVA 11. Kyydin etsiminen

Kun järjestelmä ilmoittaa laitteelle, että kyyti odottaa hakijaansa, Suhari käynnistää RideRequestActivityn ja soittaa puhelimen hälytysääntä. Tätä kestää niin kauan, kunnes käyttäjä kuittaa kyydin toisella napeista tai kunnes 15 sekuntia on kulunut. RideRequestActivityn käyttöliittymä näkyy kuvassa 12.



KUVA 12. Kyytitarjous

Jos käyttäjä päättää hyväksyä kyydin, sovellus siirtyy varattuun tilaan. Käytännössä tämä tarkoittaa sitä, että uusia kyytejä ei voi etsiä, ennen kuin nykyinen on lopetettu Home-sivun painikkeesta. Home-sivulla ilmestyy kytkimen tilalle kyydin tiedot. Map-sivu liikkuu kyydin osoitteen kohdalle kartassa. Map-sivulle ilmestyy myös kyydin hyväksyessä merkki, jota painamalla kartan oikeaan alareunaan ilmestyy kaksi painiketta. Kummastakin napista aukeaa Laitteelle asennettu Google Maps -sovellus. Erona on se, että vasen painike antaa myös kartalla olevan merkin koordinaatit mukanaan. Käyttäjä voi siis näillä tiedoilla navigoida suoraan kohteeseen. Jos kyyti hyväksytään, se lisätään aina heti Logiin. Varattu-tilan käyttöliittymä näkyy kokonaisuudessaan kuvassa 13.



KUVA 13. Varattu-tilan käyttöliittymä

5.3.2 Ulkoiset kirjastot

Suharissa käytettiin seuraavia kirjastoja:

- Retrofit2
- Sugar ORM
- EventBus.

Retrofit2 on vapaasti käytettävä kirjasto, joka on tehty rajapintakutsuja varten. SugarORM tekee puhelimen sisäiseen SQLite-tietokantaan kirjoittamisesta helpompaa. EventBus auttaa activityjen sekä fragmenttien välisessä kommunikoimisessa. Kaikki yllämainitut kirjastot ovat vapaasti käytettäviä ja ilmaisia kirjastoja.

5.3.3 Ohjelmointi

Suharissa on kaksi palvelua (engl. service), jotka pysyvät päällä, vaikka käyttäjä ei aktiivisesti käytä puhelinta. Ensimmäinen on GpsService (liite 5), joka Suharin käynnistyessä laitetaan päälle. GpsService juoksee niin kauan taustalla, kunnes käyttäjä poistaa Suharin Androidin omasta sovelluslistasta. GpsService tarkistaa aina 30 sekunnin välein, onko puhelimen sijainti muuttunut. Jos si-

jainti on muuttunut, sijainti lisätään Androidin ”Shared preferenceihin”. Shared preferences on tallennustila Androidissa, jonne sovellukset voivat tallentaa avain-arvopareja. Tässä tapauksessa tallennustilaan tallennetaan leveys (engl. latitude) ja pituus (engl. longitude) long-muuttujina. Koordinaatit tallennetaan long-muuttujina sen takia, että muuttujat eivät menetä missään vaiheessa tarkkuuttaan.

Suharin toinen palvelu on BackGroundConnectionService (liite 1). Kun käyttäjä aloittaa kyydin etsimisen painamalla Home-sivulla olevaa kytkintä, kutsutaan StartHttpService-metodia(kuva 14). Androidissa intentit sisältävät käskyjä tai muuttujia toisille activityille. PendingIntent taas on tapa antaa toiselle sovellukselle sitä kutsuvan sovelluksen oikeudet. StartHttpService-metodissa tapahtuvat seuraavat asiat: Ensin pendingIntent-muuttuja alustetaan, jos se on tyhjä. Toisin sanoen siihen lisätään intent, jossa on käsky käynnistää BackGroundConnectionService-luokka. Tämän jälkeen luodaan yhteys rajapintaan ja rekisteröidään auto palveluun. Jos rekisteröinti tapahtui onnistuneesti, BackGroundConnectionService käynnistetään käyttämällä AlarmManageria. PendingIntent toimii intentille kääreenä, joka auttaa AlarmManageria ymmärtämään sitä ja jossa samalla annetaan AlarmManagerille oikeudet käynnistää BackGroundConnectionService joka minuutti.

```
public void StartHttpService()
{
    if(pendingIntent == null)
    {
        intent = new Intent(getApplicationContext(), BackgroundConnectionService.class);
        intent.addCategory(BackgroundConnectionService.TAG);
        pendingIntent = PendingIntent.getService(getApplicationContext(), 12345, intent,
                PendingIntent.FLAG_UPDATE_CURRENT);
    }
    SharedPreferences prefs = getSharedPreferences("com.example.mylly1.suhari", Context.MODE_PRIVATE);

    CarRepo rp = new CarRepo();
    rp.setId("auto77");
    rp.setLon(Double.longBitsToDouble(prefs.getLong("lon", Double.doubleToLongBits(25.474275))));
    rp.setLat(Double.longBitsToDouble(prefs.getLong("lat", Double.doubleToLongBits(65.012737) )));

    //rp.setLoc(loc);
    restService.postCar(rp).enqueue(new Callback<Void>() {
        @Override
        public void onResponse(Call<Void> call, Response<Void> response) {
            if(response.isSuccessful())
            {
                Log.i("vastaus", response.message());
                am = (AlarmManager)getSystemService(ALARM_SERVICE);
                am.setRepeating(AlarmManager.ELAPSED_REALTIME, SystemClock.elapsedRealtime(), 3000,
                        pendingIntent);
                MakeToast("Etsit kyytiä", Toast.LENGTH_SHORT);
            }
            else{
                MakeToast("Virhe: " + response.code(), Toast.LENGTH_LONG );
                WaitingforJob frg = (WaitingforJob) getSupportFragmentManager().findFragmentByTag("WAIT");
                frg.jobSwitch.setChecked(false);
            }
        }

        @Override
        public void onFailure(Call<Void> call, Throwable throwable) {
            if(throwable.getMessage() != null)

            Log.w("vastauss", throwable.getMessage());
            MakeToast("Virhe:" + throwable.getMessage(), Toast.LENGTH_LONG);
            WaitingforJob frg = (WaitingforJob) getSupportFragmentManager().findFragmentByTag("WAIT");
            frg.jobSwitch.setChecked(false);
        }
    });

}
```

KUVA 14. Suhari Android, StartHttpService()-metodi

BackGroundConnectionService kyselee kyytiä minuutin välein rajapinnalta ja positiivisen vastauksen saatuaan käynnistää RideRequestActivityn. RideRequestActivityn käyttöliittymä on näytetty kuvassa 12. Vastauksen saatuaan RideRequestActivity kutsuu MainActivityä, jossa vastaus tarkistetaan. Jos käyttäjä on hyväksynyt kyydin, haetaan se rajapinnasta ja jaetaan sovelluksessa oleville fragmenteille.

5.3.4 Retrofit2

Suharissa käytetään Retrofit2-kirjastoa apuna HTTP-kutsujen tekemiseen. MainActivityn alussa kutsutaan getSuhClient-nimistä metodia, joka sijaitsee RestServiceGenerator-luokassa, ja liitetään se restService-muuttujaan. getSuhClient sisältää Retrofit-palvelun, johon on lisätty SuhClient-rajapinta. SuhClient on rajapinta Suharin sisällä, johon on luokiteltu etukäteen kaikki Suharin tekemät HTTP-kutsut (kuva 15).

```java
package com.example.mylly1.suhari.api.service;

import com.example.mylly1.suhari.api.model.CarRepo;
import com.example.mylly1.suhari.api.model.SuhariRepo;
import retrofit2.Call;
import retrofit2.Callback;
import retrofit2.http.Body;
import retrofit2.http.DELETE;
import retrofit2.http.GET;
import retrofit2.http.POST;
import retrofit2.http.PUT;
import retrofit2.http.Path;

/**
 * Created by Mylly1 on 18.5.2017.
 */

public interface SuhClient {
    /**
     * @param Job
     * @return
     */
    @GET("/cars/job/{id}")
    Call<SuhariRepo> fetchJob(
            @Path("id") String Job

    );
    @DELETE("/cars/job/{id}")
    Call<Void> delJob(
            @Path("id") String Job
    );

    @POST("/cars/car")
    Call<Void> postCar(
            @Body CarRepo body
    );
    @PUT("/cars/car")
    Call<Void> updateCar(
             @Body CarRepo body
    );
    @DELETE("/cars/car/{id}")
    Call<Void> delCar(
            @Path("id") String Car
    );

}
```

KUVA 15. SuhClient-rajapinta

Kuvassa 16 nähdään kutsu palvelimella olevaan rajapintaan, joka käyttää Retrofit2-kirjastoa apunaan. Kutsussa haetaan kyyti sen jälkeen, kun Suharin käyttäjä on hyväksynyt kyydin.

```
private void FetchJob(){
    restService.fetchJob("auto77").enqueue(new Callback<SuhariRepo>() {
        @Override
        public void onResponse(Call<SuhariRepo> call, Response<SuhariRepo> response) {
            if(response.isSuccessful()){
                DeleteJobFromDb();
                EventBus.getDefault().post(response.body());
            }else{
                int statusCode = response.code();

            }
        }

        @Override
        public void onFailure(Call<SuhariRepo> call, Throwable throwable) {
            Log.d("REST", throwable.getMessage());
            MakeToast("Kyydin hakeminen epäonnistui, soita keskukseen", Toast.LENGTH_LONG);
        }

    });

}
```

KUVA 16. Suhari Android, fetchJob-metodi, kyydin hakeminen koodissa

5.4 Rajapinta

ASP.NET Core on modulaarinen luonteeltaan. Tästä syystä rajapinnan sisällä olevat osat piti ns. irrottaa toisistaan mahdollisimman hyvin, jotta osien välillä oleva linkki pysyisi löyhänä. Kyseisestä asiasta on kirjoitettu jo opinnäytetyön kohdassa 4.3 Rajapinta. ASP.NET Core ei ole loppujen lopuksi muuta kuin konsolisovellus, johon on kiinnitetty erilaisia osia, kuten esimerkiksi Kestrel-web-palvelin. Kestrel ei itsessään vielä pysty käsittelemään liikennettä internetistä, vaan tarvitsee IIS-web-palvelimen itsensä eteen. Microsoftin mukaan syy tälle on se, että, Kestrel on web-palvelimena vielä todella uusi eikä tietoturvaa ole vielä ehditty hioa loppuun.(10)

Suharin rajapinnassa on yksi controller (liite 2). Controller hallitsee rajapintaan tulevia yhteyksiä. Controller on rajapinnan ns. poliisi, joka hallitsee, kuka rajapinnan palveluita pääsee käyttämään ja minkälaisia kutsuja rajapintaan voi tehdä.

Tämän lisäksi rajapinnassa on luokka nimeltä CarRepository(liite 4). CarRepository-luokkaa kutsutaan, kun halutaan tehdä kutsu tietokantaan. Koska osien välillä oleva liitos täytyy pitää löyhänä (engl. loose coupling), controllerissa ei kutsuta CarRepository-luokkaa tai luoda siitä uutta instanssia. Sen sijaan luodaan riippuvuusinjektio. CarRepository-luokka lisää itseensä rajapinnan (engl. interface) nimeltä ICarRepository(Liite 3). Tämän jälkeen ICarRepository injektoidaan controlleriin.

Näin saadaan aikaan riippuvuusinjektio. Näin controllerin ei tarvitse tietää, miten CarRepositoryssa muodostetaan palvelut tai mitä palveluita CarRepository käyttää. Tässä tapauksessa CarRepository käyttää MongoDB:n C#-ajuria. Ainoa asia, mitä controllerin täytyy tietää tässä tapauksessa, on se, mitä palveluita IcarRepository-interface antaa.

Rajapinta toimii täysin samalla logiikalla kuin teoriaosan luvussa 4.3 Rajapinta on kerrottu.

5.5 Tietokanta

Projektissa käytettävä MongoDB-tietokanta on tehty Windows-palveluksi tietokoneelle, jossa projektia kehitetään. Tässä osiossa ei tulla käymään läpi sitä, miten MongoDB:stä saadaan Windows-palvelu. Internetissä on todella iso määrä ohjeita tämän asian suorittamiseen. Asian voi käydä lukemassa suoraan esimerkiksi MongoDB:n dokumentoinnista(11).

Tässä osiossa keskitytään siihen, miten MongoDB:n C#-ajuria käytetään ASP.NET Coressa. MongoDB oli opinnäytetyön tekovaiheen aikoina tai vähän sitä ennen muuttanut ajureitaan. Arvaus tähän on se, että C#-ajurista haluttiin tehdä paremmin toimiva paketti juurikin ASP.NET Coren kanssa. Joka tapauksessa uuden ajurin käyttöohjeita oli loppujen lopuksi todella vaikea löytää, ja esimerkiksi geo-hakujen tekeminen oli aluksi todella vaikeaa asian uutuuden takia. Näin MongoDB:n C#-ajuria voi ainakin käyttää ASP.NET Coressa. Kuvassa 17 nähdään miten MongoDB:lle voidaan tehdä oma luokka, jonka muodostinta voidaan tarvittaessa kutsua.

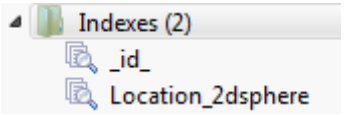
```
public class MongoDBContext
{
    private IMongoDatabase _database = null;

    public MongoDBContext(IOptions<Settings> settings)
    {

        var client = new MongoClient(settings.Value.ConnectionString);

        if(client != null)
        {
            _database = client.GetDatabase(settings.Value.DatabaseName);
        }

        var collection = _database.GetCollection<Car>("Cars");
        //create index for geoqueries
        collection.Indexes.CreateOneAsync(Builders<Car>.IndexKeys
            .Geo2DSphere(i => i.Location));

    }
    public IMongoCollection<Job> Jobs
    {
        get
        {
            return _database.GetCollection<Job>("Jobs");
        }
    }
    public IMongoCollection<Car> Cars
    {
        get
        {
            return _database.GetCollection<Car>("Cars");

        }
    }

}
```

KUVA 17. Rajapinta, MongoDBContext-luokka

Otetaan esimerkiksi geo-haut. Geo-haulla tarkoitetaan hakua, jossa tietokannasta haetaan dokumentti sen perusteella, mitkä koordinaatit dokumentissa ovat. Jos geo-hakuja haluaa tehdä, täytyy tietokantaan luoda indeksi siihen kokoelmaan, missä haettavat dokumentit ovat(kuva 18). Tämä tehdään esimerkiksi MongoDBContext-luokan muodostimessa. Tämän jälkeen käydään katsomassa Robomongosta, onko indeksi luotu(kuva 19).

```
collection.Indexes.CreateOneAsync(Builders<Car>.IndexKeys.Geo2DSphere(i => i.Location));
```

KUVA 18. Indeksin luominen MongoDB-tietokantaan



KUVA 19. MongoDB, autot-kokoelma, indeksit

Rajapinnassa MongoDBContextista luodaan instanssi CarRepository-luokassa. MongoDBContext asetetaan muuttujaan nimeltä _context. Jotta hakuja voidaan suorittaa, täytyy ensin olla jotain mitä hakea. Rajapinnassa ainoastaan autoja haetaan koordinaattien avulla. Koska ylhäällä oleva indeksi on luotu muuttujaan Geo2DSphere, täytyy autojen koordinaatit muuttaa kahdesta double-muuttujasta muuttujaan GeoJsonPoint. Kuvassa 20 auton Location-muuttujaan laitetaan GeoJsonPoint-luokka, joka sisältää ja perii GeoJson2DGeographicCoordinates-luokan. Tämän jälkeen auton tiedot tallennetaan tietokantaan.

```
car.Location = new GeoJsonPoint<GeoJson2DGeographicCoordinates>(
new GeoJson2DGeographicCoordinates(tempCar.lon, tempCar.lat));
await _context.Cars.InsertOneAsync(car);
```

Kuva 20. Rajapinta, auton lisääminen GeoJson-attribuutin kanssa

MongoDB:stä voidaan hakea dokumentteja käyttämällä apuna LINQ:ta(Language Integrated Query). LINQ:llä voidaan hakea tietoa helposti juurikin vaikka dokumenttitietokannasta(12, s.2). Kuvassa 21 näytetään esimerkki geo-hausta MongoDB:stä käyttäen apuna LINQ:ta.

```
public async Task<Car> FindClosestCar(Job job)
{
    var point = GeoJson.Point(GeoJson.Geographic(job.lng, job.lat));
    var locationQ = new FilterDefinitionBuilder<Car>().Near(t => t.Location, point, 200000);

    try
    {
        return await _context.Cars.Find(locationQ).FirstAsync();


    }
    catch(Exception e)
    {
        return null;
        throw e;
    }
}
```

KUVA 21. Rajapinta, lähimmän auton löytäminen kyydille

5.6 Taustaprosessi

Taustaprosessi toteutettiin .NET Core -konsolisovelluksena. Taustaprosessin ainoa tarkoitus on kutsua palvelimen rajapinnan Update-metodia (liite 2), kun minuutti ja 15 sekuntia on kulunut. Update-metodi sijaitsee rajapinnan controllerissa. Konsolisovellus on tarkoitus laittaa Windows-palveluksi palvelimelle. Käytännössä tämä tarkoittaa vain sitä, että sovellus käynnistyy automaattisesti, kun palvelinkin käynnistyy. Kuvassa 22 näkyy konsolisovelluksen koko koodi.

```
using System;
using System.Threading;
using System.Net.Http;
namespace SuhariService
{
    class Program
    {
        Timer _tm = null;
        AutoResetEvent _autoEvent = null;
        private int _counter = 0;
        private static HttpClient client = new HttpClient();

        static void Main(string[] args)
        {
            client.BaseAddress = new Uri("http://192.168.10.101:3000/cars/");
            Console.WriteLine("Suhari suhahtaa");
            Program p = new Program();
            p.StartTimer();
        }
        public void StartTimer()
        {
            _autoEvent = new AutoResetEvent(false);
            _tm = new Timer(Execute, _autoEvent, 1000, 61500);

            Console.Read();
        }
        public void Execute(Object stateInfo)
        {
            var response = client.GetAsync("jobs/update").Result;
            Console.WriteLine(response.StatusCode);
            Console.ReadLine();
        }

        }
}
```

KUVA 22. Taustaprosessi

Kun taustaprosessi kutsuu rajapinnassa olevaa endpointtia, tapahtuu seuraavia asioita: Haetaan kaikki kyydit listaan, joiden UpdatedOn-muuttuja on vanhempi kuin serverin tämänhetkinen aika,

josta on vähennetty minuutti ja 15 sekuntia. Tämän jälkeen kaikille listassa oleville kyydeille tehdään seuraava toimenpide. Haetaan lähin auto kyydille ja lisätään auton indeksi kyydin car.Id-kenttään. Tämän jälkeen auto poistetaan vapaana olevien autojen listalta, ja seuraavan kerran, kun auto kyselee kyytiä, rajapinta ilmoittaa kyselijälle, että tietokannassa on tarjolla kyyti.

Kun näin toimitaan, pysyy tietokanta puhtaana autoista, jotka ovat esimerkiksi menettäneet verkkoliikenteensä ja auto ei ole tämän takia poistunut tietokannasta. Jos kyydillä on valittuna auto, joka on jonkin virheen takia estynyt ottamaan kyydin vastaan, poistetaan auto silti jossain välissä järjestelmästä.

5.7 Todentamispalvelu ja testaus

Valitettavasti projektille varattu aika loppui kesken ja todentamispalvelun toteuttaminen jäi tekemättä. Todentamispalvelu on erittäin tärkeä osa palvelua ja se kannattaa saada kerralla oikein. Myös testaaminen jäi pois kokonaan ja projektin koodissa on vielä paljon ongelmia, jotka tulisi ratkaista. Työhön oli tässä vaiheessa käytetty jo reilusti yli 400 tuntia ja työn palautuspäivämäärä oli mennyt jo ohi. Koska työstä jäi puuttumaan todentamispalvelu, palvelua ei kannata vielä laittaa minkäänlaiseen pilvipalveluun, eikä aika tähän riittäisikään.

6 LOPPUSANAT

Opinnäytetyöni tarkoituksena oli tehdä järjestelmä taksikyytien levittämiseen ja siinä osittain onnistuttiinkin. Järjestelmästä jäi pois käyttäjien todentaminen sekä palvelun tarjoaminen pilvipalvelusta käsin. Työhön käytetty aika rupesi olemaan jo reippaasti yli 400 tuntia erinäisistä syistä johtuen. Projektin valmistuminen myöhästyikin parilla viikolla ja palautettiin viime tipassa. Yllä mainittuihin syihin kuuluvat mm. kehitysympäristöjen kanssa jatkuva taisteleminen, tutkimuksen määrän suuri koko sekä ASP.NET Coren uutuudesta johtuva tiedon etsimisen vaikeus. Projektin alkutaipaleella jouduin asentamaan Visual Studion sekä Node.js:n kokonaan uusiksi käyttöongelmien takia.

En ole koskaan ennen tehnyt näin kattavaa projektia yksin. Aiempaa kokemusta minulla oli Android Studion käytöstä jonkin verran ja WPF-sovelluksia olen joskus koulussa kursseilla tehnyt. En ollut myöskään käyttänyt ASP.NETiä ikinä ennen, puhumattakaan siitä, että ASP.NET Coressa on muutettu monia käytäntöjä vanhaan versioon nähden. ASP.NET Coren uutuuden takia tutkimusta tehdessäni huomasin monesti törmääväni vanhempaan tekniikkaan liittyviin ohjeisiin. ASP.NET Core kehittyy vielä valtavin harppauksin ja tästä syystä olen sitä mieltä, että ASP.NET Coren käyttöönoton kanssa kannattaa odottaa vielä vuosi tai kaksi. ASP.NET Core vaikuttaa todella hyvältä ja pienen viilauksen jälkeen tulee olemaan nopea ja paljon käytetty ohjelmistokehys. Sen sijaan Androidin kanssa oli juuri päinvastainen ongelma. Tietoa löytyi todella suuria määriä, mutta iso osa tiedosta oli jo vanhentunutta. Google on vaihtanut vuosien varrella nimeämiskäytäntöjä sekä toimintatapoja erittäin monta kertaa. Tekniikka kehittyy koko ajan eteenpäin.

Järjestelmä on vielä erittäin herkkä kaatumiselle. Myös kyytien valvojan työkaluja tulisi lisätä. Järjestelmä toimii, mutta on nykyisessä tilassaan vielä suojaamaton. Jos toteuttaisin pääjärjestelmän kyseiseen käyttötarkoitukseen, tekisin sen aivan eri tavalla.

Kaikesta tästä huolimatta projekti oli opinnäytetyön aiheena todella hyvä. Aihe kiinnosti minua paljon ja opin valtavan määrän uusia asioita. Sain myös ideoita seuraavaa projektia varten.

LÄHTEET

1. ASP.NET Core Scedule and Roadmap. 2017. GitHub. Saatavissa: https://github.com/aspnet/Home/wiki/Roadmap. Hakupäivä 5.6.2017.

2. Loose coupling. 2017. Wikipedia. Saatavissa: https://en.wikipedia.org/wiki/Loose_coupling. Hakupäivä 5.6.2017.

3. What is AWS Lambda? 2017. AWS Documentation. Saatavissa: http://docs.aws.amazon.com/lambda/latest/dg/welcome.html. Hakupäivä 6.6.2017.

4. MongoDb at Scale. 2017. MongoDB Saatavissa: https://www.mongodb.com/mongodb-scale. Hakupäivä 6.6.2017.

5. GeoSpatial Indexes and Queries in MongoDB. 2017. MongoDB Documentation Saatavissa: https://docs.mongodb.com/manual/applications/geospatial-indexes/. Hakupäivä 6.6.2017.

6. Why you should always use access tokens to secure an API. 2016. Auth0 Articles. Saatavissa: https://auth0.com/docs/api-auth/why-use-access-tokens-to-secure-apis. Hakupäivä 6.6.2017.

7. iisexpress-proxy. 2017. GitHub. Saatavissa: https://github.com/icflorescu/iisexpress-proxy. Hakupäivä 6.6.2017.

8. ISC License. 2016. Saatavissa: https://github.com/icflorescu/iisexpress-proxy/blob/master/LICENSE. Hakupäivä 6.6.2017.

9. Developing API's is hard. Postman makes it easy. Postman. Saatavissa: https://www.getpostman.com/. Hakupäivä 6.6.2017.

10. Web server implementations in ASP.NET Core. 2016. Microsoft Docs. Saatavissa: https://docs.microsoft.com/en-us/aspnet/core/fundamentals/servers/. Hakupäivä 6.6.2017.

11. Install MongoDB Community Edition on Windows. MongoDB manual. Saatavissa: https://docs.mongodb.com/manual/tutorial/install-mongodb-on-windows/. Hakupäivä 6.6.2017.

12. Vuorinen, Mikko. 2010. LINQ-ohjelmointikieleen yhdistetty hakuarkkitehtuuri. Insinöörityö. Metropolia Ammattikorkeakoulu, tietotekniikan koulutusohjelma. Saatavissa: https://publications.theseus.fi/bitstream/handle/10024/14745/Vuorinen_Mikko.pdf?sequence=2. Hakupäivä 6.6.2017.

13. MongoDB C# Driver connection pooling. 2015. Stack Overflow. Saatavissa: https://stackoverflow.com/questions/29208923/mongodb-c-sharp-driver-connection-pooling?rq=1. Hakupäivä 8.6.2017.

14. ASP.NET Core Web API Getting Started. 2016. Auth0 QuickStarts. Saatavissa: https://auth0.com/docs/quickstart/backend/aspnet-core-webapi. Hakupäivä 8.6.2017.

15. Calling APIs from a Service. 2016. Auth0 Articles. Saatavissa: https://auth0.com/docs/api-auth/grant/client-credentials. Hakupäivä 8.6.2017.

ANDROID-SOVELLUKSEN BACKGROUNDCONNECTIONSERVICE-LUOKKA LIITE 1

```
package com.example.mylly1.suhari;


import ...

/**
 * Created by Mylly1 on 25.5.2017.
 */

public class BackgroundConnectionService extends Service {
    public static final String TAG = "MyBckGrndServiceTag";
    private SuhClient restService;
    final Intent i = new Intent();

    @Override
    public IBinder onBind(Intent intent) { return null; }

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        //Toast.makeText(getApplicationContext(), "hi there", Toast.LENGTH_LONG).show();
        Log.i("LocalService", "Received start id " + startId + ": " + intent);

        SharedPreferences prefs = getSharedPreferences("com.example.mylly1.suhari", Context.MODE_PRIVATE);
        final Intent askUserIntent = new Intent(this, RideRequestActivity.class);
        askUserIntent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
        CarRepo rp = new CarRepo();
        rp.setId("auto77");
        rp.setLon(Double.longBitsToDouble(prefs.getLong("lon", Double.doubleToLongBits(25.474275))));
        rp.setLat(Double.longBitsToDouble(prefs.getLong("lat", Double.doubleToLongBits(65.012737) )));
        //rp.setLoc(loc);
        restService.updateCar(rp).enqueue(new Callback<Void>() {
            @Override
            public void onResponse(Call<Void> call, Response<Void> response) {
                if(response.code() == 200)
                {
                    Log.i("servicevast", response.message());
                    startActivity(askUserIntent);
                }
                else if(response.code() == 204){
                    MakeToast("Etsii vieläkin", Toast.LENGTH_SHORT);
                    Log.w("servicevast2", response.message());
                }
            }
            @Override
            public void onFailure(Call<Void> call, Throwable throwable) {
                if(throwable.getMessage() != null)
                    Log.w("servicevastError", throwable.getMessage());
                    MakeToast("Ei yhteyttä, uudelleenkäynnistä etsimminen", Toast.LENGTH_LONG);
                //TODO stop searching if server goes offline
            }
        });
        Toast.makeText(getApplicationContext(), Double.toString(rp.getLat()) + " and " + Double.toString(rp.getLon()),
                Toast.LENGTH_LONG).show();
        return START_NOT_STICKY;
    }

    @Override
    public void onCreate()
    {
        restService = RESTServiceGenerator.getSuhClient();
        //restService = ApiUtils.getSuhClient();

    }
    public void onDestroy(){

    }
    public void MakeToast(String msg, int duration){
        Context cxt = getApplicationContext();
        //Toast.LENGTH_SHORT;

        Toast toast = Toast.makeText(cxt, msg, duration);
        toast.show();
    }

}
```

```
[Produces("application/json")]
[Route("cars")]
public class CarController : Controller
{
    private readonly ICarRepository _carRepository;

    public CarController(ICarRepository carRepository)
    {
        _carRepository = carRepository;
    }


    [Route("car")]
    [HttpPost]
    public async Task<IActionResult> PostCar([FromBody]TempCar tempCar)
    {
        await _carRepository.AddCar(tempCar);


        return Ok();
    }
    [Route("postJob")]
    [HttpPost]
    public async Task<IActionResult> PostJob([FromBody] Job job)
    {
        if (job != null)
        {
            await _carRepository.AddJob(job);
            return Ok();
        }
        return StatusCode(StatusCodes.Status204NoContent);
    }
    [Route("car")]
    [HttpPut]
    public async Task<IActionResult> UpdateCar([FromBody] TempCar tempCar)
    {
        var job = await CheckForJob(tempCar.id, tempCar);
        if(job == true)
        {
            return Ok();

        }
        else
        {
            return StatusCode(StatusCodes.Status204NoContent);
        }
    }
    private async Task<Boolean> CheckForJob(string carId, TempCar tempCar)
    {


        var taskCheckJobStatus = _carRepository.CheckForJob(carId);
        var jobStatus = await taskCheckJobStatus;
        if(jobStatus)
        {
            await _carRepository.RemoveCar(carId);
            return jobStatus;
        }
        else
        {
            var taskUpdate = _carRepository.UpdateCar(tempCar);
            var derp = await taskUpdate;
            return false;
        }

    }
    [Route("job/{id}")]
    [HttpGet]
    public async Task<IActionResult> GetJob(string id)
    {
        TempJob job =  await _carRepository.GetJob(id);

        return Json(job);

    }
    [Route("car/{carId}")]
    [HttpDelete]
    public async Task<IActionResult> DeleteCar(string carId)
    {
        var result = await _carRepository.RemoveCar(carId);
        if(result.IsAcknowledged)
        {
            return Ok();
        }
        else
        {
            return StatusCode(StatusCodes.Status409Conflict);
        }
    }

    [Route("job/{carId}")]
    [HttpDelete]
    public async Task<IActionResult> DeleteJob(string carId)
    {
        var result = await _carRepository.RemoveJob(carId);
        if(result.IsAcknowledged)
        {
            return Ok();
        }
        else
        {
            return StatusCode(StatusCodes.Status409Conflict);
        }
    }
```

```
[Route("jobs/update")]
[HttpGet]
public async Task<IActionResult> Update()
{
    await _carRepository.UpdateJobs();
    return Ok();
}
```

RAJAPINNAN RIIPPUVUUSINJEKTIO LIITE 3

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using SuhariBE.Models;
using MongoDB.Driver;
namespace SuhariBE.Interfaces
{
    public interface ICarRepository
    {
        Task<IEnumerable<Car>> GetAllCars();
        Task<Car> GetCar(string id);
        Task AddCar(TempCar tempCar);
        Task<DeleteResult> RemoveCar(string id);
        Task AddJob(Job job);
        Task<Car> FindClosestCar(Job job);
        Task<Boolean> CheckForJob(string carId);
        Task<UpdateResult> UpdateCar(TempCar tempCar);
        Task<TempJob> GetJob(string id);
        Task<DeleteResult> RemoveJob(string carId);
        Task UpdateJobs();

    }
}
```

```
public async Task<Boolean> CheckForJob(string carId)
{
    var filter = Builders<Job>.Filter.Eq("CarId", carId);

    try
    {
        Job job = await _context.Jobs
                        .Find(filter)
                        .FirstOrDefaultAsync();
        if(job == null)
        {
            return false;
        }
        else
        {
            return true;
        }

        //return true;
    }
    catch (Exception ex)
    {
        throw ex;
        // log or manage the exception

    }
}
public async Task<DeleteResult> RemoveCar(string id)
{
    try
    {
        return await _context.Cars.DeleteOneAsync(
             Builders<Car>.Filter.Eq("Id", id));
    }
    catch (Exception ex)
    {
        // log or manage the exception
        throw ex;
    }
}

public async Task<TempJob> GetJob(string id)
{
    var filter = Builders<Job>.Filter.Eq("CarId", id);

    try
    {
        Job job =  await _context.Jobs
                        .Find(filter)
                        .FirstOrDefaultAsync();
        TempJob job2 = new TempJob();
        //TODO edit datetime into proper form
        job2.CreatedOn = job.CreatedOn;
        job2.Address = job.Address;
        job2.Customer = job.Customer;
        job2.Info = job.Info;
        job2.lat = job.lat;
        job2.lng = job.lng;
        return job2;


    }
    catch (Exception ex)
    {
        throw ex;
    }
}

public async Task<DeleteResult> RemoveJob(string carId)
{
    try
    {
        return await _context.Jobs.DeleteOneAsync(
             Builders<Job>.Filter.Eq("CarId", carId));
    }
    catch (Exception ex)
    {
        // log or manage the exception
        throw ex;
    }
}
```

RAJAPINNAN CARREPOSITORY-LUOKKA LIITE 4/2

```
public async Task AddJob(Job job)
{
    var car = await FindClosestCar(job);

    if(car != null)
    {
        job.CarId = car.Id;
        var delCar = await RemoveCar(car.Id);

    }
    else
    {
        job.CarId = null;
    }

    try
    {
        await _context.Jobs.InsertOneAsync(job);

    }
    catch(Exception e)
    {
        throw e;
    }

}

public async Task<Car> FindClosestCar(Job job)
{
    var point = GeoJson.Point(GeoJson.Geographic(job.lng, job.lat));
    var locationQ = new FilterDefinitionBuilder<Car>().Near(t => t.Location, point, 200000);

    try
    {
        return await _context.Cars.Find(locationQ).FirstAsync();


    }
    catch(Exception e)
    {
        return null;
        throw e;
    }
}


public async Task<ReplaceOneResult> UpdateCar(string Id, Car Item)
{
    try
    {
        return await _context.Cars
                    .ReplaceOneAsync(n => n.Id.Equals(Id),
                                Item,
                                new UpdateOptions { IsUpsert = true });
    }
    catch (Exception ex)
    {
        // log or manage the exception
        throw ex;
    }
}
```

```
//gets called by the backgroundprocess
public async Task UpdateJobs()
{
    var filter = Builders<Job>.Filter.Lt(p => p.UpdatedOn,
        DateTime.UtcNow.AddMinutes(-1).AddSeconds(-15));
    var jobResult = await _context.Jobs.FindAsync(filter);

    if (jobResult != null)
    {
        foreach (var job in jobResult.ToListAsync().Result)
        {

            try
            {
                var point = GeoJson.Point(GeoJson.Geographic(job.lng, job.lat));
                var locationQ = new FilterDefinitionBuilder<Car>()
                    .Near(t => t.Location, point, 200000);
                Car car = _context.Cars.Find(locationQ).FirstAsync().Result;

                if (car != null)
                {
                    var nextCarId = car.Id;
                    //if the closest car is the same one
                    await _context.Cars.DeleteOneAsync(Builders<Car>.Filter.Eq("Id", car.Id));
                    job.UpdatedOn = DateTime.Now;
                    var jobFilter = Builders<Job>.Filter.Eq(s => s.Id, job.Id);
                    var jobUpdate = Builders<Job>.Update
                        .Set(s => s.CarId, nextCarId)
                        .Set(x => x.UpdatedOn, DateTime.Now);
                    await _context.Jobs.UpdateOneAsync(jobFilter, jobUpdate);


                }
            }
            catch (InvalidOperationException e)
            {
                var jobFilter = Builders<Job>.Filter.Eq(s => s.Id, job.Id);
                var jobUpdate = Builders<Job>.Update
                    .Set(s => s.CarId, null)
                    .Set(x => x.UpdatedOn, DateTime.Now);
                await _context.Jobs.UpdateOneAsync(jobFilter, jobUpdate);

            }
            catch (Exception e)
            {
                //log it
            }

        }
    }
}
```

ANDROID-SOVELLUKSEN GPSSERVICE-LUOKKA LIITE 5/1

```
package com.example.myllyl.suhari;

import android.Manifest;
import android.app.Service;
import android.content.Context;
import android.content.Intent;
import android.content.SharedPreferences;
import android.content.pm.PackageManager;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.os.Bundle;
import android.os.Handler;
import android.os.IBinder;
import android.provider.Settings;
import android.support.annotation.Nullable;
import android.support.v4.content.ContextCompat;
import android.util.Log;
import android.widget.Toast;

/**
 * Created by myllyl on 27.5.2017.
 */

public class GpsService extends Service {

    public static final int ONE_MINUTE = 30000; // 120 seconds //60000 == one min //Suluin vaihtaa
    public static Boolean isRunning = false;
    public static double LAT;
    public static double LNG;
    public LocationManager mLocationManager;
    public GpsService.LocationUpdaterListener mLocationListener;
    public Location previousBestLocation = null;

    @Nullable
    @Override
    public IBinder onBind(Intent intent) { return null; }

    @Override
    public void onCreate() {
        mLocationManager = (LocationManager) getSystemService(Context.LOCATION_SERVICE);
        mLocationListener = new GpsService.LocationUpdaterListener();

        super.onCreate();

    }

    Handler mHandler = new Handler();
    Runnable mHandlerTask = () -> {
            if (!isRunning) {
                startListening();
            }
            mHandler.postDelayed(mHandlerTask, ONE_MINUTE);

    };

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        mHandlerTask.run();

        Toast.makeText(getApplicationContext(), "gps alkaa"+ Double.toString(LNG)+ Double.toString(LAT)
                , Toast.LENGTH_LONG).show();
        return START_STICKY;
    }

    @Override
    public void onDestroy() {
        stopListening();
        mHandler.removeCallbacks(mHandlerTask);
        super.onDestroy();
    }

    private void startListening() {
        if (ContextCompat.checkSelfPermission(this, Manifest.permission.ACCESS_FINE_LOCATION)
                == PackageManager.PERMISSION_GRANTED
                || ContextCompat.checkSelfPermission(this, Manifest.permission.ACCESS_COARSE_LOCATION)
                == PackageManager.PERMISSION_GRANTED) {
            if (mLocationManager.getAllProviders().contains(LocationManager.NETWORK_PROVIDER))
                mLocationManager.requestLocationUpdates(LocationManager.NETWORK_PROVIDER, 0, 0, mLocationListener);

            if (mLocationManager.getAllProviders().contains(LocationManager.GPS_PROVIDER))
                mLocationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 0, 0, mLocationListener);
        }
        isRunning = true;
    }

    private void stopListening() {
        if (ContextCompat.checkSelfPermission(this, Manifest.permission.ACCESS_FINE_LOCATION)
                == PackageManager.PERMISSION_GRANTED
                || ContextCompat.checkSelfPermission(this, Manifest.permission.ACCESS_COARSE_LOCATION)
                == PackageManager.PERMISSION_GRANTED) {
            mLocationManager.removeUpdates(mLocationListener);
        }
        isRunning = false;
    }
```

ANDROID-SOVELLUKSEN GPSSERVICE-LUOKKA

```
public class LocationUpdaterListener implements LocationListener
{
    @Override
    public void onLocationChanged(Location location) {
        if (isBetterLocation(location, previousBestLocation)) {
            previousBestLocation = location;
            try {
                //save coordinates to sharedPreferences
                SharedPreferences prefs = getSharedPreferences("com.example.agilyl_sukel", Context.MODE_PRIVATE);

                if(location.getLongitude() != 0.0 ( location.getLatitude() != 0.0)
                {

                    prefs.edit().putLong("lat", Double.doubleToLongBits(location.getLatitude())).apply();
                    prefs.edit().putLong("lon", Double.doubleToLongBits(location.getLongitude())).apply();

                }
                else{
                    prefs.edit().putLong("lat", Double.doubleToLongBits(60.012787)).apply();
                    prefs.edit().putLong("lon", Double.doubleToLongBits(25.674277)).apply();

                }

            }
            catch (Exception e) {
                e.printStackTrace();
            }
            finally {
                stopListening();
            }
        }
    }

    @Override
    public void onProviderDisabled(String provider) {
        stopListening();

    }

    @Override
    public void onProviderEnabled(String provider) { }

    @Override
    public void onStatusChanged(String provider, int status, Bundle extras) { }
}

protected boolean isBetterLocation(Location location, Location currentBestLocation) {
    if (currentBestLocation == null) {
        // A new location is always better than no location
        return true;
    }

    // Check whether the new location fix is newer or older
    long timeDelta = location.getTime() - currentBestLocation.getTime();
    boolean isSignificantlyNewer = timeDelta > ONE_MINUTE;
    boolean isSignificantlyOlder = timeDelta < -ONE_MINUTE;
    boolean isNewer = timeDelta > 0;

    // If it's been more than two minutes since the current location, use the new location
    // because the user has likely moved
    if (isSignificantlyNewer) {
        return true;
        // If the new location is more than two minutes older, it must be worse
    } else if (isSignificantlyOlder) {
        return false;
    }

    // Check whether the new location fix is more or less accurate
    int accuracyDelta = (int) (location.getAccuracy() - currentBestLocation.getAccuracy());
    boolean isLessAccurate = accuracyDelta > 0;
    boolean isMoreAccurate = accuracyDelta < 0;
    boolean isSignificantlyLessAccurate = accuracyDelta > 200;

    // Check if the old and new location are from the same provider
    boolean isFromSameProvider = isSameProvider(location.getProvider(), currentBestLocation.getProvider());

    // Determine location quality using a combination of timeliness and accuracy
    if (isMoreAccurate) {
        return true;
    } else if (isNewer && !isLessAccurate) {
        return true;
    } else if (isNewer && !isSignificantlyLessAccurate && isFromSameProvider) {
        return true;
    }
    return false;
}

/** Checks whether two providers are the same */
private boolean isSameProvider(String provider1, String provider2) {
    if (provider1 == null) {
        return provider2 == null;
    }
    return provider1.equals(provider2);
}
@Override
public void onTaskRemoved(Intent rootIntent)
```

```
    @Override
    public void onTaskRemoved(Intent rootIntent)
    {
        Toast.makeText(getApplicationContext(), "gps loppuu"+ Double.toString(LNG)+ Double.toString(LAT)
                , Toast.LENGTH_LONG).show();
        stopListening();
        mHandler.removeCallbacks(mHandlerTask);
        stopSelf();
        super.onDestroy();

    }
}
```