SAVONIA

OPINNÄYTETYÖ - AMMATTIKORKEAKOULUTUTKINTO
TEKNIIKAN JA LIIKENTEEN ALA

KIHO LAITEASENNUSSOVELLUS

TEKIJÄ: Lassi-Ville Mulari

SAVONIA-AMMATTIKORKEAKOULU

OPINNÄYTETYÖ
Tiivistelmä

Koulutusala Tekniikan ja liikenteen ala	
Tutkinto-ohjelma Tietotekniikan tutkinto-ohjelma	
Työn tekijä(t) Lassi-Ville Mulari	
Työn nimi Kiho Laiteasennussovellus	
Päiväys 1.12.2020	Sivumäärä/Liitteet 34
Toimeksiantaja Kiho (Mastercom Oy)	

Tiivistelmä

Kiho on kenttätyönhallintaan keskittynyt yritys, jonka toimintoihin kuuluvat muun muassa työajanseuranta ja älykkäät telematiikkapalvelut. Kiho tarjoaa ajoneuvoihin ja työkaluihin asennettavia paikantimia, jotka keräävät tietoja niiden käytöstä. (Kiho, s.a.)

Viime vuosina Kihon palvelut ovat laajentuneet paljon ja asennettavia laitteita on viikkotasolla jopa satoja. Laitteiden ylläpito ja asennus ovat työllistäneet merkittävästi asentajaa, asiakaspalvelua ja teknistä tukea eri muodoissaan. Tässä työssä perehdyttiin helpottamaan asennuksen vaiheita siten, että mahdollisimman moni työvaihe automatisoitaisiin Android -mobiilisovelluksen ja siihen liittyvän ohjelmiston avulla. Sovelluksen tuli toimia asentajan apuna laitteen asennuksessa, jotta tietokonetta ja muita organisaation jäseniä tarvittaisiin mahdollisimman vähän. Lisäksi sovellukselle asetettiin toimintatavoitteita vaikeita asennusolosuhteita varten.

Sovellus päätettiin toteuttaa Java -pohjaisesti Androidille natiivitoteutuksena. Sovelluksella pystyy tallentamaan paikantimelle tarvittavat tiedot jo asentajan toimesta, joista generoidaan lisäksi valmis dokumentaatio mahdollisia jälkiselvittelyitä varten. Lisäksi sovelluksella voidaan tarkastaa paikantimen ja sen liitännäislaitteiden, kuten moottorilämmittimen toimivuus. Projektin aikana keskityttiin myös projektinhallintaan, kuten ajankäyttöön, projektinhallintatyökaluihin ja ohjelmistoarkkitehtuurin yksityiskohtiin.

Lopputuloksena asentaja ja tekninen tuki saivat toimivan sovelluksen, joka kattaa hyvin asetetut vaatimukset. Projektin aikana tuli vielä kehitysideoita, joita jatkojalostetaan tulevaisuudessa muun työn sivussa.

Avainsanat
Android, PHP, Java, mobiiliohjelmointi

SAVONIA UNIVERSITY OF APPLIED SCIENCES

THESIS
Abstract

Field of Study Technology, Communication and Transport	
Degree Programme Degree Programme in Information Technology	
Author(s) Lassi-Ville Mulari	
Title of Thesis Kiho Device Management Tool	
Date 1 December 2020	Pages/Appendices 34
Client Organisation Kiho (Mastercom Oy)	

Abstract

The purpose of this project was to develop an Android app and the related software for the Kiho's installer and other parts of the organization. The application had to help the installer and other parts of the organization in several tasks with trackers and related devices. The application also had performance goals in difficult installation conditions.

The application was developed for Android as a native implementation and the programming language was decided to be Java. The application is able to save the necessary information for the tracker. The program creates documentation from installation parameters for possible later use. The application is also able to check the functionality of the tracker and connected devices, for example, the engine heater.

As a result, the installer and technical support got a working application that covers well the set requirements. During the project there were still some development ideas, which will be processed in the future alongside other work.

Keywords
Android, PHP, Java, Mobile Development

ESIPUHE

Haluan kiittää työnantajaani Kihoa mahdollisuudesta työskennellä erinomaisen organisaation riveissä. Aloitin Kiholla keväällä 2019, minkä jälkeen olen saanut työskennellä Kiholla opintojeni ohessa. Paljon on opittu ja valmistuminen häämöttää.

Opinnäytetyöstä erityiskiitos ohjelmistovastaava Henri Hartikaiselle, joka on alusta lähtien mahdollistanut rakettimaisen kehittymisen tuotekehityksen parissa. Lisäksi haluan kiittää koko tuotekehitystiimiä, etenkin Mobiiliohjelmoinnin tiimiä, joka oli suuri apu tämän projektin parissa.

Kiitokset myös kaikille organisaation jäsenille ja ulkopuolisille, jotka olivat mukana kannustamassa, testaamassa ja antamassa palautetta. Ilman näitä henkilöitä sovelluksesta ei olisi tullut niin hyvä, mitä se tänä päivänä on.

Kuopiossa 1.11.2020
Lassi-Ville Mulari

SISÄLTÖ

ACTIVITY

Androidin eri näkymät koostuvat Activity –pinosta. Pinossa on ikään kuin monta sivua, joiden välillä voi liikkua tarpeen mukaan. Activity voi sisältää erilaisia komponentteja ja activity toimii fragment-tien juurena.

ANDROID

Android on erilaisille älylaitteille suunniteltu ohjelmistopino, joka sisältää käyttöjärjestelmän, erilaisia väliohjelmistoja ja käyttäjälle suunnattuja perusohjelmistoja. Android on tällä hetkellä mobiilikäyttöjärjestelmistä suosituin 74,6% (heinäkuu 2020) markkinaosuudella. (Statcounter 2020)

ANDROID API –LEVEL

Android -käyttöjärjestelmän versiointiin käytetty numeraalinen järjestys.

ANDROID LIFECYCLE

Android Lifecycle tarkoittaa nimensä mukaisesti elämän kiertokulkua sovelluksen sisällä. Erinäiset sovelluksen osat poistetaan ja tuhotaan, kun toiset luodaan ja tuhotaan. Lisäksi toisen komponentin luonnissa ja poistamisessa on moninaisia vaiheista, jotka ovat kuvattuna dokumentin osassa 2.

API

Application Programming Interface on ohjelmoinnissa käytetty rajapinta, joka vastaanottaa dataa asiakkaalta ja lähettää dataa palvelimelta asiakkaalle.

BACK-END

Ohjelmoinnissa Back-end -termillä viitataan palvelimella suoritettavaan ohjelmakoodiin.

CHILD ISSUE

Child issuet ovat Epicien ala-aiheita, jossa määritellään yksittäisen Epicin osia. Child issuen voi jakaa myös vielä pienempiin tiketteihin.

COMMIT

Versionhallinnassa käytetty termi, joka osoittaa yhden vaiheen tehdyksi.

EPIC

Epic on projektinhallintatyökaluissa käytetty termi, jolla tarkoitetaan yhtä suurempaa projektin osaa, joka koostuu useista vaiheista ja ala-aiheista.

FRAGMENT

Fragmentilla tarkoitetaan Android –sovelluksissa yhtä osaa näkymässä. Yhdessä näkymässä voi olla useampia fragmentteja. Toisin kuin yksittäinen komponentti, fragment voi sisältää useampia erilaisia komponentteja. Fragment sisältää myös kyseenomaisen osion logiikan.

FRONT-END

Ohjelmoinnissa Front-end -termillä viitataan asiakkaan laitteella suoritettavaan ohjelmakoodiin ja käyttöliittymään.

GIT

Ohjelmoinnissa käytetty hajautettu versiohallintajärjestelmä. Versionhallinnalla pyritään pitämään yksittäiset ohjelmakoodin kehityshaarat toisistaan tietoisina ja yhtenäisinä.

JSON

JSON (Javascript Object Notation) on asiakkaan ja palvelimen väliseen datan siirtoon käytetty tiedostomuoto.

PHP

Lyhenne sanoista *Hypertext Preprocessor* -on ohjelmointikieli, jota käytetään pääsääntöisesti erilaisissa web -ympäristöissä. Nykyisin PHP:n käyttötavat ovat lähinnä palvelimen päässä.

REST

Repsesentational State Transfer. HTTP-protokollaan perustuva arkkitehtuurimalli rajapintojen toteuttamiseksi. (Wikipedia, s.a.)

USER EXPERIENCE (UX)

Käyttökokemus, eli sovelluksen käyttökokemus. UX suunnittelu kulkee käsikädessä UI-suunnittelun kanssa.

USER INTERFACE (UI)

User Interface, eli käyttöliittymä on sovelluksen käyttäjälle näkyvä osuus. Käyttöliittymän suunnittelu ja toteutus näyttelevät suurinta osaa, miten sovellus näyttäytyy käyttäjälleen.

1 JOHDANTO

1.1 Aihe ja aiheen valinta

Nykyaikainen digitalisaatio tuo paljon mahdollisuuksia korvata vanhat ja perinteiset järjestelmät ketterillä, älykkäillä ja tehokkailla tietojärjestelmillä. Tähän mahdollisuuteen on tarttunut myös Kiho, luodessaan omaa näkemystään nykyaikaisesta työnhallinta -järjestelmästä.

Kiho on alun perin kuopiolainen tietotekniikka-alan yritys, jonka päätuotteina ovat sähköiset kenttätyön ja kaluston hallinnat. Kihon tuotteet ovat käytettävissä selainpohjaisesti, sekä natiivisti mobiilialustoilla. Kihon mainoslause ”Kiho, tekee työstäsi helpompaa” on osuva kuvaus Kiho -järjestelmän tarkoituksesta ja toiminnasta.

Kihon telematiikka -järjestelmä on kasvanut vuosien saatossa suureksi. Asiakasyritysten määrät ovat tätä nykyä nelinumeroisia ja sen myötä paikantimien, sekä muiden telematiikka -laitteiden asennusten määrät ovat kasvaneet valtavasti. Laitteistoasennukset hoidetaan Kihon omien asentajien, teknisentuen, käyttöönottajien ja asiakaspalvelun yhteistyöllä. Monien eri tahojen ja muuttujien myötä asennusprosessi vaatii yrityksen sisältä paljon resursseja, jotka maksavat yritykselle aikaa ja rahaa.

Tässä projektissa keskitytään asennusprosessin automatisointiin nykyaikaisilla menetelmillä, luomalla mobiilisovellus ja tarvittavat palvelinohjelmistot asentajien ja asiakaspalvelun käyttöön. Sovelluksen keskeisenä tehtävänä on kertoa yksittäisen paikantimen toiminnasta, mahdollistaa paikantimen ja järjestelmän konfiguraatioiden muuttaminen, sekä luoda tarvittava dokumentaatio laitteen asennuksesta.

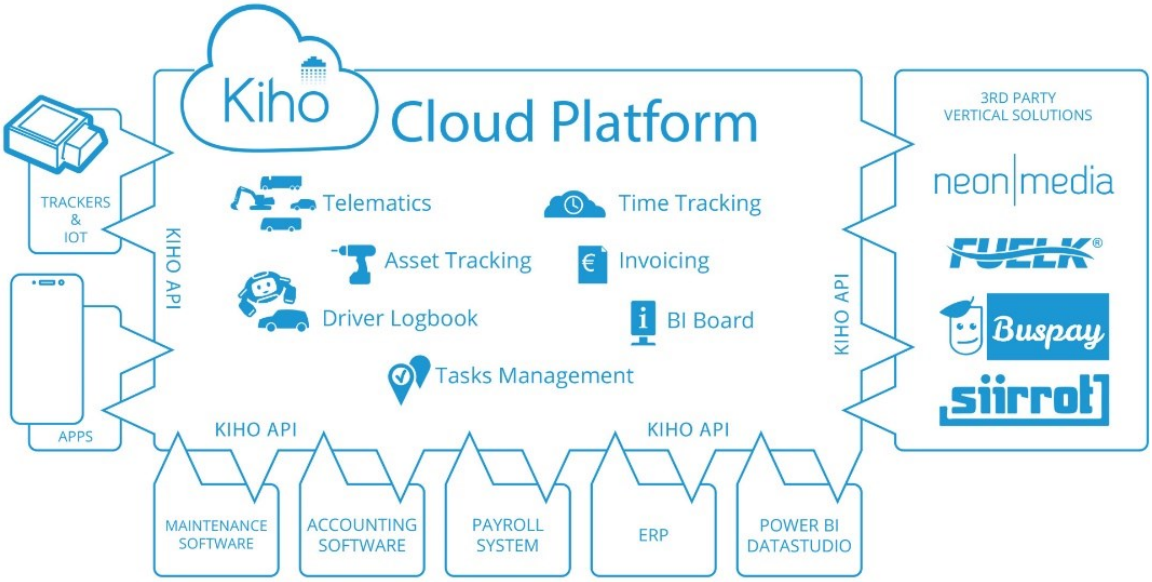
Opinnäytetyön aihe valikoitui projektin luonteen vuoksi. Projektilla ei ollut varsinaista ajallista takarajaa, joka mahdollisti myös muiden työtehtävien hoitamisen samanaikaisesti. Joustava projektin ajallinen määrittely mahdollisti työhön keskittymisen ja antoi enemmän pelivaraa kokonaisuuden, sekä yksityiskohtien suunnitteluun.

Työn aihe ja osaamisalueet olivat henkilökohtaisesti minulle osaltaan tuttuja ja osaltaan uutta asiaa. Toisaalta palvelinohjelmiston kehittäminen oli tuttua jo vanhastaan aiempien projektien ansiosta Kiholla, mutta mobiiliohjelmoinnista kokemusta oli varsin vähän. Uuden alueen oppimisen kannalta ja vapaasti suunniteltavan projektin ansiosta työ oli mieleinen. Projekti toimi myös samalla ponnahduslautana Kihon Android -sovelluksen pariin. Lisäksi uutena asiana oli suunnitella ja toteuttaa projektin aikataulutus ja määritellä tarvittavat resurssit projektia varten. Tästäkään itselläni ei ollut juurikaan kunnollista kokemusta, joten pääsin haastamaan itseäni myös näiltä osin.

1.2 Kiho yrityksenä

Mastercom Oy, eli tuttavallisemmalta aputoiminimeltään Kiho, on vuonna 2003 perustettu tietotekniikka -alan yritys. Aluksi yritys on tarjonnut laajalla skaalalla eri IT -palveluita, mutta telematiikka -järjestelmän kehittyessä yritys suuntautui ohjelmistokehitykseen. Kiho -brändi otettiin virallisesti käyttöön vuonna 2010. Kiho työllistää tällä hetkellä lähes 50 henkilöä kuudella ei paikkakunnalla, niin ohjelmistokehityksen, myynnin ja asiakaspalvelun parissa. Kihon kasvuvauhti on ollut viime vuosina hurjaa ja tästä johtuen yrityksen henkilöstömäärä on liki kaksinkertaistunut

parissa vuodessa. Tällä hetkellä Kihon päätuotteena on älykäs kenttätyönhallinta, johonka kuuluvat muun muassa työajanhallinta, älykkäät telematiikka –ratkaisut, sekä Kiho Cloud Platform -alusta, joka tarjoaa muille palveluntarjoajille älykkään palvelun kehittämisratkaisuja. Kuvan 1 mukaisesti, kenttätyönhallinta sisältää muun muassa kaluston hallinnan, työajanseurannan, tehtävien hallinnan ja erilaisia raportointimahdollisuuksia.



Kuva 1. Kihon liiketoiminnan ja rajapintojen kuvaus (Kiho, s.a.)

1.3 Projektin tavoitteet

Työn tavoitteina oli luoda toimiva sovelluskokonaisuus Kihon –laiteasennusten helpottamiseksi, käyttäen jo valmiita ja luoden uusia rajapintoja. Sovellus itsessään on Android –toteutus, joten tekijän tulee opiskella ja hallita Android –ohjelmoinnin perusteet. Sovelluksen toiminnalle asetettiin myös omat tavoitteet, kuten helppokäyttöisyys ja nopea käytettävyys. Lisäksi projektin keskeisimpänä osana oli myös resurssien ja aikataulun suunnittelu. Aikataulun ja resurssien toteutumista seurattiin projektin aikana tiiviisti.

Kaikkien näiden tavoitteiden päämääränä oli saada laajamittaista kokemusta sovelluskehityksen osalta tekijälleen. Samalla yritys hyötyy tehokkaammasta asennusprosessista, jolloin työtunteja ja yrityksen resursseja ei valu hukkaan niin paljoa.

2 PROJEKTIN ALOITUS JA SUUNNITTELU

2.1 Projektin aloitus ja suunnittelu

Projektin aloituksessa keskeisimpiä tekijöitä ovat projektin raamien hahmottelu, käytettävien työkalujen valitseminen ja projektin suunnittelu, johon kuuluvat ajankäytön ja muiden resurssien jakaminen. Tässä projektissa päävastuu projektin suunnittelusta annettiin itse työn tekijälle.

Projektissa lähtökohtana oli mahdollisimman helppokäyttöisen ja selkeän sovelluksen luominen. Sovelluksen tulisi hoitaa tehtävänsä monissa eri olosuhteissa aina lapin takametsistä toimistokäyttöön, joten sovelluksen suunnitteluun ja testaukseen päätettiin käyttää enemmän resursseja kuin normaalisti.

Projektin suunnittelu ei ollut allekirjoittaneelle sinänsä uutta. Mobiiliohjelmointi taas oli alueena varsin tuntematon, mikä haastoi ajattelemaan kokonaisuutta ja sen toimivuutta entistä enemmän. Mikään yksittäinen päätös ei siis ollut lähtökohtaisesti itsestäänselvyys vaan vaati joka kerta arviointia, jotta lopputuloksesta tulisi mahdollisimman hyvä.

2.2 Projektissa käytettävät tekniikat ja työkalut

2.2.1 Yleistä suunnittelua

Projektin suunnittelussa lähtökohdiksi otettiin ensisijaisesti pitkälle tulevaisuuteen toimivan sovellusarkkitehtuurimallin etsiminen, jotta sovellus olisi hyvin jatkokehitettävä ja helposti ylläpidettävä. Sovelluksen tärkeänä lähtökohtana oli myös valita mahdollisimman uudet ja muuhun järjestelmään sopivat tekniikat ja menetelmät.

Koska sovellus oli tarkoitus toteuttaa Android -käyttöjärjestelmälle, eikä tarvetta nähty muille alustoille, päätettiin sovellus toteuttaa niin sanottuna natiivisovelluksena Androidille. Natiivisovellus eroaa niin kutsutuista cross-platform, eli usealla eri ohjelmistoalustoilla toimivasta toteutuksesta siten, että ohjelmisto on suunniteltu suoraan kyseenomaiselle alustalle. Tällöin eri alustojen välisiä kompromisseja ei tarvita, ylläpito ja kehitys on helpompaa ja sovelluksen toimintavarmuus on selvästi parempi. Toimintavarmuus perustuu ohjelmiston kokonaisuuden yksinkertaisuuteen, eikä sovelluksessa tarvitse ottaa muiden alustojen tarpeita huomioon. (YML 2020)

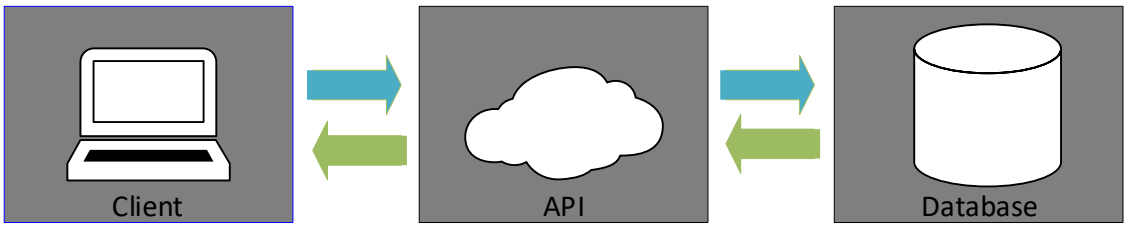
Android -sovelluskehityksessä ei ole kovin paljon vaihtoehtoja kehitystyökalujen suhteen, joten kehitys tapahtuu monesti Android Studion pohjalta, joka on ollut ja on edelleen kehittyneinpiä Android –sovelluksille toteutetuista kehitysalustoista. (Slant.co 2020; Melnichuk 2020.) Android Studion kehittäjinä ovat Google, joka vastaa Androidin pääkehityksestä, sekä JetBrains, joka puolestaan vastaa koodieditorin kehityksestä. (Google 2020)

Sovelluksen ohjelmointiin olisi voitu käyttää useita eri ohjelmointikieliä. Näistä yleisimmät ovat tällä hetkellä Oraclen kehittämä ja ylläpitämä Java, sekä Googlen kehittämä Kotlin. Kotlinin ja Javan tulevaisuutta on kokonaisuutena vaikea arvioida. Oletettavaa on, että Google siirtyy hiljalleen Jetbrainsin kehittämän Kotlinin käyttöön, Javan jäädessä taka-alalle. Tässä projektissa päädyttiin käyttämään kuitenkin Javaa, sillä yrityksen omat aiemmat Android –kirjastot ovat Java –pohjaisia. Lisäksi projektin tekijällä ei ollut projektin alkuvaiheessa tarpeeksi kokemusta Kotlinista, kuten ei

myöskään yrityksen muilla mobiiliohjelmoijilla, joten Kotlinin käyttö koettiin vaikeaksi. (Google 2020; Kotlin 2020.)

2.2.2 REST-API -rajapinnat

Kihon palvelinohjelmisto on toteutettu PHP –ohjelmakoodia käyttäen ja toteuttaen REST-API –arkkitehtuurimallia. Alkuperäiseen suunnitelmaan ei varsinaisesti sisältynyt API –suunnittelua ja toteutusta, mutta projektin aikana kävi selväksi, että projekti ei tulisi etenemään ilman API –muutoksia ja lisäyksiä.



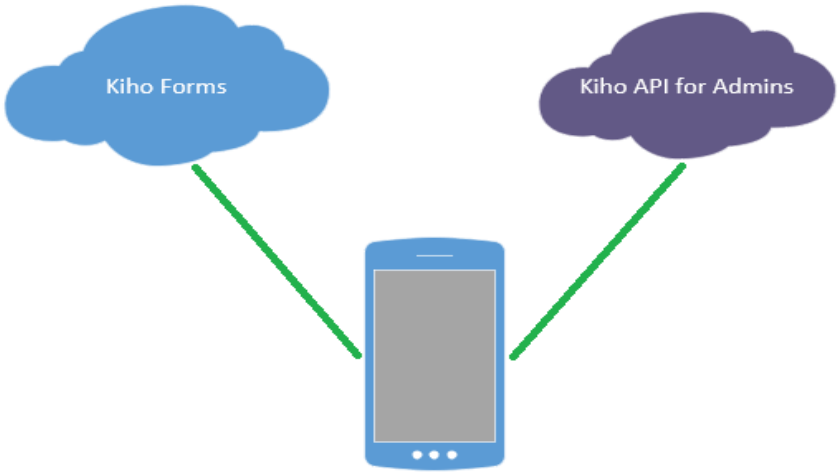
Kuvio 1. REST-API:n toiminta.

Kuvaajan mukaisesti REST-API mallissa Client, eli käyttäjän laite kommunikoi palvelimen kanssa HTTP -protokollaa käyttäen. Palvelimelle lähetetään erityyppisiä pyyntöjä, joista yleisimmät ovat;

- GET, käytetään hakiessa tietoa palvelimelta
- POST, käytetään uuden datan lisäämiseen palvelimelle
- PUT, käytetään, kun olemassa olevaa dataa muokataan
- DELETE, poistettaessa käytettävä pyyntö

Palvelin käsittelee tulevan pyynnön ja tarvittaessa käsittelee lähtevää ja tulevaa dataa. Tämän lisäksi palvelin tallentaa tai hakee dataa tietokannasta ja valvoo pääsyä tietokantaan. Tämän projektin tapauksessa Clienttina siis toimii mobiililaite ja Android -sovellus joka kommunikoi HTTP –protokollan mukaisesti API:n kanssa. (Mozilla 2020)

Sovelluksen perusvaatimuksia mukaillen, sovellus kommunikoi Kiho API:n kanssa kahteen eri paikkaan (Ks. kuva 2.). Kiho Lomakkeet –sovellus on osa Kihon tuoteperhettä, jonka avulla voidaan tehdä täysin asiakkaan käyttöön kustomoitu lomakepohja, jonka voi täyttää WEB –sivulta tai mobiilisovelluksesta. Vastauksia taas voidaan seurata ja analysoida jälkikäteen. Lomakkeet ovat myös helppokäyttöinen ohjelmistoratkaisu laiteasennusten dokumentointiin. Asennuslomake on ennakkoon suunniteltu Kiho -järjestelmässä ja muotoiltu yrityksen sisäiseen ja ulkoiseen käyttöön sopivaksi. Sovellus poimii tiedot ja muotoilee valmiiksi täytettyjen tietojen pohjalta muun muassa laitteen kanavien ja työtilojen dokumentoinnin. Dokumentaatio on suunniteltu myös siten, että asiakkaalle on mahdollista toimittaa PDF –tiedosto tai paperinen versio, mikäli asiakas haluaa dokumentaation asennuksesta.



Kuva 2. Kuvaus sovelluksen kommunikoinnista rajapintojen kanssa

2.2.3 Komponenttisuunnittelu ja koodin laatu

Yhtenä olennaisimmista osista projektin suunnittelussa otettiin komponenttimainen suunnittelu. Ohjelmoinnissa siis keskitytään yksittäisten ohjelmaosien tekemiseen siten, että kyseen omaista sovelluksen osaa pysytään käyttämään mahdollisimman useassa paikassa, mahdollisimman monipuolisesti. Projekti onkin jaettu tasaisesti UI-komponentteihin ja ohjelmakoodia suorittavat luokat niin kutsutuiksi Service ja Utility-luokiksi, joita on helppo kutsua projektin eri osista.

Ohjelmakoodissa ja sen suunnittelussa pyrittiin käyttämään ”Clean Code” -periaatteita, jotta ohjelmakoodista tulisi mahdollisimman selkeää ja yksinkertaista. Periaatteena oli, että ohjelmakoodi olisi jo itsessään niin selkeää, että sitä ei tarvitsisi kommentoida ollenkaan. Tähän päästään siten, että ohjelmakoodissa esimerkiksi muuttujat ja ohjelmoijan antamat nimet ovat mahdollisimman selkeitä ja kuvaavia. (Gupta 2020)

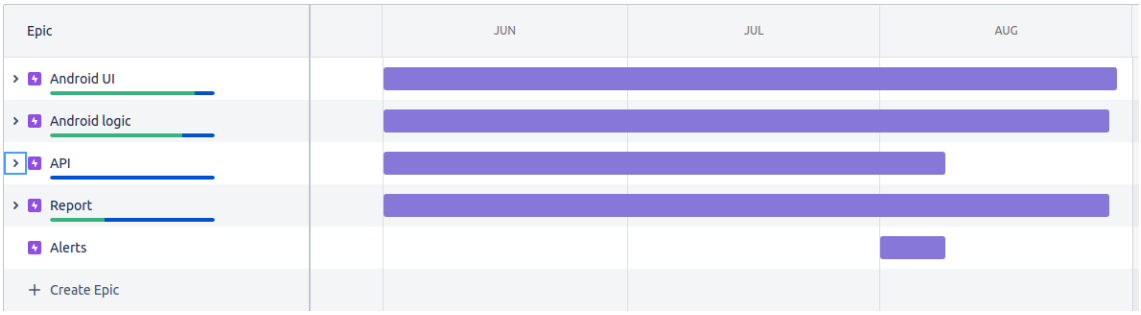
Nykyisin ohjelmakoodi –editorit ovat itsessään jo siinä määrin älykkäitä, että ne huomaavat mahdolliset epäkohdat ohjelmakoodissa. Tällaisia ovat esimerkiksi epämääräiset muuttujanimet, funktioiden tai luokkien virheelliset nimeämiset tai käyttämättä jääneet muuttujat tai luokat. Projektissa päätettiin androidin Analyzer -työkalua, joka löysi mallikkaasti ohjelmakoodin ongelmakohtia. Lisäksi editoriin oli määriteltynä erilliset koodisäännöt, jotka määrittävät esimerkiksi sen, missä järjestyksessä ylikirjoitettavat Android –ohjelman vakiofunktiot ovat tai sen, kuinka pitkiä jokaisen ohjelmakoodin muuttujan tulee olla. Näitä työkaluja käyttämällä luokkien yleisrakenteesta tuli mahdollisimman selkeä ja siisti, jotta muutkin ohjelmistokehittäjät saavat tarvittaessa selkoa ohjelmakoodista. (Google 2020)

Ohjelmakoodia suunniteltiin myös tarkistettavan yhdessä tietyin väliajoin muiden ohjelmistokehittäjien kanssa. Näin itse kehittäjän huomaamatta jääneet virheet jäävät helpommin kiinni ja selkeimmät kehityskohteet tulivat helpommin ilmi.

2.3 Projektin- ja resurssienhallinta

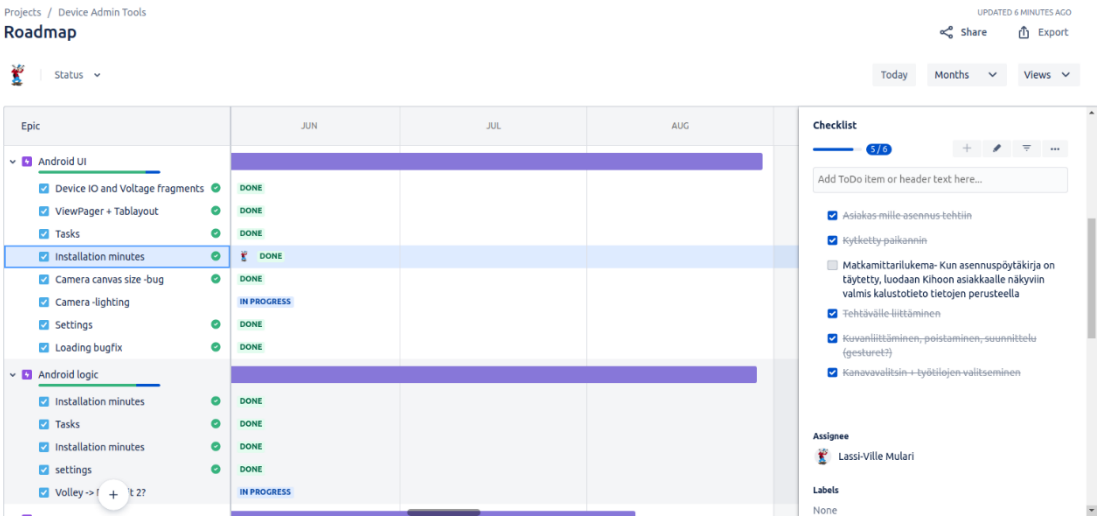
Projektin alustavaksi pituudeksi suunniteltiin kahdesta kolmeen kuukautta. Projektin hallintaan ja yleiseen suunnitteluun otettiin käyttöön Atlassian Softwaren Jira –ohjelmistopalvelu, johon viikoittaisia tehtäviä ja projektin eri vaiheita kirjailtiin. Samalla sovelluksella on myös helppo listata tekemättömiä ja tehtyjä tehtäviä, sekä ajastaa niitä kalenteriin sopivaksi. (Jira, s.a.)

Jirassa projektin voi jakaa eri segmentteihin, joita kutsutaan Epiceiksi. Itse projekti oli jaettu kolmeen eri pääsegmenttiin, eli UI –suunnitteluun, Androidin –logiikan ja rakenteen kehitykseen, sekä API –suunnitteluun ja toteutukseen. Lisäksi projektinhallintaan oli merkittynä raportointiin käytettävän materiaalin muistilista, sekä erillinen Alerts -osio jonne oli merkittynä palavereita ja muita tärkeitä muistettavia asioita.



Kuva 3. Projektin aikataulutus Atlassianin Jira -sovelluksessa.

Jokainen Epic oli jaettuna omiksi ala-aiheiksi (Child Issue) aihealueittain. Ala-aiheelle voi taas luoda oman dokumentaation, sekä muistilistan pienemmistä tehtävistä. Näin jokainen tehtävä voidaan pilkkoa pieniksi siivuiksi, jolloin itse tekemisestä tulee johdonmukaisempaa.



Kuva 4. Projektin ositus Atlassianin Jira -sovelluksessa.

Koska Jira on osa Atlassianin tuoteryhmää, oli projektissakin versionhallintajärjestelmänä Bitbucket, joka on nidottu kätevästi osaksi Jira -projektinhallintajärjestelmää. Yksittäisen Git commitin voi yhdistää suoraan oikealla tagilla yksittäiseen ala-aiheeseen tai Epic:iin, jolloin tehdyn koodin voi dokumentoida osaksi aikajanaa kätevästi.

Projektin resursseiksi määriteltäviä asioita ovat myös tarvittavat työvälineet, jotka olivat käytännössä jo olemassa. Suurin erillinen resurssi oli ehdottomasti sovellustestaajat, jotka iteratiivisesti kävivät sovelluksen toimintaa läpi ja antoivat palautetta sovelluksen eri toiminnoista ja ominaisuuksista. Koska sovelluksen käyttötarkoitus oli yrityksen sisäiseen käyttöön, oli testihenkilöidenkin oltava yrityksen työntekijöitä. Tämä aiheuttaa projektisuunnittelussa tiukat ehdot tietynlaiselle käyttäjäryhmälle ja asettaa omat haasteensa sovelluskehitykseen, mutta osaltaan myös helpottaa tuotekehitystä.

2.4 UI/UX -suunnittelu

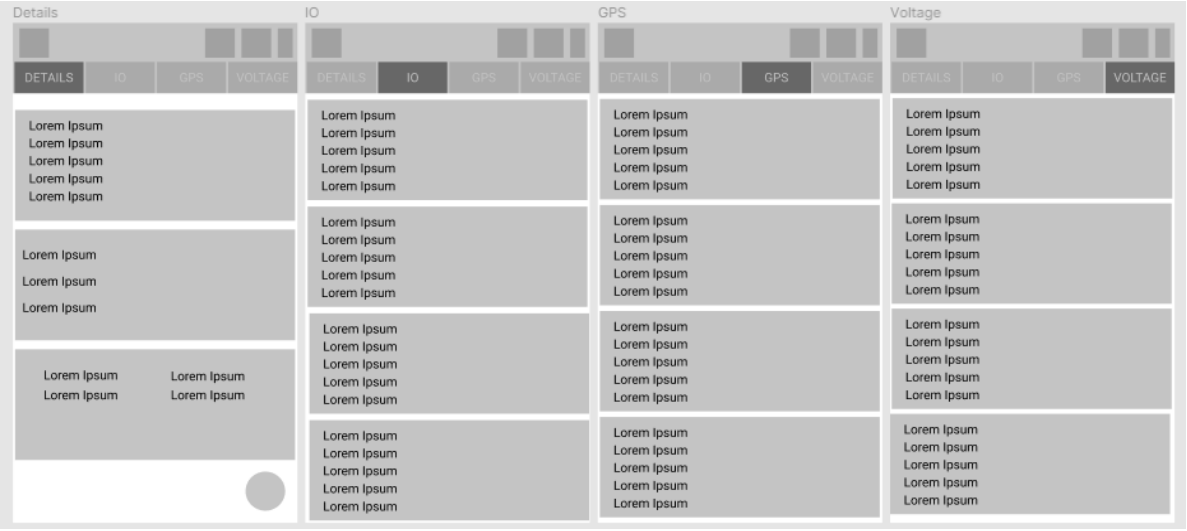
Käyttöliittymäsuunnittelussa lähdettiin liikkeelle niin sanotulla kolmen lähtökohdan periaatteella. Kolmessa lähtökohdassa pyritään miettimään mitkä asiat ovat tärkeimpiä, jotta käyttöliittymästä tulisi mahdollisimman hyvä loppukäyttäjän kannalta. Ottaen huomioon sovelluksen käyttötarkoitus ja käyttötavat, ovat kolme tärkeintä tekijää:

- Tehokkuus ja hyvät toiminnalliset edellytykset
- Selkeys ja siisteys
- Nopea ja helppokäyttöinen käyttöliittymä

Sovelluksen suunnittelulähtökohdat huomioiden ja nykyaikaisten mobiililaitteiden koon pohjalta oli lähdettävä miettimään erilaisia ratkaisuja.

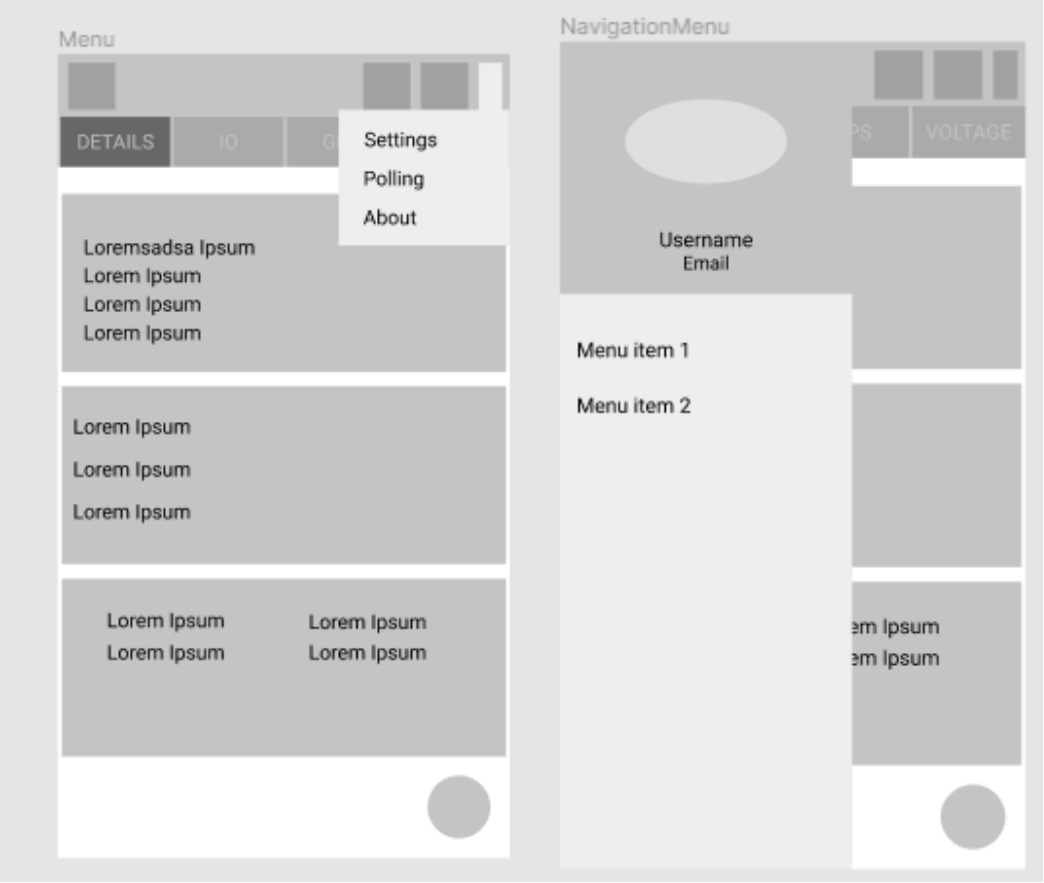
Suunnitelmavaiheen mallinnus toteutettiin Figma-vektorigrafiikkaeditorilla, joka on nopeakäyttöinen sovellus UI/UX -suunnitteluun. (Figma s.a.)

Sovellus on suunniteltu kaikilta osin käytettäväksi pystyssä. Puhelimen kääntöä kyljelleen ei nähty tarpeelliseksi, sillä sovelluksen käytössä kylkiasennosta ei ole juurikaan tietojen näytössä ja selaamisessa hyötyä, pois lukien kameranäkymä, jotta kuvista saisi horisontaalisia. Samalla yhden käden käyttö helpottuu, kun puhelimen näyttö ei kierrä automaattisesti eri tilanteissa sivuittain, kuten vaativien asennusprosessien yhteydessä monesti käy.
Samsung käyttää mobiilisuunnittelussaan käyttöliittymäratkaisua, jossa sovelluksen tärkeimmät navigoinnit ja toiminnot ovat sijoitettuna puhelimen alalaitaan, jolloin käyttöliittymä on helposti käytettävä yhdelläkin kädellä. Vastaavasti vähemmän käytetyt ominaisuudet ja staattiset –osiot ovat sijoitettuna sovelluksen yläosaan. Ylä- ja alaosan välinen osuus on aseteltu vaiheittain sopivan väljästi, jotta käyttäjälle on mielekästä käyttää sovellusta. Näin ollen sovellusta on helpompi käyttää myös yhdellä kädellä, eikä nykyaikaisilla, melko pitkillä mobiililaitteilla, ole vaikeuksia kurotella tarvittavien toiminnallisuuksien perässä. (Samsung 2020). Tämä on huomioitu myös alkuperäisessä suunnittelussa.



Kuva 5. Figma-mallinnus sovelluksen päävalikosta

Sovelluksen valikkoratkaisu noudattaa pitkälle samaa linjaa, kuin monissa muissakin mobiilisovelluksissa. Oikeasta ylälaidasta löytää sovelluksen keskeiset vaikuttavat tekijät, kuten asetukset ja ohjeet, sekä pikakomentoja.



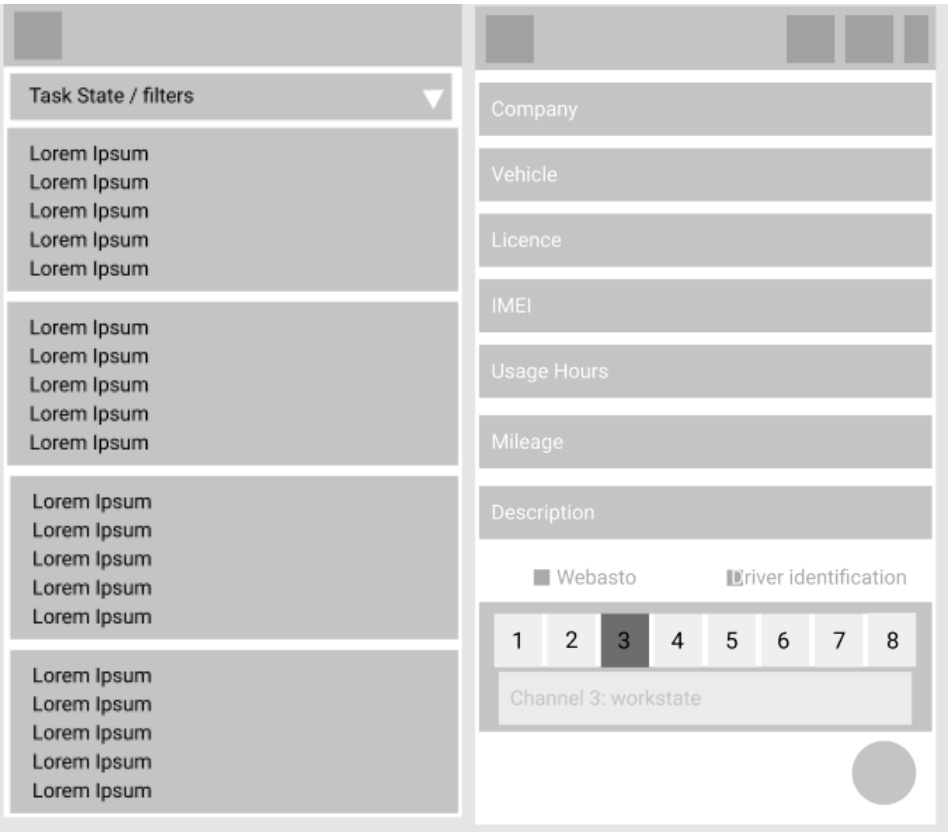
Kuva 6. Figma -mallinnus valikkorakenteesta.

Niin sanotusta navigointi –menusta taas löytyvät sovelluksen vähemmän käytetyt ominaisuudet ja käyttäjän tiedot.

Tähän ratkaisuun päädyttiin siksi, että sovellus noudattaa Kiho –sovelluksen käyttöliittymäratkaisua, joka on hyvin samankaltainen muiden markkinoilta löytyvien sovellusten kanssa. Sovelluksen

käyttöliittymää on myöskin turha muuttaa, sillä jo tutkitusti hyvät ratkaisut ovat yleensä käyttäjilleen tuttuja ja näin ollen helpompia käyttää. Lisäksi sovelluksen kehittäjä säästää aikaa ja resursseja, kun "pyörää ei tarvitse keksiä uudelleen".

Käyttöliittymän pääosa on jaettuna neljään eri lohkoon, joista ensimmäinen kertoo laitteen toiminnan oleellisimmat osat. Yhdellä silmäyksellä näkee siis, onko laite asennettu oikein ja toimiihan laite määritetyllä tavalla. Samasta välilehdestä on mahdollisuus tehdä laitteeseen ja Kiho -järjestelmään tarvittavat konfiguraatiot. Kolme muuta sivua taas täydentävät laitteen tietokokonaisuutta mahdollisten vikatilojen selvittämistä varten.



Kuva 7. Figma -mallinnus tehtävät -sivulta ja laitetallennuksesta.

Tehtävänäkymään suunniteltiin listaus annetuista työmääräyksistä, sekä erilaiset filtterit, jotta annettuja tehtäviä olisi helpompi hakea ja suodattaa.

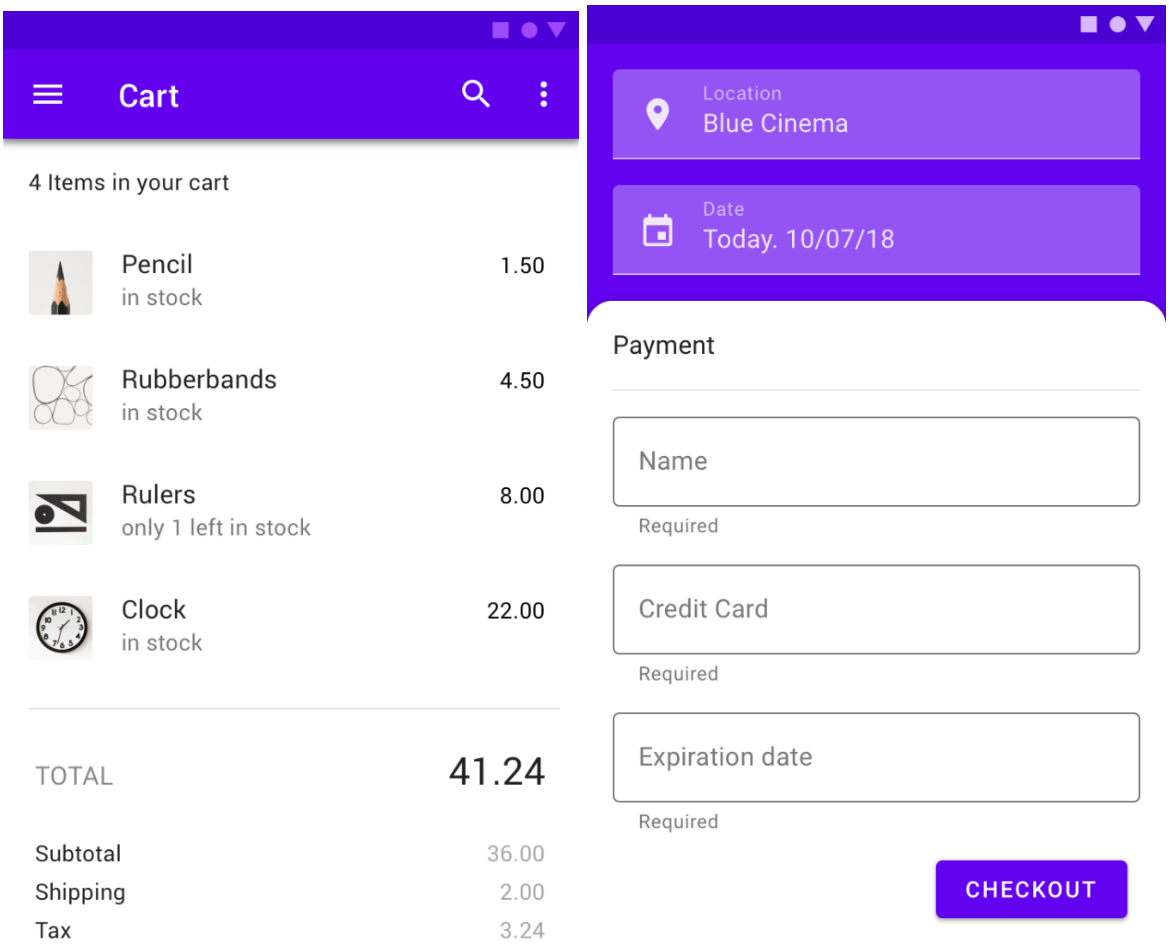
Sovelluksen tallentamissivulta löytyvät yksittäiset laitteen tallennustiedot. Tiedot ovat ryhmiteltynä siten, että käyttäjän on helppo tehdä tarvittavat muutokset laitteen ja järjestelmän tietoihin. Tiedot ovat pitkälti jo esitäytettyjä, joten laitteen tallennuksessa ei välttämättä tarvitse tehdä muita valintoja kuin kohdeyritys ja kalustovalinta. Lisäksi käyttäjä valitsee, halutaanko pelkästään tallentaa laitteen tiedot, dokumentaatio vai molemmat. Dokumentaatioon on myös mahdollista liittää kuvia laitteen asennuksesta tai ajoneuvon yleisestä tilasta, kuten digipiirturista.

Näkymiä ei vielä suunnitteluvaiheessa lyöty varsinaisesti lukkoon, vaan jokaisessa suunnitelmassa otettiin huomion mahdolliset lisätarpeet, jotka tulisivat ilmi vasta sovelluskehityksen aikana.

Lisäksi sovellukseen päätettiin lisätä Android 10 -version tuomana ominaisuutena päivä/yö -tilavaihdin. (Google 2020). Tämä vaati sovellukseen erillisen kontrastisuunnittelun, jota hiottiin käyttäjien palautteen pohjalta, sekä yhdessä Kihon UI/UX -suunnittelijan kanssa.

2.4.1 Käytettävät UI/UX -kirjastot

Sovelluksessa päätettiin käyttää Googlen kehittämää Material Design -kirjastoa, joka on muotokieleltään minimalistinen ja edustaa typografialtaan sveitsiläistä tyyliä. Material Designin toteutukset ovat nykyisin hyvin suosittuja sovelluskehittäjien keskuudessa selkeän muotokielensä ja tyylikkään olemuksensa ansiosta. (Google s.a.) Lisäksi lähes valmiit komponentit helpottavat sovelluskehitystä huomattavasti ja kyseiset komponentit on myös toteutettu IOS- ja Web -ympäristöihin tuomaan yhdenmukaisuutta. Google käyttää myös laajasti Material Design komponentteja vakio Android -jakeluissaan. (Google s.a.)



Kuva 8. Esimerkkikuvia Googlen Material Design -komponenteista ja esimerkki fontteja (Google Developers s.a.)

Kuten esimerkkikuvasta 6 voi huomata, Material Design komponentit edustavat muotokieleltään hyvin yksinkertaista ja selkeää kokonaisuutta. Pienillä asioilla, kuten lisäämällä sopiva ikoni, tai kuva listan eteen saadaan käyttäjälle aikaiseksi parempi mielikuva ostoskorin sisällöstä. Lisäksi fonttien koot, muotoilut ja asettelu selkeyttävät kokonaistypografiaa käyttäjälle ystävällisemmäksi.

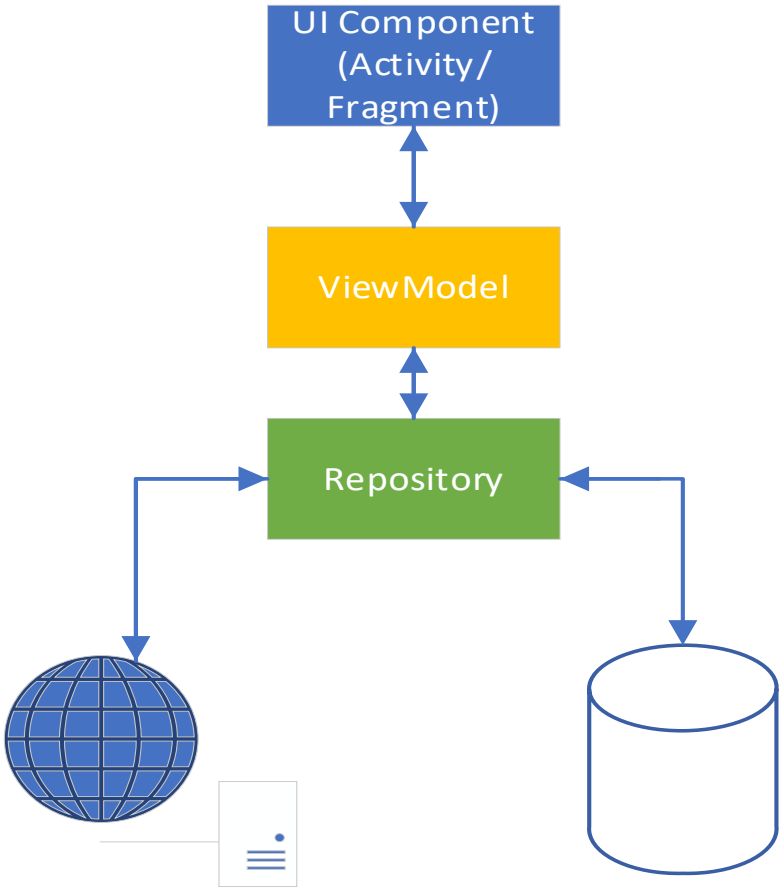
3 PROJEKTIN TOTEUTUS

3.1 Projektin aloitus

Projektia lähdettiin toteuttamaan suunnitelman mukaisesti. Sovelluskehitys oli alkuun kohtalaisen hidasta johtuen osin suunnitelman puutteellisuudesta, sekä tekijöistä, joita ei huomioitu kehitysvaiheessa. Ongelmia tuotti myös sovelluksen perusrakenteen kehitys, joka lähti alusta pitäen hieman väärään suuntaan. Projektin alussa lähdettiin toteuttamaan sovelluksen runko. Rungolla tarkoitetaan ei niinkään sovelluksen käyttäjälle näkyvää osuutta, vaan logiikkaa sovelluksen taustalla.

3.2 Sovelluksen näkymät ja rakenne

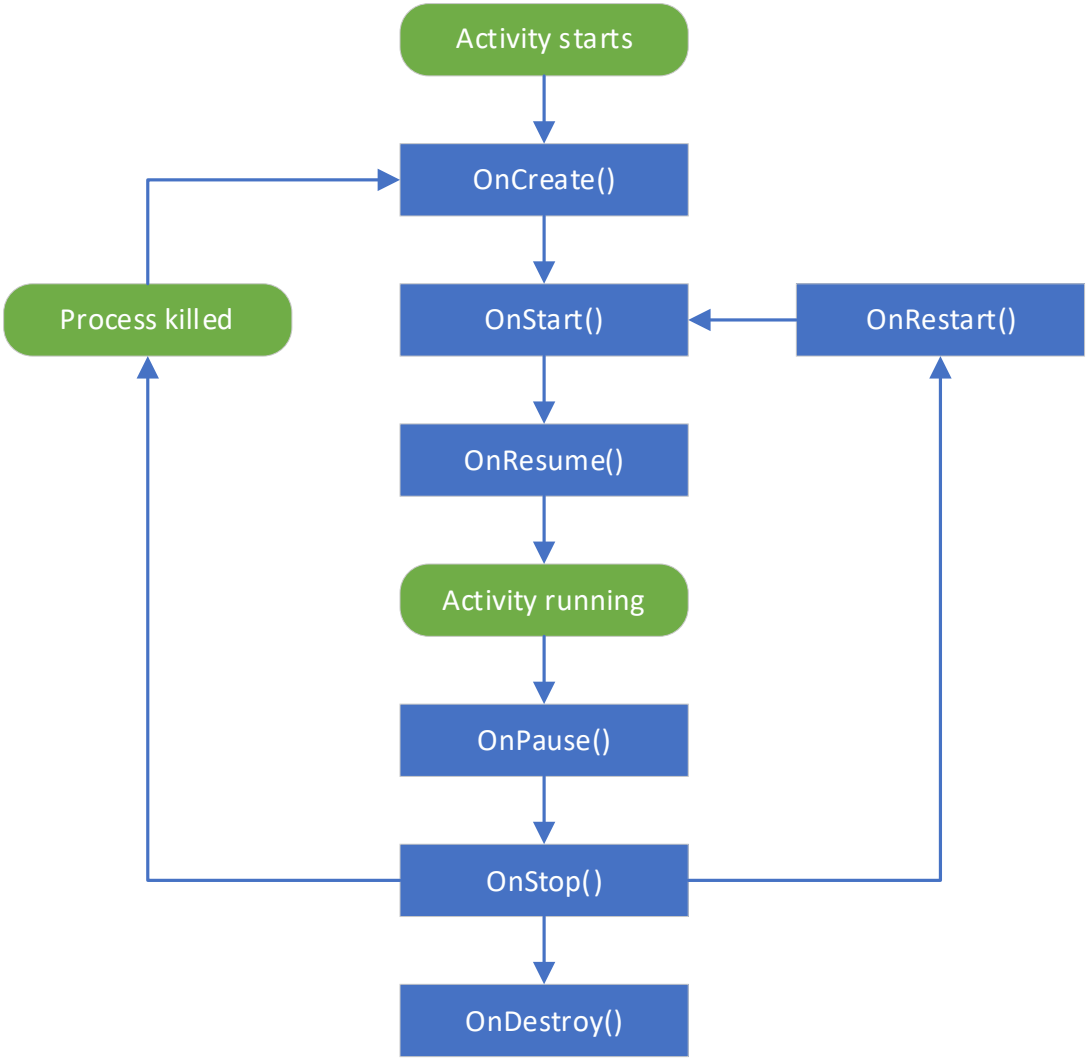
Projektin näkymät on toteutettu niin kutsutulla ViewModel –mallilla, joka on keskeinen suunnittelumalli Android -kehityksessä. ViewModel tarjoaa kommunkaatioväylän eri käyttöliittymäkomponenttien välillä ja ylläpitää dataa ja taustalogiikkaa taustalla. (Google 2020)



Kuva 9. Android ViewModel-mallinnus

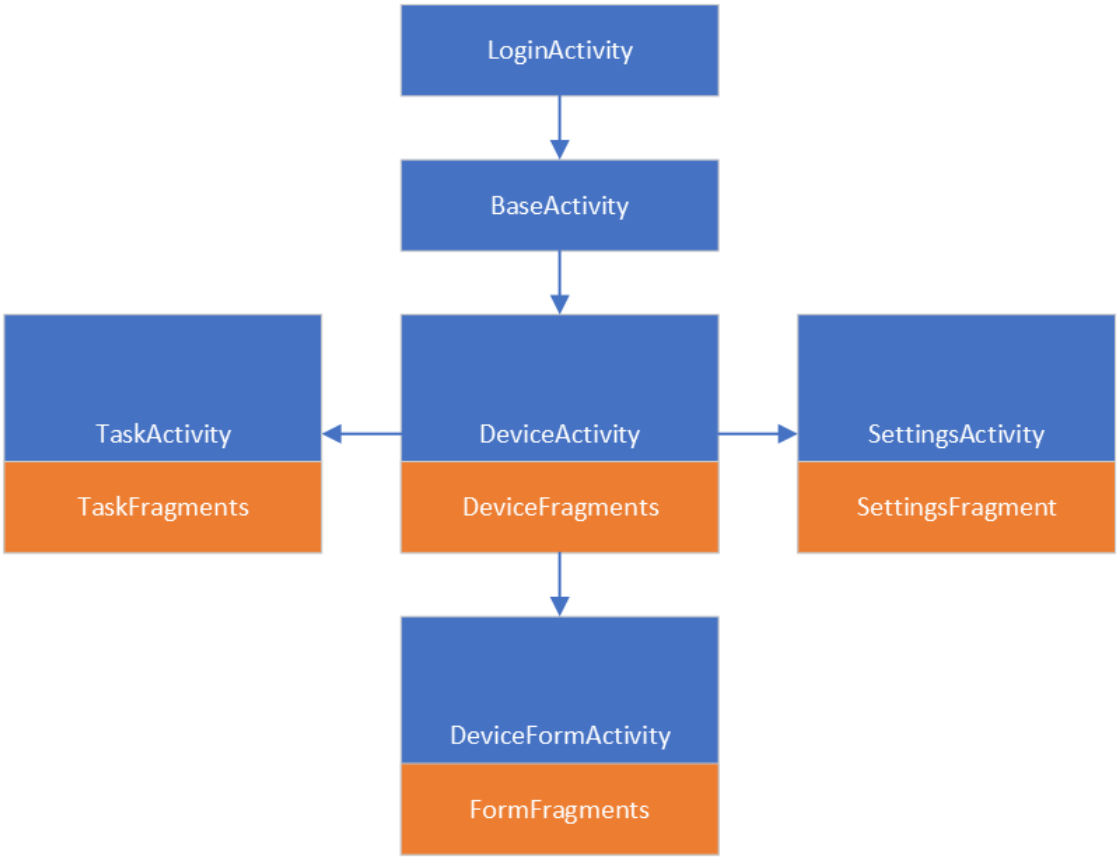
Ylimmästä laatikosta löytyvät käyttöliittymäkomponentit, jotka siis ovat osa käyttäjälle näkyvää näkymää. ViewModel ylläpitää dataa, jota tarkkaillaan käyttöliittymästä käsin. ViewModel taas kommunikoi Repositoryn kanssa, joka sisältää logiikkaa, jossa päätetään, haetaanko tietoa suoraan ulkoisesta muistista, kuten palvelimelta, vai haetaanko dataa suoraan laitteen sisäisestä muistista.

Kuten jokainen muukin android –sovellus myös tämä projekti toteuttaa androidin lifecycle järjestelmää. Andoid lifecycle on keskeisimpiä osia sovelluskehityksessä, koska se määrittää missä järjestyksessä asiat tapahtuvat ohjelmistossa. Samaan aikaan lifecycle tuo optiona ohjelmoijalle mahdollisuuden määrittää asioiden tapahtumisjärjestystä, mutta myös omat haasteensa sen suhteen, että järjestelmä toimisi oikea-aikaisesti. (Google 2020)



Kuva 10. Android Lifecycle:n kuvaaja.

Sovelluksen toteutunutta rakennetta voidaan esittää kuvan 11 kaaviolla, jossa rakenne on pilkottu activityihin ja activityjen alaisiin fragmentteihin hyvin karkeasti. Sovellus alkaa siis yksinkertaisesta kirjautumisnäkymästä, josta edetään niin kutsutun Base Activity –luokan läpi suoraan laitenäkymään. Base Activity luokka on niin sanotusti alemman tason luokka, joka ylläpitää sovelluksen läpi erinäisiä resursseja ja yleiskäyttöisiä metodeja, kuten autentikoinnin ylläpidon.



Kuva 11. Sovelluksen rakenne kuvattuna yleisellä tasolla.

Device Activity, eli laitteen perusnäkymä, koostuu neljästä eri fragmenteista, jotka ovat laitetietojen välilehtiä. Device Activitysta käsin taas voidaan selata tehtäviä Task Activityssa, joka koostuu myös pienemmistä fragmenteista, tai sovellusasetuksia. Laitenäkymästä pääsee myös laitteen asetusten tallennus ja- asennuslomakkeen lähetys näkymään eli Device Form Activityyn, joka on hajautettu lukuisiin eri komponentteihin ja fragmentteihin.

3.3 Android - ja sovellusversiointi

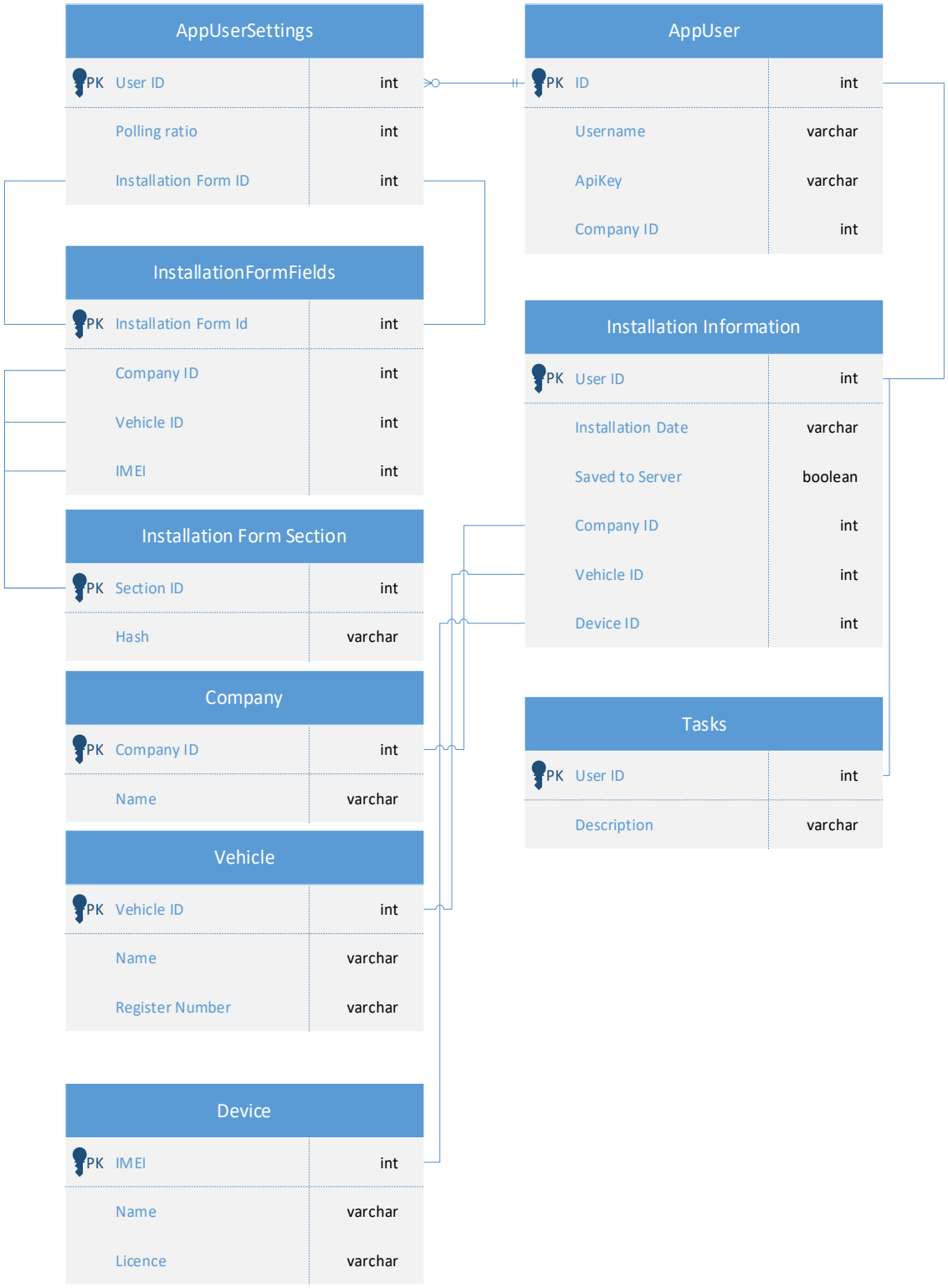
Android sovelluksissa määritellään sovelluksen toimivuus Android –versioittain. Sovellukselle voi siis määrittää androidin minimiversion, jolloin käytettävän Android –version tulee olla määritetty tai uudempi. Tässä projektissa päädyttiin käyttämään minimi API –levelinä versionumeroa 26, jota käyttää karkeasti raportin kirjoittamishetkellä reilusti yli 60% koko androidin käyttäjäkunnasta (taulukko 1). Koska sovellus ei myöskään ole varsinaisesti asiakaskäytössä vaan yrityksen sisäisessä käytössä, on sovelluksen yhteensopivuus vanhempien laitteiden kanssa suhteellisen tarpeetonta. Samalla kehityksessä voitiin käyttää uusimpia tekniikoita, eikä vanhaa tarvinnut huomioida ja ylläpitää niin paljoa.

Taulukko 1. Android versioiden jakautuminen käyttäjien kesken. Tieto oli tuorein projektin aloituksen aikaan. (Rahman 2020)

Android Alusta Versio (API Level)	**Jakauma (10.4.2020)**
Android 4.0 "Ice Cream Sandwich" (15)	0%
Android 4.1 "Jelly Bean" (16)	1%
Android 4.2 "Jelly Bean" (17)	1%
Android 4.3 "Jelly Bean" (18)	0%
Android 4.4 "KitKat" (19)	4%
Android 5.0 "Lollipop" (21)	2%
Android 5.1 "Lollipop" (22)	7%
Android 6.0 "Marshmallow" (23)	11%
Android 7.0 "Nougat" (24)	8%
Android 7.1 "Nougat" (25)	5%
Android 8.0 "Oreo" (26)	7%
Android 8.1 "Oreo" (27)	14%
Android 9 "Pie" (28)	31%
Android 10 (29)	8%

3.4 Sovelluksen datan käsittely ja tallentaminen

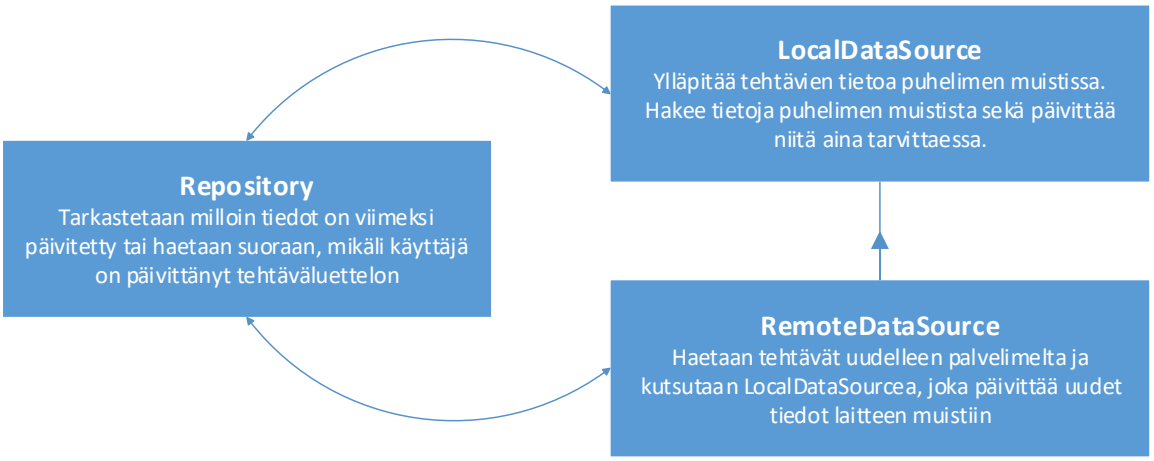
Androidissa tiedon tallentaminen muistiin on ollut viime vuosiin asti melko haasteellista. Nykyisellään Android –kirjastot antavat suoran tuen SQLite –tietokannan hallintajärjestelmälle, joka tarjoaa mahdollisuuden tallentaa dataa relaatiomallisesti suoraan "sovelluksen kylkeen". Näin ollen erillisiä tallennusratkaisuja ei tarvita. Myös tässä projektissa päädyttiin käyttämään SQLite -tietokantaa laitteen sisäiseen tallennukseen. (SQLite 2020)



Kuva 12. Sovelluksen tietokantarakenne kuvattuna yleisellä tasolla.

Kuvassa 12 on kuvattuna sovelluksen tietokantarakenne. Tietokannan kuvaus ei ole täydellinen suuren kokonsa vuoksi, eikä taulujen ja kenttien nimet vastaa oikeaa toteutusta. Sovelluksen sisäiseen muistiin siis tallennetaan siis käyttäjä, sekä käyttäjän asetukset. Näin käyttäjän ei tarvitse kirjautua joka kerta sisään ja tallentaa asetuksia uudestaan, vaan sovellus toimii siitä mihin viimeisellä käyttökerralla jäätiin. Käyttäjän asetuksiin tallentuu myös lomakekenttien tiedot, jotta lomaketta tallentaessa palvelimelle osataan yhdistää oikea kenttä oikeaan tietueeseen.

Koska tehtävien datamäärät ovat melko suuria, tallennetaan käyttäjän tehtäviä puhelimen muistiin. Näin ollen joka kerta tehtäviä tarkastellessa tietoa ei tarvitse hakea ensiksi palvelimelta vaan ne löytyvät kätevästi muistista. Tehtäviä ja niiden tietoja päivitetään aiemmin esitellyn View Model -logiikan mukaisesti.



Kuva 13. Tiedon hakeminen ja käsittely lokaaliin tietokantaan.

Vastaavaa tietojen tallennusmallia käytetään sovelluksessa myös muun muassa yrityslistauksen hakemisessa.

Lisäksi sovellukseen on toteutettu jo tässä vaiheessa täydellinen laitetallennuksen taulu jatkokehitystä varten. Tulevaisuudessa on aikomus tallentaa asennetut laitteet ja niiden tiedot suoraan sovellukseen, jälkilähetystä varten. Tämä mahdollistaa siis laiteasetusten ja asennuslomakkeen tallennuksen jälkikäteen, esimerkiksi huonojen mobiiliyhteyksien takia.

3.5 Erilliskirjastot

Kuten myös monessa muussakin projektissa, tässäkin pyrittiin hyödyntämään mahdollisimman paljon jo valmiiksi tehtyjä koodikirjastoja, jotta projektin tekemisessä säästettäisiin mahdollisimman paljon aikaa.

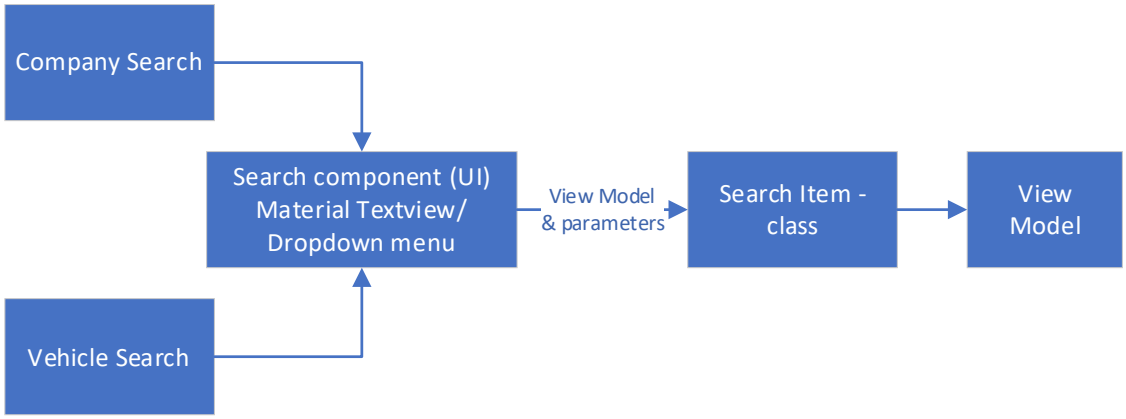
Koska Java-, sekä Androidin vakiokirjastoista ei löytynyt tarpeeksi laajamittaista aikaleima –konvertteria, joka tukisi hyvin laaja-alaisesti erilaisia aikaformaatteja, oli projektissa käytettävä ulkoista JodaTime –kirjastoa. JodaTime tarjoaa kattavat mahdollisuudet muuntaa erityyppisiä aikaleimoja, kuten ISO8061 –aikaleimoja suomalaiseen PP.KK.VVVV -formaattiin. (Joda-Time 2020)

3.6 Sovelluksen komponentit

Kuten suunnitteluvaiheessa oli tarkoitus, sovelluksen eri osia pyrittiin suunnittelemaan siten, että ohjelmakoodi toimisi monessa paikassa samalla tavalla, jolloin sovelluksessa ei tarvitsisi olla useita hyvin samantyyppisiä luokkia ja metodeja.

Parhaana esimerkkinä sovelluksesta löytyy sivunumeroitu hakukenttä, jota käytetään muun muassa yrityslistauksen ja ajoneuvojen haussa. Hakukentän UI on toteutettu Material Designin Alasvetovalikon pohjalle. Komponentti voidaan alustaa mihinkä Fragmenttiin tai Activityyn tahansa.

Komponentille vain kerrotaan käytettävä resurssi ja ViewModel, jolloin hakuluokka osaa automaattisesti kysyä oikealta View Modelilta tiettyä resurssia, joka puolestaan tekee kutsun eteenpäin kuvan 9 mukaisesti.



Kuva 14. Esimerkkikuvassa esitellään kahden erillisen hakukentän toimintaa. Taustalla toimii pitkälti sama logiikka, mutta käytettävät parametrit ovat erilaisia.

View Model löytäessään oikean resurssin palauttaa haetun datan samaa reittiä aina UI -komponentille saakka. Hakukenttä toimii niin hakusanoilla tai listaa selaamalla. Listalla näytetään muistin käytön takia ainoastaan 100 kappaletta hakukohteita, joten selaamalla listan loppuun API:lta haetaan seuraavat 100kpl hakutuloksia listan jatkeeksi. Sivunumeroitua hakua tosin tuskin tarvitaan, mutta kyseinen komponentti on samankaltainen mitä Kihon web -toteutuksissa.

3.7 Testausiteraatiot

Projektin aikana sovellusta testattiin ahkerasti aina alkumetreiltä lähtien. Testaajina toimivat yrityksen työntekijät ja muutamia ulkopuolisia henkilöitä, joilla testattiin käyttöliittymän toimivuutta. Testauksessa käytettiin käyttäjälähtöistä testimenetelmää, jossa käyttäjälle kerrottiin ainoastaan, mitä sovelluksella pystyy tekemään ja käyttäjän tehtäväksi jäi itse selvittää, kuinka sovellus toimii. Lisäksi käyttäjälle kerrottiin aluksi käyttäjätunnukset ja oikeaa asennustilannetta simuloiva tehtävä, jossa käyttäjän tuli etsiä oikealta tehtävältä asennuksen tiedot ja suorittaa vaaditut asennusvaiheet. Oheisessa kuvassa on esiteltynä sovelluksen tehtävät -näkymästä kyseenomainen testitehtävä ensimmäisestä testausiteraatiosta.

Kuva 15. Testausiteraation testitehtävä, jossa selitettynä käyttäjälle mitä sovelluksella tulisi tehdä.

Testausvaiheessa olleet testikäyttäjät olivat ikähaarukaltaan noin 20-40 vuotiaita ja heitä oli kaikkiaan 11 henkilöä. Osa käyttäjistä olivat yrityksen sisältä ja yrityksen ulkopuolisista testaajista suurin osa tiesi Kihon toiminnasta jotakin, minkä ansiosta testikäyttäjät pääsivät nopeasti kiinni sovelluksen toimintaan ja sen perusideaan.

Testien aikana huomiota kiinnitettiin ennen kaikkea siihen, mitä vaiheita käyttäjä ei oivalla ja mitkä asiat tuottavat käyttäjälle selkeästi töitä ja vaikeuksia. Sen vuoksi osa emulaattorilla tehdyistä testeistä eivät olleet hyviä, sillä käyttäjällä ei ollut fyysistä laitetta käsissään, minkä vuoksi käytettävyyden testaaminen oli pakko tehdä oikealla laitteella.

Testauksen aikana kävi selväksi, että helppokäyttöisyys ja selkeys toteutuivat melko hyvin jo alusta alkaen. Selkeimpiä ongelmia alkuperäissuunnitelmassa oli kartan puuttuminen, josta näkee käyttäjän ja valitun laitteen, sekä välilehtipalkin toivottiin olevan paremmin käsillä sovelluksen alalaidassa. Lisäksi itse asentajalta ja tekniseltä tuelta tuli muutamia pienempiä toiveita, mitkä liittyivät ennen kaikkea toimintalogiikkaan, ei niinkään käyttöliittymään.

4 LOPPUTULOS

4.1 Lopputuloksen kuvaaminen yleisellä tasolla

Projektia voidaan luonnehtia kohtalaisen onnistuneeksi kokonaisuudeksi. Sovelluksen asetetut vaatimukset täyttyivät käyttäjätestauksessa hyvin ja sovelluksen käyttö voidaan aloittaa tuotantoversiona piakkoin.

Sovelluksen UI/UX-kokonaisuudesta tuli selkeä muutamien testaus- ja kehitysiteraatioiden jälkeen. Sovellus on riittävän selkeä, jotta uusikin käyttäjä oppisi käyttämään sovellusta suunnitellulla tavalla. Lisäksi sovelluksen käyttö onnistuu uusilla puhelimilla hyvin sekä yhden ja kahden käden käytössä.

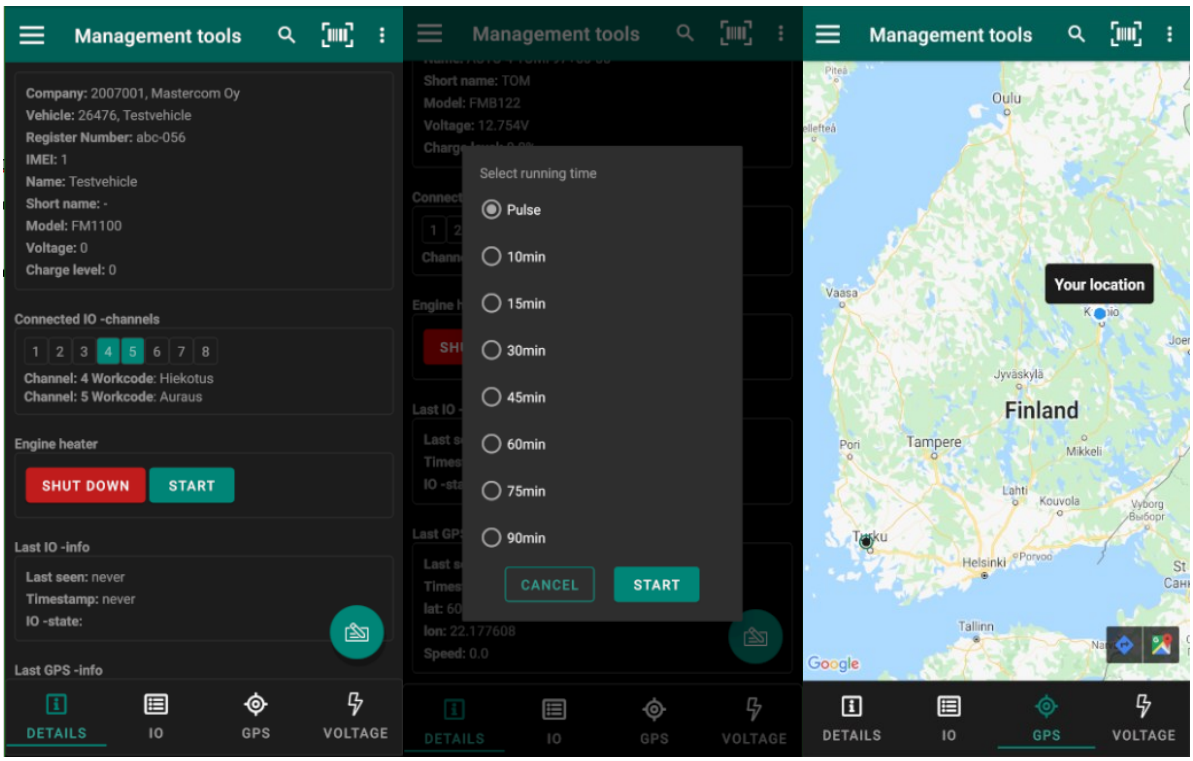
Sovelluksen komponenttirakenteeseen jäi vielä hieman toivomisen varaa. Osa komponenteista on suunnitellusti yksinkertaisia ja helposti käytettäviä, mutta esimerkiksi osa fragmenteista olisi ollut pilkottavissa pienemmiksi palasiksi, jotta yksittäisien luokkien koko ei olisi kasvanut järisyttävän suuriksi. Näiltä osin pieniä korjauksia on tulossa.

Lisäksi Local Data Sourcea olisi voinut jouduttaa esimerkiksi ajoneuvolistausten hakemisessa, jolloin hakeminen olisi ollut hieman nopeampaa, sekä API -pyyntöjen määrä olisi hieman vähentynyt.

Sovelluksen perusrakenne pysyi projektin aikana hyvin suunnitellun kaltaisena. Testi –iteraatioiden aikana huomattiin muutamia käytettävyyteen ja ominaisuuksiin liittyviä puutteita, jotka saatiin nopeasti korjattua.

4.2 Lopputuloksen yksityiskohtainen kuvaus

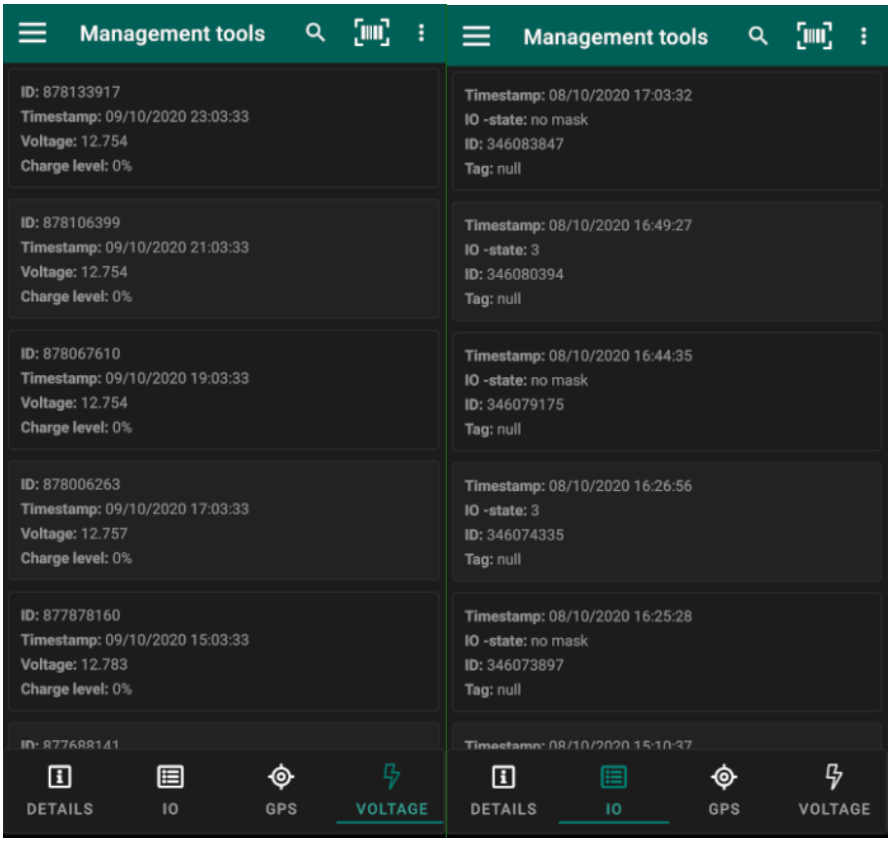
Laitteen tietojen tarkastelunäkymän yläpalkista löytyvät tarvittavat navigointimenetelmät, sekä laitehaut, kuten viivakoodinlukija. Laitenäkymä toteutettiin pitkälti myös alkuperäissuunnitelman mukaisesti, jossa oli lueteltuna laitteen perustiedot ja viimeisimmät tiedot laitteelta nopeaa tarkastelua varten. Testausiteraatioiden aikana tuli myös tarve lisätä testipainikkeet ajoneuvon moottorilämmittimen käynnistämistä varten, jotta asentaja näkee kytkentöjen toimivuuden suoraan paikan päällä. Moottorilämmittimelle on luotu vakioidut käynnissäoloajat, jotta vaikeita aikasäätimiä ei tarvitsisi lisätä sovellusnäkymään. Huomionarvoisimpana muutoksena alkuperäiseen suunnitelmaan oli myös välilehtipalkin siirtäminen näytön alalaitaan. Tähän ratkaisuun päädyttiin sen takia, että yksikätinen navigointi olisi helpompaa. Tämä asia korostuu etenkin karttanäkymässä, jossa sivuttainen liu’utus ei ole mahdollista kartan liikkumisen vuoksi.



Kuva 16. Sovelluksen päänäkymä kuvattuna ominaisuuksineen. (1/2)

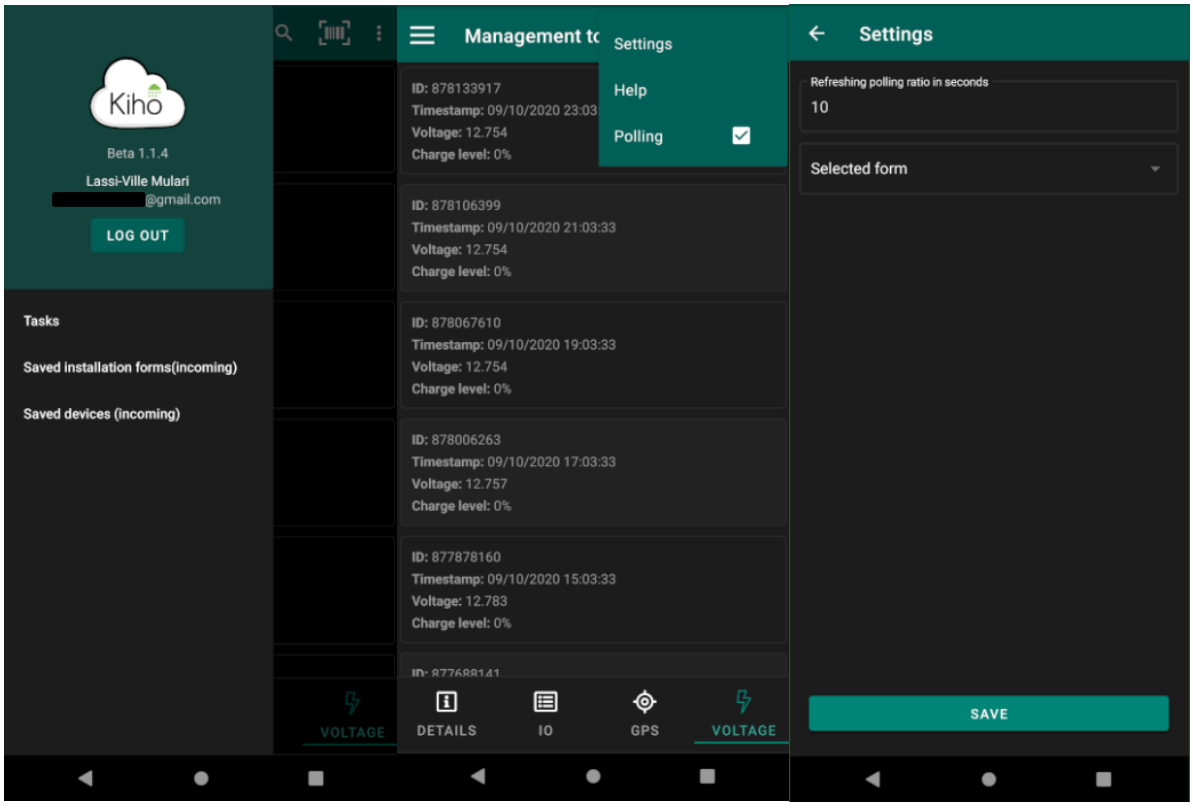
Kartta näyttää käyttäjän tämänhetkisen sijainnin ja laitteen 10 viimeisintä paikkapistettä. Näin laitteen sijainti on helppo tunnistaa. Laitteen paikkapisteet haalenevat aikajärjestyksessä uusimmasta vanhimpaan. Näin mahdolliset virhepaikannukset huomataan myös asennusvaiheessa. Laitteen paikkapistettä painamalla näkee paikkapisteen aikaleiman ja GPS –tiedot, kuten korkeuden merenpinnasta ja nopeuden.

Jännite ja IO –listaukset noudattavat tuttua listamaista esitystapaa, josta näkyvät 10 viimeisintä laitteelta tullutta tietoa. Kyseiset laitetiedot ovat tärkeitä etenkin vikatilanteiden ilmetessä (Kuva 16 2/2).



Kuva 16. Sovelluksen päänäkymä kuvattuna ominaisuuksineen (2/2)

Sovelluksen valikkoratkaisut noudattavat alkuperäissuunnitelmaa. Valikot ovat jaoteltuina kahteen eri valikkoon. Jälkikäteen ajateltuna Navigation -menun, eli oikeassa laidassa sijaitsevan valikon olisi voinut tiivistää yhteen vasemmanpuoleisen Drawer -menun kanssa.



Kuva 17. Sovelluksen valikkorakenteet ja asetukset.

Asetuksista on mahdollisuus vaihtaa päivitystaajuutta, jolla sovellus päivittää laitteen tietoja palvelimelta. Asetus on ennen kaikkea tärkeä katvealueilla. Tällöin dataa ei tarvitse hakea niin paljoa ja yhteys ei ole niin tukossa.

Lisäksi asetuksista löytyy valinta käytettävälle asennuslomakkeelle. Näin ollen tarvittaessa asennuslomake voidaan muotoilla uudelleen ja siirtyä käyttämään uutta lomaketta ilman, että sovellukseen tarvitsee tehdä muutoksia. Lomakkeiden kentät eivät sisällä muuta kuin nimen, joka näkyy kentässä, sekä geneerisen ID:n. Tästä johtuen kenttä on mahdoton yhdistää tiettyyn tietueeseen ja siksi päätettiin lisätä lomakkeen kuvaus -kenttään erillinen ID (taulukko 2).

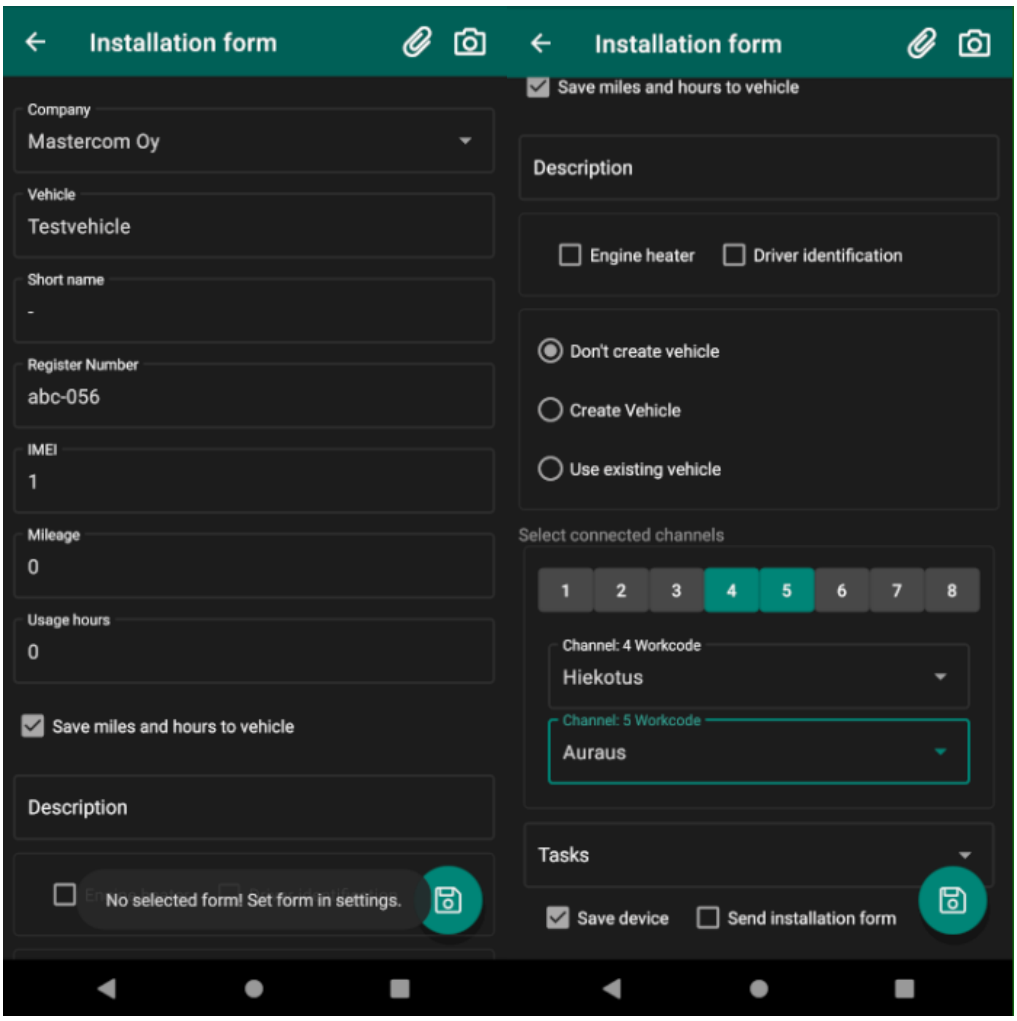
Tauluko 2. Laitetallennuksen kenttien kohdistaminen lomakkeen kenttien kanssa.

Sovelluksen kenttä	**ID (sijoitettu lomakkeen selite -kenttään)**
Yritys	#1111
Ajoneuvo	#1112
Rekisterinumero	#1113
IMEI	#1114

Lomakkeen kentät olisi ollut mahdollista kohdistaa myös sovelluksen asetuksista käsin yksitellen nimen perusteella. Tämä olisi vaatinut melko paljon klikkauksia mobiilissa ja lisätyötä lomakkeiden kehitykseen, joten kehityksessä todettiin helpommaksi määrittää kentät itse lomakkeelle.

Itse laitteen ja lomakkeen tallennussivulta löytyvät valinnat kohdeyritykselle, ajoneuvon ja laitteen nimeämiselle, lyhenteelle, rekisterinumerolle ja asennushetken kilometreille ja / tai käyttötunneille. Kuvauskenttä on tarkoitettu lomakkeen lisätieto -kentälle. Tallennussivulta saadaan valittua myös laitteen lisäominaisuudet, kuten moottorilämmitin ja kuljettajantunnistin. Asentaja pystyy myös asettamaan kytketyt kanavat ja halutut työtilat jokaista kanavaa kohti. Työtila näkyy lopuksi lyhenteen kanssa Kihon kartalla. Lisäksi lomake on mahdollista liittää halutulle tehtävälle, joten yksittäisen tehtävän tiedoista näkee mitä laitteita on asennettu, milläkin tiedoilla.

Jotta sovellus palvelisi myös teknistä tukea, on laitteelle annettu myös mahdollisuudet tallentaa joko laitteen asetukset, asennuslomake tai molemmat. Näin laitteen konfiguraatioita on mahdollista testata ongelmatilanteissa, kuten takuuvaihdoissa. Jatkossa laitteen voi siirtää ”kaatopaikalle” sovelluksella, eikä erillisiä muutoksia itse Web -kihoon tarvitse tehdä.



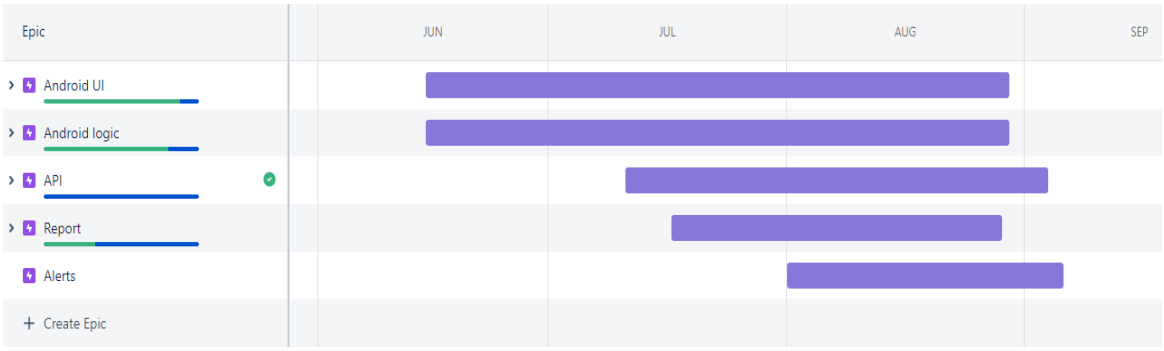
Kuva 18. Laitteen ja asennuslomakkeen tallennusnäkymä.

4.3 Jatkokehitys ja kehitysideat

Jatkokehityksen kannalta sovellus on jäänyt kohtalaisen hyvään tilaan. Tulevat toteutukset on helppo integroida nykyisen ratkaisun ympärille ja ne eivät vaadi suurien muutosten tekemistä jo olemassa olevaan ohjelmakoodiin. Projektin loppuvaiheilla testausiteraatioiden aikana ilmeni tarvetta muun muassa offline-toimintojen kehittämiselle, jotta asentaja voisi tallentaa lomakkeet ja laitteen tiedot puhelimen muistiin siksi aikaa, että pääsee katvealueen ulkopuolelle. Nämä ominaisuudet eivät kuitenkaan mahtuneet projektin aikatauluun, joten sovelluskehitys jatkuu tulevaisuudessa näiltä osin. Lisäksi sovelluksen asetuksiin toivottiin erillistä vaihdinta tumman ja vaalean teeman välille. Sovelluskehitys jatkunee osa-aikaisena muun työskentelyn ohella.

4.4 Aikataulu, resurssit ja niiden toteutuminen

Suunnitteluvaiheessa tehty aikataulutus ja projektin vaiheistus muuttui sovelluksen kehityksen aikana useasti. Moniin pieniin asioihin paloi huomattavasti suunniteltua enemmän aikaa. Ajankäytöllisesti voidaan todeta, että vanha ohjelmistopiireissä leviävä sanonta "90% toiminnallisuudesta valmistuu 20% varatusta ajasta. Loput 10% toiminnallisuudesta vie loput 80% ajasta". Tämä pitää melko hyvin paikkansa myös tässäkin projektissa.



Kuva 19. Toteutunut aikataulu Jira -järjestelmästä.

Projektin eri osa-alueille oli määritetty tietyt aikamääreet karkeasti jakaen. Android –kehitys laahasi alusta alkaen perässä aikataulusta mikä aiheutti kiirettä projektin loppuvaiheilla. Vaikka API –kehityksen tarve tulikin alkuperäissuunnitelmasta poiketen tarpeen, oli API:lle tehtävät muutokset kuitenkin nopeasti tehty ja siltä osin sovelluskehitys oli koko ajan aiottua nopeammassa aikataulussa. Sovellukseen suunnitellusta kiintiöajasta huolimatta aika ei riittänyt, minkä vuoksi kehitysaika-arviossa silti vieläkin petrattavaa ainakin aihepiireittäin. Toisaalta jatkossa perusasiat ovat paremmin hallinnassa, jolloin pieniin yksityiskohtiin ja perusrakenteisiin ei kulu yhtä paljon aikaa kuin tämän projektin aikana. Jokaisen Epicin alle kertyi runsaasti tikettejä, jotka loppupeleissä osoittivat projektin suurimman kehityksen painottuvan elokuun loppupuolelle.

Kokonaisuutena Jira toimi työkaluna hyvin. Koko projektin kulku saatiin raportoitua tehokkaasti Jiraan, josta jälkikäteen on nähtävissä projektin kehitysvaiheet.

5 YHTEENVETO

Projektin tarkoituksena oli helpottaa ja tehostaa Kihon sisäisen laitehallinnan ja asetusten kanssa tapahtuvaa työskentelyä. Samalla useita työvaiheita oli tarkoituksena karsia pois, jolloin yrityksen sisäisen työskentelyn tehokkuus kasvaa selvästi. Sovellus on projektin päättyessä valmis oikeaan käyttöön ja mahdollisuuksien mukaan sovellusta tullaan koeajamaan oikeassa käyttöympäristössään hetimmiten.

Sovelluksen kehitykselle ja toiminnalle asetetut tavoitteet täyttyvät ja ominaisuuksia saatiin jopa nidottua mukaan aiottua enemmän. Alustavasti sovelluksen käyttäjät ilmaisivat tyytyväisyytensä, joten kokonaisuutta voidaan pitää ainakin osittain onnistuneena. Sovelluksen lopullinen toimivuus nähdään heti oikean käyttötestauksen jälkeen.

Projekti oli kokonaisuutena tekijälleen mielenkiintoinen ja haastava. Suunnittelutyö oli helppoa kaikilta osin, pois lukien projektin aikataulutus, jota jouduttiin korjaamaan useaan kertaan. Alkuun mobiiliohjelmoinnin pienet haasteet hidastivat kehitystä paljon, mutta projektin edetessä edistymisvauhti kasvoi selvästi. API-kehitys luonnistui vastapainona nopeasti, vaikka alkuperäissuunnitelmassa ei API –kehitykselle nähty tarvetta.

Projektissa kertyi tekijälleen hurjasti uutta osaamista monelta uudelta osa-alueelta, joista mobiilikehityksestä eniten. Projektin jälkeen työt jatkuvat Kihon –mobiiliohjelmoinnin ja API:n parissa, joten projektin aikainen opiskelu ja kehitys heijastuvat myös tulevaisuuden tekemiseen yrityksen sisällä. Projektin aikana tekijälle selvisi myös hyvin omat heikkoudet sekä vahvuudet. Tältä pohjalta on hyvä lähteä jatkamaan uusien haasteiden pariin Kiholla.

LÄHTEET JA TUOTETUT AINEISTOT

Atlassian. (11. 10 2020). *Getting started with Jira tutorial: 6 basic steps*. Noudettu osoitteesta https://www.atlassian.com/software/jira/guides/getting-started/basics#step-2-pick-a-template

Atlassian. (s.a.). *Jira Documentation*. Noudettu osoitteesta https://confluence.atlassian.com/jira/jira-documentation-1556.html

Figma. (s.a.). Noudettu osoitteesta https://www.figma.com/

Google. (15. Huuhtikuu 2020). *Android Developers*. Noudettu osoitteesta https://developer.android.com/guide/components/activities/activity-lifecycle

Google. (28. 10 2020). *Dark theme*. Noudettu osoitteesta https://developer.android.com/guide/topics/ui/look-and-feel/darktheme

Google. (28. 10 2020). *Developers*. Noudettu osoitteesta Meet Android Studio: https://developer.android.com/studio/intro

Google. (30. 11 2020). *Kotlin*. Noudettu osoitteesta Kotlin: https://developer.android.com/kotlin

Google. (2020). *material.io.* Noudettu osoitteesta https://material.io/components/lists#anatomy

Google. (2020). *material.io.* Noudettu osoitteesta https://material.io/components/text-fields#usage

Google. (2020). *material.io*. Noudettu osoitteesta https://material.io/

Google. (30. Lokakuu 2020). *ViewModel architecture*. Noudettu osoitteesta https://developer.android.com/topic/libraries/architecture/viewmodel

Google. (2020. Heinäkuu 22). *ViewModel*. (Google) Haettu 12. Syyskuu 2020 osoitteesta https://developer.android.com/reference/androidx/lifecycle/ViewModel

Google. (s.a.). *Get started*. Noudettu osoitteesta https://material.io/resources/get-started#design

Google. (s.a.). *Introduction*. Noudettu osoitteesta https://material.io/design/introduction

Gupta, S. (25. 2 2018). *How to write clean code? Lessons learnt from "The Clean Code" — Robert C. Martin*. Noudettu osoitteesta Medium.com: https://medium.com/mindorks/how-to-write-clean-code-lessons-learnt-from-the-clean-code-robert-c-martin-9ffc7aef870c

Joda-Time. (Heinäkuu 2020). *Joda-time Userguide*. Noudettu osoitteesta https://www.joda.org/joda-time/userguide.html

Joda-TIme. (2020). *Why Joda Time*. Noudettu osoitteesta https://www.joda.org/joda-time/

Kiho. (Syyskuu 2020). *Kiho.fi*. Noudettu osoitteesta https://www.kiho.fi/

Kotlin. (2020). *Using Kotlin for Android Development*. Noudettu osoitteesta Kotlinlang.com: https://kotlinlang.org/docs/reference/android-overview.html

Melnichuk, A. (30. 11 2020). *Android IDEs for Developers*. Noudettu osoitteesta NCube: https://ncube.com/blog/android-ides-for-developers

Mozilla. (1. 2 2020). *HTTP request methods*. Noudettu osoitteesta https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods

Rahman, M. (10. 4 2020). Noudettu osoitteesta XDA-Developers: https://www.xda-developers.com/android-version-distribution-statistics-android-studio/

Rahman, M. (10. Huuhtikuu 2020). *Android Version Distribution statistics*. Noudettu osoitteesta https://www.xda-developers.com/android-version-distribution-statistics-android-studio/

Samsung. (2020). *Structure*. Noudettu osoitteesta https://developer.samsung.com/one-ui/arch/structure.html

Samsung. (2020). *Theme*. Noudettu osoitteesta https://developer.samsung.com/one-ui/arch/theme.html

Slant.co. (30. 11 2020). *What are the best IDEs for Android development?* Noudettu osoitteesta https://www.slant.co/topics/1321/~best-ides-for-android-development

SQLite. (2020). Noudettu osoitteesta https://sqlite.org/docs.html

statcounter.com. (17. Syyskuu 2020). *Android Market Share 2009-2020*. Haettu 17. Syyskuu 2020 osoitteesta https://gs.statcounter.com/os-market-share/mobile/worldwide/2015

Wikipedia. (s.a.). Noudettu osoitteesta REST: https://fi.wikipedia.org/wiki/REST

YML. (Elokuu 2020). *Native VS Hybrid Mobile Apps? The Answer is Clear*. Noudettu osoitteesta https://ymedialabs.com/hybrid-vs-native-mobile-apps-the-answer-is-clear