LAMK
Lahden ammattikorkeakoulu
Lahti University of Applied Sciences

# OFFLINE ENSIN -ARKKITEHTUURI MOBIILIKEHITYKSESSÄ

Offline-toteutus Android-alustalle

LAHDEN AMMATTIKORKEAKOULU
Liiketalouden ala
Tradenomi, tietojenkäsittelyn koulutus
Kevät 2019
Ville Pakarinen

## Tiivistelmä


| Tekijä(t) | Julkaisun laji | Valmistumisaika |
|---|---|---|
| Pakarinen, Ville | Opinnäytetyö, AMK | Kevät 2019 |
| | Sivumäärä | |
| | 28 | |


Työn nimi

**OFFLINE ENSIN -ARKKITEHTUURI MOBIILIKEHITYKSESSÄ**
Offline-toteutus Android-alustalle

Tutkinto

Tradenomi, tietojenkäsittelyn koulutus


Tiivistelmä

Opinnäytetyön tavoitteena oli offline ensin -ajattelumallin tutkiminen mobiilisovelluksien näkökulmasta. Opinnäytetyössä esiteltiin erilaisia suunnitteluperiaatteita ja työvälineitä, joita voidaan hyödyntää verkontilasta riippumattoman mobiiliratkaisun suunnittelussa ja toteutuksessa.

Opinnäytetyön teoriaosuudessa keskityttiin kuvaamaan, mitä verkkopohjaisella sovelluksella tarkoitetaan ja kuinka offline-suunnitteluperiaate sopii kyseiseen malliin. Teoriaosuudessa käsiteltiin erilaisia teknisiä toimintaperiaatteita ja työvälineitä, joita voidaan hyödyntää suunniteltaessa offline-ominaisuuksia Android-pohjaisille mobiililaitteille.

Opinnäytetyössä hyödynnettiin suunnittelutieteellistä tutkimusmenetelmää, jonka lopputuloksena on artefakti, jossa case-toiminnallisuuden määrityksiin perustuen suunniteltiin toimiva offline-toimintaperiaate. Offline-toimintaperiaatteen suunnittelua ohjasi case-toiminnallisuudelle asetut määritykset ja rajoitukset.

Käsiteltävän tapauksen pohjalta syntyi ratkaisu esiteltyyn ongelmaan, jossa hyödynnetään offline ensin -suunnitteluperiaatetta. Ratkaisua arvioitiin käytettävien suunnitteluperiaatteiden ylläpidettävyyden ja ymmärrettävyyden näkökulmasta. Tutkimuksen päätteeksi arvioitiin, kuinka suunniteltua toimintaperiaatetta voitaisiin jatkokehittää ja vastata hajautetun tietojärjestelmän synkronoinnin aiheuttamiin haasteisiin.



Asiasanat

mobiilikehitys, offline ensin, Android

## Abstract


| Author(s) | Type of publication | Published |
|---|---|---|
| Pakarinen, Ville | Bachelor's thesis | Spring 2019 |
| | Number of pages | |
| | 28 | |


Title of publication

**Offline-first Architecture in Mobile Development**
Offline Implementation on Android Platform

Name of Degree

Bachelor's Degree Programme in Business Information Technology


Abstract

The thesis addresses the offline-first approach from the point of view of mobile applications. The purpose of the thesis was to introduce various design principles and tools that can be utilized in the design and implementation of network independent mobile solutions.

The theoretical section of the thesis focuses on describing what a web-based application means and how the offline design principle fits into that model. The thesis also discusses various technical operating principles and tools that can be applied when designing offline features for Android-based mobile devices.

The thesis used the design research method. The result of the thesis was an artifact which was based on the requirements of a real-life business case that required a working offline-functionality. The design of the offline-functionality was conducted by the constraints and limitations set for the feature.

Based on the case-feature, a working solution was created which uses offline-first principles. The selected design principles used in the solution were then evaluated from the perspective of maintainability and comprehensibility. The thesis ends with an assessment of how the offline-functionality could be developed further and how the challenges of synchronization in distributed web-based applications could be overcome.



Keywords

mobile app development, offline-first, Android

SISÄLLYS

# 1 JOHDANTO

Digitalisaatio-aallon myötä monien toimialojen liiketoimintojen kriittiset operaatiot ovat siirtyneet digitaaliseen muotoon ja yritykset ovat ottaneet käyttöön monipuolisia tietojärjestelmiä. Tietojärjestelmiä hyödynnetään erilaisilla päätelaitteilla, erilaisissa tilanteissa ja sijainneissa.

Mobiililaitteet liiketoiminnan tukena nopeuttavat ja tehostavat etenkin prosesseja, jotka olisivat aiemmin vaatineet kiinteän työpisteen. Nyt työtehtävään liittyvät toiminnot voidaan hoitaa hyödyntäen kannettavaa päätelaitetta, joka työntekijälle on hankittu. Mobiililaitteiden avulla yrityksen työntekijä pystyy vaivattomasti kommunikoimaan yrityksen tietojärjestelmien kanssa.

Oxford Economics ja Samsung (2018, 2) teettivät kesäkuussa 2018 tutkimuksen, jossa selvitettiin matkapuhelinten käyttöä yritysten liiketoiminnan tukena. Tutkimuksessa haastateltiin 500 vanhempaa IT-johtajaa, toimitusjohtajaa ja muita ylemmän tason johtajia Yhdysvalloissa. 80% tutkimukseen vastanneista totesi, että yrityksen työtekijät eivät pysty tehokkaasti hoitamaan työtehtäviään ilman matkapuhelinta ja kolme neljäsosaa vastasi, että matkapuhelimien käyttäminen on oleellinen osa yrityksen henkilöstön työprosesseja.

Työntekijän mobiililaite voi käyttää saatavilla olevia WiFi-verkkoja tai operaattorien tarjoamia 3G/4G/5G -yhteyksiä. Toimialasta riippuen jatkuvan Internet-yhteyden ylläpitäminen voi olla kuitenkin haastavaa ja työtehtäviä voidaan suorittaa alueilla, jonne verkon kuuluvuus ei kanna. Edellä mainittua verkontilaa kutsutaan myös offline-tilaksi. Kyseistä tilaa pidetään usein sovelluskehityksessä virhetilana ja jos sitä ei ole tietojärjestelmän kehityksessä otettu huomioon, voi käytettävä tietojärjestelmä muuttua epäluotettavaksi tai täysin käyttökelvottomaksi.

Koska Internet-yhteyden tila voi vaihdella, tulee sovellusta suunniteltaessa ajatella mentaliteetilla ”Offline ensin”. Kun sovellus toimii offline-tilassa, voidaan siihen sen jälkeen lisätä kaikki verkkotoiminnallisuudet, joita tietojärjestelmän käyttämistä varten tarvitaan. (Google 2019a.)

# 2 TUTKIMUSMENETELMÄT

## 2.1 Motivaatio

Aloitin vuonna 2018 mobiilisovelluksen kehittämisen Android-pohjaiselle alustalle osana kolmehenkistä kehitystiimiä. Yksi sovelluksen perustason määrityksistä oli, että sen tulisi toimia saumattomasti offline-tilassa. Määrittely pohjautui asiakasyritykseltä suoraan tulleeseen tietoon, että heidän työntekijänsä työskentelevät usein sellaisissa olosuhteissa, joissa Internet-yhteyden muodostaminen onnistuu rajoitetusti.

Kehitystiimissä tiesimme kokemuksesta, että offline-toiminnallisuuden toteuttaminen mobiilisovellukseen kannattaa ottaa huomioon heti sovelluksen suunnitteluvaiheessa, sillä myöhemmin tehtävät muutokset voivat osoittautua haastaviksi. Yhtenä rajoittavana tekijänä oli asiakasyrityksen jo olemassa oleva perusjärjestelmä, johon mobiilisovelluksemme integroituisi. Idea opinnäytetyöhön syntyi tämän suunnitteluosuuden pohjalta.

## 2.2 Tutkimus

Opinnäytetyön tavoitteena on selventää offline ensin -käsitettä ja kuvailla mitä sillä tarkoitetaan käytännön tasolla. Tutkimuksessa tutustutaan erilaisiin offline-arkkitehtuureihin, joita voidaan hyödyntää mobiilisovelluksien kehityksessä offline-toiminnallisuuden tarjoamiseksi. Opinnäytetyössä vastataan seuraaviin tutkimuskysymyksiin:

- Millaisiin sovelluksiin offline-arkkitehtuuri tuo lisäarvoa.
- Miten offline ensin ajattelumalli vaikuttaa sovellusarkkitehtuuriin.

Tutkimuksessa esitellään aiheen teoriaa ja kuvataan käytännön esimerkkejä. Tutkimuksen käsittelyyn hyödynnetään suunnittelutieteellistä tutkimusmenetelmää (engl. design research) eli lyhyemmin DSRM. Tutkimuksen päätteeksi esitellään artefakti, jonka avulla voidaan kehittää offline-toiminnallisuuksia Android-pohjaiselle käyttöjärjestelmälle. Artefaktin suunnittelussa tutustutaan case-tapaukseen, jossa käytetään määrityksien pohjalta muodostettua suunnittelumallia, jonka avulla ohjelmakoodi pystytään pitämään helposti ylläpidettävänä ja ymmärrettävänä.

## 2.3 Tutkimuksen eteneminen

Suunnittelutieteellinen tutkimusmenetelmä on kehys, jota voidaan hyödyntää varsinkin teknisten alojen suunnittelutyössä. DSRM koostuu kuudesta aktiviteeteista:

- Ongelman tunnistaminen ja motivaatio
- Ratkaisun tavoitteet
- Suunnittelu ja kehitys
- Toteutuksen demonstrointi
- Arviointi
- Kommunikointi.

(Peffers, Tuunanen, Rothenberger & Chatterjee 2007, 52–56.)

Tutkimuksen alussa esitellään pääperiaatteet siitä, kuinka offline ensin -periaatteella suunniteltu sovellus poikkeaa normaalista verkkopohjaisesta sovelluksesta. Teoriaosuuden tarkoitus on avata aiheessa käsiteltäviä aihealueita lukijalle ja helpottaa ymmärtämään aihetta ilman aiempaa tuntemusta.

Tutkimuksen case-tapauksessa tutustutaan oikean asiakasyrityksen ongelmaan. Tässä osuudessa käydään lävitse asiakasyrityksen suurimmat haasteet ja motivaatiot verkontilasta riippumattoman ratkaisun takana.

Ratkaisun tavoitteiden ymmärtämisen jälkeen, kuvataan ja suunnitellaan arkkitehtuurinen malli, jonka avulla voidaan täyttää ratkaisulle annetut tavoitteet mahdollisimman tarkasti.

Toteutuksen demonstroimista varten esitellään ratkaisun arkkitehtuurinen rakenne ja esitellään työvälineitä, joita sen toteuttamisessa käytettiin.

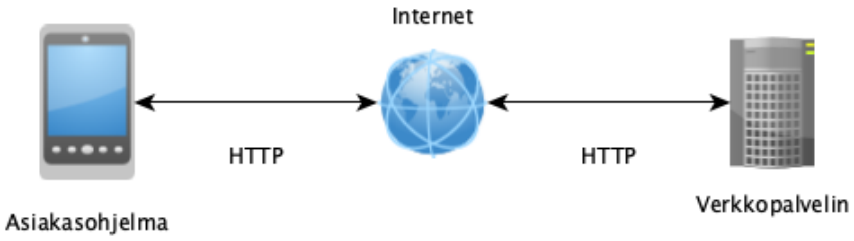
Tutkimuksen päätteeksi arvioidaan toteutetun ratkaisun onnistumisia ja asioita, joita siinä voitaisiin vielä kehittää tai parantaa.

# 3 TOIMINTAYMPÄRISTÖN KUVAUS

## 3.1 Web-pohjainen tietojärjestelmä

Web-pohjaisella järjestelmällä voidaan tarkoittaa asiakas-palvelin-arkkitehtuuria tai vertaisverkkoarkkitehtuuria. Tässä opinnäytetyössä web-pohjaisella järjestelmällä tarkoitetaan kuitenkin tietojärjestelmää, joka hyödyntää asiakas-palvelin-arkkitehtuuria, jossa asiakasohjelma ja verkkopalvelin kommunikoivat Internetin välityksellä hyödyntäen protokollaa, jota kumpikin osapuoli pystyy tulkitsemaan. (Kurose, Ross 2012, 110-112.)

Kommunikointi verkkopalvelimen kanssa tapahtuu tyypillisesti hyödyntäen HTTP-protokollaa. Verkkopalvelin asetetaan kuuntelemaan jotain avoinna olevaa porttia, johon asiakasohjelma pystyy ottamaan yhteyttä. (Kurose, Ross 2012, 116-123.) Asiakasohjelma lähettää verkkopalvelimelle pyynnön, jonka verkkopalvelin käsittelee ja palauttaa asiakasohjelmalle vastauksen hyödyntäen samaa protokollaa, jolla pyyntö lähetettiin.




Kuvio 1. Web-sovelluksen toimintaperiaate. Asiakas-palvelin-arkkitehtuuri

Pyynnön ja vastauksen mukana voidaan kuljettaa eri tyyppisiä resursseja. Resurssin tyyppi voi vaihdella kyseessä olevan kontekstin mukaan. Resurssityyppejä ovat esimerkiksi:

- JSON (JavaScript Object Notation)
- HTML (Hypertext Markup Language)
- XML (Extensible Markup Language)
- Kuva, teksti, ääni ja video

(Mozilla 2019.)

Näitä resursseja hyödyntäen voidaan asiakasohjelman ja verkkopalvelimen välillä välittää tietojärjestelmän kannalta oleellista tietoa kumpaankin suuntaan ja toteuttaa tietojärjestelmän vaatimukset hajautetussa ympäristössä.

## 3.2 REST-rajapinta arkkitehtuuri

REST on lyhenne sanoista “Representational State Transfer”. Se on hajautettujen järjestelmien arkkitehtuurinen tyyli, jonka Roy Fielding (2000) esitteli ensimmäisen kerran vuonna 2000.

REST-arkkitehtuurissa käytetään hyväksi normaalisti HTTP-protokollaa, joka mahdollistaa kuvaavien HTTP-metodien käyttämisen kommunikoinnissa verkkopalvelimen kanssa. Yleisimmin käytetyt HTTP-metodit ovat muun muassa:

- Get, jota käytetään resurssin hakemiseen.
- Post, jota käytetään resurssin luomiseen.
- Put, jota käytetään resurssin päivittämiseen.
- Delete, jota käytetään resurssin poistamiseen.

(Fielding, Gettys, Mogul, Frystyk, Masinter, Leach & Berners-Lee 1999.)

REST-arkkitehtuuri vaatii, että asiakasohjelma tekee pyynnön verkkopalvelimelle. Asiakasohjelman lähettämä pyyntö vaihtelee pyynnön kontekstin mukaan. Pyynnöt lähetetään päätepisteisiin verkkopalvelimella, joita kuvataan URI:n (Uniform Resource Identifier) avulla. Yksittäinen pyyntö ja vastaus pitää normaalisti sisällään muun muassa seuraavat tiedot:

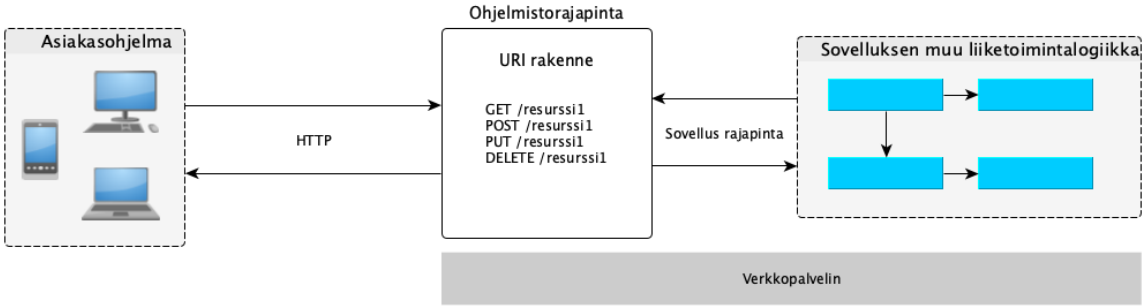
- Otsikkotiedot, jolla voidaan välittää lisätietoja pyynnöstä.
- Pyynnön runko, joka yleisesti kuvaa siirrettävän tietomallin.
- Metodin tyyppi.
- URI referenssi, joka kertoo resurssin sijainnin.
- Tilakoodi.

(Fielding ym. 1999.)

Verkkopalvelimelle toteutettu rajapinta käsittelee pyynnöt, jotka kohdistuvat määritettyihin päätepisteisiin. Päätepisteen toiminnasta ja toiminnallisuudesta vastaavat usein verkkopalvelimen kontrollerit, jotka ottavat verkkopalvelimelle tehdyn pyynnön vastaan. Käytännön tasolla kontrolleri on funktio, joka määritellyn kontekstin

perusteella käsittelee pyynnön sille välitettyjen tietojen perusteella ja toteuttaa kontekstin kuvaaman toiminnallisuuden.

Näitä periaatteita hyödyntäen voidaan kehittää ohjelmistorajapintoja verkkopalvelimille, jotka kuvaavat tavan, kuinka verkkopalvelin ja mikä tahansa asiakasohjelma voivat kommunikoida toistensa kanssa. Web-sovelluksia kehittäessä on tarjolla monenlaisia erilaisia teknologioita ja arkkitehtuureja, mutta tässä opinnäytetyössä hyödynnetään REST-arkkitehtuuria, joka mahdollistaa HTTP-pohjaisen tiedonsiirron asiakasohjelman ja verkkopalvelimen välillä.




Kuvio 2. REST-rajapinta -arkkitehtuuri

### 3.3 Offline ensin

Asiakas-palvelin-arkkitehtuurissa verkkopalvelin toimii asiakassovelluksen tietolähteenä. Kaikki sovelluksen liiketoimintaan liittyvä data voidaan säilyttää tai koostaa verkkopalvelimella. (Kurose, Ross 2012, 112.) Kun tämä data on saatavilla, tietojärjestelmän voidaan odottaa toimivan normaalisti.

Offline-tilassa asiakassovellus ei kuitenkaan pysty toimittamaan verkkopalvelimen tuottamaa dataa käyttäjälle, koska datan välitys tapahtuu hyödyntäen aktiivista Internet-yhteyttä. Asiakassovelluksen käyttäjälle tämä näkyy rikkinäisenä tuotteena.

Jos tietojärjestelmää halutaan kehittää offline-yhteensopivaksi, ei voida luottaa pelkästään siihen, että verkkopalvelin toimittaa ajankohtaisen ja oikean datan asiakassovellukselle. Hyvän käyttökokemuksen varmistamiseksi muokatut tiedot tallennetaan asiakassovellukseen ja tiedot synkronoidaan verkkopalvelimen tietovarastoon (Feyerke 2013).

Daniel Saublen (2015, 30-33) mukaan sovellus, jota on suunniteltu offline ensin -näkökulmasta, koostuu muun muassa seuraavista asioista:

- Keskeytymätön pääsy sisältöön. Tietojärjestelmän tulee tarjota mahdollisimman paljon sisältöä vahingoittamatta käyttökokemusta muilla tavoin.
- Yleinen toimintamalli on sallia asiakassovelluksessa vain lukuun tarkoitetut toiminnot offline-tilassa, mutta usein tietojärjestelmän käyttäjät haluavat työskennellä tietojärjestelmän kanssa verkon tilan vaihteluista huolimatta. Asiakassovelluksen toiminta tulisi olla dynaamista verkon tilasta huolimatta.
- Virhetilanteiden pitäisi olla hyvin selitettyjä ja niiden ei pitäisi jättää käyttäjää arvailemaan mitä on tapahtunut.
- Sovelluksen ei tule antaa käyttäjän aloittaa sellaisia operaatioita, mitä hän ei voi viedä loppuun verkon tilasta johtuen.
- Tilanteissa, jossa samaa dataa voidaan käsitellä samanaikaisesti usealla eri päätelaitteella, voi tulla vastaan tilanteita, jossa tapahtuu jotain odottamatonta. Tämänkaltaiset tilanteet tulisi selittää käyttäjälle mahdollisimman selkeästi.
- Tilanteessa, jossa dataa ei ole saatavilla, tulee käyttäjää informoida tarvittavista toimenpiteistä tilanteen korjaamiseksi.
- Palauta käyttäjä tilaan, jossa hän oli aikaisemmin. Älä vaaranna hyvää käyttökokemusta ainoastaan sen takia, että käyttäjä on offline-tilassa.

Hajautetun järjestelmän rakentaminen on yksi tietotekniikan vaikeimmista haasteista. Offline ensin -periaatteella kehitetty sovellus on luonnostaan hajautettu tietojärjestelmä. Koska tiedot voivat muuttua itsenäisesti monessa paikassa, laitteessa ja verkkopalvelimessa olevien tietosarjojen versioiden välillä syntyy ristiriitoja. (Realm 2017.) Ristiriitojen selvittäminen kannattaa ottaa huomioon mahdollisimman laajasti jo sovelluksen määrittelyvaiheessa, sillä ristiriitaiset tilanteet voivat usein aiheuttaa virhetiloja asiakassovelluksessa, joita käyttäjä ei omilla toimillaan pysty ratkaisemaan.

Offline ensin -sovellukset on suunniteltu tarkoituksenmukaisesti vastaamaan käyttäjien tarpeita kontekstin perusteella. Näissä sovelluksissa verkon joustavia ominaisuuksia ei yksinkertaisesti kiinnitetä olemassa olevaan suunnittelumalliin. Sen sijaan ne leivotaan sovellusarkkitehtuuriin ja käyttäjäkokemukseen. (Realm 2017.) Tästä syystä sovelluksen suunnitteluvaiheessa kannattaa punnita erilaisia vaihtoehtoja perustuen kehitettävän ratkaisun toimintavaatimuksiin ja tarkastella sovelluksen käyttötapauksia.

# 4 OFFLINE-SOVELLUKSEN SUUNNITTELU

## 4.1 Toteutustavat

Offline-sovellusta kehittäessä on usein tarpeen yhdistää erilaisia toteutustapoja tietojärjestelmän perusvaatimuksiin perustuen. Seuraavassa luvussa tutustutaan erilaisiin toimintatapoihin, joita hyödyntäen voidaan lähteä suunnittelemaan ja toteuttamaan tietojärjestelmää offline ensin -näkökulmasta.

### 4.1.1 Välimuistin käyttö

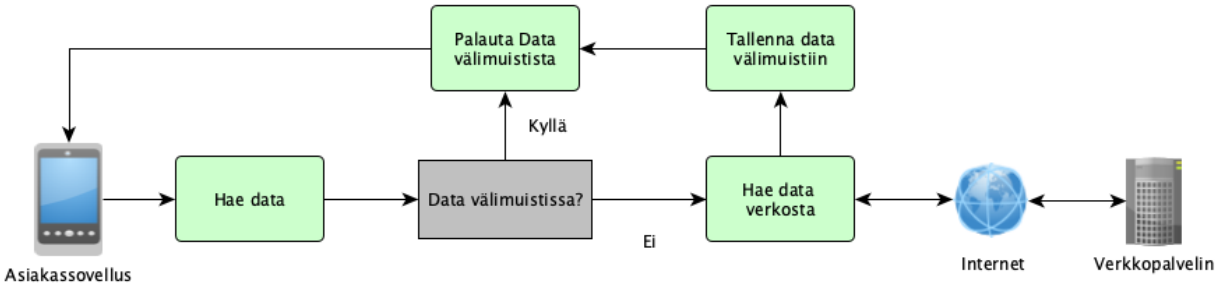
Asiakasohjelman toimintamallia, jossa hyödynnetään välimuistia, voidaan tutkia offline ensin -näkökulmasta hyödyntäen kuviota 3. Tarkasteltava malli on yksinkertaistettu esimerkki siitä, miltä lukuoperaatio voisi asiakassovelluksessa näyttää. Käyttäjän pyynnöstä aloitetaan operaatio, jonka seurauksena käyttäjälle esitetään toivottu sisältö asiakassovelluksen käyttöliittymässä.

Asiakassovellus tarkastaa ensin päätelaitteelle tallennetusta välimuistista onko kyseinen data saatavilla. Datan ollessa jo saatavilla se palautetaan suoraan pyynnön aloittaneelle ohjelmalle. Näin voidaan estää sovellusta hakemasta tuoreita tietoja ulkoisesta tietolähteestä tilanteissa, joissa tieto on jo saatavilla. Tämänkaltainen toimintamalli sopii erinomaisesti tilanteisiin, jossa kyseessä oleva tietosisältö ei päivity kovin usein ja sen uudelleenhakeminen lyhyen ajan sisällä on mitä todennäköisemmin turhaa.

Tilanteessa, jossa dataa ei ole saatavilla, pyydetään verkkopalvelinta palauttamaan viimeisin versio datasta. Jos verkkopalvelimelta saadaan vastaus, tieto tallennetaan päätelaitteen välimuistiin myöhempää käyttöä varten. Virhetilanteissa tulisi käyttäjälle ilmoittaa mistä kyseinen tilanne johtui.

Välimuistin käyttäminen sisältää tyypillisesti kaksi asiaa: välimuistin mitätöinti ja välimuistin pitäminen ajan tasalla. Välimuistin mitätöinnillä tarkoitetaan prosessia, jossa välimuistissa olevat tiedot poistetaan, koska ne eivät ole enää voimassa. (Andreu 2014.) Voimassaoloaika voidaan toimittaa verkkopalvelimelta tai asiakassovellus voi vaihtoehtoisesti itse päättää kuinka pitkään välimuistiin tallennettua tietosisältöä halutaan käyttää.

Välimuistin käyttämisellä voidaan myös optimoida laitteen virran ja kaistanleveyden käyttöä, mikä on elintärkeää varsinkin mobiilikehityksessä. Välimuistin käyttöä on syytä hyödyntää myös sovelluksissa, joita ei ole lähdetty suunnittelemaan offline ensin -näkökulmasta. Välimuistin käytön avulla voidaan parantaa kehitettävän sovelluksen suorituskykyä.




Kuvio 3. Asiakassovelluksen toimintamalli

### 4.1.2 Valikoivuus

Valikoivassa offline-toteutuksessa tietoja pyydetään yleensä palvelimelta. Valikoivalla mallilla vastuu datan tallentamisesta offline-käyttöä varten jätetään loppukäyttäjän vastuulle. (Andreu 2014.) Tätä mallia voidaan hyödyntää sovelluksissa, jossa ei ole järkevää ladata kaikkea sisältöä käyttöön offline-tilanteita varten. Tämänkaltaisesta toiminnallisuudesta hyötyvät varsinkin:

- Suoratoistopalvelut
- Palvelut, joissa käsitellään liitetiedostoja.
- Sovellukset, jotka käyttävät normaalia enemmän levytilaa.

Tietojärjestelmissä, jossa käyttäjälle annetaan mahdollisuus valikoivasti ladata sisältöä laitteelle, tulee huomioida missä muodossa tietosisältö tuodaan saataville. Data pidetään tallessa asiakassovelluksessa offline-käyttöä varten, mutta se voi olla arkaluontoista tai muuten sopimuksilla suojattua. Tätä tarkoitusta varten voidaan hyödyntää digitaalisen käyttöoikeuden hallintaan tarkoitettuja tekniikoita, joiden avulla pyritään estämään sisällön luvaton käyttö ja edelleenvälittäminen.

### 4.1.3 Manuaalinen replikointi

Manuaalinen replikointi on lähestymistapa, joka on seuraava askel välimuistin käytöstä. Tässä mallissa käyttäjän annetaan tehdä muutoksia tallennetulle datalle offline-tilassa. Sen avulla käyttäjät voivat käsitellä tietoja verkontilasta riippumatta merkitsemällä kohteet muutetuiksi. (Realm 2017.) Manuaalisen replikoinnin mallissa voidaan käyttää ajastettuja tehtäviä, joiden avulla asiakassovelluksessa käsiteltävät ja muutetut objektit välitetään verkkopalvelimelle. Vaihtoehtoisesti voidaan hyödyntää mallia, jossa muutokset asetetaan jonoon ja suoritetaan, kun jonolle määritetyt kriteerit täyttyvät.

Verkosta replikoitavat resurssit ovat usein esimerkiksi verkkopalvelimen REST-rajapinnan määrittelemässä formaatissa, jonka vuoksi resurssit tulee muuntaa tietojärjestelmässä käytettäviksi olioiksi. Tietojärjestelmän oliot voidaan tallentaa alustakohtaisesti mihin tahansa pysyvään varastoon.

### 4.1.4 Reaaliaikainen synkronointi

Tässä mallissa datan synkronointi ei ole manuaalista, vaan automaattista. Asiakasohjelmassa tehdyt muutokset lähetetään reaaliajassa. Tiedot välitetään hyödyntäen synkronointiprotokollaa, jonka välityksellä asiakasohjelman ja verkkopalvelinkerroksen välillä kulkeutuvat ainoastaan ne tiedot, joita on muutettu. (Realm 2017.)

Reaaliaikaisen synkronoinnin mallissa suurimmat hyödyt esiintyvät ristiriitojen hallinnassa. Tässä mallissa kokonaiset objektit eivät lähtökohtaisesti päällekirjoita muutoksia, jotka on tehty toiselta päätelaitteelta.

## 4.2 Työvälineet

Seuraavassa luvussa tutustumme työvälineisiin, joita hyödyntäen offline-tilassa toimivaa asiakasohjelmaa voidaan lähteä toteuttamaan. Luvussa käytettävät työvälineet ovat käytettävissä Android-pohjaisilla alustoilla.

### 4.2.1 Sisäinen muisti

Oletusarvoisesti Android-järjestelmässä sisäiseen muistiin tallennetut tiedostot ovat yksityisiä ja muut käyttöjärjestelmän sovellukset eivät voi käyttää niitä. Tämä

tekee sisäisestä tallennuksesta erinomaisen paikan yksityisille sovellustiedoille. Järjestelmä tarjoaa sisäisen ja rajoitetun hakemiston jokaiselle sovellukselle, jossa voidaan järjestää kaikki sovelluksen tarpeisiin liittyvät tiedostot. Kun käyttäjä poistaa sovelluksen, sisäiseen muistiin tallennetut tiedostot poistetaan. (Google 2019b.)

Sisäistä muistia voidaan käyttää monimuotoiseen datan tallennukseen, eikä sovellus ole sidoksissa mihinkään tiettyyn tallennusformaattiin. Sisäinen muisti sopii varsinkin suunnittelumalliin, jossa hyödynnetään valikoivuutta ja tallennettava data on monimuotoista tai sen rakenne ei ole helposti määriteltävissä.

### 4.2.2 Sisäinen välimuisti

Jos asiakassovellus haluaa säilyttää joitakin tietoja väliaikaisesti, voidaan Android-järjestelmissä käyttää erityistä välimuistihakemistoa tietojen tallentamiseen. Jokaisella sovelluksella on oma välimuistihakemistonsa. Käyttöjärjestelmä voi kuitenkin poistaa nämä välimuistitiedostot tallennustilan palauttamiseksi, eikä sitä tulisi käyttää asiakassovelluksen toiminnan kannalta välttämättömän tiedon tallentamiseen. Asiakasohjelman vastuulla on kuitenkin oletusarvoisesti aina säilyttää ja tyhjentää välimuistitiedostot itse. Kun käyttäjä poistaa sovelluksen käytöstä, välimuistiin tallennetut tiedostot poistetaan samalla. (Google 2019b.)

Sisäistä välimuistia voidaan käyttää silloin, kun halutaan säilyttää verkosta ladattavia resursseja ja optimoida kuinka usein samaa tietoja halutaan kysyä verkkopalvelimelta.

### 4.2.3 Ulkoinen muisti

Useimmat Android-laiteet tukevat jaettua ulkoista tallennustilaa. Ulkoiseen muistiin tallennetut tiedostot ovat käyttöjärjestelmän luettavissa ja laitteen käyttäjä voi myös itse käsitellä niitä. Järjestelmä tarjoaa standardeja julkisia hakemistoja tällaisille tiedostoille. Tallennustila voi olla hyödyllinen vaihtoehto, kun ei haluta hyödyntää sovelluksen sisäistä muistia. (Google 2019b.)

Ulkoisen muistin käyttäminen on hyvä vaihtoehto tilanteissa, jossa asiakassovellus haluaa esittää kuvia, videoita tai muita tiedostoja, jotka vievät paljon tilaa. Tätä

muistia hyödyntäessä tulee huomioida, että ulkoinen muisti ei ole suojattu Android-käyttöjärjestelmän toimesta samalla tavalla kuin sovelluksien sisäinen muisti.

### 4.2.4 SharedPreferences

Jos asiakasohjelman ei tarvitse tallentaa suuria määriä tietoja, eikä se ei vaadi tiettyä rakennetta, SharedPreferences-tallennustila sopii tällaisia tilanteita varten loistavasti. SharedPreferences-ohjelmistorajapintojen avulla voidaan lukea ja kirjoittaa pysyviä avainarvopareja primitiivisistä tietotyypeistä. Avainarvoparit on kirjoitettu XML-tiedostoihin, jotka pysyvät tallessa, vaikka sovelluksesi sammutettaisiin käyttöjärjestelmän toimesta. SharedPreferences tulee sanoista ”Jaetut asetukset”, mutta mikään ei estä tallentamasta sinne myös käyttäjän asetuksiin liittymättömiä tietoja. (Google 2019b.)

### 4.2.5 Tietokanta

Android-laitteella voidaan käyttää myös tietokantaa asiakasohjelman tallennustilana. Android tarjoaa täyden tuen SQLite-tietokannoille. Asiakasohjelmassa käytettävä tietokanta on ainoastaan sovelluksen käytössä. (Google 2019b.)

SQLite on prosessin sisäinen kirjasto, joka toteuttaa itsenäisen, ilman konfiguraatioita toimivan operatiivisen SQL-tietokannan moottorin. SQLite lukee ja kirjoittaa suoraan tavallisiin levytiedostoihin ja yksi levytiedosto sisältää täydellisen SQL-tietokannan, jonka formaatti on alustariippumaton. (Sqlite 2019.)

Google suosittelee käyttämään SQLite:a hyödyntäen Room-kirjastoa. Room tarjoaa abstraktiokerroksen SQLiten päälle. Sovellukset, jotka käsittelevät suuria määriä strukturoituja tietoja hyötyä suuresti siitä, että tiedot pysyvät paikallisesti tietokannassa. (Google 2019c.)

Room-kirjasto on hyödyllinen työväline silloin, kun halutaan antaa käyttäjälle mahdollisuus tehdä muutoksia asiakassovelluksessa käsiteltäviin olioihin sovelluksen ollessa offline-tilassa. Muuttuneet tiedot pidetään tallessa tietokannassa ja myöhemmin verkkoyhteyden palautuessa tiedot voidaan jälleen synkronoida verkkopalvelimen kanssa.

Kun sovelluksessa halutaan hyödyntää Room-kirjastoa, olioihin lisätään huomautus, jonka avulla Room-kirjasto tunnistaa, että kyseessä on entiteetti, jota halutaan käyttää SQLite-tietokannan kanssa.

Esimerkki Room-kirjastossa käytettävästä entiteetissä on esitelty kuvassa 1. Kuvan ohjelmakoodissa on luotu olio, johon on lisätty huomautuksia Room-kirjastoa varten. Room käsittelee ohjelmakoodiin merkityt huomautukset ja luo tämän jälkeen SQLite-tietokantaan entiteettiä vastaavan taulun. Huomautuksiin voidaan lisätä ylimääräisiä ohjaavia tekijöitä, joiden avulla pystytään vaikuttamaan siihen, miten tietokantataulu rakentuu SQLite-tietokantaan.

Room-entiteetin käyttäminen Android-pohjaisella alustalla vähentää redundanssia ja sitä kautta virheiden mahdollisuutta, kun ohjelmakoodissa joudutaan kirjoittamaan samoja olioita eri asiayhteyksiin.

```
@Entity(tableName = "customers")
data class Customer(
    @PrimaryKey
    @ColumnInfo(name = "id")
    val id: String = UUID.randomUUID().toString(),
    @ColumnInfo(name = "customer_code")
    val customerCode: String,
    @ColumnInfo(name = "name")
    val name: String,
    @ColumnInfo(name = "address")
    val address: String
)
```

Kuva 1. Room entiteetti

## 4.3 WorkManager

WorkManager on Android-sovelluksien kehitykseen tarkoitettu ohjelmistorajapinta, joka tarjoaa tavan ajastaa asynkronisesti ajettavia tehtäviä. Tehtävät ajetaan sovelluksille jaetussa ympäristössä, vaikka asiakassovellus olisi sammutettu tai käyttöjärjestelmä uudelleen käynnistetty. (Google 2019e.)

WorkManagerilla tarkoitetaan ”Töiden hallitsijaa” ja se soveltuu tilanteisiin, joissa halutaan ajaa sovelluksen toimintaan liittyviä tehtäviä ennalta määriteltyjen kriteerin täyttyessä. Sen toiminnasta vastaa käyttöjärjestelmä ja sen avulla voidaan hoitaa niin yksittäisiä tehtäviä kuin myös erityyppisiä jaksoittaisia tehtäviä.

## 4.4 Sovellusarkkitehtuuri

Offline-sovelluksen kehittäminen voi osoittautua odotettua haasteellisemmaksi tehtäväksi ja on tärkeää, että ratkaisua varten kirjoitettu ohjelmakoodi pysyy helposti ylläpidettävänä ja helppona ymmärtää. Tässä luvussa käydään läpi sovellusarkkitehtuurisia malleja, joita hyödyntäen voidaan lähteä ohjaamaan offline-kyvykkään asiakassovelluksen kehitystä ylläpidettävään ja yhtenäiseen muotoon.

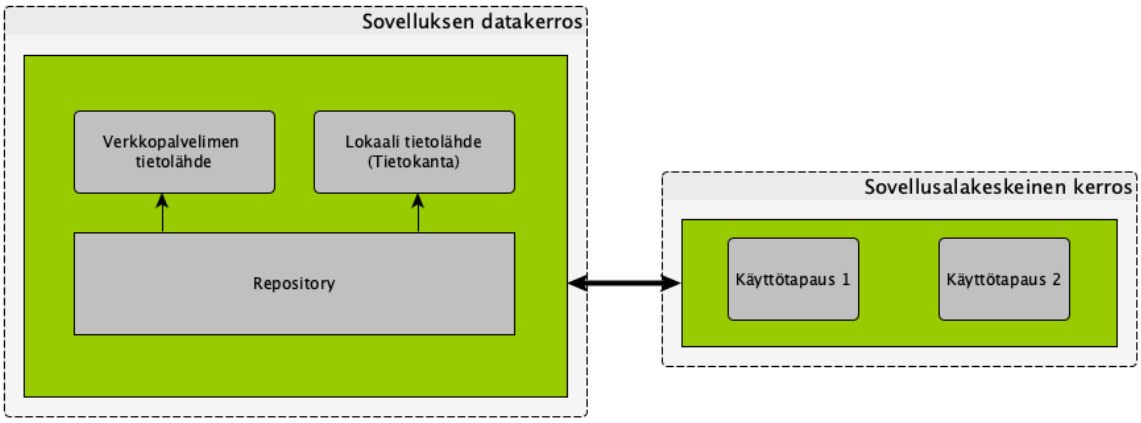
### 4.4.1 Repository pattern

Repository pattern on ohjelmistokehityksessä käytettävä malli, jonka tarkoituksena on eristää datan muodostamisen monimutkaisuus ja liitännät datavarastoihin muilta ohjelman kerroksilta. Repository esittää yksinkertaisen rajapinnan toiminnallisuudesta. (Fowler 2002, 322-325.)

Offline ensin -mentaliteetilla suunnitellun sovelluksen käsittelemät oliot voivat lähtökohtaisesti tulla erilaisista datalähteistä. Tällaisia tietolähteitä voivat olla esimerkiksi:

- tietokanta
- sovelluksen välimuisti
- käyttöjärjestelmän tarjoama tiedosto
- käyttöjärjestelmän tarjoama muu tallennustila
- verkkopalvelin.

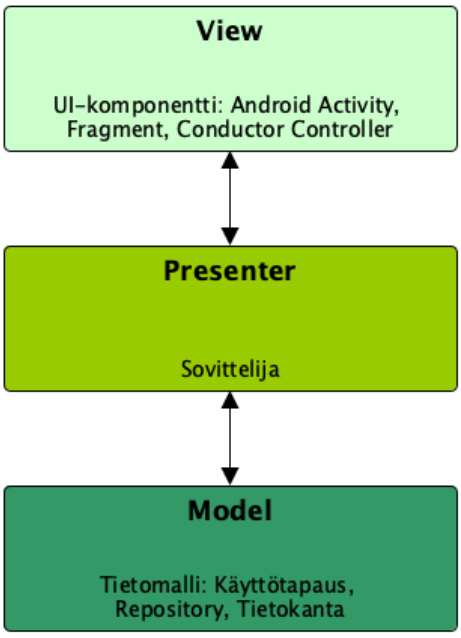
Kuten kuviossa 4 on esitelty, Repository-malli yhtenäistää datan saavutettavuuden helposti ylläpidettävään muotoon siten, että sovelluksessa käytettävän sovellusalakeskeisen logiikan ei tarvitse tietää mistä datavarastosta kyseinen data todellisuudessa haetaan. Mallia hyödyntämällä voidaan helposti lisätä erilaisia ja monimutkaisia datavarastoja erilaisia tilanteita varten, eikä yhden datavaraston lisäämisen tai poistamisen tulisi vaikuttaa muun sovelluksen toimintaan tai aiheuttaa muita muutoksia. Repository-toteutuksen vastuulla on toimia sovittelijana eri tietolähteiden välillä.




Kuvio 4. Repository pattern

### 4.4.2 Model-View-Presenter

MVP (Model-View-Presenter) on arkkitehtuurinen malli, jota yleisesti käytetty käyttöliittymien kehityksessä. Model eli tietomalli on osa sovelluksen liiketoimintalogiikkaa ja se ohjaa sovelluksessa käytettävän tietomallien käsittelyä. View eli näkymä on passiivinen rajapinta, joka on vastuussa tietojen esittämisestä. Presenter eli esittäjä toimii sovittelijana tietomallin ja näkymän välillä. Esittäjä hakee esitettävän tietomallin ja välittää sen eteenpäin näkymälle passiivista rajapintaa hyödyntäen. Näkymän vastuulle kuuluu myös käyttäjien toimintojen välittäminen esittäjätoteutukselle. (Megali 2016.)




Kuvio 5. Model-View-Presenter (mukailtu Megali 2016).

MVP-malli ohjaa erottelemaan eri osa-alueiden vastuut selkeästi kuten kuviossa 5 on esitetty. Vastuualueiden erottelu vähentää osa-alueiden riippuvuuksia toisistaan ja mahdollistaa ohjelmakoodikannan paremman ylläpidettävyyden ja testattavuuden. Koska tietomalli ja näkymä eivät ole suoranaisesti kytköksissä toisiinsa, voidaan niitä muokata ja kehittää itsenäisesti.

MVP-malli korostaa selkeää vastuualueiden erottelua ja sen yhtenä perusajatuksena on, että jokaista komponenttia voidaan testata täysin eristetyssä ympäristössä. Esimerkiksi esittäjäkerroksen funktioille voidaan kirjoittaa yksikkötestejä ilman riippuvuuksia UI-komponentteihin hyödyntäen passiivista rajapintaa, jonka näkymä toteuttaa. (Holmström 2015.)

### 4.4.3 Reaktiivinen ohjelmointi

Reaktiivinen ohjelmointi on ohjelmointiparadigma, jossa ohjelman tilaa muutetaan automaattisesti ja tehokkaasti. Tämänkaltainen paradigma sopii varsinkin sovelluksiin, joissa ohjelmakoodia suoritetaan usein tapahtumapohjaisesti. Tästä syystä reaktiivinen ohjelmointi sopii hyvin käyttöliittymäkehitykseen, jossa reagoidaan näytöllä tehtäviin tapahtumiin. (Bainomugisha, Carreton, van Cutsem, Mostinckx & de Meuter 2012, 1-3.)

Reaktiivinen ohjelmointi perustuu erilaisiin datavirtoihin. Tällainen virta voi olla tyypiltään synkroninen tai asynkroninen. Datavirrat voivat myös pitää sisällään X-määrän arvoja ja voivat käytännön tasolla olla internetistä haettavia tietueita tai käyttäjän kosketustapahtumia sovelluksen käyttöliittymässä. (Nurkiewicz & Christensen, 2016.)

Reaktiivinen ohjelmointi yhdistää erilaisia ohjelmiston suunnittelumalleja, jonka vuoksi sen käytön aloittamista pidetään haastavana. Mallien monimutkaisuuden vuoksi reaktiivista ohjelmointia varten on kehitetty ohjelmistorajapinta nimeltä ”Reactive Extensions”. Reactive Extensions eli lyhyemmin Rx on spesifikaatio, jonka pohjalta on toteutettu avoimen lähdekoodin kirjastoja käytettäväksi eri alustoilla. Rx on funktionaalisen ohjelmoinnin, Observer-mallin ja Iterator-mallin yhdistelmä (ReactiveX 2019b).

Reactive Extensions kutsuu datavirtoja nimellä Observable. Observable toteuttaa rajapinnan, jonka avulla voidaan reagoida tarkasteltavan olion tuottamiin tapahtumiin. Observablen tapahtumia voidaan kuunnella rekisteröimällä sille jokin kuuntelija. Kuuntelija toteuttaa Observer-rajapinnan, joka on esitelty kuvassa 2. (Nurkiewicz & Christensen, 2016.)

Datavirran tuottama data välitetään kuuntelijalle hyödyntäen Observer-rajapinnan onNext-metodia, jota kutsutaan aina datavirran tuottaessa uusia arvoja. Datavirrassa tapahtuvat virheet välitetään kuuntelijan onError-metodille. Virheen tapahtuessa datavirrasta ei odoteta enää saapuvan uusia arvoja ja datavirta päättyy. Datavirran päättyessä kutsutaan kuuntelijan metodia onCompleted.

```
interface Observer<T> {
    void onNext(T t)
    void onError(Throwable t)
    void onCompleted()
}
```

Kuva 1. Observer rajapinta (Nurkiewicz & Christensen, 2016)

# 5 ASIAKASPROJEKTIN TOTEUTUS

## 5.1 Asiakasprojektin johdanto

Tässä osiossa käymme läpi asiakasprojektin taustoja. Asiakasprojektin lähtökohtana oli luoda mobiilisovellus, jonka avulla asiakasyrityksen työntekijät pystyisivät hoitamaan päivittäiset rutiinitoimenpiteensä verkontilasta huolimatta. Tutkimuksessa ei syvennytä liiketoimintalogiikan toteutukseen tarkemmin, vaan keskitytään sovellusarkkitehtuuriin, jonka avulla voidaan liittää tarvittavat offline-toiminnallisuudet toteutukseen vaarantamatta sovelluksen ylläpidettävyyttä.

## 5.2 Ratkaisun arkkitehtuuri

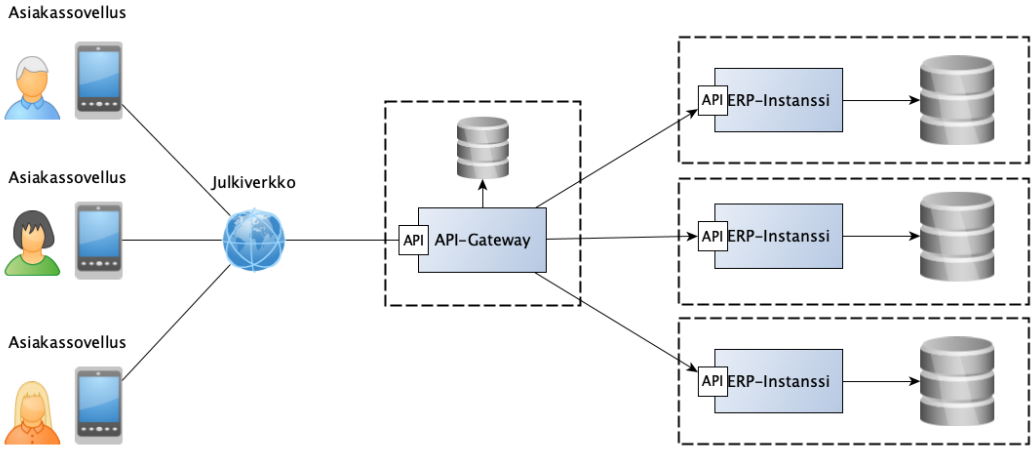
Kehitettävä mobiiliratkaisu integroituu asiakaan olemassa olevaan toiminnanohjausjärjestelmään, jota kutsutaan myös ERP-järjestelmäksi (Enterprise Resource Planning). ERP:n avulla järjestelmää käyttävä yritys pystyy organisoimaan koko tuotannonohjauksensa ja esimerkiksi henkilöstöhallintansa. ERP-järjestelmä koostuu erilaisista moduuleista, joita voidaan tarpeen tullessa aktivoida järjestelmän käyttäjille. Järjestelmästä löytyy myös REST-rajapintakerros, jonka avulla järjestelmän kanssa voidaan kommunikoida.

Asiakkaan kanssa suunniteltujen määrityksien perusteella päädyttiin siihen, että mobiiliratkaisu toimitetaan yhtenä yhtenäisenä kauppapaikalta ladattavana tuotteena, jonka on määrä integroitua ERP-järjestelmän eri instanssien kanssa. Yhdellä ERP-instanssilla tarkoitetaan tässä yhteydessä asennusta, joka ei ole liitoksissa muihin ERP-instansseihin ja toimii täysin itsenäisenä kokonaisuutena.

Koska mobiiliratkaisu on ERP-instanssista huolimatta kaikille sovelluksen käyttäjille sama, eri instanssien välillä voidaan navigoida hyödyntäen instanssikohtaisia tunnisteavaimia. Avainten avulla sovelluksen välittämät rajapintakutsut voidaan välittää oikean instanssin REST-rajapinnoille. Tätä varten Mobiiliratkaisuun päätettiin lisätä erillinen API-gateway, joka on esitelty kuviossa 6. Sen vastuulla on liikenteen ohjaaminen oikealle instanssille.

Mobiilisovellus on toteutettu suoraan Android-käyttöjärjestelmälle hyödyntäen Kotlin-ohjelmointikieltä. Se on yhteydessä API-gateway -palveluun, joka tarjoaa asiakassovellukselle REST-rajapinnan. API-gateway toimii Node.js-ajoympäristön

päällä. Tieto saatavilla olevista ERP-instansseista säilytetään PostgreSQL-tietokannassa, jonka perusteella API-gateway osaa ohjata pyynnöt oikeille ERP-instansseille.




Kuvio 6. Järjestelmäarkkitehtuuri

## 5.3 Kehitettävän ratkaisun lähtötilanne

ERP-järjestelmä koostuu monista erilaisista moduuleista, joiden avulla pystytään kattavasti hoitamaan yrityksen eri osa-alueiden hallintaa. Tätä tarkoitusta varten ERP-järjestelmä tarjoaa Windows-pohjaisen käyttöliittymän, joka voidaan räätälöidä yrityksien tarpeisiin sopivaksi erilaisin järjestelmäparametrein.

ERP-järjestelmää käyttävien yritysten työskentelymalli on projektiluontoista. Järjestelmän kautta työnjohto voi seurata ja suunnitella projektien etenemistä. Reaaliaikainen seuranta ja reagointi vaatii, että projektille suunnitellut tehtävät raportoidaan tarkasti ja mahdollisiin ongelmatilanteisiin reagoidaan nopeasti.

Ongelmaksi muodostuu usein päätelaitteiden saavutettavuus ja Windows-pohjaisen käyttöliittymän monimutkaisuus. Tästä syystä yrityksien työntekijöiden rutiinitoimenpiteiden toteuttaminen työaikana on harvoin reaaliaikaista ja tiedon täsmällisyys voi heikentyä samasta syystä.

## 5.4 Ratkaisun tavoitteiden määrittely

ERP-järjestelmän lähtökohtiin viitaten, lähdettiin asiakasyrityksen kanssa selvittämään määrityksiä, joiden perusteella mobiiliratkaisua voidaan toteuttaa iteratiivista mallia hyödyntäen. Tässä suunnitteluosuudessa keskitytään ainoastaan yhteen

ERP-järjestelmän moduuliin, jossa käsitellään tietojärjestelmän työajanseurantaan ja raportointiin liittyviä ominaisuuksia.

Työntekijöiden päivittäiset rutiinitoimenpiteet, johon hyödynnetään tietojärjestelmää, koostuvat muun muassa seuraavista asioista:

- Käyttäjät kirjautuvat sisään järjestelmään.
- Työpäivän aloittaminen ja lopettaminen.
- Työpäivän aikana tehtävät kirjaukset tehtäville.
- Työpäivän muut kirjaukset ja tauot.

Näitä toimintoja varten mobiiliratkaisuun suunniteltiin helposti käytettävä versio ERP-järjestelmän työajanseurannasta. Verkon tilasta ei voida tehdä minkäänlaisia olettamuksia, sillä projektin ja työtehtävien sijainti ja olosuhteet vaihtelevat aina tapauskohtaisesti. Verkkoyhteyden hetkellisen tai pitkäjaksoisen kuulumattomuuden ei tule olla ongelma reaaliaikaiselle työajanseurannalle.

## 5.5 Toteutuksen suunnittelu

Työajanseuranta perustuu yksittäisiin kirjauksiin. Yhdellä kirjauksella voidaan viedä kaikki tarvittavat työtapahtumaan liittyvät tiedot mobiiliratkaisusta ERP-instanssiin. Kirjauksien perusteella voidaan tulkita työaikaan ja työpäivään liittyviä tietoja.

Toteutuksen aloitusvaiheessa selvitettiin, minkälaisia toteutustapoja voidaan hyödyntää verkontilasta riippumattomassa työajanseurannassa. Mobiiliratkaisun integroituminen olemassa olevaan ERP-järjestelmään tulee ottaa huomioon myös mobiiliratkaisun suunnittelussa. Tietokannan teknologiavalinnat ja REST-rajapinnan arkkitehtuuri oli jo aikaisemmin määritelty tuotantoympäristössä pyöriville ERP-instansseille. Tämä poissulki luvussa 4.1.4 esitellyn reaaliaikaisen synkronoinnin mallin. Mallin hyödyt olisivat näkyneet ristiriitojen hallinnassa. Tätä mallia ei ehdotettu asiakkaalle, koska synkronointiprotokollan integroiminen olisi vaatinut todella suuria investointeja ERP-instanssista vastaavalle taholle.

Suunnitteluvaiheessa kävimme läpi muitakin malleja, joita on esitelty luvussa 4.1. Välimuistin käyttäminen oli luonnollinen valinta pelkästään sovelluksen optimoinnin kannalta. Työajanseurannan kirjausten tekemistä verkontilasta riippumatta ei kuitenkaan voida toteuttaa pelkällä välimuistin käytöllä. Tätä toiminnallisuutta varten

päätettiin hyödyntää manuaalisen replikoinnin mallia, jonka avulla voitaisiin antaa käyttäjälle mahdollisuus suorittaa toimenpiteitä verkontilasta riippumatta.

### 5.5.1 Työajanseurannan suunnitteluperiaate

Työajanseurannan määrityksissä mainittiin, että ominaisuutta voidaan käyttää myös tilanteessa, jossa mobiiliratkaisulla ei ole minkäänlaista yhteyttä verkkoon. Toteutusta lähdettiin suunnittelemaan malliin, jossa luotetaan pääsääntöisesti laitteen omaan tietokantaan. Mallissa, joka on esitelty kuviossa 7, tietokanta toimii sovelluksen tietolähteenä.

Verkko-osuus voitaisiin jättää pois toteutuksesta, mutta käytännöntasolla asiakassovelluksessa tarvitaan kuitenkin tietoja, jotka ovat saatavilla ainoastaan verkkopalvelimelta käsin. Koska sovellusarkkitehtuurin mallissa tietokanta hallitsee sovelluksen tilaa, täytyy ominaisuudet, jotka ovat riippuvaisia palvelinpuolen toteutuksesta, toteuttaa mobiiliratkaisuun itsenäisesti toimivana kokonaisuutena.


Kuvio 7. Työajanseurannan arkkitehtuuri

Asiakassovelluksen tietokantaan mallinnetaan tarvittavat tietomallit hyödyntäen luvussa 4.2.5 esiteltyä Room-kirjastoa. Tietokantamallit mukailevat verkkopalvelimen datamallia, mutta tietokannan entiteettejä varten päädyttiin lisäämään myös ylimääräisiä ominaisuuksia, joiden avulla voidaan seurata missä tilassa käyttäjän tekemät kirjaukset ovat. Näitä lisättyjä ominaisuuksia ovat muun muassa:

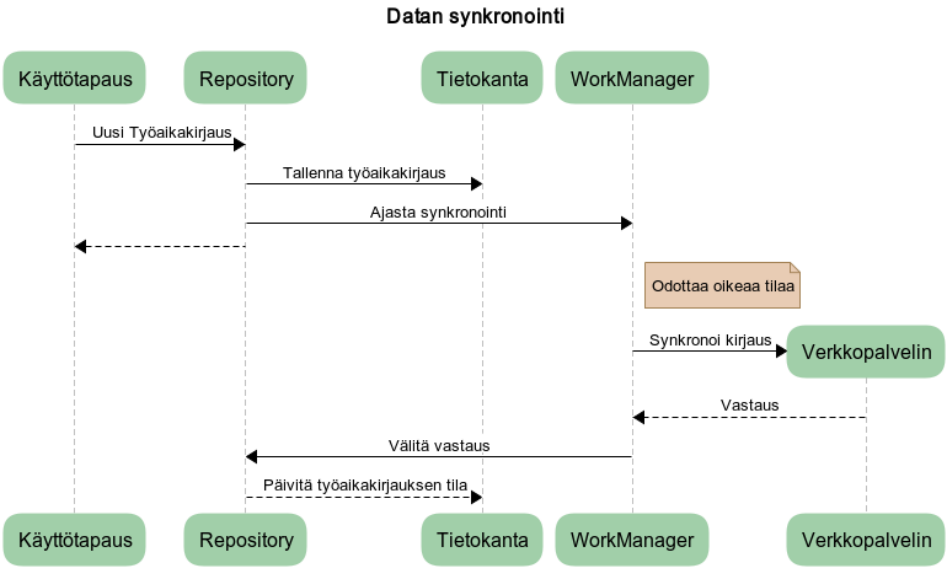
- synkronoinnin tila
- synkronoinnin HTTP-koodi
- muu virheviesti.

### 5.5.2 Tietojen synkronointi

Kuviossa 8 esitellään tapa, jolla työajanseurannassa syntyvät tapahtumat synkronoidaan ERP-instanssin kanssa. Käyttäjän tehdessä uuden kirjauksen, se välitetään toteutetulle repositorylle, jonka toimintaperiaatteet on esitelty luvussa 4.4.1. Repository vastaa datan käsittelystä, jonka seurauksena tietokantaan lisätään uusi

kirjaus ja kirjauksen vieminen verkkopalvelimelle ajastetaan hyödyntäen luvussa 4.3 esiteltyä WorkManager-työvälinettä. Ajastetulle operaatiolle annetaan rajoitetiedoksi vaadittava Internet yhteys.

Kun verkkopalvelimelta saadaan vastaus, tieto välitetään repository-toteutukselle, joka päivittää verkkopalvelimelta saadun vastauksen perusteella tilatiedon rivikohtaisesti sovelluksen tietokantaan.

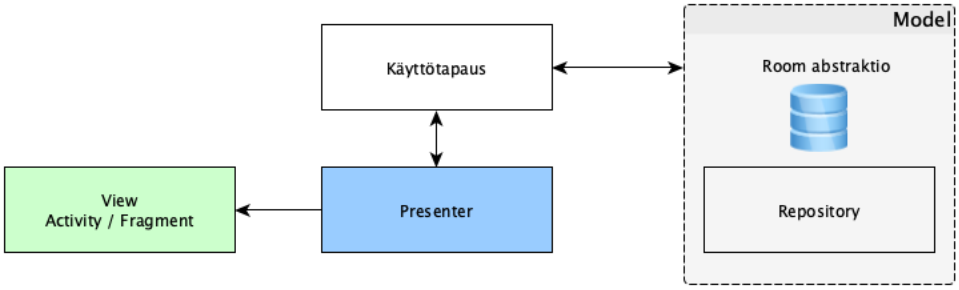



Kuvio 8. Synkronointimalli

Työajanseurannan toimintoja voidaan käyttää monelta eri päätelaitteelta. Tästä syystä on tärkeää synkronoida sovelluksessa käytettävä tietosisältö verkkopalvelimen kanssa. Mobiiliratkaisussa päädyttiin toteuttamaan synkronointi tapahtumapohjaisesti. Käyttäjän siirtyessä näkymältä toiselle kysytään verkkopalvelimelta toiminnallisuuteen liittyvää uusinta versiota tietosisällöstä. Tilanteessa, jossa verkkoyhteyttä ei ole saatavilla luotetaan viimeisimpään tietosisältöön, joka on synkronoitu verkkopalvelimelta asiakassovellukseen. Toteutuksessa käytettävä tietosisältö tulee kuitenkin kertaalleen synkronoida verkkopalvelimen kanssa sovelluksen käynnistyksen jälkeen.

### 5.5.3 Näkymäkerroksen tilanhallinta

Kuten luvussa 5.5.1 on esitelty, tietokanta toimii sovelluksen tietolähteenä. Kaikki synkronointiin liittyvät yksityiskohdat on eristetty repository-toteutuksen taakse, joten näkymäkerrosta voidaan rakentaa itsenäisesti verkontilasta riippumatta. Sovelluksen näkymien toteutuksessa on hyödynnetty luvussa 4.4.2 esiteltyä MVP-mallia. Toteutuksen yhteydessä päädyttiin käyttämään hiukan muunneltua MVP-mallia, jossa esittäjän ja tietomallin välille on luotu erillinen käyttötapaus-kerros, jonne voidaan eritellä liiketoimintalogiikkaan liittyvät toiminnot. Tämä on esitelty kuviossa 9. Käyttötapaus-välikerros päädyttiin lisäämään ratkaisuun myöhäisemmässä vaiheessa johtuen esittäjäluokkien kasvavasta monimutkaisuudesta. Tätä mallia edesauttoi myös huomattu tarve jakaa tiettyjä toiminnollisuuksia eri näkymien välillä tilanteissa, jossa toiminnallisuutta ei ollut järkevää kirjoittaa tietomalli-kerrokseen ylläpidettävyyden ja uudelleenkäytettävyyden kannalta.




Kuvio 9. Näkymäkerroksen tilanhallinta

Koska käyttäjän työajanseurantaan liittyvä tila on tallessa mobiilisovelluksen tietokannassa, voidaan näkymäkerroksen tilanhallinnan helpottamiseksi hyödyntää RxJava-kirjastoa, joka on yksi Reactive Extensionin implementaatioista. Reaktiivisen ohjelmoinnin suunnitteluperiaatteet on esitelty luvussa 4.4.3. RxJava:n avulla näkymäkerroksen päivittäminen voidaan toteuttaa tapahtumapohjaisia datavirtoja hyödyntäen.

Room-kirjaston avulla tietokantakyselyt voidaan kuvata sovelluksessa käytettävinä datavirtoina. Tämä kuvaus tehdään DAO (Data access object) -toteutukseen, jonne voidaan määrittää toteutuksessa käytettävät SQL-kyselyt SQLite-tietokantaan. Esimerkki DAO-toteutuksesta on esitelt kuvassa 3.

```
@Dao
abstract class TimeStampDao {

    @Query("""SELECT * FROM timestamp ORDER BY Datetime(stampTime) DESC """)
    abstract fun loadObservable(): Flowable<List<TimeStampEntity>>

    @Query("""SELECT * FROM timestamp
        WHERE stampTime=(SELECT MAX(stampTime) FROM timestamp) LIMIT 1""")
    abstract fun findLatestByDate(): Single<TimeStampEntity>

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    abstract fun save(stamp: TimeStampEntity)
}
```

Kuva 2. DAO-luokka

Datavirtoihin perustuvassa toteutuksessa sovelluksen näkymien päivittyminen suoraviivaistuu, sillä tietokantaa päivittäessä tieto välittyy automaattisesti kaikille rekisteröidyille kuuntelijoille. Datavirtoja tarjotaan kuitenkin melko harvoin sellaisinaan sovelluksen käyttöliittymälle. Tätä tarkoitusta varten voidaan käyttää aiemmin luvussa esiteltyjä käyttötapauksia, jossa datavirtoja voidaan muokata, niille voidaan antaa rajoitteita ja niitä voidaan yhdistää (ReactiveX 2019a).

Käyttötapaus toimii välikätenä esittäjä- ja tietomalli-kerrosten välillä ja sen tarkoituksena on toteuttaa toimialueelle tyypilliset toiminnallisuudet. Tutkittava toimialue on tässä yhteydessä työajanseuranta ja kyseessä olevan käyttötapauksen on tarkoitus tulkita käyttäjän kirjauksista työpäivän tilan ja muuta käyttäjälle tärkeää tietoa kuten työajan keston.

Käyttötapaus eristää työajanseurantaan liittyvän liiketoimintalogiikan näkymäkerrokselta ja helpottaa sen suunnittelua. Käyttötapauksiin liittyvä logiikka on syytä suunnitella niin, että sillä ei ole riippuvuuksia esimerkiksi Android-alustaan, eli siten, että se koostuu puhtaasti logiikkapohjaisista luokista. Jos käyttötapaukset on suunniteltu alustariippumattomasti, voidaan niitä jakaa eri käyttöympäristöjen kesken.

Esittäjä voidaan rekisteröidä MVP-mallissa datavirran kuuntelijaksi. Kun uusi kuuntelija rekisteröidään, kuuntelija toteuttaa kuvassa 2 esitellyn rajapinnan. Datavirran välittäessä uuden tapahtuman, se toimitetaan kuuntelijan onNext-metodille. Tämän seurauksena kuuntelija voi kutsua rajapinnan metodia, jonka Android näkymä toteuttaa. Näin käyttöliittymän tila päivittyy reaktiivisesti datajoukon muuttuessa.

Kuuntelijoita rekisteröitäessä tulee ottaa myös huomioon Android-alustan sovellusten elinkaari. Tämän takia, kun esittäjä tuhotaan, on tärkeää tuhota myös kuuntelija, jotta vältyttäisiin mahdollisilta muistivuodoilta. Alustan roskienkeräys ei osaa vapauttaa varattua muistia, jos referenssiä kuuntelijaan ei poisteta.

## 5.6 Toteutuksen arviointi

Tutkimuksen tarkoituksena oli esitellä erilaisia malleja ja työvälineitä, joita hyödyntäen offline ensin -periaatteella suunniteltua sovellusta voidaan toteuttaa. Toteutuksen case-esimerkissä käytiin läpi työajanseurantaan liittyvää toiminnallisuutta ja tavoitteena oli kuvata suunnittelumalli, jonka avulla ohjelmakoodi pystytään pitämään helposti ylläpidettävänä ja ymmärrettävänä.

Ylläpidettävyys ja vastuualueiden erottelu on tärkeä osa toteutuksen suunnittelua, sillä ilman sitä voidaan päätyä tilanteeseen, jossa sovellus koostuu tuhansien rivien mittaisista sovellusluokista. Arthur J. Riel (1996) kutsuu tämänkaltaisia toteutuksia nimellä ”God-class”, joilla tarkoitetaan luokkaa, joka toteuttaa huomattavan suuren määrä kehitettävän ratkaisun toiminnollisuuksista.

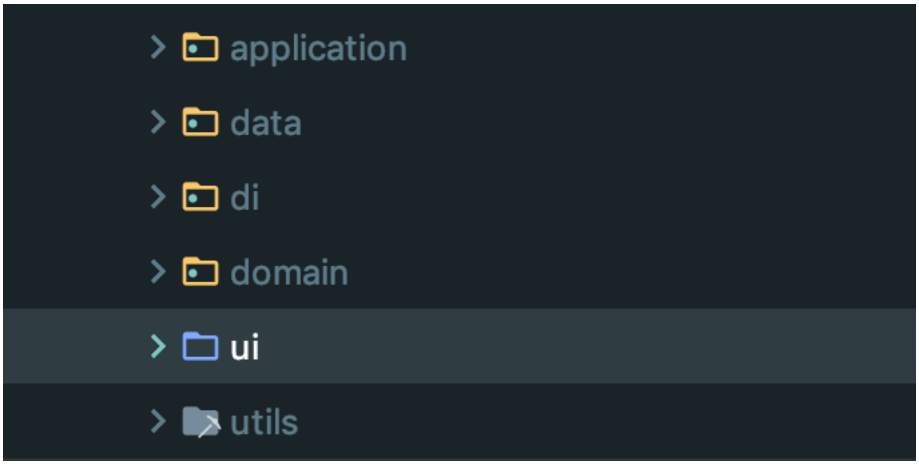
Ylläpidettävyyden kannalta sovelluksen suunnitteluosuudessa päädyttiin erottelemaan projektin osa-alueet eri pakkauksiin. Ratkaisun selkeä osa-alueiden erottelu edesauttaa sovelluksen ylläpidettävyyttä varsinkin kehitystiimeissä, jossa saman ohjelmakoodin parissa työskentelee useita henkilöitä samanaikaisesti.

Ratkaisussa käytetty projektirakenne on esitelty kuviossa 10. Application-pakkaus pitää sisällään sovellustason määritykset. DI- ja Utils -pakkaus pitää sisällään muita sovelluksessa käytettävistä aputoiminnollisuuksista.

Data-paketti pitää sisällään kaiken sovelluslogiikan, joka liittyy käytettäviin tietovarastoihin ja repository-toteutuksiin. Data-paketin taakse on rakennettu kaikki toiminnallisuudet, jotka liittyvät sovelluksessa käsiteltävien tietojen synkronointiin ja sovelluksen tietokannan ylläpitämiseen.

Domain-paketti on erillinen kerros, jonne on koostettu kaikki sovelluksessa käytettävät käyttötapaukset ja rajapintatoteutukset. Käyttötapaukset pitävät sisällään toteutuksen liiketoimintaan liittyvän logiikan.

Käyttöliittymäkerroksessa on kuvattu sovelluksen näkymät, näkymien rajapinnat ja esittäjätoteutukset. Nämä tiedot on paketoitu sovelluksen UI-pakettiin.




Kuvio 10. Projektin rakenne

MVP-suunnittelumallin ja vastuiden jakamisen hyödyt näkyvät varsinkin silloin, kun sovelluksen laadun ja ymmärrettävyyden odotetaan pysyvän samalla tasolla projektia kehittävän tiimin kokemuksesta ja koosta huolimatta. Vastuualueiden selkeä erottelu tarjoaa kehyksen ja prosessin, joka voi helpottaa ohjelmakoodin lukemista, ylläpitämistä ja ymmärtämistä. Sen avulla on mahdollista aloittaa yksinkertaisella ratkaisulla ja lisätä asteittain monimutkaisuutta tarpeen mukaan, ilman että koko toiminnallisuutta joudutaan uudelleenkirjoittamaan heikon ylläpidettävyyden vuoksi.

## 5.7 Jatkotoimenpiteet

Tutkimuksen päätteeksi voidaan pohtia erilaisia asioita, jotka nousivat esiin mobiiliratkaisun kehitysvaiheessa ja joilla voitaisiin iteratiivisesti parantaa toteutettua suunnittelumallia.

Kuten luvussa 5.5 käytiin läpi, asiakassovelluksen toimintaperiaate perustuu sovelluksen tietokannan tilanhallintaan. Tietokannan tietoja synkronoidaan verkkopalvelimen kanssa aina käyttäjän tekemien toimintojen perusteella. Koska REST-rajapinta arkkitehtuuri perustuu pyyntö/vastaus -periaatteeseen, ei tietoja tällä hetkellä synkronoida tilanteissa, jossa käyttäjä ei tee toimenpiteitä sovelluksessa. Tämä voi kuitenkin aiheuttaa ristiriitaisia tilanteita, kun verkkopalvelimen tila ja asiakassovelluksen tietokanta eivät ole synkronoituja keskenään. Tämänkaltainen tilanne voi

tulla vastaan esimerkiksi käyttäjän tehdessä kirjauksen hyödyntäen jotain toista päätelaitetta.

Edellä mainituilta tilanteilta ei voida välttyä offline ensin -toteutuksissa, mutta tilanteen välttämiseksi voidaan hyödyntää erilaisia suunnittelumalleja. Yksi malli, jota voidaan hyödyntää ongelman ratkaisemiseksi, on taustalla ajettavat synkronointioperaatiot. Synkronointioperaatioita voidaan ajastaa hyödyntäen luvussa 4.3 esiteltyä WorkManager-työvälinettä. Ajastettujen operaatioiden avulla voidaan ratkaista ongelma, joka voi tapahtua tilanteissa, joissa käyttäjä ei tee asiakassovelluksen kanssa mitään toimenpiteitä, jotka aiheuttaisivat tietojen synkronoinnin.

Taustalla ajettavien operaatioiden heikkoudeksi voidaan laskea se, että dataa kysytään myös tilanteissa, jossa se ei ole lainkaan muuttunut verkkopalvelimen päässä. Synkronointioperaatiot käyttävät mobiililaitteen kaistaleveyttä ja kuluttavat laitteen akkua. Operaatiot aiheuttavat kuormaa myös verkkopalvelimille, jos synkronoitavien laitteiden määrä on huomattavan suuri.

Synkronointihaasteen selvittämiseksi voidaan käyttää myös suunnittelumallia, jossa hyödynnetään push-ilmoituksia. Push-ilmoitukset voidaan toimittaa käyttäjän laitteelle esimerkiksi hyödyntäen Googlen tarjoamaa Firebase Cloud Messaging -palvelua.

Suunnittelumallissa käyttäjien laitteet rekisteröidään Firebase Cloud Messaging -palveluun. Rekisteröidyille laitteille luodaan laitekohtainen tunniste, joka liitetään käyttäjän perustietoihin. (Google 2019d.) Kun verkkopalvelimen tila muuttuu, voidaan muuttuneen tiedon perusteella välittää tieto eteenpäin tilamuutoksesta kiinnostuneiden käyttäjien laitteille. Ilmoituksen mukana toimitetaan normaalisti ainoastaan tieto synkronoinnin tarpeesta verkkopalvelimen kanssa

# 6 YHTEENVETO

Tutkimuksen tavoitteena oli esitellä erilaisia suunnitteluperiaatteita, joita hyödyntäen voidaan lähteä rakentamaan mobiiliratkaisuja offline ensin -suunnitteluperiaatteella. Tutkimuksessa käytiin lävitse erilaisia ohjelmistoarkkitehtuurisia ratkaisuja ja saatavilla olevia työvälineitä, joita voidaan hyödyntää, kun ratkaistaan tuotantokäyttöön suunnitellun mobiiliratkaisun skaalautuvuuteen ja ylläpidettävyyteen liittyviä ongelmia. Tutkimuksessa esiteltiin artefakti, jossa kuvattiin offline-toiminnallisuuden ratkaisu työajanseurannan toiminnallisuudessa.

AppDynamics (2014) teetti tutkimuksen, jossa haastateltiin noin 2000 käyttäjää Yhdistyneissä Kuningaskunnissa ja Yhdysvalloissa. Tutkimuksessa selvisi, että yli kahdeksan kymmenestä käyttäjästä on poistanut sovelluksen johtuen sovelluksen suorituskykyongelmista.

Offline ensin -arkkitehtuurin valinta perustuu usein ratkaistavan ongelman määrityksiin. Kuluttajille suunnatuissa sovelluksissa kannattaa kuitenkin aina panostaa käyttökokemukseen ja käytettävyyteen. Jos tiedetään, että verkkopalvelimilla voi olla paljon liikennettä ja siellä voi tapahtua suunniteltuja tai suunnittelemattomia käyttökatkoja, asiakassovellus kannattaa optimoida välttämään raskaampia pyyntöjä verkkoresursseihin. Verkkopalvelujen ollessa alhaalla käyttäjä ei välttämättä edes ehdi huomaamaan, että palvelun tilassa on tapahtunut muutosta, kun sovellus on suunniteltu offline ensin -periaatteita hyödyntäen.

Tutkimuksessa esitellyistä suunnitteluperiaatteista välimuistin käyttäminen offline-haasteen ratkaisemiseksi sopii toteutuksiin, jossa halutaan taata keskeytymätön pääsy sisältöön, jonka oletetaan olevan jo saatavilla. Laajemman toiminnallisuuden toteuttaminen vaatii erilaisten suunnitteluperiaatteiden noudattamista, jotka on hyvä ottaa huomioon sovelluksen suunnitteluvaiheessa. Niiden lisääminen jälkikäteen aiheuttaa usein merkittäviä muutoksia koko sovellusrakenteeseen. Jotta sovellusta ei tulisi suunnitella uudelleen määrityksien muuttuessa, on tärkeää hyödyntää suunnitteluperiaatteita, jotka kannustavat selkeään vastuualueiden erotteluun.

LÄHTEET

Andreu, C, 2014. IBM Worklight - Offline Patterns. [viitattu 15.3.2019]. Saatavissa: https://www.ibm.com/developerworks/community/blogs/worklight/entry/offline_patterns?lang=en

AppDynamics Inc. 2014. The App Attention Span. [viitattu 15.3.2019]. Saatavissa: https://www.appdynamics.com/media/uploaded-files/1425406960/app-attention-span-research-report-1.pdf

Bainomugisha, E., Carreton, A., Van Cutsem, T., Mostincks, S. & De Meuter, W. 2012. A Survey on Reactive Programming. [Viitattu 20.4.2019]. Saatavissa: http://soft.vub.ac.be/Publications/2012/vub-soft-tr-12-13.pdf

Feyerke, A. 2013. Designing Offline-First Web Apps. [viitattu 25.3.2019]. Saatavissa: https://alistapart.com/article/offline-first

Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. & Berners-Lee, T. 1999. Hypertext Transfer Protocol -- HTTP/1.1. [viitattu 5.4.2019]. Saatavissa: https://tools.ietf.org/html/rfc2616#section-9

Fielding, R. 2000. Architectural Styles and the Design of Network-based Software Architectures. [viitattu 22.4.2019]. Saatavissa: https://www.ics.uci.edu/~fielding/pubs/dissertation/introduction.htm

Fowler, M. 2002. Patterns of Enterprise Application Architectures. Addison-Wesley Professional

Google. 2019a. Offline First. [viitattu 23.2.2019]. Saatavissa: https://developer.chrome.com/apps/offline_apps

Google. 2019b. Data and file storage overview. [viitattu 15.3.2019]. Saatavissa: https://developer.android.com/guide/topics/data/data-storage

Google. 2019c. Save data in a local database using Room. [viitattu 15.3.2019]. Saatavissa: https://developer.android.com/training/data-storage/room/index.html

Google. 2019d. Firebase Cloud Messaging. [viitattu 22.4.2019]. Saatavissa: https://firebase.google.com/docs/cloud-messaging/

Google, 2019e. Schedule tasks with WorkManager. [viitattu 30.3.2019]. Saatavissa: https://developer.android.com/topic/libraries/architecture/workmanager

Holmström, P. 2015. Is MVP a Best Practice. [viitattu 28.4.2019]. Saatavissa: https://vaadin.com/blog/is-mvp-a-best-practice-

Kurose, J. & Ross, K. 2012, Computer Networking A Top-Down Approach. 6 painos. Harlow: Pearson Education.

Megali, T. 2016. An Introduction to Model View Presenter on Android. [viitattu 2.4.2019]. Saatavissa: https://code.tutsplus.com/tutorials/an-introduction-to-model-view-presenter-on-android--cms-26162

Mozilla. 2019. MDN Web Docs. [viitattu 2.3.2019]. Saatavissa: https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/MIME_types

Nurkiewicz, T. & Christensen, B. 2016. Reactive Programming with RxJava. O'Reilly Media, Inc.

Oxford Economics & Samsung. 2018. MAXIMIZING MOBILE VALUE: IS BYOD HOLDING YOU BACK? [viitattu 19.2.2019]. Saatavissa: https://www.samsung.com/us/business/short-form/maximizing-mobile-value/?cid=com-btb-sky-blg-122001

Peffers, K., Tuunanen, T., Rothenberger, M. & Chatterjee, S. 2007. A Design Science Research Methodology for Information Systems Research. [viitattu 3.5.2019]. Saatavissa: http://search.ebscohost.com.aineistot.lamk.fi/login.aspx?direct=true&db=bsh&AN=28843849&site=ehost-live

ReactiveX. 2019a. The Observer pattern done right [viitattu 21.4.2019]. Saatavissa: http://reactivex.io/

ReactiveX. 2019b. ReactiveX. [viitattu 21.4.2019]. Saatavissa: http://reactivex.io/intro.html

Realm. 2017. The Offline-First Approach to Mobile App Development. [viitattu 15.3.2019].
Saatavissa: https://www2.realm.io/whitepaper/offline-first-approach-registration

Riel, A. 1996. Object-Oriented Design Heuristics. [viitattu 23.4.2019]. Saatavissa: https://learning.oreilly.com/library/view/object-oriented-design-heuristics/020163385X/ch03.html

Sauble, D. 2015. Offline First Web Development. Packt Publishing Ltd.

Sqlite. 2019. About SQLite. [viitattu 28.3.2019]. Saatavissa: https://www.sqlite.org/about.html