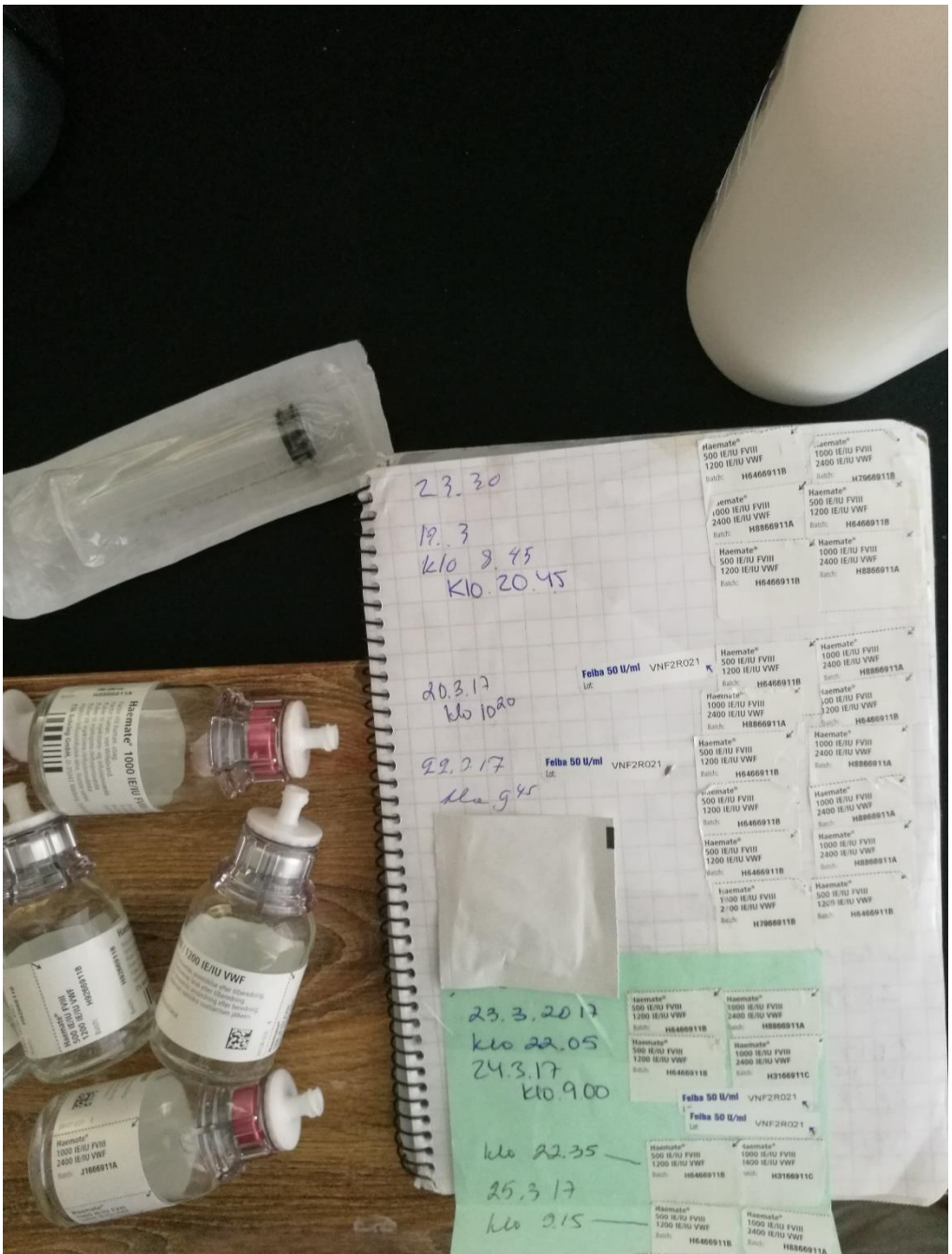
Jesse Kallio

Android-sovellus lääkkeenannon seurantaan



Tutkintonimike: Insinööri (AMK)

Koulutus: Tietotekniikka

Kevät 2017

KAJAANIN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

KAJAANIN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Tiivistelmä

Tekijä(t): Kallio Jesse

Työn nimi: Android-sovellus lääkkeenannon seurantaan

Tutkintonimike: Insinööri (AMK), tietotekniikka

Asiasanat: Ohjelmointi, Android, Android Studio, Java, C++, Tesseract-OCR, OCR, Kamera

Työn tarkoituksena oli auttaa ihmisiä, joilla on jatkuva ja kallis lääkitys, auttamalla sen seurannassa ja kirjaamisessa. Sovelluksen kohderyhmänä olivat A-hemofiliaa sairastavat vasta-aine potilaat, koska heidän lääkityksensä vaativat päivittäistä lääkitsemistä ja seurantaa. Vaikka kohderyhmä projektin alussa oli varsin rajattu, pyrittiin sovellus luomaan mahdollisimman yleiseksi, tällä tavoin mahdollistetaan sovelluksen jatkokehitys muita lääkinnällisiä sairauksia varten.

Sovelluksen kehitykseen käytettiin Android Studiota, jonka käyttö ei ollut kehitystyön alussa tuttua. Ohjelmointikielenä käytettiin Java -ohjelmointikieltä, joka myöskin vaati omaksumista koska käyttökokemukset ja taidot kielestä olivat vähäiset.

Työn tuloksena saatiin toimiva prototyyppi sovelluksesta, joka pystyy tallentamaan potilaan nimen, ja otetut lääkkeet, joko kirjoittamalla tai puhelimen kameraa hyväksi käyttäen, vaikkakin tekstin tunnistus ei vielä toimi riittävällä tarkkuudella.

KAJAANIN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Abstract

Authors: Kallio Jesse

Title of the Publication: Android Application for Tracking Medicine Distribution

Degree Title: e.g. Bachelor of Engineering, Information Technology Engineering

Keywords: Programming, Android, Android Studio, Java, C++, Tesseract-OCR, OCR, Camera

The purpose of this Bachelor's thesis was to help people with constant expensive medication to control and keep track of their medicine distribution. Primary focus during the creation of this application was patients that are suffering from difficult A hemophilia with inhibitors, since their treatment requires daily medication. Although the target group during the creation of this project was very narrow, the application was supposed to be as generic as possible, so that the application can be further developed for other health problems that require medical treatment.

The development environment during this project was Android Studio which was not familiar at the beginning. Programming language that was used was Java, which also required getting used to, since experience and skills on the language were minimal.

As result of this thesis a working prototype was achieved, which can store patient name, taken medication either by typing or with a camera of a phone, although text recognition is not yet working on the required level.

Sisällys

LYHENTEET JA MÄÄRITELMÄT

Android Studio	Ohjelmointiympäristö Android ohjelmointiin
API Level	Kokonaislukuarvo joka määrittää millaisia Android ominaisuuksia voidaan käyttää sovelluksen kehittämiseen (16)
AVD	Virtuaalinen puhelin, jolla voidaan testata sovelluksen toimivuutta (Android Virtual Device)
IDE	Ohjelmisto, joka mahdollistaa sovellusten kehityksen eri alustoille (Integrated Development environment)
Infuusio	Tiputus lääkkeenannossa
OCR	Sovelluksen lisäosa jolla voidaan muuntaa kuva tekstiviraksi (Optical Character Recognition)
SDK	Ohjelmiston kehitys paketti (Software Development Kit)

1 Johdanto

Tämän opinnäytetyön tarkoituksena oli luoda Android-sovellus prototyyppi, joka helpottaisi päivittäisiä lääkkeitä ottavien ihmisten elämää. Tarve sovellukselle syntyi omasta tarpeesta, kun joudun päivittäin lääkitsemään sekä kirjaamaan annetut lääkkeet. Lääkkeiden kirjaamiseen on tällä hetkellä käytössä perinteinen kierrevihko ja kynä, sekä lääkepullojen kyljestä eränumero tarra. Itse lääkkeenantotoimenpiteeseen kahden lapsen kohdalla kuluu noin 45 minuuttia. Lääke annetaan normaalisti kaksi kertaa päivässä, mutta jos lapsella on vuoto, niin peruslääkkeen lisäksi, tarvitaan vuotolääke, joka annetaan noin puolituntia kestävänä infuusiona. Vuotolääkkeen antamisen kokonaiskesto on noin 50 minuuttia. Joskus lääkkeet jäävätä merkitsemättä heti, koska se vaatii vihon esille ottamisen ja tarrojen repimistä ja käsienpesun tämän jälkeen. Edellä mainitusta syystä, merkinnät tehdään melko usein vasta seuraavalla lääkkeellä, tätä prosessia halusin yksinkertaistaa, niin että napataan vain puhelin käteen ja näppäillään eränumero, jonka jälkeen sovellus ehdottaa ajankohdan jolloin lääke on annettu, näin voitaisiin säästää edes hieman aikaa ja vaivaa. Lisäksi lääkkeiden merkintä puhelimella helpottaisi myös matkustusta ja lääkärikäyntejä, koska usein juuri lääkevihot unohtuvat. Mutta jos lääkitykset löytyisivät puhelimesta, olisi lääkkeet merkittävissä ja tiedot luettavissa huomattavasti todennäköisemmin.

Lähdin tekemään työtäni sillä periaatteella, että samalla tulisi oppia uutta, tästä syystä työssä käydään myös läpi työkalujen kuten Android Studio ja Java ohjelmoinnin opiskelun vaiheita sekä käydään läpi myös oleellisten ohjelmakoodien toiminnat ja tärkeimmät funktiot, joiden avulla työ on toteutettu.

Valmiin työn oli tarkoitus olla prototyyppi, jolla testataan kameran käyttöä tekstin lukemisessa sekä työn tuli ”opettaa tekijäänsä” paremmaksi ohjelmoijaksi ja lisätä kokemusta mobiiliohjelmoinnista. Näihin edellä mainittuihin tavoitteisiin päästiin, saatiin paljon kokemusta Android alustasta ja Java ohjelmointi Android ympäristössä tuli tutuksi.

2 Tausta ja aiempi tutkimus

Kroonisten sairauksien hoitamiseen liittyy usein kotihoitoa, mittauksia ja seurantaa. Tämä vaatii tietojen kirjaamiseen erilaisia menetelmiä. Perinteisesti esimerkiksi astman hoidossa tulokset kirjataan paperitaulukkoon tai vihkoon, jonka mukaan lääkitystä voi joko muuttaa itsenäisesti tai lääkärin ohjeistamana. (1, 2.)

Hoidon tueksi on kehitetty erilaisia digitaalisia testejä ja sovelluksia, kuten astmatesti (3) tai kroonista urtikariaa sairastaville Urtikaria Nyt! –sovellus (4). Harvinaisemmat sairaudet, kuten hemofilia, vaativat tarkkaa seurantaa. Hemofilian hoidon kulmakivi on puuttuvan hyytymistekijän korvaushoito (5). Tähänkin on kehitetty joitakin sovelluksia, mutta useimmat ovat saatavilla englanniksi: MicroHealth Hemophilia (Appstore), HemMobile ja niiden käyttö tarkoitus on henkilökohtaisten lääkitysten seuranta eikä useamman henkilön esim. lapsien.

Teixeira et al. kuvaavat artikkelissaan *”Improvement of Surveillance of Hemophilia Treatment through ICTs”* kuinka hemofiliapotilaiden hoitoa voidaan parantaa tietojenvälitys sovelluksilla. Hemo@care on ohjelma, jolla voidaan paremmin koordinoida lääkärien, hoitajien ja potilaiden yhteistyötä. Ohjelmalla voidaan seurata hemofiliapotilaan vuotoherkkyyttä ja muokata potilaan profylaksia hoitoa tarpeen vaatiessa. Se auttaa myös seuraaman potilaan kotihoitoa mahdollisten väärinkäytöstenvaralta ja auttaa hemofilian hoitokeskusten (Hemophilia Diagnostic and Treatment Centers) toimintaa hyytymistekijälääkkeen tilausten tekemisessä. (6.)

ManageMyCondition artikkelissa todetaan mobiililaitteiden määrän ja saavutettavuuden olevan sitä luokkaa, että se luo valtavasti mahdollisuuksia älypuhelinsovelluksille, joilla voidaan tehokkaasti tukea kroonisten sairauksien hoitoa. Artikkeli myös painottaa myös yhtenäisen viitekehyksen luomisen sovellusten kehittämiselle, jotta pystyttäisiin paremmin yhdistää erisovelluksista saadut tiedot tutkijoiden ja terveydenhuollonammattilaisten käyttöön (7.)

Mobiilisovellukset tuovat uusia mahdollisuuksia eri sairauksien seurantaan ja hoitoon. Esimerkiksi Australiassa tutkittiin ensimmäisen asteen diabetesta sairastavalla kohderyhmää. Niillä, jotka käyttivät älypuhelinsovellusta ja saivat tekstiviestejä kliinikoiltaan sekä

jatkuvaa terveydenhoitoa, verensokeritasot olivat alhaisemmat yhdeksän kuukauden jälkeen, kuin niillä, jotka eivät käyttäneet edellä mainittuja asioita apunaan. Jatkossa tavoitteena onkin reaaliaikainen seuranta sekä palautteen anto potilaan ja lääkärin välillä (8).

3 Android-sovelluksen kehittäminen

Andy Rubin perusti Android osakeyhtiö vuonna 2003 ja yrityksen tarkoitus oli valmistaa kehittyneempi käyttöjärjestelmä digitaalisille kameroille. Mutta markkinaosuuksien pienuuden takia kehityskohde vaihdettiin älypuhelimiin. Vuonna 2005 yritys joutui taloudellisiin ongelmiin, jolloin Google osti sen ja ensimmäinen kaupallinen versio julkaistiin vuonna 2008 (9).

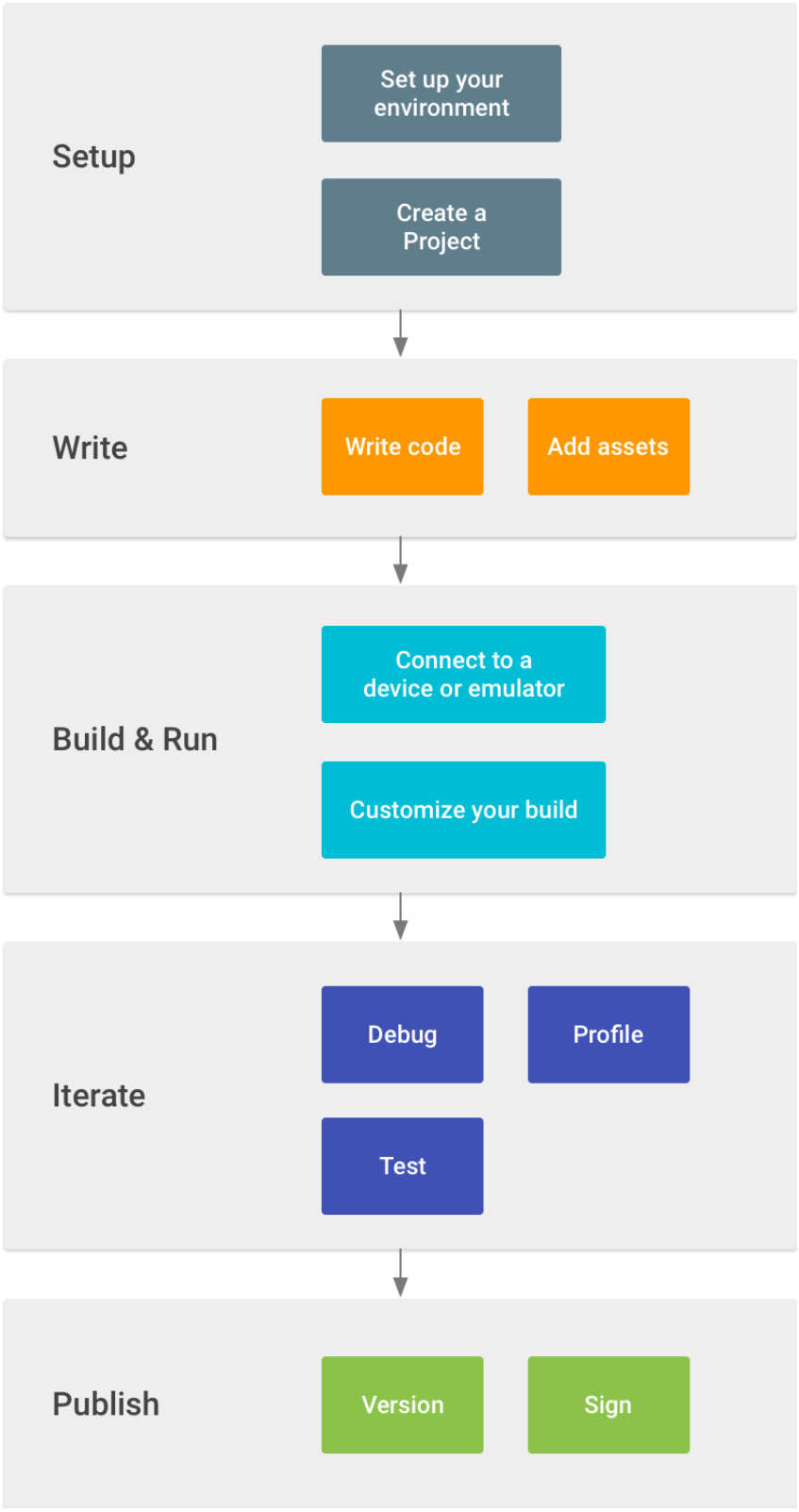
Android on Linux -pohjainen käyttöjärjestelmä (10), mobiilialustoille, jota käytetään pääasiassa puhelimissa ja tableteissa, mutta jonka voit löytää lähes laitteesta kuin laitteesta autoista rannekelloihin.

Tällä hetkellä Android on suosituin mobiilikäyttöjärjestelmä maailmassa (11). Suurin syy sen suosioon on sen avoin lähdekoodi ja muokattavuus, minkä ansiosta puhelinvalmistajat voivat muokata käyttökokemuksen sellaiseksi kuin haluavat. Lisäksi kaikki Android puhelimet tukevat Android sovelluksia joita voidaan ladata Google Play kaupasta tai suoraan kehittäjien verkkosivuilta (esim. Veikkaus.fi/fi/sovellukset).

3.1 Androidille ohjelmointi

Android sovellukset kirjoitetaan Java -ohjelmointikielellä. Android SDK työkalut kääntävät ohjelmointikoodin ja kaikki datat sekä resurssit APK:ksi, joka on arkistotiedosto päätteellä .apk. Yksi APK tiedosto sisältää kaiken Android sovelluksen sisällön ja toimii tiedostona jota kaikki Android käyttöiset laitteet käyttävät kyseisen sovelluksen asentamiseen (12).

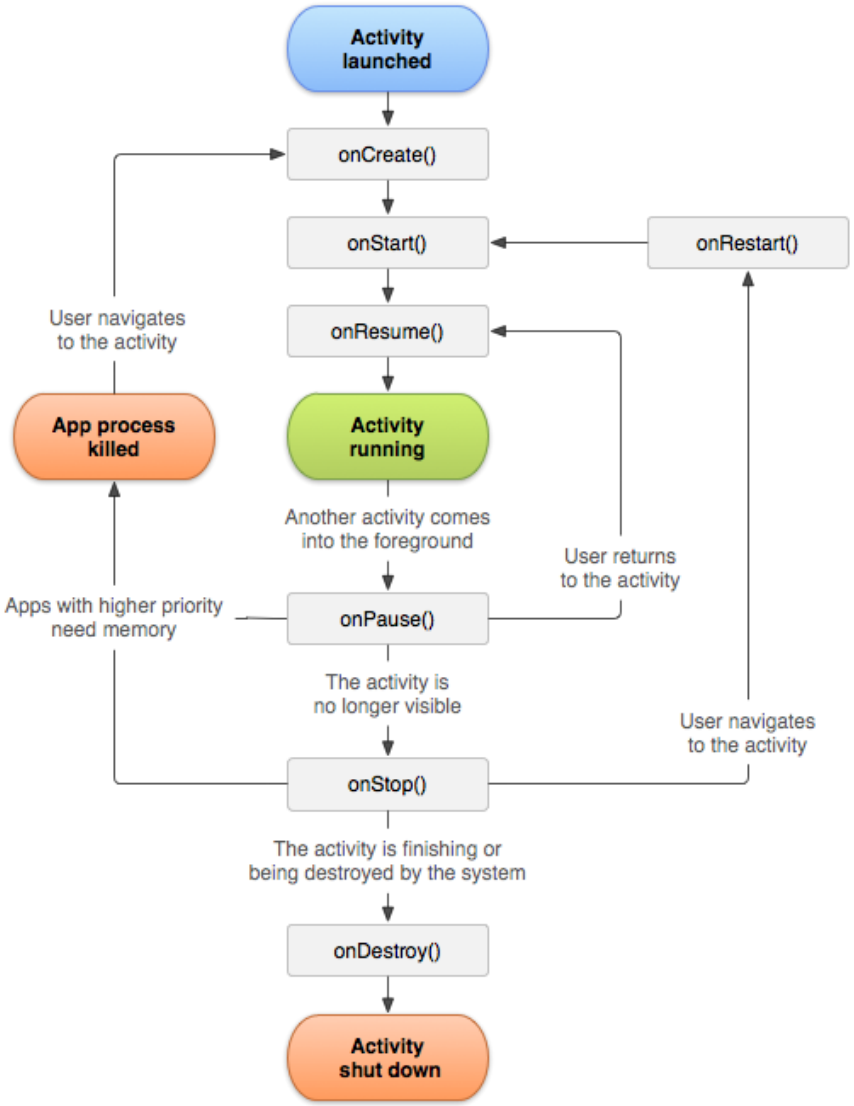
Android-sovelluksen tekeminen on hyvin samakaltaista, kuin minkä tahansa muun ohjelman. Kuten kuva 1 (13) osoittaa, ensimmäiseksi valmistellaan työympäristö, IDE. jonka jälkeen luodaan projekti, johon kirjoitetaan ohjelmakoodi ja johon lisätään komponentteja ja resursseja tarpeen mukaan. Tämän jälkeen ohjelma ajetaan laitteella tai emulaattorilla, jotta sitä voidaan testata ja etsiä virheitä. Lopuksi ohjelmasta julkaistaan ensimmäinen versio.



Kuva 1. Perus Android projektin kehitysmalli (12)

3.2 Sovelluksien elinkaari

Suurimmassa osassa tapauksista, jokainen Android sovellus toimii omassa Linux prosessissaan, tämä prosessi (Aktiviteetti) luodaan, kun jotain osaa sovelluksen koodista pitää suorittaa. Prosessia suoritetaan niin kauan, kuin tarvitaan ja järjestelmä ei tarvitse prosessin käyttämää muistia toiselle sovellukselle (15). Käyttäjän navigoidessa sovelluksessa aktiviteetti instanssit käyvät läpi elinkaarensa eri tiloja (kuva 2).



Kuva 2. Aktiviteetin elinkaari (14)

3.3 Kehitysympäristöt

Tässä osiossa on kuvattu mahdollisia kehitysympäristöjä. Aikaisempaa kokemusta minulla oli Eclipsen ja Qt:n kehitysympäristöistä.

3.3.1 Android Studio

Googlen lanseeraama, 16. toukokuuta 2013 julkaistu (16), Android-käyttöjärjestelmän virallinen ohjelmointiympäristö. Android Studiolla voidaan luoda ohjelmia kaikkiin laitteisiin, joissa on Android käyttöjärjestelmä. Ohjelmointi kielenä on Java.

3.3.2 Eclipse Java IDE

Eclipse Java IDE on Eclipse Foundationin ylläpitämä avoimen lähdekoodin Android kehitysympäristö. Ohjelmointi kielenä toimii Java ja kehitysympäristö on täysin muokattavissa omiin tarpeisiin lisäosia käyttämällä (17). Eclipsen käyttöön löytyy runsaasti materiaalia verkosta.

3.3.3 Qt Creator

The Qt Companyn Qt Creator on järjestelmärippumaton sovellusten kehitysympäristö tietokoneelle, sulaututeille järjestelmille ja mobiililaitteille. Qt:ssa voidaan ohjelmoida C++ -kieltä tai QML -komentosarjakieltä käyttäen (18).

3.4 Ohjelmointi Android -ympäristössä

Android -ympäristössä ohjelmointi tapahtuu pääsääntöisesti Java -kielen avulla, tästä johtuen Java -kielen taito on välttämätöntä. Java on Sun Microsystemsin vuonna 1995 julkaisema ohjelmointikieli.

Android sovellukset toimivat itsenäisissä virtuaalikoneissa, ne eivät tiedä toisistaan ilman erillisiä toimenpiteitä. Tämä täytyy pitää mielessä, jos halutaan käsitellä dataa jostain toisesta sovelluksesta. Android ohjelmoinnissa voidaan välttää törmäämistä odottamattomiin ongelmiin, kun ymmärretään perus-sovelluksen elinkaari kts. (3.2) ja tiedetään mitä aktiviteettia kutsutaan ja milloin.

3.5 Ohjelmisto rakenne

Kuten kuvassa 3 näkyy, Android on Linux pohjainen ohjelmistopino. Alimmaisena on Linux kernel, joka pitää huolen toiminallisuuksista, kuten säikeiden ajamisesta ja alhaisen muistin hallinnoinnista. Kernel mahdollistaa tärkeiden turvaominaisuuksien käytön ja laitevalmistajien ajureiden kehittämisen tunnetulle kernelille.

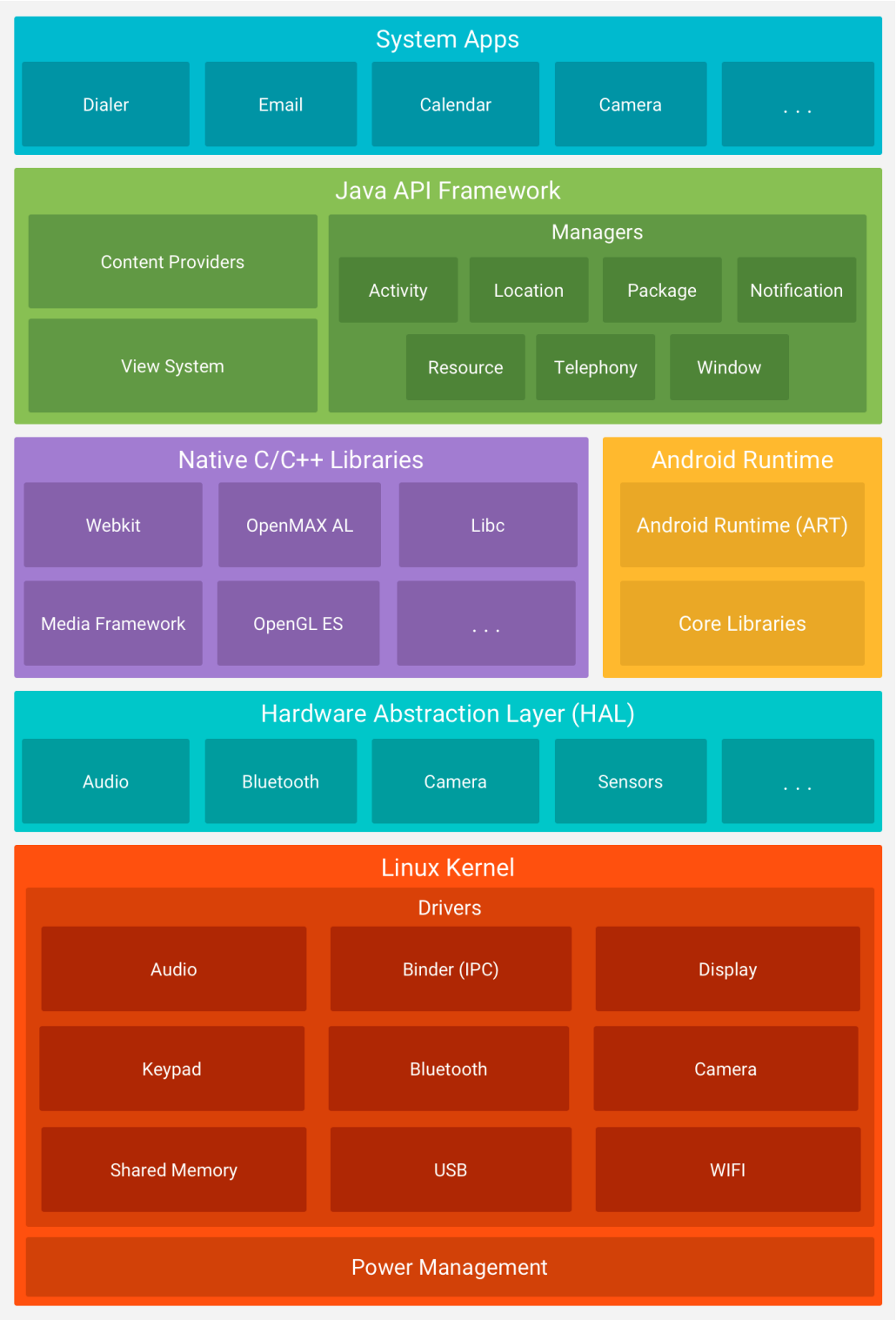
Seuraavassa kerroksessa on laite abstraktointi (HAL). Kun API kehikko kutsuu laitteen tiettyä osaa, Android systeemi lataa kirjastomoduulit sitä laitteistokomponenttia varten.

Android ajokerros (ATR) laitteille, jotka käyttävät versiota 5.0 (API taso 21) tai korkeampaa, käyttävät jokaisen sovelluksen ajossa omaa prosessia jolla on oma instanssi ART:sta

C/C++ kirjastot monet Androidin ydinkomponenteista ja palveluista, kuten ART ja HAL, on rakennettu käyttäen kirjastoja, jotka on kirjoitettu C ja C++ -kielillä. Android alusta sisältää Java API rajapintoja, jotka voivat sallia toimintoja joistain näistä kirjastoista sovelluksille.

Java API rajapinnat (Java API Framework) mahdollistavat kaikkien Android käyttöjärjestelmän ominaisuuksien käytön. API rajapinnat muodostavat rakennusosat, joita Android sovelluksen luomiseen tarvitaan, yksinkertaistamalla ydin- ja modulaaristenkomponenttien, sekä palveluiden käyttöä.

Järjestelmäsovellukset (System Apps) ovat sovelluksia, jotka tulevat alustan mukana, mutta niillä ei ole mitään erityisasemaa. Järjestelmä sovellukset toimivat käyttäjällä normaaliin tapaan, mutta voivat tarjota kehittäjälle tärkeitä toimintoja käytettäväksi kehitettävään sovellukseen, kuten esimerkiksi kamerasovelluksen käyttö. (19).



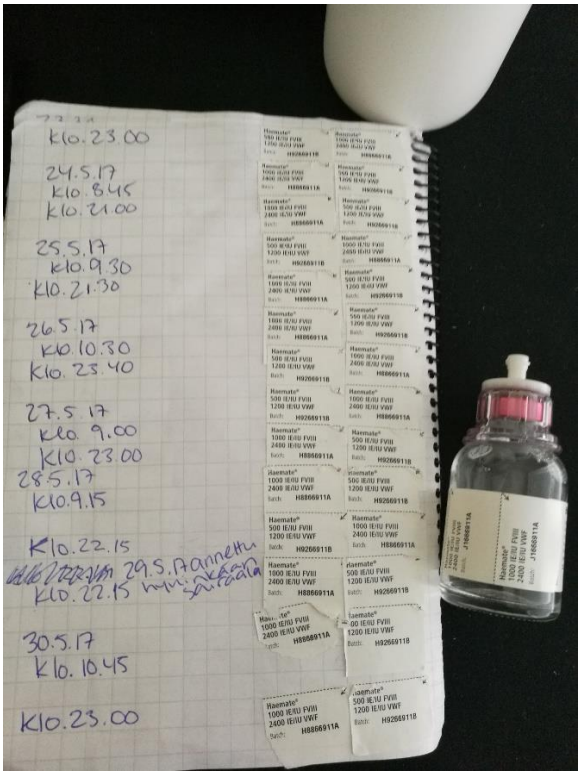
Kuva 3. Android ohjelmistopino (10)

4 Sovelluksen suunnittelu lääkkeenannon seurantaan

Suunnittelussa ja tavoitteiden asettamisessa otettiin huomioon käytettävä aika ja mahdolliset ongelmat, joita saatetaan kohdata. Tässä kappaleessa käydään läpi sovelluksen suunnittelu, aikataulutus ja tavoitteet.

4.1 Ohjelman suunnittelu

Sovelluksen idea syntyi omakohtaisesta ja muilta vastaavanlaisessa tilanteessa olevien tarpeesta, helpottaa päivittäisten lääkeannosten kirjaamisessa. Tällä hetkellä isoin ongelma on eränumerotarrojen irrottaminen lääkepullojen kyljestä ja kirjaaminen vihkoon (kuva 4). Jotta sovellus olisi hyödyllinen korvike samaiselle toiminnalle, sen tulisi olla mahdollisimman nopea- ja helppokäyttöinen, tästä syntyikin idea puhelimen kameran ja aikaleiman käyttämiseen lääkkeiden merkitsemisessä. Tällöin ei tarvittaisi mitään mekaanista merkintää, jolloin säästettäisiin aikaa.



Kuva 4. Esimerkki lääkevihkosta ja pullosta

Prototyypin tavoite vähimmillään on kirjata sovelluksen muistiin potilaan nimi ja lääkityksen eränumero ja kellonaika sekä päivämäärä. Lisäksi sovelluksen tulisi ehdottaa eränumeroa puhelimen kameraa ja Tesseractin OCR:ää käyttäen.

Valmiin sovelluksen tulisi olla käyttäjän apu lääkkeiden oton seurannassa ja muistamisessa. Sovellukseen voitaisiin tallentaa potilaan nimi, lääkitys, eränumerot, kellonaika, päivämäärä ja puhelimen kameralla voitaisiin kaapata tietoja suoraan lääkepakkausten etiketeistä. Lisäksi sovellus muistuttaisi käyttäjää ottamaan/antamaan lääkkeensä.

Kuvion 1 taulukossa on kuvattu valmiinsovelluksen vaatimusten priorisointi asteikolla 1-3 joista 1. on ehdoton, 2. vaadittu ja 3. ei välttämätön.

Nro	Vaatimus	Prioriteetti
1	Nopeakäyttöinen	1
2	Helppokäyttöinen	1
3	Kameran hyödyntäminen	3
4	Useampi potilas profiili	1
5	Tiedot varmuuskopioitu pilvipalveluun	1
6	Tietojen tulostus / lähettäminen	2

Kuvio 1. Sovelluksen vaatimukset

4.1.1 Aikataulutus ja kehitysympäristöjen valinta

Aikataulutuksessa käytettiin Microsoftin Excel taulukko ohjelmaa, jolle luotiin aikataulutus viikkotasolla. Aikataulun tavoitteet luotiin olettamaan, että projektissa ei tulisi ilmenemään paljon aikaa vieviä ongelmia.

Työvälineiden valintaan vaikuttivat kiinnostukseni uuden oppimiseen, siksi valitsin työkaluikseni välineitä joita en ollut aiemmin käyttänyt tai olin käyttänyt vähäisesti ja jotka olivat kiinnostavia. Tiesin valitessani ohjelmointiympäristöksi Androidin ja Android Studion, että joudun opettelemaan perustavammin Java -ohjelmointia, jota en aikaisemmin ollut juuri käyttänyt. Olin myös kiinnostunut selvittämään kuinka nopeasti uuden kielen omaksuminen käy, kun aikaisempi ohjelmointi taito nojautuu pelkästään C, C++, C# -kieliin.

Google Tesseract OCR oli ennestään tuttu vain nimeltä ja käyttötarkoitukselta, mutta mielestäni se sopi erinomaisesti työni tarkoitukseen, koska kyseiseen tekstinlukumoottoriin löytyi runsaasti materiaalia verkosta.

5 Android ohjelmointi

Tässä osiossa käydään läpi Android ohjelmoinnissa tarvittavia perustaitoja, kuten käytettäviä tiedostoja, hyödyllisimmät olio-luokat, kehitysympäristön opettelu ja oleellisimmat harjoitukset jotka tehtiin.

5.1 Android ohjelmoinnin perusteet

Android sovellus koostuu pääasiassa Aktiviteeteista, palveluista, lähetysten vastaanottajista (broadcast receiver) ja sisällöntoimittajista (content provider).

Aktiviteetit ovat se osa joka tuo käyttäjälle mahdolliset toiminnot ja suorittaa sovelluksen ruudulla näkyviä toimintoja joihin käyttäjä voi olla vuorovaikutuksessa. Taitava aktiviteettien hallinta takaa sujuvan käyttöliittymän, sovelluksen tietojen säilymisen eri tilanteissa ja että sovellus lopettaa prosessit, kun niin sallitaan.

Palvelut ovat puolestaan yleiskäytännöllinen tapa pitää sovellus pyörimässä taustalla tai päivittää dataa mistä syystä tahansa ilman aktiviteettia, joka saattaisi häiritä käyttäjää. Tapoja, joilla palveluita voidaan käyttää, ovat esimerkiksi musiikin soittamista taustalla, vaikka sovellus suljetaan, tai datan hakuun verkosta, vaikka sovellus ei olisikaan käynnissä. Broadcast receiver on puolestaan komponentti, joka mahdollistaa systeemin toimittaa tapahtumia sovelluksen normaalin tapahtumaketjun ulkopuolelta.

Content provider hallitsee asetettua osaa sovelluksen datasta, joka voidaan säilöä tiedosto järjestelmään, kuten SQLite datapankkiin, verkkoon tai mihin tahansa pysyvään tallennus sijaintiin. Content providerin kautta toiset sovellukset voivat tiedustella tai hallita tallennettua dataa, jos sisällön tarjoajat sen sallivat. (20).

Näiden em. ominaisuuksien lisäksi Androidilla on hyödyllinen olio nimeltä intent, jonka avulla voidaan lähettää ”pyyntöjä” systeemin eri osa-alueille. voidaan esimerkiksi luoda intent-olio, joka kertoo systeemille haluavansa käyttää kamerasovelluksen kameraominaisuutta.

Gradle on Android Studiossa toimiva koontiversion luomiseen erikoistunut lisäosa, joka pitää huolen, että käyttäjän ei tarvitse erikseen lisätä sovelluksen tarvitsemia kirjastomoduuleita. Riittää, että kehittäjä lisää kirjaston Android Studiossa olevaan Gradle.build tiedostoon (21).

Jokaisella sovelluksella pitää olla korkeimmalla käyttäjätasolla AndroidManifest.xml tiedosto. Manifesti antaa Android systeemille kaikki tarvittavat tiedot, joita sovelluksesi tarvitsee ennen kuin sovelluksen ohjelma koodi ajetaan (20).

5.2 Android Studion käytön opettelu

Koska Android Studion tai Java-kielen käytöstä ei ollut aikaisempaa kokemusta, päätin ohjelman opettelun tapahtuvan Android for Programmers (22) kirjan avulla, koska halusin, että ohjelmointi osiot selitettäisiin tarkemmin kuin mitä Android Studion omilla sivuilla. Kirjan ohjeita seuraamalla tein neljä harjoitusta, harjoitukset eivät olleet varsinaisesti Android Studiolle, vaan Eclipselle, joka on Android Studiota vastaava kehitysympäristö. Harjoituksien seuraaminen ei ollut aivan niin mutkatonta kuin olin olettanut, johtuen käyttöliittymän eroista, mutta useimmat ongelmat ratkesivat tovin niitä pähkäiltyä.

5.2.1 Harjoitussovellukset ja niistä opitut asiat

Welcome App

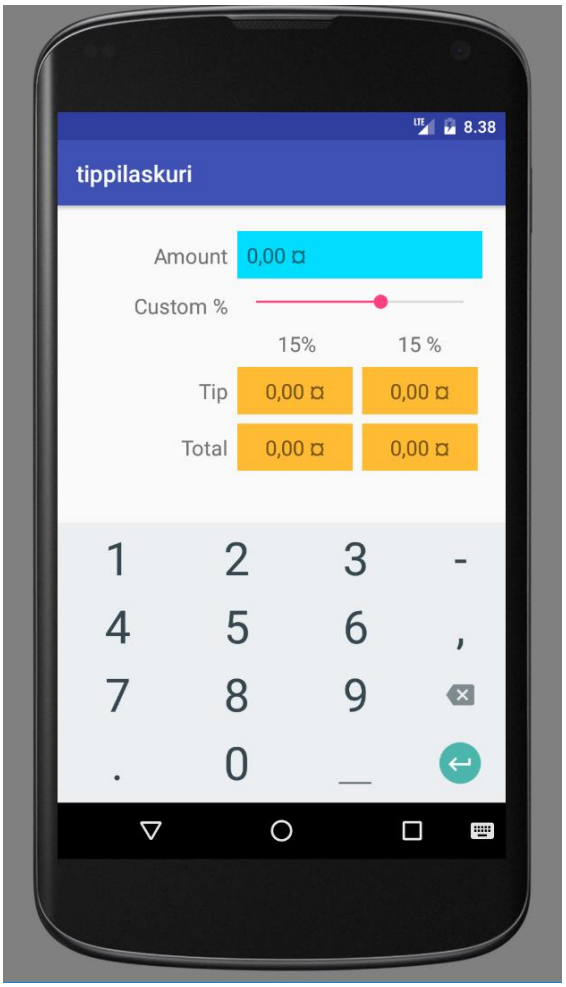
Welcome App oli perusohjelma, joka kävi läpi, miten lisätään kaksi kuvaa näytölle ja esitetään ”tervetuloa” teksti. Ohjelmassa opeteltiin lisäämään graafisessa näkymä editorissa valmiita komponentteja ”vetämällä ja pudottamalla” ne komponentti paletilta. Lisäksi muokattiin kyseisten komponenttien arvoja ominaisuus ikkunasta. Ohjelmassa myös lisättiin tekstiä ja numeerisia arvoja resurssi kansioon ja kuinka niitä voidaan käyttää lokalisaation. Osuudessa näytettiin myös, miten luodaan sisältökuvauksia, joita voidaan käyttää Androidin TalkBack ominaisuuden kanssa lukemaan ne ääneen, jotta ihmiset joilla on näkörajoitteita, pystyisivät käyttämään sovellusta. Lopuksi opeteltiin käyttämään AVD:ta ohjelman testaamiseen. Kuva 5 on valmiin sovelluksen päänäkymä.

Kuva 5. WelcomeApp sovelluksen päänäkymä suomeksi lokalisoituna.

Tip Calculator

Tippilaskuri (kuva 6) oli ensimmäinen käyttäjän ja sovelluksen vuorovaikutusta käsittelevä ohjelma, jolla käyttäjä pystyy laskemaan tipin määrän syöttämällä laskun hinnan ja liikuttamalla hakupalkkia joka lisää tai vähentää tipin kokonaismäärää. Sovelluksen luomisen aikana käytiin läpi graafisen näkymän järjestely, hyväksikäyttäen taulukko näkymää (GridLayout). Sovellus kävi myös läpi MainActivity -luokan ohjelma koodin, jossa luotiin muutama objekti (TextWatcher, OnSeekBarChangeListener) hallitsemaan tekstin vaihtumista seuraamalla hakupalkin liikehdintää (SeekBar). MainActivityssä yli kirjoitettiin OnCreate() metodi jossa käytettiin myös Activity -luokan metodia findViewByID(), jolla pystytään viit-

taamaan näkymän eri komponentteihin käyttämällä komponentin yksilöllistä ID:tä. Osiossa asetettiin, että ohjelma toimii vain muotokuva näkymällä ja että näppäimistö on koko ajan esillä muokkaamalla AndroidManifest.xml tiedostoa.

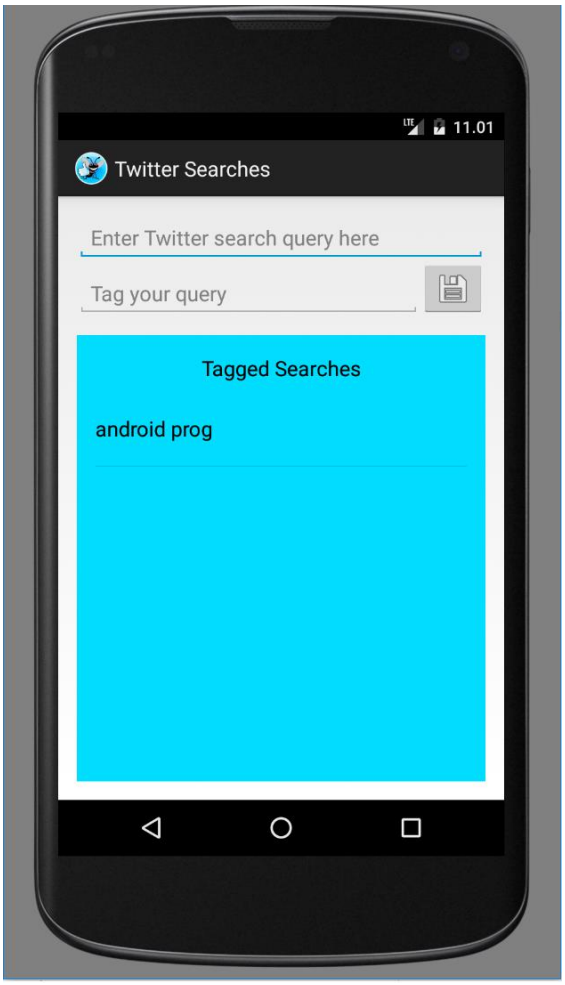


Kuva 6. Tip Calculator perusnäkymä.

Twitter Searches

Tämän sovelluksen aikana opittiin luomaa käyttöliittymä, jossa käytetään ListView komponenttia selaamaan asioita ja näyttämään niitä (kuva 7) tallennettuna listalle. Osiossa opittiin, miten Java koodissa käytetään resursseihin tallennettujen String -objektien sisältöä. Opittiin myös, kuinka SharedPreference. Editor -Objektia hyväksi käyttäen voidaan hallita, lisätä sekä poistaa tunnisteita. Ohjelmassa myös käytettiin Uri:a lataamalla se puhelimen internet selaimeen luomalla uusi Intent ja tätä käyttämällä aloittamaan uusi aktiviteetti startActivity() metodia käyttäen. Intentiä käytettiin myös, jotta käyttäjä voi valita

ohjelman, jolla jakaa käytetty haku. AlertDialog.Builder Objektia käyttäen esitettiin käyttäjälle viestejä. Lopuksi AndroidManifest.xml tiedostoa muokattiin niin, että näppäimistö ei ole näkyvissä, silloin kun sovellus käynnistetään.



Kuva 7. Twitter Searches -ohjelman päänäkymä

Flag Quiz App

Tässä harjoituksessa oli tarkoitus luoda sovellus, joka kysyy käyttäjältä, mikä on ruudussa näkyvän lipun nimi. Pääominaisuuksina olivat fragmentit, joilla luotaisiin osio graafiseen käyttäjän näkymään. Mutta koska tässä vaiheessa aikaa oli kulunut jo niin paljon, päätin jättää sovelluksen harjoittelun pintapuoliseksi, jotta ehtisin saada aikaiseksi oman prototyyppini.

5.2.2 Harjoitusten hyödyt

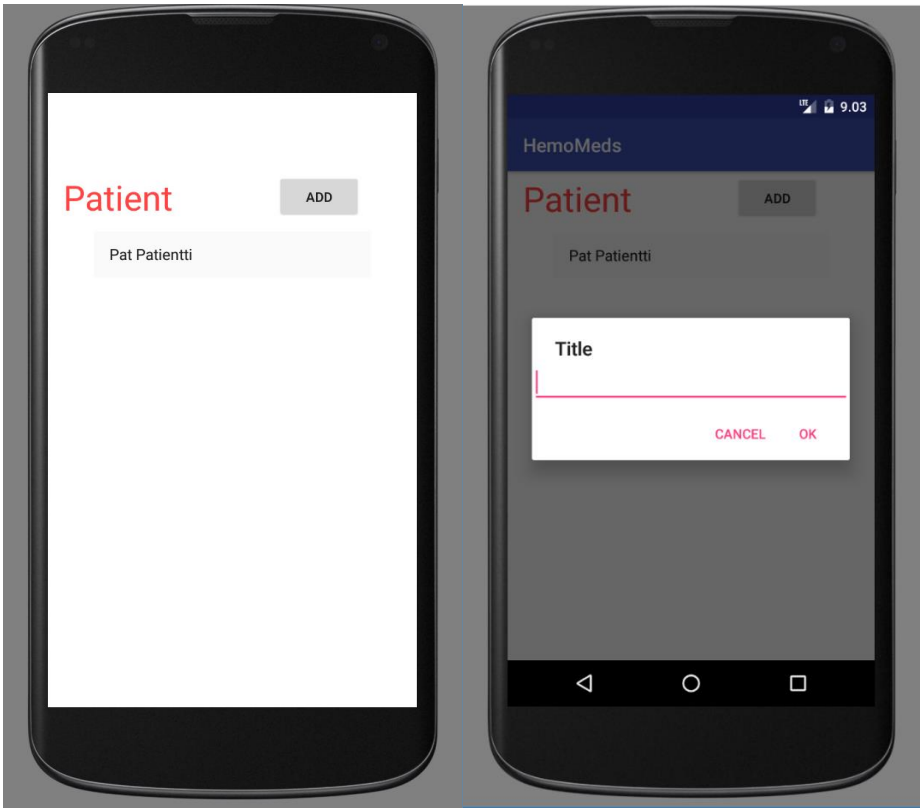
Harjoitukset auttoivat tutustumaan ohjatusti Android Studion oleellisimpiin käyttöliittymän komponentteihin. Se että kaikki harjoitukset olivat tehty Eclipselle, jonka käyttöliittymä on jokseenkin erilainen, oli sekä hyvä että huono asia. Hyvä asia siinä mielessä, että harjoitusten teko ”oikeasti” pakotti tutustumaan käyttämiinsä ominaisuuksiin, koska niitä joutui etsimään verkosta, huono asia siksi, että siihen kului ylimääräistä aikaa.

6 Sovelluksen kehitys

Kun omasta mielestä riittävä tuntuma Android-kehitykseen oli saatu, prototyypin suunnittelu alkoi ohjelman mahdollisista toiminnoista ja rakenteesta. Tässä kappaleessa kuvataan sovelluksen kehitys pääpiirteisesti.

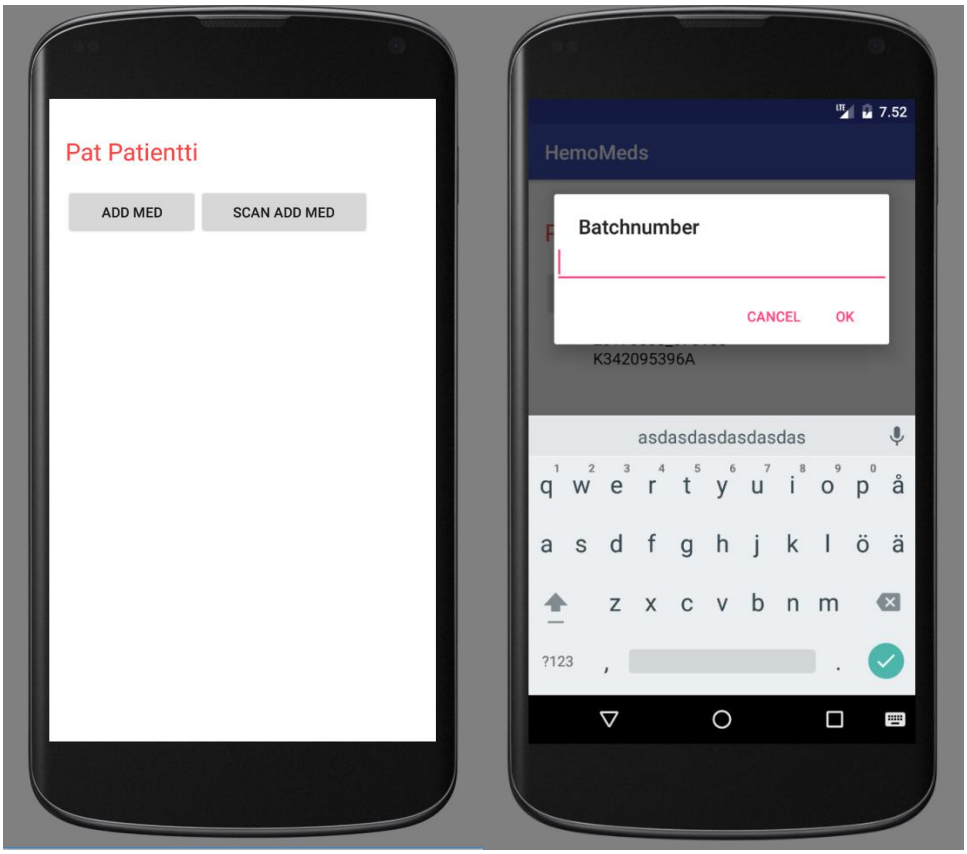
6.1 Ohjelman toimintaperiaate

Ohjelma alkaa päänäkymästä (kuva 8), jossa voidaan lisätä potilas painamalla ”ADD” painiketta, jolloin esiin tulee syöte ruutu, johon voidaan lisätä potilaan nimi (kuva 9) puhelimen virtuaalisella näppäimistöllä. Hyväksymisen jälkeen luodaan potilasobjekti, jonka nimi lisätään potilaslistaan. Painamalla potilaan nimeä, listalta siirrytään kyseisen potilaan potilasnäkymään. Mikäli käyttäjä painaa puhelimen paluunäppäintä, sovellus sulkeutuu.



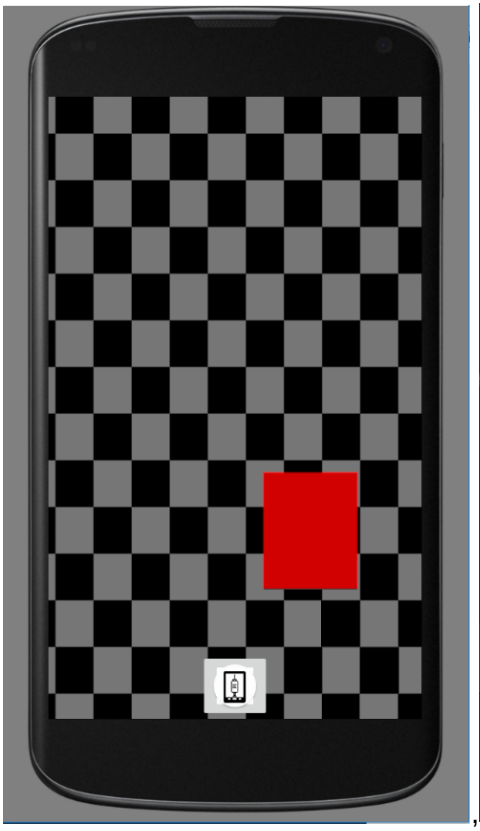
Kuva 8. Päänäkymän asettelu Kuva 9. Potilaan lisäys

Potilas ruudulta käyttäjä voi lukea annetut lääkkeet listalta (kuva 10), jota voidaan liu'uttaa ylös ja alas. ”ADD MED” painiketta painamalla, voidaan lisätä lääkkeen eränumero (kuva 11) samaan tapaan kuin potilaan nimen lisäämisessä. Kun käyttäjä on hyväksynyt syötteen, se lisätään lääkelistaan ja potilasobjektin tietoihin. Mikäli potilas painaa ”SCAN ADD MED” näppäintä, käynnistyy kamera applikaatio (kuva 12), jolla käyttäjä voi ottaa kuvan lääkepullon eränumerosta. Tämän jälkeen kuva palautetaan sovellukseen, joka pyysi kameratoimintoa. Seuraavaksi sovellus lukee kuvalta kirjaimet ja avaa ”ADD MED” painikkeen näkymän, jossa sovellus ehdottaa löytämänsä eränumeron ja pyytää käyttäjää hyväksymään sen (kuva 13). Mikäli käyttäjä painaa puhelimen paluu painiketta, siirrytään potilasnäkymään (kuva 10).

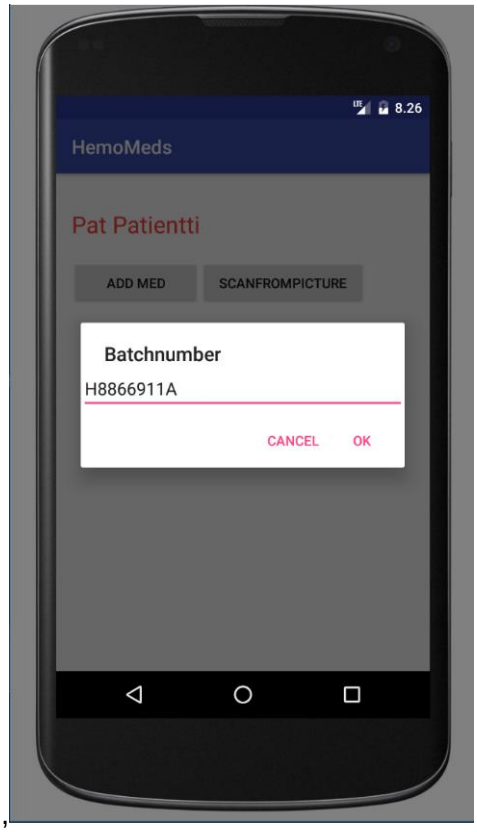


Kuva 9. Lääkkeenlisäysnäkymä

Kuva 11. Eränumeronlisäysnäkymä



Kuva 12. Kameranäkymä



Kuva 13. Kuvan ottamisen jälkeinen tilanne

6.2 Toteutus

Ohjelmointi oli alkuun varsin hankalaa, johtuen kaikesta uudesta, mitä joutui opettelemaan. Tämän vuoksi käytännössä kaiken jonka kirjoitti, joutui varmistamaan oppaista toiminnallisuuden varmistamiseksi. Ongelmia tuotti lähinnä Javan syntaksit jotka ovat hieman erilaiset, sekä luokkien käyttäminen, vaikka se ei juurikaan eroa C++:n luokkien käytöstä. Lukuun ottamatta sitä, että luokkien moniperintää ei tueta, mutta tämä on korvattu rajapinnoilla joista luokat voivat periä abstrakteja metodeita.

6.2.1 Patient -luokka

Patient on yksinkertainen potilas-luokka (kuva 14), joka pitää sisällään potilaan henkilötiedot ja otetut lääkkeet. *Patient* -luokka pitää sisällään listarakenteen johon voidaan lisätä otettuja lääkkeitä funktion *AddMedication()* ja alaluokan *Medication* avulla. Potilaan nimeä varten on myös erillinen funktio.

```
public class Patient
{
    public ArrayList<Medication> takenMedication;
    public String FullName;


    public Patient()
    {
        takenMedication = new ArrayList<Medication>();
    }

   public void AddName( String firstName, String lastName)
   {

        FullName = firstName + " " + lastName;
   }

    public void AddMedication(String batchName)
    {
        Medication med = new Medication();
        med.Batch = batchName;
        med.date = Calendar.getInstance().getTime();

        takenMedication.add(med);
        Log.d("HemoMeds","addMed");

    }

    class Medication
    {
        public String Batch;
        public Date date;
    }

}
```

Kuva 14. Patient -luokan toteutus

6.2.2 Tiedon tallentaminen

Sovelluksessa potilastietojen tallennus tapahtuu MainActivity -luokan kautta, kun käyttäjä painaa potilasnäkymässä ”ADD MED” tai ”SCANFROMPICTURE” painiketta. Mikäli käyttäjä painaa ”ADD MED” näppäintä, *AddMed()* funktio (kuva 15) tuo näytölle *AlertDialogin* (kuva 11). Tämä pyytää käyttäjää syöttämään eränumeron. Mikäli annettu eränumero on pidempi kuin kolme kirjainta, lisätään se valittuna olevan potilaan tietoihin. Tämän jälkeen Aa2 (*ArrayAdapter* objekti jota tarvitaan ListView:in käytöön) tyhjennetään ja suoritetaan *OnSelected*() funktio.

```
    public void AddMed(View view)
    {
        AlertDialog.Builder builder = new AlertDialog.Builder(this);
        builder.setTitle("Batchnumber");

// Set up the input
        final EditText input = new EditText(this);

        input.setText("H8866911A ( Default )");
        input.setInputType(InputType.TYPE_CLASS_TEXT | Input-
Type.TYPE_TEXT_VARIATION_NORMAL);
        builder.setView(input);
// Set up the buttons
        builder.setPositiveButton("OK",
new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which) {

                //todo: OCR_Text+input.getText().toString()
                String name  = input.getText().toString();
                if(name.length() > 3 )
                {
                    selectedPatient.AddMedication(name);
                    Aa2.clear();
                    OnSelected();
                }

            }
        });
        builder.setNegativeButton("Cancel",
new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which) {
                dialog.cancel();
            }
        });
        builder.show();
    }
```

Kuva 15. *AddMed* -function toteutus

OnSelected() funktiossa (kuva 16) asetetaan näkyviin potilasnäkymä ja järjestellään lääkitykset anto päivämäärän mukaan, jonka jälkeen ne lisätään ListView:iin, sekä asetetaan potilaan nimi TextView:iin.

```
public void OnSelected()
{
    ArrayList<String> tempList;
    setContentView(R.layout.patient_main);
    tempList =
        new ArrayList<String>();
    SimpleDateFormat sdf =
        new SimpleDateFormat("E  d MMMM yyyy HH:mm",
                                    Locale.getDefault());

    //sorting
    Collections.sort(selectedPatient.takenMedication,
        new Comparator<Patient.Medication>() {

        @Override
        public int compare(Patient.Medication o1,
                Patient.Medication o2)
        {
            return o2.date.compareTo(o1.date);
        }
    });

    for (int i = 0; i < selectedPatient.takenMedication.size(); i++)
    {
        Patient.Medication temp =
                selectedPatient.takenMedication.get(i);
        String strDate =
                sdf.format(temp.date);
        strDate +=
                "\n"+temp.Batch;
        tempList.add(strDate);
    }

    Aa2.clear();
    Aa2.addAll(tempList);
    patient_mainListView =
                (ListView)findViewById(R.id.listView);
    patient_mainListView.setAdapter(Aa2);
    TextView patientName =
                (TextView) findViewById(R.id.patientName);
    patientName.setText(selectedPatient.FullName);

}
```

Kuva 16. *OnSelected* -funktion toteutus

Tietojen varsinainen tallennus tapahtuu *onPause()* metodissa, *saveData()* funktiolla(kuva 17) sovelluksen sisäiseen muistiin, jonka hyvä puoli on se, että siihen eivät pääse muut

sovellukset käsiksi, mutta huonona puolena se, että ne poistuvat sovelluksen mukana. Tallennuksessa tiedostoon lisätään tietojen ympärille merkkejä, jotka helpottavat tietojen ”parsimisessa” myöhemmin.

```
@Override
 protected void onPause() {
    saveData();

    super.onPause();
}

   protected void saveData(){
        String text = new String();

        for (int i = 0; i < PatientList.size(); i++)
        {
            Patient pat = PatientList.get(i);
            text += "{\"" + pat.FullName + "\"(";
            for(int n = 0; n < pat.takenMedication.size(); n++)
            {
                Patient.Medication med =
                       pat.takenMedication.get(n);
                text +=
                       "[ \"" + med.Batch + "\"," +
                              Long.toString(med.date.getTime()) + " ]";
            }

            text += ")}\n";
        }

        FileOutputStream outputStream;

        try {

            outputStream = openFileOutput("PatientData",
                      this.MODE_PRIVATE);
            outputStream.write(text.getBytes());
            outputStream.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

Kuva 17. *onPause* ja *saveData* funktioiden toteutus

Tietojen lukeminen tapahtuu *onResume()* metodissa kutsumalla readData() funktiota (kuva 18). Mikäli potilas listarakenne on tyhjä, Avataan PatientData -tiedosto josta luetaan potilaiden nimet ja otetut lääkkeet. Nyt kun tiedetään, että tietojen molemmin puolin löytyy tietyt merkit, voidaan näitä hyväksikäyttäen tiedot hakea ja asettaa Patient objektiin. Funktion lopuksi kutsutaan *UpdatePatientList(),* joka huolehtii listanäkymien päivittymisestä.

```
protected void readData() {
    FileInputStream inputStream;
    String fileStream;

    if (PatientList.isEmpty()) {
        try {
            inputStream = openFileInput("PatientData");
            // inputStream.write(text.getBytes());
            int size = inputStream.available();
            byte[] str = new byte[size];
            inputStream.read(str);
            fileStream = new String(str);
            inputStream.close();
            int i1 = 0;
            int i2 = 0;
            //Parsing
            for (int k = 0; k < size; ) {
                int addToK = fileStream.indexOf("{", i2);
                if (addToK == -1)
                    break;
                addToK = fileStream.indexOf("}", addToK) - addToK + 1;
                i1 = fileStream.indexOf("\"", i2) + 1;
                int temp = i1 - 1;
                i2 = fileStream.indexOf("\"", i1);
                String name = fileStream.substring(i1, i2);

                i1 = fileStream.indexOf(("("), i2) + 1;
                i2 = fileStream.indexOf(")", i1);
                //temp = i2 - temp + 1;
                //Log.d("debug", temp);
                String medicalData = fileStream.substring(i1, i2);

                Patient pat = new Patient();
                pat.FullName = name;

                for (int i = 0; i < medicalData.length(); ) {
                    i1 = medicalData.indexOf(("["), i2 + 1);
                    i1 = medicalData.indexOf("\"", i1) + 1;
                    i2 = medicalData.indexOf(",", i1);

                    String batch = medicalData.substring(i1, i2 - 1);
                    i1 = i2;
                    i2 = medicalData.indexOf("]", i2);
                    String dateString = medicalData.substring(i1 + 1,
                                                i2 - 1);

                    long dateLong = Long.parseLong(dateString);
                    Date date = new Date(dateLong);
                    i = i2 + 1;

                    pat.AddMedication(batch);
                    pat.takenMedication.get(pat.takenMedication.size() - 1);
                    Patient.Medication medication =
                   pat.takenMedication.get(pat.takenMedication.size() - 1);

                    medication.date = date;

                }
                Log.d("tag", "medication = " + medicalData);
                PatientList.add(pat);
                //i2 = fileStream.indexOf(")",i2) + 1;
                k += addToK;
                i2 = k;
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        UpdatePatientList();
    }
}
```

Kuva 18. *readData* - funktion toteutus

Käyttäjä voi myös halutessaan skannata lääkepullon eränumeron painamalla ”SCAN-FROMPICTURE” näppäintä, jolloin kutsutaan OnScanMedClick() funktiota (kuva 19). OnScanMedClick() luo intentin joka kertoo sovellukselle, että aiotaan käyttää kamerasovellusta ja sovellukselta pyydetään paluutietoina kuvan Uri (tiedostopolku). Intentin luomisen jälkeen käynnistetään sovellus.

```
public void OnScanMedClick(View view){


         Intent cameraIntent =
        new Intent(android.provider.MediaStore.ACTION_IMAGE_CAPTURE);
         cameraIntent.putExtra(
                      MediaStore.EXTRA_OUTPUT, mImageCaptureUri);

         startActivityForResult(cameraIntent, CAMERA_REQUEST);

}
```

Kuva 19. *OnScanMedClick* -funktion toteutus

Kun puhelimen kameralla otetaan kuva, palataan omaan sovellukseen ja *onActivityResult*() funktioon (kuva 20) jossa luodaan Bitmap kuvasta sekä otetaan tiedostonpolku talteen. Nämä objektit syötetään sitten parametreina funktiolle getTextFromPhoto() (kuva 21).

*getTextFromPhoto()*luo tarvittavat kansiot mikäli ne puuttuvat TessBaseAPI:lle. TessBaseAPI yrittää alustuksen jälkeen lukea kuvasta mahdolliset kirjaimet, jonka jälkeen saatu tulos käytetään funktiossa *getbatchfromscan* (kuva 22), joka parsii mahdollisen eränumeron TessBaseAPI:n teksti syötteestä. Ohjelma koodi on sovellettu lähteiden (23, 24) ohjeista.

```
@Override
protected void onActivityResult(int requestCode, int resultCode,
                Intent data) {
    if (requestCode == CAMERA_REQUEST && resultCode == RESULT_OK) {
        Bitmap photo = (Bitmap) data.getExtras().get("data");

        Uri tempUri = getImageUri(getApplicationContext(), photo);
        File finalFile = new File(getRealPathFromURI(tempUri));
        getTextFromPhoto(toGrayscale(finalFile,photo));
    }
}
```

Kuva 20. onActivityResult -funktion toteutus.

```
   protected void getTextFromPhoto(Bitmap photo) {


        String DATA_PATH;
        DATA_PATH =
        Environment.getExternalStorageDirectory() + "/ocrctz/";

        File dir = new File(DATA_PATH + "/tessdata/");
        File tessfile = new File(DATA_PATH + "/tessdata/" +
"eng.traineddata");
        if (!tessfile.exists()) {
            Log.d("mylog", "in file doesn't exist");
            dir.mkdirs();
        }

        try {
            AssetManager assetManager =
               getApplicationContext().getAssets();

            InputStream in =
                assetManager.open("eng2.traineddata");

            OutputStream out =
        new FileOutputStream                    (DATA_PATH +
        "/tessdata/" + "eng.traineddata");

            byte[] buffer = new byte[1024];
            int read = in.read(buffer);

            while (read != -1) {
                out.write(buffer, 0, read);
                read = in.read(buffer);
            }
        } catch (Exception e) {
            Log.d("mylog",
        "couldn't copy with the following error : " + e.toString());
        }

        try {
            baseAPI.init(DATA_PATH, "eng");
            //treat text as one line
            baseAPI.setPageSegMode(5);
            baseAPI.setImage(photo);
            recognizedText = baseAPI.getUTF8Text();
            baseAPI.end();
            Log.d("BaseAPIResult", recognizedText);
            DialogBuilder("ScannedBatch:",getbatchfromscan());

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

        }
    }
```

Kuva 21. getTextFromPhoto -funktion toteutus.

```
public String getbatchfromscan() {

    int i1 = 0;
    int i2 = 0;

    String Text;
    int batchlength = new String("Batch:").length();


        i1 = recognizedText.indexOf(("B"));
        i2 = recognizedText.indexOf(":");

        if((i2-i1) == (batchlength - 1)) {
            Text = recognizedText.substring(i2 + 1).toString();
            return Text.trim();
        }

    else
    return Text = "No batch number found";
}
```

Kuva 22. getbatchfromscan -funktion toteutus.

7 Käyttöliittymän toiminnot ja testaus

Tässä osiossa käydään läpi käyttöliittymä ja sen toiminnot, kuvilla havainnollistettuna.

7.1 Päänäkymä

Päänäkymä on ensimmäinen asia, jonka käyttäjä näkee sovelluksen käynnistäessään. Päänäkymästä voidaan luoda uusi potilas painamalla ADD -painiketta (kuva 8.) jolloin sovellus pyytää käyttäjää kirjoittamaan potilaan nimen (kuva 9.). Valitsemalla olemassa olevan potilaan siirrytään potilasnäkymään (kuva 10).

7.2 Potilasnäkymä

Potilasnäkymässä (kuva 10) käyttäjä voi lisätä tietoihin otetun lääkkeen painamalla ”ADD MED” tai ”SCAN ADD MED” painikkeita, joko perus tekstin syötöllä tai puhelimenkameralla. Mikäli potilaalla on otettuja lääkkeitä ne näkyvät listassa painikkeiden alapuolella.

7.3 Kameranäkymä

Kameranäkymä (kuva 12) tulee esiin kun, kameran API käynnistyy, käyttäjän painaessa ”SCAN ADD MED” -painiketta. Kamera sovellusta suoritetaan, kunnes käyttäjä on ottanut valokuvan tai palannut takaisin potilasnäkymään, paluunäppäintä painamalla.

7.4 Testaus

Testaus on tärkeää suorittaa koko ajan sovellusta kehittäessä, jotta ei pääse ilmenemään virheitä, joiden korjaaminen tuhoaisi myöhemmin kirjoitetun ohjelmakoodin.

Tämän sovelluksen prototyypin kehityksessä käytettiin API taso 24, jonka valintaa ei ollut mitään varsinaista syytä. Kehityksen aikana huomattiin, että se antoi käyttöön luokkia,

joita ei alemmilla API tasoilla olisi ollut. Esimerkiksi SimpleDateFormat -luokka, jolla haettiin kellonaika ja päivämäärä.

Virtuaalista Android -puhelinta käytettiin, kun oikeaa puhelinta ei ollut käytettävissä. Puhelimeen oli asetettuna API taso 24. Emulaattori toimii hyvin, pienempien ominaisuuksien ja toimintojen testaamisessa, mutta jos virheiden etsintään tarvitaan debug -työkaluja, niin oikea puhelin antaa varmempaa tietoa siitä, mikä menee vikaan.

Emulaattoria voi suositella ajoittain käytettäväksi, mutta emulaattorin käyttö vianetsinnässä ei ole välttämättä parasvaihtoehto, koska emulaattori saattaa vain kaatua ilman vikakoodia tai vielä pahempaa, sovellus toimii hienosti, mutta oikeaan puhelimeen siirryttäessä ei, jolloin vian etsinnästä tulee huomattavasti hankalampaa.

8 Pohdinta

Prototyypin kehitys onnistui lähes suunnitellusti. Kehitettyyn sovellukseen voidaan luoda käyttäjä, lisätä käyttäjälle lääkkeitä ja kameran avulla voidaan lisätä tekstiä. Koska edellä mainitut asiat olivat sovelluksen tavoite, voidaan sanoa, että prototyyppi toteutui, vaikkakaan tekstintunnistus ei kirjoitushetkellä toimi riittävällä tarkkuudella.

Projektin työkaluja valittaessa, olisi voitu valita tuttuja ja turvallisia välineitä, mutta onneksi näin ei tehty, sillä uutta oppimalla laajennettiin omaa tietotaitoa huomattavasti enemmän, kuin menemällä sieltä mistä aita on matalin.

Android Studion omaksuminen uutena kehitysympäristönä ei loppujen lopuksi ollut ongelma, mutta niin kuin kaikessa uuden oppimisessa alku oli vaikea, varsinkin kun mukana oli vieras ohjelmointikieli. Se, mikä auttoi pääsemään sisään Android Studion käyttöliittymään, oli ohjelmointioppaat, joita alussa tuli käytyä läpi. Niissä selvitettiin riittävällä tasolla, kuinka asiat tulee tehdä.

Kaikkia Java -kielen nyansseja ei tässä ajassa tietenkään voinut oppia, mutta sain ohjelmoinnista hyvän otteen ja koodin tuottamin ei alkuvaikeuksien jälkeen ole enää ongelma. Mallista katsomista joutuu vielä tekemään huomattavasti enemmän kuin C -kielten kanssa.

Aikataulutus ei ollut niin onnistunut kuin olisi voinut toivoa. Heittoa aikataulussa oli muutama viikko, johon olisi ollut tarvetta työkalujen parempaan opiskelemiseen. Uskon kuitenkin että, vaikka opiskeluun olisi käyttänyt vielä muutaman viikon enemmän, olisi ohjelma valmistunut samassa ajassa. Sillä kaikki mitä en ollut opetellut ennen prototyypin tekemisen aloittamista, jouduin opettelemaan kehityksen aikana.

Mitä tulee sovelluksen suunnittelemiseen, niin siihen olisi voinut käyttää enemmän aikaa, mutta koska aika oli kortilla, sovelluksen kehityksen tavoitteena oli saada valmiiksi toimiva versio. Edellä mainitusta syystä prototyyppistä ei voi ottaa varsinaisen sovelluksen kehitykseen valmiita komponentteja. Prototyypissä toteutettua ideaa voidaan kuitenkin käyttää ”proof-of-concept” tyyppisenä ratkaisuna, jolla voidaan osoittaa ratkaisu esitettyyn tarpeeseen.

Lähteet

(1) Astma - Hengitys.fi. 11/2016; Saatavilla: http://hengitys.fi/astma/. Viitattu 20/5/2017.

(2) Suositukset - Käypä hoito. 2012; Saatavilla: http://www.kaypahoito.fi/web/kh/suositukset/suositus?id=hoi06030. Viitattu 25/5/2017.

(3) Astmatesti - Hengitys.fi. 2016; Saatavilla: http://hengitys.fi/astma/astmatesti/tee-astmatesti/. Viitattu 25/5/2017

(4) URTIKARIANYT – HETKESSÄ MUKANA - Urtikaria iholla. 2017; Saatavilla:

http://www.urtikariaiholla.fi/tukea/UrtikariaNYT-sovellus/. Viitattu 1/6/2017.

(5) Mitä hemofilia on? - HemofiliaBayer. 2017; Saatavilla: http://www.hemofiliabayer.fi/fi/hemofiliasta/mitahemofilia-on/.

Viitattu 2/6/2017.

(6) Teixeira L, Saavedra V, Ferreira C, Santos B. Improvement of Surveillance of Hemophilia Treatment through ICTs.

2012.

(7) Tapparello C, Heinzelman W, Conn K, Mullen CA. ManageMyCondition: A Standard Framework for the

Development of Cloud-based Medical Condition Management Applications. 2016.

(8) Smith K. Mobile Health for Hemophilia - Hemaware

. 2014; Saatavilla: https://hemaware.org/story/mobile-health-hemophilia. Viitattu 20/5/2017

(9) Hanson M. Android through the ages: Cupcake, Nougat and everything in between - Techradar. 2017;

Saatavilla: http://www.techradar.com/news/phone-and-communications/mobile-phones/android-through-the-ages-

1289518. Viitattu 5/1/2017.

(10) The Linux Kernel - Developer.android. 2017; Saatavilla: https://developer.android.com/guide/platform/index.html. Viitattu 25/2/2017.

(11) Mobile/Tablet Operating System Market Share - Netmarketshare. 2017; Saatavilla:

https://www.netmarketshare.com/operating-system-market-share.aspx?qprid=8&qpcustomd=1. Viitattu 2/5/2017.

(12) Application Fundamentals - Developer.android. 2017; Saatavilla:

https://developer.android.com/guide/components/fundamentals.html. Viitattu 20/3/2017.

(13) Developer Workflow Basics - Developer.android. 2017; Saatavilla:

https://developer.android.com/studio/workflow.html. Viitattu 2/6/2017.

(14) The Activity Lifecycle – Developer.android. 2017; Saatavilla:

https://developer.android.com/guide/components/activities/activity-lifecycle.html. Viitattu 2/6/2017.

(15) Processes and Application Life Cycle - Developer.android. 2017; Saatavilla:

https://developer.android.com/guide/topics/processes/process-lifecycle.html. Viitattu 20/5/2017.

(16) Android Studio Release Notes - Developer.android. 2017; Saatavilla:

https://developer.android.com/studio/releases/index.html. Viitattu 2/5/2017.

(17) Eclipse Forms Independent Organization - Eclipse. 2017; Saatavilla: http://www.eclipse.org/org/pressrelease/feb2004foundationpr.php.

Viitattu 5/5/2017.

(18) About Qt - wiki.qt. 2017; Saatavilla: https://wiki.qt.io/About_Qt. Viitattu 2/6/2017.

(19) Platform Architecture - Developer.android. 2017; Saatavilla:

https://developer.android.com/guide/platform/index.html. Viitattu 5/2/2017.

(20) App components - Developer.android. 2017; Saatavilla:

https://developer.android.com/guide/components/fundamentals.html. Viitattu 20/5/2017.

(21) Configure Your Build - Developer.android. 2017; Saatavilla:

https://developer.android.com/studio/build/index.html. Viitattu 20/5/2017.

(22) Deitel P, Deitel H, Deitel A. Android for Programmers an App Driven Approach Second Edition, Volume 1. Second Edition Ed. United States of America: Pearson Education, Inc.; 2014

(23) Gupta G. Making a Simple OCR Android App using Tesseract - Gaut.am. 2012; Saatavilla:

http://gaut.am/making-an-ocr-android-app-using-tesseract/. Viitattu 10/3/2017.

(24) Tesseract OCR - Github. 2017; Saatavilla: https://github.com/tesseractocr/tesseract/blob/master/README.md.

Viitattu 20/3/2017.