KARELIA-AMMATTIKORKEAKOULU
Tietojenkäsittelyn koulutus

Eetu Sormunen

ASHED-TIETOKANTASOVELLUS ANDROIDILLE

Opinnäytetyö
Toukokuu 2016

Karelia AMMATTIKORKEAKOULU

OPINNÄYTETYÖ
Toukokuu 2016
Tietojenkäsittelyn koulutusohjelma

Karjalankatu 3
80200 JOENSUU
+358 50 310 9761

Tekijä
Eetu Sormunen

Nimeke
Ashed-tietokantasovellus Androidille

Toimeksiantaja
-

Tiivistelmä

Opinnäytetyön tavoitteena oli toteuttaa Android-käyttöjärjestelmälle tietokantasovellus, joka pitää kirjaa siitä, mitä sikareita on poltettu. Android-sovelluskehityksen suosio sekä sikaritietokantasovellusten vähäinen tarjonta Google Play Storessa antoivat aiheen tämän opinnäytetyön toteuttamiseen.

Opinnäytetyön teoriaosuudessa tutustutaan Androidin tarjoamiin eri tiedontallennusmenetelmiin. Sovelluksen toteutuksessa käytettiin Androidin virallista kehitysympäristöä Android Studiota. Sovelluksen tiedontallennus toteutettiin SQLite-relaatiotietokantaa hyväksikäyttäen. Opinnäytetyössä kuvataan tietokantasovelluksen rakenteen ja toiminnallisuuden suunnittelu sekä toteutus.

Opinnäytetyön tuloksena syntyi toimiva tietokantasovellus. Valmistuneen sovelluksen pohjalta sitä voidaan jatkokehittää julkiseen levitykseen kelpaavaksi sovellukseksi.

Kieli	Sivuja
suomi	28

Asiasanat

tietokanta, Android, sikari, SQLite

Karelia
UNIVERSITY OF APPLIED SCIENCES

THESIS
May 2016
Business Information Technology

Karjalankatu 3
80200 JOENSUU
FINLAND
+358 50 310 9761

Author
Eetu Sormunen

Title
Ashed Database Application for Android

Commissioned by
-

Abstract

The purpose of this thesis was to develop a database application on Android platform which keeps track of smoked cigars. The popularity of Android programming and absence of this type of applications on Google Play Store gave the subject for this thesis.

The theoretical part of the thesis examines the different possibilities of saving data on Android platform. In the thesis Android Studio was used which is the official integrated development environment for Android platform development. The data saving of the application was implemented with the relational database management system SQLite. In the thesis the structure of the application, functionality and development progress are examined.

The result of the thesis was a successfully created database application for Android. On the base of the created application, it is ready for further development to be published the for public.

Language	Pages
Finnish	28

Keywords

database, Android, cigar, SQLite

Sisältö

1 Johdanto

Tämän opinnäytetyön tarkoitus on luoda yksinkertainen toimiva sovellus poltettujen sikareiden kirjanpitoon sekä niiden kommentointiin. Kiinnostukseni Android-sovelluskehitykseen on kasvanut vähitellen kouluaikana ja erityisesti siinä vaiheessa, kun huomasin, ettei Androidin virallisessa sovelluskaupassa Google Play Storessa ollut sellaista sovellusta, jota haluaisin itse käyttää. Sovelluskaupasta löytyy muutama ilmainen sovellus sikareiden tallentamiseen, mutta niiden kohderyhmä on pääasiassa yhdysvaltalaiset käyttäjät. Sovelluksissa on hyödyttömiä ominaisuuksia käyttäjille, jotka eivät asu Yhdysvalloissa, kuten yhdysvaltalaisten sikarikauppojen etsiminen karttasovelluksella. Opinnäytetyölle ei ollut toimeksiantajaa vaan se luotiin omaan käyttöön sovelluksen jatkokehitystä silmällä pitäen.

Opinnäytetyön toisessa luvussa tutustutaan Androidin tiedontallennukseen, tiedostojärjestelmään ja muistialueisiin sekä niiden käyttökohteita ja rajoituksia. Luvussa esitellään myös Androidin tietokantaratkaisujen vaihtoehtoja ja niiden käyttökohteita.

Opinnäytetyön kolmannessa luvussa käydään läpi opinnäytetyöprosessin suunnittelua sekä sovelluksen toiminta.

Opinnäytetyön neljännessä luvussa tarkastellaan aikaansaatuja tuloksia ja kohdattuja ongelmia. Luvussa käydään myös läpi, millaisia ajatuksia opinnäytetyöprosessi minussa herätti sekä mitä opinnäytetyö on opettanut. Lopuksi käydään läpi sovelluksen tulevaisuuden näkymiä jatkokehityksen osalta.

2 Tiedontallennus

Useimmat Android-sovellukset tarvitsevat tiedostojen tallentamista edes jossain määrin. Sovelluksen tietojen tulee pysyä laitteen muistissa myös silloin kun käyttäjä jättää sovelluksen tausta-ajoon. Useimmat hiemankin kehittyneemmät sovellukset tallentavat käyttäjäasetuksia ja mahdollisesti suuren määrän tiedostoja tai jopa tietokantoja. (Android Developers 2016a.)

Seuraavissa alaluvuissa tarkastellaan Androidin tiedostojärjestelmää yleisesti sekä eri tiedontallennusvaihtoehtoja ja niiden käyttömahdollisuuksia erilaisissa tilanteissa.

2.1 Tiedostojärjestelmä

Androidin tiedostojärjestelmä on samankaltainen kuin muillakin levypohjaisilla tiedostojärjestelmillä. Kaikilla Android-laitteilla on kaksi muistialuetta, sisäinen ja ulkoinen muistialue. Nimitykset juontavat juurensa Androidin alkuajoilta, jolloin suurimmassa osassa laitteita oli sisäinen muisti sekä ulkoinen muisti kuten muistikortti. Laitteissa, joissa ei ole irrotettavaa muistikorttia, sisäinen muisti jaetaan kahdeksi osioksi, ”sisäiseksi” ja ”ulkoiseksi” muistiksi. Näin saadaan kaksi muistialuetta, jolloin APIn (Application programming interface) eli ohjelmointirajapinnan käyttäytyminen ei muutu, oli laitteessa fyysisesti irrotettavaa ulkoista muistia tai ei. (Android Developers 2016b.)

2.2 Avain-arvopari

Avain-arvoparia (eng. shared preferences), käytetään primitiividatan eli yksinkertaisten tietojen tallentamiseen, kuten lyhyet merkkijonot, numeroita ja boolean arvoja. Nämä arvot säilyvät muistissa, vaikka sovellus suljettaisiin kokonaan. (Android Developers 2016c.)

Käytännössä avain-arvoparia voidaan käyttää esimerkiksi käyttäjätietojen kuten soittoäänen tallentamiseksi tietylle soittajalle tai herätysajan asettaminen. Esimerkin kaltaisten yksinkertaisten tietojen tallentaminen avain-arvoparina on teknisesti huomattavasti yksinkertaisempaa kuin tallentaa tietoja esimerkiksi SQLite-tietokantaan.

2.3 Sisäinen ja ulkoinen muisti

Sisäinen muisti (eng. Internal memory) on muistialue, missä sovelluksen tallentama tieto on aina sovelluksen käytettävissä. Oletuksena sisäisen muistin tiedot ovat ainoastaan tiedon luoneen sovelluksen käytettävissä eikä muilla sovelluksilla tai sovelluksen käyttäjällä ole niihin pääsyä. Sisäisessä muistissa olevat tiedot poistetaan, mikäli tietoja käyttänyt sovellus poistetaan. (Android Developers 2016d.)

Ulkoinen muisti (eng. External memory) eroaa sisäisestä muistista siten, että se ei ole välttämättä aina käytössä. Ulkoinen muisti voi olla pois käytöstä, jos käytössä on ulkoinen muistikortti, joka on otettu laitteesta pois tai mikäli käyttäjä ottaa ulkoisen muistin käyttöön USB-kaapelin avulla massamuistiksi. Muut sovellukset sekä käyttäjä voivat lukea ulkoista muistia, joten se sopii parhaiten sellaisten tiedostojen tallentamiseen, joihin käyttäjän tai muiden sovellusten halutaan pääsevän käsiksi tai joiden jakaminen toisten sovellusten kanssa on tarpeen. Ulkoiseen muistiin tallennetut tiedot poistetaan vain sovelluksen poistamisen yhteydessä, mikäli tiedot on tallennettu tiettyyn sijaintiin. (Android Developers 2016e.)

2.4 SQLite-tietokanta ja web-palvelimet

Androidissa on täysi tuki palvelimettomalle SQLite-relaatiotietokannalle. Relaatiotietokannassa taulujen välille luodaan yhteyksiä tunnisteita käyttäen. Tunnisteina toimii toisen taulun avain, jota kutsutaan yleisesti ID:ksi. ID on tunnisteen yksilöivä tieto, kuten esimerkiksi juokseva numerointi. Henkilötietoja tallentaessa ID voi olla henkilötunnus tai opiskelijalla opiskelijanumero. Taulujen välisiä suhteita voidaan kuvata äiti-lapsi-termillä. Termiä voidaan avata niin, että äidillä voi olla useita lapsia, mutta lapsilla ei useita äitejä. (Android Developers 2016f; SQLite.org 2016.)

Relaatiotietokantaan, kuten SQLiteen tallentaminen on paras vaihtoehto, kun tallennettava tieto on toistuvaa sekä yhdenmukaista. SQLite tallentaa tiedot laitteen muistiin ja ne ovat oletuksena ainoastaan tiedot tallentaneen sovelluksen saatavilla. Kun tiedot tallennetaan paikallisesti sovellusta käyttävään laitteeseen erillisiä tietokantapalvelimia ja tietokannanhallintaohjelmia ei tarvita. SQLite soveltuu erinomaisesti Android-pohjaisten mobiilisovellusten tietokantasovelluksesi, kun sovelluksen toiminta ei vaadi ulkoista tietokantapalvelinratkaisua. Käytännön esimerkki SQLite-tietokantaa hyödyntävistä sovelluksista on esimerkiksi kalenteri, muistilista ja yhteistietoluettelo. (Android Developers 2016g; SQLite.org 2016)

Asiakas-palvelin-mallinen web-palveluiden tietokantapalvelinratkaisu, kuten MySQL, sopii parhaiten käytettäväksi silloin, kun tarvitaan tietokantaratkaisu massiivisille tiedostomäärille ja monipuolisempia työkaluja tietojen käsittelyyn ja kun niihin on päästävä käsiksi useammalla laitteella. Tällöin tiedot tallennetaan tietokantapalvelimille eikä laitteen paikalliseen muistiin, kuten SQLiten tapauksessa. MySQL:ää käyttävät esimerkiksi yrityksillä Facebook, Google ja Adobe. (Wikipedia 2016a; MySQL 2016.)

3 Työn suunnittelu ja toteutus

Työnsuunnittelu projektin alkuvaiheessa karkeasti auttoi luomaan projektille rungon, kuinka sitä lähdettiin työstämään aikataulullisesti. Projektin aikataulut-taminen tapahtui noin viikon mittaisissa sykleissä alustavasti. Mikäli aikataulu ja projekti vaativat, niin syklien kestoa voitiin lyhentää, jolloin projektin työstäminen tiivistyi.
Oikeita työkaluja projektin toteuttamiseen alettiin etsiä siinä vaiheessa, kun työn toteutus oli kartoitettu sillä tasolla, että tiedettiin mitä ja miten sovelluksen tulee tehdä halutut toiminnot. Työkalujen valintaan vaikuttivat myös niiden saatavuus, saatavilla oleva ohjeistus, yhteensopivuus käytettävän käyttöjärjestelmän kans-sa sekä työkalujen hinta. Mikäli mahdollista työ toteutettaisiin ilmaisilla kehitys-työkaluilla.

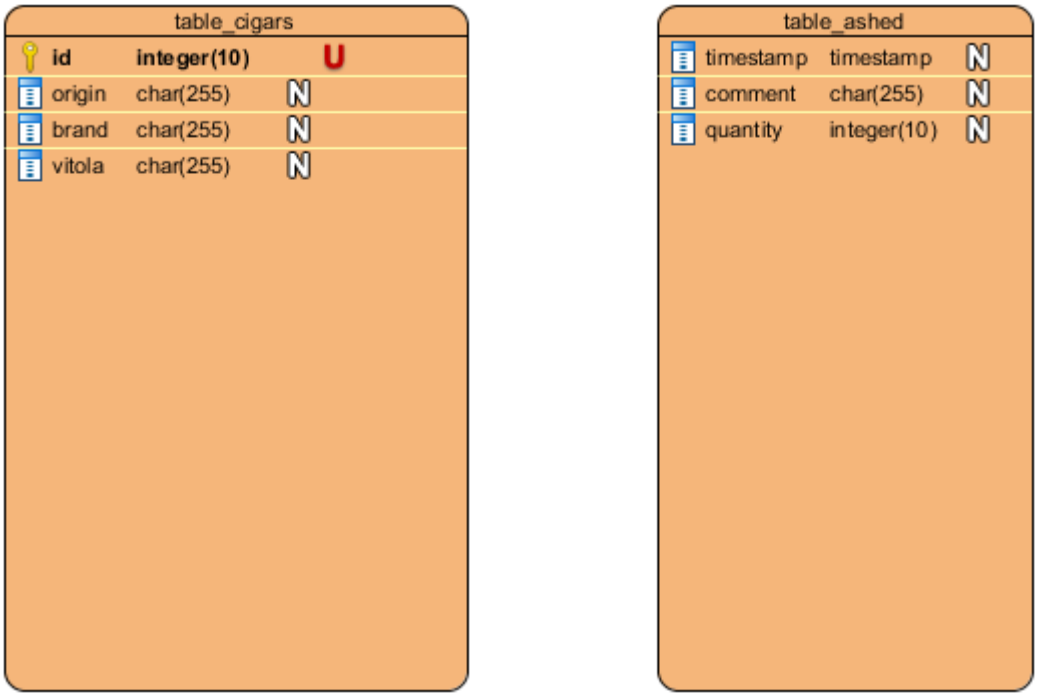
3.1 Sovelluksen suunnittelu ja kuvaus

Tavoitteena oli luoda sovellus, joka pitää kirjaa poltetuista sikareista eli toimii muistikirjana matkapuhelimessa. Sovelluksen pitää pystyä kirjoittamaan, luke-maan, muokkaamaan ja poistamaan tietoja. Tietoja sovellukseen syötetään sa-toja kertoja ja samanlaisia sikareita poltettaessa uudelleen on käytettävyyden kannalta mielekkäämpää, ettei tietoja tarvitse joka kerta syöttää uudelleen. Tie-tojen määrän ja tiedon kertautumisen vuoksi tietojen tallentamiseksi päädyttiin käyttämään SQLite-tietokantaa.

Pääpaino sovelluksen kehittämisessä oli saada toimiva prototyyppi, joka pitää sisällän aiemmin mainitut ominaisuudet. Mikäli tavoitteeseen päästään ja aikaa on käytettävissä, on mahdollista lisätä sovellukseen myös kirjanpito säilytykses-sä oleville sikareille sekä kiinnittää huomiota sovelluksen ulkonäköön.

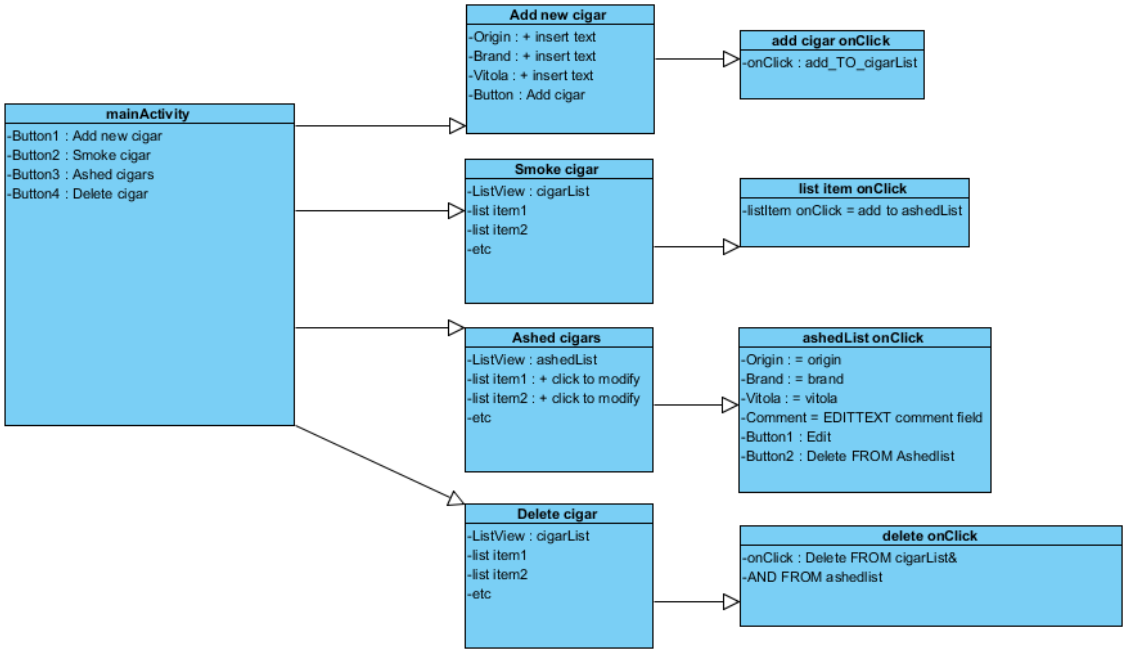
Sovelluksen suunnittelu aloitettiin määrittelemällä, mitä tietoja sovellukseen tul-laan tallentamaan ja montako tietokantataulua tarvitaan. Tallennettavia tietoja ovat sikarin yksilöivä tunniste eli ID, origin (alkuperä), brand (sikarin valmistaja), vitola (sikarin malli), päivämäärä jolloin sikari on poltettu, montako sikaria polte-

taan ja kommentti. Tietokantatauluja sovellukseen tuli kaksi kappaletta, joista toinen pitää sisällään sikareiden tiedot ja toinen tiedot polttoon liittyvistä tiedoista (kuva 1).



Kuva 1. Tietokannan taulut (Kuva: Eetu Sormunen).

Seuraavaksi suunniteltiin graafisen käyttöliittymän rakenne. Käyttöliittymän suunnittelun yhteydessä mietittiin sovelluksen toiminnallisuutta. Toiminnallisuuden suunnittelun ansiosta saatiin hyvä yleiskuva, kuinka sovellusta kannattaa lähteä toteuttamaan (kuva 2). Kuvassa 2 kuvattiin, millaisia näkymiä sovellukseen tulee ja napautuksien toiminnallisuudet.



Kuva 2. Käyttöliittymän rakenne ja toiminnallisuus (Kuva: Eetu Sormunen).

3.2 Kehitysympäristön valinta

Kehitysympäristöksi valittiin jo käytössä oleva Windows 7, Visual Paradigm, SQLiteSpy sekä Android Studio, joka on Androidin ilmainen ja virallinen kehitysalusta. Android Studiolle olisi ollut vaihtoehtona Eclipse, joka on myös ilmainen ohjelmointiympäristö. Molemmat ovat hyviä vaihtoehtoja ohjelmointiympäristöiksi, mutta jo valmiiksi asennettuna ollut Android Studio oli helpompi ottaa käyttöön.

Android Studio on Androidin ilmainen ja virallinen kehitysalusta. Android Studion ensimmäinen vakaa versio julkaistiin joulukuussa 2014. Koska Android Studio on virallinen Androidin kehitysympäristö, se on myös monipuolisempi ominaisuuksiltaan verrattuna Eclipseen (kuva 3). Android Studio osaa esimerkiksi täydentää kehittyneemmin Android-ohjelmakoodia ja refaktoroida sitä toisin kuin Eclipse. (Wikipedia 2016b.)

Feature	Android Studio	Eclipse ADT
Build system	Gradle	Apache Ant
Maven-based build dependencies	Yes	No
Build variants and multiple-APK generation	Yes	No
Advanced Android code completion and refactoring	Yes	No
Graphical layout editor	Yes	Yes
APK signing and keystore management	Yes	Yes
NDK support	Yes	Yes

Kuva 3. Ominaisuuksien vertailu (Kuva: Wikipedia 2016)

3.3 Ashed-sovelluksen toteutus

Sovelluksen käyttöliittymä määriteltiin xml-tiedostoilla. Niillä luodaan näkymiä (views), jotka käyttäjä näkee sovelluksessa. Jokaista näkymää kohden on Java-luokka, joka pitää sisällään näkymän toiminnallisuuden. Sovelluksen tiedot tallennetaan SQLite-tietokantaan SQLiteOpenHelper-luokasta perityvän Data-Handler-luokan metodien avulla.

3.3.1 Päänäkymä

Ensin luotiin päänäkymä, jonka käyttäjä näkee ensimmäisenä sovelluksen avauduttua (kuva 4). Päänäkymään lisättiin määritellyt painikkeet (kuva 2) sekä sovelluksen virheenkorjausta eli debuggausta auttamaan yksi lisäpainike DUMP DATABASE.

Kuva 4. Päänäkymä eli activity_main.xml (Kuva: Eetu Sormunen)

Päänäkymän käyttöliittymän muotoilu tapahtuu activity_main.xml-tiedostossa. Tiedosto vaaditaan käyttöliittymän näyttämistä varten. Tiedostossa määritellään käyttöliittymän ulkoasu sekä painikkeiden nimet. Päänäkymän toiminnallisuudesta vastaa MainActivity-luokka, joka myös lataa activity_main.xml-tiedoston onCreate()-metodissa, kun sovellus käynnistetään (kuva 5).

```
public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        requestWindowFeature(Window.FEATURE_NO_TITLE);
        getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN, WindowManager.LayoutParams.FLAG_FULLSCREEN);
        setContentView(R.layout.activity_main);
    }
```

Kuva 5. onCreate()-metodi (Kuva: Eetu Sormunen)

3.3.2 Tietokannan toteutus

SQLite-tietokannan käyttämiseen luotiin SQLiteOpenHelper-luokasta periytyvä DataHandler-luokka (kuva 6). DataHandler-luokka luo tietokannan, mikäli sitä ei valmiiksi ole ja päivittää sitä. DataHandler-luokkaan määriteltiin myös tietokannan nimi, versio, taulujen nimet sekä mitä tietoja tauluihin syötetään.

```
public class DataHandler extends SQLiteOpenHelper {

    private static final int DATABASE_VERSION = 1;
    private static final String DATABASE_NAME = "cigarDB.db";
    private static final String TABLE_CIGARS = "cigars";
    private static final String TABLE_ASHED = "ashed";

    //Cigars table
    public static final String COLUMN_ID = "_id";
    public static final String COLUMN_ORIGIN = "origin";
    public static final String COLUMN_BRAND = "brand";
    public static final String COLUMN_VITOLA = "vitola";

    //Ashed table
    public static final String ASHED_ID = "ashed_id";
    public static final String ASHED_COMMENT = "ashed_comment";
    public static final String ASHED_TIMESTAMP = "ashed_timestamp";

public DataHandler(Context context, String name, SQLiteDatabase.CursorFactory factory, int version){
    super(context, DATABASE_NAME, factory, DATABASE_VERSION);
}
```

Kuva 6. DataHandler-luokka (Kuva: Eetu Sormunen)

Tietokannan taulujen luomisesta vastaa onCreate()-metodi ja päivittämisestä onUpgrade-metodi (Kuva 7).

```
// Create tables
@Override
public void onCreate(SQLiteDatabase db){
    String CREATE_CIGARS_TABLE = " CREATE TABLE IF NOT EXISTS " + TABLE_CIGARS +
            "(" + COLUMN_ID + " INTEGER PRIMARY KEY, " + COLUMN_ORIGIN + " TEXT, " + COLUMN_BRAND +
            " TEXT, " + COLUMN_VITOLA + " TEXT)";

    String CREATE_ASHED_TABLE = " CREATE TABLE IF NOT EXISTS " + TABLE_ASHED +
            "(" + ASHED_ID + " INTEGER, "  + ASHED_COMMENT + " TEXT, " + ASHED_TIMESTAMP + " INTEGER)";
    db.execSQL(CREATE_CIGARS_TABLE);
    db.execSQL(CREATE_ASHED_TABLE);
}

@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) { onCreate(db); }
```

Kuva 7. Metodit onCreate() ja onUpgrade() (Kuva: Eetu Sormunen)

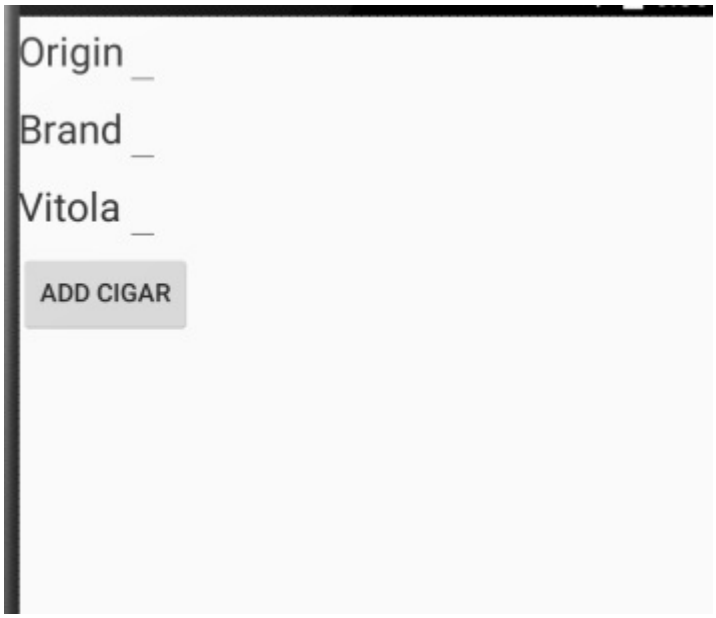
Tietojen ottamiseksi vastaan käyttäjän syötteistä ja niiden välittämiseksi Data-Handler-luokan metodeille, luotiin kaksi luokkaa Cigar (kuva 8) ja Ashed. Luokassa Cigar luodaan objekti cigar, joka ottaa vastaan tiedot, jotka tallennetaan TABLE_CIGARS-tauluun. Vastaavasti Ashed-luokassa luodaan objekti ashed, joka välittää tiedot TABLE_ASHED-tauluun.

```
public class Cigar implements Serializable {
    private int _id;
    private String _origin;
    private String _brand;
    private String _vitola;


    //Constructors
    public Cigar(){
    }

    public Cigar(String origin, String brand,  String vitola) {
        this._origin = origin;
        this._brand = brand;
        this._vitola = vitola;
    }

    //Setters & getters
    public void setID(int id) { this._id=id; }
    public int getID() { return this._id; }
    public void setOrigin(String origin) { this._origin = origin; }
    public String getOrigin() { return this._origin; }
    public void setBrand(String brand) { this._brand = brand; }
    public String getBrand() { return this._brand; }
    public void setVitola(String vitola) { this._vitola = vitola; }
    public String getVitola() { return this._vitola; }
}
```

Kuva 8. Cigar-luokka (Kuva: Eetu Sormunen)

3.3.3 Uuden sikarin lisäys

Uuden sikarin lisäämiseksi TABLE_CIGARS-tauluun luotiin uusi näkymä activity_add_new_cigar.xml (kuva 9) ja ohjausluokka AddNewCigarView.



Kuva 9. activity_add_new_cigar visuaalinen näkymä (Kuva: Eetu Sormunen)

Käyttäjän syötteen vastaanottamiseksi luotiin kolme <EditText>-kenttää joiden kautta näkymä activity_add_new_cigar saa syötteen. Otetaan esimerkiksi origin-tiedon välittäminen TABLE_CIGARS-tauluun. Käyttäjä antaa syötteen EditText-kenttään. Kentän ID on cigarOrigin, (kuva 10) jonka avulla tieto löytyy ohjausluokasta.

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceLarge"
    android:text="Origin"
    android:id="@+id/textView4" />

<EditText
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/cigarOrigin"/>
```

Kuva 10. Käyttöliittymäkomponentti cigarOrigin (Kuva: Eetu Sormunen)

Ohjausluokassa onCreate()-metodin sisällä oleva findViewById(R.id.cigarOrigin)-metodi luo objektin, joka viittaa näkymässä luotuun cigarOrigin -käyttöliittymäkomponenttiin. Näin saadaan käyttäjän syöttämä tieto talteen (kuva 11).

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_add_new_cigar);
    originBox = (EditText) findViewById(R.id.cigarOrigin);
    brandBox = (EditText) findViewById(R.id.cigarBrand);
    vitolaBox = (EditText) findViewById(R.id.cigarVitola);
}
```

Kuva 11. onCreate-metodi (Kuva: Eetu Sormunen)

Painiketta ADD CIGAR (kuva 9) napautettaessa ohjausluokan metodi newCigar() (kuva 12) luo uuden Cigar-luokan objektin käyttäjän syötteestä ja välittää sen DataHandler-luokan metodille addCigar(), joka tallentaa objektin tietokannan tauluun TABLE_CIGARS (kuva 13).

```
public void newCigar(View view) {
    DataHandler dbHandler = new DataHandler(this, null, null, 1);
    Cigar cigar = new Cigar(originBox.getText().toString(), brandBox.getText().toString(), vitolaBox.getText().toString());
    dbHandler.addCigar(cigar);
    originBox.setText("");
    brandBox.setText("");
    vitolaBox.setText("");

}
```

Kuva 12. newCigar()-metodi (Kuva: Eetu Sormunen)

```
public void addCigar(Cigar cigar){
    SQLiteDatabase db = this.getWritableDatabase();
    ContentValues values = new ContentValues();
    values.put(COLUMN_ORIGIN, cigar.getOrigin());
    values.put(COLUMN_BRAND, cigar.getBrand());
    values.put(COLUMN_VITOLA, cigar.getVitola());
    db.insert(TABLE_CIGARS, null, values);
    db.close();
}
```

Kuva 13. addCigar()-metodi (Kuva: Eetu Sormunen)

3.3.4 Polta sikari

Sikarin siirtäminen TABLE_CIGARS-taulusta TABLE_ASHED-tauluun tapahtuu kutsumalla MainActivity-luokan metodia smokeCigarViewClick() (kuva 14), joka avaa listanäkymän activity_cigar_list.xml, jonka ohjausluokkana toimii CigarList-luokka. Cigarlist-luokka luo listanäkymän kaikista sikareista, jotka ovat tallennettuna TABLE_CIGARS-tauluun.

```
public void smokeCigarViewClick(View view) {
    Intent i = new Intent(this, CigarList.class);
    i.putExtra("type", "add");
    startActivity(i);
}
```

Kuva 14. smokeCigarViewClick()-metodi (Kuva: Eetu Sormunen).

CigarList-luokassa kutsutaan DataHandler-luokan metodia getAllCigars(), joka palauttaa TABLE_CIGARS-taulusta kaikki tiedot (kuva 15).

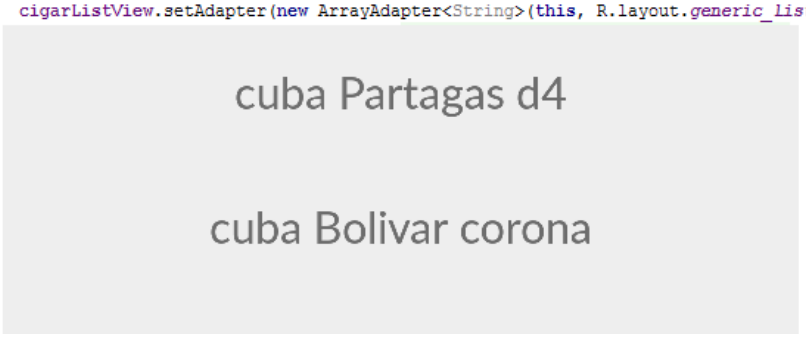
```
public ArrayList<Cigar> getAllCigars () {
    ArrayList<Cigar> cigarList = new ArrayList<>();
    String query = "Select * FROM " + TABLE_CIGARS;
    SQLiteDatabase db = this.getWritableDatabase();
    Cursor cursor = db.rawQuery(query, null);
    while (cursor.moveToNext()) {
        Cigar cig = new Cigar();
        cig.setID(cursor.getInt(0));
        cig.setOrigin(cursor.getString(1));
        cig.setBrand(cursor.getString(2));
        cig.setVitola(cursor.getString(3));
        cigarList.add(cig);
    }
    cursor.close();
    return cigarList;
}
```

Kuva 15. getAllCigars()-metodi (Kuva: Eetu Sormunen)

Metodi getAllCigars() palauttaa tiedot objekteina, jotka syötetään listaan ArrayList<cigar> cigarList. ArrayList<String> cigarNamesList-listaan asetetaan

merkkijonona cigarList-listasta palautetun objektin arvot origin, brand ja vitola ja näytetään käyttäjälle listauksena (kuva 16).

```
private void initializeCigarList() {
    final DataHandler dbHandler = new DataHandler(this, null, null, 1);
    cigarList = dbHandler.getAllCigars();
    cigarNamesList = new ArrayList<>();
    if (cigarList != null && cigarList.size() > 0) {
        for (Cigar cig : cigarList) {
            cigarNamesList.add(cig.getOrigin() + " " + cig.getBrand() + " " + cig.getVitola());
        }
        cigarListView.setAdapter(new ArrayAdapter<String>(this, R.layout.generic_list_item, cigarNamesList));
```



Kuva 16. Tietojen näyttäminen käyttäjälle (Kuva: Eetu Sormunen)

Käyttäjän napautettaessa listanäkymästä sikaria kutsutaan DataHandler-luokan addAshedCigar()-metodi, joka lisää TABLE_ASHED-tauluun valitun sikarin poltetuksi sikariksi (kuva 17).

```
public void addAshedCigar(Ashed ashed){
    SQLiteDatabase db = this.getWritableDatabase();
    ContentValues values = new ContentValues();
    values.put(ASHED_ID, ashed.getID());
    values.put(ASHED_TIMESTAMP, ashed.getTimestamp());
    db.insert(TABLE_ASHED, null, values);
    db.close();
}
```

Kuva 17. addAshedCigar()-metodi (Kuva: Eetu Sormunen)

3.3.5 Poltetut sikarit

Poltettujen sikareiden tarkasteluun tehtiin AshedList-luokka sekä vastaava näkymä activity_ashed_list.xml. Näkymä vastaa pitkälti activity_cigar_list näkymää. Listaan haetaan tiedot TABLE_ASHED-taulusta kutsumalla DataHandler-luokasta metodia getAllAshed() (kuva 18), joka palauttaa poltetut sikarit aikajär-

jestyksessä uusimmasta vanhimpaan. Samalla kutsutaan DataHAndler-luokan metodia findCigar() (kuva 19), jonka avulla palautetaan sikarin tiedot TABLE_CIGARS-taulusta.

```
public ArrayList<Ashed> getAllAshed () {
    ArrayList<Ashed> ashedList = new ArrayList<>();
    String query = "Select * FROM " + TABLE_ASHED + " order by " + ASHED_TIMESTAMP + " desc";
    SQLiteDatabase db = this.getWritableDatabase();
    Cursor cursor = db.rawQuery(query, null);
    while (cursor.moveToNext()) {
        Integer cigarId = cursor.getInt(0);
        Cigar cig = findCigar(cigarId);
        Ashed ashed = new Ashed();
        ashed.setID(cigarId);
        ashed.setComment(cursor.getString(1));
        ashed.setTimestamp(cursor.getLong(2));
        ashed.setCigar(cig);
        ashedList.add(ashed);
    }
    cursor.close();
    return ashedList;
}
```

Kuva 18. getAllAshed()-metodi (Kuva: Eetu Sormunen)

```
public Cigar findCigar(Integer cigarId){
    String query = " Select * FROM " + TABLE_CIGARS + " WHERE " + COLUMN_ID + " = " + cigarId;
    SQLiteDatabase db = this.getWritableDatabase();
    Cursor cursor = db.rawQuery(query, null);
    Cigar cigar = new Cigar();
    if (cursor.moveToFirst()) {
        cursor.moveToFirst();
        cigar.setID(Integer.parseInt(cursor.getString(0)));
        cigar.setOrigin(cursor.getString(1));
        cigar.setBrand(cursor.getString(2));
        cigar.setVitola(cursor.getString(3));
    }
    else {
        cigar = null;
    }
    cursor.close();
    db.close();
    return cigar;
}
```

Kuva 19. findCigar()-metodi (Kuva: Eetu Sormunen)

Käyttäjälle näkymä on lähes sama kuin poltetuissa sikareissa sillä erotuksella, että poltetuissa sikareissa näytetään lisäksi päivämäärä, jolloin sikari on poltettu (kuva 20).

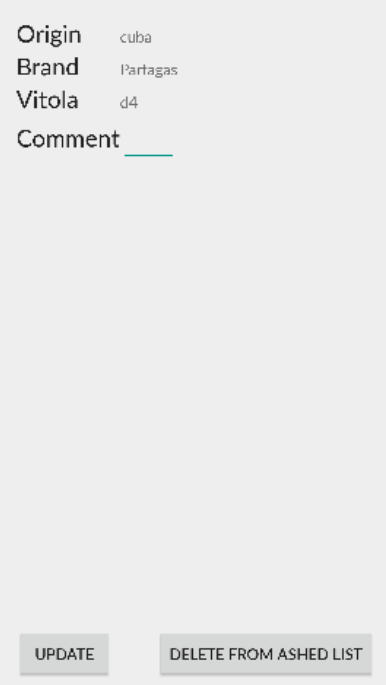
Kuva 20. Poltettujen sikareiden listanäkymä (Kuva: Eetu Sormunen)

3.3.6 Kommentin lisäys ja muokkaus sekä poltetun sikarin poistaminen

Poltettujen sikareiden kommenttien lisäämiseksi ja niiden muokkaamiseksi sekä poltettujen sikareiden poistamiseksi TABLE_ASHED-taulusta tehtiin AshedDetails-luokka (kuva 21) sekä sille vastaava näkymä activity_ashed_details.xml (kuva 22). Napautettaessa sikarin nimeä (kuva 20) avataan activity_ashed_details-näkymä ja välitetään sille Ashed-luokan objekti ashedObj. (kuva 23), joka pitää sisällään tiedot poltetusta sikarista.

```
public class AshedDetails extends AppCompatActivity {

    private TextView origin;
    private TextView brand;
    private TextView vitola;
    private EditText comment;
    private Ashed ashedObj;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        requestWindowFeature(Window.FEATURE_NO_TITLE);
        getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
                WindowManager.LayoutParams.FLAG_FULLSCREEN);
        setContentView(R.layout.activity_ashed_details);
        origin = (TextView) findViewById(R.id.cigarOrigin);
        brand = (TextView) findViewById(R.id.cigarBrand);
        vitola = (TextView) findViewById(R.id.cigarVitola);
        comment = (EditText) findViewById(R.id.ashedComment);
        ashedObj = (Ashed) getIntent().getSerializableExtra("ashedObj");
        initializeAshedDetails(ashedObj);
    }

    private void initializeAshedDetails(Ashed ashed) {
        if (ashed != null) {
            origin.setText(ashed.getCigar().getOrigin());
            brand.setText(ashed.getCigar().getBrand());
            vitola.setText(ashed.getCigar().getVitola());
            comment.setText(ashed.getComment());
        }
    }
```

Kuva 21. AshedDetails-luokka (Kuva: Eetu Sormunen)



Kuva 22. activity_ashed_details.xml-näkymä (Kuva: Eetu Sormunen)

```
ashedListView.setOnItemClickListener((adapterView, view, position, id) -> {
        Intent i = new Intent(context, AshedDetails.class);
        i.putExtra("ashedObj", ashedList.get(position));
        context.startActivity(i);
});
```

Kuva 23. ashedObj (Kuva: Eetu Sormunen)

Näkymään activity_ashed_details luotiin painike Update kommentin talteen ottamiseksi. Update-painiketta painaessa kutsutaan DataHandler-luokan metodia updateComment() (kuva 24).

```
public void updateComment(Integer ashedId, Long timestamp, String comment) {
    String query = " UPDATE " + TABLE_ASHED + " SET " + ASHED_COMMENT + " = \"" + comment +
            "\" WHERE " + ASHED_ID + " = " + ashedId + " AND " + ASHED_TIMESTAMP + " = " + timestamp;
    SQLiteDatabase db = this.getWritableDatabase();
    db.execSQL(query);
}
```

Kuva 24. updateComment()-metodi (Kuva: Eetu Sormunen)

Poltetun sikarin poistamiseksi TABLE_ASHED-taulusta kutsutaan DataHandler-luokan metodia deleteAshed(). Poltot poistetaan taulusta viiteavaimen ashedId:n sekä aikaleiman perusteella. Millisekunnin tarkkuudella oleva aikaleima on riittävän yksilöivä tieto, jotta sen perusteella voidaan varmistua halutun tietueen poistamisesta (Kuva 25).

```
public void deleteAshed(Integer ashedId, Long timestamp) {
    String query = " DELETE FROM " + TABLE_ASHED + " WHERE " + ASHED_ID + " = " + ashedId +
            " AND " + ASHED_TIMESTAMP + " = " + timestamp;
    SQLiteDatabase db = this.getWritableDatabase();
    db.execSQL(query);
}
```

Kuva 25. deleteAshed()-metodi (Kuva: Eetu Sormunen)

3.3.7 Sikarin poistaminen kokonaan sovelluksesta

Sikarin poistamiseksi kokonaan tietokannasta luotiin DataHandler-luokkaan metodi deleteCigar(), joka poistaa sikarin sekä siihen liittyvät poltot tietokannasta (kuva 26).

```
public void deleteCigar(Integer cigarId){
    String query = "DELETE FROM "+ TABLE_CIGARS + " WHERE " + COLUMN_ID + " = " + cigarId;
    String anotherQuery = "DELETE FROM " + TABLE_ASHED + " WHERE " +  ASHED_ID + " = " + cigarId;
    SQLiteDatabase db = this.getWritableDatabase();
    db.execSQL(anotherQuery);
    db.execSQL(query);
}
```

Kuva 26. deleteCigar()-metodi (Kuva: Eetu Sormunen)

3.3.8 Virheenkorjaus

Sovelluksen virheenkorjaamiseen eli debuggaukseen käytettiin fyysistä Android-laitetta. Tietokannan debuggaukseen käytettiin SQLiteSpy-sovellusta ja komentorivityökalua PowerShell, koska käytössä ollut tietokone ei tukenut virtuaalisen laitteen käyttöä Android Studiossa. Päänäkymän Dump database -painiketta painaessa aktivoituu DataHandler-luokan metodia copyDatabse() (kuva 27), joka kopio sovelluksen tietokannan laitteen juurihakemistoon. Tietokantatiedosto avataan SQLiteSpy-sovelluksessa, jonka avulla voidaan tarkastella tauluihin syötettyjä tietoja (kuva 28).

```
public void copyDatabase() {
    SQLiteDatabase db = this.getWritableDatabase();
    File databaseFile = new File(db.getPath());
    File destFile = new File(Environment.getExternalStorageDirectory().getAbsolutePath() +
            File.separator + "CigarDB.db");
    try {
        if (!destFile.exists()) {
            destFile.createNewFile();
        }
        InputStream inStream  = new FileInputStream(databaseFile);
        OutputStream outStream  = new FileOutputStream(destFile);
        byte[] buffer = new byte[1024];

        int length;
        //copy the file content in bytes
        while ((length = inStream.read(buffer)) > 0){
            outStream.write(buffer, 0, length);
        }
        if(inStream != null) {
            inStream.close();
        }
        if(outStream != null) {
            outStream.close();
        }
        Log.d("DBDUMP", "Database copied from " + databaseFile.getAbsolutePath() + " to " +
                destFile.getAbsolutePath());
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Kuva 27. copyDatabase()-metodi (Kuva: Eetu Sormunen)

Kuva 28. Tietokantataulut TABLE_CIGARS ja TABLE_ASHED SQLiteSpy:n näkymässä (Kuva: Eetu Sormunen)

4 Pohdinta

Tässä luvussa käyn läpi, minkälaisia ajatuksia sovelluksen kehittäminen ja kehitystyö herättivät minussa sekä vastaan tulleisiin haasteisiin ja teknisiin ongelmiin. Käyn myös läpi, minkälaisia ominaisuuksia sovellukseen voitaisiin lisätä tulevaisuudessa.

4.1 Opinnäytetyön lopputulos

Opinnäytetyön tavoitteena oli kehittää poltettujen sikareiden kirjanpitoon toimiva sovellus, jota voidaan jatkossa kehittää edelleen. Aikaisempaa kokemusta vastaavan laajuisesta yksin toteutetusta sovelluskehityksestä ei ollut, joten kokemus oli uusi ja haastava sekä opettavainen. Opinnäytetyön suunnittelu tarpeeksi tarkalla tasolla oli yksi tärkeimmistä edellytyksistä valmiin lopputuloksen aikaansaamiseksi käytettävissä olevan ajan sisällä. Tietokannan rakenteen ja visuaalisen käyttöliittymän sekä toiminnallisuuden määrittelyn ollessa selvillä oli huomattavasti helpompaa lähteä etenemään kohti varsinaista toteutusta kuin jos olisi alkanut ohjelmoida ilman konkreettista suunnitelmaa. Opinnäytetyöhön liittyvään sovellukseen löytyi erittäin vähän suomenkielistä dokumentaatiota. Syy lienee pääasiassa alalla hallitseva käytäntö, jonka mukaan ohjelmoinnissa käytetään pääasiallisena kielenä englantia. Englanninkielisenä dokumentaatioita löytyi huomattavasti paremmin. Kehitystyön aikana huomasin usein, ettei moneen vastaan tulleeseen ongelmaan ja kysymykseen löytynyt vastausta Androidin virallisesta dokumentaatiosta. Onneksi Android-sovelluskehitys on suosittua, joten ongelmiin löytyi monesti vastaus erilaisten Android-kehitysyhteisöjen keskustelualueilta sekä kokeneempien sovelluskehittäjien blogeista. Opinnäytetyön edetessä sain huomata, kuinka aloitteleva ohjelmoija olen. Useassa kehitysvaiheessa olin suunnitellut tekeväni toiminnon tietyllä tavalla ja siihen liittyvää lisätietoa etsiessäni, toimintoon löytyi kokeneemman oh-

jelmoijan ratkaisu, joka oli paljon järkevämpi tapa toteuttaa toiminto kuin mitä olin itse aluksi ajatellut. Laajan käyttäjäpohjan ja kehittäjäyhteisöjen tuen laskisinkin yhdeksi Android-ohjelmoinnin suurimmaksi eduksi aloittelevalle Android-kehittäjälle.

Teknisiltä ongelmilta ei työn aikana vältytty. Työtä aloittaessa huomasin, että Android Studion virtuaalinen laite toimii erittäin hitaasti ja sovelluksen debuggaus sen kautta oli tuskallisen hidasta. Onnekseni minulla oli Android-käyttöliittymää käyttävä matkapuhelin, jota pystyin käyttämään sovelluskehityksessä apuna. Myöhemmässä vaiheessa sain huomata, että tietokoneeni prosessori ei tukenut enää virtuaalilaitetta. En uskonut siitä koituvan ongelmia, sillä sovelluskehitys oli onnistunut erinomaisesti fyysisellä laitteella. Huomasin kuitenkin siinä vaiheessa tarvitsevani virtuaalista laitetta, kun tietokantatiedosto piti saada matkapuhelimen muistista tietokoneelle SQLiteSpy-sovelluksella tarkasteltavaksi. Koska SQLite tallentaa tietokannan laitteen sisäiseen muistiin, siihen ei pääse käsiksi USB-kaapelilla. Virtuaalilaitteen sisäiseen muistiin olisin päässyt suoraan Android Studion kautta, mutta uusimman Android Studion versiopäivityksen yhteydessä tietokoneeni prosessoria ei enää tuettu virtuaalilaitteen käytössä. Ongelmaan löytyi onneksi ratkaisu kopioimalla tietokanta matkapuhelimen ulkoiseen muistiin, josta tietokannan sai kopioitua työpöydälle.

Suurimmaksi haasteeksi osoittautui lopulta kirjallisen raportin käytännön työn dokumentointi. Käytännön työn dokumentointi järkevällä rakenteella oli odotettua haastavampaa ja sen valmiiksi saaminen laittoi opinnäytetyön aikataulun koetukselle. Työn alkuvaiheessa oli tarkoitus dokumentoida työtä sitä mukaa kun se edistyy, mutta varsinkin alussa koodi muuttui jatkuvasti niin paljon, että dokumentointi vei mielestäni kohtuuttomasti aikaa ja hidasti valmiin sovelluksen toteutusta. Asetinkin toimivan sovelluksen aikaansaamisen tärkeysjärjestyksessä ensimmäiseksi ja päätin dokumentoida työprosessin vasta sovelluksen ollessa valmis. Ratkaisu oli toimiva, sillä sovellus saatiin valmiiksi ajoissa. Dokumentointi oli kuitenkin oletettua huomattavasti haastavampaa jälkikäteen. Uusien vasta opeteltujen asioiden kuvaaminen raporttiin ymmärrettävästi jälkikäteen vaati useaan otteeseen palaamaan takaisin lähdekoodin pariin ja palauttamaan mieleen miten kyseinen toteutus tehtiin. Yleiskuva sovelluksen toimivuudesta oli

koko ajan vahvasti selvillä, mutta osittain uusien asioiden sisäistäminen tulee tulevaisuudessa syventymään sovelluksen jatkokehityksen aikana.

Kokonaisuudessaan opinnäytetyö oli haastava ja opin valtavasti uusia asioita Android-ohjelmoinnista sekä kiinnostukseni Android-sovelluskehitykseen syveni entisestään. Asioita joita tekisin opinnäytetyöprosessissa toisin olisi ehdottomasti dokumentointi sitä mukaa, kun työ edistyy. Silläkin uhalla, ettei toimivaa sovellusta olisi saatu aikaiseksi. Hyvä dokumentointi auttaa myös oppimisprosessissa, kun käy asioita itselleen läpi sitä mukaa, kun uusia asioita tulee eteen.

4.2 Sovelluksen kehityskohteet tulevaisuudessa

Sovelluksen kehityskohteet tulevaisuudessa ovat ulkoasun parantaminen, tietokantakyselyiden toteuttaminen turvallisemmin, kuvien lisääminen sekä tietokannan laajentaminen säilytyksessä oleville sikareille. Opinnäytetyössä aikaansaatu sovellus ja tietotaito antavat minulle hyvän pohjan lähteä työstämään sovellusta julkiseen levitykseen kelpaavaa versiota kohti sekä uusien Android-sovellusten kehittämiseen. Näiden tavoitteiden vuoksi täytyy opiskella lisää Android-sovellusten tietoturvaa sekä perehtyä tarkemmin siihen, kuinka luodaan visuaalisesti paremman näköisiä käyttöliittymiä, sillä nyt valmistuneen sovelluksen käyttöliittymä on hyvin pelkistetty.

Lähteet

Android Developers 2016a. Saving Data. http://developer.android.com/training/basics/data-storage/index.html. 16.3.2016

Android Developers 2016b. Saving Files. http://developer.android.com/training/basics/data-storage/files.html. 16.3.2016

Android Developers 2016c. Saving Key-Value Sets. http://developer.android.com/training/basics/data-storage/shared-preferences.html. 16.3.2016

Android Developers 2016d. Using the Internal Storage. http://developer.android.com/guide/topics/data/data-storage.html#filesInternal. 16.3.2016

Android Developers 2016e. Using the External Storage. http://developer.android.com/guide/topics/data/data-storage.html#filesExternal. 16.3.2016

Android Developers 2016f. Saving Data in SQL Databases. http://developer.android.com/training/basics/data-storage/databases.html. 16.3.2016

Android Developers 2016g. Android Studio Overview. http://developer.android.com/tools/studio/index.html. 23.3.2016

Eclipse 2016. http://www.eclipse.org/org/#about. 23.3.2016

SQLite.org 2016. http://www.sqlite.org/whentouse.html. 16.3.2016

Wikipedia 2016a. MySQL. https://fi.wikipedia.org/wiki/MySQL. 17.3.2016

Wikipedia 2016b. Android Studio. https://en.wikipedia.org/wiki/Android_Studio. 23.3.2016