SQLITE-TIETOKANTAA HYÖDYNTÄVÄ ANDROID-SOVELLUS

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Ammattikorkeakoulun opinnäytetyö

Tietojenkäsittelyn koulutusohjelma

Visamäki, kevät 2016

Karoliina Elo

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

TIIVISTELMÄ

Visamäki
Tietojenkäsittelyn koulutusohjelma
Systeemityö

Tekijä Karoliina Elo **Vuosi** 2016

Työn nimi SQLite-tietokantaa hyödyntävä Android-sovellus

TIIVISTELMÄ

Opinnäytetyön tavoitteena oli oppia luomaan toimiva Android-sovellus, joka hyödyntää SQLite-tietokantaa. Työssä päätettiin lähteä toteuttamaan Digital Service Development -kurssilla suunniteltua sovellusta, jonka avulla voitaisiin opiskella ja kerrata japanilaisia kirjoitusmerkkejä. Työn toimeksiantajana toimi Hämeen ammattikorkeakoulu, jolle opinnäytetyön aikana luotu sovellus tulisi opetuskäyttöön esimerkkisovelluksena. Esimerkkisovelluksen tarkoituksena oli toimia tietovisatyyppisen sovelluksen pohjana, jota voisi helposti kustomoida tarpeisiinsa tietokantaa vaihtamalla.

Koska aiempaa kokemusta Android-ohjelmoinnista ei ollut, tutustuttiin työn teoriaosuudessa Androidin perusteisiin. Tämän lisäksi käytiin perusteita läpi myös SQLite-tietokantajärjestelmästä. Aineistona käytettiin muun muassa Googlen dokumentaatiota Androidin kehittäjille sekä Jukka Harjun kirjaa Android-ohjelmoinnin perusteet. Opittuja asioita hyödynnettiin työn käytännön osuudessa, jossa toteutettiin sovellus.

Android osoittautui työtä tehdessä aloittelijalle helposti lähestyttäväksi vaihtoehdoksi, ja siitä löytyikin kattava dokumentaatio. Myös Googlen oma sovelluskehitin Android Studio oli suhteellisen helppokäyttöinen ja selkeä. Varsinkin käyttöliittymän luominen sen avulla kävi vaivattomasti.

Opinnäytetyön aikana luodun sovelluksen kehittämistä aiotaan jatkaa myös vapaa-ajalla. Seuraavina sovellukseen aiotaan lisätä myös loput kurssilla suunnitellut tehtävät sekä toteuttaa opiskelupuoli. Lisäksi kanjien valintaan aiotaan toteuttaa myös toiminto, jonka avulla käyttäjä voi luoda omia harjoiteltavien kanjien listoja.

Avainsanat Android, mobiilisovellus, ohjelmointi, SQLite, tietokanta

Sivut 35 s. + liitteet 6 s.

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Visamäki
Degree Programme in Business Information Technology
System Engineering

Author Karoliina Elo **Year** 2016

Subject of Bachelor's thesis Android application that uses SQLite database

ABSTRACT

The goal of this thesis was to learn how to create a functional Android application that uses SQLite database. The application that was created for the thesis was designed during the Digital Service Development -course. It was made for the learning and rehearsing of Japanese characters, kanji. The client of this work was Häme University of Applied Sciences to whom the application was made for as an example to be used in teaching. The example application was designed as a quiz application that could be easily modified by changing the database.

The theoretical section of this work handles the basics of Android programming since there was no previous experience in it. The SQLite database engine is also handled in this section. The theory was gathered mainly from Android Developers -site and from Jukka Harju's Finnish book about the basics of Android programming. The theory for the SQLite database was mainly from SQLite's homepage. This was all put in action in the practical section of the thesis where the application was created.

Android turned out to be rather easy to learn for a beginner and there was comprehensive documentation about it. The use of Google's own application Android Studio was also very user-friendly and easy to learn. Especially creating the user interface was quite effortless.

The development of this application is planned to continue even after this thesis. The features planned to be implemented in the application next include a studying section, more exercises and the possibility to create your own lists of kanji for the rehearsing section.

Keywords Android, mobile application, programming, SQLite, database

Pages 35 p. + appendices 6 p.

SISÄLLYS

TERMISTÖ

Android
Linux-pohjainen mobiilikäyttöjärjestelmä

Emulaattori
tietokoneohjelma tai laitteistolaajennos, joka mahdollistaa eri käyttöjärjestelmille tai laitteille suunniteltujen ohjelmien käytön.

XML (Extensible Markup Language)
Merkintäkieli, jota käytetään tiedon kuvaukseen tai formaattina tiedon tallentamiseen.

Manifestitiedosto
XML-tiedosto, jossa määritetään kaikki sovelluksen käyttämät komponentit, API-versio, tarvitut käyttöoikeudet sekä laitteistovaatimukset.

Attribuutti
SGML- ja XML-kielten elementtiin kuuluva nimetty lisämääre.

Moduuli
Itsenäinen osa, joista voi koota erilaisia kokonaisuuksia.

Metodi
Aliohjelma, joka suorittaa sille määritettyjä toimintoja.

1 JOHDANTO

Työn tavoitteena on oppia luomaan toimiva Android-sovellus, joka hyödyntää SQLite-tietokantaa. Aihe työlle muodostui omasta tarpeestani sovellukselle, jonka avulla voisin helposti kerrata japanilaisia kirjoitusmerkkejä, sekä kiinnostuksestani ohjelmointiin. Erillistä toimeksiantajaa työllä ei alun perin ollut. Sovelluksen suunnittelu aloitettiin jo Digital Service Development -kurssilla, jolloin keskityttiin suunnittelemaan sovelluksen sisältöä sekä tehtiin käyttöliittymäsuunnitelmat. Myöhemmin sovellusta lähdettiin kuitenkin työstämään ennemminkin tietovisasovelluksen pohjana, josta tietokantaa muuttamalla saisi helposti muokattua omiin tarpeisiinsa sopivan. Tällöin sovelluksen kohdeyleisöä saataisiin laajennettua.

En ollut aiemmin ohjelmoinut Androidilla, joten koin opinnäytetyön olevan hyvä tilaisuus siihen. Androidin ollessa yksi tämän hetken suosituimmista mobiilikäyttöjärjestelmistä, koen hyötyväni osaamisesta tulevaisuudessa. Keskityn opinnäytetyöni aikana luomaan sovelluksen vain puhelimille, enkä siis lähde työstämään sovellusta tableteille sopivaksi. Kehitysympäristönä toimii Android Studio. Testaukseen käytetään emulaattoria ja fyysistä laitetta. Opinnäytetyön aikana sovellusta ei tulla julkaisemaan.

Työssä käydään läpi hieman Android-ohjelmoinnin perusteita sekä tiedon varastointia SQLite-tietokantaa käyttäen. Erityisesti kiinnitetään huomiota siihen, miten sovelluksen muunneltavuus tietokantaa muuttamalla tulee ottaa huomioon. Lisäksi pohditaan myös miten erilaiset näyttökoot tulee ottaa huomioon sovellusta toteuttaessa. Työssä käydään läpi myös Digital Service Development -kurssilla tehdyt suunnitelmat sovelluksesta ja mitä osia aiotaan opinnäytetyön aikana toteuttaa. Lisäksi kerrotaan lyhyesti kanjeista, eli japanilaisista kirjoitusmerkeistä, koska niitä käytetään opinnäytetyön esimerkkitietokannassa.

2 SOVELLUS

Opinnäytetyön tavoitteena oli luoda yksinkertainen tietovisatyyppinen sovellus, josta suunniteltiin sellainen, että se olisi helposti muokattavissa tietokantaa vaihtamalla. Esimerkkitietokantana käytettiin kanjeja, mutta tavoitteena oli kuitenkin pitää sovelluksen rakenne sellaisena, että tietokannan vaihto olisi mahdollisimman vaivatonta.

Alustavasi tietokanta vaihdettaisiin käsin sovelluksen koodiin. Jatkokehityksenä sovelluksen tueksi voitaisiin myös luoda ohjelma, jonka kautta oman tietokannan luonti kävisi vaivattomasti myös peruskäyttäjältä. Sovelluksen valikkoon voitaisiin myös luoda valitse tietokanta -painike, jonka kautta käyttäjä voisi vaihtaa tietokantaa ohjelman sisällä. Valittavana voisi olla kaikki käyttäjän luomat tietokannat ja vaihtoehtoisesti voitaisiin hakea muidenkin käyttäjien luomia kantoja. Alkuperäinen sovellus suunniteltiin niin, että vastausvaihtoehdot haettiin satunnaisesti kannasta, mutta joissain tapauksissa voisi olla parempi, että ne lisättäisiin jokaiselle kysymykselle suoraan tietokantaan. Tämä on toinen asia, jota voisi lähteä toteuttamaan jatkossa.

2.1 Sovelluksen suunnittelu

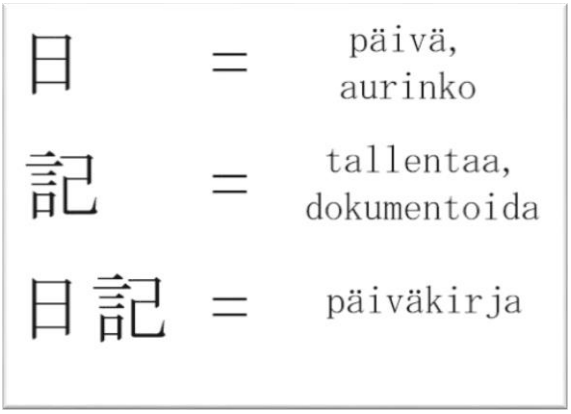
Sovelluksen suunnittelu aloitettiin jo Digital Service Development -kurssilla. Aiheenamme oli luoda mobiilisovellus, joka auttaisi opettelemaan ja kertaamaan japanin kirjoitusmerkkejä, eli kanjeja. Sovellukseen suunniteltiin kaksi eri osiota: kanjien opiskelu ja kanjien kertaus. Opiskelupuolella pääsisi tarkastelemaan kanjin käännöstä, lukutapoja sekä piirtojärjestystä. Lisäksi piirtojärjestystä voisi harjoitella piirtämällä puhelimen näytölle. Kertauspuolelta taas löytyisi erilaisia ”valitse oikea vaihtoehto” -tyyppisiä harjoituksia. Näitä suunniteltiin neljä: valitse oikea kanji, piirrä kanji, valitse oikea lukutapa sekä valitse oikea käännös.

Sovelluksen valikkoon suunniteltiin kirjoitusasun valinta (roomalaiset aakkoset/japanilaiset tavumerkit) sekä kanjien valinta (jaoteltu kouluvuosittain). Lisäksi valikosta voisi muuttaa sovelluksen teemaa ja tehtävien vaikeusastetta.

Opinnäytetyössä päätettiin keskittyä nimenomaan kertauspuolen toteutukseen, eikä siis toteutettaisi opiskelupuolta lainkaan. Tarkoituksena oli toteuttaa ainakin kaksi suunnitelluista harjoituksista, sekä valikkoon kanjien valinta- ja ohje-painikkeet. Sovellus toteutettaisiin myös vain yhdellä kirjoitusasulla: roomajilla, eli roomalaisilla aakkosilla. Opinnäytetyötä varten sovellukseen lisättäisiin ala-asteella opittavat 1006 ensimmäistä kanjia.

2.2 Kanjit

Kanjit ovat japanissa käytettyjä kirjoitusmerkkejä, jotka ovat peräisin kiinasta. Jokaisella yksittäisellä kanjilla on oma merkityksensä, ja yhdistämällä kanjeja saadaan aikaan uusia merkityksiä (Karppinen 2005, 7).



Kuva 1. Merkkejä yhdistämällä voidaan luoda uusia merkityksiä

Vaikka kanjeja onkin monia tuhansia, ei läheskään kaikkia käytetä jokapäiväisessä elämässä. Japanilaisissa peruskouluissa opetetaankin 2136 kanjia (jouyou kanji), joista ensimmäiset 1006 kanjia opitaan ala-asteella (kyouiku kanji) ja loput 1130 kanjia ylä-asteella. Näiden kanjien avulla on jo mahdollista kohtuullisen hyvin lukea japanilaisia sanomalehtiä (Ijas 2013.)

Kanji-merkeillä on tyypillisesti kaksi lukutapaa: japanilainen lukutapa kun-yomi, sekä kiinalainen lukutapa on-yomi. Japanilaista lukutapaa käytetään yleisesti silloin, kun kanji on yksin. Esimerkiksi vettä tarkoittava merkki 「水」luettaisiin ”mizu”. Yhdyssanoissa sen sijaan käytetään pääsääntöisesti kiinalaista lukutapaa. Esimerkiksi elohopeaa tarkoittava sana「水銀」(suigin), jossa yhdistetään merkit ”vesi” sekä ”hopea” luettaisiin käyttäen molempien kanjien kiinalaista lukutapaa ”sui” (vesi) ja *”gin”* (hopea). Toki poikkeuksiakin on. (Kano, Shimizu, Takenaka & Ishii 1989, 2.)

Yhdellä kanjilla voi myös olla monta erilaista japanilaista ja kiinalaista lukutapaa, joiden merkitys voi myös vaihdella jonkin verran. esimerkiksi taivasta tarkoittavalla merkillä 「空」 on kolme erilaista kun-yomia: ”sora”, ”a-ku” ja ”kara”. Näistä ensimmäistä käytetään yleisimmin tarkoittamaan taivasta (ei uskonnollista), toinen taas tarkoittaa ”aukeata” ja viimeinen ”tyhjää”. Joillakin kanjeilla taas on pelkästään kun-yomi tai on-yomi. (Ijas 2013.)

Kanjien lisäksi japanin kielessä on yksinkertaiset tavumerkistöt: hiragana ja katakana, joista ensimmäisellä kirjoitetaan japanilaisten sanat sekä taivutetaan sanoja. Jälkimmäistä puolestaan käytetään yleisemmin vierasperäisissä sanoissa sekä nimissä. Toisin kuin kanjeilla, ei näillä merkeillä siis ole kokonaisen sanan merkitystä. Tavumerkeillä voidaan myös kirjoittaa kokonaisia sanoja, ja usein lapsille tarkoitetut tekstit onkin kirjoitettu pelkästään niillä (Karppinen 2005, 8.).

3 ANDROID-OHJELMOINTI

Android on yksi tämän hetken suosituimmista mobiilikäyttöjärjestelmistä. Alustalle voidaan kehittää sovelluksia esimerkiksi Eclipsen ADT-lisäosaa tai Googlen oman sovelluskehitintä Android Studiota käyttäen. Tässä luvussa käydään läpi muutamia Android-ohjelmoinnin perusasioita, joita tarvittiin sovelluksen toteuttamisessa.

3.1 Aktiviteetit

Aktiviteetti on sovelluksen osa, jonka kanssa käyttäjä on vuorovaikutuksessa. Se luo näkymän, johon piirretään käyttöliittymäelementit, kuten esimerkiksi teksti ja painikkeet. Sovelluksessa on yleensä monia aktiviteetteja, jotka usein täyttävät koko ruudun. Aktiviteetti voidaan kuitenkin myös määritellä näkymään pienempänä ikkunana toisen aktiviteetin päällä. (Google Inc. 2015a.)

Aktiviteetti luodaan toteuttamalla Activity-luokka. Toteuttavan luokan perusrakenteen tulee aina olla sama, ja toteuttaa onCreate-metodi (kuva 2). Tätä metodia kutsutaan silloin, kun aktiviteetti käynnistyy. Metodissa onCreate tulee myös kutsua setContentView-metodia, jonka avulla määritellään haluttu ulkoasu (Google Inc. 2015a.)

```
public class TestiAktiviteetti extends Activity {

    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

    }
}
```

Kuva 2. Aktiviteetin perusrakenne

Jotta aktiviteetti voidaan ottaa käyttöön, tulee se alustaa Manifest-tiedostossa. Tämä tapahtuu lisäämällä activity-elementti application-elementin alle. Elementillä activity on useita erilaisia attribuutteja, mutta näistä ainoastaan android:name on pakollinen. Se kertoo määriteltävän aktiviteetin luokan nimen. Muita attribuutteja voidaan käyttää määrittelemään esimerkiksi käyttäjälle näkyvän otsikko (label), kuvake (icon) tai teema (theme). Yksi sovelluksen aktiviteeteista määritellään main activityksi, joka avataan sovelluksen käynnistyksen yhteydessä. Oletuksena main activityksi on määritelty automaattisesti luotu luokka MainActivity. Kuvassa (3) on esitetty manifest-tiedoston rakenne (Google Inc. 2015a.)

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.karoliina.kanjiapuri_v2" >

    <uses-sdk android:minSdkVersion="20"/>

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/OmaTeema" >
        <activity
            android:name=".MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <activity
            android:name=".TestiAktiviteetti"
            android:label="Aktiviteetin otsikko">
        </activity>

        </application>
```

Kuva 3. Manifest-tiedoston rakenne

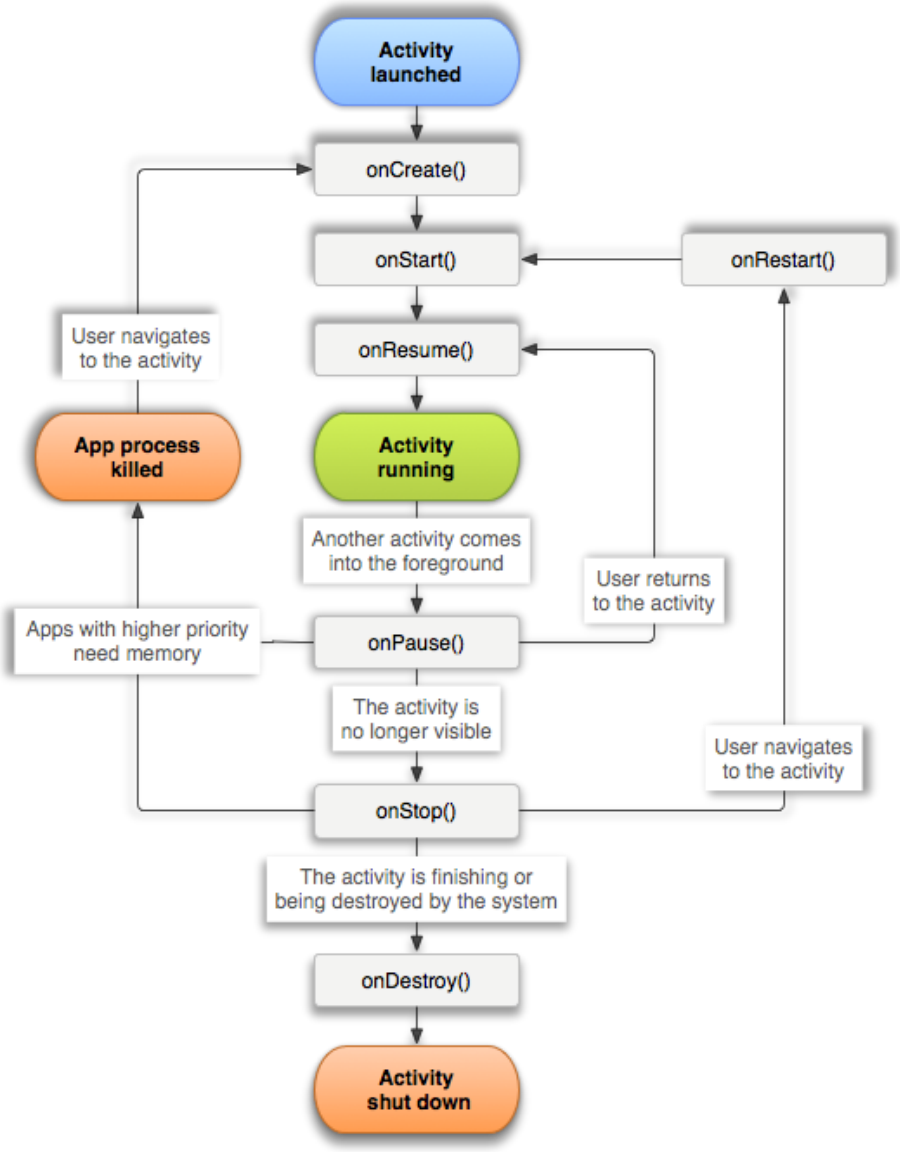
3.1.1 Aktiviteetin elinkaari

Aktiviteetilla on erilaisia tiloja, joissa se voi olla. Se voi olla aktiivinen, keskeytetty, pysäytetty tai inaktiivinen. Näiden tilojen välillä siirtyminen tapahtuu aktiviteetin käytössä olevien metodien avulla. Näitä metodeja ovat onCreate, onStart, onResume, onRestart, onPause, onStop, ja onDestroy (Kuvio 1). (Google Inc. 2015a.)

Aktiviteetin elinkaari sijoittuu kokonaisuudessaan aktiviteetin käynnistävän onCreate-metodin ja aktiviteetin sulkevan onDestroy-metodin välille. Avaavassa metodissa suoritetaan tarvittavat aktiviteetin alustustoimenpiteet. Sulkevassa metodissa aktiviteetin tulisi tallentaa tarvittavat tiedot, sekä vapauttaa kaikki yhteydessä olevat resurssit, kuten tietokannat (Harju 2013, 43–44.)

Kun aktiviteetti asetetaan käyttäjälle näkyväksi, kutsutaan sen onStart-metodia. Metodissa määritellään aktiviteetin käyttämät resurssit, kuten tietokantayhteydet. Kun seuraava aktiviteetti käynnistetään, lähetetään sitä ennen auki olleelle aktiviteetille tieto siitä, että se on pysäytetty. Kun aktiviteetti häviää näkyvistä, kutsutaan sen onStop-metodia, jossa vastaavasti vapautetaan kaikki resurssit. onStop-metodi ei kuitenkaan sulje aktiviteettia kokonaan, vaan siirtää sen pois niin kutsutun aktiviteettipinon päältä. Kun aktiviteetti käynnistetään taas uudelleen onRestart-metodilla, kutsutaan se takaisin aktiviteettipinon päällimmäiseksi. Näitä metodeja kutsutaan usein monesti aktiviteetin elinkaaren aikana (Google Inc. 2015a.)

Aktiviteetin pysyessä aktiviteettipinon päällimmäisenä, mutta laitteen mennessä lepotilaan tai aktiviteetin jäädessä esimerkiksi dialogi-ikkunan taakse, kutsutaan metodia onPause. Kun aktiviteetti palaa taas aktiiviseksi, kutsutaan onResume-metodia. Jos aktiviteetin ollessa inaktiivisessa tilassa, tarvitaan puhelimen muistia muuhun käyttöön, voidaan aktiviteetti pysäyttää (Harju 2013, 43–44.)



Kuvio 1. Aktiviteetin elinkaari (Google Inc. 2015)

3.1.2 Aktiviteetin ulkoasu

Aktiviteetin ulkoasu määritellään yleensä erillisessä xml-tiedostossa, joka tallennetaan res/layout-tiedostopolun alle. Tiedostossa määritellään aktiviteetin käyttämät käyttöliittymäelementit ja sijoitetaan ne halutuille paikoille. Elementtejä voidaan määritellä myös suoraan lähdekoodissa, mutta on kuitenkin suositeltavaa tehdä se xml-tiedostossa. Tällöin voidaan helpommin luoda sopivat käyttöliittymät erilaisille näytöille ja eri näytön orientaatioille. Lähdekoodin kautta voidaan kuitenkin muokata elementtien

tilaa tai esimerkiksi näytettävää tekstiä ohjelman ajon aikaisesti. Jotta aktiviteetti saadaan näyttämään haluttu ulkoasu, tulee toivottu layout-tiedosto määrittää aktiviteetin onCreate-metodissa. Tämä tapahtuu käyttämällä metodia setContentView(R.layout.tiedostonNimi). (Harju 2013, 53–55.)

Ulkoasulle määritellään xml-tiedoston alussa juurielementti, jonka sisään se luodaan. Kuvassa (4) juurielementiksi on määritelty LinearLayout. Alussa määritellään myös sekä ulkoasun korkeus että leveys, kuten myös se, onko ulkoasu tarkoitettu näytön horisontaaliselle vai vertikaaliselle asettelulle. Näiden määrittelyjen jälkeen voidaan juurielementin sisään luoda komponentteja, kuten tekstiä ja painikkeita. Esimerkiksi painike luodaan lisäämällä juurielementin alle *Button*-elementti. Elementin sisään kirjoitetaan vaaditut attribuutit: android:layout_width, joka määrittää komponentin leveyden, sekä android:layout_height, joka määrittää sen korkeuden. Näissä määrittelyissä kannattaa käyttää yksikköinä tiheydestä riippumattomia pikseleitä (density-independent pixel eli dp*)*. Koko voidaan myös määrittää äitielementin kokoiseksi. Jotta komponenttia voidaan kutsua lähdekoodin puolella, täytyy sille määrittää myös yksilöivä id attribuutilla android:id. Kuvassa (4) on luotu ulkoasu, joka sisältää tekstiä ja painikkeen. (Harju 2013, 53–55.)

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/teksti"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Paina nappia"
        />

    <Button
        android:id="@+id/ekanappi"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="nappi">
    </Button>


</LinearLayout>
```

Kuva 4. Ulkoasutiedosto, jossa on määritelty tekstiä ja painike

Aktiviteetissa komponentit saadaan otettua käyttöön tallentamalla ne niiden elementin mukaisiin muuttujiin. Esimerkiksi nappi määritettäisiin käyttöön komennolla: Button omaNappi = (Button) findViewById(R.id.napinId);. (Google Inc. 2015d.)

3.1.3 Aktiviteetin käynnistäminen

Aktiviteetti käynnistetään Intent-luokan olion avulla. Oliolle voidaan määritellä käynnistettäväksi joko suoraan tietty aktiviteetti, tai vaihtoehtoisesti sille voidaan kuvailla mitä käynnistettävän aktiviteetin haluttaisiin tekevän. Tällöin järjestelmä valitsee sopivan aktiviteetin, joka voi olla myös kokonaan toisesta sovelluksesta. Itse siirtyminen tapahtuu käyttämällä olion metodia startActivity (kuva 5). (Google Inc. 2015a.)

```
final Intent OmaIntent = new Intent(this,TestiAktiviteetti.class);

startActivity(OmaIntent);
```

Kuva 5. Aktiviteetin käynnistäminen

Intent-olion mukana voidaan myös siirtää pieniä määriä tietoa, esimerkiksi merkkijonoja ja taulukoita, käynnistettävälle aktiviteetille (kuva 6). Tiedon lisääminen olioon tapahtuu metodilla putExtra. Vastaanottavassa aktiviteetissa haetaan olion mukana siirretyt tiedot ja tallennetaan ne muuttujiin (kuva 7). (Google Inc. 2015a.)

```
int siirrettavaLuku = 10;

OmaIntent.putExtra("kuvaavaNimi", siirrettavaLuku);
```

Kuva 6. Siirretään Intent-olion mukana luku avattavalle aktiviteetille

```
int SiirrettyLuku = getIntent().getExtras().getInt("kuvaavaNimi");
```

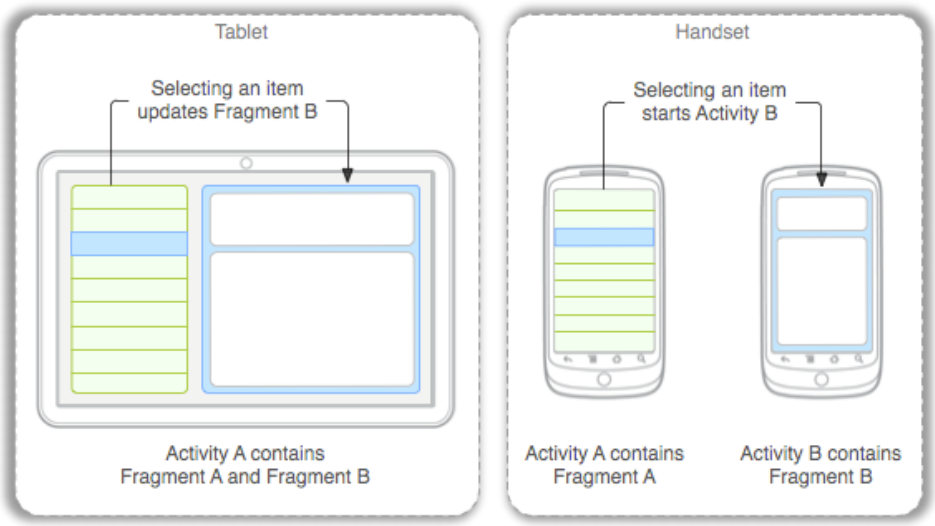
Kuva 7. Tallennetaan olion mukana siirretty luku muuttujaan

3.2 Fragmentit

Fragmentit ovat aktiviteetin sisällä käytettäviä moduuleja, joilla on oma elinkaarensa. Tämä elinkaari on kuitenkin jossain määrin riippuvainen fragmentin toteuttavan aktiviteetin tilasta. Esimerkiksi kun isäntä-aktiviteetti pysäytetään, käy samoin myös fragmentille. Aktiviteetin ollessa käynnissä, voi fragmentin tilaa kuitenkin muutella haluamallaan tavalla. (Google Inc. 2015c.)

Fragmentit lisättiin Androidiin ensisijaisesti lisäämään suurille näytöille suunniteltujen ulkoasujen dynaamisuutta. Kun tilaa on enemmän, voidaan samassa näkymässä näyttää useita fragmentteja, jotka pienemmällä näytöllä vaatisivat omat aktiviteettinsa. Esimerkiksi jos valitsisimme listasta objektin, voitaisiin sen tiedot isommalla ruudulla näyttää toisessa fragmentissa.

Pienemmällä näytöllä jouduttaisiin ne sen sijaan näyttämään omassa aktiviteetissaan (kuva 8). Lisäksi fragmenttia voidaan käyttää suoraan useissa eri aktiviteeteissa, jolloin sitä ei aina tarvitse luoda erikseen. (Google Inc. 2015c.)



Kuva 8. Fragmentit isommalla ja pienemmällä näytöllä (Google Inc. 2015c)

Fragmentti luodaan toteuttamalla Fragment-luokka, ja se käyttää lähes samoja metodeja, kuin aktiviteettikin. Fragmentin tulee kuitenkin näiden lisäksi kutsua sen onCreateView-metodia, kun se näytetään ruudulla ensimmäistä kertaa (Harju 2013, 140.)

3.3 Resurssitiedostot

Kuvat, merkkijonot ynnä muut tiedot, joita ei muuteta sovelluksen ajon aikana, tulee sijoittaa resurssitiedostoihin. Tämä parantaa sovelluksen ylläpidettävyyttä, sillä tällöin esimerkiksi merkkijonoja ei tarvitse erikseen muuttaa lähdekoodista. Resurssitiedostojen avulla voimme myös helposti optimoida sovellusta erilaisille laitteille sekä eri kielille. Kaikki resurssitiedostot sijoitetaan res/-hakemistopolun alle, mistä löytyy omat kansiot erilaisille resursseille (taulukko 1). (Harju 2013, 119.)

Taulukko 1. Kansiot, joihin resurssitiedot sijoitetaan (Harju 2013, 119–120.)

Hakemisto	**Sisältö**
animator	Hakemisto property animaatioille.
anim	Hakemisto tween-animaatioille.
color	Hakemisto tila–väri -määritystiedostoille.
drawable	Hakemisto bittikarttatiedostoille ja kuviksi muunneltaville XML-tiedostoille.
layout	Hakemisto asettelujen XML-määritystiedostoille.

menu	Hakemisto valikkojen XML-määritystiedostoille.
raw	Hakemisto resurssitiedostoille jotka eivät muotonsa tai sisältönsä puolesta sijoitu muualle *res*-hakemistorakenteeseen.
values	Hakemisto resurssitiedostoille, joissa määritetään sovelluksessa tarvittavat merkkijonovakiot.
xml	Hakemisto muille XML-määritystiedostoille kuten *searchable*, jonka avulla sovelluksessa voidaan ottaa käyttään Androidin pikahakutoiminto.

Resurssitiedostot tallennetaan yleensä kukin omaan tiedostoonsa, jolloin niihin voidaan koodissa viitata tiedostonimellä. Poikkeuksena tästä ovat kuitenkin merkkijonot, jotka määritellään kaikki strings.xml-tiedostossa, joka tallennetaan res/values-hakemiston alle. Jokaiselle merkkijonolle määritetään tiedostossa oma nimensä, jolla siihen viitataan (kuva 9). (Google Inc. 2015g.)

```
<resources>
    <string name="app_name">Testisovellus</string>
    <string name="harj1">Ensimmäinen teksti</string>
    <string name="harj2">Toinen teksti</string>

</resources>
```

Kuva 9. Strings.xml-tiedoston rakenne

Resursseihin viitataan hieman erilailla ohjelmakoodin ja xml-tiedoston puolella. Esimerkiksi kuvaan viitattaisiin ohjelmakoodissa polulla R.drawable.kuvanNimi, ja xml-tiedostossa sen sijaan polulla @drawable/kuvanNimi. (Harju 2013, 120.)

Sovellukselle voidaan myös määritellä vaihtoehtoisia resurssitiedostoja, esimerkiksi eri näyttökokoja, resoluutioita ja kieliasetuksia varten. Sovellusta käynnistettäessä Android-järjestelmä määrittää käyttöön sopivimmat resurssitiedostot laitteen tietojen mukaan. Jos sopivaa ei löydy, määritetään käytettäväksi oletusresurssi. (Google Inc. 2015g.)

Erityisesti kuville olisi hyvä määrittää vaihtoehtoiset resurssit. Käytännössä tämä tarkoittaa sitä, että kuvasta tehdään eri resoluutioille tarkoitettuja versioita ja tallennetaan ne eri näyttötarkkuuksia varten varattuihin kansioihin. Näitä kansioita ovat: res/drawable-xhdpi, res/drawable-hdpi, res/drawable-mdpi, sekä res/drawable-ldpi. Myös erikokoisille näytöille (small, normal, large, xlarge) voi olla aiheellista luoda omat ulkoasuresurssit. Resurssi voidaan myös luoda niin, että määritellään pienin näytön leveys, jolla resurssitiedostoa käytetään (esim. sw320dp). Sovelluksen tukemat näyttökoot

tulee kirjata manifest-tiedostoon supports-screens-elementin alle. (Google Inc. 2015g.)

Kieliasetuksia varten luodaan res/values-hakemistopolun alle vaihtoehtoisia kansioita. Jos esimerkiksi haluttaisiin määrittää kielivaihtoehdoiksi suomi ja englanti, luotaisiin seuraavat kansiot: res/values ja res/values-en. Näistä käytetään oletusarvoisesti res/values-kansion alta löytyvää strings.xml-tiedostoa. Jos taas laitteen kieleksi olisi määritetty englanti, käytettäisiin res/values-en-kansion alle tallennettua tiedostoa. Lisäksi jos haluaisimme jättää kääntämättä jonkin merkkijonon, kuten esimerkiksi sovelluksen nimen, voidaan se jättää pois englanninkielisestä tiedostosta. Tällöin järjestelmä hakee kyseisen merkkijonon oletustiedostosta, vaikka kieli olisikin englanti. (Google Inc. 2015e.)

Myös näytön orientaatiolle tehdään vaihtoehtoiset kansiot, jolloin oletusorientaatiolle tarkoitettu tiedosto tallennetaan suoraan res/layout-kansion alle. Näytön horisontaaliselle asettelulle tarkoitettu ulkoasu tallennetaan res/layout-land-kansion alle. Pystysuuntaisen ulkoasun vastaava kansio taas on res/layout-port. Jos kuitenkin oletustiedosto on tehty pystysuuntaiselle näytölle, ei tätä kansiota ole tarpeen käyttää. On hyvä muistaa, että näytön orientaation vaihtuessa aktiviteetti käynnistetään uudelleen, joten voi olla tarpeellista tallentaa sen tila. (Google Inc. 2015g.)

3.4 Valikot

Androidille on yleisesti käytössä kolmentyyppisiä valikoita: Options-, Contextual- ja Popup-valikot. Näistä ensiksi mainittu sijaitsee yleensä yläpalkissa, ja on ensisijainen valikko. Palkkiin tulisikin sijoittaa kaikki valinnat, joihin olisi hyvä päästä käsiksi kaikkialta sovelluksesta. Kontekstuaalinen valikko puolestaan avautuu näytölle, kun käyttäjä painaa jotakin elementtiä pitkään. Tällaiseen valikkoon on hyvä sijoittaa sen avaavaa elementtiä koskevat asetukset. Popup-valikko taas avautuu kun käyttäjä suorittaa jonkin tietyn toiminnon, esimerkiksi painaa nappia. (Google Inc. 2015f.)

Valikot alustetaan omissa xml-tiedostoissaan, jotka tallennetaan res/menus-hakemistopolun alle. Nämä xml-tiedostot koostuvat kolmesta elementistä: menu, item sekä group. Näistä kolmesta ensiksi mainittu on tiedoston juurielementti, jonka sisälle itse valikko rakennetaan. Item-elementillä puolestaan määritellään yksi valikon osa, ja sen sisälle voidaan jopa upottaa toinen menu-elementti. Group-elementti ei ole pakollinen, mutta sen avulla voidaan kategorisoida valikon osia niin, että ne jakavat asetuksia esimerkiksi näkyvyydestä. (Google Inc. 2015f.)

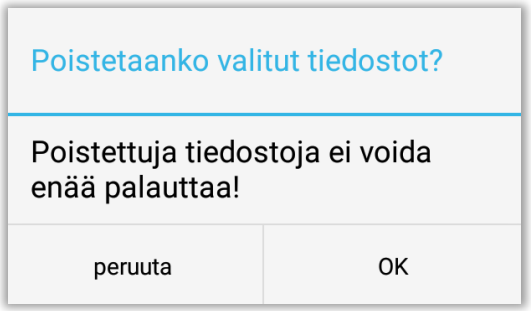
Yksittäisen valikon kohteen ulkonäköä ja käyttäytymistä voidaan määritellä erilaisten attribuuttien avulla. Ensin kohteelle pitää kuitenkin määrittää yksilöivä nimi, jolla siihen voidaan viitata koodin puolella. Tämä tapahtuu attribuutin android:id avulla. Muita käytettäviä attribuutteja ovat android:icon (kuvakkeen määritys), android:title (nimi, jolla vaihtoehto näkyy valikossa), sekä android:showAsAction (kohde on yläpalkissa näkyvissä aina, ei koskaan tai silloin jos on tilaa). (Harju 2013, 66–67.)

Options-valikko asetetaan aktiviteettiin metodilla onCreateOptionsMenu, jonka sisällä käytetään getMenuInflater-metodia. Metodille annetaan parametrina valikon ulkoasutiedoston polku. valikon kohteen avaamiseksi luodaan onOptionsItemSelected-metodin sisällä halutusta valikosta uusi DialogFragment, ja kutsutaan sen show-metodia. Kontekstuaalisella valikolla vastaavat metodit ovat onCreateContextMenu ja onContextItemSelected. (Google Inc. 2015f.)

Popup-valikko sen sijaan luodaan hieman erilailla. Jos haluttaisiin, että popup-valikko avataan silloin, kun painetaan nappia, tulee xml-tiedostossa napille lisätä attribuutti android:onClick=”showPopup”. Aktiviteettiin luodaan metodi showPopup, joka toteuttaa show-metodin. (Google Inc. 2015f.)

3.5 Dialogit

Dialogi on ikkuna, joka aukaistaan aktiviteetin päälle. Sen kautta käyttäjä voi tehdä valintoja, tai vaihtoehtoisesti syöttää tietoja. Dialogeja on hyvä käyttää esimerkiksi tilanteissa, joissa käyttäjältä tarvitaan varmistus ennen kuin viedään jokin toiminto loppuun (kuva 10). (Google Inc. 2015b.)



Kuva 10. Dialogi, jossa käyttäjältä kysytään varmennusta, ennen toiminnon suorittamista

Dialogi on suositeltavaa luoda omaan luokkaansa ja sen toteuttamiseen tulee käyttää DialogFrgment-luokkaa (kuva 11). Luokan tulee toteuttaa metodi onCreateDialog. Metodin sisällä luodaan Dialog.Builder-objekti, joka viittaa aktiviteettiin. Objektin avulla voidaan dialogille lisätä otsikko, tekstiä sekä napit. Lopuksi palautetaan dialogi-objekti. Erilaisia dialogeja saadaan tehtyä helposti käyttämällä Dialog-luokan aliluokkia, kuten AlertDialog, DatePickerDialog ja TimePickerDialog. Dialogille voidaan myös luoda oma ulkoasu. (Google Inc. 2015b.)

```
public class TestiAktiviteetti extends DialogFragment {

    public Dialog onCreateDialog(Bundle savedInstanceState) {
        // Use the Builder class for convenient dialog construction
        AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());

        builder.setTitle("Poistetaanko valitut tiedostot?");
        builder.setMessage("Poistettuja tiedostoja ei voida enää palauttaa!")
                .setPositiveButton(R.string.ok, new DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog, int id) {
                        // FIRE ZE MISSILES!
                    }
                })
        .setNegativeButton("peruuta", new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int id) {

                }
            });
        // Create the AlertDialog object and return it
        return builder.create();
    }
}
```

Kuva 11. Dialogin toteuttavan luokan rakenne

Dialogin näyttämiseksi luodaan sen avaavaan aktiviteettiin instanssi luokasta, johon oma dialogi luotiin, ja kutsutaan sen Show-metodia. Dialogi sulkeutuu automaattisesti, kun käyttäjä painaa siinä olevaa nappia tai valitsee listasta vaihtoehdon. Vaihtoehtoisesti dialogin voi sulkea myös komennolla dismiss. (Harju 2013, 111–112.)

4 SQLITE-TIETOKANTA

SQLite on täysin vapaasti yleiseen käyttöön asetettu, suosittu tietokantajärjestelmä. Toisin kuin monet tietokantajärjestelmät, ei SQLite tarvitse toimiakseen erillistä palvelinta, vaan on tiedostopohjainen. Tämä tarkoittaa sitä, että koko tietokanta sijaitsee yhden tiedoston sisällä. Koska SQLite vie vain vähän tilaa, eikä vaadi paljoa muistia suorittamiseen, on se erityisen suosittu mobiililaitteissa. SQLitea ei myöskään tarvitse erikseen asentaa mitenkään, riittää että ladataan lähdekoodi ja linkitetään se käytettävään sovellukseen. (SQLite.org 2015a.) Monista suosituimmista mobiilikäyttöjärjestelmistä myös löytyy SQLite jo valmiina. Esimerkiksi Android-käyttöjärjestelmässä on valmiina täysi tuki SQLite-tietokantajärjestelmälle. (Google Inc. 2015i.)

SQLite lukitsee koko tietokannan muutosten ajaksi, eikä vain muutoksen kohteena olevaa taulua, kuten monet muut tietokantajärjestelmät. Jos lukitus ei toimi, voi aiheutua ongelmia, mikäli moni käyttäjä käyttää tietokantaa samanaikaisesti. Pahimmassa tapauksessa moni käyttäjä voi yrittää muokata tietokantaa samanaikaisesti, jolloin tietokanta voi korruptoitua, eli hajota. Kun samanaikaisia käyttäjiä on monta, onkin viisaampaa käyttää asiakas/palvelin (eng. client/server)-tyyppistä tietokantaa, kuten esimerkiksi MySQL-tietokantaa. SQLite ei myöskään ole paras vaihtoehto, jos tietokanta on hyvin laaja. (SQLite.org 2015b.)

4.1 Tietokannan luominen

Tietokannan luomista varten on suositeltua luoda erillinen luokka, joka toteuttaa SQLiteOpenHelper-luokan. Tämä luokka toteuttaa onCreate- ja OnUpgrade-metodit. Lisäksi siinä määritetään tietokannan nimi ja versionumero sekä tietokannan taulut ja niiden kolumnit. Luotu tietokanta on sovelluksen jokaisen luokan käytettävissä, muttei näy sen ulkopuolelle. (Google Inc. 2015h.)

Metodin onCreate sisällä luodaan tietokantaan taulut SQL-lauseen avulla. Lauseessa määritellään taulun nimi ja sarakkeet, sekä niiden tietotyypit. Käytettäviä tyyppejä ovat null, integer, text, real, ja blob. Metodissa onUpgrade puolestaan tarkistetaan, onko kannassa jo valmiina taulu. Jos taulu löytyy, poistetaan se ja kutsutaan sitten onCreate-metodia. (Vogella.com 2015.) Tietokantaa päivittäessä on hyvä muistaa nostaa tietokannan versionumeroa, jotta muutokset tulevat voimaan.

Jotta tietokantaa olisi vaivatonta käyttää, luodaan sille metodit tiedon lisäämistä, lukemista, päivittämistä ja poistamista varten. Näitä metodeja kutsutaan yleisesti CRUD-metodeiksi (create, read, update, delete). Lisäksi on suositeltavaa kirjoittaa erillinen luokka, jossa tietokannan saraketta voidaan tarkastella objektina. Luokassa kirjoitetaan kolumneille set- ja get-metodit. (Androidhive.info 2015.)

4.2 Tietokantaan kirjoittaminen

Jotta tietokantaan voidaan kirjoittaa, tulee kutsua metodia getWritableDatabase. Tämän jälkeen luodaan uusi instanssi luokasta ContentValues, jonka avulla kolumneihin tallennetaan arvot. Tähän käytetään values-objektin put-metodia. Kirjoittamisessa voidaan käyttää apuna erilliseen tietokantaa kuvaavaan luokkaan kirjoitettuja get-metodeja. Kun tiedot on saatu lisättyä values-objektiin, kirjoitetaan tiedot itse kantaan käyttämällä SQL-lausetta insert. (Androidhive.info 2015.)

```
public void addKanji(Kanji kanji) {
    //for logging
    Log.d("addBook", kanji.toString());

    // 1. get reference to writable DB
    SQLiteDatabase db = this.getWritableDatabase();

    // 2. create ContentValues to add key "column"/value
    ContentValues values = new ContentValues();
    values.put(KEY_KANJI, kanji.getKanji()); // get kanji
    values.put(KEY_KAANNOS, kanji.getKaannos()); // get käännös
    values.put(KEY_KUNYOMI, kanji.getKunyomi()); // get kunyomi
    values.put(KEY_ONYOMI, kanji.getOnyomi()); // get onyomi
    values.put(KEY_KANAKUN, kanji.getKanaKun()); //get kunyomi in kana
    values.put(KEY_KANAON, kanji.getKanaOn()); //get onyomi in kana

    // 3. insert
    db.insert(TABLE_KANJIT1, // table
            null, //nullColumnHack
            values); // key/value -> keys = column names/ values = column values

    // 4. close
    db.close();
}
```

Kuva 12. Luodaan metodi, jonka avulla tietokantaan voidaan lisätä uusi kanji

Kantaan voidaan lisätä tietoa kirjoittamista varten luodun metodin avulla (kuva 13). Metodille luodaan uusi objekti, jolle annetaan parametreina sarakkeiden arvot. Jos tietokantaan on määritetty pääavaimeksi sarake ID, jolle on lisätty määre autoincrement, ei sitä tule antaa parametrina metodille. (Androidhive.info 2015.)

```
db.addKanji(new Kanji("一", "yksi", "hito-tsu", "ichi, itsu", "ひと・つ", "イチ, イツ"));
db.addKanji(new Kanji("二", "kaksi", "futa-tsu", "ni, ji", "ふた・つ", "ニ, ジ"));
db.addKanji(new Kanji("三", "kolme", "mit-tsu", "san", "みっ・つ", "サン"));
```

Kuva 13. Kanjien lisääminen tietokantaan

4.3 Tietokannasta lukeminen

Tietokannasta lukemista varten tulee kutsua metodia getReadableDatabase. Metodi palauttaa objektin, joka kuvaa käytettävää tietokantaa. Näin saadaan käyttöön muun muassa query-metodi, jonka avulla voidaan suorittaa SQL-kyselyitä. (Google Inc. 2015h.)

Jokainen tehty SLQ-kysely palauttaa kursorin, jonka avulla luetaan rivejä tietokannasta. Haettavaa saraketta ei laiteta suoraan SQL-lauseen SELECT arvoksi, vaan kyselyllä haetaan kaikki sarakkeet, joista kursorin avulla erotetaan halutut. Kun tiedot on haettu, voidaan kursoria liikuttaa komennoilla MoveToFirst, joka siirtää kursorin ensimmäiselle riville ja MoveToNext, jonka avulla päästään seuraavalle riville. kursori tarjoaa myös get-metodin, jonka avulla riveiltä voidaan lukea halutut sarakkeet. Esimerkiksi String-muotoinen muuttuja saataisiin komennolla getString(columnIndex), jossa columnIndex viittaa siihen, monennestako sarakkeesta tietoa halutaan lukea. (Google Inc. 2015h.)

```
public String getkanjiKaannos(int id) {


    // 1. get reference to readable DB
    SQLiteDatabase db = this.getReadableDatabase();

    // 2. build query
    Cursor cursor =
            db.query(TABLE_KANJIT1, // a. table
                    COLUMNS, // b. column names
                    " id = ?", // c. selections
                    new String[]{String.valueOf(id)}, // d. selections args
                    null, // e. group by
                    null, // f. having
                    null, // g. order by
                    null); // h. limit

    // 3. if we got results get the first one
    if (cursor.moveToFirst()) {
        do {
            // 4. build book object
            kaannos = cursor.getString(2);
        } while (cursor.moveToNext());
    }
    //log
    Log.d("getBooklol(" + id + ")", kaannos);

    db.close();
    // 5. return book
    return kaannos;
}
```

Kuva 14. Luetaan tietokannasta parametrina annetun ID:n mukaisen kaannos-sarakkeen arvo

Sarakkeen numeron sijasta voidaan käyttää myös cursor.getColumnIndex-metodia, jonka avulla voidaan hakea sarake sen nimen perusteella (kuva 15). (Google Inc. 2015h.)

```
do {
    // 4. build book object
    kaannos = cursor.getString(cursor.getColumnIndex("kaannos"));
} while (cursor.moveToNext());
```

Kuva 15. Tietokannasta voidaan hakea myös sarakkeen nimellä

Kun tiedot on saatu luettua, on hyvä muistaa myös sulkea kursori komennolla Cursor.close, sekä itse tietokantayhteys, komennolla yhteydeNimi.close. (Androidhive.info 2015.)

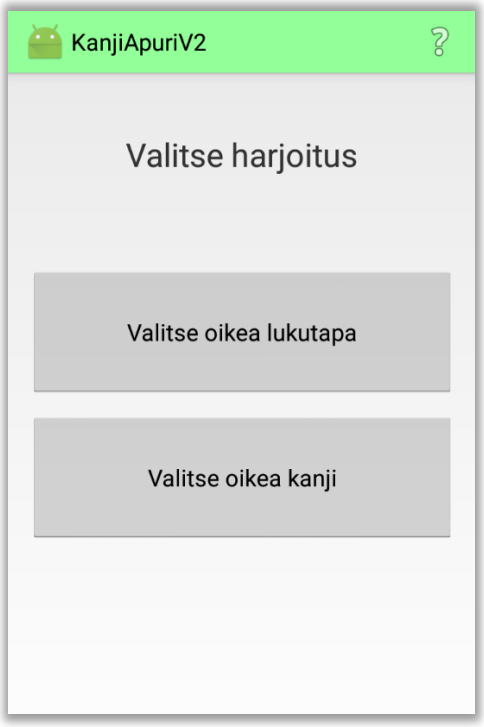
5 SOVELLUKSEN TOTEUTUS

Sovelluksen ohjelmointi aloitettiin harjoituksen valinta -näkymästä, joka on sovelluksen päänäkymä. Harjoituksista toteutettiin ensimmäisenä valitse oikea lukutapa -tehtävä, jonka jälkeen jatkettiin valitse oikea kanji -tehtävän ohjelmointiin. Molemmat harjoitukset toteutettiin alkuun vain ensimmäisen vuoden kanjeilla (80 kappaletta). Kun kaikki saatiin toimimaan, lisättiin sovellukseen loputkin suunnitellut kanjit. Sovelluksen toteuttamiseen käytettiin Android Studiota ja testilaitteina käytettiin emulaattoria sekä fyysistä laitetta.

5.1 Harjoituksen valinta -näkymä

Sovelluksen rakentaminen oli kaikkein loogisinta aloittaa aloitusnäkymästä, joka olisi myös toiminnoiltaan aktiviteeteista yksinkertaisin. Näkymä tehtiin suoraan Android Studion valmiiksi luomaan luokkaan, MainActivityyn. Ensimmäisenä aktiviteetille tarvittiin kuitenkin ulkoasu, jota se tulisi käyttämään. Aktiviteetille löytyikin res/layout-kansion alta valmis resurssitiedosto, activity_main.xml, johon MainActivity viittasi setContentView-metodissaan.

Ulkoasun juurielementiksi valittiin RelativeLayout, sillä se oli taipuisampi kuin LinearLayout, joka ei esimerkiksi antanut asetella komponentteja vieretysten. Näkymään luotiin yksi rivillinen tekstiä sekä kaksi painiketta, joista voisi valita haluamansa harjoitteen. Tämän jälkeen lisättiin strings.xml-tiedostoon kolme uutta merkkijonoa, jotka asetettiin komponenttien teksteiksi. Lopuksi kaikki komponentit sijoitettiin horisontaalisesti näytön keskikohtaan (kuva 16).



Kuva 16. Aloitusnäkymä

Kun ulkoasun komponentit oli saatu paikoilleen, haluttiin vielä muuttaa yläpalkin väriä. Tämä ei kuitenkaan tapahtunutkaan ulkoasutiedoston kautta, vaan määritys piti tehdä res/values-hakemistopolun alta löytyvään styles.xml-tiedostoon. Tiedostoon luotiin uusi teema nimeltä MyActionBar, jonka isäntä-elementiksi määritettiin elementti @android:style/Widget.Holo.Light.ActionBar. Lopuksi asetettiin android:background-attribuutilla taustan väri halutuksi. Jotta teema saatiin käyttöön, tarvittiin käyttöön vielä toinen teema, OmaTeema, jonka isäntä-elementiksi määritettiin elementti @android:style/Theme.Holo.Light. Teeman actionBarStyle-attribuutilla määritettiin yläpalkin teemaksi ensiksi luotu MyActionBar.

Jotta nappeja painamalla päästäisin siirtymään tehtäviin, tuli niihin ohjelmoida tapahtumat. Ennen tätä ne täytyi kuitenkin tallentaa Button-muuttujiin, jotta niihin päästäisiin kiinni koodissa. Napit haettiin muuttujiin findViewById-metodin avulla, jossa niihin viitattiin ulkoasutiedostossa määritetyillä nimillä.

Siirtymistä varten luotiin kaksi uutta aktiviteettia: QuizActivity ja KanjiActivity, joihin tehtävät myöhemmin toteutettaisiin. Molempia aktiviteetteja varten tarvittiin uudet intent-oliot, joiden starActivity-metodit asetettiin nappien setOnClickListener-metodien sisään. Testatessa sovellus kuitenkin kaatui, kun nappia painettiin. Aktiviteetit oli unohdettu kirjata manifest-tiedostoon. Kun aktiviteetit lisättiin tiedostoon, saatiin siirtyminen toimimaan ongelmitta.

5.2 Tietokannan luominen

Tietokantaa varten luotiin uusi luokka, MySQLiteHelper, joka toteutti SQLiteOpenHelper-luokan. Luokassa määritettiin tietokannan versionumero ja tietokannan nimi. onCreate-, sekä onUpgrade-metodien lisäksi kirjoitettiin funktiot myös kanjin lisäämistä, hakemista, päivittämistä ja poistamista varten. Lisäksi kirjoitettiin metodit, joilla saattoi id:n avulla hakea vain yksittäisen sarakkeen, esimerkiksi käännöksen, tiedot. Lisäksi kannan käyttämistä helpottamaan kirjoitettiin erillinen luokka, Kanji, jonka avulla pystyttiin käsittelemään kannan sisältöä olion tapaan.

onCreate-metodissa luotiin tietokantaan uusi taulu, KANJIT1. Taulu sisälsi aluksi vain sarakkeet id, käännös, kunyomi ja onyomi, sillä kanji lisättiin resurssitiedostoihin kuvana. Myöhemmin myös itse kanji lisättiin kuitenkin tietokantaan tekstinä, sillä arveltiin, että sovelluksesta saataisiin näin tehtyä hieman kevyempi. Kanjit toimivat tietokannassa suoraan, kunhan teksti oli muodossa UTF-8. Tietokantaan lisääminen sijoitettiin Main-activityyn, ja tapahtui MySQLiteHelper-luokkaan kirjoitetulla lisaaKanji-metodilla.

Kanji-luokkaan kirjoitettiin rakentaja, joka sisälsi muuttujina tietokannan sisältämät sarakkeet. Näille muuttujille kirjoitettiin set-, ja get-metodit, joita käytettiin hyödyksi MySQLiteHelper-luokan CRUD-metodeissa. Lopuksi kirjoitettiin vielä metodi toString, joka kirjoittaisi kaikkien sarakkeiden arvot merkkijonoon.

Muille mahdollisille sovelluksen käyttämille tietokannoille ei välttämättä tarvittaisi kanjille ominaisia sarakkeita kun-yomi ja on-yomi, joten oli tarpeen luoda tietokannan metodeista sellaiset versiot, joissa sarakkeita olisi vain kolme. Jos vastausvaihtoehdot kuitenkin haluttaisiin lisätä suoraan kantaan, tarvittaisiin sarakkeita seitsemän. Tietokannasta tarvittaisiin myös versio sellaista tietovisaa varten, joka käyttäisi kanjin sijasta resurssitiedostoihin sijoitettavaa kuvaa.

5.3 Valitse oikea lukutapa -tehtävä

Valitse lukutapa -tehtävä kirjoitettiin aiemmin luotuun luokkaan QuizActivity ja sen toteutus aloitettiin ulkoasun luomisesta. Ulkoasua varten luotiin res/layout-kansion alle uusi resurssitiedosto, activity_quiz.xml. Ulkoasu koostui alueesta, johon kanji kirjoitettiin, käännöstekstistä sekä neljästä painikkeesta. Lisäksi näytön oikeaan alakulmaan lisättiin vielä ”kanjeja jäljellä”- ja ”vääriä vastauksia”-tekstit, sekä molempien perään tekstialueet niiden määrille. Kanjin alue koostui RelativeLayoutin alle luodusta LinearLayoutista, jonka taustaväri vaihdettiin harmaaseen, sekä valkopohjaisesta tekstialueesta. Lopputuloksena saatiin kanji näkymään valkoisella taustalla, jolla oli harmaa reunus. Näin kanji erottui paremmin sovelluksen taustasta (kuva 17).

Kuva 17. Valitse lukutapa -tehtävä

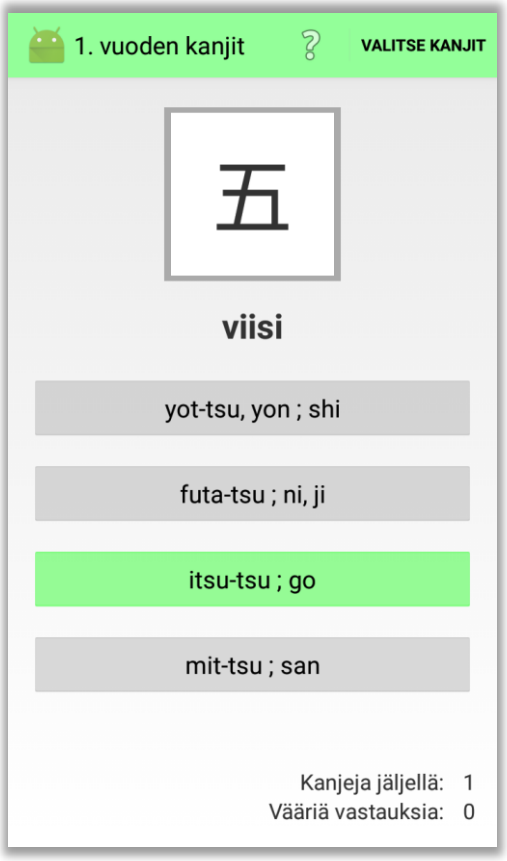
Koska haluttiin, että sovellus arpoisi kysyttävän kanjin satunnaisesti tietokannasta, arvottiin muuttujaan random satunnainen luku. Luvun perusteella haettiin tietokannasta id:n perusteella kanji, ja asetettiin se kanjille varatulle alueelle. Seuraavana haettiin käännös kanjin alla olevaan tekstikenttään. Myös nappeihin asetettavat vaihtoehdot haluttiin hakea satunnaisesti tietokannasta, joten niitä varten luotiin hashSet, johon arvottiin neljä lukua. Näiden lukujen perusteella haettiin tietokannasta lukutavat, jotka sijoitettiin nappien teksteiksi. Koska hashSet ei tallenna samaa lukua kahdesti, ei tarvinnut huolehtia siitä, että sama vaihtoehto tulisi useampaan painikkeeseen. Jos oikea vastaus ei sisältynyt arvottuihin lukuihin, poistettiin setin viimeinen luku ja lisättiin sen paikalle oikea vastaus.

Sovellus ei kuitenkaan toiminut täysin halutulla tavalla, vaan ei aika ajoin antanut oikeaa vastausta vaihtoehdoksi lainkaan. Koodia päätettiinkin muuttaa niin, että arvottiin settiin vain kolme satunnaista lukua, ja vasta tämän jälkeen lisättiin erikseen oikea vastaus settiin. Tämä kuitenkin poiki uusia ongelmia, sillä oikea vastaus oli nyt aina samalla paikalla, eli neljännessä napissa. Ongelma saatiin kuitenkin ratkaistua siirtämällä setin sisältö listaan, jonka järjestys oli sekoitettavissa.

Seuraavana lähdettiin työstämään vastauksen tarkistamista. Tätä varten tallennettiin aiemmin luodun random-muuttujan arvo uuteen muuttujaan, CorrectAnswer. Itse tarkistus sijoitettiin nappien onClick-metodeihin, ja toteutettiin vertaamalla napin tekstiä CorrectAnswer-muuttujan arvolla tietokannasta haettuihin lukutapoihin. Koska napin tekstissä lukutavat erotettiin puolipisteen avulla, piti se ottaa huomioon myös tarkistuksessa. Käyttäjälle kerrottiin tekstikuplan avulla, oliko vastaus oikein vai väärin (kuva 18).

Kuva 18. Tekstikuplalla kerrottiin menikö vastaus oikein

Myöhemmin tekstikuplan tilalle kuitenkin vaihdettiin värikoodit, eli nappi muuttui joko punaiseksi tai vihreäksi (kuva 19). Jotta käyttäjä ehtisi nähdä värikoodin, tuli sovellukseen lisätä pieni viive ennen seuraavaan kanjiin siirtymistä. Tähän käytettiin Handler-luokan metodia postDelayed. Viipeeksi asetettiin 400 millisekuntia. Värikoodien ongelmaksi voisi kuitenkin jatkossa osoittautua mahdolliset käyttäjät, joilla olisi jonkinlaista värisokeutta. tällaiset käyttäjät eivät välttämättä erottaisi menikö heidän vastauksensa oikein vai väärin. Jatkokehityksenä tarkistukseen voitaisiinkin lisätä jonkinlaista grafiikkaa, joka piirrettäisiin suoraan elementtiin, eikä tekstikupliin, jotka käyttäjän nopeasti kysymyksiin vastatessa alkaisivat kasaantumaan päällekkäin.



Kuva 19. Värikoodi, jolla käyttäjälle kerrottiin vastauksen menneen oikein

Kun vastaus saatiin oikein, haluttiin, ettei kyseistä kanjia enää kysyttäisi uudelleen sillä kierroksella. Tämä toteutettiin luomalla setti, oikeatSetti, johon vastauksen mennessä oikein, lisättiin CorrectAnswer-muuttujan arvo. Uutta lukua arvottaessa tarkistettiin while-lauseella, sisälsikö setti jo arvotun luvun. Jos näin oli, arvottiin luku uudelleen.

Kanjien ja väärien vastausten määriä varten luotiin uudet muuttujat: jaljella ja vaarin. jaljella-muuttujan arvoksi asetettiin kerrattavien kanjien määrä (esim. ensimmäisen vuoden kanjien kohdalla 80), ja vaarin-muuttujan arvoksi nolla. Jos vastaus oli oikein, vähennettiin jaljella-muuttujan arvosta yksi, kun taas vastauksen ollessa väärin, kasvatettiin vaarin-muuttujan arvoa yhdellä. Muuttujan jaljella arvo sijoitettiin sitten ”Kanjeja jäljellä” -tekstin perään ja vaarin-muuttujan arvo ”Vääriä vastauksia” -tekstin jälkeen.

Tehtävää muokattiin vielä niin, ettei se vastauksen ollessa väärin kysyisi samaa kanjia heti uudelleen. Toiminto toteutettiin uuden muuttujan, edellinen, avulla. Jos käyttäjän vastaus oli väärin, asetettiin muuttujan arvoksi CorrectAnswer-muuttujan arvo. Lopuksi tarkistettiin while-lauseella oliko edellinen-muuttujan arvo sama kuin CorrectAnswer-muuttujalla. Tämä toteutettiin samaan while-lauseeseen kuin yllämainittu kysyttyjen kanjien tarkistus. Kierroksen viimeisen kanjin kanssa ilmeni kuitenkin ongelma. Jos viimeinen vastaus meni pieleen, ei sovellus suostunut arpomaan samaa kanjia uudelleen. Koska kyseinen kanji oli kuitenkin ainoa, jonka pystyttiin

enää arpomaan, johti tämä sovelluksen kaatumiseen. Vastauksen tarkistukseen lisättiinkin ehtolause, joka tarkisti oliko kyseessä kierroksen viimeinen kanji. Jos näin oli, asetettiin edellinen-muuttujan arvoksi nolla.

Kun kaikki kanjit oli kysytty, siirryttiin yhteenvetonäkymään, josta pääsisi jatkamaan kertaamista tai palaamaan aloitusnäkymään. Jäljellä olevien kanjien määrä selvitettiin vertaamalla OikeatSetti-setin kokoa kysyttävien kanjien määrään. Yhteenvetonäkymää varten luotiin uusi aktiviteetti yhteenvetoActivity, ja siirtymä suoritettiin intent-luokan olion avulla. Olion mukana siirrettiin yhteenvetonäkymälle lista, jossa oli väärinmenneiden kanjien ID-arvot, ja int-tyyppinen muuttuja jossa oli niiden määrä.

Kaikki kanjit haluttiin alun perin lisätä samaan aktiviteettiin, jolloin se käynnistettäisiin uudelleen kanjeja vaihtaessa. Tätä ei kuitenkaan saatu toimimaan kunnolla, joten päätettiin luoda jokaisen vuoden kanjeille omat aktiviteetit.

Jotta sovelluksen sisällä navigointi olisi vaivatonta, haluttiin, että puhelimen taaksepäin-näppäintä painamalla päästäisiin takaisin aloitusruutuun. Taaksepäin siirtymiseen käytettiin Androidin valmista metodia onBackPressed. Metodin sisälle luotiin uusi intent-olio, joka viittasi MainActivityyn ja itse siirtymä toteutettiin sen startActivity-metodia käyttäen.

5.4 Valitse oikea kanji -tehtävä

Valitse kanji -tehtävä suunniteltiin toiminnoiltaan hyvin samankaltaiseksi, kuin valitse lukutapa -tehtävä. Ulkoasua piti kuitenkin muokata joissain määrin, ja sitä varten luotiin uusi ulkoasutiedosto activity_kanji.xml. Ruudun ylälaitaan asetettiin käännöstä varten tekstialue, jonka alle sijoitettiin tekstikentät lukutavoille. Nappien asemassa käytettiin valitse lukutapa -tehtävässäkin käytettyä aluetta, jolla oli harmaa reunus. Alueet sijoitettiin kaksi vierekkäin, ja toiset kaksi niiden alapuolelle. Alalaitaan lisättiin jälleen ”kanjeja jäljellä”- ja ”vääriä vastauksia”-tekstit, sekä molempien perään tekstialueet niiden määriä varten (Kuva 20).

Kuva 20. Valitse oikea kanji -tehtävä

Itse tehtävä kirjoitettiin luokkaan KanjiActivity, ja kuten valitse oikea lukutapa -tehtävässäkin, tehtiin jokaisen vuoden kanjeille omat aktiviteetit. Arvottujen lukujen perusteella haettiin jälleen tietokannasta kanjien tiedot paikoilleen. Kanjien ja väärien vastausten määrien laskemista varten luotiin jälleen muuttujat, joiden arvot sijoitettiin alalaitaan ”Kanjeja jäljellä” - ja ”Vääriä vastauksia” -tekstien perään. Myös vastauksen tarkistaminen tapahtui suurilta osin samalla tavalla, kuin toisessakin tehtävässä. Kun kanjille varattua tekstialuetta painettiin, tarkistettiin vastasiko siinä oleva teksti CorrectAnswer-muuttujan arvolla haettua kanjia. Jos näin oli, muutettiin alueen harmaa reunus vihreäksi (kuva 21). Vastaavasti vastauksen ollessa väärin, muutettiin väri punaiseksi.

Kuva 21. Värikoodi, jolla käyttäjälle kerrottiin vastauksen menneen oikein

Myös tähän tehtävään lisättiin arvottavien kanjien tarkistukset, sekä edellisen kanjin tarkistus, jottei sitä kysyttäisi heti uudelleen. Tehtävälle luotiin myös oma yhteenvetonäkymä, jonne siirryttiin, kun käyttäjä oli vastannut kaikkiin kierroksen kanjeihin oikein. Lisäksi toteutettiin myös siirtymä takaisin aloitusnäkymään kun puhelimen taaksepäin-näppäintä painettiin.

5.5 Yhteenvetonäkymä

Yhteenvetonäkymä luotiin aktiviteettiin yhteenvetoActivity, ja siihen luotiin aluksi vain kaksi nappia: ”Jatka kertaamista” ja ”Palaa päävalikkoon”. Aktiviteetin ulkoasu määritettiin tiedostossa yhteenveto_activity.xml. Myöhemmin aktiviteettiin haluttiin kuitenkin näkyville kierroksella väärin menneet kanjit. Näkymään luotiinkin uusi GridView, jonka avulla kanjit saataisiin näkymään ruudukkomaisesti aseteltuina. Myös ruudukkoa varten tehtiin ulkoasu, jossa jokaisen ruudukossa olevan kohteen ympärille piirrettiin ohut harmaa reunus. Ruudukon yläpuolelle lisättiin teksti ”Harjoittele vielä näitä kanjeja”. Jos vääriä vastauksia ei ollut, vaihdettiin tekstiksi ”Hienoa, vastasit kaikkiin oikein!”. Jotta ulkoasu olisi selkeämpi, lisättiin nappien ja ruudukkonäkymän väliin ohut harmaa viiva. Viivan yläpuolelle, oikeaan laitaan, lisättiin vielä ”vääriä vastauksia yhteensä” -teksti, sekä väärien vastausten lukumäärä (kuva 22).

Kuva 22. Yhteenvetonäkymä

Palaa päävalikkoon -napin onClick-metodiin lisättiin uusi intent-olio, joka viittasi MainActivityyn. Jotta nappia painamalla päästäisiin takaisin aloitusnäkymään, kirjoitettiin metodin sisälle vielä intent-olion onStart-metodi. ”Jatka kertaamista” -napin avulla taas haluttiin siirtyä takaisin tehtävään. Koska eri vuosien kanjit toteutettiin omiin aktiviteetteihinsa, tehtiin yhteenvetoaktiviteettejakin aluksi jokaiselle omansa. Tämä tuntui kuitenkin vaivalloiselta, joten lähdettiin etsimään parempaa ratkaisua. Jotta saatiin tarkistettua, mistä aktiviteetista näkymään siirryttiin, lisättiin tehtävän aktiviteettiin muuttuja aktiviteettiNro. Muuttuja sitten lähetettiin intent-olion mukana yhteenvetoaktiviteetille, jonne luotiin uusi intent-olio KanjiIntent. Jotta tiedettäisiin mihin aktiviteettiin tuli siirtyä, kirjoitettiin ehtolause, jossa tarkistettiin aktiviteettiNro-muuttujan arvo. Jos esimerkiksi arvo oli yksi, siirryttiin ensimmäisen vuoden kanjeihin, jonka aktiviteetissa kyseisen muuttujan arvoksi oli määritetty yksi. Valitse oikea kanji -, ja valitse oikea lukutapa -tehtäville päätettiin tehdä selkeyden vuoksi omat yhteenvetoaktiviteettinsa, mutta niiden yhdistäminen ei toki olisi ollut mahdoton ajatus.

Siirtymän mukana saatu tieto väärin menneiden vastausten lukumäärästä sijoitettiin vääriä vastauksia yhteensä -tekstin perään. Väärin menneiden kanjien lisääminen ruudukkoon puolestaan vaati hieman enemmän vaivannäköä. Kanjien lisäämistä varten luotiin uusi luokka MyAdapter, joka toteutti BaseAdapter-luokan. Luokkaan kirjoitettiin MyAdapter-rakentaja, jonka parametreiksi asetettiin konteksti, sekä String-, ja Integer-tyyppiset taulu-

kot. Nämä tallennettiin rakentajan sisällä omiin muuttujiinsa. Tämän jälkeen luotiin BaseAdapter-luokan vaatimat metodit getCount, getItem ja getItemId. Näitä metodeja ei kuitenkaan käytetty näkymän toteutuksessa. Viimeisenä luokkaan kirjoitettiin vaadittu metodi getView, joka palautti view-objektin. Metodissa asetettiin ruudukossa näkyväksi tekstiksi kanji. Yhteenvetoaktiviteetissa luotiin MyAdapter-luokasta uusi instanssi adapter. Adapterin parametreiksi annettiin lista väärinmenneiden kanjien ID-numeroista, ja lista näillä luvuilla haetuista kanjeista. luotu MyAdapter-objekti asetettiin GridView'n adapteriksi komennolla gridview.setAdapter(adapter).

Kanjia klikkaamalla haluttiin saada näkyviin sen tiedot. Tämä päätettiin toteuttaa käyttämällä dialogia, jolle luotiin uusi ulkoasu tiedostoon kanji_tiedot.xml. Ulkoasun yläreunaan lisättiin tekstikenttä käännöstä varten ja sille määriteltiin vihreä tausta, joka oli koko ruudun levyinen. Käännöksen alle sijoitettiin harmaareunuksinen alue kanjille, ja sen alle tekstikentät lukutapoja varten. Itse dialogia varten luotiin kokonaan uusi luokka, KanjiTiedotDialog, joka toteutti DialogFragment-luokan. Jotta dialogiin saatiin haettua tekstit tietokannasta, kirjoitettiin KanjiTiedotDialog-objektin palauttava metodi tiedot (kuva 23), jolle asetettiin parametreiksi kanji, käännös, sekä kun- ja on-yomit. Metodin sisälle luotiin KanjiTiedot-Dialog-objekti frag, sekä Bundle args, johon tallennetiin parametreina saadut arvot. Lopuksi lisättiin Bundlen sisältö metodissa luotuun objektiin ja palautettiin se.

```
public static KanjiTiedotDialog tiedot(String kaannos, String kanjiKuva,
                                       String kunyomi, String onyomi) {
    KanjiTiedotDialog frag = new KanjiTiedotDialog();
    Bundle args = new Bundle();
    args.putString("kaannos", kaannos);
    args.putString("kanjikuva", kanjiKuva);
    args.putString("kunyomi", kunyomi);
    args.putString("onyomi", onyomi);
    frag.setArguments(args);
    return frag;
}
```

Kuva 23. tiedot-metodin rakenne

onCreateDialog-metodissa tallennettiin Bundleen args lisätyt argumentit muuttujiin (kuva 24). Lopuksi asetettiin komponenttien teksteiksi näiden muuttujien arvot, ja lisättiin dialogiin "ok"-painike.

```
String kaannos = getArguments().getString("kaannos");
String kanjiKuva = getArguments().getString("kanjikuva");
String kunyomi = getArguments().getString("kunyomi");
String onyomi = getArguments().getString("onyomi");
```

Kuva 24. Bundlen *args* sisältö tallennettiin muuttujiin

Yhteenvetonäkymään lisättiin vielä gridview.setOnItemClickListener-metodi, jossa haettiin tietokannasta klikatun ruudukon kohteen tiedot. Nämä tiedot tallennettiin muuttujiin ja annettiin parametrina KanjiTiedotDialog-objektille. Lopuksi avattiin dialogi objektin show-metodilla (kuva 25).

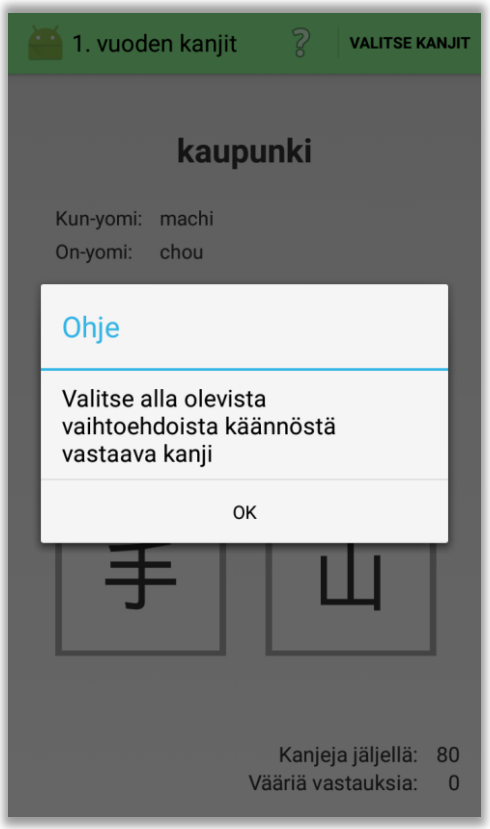
Kuva 25. Kanjin tiedot sisältävä dialogi

5.6 Valikot

Valikoita luotiin erilaisia kaksi kappaletta: menu_main ja menu_tehtava. Molemmat näistä olivat Options-valikoita, ja sijaitsivat yläpalkissa. Valikkoon menu_main lisättiin vain kohde ohje, joka määriteltiin näkymään aina. Sen kuvakkeeksi asetettiin Androidin käytettävissä oleva drawable/ic_menu_help. Tehtävä-valikkoon lisättiin ohjeen lisäksi kohde ”valitse kanjit”, joka asetettiin näkymään silloin, jos tilaa on. Kohteen otsikoksi asetettiin teksti ”valitse kanjit”.

5.6.1 Ohje-dialogit

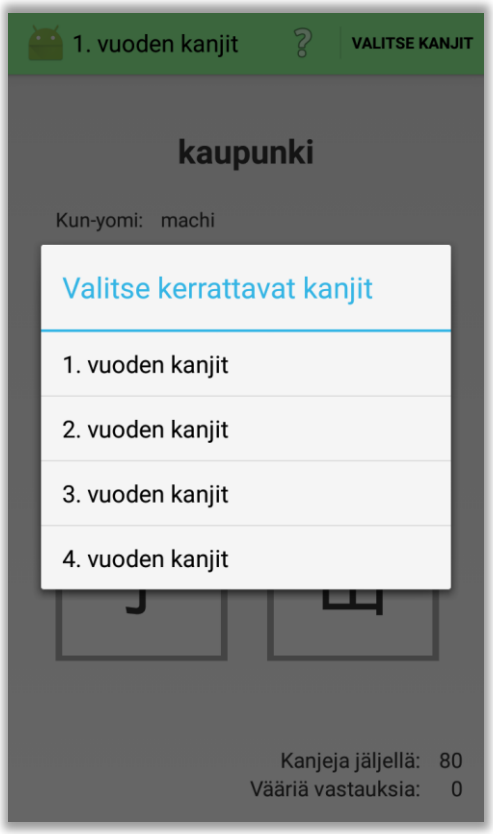
Ohjeita varten luotiin neljä dialogia: mainohje, quizohje, kanjiohje ja yhteenvetoohje. Jokaiselle dialogille lisättiin otsikko, tekstiä ja ”ok”-painike (kuva 26). Aloitusnäkymän ohjeeksi sijoitettiin mainohje, tehtävien ohjeiksi quizohje ja kanjiohje sekä yhteenvetonäkymän ohjeeksi yhteenvetoohje. Ohjeen sai näkyviin yläpalkista löytyvää kysymysmerkki-ikonia painamalla.



Kuva 26. Valitse oikea kanji -tehtävän ohje-dialogi

5.6.2 Valitse kanjit -dialogi

Kanjien valintaa varten luotiin listanäkymä-dialogi (kuva 27). Jotta listasta voitaisiin valita kohde, luotiin luokkaan dialogikäsittelijä, joka sisälsi metodin onDialogPositiveClick(String valintasi). onCreateDialog-metodissa luotiin taulukko, joka sisälsi listaan tulevien kohteiden otsikot muodossa ”n. vuoden kanjit”. Dialogille annettiin otsikko ja lisättiin taulukon sisältö sen kohteiksi. Metodin sisään luotiin vielä onClick metodi, jolle annettiin int-tyyppinen parametri valinta.



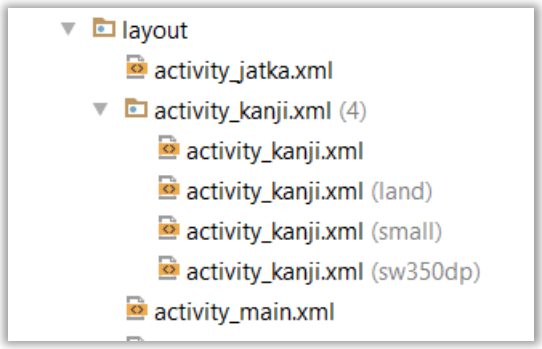
Kuva 27. Kanjien valinta -dialogi

onClick-metodin sisälle luotiin uusi muuttuja omaValinta, jonka arvoksi määritettiin valinta-parametrin arvon mukainen kohta taulukosta. Koska omaValinta-muuttujan arvoksi haluttiin lopuksi vain luku kohteen otsikon alusta, halkaistiin merkkijono pisteen kohdalta. Halkaistun merkkijonon ensimmäinen osa, eli haluttu luku, tallennettiin sitten omaValinta-muuttujan uudeksi arvoksi. Tämä arvo annettiin onDialogPositiveClickin parametriksi.

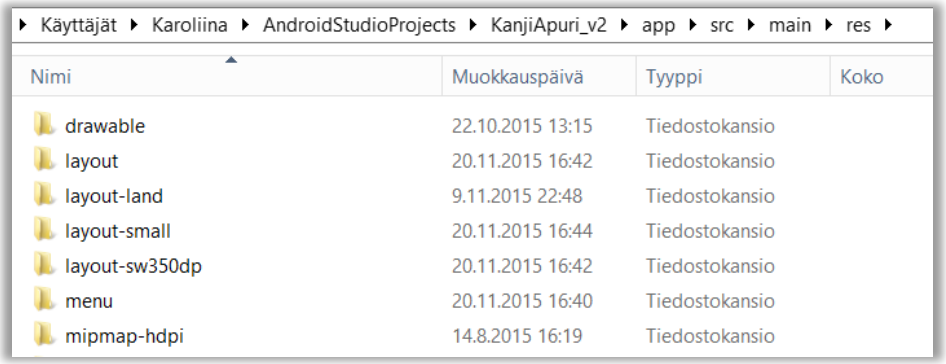
Tehtävä-luokassa toteutettiin ValitseKanjitDialog.KanjitDialogiKasittelija-rajapinta, ja toteutettiin sen metodi onDialogPositiveClick(String valintasi). Jos valinta oli arvoltaan yksi, siirryttiin ensimmäisen vuoden kanjeihin ja niin edelleen. Jos valinta osoitti jo käynnissä olevaan aktiviteettiin, näytettiin tekstikupla, jossa kerrottiin, että oltiin jo kertaamassa kyseisen vuoden kanjeja.

5.7 Näytön koko ja orientaatio

Sovellukselle luotiin ulkoasu myös näytön horisontaalista orientaatiota varten, sekä myös muutamalle eri näyttökoolle (kuva28). Valitse oikea kanji -tehtävää varten luotiin uusi ulkoasutiedosto samalla nimellä kuin aikaisemminkin, eli activity_kanji. Tiedosto kuitenkin tallennettiin /res-hakemistopolun alle (kuva 29) uuteen kansioon nimeltä layout-land. Tällöin Android osasi itse hakea käyttöön oikean orientaation ulkoasutiedoston.



Kuva 28. Valitse oikea kanji -tehtävälle luodut ulkoasut erikokoisille näytöille ja horisontaaliselle orientaatiolle



Kuva 29. Tiedostot asetettiin /res-hakemistopolun alle omiin kansioihinsa

Kun laitetta käännettiin, huomattiin kuitenkin, että aktiviteetti käynnistyi uudelleen. Jotta tietovisaa saataisiin jatkettua samasta kohtaa, otettiin käyttöön metodi onSavedInstance, jonka avulla tallennettiin orientaation vaihtuessa kaikki tarpeelliset tiedot. Tallentaminen kävi onSavedInstance-metodin put-metodia käyttäen. Esimerkiksi Integer-tyyppinen muuttuja saatiin tallennettua komennolla: savedInstanceState.putInt("kuvaavaNimi", muuttujanNimi); Aktiviteetin onCreate-metodissa tutkittiin oliko onSavedInstanceState tyhjä. Jos näin ei ollut, asetettiin taas tallennetut arvot haluttuihin muuttujiin. Esimerkiksi edellä mainittu numeromuuttuja asetettiin muuttujan arvoksi komennolla: muuttujanNimi = savedInstanceState.getInt("kuvaavaNimi"); Näin saatiin kaikki muu toimimaan, paitsi arvotut vastausvaihtoehdot, jotka eivät jostain syystä suostuneet tallentumaan oikein.

Myös eri näyttökokoja varten luotiin vaihtoehtoiset ulkoasutiedostot. Tiedostot luotiin pienelle näyttökoolle (small) sekä vähintään 350 pikseliä leveälle näytölle. Näitä tiedostoja varten luotiin /res-hakemistopolun alle uudet kansiot layout-small ja layout-sw350dp. Ulkoasutiedostoja muokattiin niin, että ne näyttivät hyviltä edellä mainitun kokoisilla näytöillä. Kun tiedostot oli tallennettu oikeisiin kansioihin, osasi käyttöjärjestelmä valita niistä käyttöön sopivimman. Koska käytössä oli vain yksi fyysinen laite, testattiin ulkoasutiedostojen toimivuutta erikokoisilla emulaattoreilla.

6 YHTEENVETO

Opinnäytetyön tavoitteena oli oppia luomaan mobiilisovelluksia Android-alustalle ja hyödyntämään niissä SQLite-tietokantaa. Tässä tavoitteessa onnistuttiin hyvin ja aikaan saatiinkin toimiva sovellus, joka sisälsi ne toiminnot, joita siihen alun perin suunniteltiin toteutettavan. Lisäksi sovellukseen toteutettiin myös yhteenvetonäkymä, josta käyttäjä näkisi, mitkä vastaukset menivät väärin. Sovelluksen lopullinen ulkoasu vastasi melko tarkasti Digital Service Development -kurssilla tehtyjä suunnitelmia, muutamia parannuksia lukuun ottamatta.

Android oli aloittelijalle helposti lähestyttävä vaihtoehto, josta löytyi kattava dokumentointi. Googlen oma sovelluskehitin Android Studio oli myös suhteellisen helppokäyttöinen ja selkeä. Varsinkin käyttöliittymän luominen ohjelman avulla kävi vaivattomasti. Yhtenä tutkimuskysymyksenä olikin, miten erikokoiset näytöt tulee ottaa huomioon sovellusta toteuttaessa. Kävi ilmi, että Androidilla tämä on helpoin toteuttaa luomalla vaihtoehtoisia resurssitiedostoja. Tämä todennettiin tekemällä ulkoasutiedostot muutamalle näyttökoolle ja testaamalla niitä emulaattorissa.

Koska opinnäytetyön tekijällä ei ollut Androidin lisäksi aiempaa kokemusta myöskään SQLite-tietokantajärjestelmästä, haluttiin selvittää miten sitä käytetään Android-sovelluksissa. Järjestelmään tutustuessa huomattiin, että monissa suosituissa mobiilikäyttöjärjestelmissä on jo valmiiksi täysi tuki SQLite-tietokannoille. Näin ollen mitään erillistä asennusta tai käyttöönottoa ei tarvittu. Myös kannan luominen sovelluksessa oli suhteellisen yksinkertaista, ja myös Androidin dokumentaatiosta löytyikin ohjeita tähän. Suositeltavaa oli määrittää kanta erilisessä luokassa, jossa toteutettiin SQLiteOpenHelper-luokka.

Viimeisessä ja haastavimmaksi osoittautuneessa tutkimuskysymyksessä pohdittiin, kuinka sovelluksen muunneltavuus tietokantaa vaihtamalla otetaan huomioon. Lopulta päädyttiin siihen tulokseen, että tietokannan rakenteen ja nimeämiskäytäntöjen suunnittelu oli tärkeintä. Kun tietokannan rakenne pysyy samana, ei lähdekoodiin tarvitse tehdä muutoksia. Tämän lisäksi kannan nimeämiskäytännöt voitaisiin pitää yhtenäisinä tai vaihtoehtoisesti hakea kannan metodeissa tietoja kolumnien numeron, eikä nimen perusteella. Näin myöskään kannan metodeja ei tarvitsisi muokata myöhemmässä vaiheessa. Kun tietokanta olisi luotu suunnitellun rakenteen mukaan, kävisi sen vaihtaminen vaivatta. Jatkokehityksenä voitaisiin myös luoda internettiin erillinen, ohjattu tietokannan luontiohjelma, jonka avulla kantoja voisi luoda kuka tahansa.

Opinnäytetyö vahvisti tekijän omaa mielenkiintoa aihetta kohtaan, ja sovelluksen parissa aiotaankin jatkaa myös työn jälkeen. Seuraavina sovelluksen kertauspuolelle aiotaan luoda loput suunnitellut tehtävät, ja sen jälkeen toteuttaa myös opiskelupuoli. Lisäksi kanjien valintaan haluttaisiin toiminto, jonka avulla käyttäjä voisi luoda omia harjoiteltavien kanjien listoja.

LÄHTEET

Androidhive.info. 2015. Android SQLite Database Tutorial. Viitattu 11.9.2015
http://www.androidhive.info/2011/11/android-sqlite-database-tutorial/

Google Inc. 2015a. Activities. Viitattu 11.9.2015.
http://developer.android.com/guide/components/activities.html

Google Inc. 2015b. Dialogs. Viitattu 11.9.2015.
http://developer.android.com/guide/topics/ui/dialogs.html

Google Inc. 2015c. Fragments. Viitattu 11.9.2015
http://developer.android.com/guide/components/fragments.html

Google Inc. 2015d. Layouts. Viitattu 11.9.2015.
http://developer.android.com/guide/topics/ui/declaring-layout.html

Google Inc. 2015e. Localizing with Resources. Viitattu 11.9.2015
http://developer.android.com/guide/topics/resources/localization.html

Google Inc. 2015f. Menus. Viitattu 11.9.2015.
http://developer.android.com/guide/topics/ui/menus.html

Google Inc. 2015g. Providing Resources. Viitattu 11.9.2015.
http://developer.android.com/guide/topics/resources/providing-resources.html

Google Inc. 2015h. Saving Data in SLQ Databases. Viitattu 2.7.2015.
http://developer.android.com/training/basics/data-storage/databases.html

Google Inc. 2015i. Storage Options. Viitattu 2.7.2015.
http://developer.android.com/guide/topics/data/data-storage.html#db

Harju, Jukka. 2013. Android-ohjelmoinnin perusteet. Books on Demand

Ijas, Silja. 2013. Yleisesti kanjeista. Viitattu 9.11.2015
http://www.kanjikaveri.net/kanjit/yleista.php#lukutavat

Kano, C., Shimizu, Y., Takenaka, H. & Ishii, E. 1989. Basic kanji book. Bonjinsha Co., Ltd.

Karppinen, Takako. 2005. Japanin kielen alkeet. Hakapaino Oy

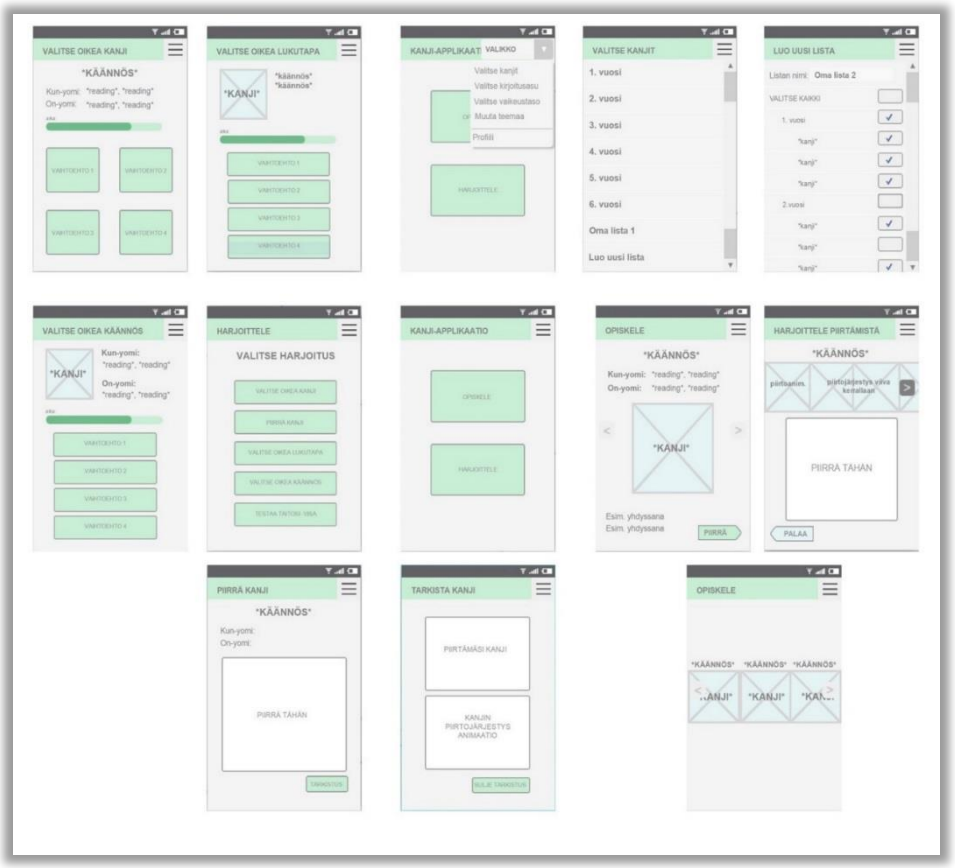
SQLite.org. 2015a. About SQLite. Viitattu 2.7.2015.
http://www.sqlite.org/about.html

SQLite.org. 2015b. Appropriate Uses For SQLite. Viitattu 2.7.2015
https://www.sqlite.org/whentouse.html

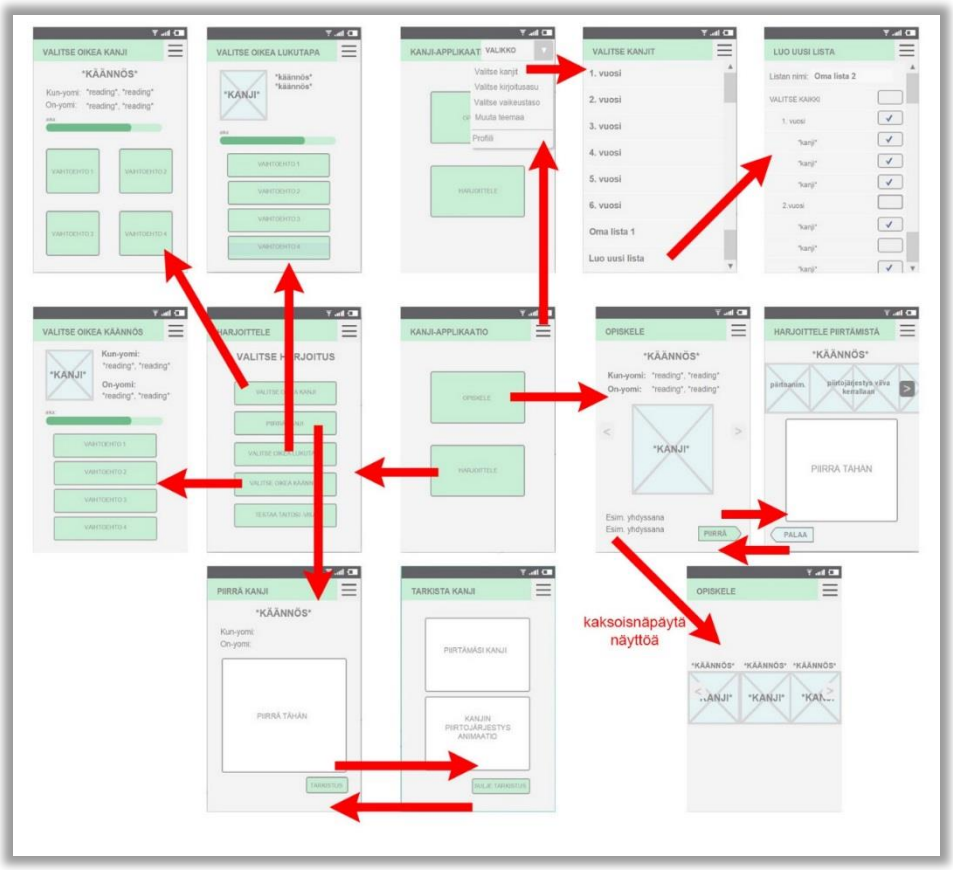
Vogella.com. Android SQLite database and content provider – Tutorial. Viitattu 2.7.2015
http://www.vogella.com/tutorials/AndroidSQLite/article.html

Liite 1/1

SOVELLUKSEN KÄYTTÖLIITTYMÄSUUNNITELMA

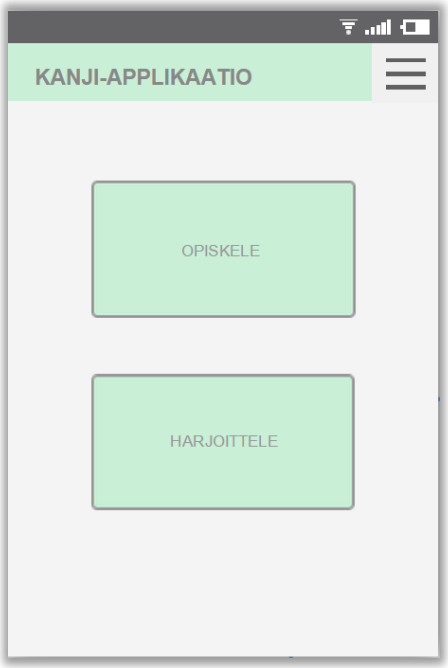


Kuva 1 Sovelluksen käyttöliittymäsuunnitelma



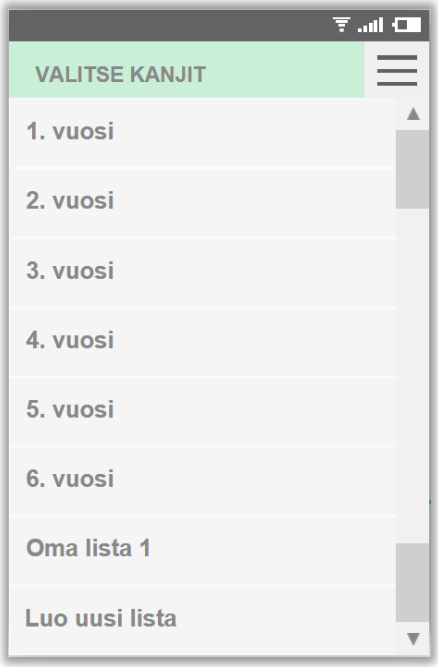
Kuva 2 Sovelluksen käyttöliittymäsuunnitelman linkitykset

Applikaation aloitusnäkymässä käyttäjä pääsee valitsemaan haluaako hän opiskella vai harjoitella kanjeja. Yläpalkin valikko on aina saatavilla, ja sen kautta pääsee valitsemaan käytetyn kirjoitusasun (kana/roomaji) ja opiskeltavat/kerrattavat kanjit sekä muuttamaan teemaa ja tehtävien vaikeusastetta. Valikon kautta pääsee myös tarkastelemaan ja muokkaamaan omaa profiiliaan.

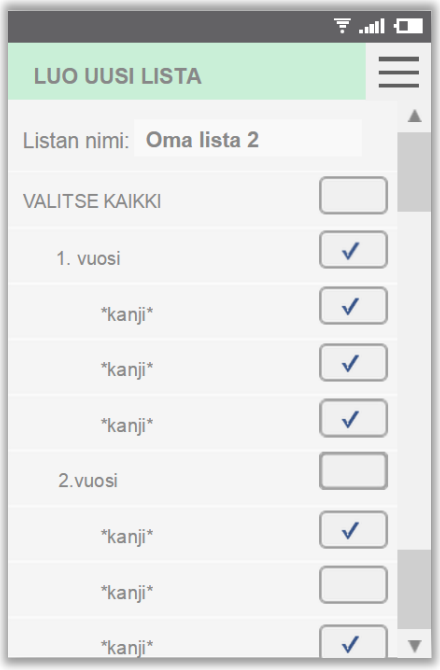


Kuva 3 Aloitusnäkymä

Kuva 4 Sovelluksen valikko



Kuva 5 Valitse kanjit



Kuva 6 Luo oma kanjilista

Opiskeluosiossa näkyviin saa valittujen kanjien tiedot, ja näyttöä pyyhkäisemällä voidaan siirtyä seuraavaan tai edelliseen. Kaksoisnäpäyttämällä näyttöä saadaan myös käyttöön selausikkuna, jonka avulla voidaan vaivattomammin löytää etsitty kanji.

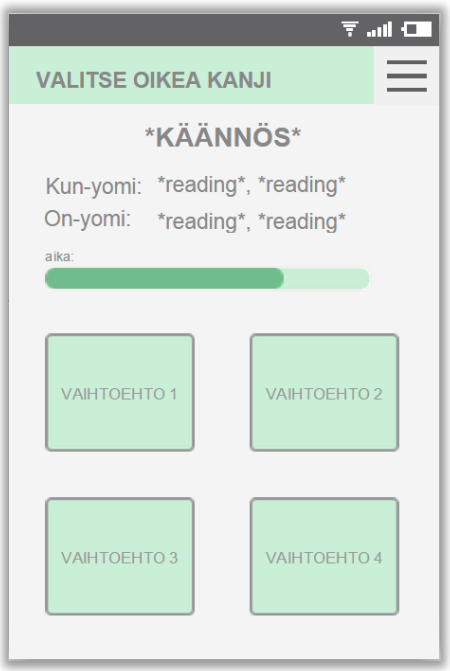
Kuva 7 Opiskele Kanjeja

Kuva 8 Harjoittele kanjin piirtämistä

Kuva 9 Opiskeltavien kanjien selaus-näkymä

Harjoittelunäkymässä voidaan valita neljästä harjoitusvaihtoehdosta haluttu. Lisäksi voidaan valita ”Testaa taitosi” – visa, joka yhdistää näitä harjoituksia, lukuun ottamatta piirtämistä. Visassa voi valita kanjit vain vuosittaisella jaottelulla, eikä voi siis luoda omaa listaa. Vaikeustason voi kuitenkin valita. Vaikeustason noustessa aikarajoja lyhennetään ja vaihtoehtoja kasvatetaan. Helpoimmalla tasolla taas aikaraja otetaan kokonaan pois, ja vaihtoehtoja on vain kolme tai neljä. Tässä tehtävämuodossa voi myös osallistua kisaan, ja saada nimensä leaderboardille.

Kuva 10 Harjoittelun päänäkymä



Kuva 11 Valitse oikea kanji – tehtävä

Kuva 12 Piirrä kanji – tehtävä

Kuva 13 Tarkista piirtämäsi kanji

Kuva 14 Valitse oikea lukutapa – tehtävä

Kuva 15 Valitse oikea käännös – tehtävä