Toni Harju

ANDROID–SOVELLUKSEN JA PALVELINRAJAPINNAN TOTEUTUS

Opinnäytetyö
CENTRIA-AMMATTIKORKEAKOULU
Tietotekniikan koulutusohjelma
Maaliskuu 2017

TIIVISTELMÄ OPINNÄYTETYÖSTÄ

Centria-ammattikorkea-koulu	**Aika** Maaliskuu 2017	**Tekijä/tekijät** Toni Harju
Koulutusohjelma Tietotekniikka		
Työn nimi ANDROID-SOVELLUKSEN JA PALVELINRAJAPINNAN TOTEUTUS		
Työn ohjaaja Sakari Männistö		**Sivumäärä** 27
Työelämäohjaaja Peter Bång		

Opinnäytetyön toimeksiantaja oli Valtacon Oy, joka on Kolpissa sijaitseva IT-yritys. Työn tarkoituksena oli kehittää Android–sovelluksen prototyyppi, jolla on mahdollista demonstroida yrityksen tuotteen toimivuus tuleville asiakkaille. Tuote on uudenlainen etukorttipalvelu, jossa skannataan QR–koodeja ja tällä tavalla kerätään erilaisia bonuksia. Perus palvelinrajapinnan toteutus oli myös osa työtä.

Työn tavoitteena oli oppia Android–sovelluksien kehittämisestä, ja miten Android–sovelluksesta kommunikoidaan erilaisten palvelinrajapintojen kanssa. Asiakkaalle toimitettava prototyyppi oli myös tärkeä tavoite.

Työ oli erittäin opettavainen ja tulokset olivat positiiviset tekijän ja asiakkaan kannalta. Prototyyppi–sovellus saatiin kehitettyä ja asiakas pystyi käyttämään sitä esittelyissä.

Asiasanat
Android, QR, SDK

Centria
UNIVERSITY OF APPLIED SCIENCES

ABSTRACT

Centria University of Applied Sciences	**Date** March 2017	**Author/s** Toni Harju
Degree programme Information Technology		
Name of thesis CREATING AN ANDROID APPLICATION AND A SERVER INTERFACE		
Instructor Sakari Männistö		**Pages** 27
Supervisor Peter Bång		

The commissioner of the thesis was Valtacon Oy, an IT company in Kolppi. The purpose of this thesis was to develop a prototype of an Android application which the commissioner could use to demonstrate the product to customers. The product is a new kind of patron service in which the customers scan QR codes and in turn receives different kinds of bonuses. A basic server interface was also part of the thesis work.

The aim of this thesis was to learn how to develop Android applications and how to communicate with different kinds of server interfaces through the application. An important aim for the commissioner was to be able to produce the actual prototype.

The experience was very rewarding and helped the author of the thesis work increase his knowledge in programming. It was also positive for the commissioner as a prototype could be produced that could be used.

Key words
Android, QR, SDK

1 JOHDANTO

Tämän opinnäytetyön tavoitteena on kehittää Valtacon Oy:lle Android–sovelluksen prototyyppi, jolla on mahdollista demonstroida yrityksen tuotteen toimivuus tuleville asiakkaille. Sovellusta tullaan mahdollisesti käyttämään KiitosEtu–nimisessä etukorttipalvelussa, mutta kortin sijaan käytetään Android–sovellusta, jolla skannataan QR-koodeja. Etukorttipalvelun tarkoitus olisi tarjota halpa ja moderni etukorttisysteemi, joka toimisi jo olemassa olevien etukorttien kanssa.

Työssä on monta ongelmaa, jotka vaativat suunnittelua ja pohdintaa. Yksi isoimmista ongelmista on kokemattomuuteni Android–sovellusten kehittämisessä, ja vaikka Java on tuttu kieli, en ole silläkään montaa projektia tai sovellusta tehnyt. Toinen ongelma tulee olemaan eri kirjastojen löytö ja käyttö, ja eri kirjastoluokkien rajapintojen hyödyntäminen.

Henkilökohtainen tavoitteeni tämän projektin suhteen tulee olemaan Androidin ohjelmointirajapintojen oppiminen ja sen selvittäminen, miten Androidille käytännössä kehitetään sovelluksia.

2 TOIMEKSIANTO

Valtacon Oy on Pohjanmaalla Kolpissa sijaitseva pieni tietotekniikanalalle sijoittuva yritys. Yrityksen perustajalla on ollut kehitteillä uudenlainen etukorttijärjestelmä, joka pääasiassa olisi hyödyllinen pienille yrityksille. Yksi tapa käyttää etukorttia olisi puhelinsovelluksen kautta, johon itse etukortti olisi liitetty numerosarjan kautta. Sovelluksella etuudet saataisiin liikkeisiin sijoitettavien QR–koodien avulla.

Toimeksiantoni on tuottaa prototyyppiversio sovelluksesta Android–alustalle. Sovellukselle on neljä pääasiallista vaatimusta. Näistä vaatimuksista yksi liittyy käyttöliittymään ja kolme tekniikkaan.

Ensimmäinen vaatimus on käyttäjäystävällisen päänäkymän kehittäminen käyttämällä selkeitä ikoneja merkitsemään eri nappien toiminnot. Toinen vaatimus on mahdollisuus skannata QR–koodeja, joka liittyy suoraan etuuksien keräämiseen. Kolmas vaatimus on soveltaa Googlen karttaa eri etukorttia tukevien yrityksien ja liikkeiden listaamiseen kategorioiden mukaisesti. Tätä prototyyppiä varten on riittävää, että kartalle on merkintöjä ja niitä klikkaamalla aukeaa infoikkuna, josta näkee jonkinlaista perustietoa ja klikattavia linkkejä, kuten Facebook ja navigointi kyseiseen paikkaan.

Viimeinen vaatimus on suunnitella ja kehittää tapa, jolla palvelimelta saadaan haettua tarvittavat tiedot yrityksistä ja etuuksista palvelimelta. Tämä vaatimus ei tule olemaan mukana prototyyppisovelluksessa erilaisten ongelmien välttämiseksi, mutta vaatimus on erittäin tärkeä mahdollisia myöhempiä versioita varten.

3 ANDROID–SOVELLUKSIEN KEHITTÄMINEN

On olemassa monia työkaluja, jotka auttavat sovellusten kehityksessä ja internetissä on monia hyviä oppaita, joilla pääsee alkuun. Yksi hyvä paikka aloittaa on Googlen oma kehittäjäsivusto. Android sovelluksen kehittäminen ei ole pelkästään Java–koodin kirjoittamista, ja vaatii jonkin verran kokemusta sovelluskehittämisestä. Alkuun pääseminen on kuitenkin suoraviivaista, vaikka työkaluja on monia ja ominaisuuksia on paljon. (Ravenscraft 2014.)

3.1 Sovelluksien rakenne

Android–projektit koostuvat moduuleista. Moduulit ovat kokoelma lähdekooditiedostoja ja kääntöasetuksia, jotka mahdollistavat ohjelman jaon osiin. Yhdessä projektissa voi olla useampi moduuli, jotka voidaan kääntää, testata ja debuggata erikseen. (Projects Overview.)

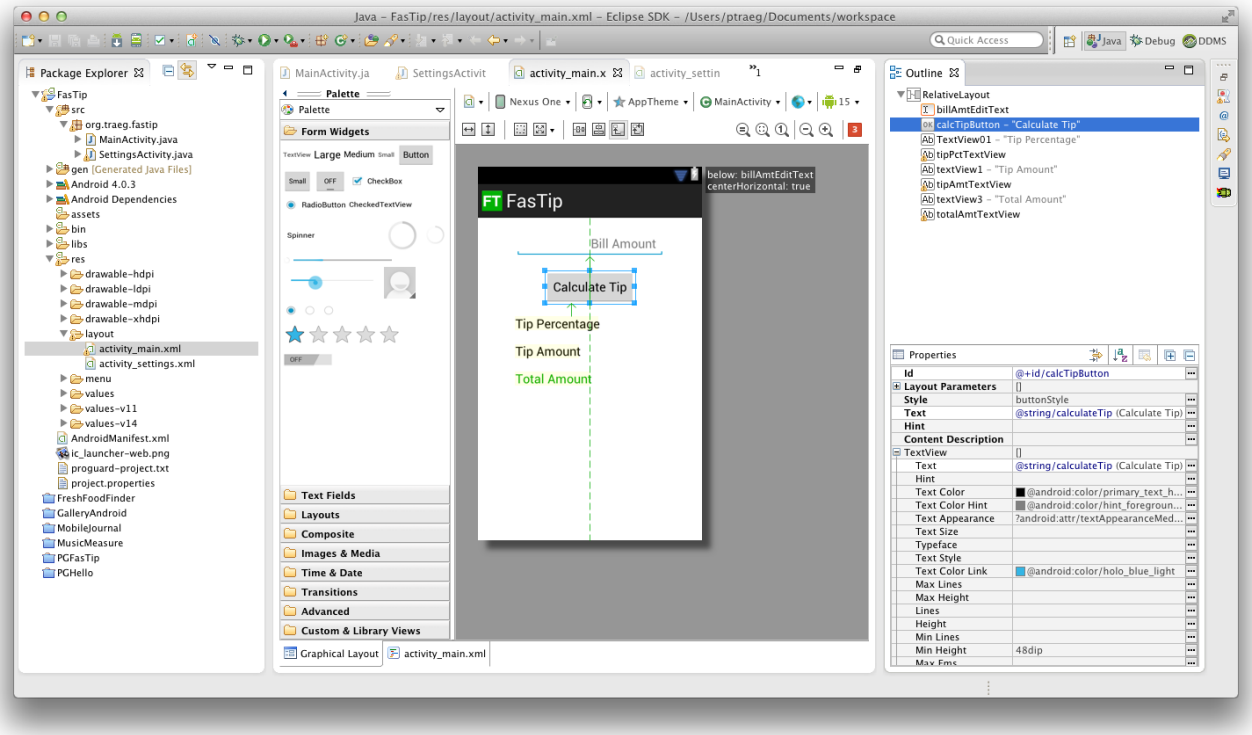
Moduulit koostuvat ohjelmamoduuleista, kirjastomoduuleista ja Google Cloud -moduuleista. Kirjastomoduulit voivat olla joko Android–kirjastoja tai Java–kirjastoja. Android–kirjastot voivat sisältää kaikkia Androidin tukemia tiedostoja kuten lähdekoodia ja resursseja. Java–kirjastot voivat sisältää ainoastaan Java–lähdekoodia. (Projects Overview.)

Sovellukset koostuvat yhdestä tai useammasta Activity:sta. Activity voidaan kuvata asiana, jota käyttäjä voi tehdä sovelluksessa, joka on yleensä kokoruututilassa, mutta voivat myös olla esimerkiksi leijailevia ikkunoita. (Activity-luokka.)

Tyypillisessä sovelluksessa on ylätasonäkymä ja tarkennettu näkymä, riippuen sovelluksen käyttötarkoituksesta ja siitä, minkälaista tietoa halutaan näyttää käyttäjälle. Ylätasonäkymän ja tarkennetun näkymän välissä voi myös olla esimerkiksi kategoria-näkymä ja erilaista navigaatiota. (App Structure.)

3.2 Kehitysympäristöt

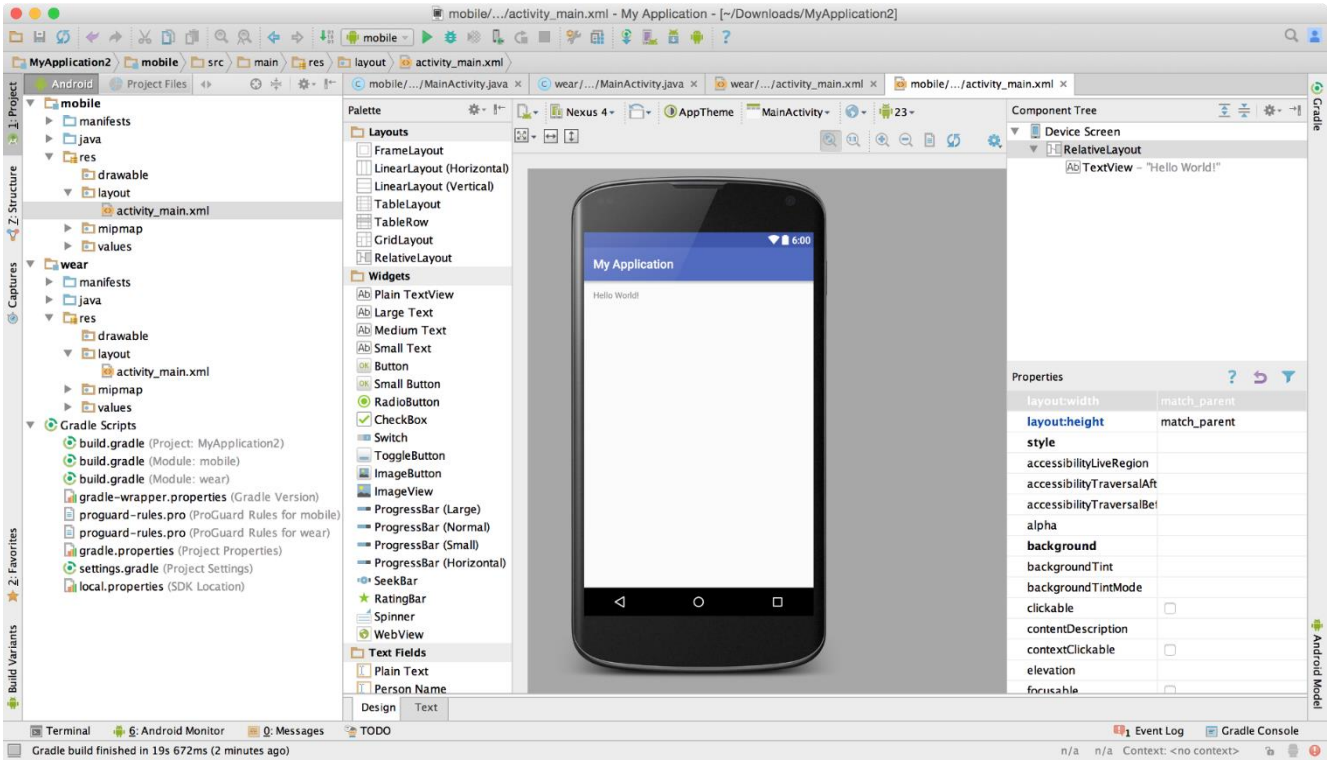
Androidin virallinen kehitysympäristö vuoteen 2013 saakka oli Eclipse-kehitysympäristö, johon Google oli kehittänyt oman Android Developer Tools (ADT) -lisäosan. Vuonna 2013 Google esitti Google I/O tapahtumassa uuden Android Studion, joka pohjautuu suosittuun IntelliJ IDEA–kehitysympäristöön. (Papageorgiou 2014.)



KUVA 1. Eclipse ADT lisäosalla

ADT oli Googlen virallinen lisäosa Eclipselle, joka lisäsi kehitysympäristöön mahdollisuuden luoda ja debuggata Android–projekteja. Lisäosa mahdollisti myös Android ohjelmien viennin Android Application Package (APK) muotoon, joko kirjattuina tai ei-kirjattuina (Android Development Tools for Eclipse 2012). Koska Android Studio on uusin virallinen kehitysympäristö Androidia varten, ei ADT:tä päivitetä enää, ja siksi Google suosittelee siirtämään projektit Eclipsestä Android Studioon (ADT Plugin Release Notes).

Android Studion ominaisuuksiin kuuluu käännös editori, jolla on helppo tehdä ohjelmista monikielisiä. Kehitysympäristössä on myös suoraan tuki esimerkkien lataamiseen GitHubista. Tämä on varsinkin aloittaville kehittäjille erittäin hyödyllistä. Myös ohjelmien testaukseen liittyvät työkalut löytyvät suoraan kehitysympäristöstä. (Android Studio Features.)



KUVA 2. Android Studio

3.3 Android Runtime

Runtime, eli ajonaikainen, on vaihe, jossa sovellus ja sen käyttämät resurssit ja kirjastot on ladattu laitteen muistiin. Tällöin sovellus on käynnissä. (Runtime Definition.)

Dalvik ja Android Runtime (ART) kehitettiin Android-projektia varten. Dalvik on ART:n edeltäkäviä, mutta ne ovat suurimmaksi osaksi yhteensopivia. Kummatkin ajonaikaiset ajavat Dalvik Executable (Dex) tavukoodia, joten Dalvikille luotu sovellus todennäköisesti toimii myös ART:lla. Jotkin tekniikat, jotka toimivat Dalvikissa eivät kuitenkaan toimi ART:ssa. (ART and Dalvik.)

Yksi suurimmista muutoksista ART:ssa on ns. Ahead-of-time (AOT) -kääntäminen. Tämä tarkoittaa sitä, että ART kääntää ohjelmat jo asennusvaiheessa, joka nopeuttaa ohjelmia. Koska jotkin Dalvikin ominaisuudet eivät ole käytössä ART:ssa, voi kääntäjä tuottaa joitakin virheellisiä tiedostoja, jos kyseessä on Dalvikille tarkoitettu ohjelma. (ART and Dalvik.)

ART:ssa on myös parannettu garbage collection (GC). Tämä parannettu GC vie vähemmän taustamuistia ja GC–paussin aikana tapahtuva prosessointi on rinnastettu. Myös GC–pausseja on vain yksi kahden sijaan (ART and Dalvik.)

3.4 Testaus

Android–ohjelmia voidaan testata emulaattorin avulla tai suoraan fyysisellä laitteella. Kohde, jossa sovellus ajetaan, valitaan Run-nappia painamalla. Android Studion ominaisuus, Instant Run, auttaa myös muutosten kanssa. Koodi- ja resurssimuutokset ovat näkyvissä melkein heti jo käynnissä olevilla sovelluksilla, eikä uutta APK:ta tarvitse luoda. (Build and Run Your App.)

Android–emulaattori vaatii Android Virtual Device:n (AVD). AVD:t ovat määrittely erilaisesta Android–laitteesta, kuten puhelimesta, tabletista, kellosta tai TV:stä ja sisältävät muun muassa laitteistoprofiilin, järjestelmäkuvan ja tallennusalueen. Android SDK:n mukana tulee AVD Manager, jolla on helppo luoda, muokata ja poistaa näitä virtuaalisia laitteita. (Create and Manage Virtual Devices.)

Ennen sovelluksen julkaisua on tärkeää testata sovellus myös yhdellä tai useammalla fyysisellä laitteella. Mitä tahansa Android laitetta voidaan käyttää testausympäristönä. Android laitteissa on hyödyllisiä kehittäjäasetuksia kuten virheenjäljitys USB:n kautta ja virheraporttien luonti. Näistä asetuksista ainoastaan virheenjäljitys USB:n kautta on pakollinen, kun halutaan testata fyysisellä laitteella. Windows–käyttöjärjestelmä vaatii myös ajurin, jotta virheenjäljitys USB:n kautta on mahdollista. (Run Apps on a Hardware Device.)

3.5 Jakelu Google Play Storeen

Android ohjelmat täytyy valmistella ennen kuin niitä voi jakaa Play Storeen. Ohjelma täytyy ensin konfiguroida, kääntää ja testata. Konfigurointivaiheet ovat helpot ja liittyvät koodin siistimiseen ja optimointiin. Kääntämisvaiheessa ohjelmasta käännetään debug-versio, jota voidaan sitten testausvaiheessa käyttää testaamiseen. Kun nämä askeleet on käyty läpi, voidaan ohjelma kirjata, ja näin saadaan kirjattu APK–tiedosto, joka voidaan jakaa käyttäjille esim. Google Play Storen kautta. (Preparing for release.)

Android–järjestelmä vaatii kaikkien ohjelmien digitaalisen kirjaamisen sertifikaatilla jonka ohjelman kehittäjä omistaa. Järjestelmä käyttää tätä sertifikaattia tunnistamaan kehittäjät, ja näin se pystyy luomaan luottamussuhteita ohjelmien välille. Sertifikaatin ei tarvitse olla minkään auktoriteetin kirjaama. (Preparing for release.)

Sertifikaatin luomiseen tarvitsee kaksi työkalua. Ensimmäinen on Keytool. Tällä työkalulla luodaan sertifikaatti salasanalla ja nimellä. Tämä työkalu on osa Androidin Software Development Kitiä (SDK). Toinen tarvittava työkalu, on jarsigner. Tällä työkalulla kirjataan APK tiedostot niin, että ne ovat tunnistettavissa. Tämä työkalu on osa Java Development Kittiä (JDK). (Signing Your Apps.)

Android Studion käyttäminen helpottaa koko jakeluprosessia. Kehitysympäristön Gradle–käännösjärjestelmällä voi automatisoida käännösprosessin. Ainoastaan sertifikaatin luominen jätetään kehittäjän vastuulle, mutta sertifikaatin puuttuessakin kehitysympäristö muistuttaa tästä. Gradle–käännösjärjestelmää voi myös käyttää Android Studion ulkopuolella, jos on tarpeellista käyttää jotain muuta kehitysympäristöä. (Preparing for release.)

4 SOVELLUKSEN SUUNNITTELU

Androidille on suhteellisen helppoa suunnitella sovelluksia, koska Googlen dokumentaatiot ovat hyvän laatuisia. Varsinkin käyttöliittymään liittyvät parhaat käytännöt ovat iso apu jo suunnitteluvaiheessa. Nämä parhaat käytännöt takaavat, että kaikki Android–sovellukset ovat yhtenäisiä ulkonäön kannalta, ja siksi niitä kannattaa seurata.

Aloitin koko suunnitteluprosessin miettimällä, kuinka saisin käyttöliittymästä toimeksiannon mukaisen, eli käyttäjäystävällisen. Koska Googlella on jo valmiit käytännöt tähän, päätin soveltaa niitä tähän. Koska käyttöliittymän kuului olla käyttäjäystävällinen ja minimalistinen ja koska kyseessä on prototyyppisovellus, käytin sen suunnitteluun 25 prosentin osuuden kehitysajasta. Käyttöliittymän karkean ulkonäön suunnittelimme Valtaconilla paperilla. Yksi keskeisimmistä ominaisuuksista etusivulla on ristikkomuodossa oleva lista, jossa kahdessa sarakkeessa ovat ikonit eri kategorioihin, ja näitä klikkaamalla suodatetaan kartta, jotta kartalla näkyy ainoastaan se mistä käyttäjä on kiinnostunut.

Toimeksiannon perusteella tiesin myös, että tulisin tarvitsemaan tavan lukea QR–koodeja ja integroida Google Maps omaan näkymään ja lisätä sinne jonkinlaista tietoa. Vaikka palvelinkutsujen vaatimus ei ollut prototyyppisovellukseen välttämätön, päätin tutkia myös tätä etukäteen ja toteuttaa tästä erillisen kirjaston testausta varten. Koska kyseessä on prototyyppisovellus, ja tieto joka näytetään kartalla ei perustu mihinkään dynaamiseen dataan, päätin olla lisäämättä oikeita yrityksiä kartalle, ja käytin muutamaa kaupunkia esimerkkinä.

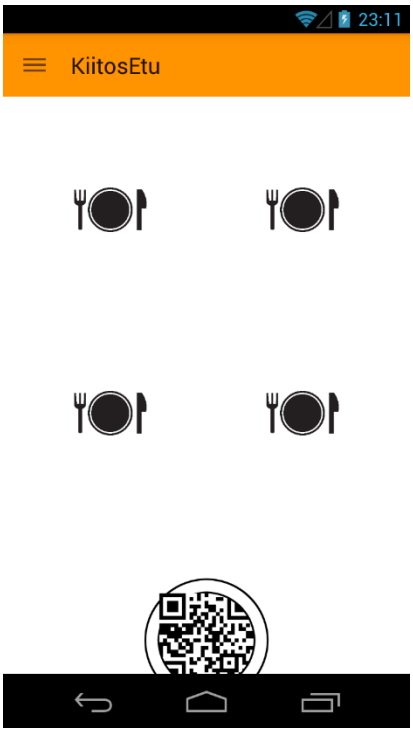
5 SOVELLUKSEN TOTEUTUS

Tässä luvussa käydään läpi sovelluksen toteutus, eri teknologiat joita käytetään ja miksi valitsin tehdä asiat tavalla, jolla tein ne. Aliluvut ovat tärkeysjärjestyksessä. Tämän järjestyksen sovimme toimeksiantajan kanssa, jotta projekti etenisi hyvässä tahdissa.

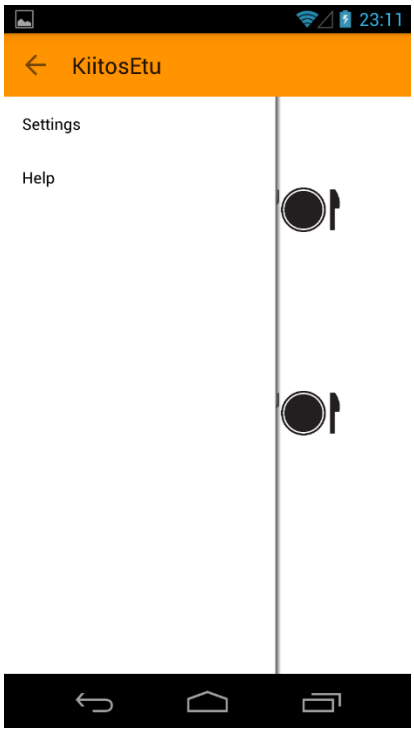
5.1 Käyttöliittymä

Projektin alussa päätin käyttää Googlen uutta Material Design -teemaa käyttöliittymän kehittämiseen. Material teema on suunniteltu antamaan käyttäjälle visuaalisia vihjeitä siitä, miten käyttöliittymää on tarkoitettu käyttää. Tähän käytetään valoa ja liikkeitä, jotka luovat realistisia jakavia saumoja käyttöliittymään. (Material Design – Introduction.)

Material teeman käyttöönotto Android–projektissa on helppoa. Lisäämällä seuraavat rivit res/values/styles.xml nimiseen tiedostoon mahdollistaa teeman käytön:



KUVA 4. Sovelluksen pääkäkymä



KUVA 3. Material teeman sivuvalikko

```
<resources>
  <!-- your theme inherits from the material theme -->
  <style name="AppTheme" parent="android:Theme.Material">
    <!-- theme customizations -->
  </style>
</resources>
```

Material–teema mahdollistaa kaikkien uusien widgettien käytön, joita ei vanhemmissa teemoissa ole mahdollista käyttää. (Creating Apps with Material Design – Getting Started.)

Yksi tärkeimmistä ominaisuuksista Androidin käyttöliittymässä on ActionBar, joka sijaitsee aina sovelluksen ylälaidassa. ActionBar on ollut osa Androidin käyttöliittymää jo versiosta 3.0, mutta siihen on lisätty toiminallisuuksia ajan myötä. Jotta sovellus toimis mahdollisimman monella laitteella, kannattaa käyttää ActionBarin tukiversiota. (Adding the App bar – Setting Up the App Bar.)

Etusivun kategorialista on GridView, johon liitetään kuvanappeja. Koska kategorialista tullaan täyttämään myöhemmin tietokannasta, täytyy sen sisältämät napit luoda koodista. Tämä tapahtuu hakemalla GridView koodissa findViewById-metodilla, ja antamalla tälle ImageAdapter. Tämä adapteri mahdollistaa kuvien lisäämisen dynaamisesti ristikkoon. (User Interface – GridView.)

Koska ristikossa sijaitsevia nappeja täytyy pystyä painamaan, täytyy GridView:lle myös lisätä onClickListener. Tämä on kuuntelija, joka kutsuu sille annettua objektin yli-kirjoitettua metodia, kun jotakin ristikon osiota painetaan.

```
gridview.setOnItemClickListener(new OnItemClickListener() {
    public void onItemClick(AdapterView<?> parent, View v,
        int position, long id) {
        Toast.makeText(HelloGridView.this, "" + position,
        Toast.LENGTH_SHORT).show();
    }
});
```

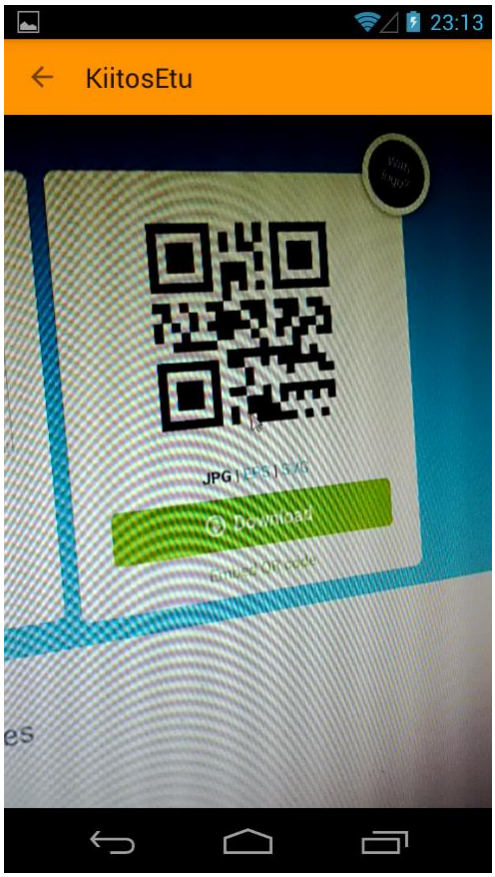
Esimerkistä näemme, kuinka helposti tämän kuuntelijan luominen on. Sama periaate pätee myös muihin nappeihin ja vastaaviin widgetteihin, joita halutaan pystyä painaa. (User Interface – GridView.)

Koska QR-koodien luku on myös tärkeä osa sovelluksen toiminnallisuutta, halusimme lisätä tämän napin näkyville sovelluksen alareunaan. Tämä onnistui lisäämällä FrameLayoutin päänäkymään. FrameLayout on tarkoitettu juuri tätä varten, näyttämään yhden ainoan napin tai muun vastaavan yhdessä osassa ulkoasua. (Google API – FrameLayout.) Esimerkki sovelluksen FrameLayoutista:

```
<FrameLayout
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <ImageButton
        android:id="@+id/qrButton"
        android:src="@drawable/qr_button_selector"
        android:background="@android:color/transparent"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="bottom|center"/>
</FrameLayout>
```

5.2 QR-koodin luku

QR-koodin luku Androidissa osoittautui osittain ongelmaksi. Helpoin tapa tehdä QR-koodien luku on pakottaa käyttäjä käyttämään jotain kolmannen osapuolen tuottamaa sovellusta. Tämä vaatisi, että puhelimella olisi asennettuna tämä toinen sovellus, joka käyttäjäkokemuksen kannalta ei ole paras mahdollinen ratkaisu. Vaatimus oli myös, että QR-koodin skannaus tulisi tapahtua suoraan ohjelman sisältä.



KUVA 5. QR-koodin luku sovelluksessa

Etsinnän jälkeen löysin kirjaston nimeltä Zebra Crossing (ZXing), joka on javalla tehty 1D/2D viivakoodin lukuun tarkoitettu kirjasto. ZXing-kirjastolla pystyy lukea iso osa erilaisia viivakoodeja, myös QR-koodia. (ZXing – Get Started Developing.) Isoin ongelma ZXing-kirjaston integroimisessa projektiin oli vähäinen määrä dokumentaatiota.

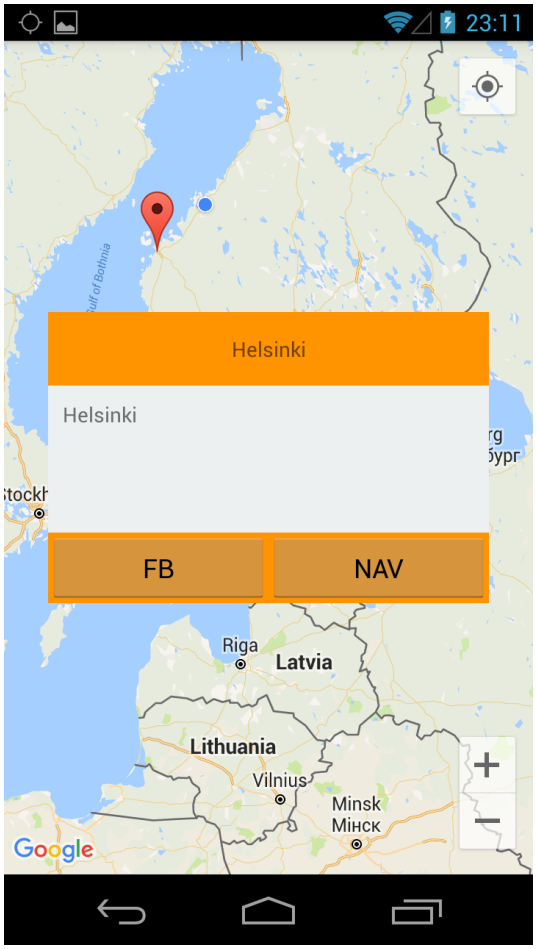
Päätin aloittaa lisäämällä mahdollisuuden avata kameran esikatselu sovelluksen sisällä. Androidin kamera-toiminnot löytyvät Camera-luokasta, ja esikatselu onnistuu lisäämällä Preview-tyyppisen luokan kameralle. (Camera – Creating a preview class.) Tämä Preview-luokka mahdollistaa PreviewCallbackin asettamisen, jonka kautta pystytään reagoimaan kuvaan, jota Preview-luokka tuottaa. Tällä tavoin pystyin lukemaan jokaisen esikatselukuvan. Jos kuva sisälsi QR-koodin, pystyttiin siitä purkaa ulos sisältö.

```
mCamera.setPreviewCallback( new Camera.PreviewCallback() {

    @Override
    public void onPreviewFrame(byte[] bytes, Camera camera) {
        LuminanceSource source = new PlanarYUVLuminanceSource(
            bytes,
            mCamera.getParameters().getPreviewSize().width,
            mCamera.getParameters().getPreviewSize().height,
            0, 0,
            mCamera.getParameters().getPreviewSize().width,
            mCamera.getParameters().getPreviewSize().height, false );
        BinaryBitmap bitmap = new BinaryBitmap( new HybridBinarizer(
source ) );
        QRCodeReader reader = new QRCodeReader();

        Result result;
        String stringResult;

        try {
            result = reader.decode( bitmap );
            stringResult = String.valueOf( result.getText() );
            mIntent.putExtra( "QR_SCAN", stringResult );
            setResult( RESULT_OK, mIntent );
            finish();
        } catch( Exception e ) {}
    }
});
```

Tästä koodista näemme, kuinka esikatselukuvan tavut muutetaan ZXing-kirjaston luokkia ja metodeja käyttäen kuvaksi, binääriseksi bittikartaksi, jota kirjaston QR-koodi lukija pystyy tulkitsemaan. Kuvan sisältö laitetaan lopuksi Intentin sisällöksi, jolloin sisältöä voidaan hyödyntää skannauksen jälkeen muualla.

5.3 Google Maps -integrointi

Google Mapsin integroiminen Android sovelluksiin on helppoa. Google Maps vaatii Google Play services SDK:n joka ladataan Android SDK:n kautta, jonka jälkeen Google play services lisätään projektiin. Android Studiossa on myös valmiiksi Google Maps-tyyppinen Activity, joka automaattisesti luo tarvittavia tiedostoja. Nämä tiedostot ovat näkymiin ja asetuksiin liittyvät xml-tiedostot ja Java-luokka joka määrittelee Activityn. (Google Maps API – Getting Started.)



KUVA 6. Karttanäkymä info-ruudulla

Koska Google Maps on osa Google Play-palvelua, tarvitaan sen käyttöön API avaimen, joka on ilmainen. Avain mahdollistaa kutsujen tekemisen Google Maps APIin. Tämä avain täytyy myös lisätä sovelluksen asetuksiin, jotta sovellus pystyy käyttämään sitä. Avaimen voi myös lisätä Google API Consoleen, jolloin itse sovellus on sallittu käyttämään Google API:a. (Google Maps API – Getting Started.)

Activityn Java–luokka on tiedosto, johon itse koodit kirjoitetaan tai jota laajennetaan esimerkiksi perimällä luokkaa. Luokka jonka Android Studio generoi on hyvin minimalistinen, mutta suoraviivainen:

```
public class MapsActivity extends FragmentActivity implements
OnMapReadyCallback {

    private GoogleMap mMap;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_maps);
        SupportMapFragment mapFragment =
            (SupportMapFragment) getSupportFragmentManager()
             .findFragmentById(R.id.map);
        mapFragment.getMapAsync(this);
    }

    @Override
    public void onMapReady(GoogleMap googleMap) {
        mMap = googleMap;
    }
}
```

Huomaamme, että koodissa kutsutaan findFragmentById, joka viittaa xml-tiedostossa sijaitsevaan fragment-määritelmään:

```
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/map"
    tools:context=".MapsActivity"
    android:name="com.google.android.gms.maps.SupportMapFragment" />
```

Tämä fragment määrittelee, minkä näköinen kartta tulee olemaan sovelluksen sisällä. Koodissa on myös viittaus itse karttaan, GoogleMap-nimiseen luokkaan. Tämä luokka sisältää metodeja, joilla pystytään manipuloida karttaa ja kartan kameraa. Muun muassa kaikki kartan manipulaatio ja niin edelleen tapahtuu onMapReady-metodin sisällä, jota sovellus kutsuu sisäisesti. (Google Maps API – Getting Started.)

Vaatimuksen mukaan karttaan täytyy lisätä merkkejä, joista löytyy yrityksiä, jotka tukevat etukorttia. Tämä tapahtuu kartan Marker-toiminnolla. Näitä on helppo lisätä karttaan:

```
@Override
public void onMapReady(GoogleMap map) {
    map.addMarker(new MarkerOptions()
        .position(new LatLng(10, 10))
        .title("Hello world"));
}
```

Kartalle kerrotaan, että lisätään marker, ja annetaan sille koordinaatit 10, 10 ja kun merkintää klikataan, näytetään info-ikkuna otsikolla ”Hello World”. On myös helppoa saada tieto siitä, onko jotain markeria klikattu ylikirjoittamalla onMarkerClick-metodin. Tämä metodi saa parametrina markerin, jota klikattiin. (Google Maps API – Markers.)

Markerit tukevat suoraan info-ikkunaa asettamalla otsikko title-metodilla ja valinnainen tekstinpätkä snippet-metodilla. Info-ikkunan ongelma on, että sen sisälle ei voida lisätä nappeja, jotka ovat yksi vaatimuksista. Info-ikkuna tukee ainoastaan kuuntelemisen, onko itse ikkunaa klikattu, eikä onko jotakin elementtiä itse ikkunan sisällä klikattu. (Google Maps API – Info Windows.)

Jotta pääsin tämän esteen ylitse, loin oman info-ikkunan, joka asetetaan näkyväksi, kun markeria klikataan. Näin pystyin lisäämään omat kuuntelijat napeille info-ikkunan sisällä ja käynnistämään sopivan Activityn riippuen napin tarkoituksesta. Koska onMarkerClick-metodi saa parametrinä klikatun markerin, on helppoa hakea kyseiselle markerille tarkoitettu data taulukosta, johon kaikki data on ladattu.

```
//Navigation Button
Button inb = (Button)view.findViewById( R.id.info_navigation_button );
inb.setOnClickListener(
    new View.OnClickListener() {

    @Override
    public void onClick(View view) {
        startActivity( new Intent( Intent.ACTION_VIEW, Uri.parse(
"http://maps.google.com/maps?daddr=" + marker.getPosition().latitude
+ "," + marker.getPosition().longitude
        ) ) );
     }
} );

mIsInfoWindowOpen = true;

getMap().getUiSettings().setAllGesturesEnabled( false );
```

Edeltävästä lyhennetystä koodista näemme, kuinka info-ikkunassa sijaitsevan napin layout haetaan findViewById-metodilla ja kuinka napille lisätään oma OnClickListener. Koska kyseessä on navigointinappi, halutaan tällä käynnistää navigointi kyseisen paikan koordinaatteihin tai osoitteeseen. Tämä onnistuu startActivity-metodilla ja Intent-luokalla.

Intentit ovat tapa pyytää toiminnallisuutta jostain muusta sovelluskomponentista. Näille on kolme pääasiallista käyttötapausta. Activityn tai Servicen käynnistäminen, ja jonkin viestin lähetys kaikille sovelluksille. (Intents and Intent Filters.)

Yllä olevassa koodissa startActivity-metodille annetaan parametrina uusi Intent, jolle annetaan parametrina osoite Googlen kartta-linkki, jossa koordinaatit ovat mukana. Näin Android osaa pyytää lupaa käynnistää navigointi eri ohjelmassa.

Koska tekemäni info-ikkuna vie ison osan näytön tilasta, halusin että karttaa ei voi manipuloida silloin, kun info-ikkuna on auki. Tämä tapahtuu setAllGesturesEnabled-metodilla, jolle annetaan arvoksi false, eli epätosi. Kun info-ikkuna suljetaan, asetetaan tälle metodille arvoksi true, eli tosi, joka sallii kartan manipuloinnin uudelleen.

5.4 Palvelinkutsut

Vaikka palvelinkutsujen ei tarvinnut olla osa itse KiitosEtu-sovellusta, haluttiin kuitenkin jonkinlainen proof-of-concept (POC), koska palvelinkutsut ovat tärkeä osa sovelluksen toiminnallisuudessa. POCin kehittäminen tehtiin testisovelluksen jälkeen.

Koska palvelinkutsuja täytyy tehdä useampia kerralla, ja koska kutsut voivat olla hitaita, täytyy ne tehdä asynkronisesti. Asynkroniset kutsut ajetaan taustaprosessissa, jolloin itse pääsovellus ei häiriinny näistä kutsuista. Helpoin tapa ajaa tällaisia asynkronisia kutsuja Androidissa on käyttää AsyncTask-luokkaa. (Google API – AsyncTask.)

AsyncTask-luokka koostuu neljästä eri vaiheesta. Nämä ovat onPreExecute, doInBackground, onProgressUpdate ja onPostExecute. Nämä neljä metodia voidaan yli-kirjoittaa, joista doInBackground yli-kirjoitetaan yleensä aina, kuten myös onPostExecute. doInBackground-metodissa määritellään mitä kyseinen AsyncTask tekee, kun se ajetaan. onPostExecute-metodi saa parametrinä tuloksen doInBackground-metodista, jolloin tulosta voidaan manipuloida haluamalla tavalla. AsyncTask-luokkalle täytyy myös antaa kolme tyyppiä template-muodossa, kun luokka peritään. AsyncTaskit täytyvät myös olla aliluokkia jonkin luokan alla. (Google API – AsyncTask.)

```
private class MyTask extends AsyncTask< Void, Void, Void > {}
```

Kolme Void-tyyppiä jotka edeltävässä luokkamäärittelyssä annetaan AsyncTaskille, kertovat luokalle mitä tyyppiä parametrit, edistys ja palaute ovat. Kun nämä ovat tyyppiä Void, kerrottaan luokalle, että kyseistä tyypistä ei välitetä ollenkaan. (Google API – AsyncTask.)

AsyncTaskit ajetaan kutsumalla execute-metodia. Tämä metodi ottaa vastaan yhden tai useamman parametrin. Parametrin tyyppi täytyy täsmätä templatessa ensimmäisen annetun tyypin kanssa. AsyncTaskin voi myös peruuttaa kutsumalla cancel-metodia. Jotta peruuttaminen toimis parhaiten kuuluisi doInBackground-metodissa tarkistaa isCancelled-metodilla, onko peruutus pyydetty, jotta peruuttamiseen voidaan reagoida paremmin. Jos AsyncTask peruutetaan, kutsutaan onCancelled-metodi onPostExecute-metodin sijaan. (Google API – AsyncTask.)

Koska palvelinkutsuissa halutaan tehdä HyperText Transfer Protocol (HTTP) -kutsuja, kuten GET ja POST kutsuja, täytyy kummallekin kutsutyypille tehdä omat AsyncTaskit. Koska AsyncTaskit täytyvät olla aliluokkia, täytyy ensin luoda pääluokka, jonka sisälle tehtävät tulevat. Tähän pääluokkaan on sopivaa myös luoda tapahtumakuuntelija, event listener, josta saadaan tieto, kun jokin tehtävä on valmis. Pääluokkaan voidaan myös luoda metodit, joilla luodaan tehtävät ja joille voidaan syöttää parametrit. Esimerkki GET-tyyppisten kutsujen tehtävätästä:

```
@Override
protected String doInBackground(String... uri) {
    HttpParams httpParams = new BasicHttpParams();
    HttpConnectionParams.setConnectionTimeout(httpParams, 3000);
    HttpConnectionParams.setSoTimeout( httpParams, 5000 );
    HttpClient client = new DefaultHttpClient( httpParams );
    String result = null;

    try {

        HttpResponse response = client.execute(
            new HttpGet( uri[0] )
        );
        StatusLine status = response.getStatusLine();
```

```
        switch( status.getStatusCode() ) {
            case RESPONSE_OK:

            ByteArrayOutputStream out = new ByteArrayOutputStream();
            response.getEntity().writeTo( out );
            result = out.toString();
            out.close();
            break;

            default:

            response.getEntity().getContent().close();
            Log.e("CustomError", status.getReasonPhrase());
            throw new IOException( status.getReasonPhrase() );

        }

    } catch( SocketTimeoutException e ) {
        Log.e( "CustomError", e.getMessage() );
    } catch( IOException e ) {
        Log.e( "CustomError", e.getMessage() );
    }

    return result;

}
```

Kuten näemme koodista, luodaan doInBackground-metodissa HTTP-yhteys, jonka jälkeen yritetään tehdä kutsu. Ilman virheitä saadaan palaute 200-tyyppisellä koodilla. Tämä palaute kirjoitetaan String–tyyppiseen muuttujaan, joka palautetaan metodin lopussa. HTTP-kutsu ajetaan try-catch-lausekkeen sisällä, jolloin teknisistä virheistä saadaan viesti. Jos HTTP-kutsu palauttaa jotain muuta kuin 200-tyyppisen koodin, saadaan siitä erikseen tieto switch-lausekkeessa, jolloin siitä kirjoitetaan virheviesti ja heitetään virhe, jonka catch-lauseke voi napata.

```
@Override
protected void onPostExecute( String result ) {
    super.onPostExecute( result );
    if( result == null ) {
        Toast.makeText(
            mContext,
            "SRA Request Failed", Toast.LENGTH_SHORT).show();
        return;
    }
    JSON data = new JSON( result );
    int response_code = (int)data.get( "RESPONSE_CODE" );
    switch (response_code) {
        case RESPONSE_OK:

        if (mSRA.hasSRAEventListener())
            mSRA.mSRAEventListener.onGet(data);
        break;
        default:

        if (mSRA.hasSRAEventListener())
            mSRA.mSRAEventListener.onError(data);
        break;
    }
}
```

Edeltävästä koodista näemme, kuinka doInBackground-metodin palaute prosessoidaan onPostExecute-metodissa. Jos palaute on null-tyyppiä, näytetään virheviesti käyttäjälle. Palautettu merkkijono muutetaan JavaScript Object Notation -tyyppiseksi objektiksi, jolloin sitä on helpompi käsitellä. Täällä myös katsotaan, oliko palautettu OK vai tapahtuiko jokin virhe. Palautteen tyypistä riippuen annetaan data tapahtumakuuntelijan onGet- tai onError-metodille.

Tapahtumakuuntelija koostuu lyhyestä rajapinnasta, jonka sovelluksen kehittäjän on tarkoitus toteuttaa omaa käyttöä varten:

```
public interface CustomEventListener {

    public void onGet( JSON data );
    public void onPost( JSON data );
    public void onError( JSON data );

}
```

AsyncTaskin käynnistys on helppoa, mutta koska GET-tyyppisten HTTP-kutsujen teko voi vaatia erilaisten parametrien lisäyksen kutsuun, on taskien pääluokkaan hyvä luoda apumetodeja tätä varten:

```
public void get( String action, HashMap< String, String > params ) {
    if( params != null ) {
        String keys_values = "";
        for (HashMap.Entry<String, String> param : params.entrySet())
        {
            String key = param.getKey();
            String value = param.getValue();
            keys_values += "&" + key + "=" + value;
        }

        new GetTask(this, mContext).execute(
            mApiUrl + "?a=" + action + keys_values
        );
    } else {
        new GetTask(this, mContext).execute(
            mApiUrl + "?a=" + action
        );
    }
}
```

Koodista näemme, kuinka taskin käynnistäminen on hieman erilainen, jos GET-kutsulle on annettu parametreja. HashMap-tyyppinen lista täytyy muuntaa String-tyyppiseksi, jotta se voidaan lisätä kutsun perään.

5.5 Palvelimen rajapinta

Jotta palvelinkutsuja pystytään tehdä Android–sovelluksesta, tarvitaan jokin palvelinsovellus, johon kutsut voidaan lähettää, ja koska Android sovellus olettaa vastausten olevan JSON-muodossa, täytyy palvelimen pystyä lähettämään vastaukset tässä muodossa. Koska minulla on paljon kokemusta PHP:stä, päätin tehdä oman kevyen kirjaston, joka mahdollistaa pyyntöjen vastaanottamisen ja lähettämisen. Tiesin myös, että tietokannasta hakeminen tulee olemaan tärkeä osa sovellusta, joten lisäsin kirjastoon tavan tehdä tietokantahakuja turvallisesti.

Kirjasto koostuu neljästä tärkeästä luokasta ja funktiosta. Nämä ovat REQUEST-luokka, QUERY-funktio, URL-funktio ja SUCCESS/ERROR-funktiot.

```
class REQUEST {
public static function GET(
    $key,
    $as_html = false,
    $return_error = false ) {
    if( !isset( $_GET[ $key ] ) ) {
        if( $return_error )
            ERROR( RESPONSE_BAD_REQUEST, 'Invalid index' );
        else
            return '';
    }
    if( $as_html )
        return htmlentities( $_GET[ $key ], ENT_QUOTES, 'UTF-8' );
    else
        return utf8_decode( $_GET[ $key ] );
}
}
```

Edeltävästä koodista näemme, että GET-metodi ottaa vastaan kolme parametria. Nämä ovat avaimen nimi Uniform Resource Identifierissä (URL), as_html ja palautetaanko virhe. Kaksi viimeistä parametria ovat valinnaiset, ja ainoastaan ensimmäinen on pakollinen. Esimerkki avaimesta URL:ssa ”a=” osuus seuraavassa:

```
index.php?a=someAction
```

Avaimen ”a” arvo olisi tässä tapauksessa ”someAction”. as_html -parametrilla voidaan valita, halutaanko avaimen sisältö HTML-käännettynä vai pelkästään UTF-8–dekoodattuna. UTF-8 dekoodausvaihe on tärkeä, koska Java käyttää vakiona UTF-8–enkoodausta merkkijonoille, kun taas PHP ei. Tämä tarkoittaa, että PHP:n merkkijonot täytyy muuntaa UTF-8 muotoon ennen kuin ne voidaan lähettää takaisin Android sovellukselle. Kolmannella parametrilla määritellään, halutaanko että metodi palauttaa virheen, vai palautetaanko tyhjä merkkijono. POST-metodi toimii samalla periaatteella.

```
function QUERY( $query, array $params = [] ) {
    $host = CONFIG::GET( 'mysql', 'host' );
    $user = CONFIG::GET( 'mysql', 'user' );
    $pass = CONFIG::GET( 'mysql', 'pass' );
    $data = CONFIG::GET( 'mysql', 'prefix' ) . Config::get( 'mysql',
'data' );

    $statement = null;
    try {
        $connection = new PDO(
        "mysql:dbname={$data};host={$host}", $user, $pass
        );
        $statement = $connection->prepare( $query );
        $statement->execute( $params );
    } catch( PDOException $e ) {
        ERROR( RESPONSE_BAD_REQUEST, $e->getMessage() );
    }
return $statement;
}
```

QUERY-funktio toimii yksinkertaisena wrapperina PHP:n PHP Data Objects-kirjastolle (PDO). PDO itsessään on kevyt rajapinta, jolla voidaan ottaa yhteys tietokantoihin. Toimiakseen PDO vaatii ajurin tietokantatyypille, jota halutaan käyttää. PDO ja monet ajurit ovat osa PHP:tä versiosta 5.1 ja eteenpäin. (PHP Data Objects.)

Funktio palauttaa tavallisen PDO:n statement-objektin, jota voidaan käyttää esimerkiksi hakemalla kaikki tieto jota PDO:lle syötetyllä SQL-kyselyllä halutaan. Esimerkki kyselystä, joka palauttaa kaikki kaupungit ”city” nimisestä taulusta. Tulokset laitetaan sitten result-nimiseen muuttujaan:

```
$query = QUERY( "SELECT city.name FROM city ORDER BY city.name" );

$result = [];
$result[ ‘CITIES’ ] =
    $query->fetchAll( PDO::FETCH_GROUP | PDO::FETCH_ASSOC );
```

SUCCESS ja ERROR-funktiot ovat apufunktioita, joilla voidaan nopeasti palauttaa kutsuvalle sovellukselle joko 200 (SUCCESS) tai jonkin muun numerisen koodin (ERROR), jotta kutsuva sovellus voi reagoida sopivalla tavalla.

URL-funktiolla määritellään, mitä toimenpiteitä palvelinsovelluksella, eli rajapinnalla on olemassa. Funktio tarkistaa jokaisessa kutsussa, täsmääkö ”a” avaimessa oleva arvo johonkin määriteltyyn toimenpiteeseen. Jos avain täsmää, ajetaan tämä toimenpide.

```
URL( [
    '' => function() {
    $query = QUERY("SELECT city.name FROM city ORDER BY city.name");
    $result = [];
    $result[ 'TOWNS' ] =
        $query->fetchAll( PDO::FETCH_GROUP | PDO::FETCH_ASSOC );

    SUCCESS( $result );
    },
```

```
    'update' => function() {
    $id = REQUEST::POST( 'id' );
    $name = REQUEST::POST( 'name' );

    $query = QUERY( "UPDATE point SET name=? WHERE id=?", [ $name,
$address, $lat, $lon, $id ] );

    SUCCESS( [] );
    }
] );
```

Edeltävässä esimerkissä URL-funktiolle on annettu kaksi eri toimenpidettä. Ensimmäinen on tyhjä merkkijono. Tämä toimii oletustoimenpiteenä ja sitä kutsutaan esimerkiksi silloin, kun ”a” avainta ei ole annettu. Toinen toimenpide on päivitystoimenpide, joka pyytää id ja name -nimiset POST-tietueet. Nämä syötetään QUERY-funktiolle, joka päivittää uuden nimen id:lle.

Tämä palvelinrajapinta on hyvin yksinkertainen ja helppokäyttöinen ratkaisu, varsinkin prototyyppisovellusta varten. Tietoturvan kannalta tarvittaisiin vielä lisätöitä, mutta totesin että tällaista prototyyppiä varten tietoturva ei ole ensimmäinen prioriteetti.

5 YHTEENVETO

Projektin alussa olin erittäin epävarma, tulisinko koskaan saamaan mitään aikaiseksi, mutta jääräpäisyyden ja omistautumisen takia jatkoin kuitenkin eteenpäin. Varsinkin QR-koodin lukuvaihe oli vaikea, ja apua oli vaikea löytää. Huomasin kuitenkin hyvin nopeasti, että pelkkä kokeileminen on hyvä tapa saada vaikeat asiat toimimaan, ja näin myös tapahtui.

Koska valitsin käyttää Android Studiota, joka oli vielä projektin alussa alkutekijöissä, aiheutti tämäkin ongelmia. Ohjeita oli vaikea löytää, ja koska Android Studio käytti Gradle-nimistä riippuvuushallintaa tunnetumman Maven-riippuvuushallinnan sijaan, oli tästäkin haasteellista löytää ohjeita.

Jälkikäteen mietittynä projekti ei ollut mitenkään erityisen haasteellinen kokonaisuudessaan. Yksi tärkeä syy tähän oli, että prototyyppi suunniteltiin suhteellisen hyvin, ja toimeksiantajani tiesi tarkalleen mitä hän halusin, joten oli helppoa aina kysyä, kun piti tehdä jonkinlainen päätös liittyen esimerkiksi käyttöliittymään tai toiminnallisuuteen.

Googlen dokumentaation laatu auttoi myös erittäin paljon, ja koska en ole mitenkään erityisen hyvä suunnittelemaan käyttöliittymiä, oli käyttöliittymä oppaasta ja Googlen parhaista toimintatavoista erittäin paljon apua.

LÄHTEET

Activity-luokka. Saatavissa: https://developer.android.com/reference/android/app/Activity.html. Viitattu: 29.10.2016.

ADT Plugin Release Notes. Saatavissa: http://developer.android.com/tools/sdk/eclipse-adt.html. Viitattu 21.4.2016.

Adding the App bar – Setting Up the App Bar. Saatavissa: https://developer.android.com/training/appbar/setting-up.html. Viitattu 29.11.2016.

Android Development Tools for Eclipse. Saatavissa: https://marketplace.eclipse.org/content/android-development-tools-eclipse. Viitattu 21.4.2016.

Android Studio Features. Saatavissa: http://developer.android.com/tools/studio/studio-features.html. Viitattu 21.4.2016.

App Structure. Saatavissa: https://developer.android.com/design/patterns/app-structure.html. Viitattu: 29.10.2016.

ART and Dalvik. Saatavissa: https://source.android.com/devices/tech/dalvik/. Viitattu 19.4.2016.

Build and Run Your App. Saatavissa: https://developer.android.com/studio/run/index.html. Viitattu 12.10.2016.

Camera – Creating a preview class. Saatavissa: https://developer.android.com/guide/topics/media/camera.html#camera-preview. Viitattu: 30.10.2016.

Create and Manage Virtual Devices. Saatavissa: https://developer.android.com/studio/run/managing-avds.html. Viitattu 29.10.2016.

Creating Apps with Material Design – Getting Started. Saatavissa: https://developer.android.com/training/material/get-started.html. Viitattu. 28.11.2016.

Google API – AsyncTask. Saatavissa: https://developer.android.com/reference/android/os/AsyncTask.html. Viitattu: 25.11.2016.

Google API – FrameLayout. Saatavissa: https://developer.android.com/reference/android/widget/FrameLayout.html. Viitattu: 29.11.2016.

Google Maps API – Getting Started. Saatavissa: https://developers.google.com/maps/documentation/android-api/start. Viitattu: 6.10.2016.

Google Maps API – Info windows. Saatavissa: https://developers.google.com/maps/documentation/android-api/infowindows. Viitattu: 10.10.2016.

Google Maps API – Markers. Saatavissa: https://developers.google.com/maps/documentation/android-api/marker. Viitattu: 6.10.2016.

Intents and Intent Filters. Saatavissa: https://developer.android.com/guide/components/intents-filters.html. Viitattu: 10.10.2016.

Material Design – Introduction. Saatavissa: https://material.google.com/#introduction-principles. Viitattu: 28.11.2016.

User Interface – GridView. Saatavissa: https://developer.android.com/guide/topics/ui/layout/gridview.html. Viitattu: 29.11.2016.

Papageorgiou, P. 2014. Android Studio vs Eclipse: What are the main differences? Saatavissa: http://www.avocarrot.com/blog/android-studio-vs-eclipse-main-differences/. Viitattu 21.4.2016.

PHP Data Objects. Saatavissa: http://php.net/manual/en/intro.pdo.php. Viitattu 28.11.2016.

Preparing for release. Saatavissa: http://developer.android.com/tools/publishing/preparing.html. Viitattu 21.4.2016.

Projects Overview. Saatavissa: https://developer.android.com/studio/projects/index.html. Viitattu 15.10.2016.

Ravenscraft, E. 2014. I Want to Write Android Apps. Where do I Start? Saatavissa: http://lifehacker.com/i-want-to-write-android-apps-where-do-i-start-1643818268. Viitattu: 29.10.2016.

Run Apps on a Hardware Device. Saatavissa: https://developer.android.com/studio/run/device.html. Viitattu: 29.10.2016.

Runtime Definition. Saatavissa: http://techterms.com/definition/runtime. Viitattu 4.10.2016.

Signing Your Apps. Saatavissa: http://developer.android.com/tools/publishing/app-signing.html. Viitattu 21.4.2016.

ZXing – Get Started Developing. Saatavissa: https://github.com/zxing/zxing. Viitattu 30.10.2016.