Högskolan i Gävle

Introduktion till att skapa appar för android

HT 2022

Projektrapport: AppDictionary

Namn: Tommy Hallqvist

e-mail: tommy.hallqvist@gmail.com

2022-12-26

Innehållsförteckning

Inledning

Som avslutande projektarbete till *kursen Introduktion till att skapa appar för Android* bestämde jag mig för att försöka utveckla en översättnings-app. Eftersom jag har det ryska språket som ett av mina stora fritidsintressen var det naturligt att språken jag arbetade med i detta projekt var just svenska och ryska. Själva idén och funktionaliteten med appen är dock helt språk-neutral, och kan implementeras med valfri språkkombination.

Även om jag nu arbetar inom data (Service Desk Analyst) har jag länge varit intresserad av rysk/sovjetisk historia, och har genom åren gjort flertalet resor till olika östeuropeiska länder. Detta var en möjlighet för mig att kombinera mina intressen och samtidigt skapa ett program, som skulle vara användbart för mig i framtiden. Jag har en relativ mångsiding IT-bakgrund med viss erfarenhet och utbildning inom programmering, dvs: webprogrammering i JavaScript, React, XHTML och CSS. Jag har lite erfarenhet av Java-programmering, men har aldrig kommit i kontakt med Kotlin förut.

Idén till projektet utgick från ett antal Excel dokument där jag sparat långa listor med olika medicinska och juridiska termer översatta till ryska. Min tanke var att försöka använda dom på något sätt och samtidigt utforma ett praktiskt och funktionellt verktyg utifrån detta.

Det skulle också vara ett verktyg som kan erbjuda lite andra features, än vad som erbjuds av andra översättningsprogram (Google-translate och dylikt). Terminologin som används inom sjukvårds- och rättstolkning omfattas inte av de generella online-översättningsresurser. Eftersom jag med tiden skapat mig en omfattande lista av sådana termer, kändes det som en bra idé att kunna integrera dem i min app. Den är i första hand riktad till dem som arbetar professionellt med tolkning/översättning, och de som vill själva kunna skräddarsy och uppdatera sina ordlistor. Appen är dessutom designad för att, med små justeringar, även kunna användas av studenter för att ”plugga” glosor

Jag ville skapa ett enkelt men praktiskt gränssnitt som skulle hjälpa mig snabbt i komplicerade översättningssituationer, där det inte finns tid att söka på Internet eller ordböcker på traditionellt vis.

Problemanalys

Hur utvecklar man en ny typ av översättningsverktyg/app, som skiljer sig från traditionella program/metoder? Traditionella hjälpmedel är ofta alltför opraktiska och tidskrävande att använda på ett effektivt sätt t.ex. i en tolksituation. Man kan notera följande problem med traditionella översättningsverktyg.

1. Ordböcker – tidskrävande att använda
2. Översättningsprogram – kräver ofta dator
3. Webbaserade översättningsprogram är oftast allmänt inriktade.

Det gällde alltså för mig att utveckla en praktisk och snabbnavigerad design, men jag ville också implementera en uppslagsverk-funktion. Även om man kan översätta ett ord, kan det vara svårt att minnas vad det specifika ordet/termen betyder. I stället för att behöva leta i flera olika program skulle det vara smidigare att ha tillgång till all information i samma app.

Till att börja med behövde jag fastställa ut en generell design såväl programmeringsmässigt som utseendemässigt, som jag kunde modellera min app utifrån. Eftersom jag är nybörjare in Kotlin gällde det också att hitta programmeringsmässiga metoder som var praktiskt möjliga att implementera under en relativt kort tidsperiod.

En av de första frågorna var att bestämma datakällan, dvs hur appen skulle hämta in sina data. Dessutom behövde jag även finna en lösning på hur man sedan på ett effektivt sätt uppdaterar/underhåller ordlistorna.

Huvudalternativen som fanns tillgängliga::

1. Ladda in informationen vi URL från Internet, genom t.ex. Retrofit
2. Ladda in informationen från en lokal Assets/Resource fil.
3. Implementera informationen genom att ”hårdkoda” den direkt i Kotlin-filerna.

Nästa fråga var i vilket filformat skulle inputdatan sparas? Det var nödvändigt att fastställa vilken filtyp som var lämplig att använda i detta projekt. Jag ville även titta på framtida alternativa versioner av denna app, och hur användaren ska interagera med appdatan. Några av alternativen för detta var:

1. Excel/CSV
2. XML
3. JSON

Jag behövde undersöka om en feature som t.ex. *Speech-to-text* kunde vara aktuell. Det var också viktigt för mig att hitta en bra lösning för sökfunktionen, eftersom den är central för att få en funktionell översättnings-app. Vilken programmeringsmetod var lämplig? Var det aktuellt att använda t.ex. **ViewModels**?

På vilket sätt kan man använda online-material som datakällor? Jag funderade över olika alternativ, bl.a.:

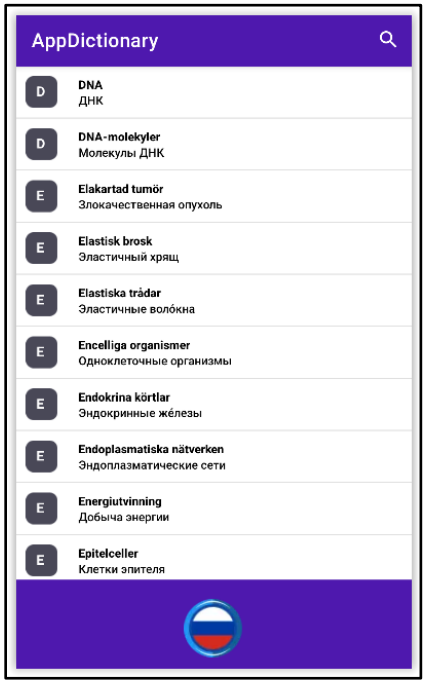
1. Google Translate
2. Firebase ML Kit
3. Internet som källa för t.ex. URL:s och bilder

.

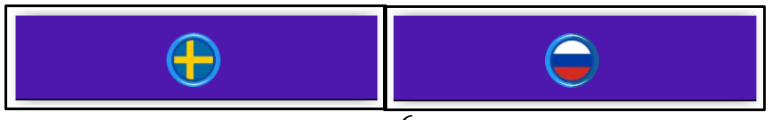
UI

Jag valde en diskret och relativt enkel grunddesign, eftersom detta är tänkt främst som ett arbetsverktyg och inte en nöjesapp. Det var inte viktigt att presentera vackra färgkombinationer eller konstnärliga smådetaljer. Det skulle vara en tydligt och praktisk design, med inte för många knappar/funktioner eller ögongodis. Den utgår från en traditionell scrollbar lista, men svarta bokstäver mot vit bakgrund, så att texten framgår tydligt. Generellt så bestämde jag mig för att undvika intensiva färger, t.ex. röd, orange, som kunde distrahera eller upplevas tröttsamma för användaren. Efter en del experimenterande beslutade jag mig för att använda *sans-serif-medium*, *normal* för brödtexten och *sans-serif-medium*, *bold* för rubrikerna. Det blev en bra balans mellan stil och lättlästhet i appen som helhet.

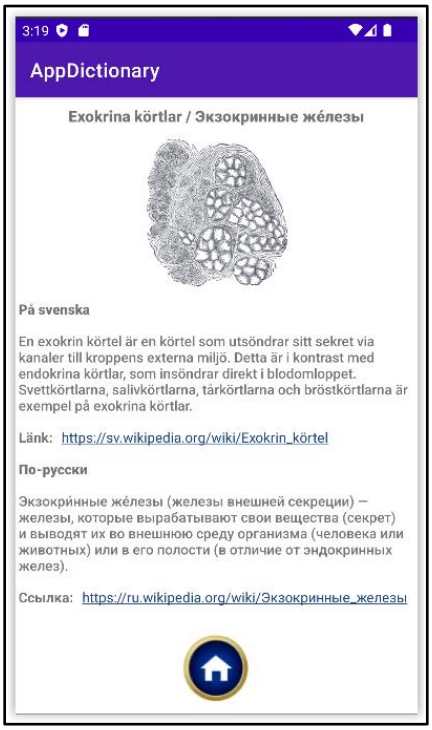
När man öppnar appen visas den första bokstaven i varje ord i bokstavsordning till vänster, vilket underlättar överblicken vid scrollning och sökning i långa lisor.



Menyknapparna i nederkanten är lättidentifierade, och lättåtkomliga vilket borde göra navigeringen intuitiv för användaren. Formen och placeringen skulle påminna om den traditionella Iphone-hemknappen. Jag strävade efter att implementera så få element som möjligt för att maximera appens praktiska användning. Om man trycker på den ryska flaggan kommer man till en identisk sida för sökning om man föredrar att söka genom att använda ryska och det kyrilliska alfabetet.



I layout-sidan *activty_selected_word.xml* valde jag en öppen scrollbar design utan meny-frame. Det ger en friare och öppnare känsla eftersom den delen ska kunna innehålla en större textmassa. Dessutom ville jag använda en responsiv struktur på min design, som inte begränsar appens praktikalitet. Hemknappen, om man inte vill använda telefonens ”tillbaka-knapp”, finner man flytande strax under där textområdet slutar.



Det var viktigt för mig att man inte bara fick fram en direkt översättning på termen, utan även skulle få en kort beskriving av t.ex. organet, sjukdomen eller kroppsdelen som var översatt. Vill man sedan läsa mer finns länken till den fullständiga artikeln under en kort sammanfattning.

Sökfältet är placerat högst upp i app-fönstret och kan snabbt nås genom den vita förstoringsglas-ikonen. När man trycker på den försvinner app namnet för att ge användaren fullt fokus på det hen skriver in i sökfältet.

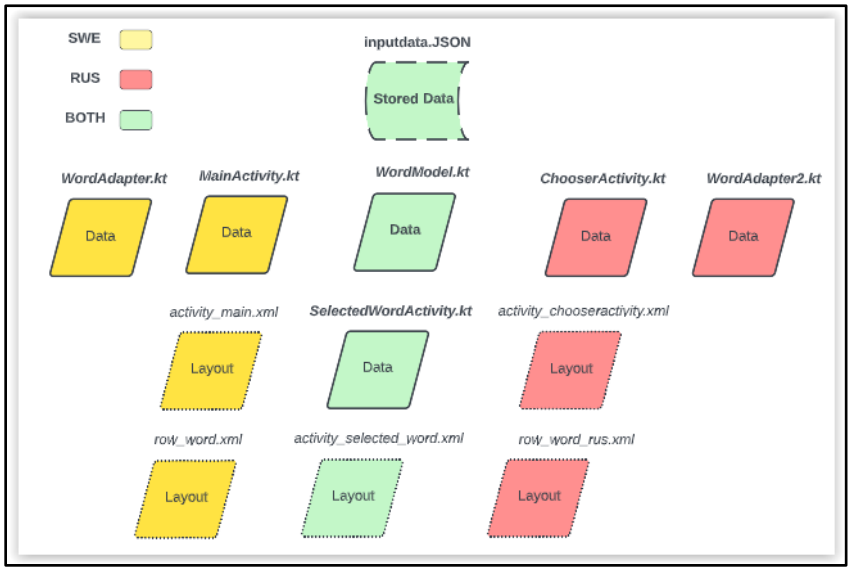
Implementation

Eftersom jag är nybörjare inom Kotlin så ville jag ha så mycket kontroll som möjlig över utvecklingsmiljön. Av praktiska skäl så valde jag därför att arbeta med datan via en lokal JSON-fil. Från början hade jag tänkt att man skulle kunna extrahera data direkt från mina Exceldokument, men metoderna för extrahering och serialisering av data i Excel-dokumentets celler, rader och kolumner verkade alltför komplicerade i dagsläget.

JSON är ett filformat jag är bekant med sedan tidigare (*Azure*, *DevOps, Node.js*), och är ett bra val när man behöver lagra en relativt liten uppsättning nyckel/värde-par sparar. Man sparar t.ex. utrymme eftersom JSON-strängar kan lagras i en enda kolumn. Jag ville strukturera appen på det sättet att all information hämtas från en fil, för att göra den mer flexibel och lättare att underhålla/uppdatera.

Ett annat val jag gjorde var att separera koden för svenska och ryska översättningen. Jag slutförde alltså först den ”svenska” delen av appen från där jag tar emot data från JSON-filen i *MainActivity.kt*, tills datan visas i **RecyclerView** med implementerad **SearchView**. Därefter kopierade jag och återanvände helt enkelt koden för den ryska delen, där jag t.ex. spegelvände strukturen layoutfilerna för rows i **RecyclerView** (*row_word.xml*/*row_word_rus.xml*). Jag valde också att ska två separata adaptrar (*WordAdapter.kt*/*WordAdapter2.kt*).

Nedanstående bild visar hur appens dataflöde är uppdelad i tre delar: 1. Svensk översättnings del, 2. Rysk översättnings del och 3. en gemensam som hanterar data oavsett översättningsspråk.



Det kändes logiskt att dela upp programkoden på detta sätt, även om det kanske är praktiskt möjligt att göra på något annat sätt i Kotlin. Som bilden visar passerar datan gemensamt genom 1/3 av filerna: *inputdata.JSON*, *SelectedWordActivity.kt*, *activity_selected_word.xml* och *WordModel.kt*.

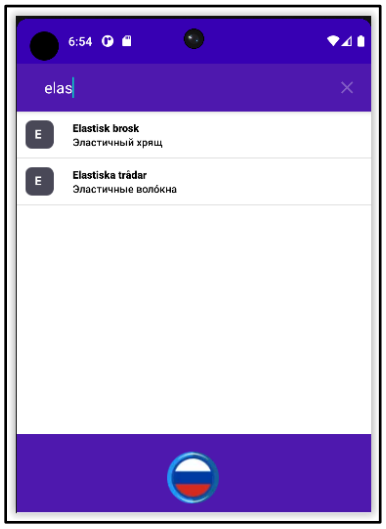
Jag kommer fortsättningsvis att utgå ifrån det för det svenska dataflödet av praktiska skäl. Appens arbetsflöde startar med funktionen **readJSONDataFromFile()** som läser in JSON-data från *inputdata.json* via *inputStream*, efter vilket ett String-objekt (*jsonDataString*) skapas. I funktionen **addItemFromJSON()** delas det upp i *itemObject* och skickas till klassen *WordModel* (*WordModel.kt*) där de konverteras till dataobjekt som därefter placeras i array-*wordModelList*.
I slutet på denna metod finns också sorteringsfunktionen (bokstavsordning) som används av **RecyclerView**, dvs. **Collections.sort()**.

```
private fun addItemFromJSON() {
    try {
        val jsonDataString = readJSONDataFromFile()
        val jsonArray = JSONArray(jsonDataString)
        for (i in 0 until jsonArray.length()) {
            val itemObject = jsonArray.getJSONObject(i)
            val Svenska = itemObject.getString( name: "Svenska")
            val Ryska = itemObject.getString( name: "Ryska")
            val ImgURL = itemObject.getString( name: "ImgURL")
            val SvenskaKort = itemObject.getString( name: "SvenskaKort")
            val RyskaKort = itemObject.getString( name: "RyskaKort")
            val LinkSV = itemObject.getString( name: "LinkSV")
            val LinkRY = itemObject.getString( name: "LinkRY")
            val model =
                WordModel(Svenska, Ryska, ImgURL, SvenskaKort, RyskaKort, LinkSV, LinkRY)
            wordModelList.add(model)
            Collections.sort(
                wordModelList,
                Comparator.comparing { obj: WordModel -> obj.ordSvenska })
        }
    } catch (e: JSONException) {
        Log.d(ContentValues.TAG, msg: "addItemFromJSON: ", e)
    } catch (e: IOException) {
        Log.d(ContentValues.TAG, msg: "addItemFromJSON: ", e)
    }
}
```

Java.io är ett s.k. markörsgränssnitt, vilket betyder att den inte behöver implementera några metoder eller fält för serialisering.
Det framstod som en smidig lösning för att konvertera JSON-objekten till data-objekt. Klassen *WordModel* (*WordModel.kt*) importerar av den anledningen paketet *java.io.Serializable*.

Adaptern i W*ordAdapter.kt*, tar sedan emot de konverterade och sorterade dataobjekten i *wordModelList*. Genom funktionen **getFilter()** kontrolleras om någon av bokstäverna som användaren matar in i **SearchView** förekommer i *searchStr* genom funktionerna **wordModel.getOrdSvenska()**, **wordModel.getOrdRyska()**.

Om så är fallet filtreras alla övriga objekt bort från listan och en ny objektlista i *wordModels* visas i **RecyclerView** gränssnittet via **publishResults().**



Det var inte helt lätt att följa dokumentation/exempel för att programmera sökfunktionen i *MainActivity.kt*, eftersom syntaxen för metoderna **onCreateOptionsMenu()** och **onOptionsItemSelected()** nyligen verkade genomgått vissa förändringar. Sorteringen som implementerades *i* ***addItemFromJSON().,*** dvs. **Collections.sort()**, var dock oväntad okomplicerad att använda.

Metoderna **onCreateViewHolder()** och **onBindViewHolder()** skapar och initierar ViewHolder och dess associerade **Views**. Den sistnämnda binder associerade data till **ViewHolder**, dvs *ordPrefix*, *ordRyska* och *ordSvenska* som utgör listan av de för användaren valbara/sökbara ord i **RecyclerView** (*rvWords2*). I klassen *WordAdapterVh* binds slutligen **ViewHolder** och dess objekt till respektive **View** i layout-filerna.

```
public static class WordAdapterVh extends RecyclerView.ViewHolder {
    private final TextView ordPrefix;
    private final TextView ordRyska;
    private final TextView ordSvenska;

    public WordAdapterVh(@NonNull View itemView) {
        super(itemView);
        ordRyska = itemView.findViewById(R.id.tvRyska );
        ordSvenska = itemView.findViewById(R.id.tvSvenska );
        ordPrefix = itemView.findViewById(R.id.tvPrefix );
    }
}
```

UserClickListener gör orden klickbara och skickar den valda informationen som *extras* till *SelectedWordActivity.kt* genom funktionen **selectedWord(wordModel: WordModel)** i *MainActivity.kt*.

I *SelectedWordActivity.kt* finns egentligen bara en enda funktion. Den tar emot Intent-extras från *MainActivity.kt* och placerar ut dem i respektive **View** i *activity_selected_word.xml*. **Glide** (*image loading library*) var oväntat enkel att implementera, då man med en enda rad kunde ladda in bilderna via URL från Internet till **TextView**: *tvLinkSV* i *activity_selected_word.xml*.

```
if(intent != null){

    wordModel = (WordModel) intent.getSerializableExtra( name: "data");
    String selectedWord = wordModel.getOrdSvenska() +" / "+ wordModel.getOrdRyska();
    String svenskaKort = wordModel.getSvenskaKort();
    String ryskaKort =wordModel.getRyskaKort();
    String imgURl = wordModel.getImgURL();
    String linkSV = wordModel.getLinkSV();
    String linkRY = wordModel.getLinkRY();

    Glide.with( activity: this).load(imgURl).centerCrop().into(imageView);
    tvSelectedWord.setText(selectedWord);
    tvSvenskaKort.setText(svenskaKort);
    tvRyskaKort.setText(ryskaKort);
    tvLinkSV.setText(linkSV);
    tvLinkRY.setText(linkRY);


}
```

Slutsats

Det kändes naturligt för mig att jobba lokalt med appen/ordlistan under utvecklingsarbetet, men i framtida versioner vill jag hämta in data via Internet. Om fler personer ska använda appen skulle det naturligtvis vara praktiskt att jobba med/uppdatera ordlistorna på en central server, för att sedan distribuera ut uppdaterade versioner kontinuerligt.

Genom implementation av makron i Excel, skulle man också öka användarvänligheten vid uppdatering/underhåll. T.ex. så skulle ordlistorna kunna sparas direkt i JSON medan man arbetar med dem i Excel-filer. Det är fullt möjligt att jag I framtiden kommer att använda CSV/Excel direkt, när jag lärt mig mer om serialisering och extraherande av information från Exceldokument.

Jag bestämde mig för att i detta projekt ladda in bilder och länkar till appen främst från *Wikipedia*. Det skulle man kunna tänka sig att ändra sedan, med hänsyn till t.ex. Copyright.

Speech-to-text valde jag att inte använda mig av, delvis för att det inte fanns tidsutrymme för att sätta sig in det under denna kurs. Om man dessutom utgår från en tolksituation är jag inte helt övertygad om att det är en passande funktion då det kan uppstå viss förvirring om tolken plötsligt börjar ”prata” med sin telefon.

Jag upplevde det som oväntat svårt att komma in i Kotlin som programmeringsspråk. Det var inte lätt att navigera sig fram bland online-resurserna. Det känns som att Kotlin är under ständig snabb utveckling och källor/tutorlials/exempel som hade något år på nacken var inte länge tillämpbara. Jag hade t.ex. svårt att hitta exempel på hur man programmerar mer komplexa funktioner med hjälp av **ViewModels**.

RecyclerView visade sig vara ett utmärkt verktyg för att visa, sortera och söka i ordlistan. Även om det inledningsvis var lite knepigt att förstå hur man skulle koppla den till adaptern och **SearchView/Action bar**.

Jag var förvånad hur lätt det var att integrera och presentera spåkspecifika tecken som t.ex. å,,ä,ö och den kyrilliska alfabetet. Allt verkade vara inbyggt i den allmänna serialiseringsprocessen, och inga extra programmeringssteg krävdes.

Android Studio var bra på att ge förslag under programmeringen, och erbjöd en ganska trevlig utvecklingsmiljö. Syntaxen kunde dock vara lätt förvirrande och verkar vara under ständig utveckling Jag gillade skarpt kopplingen till Github, som fungerade lysande och var mycket smidigare än den som finns i *Visual Studio Code*.

Nackdelen var dock att den ibland kändes ”buggig” och alltför ofta var man tvungen att spara och kompilera 2–3 ggr innan den accepterade uppdatering och uppstart av sin *virtual device*. Det var heller inte lätt att sätta sig in i när och hur det krävdes implementering av olika bibliotek i **build.gradle**.

Möjligheten att byta mellan kod vy och design vy när man arbetade med layout-filerna var en mycket praktisk feature. På det sättet kunde man enkelt sedan duplicera och multiplicera sin design till flera XML-filer.

Jag är ganska nöjd med den slutliga UI-designen, men kanske skulle man behöva genomföra användartester för att se om designen kan göras bättre/tydligare.

Överlag är jag ganska imponerad av Android Studio som gratis utvecklingsverktyg, och förhoppningsvis kommer tillgången på aktuella guider/tutorials och exempel bara att öka i framtiden.