Bill Monitor - From idea to publication, or the birth of an Android application

Roland Cseh [a]

[a] *Pentatrade Kft., Sefőződombi dülő 3, Győr, 9028, Hungary, cseh.roland@pentatrade.hu*

Abstract

In the article entitled Bill Monitor - From idea to publication, or the birth of an Android application, I would like to show what it takes for someone to create a complex, independent, unique program for the Android platform. The program I am implementing is not completely new, because a similar application is already available in the Android application store, the Google Play Store. A program that is so user-friendly, takes into account the peculiarities of Hungarian mobile phone service providers and is detailed, understandable and customizable has yet to be found. Bill Monitor, as the name suggests, will be a bill monitoring application that will help the user keep control of his mobile bill or the balance of his prepaid card. The user provides the important data for the program (e.g.: Monthly fee, Minute fee, SMS fee, etc.), then the program calculates the current total amount of the user's phone bill using this data and the phone's call log and SMS storage, or the current card balance. I will introduce what Android is. I will describe what is needed to develop an application. I design the software and explain in detail how I implemented certain parts of the program, why they are responsible for each one. I will describe how to publish an application in the Play Store and finally explain what my plans are for the future.

Keywords: programming; Android; Google

Bill Monitor – Ötlettől a publikálásig, avagy egy Android-os alkalmazás születése

Cseh Roland [a]

[a] *Pentatrade Kft., Sefőződombi dülő 3, Győr, 9028, Magyarország, cseh.roland@pentatrade.hu*

Absztrakt

A cikkben melynek a címe Bill Monitor – Ötlettől a publikálásig, avagy egy Android-os alkalmazás születése, azt szeretném megmutatni, mi is kell ahhoz, hogy valaki egy komplex, önálló működésű, egyedi programot hozzon létre Android platformra. A program, amit megvalósítok, nem teljesen új keletű, mert hasonló alkalmazás már található az Android alkalmazásboltjában, a Google Play Áruházban. Olyan programot, ami ennyire felhasználóbarát, figyelembe veszi a Magyarországi mobil-telefonszolgáltatók különlegességeit és részletesen, érthetően, személyre szabható még nem található. A Bill Monitor, mint a neve is mondja, egy számlafigyelő alkalmazás lesz, ami segít a felhasználónak kordában tartani a mobilszámláját, vagy épp a kártyás feltöltött egyenlegét. A felhasználó a program számára megadja a fontos adatokat (pl.: Havidíj, Percdíj, SMS díj, stb.), majd a program ezen adatok és a telefon hívásnaplója, valamint SMS tárolója segítségével, kiszámolja mennyi is jelenleg a felhasználó telefonszámlájának a végösszege, vagy az éppen aktuális kártyás egyenlege. Bemutatom mi is az az Android. Leírom, mi szükségeltetik egy alkalmazás fejlesztéséhez. Megtervezem a szoftvert és részletesen elmagyarázom miként valósítottam meg a program egyes részeit, azok miért felelősek egyenként. Leírom, miként lehet publikálni egy alkalmazást a Play Áruházban és végül ismertetem, milyen terveim vannak a jövőre nézve.

Kulcsszavak: programozás; Android; Google

1. Bevezető

Miért pont az Android rendszert választottam? Egyszerű a kérdés. Mindig is rajongtam az okos kütyükért. A legelső okostelefont az életemben még 16 évesen vásároltam mely egy Symbian rendszerrel szerelt készülék volt. Már abban az időben elhatároztam, hogy ha lesz elegendő időm és kitartásom, akkor egy programozási nyelvet megpróbálok a képességeimhez mérten a lehető legjobban elsajátítani, és alkotni a segítségével valami egyedit, ami segíti a készülék használóját a mindennapokban. Ahogy teltek az idők, úgy cserélődtek a készülékek is. Így szereztem be magamnak én is egy Android operációs rendszerrel működő okostelefont. Az első pillanatoktól fogva éreztem, hogy ennek a rendszernek még nagy jövője lesz. Amikor még nem volt ennyire elterjedt ez a platform, már akkor elkezdtem saját alkalmazás készítésével foglalatoskodni. Mivel még nem volt meg a kellő elszántság és talán a megfelelő mennyiségű szabadidőm se, így nem jártam sikerrel. Egy évvel később viszont újra megpróbáltam és nem adtam fel. Az újonnan szerzett programozási tapasztalataim és a magabiztosságom növekedése meghozta a várt sikert. Megszületett az ötlet, mely alapján a cikket és a programomat elkészítettem. A készítés egyben tanulás is volt számomra, ugyanis ezzel az alkalmazással sajátítottam el az Android programozás legfontosabb fortélyait.

A program ötlete onnan ered, hogy én mindig is szerettem arra figyelni, hogy a nehéz munkával megkeresett vagy éppen szüleim által megkeresett pénzt mire is költöm el. Ez igaz a mobiltelefon használatomra is. Ezért próbáltam egy olyan megoldással előállni, mely segítségével egy Magyarországon élő ember, aki rendelkezik Android-os készülékkel, könnyedén le tudja ellenőrizni, hogy jelenleg mennyi is éppen az aktuális egyenlege. A magyar szolgáltatók ennek ellenőrzésére biztosítanak lehetőséget, de csak online, azaz internetes felületen, melyhez természetesen internetkapcsolat szükséges. Az én alkalmazásom viszont a készülékben található adatok és a megadott adatok alapján számolja ki mindezt, ami könnyedén, gyorsan és értelmezhető formában jelenik meg a felhasználó számára. Az ötlet nem teljesen új, viszont az, hogy csak Magyarországra kell koncentrálnom az elkészítés során, lehetőséget biztosít arra, hogy a magyar mobiltelefon szolgáltatók sajátosságait is bele tudjam rakni a programba.

1.1. Mi is az az Android?

Az Android szó jelentése az idegen szavak szótára szerint: emberszerű robot. Ez a megállapítás nem is áll olyan messze attól, amit valójában a mai emberek számára jelent: egy zöld kis robot, mely az emberek okostelefonjainak legalább a 70%-án megtalálható. És ez miért van így? Mert

a világon a jelenleg legnépszerűbb mobiloperációs rendszer az Android. Könnyű kezelni, sokféle készüléken megtalálható, remekül személyre szabható és nyílt forráskódú.

„Az Android egy Linux kernelt használó mobil operációs rendszer, elsősorban érintőképernyős mobil eszközökre (okostelefon, táblagép) tervezve." A fejlesztést az Android Inc. kezdte meg hagyományos, nem érintőképernyős telefonokra. 2005-ben a Google először csak anyagilag támogatta, majd később fel is vásárolta a céget. Az első nagy bejelentésre 2007 novemberéig kellett várni. Ekkor jelentették be az Open Handset Alliance konzorciumot és azt, hogy gőzerővel dolgoznak az operációs rendszeren, mely nyílt forrású és Linux kernel alapokon nyugszik majd. A szövetség tagjai között vannak telefongyártók, alkalmazás fejlesztők, mobiltelefon szolgáltatók és chipkészítők is. Az első készülék melyen ez a rendszer futott, már érintők jelzővel volt szerelve és 2008.október.22-én lett elérhető a nagyközönség számára. A HTC gyártotta a T-Mobile-l karöltve és az Android 1.0-ás verziója futott rajta. Ez a készülék nem igazán váltotta be a hozzá fűzött reményeket a nehézkes kezelése miatt és csak nagyon kis rétegben tudott sikereket elérni. Az áttörést az 1.5-ös verzió megjelenése hozta el az Android számára. Erre 2009 áprilisában került sor. Horton, J. (2015), Griffiths, D., & Griffiths, D. (2017)

2. Szoftver követelmények

A szoftverfejlesztés első lépései a feladatspecifikáció és a követelmény feltárás. Sándor, S., & László, V. (2001) A következő fejezetekben megpróbálok azokra a kérdésekre válaszolni, hogy mit kell tudjon az alkalmazásom, milyen problémákat kell megoldanom és ezeknek milyen követelményei vannak.

2.1. Feladatspecifikáció

A legfontosabb funkciók, melyeket meg kell valósítani:

- Előfizetéses vagy felöltőkártyás "számla" követés
- Számlázási időszak megfelelő követése (minden előfizetésnek van egy hónapforduló napja)
- Ingyen percek, ingyen SMS-ek kezelése
- Ingyen hívható telefonszámok és ingyen SMS számok kezelése
- Hálózaton belüli és kívüli perc / SMS díj, ingyen perc, ingyen SMS követés
- Időszakok kezelése (csúcsidő és ingyen telefonálási időszak)
- Telefonadóval való számítás
- Havi limit / Egyenleg limit szolgáltatás
- Számlaértesítő funkció (Előfizetéseseknek)
- Egyenlegfeltöltés kezelő (Feltöltőkártyásoknak)

- Egyéb havi költségek kezelése
- Szép, egyszerű és HOLO témájú kezelőfelület az Android Design Irányelvek alapján
- Többféle HOLO téma
- Egységes kinézet 2.3 és 4.x-es Android-on ActionBarSherlock segítségével
- Részletes beállítási lehetőségek ÉRTHETŐEN
- Widget és/vagy értesítés a jelenlegi állapotról, mely automatikusan frissül hívás és/vagy SMS után
- Roaming értesítés
- Program eseményeinek megfelelő naplózása

2.2. Követelmény feltárás

A szoftver használatához olyan eszközre van szükség, mely rendelkezik a következő hardveres és szoftveres követelményekkel:

Hardveres követelmények

Az eszköz legyen képes futtatni az Android operációs rendszert úgy, hogy tegyen eleget minden olyan hardveres követelménynek, melyek szükségesek az Android futtatásához. Ezen felül rendelkezzen a telefonáláshoz szükséges összes hardverelemmel (rádió modul, SIM kártyaolvasó, antenna stb.) és legyen képes kapcsolódni a mobiltelefon hálózatokhoz. Erre azért van szükség, mert különben az alkalmazás elveszíti az értelmét, ha egy olyan eszközön használnánk, ami nem is képes a telefonálási funkciók használatára.

Szoftveres követelmények

Az eszközön fusson legalább a 2.3-as verziójú Android, mely rendelkezik hívásnaplóval és SMS tárolóval, melyek tartalmaznak bejegyzéseket is.

3. Az alkalmazás megtervezése

A specifikációk és az alapfeltételek megismerése után jöhet a tervezés Sándor, S., & László, V. (2001).

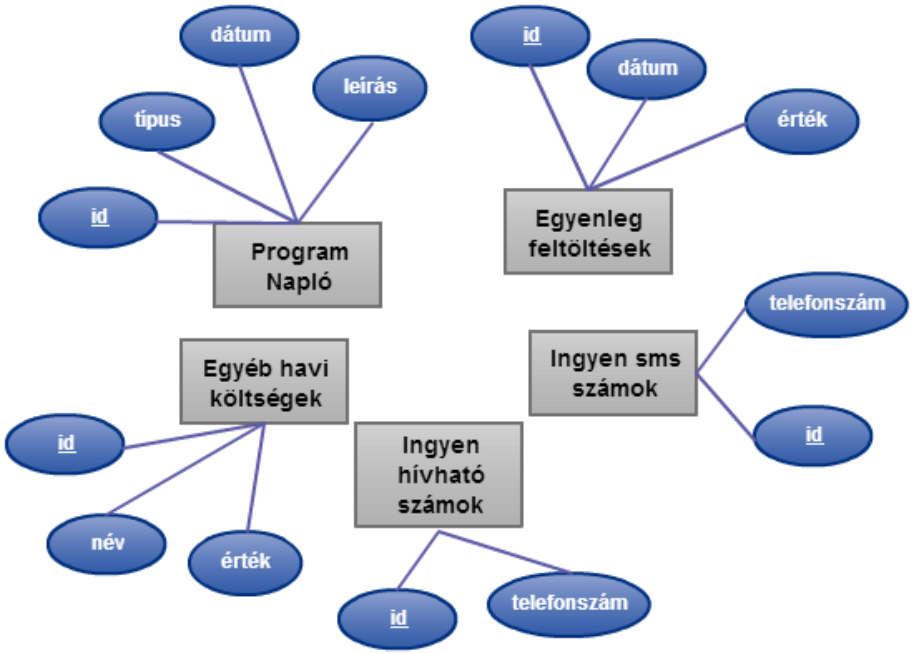
3.1. Adatbázis megtervezése

A feladatspecifikációban ismertetett funkciókból látszódik, hogy a megfelelő adattárolás szempontjából ajánlott lesz az adatbázis használata. A következő funkciók miatt kell adatbázist létrehozni:

- Ingyen hívható telefonszámok és ingyen SMS számok kezelése
- Egyéb havi költségek

- Egyenlegfeltöltés menedzser
- Eseménynapló

Minden funkciónak külön adattáblát csinálok. Közöttük kapcsolat kialakítására nincs szükség.



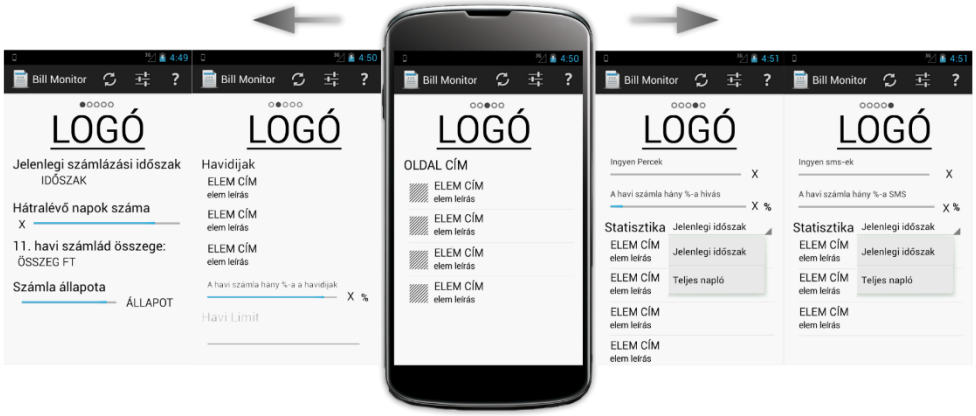
1. ábra: Az adatbázis E/K modellje

Az Android rendszer lehetőséget biztosít arra, hogy minden alkalmazás fájl alapú adatbázist hozzon létre, melyet csak önmaga ér el. Célszerű ezzel megvalósítani az előbb tervezett adatbázist.

3.2. Felhasználói felület megtervezése

A felhasználói felület tervezése során nagyon fontos szempont volt számomra, hogy kövessem az Android készítői által kiadott Design Irányelveket, mely segítségével hatékony és szép programot leszünk képesek készíteni, az Android alapértelmezett témájának a segítségével is.

Az alkalmazás fő felületének egy könnyen értelmezhető, informatív, egységes és szép designt terveztem, mely lapozhatóságának köszönhetően sokféle információ megjelenítésére képes anélkül, hogy el kellene onnan navigálni. A panoráma kép a főképernyőről a 2. ábrán látható.

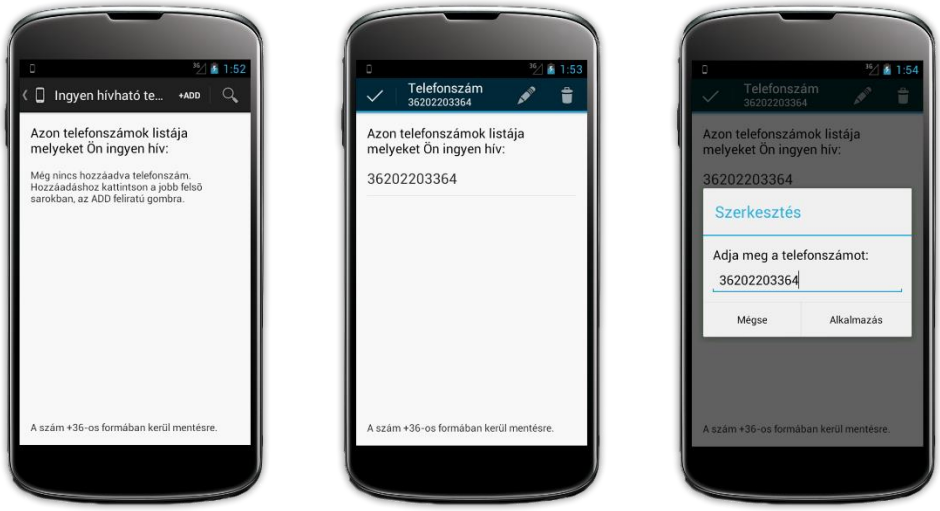


2. ábra: Panoráma kép a főképernyőről.

A Naplókat megjelenítő activity-nek egy nagyon letisztult, mégis könnyen kezelhető felületet terveztem. Egy lenyíló menü segítségével lehet kiválasztani a megfelelő kategóriát. A Naplók képernyőtervét a 3. ábra ismerteti.

3. ábra: Naplók képernyőterve.

A beállításokon belül találhatóak majd olyan részek, melyeket egyénileg tervezett képernyővel szeretnék megvalósítani. Ezek közül az egyik ilyen az ingyen hívható telefonszámok kezelésére szolgáló képernyő. Az Ingyen hívható telefonszámok képernyőterve a 4. ábrán látható.

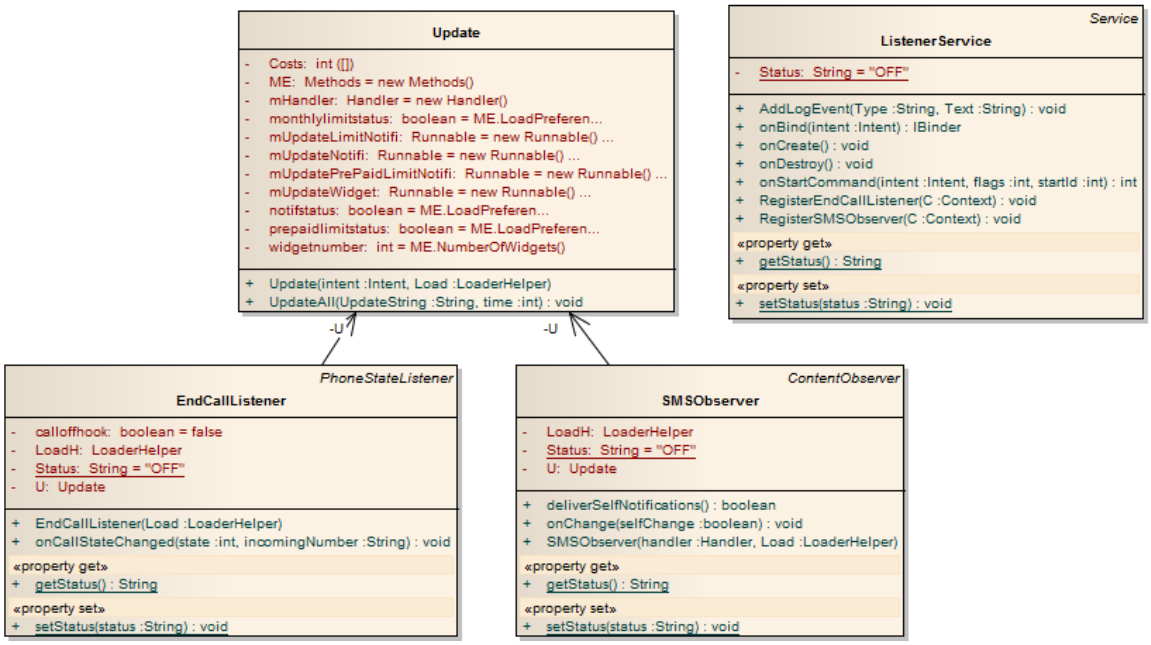


4. ábra: Ingyen hívható telefonszámok képernyőterve

Az Android rendszer lehetőséget biztosít arra, hogy minden alkalmazás fájl alapú adatbázist hozzon létre, melyet csak önmaga ér el. Célszerű ezzel megvalósítani az előbb tervezett adatbázist.

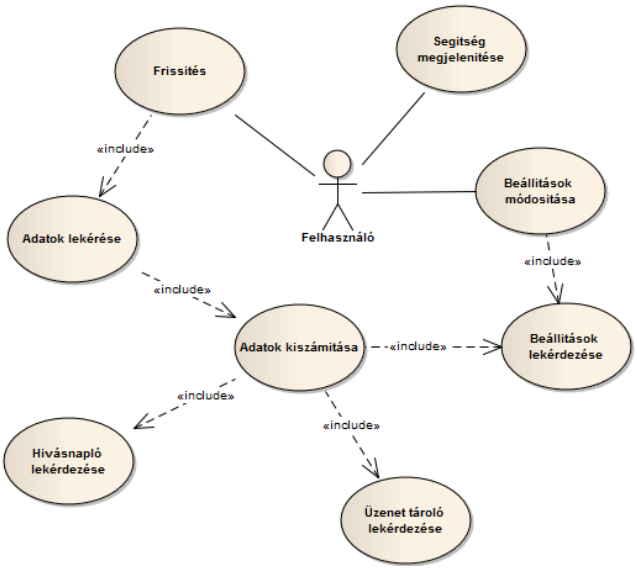
3.3. Diagramok

Az osztálydiagram a *Listener* csomagban található osztályokat ábrázolja. Az 5. ábra illusztrálja.



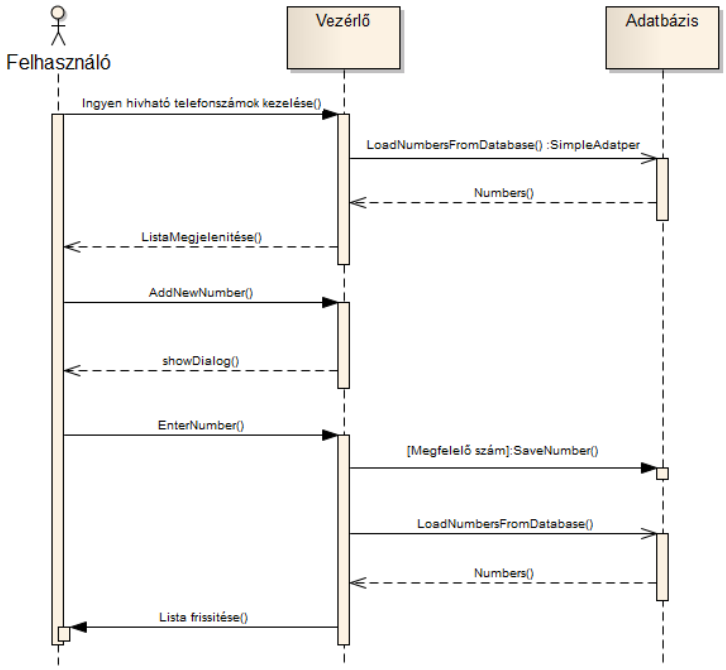
5. ábra: Az osztálydiagram.

A Use Case (Használati eset) diagramot, azért soroltam statikus diagramnak, mert véleményem szerint egy pillanatnyi állapotot szemléltet. A 6. ábra mutatja be.



6. ábra: Használati eset diagram.

A szekvencia diagram a 7. ábrán látható.



7. ábra: A szekvencia diagram.

4. Az alkalmazás implementálása és integrálása

Ebben a fejezetben bemutatott forráskódok helymegtakarítás céljából csak a legfontosabb műveleteket tartalmazzák. Az implementálás sikeres megvalósításához a következő lépésekre és eszközökre volt szükségem:

- Az ADT, SDK, AVD előkészítése, új projekt létrehozása
    - Emulátor létrehozása és beállítása
    - Készülék beállítása tesztelésre

4.1. Az alkalmazás fő activity-je

Az alkalmazás legfőbb activity-je, amely megjeleníti a megfelelő kezelőfelület, és garantálja a program megfelelő működését.

Kinézete egy `LinearLayout`, mely áll egy `CirclePageIndicator`-ból és egy `ViewPager` elemből. Használ *Action Bar* elemeket, és *Options Menu*-t is tartalmaz. Indítás után, különféle dialogok használatával jelenít meg információkat, segédletet a felhasználó számára. Ezek közül, az activity-re jellemző dolgok megvalósítását részletesen leírom.

Activity layout

Az activity elrendezését a *res/layout* mappában egy xml fájl segítségével hoztam létre.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical"
        android:paddingTop="5dp" >
        <billmonitor.insdeveloper.ViewPager.CirclePageIndicator
                android:id="@+id/CircleMainIndicator"
                android:layout_height="wrap_content"
                android:layout_width="fill_parent"
                android:paddingTop="10dp"
                app:radius="4dp"
                app:strokeWidth="2dp" />
        <android.support.v4.view.ViewPager
                android:id="@+id/viewPagerMain"
                android:layout_width="fill_parent"
                android:layout_height="0dp"
                android:layout_weight="1" />
</LinearLayout>
```

Az elrendezés nagyon egyszerű. Egy `LinearLayout`, mely egymás alatt tartalmazza a `CirclePageIndicator`-t, és a `ViewPager`-t.

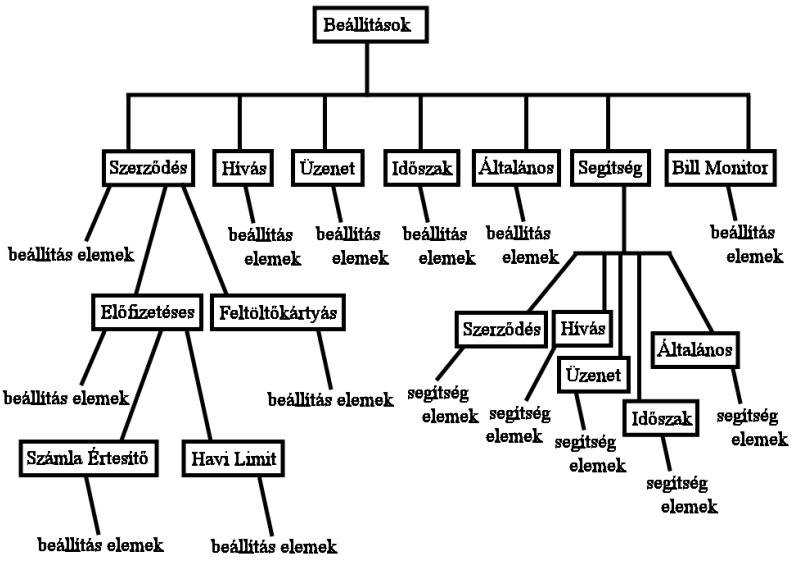
4.2. Start Wizard

Ennek az activity-nek a segítségével egy olyan felületet valósítok meg, mely csak a legelső indítás esetén indul el és jelenít meg tartalmat. Célja, egy olyan indítási segédlet megvalósítása, mely üdvözli a felhasználót, röviden leírja a program lényegét, és segít neki a kezdeti beállításokat elvégezni. Érdekessége, hogy valójában minden alkalmazás indítás estén ez az activity indul el, de csak akkor jeleníti meg az indítási varázslót, ha nem található beállítás fájl. Ellenkező esetben elindítja a MainActivity-t. Exportált beállításokat képes észlelni az SD kártyán, és megkérdezi a felhasználót, hogy szeretné-e a régi beállításait visszamásolni. Ezzel is megkönnyítve a felhasználó dolgát, ha esetleg újra kellene telepíteni az alkalmazást.

A layout-ja egy `LinearLayout`, mely mindösszesen 2 gyereket tartalmaz. Egy ViewPager-t és egy `ViewPagerIndicator`-t ami pontosabban egy `UnderLinePageIndicator`. Nagyon hasonló az előző fejezetben ismertetetthez. A ViewPager létrehozását szintén kifejtettem már, így most ezt nem ismételném meg.

4.3. Beállítások Activity

Az alkalmazásom rengeteg testre szabási lehetőséget biztosít. Ennek a könnyű megvalósíthatósága érdekében készítettem egy `PreferenceActivity`-t, mely segítségével a felhasználó egy felületen, kategorizálva, meg tudja jeleníteni a beállításokat. Első lépésként megtervezem, hogy milyen beállításokat szeretnék készíteni és, hogy ezeket milyen kategóriákba fogom majd sorolni. A 8. ábra az alkalmazásom beállítások hierarchiájáról:



8. ábra: Az alkalmazás beállítások felépítése.

4.4. Értesítés / Widget

Az alkalmazásomban fontos szerepet kapott ez a két funkció, így részletesen leírom miként is valósítottam meg ezeket.

Értesítés létrehozása és vezérlése

Nem túl bonyolult egy értesítést létrehozni. Igazából majdnem ez volt az első komolyabb dolog, amit megtanultam elkészíteni. A 3.0-ás Android bevezetésével még könnyebb dolga van a fejlesztőnek, ugyanis bevezették a `Notification.Builder` osztály-t, aminek a segítségével könnyedén lehet értesítést létrehozni. A régebbi Android verziók támogatatására is gondoltak. Megalkották a `NotificationCompat.Builder` osztály-t, mely a *Support Library* része és működése megegyezik az eredeti osztály működésével.

Első lépésként egy véglegesített statikus integer változót készítettem, mely az értesítés egyedi azonosítója lesz.

```
private static final int NOTIF_NUMBER = 1000;
```

Ezután létrehoztam a metódust, ami elkészíti és meg is jeleníti az értesítést.

```
public void createNotification(Activity A) {

      NotificationManager nManager = (NotificationManager)
      A.getSystemService(Context.NOTIFICATION_SERVICE);

      Intent notificationIntent = new Intent(context,
      BillMonitorActivity.class);

      PendingIntent pendingIntent = PendingIntent.getActivity(this,
      0, notificationIntent, 0);

      Notification notif = new Notification.Builder(Context)
            .setSmallIcon(R.drawable.ic_launcher)

            .setTicker(”Értesítés!”)

            .setContentTitle(”Minta-Alkalmazás”)

            .setContentText(”Önnek 2 új üzente érkezett”)

            .setContentIntent(pendingIntent)

            .build();

      nManager.notify(NOTIF_NUMBER, notif);

}
```

Első lépésben a rendszertől kérni kell az értesítés szolgáltatást, melyet a `getSystemService(Context.NOTIFICATION_SERVICE)` metódus segítségével teszek meg. Ez a függvény csak activity-ben vagy olyan környezetben használható, ami megegyezik egy activity környezetével. Ezután készítek egy `pendingIntent`-et. A `pendingIntent` egy olyan intent, mely feljogosít arra egy másik alkalmazást, vagy egy alkalmazás komponenst, hogy későbbi időpontban indítsa el az intent-et. Értesítés esetén ez akkor történik meg, ha a felhasználó megérinti azt. Ezután nincs más dolgom csak kiadni a `NotificationManager` objektum `notify(int, Notification)` parancsát, a megfelelő paraméterekkel. Az első paraméternek megadott szám lesz a második paraméterben megadott értesítés azonosítója. A metódus végrehajtása után megjelenik az értesítés, melyet alapértelmezetten nem követ hang és rezgés, és szabadon törölhető. Az értesítés viselkedésének módosításához jelző *flag*-eket és alapértelmezéseket kell beállítani a következő módon:

```
notif.flags |= Notification.FLAG_ONGOING_EVENT;
notif.defaults |= Notification.DEFAULT_VIBRATE;
```

Hang megadásához szükségünk lesz egy `RingtonePreference`-re, melynek az elmentett értéke egy olyan `string` lesz, melyet `URI`-ként egyenlővé tudunk tenni a saját `Notification` objektumunk `sound` változójával.

```
notif.sound =
      Uri.parse("content://settings/system/notification_sound");
```

Végül pedig megírtam azt a metódust, amivel lehet törölni egy előzőleg megjelenített értesítést, ha tudjuk az azonosítóját:

```
public void killNotification(Activity A, int notif_num) {
      NotificationManager nManager = (NotificationManager)
      A.getSystemService(Context.NOTIFICATION_SERVICE);
      nManager.cancel(notif_num)
}
```

A `NotificationManager cancel(int)` metódusa végzi a törlést.

Az értesítésnek lehet saját kinézetet is megadni. Ehhez nincs más dolgom csak előre elkészíteni egy layout-ot *xml*-ben vagy a programkódon belül, és utána a `Notification.Builder` alkalmazása során a következő sort is beilleszteni:

```
RemoteViews notifview = new
RemoteViews(context.getPackageName(),R.layout.basenotif);
```

```
...
.setContent(notifview)
...
```

Jelly Bean vagy újabb Android esetén lehetőség van kiterjesztett nézet létrehozására is. A programom során ezt is alkalmazom, természetesen csak abban az esetben, ha az eszközön futó Android verzió 4.1 vagy annál nagyobb. A `bigContentView(RemoteViews)` segítségével tudok megadni kiterjesztett egyedi kinézetet az alkalmazás számára. Vannak beépített kiterjesztett nézetek is, melyek használatához először egy új értesítés kell létrehozni, és azon belül készíteni egy `Notification.Builder(Context)`-t. Például:

```
Notification notif = new Notification.BigTextStyle(
      new Notification.Builder()
         .setContentTitle("New mail from " + sender.toString())
         .setContentText(subject)
         .setSmallIcon(R.drawable.new_mail)
      .bigText(aVeryLongString)
      .build();
```

Ezzel az értesítések készítését bemutattam. A kész értesítés kódok megtalálhatók a *Methods.java* fájlban a `triggerNotification`, `triggerLimitNotification`, `triggerPrePaidLimit`, `triggerReminder`, `triggerRoaming`, `killRoamingNotification` és `killNotification` metódusokban.

Widget létrehozása

Widget-et készíteni egy picivel nehezebb, mint értesítést, de úgy gondolom, hogy ez se okoz sokkal nagyobb gondot.

Első lépésként készítettem egy *widget_provider.xml* fájlt a *res/xml* mappában. Ez a fájl írja le a widget alapvető tulajdonságait, viszont a kinézetét nem tartalmazza.

```
<appwidget-provider
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:minWidth="240dp"
    android:initialLayout="@layout/loading"
    android:minHeight="60dp"
```

```
    android:resizeMode="horizontal|vertical"
    android:widgetCategory="keyguard|home_screen"/>
```

Az én widget-em nem rendelkezik pontos magasság és szélesség értékekkel, épp azért, hogy szabadon méretezhető legyen. Ebben az esetben, ha megadok egy minimum szélesség (`android:minWidth`) és magasság (`android:minHeight`) értéket, akkor annál csak nagyobbra lesz képes méretezni a felhasználó. Az `android:initialLayout` attribútum segítségével, a widget-nek azt a felületét adom meg, amit rögtön a kihelyezés után szeretnék, ha a rendszer megjelenítene. Általában ez olyan felület, mely arra hívja fel a figyelmet, hogy a widget még betöltés alatt van. Erre azért van szükség, mert egy widget úgy működik, hogy a kihelyezést követően a rendszer kiküld egy intent-et, melyet az arra alkalmas `AppWidgetProvider` osztály az alkalmazásban feldolgoz, és egy frissítés után megjeleníti a tényleges felületet. Ennek az adatok tisztasága miatt van fontos szerepe.

Következő lépésben el kellett készítenem a widget kinézetét leíró layout fájlt, és a megfelelő `AppWidgetProvider`-t, ami kezeli a widget életciklusát, és gondoskodik a frissítésről is.

A widget layout-ot a *res/layout/widget.xml* és a *widget_black.xml* tartalmazza.

```
public class BillMonitorWidget extends AppWidgetProvider {
      @Override
      public void onUpdate(Context context, AppWidgetManager
      appWidgetManager, int[] appWidgetIds) {
            updateWidget();
      }
      @Override
      public void onDisabled(Context context) {
            super.onDisabled(context);
      }
}
```

Az `onUpdate(...)` metódus akkor fut le, amikor a rendszer egy frissítési igényt küld ki a widget számára. Az `onDisabled(Context)` pedig akkor hajtódik végre, ha az utolsó példánya is el lett távolítva a widget-nek a főképernyőről.

A widget frissítését megvalósító metódus:

```
public void updateWidget() {
```

```
        ComponentName thisWidget = new ComponentName(context,
        BillMonitorWidget.class);
        RemoteViews remoteViews = new RemoteViews(context.getPackageName(),
        R.layout.widget);
        remoteViews.setTextViewText(R.id.widgetmaradtperc, Details[2] + " " +
        res.getString(R.string.minute_sign));
        remoteViews.setTextViewText(R.id.widgetmaradtsms, Details[3] + " " +
        res.getString(R.string.sms_sign));
        remoteViews.setTextViewText(R.id.widgetperiodText,
        DatePeriod[0]+"."+DatePeriod[3]+"."+DatePeriod[4]+" -
        "+DatePeriod[1]+"."+DatePeriod[2]+"."+DatePeriod[4]);
        Intent intent = new Intent(context, BillMonitorActivity.class);
        PendingIntent pendingIntent = PendingIntent.getActivity(c, 0, intent,
        0);
        remoteViews.setOnClickPendingIntent(R.id.widget_layout,
        pendingIntent);
        AppWidgetManager.getInstance(context).updateAppWidget(thisWidget,
        remoteViews);
}
```

Látható, hogy a widget kinézetét is `remoteViews` segítségével lehet megadni, és azokon keresztül könnyen tudok a megjelenítendő adatokon módosítani. Az `AppWidgetManager` egy példányának a meghívása után az `updateAppWidget(ComponentName, RemoteViews)` segítségével frissítem fel a widget-et.

5. További fejlesztési lehetőségek

Egy tudatos fejlesztő a megjelenés után egyből azon gondolkodik, hogy miként lehetne még ennél is jobbá, szebbé és ügyesebbé tenni az ő fáradtságos munkával létrehozott kis alkalmazását. Természetesen nagyobb programrendszerek és projektek esetén manapság már a megjelenési utáni tervek is előre meg vannak tervezve, de egyszerű programok esetén ez még nem jellemző.

A fejlesztés során és utána, én is érzékeltem azt, hogy mely pontokon tudna az alkalmazásom sokat fejlődni, hol lehetne igazán javítani rajta. A probléma ezzel leginkább az volt, hogy nagyon sok kódot kellett volna kidobnom és teljes mértékben átdolgoznom, hogy megvalósítsam a változtatásokat, amiket szerettem volna. Erre időhiányában nem volt már

lehetőségem. Ennek ellenére úgy érzem egy nagyon komplett, kerek alkalmazást sikerült kiadnom a kezemből, ami már most rengeteg funkcióval kecsegtet és próbálja meggyőzni a leendő felhasználókat, hogy tegyen vele egy próbát.

Röviden összefoglalom mit is tervezek a jövőre nézve:

- **Hívásnapló és SMS tároló alapján saját Napló készítése:** Az alkalmazásom jelenlegi gyenge pontja, hogy nagyban függ az adott készülék aktuális hívásnaplójától és SMS tárolójától. Tehát ha a rendszerben törlünk egy üzenetet, vagy hívásnapló bejegyzést, akkor onnantól kezdve nem garantálható a számítás helyessége, és körülbelüli pontossága, ugyanis az adatok hiányosak lesznek. Ezt szeretném úgy elkerülni, hogy ez első indítás alkalmával készítek egy saját naplót a hívásnapló és SMS tároló alapján, amit egy saját adatbázisban tárolok, mely segítségével már megmaradnak az adatok, ha esetleg a felhasználó a rendszerből törli is őket. Egyéb információval is el tudom látni az adott naplóbejegyzést (pl.: csúcsidőben történt-e a hívás), így az adatok feldolgozása is könnyebb feladat lenne az alkalmazás számára. A legnagyobb előnye viszont már az alkalmazás telepítése után indított hívásokban és SMS-ekben mutatkozna meg. Ezzel a módszerrel képes leszek eltárolni, egy adott hívás vagy üzenet mellett azt az információt is, hogy ROAMING körülmények között történt-e. Így a számítás során külön percdíjjal és sms díjjal tudnám a roaming hívásokat szűrni. Ez egy nagyobb átalakítást igényel a programkódban, ugyanis minden olyan metódust a *Methods* osztályban le kell cserélnem, ami jelenleg a hívásnapló és sms tár adatait kéri le.
- **Roaming hálózat érzékelése után percdíj értesítés:** Amint a készülék érzékeli a roaming-ot jelenleg egy értesítést helyez az állapotsávra, mely erről tájékoztatja a felhasználót. A jövőben ezen felül szeretnék egy értesítést, mely megérintése után, lehetőséget biztosítana a roaming percdíj azonnali megadására.
- **Main Activity-t átdolgozni:** A fő activity átdolgozása fragment-ek használatával és a *Navigation Drawer* implementálása, a funkciók könnyebb elérése érdekében. A szolgáltatásoknak is lenne ott gyorsindító.
- **Hívás utáni Toast értesítés:** Minden hívás után és sms küldés után egy apró *Toast* értesítés formájában tájékoztatnám a felhasználót arról, hogy mennyibe kerül ez a hívás vagy sms, az ingyen perceket és SMS-eket figyelembe véve.
- **Támogatói fizetős verzió létrehozása:** Amint Magyarországon is elérhetővé válik a fizetős alkalmazások feltöltése, és az alkalmazáson belüli fizetés, egy támogatói lehetőséget is készítenék a programhoz, ami egy plusz widget-et adna a sima verzióhoz.

Célja, hogy aki szeretne támogatni, vagy az alkalmazást szeretné támogatni, így könnyen megteheti és még plusz szolgáltatást is kap.

- **Havi Limit / Egyenleg Limit – hívástiltás:** Abban az esetben, amint a felhasználó eléri a beállított limitet minden kimenő hívás automatikusan tiltva lenne. A felhasználónak lehetőséget biztosítok olyan telefonszámok megadására, mely a limit elérése után is szabadon hívható marad. Jelenleg az ezzel kapcsolatos API zárt, és nem megoldható a hívástiltás komolyabb módosítások és ROOT funkciók nélkül, így nem tudom egyáltalán megvalósítható lesz-e valaha.

Ezeket a módosításokat, plusz funkciókat szeretném minél hamarabb megvalósítani, és ezekkel is még jobbá és használhatóbbá tenni az alkalmazásomat.

6. Konklúzió, avagy a projekt értékelése

Konklúzió. Következtetés levonás. Ez olyan dolog, melyet az ember önmaga nem feltétlen tud helyesen megítélni, mondom ezt azért, mert a saját munkája iránt az ember mindig is kicsit elfogult lesz. Megpróbálom ennek ellenére reálisan értékelni a munkámat.

Elérkeztem egy nagyon hosszú út végéhez. Az út során sok fejtörések, megoldandó problémák közepette megszületett egy nagyszerű alkalmazás, mely azért közel sem tökéletes jelenleg, de a jövőben még minden adott ahhoz, hogy sikeres legyen. A tesztelők közül sokan azt mondták, már az első pillanat után, hogy mennyire letisztult, mégis átlátható és informatív a felülete. Ezzel úgy gondolom elértem a célom, és megvalósítottam az elvárásokat, melyeket kitűztem magamnak az alkalmazásommal szemben.

A másik nehéz feladat ezen dolgozat megírása volt. Az alkalmazás komplexitásának, és sokrétűségének köszönhetően bőven akadt írni való, ám még ezek után is úgy érzem tudtam volna miről írni. A lehető legfontosabb elemeket megpróbáltam érthetően, és minél részletesebben bemutatni, annak érdekében, hogy egy kerek képet alkothassak arról, hogy miképpen áll össze, épül fel egy Android-os alkalmazás.

Megmutattam mi szükséges ahhoz, hogy valaki elkezdjen fejleszteni erre a remek platformra, készítettem egy tervet, mely alapján elkészítettem egy alkalmazást, aminek a legfontosabb elemeit ismertettem. Ezek után teszteltem is azt, és végül a Google Play Áruház segítségével elérhetővé tettem a nagyközönség számára is.

Nagyon sokat tanultam és rengeteg pozitív tapasztalatot is szereztem ez alatt a pár hónap alatt. A rengeteg fáradalmas és álmatlan éjszakákért kárpótol az, amit sikerült elérnem és gvalósítanom.

Irodalomjegyzék

[1] Horton, J. (2015). *Android programming for beginners*. Packt Publishing Ltd.

[2] Griffiths, D., & Griffiths, D. (2017). *Head First Android Development: a brain-friendly guide*. " O'Reilly Media, Inc.".

[3] Sándor, S., & László, V. (2001). *Szoftvertechnológia és UML*. ELTE Eötvös K..

Rövid szakmai életrajz

Jelenleg szoftverfejlesztési vezetőként dolgozom már lassan 5 éve a cégnél. Fő tevékenységeim közé tartozik a fejlesztői csapat koordinálása, projektre delegálása, kompetenciák felmérése új csapattag esetén, új munkavállaló beillesztése a csapatba, házon belüli konvenciók és fejlesztési irányelvek megalkotása és leoktatása a csapatnak, fejlesztési technológiák kutatása, újabb és hatékonyabb boilerplate-ek készítése új projektek számára. Nagy szerepet vállalok architect ként is a projektekben. Fő technológiánk a web alapú modern alkalmazások készítése, kiegészítve mobil alkalmazással. Frontend Angular (ha natív is kell akkor Electron-is), backend ASP.NET 6, mobilra Xamarin (MAUI). Adatbázisnak a bevált MS SQL-t preferáljuk, de más rendszereket is tudunk támogatni. Fő profilunk a B2B, üzleti ügyfeleinknek kínálunk meglevő termékeink mellé egyedi web alapú alkalmazásokat vagy teljesen új ötleteken alapú egyedi alkalmazásokat.