SOPRONI EGYETEM
SIMONYI KÁROLY MŰSZAKI, FAANYAGTUDOMÁNYI ÉS MŰVÉSZETI KAR
INFORMATIKAI ÉS GAZDASÁGI INTÉZET

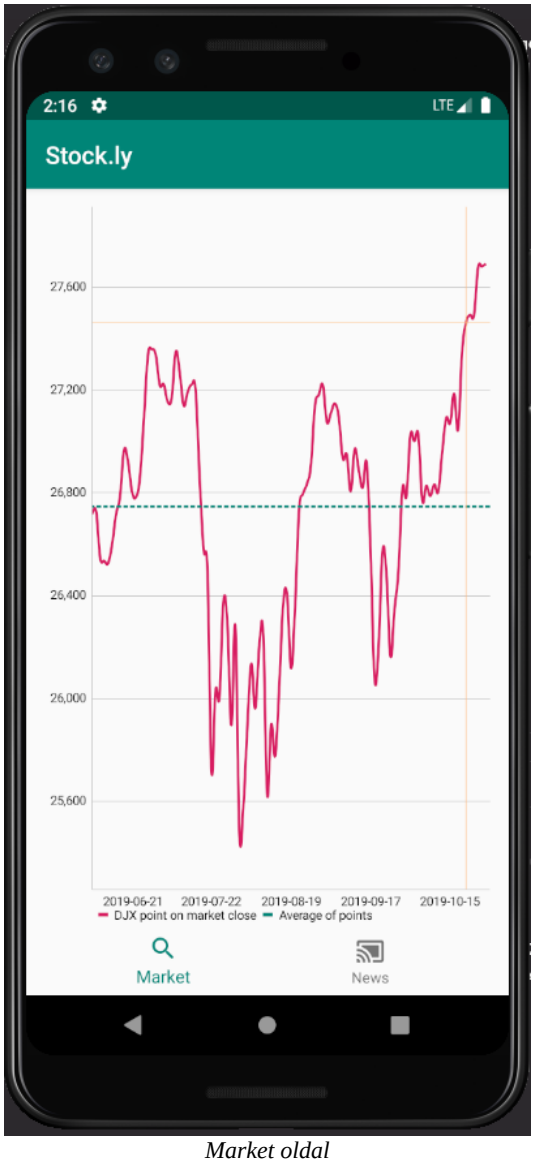
MOBIL SZOFTVERFEJLESZTÉS – STOCK.LY
DOKUMENTÁCIÓ

CSANAKI RICHÁRD
KZEADR

SOPRON,
2019. NOVEMBER 12.

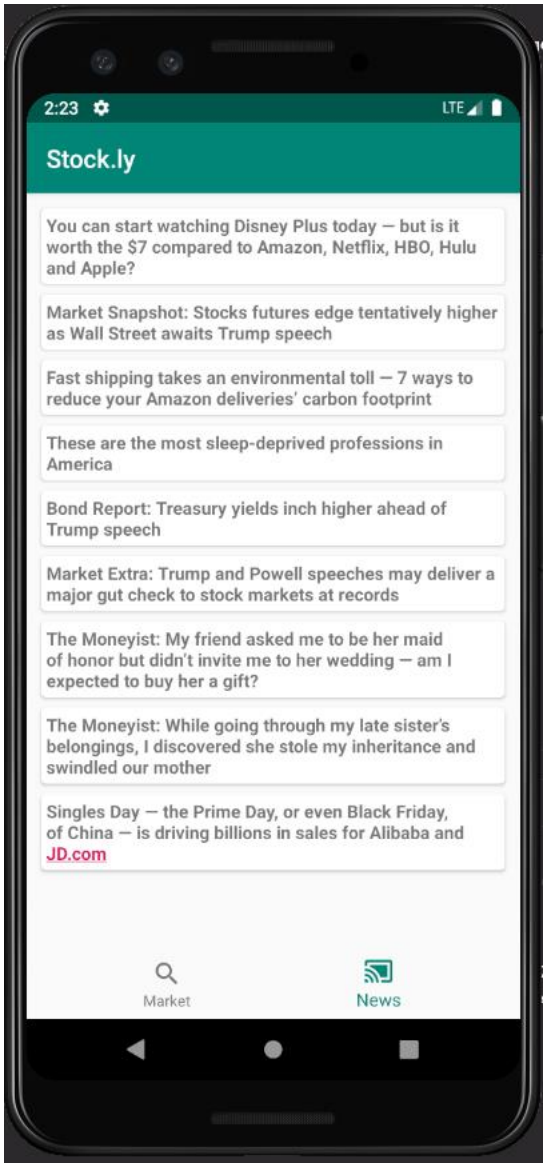
Koncepció
Az alkalmazás célja: tőzsdei adatok és hírek megjelenítése egy intuitív felületen.

Felhasználói felület és funkciók
A felhasználói felület kettő lapból áll – Market és News oldalak. A Market oldalon látható a Dow Jones Industrial Average állása 100 napra visszamenőleg (market close points), illetve az ezen idő intervallum alatt vizsgált pontok átlaga. A News oldalon a piacokkal kapcsolatos híreket olvashatunk. Az alkalmazás elforgatott pozícióban is működik.



Market oldal

A Market oldalon a grafikon nagyítható, mozgatható. Hogyha nincsen Internet kapcsolat, akkor a felhasználót erről egy Toast üzenet tájékoztatja.



News oldal

A News oldalon a hírekre kattintva a cikkek böngészőben jelennek meg, illetve az oldal tetejéről lefele húzva frissíthetjük a feed-et.

Az alkalmazás ikonja – azért négyzetes, mert így a lekerekített változat is szépen fog kinézni

Megvalósítás

Az alkalmazás felépítése

Az alkalmazás 2 Fragment-ből (FrameLayout-on elosztva) áll, amelyek közötti váltást egy BottomNavigationView oldja meg a Main Activity-ben.

Market oldal

Az oldalt elfoglaló grafikon megjelenítéséhez az MPAndroidChart könyvtárat használtam. Ezen könyvtár gyorsan átlátható és megbízható funkcionalitásával azonnal megfelelő eszköze lett a tőzsdei adatok megjelenítésének. Az adatpontok Entry objektumok, melyek a parse-olt JSON objektumokból táplálkoznak. A pontokat LineDataSet-ekhez adjuk, ezeket listába rendezzük és ezután adjuk át a LineChart objektumnak.

Mindezen adatokat az Alpha Vantage ingyenesen elérhető API-ja szolgáltatja. Az adatok eléréséhez egyszerű Http request-ekkel kell felkerseni az API-t, melyeket a Retrofit könyvtárral hajtok végre. A kérésre érkező válasz egy JSON formátumú objektum, melyet a tervezési fázisban POJO megközelítéssel akartam felbontani, azonban végül azt az utat választottam, hogy az eredményt String típusúvá cast-olom, és ezt a szöveget helyezem be egy JSON objektumba, melyet a Gson könyvtár metódusaival szépen lehet kezelni, belőle a megfelelő adatokat kinyerni. Az adatok megjelenítésekor feltűnt, hogy az adatok mindenféle rendezettség nélkül jelennek meg, így tárolásukra egy olyan adatszerkezetet használok, mely megtartja az inzertálás sorrendjét (LinkedHashMap). Továbbá ahhoz, hogy a megszokott módon jelenjenek meg az adatok, az egész adathalmazt meg kellett fordítani. Mindezen feladatokat az AlphaVantageApi, Average, ResponseParser és MarketFragment osztályok kezelik.

Az adatok TIME_SERIES_DAILY típusúak, azaz napi rendszerességűek, összesen 100 napra vonatkoznak és a piac zárásakor feljegyzett értéket használtam a megjelenítéshez.

```
public interface AlphaVantageApi {

    @GET("query?function=TIME_SERIES_DAILY&symbol=.DJI&outputsize=compact&apikey=BMOZOE9D5X9ECSP9")
    Call<ResponseBody> getResponse();

}
```

A Http request alapjául szolgáló interfész és benne a custom URL részlet

News oldal
A News oldal egy RecyclerView-n jeleníti meg a CardView-ba csomagolt híreket. Magukat a híreket a MarketWatch website RSS feed-je szolgáltatja. Az XML alapú üzenetből a hír címét és linkjét szedem ki, de csak a címet jelenítem meg, ami egy kattintható elem, a hír oldalára vezet a böngésző megnyitásával.
A híreknek van egy lifeSpan-jük, a lejárat után automatikusan frissülnek. Mindezt manuálisan is megtehetjük, hogyha a lap tetejéről lefele húzunk.
A híreket egy aszinkron lekérdezéssel érem el, mely a parse-olt eredményt egy RssFeedModel objektummá alakítja (title + link) és az adapternek átadva jeleníti meg egy kártyaként.
Az XML válasz feldolgozásakor első ránézésre a fejlesztés során nem volt probléma, teszteléskor derült ki, hogy hibás a parse logika, ugyanis adott elemre való kattintáskor az előző elem linkjét hozta be. Ezt sikerült orvosolni, így most már az elvárásoknak megfelelően működik. Mindezen feladatokat az RssFeedModel, RssFeedListAdapter és a NewsFragment osztályok kezelik.

```
if (
        eventType == XmlPullParser.START_TAG
                && name.equalsIgnoreCase( anotherString: "item")
) { isItem = true; }

if (
        eventType == XmlPullParser.END_TAG
                && name.equalsIgnoreCase( anotherString: "item")
) { isItem = true; }
```

A javított parse logika

Ismert bug-ok - azaz feature-k :')

- Ha túl gyorsan váltunk a Market és News Fragment között, akkor a Market oldal túl gyorsan küld request-et az API felé, így time out-ol a lekérdezés és nem jelennek meg az adatok – ebből is látszik, hogy nem állapot megőrző a megoldás
- A Market oldal grafikonjának legend pozíciója csúnya, túl zsúfolt az X-tengely elemeivel. Azonban a helyzete nem állítható, ez az opció csak az MPAndroidChart könyvtár előző verzióiban volt elérhető
- A dátumok megjelenítése meglehetősen ad-hoc módon történik az X-tengelyen, sajnos a könyvtár nem rendelkezik olyan funkcióval, amellyel be lehetne állítani a tengelyen megjelenítendő első és utolsó elemet. Azonban, hogyha ráközelítünk (pinchToZoom) a grafikon végére, akkor előbukkannak a keresett dátumok
- Hogyha nincsen internet elérés, akkor a News oldal néha nem jelez a felhasználó felé (viszont a Market oldalon mindig megjelenik a Toast üzenet, úgyhogy ez talán nem akkora probléma)

Tapasztalatok, kitekintés

Ezen alkalmazás elkészítésével megismertem az Android programozás alapjait és a már sokszor használt elveket sikerült mobil platformon is implementálni (Http request, JSON-XML parsing, stb.).
Természetesen az alkalmazás funkcionalitása még jócskán bővíthető, például egy közönséges átlagnál komplexebb elemzésekkel. Ennek egy lehetséges módja lehetne mondjuk egy Python alapú API, amely az alkalmazást ellátja ezekkel az adatokkal.

Az alkalmazás forráskódja és dokumentációja elérhető itt:
<https://github.com/csana23/mobilprog>

Sopron, 2019. november 12.