Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Irányítástechnika és Informatika Tanszék

Pap Gábor

Közös kódbázisú szoftverfejlesztés Android és GWT alapon

TDK Dolgozat

Konzulens
Dr. Goldschmidt Balázs

BUDAPEST, 2012

Tartalomjegyzék

Ábrajegyzék

1 Bevezető

1.1 Előszó

A számítógépek elterjedése után manapság az okostelefonok növekvő népszerűségével egyre több eszközön nyílik lehetőség az internetes alkalmazások használatára. Ezen új eszközök kialakításukból adódóan új, a személyi számítógépektől eltérő felületet nyújtanak a használat során. Teljesen más ugyanazt az alkalmazást használni egy PC-n és egy okostelefonon.

A népszerű internetes alkalmazásoknak elérhető az okostelefonra alkalmazásként megírt változata is.[1] Ezek jellemzően az eredeti webes alkalmazások megszületése után készülnek. A különálló okostelefon alkalmazások létét az indokolja, hogy a weboldalak nem működnek teljes értékűen egy okostelefonra szánt böngészőben, ami részben a készülékek használatából és méretéből fakad (érintőképernyőn más az elérhető felület-elemek optimális sűrűsége), másrészt az okostelefonok böngészői nem mindig biztosítják az asztali gépek böngészőinek funkcionalitását. Kész weboldal mellé egy új, hasonló funkcionalitású okostelefon-alkalmazás fejlesztése erőforrás-igényes lehet. Gyakran a teljes alkalmazást az alapoktól újra kell írni.

Jelen projekt a fent említett folyamat megkönnyítését, leegyszerűsítését tűzte ki célul. A kiindulási megkötés az volt, hogy a weboldalt (alkalmazás PC verziója) Google Web Toolkit, az okostelefon alkalmazást pedig Android platformon kell kifejleszteni. A két platform közös programozási nyelve a Java nyelv. Ez lehetőséget biztosít arra, hogy közös kódrészleteket használjunk fel a két különböző platformra készített alkalmazásban. Bemutatjuk, hogy milyen tervezési minták alkalmazhatók a fejlesztés során, hogyan modellezhető egy ilyen alkalmazás, és milyen gyakorlatot érdemes követni, hogy a két konfiguráció forráskódja minél kevésbé térjen el egymástól.

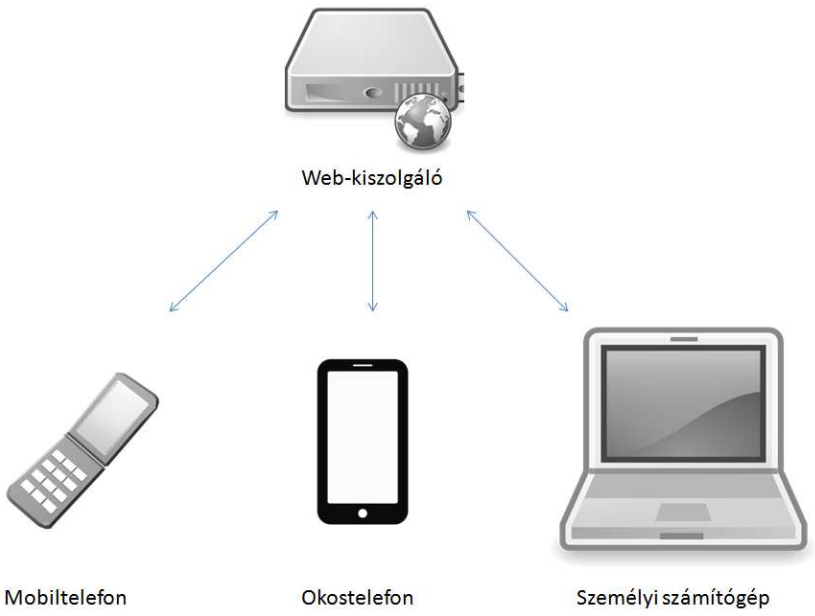
1.2 Körkép: Alkalmazások

A webes technológiák fejlődésével lehetővé vált, hogy a weboldalak ne csak statikus oldalak legyenek, melyeket az újságokhoz hasonló módon tudunk lapozgatni, hanem egyre funkciógazdagabb módon tudjuk őket használni. Így egy weboldalra ma már inkább úgy gondolhatunk, mint egy teljes értékű számítógépes alkalmazásra.[2] Megjelentek a web-alkalmazások.

A mobiltelefonok fejlődésével pedig megjelent egy új kategória, az okostelefonoké. Ezekre a készülékekre inkább tekinthetünk úgy, mint kisméretű számítógépekre, hiszen elég sokrétű használatot biztosítanak a telefonálás mellett.

1.3 Web- és mobilalkalmazások kapcsolata

A mai trendeknek megfelelően egy elkészült web-alkalmazásnak (1.3.1 pont) elérhető egy mobilböngészőre optimalizált felülete (1.3.2 pont), valamint okostelefonra megírt változata is (1.3.3 pont) a legnépszerűbb mobil platformokon. (például Facebook, Youtube, Gmail, Ebay) A mobilböngészős felület általában kevesebb funkcióval bír, de az okostelefon változat már kényelmes használatot kínál, néha teljesen helyettesíteni tudja a web-alkalmazást (egyes esetekben pedig az okostelefon verzió többletszolgáltatásokat is nyújt). Ezen verziók egyszerű esetben egy kiszolgáló számítógéppel kommunikálnak.



1. ábra: Kommunikáció a közös web-kiszolgálóval

1.3.1 Web-alkalmazások

Általában azokat az alkalmazásokat nevezzük web-alkalmazásnak, melyek futtatásához egy böngésző programra van szükség a felhasználó eszközén. Egy ilyen alkalmazás különböző technológiák sorát alkalmazza. [3] Már a 2000-es évek előtt léteztek szerver oldalon olyan technológiák, mint a CGI, Java Server Pages, Microsoft Active Server Pages, illetve PHP szerver.[4] Kliens oldalon pedig HTML böngésző, JavaScript értelmező motorral, AJAX, CSS támogatással. Ezek felhasználásával már

komplex web-alkalmazások írhatóak és modellezhetőek UML diagramok segítségével[1]. Az újabb technológiák megjelenése pedig még fejlettebb alkalmazások megjelenését tette lehetővé. A legtöbb videó megosztó oldal Adobe Flash alapú lejátszót használ. A HTML5 pedig viszonylag új szabvány, elterjedőben van. A WebGL segítségével pedig már 3 dimenziós grafikus megjelenítés is lehetséges hardveres gyorsítással egy web-alkalmazásban.

Ezeknek az alkalmazások általában kielégítő a teljesítménnyel rendelkeznek, hiszen a számítások nagy részét a szerver végzi. A kliens oldali kód viszont lassabban fut, mint egy natív alkalmazásban.

1.3.2 Web-alkalmazások mobilböngészős felülete

A fent említett alkalmazásoknak gyakran készül el egy kisebb kijelzőre, kisebb sávszélességre optimalizált változata, általában kevesebb funkcióval. A megszorítások a mobiltelefonok korlátozott erőforrásai (szoftveres és hardveres), lehetőségei miatt szükségesek. Nehezen képzelhető el, hogy egy hagyományos mobiltelefon kijelzőjén ugyanúgy jelenjenek meg a komplexebb weboldalak, mint egy számítógép monitorján.

1.3.3 Web-alkalmazások okostelefon verziója

Ma már általánossá vált, hogy a web-alkalmazások egy letölthető alkalmazással helyettesíthetőek az okostelefonokon. Az okostelefonok sokkal fejlettebb hardverrel rendelkeznek, mint pár évvel ezelőtti elődei. A web-alkalmazások mégsem használhatóak gördülékenyen, főként az érintőkijelző, és a méret miatt, hiszen a weboldalakat egér és billentyűzettel történő használatra készítik. Bár láthatunk példát teljes funkcionalitású web-alkalmazások okostelefonra optimalizált felületére (Google Maps, Gmail), azonban ezeket összehasonlítva a letölthető alkalmazással azt tapasztaljuk, hogy utóbbi sokkal gördülékenyebben, gyorsabban használható, jobban ki tudja használni a hardver adottságait.

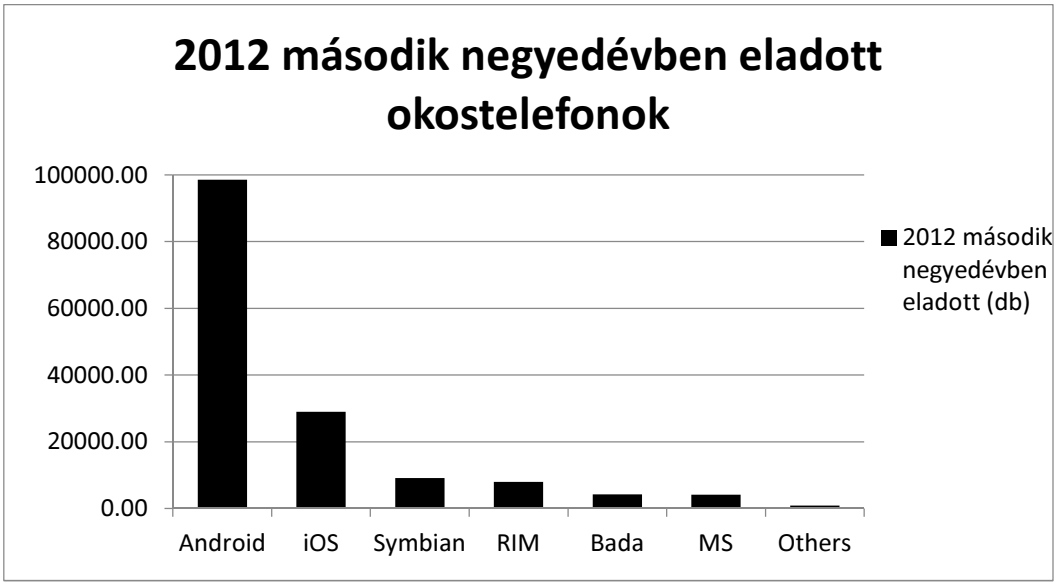
1.4 Jelen projekt kiválasztott technológiái

A projekt célja, hogy bemutassa, miként írható meg egy alkalmazás okostelefon- és web-alkalmazás verziója párhuzamosan a lehető legkevesebb kódduplikálással. Ehhez

[1] Egy lehetséges megközelítés a weboldalak UML modellezéséhez:
Jim Conallen, Rational Software - Modelling Web Application Architectures with UML - http://www.deetc.isel.ipl.pt/Programacao/Programacao_Inv_2004_2005/ti/Documentacao/webapps.pdf [5]

a választott két platform az Android és a Google Web Toolkit. Mindkét platform elsődleges programozási nyelv a Java. Ez lehetőséget nyújt közös kódrészletek felhasználására, és közös fejlesztőkörnyezet, az Eclipse használatára.

Az Android választását népszerűsége is indokolja. Az eladások száma alapján a legnépszerűbb okostelefon platform.



2. ábra: Forrás: Gartner (http://www.gartner.com/it/page.jsp?id=2120015)

A Google Web Toolkit azért tűnt kézenfekvő választásnak, mivel jelentősen leegyszerűsíti a fejlesztést, hiszen az alkalmazás készítőjének nem kell ismernie a JavaScript nyelvet a kliens oldali rész fejlesztéséhez, helyette az Androidon is használt Java a programozási nyelve.

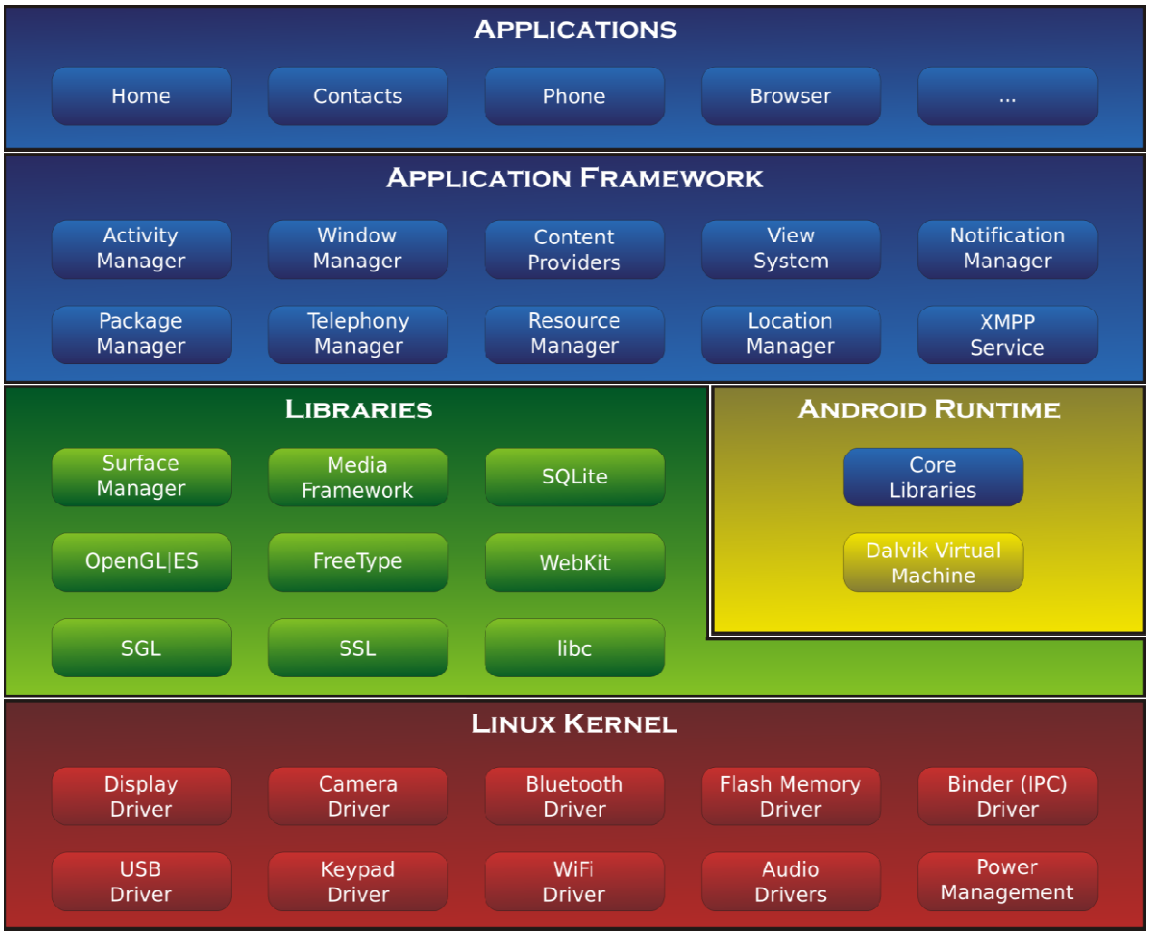
A két platformon történő alkalmazásfejlesztéshez a következő szoftvereket használtam:

- Eclipse 4.2 (Juno)
- Google Plugin for Eclipse (GWT fejlesztéshez)
- Android Development Tools és Android SDK (Android fejlesztéshez)
- Apache Tomcat 7.0 webszerver (Az alkalmazás szerver oldali kódjának futtatásához)
- Android 4.1 Jelly Bean (Google Nexus S telefonon futtatva)
- Google Chrome böngésző

2 A választott platformok bemutatása

2.1 Android

Az Android első változata 2009-ben jelent meg a felhasználók számára, Donut kódnévvel, 1.6 verziószámmal. A legfrissebb verzió 4.1 Jelly Bean. Az operációs rendszer Linux kernel-re épül. A fejlesztés szempontjából fontos kiemelni, hogy a megírt Java nyelvű alkalmazás Dalvik Executable bináris kódra fordul, majd ez a kód fut a Dalvik virtuális gép felett. [7]



3. ábra: Az Android platform felépítése

Alkalmazások fejlesztéséhez szükség van az Android-SDK-ra. Ezzel együttműködik az Eclipse nevű fejlesztőkörnyezet, a Google által kifejlesztett plugin segítségével. Az SDK tartalmaz többek között rengeteg könyvtárat, amit használhat az általunk megírt alkalmazás, valamint egy Android emulátort, melyen könnyedén futtathatjuk hibakereső módban is az alkalmazásunkat. Lehetőség van arra is, hogy közvetlenül

egy számítógéphez kötött okostelefonon teszteljünk, ez egy sokkal kényelmesebb mód, és valós képet ad arról, hogy hogyan fut az alkalmazásunk.

„Egy Android-alkalmazás forrása fejlesztői szempontból a következő fő csoportokba osztható:

- *Metainformációk: főként az alkalmazás projektleíró manifest állománya*
- *Erőforrások: XML-alapú felhasználói felület elemek, XML-alapú nyelvi állományok, egyéb XML-erőforrások, valamint különféle multimédia-elemek (képek, hangok, stb.)*
- *Forráskód: Java-alapú forrás (vagy C/C++ NDK használata esetén)*
- *Külső osztálykönyvtárak: lefordított könyvtárak, amelyeket az alkalmazásunk forrása felhasznál*" [7] (old.:33)

Felépíteni pedig a következő típusú komponensekből tudunk egy alkalmazást:

- Activity: tulajdonképpen egy képernyő, amivel a felhasználó interakcióba lép. Egy alkalmazás általában több Activity-ből áll. Ezek egymással kapcsolatba léphetnek, sőt más alkalmazások Activity-jeit is elindíthatják.
- Service: a háttérben futó feladatok megvalósítására szolgál.
- ContentProvider: tartalomszolgáltató. Több alkalmazás számára biztosít adatokat. Például a névjegyzék is egy ilyen tartalomszolgáltató, a többi alkalmazás lekérheti a telefonon tárolt névjegyeket.
- BroadcastReceiver: a telefonon bekövetkező eseményekre lehet feliratkozni egy ilyen komponenssel. Így végrehajthatjuk a saját kódunkat, ha bekövetkezik valamilyen esemény, például bejövő hívás, alacsony akkumulátorszint, stb.

Ebben a projektben az Activity-t használjuk, a többi elemre nincs szükség az esettanulmányban lévő egyszerű felépítésű alkalmazás miatt. Fontos megemlíteni a Fragment-eket. Ezeket a táblagépek támogatása óta vezették be. *„A Fragmentek tulajdonképpen önálló életciklussal rendelkező Activity-k, amelyek közül egy időben többet is megjeleníthetünk a felhasználói felületen, rugalmasan kezelhetők, és önálló, elkülönült üzleti logika rendelhető mögéjük.*" [7] (old.:331) A Fragmentek használata jelentősen megkönnyíti a több céleszközre történő fejlesztést.

2.2 Google Web Toolkit bemutatása

A JavaScript segítségével a weboldalak a kliens oldalon is képesek kód futtatására.[8] Az AJAX technológia segítségével pedig aszinkron módon tudnak interakcióba lépni a szerverrel: a háttérben küldhető adat a szerverre, és fogadható onnan anélkül, hogy újra kellene tölteni a teljes weboldalt. [9]

A JavaScript gyengén típusos[2] nyelv, fordítás nélkül futtatható, objektumorientált, de prototípus alapú[3]. Körülményes kezelni vele az AJAX hívásokat. Böngészőnként eltérő megvalósításai vannak, nehéz biztosítani, hogy a megírt kód minden böngészőben ugyanúgy működjön. Ez megnehezíti nagyobb alkalmazások fejlesztését, nem véletlen tehát, számos keretrendszer, könyvtár készült hozzá a fejlesztés könnyítéséhez.

A Google Web Toolkit is tulajdonképpen egy AJAX keretrendszer, vagyis az előbb említett aszinkron hívások kezelésére nyújt egy felületet.[10] A programozási nyelve a Java, ebből fordul le végül a kliens oldalon futtatható JavaScript-re az alkalmazás.

A GWT alkalmazásokat is (hasonlóan az Androidhoz) az Eclipse fejlesztőkörnyezet segítségével írhatunk. Ehhez szintén van egy a Google által kifejlesztett plugin. A telepítés után létrehozhatunk GWT project-et az Eclipse-ben.

Létrehozás után egy GWT project az Eclipse-ben a következőket tartalmazza:

- A Java nyelvű forráskódok, külön Java csomagban a kliens és szerver.
- /war nevű mappa, ami a weboldal statikus elemeit tartalmazza: html és css fájlok, képek.
- /war/WEB-INF, ami a lefordított Java fájlokat tartalmazza
- /war/WEB-INF/lib, a külső .jar fájlok számára
- MyApp.gwt.xml: a MyApp nevű GWT modul leíró fájlja, szerepel benne például a modul belépési pontjául szolgáló osztály neve
- web.xml: A szerver oldali alkalmazás leíró fájlja.

Egy-egy GWT modult egy HTML fájlba kell beágyazni, általában egy blokknak veszi át a helyét. Az oldal betöltődésekor lefut a modul belépési pontjának onModuleLoad() metódusa. Ebben a függvényben állíthatjuk össze az alkalmazás kezelői felületét widget-ek (pl. gombok, szövegmezők, összetett elemek, stb.) segítségével, és rendelhetünk eseménykezelést a widget-en történő eseményekhez.

[2] Gyengén típusos: nem szükséges megadni egy változó típusát, mely futás közben változhat. Így nehéz ellenőrizni, hogy megfelelő típusú paraméterekkel hívtunk-e meg egy függvényt

[3] Prototípus alapú: a JavaScript is objektumorientált, azonban az objektumok létrehozásához nem kell definiálnunk egy osztályt. Az öröklést az objektumok lemásolásával lehet megvalósítani.

Az távoli eljáráshívás (RPC) egy hatékony technika kliens-szerver alkalmazások fejlesztése során, lehetőséget biztosít a programozónak arra, hogy a hálózaton keresztül hívjon meg egy távoli számítógépen megvalósított eljárást.[11]

A GWT alkalmazásban a Callback tervezési minta segítségével valósítható meg. Ennek használatát a most következő esettanulmányban követhetjük nyomon.

3 Esettanulmány

Ahhoz, hogy megvizsgálhassuk, miként történhet a közös kódbázisú fejlesztés, először egy esettanulmányt mutatnék be arról, hogyan történik a fejlesztés a két platformon külön-külön, majd hogyan próbáljuk meg átalakítani a kommunikációt úgy, hogy ugyanaz a szerver ki tudja szolgálni a mindkét alkalmazást, végül a klienseket hogyan valósítjuk meg úgy, hogy minél közelebb álljon a megvalósításuk, minél több közös kódot használjanak.

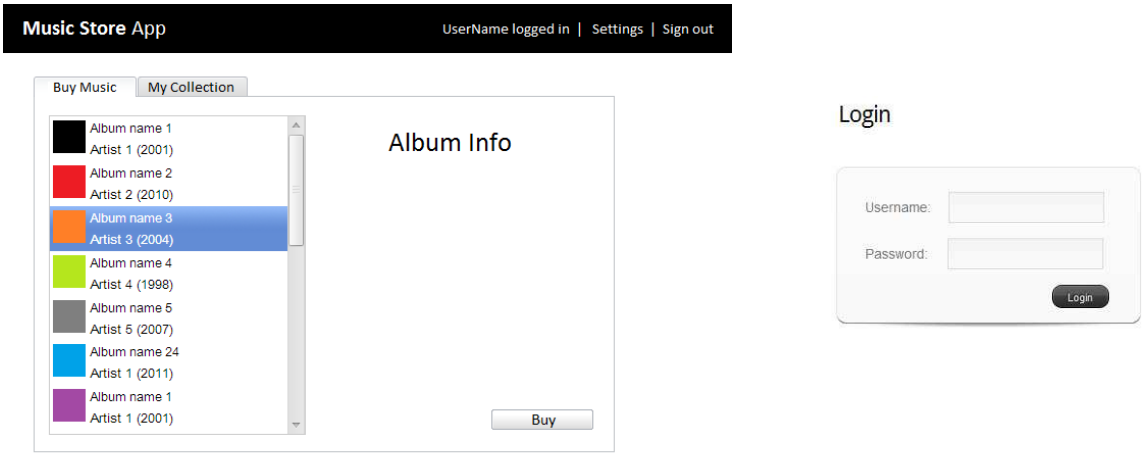
A választott alkalmazás egy egyszerű zenebolt. Ehhez szükség van egy szerverre, ahol tárolódnak az információk a megvásárolható zenékről, az regisztrált felhasználókról. Valamint meg kell valósítani egy-egy klienst is, ami kommunikál a szerverrel, lekéri a szükséges adatokat, és megjeleníti a felhasználó számára.

3.1 Egy-egy különálló alkalmazás fejlesztése a két platformra

3.1.1 Google Web Toolkit

A GWT könnyen használható lehetőséget biztosít arra, hogy kliens-szerver alkalmazásokat írjunk a Java nyelv segítségével. Az egyik előnye a távoli eljáráshívás támogatása, tehát a kliensről lehetőség van a szerveren megvalósított metódusok meghívására anélkül, hogy törődni kellene a kommunikációs protokoll kiválasztásával, használatával. Ebben a példában is használni fogjuk ezt a funkciót.

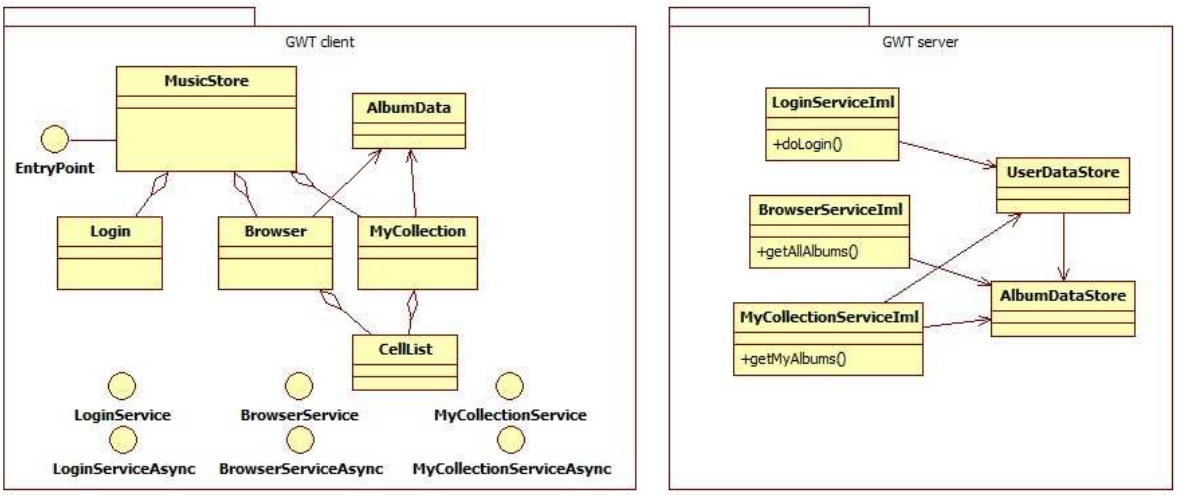
A Zenebolt alkalmazásnak először megterveztem a felhasználói felületét. Szükség van egy bejelentkező ablakra, a megvásárolható zenék listázására, illetve a megvásárolt zenék megtekintésére. Az alkalmazás felhasználói felületének kinézete:



4. ábra: Music Store alkalmazás GWT-ben

Tervezői döntés volt, hogy nem helyeztem el statikus elemeket a web-alkalmazásban, minden dinamikusan jelenik meg, Java kódban megvalósítva. Így könnyebb modellezni, és megvalósítani is.

Az alkalmazás belső felépítésének szemléltetése a következő osztálydiagramon történik.



5. ábra: Music Store alkalmazás GWT-ben

Az egyes kliens oldali osztályoknak a szerepe, felelőssége, és hogyan kapcsolódnak a felhasználói felület elemeihez:

- **MusicStore**

  Ez az osztály implementálja az EntryPoint interfészt. Ez felelős azért, hogy megjelenjen a weboldal. Ez helyezi el a szükséges elemeket az oldalon: bejelentkező oldal (Login osztály felhasználásával), megvásárolható albumok listázása (Browser osztály felhasználásával) és megvásárolt albumok (MyCollection osztály felhasználásával).

- **Login**

  A bejelentkező ablak implementációját tartalmazza. Elküldi a szervernek a bejelentkezési adatokat: meghívja a szerver oldali LoginServiceImpl osztály doLogin metódusát. Valójában egy GWT widget, ami már meglévő widgetekből áll (kompozit elem).

- **Browser**

  Tartalmaz egy CellList osztálypéldányt, ami a megjeleníti a szervertől lekért megvásárolható albumok listáját. A BrowserServiceImpl osztály getAllAlbums metódusát hívja meg a szerveren.

- **MyCollection**

Ez is tartalmaz egy CellList osztálypéldányt, ami a már megvásárolt albumokat jeleníti meg. A MyCollectionServiceImpl getMyAlbums metódusa segítségével kéri le az adatokat a szervertől.

- **AlbumData**
  A fogadott albumok adatainak osztálya. Egy ilyen típusú objektummal tér vissza a getMyAlbums és getAllAlbums osztály.
- **CellList**
  Egy GWT widget, lista megjelenítésére használható.

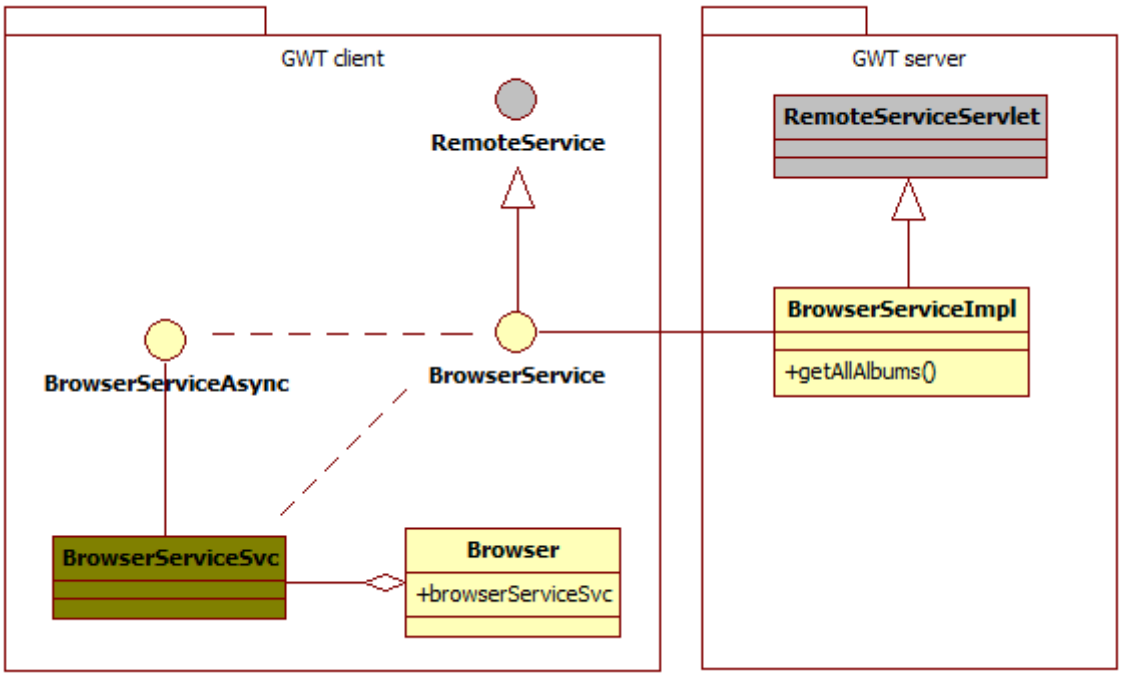
A szerver oldali osztályok bemutatása:

- **LoginServiceImpl**
  A doLogin metódusa elvégzi a bejelentkező felhasználó adatainak ellenőrzését, a UserDataStore osztály segítgésével.
- **BrowserServiceImpl**
  A megvásárolható albumokat küldi el JSON string-be szerializálva. Az AlbumDataStore osztálytól kéri el az adatokat.
- **MyCollectionServiceImpl**
  A már megvásárolt albumokat küldi el.
- **UserDataStore**
  Tárolja a felhasználók adatait, megvásárolt albumaik listáját
- **AlbumDataStore**
  Tárolja az albumok adatait.

A web-alkalmazás vázlatos működése: Ha megnyitjuk az oldalt, akkor a bejelentkező ablak (Login osztály) fogad minket. Ez meghívja a szerveren a LoginServiceImpl osztály doLogin metódusát, ami elvégzi az adatok ellenőrzését. Amikor visszatér a függvény a kliens oldalon, akkor megjelenik a megvásárolható albumok listája, vagyis magának az alkalmazásnak a felülete. A felhasználóhoz létrejön egy session[4], ami kijelentkezésig (vagy egy megadott időkorlátig) érvényes, utána újra be kell jelentkezni. (A GWT támogatja a session-kezelést. [12])

[4] Session: a kliens és a szerver közötti kommunikációhoz jön létre, amíg adatokat küldenek egymásnak. Mindkét fél tárolhat információkat a beszélgetésről. Ezt az információt is felhasználva tudja a szerver megkülönböztetni a bejelentkezett felhasználókat.

A távoli eljáráshívást megvalósító osztályok leírásán keresztül bemutatom, hogyan működnek. Ehhez a BrowserService működését veszem példaként. Csak a magyarázat szempontjából szükséges osztályok láthatóak a következő osztálydiagramon.



6. ábra: A BrowserService bemutatása

A szürkével jelölt elemeket a GWT keretrendszer biztosítja, a BrowserServiceSvc-t pedig az általunk megírt kód alapján generálja a keretrendszer. Emellett megkötések is vannak az osztályok neveit illetve tartalmát illetően. A megvalósítani kívánt szolgáltatás neve BrowserService. Így a következő elemeket kell létrehozni:

- **BrowserService** interfész. Ez definiálja, hogy milyen eljárásokat hívhatunk meg a kliensből a szerveren.
- **BrowserServiceAsync** interfész. Kötelező párja a BrowserService-nek. Minden BrowserService-ben szereplő eljárásnak van egy párja a BrowserServiceAsync interfészben. Ezeket kell hívni a kliens oldalon, melyek meghívják a szerver oldali metódusokat.
- **BrowserServiceSvc**. Ezt a keretrendszer generálja az általunk megírt osztályokból, a BrowserServiceAsync megvalósítása.
- **Browser:** az általunk írt osztály. Ebben példányosodik egy BrowserServiceSvc objektum, melynek a függvényeit meghívva történik meg a távoli eljáráshívás.

- **BrowserServiceImpl:** a szerveren lévő távolról hívható eljárások megvalósítása. Ősosztály a RemoteServiceServlet, ami tartalmazza a GWT-RPC belső megvalósítását

3.1.2 Android

Az Android alkalmazásból nincs lehetőség meghívni az előző pontban megvalósított szerver oldali függvényeket, mert nincs erre támogatás. Helyette új szerver implementációt kellett írni, és kiválasztani hozzá a megfelelő kommunikációs módot. Másik probléma, hogy a fent megvalósított session-kezelés itt már nem működik, hiszen nem böngésző programot használunk.

3.1.2.1 Servlet

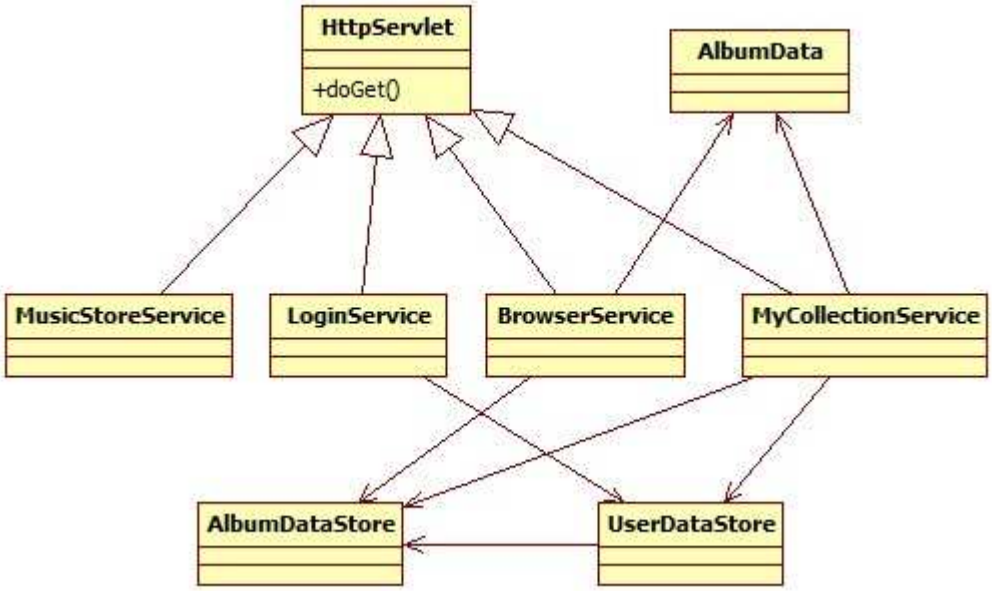
A kliens és szerver közötti adatforgalom HTTP felett küldött JSON objektumok segítségével történik.

„A JSON (JavaScript Object Notation, JavaScript objektumjelölés) emberek számára is olvasható–írható, programozottan pedig könnyen feldolgozható és előállítható, pehelysúlyú adatcsere-formátum. … A JSON noha programozási nyelvtől független szöveg-formátum, a C családú nyelvekben – C, C++, C#, Java, JavaScript, Perl, Python stb. – jártas programozó számára ismert konvenciókhoz igazodik. Ezek a tulajdonságok teszik a JSON-t ideális nyelvvé adatcseréhez.”[13]

Megvalósítására a Google GSON könyvtárát használtam, aminek segítségével egyszerűen lehet már meglévő osztályainkat szerializálni és deszerializálni. „A Gson egy Java könyvtár, mely arra használható, hogy Java objektumokat átkonvertálhassunk JSON megfelelőikké. Arra is fel lehet felhasználni, hogy egy JSON string-et egy ekvivalens Java objektummá konvertáljunk. A Gson tetszőleges Java objektumokkal együtt tud működni, beleértve azokat is, melyeknek nem ismerjük a forráskódját.” [14]

A felhasználók bejelentkezésének kezelésre pedig saját implementációt használtam: A szerver bejelentkezéskor generál egy azonosítót, amit eltárol a kliens. Ezután minden kérésben szerepel ez az azonosító. Ha egy meghatározott ideig nem érkezik kérés az azonosítóval, akkor kijelentkezettnek tekintjük a felhasználót. (Persze ez a módszer biztonsági szempontból vitatható. Itt annak szemléltetésére szolgál, hogy bizonyos funkciókat újra meg kell valósítani, amik a másik platformon léteznek.)

A servlet osztálydiagrammja a következő.



7. ábra: A servlet osztálydiagramja

Az osztályok szerepe hasonlít a GWT szerver osztályaihoz, azzal a különbséggel, hogy most a MusicStoreService, LoginService, BrowserService és MyCollectionService osztályok a doGet metódust valósítják meg, a doLogin, getAllAlbums és getMyAlbums helyett. A doGet metódusban lehetőség nyílik válasz küldésére egy string puffer segítségével, és nem tudunk objektummal visszatérni, mint a GWT esetében. Ebbe a string pufferbe kell szerializálni JSON formába az átküldeni kívánt adatot.

Az új (GWT server-ben nem szereplő) osztályok listája:

- **HttpServlet**
  A servlet-ek ősosztálya, ez tartalmazza a servlet-ek működéséhez szükséges kódot.
- **MusicStoreService**
  Ezzel kommunikál először az Android kliens. Ellenőrzi, hogy session azonosító alapján, hogy be van-e jelentkezve a felhasználó.
- **LoginService**
  A bejelentkezést, és a session azonosító generálását, karbantartását végzi.
- **BrowserService**
  A megvásárolható albumok listáját küldi el JSON-ba szerializálva

- **MyCollectionService**

  A felhasználó megvásárolt albumainak listáját küldi el, szintén JSON formában.

3.1.2.2 Kliens

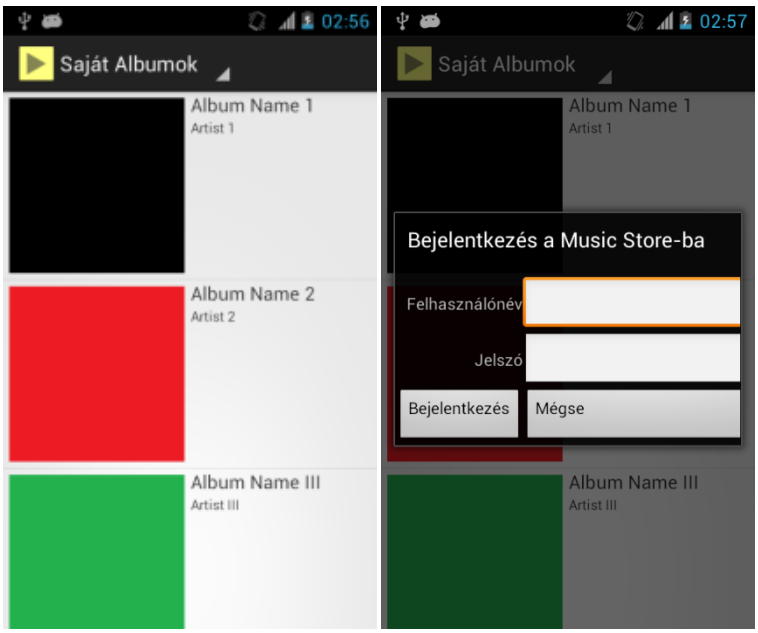
A felhasználói felület megtervezésekor Fragmenteket használatunk az Activity-kben. Ez egyébként az új követendő módszer, hiszen megkönnyíti a több kijelző méretre (okostelefon, táblagép) történő fejlesztést.

Fontos figyelembe venni a dizájn alapelveket[5] is a tervezésekor, hogy a felhasználói felület könnyen kezelhető legyen, és egységet alkosson a telefonon lévő többi alkalmazással. Így a felhasználónak nem kell tanulni a kezelést, minden egyértelmű lesz számára. Erre példa ebben az alkalmazásban a legördülő menü, amellyel lehet választani a nézetek között.

Az alkalmazás kezelői felülete igen egyszerű: szükség van egy bejelentkező ablakra, egy képernyőre, ami megjeleníti a megvásárolható albumok listáját, és egy olyanra, ami a megvásároltakét. Ennek megvalósítására két Activity-t használtam:

- **ActivityMain**: a tartalom megjelenítésére szolgál. Tartalmaz két Fragmentet, BrowseFragment és MyCollectionFragment, melyek a megvásárolható és megvásárolt albumok megjelenítéséért felelősek
- **ActivityLogin**: a bejelentkező ablak megjelenítését végzi.

Az alkalmazás kinézete a következő.



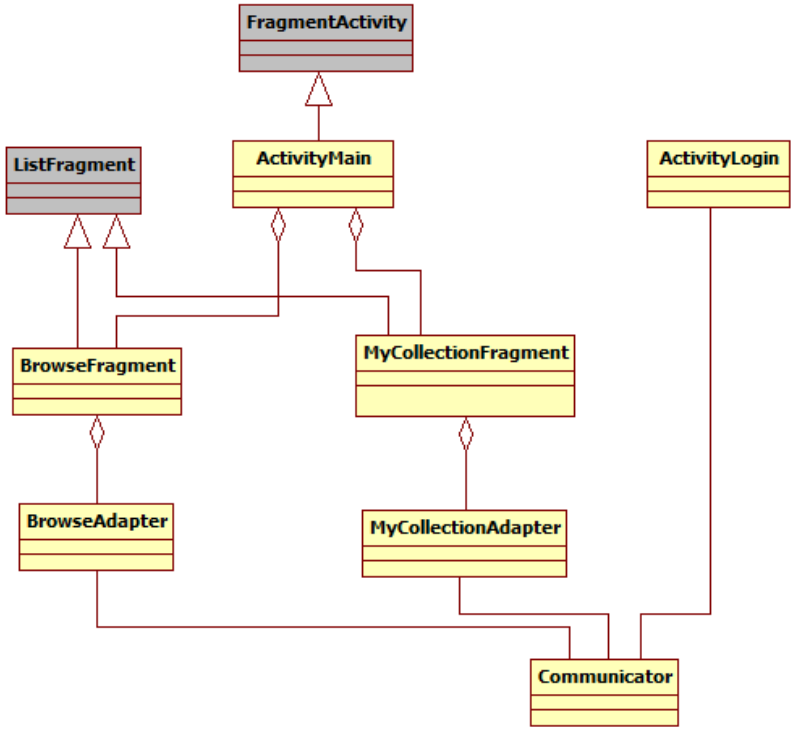
8. ábra: Music Store alkalmazás Androidon

[5] Android Design Guide: http://developer.android.com/design/index.html

A felhasználók bejelentkezésének kezelésére külön implementációt írtam. GWT-ben használhattuk a Browser Session-t, de ez Android-on már nem működött volna.

A belső felépítés modellezésénél nem hagyatkozhatunk csak UML osztálydiagramokra, hiszen egy Activity vagy egy Fragment vizuális felépítését XML fájlok segítségével definiáljuk, és az alkotóelemeit egy központi osztályon keresztül érjük el. Egy XML fájlban definiáljuk például, hogy egy Activity a létrehozás után milyen Fragmentet tartalmazzon. Az alábbi osztálydiagramon az ActivityMain és ActivityLogin Intent objektumok segítségével kommunikálnak egymással az Android operációs rendszer komponensein keresztül.

A hálózati adatforgalom bonyolítását nem ajánlott a fő szálból kezelni. Hogy a kommunikáció elkülönüljön, létrehoztam egy külön osztályt a számára, ami külön szálon végzi az adatküldést és -fogadást.



9. ábra: MusicStore alkalmazás Android változatának osztálydiagramja

A kliensben megvalósított osztályok és felelősségeik:

- **ActiviyMain**

  Ennek az Activity-nek az indításával lehet elindítani az alkalmazást. Tartalmaz két Fragmentet: BrowseFagment és MyCollectionFragment. Ha nincs bejelentkezve a felhasználó, akkor kéri az AndroidOS-t, hogy jelenítse meg az ActivityLogin-t

- **ActivityLogin**

  A bejelentkező ablak elemeit tartalmazza, illetve elküldi a Communicator osztálynak a bejelentkezési adatokat.
- **BrowseFragment**

  Ez a Fragment jeleníti meg a megvásárolható albumok listáját. A BrowseAdapter osztálytól kéri el az adatokat
- **MyCollectionFragment**

  A megvásárolt albumokat jeleníti meg. Szintén van egy Adapter osztálya, ami az adatokat tárolja
- **BrowseAdapter**

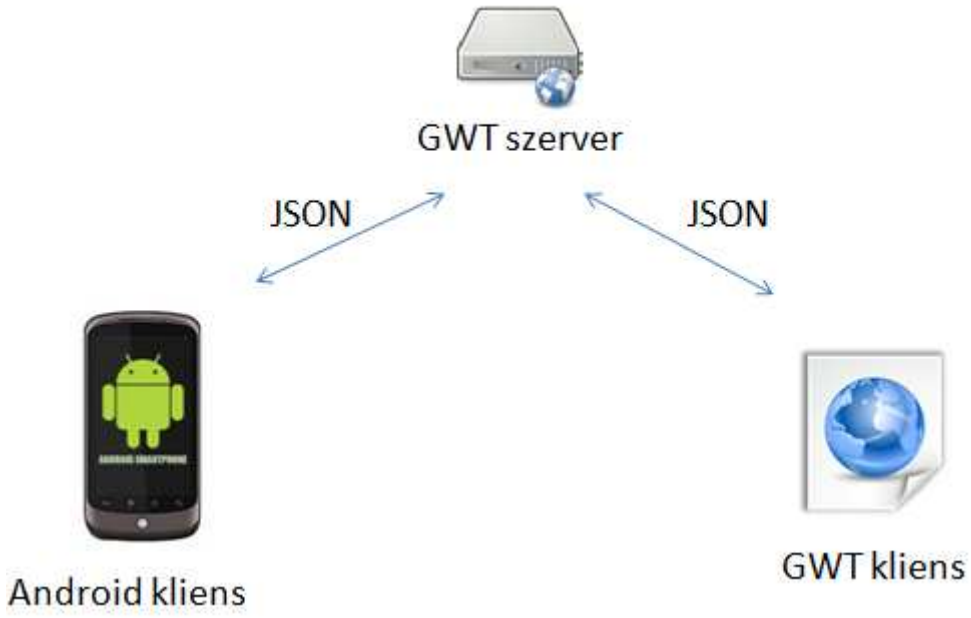
  Szükség esetén lekéri az albuminformációkat a Communicator osztályon keresztül a szerverről, és tárolja, kezeli őket.
- **MyCollectionAdapter**

  A megvásárolt albumok információit tárolja, hasonlít a BrowseAdapter osztályhoz.
- **Communicator**

  Tartalmaz egy szálat, ami a hálózati adatforgalom lebonyolítását végzi. A többi osztály ezen keresztül lép kapcsolatba a szerverrel.

3.2 Közös szerver megvalósítása

Az előző pontban történt megvalósítás során kétszer kellett megvalósítani a szerver alkalmazást. Ebben a pontban a cél az, hogy csak egy közös szerverrel kommunikáljon a két kliens alkalmazásunk.



10. ábra: A cél, hogy közös szerverrel kommunikáljanak a kliensek

Tervezői döntések:

- Az Android nem támogatja az RPC-t, így a GWT kliensében sem használható, ha közös szervert szeretnénk. Helyette olyan kliens-szerver kommunikációt választottam, ami mindkét platformon elérhető. A választás a JSON-ra esett, egyszerű használhatósága és kis sávszélesség igénye miatt. És az eddig megvalósított Android kliens is ezt használja. (GWT-ben JSNI-n keresztül érhető el, JavaScript osztályként a fogadott JSON objektum.)
- Az Android kliensben szintén megvalósításra került már egy saját session kezelés. Ezt most a GWT kliensben is meg kell valósítani.

3.2.1 Közös szerver

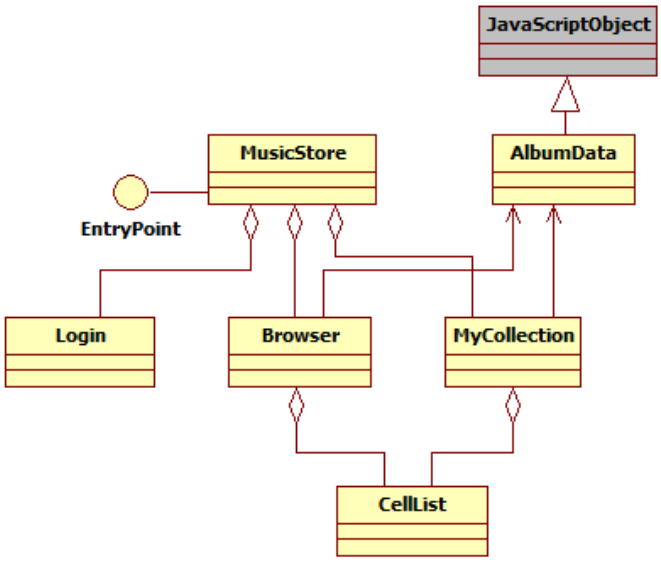
A közös szerver implementációját a fejlesztőkörnyezetben az GWT project server részében készítettem el. Az implementáció megegyezik a 3.1.2.1 pontban megvalósított servlet-tel, itt már nem használtam GWT specifikus osztálykönyvtárakat.

3.2.2 GWT kliens

A megvalósítás során a JSON formátumú fogadott adatokat ú.n. overlay osztályok segítségével érjük el[6]. A kliens kódja JavaScript nyelvre fordul, a JSON objektumok pedig JavaScript objektumok. Overhead-et jelentene őket Java objektumokká konvertálni, majd fordításkor újra, immár bonyolultabb JavaScript objektumra leképezni. [15]

A 3.1.1 pontban megvalósított kliens tehát annyiban módosul, hogy az adatokat a szervertől nem távoli eljárások meghívásával kérjük le, hanem HTTP GET kérések segítségével JSON formátumban. A fogadott JSON objektum fölé overlay osztályt készítünk. (AlbumData osztály a következő osztálydiagramon.)

Az kliens alkalmazás módusult osztálydiagramja:



11. ábra: Az alkalmazás GWT kliensének osztálydiagramja

3.2.3 Android

Jelentős változtatásokra nem volt szükség, mert már a 3.1 pontban megvalósított alkalmazás is JSON adatokat fogadott a szervertől. Azonban ellenőrizni kell a helyes működést, mert a 3.1 pontban egy szabványos servlet-tel történt a kommunikáció, most viszont egy GWT alkalmazás szerver oldalával, amit eltérően kezel az Eclipse fejlesztőkörnyezet.

[6] A GWT overlay osztályok abban segítenek, hogy már meglévő JavaScript osztályokat úgy érhessünk el és dolgozhassunk velük a Java kódból, mintha azok is Java osztályok lennének. Itt a fogadott JSON adat valójában egy JavaScript osztály. Forrás: https://developers.google.com/web-toolkit/doc/latest/tutorial/JSON

3.3 Közös kód a kliensekben

A fejlesztés során megfigyelhetjük, hogy a két kliens alkalmazásban megvalósítottuk ugyanazokat a funkciókat. Ezek között a funkciók között vannak platformfüggőek, például más felhasználói felület elemeket kellett használni, vagy használhattunk overlay osztályokat a JavaScript osztályok elfedésére. Voltak viszont platform független funkciók: a fogadott adatokat általunk megvalósított osztályok tárolják, kezelik.

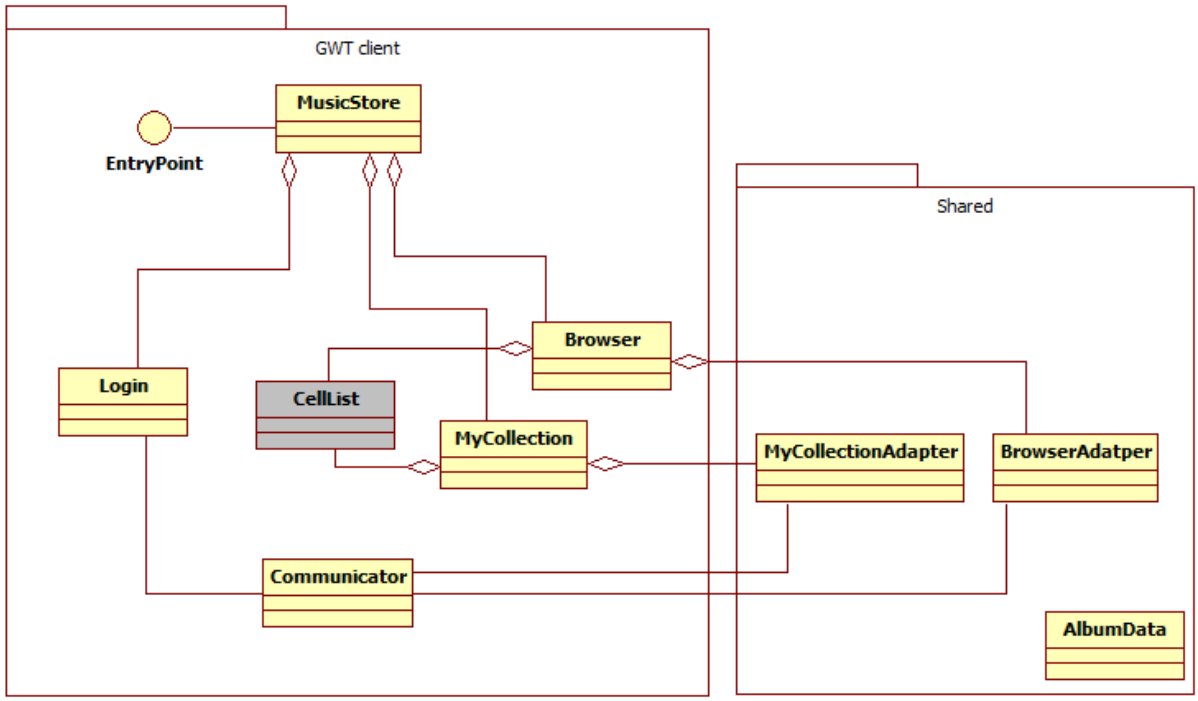
Ha a platform független kódrészleteket kiemeljük egy külön Java csomagba, csökkenthetjük a kódduplikálást.

Mivel a GWT kliens oldali része JavaScript nyelvre fordul le, nem használható a Java API minden függvénye. Az Android platform sem sztenderd JVM-en fut, hanem az ú.n. Dalvik VM-em. Tehát mindkét platform szűkíti a használható csomagok körét. Ha olyan kódot akarunk írni, ami mindkét platformon fut, ellenőrizni kell, hogy használhatóak-e.

Jelen esetben a kliens logika elég egyszerű, így probléma nélkül ki lehetett emelni.

3.3.1 GWT kliens

Az eddig megvalósított kliensben a MyCollection és Browser osztályok nem csak a megjelenítésért voltak felelősek, hanem az adatokat is ők tárolták, és lekérték a megfelelő információt a szervertől. Most azonban szétválasztjuk a megjelenítésért (View) és a belső logikáért, tárolásért, kommunikációért (Presenter) felelős osztályokat külön Java csomagokba. Ez közelítőleg megfelel az Model-View-Presenter tervezési mintának. Arra kell törekedni, hogy a Presenter-be kerüljenek azok a kódrészletek, amelyeket a 3.2-es pontban mindkét kliensben megvalósítottunk. Ennek során eltérhetünk a tervezési mintától, a fontosabb cél a közös részek megtalálása.



12. ábra: GWT kliens kimenelt közös kóddal

A módosult vagy új osztályok leírása, felelősségei:

- **Communicator:**

  A szerverrel való kommunikációt végzi. A fogadott JSON adatokat átalakítja, és feltölti velük az Adapter osztályokat

- **Browser:**

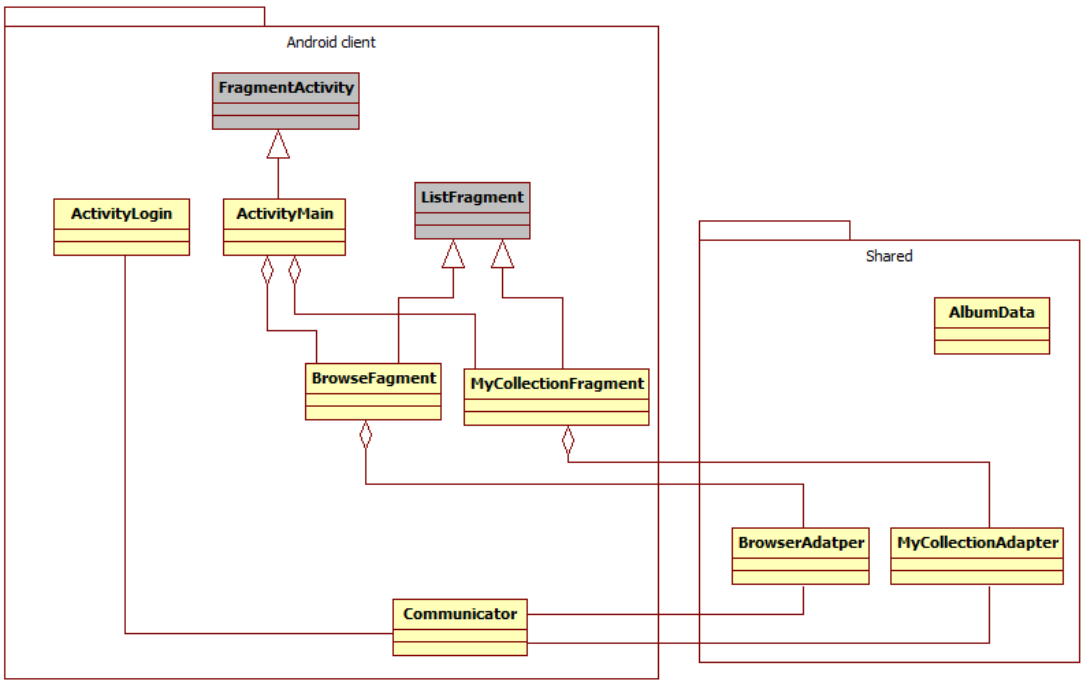
  Most már csak a megjelenítésért felelős, az Adapter osztálya (BrowserAdapter) tárolja és kezeli az általa megjelenített adatokat.

- **MyCollection:**
  Szintén szűkült a felelősségi köre, csak a megjelenítésért felelős. A megjelenítendő adatokat a MyCollectionAdapter osztálytól kéri el.
- **MyCollectionAdapter:**
  Tárolja, kezeli a megvásárol albumok információit a kliens oldalon. A megvalósításánál ügyelni kell arra, hogy a kód Android platformon is futtatható legyen.
- **BrowserAdapter:**
  A megvásárolható albumok adatait tárolja. Szintén ügyelni kell, hogy Android-on is futtatható legyen.
- **AlbumData**
  Egy album adatait tárolja.

3.3.2 Android kliens

Itt egyszerűbb dolgunk van, mint a GWT kliensnél, mert már az előző megvalósítás során külön osztályok felelőssége volt a megjelenítés és a tárolás, kezelés. Így csak külön Java csomagba kellett tenni a megfelelő osztályokat (AlbumData, BrowseAdapter, MyCollectionAdapter). Ezen osztályok implementációjának meg kell egyeznie a 3.3.1 pontban megvalósított Shared csomagban lévő osztályokéval. Tehát itt sem használhatunk Android specifikus könyvtárakat.

Az osztálydiagram a következő.



13. ábra: Android kliens kiemelt közös kóddal

4 Ajánlások, modellezés

Ha már a fejlesztés megkezdése előtt tudjuk, hogy készüli fog mindkét platformra egy verzió, akkor fontos, hogy ezt már a tervezési fázisban figyelembe vegyük. Ezzel jelentősen csökkenthető a fejlesztés erőforrásigénye. Az esettanulmányban bemutattunk különböző lehetőségeket arra, hogy miként lehet összevonni a két verzió komponenseit, csökkenteni a duplikált kódot.

4.1 Kommunikáció a szerverrel

A GWT környezet nagy előnye a beépített támogatás egy Java Servlet-eken alapuló RPC mechanizmushoz, mely hozzáférést szolgáltat a szerver erőforrásaihoz. Egy Android alkalmazásból viszont nem lehetséges meghívni a GWT távolról elérhető metódusait, mert nem készült még ilyen implementáció. Helyette HTTP GET kérésekkel érhetőek el a Java Servlet-ek szolgáltatásai. Így a szerializálást és a deszerializálást is meg kell valósítani, ha objektumokat szeretnénk küldeni illetve fogadni.

A különböző kommunikációs módok különböző Servlet implementációt igényelnek. Ha kiválasztjuk a megfelelő módot, akkor elegendő lehet egyetlen Servlet implementációt írni. A 3.2 pontban történt egy ilyen Servlet, és a vele interakcióba lépő kliensek fejlesztésének bemutatása.

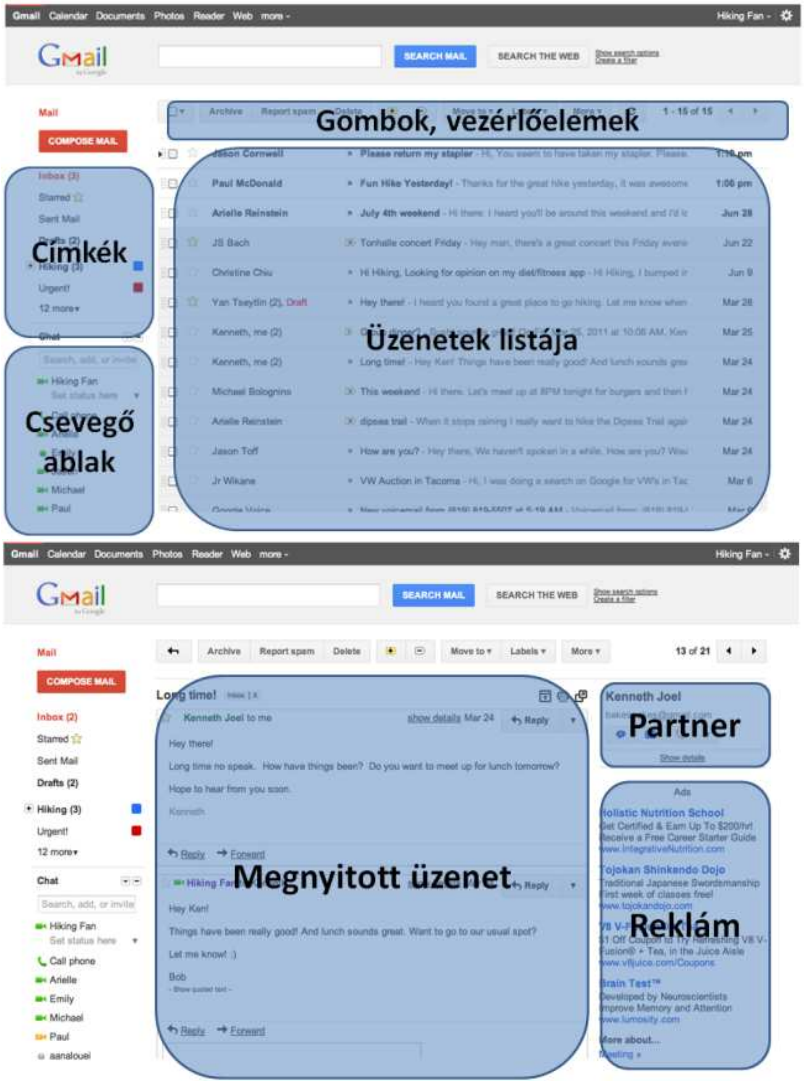
Eszerint nem használtuk a GWT-RPC-t, helyette HTTP GET kéréseket és JSON objektumokat használtunk. Így a szerver oldalon megszűnt a különböző kommunikáció miatti duplikált kód. A szolgáltatások számának növekedésével egyre indokoltabbá válik az utóbbi módszer alkalmazása, ha biztosítani akarjuk a hasonló funkcionalitást a GWT és Android kliensekben egyaránt.

4.2 A felhasználói felület elemei

Szintén tervezési fázisban, az alkalmazás kezelői felületének megszületésekor érdemes funkció szerint szétbontani kisebb elemekre az alkalmazást. Majd a kliens alkalmazások megvalósításakor ezekből az elemekből újra felépíteni a felhasználói felületet a kijelző méretéhez igazodva. Ez az alapötlete az Android táblagépekre szánt verziójának, a Fragmentek bevezetésének. Azonban ezt az ötletet alkalmazhatjuk web-alkalmazás fejlesztése esetén is.

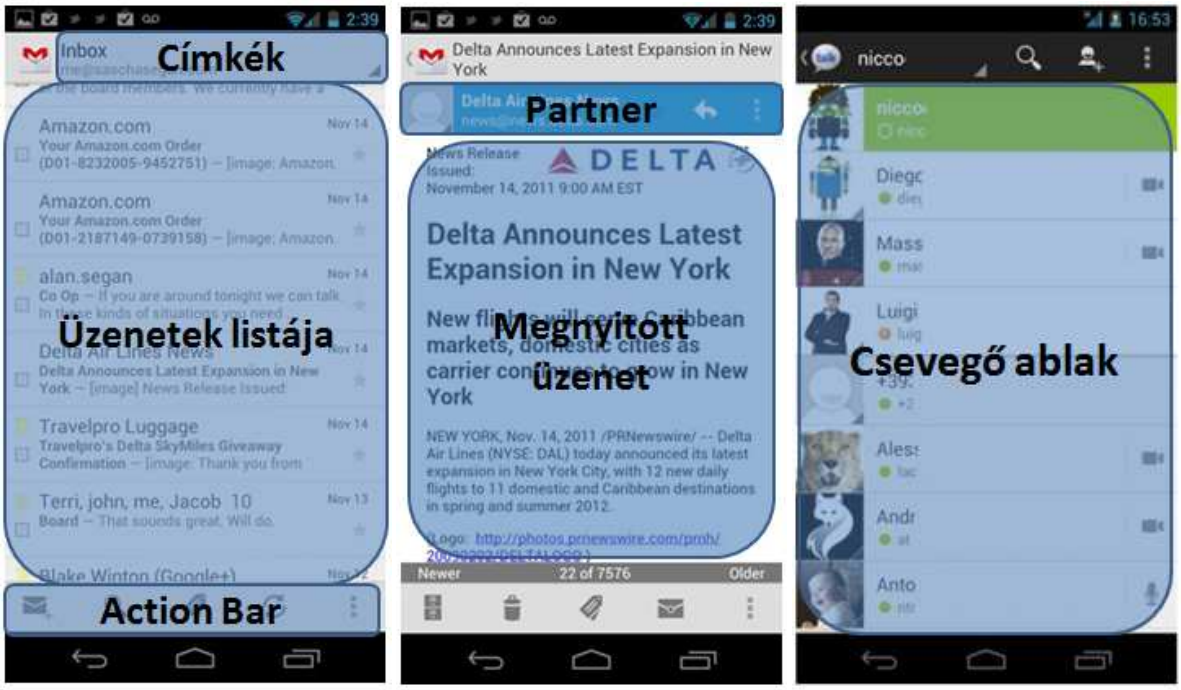
Egy példán keresztül bemutatom, hogyan bontható részekre egy web-alkalmazás és Android párja. A példa alkalmazás a Google Gmail web-alkalmazás és Android verziója:

A Gmail böngésző programban:



14. ábra: A Gmail web-alkalmazás elemekre bontása

A Gmail és Gtalk Android-on:



15. ábra: A Gmail és Gtalk Androidos alkalmazások elemekre bontása

Látható, hogy ugyanazok az elemek megtalálhatóak mindkét verzióban. Ha a kódot is eszerint csoportosítjuk, könnyen valósíthatjuk meg a két platformra a kliens alkalmazásokat kevés kódismétléssel.

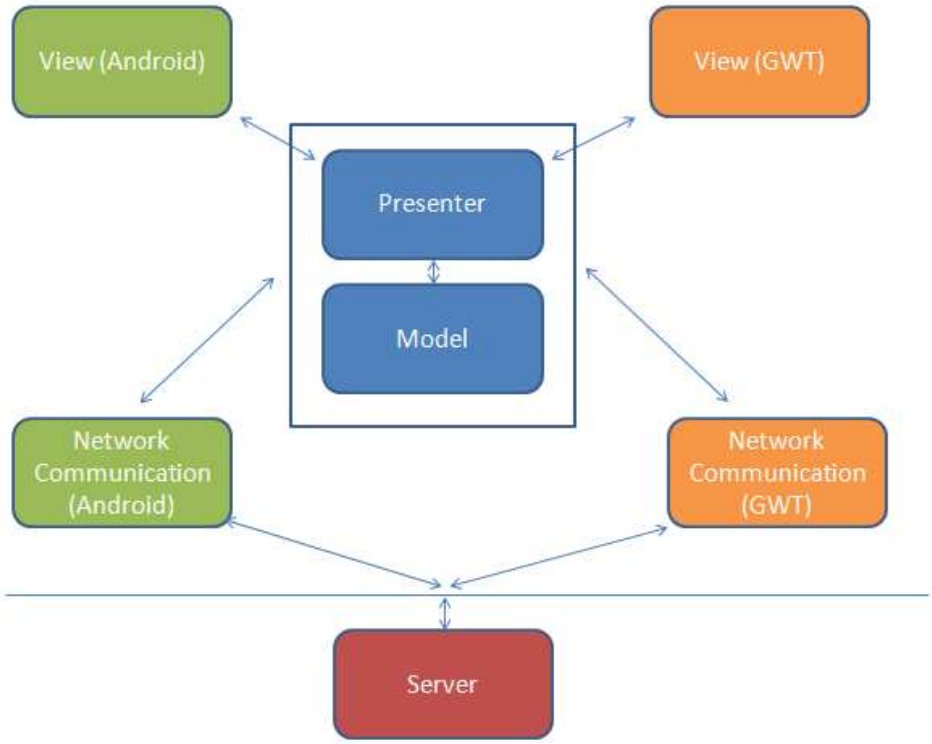
Egy-egy ilyen elemhez rendelhetünk egy olyan Java csomagot, ami mindkét platformon futtatható, és ugyanazt a feladatot látja el, beépülve a többi alkalmazáskomponens közé.

4.3 Program logika elemei

A Model-View-Presenter (MVP) [16] tervezési minta hatékonyan alkalmazható GWT web-alkalmazások fejlesztéséhez, azonban újra kell gondolni, ha szeretnénk felhasználni egy párhuzamos fejlesztés esetén.

Az esettanulmány 3.3 pontjában lévő alkalmazás (Közös kód a kliensben) osztálydiagramjait megvizsgálva láthatjuk, hogy többé-kevésbé megfeleltethetőek az osztályok az MVP tervezési minta elemeinek, ahol a közös osztályok a Presenter és Model részei. Szükség van némi kiegészítésre, mert nem alkalmazható egyértelműen a tervezési minta.

A következő ábra szemlélteti az MVP minta kiegészítését, mely már alkalmas párhuzamos fejlestésre.



16. ábra: Módosított MVP

Kék színnel jelöltem azokat az elemeket, amelyek a két platformon azonos megvalósítással rendelkeznek. Az elemek áttekintése:

- **View:** egyértelműen platform függő. Egy platformon nem jeleníthetjük meg a másik elemeit.[7]
- **Presenter:** az alkalmazás logikáját tartalmazza. Itt nem fordulhat elő platformfüggő kód, hiszen az volt a cél, hogy közös legyen a megvalósítás. Azonban elkerülhetetlen, hogy mégis szükség van platformfüggő alkalmazáslogikára. Ezeket kiemeltem külön csomagba. Az ábrán a Network Communication egy-egy ilyen elem. Ez az esettanulmányban is különböző implementációt tartalmaz a két platformon.
- **Model:** Csak adatokat tárol, a legtöbb esetben platform független.

[7] Erre van kivétel: az Android WebView eleme, amely weboldalak megjelenítésére szolgál egy alkalmazásban. Ekkor azonban nem Android építőelemekből van felépítve az alkalmazás, így a teljesítménye is romlik.

5 Mérések

A web-alkalmazások okostelefon verziójának elkészítése olyan előnyökkel jár, mint például a hardver jobb kihasználása (ezáltal jobb felhasználói élmény), kevesebb hálózati adatforgalom. Utóbbit méréssel is alátámasztom az 5.1 pontban végzett méréssel.

Egy ilyen alkalmazás utólagos elkészítése azonban erőforrás-igényes lehet. Az általam bemutatott módszer viszont csökkentheti a megírt programkódok sorainak számát (Line Of Code, LOC), ezáltal segítheti a fejlesztés folyamatát, és a karbantartási költségeket. A kód sorainak számát az 5.2 pontban vizsgáljuk.

5.1 A kommunikáció vizsgálata

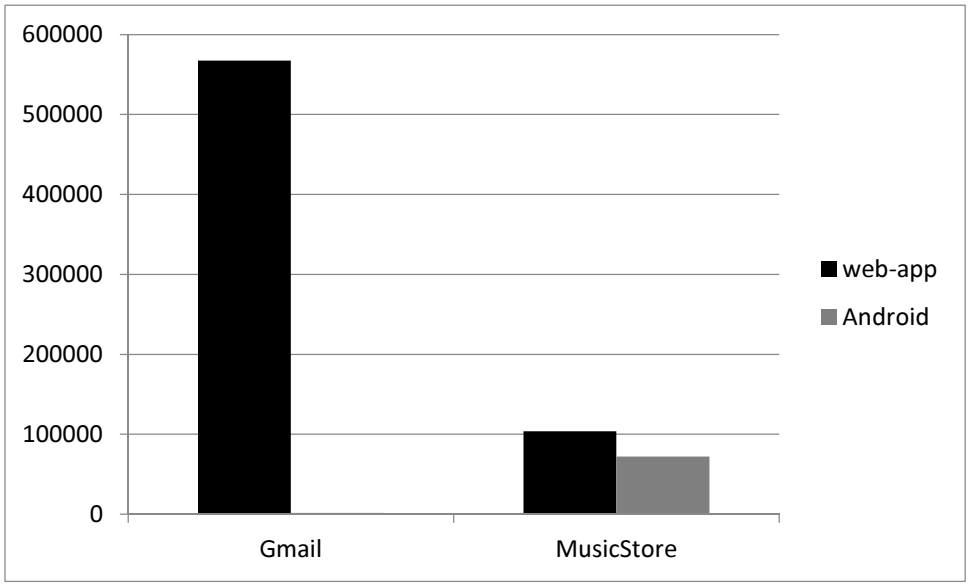
Az alkalmazások használata során a Wireshark nevű programnak a Statistics funkcióját használtam az adatforgalom vizsgálatára. A mérési elrendezés az alábbi ábrán látható.

17. ábra: Mérési elrendezés

(Megjegyzés: a web-alkalmazás adatforgalmának mérésénél a böngésző beállításait módosítottam úgy, hogy egy proxy szerveren keresztül történjen az adatküldés. Így csak a web-alkalmazás forgalmát mérte a Wireshark. A proxy szerver nem használt gyorsítótárazást.)

A mérés során azt vizsgáltam, hogy a megnyitás során mennyi adatforgalmat használnak az egyes alkalmazások az egyes platformokon.

	Gmail	**MusicStore**
web-alkalmazás	567287 byte	103852 byte
Android	1538 byte	72168 byte



18. ábra: Adatforgalom mérés eredménye byte-okban

Látható, hogy a web-alkalmazások jelentősen több adatforgalmat használnak, mivel az alkalmazás kliens oldali kódját újra le kell tölteni a megnyitáskor. A telefonon az alkalmazás telepítve van, így ott kevesebb az adatforgalom.

További oka a nagy különbségnek, hogy az Android alkalmazások tudják gyorsítótárazni a tartalmat, azaz eltárolni későbbre a kliens oldalon a már letöltött tartalmat, így nem kell újra letölteni. Erre minden alkalmazásnak van egy saját mappája az eszközön, ahová menthet. A MusicStore alkalmazás nem használja ezt a lehetőséget, így ott kisebb a különbség. (A leöltött képek mérete: 68365 byte. Ennyit lehetne megtakarítani a cache használatával.)

5.2 A kód vizsgálata

Ha külön-külön fejlesztjük ki a kliens alkalmazásokat, akkor jobban ki lehet használni a platform adta lehetőségeket, viszont megnő a szoftver karbantartási költsége, mivel két változatot is támogatni kell.

Közös fejlesztés esetén viszont a kódduplikálást minimalizálhatjuk, jobb a karbantarthatóság, csökken a fejlesztés erőforrásigénye.

A programkód vizsgálatához a megírt sorok számát vizsgáltam (LOC). Ez a módszer nem feltétlenül mutat pontos képet a fejlesztés nehézségéről, költségeiről. Az Eclipse fejlesztőkörnyezetben például igen sok lehetőségünk nyílik kódgenerálásra, például a GWT és Android projectekben is. Azonban egy alkalmazás két verziójának összehasonlítására alkalmas, kiválóan mutatja a verziók közötti különbséget. Az esettanulmányban bemutatott alkalmazás forráskódjának sorainak a számát az alábbi táblázat tartalmazza. Ezek már tartalmazzák az Eclipse által generált kódrészleteket is, de csak az src mappákból.

	GWT kliens	**Kliens megosztott**	**Android kliens**	**Server**	**Összes**
Különálló alkalmazások	1248	-	938	1038	3224
Közös szerver	1424	-	938	407	2769
Kliens tartalmaz közös kódot	1351	120	850	407	2728

19. ábra: A forráskódok sorainak száma

A táblázat adataiból azt a következtetést vonhatjuk le, hogy összesítésben csökkent a kód mérete.

A GWT kliensben a legrövidebb kód akkor keletkezett, amikor az RPC-t használtuk (1. sor). A második esetben (2. sor) lemondtunk az RPC használatáról, elég volt egy Servlet implementácó, ami összességében csökkentette a kód mennyiségét. A 3. sorban is tapasztalható némi csökkenés, a közös kód kiemelésének hatására.

Mivel az esettanulmányban megvalósított kliens alkalmazások kevés üzleti logikát tartamaztak (főként a megjelenítést, és kommunikációt kellett implementálni), kicsi volt a kiemelhető kód mennyisége, ami csökkenti az összesen megírt kódsorok számát.. Viszont így is megfigyelhető a verziók közötti különbség.

6 Összefoglalás

Az okostelefonok gyors terjedésének köszönhetően egyre népszerűbbek az okostelefon-alkalmazások. Egy-egy weboldalnak általában elérhető okostelefon-alkalmazás változata is.

Kiválasztottunk egy népszerű okostelefon platformot, és egy web-alkalmazás platformot: az Android-ot és a Google Web Toolkit-et. Közös tulajdonságuk, hogy Java nyelven programozhatók, és modern web-alkalmazások fejleszthetőek segítségükkel.

Célunk az volt, hogy megvizsgáljuk annak lehetőségét, hogyan lehet párhuzamosan fejleszteni a két platformra, hogyan lehet egyszerűsíteni a fejlesztés menetét és csökkentve a kódduplikálást, ahol lehet közös kódot használni a két platformra készített alkalmazásban.

A 3. fejezetben egy esettanulmányon keresztül bemutattam, hogyan zajlik a fejlesztés, ha a két platformra teljesen különálló alkalmazást fejlesztünk. Utána megvizsgáltuk, hogyan lehet átalakítani, hogy a kliens alkalmazások ugyanazzal a szerver alkalmazással kommunikáljanak ugyanolyan kommunikációs módot használva. Végül pedig megnéztük annak lehetőségét, hogyan tudunk közös implemetációt használni a két kliens alkalmazásban. Modelleztük az egyes verziókat.

A 4. fejezetben általánosan is megfogalmaztam az esettanulmány alkalmazásának fejlesztése során szerzett tapasztalatokat. Egy alkalmazás kezelői felületének tervezésére fogalmaztam meg ajánlásokat, melyek a kód átláthatóságára is pozitív befolyással vannak. Felállítottam egy modellt (az MVP tervezési minta módosításával), aminek alkalmazása megkönnyíti a fejlesztést.

Az 5. fejezetben méréseket végeztem az alkalmazások adatforgalom-használatát és a megírt kód mennyiségét illetően. Ezek a mérések alátámasztják, hogy okostelefonon előnyökkel jár az alkalmazások használata a weboldalak használatával szemben, és a bemutatott módszer jelentősen csökkentheti a fejlesztés költségeit. Bár ez megköveteli bizonyos szabályok betartását, mégis az esetek többségében indokolt lehet a módszer használata.

A lehetséges további kutatási irányok között említeném a GWT HTML5 támogatásának vizsgálatát, és hasonló funkcionalitással bíró Android alkalmazás fejlesztését. A további mobilplatformokra történő kiterjesztés vizsgálatát szintén

fontosnak tartom, hiszen az Android nem egyeduralkodó a kategóriában. Jobb kommunikáció megvalósítása Android platformon, például távoli eljáráshívás használatával.

7 Köszönetnyilvánítás

Köszönettel tartozom konzulensemnek, Dr. Goldschmidt Balázsnak, akihez bármikor fordulhattam kérdéseimmel, hasznos tanácsaival mindvégig segítette munkámat.

Szeretném megköszönni az Irányítástechnika és Informatika Tanszéknek, hogy biztosítottak számomra az Android-os fejlesztés nagymértékű megkönnyítésére egy Google Nexus S telefont.

8 Irodalomjegyzék

[1] *Andre Charland, Brian Leroux* Mobile application development: web vs. native [Online] – 2012. október 26. - http://dl.acm.org/citation.cfm?id=1941504

[2] Reducing the Cost of Web Site Development and Maintenance [Online] – 2012. október 26. - http://www.ctg.albany.edu/publications/guides/roi?chapter=5&PrintVersion=2

[3] Web Application Description Language [Online] – 2012.október 26. - http://www.w3.org/Submission/wadl/

[4] History of the World Wide Web [Online] – 2012. október 26.- http://www.nethistory.info/History%20of%20the%20Internet/web.html

[5] Jim Conallen, Rational Software - Modelling Web Application Architectures with UML [Online] – 2012. október 26. http://www.deetc.isel.ipl.pt/Programacao/Programacao_Inv_2004_2005/ti/Documentacao/webapps.pdf

[6] Gartner Says Worldwide Sales of Mobile Phones Declined 2.3 Percent in Second Quarter of 2012 [Online] – 2012. október 26. http://www.gartner.com/it/page.jsp?id=2120015

[7] *Ekler Péter, Fehér Marcell, Forstner Bertalan, Kelényi Imre* Android-alapú szoftverfejlesztés [Könyv]. - 2012.

[8] JavaScript [Online] – 2012. októbe 26. - https://developer.mozilla.org/en-US/docs/JavaScript

[9] AJAX Tutorial [Online] – 2012. október 26. - http://www.w3schools.com/ajax/default.asp

[10] Google Web Toolkit [Online] – 2012. október 26. - https://developers.google.com/web-toolkit/

[11] Remote Procedure Calls (RPC) [Online] – 2012. október 26. - http://www.cs.cf.ac.uk/Dave/C/node33.html

[12] Using Servlet Sessions in GWT [Online] – 2012. október 26. - http://developerlife.com/tutorials/?p=230

[13] Introducing JSON [Online].- 2012. október 26.- http://json.org/json-hu.html.

[14] Google Gson [Online]. - 2012. október 26. - https://sites.google.com/site/gson/

[15] Retrieving JSON Data [Online] – 2012. október 26. - http://msdn.microsoft.com/en-us/magazine/cc188690.aspx

[16] Design Patterns : Model-View-Presenter [Online] – 2012. október 26. - http://msdn.microsoft.com/en-us/magazine/cc188690.aspx