

# SZAKDOLGOZAT

Lakatos Gergely

2014

Pannon Egyetem

Rendszer- és Számítástudományi Tanszék

Gazdaságinformatikus BSc

## SZAKDOLGOZAT

Androidos alkalmazás fejlesztése workflow  
menedzsment rendszerhez

Lakatos Gergely

Témavezető: Frits Márton

2014

## Feladatkírás

## Nyilatkozat

Alulírott Lakatos Gergely hallgató, kijelentem, hogy a dolgozatot a Pannon Egyetem Rendszer- és Számítástudományi tanszékén készítettem a gazdaságinformatikus végzettség megszerzése érdekében.

Kijelentem, hogy a dolgozatban lévő érdemi rész saját munkám eredménye, az érdemi részen kívül csak a hivatkozott forrásokat (szakirodalom, eszközök, stb.) használtam fel.

Tudomásul veszem, hogy a dolgozatban foglalt eredményeket a Pannon Egyetem, valamint a feladatot kiíró szervezeti egység saját céljaira szabadon felhasználhatja.

Veszprém, 2014. december 10.

.....

Lakatos Gergely

Alulírott Frits Márton témavezető kijelentem, hogy a dolgozatot Lakatos Gergely a Pannon Egyetem Rendszer- és Számítástudományi tanszékén készítette gazdaságinformatikus végzettség megszerzése érdekében.

Kijelentem, hogy a dolgozat védelemre bocsátását engedélyezem.

Veszprém 2014. december 10.

.....

Frits Márton

## **Köszönetnyilvánítás**

Szeretném megköszönni témavezetőmnek, Frits Mártonnak a projekt elkészítése során nyújtott segítséget és folyamatos iránymutatást.

Köszönöm továbbá családomnak és barátaimnak a támogatásukat a szakdolgozat elkészítéséhez.

## TARTALMI ÖSSZEFOGLALÓ

Az okos-telefonok és a tabletek rohamos mértékű fejlődéséből, és népszerűségéből adódóan, manapság már egy átlagos személy több időt tölt a készüléke használatával, mint a számítógépe előtt. Ennek következtében nagy rendszerekhez, amit napi rendszerességgel, sokan használnak, elengedhetetlen fontossággal bír olyan kliensprogram készítése, ami okos telefonokon futatható.

Az alkalmazásom egy olyan workflow menedzsment rendszerhez készült, amivel könnyedén lehet munkafolyamatokat létrehozni és végrehajtani. A munkafolyamat egyes lépéseihez űrlapok kapcsolódnak, amelyeket az adott munkafolyamat résztvevői ki tudnak tölteni, és az arra adott válaszok függvényében a munkafolyamat tovább tud haladni.

Az applikációm ezeket az űrlapokat tudja strukturált üzenet formájában fogadni, visszaküldeni, illetve dinamikusan generálva megjeleníteni. Maguk az űrlapok tartalmazhatnak speciális részeket is (számoló mező, nagyméretű kép), amiket az alkalmazásnak tudnia kell kezelni. Ezen felül az adatlapok leadása előtt ellenőriznie kell tudnia, az űrlap kitöltésének helyességét. Továbbá az alkalmazásnak biztosítania kell az űrlap leadása mellett az űrlap mentését is.

Ez a dokumentum bemutatja azokat a technológiákat, amiket felhasználtam a fejlesztés során, illetve vázolja az alkalmazás specifikációját, ezen felül részletes betekintést ad az alkalmazás és a teszt szerver tervezési, fejlesztési és tesztelési folyamatába.

**Kulcsszavak:** Android alkalmazás, Workflow, Űrlapkezelés

## **ABSTRACT**

In the consequence of the rapid development and popularity of the smart phones and tablets, nowadays an average person uses his gadget rather than his personal computer. For this reason, huge systems, that are used every day by increasing number of people, bring the indispensable necessity of the implementation of client-application, what can be run by smartphones.

My application was made for a workflow management system and by the help of this system one can create workflows with ease. The pieces of the workflow include forms, which can be filled out by the participants of the given task. Moreover as a function of the answers of the task the workflow can proceed further.

The structured messages -which contain the forms-, can be received, sent back respectively and visualized dynamically by the application. The forms may include special segments (computable field, large-sized picture) which had to be handled by the application. Furthermore it must be able to check the correctness of the filling of the form before the transmission. In addition it has to secure the saving of the forms besides the transmission.

This document introduces the technologies what were used for the implementation, furthermore presents the design, the implementation and the testing of the application and the test server

**Key Words: Android application, Workflow, Form-management**

## Tartalomjegyzék

Feladatkiírás .....	2
Nyilatkozat .....	3
Köszönetnyilvánítás .....	4
TARTALMI ÖSSZEFOGLALÓ .....	5
ABSTRACT .....	6
Tartalomjegyzék .....	7
1. Bevezetés .....	10
2. Workflow Management System projekt.....	12
2.1 A Workflow projekt.....	12
2.2 Workflow Management System architektúrája .....	13
3. Használt technológiák .....	15
3.1 Android .....	15
3.1.1 Android verziók.....	15
3.1.2 Android architektúrája.....	16
3.1.3 Android felületek életciklusa.....	17
3.2 JSON.....	19
3.3 PHP .....	19
3.4 MySQL .....	20
4. Specifikáció .....	21
4.1 Funkcionális követelmények .....	21
5. Tervezés.....	23
5.1 Alkalmazás felületei .....	23
5.1.1 Nyitó felület.....	23
5.1.2 Főoldal.....	23



## Tartalomjegyzék

5.1.3	Űrlapok listázása .....	24
5.1.4	Űrlap megjelenítése .....	24
5.2	Szerverkapcsolat .....	25
5.2.1	Kapcsolatoptimalizálás .....	25
5.2.2	Üzenet architektúrája .....	26
5.3	Védelem, adatok kezelése .....	27
5.4	Folyamatok .....	27
5.4.1	Bejelentkezés .....	27
5.4.2	Űrlap megjelenítés .....	29
5.5	Űrlap felépítése, megjelenítése .....	30
5.6	Űrlap adatmodell .....	31
5.7	Adatbázismodell .....	32
6.	Fejlesztés .....	34
6.1	Alkalmazás felületei .....	34
6.1.1	Nyitó felület .....	35
6.1.2	Fő oldal .....	37
6.1.3	Űrlapok listázása .....	38
6.1.4	Űrlapok megjelenítése .....	40
6.1.5	Nyelvbeállítás, kijelentkezés .....	42
6.2	Szerverkapcsolat .....	43
6.2.1	Üzenetek .....	43
6.2.2	Szerverkapcsolat létrehozása .....	44
6.2.3	Űrlapok parzolása .....	45
6.3	Űrlap generálása .....	46
6.4	Űrlapegységek generálása .....	46
6.4.1	Címke .....	47

## Tartalomjegyzék

6.4.2	Szövegmező.....	47
6.4.3	Beviteli mező.....	47
6.4.4	Jelölőnégyzet .....	48
6.4.5	Rádiógomb-csoport .....	48
6.4.6	Kép .....	48
6.4.7	Számolómező .....	49
6.5	Űrlap mentése .....	52
6.5.1	Űrlap beviteli mezőinek ellenőrzése .....	52
6.6	Segédszerver megvalósítása .....	53
6.6.1	Adatbázis kapcsolat megvalósítása .....	53
6.6.2	Bejelentkeztetés, kijelentkeztetés .....	54
6.6.3	Űrlapok menedzselése .....	55
7.	Tesztelés .....	59
7.1	Alkalmazás tesztelése .....	59
7.1.1	Monkey teszt .....	59
7.1.2	Felhasználói teszt .....	59
7.2	Segéd szerver tesztelése .....	60
8.	Összefoglalás .....	61
	Irodalomjegyzék.....	62
	Ábrajegyzék.....	63
	CD melléklet.....	64

## 1. Bevezetés

A szakdolgozatom témája egy olyan Android alkalmazás fejlesztése, ami a Workflow management rendszer felhasználóinak teszi kényelmesebbé, és gyorsabbá a hozzájuk rendelt feladatok kezelését.

A Workflow rendszer, egy projektek menedzselésére készülő rendszer. A Workflow fő célja hogy a projekteket létrehozó, és kezelő menedzserek munkáját megkönnyítse, azáltal hogy átlátható, könnyen kezelhető felületeket biztosít a projektek munkafolyamatainak összerakásához. A munkafolyamataihoz tartozó részfeladatokhoz különböző munkatársakat lehet hozzárendelni, akiket az adott feladathoz tartozó munkákról illetve döntésekről értesíteni kell. Ezek az értesítések, űrlapok formájában kerülnek konstruálásra. A munkafolyamat gördülékeny haladásának érdekében, a rendszer célja az, hogy a felhasználók ezeket az űrlapokat a lehető leggyorsabban, és a legegyszerűbb úton megkapják, és kitöltsék.

A technológia rohamos fejlődésével, mára a háttérbeszoruló számítógépek helyét átvették az okos telefonok, tabletek. Ezek a készülékek, kisebb helyen elférnek, egyszerűbben és gyorsabban lehet őket kezelni, mint a teljesítményben közel azonos laptopokat, asztali gépeket. Ezen okokból kifolyólag a Workflow rendszer felhasználóknak szánt kliens programját, célszerű volt okos telefonokon futtatható applikációként létrehozni. Azért írtam a szakdolgozatom applikációját Androidos készülékekhez, mert jelenleg a mobilpiac három vezető operációs rendszere közül az Android birtokolja a legnagyobb piaci részesedést.

Az alkalmazás fő feladata, hogy a workflow rendszer által előállított, munkafolyamatok részfeladataihoz tartozó, űrlapok formájában kiosztott alfeladatokat, és döntéseket a felhasználó minél előbb megkapja, és minél egyszerűbben, és gyorsabban arra reagálni tudjon. A workflow rendszeren keresztül definiált munkafolyamatok komplexitása miatt, több fajta űrlapot kell tudnia az alkalmazásnak kezelni. Egy űrlap lehet csak egy elolvasandó dokumentum, de lehet egy szűk határidős gyors reakciót igénylő szerződés is. Ebből kifolyólag az űrlapfajták rugalmas kezelése fontos szerepet játszik az

alkalmazás működésében. Az űrlapok tartalmazhatnak titkos információkat, mik nem kerülhetnek illetéktelen kezekbe, ezért az adatok kezelése, védelme is prioritást élvez.

A szakdolgozatom megírása ideje alatt még nem készült el teljes egészében a workflow rendszer interfész szervere (ami szükséges a szerverkapcsolat és az űrlapok lekérésének és mentésének teszteléséhez) ezért az alkalmazáson kívül egy segéd szervert is készítettem az alkalmazás tesztelésének céljából.

## **2. Workflow Management System projekt**

Ebben a fejezetben röviden összefoglalásra kerül a Workflow Management System működése, és az Android alkalmazás szerepe a rendszerben.

### **2.1 A Workflow projekt**

A projekt célja egy olyan rendszer fejlesztése, amelynek használatával a felhasználó képes BPMN szabványos jelölésrendszert használó munkafolyamatot létrehozni, illetve maga a rendszer tud ilyen munkafolyamatokat értelmezni, a saját workflow motorja segítségével (Workflow Engine, WFE). A felhasználóknak (a szerepköreiktől függően), munkafolyamatok létrehozása és szerkesztése közben lehetőség nyílik manuálisan definiálni a munkafolyamatokhoz tartozó erőforrásokat, a folyamat részfeladataihoz tartozó jogosultságokat és függőségeket. Ezen felül ezeket a folyamatokat illetve a folyamatokhoz tartozó erőforrásokat lehet a rendszer segítségével optimalizálni, generálni.[1]

Maga a workflow rendszer két részegysége foglalkozik a felhasználói döntések támogatásával, az egyik az ágens szimulációs modul (Agent Simulation Module, ASM), a másik a naplóinformációk feldolgozására szolgáló modul (Log Analysis Module, LAM). Az ASM fő feladata hogy az adott munkafolyamatot lefuttassa, leszimulálja, és ajánlásokat tegyen a kétes, kényes erőforrás hozzárendeléseknél. A LAM modul pedig a naplózott adatok alapján próbálja az erőforrások paramétereit beállítani. A felhasználóknak a megvalósítandó és már futó folyamatok megtekintéséhez, szimulálásához, kezeléséhez, külön web alapú felület (speciális weboldal: Task List Manager, TLM) kerül fejlesztésre, ami a folyamatok lehetséges és szükséges felhasználói interakciókat képes kezelni. Minden műveleti egység dinamikusan előállított (generált) űrlapok segítségével oszthatóak ki a felhasználók között. Ezeknek az űrlapoknak definiálni lehet a célját (olvasás, kitöltés, letöltés stb..) és a tartalmát (kép, szöveg, beviteli mező stb..). Az űrlapok elkészítése a részfeladatokhoz kapcsolódó tulajdonságok (mint például jogosultságok) alapján az LAM modul bevonásával történik. A LAM modulra azért van szükség az űrlapok elkészítésénél, mivel ez a modul a log

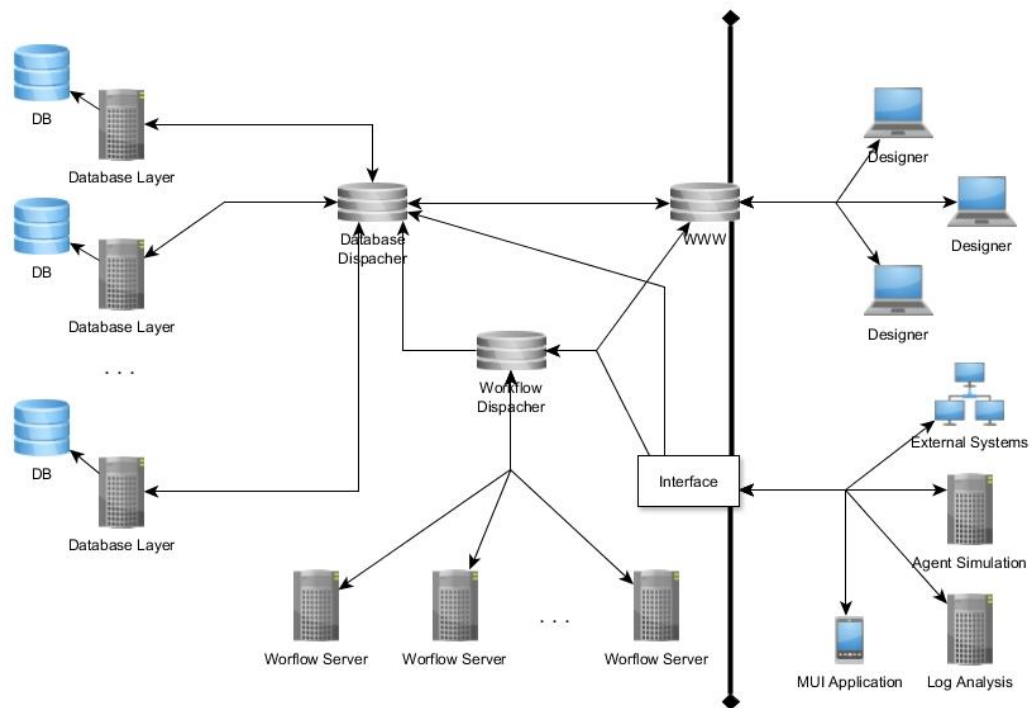
fájlok alapján meg tudja határozni, kik azok a munkavállalók (felhasználók), akik a legalkalmasabbak az adott munkára, tehát leghatékonyabban tudják ellátni az adott részfeladatot. Az űrlapok generálása után kiosztása kerülnek az adott felhasználóknak, amikre adott válaszok feldolgozása után összeáll a teljes folyamat. Az űrlapokat a felhasználók webes felületen és mobilalkalmazás segítségével tekinthetik meg, tölthetik ki.[1]

### **2.2 Workflow Management System architektúrája**

A rendszer architektúrájának kialakításnak az az alapelve, hogy a rendszert dinamikusan lehessen bővíteni, leépíteni a kihasználtságtól függően. A szerverek egymás között kommunikációja websocket protokoll segítségével, illetve egy erre a kommunikációra kifejlesztett JSON alapú csomagok használatával kerül megvalósításra. Egy ilyen üzenet (csomag) magában foglalja azokat az információkat, amelyek segítségével egyértelműsíthető melyik szervernek kerül majd kiosztásra az adott feladat.

Magát a WMS rendszer architektúrája az 2-1. ábra mutatja be. A függőleges fekete vonal jelképezi a rendszer fizikai határát, amelyen található http és interfész szerver. Ez a két szerver biztosítja a kapcsolatot a külvilággal. Míg a http szerver (www) a web böngészőn keresztül elérhető szolgáltatásokért felelős, addig az interfész szerver, felelős a külső, más rendszerekkel és szerverekkel történő kommunikáció lebonyolításáért. A MUI applikáció is ehhez a szerverhez kapcsolódva folytat kommunikációt a rendszerrel. Ez a szerver támogatja websocket kapcsolatot és a POSTGET kapcsolatot egyaránt. Az Android alkalmazás az utóbbit használva fogja kiépíteni a kapcsolatot a WMS rendszerrel. [2]

## Workflow Management System projekt



**2-1. ábra: Workflow rendszer architektúrája**

### 3. Használt technológiák

Ebben a fejezetben bemutatásra kerülnek azok a technológiák, amiket a szakdolgozatom megírása alatt alkalmaztam. Először magát az Android operációs rendszert ismertetem, ezt követően a segédszervernél alkalmazott technológiákat fejtem ki.

#### 3.1 Android

Az Android egy mobil operációs rendszer, ami főként érintőképernyős készülékekhez készült. Az operációs rendszer teljes mértékben összefonódott a Google fiókkal, hogy ahhoz hogy az operációs rendszert használni lehessen, a felhasználónak Google fiókkal kell, hogy rendelkezzenek. Ez a fajta Google-függés, nagy előnyt jelent a felhasználó számára, hiszen az Android szinkronizálja a már meglévő Google szolgáltatásokkal a készüléket, így egyből megjelenítésre kerülnek a leveleink, a böngészési előzményeink, a felhőszolgáltatásunk, illetve a Google naptárunk a telefonon. Új alkalmazásokat a GooglePlay-en (egy előre telepített alkalmazáson) keresztül tölthetjük le, mihez szintén szükséges a Google fiók.[3,4]

Az Android alkalmazást az Android Inc. vállalat kezdte el fejleszteni, mit a Google 2005-ben felvásárolt. A felvásárlást követő harmadik évben megjelent az első Android operációs rendszert használó telefon. Ezt követően az operációs rendszer agresszív terjeszkedésbe kezdett, és 2013-ra piacvezető szerepkört ért el a mobil piacon. Az Android jelenleg is listavezető, és olyan új iparágakba próbál betörni, mint az autóipar.[3,4,5]

##### 3.1.1 Android verziók

2009 óta az Android operációs rendszernek 12 (többnyire édességekről elnevezett) verziója jelent meg. A jelenleg aktív, forgalomban lévő készülékek Android operációs rendszert használó eszközök piaci részesedése:

- Froyo (2.2): 0.5%
- Gingerbread (2.3.3- 2.3.7): 9.1%



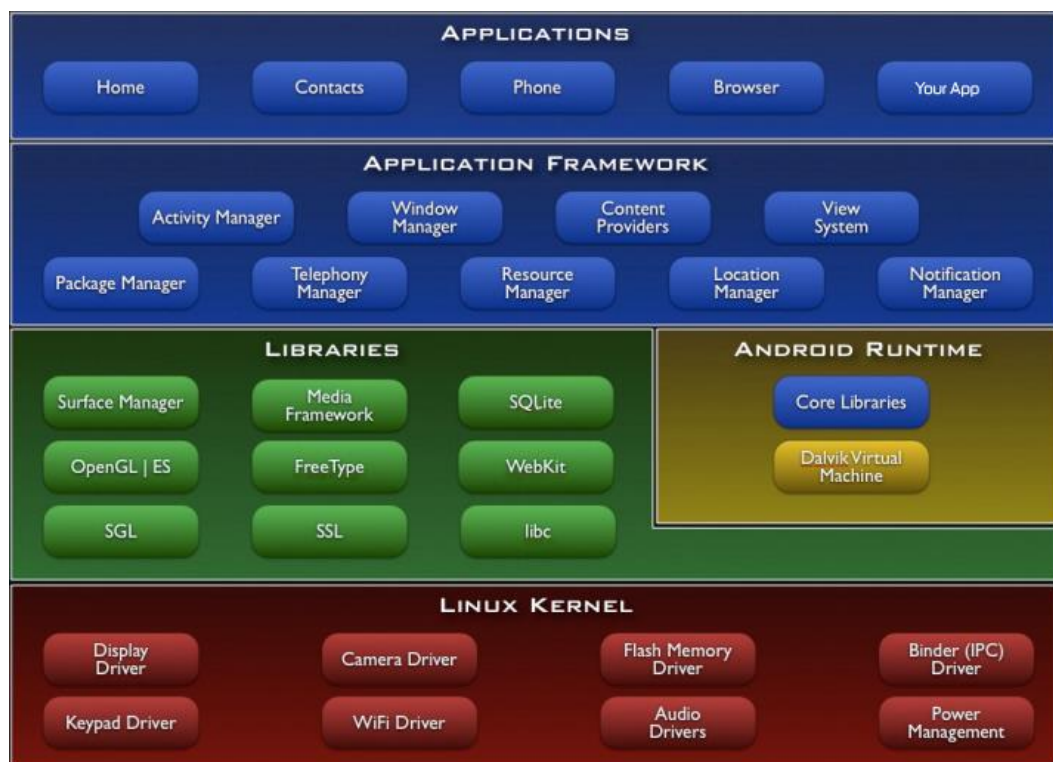
- Ice Cream Sandwich (4.0.3-4.0.4): 7.8%
- Jelly Bean (4.1.x ,4.2.x, 4.3.x): 48.7%
- Kitkat (4.4.x) piaci részesedése: 33.9%

Az eloszlásból az látszik, hogy az aktív készülékek több mint 80% már 4.0 feletti Android verziót használ, míg csak olyan 10% körüli azoknak a mobiltelefonoknak az aránya, akik régi verziót futtatnak (2.3.6 vagy ez alatti). Azok az alkalmazások, amik régi telefonokra lettek íródva elindulnak az új rendszert futtató készülékeken, viszont az új Androidra írt alkalmazások nem futtathatók a régebbi készülékeken. Ez a fejlesztőknek azt jelenti, hogy ha egy olyan alkalmazást akarnak írni, ami (szinte) minden aktív, Android operációs rendszert használó telefonon fusson, érdemes API 10-es szinttől fejleszteni. [6]

A 10-es és a 15-ös API szint közötti a különbség jelentős, sokkal körülményesebb régebbi verziót használó készülékekre fejleszteni, mivel azok nem támogatnak több később bevezetett változtatásokat, újításokat.

### 3.1.2 Android architektúrája

Az Android, Linux kernel alapú operációs rendszer, aminek elsődleges célja, feladata az alap hardverek (WIFI, hangerőszabályzó, képernyő, Bluetooth, stb.) hibamentes működtetése. Ezeket a alap kernel-programokat a készülék gyártói írják meg, hiszen szinte minden készülékbe különböző hardverek kerülnek. A kernelbe található programokat a felette elhelyezkedő könyvtárak (például SQLite vagy Webkit) használják, azokat a programokat, amik közvetlenül a kernelbe futnak. Főként ezeket a könyvtárak használatával működik a Dalvik virtuális gép, minek egyedüli célja az Android alkalmazások futtathatóvá tétele, és futtatása. A legfelső két kék rétegben találhatóak azok a Java alkalmazások, amik adják az Android egészét, itt található az összes alkalmazás, amivel a felhasználó az Android használata közben találkozhat. Magát az architektúrát a 3-1. ábra mutatja be.[3,4,7]



3-1. ábra: Android architektúrája

### 3.1.3 Android felületek életciklusa

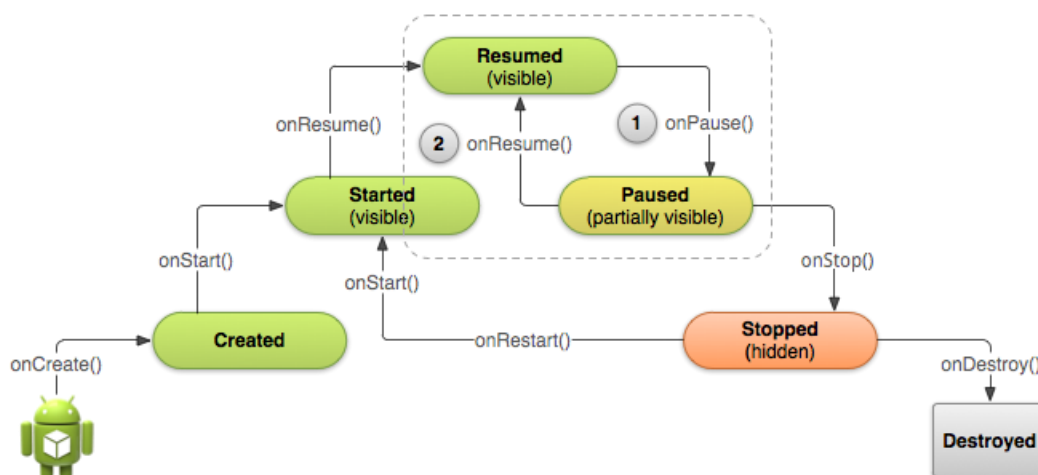
Android alkalmazás felületei *activity*-ei nem teljesen egyeznek meg az asztali gépeken futó operációs rendszerek ablakaikkal, mert ezekhez a felületekhez nem tartozik ablakkezelő rendszer. Egy adott alkalmazás több lazán kapcsolódó felületből (ablakból) áll össze, amik között speciális navigációk vannak definiálva.[8]

Ez azt jelenti, hogy minden alkalmazás felülethez meg lehet határozni olyan eseményeket, amelyek bekövetkezésekor új felületeket indítson el (az adott felület). Viszont ha egy új oldalt (*activity*-t) indítunk, akkor az előző működése leáll, de nem szűnik meg, mert a rendszer megőrzi a *Back Stack* tárolójában. Ez a speciális tároló, a veremtárolóhoz hasonló jellemzőkkel rendelkezik, tehát ha egy elindított felületen back gombot nyomunk, akkor a *Back Stack*-ból kikerül az utoljára betett felület, és folytatja működését ott ahol abbahagyta. Az Activity mikor egy másik miatt elindul, vagy leáll, akkor egy *callback* értesítést kap az eseményről. Ebből következik, hogy ezektől a *callback* eseményektől függően az

*activity* reagálhat ezekre a történésekre. Az ilyen a feladatok implementálására lettek definiálva a következő metódusok:

- *onCreate*: az *activity* első létrehozásánál hívódik meg.
- *onStart*: az *onCreate* vagy az *onRestart* után fut le a benne lévő programkód.
- *onResume*: akkor fut le, mikor az alkalmazás (újra) előtérbe kerül, láthatóvá válik.
- *onPause*: abban az esetben hívódik meg, amikor az oldal háttérbe kerül (a *Back Stack*-be), mert valamilyen felugró ablak kitakarja
- *onStop*: mikor az oldal egyáltalán nem látható, mert a *Back Stack*-be kerül, akkor fut le ez a függvény.
- *onRestart*: az *onStop* leállítás után, de még az *onStart* lefutása előtt meghívódó függvény.
- *onDestroy*: A felület végleges leállítása, és törlése esetében hívódik meg.

A 3-2.-es ábra egy Android oldal életciklusát illusztrálja. [8][9]



3-2. ábra: Android oldalak életciklusai

Az alkalmazás *activity*-jeinek pontos életciklusának meghatározása a miatt kulcsfontosságú, mivel azok a háttérbe került felületek, amiket a rendszer nem talál fontosnak, leállíthatja, ha túlságosan is nagy helyet foglal a memóriában.

### 3.2 JSON

A JSON egy programnyelv-független, szöveg alapú adatcserére kitalált szabvány, ami kevés többletkezelő karakter segítségével ír le strukturált adatokat (objektumokat, attribútumokat, tömböket). A nyelvet 2001-ben Douglas Crockford fejlesztette ki és kezdte el népszerűsíteni. Öt év alatt olyan gyorsan növekedett a népszerűsége, hogy, a 2006-ban a Google GData webes protokollja már JSON nyelvet használt. Ennek a leíró nyelvnek egy alternatívája az XML, mi hasonló célt szolgál, viszont nyelvezete nem olyan tömör, mint a JSON-é. [10]

A szakdolgozatomhoz azért választottam a JSON leíró nyelvet, mert manapság azt használják a legtöbben, illetve a JAVA programnyelv támogatja annak egyszerű feldolgozását.

### 3.3 PHP

A PHP egy dinamikus tartalmakat használó weblapok megvalósítására szolgáló szerver oldali szkriptnyelv. [11]

Elsősorban egyszerű makró készletnek tervezték, még 1995-ben, ami C nyelven íródott, és képes volt primitív weboldalak generálására és adatbázis-kapcsolat kialakítására. Az első kiadásban sok apró hibát találtak, amik javítása után létrejött a PHP 2. Ez a nyelv már űrlapok készítésére, kezelésére is képes volt. Két évre rá két izraeli fejlesztő (Andi Gutmans, Zeev Suraski), újradefiniálta és implementálta a PHP alapját, ezzel megteremtve a ma használt PHP alapjait a PHP 3-at. (Ebben időben alapították a Zend Technologies céget, ami a mai napig ellenőrzi a PHP nyelv fejlesztését.) A Zend Technologies ezt követően újraírta a PHP magját, amit átkeresztelt Zend Engine-re. Az ezredforduló után a cég tovább korszerűsítette a Zend Engine-t, és 2004-ben kiadásra került a PHP 4. Viszont ez a verzió se volt tökéletes, hiszen még nem volt objektumorientált. Ezt a hiányosságot a fejlesztők is észlelték, ezért pár évre rá kiadták a PHP 5-t, ami már támogatja az objektumorientált programozást. Továbbá a verzió már tartalmazza a PDO (PHP Data Objects) objektumot, ami nagyban megkönnyíti az adatbázis-kapcsolat kiépítését, illetve a PHP magja már Zend Engine II-t használ. Jelenleg is a PHP 5 a legfrissebb kiadás, azon pedig az 5.6.3-as verzió. [11,12]

Ezt a nyelvet a szerver oldalon futó Apache szervernél használtam fel, a beérkező kérések feldolgozásához.

### **3.4 MySQL**

A MySQL egy SQL-alapú relációs adatbázis-kezelő rendszer. Az egyik leghíresebb, és legelterjedtebb adatbázis-kezelő rendszerek egyike. Felkapottságát legfőképp annak köszönheti, hogy ingyenes licenccel rendelkező LAMP (Linux, Apache HTTP szerver, MySQL, PHP) szoftvercsomag része, ami költséghatékony, gyors fejlesztést nyújt dinamikus web-szolgáltatások kialakításához, működtetéséhez. Menedzselése vagy parancssoron keresztül, vagy nyílt forráskódú phpMyAdmin segítségével történik. [13]

A MySQL-re azért volt szükségem, hogy a próbaszerveren tárolt adatokat egyszerűen, és hatékonyan tudjam kezelni.

## 4. Specifikáció

A szakdolgozatom fő célja egy olyan Androidos alkalmazás tervezése és implementálása volt, amely képes a Workflow rendszer felhasználóihoz rendelt űrlapokat telefonos környezetben is menedzselni. A dolgozatom írása közben még a Workflow rendszer teljes egészében nem állt még rendelkezésemre, ezért annak imitálására, létre kellett hoznom egy teszt szerveret, amivel az alkalmazásomat teszteltem.

### 4.1 Funkcionális követelmények

Az Android applikáció segítségével a Workflow felhasználónak be és ki kell tudnia jelentkezni az alkalmazásból, az interfész szerver segítségével. Az Android alkalmazás ügyel arra, hogy úgy ne tudjon az alkalmazásba bejelentkezni a felhasználó, hogy nincs élő internetkapcsolata.

Az alkalmazás, felhasználónként le tudja kérni és meg tudja jeleníteni a felhasználókhöz tartozó űrlapok statisztikáit (mennyi űrlap tartozik a felhasználóhoz, és abból mennyi új, és mennyi leadott), a könnyed áttekinthetőség érdekében. Az applikáció listázni tudja kitöltött, és kitöltésre váró űrlapokat, amik informatív formázással (fontos űrlapokat kiemelve) jelennek meg a készülék képernyőjén. Ehhez a listafelület úgy kell implementálni, hogy a felhasználó kényelmesen át tudja az űrlapokat látni, illetve könnyedén tudja kezelni azokat.

Az űrlapokat az applikáció a szervertől érkező JSON leírónyelv által definiált strukturált adatokból generálja. Az űrlapegységek a hozzájuk tartozó tulajdonságaik szerint kelljenek, hogy megjelenjenek a felületen. Egy űrlap egység lehet általános űrlapegység típus, mint szöveg, jelölő négyzet, rádiógomb, de lehet speciális is, mint például egy kép, vagy számoló mező. Az utóbbi a hozzá rendelt képlet alapján számolja ki a képletben meghatározott beviteli mezők összegét, különbségét, szorzatát, hányadát.

Kitöltés közben ezeket az űrlapokat el kell, hogy tudja menteni az alkalmazás, és lehetőséget kell biztosítani a felhasználóknak a félbehagyott űrlapok kitöltésének folytatására. Az űrlap befejezésekor az űrlapot le lehessen adni,

viszont leadás után már nem lehessen azt módosítani (csak megtekinteni). A szerver naplózza azt, hogy egy adott űrlap volt e már megnyitva, és ha igen mikor nyitotta meg utoljára az adott felhasználó. Erre a funkcióra az alkalmazást kompatibilissé kell tenni.

Az alkalmazás felé támasztott elvárások között szerepel, hogy az applikációnak futnia kell a régebbi verziót használó készülékeken is. Ennek érdekében az alkalmazást úgy kell megtervezni és fejleszteni, hogy régebbi készülékkel is kompatibilis legyen.

Az applikációnak a szervertől érkező adatokat, a lehető legbiztonságosabb úton kell, hogy kezelje. Ennek érdekében olyan módszerekkel kell megvalósítani az adattárolást, hogy más, külső alkalmazás ne férhessen azokhoz hozzá.

Az alkalmazásnak támogatnia kell a többnyelvűséget, tehát úgy kell megtervezni, hogy későbbiekben egyszerűen lehessen a magyar és az angol nyelv mellé további nyelveket hozzárendelni.

## 5. Tervezés

Ez a fejezet áttekintést ad az alkalmazás tervezésének folyamatáról. Elsősorban bemutatásra kerülnek a felület tervezetek, illetve a szerverkapcsolat majd, kifejtésre kerül az űrlap és adatbázis adatmodell, végezetül bonyolultabb folyamatok.

### 5.1 Alkalmazás felületei

Az alkalmazás tervezését az alkalmazás lehetséges felületeinek (*activity*-knek) definiálásával kezdtem. Törekedtem arra, hogy ne legyen túl sok felület, viszont egy felület se legyen túlszűfolt. Ebből kifolyólag négy felületet terveztem az alkalmazáshoz: a bejelentkező felületet, a főoldalt, az űrlapok listázását megvalósító *activity*-t és magát az űrlap-megjelenítő, kezelő felületet.

A nyelvbeállítást, és a kijelentkezést az alkalmazás címrészében (*actionbar-activity*-ként) kerül implementálásra Ez a menü csak a főoldalon és az űrlapok listázásánál jelenik meg, a másik két felületnek nincs felső sávja.

#### 5.1.1 Nyitó felület

Az alkalmazás indítása után ezzel a kezdőfelülettel találkozik a felhasználó. Címként megjelenik az üdvözlő felirat, alatta két beviteli mező található ahova a felhasználónevet és a jelszót kell beírni. A beviteli mezőknek nincs címe, halványan bele van írva melyik a felhasználónév és melyik a jelszó beviteli mezeje. Helytelen bejelentkezés esetén figyelmeztetés jelenik meg a bejelentkezés sikertelenségéről, és a hiba okáról.

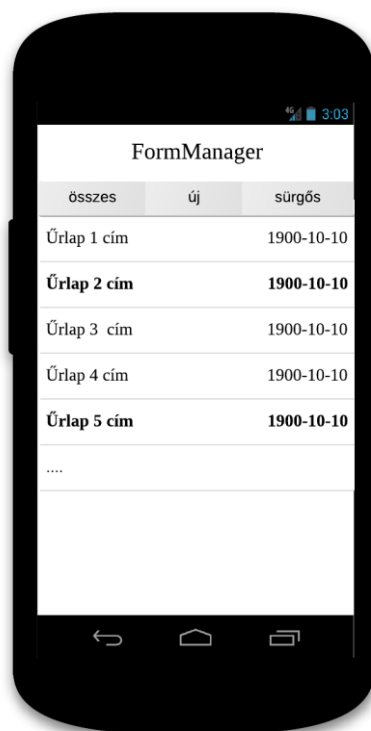
#### 5.1.2 Főoldal

A bejelentkezés után az alkalmazás a főoldalra kalauzolja a felhasználót, ahol az űrlapjait összefoglaló táblázattal találkozik. Ebben a táblázatban szerepel, hogy eddig mennyi űrlapot töltött ki, és mennyi vár kitöltésre az adott hónapban. Az összesítő táblázat alatt található egy gomb, mire kattintva megjelennek az űrlapok listanézetben is.



### 5.1.3 Űrlapok listázása

A felhasználó mikor ezt a felületet nézi akkor lát egy listát, amikbe fel vannak sorolva a hozzá rendelt űrlapok. A listanézetben négy jellemzője jelenik meg az űrlapnak. Első helyen a címe, majd a határideje szerepel az adott űrlapnak. A listanézetben szereplő űrlapok formázása is további információkat ad a felhasználónak. Az űrlap háttere jelzi majd az adatlap prioritását. A félkövér betűtípus azt mutatja, hogy az adatlapot még sose nyitotta meg az adott felhasználó tehát új, míg a sima betűtípus ennek ellenkezőjét jelzi. Az *activity* ezen felül rendelkezik szűrő gombokkal. A felület kinézetének tervezetét az 5-1.es ábra mutatja be.



5-1. ábra: Űrlapok listázása tervezet

### 5.1.4 Űrlap megjelenítése

A listanézetből jutunk el erre a felületre, ahol az adott űrlap rajzolódik ki. Ha az űrlapot csak olvasnia kell a felhasználónak (tehát csak egy dokumentációt kapott) akkor egy olvasva felirattal ellátott gombot talál az űrlap végén, mit megérintve

olvasottnak és leadottnak állítja az űrlapot. Ezzel ellentétben, ha az űrlapnak vannak olyan részei mit ki is kell tölteni, akkor az űrlap végén a mentés, és leadás gombok jelennek meg. Ebben az esetben a felhasználó el is tudja menteni az adott állást, (és később visszatérve folytatni tudja), vagy a kitöltött adatlapot le tudja adni. Egy űrlap leadása után se menteni se újra leadni, nem tudja. A leadás és mentés sikerességéről egy figyelmeztető üzenet jelenik meg

A kitöltendő űrlapokhoz előre vannak definiálva szekciók, amik az űrlap-részegységeket (beviteli mező, szövegrész, jelölő négyzet, rádió gomb stb.) tartalmazzák. Minden ilyen egységhez tartozik címke is, ami az egység előtt jelenik meg, az adott egység címeként, címkéjeként. Az űrlapok tartalmazhatnak egy speciális beviteli mezőt, minek az értékét módosítani nem lehet, mert dinamikusan változik. Ennek az űrlap részegységnek a neve számoló mező (*computable field*). Ennek a beviteli mezőnek a sajátossága, hogy az űrlapon előforduló két másik mezőbe beírt (szám) adatának összegét különbségét szorzatát vagy hányadát jeleníti meg. Ez a számoló mező azt is lekezezi, ha a beírt mező dátum, ilyenkor a másik értéket hozzáadja (vagy kivonja).

## 5.2 Szerverkapcsolat

A kommunikáció létrehozásához POST-GET kapcsolatot terveztem, ami azt jelenti, hogy szükség van egy szerverre, ami fogadja az adatokat és azoknak tükrében adatokat küld vissza. Ennek a szervernek egy Apache szervert terveztem használni, amin PHP nyelvben íródott weboldalak találhatók. Ezek a weboldalak a beérkezett kérések tükrében elvégzik a szükséges adatlekérést, adatmódosítást a szerveren, majd weblapként megjelenítik a visszaküldendő adatot.

### 5.2.1 Kapcsolatoptimalizálás

Az alkalmazás és a szerver között, előfordulhat (ha egy adott felhasználóhoz sok, nagy terjedelmű űrlapok tartoznak), hogy nagy mennyiségű adatcsere is végbemehet az alkalmazás futtatása közben. Ennek érdekében optimalizálni kellett a lekéréseket, és a feltöltendő adatokat. Először az volt a tervem hogy, egy felhasználóhoz tartozó összes űrlapot, űrlap adataival és vázával együtt fogja letölteni az alkalmazás (majd módosítás alkalmával, az összes adatot visszaküldi,

a szervernek ki eldönti mi módosult). Ezt a tervet elvetettem, mivel a kapcsolatnak nem megterhelő, viszont az applikáció lassabban dolgozna fel az így megkapott adatokat. Ennek orvoslására vezettem be azt, hogy mikor a felhasználóhoz tartozó űrlapokat listázza az alkalmazás, akkor a szervertől csak az űrlap tulajdonságokat kéri le (űrlap címe, azonosítója, határideje stb.), magát az űrlap vázát nem. Az űrlap vázát akkor kéri le az alkalmazás, mikor azt meg akarja nyitni, és kitöltés után csak az adott űrlap adatokat küldi el a szervernek. Ez által felgyorsul az üzenet-feldolgozás, és nem kell több másodpercet várni, míg feldolgozza az adatokat az applikáció.

### 5.2.2 Üzenet architektúrája

A szerver és az alkalmazás között lévő kommunikáció üzenetek váltásaként történik. Ezek az üzenetek JSON leíró nyelv segítségével került definiálásra. Az üzenet két fő részből áll össze, a cím (*title*) szekció, és az üzenet (*message*) rész. A címben található a küldő és fogadó ID azonosítója, amivel pontosan definiálni lehet a küldőt és a fogadót. Ennek megléte azért fontos, hogyha a későbbiekben tovább akarják bővíteni az alkalmazást a felé, hogy két applikáció is tudjon kommunikálni egymással (szerver aktív jelenléte nélkül) akkor is használható legyen ez az üzenet struktúra.

Az üzenet (*message*) részben, megtalálható az üzenet célja (ami lehet; adatlekérés, bejelentkezés stb.) illetve maga az adat (például: űrlap adatok, bejelentkezési adatok stb.). Az üzenet célját az adatfeldolgozás megkönnyítése céljából érdemes definiálni. Az üzenetre egy példa (ahol a felhasználó űrlapjainak statisztikáit kéri le a szerverről):

```

{ "title":{
  "sender":0,
  "to":1},
  "message":{
    "formaction":"getssummarydata",
    "data":{
      "allformsCount":4,
      "filledformsCount":1,
      "unseenformsCount":0
    }
  }
}

```

### 5.3 Védelem, adatok kezelése

Az alkalmazás tervezése során fontosnak tartottam az űrlap adatok kezelését, védelmét. Ennek következtében az alkalmazás úgy lett kialakítva, hogy lokálisan ne tároljon adatot se a felhasználóról, se a felhasználó űrlapjairól. Az alkalmazás futása közben minden adatot a felhőből (a szervertől) kér le, és oda menti azokat. Az applikáció a RAM-ban (hosszútávon), csak a felhasználó azonosítóját (ID-át) tárolja. Ennek következtében külső applikáció nem férhet hozzá az alkalmazás adataihoz.

A kliens-szerver kapcsolatot úgy alakítottam ki, hogy a szerverre küldendő, és a szervertől kapott adatok könnyen titkosíthatóak legyenek, tehát ha a későbbiekben a szerver kódolva szeretné küldeni az adatokat, az alkalmazás forrását gyorsan át lehet alakítani.

### 5.4 Folyamatok

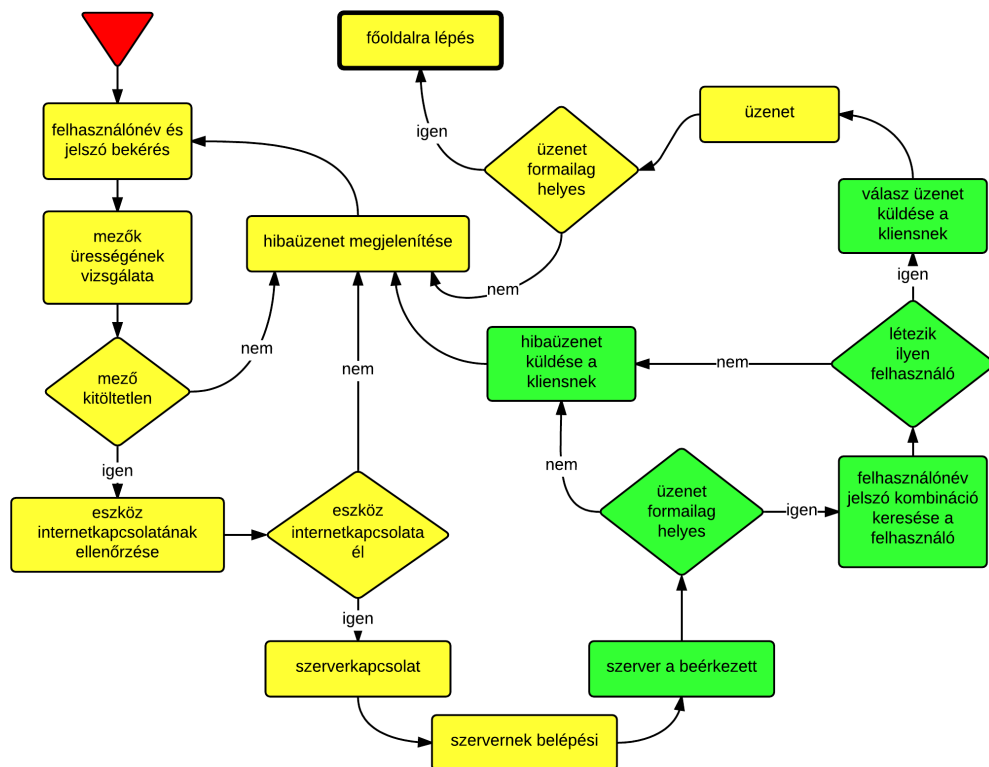
A következő alfejezetekben két folyamatot mutatok be, a bejelentkezést, illetve az űrlap kitöltését. A folyamatokhoz tartozó ábrák úgy lettek megszerkesztve, hogy egyértelműen elkülöníthető legyen, mely folyamategységek zajlanak az applikáción, és melyek a szerveren.

#### 5.4.1 Bejelentkezés

A felhasználó bejelentkezés alkalmával egy bonyolult folyamat zajlik le, amit a következő ábra (5-2. ábra) szemléltet. A készüléken történő folyamatokat

sárgával, míg a szerveren folyókat zölddel jelöli az ábra. A folyamat kezdetét egy piros háromszöggel szemléltetem.

A bejelentkezés folyamata azzal kezdődik, hogy az alkalmazás bekéri az adott felhasználó nevét és jelszavát. Ha valamelyik beviteli mező üres, hibaüzenetként jelzi azt.



5-2. ábra: Bejelentkezés folyamata

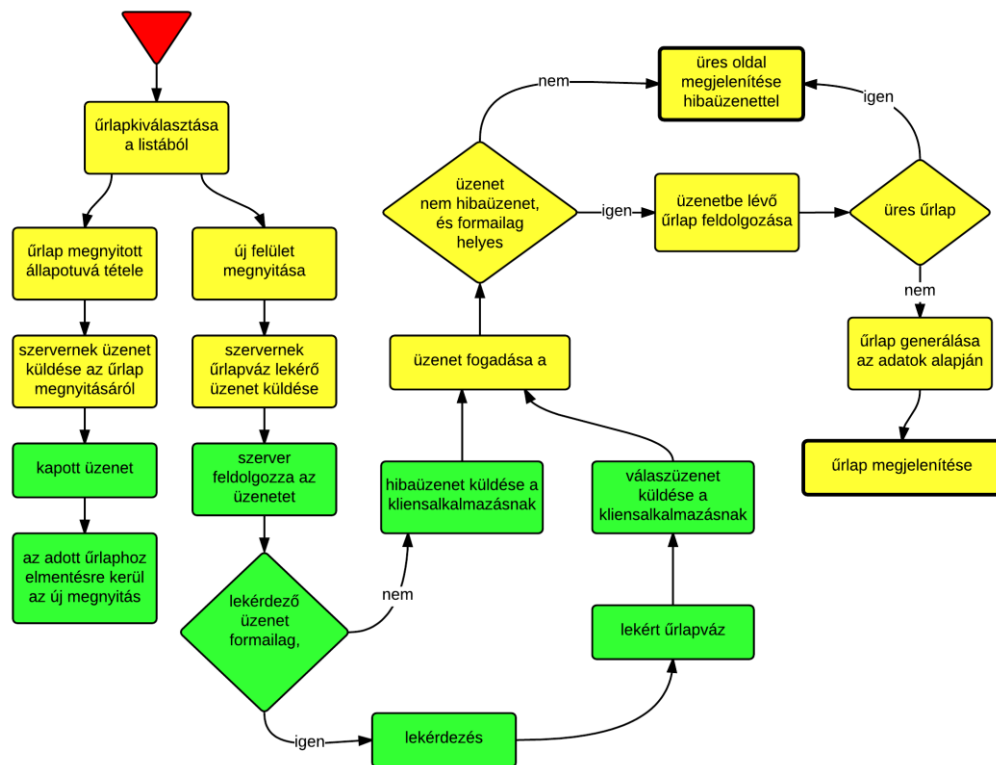
Viszont ha mindkét mezőben szerepel érték, akkor az alkalmazás ellenőrzi a készülék internet kapcsolatát, ha az jól működik, akkor a szervernek (előre definiált) üzenet formájában elküldi a bejelentkezési adatokat, ellenkező esetben (ha nincs internet kapcsolat) az alkalmazás jelez a felhasználó felé a problémáról. Ezt követően az alkalmazás vár a szerver válaszára. A szerver mikor megkapta az adatokat ellenőrzi annak formai, és tartalmi helyességét. Megnézi, hogy feltudja-e dolgozni az üzentet (nem érkezett-e meg hibásan) illetve, hogy a szerveren van-e olyan felhasználó, aminek ugyan az a jelszava és a felhasználó neve, mint amit az üzenetbe kapott. Ha van olyan felhasználó, akkor üzenetként visszaküldi a felhasználó nevét és azonosítóját, viszont ha nem található ilyen felhasználó az

adatbázisban hibaüzenetet ad vissza a szerver. Végezetül az alkalmazáshoz érkezett üzenetből eldönti, hogy a bejelentkezés sikeres volt e vagy sem, és ha sikeres, akkor belépteti a felhasználót a főoldalra, ha nem, akkor a folyamatot újrakezdeti vele, és újból kéri a felhasználónevet és a jelszót.

#### **5.4.2 Űrlap megjelenítés**

Az űrlap megjelenítésének folyamatát az 5-3. ábra mutatja be. Ahogy az előző ábránál, az alkalmazás futása közbe végbemenő folyamatokat sárgával, a szerver oldali folyamatokat pedig zölddel van szemléltetve. A kezdőpontja a folyamatnak egy piros háromszög.

Az űrlap készítése azzal kezdődik, hogy az űrlapok listájából kiválasztjuk a megjelenítendő űrlapot. Ekkor először az alkalmazás egy üzenetet küld a szervernek arról, hogy a felhasználó melyik űrlapot nyitotta meg. Erre azért van szükség, hogy naplózni lehessen azt, hogy egy felhasználó hányszor nyitotta meg az adott űrlapot, mielőtt leadta. A szerver mikor megkapta a készüléktől jövő üzenetet azt feldolgozza, és ha az üzenet tartalmilag és formailag helyes, akkor bekerül az új megnyitási dátum a naplóba. Ezt követően az activity elindul, amibe az űrlap kerül megjelenítésre. Mivel az űrlapok listanézetben nem tartalmazzák magát az űrlapok vázát, csak az űrlapok adatait, ezért az űrlap kitöltő felület indítása után a szervertől lekéri az alkalmazás az adott űrlap tartalmát, vázát. Ezt a kérést a szerver feldolgozza, és válaszüzenetbe visszaküldi az adott űrlap adatait. (Ha nem találja a lekérendő űrlapot hibaüzenettel válaszol a kérésre.) Ha az adatok rendben megérkeznek, akkor az üzenet feldolgozásra kerül, ellenkező esetben egy üres űrlap kerül megjelenítésre, egy hibaüzenettel. Abban az esetben is ez a hibaüzenettel ellátott oldal jelenik meg, ha maga az üzenet nem tartalmaz űrlapot. Ha az üzenetben lévő űrlapot az applikáció fel tudja dolgozni, akkor megjeleníti azt.



5-3. ábra: Űrlap megjelenítés folyamata

## 5.5 Űrlap felépítése, megjelenítése

Mindegyik űrlap azonos szerkezettel rendelkezik. Erre azért van szükség, hogy az alkalmazás fel tudja dolgozni és meg tudja jeleníteni. Minden űrlap rendelkezik egy címmel és egy űrlap vázzal, ami szekciókat tartalmaz. A cím minden esetben az űrlap tetején helyezkedik el, alatta pedig sorban a szekciók. Minden szekciónak van címe (ami címkeként van tárolva) és vannak űrlapegységei, amik lehetnek jelölőnégyzet (*checkbox*) rádiógomb-csoport (*radiobuttongroup*), szöveg (*textfield*), kép (*picture*) és két fajta beviteli mező, a sima beviteli mező (*edittext*), és a számoló mező (*computable*). Az olvasásra szánt űrlapok végén (ha az adott űrlap nincs leadva) olvasva gomb helyezkedik el. Azoknak az űrlapoknak a végén, amik kitöltésre lettek létrehozva két gomb jelenik meg, egy mentés gomb, és egy leadás gomb, amikkel menteni és leadni lehet az űrlapot.

Az űrlap minden egyes részegysége (a címkéket leszámítva) egyedi azonosítóval rendelkezik. Az egyéni, speciális azonosítóra azért van szükség, mert ezek az azonosítók adódnak át a felületre felkerülő részegységeknek (*view*-knak)

és ezekkel az azonosítókkal lehet egyesével lekérni az adatlap mentésekor az űrlapegységeket, és ez által a tartalmukat. A másik oka az egyedi azonosítóknak, hogy ezek az azonosítók növekvő sorrendben kerülnek meghatározásra, ami azért fontos, mert a JSON parzolása (átalakítása) nem sorrendtartó, tehát előfordulhat az (ha nem rendezzük sorrendbe a generálás során), hogy a szekciók, és a szekciókon belüli egységek sorrendje felcserélődik, és az űrlap nem az eredeti formájában fog megjelenni. Az azonosítók generálásának és hozzárendelésnek a logikája a következő:

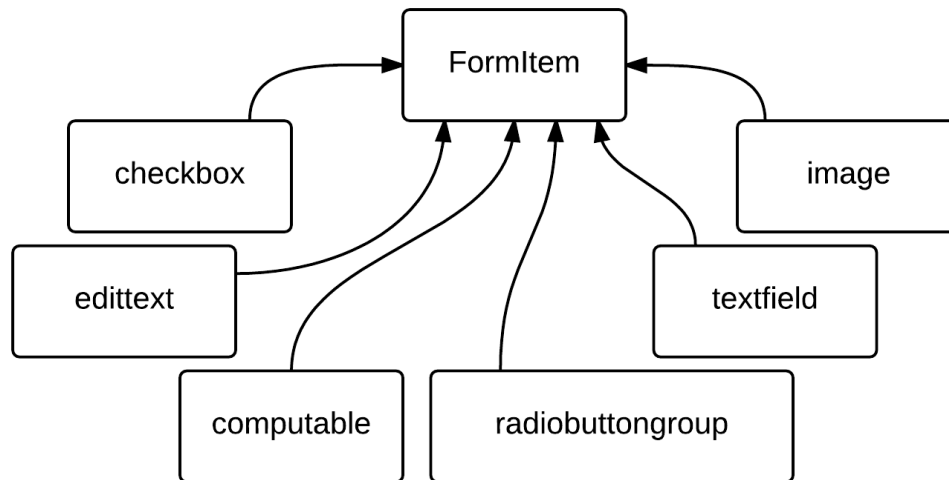
- A szekciók azonosítója maradék nélkül osztható 10000-rel és ugyan ennyivel növekednek.
- Az adott szekcióhoz tartozó űrlapegységek, azonosítója egyesével növekszik, és mindegyikhez hozzáadódik, a szekciója azonosítója (például: szekció-azonosító: 120000, a benne található egységek azonosítója rendre; 120001, 120002, 120003.. stb.)

## 5.6 Űrlap adatmodell

A tervezés során nagy prioritást élvezett a precíz adatmodell kialakítása. Elsődleges szempontot játszott az, hogy úgy alakítsam ki az objektumokat és a modellt, hogy egyszerűen kezelhető, átlátható legyen, de egyben könnyen lehessen módosítani, bővíteni.

Egy űrlap egy olyan tároló, ami szekciókat tartalmaz. Az űrlap szekciói tárolják a hozzájuk tartozó űrlapegységeket. Ezeknek az űrlapegységeknek van egy ős osztályuk, minek olyan attribútumai vannak, amik minden részegységben megtalálhatóak (azonosító, címke, szerkeszthetőség, láthatóság, típus). Ebből az ősosztályból öröklődik: a rádiógomb-csoport (radiobuttongroup), a kép (image) a jelölőnégyzet (checkbox), a szöveg (textfield), és két fajta beviteli mező, a sima beviteli mező (edittext), és a számoló mező (computable). Ős osztály létrehozására azért volt szükség, mert így könnyebben kezelhetőek az űrlap-részegységek.





5-4. ábra: Űrlapegységek adatmodellje

## 5.7 Adatbázismodell

Maga a (szerver oldali) adatbázis 4 táblából áll; egy űrlapokat tároló, egy felhasználókat tároló, egy űrlapok log adatait tartalmazó tároló és egy felhasználót űrlapokkal összekapcsoló tábla.

Űrlapokat tároló tábla attribútumai tartalmazzák az összes adatot, amivel egy űrlap rendelkezik; határidővel, legutóbbi megnyitás dátumával, leadás állapotával (le lett e adva vagy sem), címmel, azonosítóval, magával az űrlappal (JSON formátumban eltárolva) és az űrlapművelettel (olvasni kell vagy kitölteni). A tábla létrehozása a következő SQL kóddal lehetséges:

```

CREATE TABLE IF NOT EXISTS `forms` (
  `FORMS_ID` int(11) NOT NULL AUTO_INCREMENT,
  `FORMS_DEADLINE` varchar(40) NOT NULL,
  `FORMS_SEEN` date DEFAULT NULL,
  `FORMS_FILLED` int(11) NOT NULL,
  `FORMS_FORMBODY` text NOT NULL,
  `FORMS_TITLE` varchar(200) NOT NULL,
  `FORMS_FORMACTION` varchar(40) NOT NULL,
  PRIMARY KEY (`FORMS_ID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 AUTO_INCREMENT=5;
  
```

Felhasználó tábla a felhasználók adatait tartalmazza; a felhasználónevet, a jelszót, a felhasználó állapotát (be van e jelentkezve vagy sem) illetve az azonosító kódját. Maga a táblát ezzel a kóddal lehet létrehozni:

```
CREATE TABLE IF NOT EXISTS `users` (
  `USERS_ID` int(11) NOT NULL AUTO_INCREMENT,
  `USERS_USERNAME` varchar(150)
    COLLATE utf8_hungarian_ci NOT NULL,
  `USERS_PASSWORD` varchar(150)
    COLLATE utf8_hungarian_ci NOT NULL,
  `USERS_LOGIN` int(11) NOT NULL,
  PRIMARY KEY (`USERS_ID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_hungarian_ci
AUTO_INCREMENT=3;
```

Az űrlapok log adatait tartalmazó tábla azért lett készítve, hogy számon legyen tartva az, hogy egy adott felhasználó hány alkalommal nézte meg az űrlapot mielőtt kitöltötte. Ez a tábla három attribútummal rendelkezik, egy azonosítóval, egy űrlap-azonosítóval, és egy dátummal. A tábla létrehozásához szükséges SQL kód:

```
CREATE TABLE IF NOT EXISTS `formslog` (
  `FORMSLOG_ID` int(11) NOT NULL AUTO_INCREMENT,
  `FORMSLOG_FORMID` int(11) NOT NULL,
  `FORMSLOG_DATE` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (`FORMSLOG_ID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_hungarian_ci
AUTO_INCREMENT=1;
```

A kapcsolótábla csak egy azonosítót, és a felhasználó és az űrlap azonosítóját tartalmazza, ami egyértelmű relációt eredményez, így egyszerűen le lehet kérni a felhasználókhoz tartozó űrlapokat.

```
CREATE TABLE IF NOT EXISTS `usersforms` (
  `USERSFORMS_ID` int(11) NOT NULL AUTO_INCREMENT,
  `USERSFORMS_USERID` int(11) NOT NULL,
  `USERSFORMS_FORMID` int(11) NOT NULL,
  PRIMARY KEY (`USERSFORMS_ID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_hungarian_ci
AUTO_INCREMENT=6;
```

## 6. Fejlesztés

Ebben a fejezetben kifejtésre kerül az alkalmazás és a teszt szerver fejlesztése, és a kritikusabb programkód-részletek bemutatása.

### 6.1 Alkalmazás felületei

A felületek úgy lettek kialakítva (ahogy a tervezés fejezetben már említettem), hogy egyik felület se legyen túlzsúfolt, és ne végezzen túl sok feladatot.

Az *activity*-k (felületek) kinézetét két módon lehet meghatározni, xml fájlban, vagy a felülethez tartozó java osztályban. A statikus elemeket az xml fájlban (mint például a bejelentkező felület beviteli mezőit), míg a dinamikus elemeket (például: űrlapok elemeit) a java osztályban definiáltam. A felületek osztályaiban csak olyan feladatokat ellátó függvények szerepelnek amik, specifikusak, tehát csak az adott activity-hez tartozik (például: bejelentkezés). Azok a feladatok, amik több helyen is szükségesek (például szerverkapcsolat), külön osztályokba kerültek megvalósításra, amik példányosítva vannak az adott felülethez tartozó osztályban. Ezeket az osztályokat külön alfejezetben mutatom be.

A mivel projekt alapkövetelményei között szerepel, hogy az applikációnak futnia kell a legalább Android 2.3.3 operációs rendszerrel rendelkező készülékeken, ezért a felületek készítésekor *fragment*-ek helyett is *activity*-ket használtam.

Az alkalmazásban implementált összes felületekhez tartozik egy Java osztály. Ebben a Java osztályban lehet megírni a dinamikus működést biztosító függvényeket. Magát az osztályt vagy az *ActionBarActivity*-ből vagy a sima *Activity*-ből kell örököltetni ahhoz, hogy az előre definiált, alapértelmezett *Activity* osztályokat tudjuk használni. A két osztály közötti különbség az, hogyha az előbbit örököltetjük, akkor megjelenik egy cím-sáv a felületen ahol az alkalmazás neve és logója, valamint (ha definiálva van hozzá) menü helyezkedik el rajta. Ha viszont az utóbbiból öröklődik a felülethez tartozó osztályunk, akkor nem jelenik meg ilyen címsáv. Ahhoz hogy a java osztály tudja, melyik felülethez tartozik, definiálni kell a *setContentview* függvény segítségével.

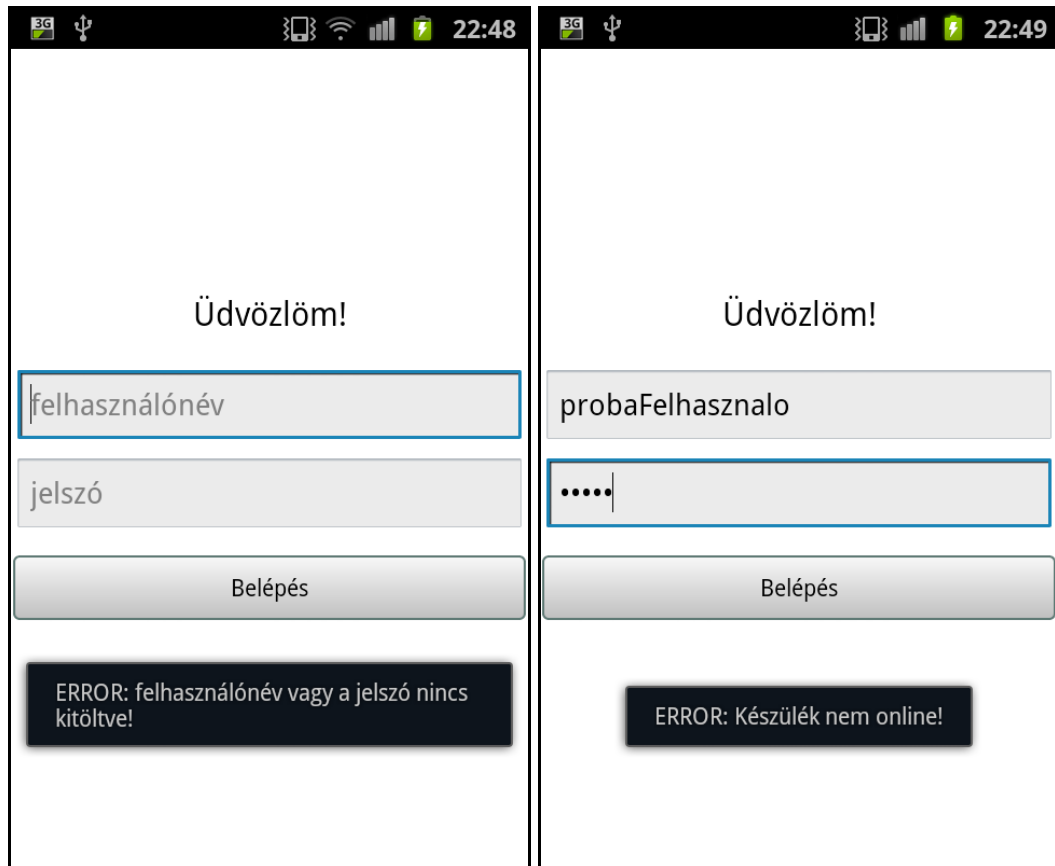
Az alkalmazás helyes működése érdekében még az új *activity*-t az *AndroidManifest.xml* fájlban is definiálni kell a következő módon:

```
<activity android:name=".activities.FormActivity"></activity>
```

### 6.1.1 Nyitó felület

A nyitó felület (*activity\_login.xml*) fogad minket mikor elindítjuk az alkalmazást. Itt látható a bejelentkezés rész, a bejelentkezés cím, illetve két beviteli mező, mi a belépési adatokat várja. A felületet kinézetét teljes mértékben az xml fájl határozza meg, mert nincs benne semmilyen dinamikus rész.

A bejelentkezéshez, és a továbblépéshez ki kell tölteni a felhasználónév és jelszó beviteli mezőket, majd meg kell érinteni a belépés gombot. Ezt követően az alkalmazás leellenőrzi, hogy a beviteli mezők ki lettek-e töltve, ha nem akkor erről értesíti a felhasználót (6-1. ábra bal oldali képe) *Toast* segítségével. Ha az alkalmazás meggyőződött arról, hogy helyesen lett kitöltve a két beviteli mező, ellenőrzi, hogy a készüléken a wifi be van a kapcsolva. (Az internetkapcsolat meglétét a *ConnectivityManager*-ből lekérdezhető *NetworkInfo isConnected* metódusából kérdezi le az alkalmazás.) Ha a készülék nem online, akkor a szervernek meg se próbál adatot küldeni, hanem értesíti a felhasználót a problémáról (6-2. ábra jobb oldali képe), viszont ha a készülék internetre van csatlakozva, akkor megpróbálja elérni a szervert. Ha sikeresen eléri, és az adatok rendben voltak akkor a szerver visszaküldi az alkalmazásnak a felhasználó azonosító kódját az ID- kódját.



6-1. ábra: Bejelentkezés

Ezt az ID kódot megkapván az alkalmazás elindítja a főoldal *activity*-jét, minek átadja *intent*-ként a felhasználói azonosítót, majd saját magát bezárja (meghívja a *finish* függvényt). Saját magát azért kell bezárnia a kezdő oldalnak, mivel ha az a felület nem szünteti meg önmagát, akkor a főoldalról bármikor vissza lehetne lépni a back gomb érintésével, ami logikai hibát eredményezne az alkalmazás működésében. Abban az esetben, ha rossz adatokat, továbbit a felület a szervernek akkor a szerver hibaüzenettel válaszol, amiről értesíti a felhasználót, és újra bekéri a jelszavát és a felhasználó nevét.

A bejelentkezés folyamata (*LoginActivity.java*-ban) úgy lett megírva, hogy csak az *onCreate activity* függvény lett felül írva. Azért nem volt szükség a többi *callback* eseménykezelő metódust is felülírni, mert ez a felület megszűnik miután a felhasználó bejelentkezett, ezért nem kell lekezelni azt az eshetőséget, ha a felhasználó vissza akarna lépni erre az oldalra. Ebben a metódusban lett példányosítva a (kapcsolatok menedzselésére létrehozott) szerverkapcsolat

objektum. Ennek az osztálynak (*ServerConnection*) a *login* metódusa van meghívva, ami (ha a bejelentkezés sikeres) a felhasználó azonosítójával tér vissza, ha viszont nem sikeres, akkor egy negatív számmal. A negatív szám lehet -1 abban az esetben szerver által leellenőrzött adat hibás, és lehet -2, amit akkor kap a bejelentkező osztály, ha a készülék nem online. Azért van szükség arra, hogy két negatív értéket is visszaküldjön az alkalmazás, mert ez alapján értesíti az alkalmazás a felhasználót a felmerülő problémáról.

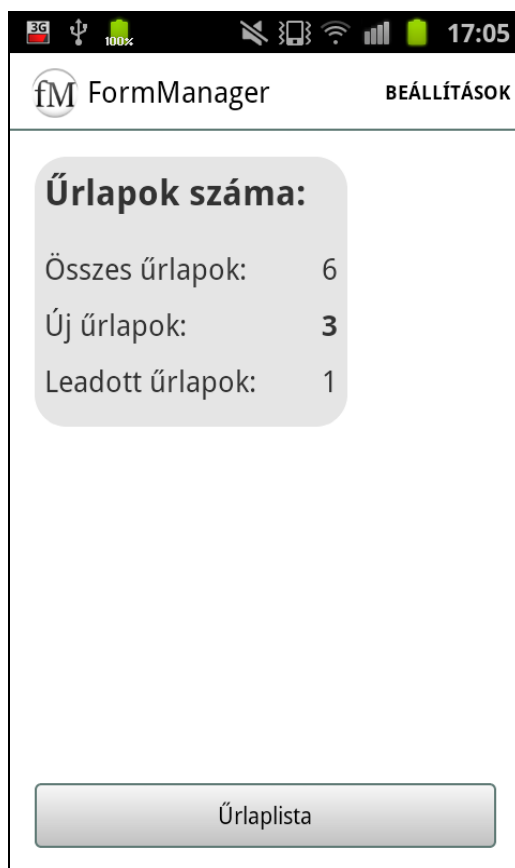
A Bejelentkezés oldal stílusa az alapértelmezett beépített *HoloLight* stílus. Csak a gomb stílusa került felülírásra a *drawable/rounded\_button.xml* fájlban definiáltak alapján.

### 6.1.2 Fő oldal

A sikeres bejelentkezés után a főoldal jelenik meg (*activity\_main.xml*). Ez a felület a (bejelentkező felülettel ellentétben) rendelkezik egy cím-sávval, ahol az alkalmazás logója, az alkalmazás neve, és a beállítások menü látható. Ez alatt a *LinearLayout* felület tartalmaz egy összefoglaló táblázat, mi kijelzi a bejelentkezett felhasználónak, hogy mennyi űrlap van hozzárendelve, és ezek közül mennyi az új, és mennyi a leadott státuszú. Az új űrlapok darabszámát (hangsúlyozás végett) félkövérrel kiemeli az alkalmazás. A felület alján pedig egy gomb található, amit megérintve az űrlapok listázásáért felelős *activity*-t indítja az alkalmazás. A felület statikus részeit az xml fájlban definiáltam.

A főoldali felülethez a *LoginActivity.java* tartozik. Ebben a Java osztályban felülírtam az *onCreate* metódust, hogy ott kinyerjem a felületnek *intent*-ként átadott változókat. ebben a metódusban kerül példányosításra a szerverkapcsolatért felelős osztály is. A felületen elhelyezkedő összesítő tábla feltöltését, az *onResume* felülírt metódus végzi, mivel előfordulhat az, hogy a felhasználó visszatér erre a felületre, és a legfrissebb adatokat tartalmazó összesített táblázatot szeretné látni. Ebben a függvényben kéri le az adatokat az Android alkalmazás a workflow szerverről a *ServerConnection* osztály *updateFormsSummary* objektum segítségével. Maga a függvény egy *map* tárolóval tér vissza, minek a kulcsai a lekérdező adatok nevei az adat tagjai pedig

maguk az értékek. A függvény *null* értéket ad vissza abban az esetben, ha a kapcsolatban hiba történt. Ezt vizsgálva tudja eldönteni az applikáció felülete, hogy az adatok megérkeztek-e. Ha az adatokat rendben találta, akkor az alkalmazás beírja azokat a táblázatba, és megjeleníti az űrlapok listájára kalauzoló gombot.



6-2. ábra: Főoldal

A főoldal formázását a felülethez tartozó xml fájlban valósítottam meg. Az összesítő táblázat háttérét és a sarkainak lekerekítésének mértékét egy külön fájl írja le (*drawable/shape\_summarytable.xml*). Az felületen található gomb háttere felül lett írva a *rounded\_button*-nal. Az *ActionBar* felső sáv már az oldalhoz tartozó java kódban került meghatározásra, a *header\_stlye* segítségével.

### 6.1.3 Űrlapok listázása

Az főoldalról elérhető egy olyan felület, ahol a felhasználó az összes űrlapját listázni tudja. A felületen található egy címsáv (*ActionBar* sáv) ahol az applikáció

logója, az alkalmazásneve található, és a beállítások menü érhető el. Ez alatt található három gomb (összes, új, sürgős), amikkel a listaelemeket lehet szűrni. Ez alatt található egy olyan tábla (*TableLayout*) aminek a sorai maguk az űrlapok. Egy űrlapnak megjelenik az címe, és a kitöltési határideje. A tábla sorait (űrlapjait) érintve lehet megnyitni az adott űrlapot egy új felületen.

Az oldal java fájljában (*FormListActivity.java*) felülírássra került az *onCreate* metódus, ahol a szerverkapcsolatért felelős osztály lett példányosítva illetve az *intent*-ek kezelve. A tábla feltöltését, és a szűrőgombok implementálását az *onResume* metódus hajtja végre. A feltöltés metódus a szerverkapcsolatért felelős osztálytól megkapja az összes űrlap adatát (kivéve az űrlap vázát), és azt egyesével megjeleníti, a táblában.

Az oldal formázása xml fájlokban és a felülethez tartozó Java fájlban kerül megvalósításra. Az xml fájlban (*activity\_formlist.xml*) vannak az *activity* elrendezése és statikus elemeinek stílusai; a gombok és a címsáv. Ezzel ellentétben a dinamikus elemek (az űrlapsorok) a java osztályban kerültek megvalósításra. Erre azért volt szükség, mert java osztályon belüli metódusok döntenek el, az űrlap adatai alapján hogy milyen formázást kapjon az adott űrlap-táblasor. Tehát egyesével kerül beállításra a háttérszín (mi azt jelzi, hogy az űrlap leadási határideje milyen közel van, illetve hogy le lett e adva), illetve a betűtípus (félkövér az űrlap címe és a leadási határideje, ha olvasatlan az űrlap). Az tábla sorainak háttér és keretszíne, illetve a keret vastagsága és lekerekítettsége külön xml fájlokban vannak definiálva, míg a betűtípus nem.

Az 6-3-as ábrán található a listázó felület. Bal oldalt látható egy kép arról, mikor a felhasználónak az összes adatlapja megjelenik, jobb oldalt pedig egy olyan, amikor az új, még nem olvasott űrlapokra lett szűrve.



fm FormManager BEÁLLÍTÁSOK	
összes	új
Android alkalmazásfejlesztés II.	2014-12-30
Android alkalmazásfejlesztés III.	2015-01-10
GE jogi szerződés	2014-11-22
OMV szerződés	2014-11-20
GE Android fejlesztés	2015-01-20
MIK weboldalfejlesztés, háttérkép	2014-11-30
KickOff Meeting	2014-12-10

fm FormManager BEÁLLÍTÁSOK	
összes	új
Android alkalmazásfejlesztés II.	2014-12-30
GE jogi szerződés	2014-11-22
KickOff Meeting	2014-12-10

6-3. ábra: Űrlapok listázása


#### 6.1.4 Űrlapok megjelenítése

A felhasználó egy űrlapot, az űrlaplistából kiválasztva (azt megérintve) tudja megnyitni. Ekkor elindul az űrlapok megjelenítésért felelős oldal. Az oldal a megkapott felhasználói azonosító és űrlap azonosító segítségével lekéri az űrlapot a szervertől, amit feldolgoz és megjelenít. Ha az űrlap még nem volt leadva az űrlapot a felhasználó menteni, és leadni tudja az űrlapok végén elhelyezkedő gombok segítségével.

Az activity-hez tartozó java objektum kezel, és végez el mindent. Maga az oldal xml fájlban csak az űrlap címét megjelenítő *TextView* és egy az űrlap vázának megjelenítésre szolgáló *LinearLayout* van definiálva. A felület objektuma az *onCreate* metódusban először példányosítja a *ServerConnection* osztályt, majd ezt követően segítségével lekéri az űrlapokat. Ezt követően a *FormGenerator* objektum segítségével készíti el az űrlap vázát. (Az 5.4 és az 5.5 fejezet részletesen kifejtésre kerül, hogy hogyan generálja az alkalmazás az

űrlapegységeket.) Ha a felhasználó leadja vagy elmenti az űrlapot, akkor elsősorban felülírja a memóriában tárolt űrlapot a legújabb verzióval majd a már példányosított *ServerConnection* osztálynak átadja, ami elküldi a szervernek. Ha minden rendben történt, akkor azt egy *Toast* üzenet segítségével az *activity* a felhasználó tudtára adja. A következő pár képen pár generált űrlap látható.

A következő két ábrán láthatóak a megjelenített, űrlapok. Az 6-4.-es ábra bal oldala szemlélteti azt, hogy a leadás előtt a kitöltés végén, milyen gombok fogadják a felhasználót, az ábra jobb oldalán pedig a méretre igazított képmegjelenítés látható. Az 6-5.-ös ábra bal oldal megmutatja hogyan vált a billentyűzet e-mail cím bevitelét elősegítő módba, illetve a képen látható hogy jelenik meg a kitöltési minta a beviteli mezőben. Ezen ábra jobb oldalán pedig a számoló mező működése van illusztrálva egy speciális űrlap által.

<div> <div> <div>3G</div> <div>100%</div> <div>18:35</div> </div> <div> <div>Kutató munka</div> <div> <i>Suspendisse hendrerit scelerisque lorem ut viverra. Donec sit amet lectus lacus. Nulla ornare aliquam consectetur. Phasellus viverra, lorem sit amet interdum mattis, arcu lectus lacinia orci, eu lobortis metus augue iaculis orci. Praesent augue dolor, fermentum in lacinia nec, rhoncus non purus. Pellentesque sollicitudin iaculis nisi, a mattis neque tincidunt ut.</i> </div> <div> <div>Felelős Neve:</div> <div>Kiss János</div> </div> <div> <div>alporjektek száma:</div> <div>11</div> </div> <div> <div>Részt kíván venni benne:</div> <div> <input checked="" type="radio"/> igen                     <input type="radio"/> nem                 </div> </div> <div> <div>leadás</div> </div> <div> <div>mentés</div> </div> </div> </div>	<div> <div> <div>3G</div> <div>100%</div> <div>17:06</div> </div> <div> <div>MIK weboldalfejlesztés, háttérkép</div> <div>Fotós adatai:</div> <div> <div>Neve:</div> <div>Kiss Janos</div> </div> <div> <div>Évfolyam:</div> <div>2</div> </div> <div> <div>Szak:</div> <div>gazdaságinformatikus BSc</div> </div> <div> <div>Kép</div> <div>Háttérkép:</div> <div>  </div> </div> </div> </div>
--	--

6-4. ábra: Űrlap megjelenítés 1.

The image displays two side-by-side screenshots of an Android application interface.

**Left Screenshot: Autópálya használat**

- Header: Autópálya használat
- Form fields:
  - Name: Kiss Janos
  - Életkora: 10
  - Autó rendszáma: ERF-111
  - email cím: example@domain.jp
- Keyboard: A standard QWERTY keyboard is visible at the bottom.
- Status bar: 3G, 100% battery, 18:03.

**Right Screenshot: Számológép**

- Header: Számológép
- Section: Első rész!
- Form fields:
  - A: 23
  - B: 15
  - A+B: 38
  - Dátum(C): 2014-11-12
  - Napszám(D): 56
  - C-D: 2014-09-17
- Button: leadás
- Status bar: 3G, battery, 20:19.

6-5. ábra: Űrlap megjelenítés 2.

### 6.1.5 Nyelvbeállítás, kijelentkezés

A kijelentkezés és a nyelvbeállítás a főoldalon, és az űrlapok listázása felületen érhető el a beállítások menüpontra belül. Mikor rákattintunk a beállításokra, akkor, ha 4.0 vagy újabb Android verziót használó készüléken fut akkor legördülő menü, ellenkező esetben középen megjelenő lista jelenik meg.

A nyelvbeállítás alapértelmezettként működik az alkalmazásnál, tehát átveszi a telefon által használt nyelvet, viszont ha magyarról angolra, vagy angolról magyarra akar váltani, azt a beállítások menüpontra éri el. Itt rámegy nyelvváltásra és az alkalmazás átváltja a nyelvét. Fontos megemlíteni, hogy az alkalmazás az űrlapok nyelvét nem váltja át, csak az alkalmazását.

Ha a kijelentkezésre megyünk, akkor az alkalmazás először üzenetet küld a szervernek, amiben tudatja, hogy az adott felhasználó kijelentkezik. Ezt követően az adott *activity* a loginfelület kivételével az összes *activity*-t elindítja úgy, hogy

átad neki egy *intent*-ben a *LOGOUT-EXIT* kulcsú *boolean* értékű üzenetet. Ez által minden felület elindul (függetlenül attól, hogy eddig futott-e) és megvizsgálja, hogy az átadott *intent* tartalmaz-e *LOGOUT-EXIT* kulcsú értéket. ha igen, akkor meghívja a *finish* függvényt, mivel leállítja magát. miután az adott oldal ahol kijelentkeztünk, leállította az összes oldalt, saját magára is meghívja a *finish* függvényt, így az is leáll, így pedig az alkalmazás is.

## 6.2 Szerverkapcsolat

A szerverkapcsolat kialakítása és az onnan érkező adatok feldolgozása olyan metódusok, mit több felület is használ így ezt a feladatot külön osztályokként definiálva hoztam létre. A szerverkapcsolat (*ServerConnection*) az osztályban kerül megvalósításra a szervernek küldött, és a szervertől kapott üzenetek kezelése. Az üzenet elkészítését részben ez az osztály, részben a JSON menedzser (*JsonManager*) osztály végzi.

### 6.2.1 Üzenetek

A szervernek az alkalmazás több fajta üzenetet küldhet. Az üzenetek előre meghatározásra kerültek a tervezés során, mert a kliens és szerver oldal is ezeket az üzeneteket használja, kezeli, és ha az üzenetek felépítése különbözne, akkor a kommunikáció se tudna létrejönni. Az üzenetek fajtái:

- Bejelentkező üzenet: Ennek az üzenetnek a segítségével jelentkezik be a felhasználó, erre válaszként a szerver vagy hibaüzenetet küld (ha hibás a felhasználónév vagy a jelszó) vagy a felhasználó azonosító kódját.
- Kijelentkező üzenet: Mikor a felhasználó kijelentkezik a mobil alkalmazásból, akkor erről a szervernek is értesítést küld az applikáció, hogy a rendszer naplózni tudja, ki van be és ki van kijelentkezve.
- Űrlapösszegző üzenet: A bejelentkezés után, a főoldalon lévő adatok megjelenítéséhez az alkalmazás lekéri az adott felhasználóhoz tartozó összes, kitöltetlen, és leadott űrlapok számát, ami ha a kapcsolat

működése zavartalan, válaszként megkap és feldolgozásra majd megjelenítésre kerül.

- Űrlapok adatait lekérő üzenet: Az alkalmazás ahhoz, hogy a felhasználó meg tudja tekinteni a hozzá tartozó űrlapokat, lekéri az űrlapok adatait. Itt fontos megjegyezni, hogy ebben az üzenetben nem szerepel az űrlapok váza, csak az űrlapok adatai (űrlap címe, kitöltési határideje stb..).
- Űrlapok megnyitását rögzítő üzenet: A MUI applikáció minden egyes alkalommal, mikor egy űrlap megnyitásra került, üzenetet küld a szervernek arról, hogy melyik űrlapot nyitotta meg a felhasználó.
- Űrlap tartalmát lekérő üzenet: Az alkalmazás, ha meg akar nyitni egy űrlapot, akkor (csak) annak az űrlap vázát lekéri a szerverről és továbbadja az információt az adott felületnek.
- Űrlapmentés üzenet: Ha a felhasználó elmenti az adott űrlapot, az nem a készüléken, hanem a szerverre kerül eltárolásra. Ilyenkor egy üzenetbe téve az adott űrlap vázat visszaküldi az alkalmazás a szervernek, ami menti azt.
- Űrlap leadás üzenet: Ez az üzenet szinte megegyezik az előző üzenet típussal (Űrlapmentés üzenettel) annyi a különbség, hogy mikor a szerver ezt az üzenetet kapja meg akkor, az űrlapot véglegesíti, leadottnak állítja.
- Hibaüzenet: A Hibaüzenetet a szerver küldi az alkalmazásnak abban az esetben ha, valami probléma lép fel a működésében, vagy a szervernek intézett lekérés hibás.

Az első két üzenetet a felhasználókat menedzselő oldal, míg a maradék üzenetet az űrlapmenedzser oldalnak küldi az alkalmazás.

### 6.2.2 Szerverkapcsolat létrehozása

Mint azt korábban említettem a szerverrel a kliensalkalmazás POST-GET kapcsolat segítségével kommunikál. A kapcsolat kialakítása a következő módon valósul meg:

1. Az adott felület, ami adatokat szeretne lekérni, vagy módosítani, meghívja a példányosított szerverkapcsolat osztály (*ServerConnection*) adott függvényét.
2. A *ServerConnection* objektum adott függvénye az attribútumként átadott adatok segítségével, legenerálja az üzenetet, amit a szervernek akar küldeni.
3. A meghívott függvény elkészíti az URL-t amibe belekerül a generált üzenet, majd ezt a webcímet átadja a *getStringFromServer* függvénynek.
4. A *getStringFromServer* metódus megnézi, hogy a készülék online-e, ha igen, létrehozza a kapcsolatot, tehát megnyitja az attribútumban megkapott weboldalt. Ha a weboldal megnyitása közben hiba történik, vagy a készülék nem online akkor *null* értékkel tér vissza a függvény, jelezvén hogy a kapcsolat sikertelenül zárult. Viszont ha az oldal jól letöltődik, és a letöltött oldal tartalma nem hibaüzenet, akkor az oldal tartalmával tér vissza a függvény.
5. Miután a felület által meghívott függvény megkapta az adatokat, azt tovább adja a felületnek, ami majd egy másik metódus meghívásával feldolgozza azt.

A szerverkapcsolat kialakításáért felelős osztályban az előre definiált üzeneteknek csak a váza készül el, maga az üzenet tartalmának parzolása (elkészítése) olyan sok függvényt igénylő feladat, amit külön osztályokba lett megvalósítva. A következő alfejezetben ezeket az osztályokat mutatom be.

### 6.2.3 Űrlapok parzolása

A szervertől a legtöbb esetben űrlapokat, vagy űrlapokkal kapcsolatos információkat kapunk. Ezeket az adatokat, hogy könnyedén tudjuk a szervernek küldeni, és azokat onnan fogadni, JSON leíró nyelvben definiálnunk kell, az egyszerű feldolgozhatóság érdekében.

*JsonManager* osztály gondoskodik arról, hogy az általa megkapott szöveget, űrlapobjektumokká alakítsa. Ez az osztály arra való, hogy kiolvassuk az

üzenetben érkezett űrlapok adatait. Ez az objektum a *JsonFormManager* függvényeit használja. Ha egy adott űrlapot kapunk (vázsal együtt) akkor az előbb említett *JsonFormManager* objektum segítségével parzoljuk. *JsonFormmanager* csak az űrlapok adatait szedi ki a JSON üzenetből, magát az űrlap vázát *JsonFormItem* osztály állítja elő. Ez az osztály űrlap szekciónként, egyesével megvizsgálja, hogy a soron következő űrlap elem milyen típusú, és annak tükrében meghívja az adott űrlapelemhez tartozó parzert, ami űrlapelemet készít az üzenetből. végezetül a memóriába elmentett feldolgozott adatok átadásra kerülnek az űrlapokat generáló osztálynak, az űrlapfelület által. A *JsonFormManager* mielőtt visszaadná az űrlap vázát, a szekciókat, és a szekciókon belüli űrlapegységeket sorrendbe helyezi az azonosítójuk alapján. (Erre azért van szükség, mert a JSON parzolás művelet nem sorrendtartó, tehát az adatok nem mindig ugyan abban a sorrendbe kerülnek átalakításra, mint ahogy megadjuk.)

Az RAM-ban lévő űrlapot a *FormParsertoJson* osztály meghívásával lehet szöveggé alakítani. Ez az osztály (ahogy a *JsonFormItems*) egyesével végig megy az űrlap szekcióin, és a szekciókon belül található űrlapelemeket JSON leírónyelvet használva szöveggé alakítja át.

### 6.3 Űrlap generálása

Az Űrlap-megjelenítő felület a beolvasott, feldolgozott adatokat átadja a *FormGenerator* osztálynak, ami a kapott értékek alapján felépíti az űrlapot. Az űrlapgenerálást a *create* metódusával lehet elkezdni. Maga az űrlapgenerálás rekurzív elven zajlik, ami azt jelenti, hogy egyesével végig megy a beolvasott szekciókon, a szekciókon belül eldönteni, hogy a soron következő elem milyen típusú, és az alapján meghívja az elemhez tartozó függvényt, mi azt a részegységet létrehozza, és felteszi a felületre.

### 6.4 Űrlapegységek generálása

Az űrlapegységeket egyesével dinamikusan kell generálni. Ennek következtében minden elemnek külön, egyedi azonosítót kell megadni. Erre azért van szükség, hogy megkönnyítsük a mentés metódus működését. (Ezek az egyedi azonosítókat

a szerver definiálja és rendeli hozzá egyesével az űrlapegységekhez.) Mindegyik űrlapegység rendelkezik Címkével, ami adott az űrlap egység elkészítése előtt létrejön. Ezen kívül közös tulajdonságok között van még a láthatóság (megjelenítésre kerüljön az adott űrlapelem vagy sem) illetve a szerkeszthetőség (ami azt tárolja, hogy az adott felhasználónak joga van-e az adott űrlapegységet szerkeszteni).

### 6.4.1 Címke

Minden űrlapegység (és szekció) rendelkezik címke objektummal. Ennek az osztálynak a célja, hogy tárolja az adott egység címét, és a cím szöveg stílusjegyeit. Címkének meg lehet adni a méretét, az elhelyezkedését (megjelenítése balra, jobbra, vagy középre igazított) a stílusát (félkövér, dőlt, félkövér és dőlt, vagy alapértelmezett normál).

### 6.4.2 Szövegmező

A szövegmező szövegeket megjelenítő mező. A szöveg egység a *TextView* objektum segítségével kerül megjelenítésre, amelyik objektumon, az előre definiált jellemzőket (szövegstílus, elhelyezkedés, méret) kell beállítani. A szövegmező azonos stílusbeállításokkal rendelkezik, mint a címke.

### 6.4.3 Beviteli mező

A beviteli mező egy olyan egység, amibe előre definiált adatokat (szöveget, számot stb..) lehet rögzíteni. A beviteli mezőnek be lehet állítani a segédszöveget (*hint-et*) ami hasznos tud lenni abban az esetben, ha a kitöltés nem egyértelmű. Ezen kívül be lehet állítani a bevitelhez milyen virtuális billentyűzet jelenlen meg. Ez a funkció azért hasznos, mert az Android támogatja azt, hogyha egy email-t kell beírni a beviteli mezőbe, akkor az alap, kezdőbillentyűzeten megjelenik a kukac gomb (@) illetve a com domain gomb (.com), ezzel is gyorsítva a bevitel sebességét. A bevitel mezőkhöz tartozhat reguláris kifejezés, ami ha definiálva van az űrlap végén, ez alapján ellenőrzésre kerül, hogy a beviteli mezőben megadott adat helyes e. (Ezt az ellenőrzést nem az űrlapgenerálás metódus végzi,



ez az űrlap beviteli mezőinek ellenőrzése alfejezetben az űrlap mentése alatt került kifejtésre.)

#### 6.4.4 Jelölőnégyzet

Jelölőnégyzet (checkbox) létrehozása közben beállítja az applikáció a szerver által hozzárendelt jellemzőket; megjelenjen-e a felületen a jelölőnégyzet, a jelölőnégyzet szerkeszthetőségét, és elhelyezkedését, majd bepipálja azt, ha alapértelmezett értéke igaz. Magát a jelölőnégyzet a *CheckBox* Android osztállyal hozza létre a metódus.

#### 6.4.5 Rádiógomb-csoport

A rádiógomb csoport, rádiógombokat tartalmaz, és tárolja, hogy melyik rádiógomb van kiválasztva. Implementálása során először a rádiócsoportot hozza létre az alkalmazás, aminek beállítja a servertől kapott azonosítót, és az általános jellemzőket (például szerkeszthetőséget), majd egyesével hozzáadja a rádiógombokat a hozzá tartozó szöveggel és azonosítóval. Ezt követően bejelöli azt a rádiógombot, ami alapértelmezetten ki van választva. Rádiócsoportot a *RadioGroup* objektummal, a rádiógombot pedig a *RadioButton* osztállyal kell implementálni.

#### 6.4.6 Kép

Az űrlap tartalmazhat képet is, amit az alkalmazásnak meg kell tudni jeleníteni. A kép az űrlapot tartalmazó üzenetben egy URL formájában jelenik meg. Természetesen ezen információ mellett rendelkezik azokkal az általános jellemzőkkel, mint a többi; láthatóság, szerkeszthetőség, címke, azonosító. Ezek mellett két egyedi paramétert is tartalmaz az üzenet ezen része, a kép szélességét és hosszát. A kép méreteire azért van szükség, mert az alkalmazás megvizsgálja, hogy a kép szélessége nagyobb-e mint a képernyő szélessége, és ha igen, akkor méretarányosan lecsökkenti. Ennek a folyamatát a következő kódsor szemlélteti:

```

DisplayMetrics dMetrics = new DisplayMetrics();
activityForm.getWindowManager().getDefaultDisplay().getMetrics(dMetrics);
int screenWidth = dMetrics.widthPixels;
int pHeight = screenItemImage.getHeight();
int pWidth = screenItemImage.getWidth();
double height = pHeight;
double width = pWidth;
if (pWidth > (screenWidth - 4.0)) {
    double temp = (screenWidth - 4.0) / pWidth;
    width = (screenWidth - 4.0);
    height = height * temp;
}

```

A példányosított *DisplayMetrics*-be a függvény elmenti először a képernyő méreteinek adatait, majd ezt követően tudjuk kinyerni a képernyő szélességét. (A képernyő hosszára azért nincs szükség, mert a kép bármilyen hosszúságú lehet, azt nem kell vizsgálni, hiszen ha nem fér ki a képernyőre, akkor túlnyúlik, ami könnyen kezelhető görgetéssel). Miután kinyertük a képernyő szélességét, beállítjuk a *pHeight*, és a *pWidth* változóba az workflow szerverről érkező alap képméreteket. Ha ez a szélesség kisebb, mint a képernyő szélessége (mínusz 4 pixel) akkor nem történik átméretezés, viszont ha nagyobb, akkor arányosan átméreteződik a kép az *if*-ben foglaltak szerint. Azért nem a teljes képernyőszélességgel számol a metódus, mert így nem 100%-osan tölti ki a képernyőt a kép.

A képet annak megjelenítéséhez le kell tölteni a megadott weboldalról. Ennek megvalósítása az *AsyncTask*-ból örököltetett, *DownloadImageTask* osztály feladata. Maga az osztály megkapja a kép webcímét, a képernyő méreteire igazított kép hosszát és szélességét illetve az *ImageView* változót. Ezt követően megpróbálja a megkapott *ImageView* változóba menteni az url képtartalmát. Ha sikerül neki akkor méretre igazítottan beleteszi a kép változóba, ellenkező esetben a *try catch* által elkapott hiba kiírásra kerül, a kép pedig nem kerül bele az *ImageView* osztályba tehát nem kerül megjelenítésre (viszont az alkalmazás folytatja tovább a működését).

#### 6.4.7 Számolómező

A számolómező egy speciális beviteli mező, ami rendelkezik az általános űrlapegység jellemzőkkel. Viszont ezt a mezőt nem lehet szerkeszteni, mivel

automatikusan veszi fel az értékét a hozzá tartozó képlet alapján. Maga a képlet tartalmaz két beviteli mezőhöz tartozó azonosítót, és egy műveleti jelet. A két beviteli mező lehet számokat, és lehet dátumokat váró beviteli mező is. Az előre definiált a matematikai kifejezésnek az eredménye lesz a beviteli mező értéke. Ahhoz hogy ez a funkció az űrlap kitöltése alatt működjön (tehát egyből számoljon amint a felhasználó kitöltötte az adott beviteli mezőt), ahhoz egy speciális objektumot kell hozzáadni a két figyelendő, fókusz alatt lévő szövegmezőre. Ez az objektum a *TextWatcher* osztály. Annak érdekében, hogy számot számmal, vagy számot dátummal tudjunk összeadni ahhoz 3 különböző *TextWatcher* szükséges. Első lehetséges opció, mikor számot számmal adunk össze akkor 1 fajta *Textwacher* kell, ami számot vár, és egy másik számhoz hozzáadja az értéket-mit beolvasott, és a kapott összeget beírja a kívánt szövegmezőbe. Viszont ha dátum érték is szerepel a képletben, akkor két lehetőség áll fent; a képlet első tagja a dátum vagy a képlet második tagja. Ezt a két opciót azért kell szétválasztani, mert nem mindegy hogy a szövegmezőtől szám értéket, vagy dátum értéket várunk. A következő kódrészlet bemutatja, hogy hogyan valósítottam meg egy olyan szövegfigyelő (*TextWatcher*) objektumot, ami egy számmezőre lett rátéve, és a képlet másik eleme egy dátummező, tehát a képlet eredménye is képlet lesz.

```
private TextWatcher setTextWachernumberofdate(
    final EditText editTextTemp, final EditText
    focusableEdittextdate, final String operation) {
    TextWatcher textWatcher = new TextWatcher() {

        @Override
        public void onTextChanged(CharSequence s, int start, int
        before, int count) {
            int count_temp1 = 0;
            try {
                count_temp1 = Integer.parseInt(s.toString());
            } catch (NumberFormatException ne) {
                count_temp1 = 0;
            }
            Date date =
            stringDateparse(focusableEdittextdate.getText()
            .toString());
            if (date != null) {
                if (operation.equals("+")) {
                    Calendar cal = Calendar.getInstance();
                    cal.setTime(date);
                    cal.add(Calendar.DATE, count_temp1);
                    editTextTemp
```

```

        setText(dateDateparsetoString(
            cal.getTime()));
    }
    if (operation.equals("-")) {
        Calendar cal = Calendar.getInstance();
        cal.setTime(date);
        cal.add(Calendar.DATE, (-1 *
            count_temp1));
        editTextTemp
            .setText(dateDateparsetoString(
                cal.getTime()));
    }
}

@Override
public void beforeTextChanged(CharSequence s, int start,
    int count,int after) {}

@Override
public void afterTextChanged(Editable s) {}
};
return textWatcher;}

```

A programrészlet egy olyan függvényt mutat be mi egy szám beviteli mezőre rátett szövegfigyelőt készít el a neki átadott objektumok alapján és tér vissza vele. Ahogy látható ahhoz hogy implementálni tudjuk a *TextWacher* osztályt felül kell írni az alapértelmezett függvényeit az *@override* annotációval. Mivel a célunk az, hogy a beviteli mezőbe beírt szöveg pillanatában már megváltozzon az eredmény azért az *onTextChanged* függvényt kell módosítani.

Az *onTextChanged* metódusnak az attribútumai között található egy szövegváltozó (*CharSequence s*) amiben tárolva van az a fókusz alatt lévő beviteli mező tartalma. Ezt a szöveget először át kell alakítani számmá (integerré) ahhoz, hogy később számolni tudjunk vele. azért kell *try-catch* közé tenni az átalakítást, mert hibát okozhat (és lefagyhat az alkalmazás) abban az esetben, ha nem tudja kiolvasni az adott értéket a beviteli mezőből. Ha ez a hiba esetlegesen bekövetkezik, akkor a *catch* ágba kerül a program, ami visszaad egy nullát, így nem áll meg az applikáció működése.

Miután sikeresen kinyertük az éppen beírt számot, akkor le kell kérni a képlet másik eleméhez beírt dátumot. Ez a dátum változó nem *null* értékkel rendelkezik, akkor a dátumhoz hozzáadódik, vagy kivonódik a beírt szám. Ezt hogy elérjük, a dátum változót *Calendar* változóra alakítjuk, és így hozzáadható (vagy

kivonható) a hozzáadandó nap mennyiség. Végezetül, pedig ha a dátum értéke változott, akkor átadjuk az értéket a számoló mezőnek, mi megjeleníti azt.

## 6.5 Űrlap mentése

Az űrlap mentését egy külön osztály végzi (*FormSaver*). Ennek az osztálynak meg kell adni az űrlap vázát, amibe el akarjuk menteni az űrlapba felvitt adatokat, majd a *save* függvény elmenti azokat. Maga a függvény úgy lett meghívva, hogy végigmegy az űrlap listaelemeken, és a listaelem egy olyan űrlapegység mibe adatot lehet tárolni, (tehát azokon ami nem szöveg, vagy kép), abban az esetben elmenti a felhasználó választ. Maga a ciklus úgy lett megírva, hogy azokat a listaelemeket meg se nézze, ami nem módosítható, ezzel is gyorsítva a függvény működését.

Az űrlapmentés két esetben hívódik meg, ha a felhasználó menteni akarja az adatokat, vagy ha leadni az űrlapot. Az applikáció mindkét esetben ugyan azt a mentés metódust használja, a különbség csak a szervernek küldött üzenetbe mutatkozik meg.

### 6.5.1 Űrlap beviteli mezőinek ellenőrzése

A felhasználó mikor le szeretné adni az űrlapot, az alkalmazás ellenőrzi a kitöltés helyességét. Egy ciklus végigfut a leadásra szánt űrlap összes beviteli mezőjén, ami elsősorban azt ellenőrzi, hogy az adott beviteli mező kitöltésre került-e. Ha igen akkor megvizsgálja, hogy az adott beviteli mezőhöz lett-e reguláris kifejezés hozzárendelve. Ha lett, akkor leellenőrzi, hogy passzol-e a kifejezéssel a beviteli mező, és ha igen tovább megy a ciklus. Ha valamelyik beviteli mezőt hibásnak tartja a helyességvizsgáló függvény, akkor az űrlapegység címkéjének értékével tér vissza, ellenkező esetben *null* értéket ad vissza. Az *activity* visszatérési értékből tudni fogja, hogy az űrlap helyesen lett-e kitöltve, vagy ha nem, akkor melyik az első beviteli mező, ami hibásan került leadásra. A rosszul kitöltött beviteli mező címkéjének értéke *Toast* segítségével kiírásra kerül, ezzel informálva a felhasználót, melyik beviteli mezőt írta el.

## 6.6 Segédszerver megvalósítása

Az alkalmazás teszteléséhez létre kellett hoznom egy segédszervert, ami a workflow rendszer interfész szerverét szimulálja. Ahhoz, hogy a szerver jól működjön, a szerver mellé implementálnom kellett egy adatbázist is. Az adatbázist MySQL, magát a web-szervert pedig PHP-t futtató Apache szerverrel oldottam meg.

A szerver fejlesztése során odafigyeltem arra, hogy PHP és SQL hibaüzenet ne jelenlen meg a weboldalon, minden esetben a hibaüzenet kerüljön kiírásra. (Viszont az éles rendszerben előfordulhat az, hogy valami oknál fogva hibaüzenet is megjelenjen a weblapon, ezért ezt a problémát kivételkezeléssel lekezeltem alkalmazás szinten.)

Az alkalmazás onnan értesül a kérésének sikerességéről, hogy a válaszüzenetben a szerver beállítja a kérés célját, és ha a beérkező parancs lekérés, akkor a *data* részbe beilleszti a lekért információkat. Ellenkező esetben a szerver hibaüzenetet küld vissza, minek a célját *error*-ra állítja.

A web szerveren különválasztottam a bejelentkezéssel, kijelentkezéssel kapcsolatos teendőket, az űrlapokkal kapcsolatosaktól.

### 6.6.1 Adatbázis kapcsolat megvalósítása

Minden weboldal elején példányosítom a PHP nyelvbe beépített, adatbázis kapcsolat létrehozására fejlesztett PDO osztályt aminek, a konstruktorának a következő képen adom át az adatokat:

```
$db = new PDO('mysql:host=127.0.0.1;dbname=szakdolgozat;
charset=UTF8','felhasznalonev','jelszo');
```

Ezt követően, ha az adatbázist eléri a weboldal, akkor az SQL parancsok végrehajtásához elég a PDO osztályban definiált metódusokat használni. ellenkező esetben hibaüzenetet ír ki az oldalra.

### 6.6.2 Bejelentkeztetés, kijelentkeztetés

A bejelentkezést a */login* mappában lévő *index.php* fájlban lévő PHP oldal végzi. Maga az oldal betöltés pillanatában leellenőrzi, hogy az oldal meghívásánál GET-be kap-e *message* kulcsú adatot, ha nem akkor hibaüzenet jelenít meg az oldalon. Ha kap üzenetet, utána ellenőrzi le, hogy az üzenet parzolható-e tehát a JSON leíró nyelv szerint van-e megfogalmazva az üzenet tartalma. Ha minden rendben talált, megnézi, hogy az üzenet a felhasználót be- vagy kijelentkeztetni akarja, és annak függvényében végrehajtja a feladathoz rendelt metódust.

A következő kódrészletben a bejelentkezés függvény látható:

```
function login($db, $username, $password) {
    $sql="SELECT `USERS_ID`, `USERS_NAME`,
    `USERS_PASSWORD`, `USERS_LOGIN`, FROM `users` WHERE
    USERS_USERNAME=:username AND USERS_PASSWORD=:password
    ";
    $query = $db->prepare($sql);
    $query->execute(array(':username' => $username,
    ':password' => $password));
    $users = $query->execute->fetchall(PDO::FETCH_CLASS);
    if($users == false){
        errorRespond();
    }else{
        $user = $users[0];
        updateOnline($db, $username, $password);
        $response = array('title' => array('sender' =>
        0, 'to' => $user->USERS_ID), 'message' =>
        array('action' => 'login', 'data' => array('id'
        => $user->USERS_ID)));
        echo json_encode($response);
    }
}
```

A függvény attribútumként megkapja a PDO adatbázis objektumot (ami a kód elején van implementálva, megkapja a felhasználónevet (*\$username*) és a jelszót (*\$password*). A függvény elsősorban definiálja az SQL parancsot, amit először a PDO *prepare* függvényével leellenőriztet, hogy maga a SQL parancs helyes-e. Ezt követően a parancsba beillesztésre kerül a felhasználónév, és a jelszó adat az *execute* függvény által. Ez a függvény vizsgálja azt is, hogy a két változó tartalmaz-e illegális karaktereket, olyan kifejezéseket, amivel SQL-injekciót lehetne végrehajtani. Ha a változókat helyesnek találta, akkor beilleszti a parancsba. Magát a lekérdezést a *\$query* változón meghívott *fetchall* függvény végzi el. Ennek a metódusnak a visszatérési értékét vizsgálja a soron következő *if*

metódus, hiszen ha a lekérés nem tartalmaz adatot, akkor hamis (*false*) értékkel tér vissza, ami azt jelenti, hogy nincs ilyen felhasználónevű és jelszóval rendelkező felhasználó az adatbázisban. Ebben az esetben a szerver hiba üzenetet jelenít meg az *errorRespond* függvény segítségével. Ellenkező esetben, ha van ilyen felhasználó az adatbázisban (tehát a bejelentkezési kísérletben elküldött adatok helyesek) akkor a szerver bejelentkezettre (*online-ra*) állítja a felhasználó státuszát. Ezt a *updateOnline* metódussal hajtja végre. Végezetül a *\$response* metódusba függvényként előkészíti a válaszüzenetet az alkalmazásnak. Ezt a függvényt alakítja át JSON nyelvvé, majd kiírja a weblapra.

A kijelentkezés folyamat szinte ugyan úgy zajlik, mint a bejelentkezés, annyi különbséggel, hogy az alkalmazástól, csak a felhasználó azonosító kódját kapja meg a szerver, ami alapján megkeresi a felhasználót. Ha megtalálja a felhasználót, akkor a státuszát kijelentkezettre (*offline-ra*) állítja. Amiben még különbözik a bejelentkezés metódustól az az, hogy a kijelentkeztetés nem ad vissza se hiba, se megerősítő üzenetet, mivel az alkalmazás a kijelentkezés alkalmával, már nem figyeli azt, hogy milyen üzenettel válaszol a szerver a kérésre.

### 6.6.3 Űrlapok menedzselése

Az űrlapok kezelését a */formanager* mappában lévő *index.php* fájl végzi. a weboldal meghívása pillanatában (ugyan úgy, mint a felhasználó menedzser weblap), GET-be várja a feldolgozásra váró üzenetet. Itt bármiféle hiba felmerülésekor JSON hibaüzenet ír ki a weblap.

A weboldalba befutó kéréseket, a kérések fajtánként külön függvények hajtják végre. Mikor az üzenetet megkapta a szerver kiolvassa az üzenet célját, és az alapján hívja meg a hozzá tartozó függvényt. Az összes meghívandó függvény megkapja a példányosított adatbázisosztályt, illetve az üzenetet, amiből majd a függvény maga olvassa ki a szükséges információkat.

A soron következő kódrészletben a leadott kitöltött űrlap mentését szemléltetem. Azért választottam ezt a metódust bemutatásra, mert kicsit komplikáltabb, mint a többi metódus, és nagyon hasonló a sima űrlapmentést végző függvénnyel.



```

function updateSavedFilledForm($db, $message) {
    $userid = getsender($message);
    $formid = getformid($message);
    $formbody = getformbody($message);

    if(isset($userid) && isset($formid) && isset($formbody)){
        if($userid != -1 && $formid != -1 && $formbody != -1){
            $sql="UPDATE 'forms' SET
            FORMS_FORMBODY=:formbody, FORMS_FILLED=1 WHERE
            FORMS_ID=:formid";
            $query = $db->prepare($sql);
            $query->execute(array(':formid' => $formid,
            ':formbody' => $formbody));

            $response = array('title' => array('sender' =>
            0, 'to' => $userid), 'message' =>
            array('formaction' =>
            'getupdatesavedfilledform', 'data' =>
            array('formid' => $formid)));
            echo json_encode($response);
        }else{
            errorRespond();
        }
    }else{
        errorRespond();
    }
}

```

A függvény első sorban kinyeri a felhasználó azonosítóját, az űrlap azonosítóját, illetve a mentésre váró módosított űrlap tartalmát az üzenetből. Ezt követően megvizsgálja egy dupla *if* segítségével, hogy a kellő információk tényleg azokban a változóban vannak elmentve vagy sem. Ha az egyik adat is hiányos, akkor hibaüzenetet ír ki a függvény, viszont ha az adatok rendben vannak, a függvény folytatja működését. Az adatok meglétét azért kell ellenőrizni, mert ha a felhasználó azonosító, vagy az űrlap-azonosító hiányzik, nem fog lefutni az adatbázis parancs, ha pedig az űrlapra leadott válasz hiányzik, akkor lefut a frissítő parancs, viszont az űrlap vázának *null* értéket állít be. A dupla *if*-en belül a metódus először definiálja a futtatandó parancsot, amit le is ellenőriztet és előkészítet a PDO osztály *prepare* függvényével. (Maga a parancs egy frissítő (update) parancs, ami azt jelenti, hogy egy már meglévő adatbázis sorát fogja módosítani az SQL kifejezés futtatása. Ebben a kódrészletben a parancs lefuttatása az űrlapnak leadott állapot állít be, illetve felülírja az űrlap vázát, a leadott adattal.) Ezt követően meghívja rajta az *execute* parancsot, ami paraméterbe megkapja az SQL parancs dinamikus változóit. Miután a parancs lefutott, a

függvény kiírja a képernyőre a sikeres SQL parancs lefutását megerősítő üzenetet, amiben adatként visszaadja a leadott adatlap azonosítóját.

Nem sokkal különbözik ettől a kódrészlettől a sima mentést végző metódus. Mikor a szerver olyan üzenetet kap ahol az űrlap csak mentésre kerül akkor a frissítő lekérdezésbe (a példával ellentétben) a *filled* attribútum nem íródik felül, csak az űrlap váza.

Az alkalmazás az űrlapokat egyesével (megnyitásuk pillanatában) töltik le a szerverről. Az űrlaplekérő kérések is ehhez a weblaphoz futnak be. Ebben az esetben nem *update* SQL parancs hanem *select* lekérést definiál az erre megírt függvény. Természetesen ez a metódus is leellenőrzi az üzenet tartalmának formai helyességét, illetve a kellő adatok meglétét (máskülönben nem tudja a lekérést implementálni). A lekérés eredménye egy JSON leíró nyelvvel meghatározott szöveggként tárolt űrlap lesz, amit a függvény üzenetbe rakva megjelenít a honlapon. Ha a lekérés sikertelen, vagy az applikációtól érkezett üzenet hibás, akkor a hibaüzenetet írja ki a szerver.

Az űrlapok összesítő adatait visszaadó metódust is ez a weblap végzi el. Amikor a főoldal betöltődik az alkalmazásban, akkor megjeleníti a felhasználóhoz tartozó űrlapok összesítő táblázatát. Ebben a táblázatban található, hogy mennyi az összes hozzá tartozó adatlap, ezek közül mennyit adott le, és mennyi az új (amit még nem nyitott meg egyszer se). Ezeket az adatokat az ennek az oldalnak címzett üzenet tárgyához rendelt metódus lefutásával kapja meg az alkalmazás. Maga a metódus (ha a felhasználó azonosítót rendben találta) egy *select* lekérdezést intéz a MySQL adatbázishoz. Ez a lekérdező parancs (ahogy a példakód is mutatja) annyiban különbözik az eddigiektől, hogy itt az eredményt össze kell adni, illetve meg kell adni, hogy milyen néven küldje vissza az adatokat.

```
$sql="SELECT COUNT(*) as 'result' FROM forms f, userforms uf
WHERE uf.USERSFORMS_FORMID=f.FORMS_ID AND
uf.USERSFORMS_USERID=:userid AND f.FORMS_SEEN IS NULL";
```

Ezzel a lekéréssel nem az összes sor-találattal tér vissza az adatbázis lekérés, hanem csak a szűrt sorok számával. Azért van szükség a visszatért oszlop átnevezésére (*as 'result'*) mert ha ezt nem tesszük meg, akkor olyan elnevezésű

objektumot ad vissza a lekérdezés, amiben zárójel van, és arra a PHP-ban nem lehet hivatkozni.

Az űrlap naplózásával járó feladatokat is ez az oldal látja el. Mikor a felhasználó az alkalmazásban megnyit egy űrlapot, az alkalmazás erről üzenetet küld a szervernek, ami elmenti az üzenet beérkezése időpontját, illetve az űrlap azonosítóját a naplózó táblába, illetve az űrlap megtekintve (*seen*) attribútuma felülíródik a megtekintés idejével. Ez által nyomon követhető, hogy az űrlapot megtekintette-e már az adott felhasználó, illetve ha igen, akkor hányszor, és mikor.

## 7. Tesztelés

A következő oldalakon az kerül bemutatásra, hogy hogyan teszteltem le az alkalmazást, és a szerveret, illetve milyen eredménnyel zárultak a tesztek.

### 7.1 Alkalmazás tesztelése

Az alkalmazás tesztelését, emulátoron, és Android operációs rendszert használó készüléken is leteszteltem. Mivel az alkalmazástól elvárás volt, hogy a régebbi Androidon futtató telefonokon is működjön, így az alkalmazás tesztelve lett több Android verzió is Eclipse Developer Tool emulátorának segítségével.

#### 7.1.1 Monkey teszt

A monkey teszt (majom teszt) egy olyan, Android operációs rendszerre megírt applikációkon végrehajtható stressz teszt. A tesztet konzolból lehet elindítani a következő paranccsal:

```
$ adb shell monkey -p hu.pemik.formmanager -v 100
```

A teszt körülbelül egy percet vesz igénybe, célja hogy szimulálja azt az esetet, ha a felhasználó feszült, és mindenhova tapint erősen a felületen. [14]

A teszt eredménye negatív lett, tehát nem lépett fel semmilyen probléma a teszt futtatása közben.

#### 7.1.2 Felhasználói teszt

A monkey teszt után saját magam teszteltem le az alkalmazás funkcióit egyesével, különböző problémás helyzetek (például internetkapcsolat hiánya) szimulálásával. Az alkalmazás működését Samsung Ace 2 készüléken, valamint több emulátoron elkészített mobileszköz segítségével ellenőriztem. Azért teszteltem emulátoron is, mert az Android 2.3.6 operációs rendszert alkalmazó készülékekre írt alkalmazás másképp fut az újabb operációs rendszert használó készülékeken. Míg az emulátoron kipróbált alkalmazás, akadozás nélkül, probléma nélkül működött, addig a Samsung Ace 2 telefonon észrevettem némi lassúságot a felületek váltása

közben. Ezeket az akadozásokat viszont nem maga az alkalmazás működése váltja ki, hanem a készülék kora.

Azt követően, hogy saját magam is több alkalommal saját kezűleg teszteltem az alkalmazást, különböző, direkt elkövetett hibákkal, szaktársakat és kollegákat is megkértem, hogy használják az alkalmazást, és jelezzék nekem az esetlegesen felmerülő hibákat.

### **7.2 Segéd szerver tesztelése**

A segédszervert egy lokálisan futó XAMPP (v3.2.1) szerveren teszteltem. A XAMPP által futtatott PHP oldalak a XAMPP MySQL adatbázisában lévő táblákat használta. A tesztelés alkalmával kipróbáltam, hogy GET-ként megadott hibás üzenetek alkalmával, minden egyes alkalommal hibaüzenetet ad-e vissza, illetve azt, hogyha helyes üzenetet kap, akkor végrehajtja-e a benne foglaltakat. Külön, egyesével leellenőriztem az összes üzenettípust, hogy mindegyik megfelelő működésre bírja-e a szervert. Az oldalak meghívása közben párhuzamosan követtem az adatbázis változásait is, tehát egy módosító, vagy naplózó üzenet érkezik, akkor be kerül-e az új sor az adatbázisba, vagy módosul-e a kívánt adat.

## 8. Összefoglalás

A szakdolgozatom keretein belül implementáltam egy Androidos alkalmazást, ami a Workflow menedzsment rendszer felhasználóinak nyújt kényelmes lehetőséget a hozzájuk rendelt űrlapok kezeléséhez.

A dolgozatom elkészítésének első fázisában elemeztem magát a Workflow rendszert, annak érdekében, hogy az alkalmazásom kompatibilis legyen a fő projekttel illetve azért, hogy teljes mértékben kitudja szolgálni a felhasználók igényeit. A menedzsment rendszer tanulmányozásával párhuzamosan, utána jártam, hogy milyen technológiai lehetőségek állnak rendelkezésre a probléma megoldásához.

A szakdolgozatom írásának második szakaszában, mikor már ki lett választva az összes technológia, az applikáció specifikációja alapján elkezdtem megtervezni az alkalmazást. Itt igyekeztem úgy megtervezni a kinézetet és a funkciókat, hogy egyszerű legyen az applikációt későbbiekben tovább fejleszteni. Az alkalmazás mellett, egy teszt szervert is terveznem kellett, mert a workflow rendszer a szakdolgozatom írásának évében még nem készült el teljes egészében.

A tervezést az alkalmazás és a teszt szerver implementálása követett. Először az adott részfeladathoz tartozó segédszerver egységet készítettem el, majd az ahhoz tartozó alkalmazás-részt. Ennek következtében folyamatosan tudtam tesztelni, hogy az adott feladategység működését.

Miután az alkalmazás teljes mértékben elkészült, tehát a kielégítette a feladatkiírásban megfogalmazott összes követelményt, egészébe teszteltem a rendszert, ami működésében nem találtam hibát.

## Irodalomjegyzék

- [1] Általános workflow motor ágens alapú döntéstámogatással\_v5.docx  
Workflow rendszer specifikáció – Feladat leírása (2.-4. oldal)
- [2] Általános workflow motor ágens alapú döntéstámogatással\_v5.docx  
Workflow rendszer specifikáció – Infrastruktúra (10.-17. oldal)
- [3] [http://hu.wikipedia.org/wiki/Android\\_\(oper%C3%A1ci%C3%B3s\\_rendszer\)](http://hu.wikipedia.org/wiki/Android_(oper%C3%A1ci%C3%B3s_rendszer))  
(letöltés dátuma: 2014.12.09) *Android operációs rendszer*
- [4] [http://en.wikipedia.org/wiki/Android\\_\(operating\\_system\)](http://en.wikipedia.org/wiki/Android_(operating_system))  
(letöltés dátuma: 2014.12.09) *Android operation system*
- [5] [http://en.wikipedia.org/wiki/Usage\\_share\\_of\\_operating\\_systems](http://en.wikipedia.org/wiki/Usage_share_of_operating_systems)  
(letöltés dátuma: 2014.12.09) *Usasage share of operation systems*
- [6] <https://developer.android.com/about/dashboards/index.html>  
(letöltés dátuma: 2014.12.09) *Android Dashboards*
- [7] <http://www.android-app-market.com/wp-content/uploads/2012/02/Android-architecture.jpg> (letöltés dátuma: 2014.12.09) *Android Architecture*
- [8] EKLER P., FEHÉR M., FORSTNER B., KELÉNYI I. (2012). Android-alapú szoftverfejlesztés Az Android rendszer programozásának bemutatása: *Activity-életciklus és –környezet* 49-55.
- [9] <http://developer.android.com/training/basics/activity-lifecycle/pausing.html>  
(letöltés dátuma: 2014.12.09) *Pausing and Resuming an Activity*
- [10] <http://hu.wikipedia.org/wiki/JSON>  
(letöltés dátuma: 2014.12.09) *JSON bemutatása*
- [11] <http://hu.wikipedia.org/wiki/PHP>  
(letöltés dátuma: 2014.12.09) *PHP bemutatása, története*
- [12] <http://php.net/downloads.php>  
(letöltés dátuma: 2014.12.09) *PHP Downloads*
- [13] <http://hu.wikipedia.org/wiki/MySQL>  
(letöltés dátuma: 2014.12.09) *JSON bemutatása*
- [14] <http://developer.android.com/tools/help/monkey.html>  
(letöltés dátuma: 2014.12.09) *Monkey test*

## Ábrajegyzék

2-1. ábra: Workflow rendszer architektúrája.....	14
3-1. ábra: Android architektúrája .....	17
3-2. ábra: Android oldalak életciklusai .....	18
5-1. ábra: Űrlapok listázása tervezet .....	24
5-2. ábra: Bejelentkezés folyamata .....	28
5-3. ábra: Űrlap megjelenítés folyamata .....	30
5-4. ábra: Űrlapegységek adatmodellje .....	32
6-1. ábra: Bejelentkezés .....	36
6-2. ábra: Főoldal .....	38
6-3. ábra: Űrlapok listázása .....	40
6-4. ábra: Űrlap megjelenítés 1. ....	41
6-5. ábra: Űrlap megjelenítés 2. ....	42



## CD melléklet

A CD melléklet tartalma:

- Szakdolgozat\_LakatosGergely\_QQEM9L.docx
- Szakdolgozat\_LakatosGergely\_QQEM9L.pdf
- abrak/
  - Workflow\_rendszer\_architekturaja.png
  - Android\_architekturaja.png
  - Android\_oldalak\_eletciklusai.png
  - Urlapok\_listazasa\_tervezet.png
  - Bejelentkezés\_folyamata.png
  - Urlap\_megjelenítés\_folyamata.png
  - Urlapgyűsek\_adatmodellje.png
  - Bejelentkezés.png
  - Fooldal.png
  - Urlapok\_listazasa.png
  - Urlap megjelenítés\_1.png
  - Urlap megjelenítés\_2.png
- alkalmazas\_forraskod/
  - alkalmazas\_forraskod.rar
- tesztserver\_forraskod/
  - tesztserver\_forraskod.rar
- fuggosegek/
  - adt-bundle-windows-x86\_64-20140702.zip
  - xampp-win32.zip
- hivatkozasok/
  - hivatkozasok.rar