Miskolci Egyetem
Gépészmérnöki és Informatikai Kar
Általános Informatikai Tanszék

RestaurAPP

Az ételrendelő alkalmazás

Szakdolgozat

Készítette:
Név: Gál Gábor
Neptunkód: GQEXYY
Szak: Mérnök Informatikus
Infokommunikációs szakirány
Cím: Sárospatak Árok út 5.
Tervezésvezető: Szűcs Miklós

Tartalomjegyzék

1.Bevezetés

Az Internet terjedésével egyre több szolgáltatás jelent meg az emberek számára. A statikus tartalmakat hamar felváltották a dinamikus tartalmak, ezzel utat engedve a különböző script nyelveknek (PHP, esetleg Javascript –AngularJS es extJS -) illetve a programozási nyelveket is bővítették HTTP szolgáltatásokkal

Hamar elérhetővé váltak a különböző webáruházak is. Ahol a vásárlók a legkülönbözőbb tárgyakat szerezhették be. Megjelentek az online piacterek is, ahol a felhasználók egymás között értékesíthetik megunt vagy feleslegessé vált értéktárgyaikat. Így viszonylag hamar megjelentek az online ételrendelő oldalak is.

Az Internet fejlődését felborította a különböző okos eszközök megjelenése. Fontossá vált ezekre a készülékekre optimalizálni a weboldalak megjelenését, valamint elkészültek a különböző szolgáltatások platformra szánt native alkalmazásai.

A mobil alkalmazások fejlődését aktívan figyelve vettem észre, hogy az alkalmazások között már megtalálhatóak a legnépszerűbb szolgáltatások (Facebook, instagram, Youtube stb…), de ,mint online étel rendelős alkalmazások nem találhatóak meg.

Így szakdolgozatom elkészítése során célul tűzöm ki számomra, hogy egy ilyen alkalmazást készítek el. Megismerve, összehasonlítva a felhasználható technológiákat, úgy mint mobil eszköz platformok, programozási nyelvek, valamint, hálózati kommunikációra használható megoldások.

2. Irodalom feldolgozás

Szakdolgozatom elkészítése során belemerültem a felhasználható technológiák sokaságába, ezért a következő pontokban röviden a teljesség igénye nélkül bemutatnám a három főbb mobil operációs rendszert, melyeket áttanulmányoztam munkám során. Napjainkban az okos telefonok piacán gyakorlatilag három operációs rendszer verseng egymással. Az Apple által fejlesztett IOS, a Google által fejlesztett Android, és a Microsoft Windows Phone-ja. Természetesen léteznek még egyéb operációs rendszerek de ezek az okos telefon piac elenyésző részét teszik ki.

2.1 Windows phone

Az első általam vizsgált platform a Microsoft által fejlesztett Windows Phone nevű mobil operációs rendszer.

1. ábra Windows Phone logo

Kezdetekben még Windows Mobil néven ismerhettük, ám amikor a Microsoft felvásárolta a Nokia céget létrehozták a mai Windows Phone operációs rendszert. A Microsoft 2010 február 15.-én bemutatta a Windows Phone 7-et. Az operációs rendszer CE alapokon nyugszik. A Microsoft fejlesztésű operációs rendszer nem meglepő módon a .net keretrendszerre épül, és C# programozási nyelvet alkalmaz. Könnyedén fejleszthető rá alkalmazás, csupán egy Visual Studio nevű fejlesztő környezetet kell hozzá beszereznünk.

2011-ben megérkezett a Windows Phone 7.5-ös verziója amit a Mango névvel illettek. Az új verzió rengeteg új funkciót hozott, többek között az elfordítható rendszerszintű

szolgáltatásokat, több nyelvvel bővült a rendszer, közöttük a magyarral is. Lehetőség volt például saját csengőhang beállítására, amire az előző verzióban még nem volt lehetőség.

2012. október 29.-én a nagyközönség előtt bemutatták a Windows Phone 8-at. Az új generáció szakítva az elődje által használt CE alappal, a Windows NT kernelt választotta alapjaként. Ez volt az első operációs rendszer ami ilyen alapot használt, ez gyakorlatilag azt jelenti, hogy a kernel megegyezik a Windows 8 kernelével. Az új operációs rendszer javított a fájlkezelésen, növelte a média és grafikai támogatást, és nagyobb biztonságot tett lehetővé.

Az NT kernel segítségével a Windows Phone most már támogatja a többmagos processzorokat. Mivel az NT kernel is ezáltal a Windows Phone 8 is támogatja a natív 128 bites BitLocker titkosítást, és a biztonságos rendszerindítást.[4]

Az előző verzióval ellentétben a 8-as verzió már használ valódi multitasking-ot amely lehetővé teszi a fejlesztők számára, hogy alkalmazásokat futtassanak a háttérben, illetve egy gombnyomásra előhívják azokat. A felhasználó kedve szerint váltogathat a futó alkalmazások közül, felfüggesztheti, illetve törölheti azokat, ha például már épp lemerülő félben van a telefon akkumulátora, és tartalékolni szeretné.

2014-ben jelent meg a 8.1 es verzió aminek a legfőbb újítása a Cortana személyi asszisztens, hasonlóan az Apple által fejlesztett Siri-hez.

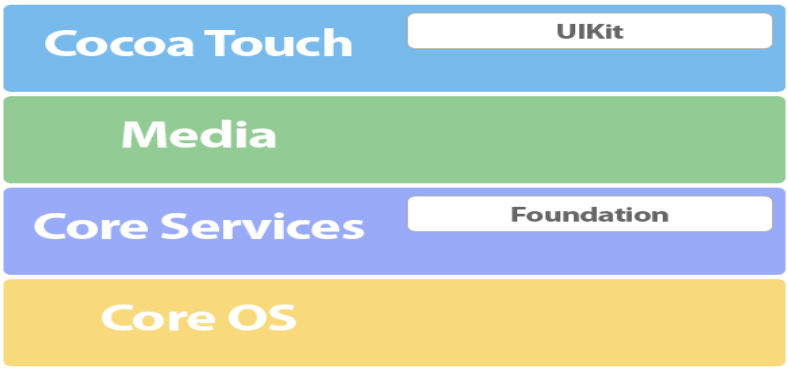
2015 január 21.-én adták ki a Windows 10 Mobil-t, jelenleg ez a legfrissebb verzió.

2.2 IOS

2. ábra Apple logo

Az iPhone OS rövidített nevén IOS-t, az Apple cég fejleszti kizárólag saját készülékeire. Az operációs rendszer első verzióját 2007 januárjában mutatták be a nagyközönség előtt Steve Jobs kezében az első iPhone-nal együtt a WWDC fejlesztői konferencián. 2007 októberében az Apple úgy döntött, hogy kiad egy natív SDK-t vagyis egy szoftver fejlesztő környezetet amivel a felhasználók kezébe adja, hogy alkalmazásokat fejlesszenek. Azzal, hogy az IOS csak Apple termékekre érhető el, nincsen töredezettség, és a fejlesztőknek sem kell a rengeteg különböző felépítésű és funkciójú készülék hardverére optimalizálni a rendszert és az alkalmazásaikat. Ezáltal a legstabilabb rendszert érték el az okos telefon operációs rendszerek közül. Rendszerük robosztus, de sok esetben korlátolt. A korlátokat inkább a személyre szabhatóságra értem, mivel itt jóval kisebb a személyes szabadság, nem tehet meg bármilyen módosítást a felhasználó kedve szerint.

Absztrakciós rétegek az IOS-ben



3. ábra Absztrakciós rétegek

Alulról felfelé haladva az első réteg a Core Os. A Core Os az operációs rendszer alapjául szolgál. Ez a réteg felel a hálózati kapcsolattartásokért a memóriakezelésért, a fájlrendszerért és az operációs rendszer egyéb feladataiért is. A réteg néhány összetevője: socketek, BSD, fájlrendszer OS X kernel stb.

Második réteg a Core Services amely az alapszintű hozzáférést biztosítja az IOS szolgáltatásaihoz. Ezen réteg az alábbi komponensekből épül fel: címtár, gyűjtemények, Core Location, szálkezelés, URL-segédprogramok, stb.

Harmadik rétegünk az alkalmazásokban használatos multimédia szolgáltatásokat biztosítja számunkra. Néhány összetevője: OpenGL, videó lejátszás, JPG, PNG, PDF, hangfelvétel, stb.

Utolsó réteg a Cocoa Touch amely egy olyan réteget ami különböző könyvtárakat biztosít iPhone-ok programozásához, például: vezérlők, többérintéses vezérlőelemek, gyorsulásmérő, lokalizáció, stb.

Napjainkban az IOS 9-es verziója a legfrissebb amit 2015 júniusában adtak ki. A verziók alatt nagyon sok változás ment végbe az operációs rendszerben de mindvégig törekedtek a megbízható stabil működésre. Különböző technikai megoldásokat is beiktattak a különböző verziókba, mint például az újlenyomat olvasó, amivel lezárás után feloldhatjuk a telefonunkat. Illetve a legújabb dolog a 3D touch, ami azt jelenti, hogy különböző erősséggel nyomjuk meg a telefonunk érintő felületét más-más funkció érhető el vele.

Összevetve más operációs rendszerrel az IOS rendszer a futáshoz jóval kisebb hardvert igényel, ugyanis sokkal jobban vannak optimalizálva az alkalmazások, és minden egyéb a rendszerhez.

2.3 Android

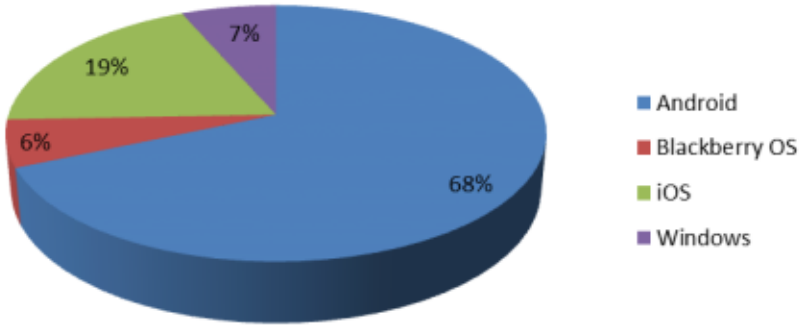
Harmadik egyben utolsó mobil operációs rendszer az Android. Mivel az alkalmazásomat erre az operációs rendszerre fejlesztem, ezt valamivel bővebben mutatnám be mint a társait.

4. ábra Android logo

Manapság az okos telefon operációs rendszerek közül a legnagyobb hányadát az android teszi ki. Ennek az egyik oka, hogy a legtöbb telefonra az android érhető el, és nem csak a csúcskategóriás telefonokon találkozhatunk vele.

Az Android platform abból a célból született, hogy egy egységes nyílt forrású operációs rendszere legyen a mobil eszközöknek. Az alap ötlet egy Linux alapú operációs rendszer volt, amelyen olyan módosításokat kellett végrehajtani, hogy gond nélkül tudja kezelni a mobilok integrált hardvereit, mint például az érintőképernyőt. A rendszer elsődleges fejlesztési nyelve a Java. A Google felvásárolta az Android nevű céget, ezzel új irányt és új lendületet adva a fejlesztéseknek. A korábbi Linux kernel fölött most már egy Dalvik virtuális gép üzemel, ami korlátozott CPU sebességű és memóriával rendelkező mobil eszközökre lett optimalizálva. A fejlesztők részére ingyenes fejlesztési lehetőséget biztosít.

Operációs rendszerek eloszlása:

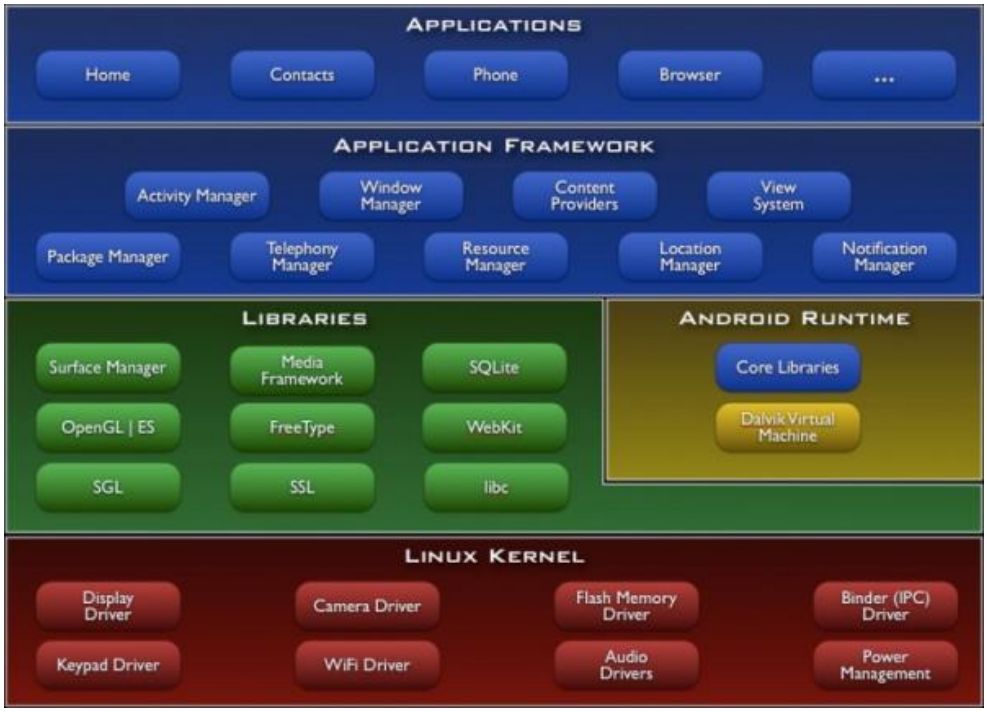


5. ábra operációs rendszerek piaci eloszlása

Lényegesebb android verziók:

Verzió	Elnevezés	Kiadás dátuma
2.2	Froyo	2010. május 20.
2.3.3-2.3.7	Gingerbread	2011. február 9.
4.0.3-4.0.4	Ice Cream Sandwich	2011. december 16.
4.1-4.3	Jelly Bean	2012. július 9.
4.4	KitKat	2013. október 31.
5.0.0-5.1.1	Lollipop	2014. november 3.
6.0	Marshmallow	2015. szeptember 29.

A rendszer szerkezete:



6. ábra Android rendszer szerkezete

A platform egyszerűnek tűnik, és messziről nézve egyszerű is. Mint láthatjuk, a platform alapját a vörös színnel jelölt Linux kernel adja, amely tartalmazza a hardver által kezelendő eszközök meghajtó programjait. Ezeket azon cégek készítik el, amelyek az Android platformot saját készülékükön használni kívánják, hiszen a gyártónál jobban más nem ismerheti a mobil eszközbe integrált perifériákat. Ez a kis méretű kernel adja a memória kezelését, a folyamatok ütemezését és az alacsony fogyasztást elősegítő teljesítmény-kezelést is.[5]

A kernel szolgáltatásait használják a Linux rendszerekben meglévő különféle programkönyvtárak, mint a libc, az SSL vagy az SQLite. Ezek C/C++ nyelven vannak megvalósítva, és a Linux kernelen futnak közvetlenül. Részben ezekre épül a Dalvik virtuális gép, amely egyáltalán nem kompatibilis a Sun virtuális gépével, teljesen más az utasítás készlete, és más bináris programot futtat. A Java programok nem egy-egy .class állományba kerülnek fordítás után, hanem egy nagyobb Dalvik Executable formátumba, amelynek kiterjesztése .dex, és általában kisebb, mint a forrásul szolgáló .class állományok mérete, mivel a több Java fájlban megtalálható konstansokat csak egyszer fordítja bele a Dalvik fordító. A virtuális gép más, mint a Java alatti megszokott virtuális gép, vagyis a Java csak mint nyelv jelenik meg![5]

A kék színnel jelölt részekben már csak Java forrást találunk, amelyet a virtuális gép futtat, s ez adja az Android lényegét: a látható és tapintható operációs rendszert, illetve a futó programokat. A virtuális gép akár teljesen elrejti a Linux által használt fájlrendszert, és csak az Android Runtime által biztosított fájlrendszert láthatjuk.[5]

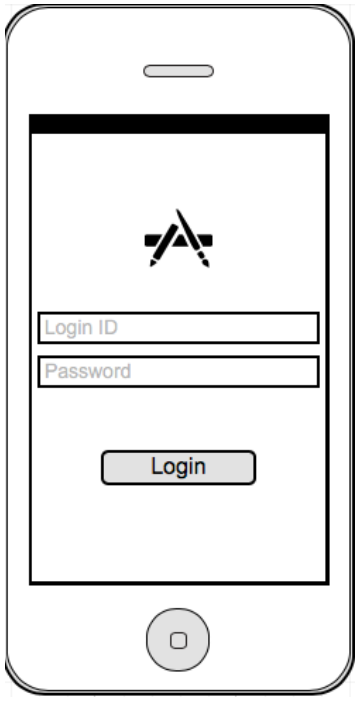
3. Esettanulmány

Szakdolgozatom készítése során vizsgálódtam mind a Google Play Store, mind az Apple AppStore nevű alkalmazás áruházában is. Mindkét központban megtalálhatóak a különböző ételekkel kapcsolatos alkalmazások, mint pl. étel vagy étterem kvízek, recept alkalmazások melyekkel saját konyhánkban készíthetünk el szinte bármit. A Google Maps-en illetve Apple Maps-en de akár már a közösségi oldalakon is (Facebook, Google+) értékelhetünk különböző vendéglátóhelyeket.

Megfigyeléseim alapján otthonunkba csakis böngészőn keresztül áll módunkban ételt rendelnünk. Ezért szakdolgozatom cél kitűzése, hogy egy olyan alkalmazást készítsek el, amely az online étel rendelésre szolgáló weboldalak által nyújtott és jól ismert szolgáltatásokat nyújtja.

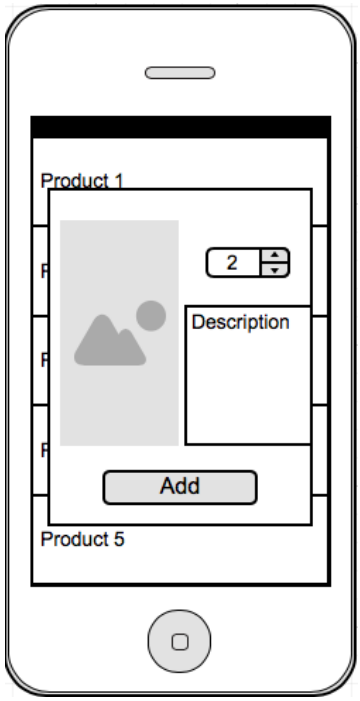
3.1 Technikai Specifikáció

Így egy rövid regisztrációt követően, ahol szükséges felhasználónevet és jelszót választani illetve egy címet megadni az alkalmazás már használható is. A jelszavas védelemre szükség van, hogy ne rendelhessen senki sem helyettünk, ezzel előidézve, hogy váratlanul is megjelenjen a futár az ajtónk előtt.



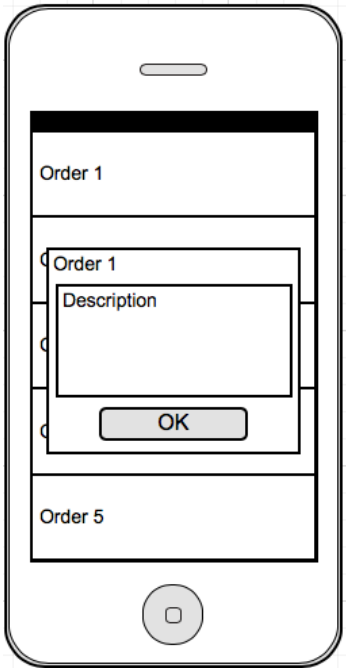
7. ábra Login képernyő

Az alkalmazás funkciói közé tartozik az éppen aktuális kínálat listázása. Ahol megtekinthetjük az étel nevét, rövid leírását illetve képet is megtekinthetünk róla. A felhasználó rendelkezik egy kosárral, amibe ételeket tehet be illetve vehet ki onnan. Beállíthatja, hogy miből mennyit kíván elhelyezni ott, amelyet később is módosíthat.



8. ábra Rendelhető ételek, italok

Lehetősége van a felhasználónak megtekintenie az eddigi rendeléseit is, pontos leírással, hogy miből mennyit rendelt. Továbbá napra pontosan, hogy mikor adta fel a rendelését és, hogy mi a rendelése státusza, éppen kiszállítás alatt van vagy azt már kézbesítették.



9. ábra Rendelések

Végezetül pedig a felhasználó tud felvenni több szállítási címet is. Mivel az az általános tapasztalat, hogy erre szüksége van az embereknek, már ha csak azt vesszük alapul, hogy különböző címen van az otthonuk és a munkahelyük is, ha már csak a két leg alapabb opciót vesszük alapul.

10. ábra Szállítási cím hozzáadás

A felhasználók minden adatai szerver oldalon tárolódnak relációs táblákban. Melyek a megfelelő menük meglátogatásakor kerülnek letöltésre a mobil alkalmazásba, valamint tárolásra ideiglenes cachekbe. Mikor új adatok kerülnek felvételre, az adatok újra letöltésre kerülnek.

3.2 Felhasznált technológiák

Mivel az éttermek fő profilja nem a web fejlesztés, ezért nagyon fontos, hogy ingyenes technológiák kerüljenek felhasználásra, mivel tanulmányaim során több opensource illetve szabadon felhasználható technológiával ismerkedhettem meg, volt miből technológiát választanom.

Szakdolgozatom elkészítése során számos technológiával megismerkedtem és alkalmaztam is őket. Ahol csak tehettem törekedtem az ingyenes verziók használatára.

Adatbázis kezelő alkalmazások közül a MySQL re esett a választásom, amely a céljaimnak teljesen megfelel és ingyenesen letölthető. Szintén ingyenesen elérhető a grafikus fejlesztőkörnyezet a MySQL Workbench.

A fejlesztéshez különböző keretrendszereket is használtam, hogy egy sokkal rugalmasabb jól integrálható alkalmazást hozhassak létre, így esett a választásom a Spring keretrendszerre az alkalmazásban a munkamenetek megvalósítására. A MyBatist az adatbázis kapcsolat kialakítására, lekérdezések futtatására, valamint új adatok felvételére használtam. Ezek a keretrendszerek terjedelmességük miatt nem érhetőek el az interneten exportált lib formájában, ezek letöltéséhez és menedzselésére a Maven alkalmazásra van szükség, amelyet preferálnak Enterprise alkalmazások fejlesztése során.

3.2.1 MySQL

A MySQL népszerűségének köszönhetően, sok operációs rendszerre elérhető, így többek között telepíthetjük különböző Windows változatokra, Linux verziókra, és a MAC által használt OS X rendszerekre. Forráskódját tekintve nagyrészt C-ben illetve C++-ban íródott, valamint a mai legnépszerűbb, és legtöbbet használt programozási nyelvekhez készítettek API-t is, így használhatunk MySQL adatbázist ha Java-ban vagy C#-ban készítünk alkalmazást.

Felhasználása széles körű az ingyenesen használható web alkalmazásoktól, a manapság igen népszerű, nagy rendszerekig, például Facebook, Google, Twitter.

A MySQL relációs adatbázis kezelő rendszer a telepítés során nem áll rendelkezésünkre grafikus felhasználói felület, ezzel próbálták a fejlesztők a lehető legkisebbre venni az alkalmazás méretét és a lehető legszélesebb körben elérhetővé tenni.
Telepíthető olyan rendszerekre is, amelyeken nem fut grafikus ablakkezelő felület. A telepítést követően mindössze egy parancssoros kezelőfelület áll rendelkezésünkre a szükséges adminisztratív jellegű beállítások elvégzéséhez, de a fejlesztéshez használhatjuk a készítők által külön letölthető alkalmazást a MySQL Workbenchet.[9]

3.2.2 Spring Keretrendszer

A Spring Framework egy nyílt forráskódú keretrendszer a Java platformhoz. Az első verziót Rob Johnson készítette el ami 2003 júniusában jelent meg Apache 2.0 licensz alatt. A Spring keretrendszer magját képező szolgáltatásokat főként Java alkalmazás fejlesztésére használják a programozók. Ugyanakkor a Java EE platformra is elérhetők a bővítményei, amelyek web-alkalmazás fejlesztését segítik elő. Nem rendelkezik külön specifikált fejlesztési modellel, hanem az Enterprise JavaBean(EJB) modell kiegészítése, helyettesítője, vagy alternatívájaként vált népszerűvé a Java fejlesztők között.[6]

A Spring keretrendszer több önálló modulból épül fel, amelyek az alábbi szolgáltatásokat nyújtják a fejlesztők számára:

- Spring Core Container: az egész rendszer alapjául ez a container szolgál, ebben helyezkedik el az egész alkalmazás context.
- Inversion of control kontainer: a Java objektumok életciklusának kezelése és az alkalmazás-komponensek testreszabása.
- Hitelesítés és engedély kezelés: beállíthatunk biztonsági folyamatokat az alkalmazásunkban, jelenleg a Spring Security project néven fut.
- Tranzakciókezelés: többféle tranzakció kezelő API-t tartalmaz.
- Adat elérés: lehetőségünk van relációs adatbázis kezelő rendszer elérésére JDBC segítségével.
- Tesztelés: jelentős segítség, hogy integritásteszteken kívül lehetőségünk van egységteszteket is végeznünk.[6]

3.2.3 MyBatis

A MyBatis egy ingyenes Java perzisztens keretrendszer, amely XML fájlokban illetve speciális Java osztályokban tárol SQL utasításokat. Apache 2.0 alatt érhető el. A MyBatis az iBatis 3.0 egy mellék ága, amelyet az eredeti iBatis fejlesztői tartanak karban.[7,8]

Ellentétben az ORM(object relation mapping) rendszerekkel szemben a MyBatis nem tárolja a Java Objektumokat relációs adatbázisban, hanem Java metódusokat társít SQL utasításokhoz. Leegyszerűsíti az SQL lekérdezések elkészítését, és a JDBC kapcsolatokat. Könnyedén integrálható más rendszerekkel így például az általam használt Spring keretrendszerrel sem okozott gondot.[7,8]

A Spring keretrendszerrel való konfigurálás során a MyBatis egy a Java EE Bean-ként fog megjelenni, amelyet konfigurálhatunk a XML alapú Spring konfigurációs fájlban, illetve speciális Java osztályokkal is.[7,8]

3.2.4 Android SDK

Az Android publikálásakor két fejlesztő eszköz is publikálásra került, az egyik az általam is felhasznált Android SDK, mely az Oracle által specifikált Java nyelvre épül szakdolgozatom írásának idejében, de a nem rég kimondott bírói végzés értelmében a Google megszegte annak licenszelését, ezért azt a jövőben lecserélik az OpenJDK nevezetű opensource terjesztésre. Megjelent továbbá az Android NDK, amellyel teljesen natív alkalmazásokat készíthetünk Android platformra C++ nyelvben. Mivel az egyetemi tananyag jelenleg nem tért ki részletesebben a C++ programozási nyelvre, ezért nem döntöttem az NDK használata mellett.

Az Android SDK-t használva lehetőségünk van a Java nyelvben elkészítenünk Androidra szánt alkalmazásunkat. A felhasználói felületet személyre szabhatjuk XML-ben definiált úgy nevezett layout fájlokkal, valamint meghatározhatjuk azok pontos elhelyezkedését a Java nyelv adta lehetőségekkel.

Habár teljesen más fordító került az Android SDK-ba, mint az Oracle definiált és a JVM virtuális gépet is felváltotta először a Dalwik virtuális gép majd az ART (Android Runtime), az nem jelent semmilyen korlátozást a nyelvben magában, így a Java SE és Java EE minden

egyes elemet használhatjuk a fejlesztés során. Természetesen rengeteg védelmi mechanizmus van beépítve az Android rendszerbe, amely a felhasználók személyes adatait hivatottak védeni, ilyen pl. az, hogy sem a felhasználók, sem az alkalmazások nem kaphatják meg a UNIX rendszerekből nagyon jól ismert, kitüntetett felhasználói jogosultságokat, amelyekkel a root felhasználó rendelkezik.

3.3 Követelmények

Az alkalmazásban nem kerülnek tárolásra szenzitív adatok, nem tárolódik benne sem bankkártya adat, sem személyes információ a szállítási címeket leszámítva. De már a címek és az eddigi rendelések megtekintéséhez is szüksége van a felhasználónak megadni a felhasználónevét és jelszavát. Mivel nem tárolódik a készüléken személyes adat és a szerverről történő lekéréshez felhasználónév és jelszó szükséges, így mondhatjuk, hogy az alkalmazás kellően biztonságos a felhasználás szempontjából.

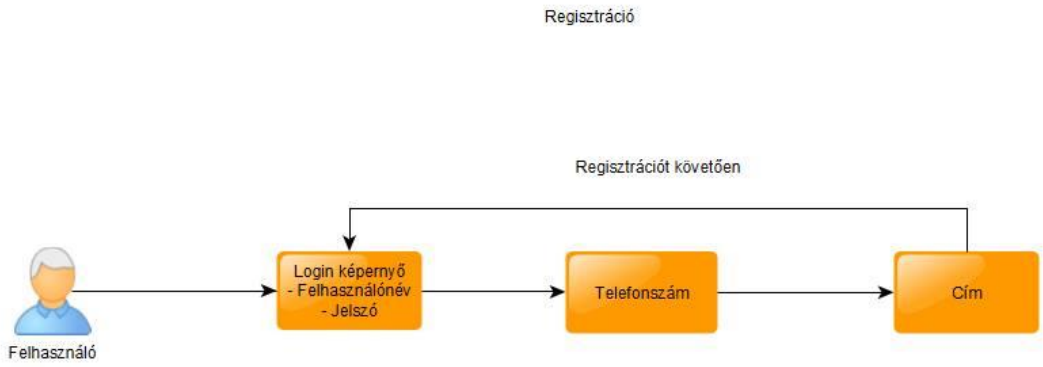
Így az elvárt követelmények az alkalmazással szemben a könnyen megérthetőség és egyszerű használat, amelyet a Google legtöbb alkalmazásához hasonlóan jól áttekinthető listákkal és oldalsó navigációs sávval igyekeztem megvalósítani, hogy a felhasználók minél könnyebben tudjanak navigálni az alkalmazáson belül. Nagyon fontos továbbá a gyors és megbízható kommunikáció a vásárlók és az üzemeltetők között, hogy megrendeléseik minél gyorsabban kerüljenek feldolgozásra és szállításra.

A szerver alkalmazása is nagyon fontos, hogy az a nap 24 órájában megbízhatóan folyamatosan a felhasználók rendelkezésére álljanak, ne legyenek fennakadások a működésében, az adatok legyenek perzisztensen tárolva, valamint az üzemeltetése is lehetőleg ingyenes vagy kis költségű legyen.

4. Implementáció

Szakdolgozatom célkitűzése egy Androidra szánt ételrendelő alkalmazás elkészítése, amely a már web böngészőkben megszokott társukhoz hasonló funkciókat nyújt a vásárlók számára. Ahhoz, hogy az alkalmazás megfelelően elláthassa azokat a funkciókat, amelyre egy ilyen alkalmazásnak a mindennapokban is szüksége lenne, nem elegendő a mobil alkalmazás elkészítése. Szükségem van egy szerverre is, amelyről a mobil alkalmazás letöltheti az éppen az aktuális ajánlatokat, a felhasználóhoz tartozó már lebonyolított rendeléseket és képes legyen új rendelések fogadására. Mivel a szerver alkalmazás, csak a mobil alkalmazással történő kommunikációt hivatott kiszolgálni szakdolgozatom keretein belül, így nem készítem el, ahhoz azt a menedzser alkalmazást, amivel az étteremben a rendeléseket kezelik. Valamint ott a felhasználó számára információkat közölhetnek az aktualitásokról, a szállítás megkezdéséről stb.

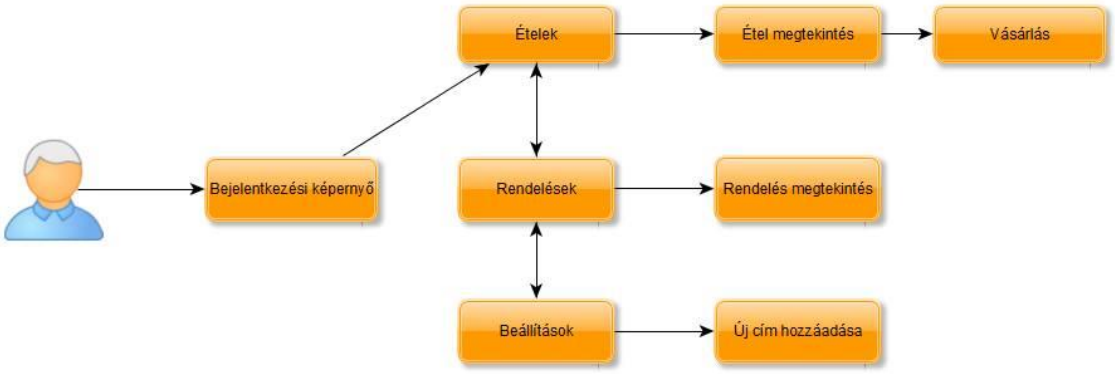
A felhasználónak az alkalmazás használata előtt szükséges van regisztrálni a szolgáltatásokhoz, ezt a folyamatot a következő ábra (11. ábra) szemléltet.



11. ábra Regisztráció

A felhasználó az alkalmazás indulásakor a bejelentkezési képernyőn találja magát, ahol kiválasztva egy tetszőleges felhasználónevet és egy kellően biztonságos jelszót, majd a regisztráció gombra kattintva érheti el a regisztráció többi szakaszát. A megjelenő dialógus ablakban kötelező megadni a telefonszámát, bármilyen hiba, késedelem esetén az étterem ezen a telefonszámon érheti el a vásárlót. Végezetül pedig a megjelenő dialógus ablakban meg kell adnia a szállítási címet, ahol majd az étel szállítva lesz. A szállítási címek listája

később bővíthető. A regisztrálást követően a felhasználó készen áll az alkalmazás használatára.



12. ábra

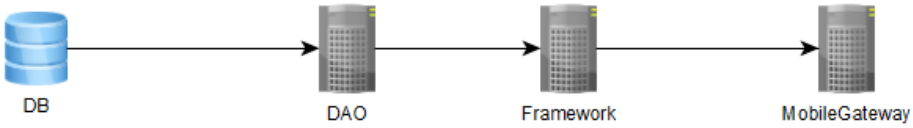
A felhasználó a sikeres belépést követően a megrendelhető ételek listáját látja. A listában megjelenő adatok a név valamint a termékhez tartozó ár. A lista elemekre kattintva részletes leírást láthat az ételről, valamint beállíthatja a mennyiséget, melyet a kosárba kíván helyezni. A kosár tartalma a jóváhagyás után frissül. Több étel is elhelyezhető a kosárban. Vásárlást megelőzően áttekinthető a kosártartalma az még módosítható. Majd a jóváhagyást követően a rendelt ételek listája továbbításra kerül a szerverre a szállítási címmel együtt.

Az ujjunkat a kijelző bal széléről befelé söpörve előhozhatjuk a navigáció sávot, ahol a további opciók közül választhat a felhasználó: Rendelések illetve Beállítások. A vissza gomb megnyomására az alkalmazás megkérdezi, hogy biztosan ki szeretnénk-e lépni, amennyiben a válasz igen a belépési képernyőre kerül a felhasználó.

A Rendelések menüpontban az eddigi rendelések kerülnek listázásra. Az itt megjelenő információk, hogy mikor történt a rendelés illetve a rendelés állapotáról. Az állapot lehet folyamatban illetve teljesült. A lista elemekre klikkelve megjelenésre kerülnek a részletes információk a rendelésről egy dialógus ablakban. Plusz információként megjelenik, hogy mit tartalmazott a rendelés.

A beállítások menüt választva a felhasználó láthatja a megadott címeit, valamint új címet vehet fel. A regisztráció követően a felhasználónak csak itt van lehetősége új címet felvenni, ami a rendelések szállítását tekintve fontos.

Mivel a szerver alkalmazás tárolja a szükséges adatokat, így az alkalmazás bemutatását innen kezdem. Az alkalmazás 3 Maven projectből épül fel (13. ábra), amelyek egymásra hivatkozva és az Android alkalmazással együtt az MVC (model-view-controller) modell hivatott megvalósítani.



13. ÁBRA SZERVER ARCHITEKTURÁLIS ELRENDEZÉSE

A DB vagyis adatbázis réteget az általam választott MySQL Server valósítja meg. Itt InnoDB-ben egy külön sémában tárolódnak az alkalmazás számára létrehozott táblák, amelyeket SQL utasításokon keresztül manipulál az alkalmazás.

Ehhez a művelethez szükséges SQL utasítások a DAO rétegben tárolódnak MyBatis Mapper XML-ekben. Ez a Framework hozza létre a kapcsolatot az adatbázissal, saját beállító állománya alapján, melyről a következő alfejezetben (alfejezet száma) részletesebben fogok ismertetni.

A Framework project végzi a DAO projectben DBből kinyert adatok konverzióját, valamint a mobil alkalmazás felől érkező adatok tárolásának menedzselését és letárolásáért is felelős. Így homlokzatot épít a DAO rétegre elfedve annak pontos működését és függőségeit, ezáltal egyszerűsítve a tárolási és lekérdezési folyamatokat. Illetve plusz szolgáltatásokat is nyújt a helyes működés érdekében. A Framework project által nyújtott néhány szolgáltatást az alfejezetben fogom részletezni.

Az utolsó project, a Mobilegateway végzi a tényleges kommunikációt a mobil alkalmazással, JSON objektumok segítségével REST Controllereken keresztül. Az általam használt és beállított vezérlők részei a Spring keretrendszernek (14. ábra). A project pontos működését az alfejezetben fogom részletezni.

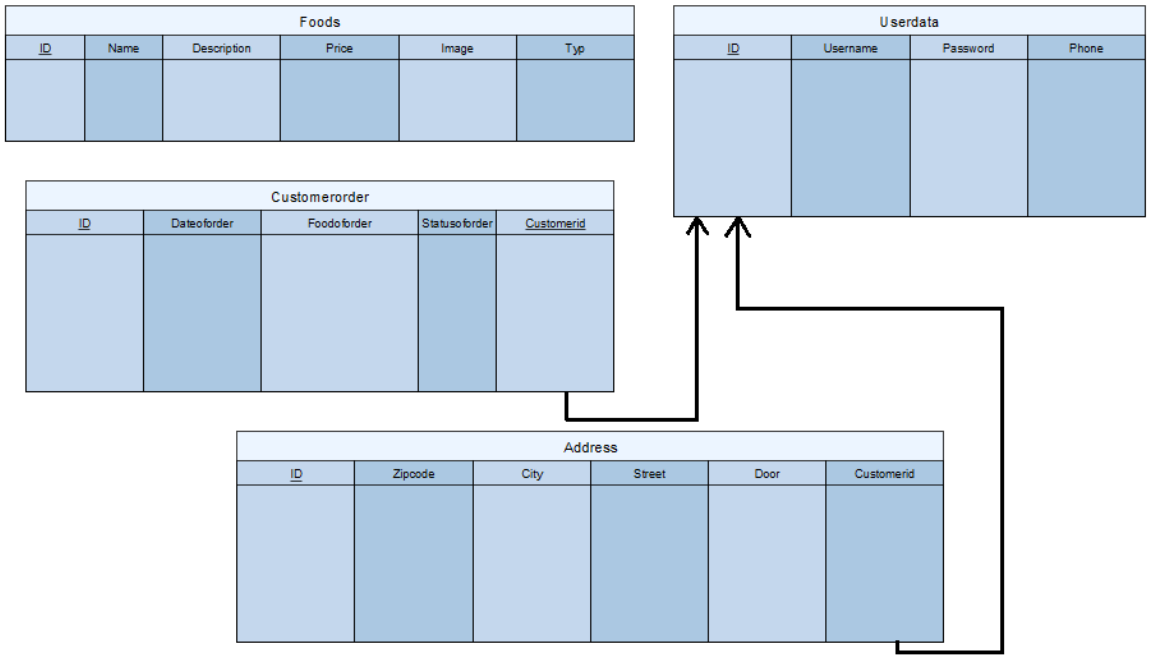
14. ábra Szerver-mobil kommunikáció

A mobil alkalmazás egyetlen projectből áll, a megfelelő tagoltságról itt a csomagok gondoskodnak, a több project híján. Valamint különböző csomagokban helyeztem el a sokszor használt funkciókat ezzel is elősegítve az alkalmazás megfelelő tagoltságát. A mobil alkalmazás projectet az alfejezetben fogom részletesen bemutatni.

Mivel a funkciók egytől egyig JSON alapú kommunikációt használnak, egyetlen kivétellel, így a Login funkcióhoz használt, általam megvalósított kódot fogom bemutatni szakdolgozatomnak ebben a fejezetében, illetve azt az egy kivételt, amely nem, ez a rendeléseknél megjelenő képek betöltését végzi, amely bájt alapú kommunikációt végez.

4.1 Az adatbázis

A mobil alkalmazás működését segítő Java EE szerver, MySQL adatbázishoz kapcsolódik és tárolja a szükséges adatokat. Az általam kialakított adatstruktúra a következő ábrán látható (15. ábra).



15. ábra Relációs modell

Az ábrán önállóan látható a Foods tábla, amely nem tartalmaz hivatkozást más tábla elsődleges kulcsára, illetve ennek a táblára sem történik hivatkozás más táblából. Itt a megvásárolható ételekről tárolódik információ, ezek az információk:

- name: a termék megnevezése
- description: a termék leírása
- price: a termék ára
- image: a termékhez tartozó képfájl neve és kiterjesztése
- typ: az étel típusa, amely lehet ital vagy étel

A Userdata tábla tartalmazza a felhasználóról tárolt adatokat, amelyek a következőek:

- username: a regisztrációnál megadott felhasználónév
- password: a regisztrációnál megadott jelszó
- phone: a regisztrációnál megadott telefonszám

A Userdata tábla kapcsolatban áll az Address és a Customerorders táblákkal. Ezekből a táblákból egy-egy idegen kulcs mutat a Userdata tábla elsődleges kulcsára, ezzel azonosítva azt, hogy melyik cím illetve rendelés melyik felhasználóhoz tartozik.

Az Address tábla a Userdata táblára mutató idegen kulcson kívül a következő mezőket tartalmazza, mint az az ábrán is látszik:

- zipcode: A város irányítószáma
- city: a város neve
- street: az városban megtalálható utca, valamint a házszám is itt kerül tárolásra
- door: a társasházak emelet illetve ajtó számát itt tároljuk

Az ábrán látható utolsó tábla a Customerorders tábla, ahol a vásárló eddigi rendelései kerülnek tárolásra, a következő információkkal:

- dateoforder: a vásárló, mikor adta fel a rendelést kerül itt tárolásra
- foodoforder: itt kerül tárolásra a rendeléshez tartozó ételek nevei, valamint azoknak a mennyisége is
- statusoforder: itt tároljuk, hogy a rendelés milyen állapotban van, folyamatban van a kiszállítása vagy a szállítás már megtörtént

A MySQL adatbázishoz történő kapcsolódást a DAO projectben valósítottam meg.

4.2 DAO – Adatbázis kapcsolati réteg

Ez a project mindössze a MyBatis beállító állományokat, mappereket és interfaceket tárolja. Az adatbázishoz történő csatlakozáshoz szükséges paraméterek a database-config.xml fájl tartalmazza (16. ábra). Ahol megadtam a kapcsolat típusát, merre található az adatbázis, esetünkben ugyan azon a számítógépen a 3306 porton. Valamint a bejelentkezéshez szükséges felhasználónevet és jelszót szükséges még megadni.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE configuration PUBLIC "-//mybatis.org/DTD Config 3.0//EN" "http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
    <environments default="development">
        <environment id="development">
            <transactionManager type="JDBC"/>
            <dataSource type="POOLED">
                <property name="driver" value="com.mysql.jdbc.Driver"/>
                <property name="url" value="jdbc:mysql://localhost:3306/restaurapp"/>
                <property name="username" value=""/>
                <property name="password" value=""/>
            </dataSource>
        </environment>
    </environments>
    <mappers>
        <mapper resource="mybatis/mapper/AddressMapper.xml"/>
        <mapper resource="mybatis/mapper/CustomerOrdersMapper.xml"/>
        <mapper resource="mybatis/mapper/FoodMapper.xml"/>
        <mapper resource="mybatis/mapper/UserdataMapper.xml"/>
    </mappers>
</configuration>
```

16. ábra database-config.xml

Ezt a konfigurációs fájlt az alkalmazás akkor olvassa fel, amikor egy új kapcsolatot létesítünk a DBvel. Ennek a kapcsolatnak a kiépítéséért felelős a Connection, amely az ábrán látható (Ábrajegyzék 17. ábra).

Ennek az osztálynak a megírásánál a jól ismert Singleton mintát vettem alapul, amely egy SQLSessionFactory-t tárol egy statikus adattagként, valamint privát konstruktorát egy látható statikus metóduson keresztül ér el, ami a getInstance nevet kapta. Ezzel garantálva, hogy az adatbázishoz a kommunikáció során mindössze csak egyetlen egy kapcsolat létezzen.

A kialakított kapcsolaton áthaladó utasításokat a MyBatis keretrendszerhez létrehozott Mapper.xml-ek tárolják. A bejelentkezésnél szükség van a Userdata table olvasásához, hogy megtudjuk, hogy a felhasználó létezik-e, valamint, hogy a felhasználónévhez rendelt jelszó megegyezik-e azzal amit a felhasználó megadott.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="hu.unimiskolc.iit.thesis.restaurapp.dao.mybatis.interfaces.IUserdata">
    <!-- ResultMap -->
    <resultMap type="hu.unimiskolc.iit.thesis.restaurapp.dao.objects.Userdata" id="UserdataResult">
        <result property="id" column="id"/>
        <result property="loginId" column="loginid"/>
        <result property="password" column="password"/>
        <result property="phone" column="phone"/>
    </resultMap>
    <!-- Transactions -->
    <select id="selectAllUser" resultMap="UserdataResult">
        SELECT id, loginid, password, phone FROM USERDATA
    </select>
    <select id="selectUser" parameterType="String" resultMap="UserdataResult">
        SELECT id, loginid, password, phone FROM USERDATA WHERE loginid = '${value}'
    </select>
    <insert id="addUser" parameterType="hu.unimiskolc.iit.thesis.restaurapp.dao.objects.Userdata">
        INSERT INTO USERDATA (loginid, password, phone) VALUES(#{loginId}, #{password}, #{phone})
    </insert>
    <update id="updateUser" parameterType="hu.unimiskolc.iit.thesis.restaurapp.dao.objects.Userdata">
        UPDATE USERDATA SET password = '#{password}', phone = '#{phone}' WHERE loginid = '#{loginId}'
    </update>
    <delete id="deleteUser" parameterType="hu.unimiskolc.iit.thesis.restaurapp.dao.objects.Userdata">
        DELETE FROM USERDATA WHERE loginid = '#{loginId}'
    </delete>
</mapper>
```

18. ábra Userdata mapper

Minden SQL utasítás az adott táblához ebben a fenti ábrán (18. ábra) látható Mapperben található meg. Az XML fájl felépítése nagyon hasonlít a Javaból már jól ismert prepared statementekhez. Meg kell adnunk, hogy milyen parameter típust vár a kifejezésünk, majd behelyettesítenünk azt, egyszerű típusoknál, mint a String vagy Integer a ${value} vagy complex példány esetén a hash{paraméternév} segítségével végezhetjük el a behelyettesítést.

Minden egyes Mapper fájlnál definiálnunk kell egy Java interfacet, amelyet meg kell adnunk a Mapperen belül is. Ezt fogja használni a MyBatis a Java-ból történő hívásokhoz, ehhez a Mapperhez tartozó Java interface a lenti ábrán látható (Ábrajegyzék 19. ábra).

Itt publikus statikus metódusokat gyűjtünk össze, amin keresztül majd meg tudjuk hívni a MyBatis funkcióit. Visszatérési érték és paraméterek megegyeznek a Java Interfacek és a MyBatis mapperek esetében. Ez fontos, mivel ellenkező esetben fordítási hibát kapunk. Ezért a DAO project tartalmaz az adatbázis tábla struktúráknak megfelelő osztályokat is, amelyeket példányosítva megkaphatjuk a megfelelő objektumokat.

Ezek Java Beanekként működnek, minden adattagjához rendelünk egy setter és egy getter metódust is. Valamint tartalmaz konstruktort is, ami beállít minden adattagját az objektumhoz.

4.3 Framework

A framework project 3 funkciónkét szétosztott homlokzaton keresztül éri el a DAO projectet. Ezek a homlokzatok valósítják meg a tényleges kapcsolat létrehozást, lekérdezést, insertálást, kapcsolat bontást. Ha a felhasználóról szeretnénk az adatbázisban tárolt információt megtudni, akkor a következő ábrán (20. ábra) látható metódusok közül az egyiket kell meghívnunk a QueryFromDB osztályból:

```
public static Userdata getUserdata(String username){
        SqlSession sqlSession = Connection.getInstance().openSession();
        IUserdata iUserdata = sqlSession.getMapper(IUserdata.class);
        Userdata userdata = iUserdata.selectUser(username);
        sqlSession.close();
        return userdata;
}
```

20. ábra QueryFromDB osztály getUserData metódusa

Míg, ha új felhasználót szeretnénk hozzáadni a rendszerhez, akkor a következő metódus lesz hasznunkra az AddingToDB osztályból (Ábrajegyzék 21. ábra).

Mindkét ábra tökéletesen szemlélteti az egységes működést. Ahol létrehozunk egy kapcsolatot az adatbázissal először is. A létrejött kapcsolathoz kiválasszuk a megfelelő Mapper osztályt, amely rendelkezik és képes lesz futtatni a megfelelő metódusokat. Ezután már csak futtatnunk kell a metódust a megfelelő paraméterrel, vagy helyet biztosítani a visszatérési értékének. A beillesztés során szükségünk van a változásokat véglegesíteni az adatbázisban egy commit utasítással, míg lekérdezésnél erre nincs szükség, végezetül pedig bezárjuk az aktív kapcsolatot.

Bejelentkezési folyamaton belül ebben a projectben az utolsó osztály, ahol az információ végig halad, az a LoginController (22. ábra).

```java
package hu.unimiskolc.iit.thesis.restaurapp.thesis.framework.controllers;

import hu.unimiskolc.iit.thesis.restaurapp.dao.objects.Userdata;
import hu.unimiskolc.iit.thesis.restaurapp.framework.database.connection.QueryFromDB;
import hu.unimiskolc.iit.thesis.restaurapp.framework.objects.Status;

public class LoginController {

    public static Status login(Userdata userdata){
        try{
            Userdata queriedUser = QueryFromDB.getUserdata(userdata.getLoginId());
            return (queriedUser != null && queriedUser.getPassword().equals(userdata.getPassword())) ? new Status(true, false) : new Status(false, true);
        }
        catch(Exception e){
            return new Status(false, true);
        }
    }

    public static Status logout(){
        return new Status(true, false);
    }
}
```

22. ábra LoginController osztály

Az osztály login metódusa paraméterben kapja meg a felhasználó által kitöltött form adatait, majd a felhasználónévhez rendelt jelszót kiolvassa a DBből: ha nincs ilyen felhasználó, akkor ennek megfelelő Status objektummal tér vissza, sikeres belépésnél az objektum sikert eredményez. Valamint az eddig megismert kapcsolatnál, bármilyen okból kifolyólag Exception dobódik futásidőben akkor sikertelen eredménnyel tér vissza a metódus.

Az osztály logout metódusa felelős a sikeres kiléptetésért, amint látszik ebben az esetben nem sok hiba történhet, a sikertelen eredményt csak hálózati hiba eredményezheti.

Az osztályban újonnan megjelent Status típus (22. ábra) ugyan úgy Java beanként működik, mint a már ismertetet Userdata objektum.

4.4 Mobilegateway

Ez a project tartalmazza a Java EE alkalmazásokból megszokott web.xml leíróállományt, valamint ami a Spring keretrendszer és a RESTful WebServices miatt szükséges plusz beállító állományokat.

Ezeknek a követelményeknek megfelelően a web.xml tartalma a következő ábrán látható (Ábrajegyzék 23. ábra):

Amiben láthatjuk, hogy rest néven fogjuk használni a Spring által definiált DispatcherServletet, valamint ezt a szervletet a /* URL-re fogjuk alkalmazni, tehát minden egyes URL ehhez a servlethez fog érkezni, ami a Tomcatben telepített alkalmazáshoz

érkezni fog. Illetve láthatjuk azt is, hogy ezt a servletet induláskor mindössze egy példányban szeretnénk elindítani.

További szükséges beállító állomány a rest-servlet.xml, amelyet a web.xml fájl mellett találhatunk a WEB-INF mappában és a következő ábrán (Ábrajegyzék 24. ábra) látható:

Az xml fájlban látható számunkra fontos információ a context:component-scan tagen belül található base-package attribútum értéke. Az itt megadott package azonosító fogja meghatározni azt az alkalmazásunk számára, hogy az hol fogja keresni a REST Controllereket.

Természetesen a Spring keretrendszer nem az összes osztályból véletlenszerűen fogja eldönteni azt, hogy az le tudja-e kezelni az éppen érkezett kérést, az itt szereplő osztályokat, vagy ha úgy jobban tetszik Controllereket annotációkkal kell ellátni, amely a kérésben URI útvonalként fog megjelenni, míg az osztályokban RequestMappekként. A bejelentkezéshez szükséges UserController az alábbi ábrán látható (25. ábra):

```
@RestController
@RequestMapping("/UserService")
public class UserController {

	@RequestMapping(value="login", method=RequestMethod.GET, headers="Accept=application/json")
	public String login(@RequestParam("loginData") String loginDataJson){
		Gson gson = new Gson();
		Userdata userdata = gson.fromJson(loginDataJson, Userdata.class);
		return gson.toJson(LoginController.login(userdata));
	}

	@RequestMapping(value="register", method=RequestMethod.GET, headers="Accept=application/json")
	public String register(@RequestParam("regData") String regDataJson){
		Gson gson = new Gson();
		RegData regData = gson.fromJson(regDataJson, RegData.class);
		return gson.toJson(RegController.register(regData));
	}

	@RequestMapping(value="addAddress", method=RequestMethod.GET, headers="Accept=application/json")
	public String addAddress(@RequestParam("address") String  addressJson, @RequestParam("username") String username){
		Gson gson = new Gson();
		Address address = gson.fromJson(addressJson, Address.class);
		return gson.toJson(RegController.registerAddress(address, username));
	}

	@RequestMapping(value="logout", method=RequestMethod.GET, headers="Accept=application/json")
	public String logout(){
		Gson gson = new Gson();
		return gson.toJson(LoginController.logout());
```

25. ábra UserController osztály

Az osztályban található első annotáció a @RestController, ez mondja meg a fordítónak, hogy ez az osztály tud fogadni rest kéréseket, így azt az előbb bemutatott rest-servlet.xml (ábra) fogja tudni használni. A következő annotációban megadunk a Controller számára egy elérési utat, ami ebben az esetben a /UserService lesz.

Ezután az osztályban szereplő publikus statikus metódusokat látjuk el plusz információval a Spring számára. Itt kerül meghatározásra, hogy az előzőleg megadott pathon belül a /login úton érjük el a bejelentkezés szolgáltatást, amely a HTTP protokollban definiált POST és GET metódusok közül a GET metódust támogatja. Utolsó plusz információnk a metódusról, hogy az csak application/json objektumokat fogad el, ha meghívják. A metódusnak paraméterképpen kötelezően meg kell adnunk egy loginData parameter, amit majd később a metódusban String objektumként fogunk kezelni.

A Google Gson APIja segítségével a Stringben megkapott adatokat konvertáljuk objektummá, majd rögtön meghívjuk a LoginController login metódusát. A metódus álltal kapott visszatérési értékét ugyan úgy a gson api segítségével Stringgé alakítjuk és elküldjük válaszol a mobil alkalmazásnak.

4.5 Mobil alkalmazás

A mobil alkalmazás egyetlen osztállyal tartja a kapcsolatot a szerverrel, ami a JSON kommunikációt illeti. Ez az osztály a Communicator (Ábrajegyzék 26. ábra):

Mely nem rendelkezik statikus metódussal, minden egy kérésnél példányosítanunk kell, megadva neki az alkalmazás contextust, a szolgáltatást, amit szeretnénk elérni és ahhoz a szolgáltatáshoz tartozó paramétereket.

Ezután van lehetőségünk meghívni a callService metódust, amely meghívja példányosítja és elindítja a kommunikációt az osztályba ágyazott CommunicatorAsyncTask osztály segítségével.

A CommunicatorAsyncTask osztály kibővíti az AsyncTask által definiált szolgáltatásokat és a kommunikációt egy új szálon indítja el, mivel a hálózati kommunikációt nem futtathatjuk Android esetén a főszálon.

Kezdetben az onPreExecute metódus fut le, amely egy töltés animációt helyez ki a felhasználói felületre. A tényleges kommunikáció a doInBackground (27. ábra) metódusban történik, ahol összeillesztjük a szerver címével a szolgáltatás elérési útját, valamint a szolgáltatáshoz szükséges paramétereket. A paraméter értékeket szükségünk van átkódolni, hogy az értelmezhető legyen a szerver számára. Ez az átkódolás a speciális karakterekre igaz, amelyek nem szerepelhetnek URL-ekben, ilyenek például az ékezetes karakterek.

```
@Override
protected String doInBackground(String... params) {
	HttpClient httpClient = new DefaultHttpClient();
	HttpResponse httpResponse;
	String response = null;
	String address = context.getString(R.string.hu_unimiskolc_iit_thesis_restaurapp_framework_network_connection_url);
	StringBuilder stringBuilder = new StringBuilder(address + service);
	stringBuilder.append(URLEncoder.encode(parameters));
	String uri = stringBuilder.toString();
	Logger.getGlobal().log(Level.WARNING, "Called URL: " + uri);

	try {
		httpResponse = httpClient.execute(new HttpGet(uri));
		if(httpResponse.getStatusLine().getStatusCode() == HttpStatus.SC_OK){
			ByteArrayOutputStream byteArrayOutputStream = new ByteArrayOutputStream();
			httpResponse.getEntity().writeTo(byteArrayOutputStream);
			byteArrayOutputStream.close();
			response = byteArrayOutputStream.toString();
		}
		else{
			Toast.makeText(context, R.string.hu_unimiskolc_iit_thesis_restaurapp_natural_try_again, Toast.LENGTH_LONG).show();
			throw new IOException(httpResponse.getStatusLine().getReasonPhrase());
		}
		//httpResponse.getEntity().getContent().close();
	} catch (ClientProtocolException e) {
		e.printStackTrace();
	} catch (IOException e) {
		e.printStackTrace();
	}
	Logger.getGlobal().log(Level.WARNING, "Response returned from server.");
	return response;
```

27. ábra CommunicatorAsyncTask osztály doInBackground metódusa

Miután a szerver válaszolt a mobil alkalmazásnak, eltávolításra kerül a töltés animáció az onPostExecute metódusban és feldolgozzuk a kapott választ. Itt ellenőrizzük, hogy semmilyen hiba nem történt a kommunikáció során, ilyen lehetne például, hogy a szerver 404-et válaszol vagy üres JSON-el válaszol. Miután ezeket az ellenőrzéseket elvégeztük, vissza kell hívnunk a főszálba. Így a vezérlés a Communicator osztály callBack metódusára kerül. Amelyet lentebb láthatunk (Ábrajegyzék 28. ábra):

A Callback metódus elvégzi a válaszban kapott JSON feldolgozását. Először is meghatározza, hogy melyik vezérlőről érkezett a válasz. Fontos kiemelnem, hogy az Android alkalmazásokat Java 1.6 fordítóval fordítjuk és a Stringere értelmezett switch utasítás, csak Java 1.7-ben jelent meg ezért, amikor a vezérlők nevét ellenőrizzük kénytelenek vagyunk if utasításokat használni.

Belépéskor mivel a login szolgáltatást hívtuk, így a vezérlés a második if utasításra kerül. Itt a kapott JSON eredményt a már szerver oldalon bemutaott Gson API konverzióval Status objektummá alakítjuk, megvizsgáljuk a benne tárolt adattagok értékét, melyek a sikerességre utalnak. Ha sikeres eredmény és nem hibás eredmény érkezett akkor az alkalmazás a

LoginActivityről elnavigálja a felhasználót a MainActivity-re, ahol kezdetben a FoodsFragment kerül betöltésre.

A navigáció:

Az alkalmazáson belüli navigáció egyetlen osztályban van megvalósítva, amely a ViewLoader nevet kapta (Ábrajegyzék 29. ábra.)

Mint az ábrán is látható 3 statikus metódust tartalmaz, amelyek közül az utolsó, exit metódus a LoginActivity-re navigál. Itt megszünteti a visszalépés lehetőségét a ManActivity-be. Majd az Androidból ismert új Intent létrehozásával a context segítségével elvégzi a navigálást.

Míg a fenmaradó másik két metódus az alkalmazáson belül bármelyik felületről bármelyik felületre elnavigálhatnak. Új Intentet létrehozva, ahol megadjuk azt az Activity osztályt, hogy hova szeretnénk navigálni, majd a kontextussal elvégeztetjük a betöltést. Az egyetlen különbség a két metódus közül, azok a String tömbként kezelt extrák, mivel a három parameters loadView metódus segítségével lehetőségünk van extra tartalom átadására a betöltendő Activity számára.

MainActivity Fragment kezelése:

A MainActivity indulásakor az onCreate metódusban betölti a FoodsFragmentet. Acitivityben Fragment töltésekor az Activity FragmentManager adattagján szükséges egy új tranzakciót indítanunk, amikor szükséges megadnunk, hogy hova kerül betöltésre az új elem, valamint melyik Fragment osztály példányát szeretnénk ide betölteni. Majd a módosításokat érvénybe léptethetjük a commit metódussal.

A sikeres betöltés után a Fragment onCreateView metódusa fog lefutni. Ahol lehetőségünk van beállítani a megjelenítendő nézetet, illetve az Activity működését kódolhatjuk.

4.6 Képek továbbítása:

A mobil alkalmazásból, amikor szükségünk van egy kép betöltésére, akkor azt az ImageLoader osztályon keresztül érhetjük el (30. ábra).

```
public class ImageLoader{

	private final static String ADDRESS = "http://192.168.2.105:8080/restaurappserver-mobilegateway/ImageService/getImage?image=";

	private String image;
	private ImageView imageView;

	public ImageLoader(ImageView imageview){
		this.imageView = imageview;
	}

	public void getImage(String image){
		this.image = image;
		new ImageLoaderAsyncTask().execute(ADDRESS + this.image);

	}

	private class ImageLoaderAsyncTask extends AsyncTask<String, String, Bitmap>{

		@Override
		protected Bitmap doInBackground(String... params) {
			Bitmap bitmap = null;
			try {
				bitmap = BitmapFactory.decodeStream((InputStream) new URL(params[0]).getContent());
			} catch (MalformedURLException e) {
				e.printStackTrace();
			} catch (IOException e) {
				e.printStackTrace();
			}
			return bitmap;
		}

		@Override
		protected void onPostExecute(Bitmap result) {
			if(result != null){
				imageView.setImageBitmap(result);
			}
		}
	}
```

30. ábra ImageLoader

Új példány létrehozásakor szükséges a konstruktornak paraméterül megadni azt az ImageView-t, ahova majd a kép kerülni fog. Majd a getImage metódus meghívásakor a képfájl nevét és kiterjesztését kapja a metódus. Ahogy a Communicator osztály úgy az ImageLoader osztály is tartalmaz egy AsyncTaskot kiterjesztő privát osztályt.

Mivel a kép bájtfolyamként érkezik a szerverről, így a doInBackground metódus megnyitja azt a folyamot, olvassa, majd az onPostExecute metódusban beállítja az ImageViewnak a megfelelő képet, amelyet a bájtfolyamból koncentrált bitmappá.

Képek továbbítása szerver oldalon:

A mobil alkalmazástól a kérés az ImageController osztályhoz érkezik be elsőként. Akárcsak a többi, ez is egy Spring keretrendszerben definiált REST vezérlő (31. ábra).

```
@RestController
@RequestMapping("/ImageService")
public class ImageController {

	@RequestMapping(value="/getImage", method=RequestMethod.GET, headers="Accept=Application/json")
	public byte[] getImage(@RequestParam("image")String imageName){
		ImageManager imageManager = new ImageManager();
		return imageManager.getImageBytes(imageName);
	}
}
```

31. ábra ImageController

Mely a Frameworkben található ImageManager osztályhoz továbbítja a kérést. Itt az operációs rendszernek megfelelően kiválasztja az alkalmazás, hogy az előre elkészített képek hova vannak másolva. (32. ábra)

```
public class ImageManager {

	private static String LINUX_PATH = "/home/doctor/Pictures/thesis/";
	private static String WINDOWS_PATH = "C:\\Images\\";

	private String imagePath;

	public ImageManager(){
		if(System.getProperty("os.name").equals("Linux")){
			imagePath = LINUX_PATH;
		}
		else{
			imagePath = WINDOWS_PATH;
		}
	}

	public byte[] getImageBytes(String image){
		File file = new File(imagePath + image);
		byte[] bytes = null;
		try {
			FileInputStream fileInputStream = new FileInputStream(file);
			System.out.print("file was loaded into input stream");
			bytes = IoUtils.toByteArray(fileInputStream);
			System.out.print("inputStream was converted");
			fileInputStream.close();
			System.out.print("stream was closed");
		} catch (FileNotFoundException e) {
			e.printStackTrace();
		} catch (IOException e) {
			e.printStackTrace();
		}
		return bytes;
	}
```

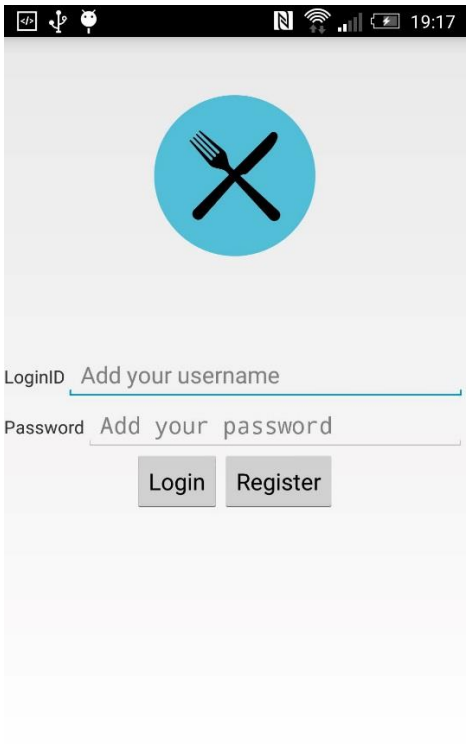
32. ábra ImageManager

Az elérési út és a kép neve segítségével megnyitjuk, a fájlrendszerből beolvassuk a bájtokat, majd zárjuk a bájtfolyamot. Ha a művelet sikeres volt, akkor a szerver alkalmazás a bájtokat továbbítja a hálózaton a mobil alkalmazás számára.

5. Felhasználói élmény

A felhasználói élmény eszméletlenül fontos a mobil alkalmazások, de manapság már webes alkalmazások illetve asztali alkalmazások esetében is, ezért különös figyelmet szenteltem az általam tervezett alkalmazás megjelenésére.

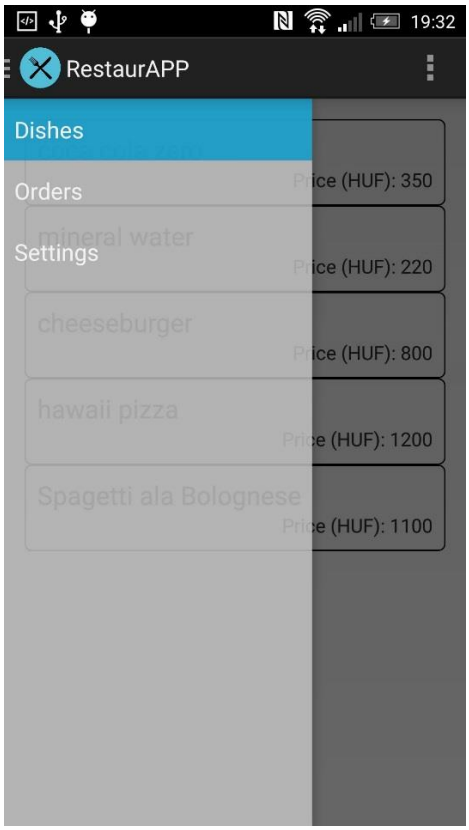
Legfőbb célom az volt, hogy a felhasználói felület kövesse a manapság népszerű flat dizájn, amely végtelenül egyszerű GUI elemekről vált híressé, valamint a felesleges effekteket és 3D elemeket is nélkülözi. Fontos az alkalmazás egyszerű megjelenése a manapság kínai piacon lévő telefonok gyengébb hardvere miatt is.



33. ábra: Bejelentkező képernyő

A bejelentkező képernyőn a felhasználó semmilyen funkciót nem érhet el, itt csak a bejelentkezési adataik megadására illetve új regisztrációra van lehetőségük. Ha a LoginID vagy a Password mezőkre kattintanak akkor az Android rendszer által nyújtott virtuális billentyűzet segítségével adhatják meg belépési adataikat, amely sikertelenségéről egy felugró ablak értesíti őket, valamint ha sikerült belépniük, akkor az alkalmazás védett része válik elérhetővé.

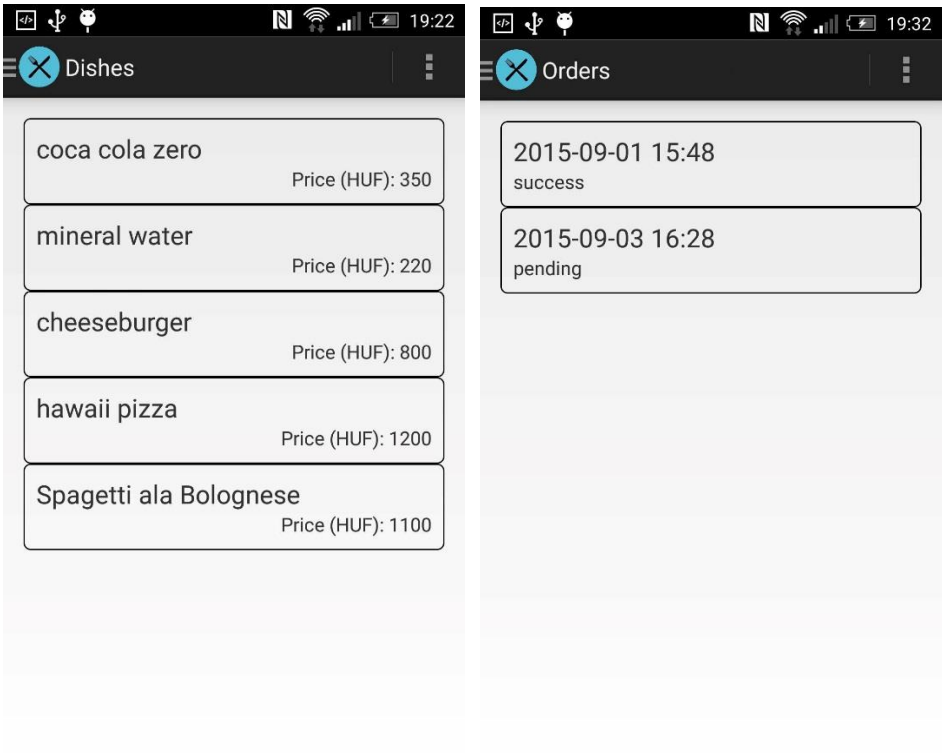
Az alkalmazás funkciói között a felhasználóknak az Android rendszerből, valamint a Google által készített alkalmazásokból megszokott módon egy a kijelző bal széléről induló jobb oldalra irányuló söprő mozdulattal képesek előhozni a navigációul szolgáló oldalsó menüt.



34. ábra: Oldalsó navigációs sáv

Amennyiben ebben a menüben kiválasztanak egy menüpontot a funkció rögtön betöltésre kerül az alkalmazáson belül és az oldalsó sáv automatikusan eltűnik. Amennyiben nem kívánnak navigálni másik funkcióra egy jobb oldalról bal oldalra irányuló söprő mozdulattal bezárhatják az oldalsó sávot. Valamint ez a sáv bezáródik a vissza gomb megnyomására is.

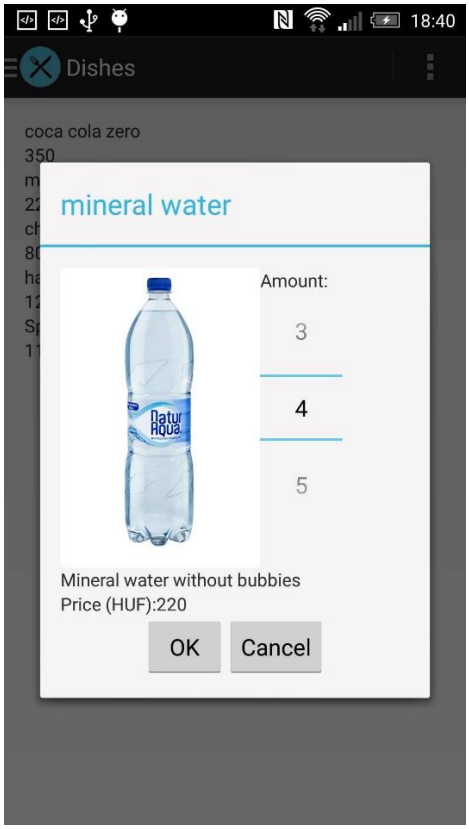
Az alkalmazás további fontos részeit képezik még a listák. Minden felhasználói adat listába szedve jelenik meg, ilyenek például az eddigi rendelések vagy a felvett szállítási címek is. Nem felhasználói adat, de listába szerveződve jelennek meg az ételek is, amelyeket a felhasználó a kosarába helyezhet a vásárlás során.



35. ábra: Listák

Ezekre a lista elemekre kattintva a felhasználó további információkat tudhat meg az elemről. Így tekintheti meg az eddigi rendeléseinek a leírását is vagy annak státuszát, de a rendelhető ételek közül is további információkat tudhat így meg. Utóbbihoz tartozó részletek nézeten a felhasználó láthat egy képet is a termékről valamint beállíthatja a mennyiséget, hogy mennyit szeretne abból a kosárba helyezni.

Miután megtörténtek a szükséges beállítások (mennyiség kiválasztása) az OK gomb megnyomásával a kosárba kerül az adott termék, míg a Cancel gomb megnyomására a részletek nézett eltűnik a felhasználó szemei elől.



36. ábra: Részletek megjelenítése

Úgy gondolom, hogy a felhasználói felületet sikerült kellőképpen egyszerűre megalkotnom, amely képes betölteni a funkcióit, de kellően egyszerű maradt ahhoz, hogy a gyengébb hardverrel rendelkező készülékeken is megfelelően rugalmasan futhasson.

Az alkalmazás használata során a felhasználók a már jól meg szokott gesztusokat használhatják és a jól megszokott felületi elemekkel találkoznak, így akár egy kezdő Android felhasználó is kiismeri magát a grafikus felületen és megtalálja azt a funkciót, amire éppen szüksége van.

6. Telepítés Android rendszerekre

Telepítés Androidra

Helyezzük a mellékelt lemezt a meghajtóba és nyissuk meg azt egy fájlkezelő alkalmazással (pl. Total Commander, Windows Intéző).

A megnyitást követően találunk egy *Restaurapp.apk* fajlt, amit Android készülékünkre másolva (ami legalább Android 4.3-t futtat) telepíthetünk. A telepítéshez szükséges a telefon beállításaiban az ismeretlen forrásokból származó alkalmazások telepítését engedélyezni. Ezután a program indítóikonja megjelenik a készülékre telepített alkalmazások listájában.

7. Összefoglalás

Napjainkban mindenki törekszik arra, hogy a lehető legkényelmesebben élje a mindennapjait. Életünk szerves részét képezik az okos telefonok amik gyakorlatilag mindig a kezünk ügyében vannak.

Kényelmes életet élve gyakran rendelünk ételt otthonra, ezzel mellőzve a főzést, és minden más munkát. Rengeteg étteremből rendelhetünk telefonon, és néhány komolyabb étteremnek weboldala is van, ahol szintén leadhatjuk a rendelésünket. Kicsit belemélyülve a témába észrevettem, hogy gyakorlatilag nem találtam mobil applikációt amivel a legegyszerűbben ételt tudnék rendelni. Ezért célul tűztem ki, hogy majd akkor én elkészítek egy alkalmazást erre a célra. Körülnéztem az okos telefon operációs rendszerek között, hogy melyik platformra lenne érdemes fejlesztenem, hogyan juthatna el a legtöbb felhasználóhoz az alkalmazásom.

Az irodalomkutatás során megismerkedtem az Android, iOS, és Windows Phone operációs rendszerekkel. Alaposan tanulmányoztam a különböző operációs rendszerek felépítését, előnyeit, hátrányait. A piackutatási adatokból kiderült, hogy legtöbben Android rendszerrel ellátott okos telefont használnak, ezért erre esett a választásom, mert így lehet a legnagyobb a célközönségem. Szakdolgozatom elkészítése során számos technológiával megismerkedtem és alkalmaztam is őket. Ahol csak tehettem törekedtem az ingyenes verziók használatára. Szükségem volt egy adatbázis kezelő alkalmazásra is melyek közül a MySQL-re esett a választásom, ami ingyenesen hozzáférhető és számomra teljesen megfelelő volt. Továbbá a fejlesztés során jó hasznát vettem a Spring Java EE keretrendszernek is. A forráskód megírását szintén az ingyenes Eclipse fejlesztőkörnyezetben végeztem el. Az Android alkalmazáshoz a Google által kiadott Android SDK-t használtam.

Összességében úgy érzem sikerült teljesítenem a célomat, hogy fejlesszek egy alkalmazás rendszert, mellyel az emberek/felhasználók életét tehetem könnyebbé, azzal, hogy ételrendelésüket így már mobil készülékről, natív alkalmazás segítségével is leadhatják. Sajnos az alkalmazást nem sikerült nagy közönség előtt letesztelni, mivel ennek rengeteg jogi feltétele lenne, amelyeket nem állt módomban megkötni a szakdolgozat keretei között.

8. Summary

Nowadays all the people wants to live their life on the easiest and most comfortable way as they can. As the result of this endeavour smart devices are the port of our everzdays.

We often order dishes using the internet to our home. We avoid preparing food in our kitchen and the afterworks just like cleaning in this way. We can order food using phone calls from several restaurants but also a lot of restaurant has an own website for orders. Making my researching I discovered that there is no native mobile application for this funcionality. That is why I made the decission to code my own application. The first step was to get to know mobile operation systems and choose when to develop my app.

I got know with Android, iOS and Windows Phone during my research. I studied the archutecture of different mobile operating systems. I choosed Android OS for my mobile application, because of I had discoverd during my market researches that most people uses this kind of mobile OS. My app can reach these people what is important for me. I got know and used several technologies during the writing of my thesis. I used several free software where I could. That is why I used MySQL as the database of my application what is free and fit for my needs. I also used the free Spring Java EE Framework for my thesis. I used to code my application in the free Eclipse IDE.

In total, I had reach my objects with my thesis. I developed an integrated system what can server the customers/users to order dishes using a native application on their mobile devices. I think this system makes their life more easier. Unfortunately I cannot test my application with several user on Google Play, because it needs more legal contract than I can solve within the limits of my thesis.

Irodalomjegyzék

[1]Kovács László: Adatbázisok tervezésének és kezelésének módszertana, Computerbooks kiadó, ISBN: 963618321X, 2004, p. 458

[2] Balogh Péter, Berényi Zsolt, Dévai István, Imre Gábor, Soós István, Tóthfalussy Balázs
Szoftverfejlesztés Java EE platformon ISBN 978-963-9131-97-2

[3]Ekler Péter – Fehér Marcell- Forstner Bertalan – Kelényi – Imre
Android alapú szoftverfejlesztés
ISBN:9789639863279

[4] Wikipedia Windows Phone cikke
https://en.wikipedia.org/wiki/Windows_Phone (2016.04.25)

[5] Szak.hu Android cikke
http://www.szak.hu/android-fejl/sample_chapters/chap1.pdf (2016.04.25)

[6] Spring keretrendszer
http://www.tutorialspoint.com/spring/spring_overview.htm (2016.04.25)

[7] Wikipedia MyBatis cikke
http://www.tutorialspoint.com/mybatis/mybatis_quick_guide.htm (2016.04.25)

[8] MyBatis
http://www.mybatis.org/mybatis-3/index.html (2016.04.25)

[9] Wikipedia MySQL cikke
https://en.wikipedia.org/wiki/MySQL (2016.04.25)

Ábrajegyzék

```
public class Connection {

	private static SqlSessionFactory sqlSessionFactory = null;

	private Connection(){
		Reader reader = null;
		try{
			reader = Resources.getResourceAsReader("database-config.xml");
		}
		catch(IOException e){
			e.printStackTrace();
		}
		sqlSessionFactory = new SqlSessionFactoryBuilder().build(reader);
	}

	public static SqlSessionFactory getInstance(){
		if(sqlSessionFactory == null){
			new Connection();
		}
		return sqlSessionFactory;
	}
}
```

17. ábra Connection

```
1  package hu.unimiskolc.iit.thesis.restaurapp.dao.mybatis.interfaces;
2
3  import java.util.ArrayList;
4
5  import hu.unimiskolc.iit.thesis.restaurapp.dao.objects.Userdata;
6
7  public interface IUserdata {
8
9  	public ArrayList<Userdata> selectAllUser();
10
11 	public Userdata selectUser(String username);
12
13 	public void addUser(Userdata userdata);
14
15 	public void updateUser(Userdata userdata);
16
17 	public void deleteUser(Userdata userdata);
18 }
```

19. ábra Userdata interface

```
public class AddingToDB {

	public static void addAddress(Address address){
		SqlSession sqlSession = Connection.getInstance().openSession();
		IAddress iAddress = sqlSession.getMapper(IAddress.class);
		iAddress.addAddress(address);
		sqlSession.commit();
		sqlSession.close();
	}

	public static void addUserdata(Userdata userdata){
		SqlSession sqlSession = Connection.getInstance().openSession();
		IUserdata iUserdata = sqlSession.getMapper(IUserdata.class);
		iUserdata.addUser(userdata);
		sqlSession.commit();
		sqlSession.close();
	}

	public static void addCustomerOrder(CustomerOrder customerOrder){
		SqlSession sqlSession = Connection.getInstance().openSession();
		ICustomerOrder iCustomerOrder = sqlSession.getMapper(ICustomerOrder.class);
		iCustomerOrder.addOrder(customerOrder);
		sqlSession.commit();
		sqlSession.close();
	}

	public void addFood(Food food){
		SqlSession sqlSession = Connection.getInstance().openSession();
		IFood iFood = sqlSession.getMapper(IFood.class);
		iFood.addFood(food);
		sqlSession.commit();
		sqlSession.close();
	}
```

21. ábra AddingToDB osztály

```
<web-app id="WebApp_ID" version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">

    <display-name>Restaurapp MobileGateway</display-name>

    <servlet>
                <servlet-name>rest</servlet-name>
                <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
                <load-on-startup>1</load-on-startup>
        </servlet>

        <servlet-mapping>
                <servlet-name>rest</servlet-name>
                <url-pattern>/*</url-pattern>
        </servlet-mapping>
</web-app>
```

23. ábra web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans" xmlns:context="http://www.springframework.org/schema/context"
           xmlns:mvc="http://www.springframework.org/schema/mvc" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:p="http://www.springframework.org/schema/p"
       xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-4.0.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context-4.0.xsd
        http://www.springframework.org/schema/mvc
        http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd">
    <context:component-scan base-package="hu.unimiskolc.iit.thesis.restaurapp.mobilegateway.communication.rest.controllers" />
    <mvc:annotation-driven />
</beans>
```

24. ábra rest-servlet.xml

```
public Communicator(Context context, String service, String parameters){
        this.service = service;
        this.parameters = parameters;
        this.context = context;
        Logger.getGlobal().log(Level.WARNING, "Communicator instance was created");
}

public void callService(){
        Logger.getGlobal().log(Level.WARNING, "Communication started.");
        String address = context.getString(R.string.hu_unimiskolc_iit_thesis_restaurapp_framework_network_connection_url);
        new CommunicatorAsyncTask().execute(address + service + parameters);
}
```

26. ábra Communicator osztály

```
    private void callBack(String result){
            Logger.getGlobal().log(Level.WARNING, "Called back to main thread.");
            try {
                    if(service.equals("UserService/register?regData=")){
                            JSONObject jsonObject = new JSONObject(result);
                            Logger.getGlobal().log(Level.WARNING, "Register process ended.");
                            Status status = new Status(jsonObject.getBoolean("success"), jsonObject.getBoolean("fail"));
                            if(!status.getSuccess() && status.getFail()){
                                    new NetworkWarningDialog(context).show();
                            }
                    }
                    if(service.equals("UserService/login?loginData=")){
                            JSONObject jsonObject = new JSONObject(result);
                            Logger.getGlobal().log(Level.WARNING, "User has been logged in.");
                            Status status = new Status(jsonObject.getBoolean("success"), jsonObject.getBoolean("fail"));
                            if(status.getSuccess() && !status.getFail()){
                                    Logger.getGlobal().log(Level.WARNING, "Navigating to MainActivity.");
                                    ViewLoader.loadView(context, MainActivity.class);
                            }
                            else{
                                    Shared.getInstance(context).deleteString("LOGGED_IN_USER_ID");
                                    new NetworkWarningDialog(context).show();
                            }
                    }
```

28. ábra Communicator osztály callBack metódusa

```java
package hu.unimiskolc.iit.thesis.restaurapp.framework;

import android.content.Context;
import android.content.Intent;

public class ViewLoader {

	public static void loadView(Context context, Class<?> clazz){
		Intent intent = new Intent(context, clazz);
		context.startActivity(intent);
	}

	public static void loadView(Context context, Class<?> clazz, String[] extras){
		Intent intent = new Intent(context, clazz);
		intent.putExtra("extras", extras);
		context.startActivity(intent);
	}

	public static void exit(Context context){
		Intent intent = new Intent(Intent.ACTION_MAIN);
		intent.addCategory(Intent.CATEGORY_HOME);
		intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
		context.startActivity(intent);
	}
}
```

29. ábra ViewLoader