DIPLOMAMUNKA

Koncz Tamás

Debrecen

2015

Debreceni Egyetem
Informatikai Kar

IoT jellegű, RFID alapú leltározás SQL szerverre

Témavezető:

.....
Dr. Buchman Attila
Egyetemi adjunktus

Szerző:

.....
Koncz Tamás
Programtervező informatikus
MSc

Debrecen
2015

Tartalomjegyzék

1. Bevezetés

1.1. Célkitűzés és témaválasztás indoklása

2014. nyarán egy kis debreceni cégnél csodálatos dolgokat alkottunk egy pár fős gyakornoki gárdával. Ez a csapat fokozta az érdeklődésemet a mikrokontrollerek iránt, ami eddig is számottevő volt, tekintve az előző PIC16F628-as mikrokontrolleres szakdolgozatomat (7). A cégnél egy sokkal közismertebb, egyszerűbben programozható mikrokontroller családdal foglalkoztunk, az Arduinoval. Az a kis kékség, ami nem olyan gyors, memóriája is apró-cseprő, néha megtelik és észre se vesszük, mert a saját kódjainkban keressük a megfagyás okát. Az Arduino az nem más, mint chip a körülötte lévő érrendszerrel együtt, mint ahogyan a PC alaplapja a processzorral együtt. Az Arduino UNO chipje maga az Atmel ATmega 328-as, ami lefuttatja a FLASH-be betöltött algoritmust.

A 2014-es gyakornoki gárdával egy automatizált permetező rendszert szerettünk volna létrehozni, amit egy traktorra felszerelve üzemeltethet az ügyfél. Az ügyfélnek egy Androidos okostelefon kellett a vezérléséhez. Tehát a feladat 2 részre osztódott az Androidosra és az Arduinosra. A kettő között bluetooth tartotta a kapcsolatot. Az ügyfél láthatta okostelefonján a traktor sebességét, és a permetező szivattyújának aktuális erejét.

Ez adta az ihletet a diplomamunka feladatomhoz, ami folyamatban van. Az alapvető elgondolás az volt, hogy keresek egy másik felhasználást, ahol egy Arduinot és egy Androidot köthetek össze. Egyik este édesanyám előrukkolt egy ötlettel, hogy mi lenne, ha automatikusan rendelné egy rendszer a hűtőszekrényből hiányzó ételt vagy italt? Én meg erre alapozva létrehoztam egy 3 oldalas látványtervet egy esti ábrándozásom során, amelynek legfontosabb elemeit az utolsó fejezetben felvázolom. Így látható lesz a teljes projekt komplexitása és az, hogy ezt egyedül lehetetlenség létrehozni, még egy élet munkája is kevés. A későbbi fejezetekben kielemzem a már megvalósított részeket.

1.2. A megvalósítás módszere

A feladat egyszerűnek hangzik. Kielemezzük mi található a hűtőszekrényben. Eltároljuk egy SQL adatbázisban, amely adatbázist figyel egy program és leadja a rendeléseket. Ebből az valósult meg, hogy van egy SQL adatbázisom, amit Arduinorol próbálok feltölteni, és bízok benne, hogy mire a bizottság elé kerül a projekt fel is tudom tölteni az adatbázist. Az Arduino jelenleg képes RFID biléták beolvasására. Mindegyik biléta 1 terméket jelöl. Az Arduino kapcsolatba lép az adatbázissal, de már a jelszavas bejelentkezéssel gondok adódtak. A

témavezető fel is világosított, hogy lehetetlenség különböző adatbázis műveletek végezni Arduinon, hiszen túl kevés a benne lévő SRAM, amit a futása során használ. Ezt én is tanúsíthatom, miután sikerült működés közben lefagyasztanom többször az SRAM telítődése miatt. Így egy köztes JAVA programot használok majd az Arduino és az SQL adatbázis között, amit a tervem szerint az SQL adatbázis szerverén futtatok majd. Így az Arduino csupán a biléta sorszámát küldi el a szerveren futó programomnak, ami már tudni fogja melyik termékre vonatkozik a sorszám. Ha már szerepel az adatbázisban leveszi azt, ha még nem szerepel benne, akkor hozzáadja.

Ami mondhatni 100%-osan megvalósításra került, az nem más, mint az Android applikáció, ami egy táblagépen vagy okostelefonon kijelzi az adatbázis tartalmát. Roppant egyszerűen lehet benne válogatni a hűtőszekrények között. (A prototípusom 1 hűtőhöz való, de a rendszer több hűtőre is alkalmazható.)

Édesanyám kezdeti ötletétől még ez fényévek távolságában áll. A rendelések leadásához még szerződésre is szükség van a futárszolgálatokkal és a boltokkal. Továbbá a rendszer nem csak hűtőszekrényekhez jó, hanem minden raktárhelyiséghez megfelel, ahol a raktározott termékekre rá lehet akasztani RFID bilétákat. Jelenleg az iparban a raklapokat bilétázzák RFID-vel, hiszen 20-30 Ft-ból jön ki egy chip, de a tejesüvegre már nem éri meg rárakni, ha egyszer a vonalkód olcsóbb.

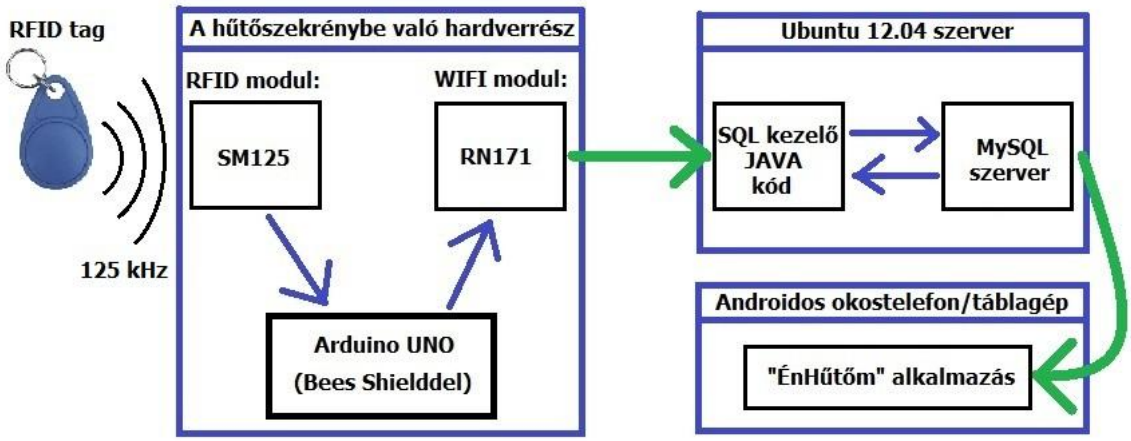
Felvetettem az ultrahangos ételfelismerést is. Ez úgy működött volna, hogy, ha valamit az ultrahang elé tolunk, akkor a mért távolság alapján megállapította volna, hogy ott van valami. Ezt is egyszerű lett volna leprogramozni, mint az RFID alapú felismerést, csak arra is gondoltunk, hogy a felhasználó nem fog mindig mindent ugyanoda rakni. Szóval maradtunk az RFID-nál.

A címben lévő IoT (Internet of Things) jelleg meg egy 1999-es fogalom (9). Azon dolgok tartoznak bele, amelyek kevés emberi beavatkozással használják az internetet. Olyan apróságokra kell gondolni, amiket észre se vesz az ember, hogy a netre csatlakoznak. Pl.: Egy tubus fogkrém, ami önmagát újrarendeli. Én is arra törekszek, hogy a felhasználó észre se vegye, hogy dokumentálja a hűtő adatbázisát.

1.3. Felhasznált eszközök

Az én megoldásomban 5 db átprogramozhatatlan RFID chip állt a rendelkezésemre. Az Arduinot Microsoft Visual Studioval programoztam be C++-t használva. Feltelepítettem a Visual Studiohoz egy Arduino programozáshoz való Visual Micro bővítményt. Az Arduino

UNO-ra (19) rácsatlakoztattam egy RN171-es wifi (26) egyéget és egy SM125 RFID (21) olvasót. A google-től béreltem egy apró Linux szervert, amin az SQL adatbázis fut. Továbbá arra tervezem rátelepíteni a JAVA applikációt, ami összeköti majd az Arduinot és az SQL adatbázist. Az egyetemtől kaptam kölcsönbe egy táblagépet is az előző eszközökön kívül. A táblagépen az Android applikációm fut ("ÉnHűtőm"). Az Androidos applikációt JAVA-ban programoztam az Android Studiot használva.



1. ábra - Az adatáramlás folyamata

2. Tőlem független megvalósítások

A megvalósítandó ötlet nem egyedi, mások is próbáltak okoshűtőket készíteni kisebb-nagyobb sikerrel. Úgy érzem az LG és a Samsung közelítette meg valamelyest a szemléletet, melynek lényege, hogy az ügyfélnek ne legyen semmi dolga. A wikipédiában az okoshűtő fogalma alatt ezt találjuk. (8) Lefordítottam magyarra: "Egy hűtő, ami arra lett programozva, hogy észrevegye, milyen termékeket tárolnak benne, és vonalkód vagy RFID chipek beolvasásával frissíti a benne lévő termékek listáját."

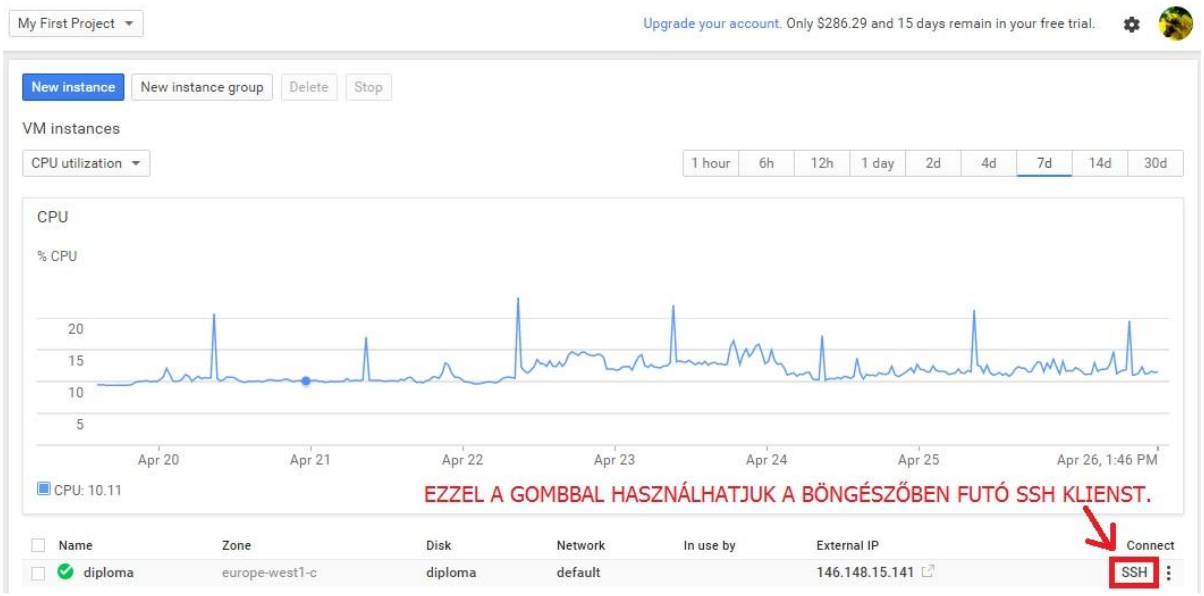
Tehát a termékfelismerés közel sem felhasználóbarát. Kevesen fizetnek 100ezreket egy hűtőszekrényért, hogy a végén minden kivételnél és berakásnál vonalkódokat vagy RFID-ket olvastassanak be vele. Ez plusz fáradtság és munka. A legjobb az lenne, ha a felhasználónak nem is kellene foglalkoznia az ételek, italok elhúzásával az érzékelők előtt. Ha meg google-be beírjuk: "hűtőbe beépíthető ételfelismerő", akkor sorozatban dobja fel a találatokat a beépített/beépíthető hűtőszekrényekről. Tehát nem találtam a piacon olyan rendszert, ami bármely hűtőbe beszerelhető, és az átlagember igényeit kielégíti. Pedig sok körülmény kecsegtető. A 0 °C körüli hőmérsékletben élvezik a legjobban az akkumulátorok magukat, és még a wifi is lazán be tud hatolni a hűtő belsejébe. A hűtő ajtaja vasból van, tehát egy mágneses tabletet rápattintva lehetne vezérelni. Vásárlásnál meg csak lepattintjuk és visszük magunkkal. Szerintem a felhasználóbarát megoldás egy videó alapú képfelismerő lehetne, amihez persze lassú lenne az Arduino, így más technológiákhoz kéne folyamodni.

Ha a termék oldaláról nézzük a dolgokat, akkor RFID esetén mindig rá kellene akasztani a bilétát vagy ráragasztani egy apró RFID chipet. A vonalkód esetén pedig nem mindenen van vonalkód. És amin van vonalkód és kibontjuk, az hamarabb megromlik, mint ami zárva áll a hűtőben. Ezt azért emeltem ki, mert a legtöbb ilyen rendszer odafigyel a romlandó tartalomra. A vonalkód pozitívuma, hogy az első fele a gyártó száma (cégprefix), és a második fele a termék cikkszáma. Így nem kell manuálisan beprogramozni a chipeket vagy a chipekhez tartozó jelentéstartalmat.

3. A szerver kialakítása
3.1. A szervergép beüzemelése

Első lépésként egy szervert kerestem, ahova rátelepíthettem a MySQL-t. Szerverre azért volt szükségem, hogy akkor is elérhető legyen az adatbázis, ha a gépem ki van kapcsolva. A második indok, hogy gépemen Windows fut és mindenki, aki egy komoly szervert szeretne, Linux alapú operációs rendszeren futtatja azt.

Az interneten keresgélve megtudtam, hogy a Google Developers Console segítségével egyszerűen nyithatok egy szervert. Beállítható volt a processzor sebessége, a memória mérete és a winchester tárterülete is. Kiválasztottam rá a telepítendő operációs rendszert (Ubuntu 12.04.5) és azt, hogy a bolygó melyik részén helyezkedjen el. Nyugat-Európára esett a döntésem, mert az volt a legközelebbi szerverpark Magyarországhoz. Megnyomtam az "OK"-ét és 5 percen belül létrejött a szerver. A google biztosít egy 60 napos, ingyenes szerver bemutatót, ahol 300 dollár költőpénzt kap a kísérletező, de mivel nem kértem komoly szervert, ezért egy mozijegy árából kitelne nekem havonta. SSH-án keresztül be lehet tekinteni a szerverre, ami Linux terminálon keresztül fogad el parancsokat. Semmilyen grafikus felülete nincsen. Az SSH, böngészőben futó kliensüket használtam. Ez a szerverem, aminek a "diploma" nevet adtam:



2. ábra - A szerverem adatai (10)

3.2. A MySQL adatbázis létrehozása a szerveren

Ahhoz, hogy egy külső program rácsatlakozzon az interneten keresztül erre a szerverre, le kell kapcsolni a tűzfalakat mind a programnak a szerver irányába, mind a szervernek a program irányába. Továbbá több adatra van szüksége a programnak a szerverre csatlakozáshoz. Ilyen adatok a szerver IP címe és az adatbázis kezelő port száma, amin keresztül a MySQL adatbázis kezelő kommunikál a külvilág felé. A fenti képről kiderül a szerver IP címe, ami 146.148.15.141. A MySQL port száma meg alapértelmezetten a 3306-os port, de fogok rá mutatni parancsot is, ami megmutatja. Szükséges még a csatlakozáshoz a MySQL felhasználónév, jelszó és az adatbázis neve, amiben van a táblázatunk.

Az első parancs, amit beírtam a szervernek, hogy telepítse a MySQL-t: "sudo apt-get install mysql-server" Aki nem jártas Linuxban annak jó tudnia, hogy a "sudo" előtag arra jó, hogy root felhasználóként közöljük a géppel a parancsot. A root felhasználónak vannak különleges jogai, adminnak is szokták nevezni. Az "apt-get install" parancs az operációs rendszer tárhelyéből installálja a kiválasztott programot. Telepítés után ezzel a paranccsal lehet leellenőrizni a verziószámot: "mysqladmin -u root -p version" A verziószáma az általam telepített MySQL-nek 5.5.45-ös. A "-u root -p" jelzi, hogy root-ként akarunk bejelentkezni és meg szeretnénk adni a jelszót. (-u, mint user és -p, mint password) A "version" pedig maga a parancs, ami a verziószám lekérése. A verziószámra azért volt szükségem, hogy megfelelő JAVA osztályokat keressek hozzá, amikkel fel tudok lépni a szerverre.

A MySQL szoftverének vezérlése:

- indítása: "sudo service mysql start"
- leállítása: "sudo service mysql stop"
- újraindítása: "sudo service mysql restart"

Miután elindítottam a MySQL alkalmazást, leellenőriztem az általa használt port számát, hogy valóban 3306-os. Ezt a "sudo netstat -tlnp" paranccsal tehetjük meg, ami felsorolja azokat a portokat és a hozzájuk tartozó alkalmazásokat, amik az internet irányába hallgatnak. A "sudo netstat -tlnp | grep mysql" parancsra ezt a választ kaptam, látszik, hogy minden irányba hallgatózik:

"tcp 0 0 0.0.0.0:3306 0.0.0.0:* LISTEN 16966/mysqld"

A "grep mysql" kiválasztja a válaszból azokat a sorokat, amelyekben benne van a "mysql" szó. A 0.0.0.0:* mutatja, hogy az összes számítógép összes portjáról elérhető a MySQL.

Eredetileg nem volt elérhető. Át kellett engednem mindenkit a tűzfalon. Ahhoz, hogy mindenkit átengedjek 3 lépést kellett megtegyek:

1. Az etc/mysql/ könyvtárban lévő my.cnf szövegfájlban át kellett írjak egy sort erre: "bind-address = 0.0.0.0" Ami megadja, hogy a MySQL minden irányba hallgasson. Ezt a Linux VI szövegszerkesztő programjával tettem meg.
2. Át kellett engedjek mindenkit befelé a Linux tűzfalán ezzel a paranccsal: "sudo iptables -A INPUT -p tcp --dport 3306 -j ACCEPT". Ezzel a fajta paranccsal az IP táblázatot lehet konfigurálni. A "-A" hozzáad az IP táblához. ("-A", mint "append", ami magyarul hozzácsatol.) Módszereket az IP táblázatok szerkesztésére itt találtam: (11)
3. Át kellett engedjem a google tűzfalán a Google Developers Console webes kezelőfelületén keresztül. A webes felületen beállítottam egy tűzfalszabályt, ami mindenkit átenged TCP protokollal a 3306-os portra. Ez ugyanaz a tűzfalszabály, amit a 2-es lépésben használtam, csak az a Linux terminálon belül volt.

Ezekből a leírásokból sajátítottam el a MySQL használatát Linux alatt: (12) és (13). Régebben is használtam SQL-t, több tantárgyunknak volt már főtémája. A Linux alatti SQL parancsok megegyeznek az Oracle rendszereiben használt parancsokkal. Szóval miután fut a MySQL, és beléptünk a shelljébe, a vezérlése ugyanúgy működik. A keresés a jól ismert "SELECT" utasítással történik. Míg a sorbeillesztés a szokásos "INSERT INTO" utasítással. Törléshez pedig a "DELETE" utasítást használjuk. Csak a programba belépés módja különbözik a grafikus megoldástól. A terminálban a "mysql -u root -p" paranccsal lépünk be. A "-p" hatására elkéri a jelszavunkat, és a helyes beírása után belépünk a MySQL shelljébe. A shell nem más, mint a belső felület, amin keresztül vezérelhetjük a programot. Az előzőleg megemlített parancshármast: "SELECT"-et, "INSERT INTO"-t és "DELETE"-et, az adatbázist vezérlő programok JAVA kódjaiban fogjuk hasznosítani.

3.3. Az adatbázis és felhasználói jogosultságok létrehozása

A hűtőszekrényes alkalmazáshoz létre kellett hozzak egy adatbázist 1 db táblával. A "CREATE DATABASE firdge_network;" parancs létre is hozta a "fridge_network" nevű adatbázist. Ebben az adatbázisban létrehoztam egy táblát a "CREATE TABLE" paranccsal. Egy tábla bőven elég a feladat megvalósításához. Ez a tábla úgy lett tervezve, hogy több hűtő tartalmát eltárolja. A tábla egy sora az egy terméket modellez egy adott hűtőben.

A táblázat neve "content" és ezen attribútumokat tartalmazza:

- **ID:** Ez lesz a tábla kulcsa. Ez azonosítja a terméket. Típusa UNSIGNED BIGINT.
- **Fridge_ID:** Ez azonosítja a hűtőket. Ez mutatja meg, hogy a termék melyik hűtőben található. Típusa UNSIGNED INT.
- **brand:** Ez lesz a márka azonosítószáma. Típusa UNSIGNED INT. Pár használt jelölés a megoldásomban:

  1 = "Nagyné és fiai" 2 = "Alföldi tej" 3 = "Gyulai"

  4 = "Győri édes" 5 = "Nádudvari"
- **type:** Ez lesz a termék típusának az azonosítója. Típusa UNSIGNED INT. Pár használt jelölés megoldásomban:

  1 = "tej 2 = "vaj" 3 = "tejföl"

  4 = "sajt" 5 = "paprika" 6 = "tojás"

  7 = "uborka" 8 = "citrom" 9 = "kolbász"

  10 = "szalámi" 11 = "túró"

Egyik barátomat zavarta a tudat, hogy a google mostantól már azt is tudni fogja, hogy mi van a hűtőjében, ezért közöltem vele, hogy a google csak számokat fog látni a szervereken. A számok használatának a másik oka, hogy így gyorsabb a szerverrel a kommunikáció. A kapott számok alapján az Androidos applikáció a felhasználó számára érthetően kiírja a termékeket.

- **mass:** Ez lesz a termék tömege kg-ban. Típusa UNSIGNED DOUBLE.
- **expires:** Ez lesz a termék lejárati dátuma. Típusa DATE.

Miután készen lett az adatbázisom, létrehoztam egy felhasználót, és jogosultságokat adtam neki. Erre azért volt szükség, hogy a használatára programozott JAVA programokkal be tudjak lépni a felhasználónevet és jelszót használva.

2 felhasználóra lesz szükségem. Az egyikkel az Android alkalmazás csak olvasni fog az adatbázisból, ezért neki csak olvasási jogokat adok. Felhasználót a következő paranccsal hoztam létre: "CREATE USER koncztom_tablet IDENTIFIED BY 'jelszó';"

Miután létrejött, ezzel a paranccsal adtam hozzá jogosultságokat, hogy hozzáférhessen a "fridge_network" nevű adatbázishoz és azon belül a content táblához:
"GRANT SELECT ON fridge_network.content TO koncztom_tablet;"
A "GRANT SELECT ON" parancs kizárólag a "SELECT" parancs használatát engedélyezi.
A második felhasználó a majd jövőben elkészülő JAVA programhoz szükséges, ami egy hidat képez az Arduino UNO és a MySQL között. A második felhasználót úgy hozzuk létre, mint az elsőt, csak más felhasználónevet és jelszót adunk neki:

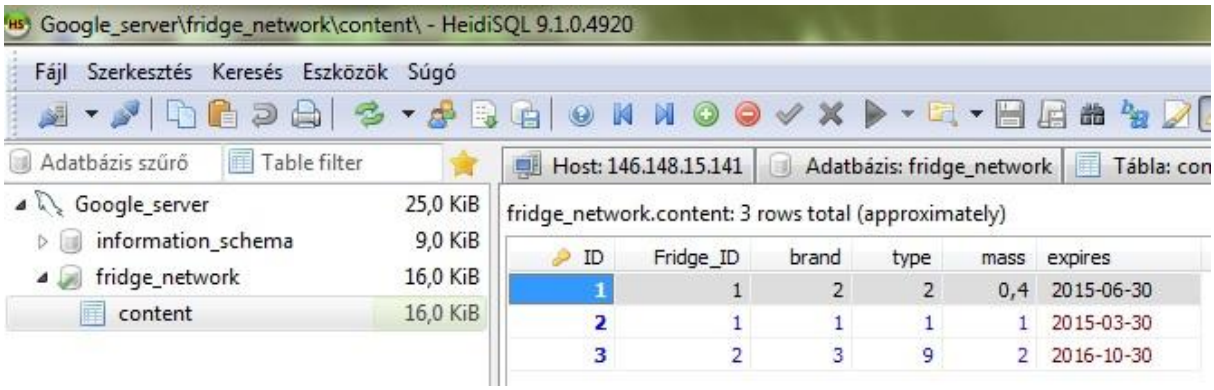
"CREATE USER koncztom_UNO IDENTIFIED BY 'jelszó';"
A koncztom_UNO nevű felhasználó megkapja a 3 számára legfontosabb táblázatkezelő jogot:
"GRANT SELECT, INSERT, DELETE ON fridge_network.content TO koncztom_UNO;"
Végül a "FLUSH PRIVILEGES" paranccsal véglegesíthetjük a hozzáférési jogosultságokat. A jogosultságok kezelésének elsajátításához ajánlom a (14)-os irodalomjegyzéket.

Érdekesség, hogy egy felhasználónévvel egyszerre több eszközről lehetünk bejelentkezve.

Mielőtt áttértem az Androidos alkalmazás fejlesztésére, beraktam pár értéket az elkészült "content" táblázatba. Az "INSERT INTO" paranccsal feltöltöttem pár értéket a táblázatba. Így most van 2 hűtőszekrényem: az 1-es hűtő és a 2-es hűtő, és mindkettőben vannak példatermékek.

A MySQL shell-jéből az "exit"-tel tudunk kilépni. A Google Developers Console böngészőben futó SSH klienséből szintén az "exit" beírásával tudunk kilépni.

Ezután letöltöttem egy ingyenes, kis helyet igénylő programot a PC-émre, amelynek segítségével beléphetünk bármelyik SQL adatbázisba, ha tudjuk az IP címet, portot, felhasználói nevet és jelszót. A program neve HeidiSQL (15). Íme a content táblázat az általam megadott példa értékekkel:



3. ábra - A "content" táblázat tartalma a HeidiSQL-lel (15) megjelenítve

Sajnos semmi se ment olyan simán, mint ahogyan leírtam, de bízok benne, hogy aki elolvassa, a leírások alapján létre tud hozni egy hasonló konfigurációt. Főleg a tűzfalak kiiktatásával volt problémám, mert nem gondoltam volna, hogy a Google Developers Console-nak is van saját tűzfala. Elakadásom miatt bementem a cégemhez tanácsokért. Arra is a főnököm világított rá, hogy a Linuxnak van saját tűzfala, amit le kell kapcsolni.

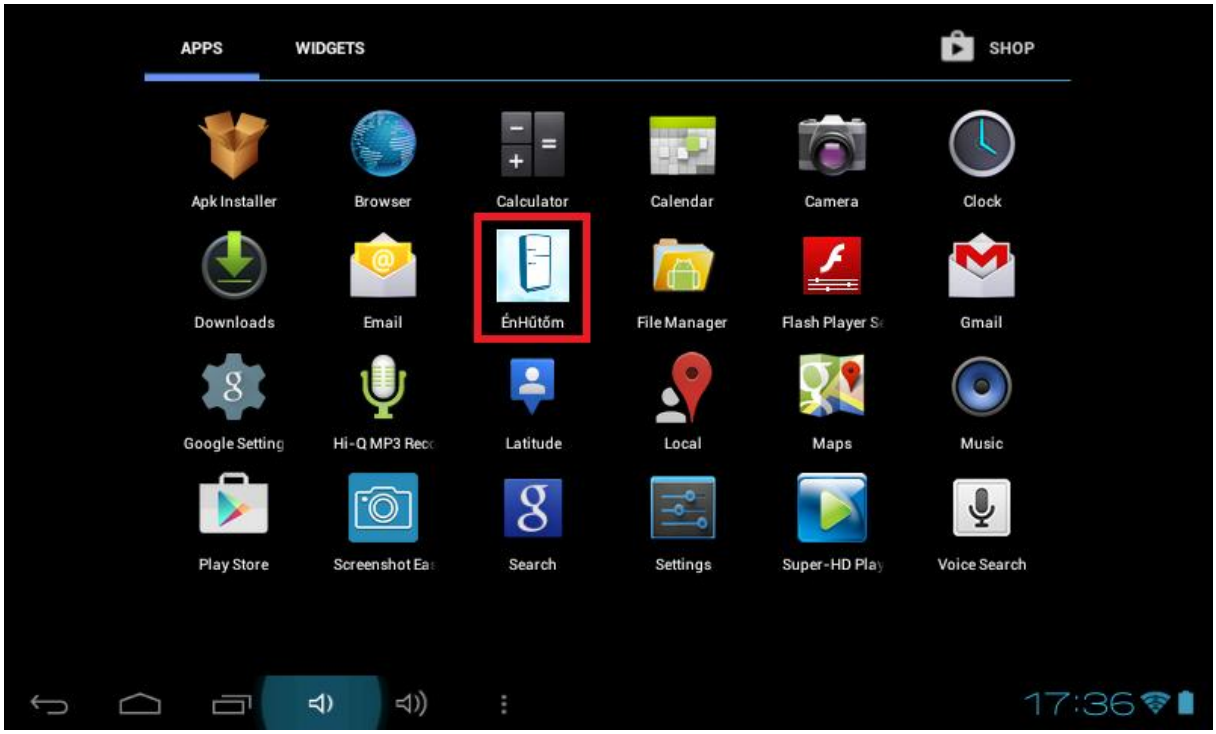
4. "ÉnHűtőm" Androidos alkalmazás

Ez az applikáció teszi emberközelivé az egészet. Ez a program mutatja meg az ember okostelefonján, tabletjén a hűtőjének a tartalmát. Viszont az alkalmazáson keresztül a tartalmat módosítani nem lehet. Érdekes kihívás volt, mert ez volt az első program, amit Androidra fejlesztettem. Ez a fejezet 2 alfejezetre bomlik. Az első alfejezetben bemutatom a program használatát egy rövid felhasználói kézikönyv által, a második alfejezetben betekinthetnek a kulisszák mögé, hogy megtudják hogyan programoztam.

A termék jelenleg nem letölthető Android boltokból, mert még van mit fejleszteni rajta. Az "ÉnHűtőm" applikáció jelenleg csupán a saját szórakoztatásomat, személyes fejlődésemet szolgálja.

4.1. "ÉnHűtőm" felhasználói kézikönyv

Töltse le számítógépére az "ÉnHűtőm" Android applikációt a diplomamunka CD mellékletéről vagy a Debreceni Egyetem könyvtárából. Másolja fel az okostelefonjára egy hozzá tartozó USB kábel segítségével. Indítsa el az okostelefonon az újonnan megjelenő kis kék hűtő ikon segítségével:



4. ábra - Pár megjelenített Android applikáció, köztük az "ÉnHűtőm"

Egy fehér töltőképernyőt követően megjelenik a hűtőválasztó képernyő. A menüt mindig a program jobb felső sarkában találja. A menüből lehet kilépni a programból vagy a jelenleg kiválasztott hűtőazonosítót elmenteni. A hűtőazonosító elmentése esetén a program újraindítása után nem szükséges a hűtő kiválasztása. Amennyiben az ön hűtőazonosítója különbözik az alapértelmezettként megadott hűtőazonosítótól (feltéve, ha felkínál alapértelmezettet), akkor adja meg az ön hűtőazonosítóját (1. lépés), majd mentse el a menüből (2. lépés), és a végén férjen hozzá a hűtő tartalmához az "OK" gombbal (3. lépés).

5. ábra - A belépés egy lehetséges folyamata hűtőazonosítójával

Ha sikertelen a belépés, akkor kiírja, hogy miért nem tudott belépni. Ennek főbb okai:

- Hibás adat kerül be a mezőbe az 1. lépésnél.
- A keresett hűtő üres vagy nem létezik.
- Nincs internetre csatlakoztatva az okostelefon.

Amennyiben sikeres a belépés, akkor megjelenik a hűtő tartalma. 4 oszlop jelenik meg előttünk. Az első oszlopban az étel vagy ital típusa jelenik meg. Például tej, vaj, túró, rostoslé stb. A második oszlopban a termék tömege jelenik meg kg-amban. Ez a vásárláskor hozzárendelt tömeg, és a fogyasztás során nem csökken. Viszont, ha a felismerő technológia elég jó lenne, akkor akár az étel fogyását is lehetne rögzíteni. A harmadik oszlopban a termék lejárati dátuma jelenik meg év-hónap-nap formátumban. Az utolsó, negyedik oszlopban a termék márkája jelenik meg. Most nézzünk egy példát a hűtő tartalmát mutató képernyőre:

6. ábra - Egy példa az 1-es hűtő tartalmára

Az ablak fejlécében megjelenik az adott hűtő azonosítója, ami jelen esettben az 1-es hűtő. Ez a képernyő nem frissül folyamatosan, tehát, ha a felhasználó frissíteni szeretné vissza kell lépnie, és be kell lépnie az "ok" gombbal újból.

Bizonyára feltűnt, hogy a képek fektetett képernyőjű tableten készültek. Az alkalmazást nem lehet csak fektetett képernyőn használni, mert állított képernyőn a táblázat átláthatatlanná válik.

4.2. "ÉnHűtőm" Androidos alkalmazás készítése

A munkához szükségem volt egy tabletre vagy okostelefonra, ami nekem nem volt, ezért az egyetem biztosított egy vonino Luna 70C tabletet, amiben van beépített wifi és rátelepítve egy Android 4.0.4 operációs rendszer. Az Android operációs rendszer mellett döntöttem, mert a mobileszközök 46,87%-a Androidot használ, 42,61%-a meg iOS-t (16). A Windows Phone a maga 2,66%-os arányával meg kiszorult a piacról.

A tabletet rácsatlakoztattam az otthoni wifimre, beállítottam, hogy ne forogjon a képernyője, amikor forgatom, mert zavart és a táblázat csak fektetett képernyőn fér el. Beállítottam, hogy amennyiben be van dugva a számítógépbe ne kapcsoljon be a képernyővédője, hogy könnyedén tudjam tesztelni a módosított programkódokat. A

számítógépem kezdetben nem ismerte fel, így fel kellett telepítenem rá a vonino Luna 70C-hez a drivert.

4.2.1. A MySQL szerverre csatlakozás JAVA kóddal

Mielőtt belefogtam volna életem első Android programjába, kipróbáltam, hogy rátudok-e kapcsolódni JAVA-val a MySQL szerverre, így telepítettem a legfrissebb JAVA-t (JDK 8-at) és az eclipse LUNA nevű fejlesztői környzetet. A MySQL 5.5-höz való csatlakozáshoz egy JDBC driver szükséges. Nem mindegyik JDBC driver működött, így a folyamat rengeteg próbálgatást igényelt, ami nekem bevált az az 5.0.8-as verzió (17), amelyhez tartozó jar file neve: "mysql-connector-java-5.0.8-bin.jar". A JAVA-hoz folyamatosan készülnek új osztályok és a kiterjesztésük jar. A JAVA objektumorientált nyelv, ezért szerencsére könnyen bővíthető, és minden eszközön működnek, amelyre feltelepül a JRE vagy JDK. A MySQL 5.5-höz valós csatlakozáshoz bőven elég a referencia könyvtárak közé berakni a "mysql-connector-java-5.0.8-bin.jar"-t. Ez volt a legjobb leírás, amit a MySQL adatbázishoz csatlakozáshoz találtam: (18) Hogy a csatlakozás helyes legyen fontos a csatlakozási paraméterek fontos megadása:

```
String url = "jdbc:mysql://146.148.15.141:3306/";
String dbName = "fridge_network";
String userName = "koncztom_tablet";
String password = "**********";
```

Az is nagyon fontos, hogy a csatlakozás és az adatok kiolvasása egy try és egy catch utasítás közé kerüljön. A JAVA-ban try után kerülnek azok a műveletek, amelyek gyakran nem hajtódnak végre, mert hibát dobnak ki valami oknál fogva. Ha a try utáni {} zárójelek közt valamelyik műveletnél hiba keletkezik akkor a catch utáni {} zárójelek közti műveletek kerülnek végrehajtásra. Példának okáért így csatlakozok fel a Linux szerveremre és olvasom le az 1-es hűtő tartalmát:

```
try {
        Class.forName("com.mysql.jdbc.Driver").newInstance();
        Connection conn = DriverManager.getConnection(url+dbName,userName,password);

        Statement st = conn.createStatement();
        ResultSet res = st.executeQuery("SELECT * FROM content WHERE Fridge_ID=1");
        while (res.next()) {
                row[row_count] = new SQL_row(Fridge_ID,res.getLong("ID"),
                res.getInt("brand"),res.getInt("type"),
                res.getDouble("mass"),res.getString("expires"));
```

```
                row_count++;
                }
        conn.close();
        }
catch (Exception e){
        e.printStackTrace();
        System.out.println("SQLException: " + e.getMessage());
        }
```

A conn objektum az TCP kapcsolat az SQL szerverrel. Az SQL szerverek általában TCP protokollt használnak. Magától értendően a conn.close(); zárja be a kapcsolatot.

A "SELECT * FROM content WHERE Fridge_ID=1" SQL parancs leadása után megkapja eredményként a táblázatot a res objektumba. A res.next()-el lépkedünk előre a táblázat sorain, és minden lépésnél beolvassuk azokat az értékeket, amelyek számunkra szükségesek.

Sokan írják a neten, hogy az Androidos készülékünkön az a JDBC driver nem működik, amelyik a PC-énken futó JAVA-val működött, de ez nem igaz. Ugyanakkor vannak alapjai, mert hamarosan ismertetni fogom mennyi mindent kellett babráljak az Android JAVA kódjával, hogy rácsatlakozzon az SQL adatbázisomra.

Miután láttam, hogy működik a JDBC driver, letöltöttem az Android programfejlesztő környezetet, melynek neve Android Studio, verziószáma 1.1.0.

4.2.2. Az Android programozás alapjai I. (activity-k)

Ebben a részben a program ablakairól és ablakai közötti ugrálásról fogunk beszélni. Ez a legfontosabb amikor mozgunk a programban.

Az Android programozása is JAVA-ban folyik, vannak hozzá külön Android-os osztályok (jar file-ok), hogy ez lehetséges legyen, de 2 fontos dolog, ami megkülönbözteti az általános JAVA-tól:

- activitykből áll
- minden activityhez tartoznak XML file-ok (html file-ok)

Ez egy kis magyarázatra szorul. Az Android kezelőfelülete olyan, mint egy honlap, és éppen ezért html-ben íródik. Vannak rajta gombok, menük, amiket mind html-ben kell leprogramoznunk. Egy képernyő mögött egy adott activity áll, mint gépezet, ami működteti. Ha megnyomunk egy gombot, ami behív egy újabb képernyőt az már egy újabb activity lesz, ami a létrejötte után behív egy újabb html filet. Pl.: Megnyomjuk az "OK" gombot a hűtőszekrényválasztásnál és meghívódik a táblázat XML file-a:

setContentView(R.layout.***activity_main***); *//ez tölti be az activity_main.xml-t*

Legegyszerűbben nézve, mint felhasználó, úgy érthetjük meg, hogy amikor lépkedünk egy Android applikációban különböző ablakok között, akkor minden ablak egy újabb xml file, amit egy újabb activity vezérel.

Hogy lássuk, hogyan kell a JAVA kódban az activityk között ugrálni, mutatok egy példát, amikor a bejelentkező képernyőről átugrunk a hűtő tartalmára:

Intent intent = **new** Intent(MainActivity.**this**, DataBaseActivity.**class**);

Ezzel létrehozunk egy új objektumot (intent), amivel képesek vagyunk a jelenlegi MainActivity-ből átugrani a DataBase osztályba, ami egy új Activity lesz, hiszen kibővíti az Activity osztályt:

public class DataBaseActivity **extends** Activity {...}

Egy nehézség van activity váltásnál, hogy a változókat, amikre szükségünk lesz, át kell vinnünk magunkkal az előző activityből az új activitybe, ez pedig az előző példánkban deklarált "intent" segítségével lehetséges.

A programomban az SQL adatbázisból még activityváltás előtt letöltöm adatokat, így kénytelen voltam egyesével átvinni őket:

```
intent.putExtra("number_of_rows",google_SQL.number_of_rows);
for(i=0;i<google_SQL.number_of_rows;i++){
   intent.putExtra("ID"+i,google_SQL.lines[i].ID);
   intent.putExtra("Fridge_ID"+i,google_SQL.lines[i].Fridge_ID);
   intent.putExtra("brand"+i,google_SQL.lines[i].brand);
   intent.putExtra("type"+i,google_SQL.lines[i].type);
   intent.putExtra("mass"+i,google_SQL.lines[i].mass);
   intent.putExtra("expires"+i,google_SQL.lines[i].expires);
}
```

Erre azért van szükség, mert, ha üres a hűtő, akkor fölösleges activityt váltani, ahhoz meg be kell nézni az adatbázisba, hogy megtudjuk, hogy üres-e. Azaz üres hűtőnél nem váltok activityt, hanem kiíratom a programmal, hogy a hűtő üres vagy nem létezik.

Ha nem üres, akkor meg mehet az activityváltás:

startActivity(intent);

Innentől kezdve bent vagyunk a DataBaseActivity osztályba és az okostelefonunkon felugrik az új ablak és minden változót átveszünk, amit az /*intent.putExtra();*/ metódusokkal átküldtünk. Ez olyan, mint Budapesten becsomagolni egy csomagot, és Debrecenben átvenni és kicsomagolni. Budapest a MainActivity és Debrecen a DataBaseActivity, akkor most csomagoljuk ki Debrecenben (a DataBaseActivity-ben):

Bundle bundle = getIntent().getExtras();

Ezzel át is vettük az extrákat, amiket az előbb becsomagoltunk. Megmutatom, hogy lehet egy értéket kivenni a "bundle"-ből (érdekesség, hogy a bundle magyar jelentése csomag), jelen esettben a kapott táblázat sorainak számát vesszük ki:

```
int number_of_rows = bundle.getInt("number_of_rows");
```

Megnéztük milyen az activityk között lépkedni. Most nézzük meg közelebbről mi is történik az új activitybe lépés után. Minden activityben van egy metódus, ami egyszer lefut az activity indításakor, ez az úgynevezett onCreate() metódus:

```
protected void onCreate(Bundle savedInstanceState) {...}
```

A többi metódust általában különböző felhasználói interrakciók befolyásolják. Pl.: Ha a felhasználó megnyom egy gombot, akkor végrehajtódik egy adott metódus. Amennyibe a jobb felső sarokban lévő menüben lépked szintén végrehajtódik egy metódus, méghozzá ez a metódus:

```
public boolean onOptionsItemSelected(MenuItem item) {...}
```

Amennyibe egy gombnyomás eredményeképpen szeretnénk elindítani egy metódust az az XML fájlból lehetséges ezzel a módszerrel:

```
<Button
  android:text="OK"
  android:id="@+id/OK_button"
  android:onClick="load_database" />
```

Ez esettben az "OK" gombra kattintás esetén lefut a load_database() metódus, ami betölti az SQL adatbázis megfelelő részét az Androidba, majd elindítja a DataBaseActivity-t. Ez a példa megmutatta az XML kódok és JAVA kódok szoros kapcsolatát.

4.2.3. Az Android programozás alapjai II. (Thread-ek)

Még egy jellemző eleme az Android kódoknak a párhuzamosan futó folyamatok (thread-ek). Meg kellett tanulnom a párhuzamos szálak használatát, mert a főszálban nem volt megengedett, hogy rácsatlakozzak az SQL adatbázisra, így létre kellett hoznom egy mellékszálat is.

Párhuzamosításhoz egy olyan osztályt kell létrehoznunk, ami implementálja az úgynevezett Runnable (futtatható) osztályt. Így kell létrehozni egy osztályt, ami párhuzamosan futtatható:

```
public class SQL_connection implements Runnable {...}
```

Az osztályon belül létre kell hoznunk egy "run" nevű metódust, ami párhuzamosan fut, ha meghívjuk. Itt egy példa a run() metódusra:

```
public void run(){
   this.download(); //ez tölti le a hűtő tartalmát az SQL szerverről
}
```

Észben kell tartanunk, hogy ezt a run metódust nem olyan egyszerű meghívni, mintha egy klasszikus metódus lenne. Egy Runnable osztály run() metódusát, így kell meghívni:

```
google_SQL = new SQL_connection("koncztom_tablet","hellotablet",fridge_ID);
Runnable SQL = google_SQL;
Thread SQL_thr = new Thread(SQL);
SQL_thr.setDefaultUncaughtExceptionHandler(_unCaughtExceptionHandler);
SQL_thr.start();
while(SQL_thr.isAlive()){}
```

Elsőnek létrehozunk egy olyan osztályú objektumot, amiben a run() metódus van (példánkban ez az SQL_connection osztály). Mindezt abban a szálban hozzuk létre, ahonnan indítani akarjuk az új szálat. Mivel az én programom maximum két szálú volt, minimum egy, ezért én a főszálban hoztam létre az objektumot. Ezután létrehozunk egy Runnable objektumot és a Runnable objektumból létrehozunk egy threadet (szálat):

```
Runnable SQL = google_SQL;
Thread SQL_thr = new Thread(SQL);
```

A szálnak metódusokkal parancsolunk. Előbb beállítunk neki egy kivételkezelőt:

```
SQL_thr.setDefaultUncaughtExceptionHandler(_unCaughtExceptionHandler);
```

Ezt a kivételkezelőt én deklaráltam. Abban az esettben lép bele, ha a szálban lévő kivételkezelő se képes lekezelni a kivételt. Tesztelés közben volt hasznos, mert egyes esetekben, amikor a kivételt nem lehetett lekezelni kiírta, hogy "nem tudta elkapni a kivételt". Így néz ki az _unCaughtExceptionHandler kivételkezelő:

```
private Thread.UncaughtExceptionHandler _unCaughtExceptionHandler =
    new Thread.UncaughtExceptionHandler() {
        @Override
        public void uncaughtException(Thread thread, Throwable ex) {
            System.out.println("NEM TUDTA ELKAPNI A KIVÉTELT");
        }
    };
```

A szálat a start() metódussal indítjuk el és ebben a programban addig fut, amíg le nem töltötte a hűtőszekrény tartalmát a szerverről. Miután sikerült letölteni átlép az új activitybe:

```
SQL_thr.start();
while(SQL_thr.isAlive()){} //a főszál vár, amíg a mellékszál dolgozik
/*az ezutáni kódrészben jön az új activitybe lépés*/
```

Az én megoldásom teljesen értelmetlenné tette a több szál használatát, mert a főszál addig vár, amíg a mellékszál le nem fut, hiszen mindenképpen meg kell várni az adatokat a szerverről. Nem is értettem miért nem tudtam elkerülni a 2 szál használatát a kódomban. Egyszerűen nem működhet hálózati kommunikáció a főszálban, mert a főszálat arra tervezték, hogy az ügyféllel tartsa fenn a folyamatos kapcsolatot. Például ne legyen az, hogy az ügyfél megnyom egy gombot, de a program nem vette észre, mert éppen a szerverrel kommunikált.

Az internettel való kommunikációs képességet külön kellett bekapcsolni az "AndroidManifest.xml"-ben. Ezt a 2 sort kellett hozzá beírni:

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

A folyamat, ami párhuzamosan fut az alkalmazásomban a főszállal ugyanaz a folyamat, amit "4.2.1. A MySQL szerverre csatlakozás JAVA kóddal" című fejezetben már kielemeztem, tehát erre nem térnék külön ki.

4.2.4. A menü létrehozása

Ebben a fejezetben a jobb felső sarokban lévő menü létrehozásáról fog szó esni. Először egy XML filet hozunk létre, ami a menü formáját, feliratait adja meg:

```
 <menu xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
xmlns:tools="http://schemas.android.com/tools" tools:context=".DataBaseActivity">
<item android:id="@+id/action_save_ID2" android:title="@string/action_save"
   android:orderInCategory="100" />
<item android:id="@+id/action_back" android:title="@string/action_back"
   android:orderInCategory="100" />
<item android:id="@+id/action_quit2" android:title="@string/action_quit"
   android:orderInCategory="100"/>
</menu>
```

Az előző XML 3 gombot definiál a menüben, abban a menüben, amelyik az adatbázis megjelenítése activityben előhívható. A gombokhoz így rendeltem JAVA kóddal parancsokat, ezt mind a DatabaseActivity osztályban tettem meg:

```
public boolean onOptionsItemSelected(MenuItem item) {
  int id = item.getItemId();
  if (id == R.id.action_save_ID2) {
    tablet.WriteToFile(fridge_ID);
    kiir = fridge_ID + " lesz az alapértelmezett hűtőazonosító.";
    Toast(); //kiírja a képernyő aljára a kiir tartalmát
    return true;
  }
```

```
    if (id == R.id.action_back) {
        System.out.println("CLOSED DATABASE ACTIVITY");
        finish();
        return true;
    }
    if (id == R.id.action_quit2) {
        Intent intent = new Intent(getApplicationContext(), MainActivity.class);
        intent.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
        intent.putExtra("EXIT", true);
        startActivity(intent); //elindítja a MainActivity-t
        return true;
    }
    return super.onOptionsItemSelected(item);
}
```

Az "action_save_ID2" gomb, elmenti az aktuális hűtőazonosítót alapértelmezettként, mindezt egy txt fileba menti el az Android készüléken. Az "action_back" gomb bezárja az aktuális activityt (DatabaseActivity), míg az "action_quit2" gomb bezárja az egész alkalmazást. Mindezt úgy teszi, hogy belép a MainActivity-be, és átküld neki egy EXIT=true-t. Az EXIT=true-t megvizsgálja a MainActivitybe és kilép. Így vizsgálja meg:

```
if (getIntent().getBooleanExtra("EXIT", false)) { //a false az alapértéke
    finish();
}
```

4.2.5. A hűtő tartalmának kiírása:

Az DataBaseActivity létrehozásánál hozzuk létre a táblázatot. Ekkor az előzőekben említett onCreate metódus fut le és ebben írjuk ki a táblázatot. Meg kell adnunk, hogy az activityhez kapcsolt XML fájl mely elemébe töltsük bele a táblázatot. A táblázat a data nevű elembe került:

```
TextView data = (TextView) findViewById(R.id.data);
```

Miután a MainActivity-ből átkerültek a táblázat elemei a DataBaseActivity-be, átmentjük őket egy szövegfájlba. melynek neve: "tablazat". Végül a szövegfájlt kiírjuk a képernyőre az XML fájl data elemén keresztül:

```
data.setText(tablazat);
```

Magyarul a fenti sor kiírja a "tablazat" stringet ebbe az elembe:

```
<ScrollView android:id="@+id/ScrollView01"
android:layout_width="wrap_content" android:layout_height="wrap_content">
<TextView android:id="@+id/data" android:layout_width="wrap_content"
```

```
android:layout_height="wrap_content" android:textSize="20dp"></TextView>
</ScrollView>
```

A "data" TextView egy ScrollView-ba van becsomagolva, erre azért volt szükség, hogy gördíthető legyen a táblázat, ha a hűtő tartalma nem fér ki egy képernyőre.

Pár napom elment, hogy a hűtő tartalmát táblázatosan szerettem volna kiírni <TableLayout>-ot használva. Az XML-ben ugyebár van <TableLayout>, amin belül <TableRow> segítségével írunk ki elemeket. Az volt a célom, hogy egy <TableRow>-on belül több <TextView>-t írok egymás mellé. Ez fix sorú és oszlopú táblázatnál működőképesnek látszott, de az én táblázatom dinamikus. Ebből adódóan csak akkor működne reálisan, hogy táblázat formájában írom ki a hűtő elemeit, ha maga a JAVA kódom generálja le a teljes XML file-t, de erre nem találtam hasznos megoldást. A későbbiekben még visszatérek erre a problémára. Ugyanis jelenleg a "tablazat" String ezen komplikált folyamattal generálódik:

```
for(j=0;j<(51-row[i].type.length());j++) {
   space1 = space1+" ";
}
for(j=0;j<(51-3-row[i].mass.toString().length());j++) {
   space2 = space2+" ";
}
tablazat=tablazat+row[i].type+space1+row[i].mass+space2+row[i].expires+
      "                    "+row[i].brand;
if(number_of_rows-1!=i){tablazat=tablazat+"\n";}
space1=""; space2=" kg";
```

A fenti kód fűzi össze a "tablazat" Stringbe a sorokat. Továbbá a fenti kód minden sor beolvasásánál ciklikusan fut le. Közben az i növekedik minden ciklusban. Ez egy elég favágó módszer, és szeretném, ha maga a JAVA kódom módosítaná az XML fájlt is. Miután oly sok napom ráment egy dinamikus táblázat lehetséges előállításával, beletörődtem és a "tablazat" Stringet egyszerűen kiírattam.

4.2.6. A hűtőazonosító elmentése/visszaolvasása:

Ez egy menüopció, ami mindkét activityben lehetséges a felhasználó számára. Ez egy lehetőség, hogy elmenthessük a hűtőazonosítót és a program újbóli bekapcsolásakor ne kelljen beírni. Egy Tablet típusú objektum által vezéreljük ezt a folyamatot. Ez az objektum a tabletünk operációs rendszerének tud txt file olvasó és író parancsokat kiadni. A Tablet objektumok általánosságban az operációs rendszerrel tartják fenn a kapcsolatot. Ez egy

általam deklarált objektum. Jelen esettben egy txt fájlba szeretnénk elmenteni az aktuális hűtőazonosítót, majd a később visszaolvasni azt.

Első lépésben lehetővé tesszük az írást és olvasást az Arduino tárhelyéről. Ehhez az "AndroidManifest.xml"-be bele kell írni ezt a sort:

```
<uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

Létrehozunk egy Tablet objektumot:

```
Tablet tablet;
```

Miután létrehoztuk a "tablet"-et, az onCreate metódusban hozzárendeljük a tablet aktuális állapotát:

```
tablet = new Tablet(getApplicationContext());
```

Ezután az általam deklarált 2 metódust lehet használni íráshoz vagy olvasáshoz. Íráshoz a WriteToFile(long s) metódusom használható, ami bármilyen számot képes kiírni egy txt fálba. Ezt a metódust legutóbb a menüopciót kiválasztó programkódban használtuk (4.2.4. A menü létrehozása fejezet). A WriteToFile(long s) metódust érdekességképpen megmutatom:

```
public void WriteToFile(long s) {
    File root = android.os.Environment.getExternalStorageDirectory();
    File dir = new File (root.getAbsolutePath() +
"/Diplomamunka/EnHutom/EnHutom_mentesek");
    dir.mkdirs();
    File file = new File(dir, "mentett_huto_ID.txt");

    try {
        FileOutputStream f = new FileOutputStream(file);
        PrintWriter out = new PrintWriter(f);
        out.println(s);
        out.flush();
        out.close();
        f.close();
    }
    catch (Exception e) {
        System.out.println("NEM írta KI a fájlba");
        System.out.println("WriteToFile error: " + e.getMessage());
    }
}
```

A WriteToFile(long s) újbóli használatánál átíródik az előző szám, és az új szám (long s) veszi át a helyét.

A fájlból kiolvasást meg csak a MainActivity onCreate metódusában használjuk:

```
txtView.setText(tablet.ReadFromFile()); //előzőleg elmentett hűtőazon
```

Ha belegondolunk csak a program indításánál kell automatikusan a mezőbe beírni az elmentett hűtőazonosítót. A másik szituáció, amikor automatikusan illene beírni a hűtőazonosítót az az, amikor a DataBaseActivity-ben elmentjük azt, és a menüből visszalépünk a MainActivity-be, de ekkor megmarad az előző MainActivity, mert nem került bezárásra, így nem is kell betölteni az elmentett hűtőazonosítót sem. Például beírjuk, hogy 2-es hűtő, megnyomjuk az "OK"-ét, és betölt a táblázat. Elmentjük a hűtőazonosítót és visszalépünk. Ekkor a rubrikában megmarad a 2-es szám, így azt rubrikát valóban csak a program indításakor szükséges automatikusan feltöltenünk.

Nézzük akkor meg, hogyan is működik a fájlból kiolvasó kód:

```
public String ReadFromFile() {
   File root = android.os.Environment.getExternalStorageDirectory();
   File dir = new File (root.getAbsolutePath() +
"/Diplomamunka/EnHutom/EnHutom_mentesek");
   dir.mkdirs();
   File file = new File(dir, "mentett_huto_ID.txt");
   String line="";
   try {
      BufferedReader in = new BufferedReader(new FileReader(file));
      line = in.readLine();
      in.close();
   }
   catch (Exception e) {
      System.out.println("NEM OLVASOTT KI a fájlból");
      System.out.println("ReadFromFile error: " + e.getMessage());
   }
   return line;
}
```

Hasonlóképpen a fileba író kódhoz ez is a /Diplomamunka/EnHutom/EnHutom_mentesek/mentett_huto_ID.txt filehoz fér hozzá. Csak ez nem beír egy számot a fileba, hanem kiolvassa a számot String formában és azt adja vissza. A visszaadott érték meg belekerül a MainActivity ablakának a szövegmezőjébe (az "OK" fölé).

4.2.7. Toastok:

Az Android képes rövid ideig egy szöveget megjeleníteni a képernyő alján, ami szép lassan elhalványul, majd eltűnik. Ezt a szöveget hívják Toastnak. Az emberek valamilyen siker után szoktak toastot (pohárköszöntőt) mondani, innen ered az osztály neve. A Toastok használata igen egyszerű. Én úgy használtam őket, hogy csináltam egy Toast() metódust, ami ciklikusan

lefut a főciklusban, de csak akkor "mond" Toast-ot, ha a "kiir" String értéke nem üres. Így néz ki a Toast() metódusom, ami 3,5 másodpercig jeleníti meg a "kiir" tartalmát:

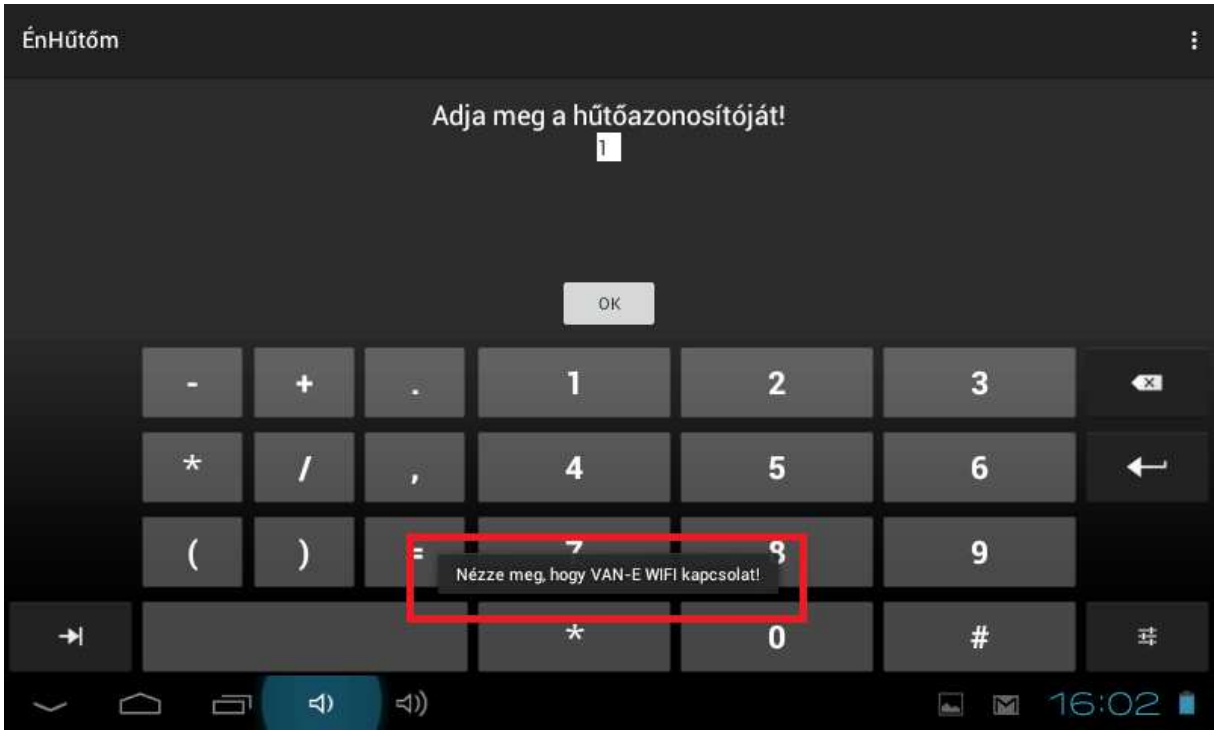
```
public void Toast(){
   if(!kiir.isEmpty()) {
      Toast.makeText(getApplicationContext(),
            kiir,
            Toast.LENGTH_LONG).show();
      kiir="";
   }
}
```

Toastokat használtam, hogy problémákat és sikereket jelezzek a felhasználó felé. Például: nincs internetkapcsolata, nem megfelelő a hűtőazonosító, a hűtőazonosító mentése sikeres vagy azt, hogy a hűtő üres.

Amennyiben nincs internetkapcsolat, erről a tablet objektumunk tudni fog. Ezzel az egyszerű módszerrel:

```
if (!tablet.isConnectedToInternet() && kiir.isEmpty()) {
   kiir = "Nézze meg, hogy VAN-E WIFI kapcsolat!";
}
```

Ennek hatására a program ki is írja, amikor belép a Toast() metódusba:



7. ábra - Példa egy toastra pirossal bekeretezve

Tehát, ha nincs internet, akkor kiírja az aljára toastként: "Nézze meg, hogy VAN-E WIFI kapcsolat!" Az internetkapcsolatra persze csak abban az esettben néz rá, amennyiben

megnyomtuk az "OK" gombot. Ez lesz nála az elsődleges, hogy megnézze, ezután nézi meg, hogy megfelelő-e a bemenet formátuma (a hűtőazonosító egy 1-től 4294967295-ig terjedő szám lehet) és végül azt nézi meg, hogy üres-e az adott hűtő vagy sem.

Érdekesség, hogy mivel ekkora számokat nem képes egy sima relációs jel összehasonlítani, ezért így jártam el, ez esettben lesz a szám kisebb, mint 4294967296:

```
if (fridge_ID / 1000 < 4294967 ||
(fridge_ID / 1000 == 4294967 && fridge_ID % 1000 < 296))
```

Azt meg, hogy üres-e az adott hűtő vagy sem, úgy állapítja meg, hogy van-e olyan termék az adatbázisban, ami az adott azonosítójú hűtőhöz tartozik. (megvizsgálja, hogy van-e az SQL adatbázisban megfelelő sor, megfelelő Fridge_ID attribútummal).

5. A hűtőtartalom felismerése

Egy olyan módszert kellett alkalmazzak, ami a programozás nehézsége alapján megfelel a tudásomnak. Így esett a választás az RFID technológiára. A következő részekben a felhasznált hardverekről lesz szó és a C++-ban alkalmazott kódjaimról. Mivel ez viszonylag rövid kód, ezért nem volt szükségem a C++ objektumorientáltságára.

5.1. Felhasznált hardvereszközök

A központi hardver az az Arduino UNO. Melynek processzora egy Atmel ATmega 328-as. Aki még nem hallott mikrokontrollerekről, azoknak dióhéjban elmondom a működésüket. Egy tenyérben elférő számítógépet képzeljünk el, ami lehet akár körömnyi is, mint a PIC16F628as mikrokontroller, amivel az előző szakdolgozatomban foglalkoztam (7). Van bennük processzor, ami logikai műveletek elvégzésére képes. Találhatóak bennük memóriaregiszterek, ahol eltárolják a program futása közben az adatokat. Olyan regiszterek is létezhetnek, amik megtartják az adatokat újraindítás után is. Található bennük egy úgynevezett watchdog timer, ami újraindítja a mikrokontrollert abban az esettben, ha bennragad a program egy végtelen ciklusban, magyarul lefagy. Képzeljünk el egy processzort sok lábbal, amiket lehet szabályozni, hogy melyik lábról menjenek ki az adatok és mely lábakon menjenek be. Ilyen lábai vannak a mikrokontrollernek. A lábakból kimenő és bejövő adatokat meg a memóriaregiszterek tárolják ideiglenesen. Ha a programkód elég odafigyelő, akkor le is olvassa őket. Ha még sem elég gyors a programkód, akkor elveszíthet bemenő adatokat. A kimenő adatok nyilván a programkód tempójában haladnak kifele.

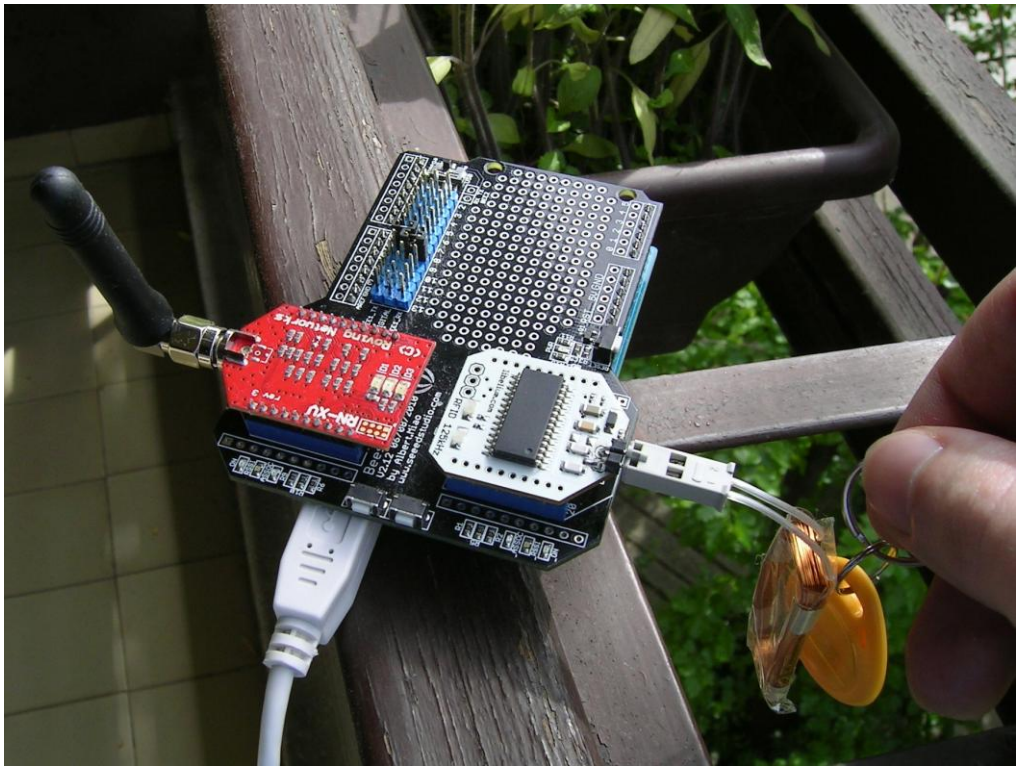
A mikrokontrollerekben található UART modul is, ami a gyors és RS232 protokoll szerinti adattovábbításért és adatfogadásért felelős a soros porton. Az UART modulhoz tartozik egy saját memóriaregiszter. Ha ennek a regiszternek a tartalma megváltozik, akkor a regiszter tartalmát kiküldi.

A bennük lévő processzor lábait kivezetik a mikrokontroller lábaira. Tudjuk, hogy a processzor 1-eseket és 0-ákat tud küldeni és fogadni. Minden mikrokontroller lába egy meghatározott feszültségszintre kapcsol, amennyiben a processzor 1-est akar küldeni, és megint csak egy adott feszültségszintre, ha 0-át. Pl.: Az Arduino UNO esetébe 5 V felel meg 1-esnek és 0 V felel meg 0-nak.

Gyakran található bennük AD átalakító a bemenő jelekre. Ilyenkor diszkrét időpillanatokban megvizsgálja az átalakító a bemenő jelet és egy egész számot rendel hozzá, amit az egyik regisztere kap meg. 8 bites regiszterek esetén 0-tól 255-ig kaphat értéket az analóg jel feszültségének nagysága. Pl.: Az Ardunio UNO-nál 10 bitre van elosztva, így 5 V felel meg 1023-nak és 0 V 0-nak.

Kimeneti konstans jelként nem képesek a mikrokontrollerek analóg jelet előállítani, hanem egy PWM jelet állítanak elő. Ez alkalmazható, amennyiben egy analóg jelet szeretnénk leutánozni. A PWM jel az egy négyzetes jel. Pl.: Ha 2 V-ot szeretnénk előállítani, és a négyzetes jel egy ciklusa T ideig tart, akkor 3T/5 ideig 0 V-ot ad ki, és 2T/5 ideig 5 V-ot. Mivel T nagyon kicsi, ezért a jel olyan hatást fejt ki a legtöbb esettben, mint egy 2V-os analóg jel.

Tehát mikrokontrollerként egy Arduino UNO-t (19) választottam. Kellett hozzá választani egy wifi modult és egy RFID modult. Szerencsém volt, mert az egyetemnek pont volt egy Bees Shield-je (20), amire mindkettő rácsatlakoztatható (a Bee1 és Bee2 foglalatokra). Az egyetem biztosított még egy SM125-ös RFID modult (22), hozzá 5 db EM4100 RFID tag-et (csak olvasható, 5 bájt tárolására képes, 125 kHz-es) (24) és egy RN171-es wifi modult (28).
Így néz ki az Arduino UNO összerakva a Bees shielddel és a rajta lévő modulokkal:



8. ábra - Az Arduino UNO a rajta lévő eszközök takarásában

A fehér modul a réztekercsel az RFID olvasó, és a piros modul az antennával a wifi modul. A sárga kulcstartó meg az RFID tag.

5.2. A hardvereszközök programozása

Egyedül az Arduino UNO-t kellett közvetlenül programozzam a számítógépemről. A két modul programozása közvetetten az Arduino UNO-n keresztül történt. Úgy kell elképzelni, hogy az UNO az agy, ami a végtagokat és érzékszerveket vezérli. Az UNO-t a Microsoft Visual Studioval programoztam úgy, hogy letöltöttem rá a Visual Micro bővítményt, aminek segítségével bármelyik Arduino programozható. A C++ nyelv meg tökéletesen testközeli volt, hisz a munkahelyemen is azt használom.

5.2.1. SM125 RFID érzékelő:

Az RFID érzékelők működési elve az, hogy gerjesztenek egy erős elektromágneses teret. Ha közel rakunk egy RFID kulcsot (tag-et) az elektromágneses térhez, akkor a kulcsban áram indukálódik. Ez az áram pont elég, hogy a kulcs rádiójeleket bocsásson ki az RFID érzékelő irányába.

Az SM125 esetében 125 kHz-esek ezek a rádiójelek, éppen ezért a kulcsoknak is kompatibilisnek kell lenniük a 125 kHz-es rádiójelekhez.

Fejlettebb RFID kulcsokra lehet adatokat írni, az enyémről csak olvasni lehet. Magyarul minden kulcsom kizárólag a saját azonosítóját (5 bájt) küldi át az RFID érzékelőnek, és ezt az azonosítót továbbítja az érzékelő az UNO irányába, ami a terv szerint továbbítani fog a neten keresztül 1-5-ig egy számot a szerver irányába. Azért 1-5-ig, mert 5 db tag-je van a prototípusnak. A szerver JAVA kódja megkapja a számot, és átírja a rajta futó SQL adatbázist.

Az UNO az SM125-tel az alapértelmezett soros porton kommunikál. Problémát okoz, hogy a programkódot is ezen a soros porton töltöm fel, tehát nem lehet programozni és tesztelni egyszerre, hanem előbb feltöltöm a kódot, aztán rárakom az RFID olvasót és tesztelem. Ha bármilyen adatot küldenék a PC irányából az UNO felé, akkor az RFID olvasó is megkapja azt és lefagy, hiszen mindketten ugyanazt a vezetéket használják. Ez a működő prototípusban nem okoz majd gondot, hiszen az nem lesz rákötve PC-re, hanem csak egy akkumulátorra a hűtőszekrényben. Az akkumulátorok úgyis 0 fok körül érzik a legjobban magukat.

Mielőtt az SM125-tel dolgozhatnánk, be kell állítanunk azt automata üzemmódra, és Manchester RF/64 kódolás használatára, hiszen az EM4100-as kulcsok is ezt használják (25). Ehhez meg kell tanulnunk hogyan kell az SM125-tel kommunikálni, ami a leírásából kiderül (22). Alapértelmezetten 19200 baud bitrátával kommunikál, ezért ezzel a sebességgel küldtem neki az adatcsomagokat. Ez azt jelenti, hogy 19200 bitet tud küldeni és fogadni másodpercenként. Fontos, hogy az UNO is 19200 baudra legyen állítva, ezt a setup()-ban érdemes megtenni: Serial.begin(19200);

Akkor hát nézzük is meg, hogyan épül fel egy SM125 felé tartó adatcsomag:

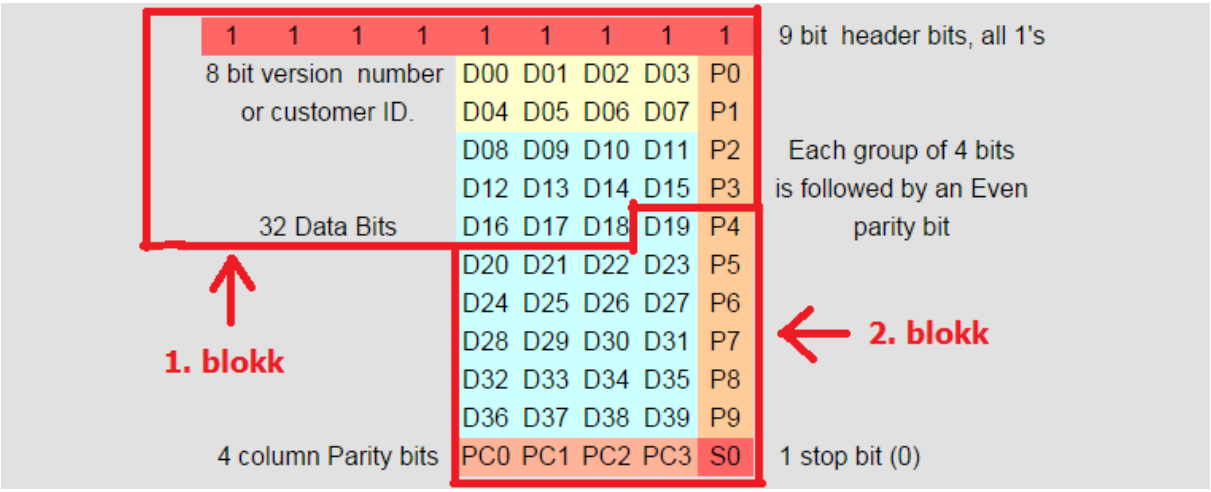
1. Header (1 bájt): 0xFF
2. Reserved (1 bájt): 0x01
3. Length (1 bájt): az adatcsomag hossza, ami 1+N bájt
4. Command (1 bájt): a használt parancs
5. Data (N bájt): a parancshoz szükséges adat
6. CSUM (1 bájt): a headert kihagyva összeadjuk az összes bájtot (checksum)

Az SM125 felől is pont ugyanígy néznek ki az adatcsomagok.

Miután parancsként elküldjük neki a 0x87-es parancsot (CMD_SET_AUTO_MODE) és a hozzá tartozó adatmezőt, beáll automata üzemmódba és Manchester RF/64 kódolásra.

A 0x87 parancshoz tartozó adatmező a mi esetünkben ezeket az értékeket tartalmazza:

1) Auto Mode Enable/Disable (1 bájt): 0x01 (Enable)
2) Mode (1 bájt): 0x01 (Byte Track Mode – Manchester RF/64)
3) Block Number (1 bájt): 0x02 (ez a kiolvasandó blokkszám, 1 blokk = 4 bájt):



9. ábra - Egy példa a 2 blokkra az EM4100 RFID tag esetében ((23) és (25))

Ezt a 2 blokkot küldi az RFID tag ciklikusan: ...blokk 1,blokk 2, blokk 1, blokk 2...

4) Password Enable/Disable (1 bájt): 0x00 (Disable)
5) Password (4 byte): pl.: 0x10, 0x20, 0x30, 0x40 (mivel nem használjuk mindegy, de kötelező a megadása)

A válasz erre a parancsra egy Success (siker) üzenet (0x99), ami így néz ki:

0xFF 0x01 0x01 0x99 0x9B

Röviden a Manchester RF/64-es kódolásról annyit illik tudni, hogy bitek küldésére képes úgy, hogy egy bit küldésének a felénél paritásváltás történik. 1-es bit küldése esetén alacsony feszültségről magas feszültségre vált, és 0-ás bit küldés esetén magas feszültségről alacsony feszültségre. Két bit között nem feltétlen van paritásváltás. Az RF/64 meg azt jelenti, hogy egy RFID tag 64 bites, azaz 2 blokkot küld el, ahogyan az előző ábra mutatta.

Miután elküldtük a 0x87 parancsot (CMD_SET_AUTO_MODE) az adatmezőjével együtt, nincs szüksége több parancsra. Az automata üzemmód azt jelenti, hogy, ha érzékel egy tag-et, akkor automatikusan továbbítja a rajta lévő 5 bájtot az Arduino felé, ami pedig leolvasva azt, és felismeri a tag-et. Erre alkalmas az RFID_tag() metódusom:

```
int RFID_tag(){
    if (Serial.available() >= 10){
        for (i = 0; i < 10; i++){
            RFID_data[i] = Serial.read();
        }
        if (RFID_data[0, 1, 2, 3] != (0xFF, 0x01, 0x06, 0x10)) //így kezdődik mind
            return 0;
        if (RFID_data[4, 5, 6, 7, 8] == (0x0B, 0x00, 0x53, 0x58, 0x03))
            return 1;
        if (RFID_data[4, 5, 6, 7, 8] == (0x06, 0x00, 0x6A, 0xFA, 0x42))
            return 2;
        if (RFID_data[4, 5, 6, 7, 8] == (0x0A, 0x00, 0x38, 0xB4, 0x75))
            return 3;
        if (RFID_data[4, 5, 6, 7, 8] == (0x0B, 0x00, 0x64, 0xA8, 0xF3))
            return 4;
        if (RFID_data[4, 5, 6, 7, 8] == (0x0B, 0x00, 0x74, 0x44, 0x43))
            return 5;
        return 0;
    }
    return 0;
}
```

A metódusból látszik, hogy 0xFF, 0x01, 0x06, 0x10-zel kezdődik mindegyik adatcsomag, amire szükségünk van. A 0x10 az a CMD_READ parancs, ami továbbítja a beolvasott RFID azonosítókat a parancs utáni adatmezőben. Ahhoz, hogy az UNO valós időben tudja, hogy

mikor melyik tag-et húztuk el az érzékelő előtt a loop()-ba raktam be az RFID_tag() metódust: tag=RFID_tag();

A tag aktuális értéke kerül a későbbiekben továbbításra a szerver felé (amennyibe nem egyenlő nulla és kész lesz a szerveren futó JAVA kód). A szerveren futó JAVA kód átírja az SQL adatbázist, ami pedig hatást fejt ki az Android alkalmazásra.

5.2.2. RN171 wifi modul:

Most összpontosítsunk arra az egységre, ami internetkapcsolatot létesít a jövőben az Arduino UNO és a szerver közt, ha kész lesz a prototípus.

Az RN171 alapértelmezetten 9600 baud bitrátával kommunikál egy általunk kiválasztott szoftveres soros porton. Szerencsére ez nem akad össze az UNO hardveres soros portjával, mint ahogyan az SM125 összeakadt. Továbbá nem szabad a Bees Shield jumpereléséről (20) sem megfeledkeznünk, ugyanis nem elég a C++ kódban beállítani az RX/TX lábakat, hanem a shielden is be kell állítanunk. Az én esetemben a Bee1 foglalat TX lába a 8-as, és RX lába a 9-es pinre van jumperelve. A Bee1 foglalatban az RN171-es modul van, míg a Bee2 foglalatba az SM125-ös.

Csináltam egy roppant egyszerű kódot, amivel kiírtam PC-re a wifi modultól kapott jeleket, és tudtam üzenni a wifi modulnak. Ez kezdetben roppant furcsa jeleket küldött, és nem azért, mert a baud ráta volt rossz. A munkahelyemen is volt már hasonló probléma. Ha egy eszköz a megfelelő baud ráta mellett sem küld tiszta jeleket, akkor meg kell próbálkozni az AltSoftSerial-lal. Ezt be is raktam a kódomba:

#include <AltSoftSerial/AltSoftSerial.h>

Ennek érdekessége, hogy pontosabb, mint egy sima szoftveres soros kommunikáció, mert az ütem tartására felhasználja a timert. Hátránya, hogy, ha használjuk, akkor az analogWrite() parancs hatástalanná válik, ugyanis annak szüksége van a PWM-re. A PWM-nek meg szüksége van a timer-re. Viszont mi a feladatunkhoz nem használjuk az analogWrite()-ot, így az "AltSoftSerial.h" nyugodtan alkalmazható.

Elsőnek deklaráljuk:

AltSoftSerial wifi; //RX=8 TX=9

Majd a setup()-ban adunk neki baud rátát:

wifi.begin(9600);

Ezután ugyanazzal a módszerrel kommunikálunk a wifi modullal, mint ahogyan az RFID modullal tettük, csupán a parancsok mások. Hozzáteszem, hogy a wifi modullal kommunkálni sokkal emberközelibb, mindez abból adódik, hogy angol kifejezésekkel lehet vele szót érteni, ellentétbe az RFID modullal, amivel hexadecimális számokkal kommunikáltunk.

Ha sikeresen beállítottuk a kapcsolatot a wifi modul és az UNO közt, be kell lépnünk command módba, hogy rácsatlakoztassuk a szobánkban vagy lakásunkban található wifi routerre. Ezt 3 $ jellel tesszük: "$$$" (enter NÉLKÜL) elküldjük neki, és, ha minden jól ment válaszként kapunk egy: "CMD"-ét. Ezután minden parancsot (28) követően használjunk ENTER-t.

Amennyiben a wifi routerünknek van tűzfala, át kell engednünk az RN171-est a routeren. Ezt úgy tehetjük meg, hogy az RN171-es MAC címét be kell tennünk a kivételek közé. MAC címét ezzel a paranccsal kérdezhetjük le: "get mac" Én a saját wifi routerem beállításaiban beraktam a kivételek közé.

Most csatlakozzunk rá a wifi routerre, amit ezzel a parancssorozattal tudunk megtenni:

1) **"set ip dhcp 1":** Ez a parancs bekapcsolja a DHCP módot. A DHCP protokoll az, ami a wifi routertől IP címet kér. Erre szükségünk lesz, hiszen mi szeretnénk egy saját IP címet.
2) **"set ip protocol 1":** Beállítja a használt internet protokolt TCP szerver és kliens protokolra. Úgy tervezem, hogy TCP protokollal kommunikálok a szerverem JAVA kódjával, de a "set ip protocol 0"-val UDP-ét is használhatok.
3) **"set wlan phrase *******":** Ez a parancs beállítja a routerjelszót.
4) **"join TurboNet":** Ez a végső csatlakozó parancs, ami a megadott SSID-jű routerre csatlakozik. Az én esetemben ez a "TurboNet" nevű router.

Rögtön a setup()-ban csatlakoztatom az Arduino UNO-t a netre a Start_WIFI() metódussal. Ez a Start_WIFI() metódus tartalmazza az összes wifi-re csatlakozó parancsot, amit az előző 4 lépésben említettem.

Egy sikeres csatlakozási folyamat így néz ki, ha csak a RN171-esből kijövő szöveget figyeljük. Ez az előző 4 parancsra kapott válasz:

```
Joining TurboNet now..
Associated!
DHCP: Start
DHCP in 2221ms, lease=86400s
IF=UP
```

DHCP=ON
IP=192.168.0.15:2000
NM=255.255.255.0
GW=192.168.0.1

A pirossal jelölt "GW=" szövegrészt észreveszi a kódom és ebből tudja, hogy a csatlakozás sikeres. Az IP címet a wifi modul 86400 s-re, azaz 24 órára kapja.

A sikeres csatlakozás után TCP protokoll segítségével rácsatlakozunk a szerveremre, melynek IP címe 146.148.15.14. JAVA kódomnak jelenleg még nincs portja, mert azt kell még kisilabizálnom hogyan töltsem rá a szerverre miután megírtam. Tervezem, hogy 1-2 hónapon belül megírom, mert szeretném a bizottságnak megmutatni. Tehát akkor így lépnék rá a wifi modulról a szerveremre:

open 146.148.15.14 <portszám>

Miután a kapcsolat megnyílik kapunk egy *OPEN* üzenetet.

Ezután egy "Hello UNO!"-val köszönteném a wifi modult a szerveren futó JAVA programmal.

Ha meg valamelyik fél megszakítja a kapcsolatot, akkor azt egy *CLOS* jelzi (nincs a végén E betű).

Azt tudni illik, hogy a wifi modul alapértelmezett portszáma a 2000-es, bár nem hiszem, hogy bármelyik programkódomnak szüksége lenne rá, hisz az egész úgy van tervezve, hogy az UNO kezdeményez. Remélem sikerül létrehozni egy ehhez hasonló TCP kommunikációt:

https://youtu.be/LI0KQlUAzQ8 (31)

Annyi a különbség, hogy ebben a videóban a szerver szólítja meg az RN171-est, én pedig az RN171-essel szólítanám meg a szervert.

6. A szerveren futó JAVA program

Ennek a programnak csak az alapjait fektettem le és programoztam meg. Elvileg egyszerűnek hangzik a feladata. Rácsatlakozik a programra az Arduino UNO. Átküldi neki a saját kis azonosítóját, hogy a JAVA program tudja melyik hűtő csatlakozott rá. Aztán számokat kap 1-től 5-ig az UNO-tól. Minden szám egy sort képvisel a szerveren pl.: 1 = (14-es azonosítójú, 0.5 kg kezdeti tömegű, sajt, 2015-7-10 lejárati dátum, Trapista), tehát az a konkrét sajt, amit ki-be pakolgatunk a hűtőből. Ha az a konkrét sajt már szerepel az adatbázisban, mert már régebben betettük, akkor töröljük a sorát. Ha meg még nincs az adatbázisban, akkor beillesztjük a sorát.

"3.3. Az adatbázis és felhasználói jogosultságok létrehozása" című fejezetben írtam, hogy létrehoztam egy "koncztom_UNO" nevű MySQL felhasználót, akinek SELECT, INSERT és DELETE jogosultságokat adtam. "koncztom_UNO" lesz az a felhasználó, akit a JAVA kódom használni fog. A SELECT utasítással megnézi mi van a hűtőben, ha a példában szereplő konkrét sajt nincs benne, akkor INSERT-tel berakja, ha meg benne találja, akkor DELETE-tel kitörli.

Jelenleg ez a program még nagyon kezdeti stádiumban van, annyira képes, hogy elmentse a táblázat tartalmát egy tömbbe. A tömb egy eleme az egy sor a táblázatban. Remélhetőleg nemsokára ellátja a feladatát. A feladatát, amire terveztem, hogy egy hidat képezzen a kevés memóriájú Arduino UNO és az SQL szerver között. Az UNO nem SQL kezelésre született, hiszen nagy adatmennyiségről van szó az ő kicsi memóriájához mérten. A témavezetőm mondta, hogy az UNO nem fogja bírni és javasolta ezt a köztes áthidaló programot.

Így újabb reményekkel nekilátok.

7. A látványterv

Mielőtt megírnám az összefoglalást, felvázolnám e rövidke részben a víziómat, hogyan képzelem el az ehhez hasonló rendszereket a jövőben. Sok embernek vannak víziói a jövővel kapcsolatban.

Mikor édesanyám fejéből kipattant az igény, hogy szeretne egy rendszert a hűtőszekrényünkre. Gondolkodtam és sorban jöttek az ötletek. A látványterv nem ábrákban és fotókban rajzolódik az olvasónak, hanem az agyában, ahogy olvassa a tervet. Ebben a fejezetben tárgyaltak nagy része megvalósítatlan, és csak a képzelet szüleménye. A terv arra épül, hogy az átlagembernek megfizethető legyen, időt és pénzt spóroljunk nekik. Hogy egy okoshűtő birtoklásához ne kelljen megvenni egy teljes hűtőszekrényt, hanem csupán egy kiegészítőt.

Az emberek a megkérdezettek alapján egyszerű és létfontosságú termékekre kíváncsiak. Például a tej, vaj, sajt, tojás és valami húsféle. Továbbá sokan véletlenül nyitva hagyják a hűtőt, velem is előfordult már. Valamint nem mindenkinek van okostelefonja, ezért az ajtón lévő mágneses tablet csatlakozhat a wifi-re. Efféle funkciókat képzeltem el neki: hangüzenetet hagyni az éppen hazaérkező anyukának, hogy tudják hova ment a gyerek, hiszen a szülők rengetegszer izgulnak, amikor a gyerek nincs otthon. Ez a képernyő jelenítené meg, ha valami nemsoká megromlik, és javaslatot tenne mit érdemes készíteni a hamarosan megromló ételekből. Erre már vannak megoldások. (32) A Samsung az http://www.epicurious.com/ receptes oldallal összeállt. Azon termékeket, amiket fel szeretnénk használni bejelölgetjük, és kiírja a program a lehetséges recepteket. Valamint lehetséges receptek alapján tud összeállítani bevásárlási listát. Továbbá ezen a tableten jelenne meg, ha valamelyik családtag éppen vásárolni van. Sípolna az eszköz, ha az ajtó egy bizonyos időn túl nyitva marad, hiszen giroszkópokkal tudná érzékelni az ajtó nyitását és zárását. Vagy egy hőmérővel is visszaigazolhatjuk, hogy túl meleg lett a hűtőben.

Ha már van egy fölösleges tabletje a családnak, akkor rárakhat mágneseket az aljára és letöltheti rá az Android applikációt. Egy központi szerver vezérelné az egészet, hasonlóan a facebookhoz. Összeállíthatunk vásárlási listákat, átküldhetjük a barátainknak egy kerti partihoz vagy házastársunknak a bevásárláshoz. Mindig tudnánk ki vállalta el a bevásárlást, mert megnyomta az: "Én szeretnék ma bevásárolni" gombot. A hűtő kiírja, hogy ma feleség bevásárol, és behúzhatunk a vásárlási listájába például 1 üveg sört. Fontos, hogy egybe tudja

kezelni különböző emberek vásárlási kívánságait. Így a vásárlás után a blokkot lefotózva hozzárendeli a szoftver az árat a megfelelő termékekhez, hogy könnyű legyen hazaérve beszedni a pénzt a lakóközösség tagjaitól. Valamint a blokkot lefotózva betölti a termékeket a hűtő adatbázisába. Vagy, ha akarjuk, akkor a központi szerverről leolvashatjuk egy termék árgrafikonjait, amit a lefotózott blokkokból gyűjt össze. Amint az első árak beérkeztek hajnalban a szerverre megtudtuk, hogy kinyílt a bolt vagy a piac. Már kint van a paradicsom, rohan is érte a kifőzde tulajdonosa. Megnézhetjük a hűtőajtó nyitogatási statisztikáit, hogy melyikünk hagyja általában nyitva a legtovább. Ki eszik a legtöbbet? A fogyasztás alapján a szoftver személyre szabott edzésprogramot tervezne. A lényeg, hogy a beérkező rengeteg adatból a lehetőségek tárháza végtelen, és pontosan ezért közöltem a bevezetésben, hogy a teljes terv megvalósításához egy élet munkája is kevés lenne.

A cégek ott akadtak el, hogyan ismerjék fel az ételeket, italokat. Sok okoshűtő még az okoshűtő jelzőt se érdemli meg, mert manuálisan kell beütnünk az ételeket a képernyőn, és amikor kivesszük őket, akkor manuálisan kell őket kitörölni szintén a képernyőt nyomkodva.

Az újak a blokkok felismerésére álltak rá. A legbarátságosabb megoldása szerintem az LG-nek van, mert vásárlás után csak lefotózzuk a blokkot és az árut be is rakja a hűtő adatbázisába. (33). Viszont, amikor kikerül a hűtőből, akkor manuálisan kell kitörölnünk (34).

Egy amerikai rokonom is tanácsolt egy felismerési lehetőséget amerikai ügyfelek részére, ahol szokás filctollal ráírni a húst tároló műanyag dobozra, hogy milyen húst tárolunk benne. Ott elterjedtek a műanyag rekeszek és a ráírási technikák. Hogy felismerjük a húst, az a dolgunk, hogy felismerjük a kézírást és, hogy melyik rekeszre írja az ügyfél.

Egy véglegesen használható rendszerben kitalálhatunk újabb és könnyebben használható felismerési technikákat. Ezen technikákat kombinálva, pedig kihozhatunk valami igazán különlegeset.

8. Összefoglalás

Az eddig végzett munka örömmel töltötte meg szívemet. A művészibb részeket, pl az Android alkalmazás kinézetét hajnalig pingáltam, közben zenét is hallgattam, mert ahhoz nem kell nagy ész. Lehet színezgetni, szépíteni, hogy a színkonstrukció harmonikus legyen, az ikonja szép legyen. Az alkalmazáshoz az ikont 3 különböző felbontásban kellett létrehozzam. Nappal meg a dizájn felett uralkodó természeti rendet, gépezetet komponáltam, halkan, csendben.

Őszintén szólva nem gondoltam volna, hogy Android alkalmazást készíteni ilyen körülményes. Eddig az Android kódokba fektettem a legtöbb időt és energiát. Mentségemre szól, hogy életem első Androidos programja volt. Rengetegszer elakadtam, de a google keresőmotor gyakran kisegített, de gyakran félre is vezetett. Pl.: a netes közösség meg volt arról győződve, hogy a JDBC driver (JAVA-ára SQL driver), ami nekem PC-én működött, Androidon nem fog működni. Így olyanokat próbáltam, amiket ajánlottak, de egyiket se sikerült beüzemelnem. Végső elkeseredésemben kipróbáltam azt, amiről azt mondták ki se próbáljam, és bevált. Az esetek 90%-ában nem vezet félre a google, így még a jövőben is hasznos munkaeszközöm marad.

Az alkalmazásban három dolgot szeretnék még megvalósítani, ami úgy érzem kimaradt, de mivel voltak a feladat működése szempontjába lényegesebb részek, ezért átugrottam őket. Az első az a hűtőtartalom szinkronizálása a szerverrel t időtartalmanként. A második az, hogy a már megromlott ételek sorát befestem pirosra, az x időn belül megromló ételeket meg sárgára. Az x-et és a t-ét a menüben lehet majd beállítani. A harmadiknak akkor van értelme, ha sok ételt tárolunk. Egyszerűen rákattintva az attribútum nevére a táblázat első sorában, az adott attribútum szerint sorba rakja az elemeket pl.: ha lejárati sorrendben akarjuk meginni a tejeket. Ehhez a harmadik bővítéshez tartozna egy keresőmotor is, ami segíti az étel vagy ital megtalálását.

A MySQL szerver létrehozása után is volt egy nagy elakadásom. Nem gondoltam, hogy napokat vesz majd igénybe mire rájövök, hogy mi az oka annak, amiért nem tudok a Heidi SQL kezelőmmel belépni az adatbázisba. Bementem a cégemhez dolgozni, egy munkanapnak indult, de az lett belőle, hogy a főnököm segített új felhasználót létrehoznom a MySQL-ben és kikapcsolni az Ubuntu tűzfalát, így aznapra a fizetésem se kértem. Nem tervezek a jövőben változtatni az adatbázison. Ha több lenne a termék, akkor annyit tennék még, hogy minden

hűtőszekrény tartalmát külön táblázatba raknám. A táblázatokat nem szeretem összekapcsolni, tudni illik a JOIN művelet nagy erőforrásigénnyel jár. A google megnyugtatott, hogy még maradt az általuk biztosított 300 dollárból 280 dollár, így még körülbelül 3 évig ingyenesen fenntartják számomra a szervert. Lehet ki fogom használni, ha már ennyire támogatják a fejlődő elméket. Kipróbálok pár dolgot. Azt szeretném a legjobban tudni, hogy hogyan lehetséges a szerveremről weblapot futtatni. Ráadásul a szerver kezelőfelülete megadja a hackerfilmekre jellemző élményt, hogy fehér betűk egy fekete képernyőn és semmi grafikus megjelenítés. Ha meg megjelenik a felületén pár színes betű, akkor már giccsessé válik.

Ezek után túlságosan kitűnne, ha azt mondanám Arduino UNO-val töltött időm sima ügy volt, akár vajas kenyeret kenni. Be kell vallanom, hogy az Arduino kódom fölöslegesen túl van bonyolítva, és ez növeli az átláthatatlanságát. Főleg, hogy a kezdeti terv az volt, hogy egy SELECT utasítással letöltöm az UNO-ra a hűtő teljes aktuális tartalmát, és ennek alapján módosítom azt, de már a szerverre csatlakozásnál beletört a bicskám. Saját magam próbáltam egy drivert összerakni Arduino-ra, ami belép a szerverre, de be se tudtam lépni, hisz a jelszót sha1-es titkosítással kellett volna elküldeni a szervernek, de ez az egész folyamat egyszerűen megtelítette az UNO SRAM-ját. Nem írta miért fagy le, azt hittem elrontottam a kódot, de abból, hogy az IDE a programnak már rögtön az indításánál kiírta, hogy majdnem 90%-a foglalt az SRAM-nak, arra engedett következtetni, hogy SRAM telítődés miatt fagy le. Így letettem az eredeti tervről, elmondtam a problémát a témavezetőmnek, aki szinte azonnal kipattant egy ötlettel, hogy ékeljek be egy programot az UNO és a MySQL szerver közé. Ezzel kapcsolatos, hogy hasznos, és érdekes dolgokat olvastam az Arduino memória túltelítődéséről ezen a blogon: (30)

Miután újra egyszerűvé és átláthatóvá tettem az Arduino kódot, két bővítést tervezek benne. A legfontosabb, hogy a témavezetőm által javasolt áthidaló JAVA kóddal kommunikálni tudjon. A második meg egy opcionális bővítés, amit szintén a témavezetőm javasolt, hogy egy kapcsolóval, ami lehet akár egy drót is, a felhasználó be tudná állítani, hogy most éppen ki akar venni termékeket a hűtőből vagy be akar rakni termékeket a hűtőbe. Ezt a kódban egy flag-gel jelölném, amit szintén megkapna a JAVA kód.

Ezek a fejlesztési lehetőségek, amelyeket az összefoglalásban felsoroltam, mind olyanok, amiket egyszerű tudásommal és kevés tapasztalatommal végre tudok hajtani. Most 1-2 napot pihenek a májusi hőségben, aztán nekilátok az Arduino kód leegyszerűsítésének, majd az áthidaló JAVA kód megírását folytatom. Végül TCP vagy UDP protokollal összekötöm a

JAVA kódot és az Arduinot. A TCP felé hajlok, mert az biztonságosabb és az adatmennyiség mérete se indokolná meg az UDP-ét.

Úgy érzem akkor lesznek újból nagy elakadásaim, amikor az áthidaló JAVA kódot futtatnám a Linux szerveremen, hogy napi 24 órában elérhető maradjon a MySQL szerverrel párhuzamosan.

A feladat elvégzése során kicsit úgy érzem magam, mintha egy egyszemélyes multinacionális vállalat lennék egy Nyugat-Európai szerverrel, amiről havonta kapom a számlakivonatokat. Betekintettem több programozási nyelvbe. Dolgozok felváltva több "részlegen".

Zárásként kijelenthetem, hogy megérte. Hiszem, hogy az adatbázisok távoli karbantartása nagy jövő előtt áll, viszont pár hónap múlva visszamegyek kis cégemhez programozni azt, amit oly sokszor programoztam, Arduinot.

9. Irodalomjegyzék

1) Charles Bell:Beginning sensor networks with Arduino and Raspberry Pi:2013, Technology in Action
2) Faludi Róbert:Building wireless sensor networks:2011, O'Reilly Media
3) V. Daniel Hunt, Albert Puglia, Mike Puglia:RFID: A guide to radio frequency identification:WILEY
4) Ruzsinszki Gábor:Programozható elektronikák:2014, Ruzsinszki Gábor
5) Ekler Péter, Fehér Marcell, Forstner Bertalan, Kelényi Imre:Android-alapú szoftverfejlesztés - Az Android rendszer programozásának bemutatása:2012, Szak Kiadó Kft.
6) Paul Hyde:JAVA thread programming:1999, Sams publishing
7) Koncz Tamás:Költséghatékony, intelligens házfelügyeleti rendszer kialakítása PIC mikrovezérlővel:https://www.dropbox.com/s/8ghwgkgjmxgv996/Koncz_Tam%C3%A1s_szakdolgozat.rar?dl=0:DEIK, 2012, Debrecen
8) ***:Internet refrigirator:http://en.wikipedia.org/wiki/Internet_refrigerator
9) ***:Internet of Things:http://en.wikipedia.org/wiki/Internet_of_Things
10) ***:Google Developers Console:https://console.developers.google.com/
11) Vivek Gite:MySQL/MariaDB Server: Bind to multiple IP Address:http://www.cyberciti.biz/faq/unix-linux-mysqld-server-bind-to-more-than-one-ip-address/
12) ***:MySQL:https://help.ubuntu.com/12.04/serverguide/mysql.html
13) Etel Sverdlov:A basic MySQL tutorial:https://www.digitalocean.com/community/tutorials/a-basic-mysql-tutorial
14) Etel Sverdlov:How to create a new user and grant permissions in MySQL:https://www.digitalocean.com/community/tutorials/how-to-create-a-new-user-and-grant-permissions-in-mysql
15) ***:HeidiSQL:http://en.wikipedia.org/wiki/HeidiSQL
16) ***:Usage share of operating systems:http://en.wikipedia.org/wiki/Usage_share_of_operating_systems
17) ***:Download Connector/J:https://dev.mysql.com/downloads/connector/j/5.0.html
18) Anurag Jain:How to connect with MySQL database using JAVA:http://mrbool.com/how-to-connect-with-mysql-database-using-java/25440
19) ***:Arduino UNO: http://www.arduino.cc/en/Main/ArduinoBoardUno
20) ***:Bees Shield: http://www.seeedstudio.com/wiki/Bees_Shield

21) ***:RFID 125 kHz module for Arduino Tutorial:http://www.cooking-hacks.com/documentation/tutorials/arduino-rfid
22) ***:SM125 Firmware V3.0 data sheet:http://www.apdanglia.org.uk/SM125_V30-MANUAL%2051%20PAGES.pdf
23) ***:SM125 sytem user manual:http://www.sonmicro.com/en/downloads/125/um_SM125.pdf
24) ***:RFID tag combo (125 kHz) - 5 pcs: http://www.seeedstudio.com/depot/rfid-tag-combo-125khz-5-pcs-p-700.html
25) ***:EM4100 protocol description:http://www.priority1design.com.au/em4100_protocol.html
26) ***:Wifi module for Arduino "Roving RN-XVee":http://www.cooking-hacks.com/documentation/tutorials/wifi-module-for-arduino-roving-rn-xvee
27) ***:RN-171 802.11 b/g Wireless LAN module:http://www.farnell.com/datasheets/1669946.pdf
28) ***:WiFly command reference, advanced features and applications user's guide:http://ww1.microchip.com/downloads/en/DeviceDoc/50002230A.pdf
29) ***:AltSoftSerial library:https://www.pjrc.com/teensy/td_libs_AltSoftSerial.html
30) Liudr:How to optimize your Arduino memory usage:https://liudr.wordpress.com/2011/02/04/how-to-optimize-your-arduino-memory-usage/

10. Videó lista

31) ***:Map WiFly wireless module serial port to TCP IP port:https://youtu.be/LI0KQlUAzQ8

32) ***:Samsung Wi-Fi smart fridge - Navigating Epicurious application:https://youtu.be/WmgTAbW8Oi4

33) ***:CES 2012 -- LG Smart Fridge:https://youtu.be/rOLEqtWhsw4

34) ***:LG smart refrigerator - Food manager:https://youtu.be/hesFYrXqwyc

Köszönöm kedves témavezetőmnek, Dr. Buchman Attilának, a tanácsokat, tippeket! Sokat segített, amikor elakadtam.
Továbbá köszönöm neki, hogy segített megszerezni a feladat elvégzéséhez szükséges eszközöket!

Köszönöm segítőkész főnökömnek, Rábai Gyulának, hogy rávilágított a Linux tűzfal és a JDBC driver létezésére, ezzel elgördítette az akadályokat a szerverre lépés elől!

Köszönöm a google-nek, hogy a szervert díjmentesen biztosították, és még 2-3 évig biztosítani fogják, bármilyen feladatot is szánok neki a jövőben!

Köszönöm barátaimnak, hogy a barátaim maradtak! Annak ellenére is, hogy a diplomamunkám ideje alatt kevesebbet úsztunk vagy söröztünk.

Köszönöm egész családomnak, legfőképpen Édesanyámnak és Édesapámnak, hogy végig mellettem álltak, és állnak most is!