**Szegedi Tudományegyetem**

**Informatikai Intézet**

# C++ alapú alkalmazás készítése magyarországi statisztikai adatok megjelenítésére

Szakdolgozat

Készítette:

**Forgó Martin**

gazdaságinformatikus Bsc szakos hallgató

Témavezető:

**Dr. Alexin Zoltán**

egyetemi adjunktus

Szeged

2021

**Szakdolgozati tématerv**

**Név:** Forgó Martin

**Szak:** gazdaságinformatikus BSc nappali tagozat

**Neptun kód:** S954ZF

**Téma címe:** C++ alapú alkalmazás készítése magyarországi statisztikai adatok megjelenítésére Qt framework használatával
**Téma címe angolul:** Development of a Qt C++ application for visualization of statistical data
**Témavezető:** Dr. Alexin Zoltán, TTIK, Szoftverfejlesztés Tanszék

**A feladat leírása és célja:**
A szakdolgozatom témájaként a feladat egy olyan alkalmazás elkészítése, mely a Magyarországra vonatkozó statisztikai adatokat grafikonosan jeleníti meg. A cél az, hogy egyszerűen lehessen a különböző statisztikai adatokat megjeleníteni, elősegítve az átláthatóságukat.

A programban egy Magyarország térkép jelenik meg a végfelhasználónak, a megyeközpontokkal. A városokat lineáris interpolációval számolom ki és helyezem el a térképen.

A különböző, Magyarország megyéire vonatkozó statisztikai adatokat adott formátumú fájlban tárolva betöltheti a felhasználó. A betöltés után az adatokat megyékre lebontva jelenítheti meg különböző grafikonokon. A főbb városok a megyével a térképen kiíródnak, melyeket kiválasztva adhatja hozzá a felhasználó a grafikon(ok)hoz.

A rendszer nyitott lesz az adott formátumú adatfájlokra.

A statisztikai adatok betöltését követően lehetőség lesz azok permanens elmentésére adatbázis használatával. SQLite adatbáziskezelő rendszert fogok használni. Ezek után adatbázisból is betölthetőek lesznek a statisztikai adatok. Az adattáblákban tárolt rekordok később módosíthatóak lesznek a felhasználó számára.

**Szakdolgozat elkészítésének ütemezése:**

| **Időszak:** | **Tervezett feladat(ok):** |
|---|---|
| Május | Információgyűjtés: Qt, SQLite, C++ ismeretek bővítése és a tanultak egyeztetése a témavezetővel. |
| Június | |
| Július | Alap program (Qt applikáció térképpel, gombokkal)<br>Legyenek meg az adatfájlok |
| Augusztus | Megyékhez tartozó fővárosok megjelnítése<br>Adatbetöltés és SQLite használata |
| Szeptember | Adatok menthetősége, módosíthatósága<br>Grafikonok megjelenítése |
| Október | Grafikonok megjelenítése |
| November | Szakdolgozat összeállítása formailag |
| December | Szakdolgozat leadása |

## Tartalmi összefoglaló

- **A téma megnevezése:**

  A téma egy olyan asztali szoftver készítése, mely képes adott formátumú statisztikai adatok vizuális megjelenítésére Magyarország megyéire, régióira, nagyrégióira vonatkozólag.

- **A megadott feladat megfogalmazása:**

  A feladat, hogy egy adott formátumú fájlban – formátum alatt itt most a belső szerkezetét kell érteni – lévő Magyarország megyéire, régióira, nagyrégióira vonatkozó különböző statisztikai adatokat egy asztali alkalmazással vizuálisan megjeleníteni lehessen, elősegítve azok átláthatóságát. Cél a statisztikai adatok egyszerűbb és gyorsabb feldolgozásának elősegítése az emberek számára.

- **A megoldási mód:**

  A megoldás C++-ban, multiplatformos Qt framework-kel készült.

- **Alkalmazott eszközök, módszerek:**

  C++-os keretrendszernek a Qt lett választva. Ehhez jött a CMake, mint buildrendszer generátor. Az adatok temporális, majd permanens tárolásához SQLite lett választva a specifikációnak megfelelően. A program tervezéséhez egy félig MVC (Modell-View-Controller) programtervezési minta lett használva.

- **Elért eredmények:**

  Egy olyan desktop program, ami megyékre, régiókra, nagyrégiókra vonatkozólag évenkénti adatokat képes megjeleníteni kétfajta (Piechart, Barchart) diagramon.

- **Kulcsszavak:**

  C++, Qt, SQLite, CMake, Lineáris interpoláció, Diagram

# *Tartalomjegyzék*

# BEVEZETÉS

Napjainkban mindenkinek az életében jelen van az a szó, hogy ”adat”. Az adat, mint fogalom önmagában nem jelent sokat, de ha megadunk neki egyfajta értelmezést, hogy mit jelöl, milyen mértékegységben, milyen időpontban, akkor az az érték, amit eddig csak adatnak hívtunk már egy olyan jelentéssel bír, mely az emberek életének egy nagyon kis szeletét reprezentálja.

Számos szervezet alapult az adatok gyűjtése, tárolása, feldolgozása, közzététele, eladása köré és egyre több az az információmennyiség, amit az emberekről gyűjtenek a különböző szervezetek, weboldalak, alkalmazások vagy éppen maga az adott állam. A KSH (Központi Statisztikai Hivatal) is egy ilyen országos szervezet, mely szakmailag ugyan önálló, de a kormány felügyelete alatt áll. A szakdolgozat alapjául a KSH adatai lettek választva, ugyanis nagy adatmennyiség áll ingyenesen rendelkezésre tőlük és sokkal érdekesebb is valós adatokkal dolgozni, mint kitaláltakkal.

A cél a dolgozattal, hogy ezeket a gyűjtött adatokat ne csak számként és szövegként láthassuk és értelmezhessük, hanem vizuálisan, grafikonokon is, ami egy ember számára sokkal könnyebben feldolgozható forma, mint az írás.

Jelenleg az elkészült szoftver csak magyarországi adatokra lett tervezve, ugyanis maga a térkép, ahol az adatpontok helyezkednek el, egy Magyarország térkép. De az adott adatformátumot betartva ez akár kibővíthető később az egész Földről gyűjtött további adatokkal is.

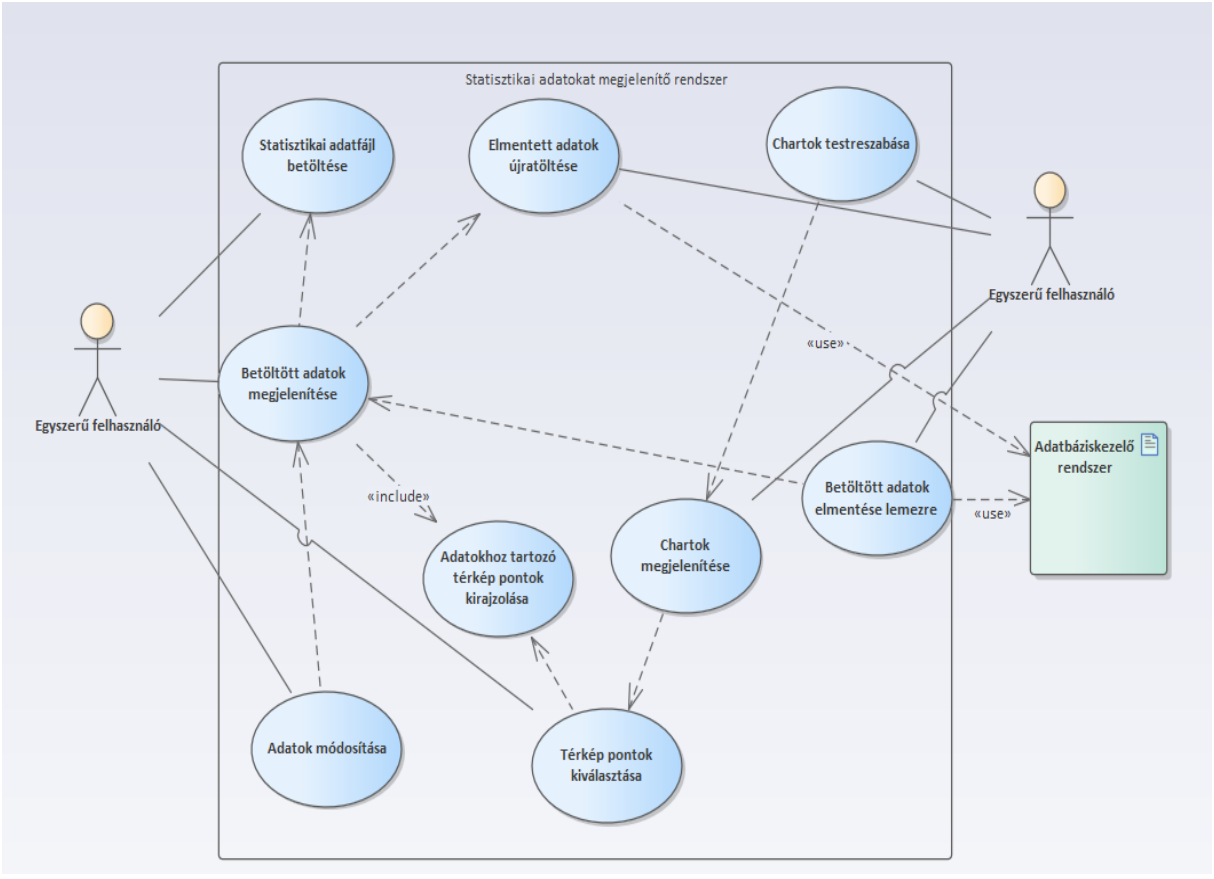
A szoftver a Creative Commons által nyújtott, *Attribution-NonCommercial-ShareAlike 4.0 International* license típus alá lett besorolva. Azaz, bárki szabadon megoszthatja bármely médiumon keresztül és felhasználhatja mindaddig, míg ennek kereskedelmi vonzata nem történik.[1]

[1] https://creativecommons.org/licenses/by-nc-sa/4.0/

# 1. Specifikáció, felhasználói követelmények

A programnak több különböző feladatot kell ellátnia, melyeket általában a megrendelő szokott kérvényezni a fejlesztő csapat felé. Scrum csapatoknál, a Product Owner a felelős az ilyen epic feladatok átvételéért, majd a fejlesztők felé történő továbbításukért, alfeladatokra való bontásukért. Persze a megrendelő megadhat konkrét funkciókat, melyeket szeretne, ha a program ellátna, illetve egyéb kérései is lehetnek melyek konkrét funkciókhoz nem köthetők, hanem inkább a program egészére vonatkoznak, mint nem funkcionális követelmények halmaza. Ezeket a követelményeket, mind a funkcionális, mind a nem funkcionális halmazt kétféleképpen írhatjuk le: magas szinten, természetes nyelven, ami bárki által megérthető, illetve rendszer szinten, melyek a rendszer funkcióit, szolgáltatásait, megszorításait részletesen írják le, de nem elég részletesen ahhoz, hogy a fejlesztési folyamatra ez ne hasson ki zavaróan.

1.1. ábra: Felhasználói szintű Use Case diagram



A Use Case diagram egy UML (Unified Modeling Language) diagram típus. A UML egy szabványos modellező nyelv, ami egységesített diagramrendszerből áll, segítve a vizualizációját, specifikációját, dokumentációját az adott szoftverrendszer elemeinek. De nem csak szoftverekre alkalmazhatóak ezek a diagramok, hanem bármire, ha illik annak kifejezésére.[2]

Az 1.1-es ábrán a szoftver magasabb szintű specifikációja látható. A Use Case diagram actorokból, use case-ekből (használati esetek), kapcsolatokból áll. A szoftver használati eseteit egy határoló vonal veszi körül, ami azon kívül van az a programon kívül esik. Az actorok ilyenek rendszerint. Egy actor egy személyt vagy másik rendszert jelöl, amik interakcióba léphetnek az adott rendszerrel. Jelen esetben egy actor egy egyszerű felhasználó lehet. Egy másik rendszer, ami szintén téglalappal van körülvéve egy valamilyen adatbáziskezelő rendszer, amivel interakcióba léphet a statisztikai rendszer. Adatok elmentésekor és betöltésekor használja a DBMS-t (Database Management System). A use case-ek pedig a rendszer funkcionalitását jelölik. A közvetlen funkcionalitás az, ami közvetlenül az actor tevékenységéhez köthető, pl. adatfájl betöltése. A közvetett funkcionalitás pedig a közvetlen funkcionalitás következménye, pl. a betöltött adatok megjelenítésekor az azokhoz tartozó térkép pontok is kirajzolódnak. Ezt viszonnyal lehet jelölni, ennél a példánál ez az `<<include>>`. A user interakciókat asszociációs vonal jelöli, a függőségeket szaggatott. Például adatokat módosítani csak akkor lehet, ha azok meg is vannak jelenítve. Megjelenítve pedig betöltés után lesznek. A diagram ezekből az egyszerű elemekből építkezik és akkor éri el célját, ha könnyen érthető. A szoftverfejlesztés bármely szakaszában lehet használni ezt az egyszerű diagram típust. Az adott diagram nem részletes, hiszen az adott funkciók bekövetkezése nincsen levezetve, vagy éppen a megjelenítendő chartok típusai sincsennek említve.

A diagramból kivehető még, hogy statisztikai adatfájlok a program bemenete. Ezek az adatfájlok adott formátumúak (formátum alatt itt nem az encoding-ra kell gondolni, hanem a fájl belső szerkezetére). A kimeneti adatok pedig ezeknek a statisztikai fájloknak az adatbázisba történő elmentése, illetve a felhasználó számára lehet még kimeneti adat azok vizualizációja.

A statisztikai adatfájlokra konkrét megkötés nem volt, hogy milyen fajta statisztikára vonatkozzon, van e valami speciális szerkezete, vagy éppen hány szempontot figyelembe véve lett összeállítva a statisztika. Az adatok a KSH oldaláról lettek letöltve és a program

[2] https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/

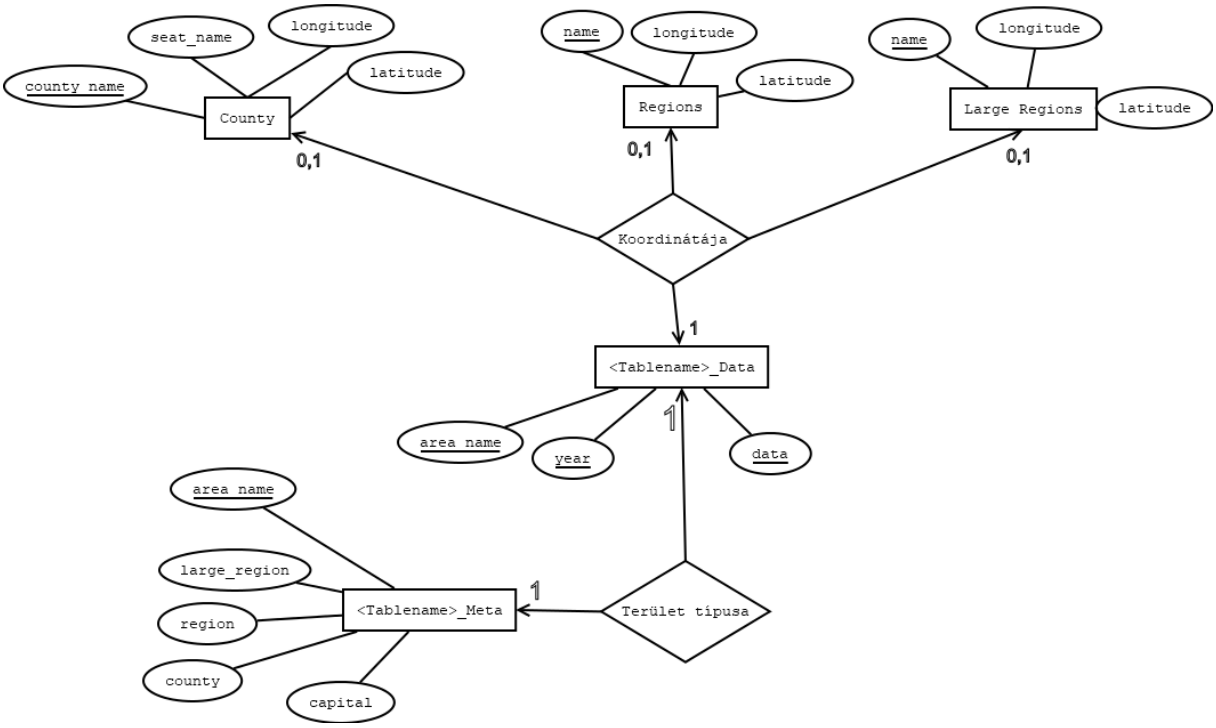
bemeneti adatfájljainak a szerkezete is azokhoz próbál hasonlítani. A KSH többféle szempont szerint statisztikát készít országos szinten, mely statisztikák az oldaláról ingyenesen le is tölthetők. Általában megyékre, régiókra, nagyrégiókra bontva kerülnek fel az adatok, de talán városokra lebontva is lehet igényelni külön a hivataltól. Az adott statisztikákhoz tartozó adatfájlok le lettek töltve xls formátumban, majd azok csv formátumba lettek konvertálva és a csv formátumra lettek alkalmazva a végső belső formátum karakterisztikái, amit a program képes már elfogadni. Jelenleg csak olyan formátumot támogat a program, amihez az ország megyéire, régióira, nagyrégióira vonatkozó statisztikai adatokat éves bontásban szükséges megadni. Tehát egy területhez tartozó adatot adott évre, évekre visszamenőleg.

A diagramból továbbá kivehető még, hogy a térképen adatbetöltéskor ponthalmaznak kell megjelennie az adott adathoz tartozó területet reprezentálva. Egyéb megkötés erre sem volt, a fejlesztőre lett bízva, hogyan valósítja meg. A Föld geo koordinátái lettek felhasználva erre a célra, melyeknek a forrása a Google Maps volt. Egyéb forrásokat is lehetett volna igénybe venni, viszont mivel a Föld görbülete igen csekély Magyarországot nézve, így század pontos adatokra volt szükség, hogy a térképre viszonylag pontosan lehessen illeszteni a területi pontokat. A geo koordináták illesztései a térkép pixel pontjaira lineáris interpolációt használva lettek kiszámolva, ami adott két pont közötti ponthalmazra a görbületet nem számolva (Magyarország területére nézve a görbület nagysága elhanyagolható) viszonylag jó közelítést tud adni a többi pont elhelyezkedését illetően. A program bemenetei közé sorolhatók így a területhez tartozó geo koordináták is, melyek a térképen történő rajzoláshoz szükségesek.

A különböző chartok megjelenítése is a specifikációból ered. Viszont külön a chart típusokra sincsennek megkötések. Lehetett volna például kinézetbeli megkötést, chart típusra való megkötést tenni, de mivel ezek nem történtek meg, a fejlesztő saját elképzelést alkalmazhatott.

A tématervből még kiderül, ami szintén a specifikáció része, hogy SQLite adatbázis-kezelő rendszert kell majd használni. Az SQLite-ból pedig adja magát, hogy az adatok tárolására, lekérdezésére, módosítására az SQL nyelvet kell használni és az adatok relációs adatbázis sémákban fognak tárolódni kis méretű adatbázisfájlokban. Az adatbázis sémákra meghatározott Egyed-Kapcsolat diagramot mutatja az 1.2-es ábra. Mint látható öt adatbázissémát lehet majd létrehozni ezalapján, melyek az adattáblák szerkezetét határozzák meg. A statisztikai fájlok tartalma a redundancia elkerülése érdekében két adattáblában vannak eltárolva, aminek az egyik suffixe _Data, a másiknak _Meta. Az előbbi adattáblához rendelt rekord a területhez tartozó adatokat tartalmazza adott évekre vonatkozólag, míg a

_Meta végződésű táblák rekordjai a terület jellegét tartalmazzák, például, hogy az a terület megye-e. A két fő sémán kívül három másik is szerepet játszik, amik rendre a megyék, régiók és nagyrégiók geo koordinátáit tartalmazzák. Ezeket a fentebb említett térképre mappelés miatt szükséges eltárolni, ezeket használja a program a területek interpolációjához. A kulcsok aláhúzással lettek jelölve. A _Data sémában a terület neve is lehetne kulcs, ha élnénk azzal a feltételezéssel, hogy nem fog két ugyanolyan nevű területű adat rekordként bekerülni a táblába, de mivel egy adott terület a több különböző év miatt bekerülhet többször is, így nem lehet önmagában kulcs a területnév abban a sémában.




1.2. ábra: Az adatbázis EK diagramja

A felhasználói követelmények másik aspektusa a nem funkcionális követelmények. A funkcionálistól eltérően ez a követelmény csoport a program egészére vonatkozik és inkább annak tulajdonságait jelöli. Mint előbb említve lett az adatbázis használata a funkcionalitás szemszögéből nézve, annak nem funkcionális követelménye a gyorsaság lehet, azaz az adatok elmentése, kinyerése viszonylag gyorsan történjen valós időben. Ez konkrétan nem lett lefektetve, mint ahogy más nem funkcionális követelmény sem, mint például a hordozhatóság, könnyen kezelhetőség, átláthatóság, reszponzív megjelenés, de mindezek ellenére a fejlesztési fázisban ezekre a követelményekre is figyelem lett fordítva.

# 2. Felhasznált toolok, technológiák

## 2.1. CMake

Az statisztikai program jelen esetben a Microsoft által kiadott MSVC compilerrel lett fordítva Windows környezetben. A fordítási folyamat során az adott compiler fogja az adott forráskódot, majd object kódra fordítja azt, azaz gépi kódra, ami egy többlépcsős hosszabb folyamat eredménye. Ha a forráskód több fájlban helyezkedik el, akkor úgynevezett translation unitok mentén zajlik a fordítás. Egy translation unit egy, a preprocessor által feldolgozott fájlok tartalma. A preprocessor lényegében egy szövegszerkesztő. Minden neki szóló #-vel kezdődő direktívát, ha képes rá, akkor végrehajtja.

### 2.1.1. Build rendszerek

A build rendszerekre az igény nagyobb projektek esetében jelentkezik, amikor sok fájlról vagy éppen projektről van szó. Konzolból mindig meg kell adni az összes fájlt, amit buildelni szeretnénk, ami fáradtságos munka, nem beszélve arról, hogy vannak olyan fájlok, amiket nem is biztos, hogy újra kell fordítani, ha már egyszer lefordították őket. A build rendszerek a fordítási folyamat automatizálására jöttek létre. Ismertebb build rendszerek például a make, amit UNIX környezetben használnak vagy az Apache Ant, amit inkább Java rendszereknél használnak szélesebb körben és XML alapú. A Microsoft build rendszere az MSBUILD, ami projekt fájlokat használ és a projekt fájlok összefogására egy solution fájlt. A projekt fájlok eredménye vagy valamilyen könyvtár (lib vagy dll) vagy pedig egy futtatható állomány (exe).

### 2.1.2. Build rendszer generátorok és a CMake

Egy build rendszer generátor nem más, mint egy olyan program, aminek egy adott build rendszerhez tartozó build fájlok a kimenete és a saját projekt vagy konfigurációs fájljai a bemenete. Magas szintű sajátos utasításkészlettel rendelkeznek, ami alapján a legenerált build rendszer beállításait lehet elvégezni automatikusan, azaz a generátor ezeket is beállítja nekünk. Fontos még ehhez kapcsolódóan megemlíteni, hogy az ilyen generátorok képesek

akár multiplatformra is build rendszer fájlokat generálni, azaz elég az input fájljaikat egyszer megírni és többplatformú rendszereket képesek ezekből létrehozni.

Ilyen generátor például a Qt-nek a qmake-je, ami a generátor projektfájljainak tartalma alapján Makefile-okat állít elő az adott platformra. Ez is működik multiplatformon, és mivel Qt termék, a Qt alapú projektekhez nagyobb támogatást tud nyújtani. De ettől függetlenül Visual Studio projektfájlokat is ugyanúgy lehet vele generálni a qmake projektfájljainak megváltoztatása nélkül.[3]

Egy másik ilyen generátor, amit a vállalatok is szélesebb körben alkalmaznak a CMake. Ennél a projektnél is ez lett felhasználva. Ez egy nyílt forráskódú program, ami több platformon is használható build fájlok előállítására, azok build folyamatainak kezelésére, a buildelt program tesztelésére, becsomagolására. Platform és compiler független konfigurációs fájlokkal dolgozik. Ezeket a fájlokat, melyeknek alapból CMakeLists.txt a nevük, olyan forráskódot tartalmazó könyvtárakba kell helyezni, melyek egy projektet tesznek ki és az ott található forráskódok alapján állítja elő a kívánt build fájlokat a CMake. Komplex könyvtár hierarchiákat tud kezelni és az sem jelent akadályt számára, ha számos librarytől függ a saját programunk. Továbbá, mivel open source a CMake, saját feature-el egészíthetjük ki, ha éppen erre van szükségünk. Részletesebb információ az [1]-es irodalomban található.

Mielőtt a statisztikai program CMake fájljai ismertetésre kerülnének a 3. fejezetben, legyen egy gyors áttekintés még a CMake névtereit illetően. A CMake-ben a változó az alapvető tárolási egység, amelyben információ tárolható. A változók értékeit stringként kezeli minden esetben, kivéve amikor a stringet egy másfajta típusnak értelmezi a nyelv, de az attól még string marad. Vannak beépített változók melyek általában `CMAKE_`, `_CMAKE_`, `_` prefixszel kezdődnek. Ezek némely parancsoknál automatikusan beállítódnak, de mi magunk is beállíthatjuk őket explicit a `set` paranccsal, illetve törölhetjük a beállított értéket az `unset`-tel. Ezen kívül saját változókat is definiálhatunk ugyanúgy, mint más programozási nyelvekben. A változók beállításakor azok értéke érvényes lesz az adott scopeban, melyben beállításra kerülnek.

Egy scope itt nem csak a megszokott blokk scope, mint ami a procedurális nyelvekben szokott lenni. Egy scope lehet function, directory, global, illetve egy perzisztens cache. Egy `set` vagy `unset` paranccsal az adott változó értéke elérhető lesz az adott scopeban, illetve az alatta levő scope-okban, ha ez a lehetőség nincs külön letiltva. A function scope az egy szokásos függvényen belüli láthatóságot jelöl.

[3] https://doc.qt.io/qt-5/qmake-manual.html

A directory scope egy adott könyvtárban, CMakeLists.txt-ben definiált változók láthatóságára vonatkozik. Ebben az az érdekes, hogy ha egy source tree-ben van több directory, akkor amikor egyik CMakeLists.txt-ből áthívunk az alkönyvtárbeli CMakeLists.txt-be, akkor a magasabb szinten definiált változók átmásolódnak az alacsonyabb szintekre is és ott is elérhetőek lesznek az értékeik.

A globális scope-ban a környezeti változók helyezkednek el, ezek soha nem kerülnek a cache-be, illetve a CMake folyamatban mindenhonnan elérhetőek, egyébként ugyanúgy viselkednek, mint egy átlagos változó.

Végül, ami szokatlan lehet még a directory scope után egy objektum orientált világból érkező programozónak, az a perzisztens cache. Ez nem más, mint egy adott CMake projekt első futtatása során létrejövő txt kiterjesztésű fájl, melyben perzisztens módon (a merevlemezen) megmaradnak az adott CMake változók az értékeikkel több egymás utáni CMake folyamat futtatása alkalmával is. Ez akkor törlődik amikor egy teljes újragenerálás történik a projektfájlok kapcsán, vagy pedig, ha manuálisan töröljük.

A változókra történő hivatkozások kiértékelése egy adott sorrendben történik a scope-ok figyelembevételével. Ha a CMake a futása során egy változó hivatkozásra bukkan, akkor a következő sorrendben keresi annak az értékét a scope-okban: function call stack, directory scope, cache, és ha ott sem talál ilyen nevű változót, akkor üres stringre értékelődik ki a hivatkozás. A globális, környezeti változókra explicit kell hivatkozni, illetve a cache entry-re is lehet külön hivatkozni, hogy ott keressen.[4]

[4] https://cmake.org/cmake/help/latest/manual/cmake-language.7.html#cmake-language-variables

## 2.2 Qt Framework

### 2.2.1. A Qt-ról általánosan

A Qt framework a The Qt Company (korábbi nevén Trolltech) szoftvercég felügyelete és gondozása alatt álló többplatformos programokat támogató keretrendszer, melyet asztali, mobil és beágyazott környezetekben is használnak például az autóiparban, de első ügyfelei között szerepel az Európai Űrügynökség is. A támogatott platformjai közé sorolhatók a Linux, Windows, macOS, Android, IOS és sok egyéb Operációsrendszer rendszer is. A Qt framework fejlesztését a The Qt Company végzi jelenleg, de egy nagyobb közösség is hozzájárul ehhez a munkához a The Qt Project néven, ami számos vállalatból és egyénekből tevődik össze szerte a világon. Továbbá mivel open source license-ek alatt is elérhető, így lényegében bárki hozzájárulhat a fejlesztéséhez. A license típusok, melyek alatt a rendszer elérhető, a kereskedelmi license, nyílt forráskódú GPL2.0, GPL3.0 és a LGPL3.0 license-ek. A rendszer nem programozási nyelvet takar, hanem egy C++ programozási nyelven írt framework-öt, ami az objektum orientált és esemény vezérelt programozást támogatja számos modullal, de nem csak C++ programokhoz használhatók fel ezek a Qt modulok, hanem például python programokhoz is. Alapvetően egy GUI toolkit, de nem csak user interface építését támogatja, hanem adatbáziskezelést, XML és JSON fileok parsolását, network programozást vagy éppen a multithreadinget is.[5]

A build rendszer generátorok támogatják a Qt-t - mint fentebb említve lett a CMake is, amivel a statisztikai program is konfigurálva lett – de a Qt-nek is van saját ilyen generátora vagy más néven front end-je natív build rendszerekhez, a qmake. Lehet vele Visual Studio projekteket, GNU Make fájlokat, vagy akár Xcode projekteket is generálni.

A Qt-nek van saját IDE-je, a Qt Creator, ami Linux, Windows, OS X-en is elérhető. Támogatja a UI tervezést, debuggolást, profiler-ezést és még sok más feature-t is, amivel hasznos, ha egy ilyen IDE rendelkezik. A Designer elérhető külön programként is, ami a UI tervezését és összeállítását támogatja. De UI elemeket is definiálhatunk benne, ha a beépítettek nem elégségesek az igények kielégítésére.

---

[5] https://www.qt.io/company-updates/qt20 , https://doc.qt.io/archives/qt-6.0/index.html, https://en.wikipedia.org/wiki/Qt_(software)

Sokszor szokott igény lenni arra, hogy egy program több nyelven is elérhető legyen, ez a localization (l10n). De pontosan mit is takarhat ez a többnyelvűség? Arra, hogy az adott programot más nyelvű és kultúrájú emberek is használni tudják. Ehhez szükséges a UI szövegeket, dokumentációkat a kívánt nyelvre fordítani. Továbbá, ha a programban szerepel, akkor az időformátumot, billentyűzet sajátosságait, valuta típusát, szimbólumokat, ikonokat, törvényes jogi szabályokat az adott nyelv vagy régió sajátosságaihoz igazítani. A Qt támogatja az internationalization-t (i18n), ami olyan programtervezést jelent, mely a localization-t lehetővé teszi.[6]

A UI felületeket a C++ Widgetek mellett a JavaScripthez hasonló nyelvvel is fel lehet építeni, ez a QML nyelv, ami szintén Qt termék, így talán könnyebb lehet a logikai (C++ kód) szétválasztása a UI kódtól. A QML nyelv a JavaScriptet integrálja, mint scripting nyelv, mellyel egyfajta backendet adhatunk a UI kód mögé. Tehát lehet választani, hogy C++ vagy JavaScript backendet szeretnénk az alkalmazás mögé, ha valaki a QML nyelvet választja a UI elemek definiálásához.[7]

Nem utolsó sorban a Qt nagyon jól, verziók alapján dokumentált és számos fórumja van, ahol kérdéseket lehet feltenni adott témával kapcsolatban. Széles a felhasználói bázisa, illetve más nyelvekben is használható. Viszont ahhoz, hogy más nyelvekben elérhető legyen binding-okat kell hozzá implementálni. Ezek third party termékek és nem a Qt részei. Ilyen például a Python-nál a PyQt binding.[8]

A továbbiakban a Qt három fordítójáról lesz szó nagyobb vonalakban, melyek fel lettek használva - a uic-t kivéve - a statisztikai program elkészítése során.

### 2.2.2. Meta Object Compiler (moc)

A MOC (Meta Object Compiler) egy preprocessornak nevezett fordító program, amit a C++ nyelv kibővítésére használnak. Azért nevezik preprocessornak, mert a tényleges fordítási folyamat előtt kell futtatni, viszont fordítási folyamatot lát el azzal, hogy a megadott macro és osztály definíciók alapján külön forrásfájlba meta object kódot állít elő.

---

[6] Erről bővebben: https://doc.qt.io/qt-5/qtlinguist-index.html

[7] Bővebben: https://wiki.qt.io/Introduction_to_Qt_Quick

[8] Szintén bővebben: https://riverbankcomputing.com/software/pyqt/intro

A Meta-Object System adja azokat a plusz funkcionalitásokat, melyekkel a C++ nyelvet kiegészítette a Qt, de ez a kiegészítés természetesen C++ szerkezeteket takar a háttérben. Ilyen például az objektumok közötti signal/slot kommunikáció, melyről bővebben a következő nagy fejezetben lesz szó, runtype type information kinyerése, vagy a property system.

Az igény egy ilyen compilerre onnan ered, hogy a C++ nyelv natívan nem támogatja az objektumok futási időben történő introspekcióját. Introspekcióval az objektumok metódusainak, a metódusok paramétereinek, adattagjainak futási időben történő típusmeghatározását és egyéb információkat lehet lekérdezni. Bár fáradtságos munkával írni lehetne C++ kódot, hogy ezeket az információkat ki lehessen nyerni, a moc éppen ezt támogatja. A Qt signal/slot és a property system pedig az introspekción alapszik.

A moc tool header fájlokat olvas be. A header fájlok tartalmazhatnak osztály definíciókat melyekben, ha a tool neki szánt macrot talál, akkor előállítja ezekből a már szabványos C++ forrásfájlokat. A generált C++ forrásfájlokban az introspekcióhoz szükséges dolgok már benne vannak. Tehát a moc egy kódgenerátor. A legenerált fájlokat is természetesen majd le kell a C++ fordítójával fordítani, majd linkelni kell az adott projekthez. A macro-k az eredeti fájlokban a moc-nak csupán jelzés, hogy generáljon kódot az adott osztályhoz, illetve rendes szabványos macro-k is egyben, amik fordítás előtt a többi preprocessor direktívával feloldásra kerülnek.[9] Ezért is nem library a Qt, hanem framework, ugyanis nemcsak modulokat és különböző könyvtárakat nyújt a fejlesztők számára, hanem ilyen rendszereket is, mint a MOC, UIC, vagy RCC.[10]

### 2.2.3. Resource Compiler (rcc)

Az rcc vagy erőforrás fordító egy olyan tool, amit erőforrások adott Qt alkalmazásba történő beágyazására használnak. Qt resource fájlokba (qrc kiterjesztésű fájlok) XML-szerűen megadhatjuk az erőforrásainkat, például képeket, amiket használni szeretnénk majd az alkalmazásban. Az rcc a moc-hoz hasonlóan egy C++ szabványnak megfelelő forrásfájlt állít elő, ami majd hozzáfordítódik és linkelődik az adott Qt alkalmazáshoz. A generált forrásfájl

[9] https://woboq.com/blog/how-qt-signals-slots-work.html

[10] https://wiki.qt.io/About_Qt

tartalmazza az erőforrás bájt reprezentációját és ezekután nem lesz szükség a külső erőforrásra már, mivel maga a bináris fogja tartalmazni azt.[11]

### 2.2.4. User Interface Compiler (uic)

A Qt frameworkben minden Widget, ami képes megjelenni grafikusan a képernyőn. A Widgeteket és az azok elrendezéséhez szükséges további osztályokat, például Layoutokat, nem csak kód szinten lehet a programra vonatkozólag definiálni, hanem ui kiterjesztésű fájlokban is. Ezek a fájlok a qrc fájlokhoz hasonlóan XML alapú szintaktikával rendelkeznek. A Qt Designer is képes ilyen fájlokat generálni, de ott az összeállításuk automatikus. Ha a UI elemeket összepakoljuk a Designerben, akkor az elkészíti számunkra a ui fájlokat automatikusan. A többi Qt compilerhez hasonlóan a uic is szabványos C++ forrásfájlokat generál a ui fájlokból, amiket a programunkkal kell egyidőben fordítani. Viszont itt szükséges a generált fájlok include-olása, mert azok header fájlok. Include nélkül a UI felületet reprezentáló generált osztályt nem is lehetne elérni, de include után használhatóvá válik, és akár leszármaztatni is lehet belőle, ha épp azt kívánja meg a projekt. A statisztikai programnál nem voltak ui fájlok használva, mert kód szinten jobban átláthatóbb és kezelhetőbb, hogy mi, hogyan történik, mint egy generált fájl esetében, nem beszélve azokról az esetekről, amikor olyan funkcionalitást volt szükséges az osztályhoz hozzáadni, amit nem lehetett volna Qt Designerben megvalósítani. Továbbá a Designer nem is tartalmaz minden Widgetet, amik a libraryk-ból elérhetőek. Bővebben a [4]-es irodalomban a uic-ről és a designer-ről.

## 2.3. SQLite

Az SQLite C nyelvben íródott könyvtár, ami az SQL nyelvet támogatja teljeskörűen és egy adatbázis motort biztosít is hozzá, hogy az SQL nyelvi elemeit használni lehessen. Az SQLite a leggyakrabban használt adatbázis motor a világon. Használják a mobilokban és a legtöbb számítógépben is egyaránt. Maga a termék közkincs, azaz bárki szabadon felhasználhatja akár magán akár kereskedelmi célra, szerzői jog nem védi. Tulajdonképpen egy beépített (embedded) SQL adatbázis motorról van szó, mely nem kliens – szerver alapú. A relációs

[11] Bővebben az rcc-ről és a Resource systemről: https://doc.qt.io/qt-5/resources.html

adattáblákat, sémákat merevlemezen tárolja kis méretű fájlokban. A fájlok formátuma úgy van kialakítva, hogy a többplatformúságot támogassa. Nem számít, hogy big-endian vagy little-endian az adott rendszer, hogy 32bites vagy 64, az adatbázis fájlok hordozhatósága megoldott, különböző rendszereken lehet ugyanazt a fájl használni.[12] A statisztikai program zavartalan működése érdekében a PATH környezeti változónak tartalmaznia kell az útvonalat az SQLite oldaláról letölthető sqlite3 binárishoz.

Ahogy fentebb említve lett, a rendszer közkincs, a forráskód is elérhető bárki számára, de van, amikor egy cég mégis license-et igényel, mert garanciális jogot szeretne például vagy mert a cég ügyvédjei ezt tanácsolják valamilyen oknál fogva. Örökös license és tulajdonjogot lehet vásárolni, ami semmi előnnyel nem jár, csak jogilag lehet szükséges. Ami még kereskedelmi jellegű szolgáltatás lehet, az a supportja, és minél profibb szakmai segítséget igényel az adott vállalat, annál drágább az éves díja ennek a szolgáltatásnak, akár évi 25millió Ft is lehet.[13]

Az SQLite fejlesztői egy nemzetközi csapat, akik teljes munkaidőben dolgoznak rajta és profi szinten fejlesztik tovább, de nyílt forráskódú szoftver lévén bárki hozzáférhet és fejleszthet egyéni funkcionalitást hozzá. Továbbá a dokumentációja nagyon részletes ahhoz, hogy bárki elkezdje használni.

Az SQLite-hoz tartoznak bővítmények is, mint például az FTS, ami a Full Text Search funkcionalitást támogatja virtuális táblákkal.[14] Ami a tesztelését, megbízhatóságát illeti, az SQLite core az, aminek 100%-os branch lefedettsége van és ezt milliós nagyságrendű teszteset számmal érték el, habár ennek ellenére hibák ritka esetekben még előfordulhatnak. Bővebben a [2]-es számú irodalomban lehet róla olvasni.

### 2.4. Egyéb felhasznált toolok

Ide sorolható a git, mint verziókövető rendszer, mely a program fejlesztési ciklus elejétől kezdve szerepet játszott, mint fejlesztői eszköz. Ez egy ingyenes, nyílt forráskódú osztott verziókövető rendszer. Nemcsak programozás támogatására lehet felhasználni, hanem bármilyen fájl verzióinak követésére, akár egy txt fájlhoz is. A legfőbb különbség a git és a

[12] https://www.sqlite.org/about.html

[13] https://www.sqlite.org/prosupport.html

[14] https://sqlite.org/fts5.html

többi verziókövető rendszer (például Subversion, ClearCase) között az, hogy mit tekintenek adatnak. Ez a többi rendszer úgy tekint az adatra, mint egy kezdeti adathalmaz és a verziók készítése során, az azokhoz az adatokhoz hozzátett változtatásokra, ezért ezek a rendszereket másnéven delta-alapú verziókövetőknek hívják. A git egy commit alapú rendszer, ami másként tekint az adatokra. Minden egyes commit során a git készít egyfajta pillanatfelvételt a working directory tartalmáról és az egészre vonatkozólag egy referenciát vagy hash-t tárol el, amire hivatkozva az adott fájlok állapotát vissza lehet kapni. Azért lett használva verziókövető, hogy a különböző állapotait a programnak nyomon követni lehessen, illetve nehogy véletlen elvesszen bármi adat egy esetleges rossz módosítás esetén. A repository egy remote repository-ba is fel lett töltve a GitHub-ra, hogy bárhonnan elérhető legyen, ahol van internetkapcsolat. A gitről bővebben a [3]-as irodalomban van szó.

Még egy tool, ami használva lett, mégpedig a program exe fájljának kinézetéhez, az ImageMagick nevű program. Ez adott kép formátumból (például png) készít ico formátumot, ami az exe megjelenítéséhez szükséges formátum és amit egy rc formátumú fájlban kell megadni, mint az alkalmazás ikonját.[15]

A különböző diagramok a Dia nevezetű ingyenes programmal és az Enterprise Architect programmal készültek. Ezek vizuális tervező eszközök, amik a szoftverrendszerek sémájának, architektúrájának tervezését, megjelenítését támogatják egészen a felhasználói szintű részletességtől a fejlesztők által értelmezhető részletességig.

# 3. A statisztikai program részletes ismertetése

## 3.1. A program CMake projektje

A CMake projekt egy vállalati programnál előforduló projekt méretéhez képest kisebb, de így is a lényegi nyelvi elemeket és struktúrákat tartalmazza, ami egy ilyen projekt elkészítéséhez szükséges. Két CMakeLists.txt fájlból áll, az egyik a projekt fő könyvtárában, a másik egy alkönyvtárában vannak. Azért lett így, mert ez egy használható szerkezet lesz egy esetleges bővítésnél is. A főkönyvtárbeli konfigurációs fájl olyan alapvető dolgokat, paraméterbeállításokat tartalmaz, mely az összes többi alkönyvtárában is igaz lesz. Fentebb

[15] https://doc.qt.io/qt-5/appicon.html

pedig ahogy említve lett a directory scope miatt ezek a beállítások az összes alkönyvtárban is láthatóak lesznek az értékeikkel együtt (hacsak felül nem definiálódnak).

A szokásos `cmake_minimum_required` paranccsal kezdődik a fájlok első sora. Ezzel lényegében a projektnek meg lett határozva, hogy melyik CMake versiont használja mely policy-val. Csak ezután használható a többi parancs, hisz van olyan parancs, ami egy adott verzióban nem is létezik, de a CMake is így tudja, hogy milyen parancskészlettel dolgozhat.

A főkönyvtárbeli konfigurációs fájlban a `project` paranccsal lett elnevezve a projektem. Majd be lett állítva néhány tulajdonság, amik majd átkerülnek az alkönyvtárakba, jelen esetben az egyetlenbe. A beállított tulajdonságok a c++ szabvány, a build type, a bin és library kimeneti mappák helyei, majd pedig mivel Qt alapú a program, a Qt egyes paraméterei is.

A Qt meta object compilere (MOC), a user interface compilere (UIC) és a resource compilere (RCC) be lett állítva automatikusra, így, ha ilyen fájlokat a projektben külön fordítani kell – például mert egy QObject leszármazott saját osztályt minden esetben a moc-al fordítani kell – akkor ez automatikusan megtörténik. A `CMAKE_INCLUDE_CURRENT_DIR` változó azért lett beállítva, mert az auto MOC által generált fájlok a build könyvtárba kerülnek majd és azokat is szükséges include-olni a projekthez. A `CMAKE_CXX_FLAGS`-hez azért lett hozzáadva a `Qt5Widgets_EXECUTABLE_COMPILE_FLAGS` változó mert így az `-fPIE` flaggel fog történni az exe buildelése.[16] A PIE a Position Independent Executable rövidítése, ami annyit jelent, hogy az exe-t tetszőleges memóriacímre lehet betölteni ezután a program indulása során. Minden Qt modul rendelkezik ilyen flaggel, de ezeket csak akkor érdemes beállítani, ha exe készül, nem pedig valamilyen library (azoknak az `-fPIC` falg lesz szükséges erre a célra).

A `find_package` parancsot is támogatja Qt esetében a CMake. Ezzel az aktuális projektnek meg lehet adni a további függőségek binárisainak, include könyvtárainak a helyét. A parancsot kétféleképpen lehet használni. Vagy egy, már más által létrehozott package keresésére vagy egy saját másik CMake projekt build fájljainak a keresésére. Az első esetben egy ilyen parancsnak kell egy külön Find<package_name>.cmake file valamelyik könyvtárban, amit a `CMAKE_MODUL_PATH` tartalmaz. A második esetben egy <name>Config.cmake-et hoz létre a CMake a saját projekt buildelésekor, és ha arra dependál

[16] `-fPIE` flag megadása GNU fordítóknál szükséges, a Visual Studio-nál nem kell beállítani, mert alapból ilyet készít.

az adott projekt, akkor <name>_DIR cache változóban meg kell adni az elérését a config fájlnak, amire dependál. A Qt modularizálva van és a CMake minden modul keresését támogatja a `find_package` paranccsal. A statisztikai projektben pedig ez lett alkalmazva épp a Widgets, Charts, Sql modulok megkeresésére.

Az `add_subdirectory` paranccsal a megadott alkönyvtár CMakeLists.txt fájlja került futási állapotba. Ebben az alfájlban már a specifikusabb dolgok lettek beállítva. Például a generált build fájlok milyen fájlokat használjanak fel a megadott exe buildeléséhez, milyen include directory-kat használjanak, milyen további könyvtárakat (ezek azok lesznek jelen esetben, melyeket a `find_package`-el meg lettek keresve) linkeljen az exe-hez. Az adott erőforrások is hozzá lettek adva még a projekthez, amit majd a Resource Compiler fog cpp fájlokra buildelni automatikusan, mivel az `AUTORCC`-t be lett állítva egy könyvtár szinttel feljebb.

A targethez (exe) linkelendő, include-olandó fájlok target property-vel lettek beállítva. Ha a parancs elől a target_ prefix ki lett volna hagyva, akkor az összes targetre értelmezte volna a beállításokat, így pedig csak ahhoz, ami specifikálva lett. A target propertyk továbbá lehetővé teszik, hogy az adott targetnél a linkelendő, include-olandó elemeket egy adott scope-hoz rendeljük. A scope lehet `PUBLIC, PRIVATE, INTERFACE`. Jelentésük pedig: az `INTERFACE`-nél csak a jelenlegi targetre dependáló targetnek kell linkelnie, include-olnia a megadott elemeket, `PUBLIC`-nál a dependálónak és a saját targetnek is, `PRIVATE`-nál pedig csak a saját targetnek kell azokat használnia. Ennél a projektben `PRIVATE`-re lettek állítva, minél szűkebb scope-ra, hogy a felesleges függőségek elkerülhetők legyenek.

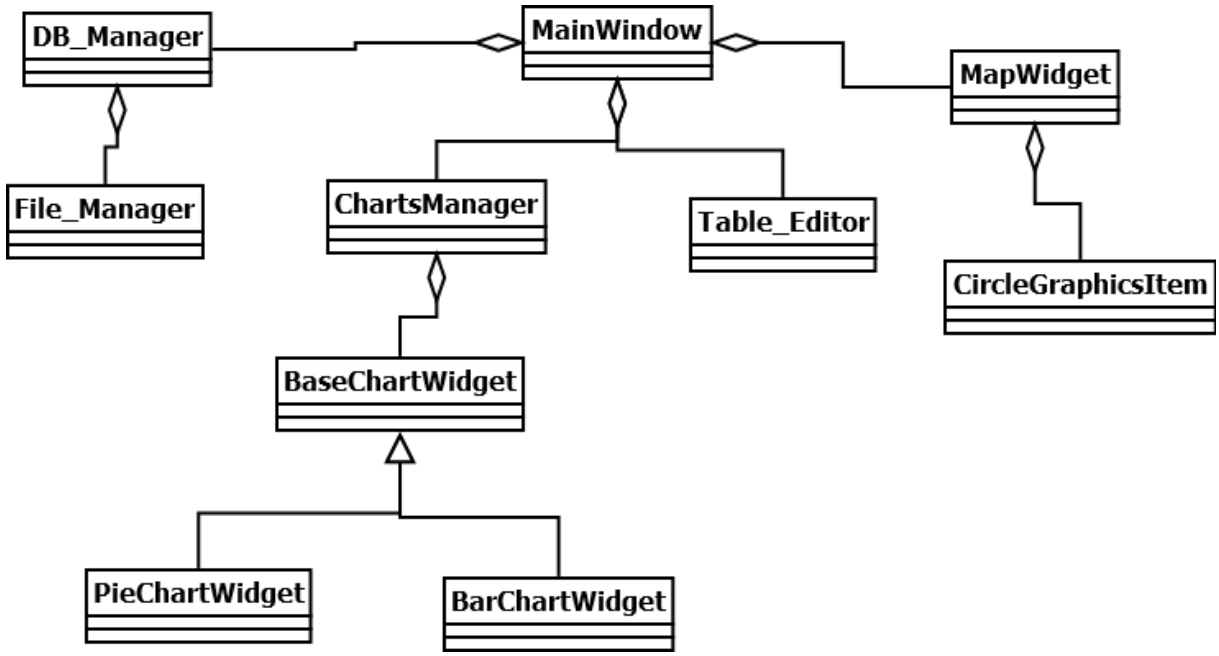
Ez a kis CMake projekt ennyi elemből áll összesen, viszont ami a nagyszerű benne, hogy a megírásuk után felhasználhatók lennének ezek a konfigurációs fájlok más platformokon is (esetleges minimális módosítást téve) és ugyanúgy működne a program azokon is.

### 3.2 A programban használt MVC-szerű minta

Az MVC programtervezési minta a megjelenítés és tárolás funkcionalitások szeparációján alapszik és tervezésbeli rugalmasságot nyújt használójának. Az MVC-ben három főbb objektum létezik a program futása kezdetétől a végéig. A model objektum, vagy alkalmazás réteg, a megjeleníteni kívánt adatokat és az azokon végzett műveleteket reprezentálja a UI réteg számára. Ez egy közvetítői réteg a UI és az adatforrás (ami egy fájl vagy adatbázis

például) között. Ugyanis a UI semmit nem tud az adatok belső tárolásáról és az adatforrás, sem tud semmit a UI-ról, ami megjeleníti. A view vagy UI réteg az adatok képernyőn való megjelenítéséért felelős. A controller pedig egy közvetítői réteg a view és a model között, illetve úgy közvetít, hogy user interakció során a kérés ide fog befutni először, tehát definiálja, hogy a UI hogyan kezelje az interakciókat. Továbbá a controllert szokták még használni a model vagy view futási időben történő lecserélésére, ha szükséges. Az MVC előtt ezeket a rétegeket rendszerint keverve használták, az MVC vezette be ezek szeparációját növelve az alkalmazások rugalmasságát.

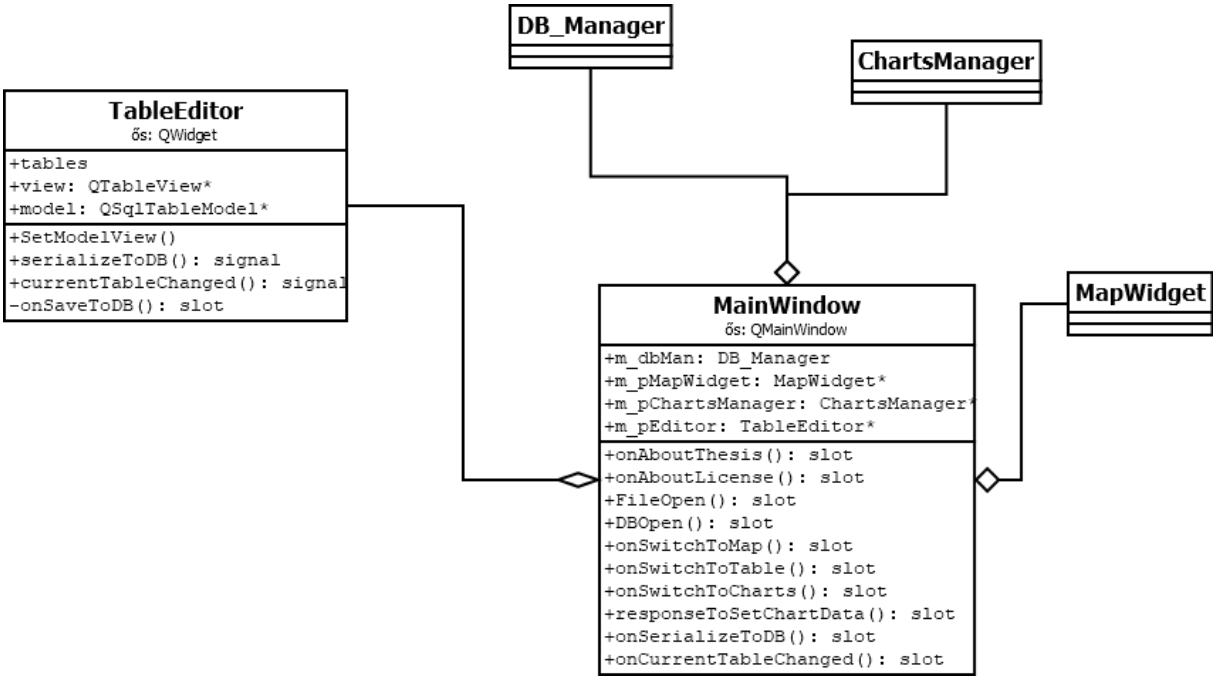
A programban a UI réteg és a model el lett különítve egymástól, de a controller szerepe a UI-ba lett beletéve, ezért MVC-szerű alapokra épült a program. A UI réteg tartalmazza a model réteg manager-jeit, mint az adatbázis manager, file manager. Ezeken keresztül történik a fájl kezeléssel és adatbázis kezeléssel a kommunikáció. Továbbá a UI réteg egyéb UI osztály kezelőkkel is kapcsolatban van, mint a `ChartsManager`, `Table_Editor` vagy `MapWidget`. A 3.2.1-es ábrán a program főbb osztálydiagramjai láthatók. Ezek az osztályok a továbbiakban egyesével végig lesznek mutatva, elmagyarázva a főbb feladataikat és a hozzájuk tartozó főbb Qt funkcionalitásokat is, amiket használnak. A soron következő osztálydiagramok nem tartalmazzák a ténylegesen implementált összes funkcionalitást, csak annyit, amennyi a főbb feladatuk megértéséhez szükséges.




3.2.1. ábra: A program magas szintű osztály hierarchiája

### 3.3. A program fő ablaka

A `MainWindow` képezi a program magját, ez a window látszódik a program elejétől a végéig ezzel alapot adva a UI rétegnek. A program indulása során jön létre az adatbázis managerrel együtt és inicializálja rögtön a UI felületet a menürendszerrel együtt. Ahogy a 3.3.1-es ábrán látható, egy kompozit objektum a `MainWindow` és a managereket tartalmazza, illetve további widgeteket is. A tag objektumaival így közvetlen hívásokkal tud kommunikálni, amit az objektumok interfésze biztosít számára. Ami még feltűnő a funkcióit nézve, hogy mindegyik egy slot. Ez azért van, mert a többi objektum a `MainWindow`-val ellentétben nem közvetlen hívásokkal, hanem a framework signal/slot mechanizmusát felhasználva signal-ok mentén kommunikál. A slotok, ahova a kérések befutnak, pedig azért lettek private láthatóságúak, mert ezeket a slotokat csak a `MainWindow` használja fel feladatai ellátására. A következőkben ez a mechanizmus, mely a Qt központi része és ami átszövi ezt a programot is, lesz nagyvonalakban bemutatva.




3.3.1. ábra: A MainWindow és TableEditor osztályok

#### 3.3.1. A felhasznált signal/slot mechanizmus

A signal és slot kommunikáció a Qt egyik központi mechanizmusa és a már korábban említett Meta Object System a mozgatórugója. Ez nem más, mint az objektumok közötti

kommunikáció megvalósítása. Például, ha egy gombra kattint valaki, akkor szeretné, hogy valami történjen, mondjuk bezáródjon egy ablak. Más framework-ök ezt callback függvényekkel oldják meg, azaz függvénypointert adnak át egy másik függvény argumentumaként, mint callback, majd a másik függvény meghívja a callbacket, ha az számára alkalmas lesz. Például ha a callback egy close() metódusa az alkalmazásnak, akkor ha ezt, mint callback átadnánk egy closeButton-nak és annak a button-nak az event handlerét felülírnánk, hogy ezt a callbacket hívja meg, ha kiváltódik a click event, akkor fáradságosabb munkával hasonló eredményt lehetne elérni, mint amit a signal/slot kommunikáció nyújt.

A signal nem más, mint egy üzenet, egy objektum publikus metódusa, amivel, ha jelezni szeretne valaki egy másik (vagy ugyanazon) objektum metódusa felé, meghívja azt vagy másnéven emittálja. Vagy kiváltódhat még adott esemény bekövetkezésekor, például, ha egy gombra kattint valaki, de igazából ott is végtére emittálva lesz. A signal egy broadcast üzenet, azaz egyirányú kommunikáció és nem foglalkozik vele az emmitáló objektum, hogy ki fogja épp az ”adást”, ha kiváltódik. A signal-ra slot-okat-ot lehet kötni, erre szolgál a `QObject`-ből eredő `connect` metódus, de meg is lehet szüntetni a kapcsolatot a `disconnect`-tel. Továbbá a signalra másik signalt is lehet kötni, ami továbbítja az eredeti signalt.

A slot egy objektum tag metódusa, bármilyen láthatóságú lehet és ha egy hozzá társított signal kiváltódik, akkor ez a slot metódus meghívódik. A programban is pontosan ez történik, amikor egy objektum a fő ablakkal szeretne kommunikálni. De mivel a slot szabványos C++ metódus - a signalal ellentétben, aminek az implementációját a moc generálja – így bárhogy meghívható külön is, jelen esetben a fő ablak is meghívhatja, mint külön tag metódusát.

A signal és slot mechanizmus típusbiztos, ugyanis a szignatúrájuknak meg kell egyezni, azaz a signalba helyezett argumentumok a slot-nak fognak átadódni. Illetve a slotnak megengedett, hogy ignoráljon bizonyos argumentumokat a signal argumentumai közül.

A connection-nek többféle típusa van, de a típusai közül a queued connection használja az event loop-ot. Ezzel ellentétben egy sima connection-nél pedig a slot azonnal végrehajtódik, ha a hozzá tartozó signal kiváltódik. Az adott signalhoz, ha több slot-van kapcsolva, akkor mindegyik slot végrehajtódik egymás után a connect-ek sorrendjében.[17]

A connectálást többféleképpen meglehet valósítani, de függvénypointer alapú szintaxissal lett írva a statisztikai programban, ugyanis a szignatúra mismatch-ek így már fordítási időben kiderülnek. A szintaxis a következő: `connect(obj1-re pointer, &OBJ1::signal, obj2-re pointer, &OBJ2::slot)`. Egy érdekes jelenség

[17] https://doc.qt.io/qt-5/signalsandslots.html

viszont bajt okozhat ebben a függvénypointeres szintaxisban, ami előjött több helyen a statisztikai programban is, mégpedig a signal overriding.

Annak ellenére, hogy a signal-t a moc implementálja, a saját programkódban a signal deklaráláskor még a programozónak kell biztosítani hozzá a deklarációt, ami lehet egy overload is, azaz ugyanazon nevű metódus más paraméterlistával. A függvénypointeres változatban paraméterlista nincs, így overload esetén több ugyanolyan nevű metódus létezik, és név alapján nem fog tudni a fordító program különbséget tenni közöttük. Mi lehet a megoldás? Castolni szükséges a függvénypointert. A függvénypointer igazából egy pointer to member, ami a függvény kezdő memóriacíme az osztályon belül. Ez a pointer - mivel osztályon belüli címre mutat, az objektumok pedig az osztály példányai – bármelyik objektumra alkalmazható. Viszont overloadnál nem lehet megállapítani csupán név alapján melyik osztályon belüli címre mutasson a pointer, ezért szükséges a cast, ahol a paraméterlista megadásával már egyértelmű lesz a memóriacím. Ha erre a memóriacímre castoljuk az ambiguous függvénypointert, akkor a megfelelő tag függvényt kapjuk. Például a programban egy `QComboBox`-nak a `currentIndexChanged()`-je, ami integer paramétert vár, egy overloadolt metódusa az osztálynak. És ha például a Chart osztályban évet változtatunk a ComboBox-on keresztül, akkor a következő connectet kell írni: `connect(m_pYearCombo,static_cast<void(QComboBox::*)(int)>(&QComboBox::currentIndexChanged),this,&PieChartWidget::onComboYearChanged)`. A cast típusa pedig statikus, mivel fordítási időben lesz rá szükség, nem pedig futáskor.

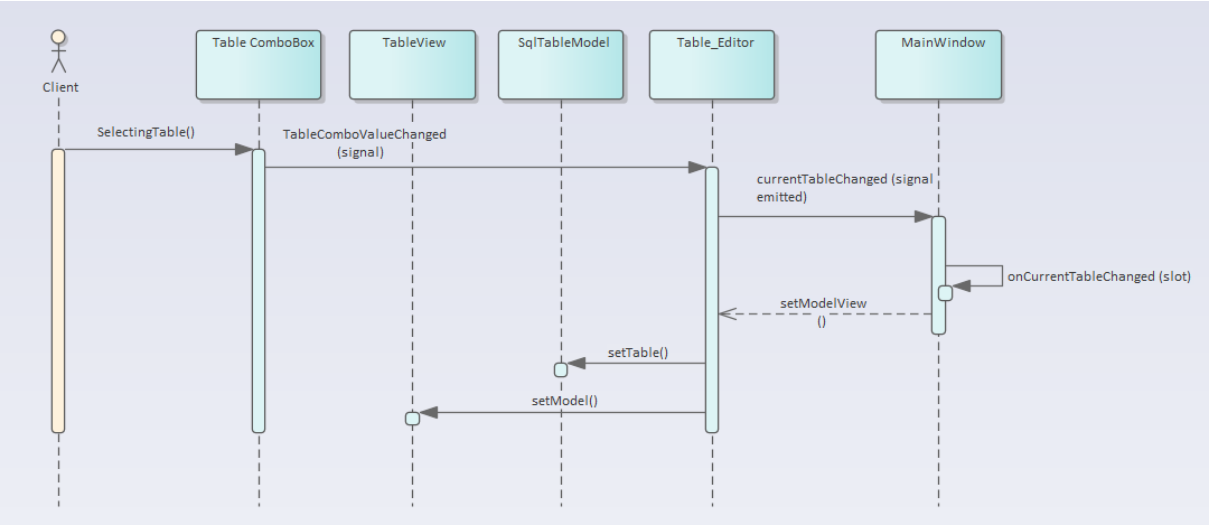
Ha a `SIGNAL` és `SLOT` macrokat használná a connect, akkor minderre a castra nem lenne szükség, ugyanis itt a teljes függvényt paraméterlistával kell megadni, ami egyértelművé teszi a helyzetet, viszont ahogy említve lett az csak futási időben derül ki ennél a változatnál, hogy a signal és a slot kompatibilis-e egymással.

### 3.4. A program tábla nézete

A fő ablakban megjelenített tartalom három nagyobb csoportba sorolható, amik a betöltött statisztikai adatokat különböző nézőpontokból jelenítik meg, ezek a csoportok a nézetek vagy view-k. A program indulása után alapból a térkép vagy map nézetet mutatja, de ez csak egy üres térképet mutat adatok nélkül. Csak a statisztikai adatok betöltése után használhatók a view-k. A különböző nézeteket a program fő ablaka tárolja el és jeleníti meg egy

QStackWidget-ben, ami azt a funkciót szolgáltatja, hogy egyszerre csak egy nézet jeleníthető meg a központi ablakban.

A betöltött adatokat meg lehet tekinteni, módosítani lehet rajtuk, majd el is lehet menteni a merevlemezre azokat egy adatbázis kapcsolaton keresztül. Ezeket a lehetőségeket a program tábla nézete biztosítja, amit a 3.3.1-es ábrán a Table_Editor osztály reprezentál. A Table_Editor tartalmazza az adatbázis táblákat, amik az adott adatbázis kapcsolaton keresztül érhetők el számára. Az adott adatbáziskapcsolattal a kommunikációt mindig a MainWindow biztosítja, így, ha az editorba szeretnénk adatot megjeleníteni ezt egy signal-al jelezhetjük, amire a MainWindow fog választ adni. Az adatok merevlemezre történő mentését is signallal jelezhetjük, amit majd az adatbázis-kezelő objektum fog kiszolgálni. Ezt a fajta objektum kommunikációt a 3.4.1-es ábra szemlélteti. A user kiválasztja a kívánt táblát (ez csak adatbázisból történő adatbetöltéskor opció), és a QComboBox ezen signálját a Table_Editor saját signal-jára kötve továbbítódik a kérés a MainWindow slot-jának. A Table_Editor signal-ja, csakúgy, mint az összes többi objektum signal-ja az adott objektum interfészéhez tartozik. Ezt használja fel a főablak a connection létrehozására. A MainWindow feldolgozza a kérést a sloton keresztül, majd beállítja a model és view objektumokat az editorban. A model és view objektumok a Qt model/view programtervezési mintáját implementálják, és ezzel van megoldva a QComboBoxban kiválasztott adattáblabeli adatok táblázatban való megjelenítése.




3.4.1. ábra: Adattábla kiválasztása az editorban

### 3.4.1. A Qt model/view-ja

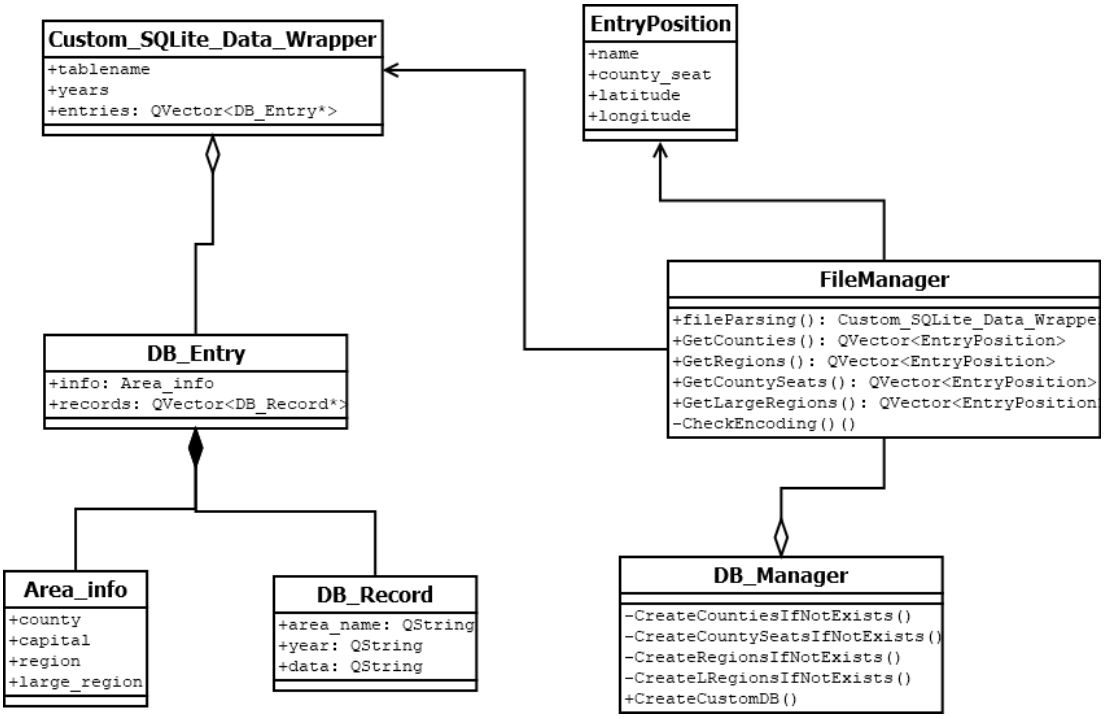
A Qt-ban a model/view biztosítja a különböző program rétegek szeparációját. A model a felelős az adatforrás elemeinek az adott view számára történő biztosításukért, mint adatforrás és view közötti köztes réteg. A view és az adatforrás ugyanis nem tud egymásról semmit sem. Ez a különböző rétegek rugalmasságát segíti elő és mondjuk, ha van egy model, akkor azt több view-nak oda lehet adni, így a sok különböző megjelenítési módhoz elég egyszer megírni a megjeleníteni kívánt adatot, amit a model zár egységbe. Hasonlóan ugyanazon view több különböző model-hez tartozó adatforrások elemeit képes ugyanazon módon megjeleníteni. A model/view az MVC kompaktabb változatának mondható. Ha a view és controller objektumokat összekombináljuk, akkor az eredmény a model/view minta. Ugyanazon a pilléren alapul, mint az MVC és ugyanúgy az adatok tárolása el van szeparálva azok megjelenítésétől, de egyszerűbb architektúrát biztosít az MVC-vel szemben. A model/view tervezési mintához három csoportot lehet rendelni: model-ek, view-k és delegate-ek. Qt-ben mindegyik csoportnak absztrakt interfészei vannak és azokat implementálni szükséges a speciálisabb típusoknál, ha a mintát használni szeretnénk. A view egyébként a model által biztosított indexek alapján kérdezi le a megjeleníteni kívánt adatokat a model-től és annak egyéb interfész metódusait is használja, ezért van az absztrakció, hogy az interfészek biztosítva legyenek a rétegek számára. A három réteg a signal/slot mechanizmust használja a kommunikációra. Bővebb leírás és példák a [4]-es irodalomban találhatóak.

A model és view-n kívül a harmadik csoport a delegate. A delegate felelős az adat elemek view-ra való rajzolásáért, illetve, ha azok az elemek szerkeszthetőek, akkor a delegate helyez egy editort a view réteg fölé. A szerkesztett elemeket a model-nek és view-nak elküldi.[18]

Mivel a model réteg specializálva lett már Sql tábla model-re is, azaz olyan modelre, ami egy SQL alapú adatbázissal tartja a kapcsolatot, így külön osztályt erre nem kellett már létrehozni, elég volt a meglévő SQLite-os adatbáziskapcsolatot megadni a specializált `QSqlTableModel` model konstruktorában. A view pedig a táblázatokhoz használt `QTableView` lett. Az alap delegate-en sem kellett változtatni, az adatbázis elemeinek szerkesztését és megjelenítését teljes mértékben támogatja a view által biztosított delegate is.

[18] https://doc.qt.io/qt-5/model-view-programming.html#delegate-classes

### 3.5. A statisztikai fájlok betöltése és a FileManager




3.5.1. ábra: A fájl manager osztálydiagramja

A File Manager felelős a különböző funkcionalitásaival a statisztikai fájlok parsolásáért, adatstruktúrákba történő mentésükért. A kezelő osztálydiagramján, ami a 3.5.1-es ábrán látható, szembetűnő, hogy az adatbázis-kezelő tartalmazza tulajdonképpen a fájlkezelőt. De ez érthető, mivel az használja a fájlkezelő által kreált struktúrákat. Így például az adatbáziskezelő `CreateRegionsIfNotExists()` metódusa a fájlkezelő `GetRegions()`-át hívja a lemezre menteni kívánt adatok eléréséhez.

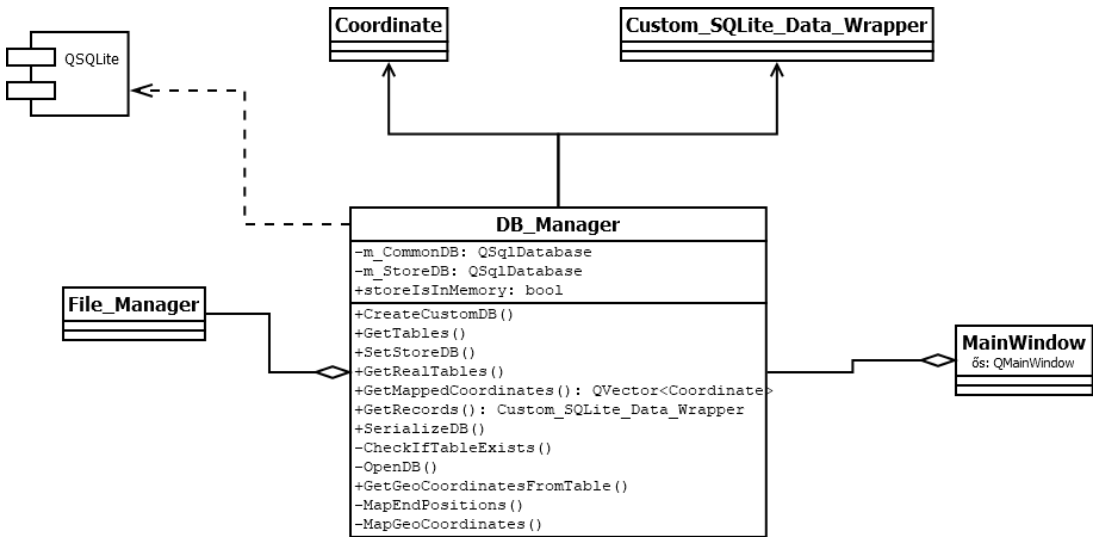
A FileManager osztály az adatok parsolásakor különböző struktúrákat használ, hogy azokba mentse a beolvasott adatokat. Ezek a struktúrák az 1.2-es diagramon szereplő sémáknak megfelelően épülnek fel. A bevezetésben említett _Meta adatoknak van az `Area_info` struktúra, míg a _Data rekordok struktúrája a `DB_Record`. Ezeket a szerkezeteket csomagolja be a `DB_Entry`, ami lényegében egy adott területnek az összes megadott évre vonatkozó adatait tartalmazza. A több különböző terület becsomagolásáért a `Custom_SQLite_Data_Wrapper` osztály felel, ami az egész beolvasott statisztikai fájl adatait tartalmazza az évekkel és táblanévvel együtt.

Egy ilyen bemeneti statisztikai fájl a KSH oldaláról letölthető fájlok szerkezetéhez hasonlít[19] (miután azok csv formátumra vannak konvertálva). Az első sor tartalmazza a tábla nevét, ami jó ha nem tartalmaz szóközt és alul vonást, a második sor az éveket a csv formátumnak megfelelően ';'-vel elválasztva. A harmadik egy olyan sor, amit átugrik a program jelenleg, de ez van fenntartva az adatok megnevezésére, hogy például "ezer fő" stb. És a negyedik sortól következnek a területi egységek nevei, típusai és az adatok. Ha a fájlban valahol `END` van, vagy egyszerűen végére érünk, akkor a beolvasás leáll. A FileManager `parse()` metódusa eszerint a logika szerint parsolja a fájlt `Custom_SQLite_Data_Wrapper` struktúrába és adja oda az adatbáziskezelőnek.

A statisztikai fájlokon kívül, további segéd fájlok léteznek szintén csv formátumban a területi adatokhoz és azok geo koordinátájukhoz kapcsolódóan. Ezeket eltárolni a térképre történő interpoláció miatt szükséges. Ezen adatok az `EntryPosition` struktúrába kerülnek a fájlok parsolása során.

Továbbá még egy feladatot ellát a fájlkezelő, mégpedig a fájlok encoding validálását, hogy azok UTF-8 karakterkódolási eljárással vannak-e eltárolva. Ugyanis szükséges az ékezetet tartalmazó betűk kezelése, és ezt a Unicode szabvány biztosítja.

### 3.6. Az adatbázis-kezelő



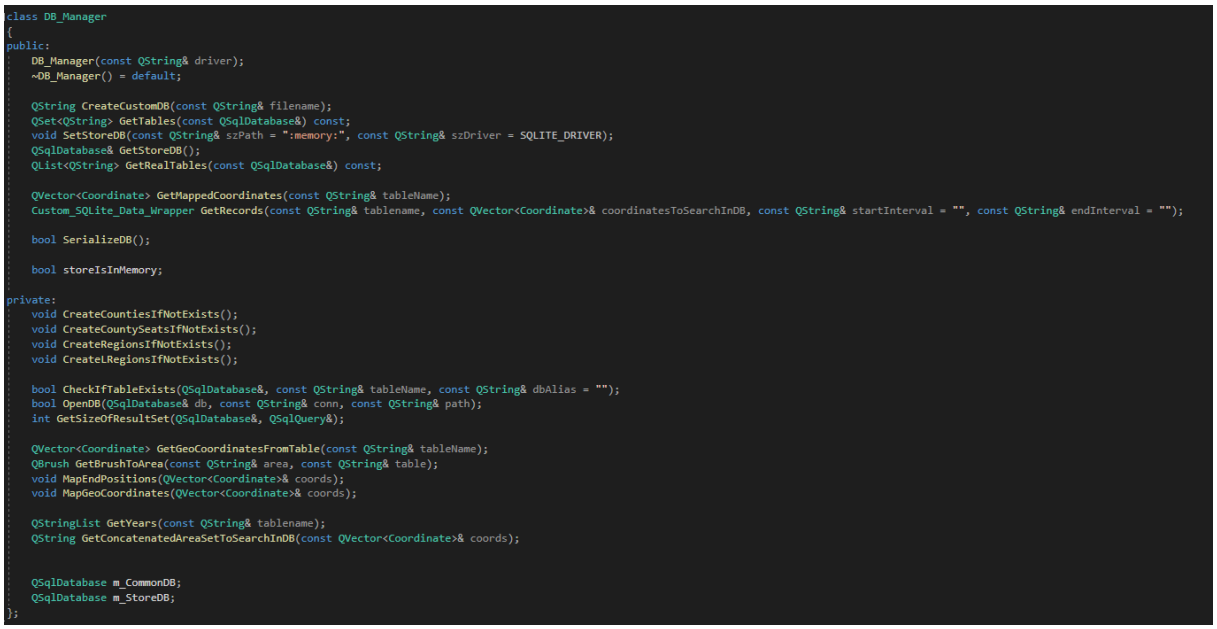

3.6.1. ábra: Az adatbázis-kezelő osztálydiagramja

[19] A program készítését követően a KSH oldalán elérhető fájlok újabb formátumot kaptak a weboldalon, viszont a programban is használtak továbbra is elérhetőek maradtak.

A `DB_Manager` felelős az adatok adatbázisfájlokba való mentéséért és mindenféle adatbázissal kapcsolatos tevékenység kivitelezéséért, például a szérializációért, vagy adatbáziskapcsolat megnyitásáért ahogyan az a 3.6.1-es ábrán látható. A sémákról és az azok alapjául szolgáló EK diagramokról az első fejezetben volt szó. A `MainWindow` tartalmazza a `DB_Manager`-t és a program indulásakor inicializálódik az. A kezelő van kapcsolatban az előbb említett fájlkezelővel és mindennemű adatot a fájlkezelő szolgáltat neki a különböző struktúrákban. Továbbá, mivel a geo koordináták lekérdezését is ez végzi, így a `Coordinate` struktúrát is felhasználja a kívánt adatok eltárolására.

Az adatok formátumáról volt már szó, de hogy ezek milyen formában vannak lementve az adatbázisba, arról még nem. A segédtábláknak, azaz a megyéket, régiókat, nagyrégiókat és azok geo koordinátáit tartalmazó adattáblák egy adatbáziskapcsolatba kerültek, a commons-ba. A fájlból beolvasott adatok (a _Meta és _Data táblák) pedig egy külön kapcsolatba, a store-ba. A program indulásakor a commons, ha még nem létezik, akkor létrejön az adott fájlokból kiolvasott területi adatokkal. A store kapcsolat és a hozzá tartozó adatbázisfájl tartalma pedig igény szerint (ha szerializálni szeretne a felhasználó) növekszik.

Alapvetően a specifikációnak megfelelően SQLite adatbázis van használatban mind a store, mind a commons kapcsolatoknál. Viszont SQLite-nál van egy lehetőség in-memory adattárolásra, azaz a RAM memória használata a rekordok tárolására. Ez van kihasználva, amikor a fájlok tartalma beolvasásra kerül, ugyanis a fájlkezelő által generált struktúra tartalmát egy ilyen in-memory adatbázisba menti az adatbázis-kezelő. Ezt pedig a felhasználó tovább mentheti a merevlemezre, ha szeretné. Azért jó ez a tárolás, mert ahogyan a korábbi alfejezetben említve lett a rekordok megjelenítéséért felelős `Table_Editor` a model/view nézetet használja mégpedig Sql alapú adat model-t. Ha in-memory adatbázisba vannak az éppen fájlból beolvasott adatok, akkor is képes lesz azokat is ugyanúgy megjeleníteni, mint a már adatbázisfájlokba mentetteket. Az in-memory állapot kezelésére van egy külön változója, illetve a store kapcsolatot attól a változótól függően állítja in-memory-ra, vagy pedig a store kapcsolathoz tartozó közös adatbázisfájlra, ahol a tábla rekordok elmentve vannak tárolva. A 3.6.2-es ábrán a fentebb leírtak láthatóak kódszinten is.



```
class DB_Manager
{
public:
    DB_Manager(const QString& driver);
    ~DB_Manager() = default;

    QString CreateCustomDB(const QString& filename);
    QSet<QString> GetTables(const QSqlDatabase&) const;
    void SetStoreDB(const QString& szPath = ":memory:", const QString& szDriver = SQLITE_DRIVER);
    QSqlDatabase& GetStoreDB();
    QList<QString> GetRealTables(const QSqlDatabase&) const;

    QVector<Coordinate> GetMappedCoordinates(const QString& tableName);
    Custom_SQLite_Data_Wrapper GetRecords(const QString& tablename, const QVector<Coordinate>& coordinatesToSearchInDB, const QString& startInterval = "", const QString& endInterval = "");

    bool SerializeDB();

    bool storeIsInMemory;

private:
    void CreateCountiesIfNotExists();
    void CreateCountySeatsIfNotExists();
    void CreateRegionsIfNotExists();
    void CreateLRegionsIfNotExists();

    bool CheckIfTableExists(QSqlDatabase&, const QString& tableName, const QString& dbAlias = "");
    bool OpenDB(QSqlDatabase& db, const QString& conn, const QString& path);
    int GetSizeOfResultSet(QSqlDatabase&, QSqlQuery&);

    QVector<Coordinate> GetGeoCoordinatesFromTable(const QString& tableName);
    QBrush GetBrushToArea(const QString& area, const QString& table);
    void MapEndPositions(QVector<Coordinate>& coords);
    void MapGeoCoordinates(QVector<Coordinate>& coords);

    QStringList GetYears(const QString& tablename);
    QString GetConcatenatedAreaSetToSearchInDB(const QVector<Coordinate>& coords);


    QSqlDatabase m_CommonDB;
    QSqlDatabase m_StoreDB;
};
```

3.6.2. ábra: A DB_Manager osztály kódszinten

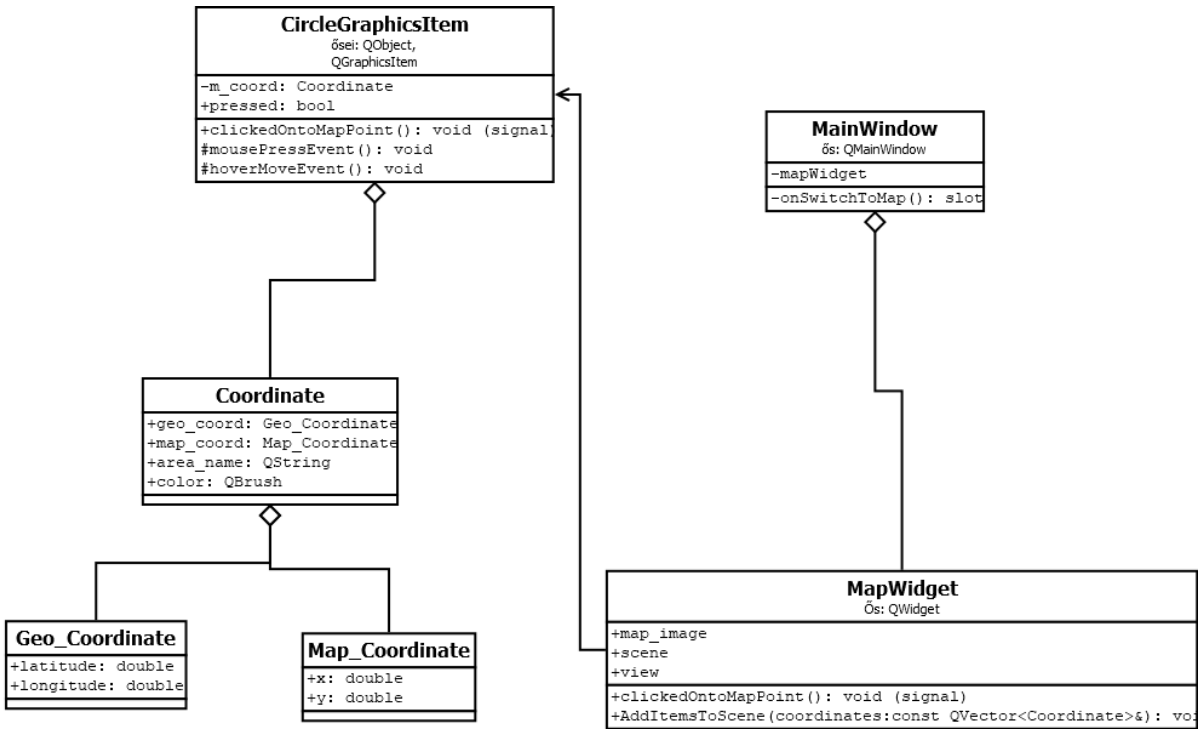
A használt SQLite adatbázis-kezelő motor több száz API-val rendelkezik, de a core API nem túl bonyolult és viszonylag kisméretű. A statisztikai programban nem közvetlen az SQLite API-ja lett felhasználva, hanem a Qt által nyújtott Qt SQL modul volt használva, ami adatbázis független. A modul, ahogyan a többi is, API-val rendelkezik, amin keresztül a fejlesztő kezelni tudja azt. A Qt modulját úgy írták meg, hogy több adatbázis motor API-jával is tudjon kommunikálni, köztük az SQLite API-jával is. Ehhez több plugint és a pluginokhoz tartozó drivert írtak. A driver egy közös interfészt és specifikus implementációt (az adott DBMS API-jához) biztosít, ez hajtja meg az adott DBMS API-ját, ezért is a driver név. Qt-ben csak a `QSqlDriver` absztrakt ősosztályból kell leszármaztatni, hogy újabb adatbázis drivert, pontosabban annak implementációját, adhassunk hozzá.[20] A plugin pedig a driver létrehozásáért felelős, illetve a plugin lesz dinamikusan betöltve a program futása során, ezért is a plugin név, ami egy dll tulajdonképpen.[21] Qt-ben az absztrakt plugin ősosztály a `QSqlDriverPlugin`. Tehát akár saját adatbázis plugint és drivert is lehet írni Qt-hez, ha implementáljuk ezeket az ősosztályokat. A `QSqlDatabase` (ez már az SQL modul része, ami direktben már használva volt a statisztikai programban is) pedig a Qt-ben az adott adatbáziskapcsolatot reprezentálja. Jelen esetben ez az SQLite driver által biztosított interfészen keresztül az SQLite API-val való adatbáziskapcsolatot jelöli.[22]

[20] https://doc.qt.io/qt-5/qsqldriver.html

[21] https://doc.qt.io/qt-5/qsqldriverplugin.html

[22] https://doc.qt.io/qt-5/qsqldatabase.html

### 3.7. A program térképnézete




3.7.1. ábra: Koordináták és a Térképnézet

A következő nézet, ami az adatokat egy másik szemszögből jeleníti meg a felhasználónak a térképnézet, ahol az adatok a térképen úgy jelennek meg, hogy az adathoz tartozó terület van megjelölve. Mivel Magyarországi statisztikai adatokról van szó, így kezdésként egy Magyarország térképet lát a felhasználó, majd, ha adatot tölt be fájlból vagy a kapcsolódó adatbázisból, akkor a területeket reprezentáló pontok, amikhez az adathalmaz tartozik, kirajzolódnak a térképen. Két különböző technikára volt ehhez szükség, a Qt Graphics View technológiájára és a lineáris interpolációra.

#### 3.7.1. Interpoláció

A lineáris interpolációról esett már szó a bevezetőben, ahol ugyanis említve volt, hogy a linearitás felhasználása elegendő a térkép pontok kiszámításához. Magyarország méretű területnek ugyanis a Földön való görbülete elenyésző, így azzal nem kell számolni. Az interpoláció egy matematikai közelítő módszer, amely kiszámolja egy adott függvénynek a

még nem ismert értékeit az ismert értékek birtokában. A közelítő módszereknek, melyek modellfüggvényt alkalmaznak ponthalmazok közelítésére két fajta változata van. A közelítés történhet az ismert pontok közötti értelmezési tartományhoz rendelt értékekkel, ez az interpoláció, amikor ugyanis a köztes pontok értékeire becslést adunk a már ismert pontok felhasználásával felírt modellfüggvénnyel. A másik változata az extrapoláció, ahol másmilyen módszereket alkalmaznak és az ismert ponthalmaz folytatására, vagy előzményére adnak egyfajta közelítést.

A programban interpoláció volt használva, mégpedig úgy, hogy az értelmezési tartomány a területek geo koordinái voltak. Az értékkészlet pedig, amire a becslést számolta ki a program, a koordinátákhoz rendelt pixelek helyzete. A két kiindulópontot meg kellett külön adni, ez úgy lett megoldva, hogy ki lett jelölve két referencia pont a térképen (a térkép két széle), aminek a pixel értékeihez (ami pixelek egy-egy területet jelölnek) a terület geo koordinátája is meg lett adva explicit. Ebből kiindulva a többi térképpont pontosan kiszámítható volt a 3.7.1-es képlet alkalmazásával.

$$y = y_1 + ((x - x_1)\,(y_2 - y_1)) \,/\, (x_2 - x_1)$$

3.7.1.1. képlet[23]

A képletben az $x$ mindig az adott kiszámolandó terület geo koordinátája, ami egy latitude és longitude páros ebben a sorrendben, ezek a koordináták vannak az adatbázis segédtábláiban eltárolva. Az $y$ pedig mindig az adott területhez tartozó pixel érték, amit kiszámol a program. Az $x1$ az egyik segédponthoz, míg $x2$ a másikhoz tartozó koordináta. Az $y1$ az $x1$ segéd koordinátához tartozó, míg $y2$ az $x2$-höz tartozó pixel érték.

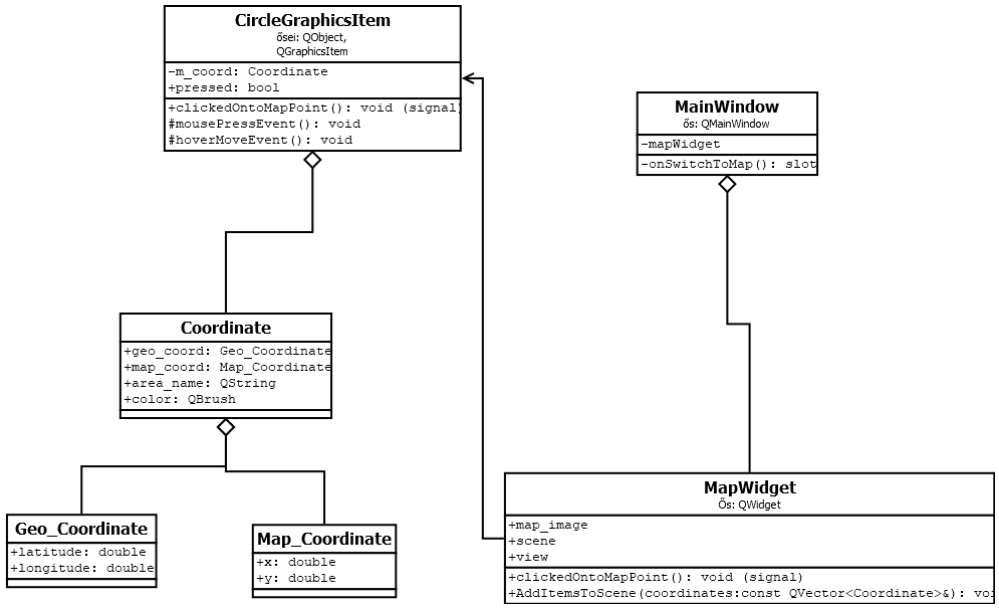
### 3.7.2. A Qt Graphics View-ja a programban

Az interpoláción kívül a másik technika, ami használva volt a térképnézetnél az a Qt Graphics View technológiája, ami a térkép és az azon lévő pontok megjelenítését tette lehetővé. A Graphics View egy item alapú megközelítése a megjelenítésnek, ami hasonló a model-view minta item alapú megközelítéséhez (amiről nem volt szó a dolgozatban, mert a táblanézetnél nem az item alapú model-view volt használva).

[23] https://hu.wikipedia.org/wiki/Interpol%C3%A1ci%C3%B3

A lényeg, hogy van egy vagy több view, amivel egy adott színteret (scene-t) figyelhetünk meg. A scene tartalmazhat elemeket, mint például jelen esetben térképet, vagy a térképre rajzolt pontokat. Ebből a három réteg típusból épül fel, amit a felhasználó lát. Minden rétegnek saját Descartes-féle koordináta-rendszere van. Ha az alsó rétegekben történik tarnszformáció, például pozíció változtatás, az a felsőbb rétegekben lévő koordinátarendszerekben meglátszik, míg, ha a felsőbb rétegek kapnak valamilyen transzformációt, az alsóbb rétegekben lévő koordinátarendszerek változatlanok maradnak. Viszont maga a tartalmazott elem ezekkel a parent transzformációkkal a parent koordináta rendszerében transzformálva van implicit, így nem kell utána húzni a child-okat a parentnek. Egy item-nek parent-je egy másik item vagy a legfelsőbb parent, a scene lehet.

Az scene elemeinek megvan a saját, lokális koordinátarendszerükben is a pozíciójuk, illetve négyzetbe foglaló koordinátájuk, de megvannak az elemeknek külön scene koordinátákban is ezek a tulajdonságaik. A scene a view-tól kapja az ott történt eventeket és továbbítja az itemek-nek azokat. A view-n lévő változások nem befolyásolják a scene-t, tehát a view egyfajta ablak a scene-re (a színtérre). A view koordináták egy pixelnek felelnek meg és ezek láthatók a felhasználó számára. A view, scene és item koordináták közötti konverziók megoldottak különböző függvényhívásokkal.[24] A statisztikai program ezeket a tulajdonságait a Graphics View-nak figyelembe veszi és alkalmazza a térképnézetén, aminek az osztálydiagramja a 3.7.2.1-es ábrán látható.




3.7.2.1. ábra: A MapWidget

[24] https://doc.qt.io/qt-5/graphicsview.html

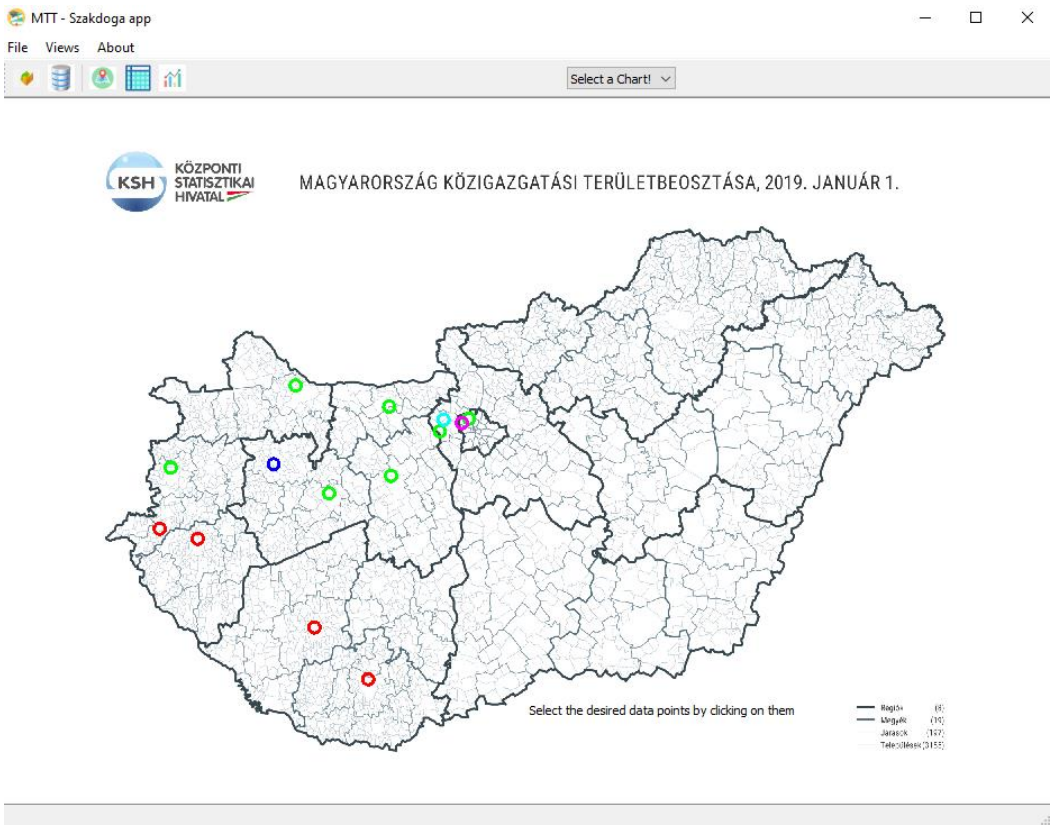
A térképnézet a scene-t, view-t, térképet tartalmazza, illetve a statisztikai adatokat térképen reprezentáló pontok egy saját GraphicsItem-ben vannak, amik a scene-hez vannak rendelve. A `MapWidget` csakúgy, mint a többi nézet a `MainWindow`-val signalokon keresztül kommunikál, ha például egy pontra kattint a felhasználó, az signalon keresztül lesz továbbítva a `MainWindow`-nak. A saját item a Qt `QGraphicsItem`-ből van származtatva, és a koordinátáit tárolja, valamint azt, hogy jelenleg pressed állapotban van-e. Ha rákattint a user egy item-re, akkor az itemhez rendelt felülírt `mousePressEvent()` event handler propagálja tovább az eventet a térképnézetnek, de már signal formájában. Az event handlerek a Qt-ban a következőképpen működnek.

Minden Qt alapú program, mely rendelkezik GUI-val esemény vezérelten működik, az események (eventek) a user interakció során létrejött csomagok, amik az interakció részleteit tartalmazzák. Ehhez szükséges egy event loop-nak léteznie, amit a `QCoreApplication` biztosít. A program indulásakor indul el a fő event loop az `exec()` függvényhívással és egészen a terminálásig, vagy crash-ig működni fog. Az event loop a program aszinkronitásához járul hozzá. Van egy queue-ja, amibe a kiváltódott eventek kerülnek a háttérben, ha azok postolva lettek. Az eventek amiket fogad származhatnak a window systemtől (amiket majd Qt `QEvent*`-ekre transzformál), vagy a Qt alkalmazás is generálhat `QEvent`-ből származtatottakat. Ha a queue-ba szeretnénk helyezeni eventet a későbbi feldolgozás igényeként, akkor postolni kell az eventet a `QCoreApplication::postEvent()`-el, ha azonnal feldolgozni szeretnénk, akkor ennek a sending változatát kell hívni. Ha a program éppen valamilyen szinkron folyamatot végez, akkor nem történik semmi a queue-ban lévő eventekkel, ha viszont éppen várakozik, azaz elfogytak a szinkron műveletek, akkor az event loop kivesz egy eventet a queue-ból (illetve előtte még az összetartozókat optimalizálhatja egy közös event-be) majd elküldi az eventet egy event handlernek. Eventet fogadhat és kezelhet minden `QObject` leszármazott, de a widgetek kezelik ezeket általában. Továbbá saját eventet is lehet létrehozni a `QEvent`-ből való leszármaztatással.[25] Az event loopból dispatchelt eventet megkapja az a `QObject` leszármazott, amit megadtunk még a `postEvent` hívásakor. Ennek meghívódik az `event()` metódusa, ami az általános event handler. Ez általában dispatcheli az eventet az objektum egy specifikusabb handlerének az event típusa alapján, amit futási időben lekérdez.

[25] Részletek: https://doc.qt.io/qt-5/qevent.html

Az `event()` fő handler és a specifikusabb handlerek is virtuális metódusok, amiket felül lehet definiálni a custom leszármazott osztályokban, jelen esetben a `QGraphicsItem` `mousePressEvent`-je lett hasonlóképpen felüldefiniálva. Ha csak részben vannak felüldefiniálva, akkor mindenképpen ajánlott az ős handlerét is meghívni, hogy kezelve legyen minden eset.[26] Léteznek még event filter objektumok, amik proxyként funkcionálnak. Az event címzett objektuma előtt megkapják az adott eventet ők is (lehet egy objektumnak több filtere is), és feldolgozzák azt, vagy tovább küldik a címzett objektumnak. Az összes filteren végig megy az adott event csomag mielőtt megkapná azt a címzett objektum handlere ténylegesen.

Visszatérve az osztálydiagramhoz, a custom item egy `Coordinate`-et tartalmaz, ami pedig további két koordinátát, amikről az interpoláció részben volt szó. A geo koordinátát, mint értelmezési tartomány egy pontját, és a map koordinátát, ami az értékkészlet egy pontja. A kompozit koordináta továbbá eltárolja a színét is, ahogy a térképen fog kirajzolódni a felülírt `paint()` metódusa által. A 3.7.2.2-es ábrán a program térképnézete látható Dunántúli adatokkal és pár kiválasztott területtel (piros színűek a kiválasztott területek).

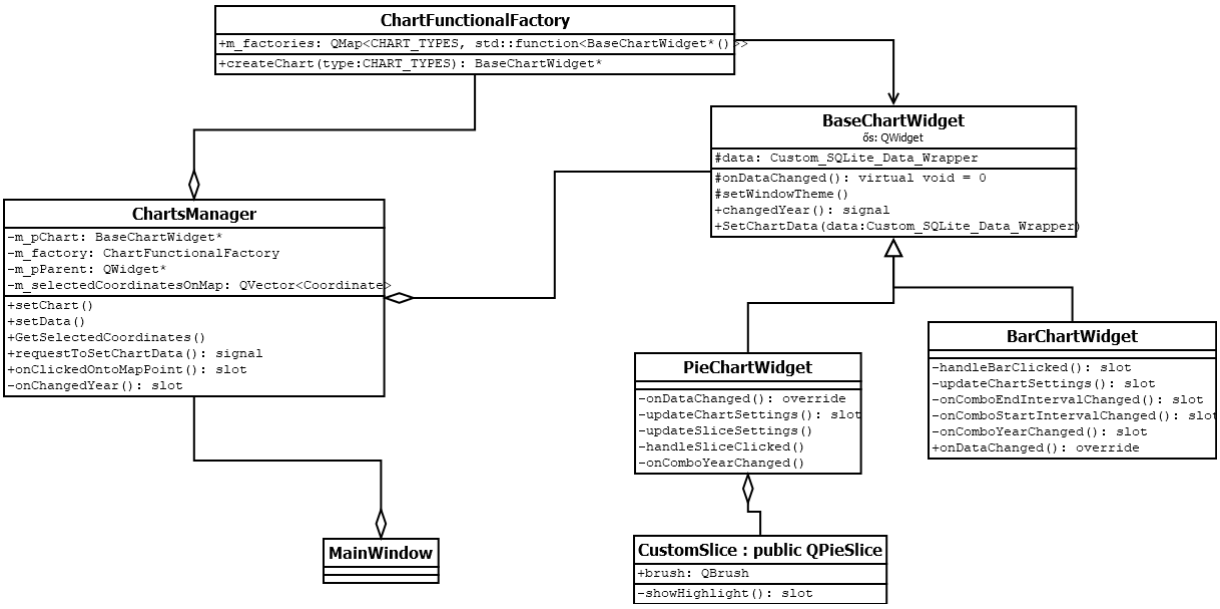



3.7.2.2. ábra: A program térképnézete

[26] Bővebben: https://doc.qt.io/qt-5/eventsandfilters.html

### 3.8. A diagram nézet

Ehhez a nézethez, ami az utolsó is egyben, egy kezelő lett hozzáadva, a `ChartsManager`, amit a `MainWindow` szintén tartalmaz és ami a `MainWindow`-val szintén signal-okkal kommunikál a többi programelemhez hasonlóan. Többek között az aktuális chart létrehozásáért felelős (egyszerre csak egy chart lehet a nézetben) és annak adatokkal feltöltéséért. A chartok létrehozásának az alapja egy közös chart, ami a chartok közös interfészét biztosítja. A közös base chart van továbbá a polimorfikus hívásokkor is felhasználva. A 3.8.1-es ábrán látható a chartokhoz tartozó osztálydiagram.



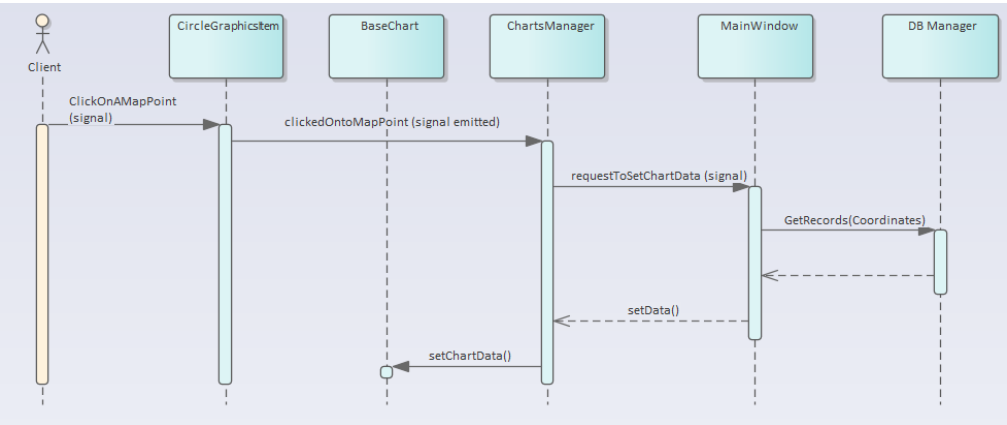

3.8.1. ábra: Charts

Jelenleg két chart típus-t támogat a program, amik a statisztikai adatokat különböző diagramokon jelenítik meg, a piechart típust és a barchart típust. Mindkettő charthoz rendelt widget a fent említett közös ősosztályból származik, a `BaseChartWidget`-ből. A base tartalmazza az interfészt (amibe a signalok is beletartoznak), és magát az adatot, ami a chartokon megjelenik. A specifikusabb chart widgetek a chartokon kívül az azokhoz tartozó customization beállításokat is tartalmazzák, mint a téma, legend stb.

A chartokat a `ChartsManager` hozza létre factory osztállyal, ami egy functional factory tulajdonképpen, mert functorokat tárol egy map-ben az adott chart típusokhoz. Így a

típus alapján hozza létre a különböző chartokat és kezeli azokat polimorfikusan utána (a virtuális metódusokon keresztül) a manager. A programtervezési mintáról bővebben a [6]-os irodalomban lehet olvasni.

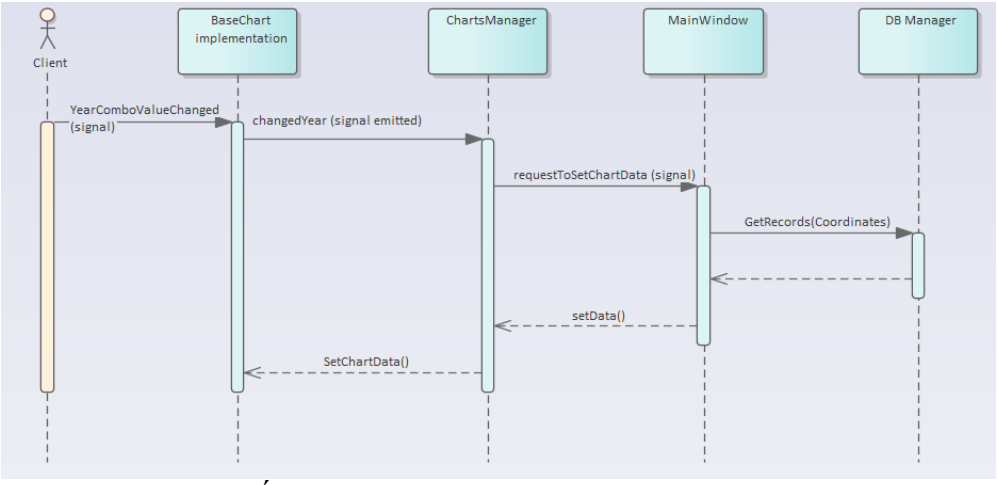
A `ChartsManager` egy `QObject`, hogy a `MainWindow`-val való signal-slot kommunikáció biztosítva legyen. A 3.8.2-es ábrán egy sequence diagram szerepel, amin először is a map pontok kiválasztása történik meg, ugyanis ez szükséges feltétele a chartok megjelenítésének, mert így töltődnek fel a chartok adatokkal a kiválasztott pontok alapján. A felhasználó kiválasztja a map ponto(ka)t, ekkor a custom `QGraphicsItem` signálja emittálódik és ez a `ChartsManager`-be érkezik be, ami eltárolja a kiválasztott ponto(ka)t, mint koordinátákat. A kezelő tovább küld egy signalt a `MainWindow` felé, ami a koordinátákhoz tartozó adatokat lekéri az adatbázis kezelőtől és visszaküldi az adatcsomagot a `ChartsManager`-nek, ami beállítja az aktuális Chart-jának az adatcsomagot a `BaseChart` interfészen keresztül.



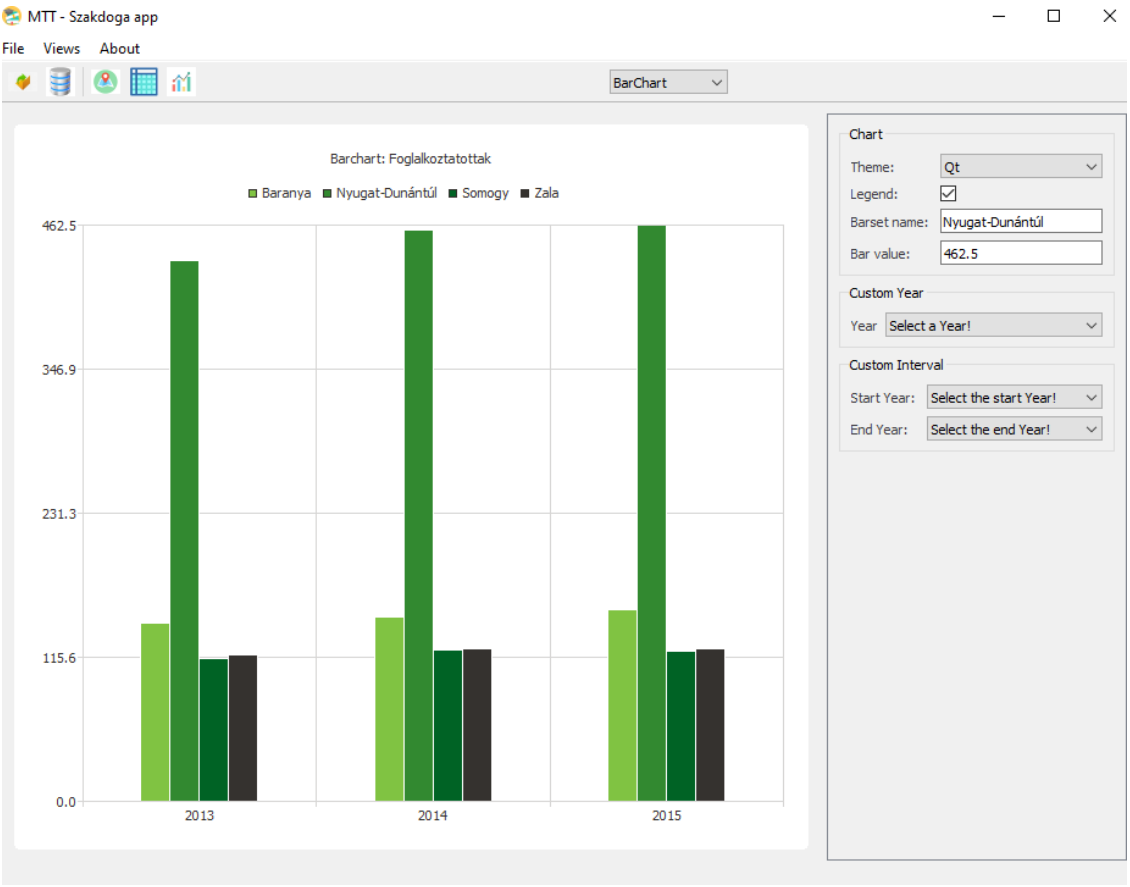

3.8.2. ábra: Térkép pontok kiválasztása (Sequence diagram)

Ezután már megjeleníthetők a kívánt chartok, hiszen ugyanazzal az adatcsomaggal dolgoznak. A Chartok settings-je többek között tartalmazza az egyes területekhez tartozó éveket, melyekhez adat kérdezhető le. Az év combo box-al így további évek adatai kérdezhetők le a területekhez. Ennek a folyamatnak a sequence diagramja a 3.8.3-as számú ábrán látható. Nagyon hasonló a folyamat a térkép pontok kiválasztásához, viszont a map point-ok most ki vannak hagyva a folyamatból. A chartokon másképpen lehet évet választani. A barchart az támogatja, hogy egy intervallumot is ki lehessen választani, de a folyamat ugyanaz lesz, mint egy önálló év kiválasztásnál. A barchart-ról képernyő kép a 3.8.4-es ábrán látható, ahol ugyanis 2013-tól 2015-ig terjedő időszakra vonatkozólag a foglalkoztatottak száma látható három Dunántúli megyében és egy régióban százezer főre vetítve. Az említett

`ChartsManager`, `BaseChartWidget` kódszintű reprezentációját pedig rendre a 3.8.5, 3.8.6. ábrák szemléltetik.




3.8.3. ábra: Év(ek) kiválasztása a Charton (Sequence diagram




3.8.4. ábra: Barchart példa

```cpp
class ChartsManager : public QObject
{
    Q_OBJECT

public:
    explicit ChartsManager(QWidget* parent);
    ~ChartsManager() = default;

    void setChart(CHART_TYPES type, const Custom_SQLite_Data_Wrapper& dataWrapper);
    void setData(const Custom_SQLite_Data_Wrapper& dataWrapper);
    BaseChartWidget* GetChart() const
    {
        return m_pChart;
    }
    QVector<Coordinate> GetSelectedCoordinates() const
    {
        return m_selectedCoordinatesOnMap;
    }

signals:
    void requestToSetChartData(QVector<Coordinate> coords, const QString& year = "");
    void requestToSetChartData(QVector<Coordinate> coords, const QString& startInterval, const QString& endInterval);

public slots:
    void onClickedOntoMapPoint(const Coordinate& coord);

private slots:
    void onChangedYear(const QString& newYear);
    void onChangedYear(const QString& startInterval, const QString& endInterval);

private:
    //egyszerre egy chart tartozik hozza jelen esetben
    BaseChartWidget* m_pChart;
    ChartFunctionalFactory m_factory;
    QWidget* m_pParent;

    QVector<Coordinate> m_selectedCoordinatesOnMap;
};
```

3.8.5. ábra: ChartsManager

```cpp
enum CHART_TYPES : int
{
    UNKNOWN = -1,
    PIECHART = 0,
    BARCHART = 1,
};

QT_CHARTS_BEGIN_NAMESPACE
class QChart;
QT_CHARTS_END_NAMESPACE

QT_CHARTS_USE_NAMESPACE

class BaseChartWidget : public QWidget
{
    Q_OBJECT
public:
    explicit BaseChartWidget(QWidget* parent = nullptr);
    virtual ~BaseChartWidget() = default;

    void SetChartData(const Custom_SQLite_Data_Wrapper& data);

signals:
    void changedYear(const QString& newYear);
    void changedYear(const QString& startInterval, const QString& endInterval);

protected:
    Custom_SQLite_Data_Wrapper m_data;

    virtual void onDataChanged() = 0;
    void setWindowTheme(QChart::ChartTheme);
};
```

3.8.6. ábra: BaseChartWidget

# 4. Továbbfejlesztési lehetőségek

Nem túlságosan részletekbe menően így áll össze a dolgozat témájaként megvalósított program, ahogyan a korábbi fejezetben az ismertetve lett. Azonban, mint minden program, ez sem tökéletes és számos továbbfejlesztési lehetőségei lehetnének. A továbbiakban a program írása közben felmerült lehetőségek említéséről lesz pár szó.

Az egyik fő továbbfejlesztési irányvonal a statisztikai fájlok köre lenne. Ugyanis, mint említve volt, a program jelenleg adott területekhez tartozó adatokat képes feldolgozni évek szerint. De számos olyan statisztikai mérés létezik, ahol másmilyen szempont szerint vannak a gyűjtött adatok feldolgozva, például nem évek szerint, hanem nemek szerint. Vagy ezt bonyolítani lehet többféle alszemponttal például nemek és korcsoport szerinti eloszlás. Ezeket a fajta adatokat sajnos még nem képes feldolgozni a program. A megoldás talán egy xml-szerű struktúra lenne csv formátum helyett, ahol a különböző tagekkel és azok egybeágyazásával az említett statisztikai adatokat is fel lehetne dolgozni egyfajta magasabb absztrakciót használva. Ezen felül lehetne még kisebb egységekre is az adatok feldolgozását alkalmazni, mégpedig az egyes településekre is, illetve nem Magyarországi viszonylatokban is akár lehetne gondolkodni. A bemeneti fájl validálásán is lehetne továbbá finomítani még. Az adatok feldolgozása is kerülhetne egy külön szálra - jelenleg egyszálon fut a program - de a betöltés és szérializáláskor nagyobb adathalmazok feldolgozása gyorsabb volna (a KSH oldaláról letöltött adatok feldolgozása lassulás nélkül történik jelenleg). A threading megoldás történhet többféle módszer közül való választással, a lényeg, hogy nem szabad összekeverni a különböző módszereket, csak egy bizonyos módszerrel jó az ilyeneket megvalósítani. A Qt is nyújt threadinghez támogatást és a C++ szabvány is.

A továbbfejlesztés másik irányvonala a diagramok lennének. Ami a megjelenítést illeti akár lehetne toggle gombok használata is a combo box helyett, tartalmi szempontból pedig támogathatna többfajta chart típust a program. Chart típusokkal kibővíteni a BaseChartWidget és a ChartsManager miatt nem lenne túl bonyolult. Most, mint a többi ötletre is idő hiányában nem jutott már implementáció. A chartokat továbbá lehetne úgy is megjeleníteni, hogy nem egy külön nézetben lennének, hanem a térképen a kiválasztott pontokhoz rendelődnének. Ekkor viszont az egymáshoz közeli megyék mellett megjelenő diagramok esetében számolni kellene a diagramok ütközésével, egyfajta ütközés detektálást kellene leimplementálni hozzá. A megyék, régiók, nagyrégiók is a pontok kiválasztásakor kiszíneződhetnének egy adott

színre, amihez viszont a megyehatárokat, régióhatárokat, nagyrégió határokat kellene valamilyen képfeldolgozó eljárással feltérképezni.

Az utolsó nagyobb továbbfejlesztési irányvonal a program tesztelése lehetne. Ahol ugyanis az automatizált unit teszteket meglehetne írni a backend logikához például a GoogleTest használatával. Integrációs teszttel a különböző kapcsolatokat lehetne tesztelni további függőségek valamilyen helyettesítésével, például az adatbázist fake-elni, a további függőségeket, mint függvényhívások például, stub-okkal vagy driverekkel helyettesíteni. A unit és integrációs teszteknél a felesleges dependenciák megszüntetése a bonyolult feladat főleg. A unit teszteken kívül az automatizált UI tesztekhez pedig léteznek különböző programok, például a Squish, amiben python nyelven lehetne teszteket írni, a különböző scenáriókat tesztelve. A Squish egyébként nagyban az objektum elnevezések alapján, ami Qt-ban támogatva van, tudja meghatározni, hogy ha valahova automatizáltan kattint a képernyőn, az melyik program objektumnak felel meg. Rendszer integrációs teszttel esetleg a használt adatbázissal való interakciót lehetne tesztelni.

Összességében sok több továbbfejlesztési irányvonal közül lehet választani, és ami a legfőbb szempont szokott lenni a megfelelő irány kiválasztásában, az mindig az ügyfél vagy felhasználó elgondolása. Ebben nagy segítséget és eszközt nyújthat a folyamatos agilis kommunikáció a megrendelővel, ami történhet a különböző metodológiák, például az egyik legszélesebb körben használt a Scrum, alkalmazásával a fejlesztői csapatban.

# *Irodalomjegyzék*


[1] Ken Martin, Bill Hoffman (2015): *Mastering CMake*, Kitware Inc.

[2] Paul Sanderson (2018): *SQLite Forensics*, Independently published

[3] Scott Chacon, Ben Straub (2014): *Pro Git*, Apress, New York

[4] Guillaume Lazar (2018): *Mastering Qt 5: Create stunning cross-platform applications using C++ with Qt Widgets and QML with Qt Quick, 2nd Edition*, Packt Publishing Ltd., Birmingham

[5] Daniel Molkentin (2007): *The Book of Qt 4: The Art of Building Qt Applications*, No Starch Press, San Fransisco

[6] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides (1994): *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley Professional

## *Nyilatkozat*

Alulírott, Forgó Martin, gazdaságinformatikus Bsc szakos hallgató, kijelentem, hogy a dolgozatomat a Szegedi Tudományegyetem, Informatikai Intézet Szoftverfejlesztés Tanszékén készítettem, gazdaságinformatikus Bsc diploma megszerzése érdekében.

Kijelentem, hogy a dolgozatot más szakon korábban nem védtem meg, saját munkám eredménye, és csak a hivatkozott forrásokat (szakirodalom, eszközök stb.) használtam fel.

Tudomásul veszem, hogy szakdolgozatomat a Szegedi Tudományegyetem Informatikai Intézet könyvtárában, a helyben olvasható könyvek között helyezik el.

Szeged, 2021.05.18.

Aláírás

## *Köszönetnyilvánítás*

Meg szeretném ezúton köszönni Dr. Alexin Zoltán témavezetőmnek a szakdolgozat során adott ötleteit, hasznos tanácsait és a rendszeres egyeztetéseket, még ha azok online is zajlottak ebben a járványidőszakban.

Köszönettel tartozom családomnak a dolgozat írása alatt nyújtott támogatásukért és biztatásukért, a finom ételekért, melyek erőt adtak a folytatásokhoz és a családias légkörért, mely az otthon biztonságát jelentette számomra.

És nem utolsó sorban szeretném hálatelt szívvel megköszönni barátnőmnek, Zsuzsinak azt a sok bátorítást, és gyengéd biztatást, mely nélkül nem készülhetett volna el a szakdolgozatom. A fejezetek átnézéséért, módosítási javaslataiért pedig külön köszönet illeti őt.