
UNIVERSITATEA „SAPIENTIA” DIN CLUJ-NAPOCA
FACULTATEA DE ȘTIINȚE TEHNICE ȘI UMANISTE,
TÎRGU-MUREȘ
SPECIALIZAREA CALCULATOARE

SISTEM DE VIZUALIZARE
VOLUMETRICĂ A DATELOR
IMAGISTICE MEDICALE
PROIECT DE DIPLOMĂ

Coordonator științific:

Prof Dr. habil Szilágyi László

Absolvent:

Ujfalvi Csongor-Robert

2021

UNIVERSITATEA „SAPIENTIA” din CLUJ-NAPOCA
Facultatea de Științe Tehnice și Umaniste din Târgu Mureș
Specializarea: **Calculatoare**

Viza facultății:

LUCRARE DE DIPLOMĂ

Coordonator științific:
Prof. Dr. habil Szilágyi László

Candidat: **Ujfalvi Csongor-Robert**
Anul absolvirii: **2020**

a) Tema lucrării de licență:

SISTEM DE VIZUALIZARE VOLUMETRICĂ A DATELOR IMAGISTICE MEDICALE

b) Problemele principale tratate:

- Studiu bibliografic privind sistemele de vizualizare imagini medicale
- Realizarea unei aplicații pentru vizualizarea eficientă a datelor imagistice măsurate și prelucrate (segmentate)
- Optimizarea aplicației de vizualizare pe monitor 4K

c) Desene obligatorii:

- Schema bloc al aplicației
- Diagrame UML privind software-ul realizat.

d) Softuri obligatorii:

- Aplicație de vizualizare volumetrică a datelor imagistice medicale

e) Bibliografia recomandată:

1. Milan Sonka, Vaclav Hlavac, Roger Boyle: Image processing, analysis, and machine vision. Stamford, Conn : Cengage Learning, 2015
2. Menze BH, Jakab A, et al: The Multimodal Brain Tumor Image Segmentation Benchmark (BRATS). IEEE Transactions on Medical Imaging 34(10):1993-2024, 2015
3. Szilágyi L, et al: Low and high grade glioma segmentation in multispectral brain MRI data, Acta Univ. Sapientiae – Informatica 10(1):110-132, 2018
4. Lefkovits L, et al: Brain tumor segmentation with optimized random forest. MICCAI BraTS 2016, LNCS vol. 10154, pp. 88-99, 2017

f) Termene obligatorii de consultații: săptămânal

g) Locul și durata practicii: Universitatea „Sapientia” din Cluj-Napoca,
Facultatea de Științe Tehnice și Umaniste din Târgu Mureș
Primit tema la data de: 31.03.2020
Termen de predare: 06.07.2021

Semnătura Director Departament

Semnătura coordonatorului

Semnătura responsabilului
programului de studiu

Semnătura candidatului


Declarație

Subsemnatul Ujfalvi Csongor-Robert absolvent al specializării Calculatoare., promoția 2020 cunoscând prevederile Legii Educației Naționale 1/2011 și a Codului de etică și deontologie profesională a Universității Sapientia cu privire la furt intelectual declar pe propria răspundere că prezenta lucrare de licență se bazează pe activitatea personală, cercetarea/proiectarea este efectuată de mine, informațiile și datele preluate din literatura de specialitate sunt citate în mod corespunzător.

Localitatea, Targu-Mures

Data: 06.07.2021

Absolvent

Semnătura..........

Sistem de vizualizare volumetrică a datelor imagistice medicale

Extras

Chiar și astăzi, numărul deceselor cauzate de tumorile cerebrale la nivel mondial este de sute de mii. Datorită îmbunătățirii rapide a științelor medicale și a altor ramuri ale științei în zilele noastre, există numeroase sisteme diferite care ajută profesioniștii din domeniul medical să diagnosticheze tumorile cerebrale mai repede decât înainte.

După diagnostic, în medie pacienții cu tumori cerebrale trăiesc doar câteva luni mai mult de un an, deci este esențial să accelerăm procesul de diagnosticare a tumorilor pentru a începe tratarea pacienților înainte de a fi prea târziu.

Subiectul proiectului meu de diplomă este un software care ajută dezvoltarea și testarea algoritmilor care facilitează detectarea și cartarea tumorilor cerebrale pe baza datelor volumetrice RMN. Aplicația este capabilă să afișeze în rezoluție 4K înregistrările cerebrale segmentate și mapate, împreună cu indicatorii cheie, rezultatele și datele statistice pe ecran. De asemenea, poate stoca aceste date într-o bază de date și le poate face ușor disponibile pentru a le interoga ulterior pentru utilizări viitoare.

În timpul procesului de proiectare a software-ului, am folosit multe modele și principii de proiectare a software-ului. În timpul dezvoltării aplicației, am folosit multe produse open source și gratuite Microsoft, inclusiv cadre, instrumente pentru dezvoltatori și pachete de biblioteci.

Cuvinte cheie: Interfața cu utilizatorul, Înregistrări RMN, Dezvoltare de software

**SAPIENTIA ERDÉLYI MAGYAR
TUDOMÁNYEGYETEM
MAROSVÁSÁRHELYI KAR
SZÁMÍTÁSTECHNIKA SZAK**

**BÖNGÉSZŐ ALKALMAZÁS TÉRBELI
ORVOSI KÉPEK MEGJELENÍTÉSÉRE
DIPLOMADOLGOZAT**

Témavezető:

Dr. habil Szilágyi László
egyetemi tanár

Végzős hallgató:

Ujfalvi Csongor-Robert

2021

Kivonat

Több százezres nagyságrendű manapság is az agyi tumorok miatti elhalálozások száma globális szinten. Az orvostudomány illetve más tudományágak fejlődésének hála manapság megannyi rendszer létezik, amely segíthet az orvosoknak korábban diagnosztizálni a tumorokat. Mivel a diagnózis után átlagosan egy évnél alig pár hónappal több időt élhetnek azok akiket agyi tumorról diagnosztizáltak létfontosságú a lehető leghatékonyabban felgyorsítani a diagnózis folyamatát.

Dolgozatom témája egy olyan szoftver, amely segíti a fejlesztését és könnyed tesztelését olyan algoritmusoknak, amelyek megkönnyítik az agyi tumorok felismerését illetve feltérképezését MRI felvételek alapján. Az alkalmazás képes megjeleníteni 4K felbontásban az algoritmus tesztelése során létrejött feltérképezett agyszeleteket és az adott kötethez tartozó fontosabb mutatókat, adatokat, illetve statisztikai eredményeket. Továbbá ezeket az adatokat a későbbi elérés érdekében egy adatbázisba menti az alkalmazás.

A szoftver tervezése során számos programtervezési mintát alkalmaztam, a megvalósításhoz pedig nyílt forráskódú és ingyenes Microsoft keretrendszereket, fejlesztői eszközöket és könyvtár csomagokat használtam.

Kulcsszavak: Felhasználói felület, MRI felvétel, Szoftverfejlesztés

Abstract

Even today deaths caused by brain tumors are in the scale of hundreds of thousands globally. Thanks to the rapid improvement of medical sciences and other branches of science nowadays there are numerous different systems helping medical professionals to diagnose brain tumors faster than before they have started using them. After the diagnosis, on average brain tumor patients live only a few more months than a year, so it's essential to accelerate the process of diagnosing the tumors to start treating the patients before it's too late.

The topic of my thesis is a software that makes the development and testing of algorithms which are segmenting and mapping brain tumors from volumetric MRI data. The application is able to display in 4K resolution the segmented and mapped brain records alongside with the key indicators, results and statistical data on screen. Also it can store these data in a database and make it readily available to query it later for future uses.

During the process of designing the software I've used many software design patterns and principles. While developing the application I've used many open source and free Microsoft products, including frameworks, developer tools and libraries.

Keywords: MRI record, User interface, Software development

Tartalomjegyzék

Ábrák jegyzéke	1
Táblázatok jegyzéke	1
1. Bevezető	2
2. Elméleti megalapozás és szakirodalmi tanulmány	4
2.1 MRI	4
2.2 MICCAI BRATS adatok	5
2.3 Szakirodalmi tanulmány	5
2.4 Felhasznált technológiák	7
2.4.1 .NET keretrendszer	7
2.4.2 C# programozási nyelv	8
2.4.3 Entity Framework	9
2.4.4 WPF - Windows Presentation Foundation	9
2.4.5 Microsoft SQL Server	9
2.5 Felhasznált programtervezési minták	9
2.5.1 Dependency injection (Függőség befecskendezés)	10
2.5.2 Binding properties (Kötési tulajdonságok)	10
3. A rendszer specifikációi	11
3.1 Követelmény specifikáció	11
3.1.1 Felhasználói követelmények	11
3.1.2 Rendszerkövetelmények	17
3.1.2.1 Funkcionális követelmények	17
3.1.2.2 Nem-funkcionális követelmények	17
3.2 A megjelenített adatok	18
3.2.1 Statisztikai adatok	18
3.2.2 Képek	19
4. A rendszer architektúrája	19
4.1 A rendszer komponensei	21
4.2 A komponensek modularitása és függőségek	21
4.3 A felhasznált technológiák specifikációi	21
4.3.1 .NET keretrendszer	22
4.3.2 Entity keretrendszer	22
4.3.3 WPF	22
4.3.4 Microsoft SQL Server	22
4.3.5 C# programozási nyelv	23
4.4 Használt eszközök	23
4.4.1 Microsoft Visual Studio	23

4.4.2 Microsoft SQL Server Management Studio	23
4.4.3 Github	24
5. Részletes tervezés	24
5.1 A szoftverkomponensek részletes leírása	24
5.1.1 Felhasználói felület	24
5.1.1.1 Főmenü	25
5.1.1.2 Kijelző ablak	26
5.1.2 Code behind	28
5.1.3 Model	31
5.1.4 Adatbázis	32
5.2 Adatfolyam	33
6. Üzembe helyezés	34
6.1 Üzembe helyezési lépések	34
6.2 Felmerült problémák és megoldásaik	35
6.3 Kísérleti eredmények	35
7. A rendszer felhasználása	35
8. Következtetések	36
8.1 Továbbfejlesztési lehetőségek	37
9. Irodalomjegyzék	38

Ábrák jegyzéke

1. Ábra: Az agyi tumorok miatti elhalálozások gyakorisága (bal) és előfordulása (jobb) az EU 27 tagállamában	3
2. Ábra: A .NET keretrendszer komponensei a 4.5-ös verzióig	8
3. Ábra: Használati eset diagram	12
4. Ábra: A szoftver architektúrájának blokksémája	21
5. Ábra: Az alkalmazás főmenüjének felhasználói felülete	25
6. Ábra: A kijelző ablak felhasználói felülete	27
7. Ábra: A kijelző ablak felső panelja közelről	28
8. Ábra: A főmenühöz tartozó osztály osztálydiagramja	29
9. Ábra: Az kijelző ablakhoz tartozó osztály osztálydiagramja	30
10. Ábra: Az adatbázist és alkalmazást összekötő model réteg osztálydiagramja	32
11. Ábra Adatbázis tábláinak és relációinak diagramja	33
12. Ábra: Az alkalmazás adatfolyam diagramja	34

Táblázatok jegyzéke

1. Táblázat: Első használati eset	13
2. Táblázat: Második használati eset	14
3. Táblázat: Harmadik használati eset	15
4. Táblázat: Negyedik használati eset	16

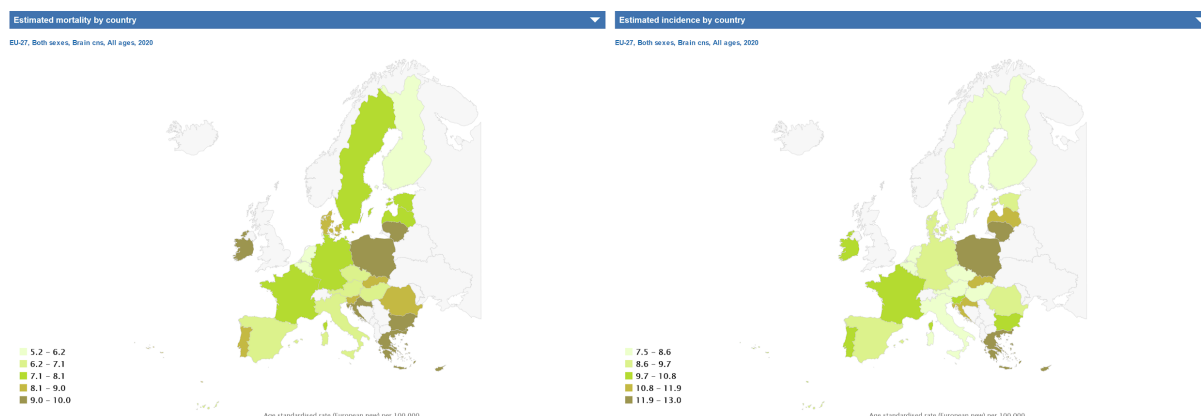
1. Bevezető

Több mint ötven év eltelt amióta az emberiség számítógépes rendszereket is használ képfeldolgozásra. Ezen rendszerek fejlődésének köszönhetően ma már számos tudományág alapozhat megbízható képfeldolgozó eszközökre. Az elmúlt évtizedek elején inkább az úrkutató intézmények repertoárjába tartoztak ezek a számítógépes képfeldolgozó rendszerek, de az idő múlásával és a technológiai fejlődés gyorsulásával meglehetősen hamar használatba kerültek számos tudomány-, illetve iparágban. Manapság a digitális képfeldolgozás már széles körben elterjedt és rengeteg olyan digitális eszköztár elérhető bárki számára, amelyek segítségével akár a mindennapi felhasználók is képesek elvégezni viszonylag bonyolultabb képfeldolgozási műveleteket is.

Viszont ezen folyamatok mai napig meglehetősen időigényesek maradtak bizonyos esetekben, mivel a feldolgozandó adat mennyiségével a feladat időtartama is nő. Ez a probléma orvosolható automatizált rendszerek segítségével, amelyeket a megfelelő felhasználói felülettel ellátva használhatóvá tehetünk egy átlagos felhasználó számára is, legyen szó egy olyan rendszerről amely a hétfégi utazásról készült több száz fotót szerkeszti számunkra, vagy akár egy olyan rendszerről amely különböző orvosi felvételeket elemez.

Az orvosi diagnosztika egy olyan terület amelyet könnyedén forradalmasíthatnak ezek a digitális képfeldolgozó rendszerek. Rendkívül sok betegség észlelésére lehet ilyen rendszereket használni. A technológia fejlődésével ezen rendszerek pontossága és gyorsasága is fejlődik, így a megfelelő hatékonyságot elérve meglehetősen felgyorsíthatják egy-egy betegség diagnosztizálását. Bizonyos betegségek esetében életbevágóan fontos az, hogy időben tudomást szerezzenek róla az orvosok, mielőtt az túlságosan elfajulna. Ilyen esetekben a digitális képfeldolgozó rendszerek nyújtotta segítség bizony életet menthet.

Az agyi tumorok az előbb említett betegségek csoportjába tartoznak. Nagy mértékben függ az észlelés idejétől az, hogy lehetséges-e gyógyítani vagy már túl késő. Manapság is világszerte évi százezres nagyságrendű elhalálozások köthetők agyi tumorokhoz. Bár az ECIS ([European Cancer Information System](#)) 2020-as statisztikái alapján Románia az Európai Unió átlagához közeli szinten helyezkedik el az az agydaganatok előfordulását illetően, az elhalálozási rátát figyelembe véve már jóval az átlag fölött található a listán, a felső középmezőnyt képviseli. Rengeteg befolyásoló tényező van, amely miatt viszonylag többek számára lehet halálos egy agyi daganat Romániában, illetve sok más országban, amelyek szintén az átlag fölött helyezkednek el. Az európai adatok az alábbi ábrán megtekinthetők [1].



1. Ábra: Az agyi tumorok miatti elhalálozások gyakorisága (bal) és előfordulása (jobb) az EU 27 tagállamában

Egyik ilyen tényező lehet az észleléshez szükséges korszerű eszköztár hiánya, amelyet orvosolva lényegesen csökkenhetne a diagnosztikát végző szakemberekre nehezedő teher mennyisége. Tapasztalt orvosok számára sem könnyű feladat felismerni és behatárolni egy agyi tumort MRI (magnetic resonance imaging - mágneses rezonanciás képalkotás) vagy CT (computed tomography - számítógépes tomográfia) felvételek segítségével.

Bár ezen felvételek segítségével külső beavatkozásoktól mentesen képesek megvizsgálni a pácienseket mégis roppant időigényes folyamat. Szeletenként kell elemezni a felvételeket és úgy kell megtalálniuk egy potenciális tumorhoz tartozó részeket. Egy ilyen folyamat óriási precizitást igényel, így roppant időigényes, illetve kellő szakértelem szükséges hozzá. A szakértők és az idő mennyisége is véges, így hát a leghatékonyabb megoldás a folyamat erőforrás igényességének csökkentése lenne.

Ebben segíthetnek olyan algoritmusok, amelyek automatikusan képesek észlelni az elváltozásokat a felvételeken, illetve továbbá képesek ezeket behatárolni. Hasonló algoritmusokat használva leszűkülhet a száma azoknak a felvételeknek, amelyeket szükséges elemeznie az orvosoknak.

Természetesen ezen algoritmusok abban az esetben a leghatékonyabbak, amennyiben könnyedén használhatóak az orvosok által. Ehhez egy olyan felhasználói felület szükséges, amely lehetővé teszi azt, hogy egy orvos, vagy számítástechnikai tudással nem rendelkező felhasználó is képes legyen gyorsan és egyszerűen böngészni az így nyert adatok között. A megfelelő felhasználói felület segítségével felesleges előfeltételek teljesítése nélkül is elérhetővé válnának az említett algoritmusok használatával elnyerhető előnyök.

Dolgozatom témája egy olyan szoftver, amely egyszerű módon képes megjeleníteni az említett algoritmusok által alkotott szegmentált felvételeket és megjeleníteni meg adatbázisba szinkronizálni az ezekhez tartozó további adatokat, illetve statisztikákat.

2. Elméleti megalapozás és szakirodalmi tanulmány

2.1 MRI

Az MRI egy olyan képalkotási eljárás, amely mágneses magrezonancián alapul. Bár már több mint negyven éve használják különböző diagnosztikai célokra, a technológia feltalálói csupán alig két évtizede nyerték el érte az Orvosi Nobel Díjat [10].

A mágneses rezonanciás képalkotás egy nagyon megbízható technológia, lehetővé teszi az orvosok számára, hogy az emberi test nehezen hozzáférhető részein is diagnosztizáljanak. Pontossága mellett említésre méltó az is, hogy a vizsgálat teljesen fájdalommentes, semmiféle káros sugárzásnak nincs kitéve a páciens, lehetővé teszi a diagnosztikát invazív eljárások (olyan eljárások, amelyek esetében szükséges a fizikai mintavétel) nélkül is.

Az MRI képalkotást egy erős mágneses térrel működő gép végzi. A legtöbb MRI berendezés 1,5T erősségű mágneses teret használ, de a 0,2T-től 7T-ig tartó skálán mozog ezen berendezések mágneses terének erőssége. A képalkotás során a mágneses tér meghatározza a hidrogénatomok protonjainak forgásmomentumát. A protonokat folyamatosan plusz energia éri, amelyet visszatérít a mágneses tér megszűnése után. Ez alapján képes a berendezés mérni és térbeli képet alkotni. A páciens egy bizonyos ideig mozdulatlanul kell maradjon a mágneses térben, amíg a képalkotási folyamat zajlik.

Az így kapott felvételek segítségével térben megvizsgálhatják az orvosok a páciensről készült felvételeket és felállíthatják a helyes diagnózist ezek segítségével. Az MRI előnye más képalkotási rendszerekkel szemben agyi tumorok felismerésénél az, hogy részletgazdag felvételeket készít lágy szövetekről is, illetve teszi ezt nagy felbontásban és erős kontrasztot biztosítva.

Bár sokak szerint az MRI az egyik legjobb képalkotási eljárás orvosi célokra megannyi zavaró tényező létezik, amelyek képesek lassítani a folyamatot és rontani a pontosságán. Egyik ilyen az, ha a páciens elmozdul. Ilyenkor a képek elmozdulnak egymáshoz képest és ez megnehezíti a felvételek használatát. Továbbá erőforrás igényes és időigényes folyamat a felvételek elkészítése. A diagnosztika is meglehetősen hosszadalmas folyamat, így hát rendkívül fontos minden lehetőség, amely ezt a teljes folyamatot képes lerövidíteni és/vagy költséghatékonyabbá tenni. Hosszútávon a kisebb optimalizálás is kifizetődik hasonló hosszú folyamatok esetében. Arról nem is beszélve, hogy a pontosság és hatékonyság javítása akár emberéleteket is menthet, így nem csak az anyagiak vagy idő szempontjából tűnik indokoltnak egy olyan rendszer használata amely segíti az orvos munkáját a diagnosztika során.

2.2 MICCAI BRATS adatok

Már 8 éve, 2012 óta évente megszervezik a MICCAI (Medical Image Computation and Computer Assisted Interventions) konferencia keretén belül a BRATS (Brain Tumor Segmentation) nevű versenyt.

A megoldandó feladat egyszerűsítésének érdekében a felhasználandó adatok szabványosított formában vannak közzé téve. Ezen szabványosítás célja az, hogy csupán az agyi szövetek osztályozására fókuszáljanak a kutatók és ne kelljen a zajok és felesleges struktúrák eltávolításával foglalkozni.

A rendelkezésünkre álló felvétel sorozatok azonos felbontásúak, ahol egy voxel egy 1 mm élhosszú kocka az agyi szövetből. A felvételek színsatornái a következők: T1, T2, TIC, FLAIR [6].

A felvételekhez mellékeltek az alapigazságot (GT - ground truth) megadó adatot is, amely azt határozza meg, hogy egy voxel normál szövet vagy egy tumorhoz tartozik. Ez szakértők általi értékeléssel jött létre, azért hogy tumor detektáló eljárások tervezése és tesztelése során segítséget nyújtson és egy egységes kiindulópontot adjon a kutatók számára [6].

2.3 Szakirodalmi tanulmány

Az elmúlt néhány évtizedben a tudomány számos ága forradalmi fejlődéseken ment keresztül. Ezen fejlődések és a számítógépek számítási kapacitásának rohamos növekedése lehetővé tették azt, hogy bizonyos tudományágak esetében a korábban kizárólag analóg módon történő metodológiák mellé digitális segítséget is igénybe vehessenek a kutatók.

Ilyen kategóriába tartozó szoftvereknek tekinthetők az orvosi képalkotási rendszerek is. Ezek digitálisan is elérhetővé tettek olyan adatokat és felvételeket, amelyek böngészése meglehetősen időigényes a hagyományos módszerek segítségével.

Továbbá a digitális módszer előnyei közé tartozik az is, hogy egyszerűbbé és gyorsabbá vált az adatok feldolgozása, ugyanis különböző statisztikai megadattudományi eszközök használatát is lehetővé tették a digitalizált adatok. Ez miatt különböző intelligens és automatikus rendszerek segítségét is igénybe vehetik ezeken a területeken különböző adatfeldolgozási, vagy adatbányászati folyamatok során. Ez által az emberek által elvégzett feladatok automatizálható és repetitív részei rendkívül felgyorsultak.

Ennek ellenére ezen rendszerek és algoritmusok is folyamatos tesztelésre szorulnak úgy a fejlesztési mint a finomhangolási fázisok alatt is. Rengetegen manuális tesztelést alkalmaznak és a különböző eredményeket különböző fájlokból kell megjeleníteni és ellenőrizni. Ez egy

elég lassú folyamat tud lenni, ezért idővel elkezdtek olyan rendszereket is fejleszteni, amelyek segíthetnek a hasonló folyamatok felgyorsításában.

Egy ilyen szoftver a 3D Slicer nevű alkalmazás is. A Slicert egy nemzetközi tudós közösség hozzájárulási eredményezték. Ezen hozzájárulások különböző területekről érkeztek, mint például mérnöki tudományok, vagy biomedicina [2].

A Slicer egy olyan alkalmazás, amely képes megjeleníteni és elemezni orvosi vonatkozású adathalmazokat és képeket. Minden gyakran használt adathalmaz formát támogat: képek, szegmentációk, felületek, annotációk, transzformációk. Mindezeket akár 2D, 3D és 4D verzióban is.

Továbbá a Slicer egy kutatói szoftver platform, amely lehetővé teszi, hogy a kutatók gyorsan és hatékonyan fejlesszenek és kiértékeljenek új módszereket, majd eljuttassák ezeket azokhoz az orvosokhoz, akik klinikai célokra is felhasználhatják munkájukat [2].

A Slicer egy termékfejlesztési platformként is szolgál. Segítségével különböző cégek vagy fejlesztők könnyedén létrehozhatnak prototípusokat amelyek esetében egy új módszer, metodológia kidolgozására fókuszálhatnak az helyett, hogy alap adatkezelési, adatvizualizációs vagy adat interakciós műveletek implementálásával töltenék el az időt [3].

A Slicer klinikai kutatási platformként is funkcionál. Ennek ellenére a hasonló kereskedelmi alkalmazásoktól eltérően nem alkalmas klinikai használatra munkaállomások beépített részeként, csupán kutatási célokra ezen a területen. Ez azért van mert rengeteg kísérleti jellegű eszközt és megoldást tartalmaz.

Többi orvosi képalkotási rendszer felvételeit támogatja a Slicer, többet között MRI, CT és PET felvételekkel is képes dolgozni. A legnépszerűbb felhasználási mód a 3D felvételek feldolgozása és vizualizációja. Ezeket térben is lehetséges megjeleníteni, de lehetséges különböző keresztmetszetek, szeletek megjelenítése is 2D vizualizációs felület segítségével. A megjelenítés mellett lehetséges különböző módosítások végrehajtása is a képen, illetve közelítés meg egyéb alpműveletek használata [3].

Ezeket kívül virtuális valóság és kiterjesztett valóság funkcionalitásokkal is ellátták a Slicert, ezek segítségével sokkal jobban átláthatóak térben a megjelenített felvételek.

A Slicer moduláris design követ, így a különböző főbb funkcionalitások különböző modulokban vannak elhelyezve. A szoftver fő komponensének felhasználói felületét eseményvezérelt szoftverfejlesztési könyvtár csomagok segítségével fejlesztették, mint a QT, viszont több hasonló könyvtárcsomag is használva volt, ezért nagyon bonyolult folyamatok állnak az események vezérlése hátterében [3].

A modulok esetében valamivel egyszerűbb megoldásokat használtak és az MVC (Model-View-Controller) szoftverarchitektúrát követték.

A Slicert a Quantitative Imaging Network (QIN) több használati esettanulmány során alkalmazta, többek között prosztatarák szűréssel illetve agyi tumor (glioblastoma) felismeréssel kapcsolatos kutatási projektek keretén belül [3].

A QIN az amerikai National Cancer Institute egy része, kvantitatív képalkotási eszközökkel és metodológiákkal kapcsolatban kutatnak, hasonló megoldásokat fejlesztenek és hitelesítik őket klinikai célokra [4].

2.4 Felhasznált technológiák

Az alábbiakban egy rövid áttekintés keretében bemutatom általánosan a szoftver megvalósításához használt technológiákat, keretrendszereket és könyvtár csomagokat. Ezen alfejelez során a szoftver implementációjától függetlenül, elméleti szinten mutatom be a technológiákat. Egy későbbi részben, a 4.3 A felhasznált technológiák specifikációi alfejezetben a tervezésre és megvalósításra is kitérek ezen technológiákkal kapcsolatban.

2.4.1 .NET keretrendszer

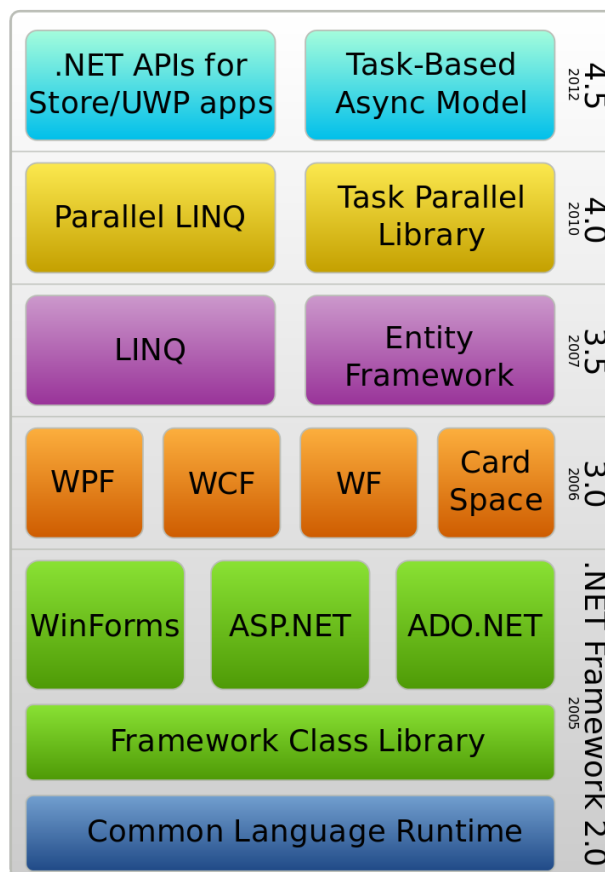
A .NET keretrendszer egy olyan fejlesztői platform amelyet a Microsoft nevű amerikai szoftverfejlesztő cég hozott létre és fejleszt a 2002-ben debütált első verziója óta.

Főbb sajátosságai közé tartozik többek között az, hogy gyors alkalmazásfejlesztés céljára igencsak alkalmas, továbbá lehetőséget nyújt platformfüggetlen alkalmazások fejlesztésére, illetve hálózati átláthatóság szempontjából is megkönnyíti a fejlesztők munkáját.

A .NET keretrendszer a Microsoft egy korábbi komponens alapú fejlesztői platformja, a COM (ActiveX), leváltása céljából jött létre és különböző fejlesztői eszközök (hardverek illetve szoftverek egyaránt) halmazából nőtte ki magát egy teljes fejlesztői platformmá.

A keretrendszer rendkívül sok mindenben segíti a fejlesztőket, a szoftverfejlesztés folyamatának számos aspektusában képes felgyorsítani a fejlesztést és hatékonyabbá tenni a fejlesztők munkáját. Segítséget nyújt kliens oldali alkalmazások fejlesztésében, de a szerver oldali résznél is hasonló módon hasznos megoldásokat tud nyújtani.

Az adatbázisok kezelését is megkönnyíti a keretrendszer, ami óriási előny lehet amennyiben többféle adatbázissal is dolgozik egy fejlesztő, vagy nem igazán jártas még az adatbázisok kezelésében.



2. Ábra: A .NET keretrendszer komponensei a 4.5-ös verzióig *

A .NET keretrendszer több komponensből áll, amelyek különböző kliens vagy szerver oldali megoldásokkal látják el a fejlesztőt. Kezdetben csak néhány komponensből állt a keretrendszer és az évek során, az újabb verziók megjelenésével, egyre inkább kibővül ezen komponensek listája. Ezek teljes listája és különböző verziókhoz tartozó felosztása látható a 2. ábrán.

A keretrendszer egy CLI rövidítésű, teljes nevén Common Language Infrastructure nevezetű alrendszerre épül, amely a .NET keretrendszer segítségével fejlesztett különböző programozási nyelvű programok futtatását teszi lehetővé. A CLI esetében egy sajátos osztálykönyvtár valamint virtuális gép (CLR - Common Language Runtime) segít abban, hogy a lefordított kód futtatható legyen.

2.4.2 C# programozási nyelv

A C# egy általános célú objektumorientált programozási nyelv, amelyet a .NET keretrendszerhez hozott létre a Microsoft. A nyelv alapjaként a Java és C++ programozási nyelvekre tekintenek. Mindkét nyelv hasonlóan általános célú és objektumorientált. A két

* <https://commons.wikimedia.org/wiki/File:DotNet.svg#/media/File:DotNet.svg>

nyelvhez hasonlóan a C# is erősen típusos és a Javához hasonlóan automatikusan történik a garbage collection, így egyszerűbb a memóriakezelés.

A nyelv típusai és natív megoldásai nagy mértékben kötődnek a .NET keretrendszerhez. A C# lehetőséget nyújt több programozási paradigmára is, nem csak objektumorientált fejlesztésre. Többek között funkcionális programozásra is alkalmas. Néhány évente új verziót jelenít meg a Microsoft, így inkrementálisan változik és fejlődik a nyelv.

2.4.3 Entity Framework

Az Entity Framework egy nyílt forráskódú objektum-relációs leképezési keretrendszer. Kezdetben a .NET keretrendszer beépített részeként fejlesztette a Microsoft, viszont idővel (a 6. verzió óta) önálló keretrendszerré vált. Az Entity Framework olyan ADO.NET technológiák halmaza, amelyek adatorientált szoftverek fejlesztését teszik egyszerűbbé.

A keretrendszer segít abban, hogy a fejlesztés során gyorsan és könnyen modellezhetőek legyenek az alkalmazás adatbázisához tartozó entitások és az azok közötti relációk. Továbbá leegyszerűsíti az adatok elérését és módosítását, mozgatását és még más hasonló műveleteket.

2.4.4 WPF - Windows Presentation Foundation

A WPF (Windows Presentation Foundation) egy nyílt forráskódú grafikus felhasználói felületek fejlesztéséhez használt osztálykönyvtár. Kezdetben a WPF is a .NET keretrendszer részeként indult, majd különálló alrendszer lett belőle. Célja az, hogy elválassza a felhasználói felületet az alkalmazások üzleti logikájától és lehetőséget nyújtson arra, hogy deklaratív módon fejleszthető legyen egy felhasználói felület a jelölőnyelv segítségével.

A WPF vektorgrafikus elemeket használ, ezeket DirectX segítségével rendereli a képernyőre. Továbbá lehetőséget nyújt adatkötésre, így a megjelenített adatok automatikusan szinkronizálhatók.

2.4.5 Microsoft SQL Server

A Microsoft SQL Server egy relációs adatbázis kezelő rendszer. Adatbázis szerverként használva lehetőséget nyújt adatok tárolására és lekérdezésére más alkalmazások által lokálisan, vagy akár egy hálózaton/interneten keresztül. Az SQL Server lekérdezési nyelve egy az SQL-hez hasonló T-SQL nevezetű procedurális nyelv.

2.5 Felhasznált programtervezési minták

Programtervezési mintáknak nevezhetünk olyan többször is használható, általánosított megoldásokat amelyek lehetővé teszik gyakori programozási feladatok megoldását. Ezen

programtervezési minták általában olyan osztályok leírásai amelyek egymással szimbiózisban működnek, tehát segítenek egymásnak ellátni a bizonyos feladatokat.

A programtervezési minták nem nyújtanak teljeskörű implementációt, inkább a megoldást formalizálva egy sablont biztosítanak amely alapján a programozó képes implementálni egy olyan kódrészletet, amely az adott feladat megoldásaként szolgál.

2.5.1 Dependency injection (Függőség befecskendezés)

A dependency injection egy olyan technika amely során bizonyos függőségeket bizonyos objektumokon belül egy másik objektum elégít ki. A függőséget használó objektum egy bizonyos szolgáltatást fog nyújtani a függőséget szolgáltató objektum számára és a függőséget a külső objektum kell átadja a szolgáltatást végző objektumnak az helyett, hogy az lokálisan hozza létre az adott függőséget. Így a szolgáltató objektum nem használhatja a függőséget statikus metódusokban, vagy konstruktor hívás esetén (utóbbi esetben csak akkor ha még nem kapta meg a függőséget).

A dependency injection-nek az a célja, hogy a szolgáltató objektum annyira elkülönüljön a függőséget bizonyító objektumtól, hogy ha lecserélik akkor ne kelljen módosítani a másik objektumot. Így remekül megfordítható a vezérlés formája [8].

2.5.2 Binding properties (Kötési tulajdonságok)

A property binding egy olyan konkurens minta amely során egy objektum bizonyos tulajdonságainak frissítése, szinkronizálása, befolyásolása olyan folyamatok során történik amelyek több megfigyelőt is bevonnak a feladat végrehajtására [9].

A property binding lehet egy- vagy kétirányú. Az előbbi abban a helyzetben előnyös, amikor valamelyik tulajdonság csak olvasható, más esetben a kétirányú használata szükséges.

Amennyiben a kötött tulajdonságok típusa nem egyezik szükséges lesz típuskonverziót alkalmazni. Továbbá bizonyos esetekben szükséges lehet sajátos konverterek, vagy akár illesztő minta implementálása is. Ezek abban az esetben is hasznosak lehetnek ha kötött tulajdonságok valamilyen bonyolult logika/algorithmus alapján befolyásolják egymást.

3. A rendszer specifikációi

A következő fejezetben az alkalmazás specifikációról lesz szó, amely magába foglalja a követelményeket amelyeknek eleget kell tennie a rendszernek, a felhasználónak és a rendszer által használt adatoknak, fájloknak.

3.1 Követelmény specifikáció

A rendszer tervezése megannyi követelményt szem előtt tartva történt meg. Ezen követelmények irányadóként szolgálnak a kivitelezés során illetve későbbi tervezési fázisok során. Továbbá különböző követelmények különböző határokat szabnak a felhasználó, illetve a rendszer számára, ezek a határok lesznek azok, amelyek egyértelműen körülírják azt, hogy az alkalmazás hogyan kell viselkedjen és a felhasználó miként érheti azt el, hogy az alkalmazás minden funkcionalitását használhassa.

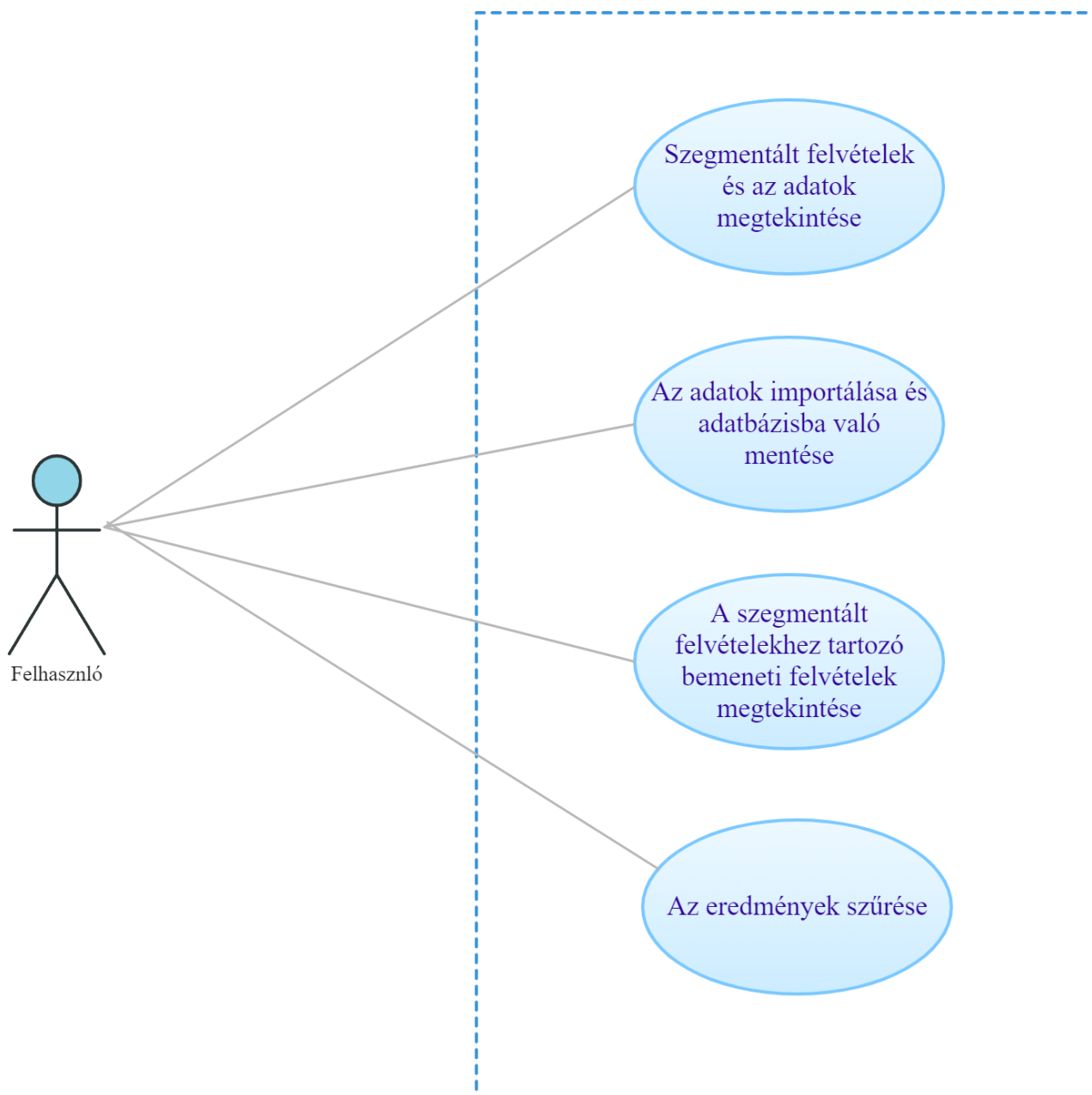
3.1.1 Felhasználói követelmények

A felhasználói követelmények olyan személyek számára készült diagramok illetve leírások, akik részletes ismeretek nélkül fogják használni az alkalmazást. Ide tartoznak olyan információk, amelyek az alkalmazás funkcióit írják le és mutatják be.

Az alkalmazás elsősorban az agytumor szegmentáló algoritmusokon dolgozó kutatók munkájának egyszerűsítésére és felgyorsítására céljából lett kitalálva.

A felhasználónak szüksége van egy számítógépre, amelyen Windows operációs rendszer fut, egy lokális SQL Server adatbázisra és a 3.2 A megjelenített adatok alfejezetben specifikált adatokra az alkalmazás használatához.

A 3. ábrán található az alkalmazás használati eseti diagramja. Ezen láthatóak a különböző használati esetek, amelyeknek eleget kell tennie a rendszernek. A diagram célja az, hogy átláthatóbbá tegye a felhasználó számára az alkalmazás teljes használati eset halmazát.



3.Ábra: *Használati eset diagram*

A továbbiakban a 3. ábrán látható használati eset diagram által felsorolt használati esetek elemzésére és részletes ismertetésére:

Szegmentált felvételek és az adatok megtekintése	
Felhasználói eset rövid leírása	
	A szegmentált felvételek és azokhoz tartozó adatok képernyőn való megjelenítése és böngészése
Cél	
	Az eredmények könnyed böngészhetősége
Előfeltétel	
	A bemeneti adatok el vannak mentve az adatbázisba
Siker feltétel	
	Minden fájl és az adatbázis elérhető
Kudarac	
	Nem elérhetőek a felvételek és az adatok
Szereplők	
	A felhasználó
Elvégzendő lépések	
	1. Az alkalmazás elindítása
	2. Az adatokat tartalmazó könyvtár kiválasztása
	3. Adatok importálása
	4. A megjelenítő képernyő megnyitása
	5. A keresett eredmény kiválasztása
	6. Adatok böngészése

1. Táblázat: Első használati eset

Az adatok importálása és adatbázisba való mentése
Felhasználói eset rövid leírása:
A szegmentálás eredményeinek importálása fájlból
Az betöltött eredmények adatbázisba mentése
Cél
Az adatok lokális adatbázisban való elmentése és későbbi lekérdezésekhez elérhetővé tévése
Előfeltételek
Bemeneti adatok
Siker feltétel
Sikeres fájlműveletek és adatbázis műveletek
Kudarac
Az adatok nem kerülnek be az adatbázisba
Szereplők
A felhasználó
Elvégzendő lépések
1. Az alkalmazás elindítása
2. Az adatokat tartalmazó könyvtár kiválasztása
3. Az adatokat tartalmazó CSV állomány kiválasztása

2. Táblázat: Második használati eset

A szegmentált felvételekhez tartozó bemeneti felvételek megtekintése	
Felhasználói eset rövid leírása	
A szegmentált képekhez tartozó bemeneti felvételek képernyőn való megjelenítése és böngészése	
Cél	
A képek a kijelzőn legyenek és böngészhetőek legyenek	
Előfeltételek	
Bemeneti fájlok és adatok importálva vannak	
Siker feltétel	
Sikeres fájlműveletek és adatbázis műveletek	
Kudarcc	
Nem elérhetőek a fájlok vagy az adatbázis	
Szereplők	
A felhasználó	
Elvégzendő lépések	
1. Az alkalmazás elindítása	
2. Az adatokat tartalmazó könyvtár kiválasztása	
4. A megjelenítő képernyő megnyitása	
5. A keresett eredményhez tartozó kép kiválasztása	
6. A megfelelő bemeneti csatorna kiválasztása	
7. Adatok böngészése	

3. Táblázat: Harmadik használati eset

Az eredmények szűrése	
Felhasználói eset rövid leírása	
A szegmentált képek és az azokhoz tartozó adatok szűrése különböző szűrők alapján	
Cél	
Szűrve böngészés az adatok és felvételek között	
Előfeltételek	
Bemeneti fájlok és adatok importálva vannak	
Siker feltétel	
Sikeres fájlműveletek és adatbázisból való betöltés	
Kudarcc	
Nem érhetőek el a fájlok és/vagy az adatbázis	
Szereplők	
A felhasználó	
Elvégzendő lépések	
1. Az alkalmazás elindítása	
2. Az adatokat tartalmazó könyvtár kiválasztása	
4. A megjelenítő képernyő megnyitása	
5. A szűrőhöz tartozó adattag nevének kiválasztása	
6. A szűrőhöz tartozó adattag értékének kiválasztása	
7. Adatok böngészése	

4. Táblázat: Negyedik használati eset

3.1.2 Rendszerkövetelmények

A rendszerkövetelmények leírják az alkalmazás funkcióit és az ezekhez tartozó különböző megkorlátásokat, illetve minden más releváns információt ami ezekhez tartozik. Ezek lehetnek funkcionális és nem-funkcionális rendszerkövetelmények.

3.1.2.1 Funkcionális követelmények

A funkcionális követelmények azt foglalják össze, hogy a rendszer pontosan milyen funkciókat is kell ellásson és hogyan.

Az alkalmazás indítása után az első elérhető funkció az adatok illetve eredmények importálása és adatbázisba való mentése. Ez úgy történik meg, hogy a felhasználó kiválasztja a munkakönyvtárat ahol találhatóak az adatok, majd ez után kiválasztja az adatokat tartalmazó állományt. Ezt követően az alkalmazás automatikusan importálja az adatokat, majd elmenti őket az adatbázisban,

A második funkcionalitás az adatok és felvételek megjelenítése és böngészése. Ehhez a felhasználónak már korábban importálnia kellett az adatokat. Ezt a funkciót a kijelző ablak megnyitásával éri el a felhasználó és a megnyitás alkalmával az első kép illetve a hozzá tartozó adatok egyből megjelenítődnek.

A harmadik funkcionalitás a bemeneti felvételek megjelenítése. Ehhez a felhasználó ki kell válassza az előbbi kijelző ablakon, hogy melyik bemeneti színcsatornát akarja megjeleníteni, majd ki kell választania a bemeneti képeket tartalmazó könyvtárat. Ezt követően megjelenik a kért bemeneti felvétel.

A negyedik funkcionalitás a felvételek és adatok szűrése. Ehhez a már használt kijelző ablakon ki kell választani a kritériumként szolgáló adattag nevét, illetve értékét. Ez után az alkalmazás automatikusan frissíti az elérhető felvételek és adatok listáját az adott kritériumok alapján.

3.1.2.2 Nem-funkcionális követelmények

A nem-funkcionális követelmények tartalmazzák azokat a feltételeket amelyek mellett működik a rendszer. Ha ezeket nem lehet teljesíteni a rendszer vagy nem fog megfelelően működni, vagy használhatatlanná válik.

Szükség van egy legalább HD felbontású képernyőre, mivel a kijelző ablak felülete ezen felbontásnál nagyobb felbontásokon működik optimálisan.

Továbbá legalább 7MB tárhelyre van szüksége az alkalmazásnak. Ezen túl a megjelenítendő képeket is el kell tárolni, így a képek mennyiségétől függően fel kell mérnie a felhasználónak a

szükséges plusz tárhely mennyiséget. Az alkalmazáshoz használt lokális adatbázis is tárhelyet igényel, legalább 6GB tárhelyet vesz igénybe az SQL Server és a hozzá tartozó eszközök.

3.2 A megjelenített adatok

Az alkalmazás képek illetve statisztikai adatok formájában jeleníti meg a szolgáltatott adatokat. Ezen adatoknak meg kell felelniük bizonyos feltételeknek ahhoz, hogy az alkalmazás összes funkciója elérhető legyen.

A többi adat CSV kiterjesztésű fájlok formájában támogatott. Egy előre definiált sorrendben kell legyenek az adatok a hibátlan szinkronizációhoz.

3.2.1 Statisztikai adatok

A CSV állományok tartalma a következő struktúrát kell követnie: 21 érték vesszővel elválasztva, a következő sorrendben:

1. ID - egyedi azonosító a teszteredményhez
2. AlgoID - a szegmentáló algoritmus azonosítója
3. PatientID - a páciens azonosítója
4. LearningSize - a tanulási adat mérete (pixel mennyiség/record)
5. AlgoParam - az algoritmus paramétere
6. FeatureCount - az osztályozáshoz használt jellemzők mennyisége
7. LearnDuration - a tanulás időtartama (msec)
8. TestDuration - a tesztelés időtartama (msec)
9. TN - True negative - helyesen tumormentesnek nevezett pixelek
10. FP - False positive - hibásan tumorosnak nevezett pixelek
11. FN - False negative - hibásan tumormentesnek nevezett pixelek
12. TP - True positive - helyesen tumorosnak nevezett pixelek
13. NDS - negatív osztály Dice Score [\[5\]](#)
14. NTPR - negatív osztály true positive rate [\[7\]](#)
15. NTNR - negatív osztály true negative rate [\[7\]](#)
16. NPPV - negatív osztály positive predictive value [\[7\]](#)
17. DiceScore - pozitív osztály Dice Score - ez a fő mutató [\[5\]](#)
18. PTPR - pozitív osztály true positive rate [\[7\]](#)
19. PTNR - pozitív osztály true negative rate [\[7\]](#)
20. PPPV - pozitív osztály positive predictive value [\[7\]](#)
21. Accuracy- milyen arányban volt eltalálva a ground truth [\[7\]](#)

3.2.2 Képek

A képmegjelenítés 4K felbontásra optimalizált, így a minőség megőrzése érdekében veszteségmentes tömörítésű PNG formátumú képeket jelenít meg a szoftver.

A teszteredmények amelyek megjelenítésre kerülnek, 9x16-os rácsban kell legyenek elrendezve, minden cellában egy bizonyos része kell látható legyen az agynak, azaz egy agyszeletet, ezeken pedig különböző színekkel kell jelölve legyenek az adatok.

Hasonló formátum elvárt a bemeneti felvételek esetében is.

Kékkel kell legyenek jelölve a tévesen tumormentesnek osztályozott pixelek, zölddel kell legyenek jelölve a helyesen tumornak szegmentált pixelek, pirossal pedig azok, amelyek tévesen tumornak voltak osztályozva.

4. A rendszer architektúrája

A rendszer több komponensből áll és moduláris architektúrát követ. A tervezés során több szempontból is erre a kialakításra esett a választás:

- Bővíthetőség
- Átláthatóság
- Hibakeresés és javítás
- Módosíthatóság

Fontos szempont volt olyan kialakítást választani, amely lehetővé teszi azt, hogy a későbbiekben változó felhasználói igényeket is el tudjon látni a rendszer amennyiben tovább szeretné fejleszteni valaki. A használt architektúra lehetővé teszi bármelyik komponens bővítését úgy, hogy a többi komponenst minimálisan szükséges módosítani, bizonyos esetekben ezt automatikus módon meg tudja oldani valamelyik keretrendszer is.

Az átláthatóság is egy fő szempont volt a tervezés során, ugyanis egy hasonló projektnél nagyon fontos az, hogy minden folyamat átlátható legyen, így nem csak a továbbfejlesztés, de maga a fejlesztés is gördülékeny lehet és rendszerezett. Továbbá a fokozott átláthatóság segít kiküszöbölni a felesleges függőségeket és komplexitást a szoftver olyan részeinél, ahol nem jó megoldás túl sok mindent összevonni.

A hibák keresése és javítása szempontjából is rendkívül előnyös a használt többkomponensű architektúra. Sokkal egyszerűbb feltárni és megoldani a problémákat amennyiben izolálva vannak egy bizonyos környezetben. Külön komponensek esetében a függőségek jól átláthatók a komponensek közötti összeköttetések által, így könnyebb egy-egy hibát kiváltó okot a jó helyen keresni.

Akárcsak a bővíthetőség, a módosíthatóság is kulcsfontosságú tervezési szempont volt, ugyanis rengeteg mindenben szükséges lehet változtatni a jövőben, ahogy változnak a felhasználási esetek különböző részletei. Továbbá mint az átláthatóság, ez is leegyszerűsíti illetve felgyorsítja a fejlesztés különböző fázisait.

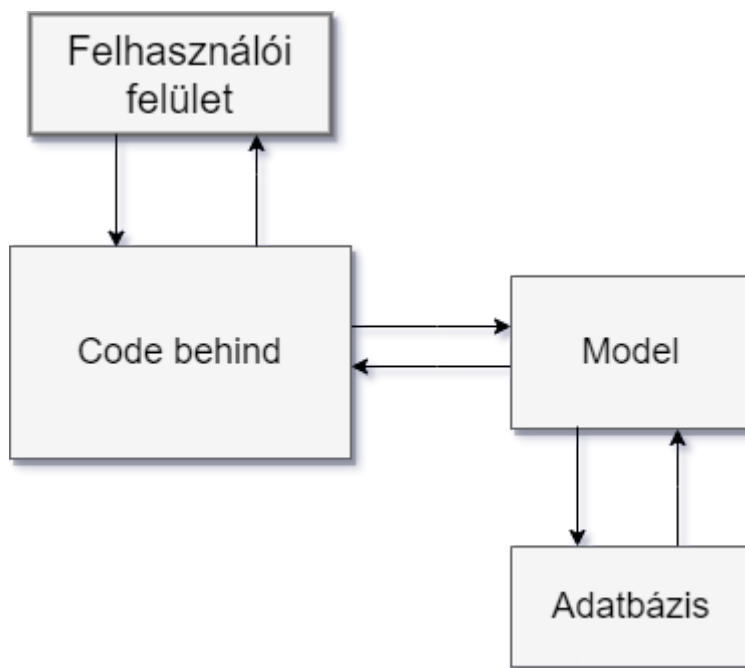
A rendszer komponensei közötti összeköttetések a mellékelt blokkséma alapján valósulnak meg. Fontos részlet az, hogy az összes kapcsolat a komponensek között kétirányú, azaz a komponensek képesek adatcserére egymás között. Ez azért számít jelentősnek, mert egy hasonló alkalmazás esetében az adatfolyam mindkét irányban lehetséges kell legyen, így a legalsó (adatbázis) rétegtől kezdve a legfelső (felhasználói felület) rétegig és fordítva is lehetséges az adatok mozgatása.

A Microsoft által biztosított keretrendszerek, könyvtár csomagok és adatbázis kezelő megoldások együttes használata rendkívül megkönnyítette a fejlesztést. Ezen fejlesztői eszközök és segédkönyvtárak úgy voltak kifejlesztve, hogy a megfelelő verziókat használva optimálisan képesek legyenek egymással kommunikálni.

Mivel a legtöbb ezek közül eredetileg a .NET keretrendszer része volt valamikor viszonylag mai napig egyszerű úgy együtt használni őket mintha egyetlen szoftverfejlesztői csomag részei lennének. Így viszonylag könnyű a kapcsolatokat és függőségeket kezelni a komponensek között, úgy hogy megmaradnak a különálló komponensek által nyújtott modularitás előnyei is.

Bár a projekt fájl struktúráját nézve a felhasználói felület *xaml* kiterjesztésű mark-up fájlok és a Code behind, azaz üzleti logika réteg *xaml.cs* kiterjesztésű fájlok összetartoznak, külön komponensekként viselkednek a kötött tulajdonságok miatt. A felhasználói felületen az adatok automatikusan szinkronizálva vannak, ezért a code behind résznél nincs szükség arra, hogy a felhasználói felületet bármilyen módon is manipuláljuk.

A 4. ábrán látható a szoftver architektúrájának blokksémája.



4. Ábra: A szoftver architektúrájának blokksémája

4.1 A rendszer komponensei

- Felhasználói felület
- Code behind
- Model
- Adatbázis

4.2 A komponensek modularitása és függőségek

A modularitás ellenére fennállnak bizonyos függőségek a komponensek között. Az egyik ilyen az említett üzleti logika és felhasználói felület közötti függőség. Egy másik függőség a Model és az adatbázis között jön létre. Mivel a Model automatikusan generálódik az adatbázis struktúrája és relációi alapján a Model osztályai és az azok közötti kapcsolatok teljes mértékben az adatbázistól függenek. Az üzleti logika részben valósul meg az importált adatok adatbázisba való mentése illetve betöltése amikor megjelenítjük az adatokat a felhasználói felületen. Ezen funkciók nagy mértékben függenek a Modeltől, így egy indirekt függőség alakul ki az adatbázis és az üzleti logika között.

4.3 A felhasznált technológiák specifikációi

Az alkalmazás fejlesztéséhez több Microsoft által biztosított technológiát illetve könyvtár csomagot vagy keretrendszert is használtam. Az alábbiakban ismertetem mindenik verzióját, specifikációit illetve a szerepét a szoftveren belül.

4.3.1 .NET keretrendszer

A .NET keretrendszer 4.8-as verzióját használtam, mivel az volt a legújabb verzió amikor a fejlesztést kezdtem. Ez a verzió nem hozott annyi újdonságot mint néhány korábbi, viszont volt néhány elsősorban a WPF magas DPI-s képernyőkre optimalizált részeinél olyan változtatás ami potenciálisan hasznos lehet bizonyos esetekben, ahol fontos a megjelenítésnél a magas DPI.

A .NET keretrendszer biztosítja azokat a könyvtár csomagokat és natív megoldásokat, amelyek lehetővé tették a WPF-el fejlesztett felhasználói felület mögött rejlő interakciók és egyéb funkciók kivitelezését.

Többek között segített egyszerű módon implementálni az adatok importálását CSV állományokból, kezelni különböző könyvtár vagy fájl elérési útvonalakat, szinkronizálni a megjelenített adatokat a felhasználói felületen. Továbbá lehetőséget nyújtott a megjelenítendő adatok formázására is.

4.3.2 Entity keretrendszer

Az Entity keretrendszer segített abban, hogy könnyed módon elérhessem az adatbázist és C# kód segítségével tudjak lekérni adatokat, vagy épp elmenteni őket az adatbázisban.

Az Entity keretrendszer 6-os verzióját használtam, jelenleg ez a legújabb, és ez a leggyakrabban használt a .NET keretrendszer 4-es verziónál újabb verziói esetében

Database first módszert alkalmaztam, amely azt jelenti, hogy először létrehoztam egy adatbázist a Microsoft SQL Serveren, majd ezt összekötve a projekttel az Entity keretrendszer kigenerálta az entitásokat és az ezek relációit tartalmazó modellt.

4.3.3 WPF

A WPF szerves részét képezi az alkalmazásnak. A teljes felhasználói felületet WPF segítségével hoztam létre. Ami a felhasználói felület kinézetét illetve amit csak lehetett megoldottam a WPF markdown nyelvét, az XAML-t használva, azaz procedurális módon.

A WPF 4-es verzióját használtam, ez az egyik legújabb verzió. Rengeteg hasznos eszközt nyújt felhasználói felületek tervezésére, fejlesztésére, tesztelésére. Egy vizuális interaktív tervezői felületet biztosít, ami nagyon felgyorsítja egy felhasználói felület elkészítését egy egyszerű szövegszerkesztőhöz képest.

4.3.4 Microsoft SQL Server

Az alkalmazás adatbázisa a Microsoft SQL Server által biztosított lokális adatbázis szerveren került megvalósításra.

A Microsoft SQL Server 2019-es Express verzióját használtam, amely nem tartalmaz annyi funkciót mint a többi verzió, viszont cserébe könnyedebb és ingyenes.

Az SQL Server szoftvercsomagja teljesen lefedte az alkalmazáshoz szükséges adatbázissal kapcsolatos szükségleteket. Úgy hosting, menedzsment, tervezés, tesztelés mint adatbázis használat terén is.

4.3.5 C# programozási nyelv

A alkalmazáslogika fejlesztéséhez a C# programozási nyelvet használtam, egész pontosan a 8-as verzióját, amely a legújabb amelyet támogat a .NET keretrendszer 4.8-as verziója. A C# 8 sok új és hasznos megoldást hozott a fejlesztők számára, amelyek segítségével jelentősen lerövidült néhány gyakran használt technika vagy módszer implementációja. Továbbá a C# a .NET keretrendszer által támogatott nyelvek egyik legnépszerűbb lehetősége, így rendkívül előnyös választásnak bizonyult a fejlesztéshez.

4.4 Használt eszközök

Az alkalmazás komponenseit a Microsoft által biztosított ingyenes eszközökkel illetve fejlesztői környezetével valósítottam meg.

4.4.1 Microsoft Visual Studio

A fejlesztés és tesztelés során legtöbbet használt eszköz a Microsoft Visual Studio 2019 Community Edition volt. Ez egy integrált fejlesztői környezet azon verziója amelyet a Microsoft diákok, egyetemisták, nyílt-forráskódú projektek fejlesztői és magánszemélyek számára hozott létre.

Elsősorban a forráskód szerkesztésében segített ez az eszköz, de a tesztelésben és a felhasználói felület tervezésében is fontos szerepet játszott. Továbbá a verziókövető rendszert is a fejlesztői környezet felhasználói felületén keresztül használtam egy kiegészítő modult alkalmazva.

A fejlesztés mellett rendkívül sokat segített a tesztelésben meg hibajavításban is a Visual Studio. Rendkívül kifinomult és egyszerűen használható beépített hibakeresővel rendelkezik, amely olykor meglehetősen lerövidíthet egy amúgy hosszas és komplex hibakeresési folyamatot.

4.4.2 Microsoft SQL Server Management Studio

Az adatbázis megtervezésére, létrehozására és menedzselésére a Microsoft SQL Server Management Studio 2018-as változatát használtam, amely egy adatbázis management eszköz szintén a Microsofttól. Ezt a 2019-es Microsoft SQL Server csomagjával együtt használtam.

Fontos szerepet játszott abban, hogy könnyedén böngészhessenek az adatbázisban található adatok között és megtervezhessem az adatbázis tábláit, azok struktúráját és a táblák közötti relációkat. Továbbá megkönnyítette azt is, hogy utólagos módosításokat is megejtsek az adatbázis tábláin.

4.4.3 Github

A forráskód fájljait a Github nevezetű verziókövető rendszer segítségével szinkronizáltam egy online felhő alapú adattárba, így biztosítva azt, hogy egy esetleges hardver hiba esetén is megmarad a forráskód, illetve korábbi verziókat is el tudok érni amennyiben vissza szeretnék lépni valamilyen oknál fogva.

A Github szintén a Microsoft tulajdonát képezi, egy GIT alapú verziókövető rendszer amely az évek során egy komoly szoftverfejlesztői platformmá nőtt ki magát, amely már projektmenedzsment eszközöket is biztosít.

5. Részletes tervezés

5.1 A szoftverkomponensek részletes leírása

A szoftver komponenseinek tervezése és implementálása, majd beüzemelése egy több lépéses folyamat volt. Ezt részletezve szeretném bemutatni az alábbiakban a különböző komponensek részletes leírásával, illetve az ezekkel kapcsolatos alkalmazott megoldások felsorolásával és magyarázatával.

5.1.1 Felhasználói felület

A felhasználói felület tervezése során szempont volt az, hogy egyszerű és egyértelmű legyen az alkalmazás használata és bárki aki az alkalmazás használatához szükséges előfeltételeket teljesíti képes legyen könnyedén, problémák nélkül használni az alkalmazást.

Ehhez elsősorban a WPF által nyújtott egyszerű UI elemeket használtam építőelemekként. Ezeket felhasználva megterveztem egy alap felhasználói felületet amelyen elhelyeztem a szükséges gombokat, címkéket, szöveges mezőket, illetve a különböző paneleket amelyek az említett felhasználói felület elemek tárolóiként szolgáltak.

Másodsorban minden egyes gombhoz hozzárendeltem egy-egy függvényt amely ellátja a gombnak szánt funkcionalitás szerepét. Ezek az eseménykezelő függvények gombnyomásra hívódnak és a gombok számára specifikus adatokat adják át a futtatási kontextus mellett az üzleti logikát tartalmazó osztálynak.

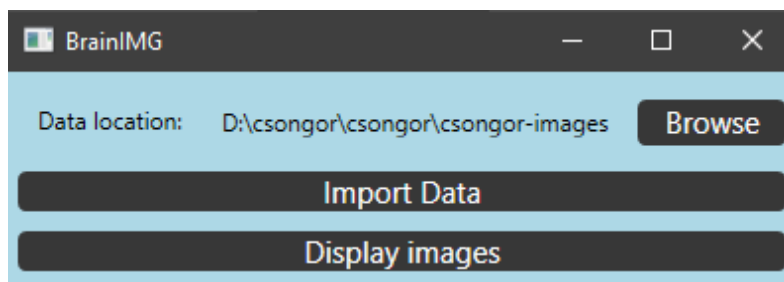
Továbbá létrehoztam a képeket megjelenítő felhasználói felület elemet is, amelyet úgy méreteztem, hogy képernyőmérettől függetlenül eredeti méretben jelenítse meg a megjelenítendő képeket, így amennyiben kisebb a képernyő 4K-nál az egér görgőjével illetve az oldalsó görgő panelekkel lehet navigálni a képen különböző irányokban. Ez fontos volt mivel a megjelenítendő képek esetében fontos a minőség megőrzése, amit nagyon rontana a képek átméretezése, legyen szó zsugorításról, vagy széthúzásról.

Ezt követően hozzáadtam egy panelt amelyen megjelenítettem a képeken található színjelölések magyarázatát, azaz azt, hogy milyen színű pixel milyen adatot jelent az agyszeleteken: a kék pixelek olyan pixelek, amelyek hibásan tumoros szövetnek voltak osztályozva; a zöld pixelek olyan pixelek, amelyek tévesen tumormentes szövetnek voltak osztályozva; a piros pixelek pedig azok, amelyek helyesen tumoros szövetnek voltak felismerve.

5.1.1.1 Főmenü

A Főmenü az első képernyő, amely megjelenik amikor elindul az alkalmazás, ez a kiindulópontja és az irányítóközpontja is az alkalmazásnak. Itt történik meg az adatokat tartalmazó könyvtár vagy meghajtó kiválasztása. Továbbá itt lehetséges kiválasztani a CSV állományt, amely tartalmazza a betöltendő adatokat. Ezen kívül az adatok megjelenítésére használatos ablak is innen érhető el.

Az 5. ábrán található a főmenü ablak felhasználói felülete.



5. Ábra: Az alkalmazás főmenüjének felhasználói felülete

A Főmenü ablak tervezése során dinamikus méretezés mellett döntöttem, így nem foglal túl sok helyet a képernyőn a viszonylag kis méretű felhasználói felület. A dinamikus méretezés azt foglalja magába, hogy amennyiben a kiválasztott könyvtár neve hosszú és a megjelenítéshez szükséges hely több lesz mint korábban rendelkezésre állt alaphelyzetben az ablak szélessége automatikusan igazodni fog az új mérethez, így dinamikusan megváltoztatja saját szélességét arra a méretre, amelybe tökéletesen belefér minden elem a felhasználói felületen.

Az ablakon megjelenített elérési út megjelenítése kötési tulajdonság segítségével volt megvalósítva. Miután a felhasználó kiválasztotta a könyvtárat a code behind fájlban megkapja az adott tulajdonság az elérési utat és ez automatikusan szinkronizálódik a felhasználói felületen

megjelenített szövegmező tartalmával. Az elérési útvonal megadásának pontossága érdekében csak a Windows által biztosított fájlböngésző segítségével adható meg az elérési útvonal,

5.1.1.2 Kijelző ablak

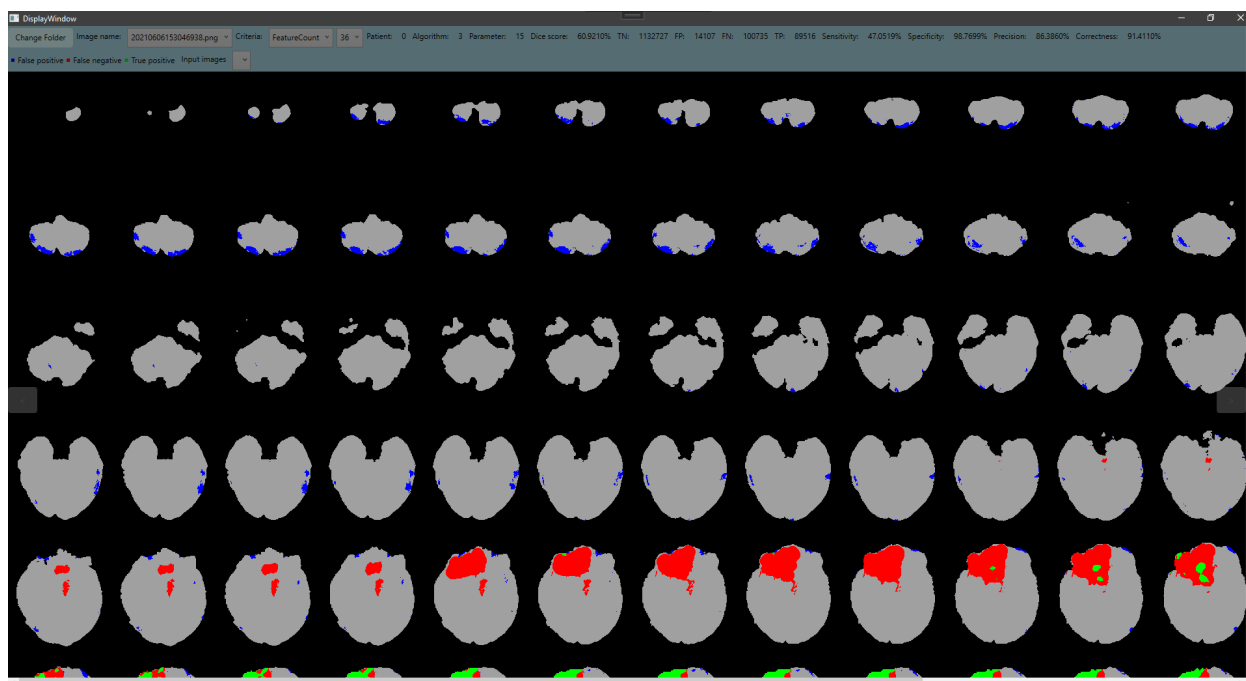
A kijelző ablak az a része az alkalmazásnak, amelyen a képek illetve az ezekhez tartozó adatok megjelenítése történik. A képek minden esetben eredeti felbontásban kerülnek megjelenítésre, azért, hogy a minőségük ne romoljon a megjelenítés során. Hasonló célokra használt képek esetén minden pixel épsége kritikus, így a fájlok veszteségmentes PNG kiterjesztésűek.

Amennyiben a használt számítógép kijelzője nem támogatja az adott méretű felbontást a kép ugyanúgy az eredeti felbontásban fog megjelenni, viszont egyszerre csak annyi része lesz látható, amennyi még elfér a képernyőn. A többi rész az egér görgőjét használva vagy az oldalsó görgő paneleket használva tekinthető meg.

A képet leszámítva minden elem ezen a képernyőn átlátszó bizonyos mértékben, azért, hogy ha görget a felhasználó ne takarja el teljesen a képet sem a felső panel sem az oldalsó gombok.

Jobb és bal oldalt található két gomb, a jobb oldali a könyvtárban található következő képet jeleníti meg, a legutolsó esetében pedig visszatér a legelsőhöz. A bal oldali gomb a könyvtár előző képét jeleníti meg, az első esetén pedig az utolsó képre ugrik.

A 6. ábrán megtekinthető a kijelző ablak felhasználói felülete egy FullHD felbontású monitoron amint teszt eredményeket jelenít meg.



6. Ábra: A kijelző ablak felhasználói felülete

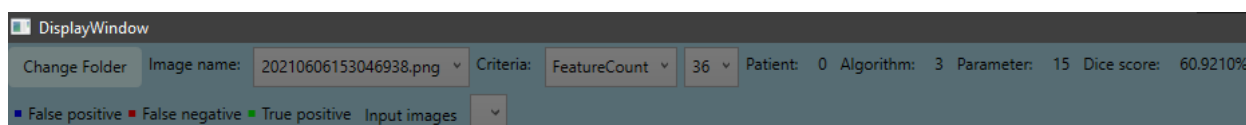
A felső panelen a megjelenített adatok mellett még más fontos elemek is helyet kaptak. Az egyik egy gomb amely segítségével munkakörnyvtárat válthatunk. Ez hasznos lehet abban az esetben ha még meg szeretnénk tekinteni más eredményeket is anélkül, hogy vissza kelljen lépni a főmenübe.

Egy másik elem egy legördülő lista a képek neveivel. Itt azt is kiválaszthatjuk a listából, hogy melyik képet és hozzá tartozó adatokat szeretnénk látni a képernyőn. Rendkívül hasznos lehet arra, hogy böngésszünk a képek listájában, vagy ha egy bizonyos képet szeretnénk elérni a könyvtárból.

Két másik elem együttesen tesz lehetővé egy igencsak fontos funkcionalitást, az adatok szűrését. A képválasztó legördülő listától jobbra még két lista található. Ezek közül az elsővel lehet kiválasztani a szűrés kritériumához használt adattag nevét, a másikkal pedig az adattag értékét. Ezt követően az elérhető képek listája frissül a kritérium alapján.

Az utolsó említésre méltó elem egy utolsó legördülő lista az Input images fejléc melletti. Ez segítségével kiválaszthatunk egy bemeneti színcsatornát, ezt követően az alkalmazás megjeleníti az épp kiválasztott képhez tartozó kiválasztott színcsatorna felvételeit.

A kijelző ablak felső panelja kötelebből is megtekinthető a 7. ábrán.



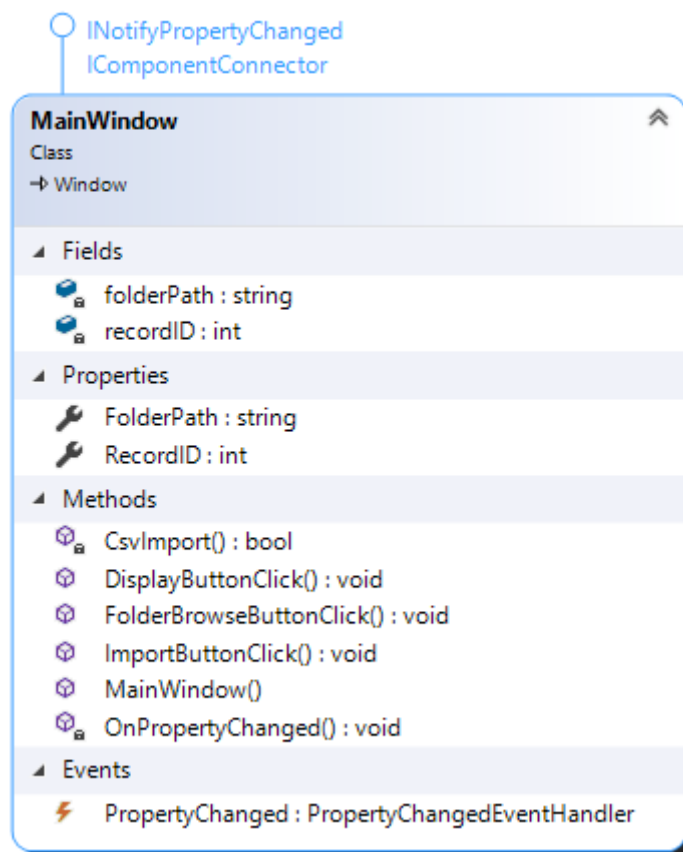
7. *Ábra: A kijelző ablak felső panelja közelről*

5.1.2 Code behind

A code behind fájlokban valósul meg az üzleti logika. Ezek a fájlok szorosan csatolódnak a felhasználói felület vizuális részét megvalósító XAML markup fájlokhoz. Az adatkötéseknek hála ennek ellenére jól elválasztható a két típusú fájl szerepe. A code behind fájlokban található függvények és metódusok fontos szerepet játszanak, ugyanis ezekben történik meg a CSV fájlokból történő adatimportálás és az adatok feldolgozása. A feldolgozás során minden adat elmentődik az adatbázisba. Továbbá a képek elérése és megjelenítése is itt valósul meg.

A MainWindow osztály egyik legfontosabb függvénye a *CsvImport()*. Ebben a függvényben az alkalmazás megnyitja a kiválasztott CSV állományt és soronként beolvassa az adatokat. A sorokban található adatokat vesszőnként elválasztja, majd a megfelelő típusú alakítja. Ezt követően minden adatot elment az adatbázisba. A kritikus műveletek egy try catch szerkezeten belül történnek meg, így amennyiben kivétel keletkezik az alkalmazás képes ezt kezelni és így nem okoz leállást egy esetleges hiba. Az adatbázis műveletek Entity keretrendszeres megoldásokkal történnek, így egy külön kontextusként meg lehet adni a függvényen belül az entitások elérését és ez által izolálva van az adatbázis elérés. Ez azért előnyös, mert így az adott kód-blokkból kilépve a garbage collector felszabadíthatja a lefoglalt erőforrásokat és nem kell manuálisan kezelni az adatbázis kapcsolatot, műveleteket SQL szintjén illetve sokkal egyszerűbb így az adatbázissal kapcsolatos kivételek kezelése, valamint hibák megtalálása, ezek javításáról nem is beszélve.

A 8. ábrán található a főmenühoz tartozó osztály osztálydiagramja az osztály adattagjaival, metódusaival, eseményeivel és az implementált interfészekkel.



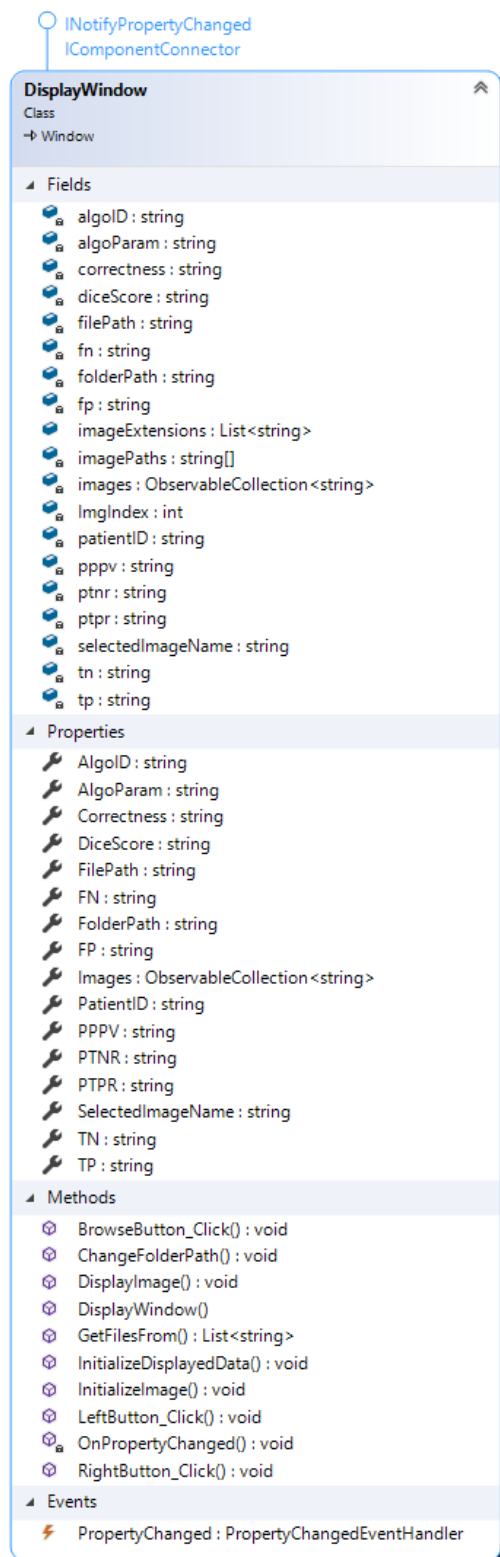
8. Ábra: A főmenühöz tartozó osztály osztálydiagramja

A DisplayWindow osztályt a MainWindow osztály példányosítja amikor a felhasználó megnyitja a képeket és adatokat megjelenítő ablakot. A MainWindow dependency injection során átadja a már kiválasztott könyvtár elérési útvonalát a DisplayWindow konstruktorának. Így ahogy megnyílik az ablak az osztály be tudja tölteni az első képet a könyvtárból és a hozzá tartozó összes adatot le tudja kérni az adatbázisból. Ez a külső függőség átadása rendkívül praktikus, ugyanis legtöbb esetben a könyvtár elérését első körben megadja a felhasználó ahhoz, hogy a CSV állományból importálja az adatokat. Így mivel már ez az adat elérhető célszerű úgy létrehozni a DisplayWindow osztályt, amelynek szüksége van rá már a konstruktorban, hogy a konstruktorának egy paramétere a szóbanforgó elérési út egy karakterlánc formájában.

Az osztály megannyi tulajdonsággal rendelkezik. Ezek közül a karakterlánc típusúak célja az, hogy az adatkötés során ezekre lesznek rákötve a felhasználó felületen megjelenített adatok.

Továbbá egy ObservableCollection tároló is található az osztályban. Ennek az a célja, hogy a képek listáját tárolja el. Azért van szükség ObservableCollection típusú tárolóra mert az egy olyan tároló, amely lehetővé teszi azt, hogy valós időben frissüljenek a lista elemei abban az esetben ha megváltozna a tartalma.

A 9. ábrán található a kijelző ablakhoz tartozó osztály osztálydiagramja az osztály adataival, módszereivel, eseményeivel és az implementált interfészekkel.



9. Ábra: Az kijelző ablakhoz tartozó osztály osztálydiagramja

Az osztály fontosabb függvényei és metódusai a következők:

- GetFilesFrom
- ChangeFolderPath
- InitializeImage
- DisplayImage
- InitializeDisplayedData

A GetFilesFrom függvény visszatérít egy karakterlánc listát a fájlok neveivel a megadott elérési úton található könyvtárból.

A ChangeFolderPath metódus betölti az előbbi függvény segítségével a fájlok listáját, majd ezzel feltölti a legördülő fájllista tartalmát a felhasználói felületen. Ezt követően kiválasztja a legelső képet.

Az InitializeImage metódus ellenőrzi egy elérési útról, hogy található-e ott kép, ha igen akkor meghívja a DisplayImage és az InitializeDisplayedData metódusokat.

A DisplayImage metódus egyszerűen betölti és megjeleníti az adott elérési úton található képet.

Az InitializeDisplayedData metódus lekéri az adatbázisból az összes adatot ami a megjelenített képhez tartozik és beállítja a kötött tulajdonságokat amelyek frissítik a felhasználói felületet.

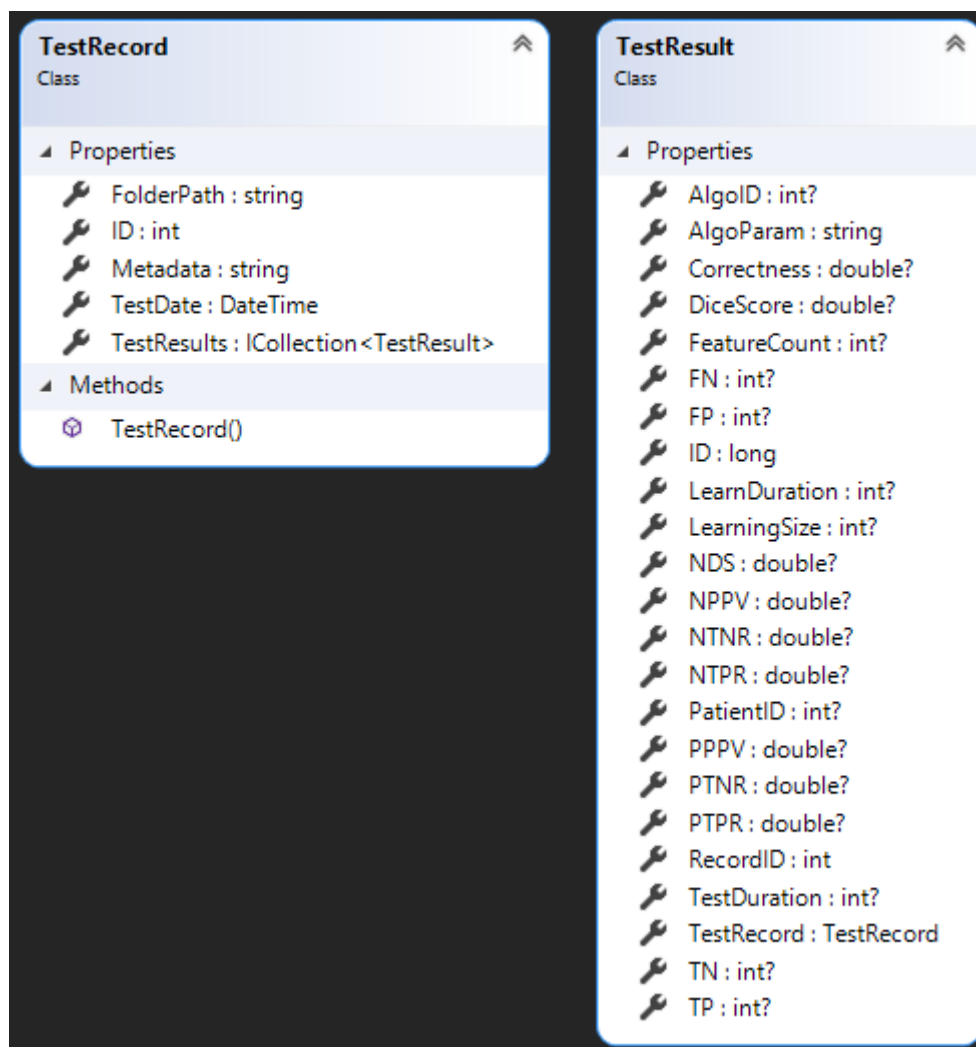
Mindkét osztály implementálja az INotifyPropertyChanged nevű interfészt. Ez egy olyan eseményvezérelt megoldást nyújt amely segítségével a code behind rész képes jelezni a megjelenített felhasználói felület elemeknek azt ha változott egy kötött tulajdonság, vagy fordítva. Lényegében ez az interfész implementációja biztosítja a kötött tulajdonságok automatikus szinkronizációját.

5.1.3 Model

A Model osztályai szorosan követik az adatbázis tábláinak és relációinak struktúráját. Ezen osztályok az Entity keretrendszer által lettek automatikusan létrehozva. Módosítások esetén a keretrendszer képes frissíteni ezeket az osztályokat a Visual Studio vizuális felülete segítségével, viszont ha vannak hivatkozások ezekre az osztályokra a kódban szükséges módosítani a kódot úgy, hogy találjon a módosított entitás osztályok struktúrájához.

Az adatbázisbeli táblák oszlopainak típusai valamelyest eltérnek a model osztályainak típusaitól, nem minden adatbázisbeli típus neve ugyanaz a .NET keretrendszerben, egy ilyen eltérés a .NET esetében *double* és adatbázis esetében *float* típus.

A 10. ábrán találhatóak a modelhez tartozó osztályok osztálydiagramjai az osztályok adattagjaival.



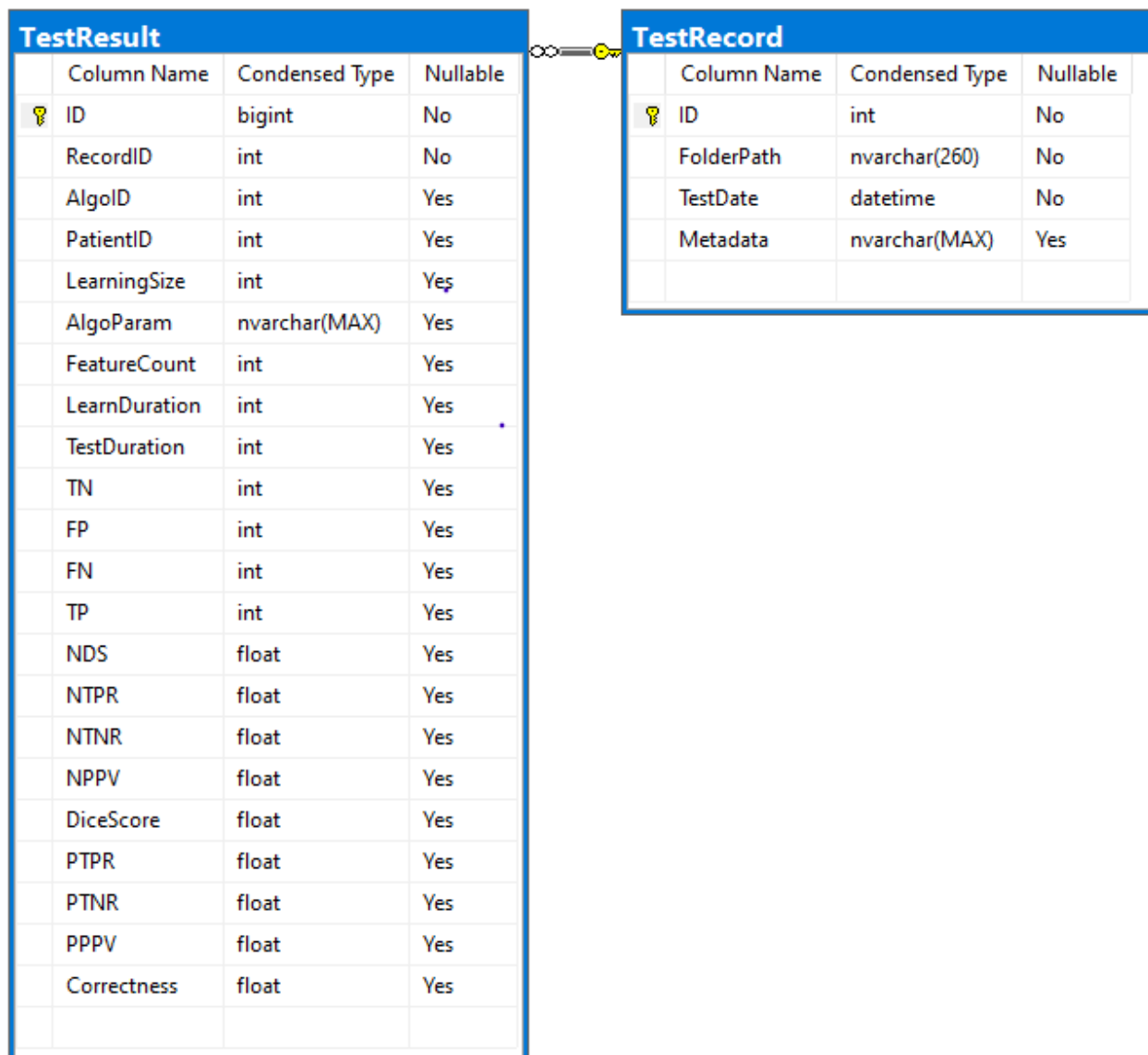
10. Ábra: Az adatbázist és alkalmazást összekötő model réteg osztálydiagramja

5.1.4 Adatbázis

Az adatbázis két táblából áll. A TestRecord táblában tároljuk egy bizonyos adat import részleteit, azaz egész pontosan azt, hogy mikor történt az adott import, milyen elérési úton érte el az alkalmazás az adatokat és egy plusz adatcella, amelybe különböző egyéb releváns információk kerülhetnek amik még esetleg hasznosak lehetnek ahhoz, hogy elkülönítsünk egy importot.

A TestResult táblában tároljuk az importált adatokat. Ide kerül elmentésre egy egyedi ID-ként a képek neve, illetve az összes képhez tartozó adat, amelyekről részletesen a [2.6 A megjelenített adatok](#) résznél ismertettem. Továbbá egy RecordID cella idegen kulcsként szolgálva hivatkozik az arra a bizonyos importra a TestRecord táblában amely során az adott képhez tartozó adatokat elmentettük az adatbázisba.

A 11. ábrán található az adatbázis diagram a táblákkal és relációkkal.



11. Ábra Adatbázis tábláinak és relációinak diagramja

5.2 Adatfolyam

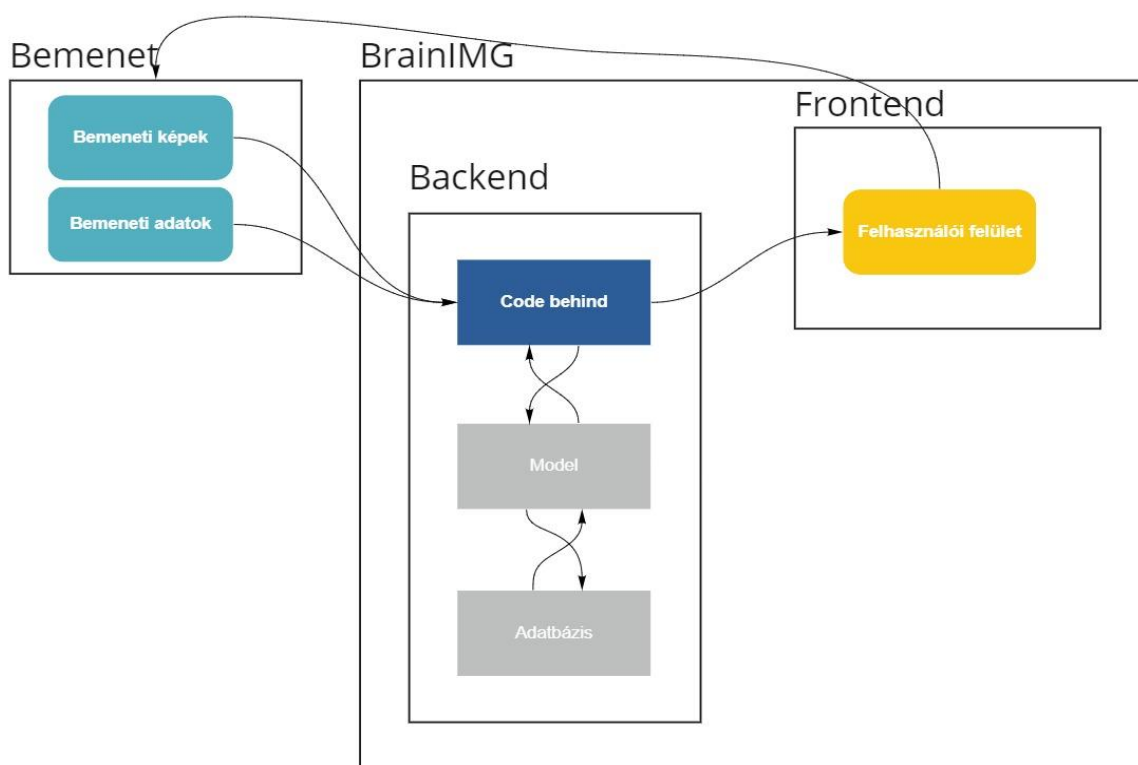
Ahogy az adatfolyam ábrán is látható különböző komponensek között különböző formában történik az adatsere. A Felhasználói felület úgy bemenetként mint kimenetként szolgál. Bemenetként indirekt módon, mivel innen kell kiválasztani az adatokat tartalmazó fájlt illetve a képeket tartalmazó könyvtárat ami után a a code behind elvégzi a kellő műveleteket ezekkel a bemenetekkel.

Továbbá a felhasználói felületen jeleníti meg az alkalmazás a betöltött adatokat és képeket, így ez a fő kimenete az alkalmazásnak

A code behind rész a Modellel és felhasználói felülettel kommunikál azon kívül, hogy feldolgozza a bemenetként kapott fájlokat. A modellel kétirányú adatfolyamot valósít meg, itt mindkét irányban történhet adatcsere, azaz lehet épp el akar menteni valamit az alkalmazás, de az is lehetséges, hogy egy korábban mentett adatot kell lekérdezni az adatbázisból a modellen keresztül.

A Model és adatbázis között is ugyanúgy kétirányú adatkapcsolat van mint a Model és a code behind rész között. Ez azért van mert a Modelt és az adatbázist az Entity keretrendszer köti össze és a működéséhez kettős irányú adatcsere szükséges.

A 12. ábrán megtekinthető az említett adatfolyamokat ábrázoló adatfolyam diagram.



12. Ábra: Az alkalmazás adatfolyam diagramja

6. Üzembe helyezés

6.1 Üzembe helyezési lépések

5.1.1 A szoftver telepítése a mellékelt telepítővel, a telepítő során talapítani kell a javasolt szoftvercsomagokat, amelyekhez internetkapcsolat is kell

5.1.2 A telepítőnek telepítenie kell a Microsoft SQL Server 2019 ingyenes Express verzióját

5.1.3 Meg kell nyitni az SQL Server Management Studiot, ahol létre kell hozni egy új relációs adatbázist amelyben a forráskódban található SQL script segítségével létrehozható a tábla struktúra. Ebben segíthet az SQL Server Management Studio használati útmutatója ami elérhető a szoftver megfelelő menüjében.

5.1.4 Az Application Files nevű mappában található BrainIMG.exe.config.deploy nevű XML fájlt szövegszerkesztővel meg kell nyitni, majd a *connectionStrings* nevű elem alatt található elemnél a *data source* részhez be kell írni az adatbázis szerver nevét, illetve a *initial catalog* részhez be kell írni az adatbázis nevét.

5.1.5 Használható az alkalmazás

6.2 Felmerült problémák és megoldásaik

Egy felmerült probléma az volt, hogy a lokális adatbázis miatt nehézkes a beüzemelése a rendszernek. Viszont a telepítési varázsló internet kapcsolaton keresztül képes telepíteni a szükséges szoftvert a lokális adatbázishoz.

Egy másik probléma az volt, hogy különböző felbontású képernyőket is támogasson a szoftver. Erre az volt a megoldás, hogy a felhasználói felület dinamikusan újraméretezi önmagát és a képek megjelenítése mindig eredeti méretben történik, görgethetőséggel kiegészítve amennyiben kisebb a képernyő felbontása a kép felbontásánál.

6.3 Kísérleti eredmények

A rendszer tesztelése során rengeteg észrevétel született. Egyik ilyen volt az, hogy HD felbontás alatti képernyők esetében a felhasználói felület nem működik optimálisan, így követelménnyé vált a megfelelő minimum felbontás.

Egy másik eredmény volt az, hogy amennyiben kép teljes felbontásban megjeleníthető görgők nélkül akkor szükségtelen a felhasználói felület elemeinek elhalványítása, amely azt a cél szolgálta, hogy a görgetés során ne takarják el a képet.

7. A rendszer felhasználása

Az alkalmazás elindítása után célszerű azzal kezdeni, hogy kiválasztjuk a célkönyvtárat, ahol a megjelenítendő kimeneti képek illetve az eredményeket tartalmazó CSV állomány található.

Ezt követően annak függvényében, hogy már korábban megtörtént vagy sem importálása azoknak az adatoknak, amelyeket meg szeretnénk jeleníteni két választásunk van. Amennyiben

szükséges az adatok importálása az Import data nevű gomb segítségével kiválasztjuk az adatokat tartalmazó CSV állományt, ez után az alkalmazás importálja a kért adatokat az adatbázisba.

Ha már importálva vannak az adatok akkor a Display images nevű gombot kell használnunk a kijelző ablak megnyitására. Itt a megadott munkakönyvtár legelső képe lesz megjelenítve és a felső panelen a hozzá tartozó adatok, amelyek importálva voltak az adatbázisban.

A képek közötti navigáció több módon is megtörténhet. Egyik mód a nyíl billentyűk használata, egész pontosan a jobb és bal nyilak. A jobb nyíl a következő és a bal pedig az előző képet jeleníti meg. Hasonló módon működnek a képernyő jobb és bal oldalán található gombok. Amennyiben elértünk a könyvtár végére és a következő képre szeretnénk navigálni a legelső képet jeleníti meg az alkalmazás. Hasonlóan ha a legelső kép van megjelenítve és az előzőre kíváncsi a felhasználó akkor a legutolsó kép kerül megjelenítésre.

Egy másik eszköz a navigációra a felső panelben található legördülő lista, amely az összes kép nevét tartalmazza. Ez segítségével kiválaszthatjuk név szerint pontosan, hogy melyik képet és a hozzá tartozó adatokat szeretnénk megjeleníteni.

Ennek a listának a tartalmát módosíthatjuk különböző kritériumok alapján, egyik ilyen az, hogy melyik páciens eredményei legyenek a listában, egy másik pedig az, hogy csak egy bizonyos algoritmus eredményei közül választhassunk.

Ehhez a mechanizmushoz két másik legördülő listát kell használnunk. Egyikben kiválasztjuk, hogy melyik adat szerint szeretnénk szűrni az eredményeket, a másik listában pedig azt választjuk ki, hogy melyik értéke legyen az adott adatnak a szűrőben.

A Change folder nevezetű gomb segítségével cserélhetjük a munkakönyvtárat. Ez hasznos lehet abban az esetben ha több könyvtárhoz tartozó adat van már az adatbázisba elmentve és szeretnénk egy másik könyvtárban is böngészni.

Továbbá egy másik legördülő lista segítségével megjeleníthetjük az adott eredményhez tartozó bemeneti képeket is, 4 különböző színcsatornát és az alapigazságot tartalmazó képet.

8. Következtetések

A Microsoft által biztosított ingyenes és nyílt forráskódú technológiák használata rendkívül leegyszerűsítette a fejlesztést és lehetővé tette a moduláris kialakítást úgy, hogy viszonylag ugyanannyi időbe telt a fejlesztés mintha monolitikus kialakítású lett volna a projekt.

A több komponenses kialakításnak számos előnyét kihasználhattam a fejlesztés során és ezeknek is köszönhetően az átlagosnál kevesebb időt kellett felesleges hibakereséssel töltenem. Továbbá könnyebb volt jól elhatárolt komponensenként megtervezni az alkalmazás különböző

funkcióit ellátó részeket, mert így mindig tiszta volt minden egyes feladatkör és az azt ellátó felelős komponens relációja.

Átláthatóság és rendszerezettség szempontjából is előnyös volt hasonló struktúrát követve kialakítani az alkalmazás architektúráját, jelentősen könnyebb volt eligazodni és új funkciókat fejleszteni.

Az alkalmazást használva sokkal könnyebben elérhetőek az adatok illetve az azokhoz tartozó feltérképezett agyszeleteket tartalmazó szegmentált képek. A szoftver használatával könnyen rendszerezhetővé válnak a különböző algoritmusokhoz vagy akár páciensekhez tartozó teszteredmények. Az alkalmazás teljes repertoárját felhasználva a felhasználó számos módon böngészhet az adatok között, ez a sokoldalúság miatt viszonylag felhasználóbarát a felület.

8.1 Továbbfejlesztési lehetőségek

Egy továbbfejlesztési lehetőség lenne az, ha az alkalmazás segítségével különböző osztályozó modelleket lehetne tanítani és tesztelni, ehhez integrálni lehetne már működő ilyen rendszereket a szoftverbe és így akár automatikusan összeköthető lenne egy teszt lefutása és az eredményeinek adatbázisba való szinkronizálása.

Egy másik lenne az, ha valamivel több szűrési és rendezési kritérium lenne biztosítva a megjelenítési képernyőn, például több különböző kritérium együttes alkalmazása egy vizuális adatbázis lekérdezés építő eszközzel.

Az alkalmazás adatbázisának felhő alapú adatbázisba migrálása lehetővé tenné azt, hogy az adatokat mások is elérhessék és esetleg nyilvánosan is elérhetőek legyenek.

9. Irodalomjegyzék

1. ECIS statisztika:
<https://ecis.jrc.ec.europa.eu/explorer.php>
2. Slicer website:
<https://www.slicer.org>
3. A. Fedorov, et al., 3D Slicer as an Image Computing Platform for the Quantitative Imaging Network, Magn Reson Imaging. 2012 November ; 30(9): 1323–1341. doi:10.1016/j.mri.2012.05.001.
4. Quantitative Imaging Network (QIN) about:
https://imaging.cancer.gov/programs_resources/specialized_initiatives/qin/about/default.htm
5. B.H.Menze, et al., The Multimodal Brain Tumor Image Segmentation Benchmark (BRATS), IEEE TRANSACTIONS ON MEDICAL IMAGING, VOL. 34, NO. 10, OCTOBER 2015
6. MICCAI Brain Tumor Segmentation (BraTS):
<http://www.braintumorsegmentation.org/>
7. Miroslav Kubat, An Introduction to Machine Learning, Second Edition, pp. 219-221
8. Dependency injection in C#:
<https://dotnettutorials.net/lesson/dependency-injection-design-pattern-csharp/>
9. A better way to implement data binding in .NET:
<https://docs.microsoft.com/en-us/archive/msdn-magazine/2016/july/data-binding-a-better-way-to-implement-data-binding-in-net>
10. Tal Geva, MD, Magnetic Resonance Imaging: Historical Perspective, Journal of Cardiovascular Magnetic Resonance (2006) 8, 573–580

UNIVERSITATEA SAPIENTIA DIN CLUJ-NAPOCA

FACULTATEA DE ȘTIINȚE TEHNICE ȘI UMANISTE, TÎRGU-MUREȘ

SPECIALIZAREA CALCULATOARE

Vizat decan

Conf. dr. ing. Domokos József

Vizat director departament

Ș.l. dr. ing Szabó László Zsolt