FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání diplomové práce

Název:	Webové rozhraní pro zpracování SPAMM tagovaných dat z magnetické rezonance
Student:	Bc. Tomáš Taro
Vedoucí:	Ing. Petr Pauš, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Webové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2022/2023

Pokyny pro vypracování

Cílem práce je vytvoření webového rozhraní pro existující aplikaci, která provádí zpracování SPAMM tagovaných dat získaných pomocí magnetické rezonance (MR). Webové rozhraní by mělo umožnit nahrání dat typu DICOM, jejich zpracování a zobrazení v prohlížeči, zadání parametrů pro algoritmus SPAMM včetně interaktivní úpravy mřížky, spuštění algoritmu a následné zobrazení výsledků.

1) Analyzujte možnosti zpracování medicinských dat ve formátu DICOM pomocí webových technologií.
2) Analyzujte možnosti propojení webového rozhraní s existující aplikací.
3) Analyzujte vhodné frameworky pro tvorbu webového rozhraní.
4) Navrhněte prototyp webové aplikace pro zpracování SPAMM tagovaných dat z MR.
5) Prototyp implementujte.
6) Proveďte vhodné testování na reálných SPAMM tagovaných datech z MR.

Elektronicky schválil/a Ing. Jaroslav Kuchař, Ph.D. dne 26. ledna 2022 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práca

Webové rozhranie pre spracovanie SPAMM tagovaných dát z magnetickej rezonancie

Bc. Tomáš Taro

Katedra softwarového inženýrství

Vedúci práce: Ing. Petr Pauš, Ph.D.

2. mája 2023

Poďakovanie

Touto cestou by som sa chcel poďakovať Ing. Petrovi Paušovi Ph.D. za vedenie a trpezlivosť pri vypracovávaní tejto diplomovej práce. Obrovské ĎAKUJEM patrí prevažne mojej rodine a priateľke za neustálu podporu, najmä počas posledného roku, kedy som dosiahol na dno svojich síl. Sľuby by sa mali plniť, a preto by som sa touto cestou chcel poďakovať Mekymu Žbirkovi, ktorého jedinečná hudba a svojrázny hlas bol tým potrebným spríjemnením počas písania tejto diplomovej práce.

Prehlásenie

Prehlasujem, že som predloženú prácu vypracoval(a) samostatne a že som uviedol(uviedla) všetky informačné zdroje v súlade s Metodickým pokynom o etickej príprave vysokoškolských záverečných prác.

Beriem na vedomie, že sa na moju prácu vzťahujú práva a povinnosti vyplývajúce zo zákona č. 121/2000 Sb., autorského zákona, v znení neskorších predpisov, a skutočnosť, že České vysoké učení technické v Praze má právo na uzavrenie licenčnej zmluvy o použití tejto práce ako školského diela podľa § 60 odst. 1 autorského zákona.

V Prahe 2. mája 2023 . . . . . .

České vysoké učení technické v Praze

Fakulta informačních technologií

Odkaz na túto prácu

Taro, Tomáš. *Webové rozhranie pre spracovanie SPAMM tagovaných dát z magnetickej rezonancie*. Diplomová práca. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2023.

Abstrakt

Klíčová slova

Abstract

Keywords

Obsah

Zoznam obrázkov

Kapitola 1

Úvod

Kapitola 2

Magnetická rezonancia

Magnetická rezonancia (MR) je jedna zo zobrazovacích techník, ktorá je používaná k zobrazeniu vnútorných orgánov tela. Narozdiel od röntgenového žiarenia a počítačovej tomografie (CT), magnetická rezonancia nepoužíva ionizujúce žiarenie. Avšak medzi spoločné znaky týchto troch zobrazovacích techník patrí ich neinvazívnosť a bezbolestné vyšetrenie [1] (vlastný preklad).

Magnetická rezonancia sa používa najmä pri:

- podozrení na anomálie mozgu a miechy, nádory a cysty,
- poranení kĺbov a mäkkých tkanív,
- podozrení na srdcové problémy a
- pri rozličných ochoreniach pečene a iných brušných orgánov [2] (vlastný preklad).

Pred niektorými MR procedúrami sa pacientovi môže intravenózne podať kontrastná látka, ktorá zlepší kontrast a vzájomnú odlíšiteľnosť orgánov a mäkkých tkanív [3].

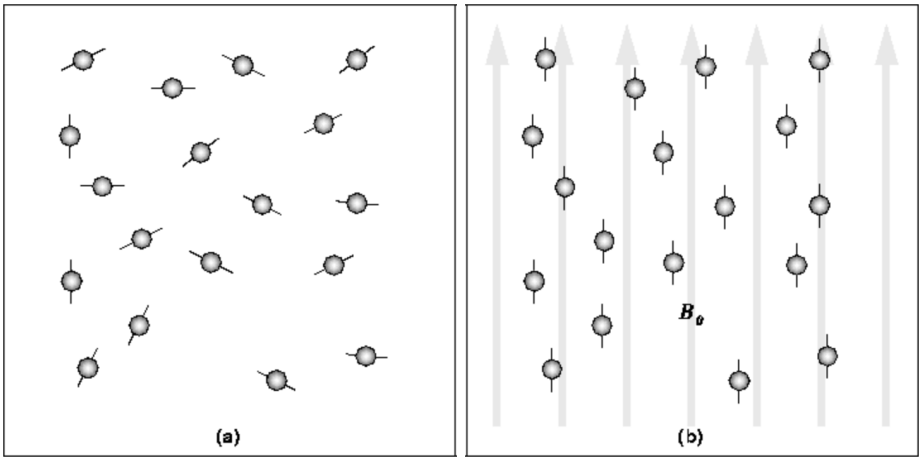
Bohužiaľ, existujú aj určité kontraindikácie, pri ktorých použitie MR nie je pre človeka vhodné. Jedným z kontraindikácií je implantovaný kardiostimulátor, v prípade že nie je kompatibilný s MR prístrojom.

Všeobecne sa za kontraindikáciu považuje použitie akéhokoľvek magnetického materiálu v tele. Taktiež je MR vyšetrenie kontraindikované ženám v prvom trimestri tehotenstva [4].

2.1 Princíp magnetickej rezonancie

Princípom magnetickej rezonancie je smerové magnetické pole (moment - $\mathcal{B}_0$) spojené s pohybom voľných jadier vodíku v tele pacienta. Tieto jadrá majú charakteristický pohyb (spin) vytvárajúci malý magnetický moment s určitým náhodným smerom a veľkosťou [1] (vlastný preklad).

Keď je pacient umiestnený vo veľkom magnetickom poli (v tubuse MR prístroja), voľné vodíkové jadrá sa zarovnajú v smere $\mathcal{B}_0$ (smer y) a vytvoria magnetický moment $\mathcal{M}$ paralelne k $\mathcal{B}_0$. Vodíkové jadrá začnú náhle prechádzať okolo smeru magnetického pola ako gyroskopy. Toto správanie sa nazýva Larmorova precesia [1] (vlastný preklad).



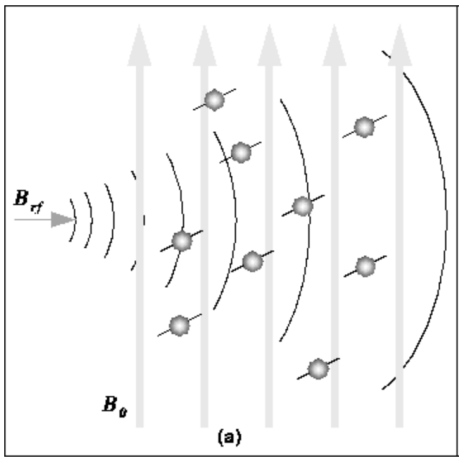
Obr. 2.1 Na ľavom obrázku je možné vidieť voľný pohyb vodíkových jadier, na pravom obrázku ich zarovnanie v smere $\mathcal{B}_0$ [1].

Následne sa aplikuje rádiofrekvenčný impulz $\mathcal{B}_{rf}$ kolmo na $\mathcal{B}_0$.
Tento impulz rovnajúci sa frekvencii Larmorovej precesie spôsobí posun $\mathcal{M}$ od $\mathcal{B}_0$ [1] (vlastný preklad).

Frekvencia Larmorovej precesie, tzv. Larmorova frekvencia, je definovaná ako:

$$\omega_0 = -\gamma * \mathcal{B}_0,$$

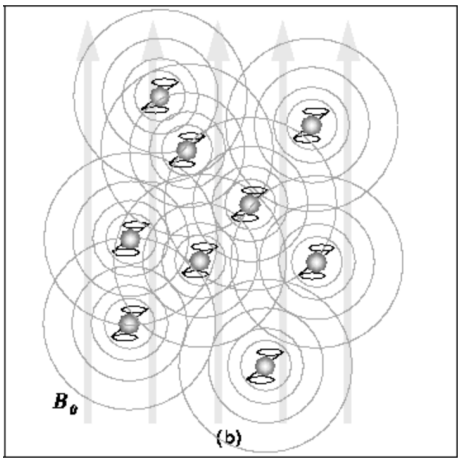
kde γ predstavuje gyromagnetický pomer a $\mathcal{B}_0$ intenzitu magnetického pola. Gyromagnetický pomer je konštanta závislá na jadre danej častice.
Pre vodík sa táto konštanta rovná 42.6 MHz/Tesla [1] (vlastný preklad).



Obr. 2.2 Kolmá aplikácia RF impulzu $\mathcal{B}_{rf}$ na vodíkové jadrá [1].

Akonáhle prestane pôsobiť rádiofrekvenčný impulz $\mathcal{B}_{rf}$, vodíkové jadrá sa presunú naspäť tak, že ich $\mathcal{M}$ je znovu paralelný s $\mathcal{B}_0$. Tento návrat vodíkových jadier sa nazýva relaxácia. Počas nej jadrá strácajú energiu vysielaním ich vlastného rádiofrekvenčného signálu [1] (vlastný preklad).

Tento signál sa nazýva „voľný indukčný rozpad“ – z anglického Free Induction Decay (FID). Ten sa zmeria vodivým poľom MR prístroja za účelom vyhotovenia 3D MR snímky v odtieňoch šedej [1] (vlastný preklad).



Obr. 2.3 Emitovanie FID signálu vodíkovými jadrami [1].

Avšak na jeho vytvorenie musí byť FID signál enkódovaný pre každý rozmer pomocou frekvenčného a fázového kódovania [1] (vlastný preklad).

Kódovanie v axiálnom smere sa dosiahne pridaním gradientového magnetického pola $\mathcal{G}_y$ v smere $\mathcal{B}_0$ (v smere y). Po pridaní $\mathcal{G}_y$ sa hodnota Larmorovej frekvencie zmení lineárne v axiálnom smere, tzn. že pre konkrétny axiálny rez existuje konkrétna Larmorova frekvencia, ktorá sa aplikuje vyslaním rádiofrekvenčného impulzu $\mathcal{B}_{rf}$ [1] (vlastný preklad).

Pole $\mathcal{G}_y$ sa následne odstráni a ďalšie gradientové magnetické pole – $\mathcal{G}_x$ – sa aplikuje kolmo na $\mathcal{G}_y$. Výsledkom je, že rezonančné frekvencie jadier sa menia v smere x vďaka $\mathcal{G}_x$ a majú fázovú variáciu v smere y v dôsledku predtým aplikovaného $\mathcal{G}_y$. Vzorky v smere x sú teda kódované frekvenciou a v smere y fázou [1] (vlastný preklad).

Následne sa použije 2D inverzná Fourierova transformácia pre transformáciu vzoriek na snímku [1] (vlastný preklad).

Kontrast získanej snímky závisí od nasledujúcich dvoch parametrov:

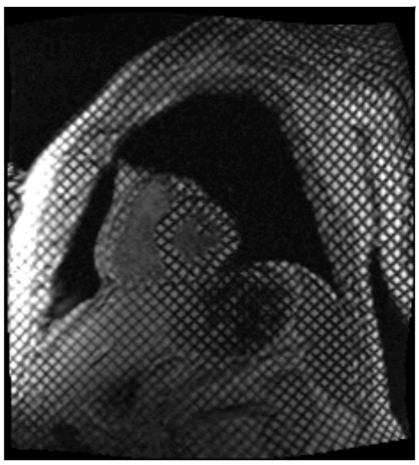
- času pozdĺžnej relaxácie – čas T1 a
- od času priečnej relaxácie – čas T2.

Čas T1 je čas potrebný pre jadrá vodíkov k ich relaxácii a čas T2 predstavuje čas, za ktorý sa FID signál prechádzajúci cez dané tkanivo rozpadne.
Oba časy závisia od daného typu tkaniva nachádzajúceho sa v pacientovi [1] (vlastný preklad).

Po získaní MR snímky sa impulz $\mathcal{B}_{rf}$ opakuje vopred stanovenou rýchlosťou. Zmenou sekvencie impulzov ($\mathcal{B}_{rf}$) sa vytvárajú rôzne typy snímiek.
Čas opakovania (TR) je množstvo času medzi po sebe nasledujúcimi pulznými sekvenciami aplikovanými na rovnaký rez. Time to Echo (TE) je čas medzi dodaním impulzu $\mathcal{B}_{rf}$ a prijatím odozvy. Úpravou TR je možné meniť výsledný kontrast na snímke medzi rôznymi typmi tkanív [1] (vlastný preklad).

2.2 SPAMM

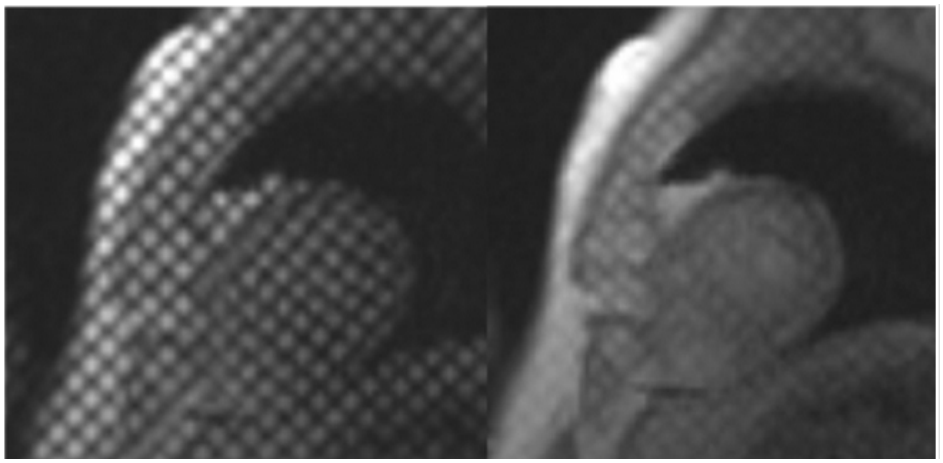
SPAMM – z anglického (SPAtial Modulation of Magnetization), čo v preklade znamená „priestorová modulácia magnetizácie“ – je technika používajúca rádiofrekvenčné saturačné impulzy pre umiestnenie mriežky na myokard, za cieľom sledovania jeho pohybu počas srdcového cyklu [5] (vlastný preklad).



Obr. 2.4 Otagovaná snímka myokardu pomocou techniky SPAMM [5].

V súčasnej praxi sa SPAMM technika používa v situáciách, kde informácia o kontrakcii myokardu je kľúčová, ako napr. podozrenie na ischemickú chorobu srdca alebo na abnormality týkajúce sa neprirodzeného pohybu stien myokardu [5] (vlastný preklad).

Nevýhodou použitia tejto techniky je skutočnosť, že táto mriežka sa stráca s blížiacim sa koncom srdcového cyklu. Samotné čiary mriežky sa pri konci systoly (časť srdcového cyklu, počas ktorej sa komory srdca sťahujú po naplnení krvou) môžu zlúčiť alebo úplne vyblednúť, čo sťažuje následnú analýzu pohybu srdca [5] (vlastný preklad).



Obr. 2.5 Ľavý obrázok zobrazuje začiatok systoly, pravý jej koniec.

Kapitola 3

Analýza súčasnej aplikácie

Táto kapitola sa zaoberá účelom súčasnej aplikácie a analýzou použitých technológií, ktoré boli použité pri vývoji súčasnej aplikácie. Následne sú popísané podprogramy, ktoré riešia výpočtovo náročnejšie úlohy v rámci aplikácie. Jednému z najdôležitejších podprogramov – `grid-tracker` – je venovaná väčšia pozornosť. Ďalej je venovaná pozornosť zostaveniu a spusteniu aplikácie, bez ktorej by nebolo možné prejsť k analýze používateľského rozhrania. Posledná sekcia tejto kapitoly prináša postrehy z testovania aplikácie a popis problémov s ňou spojených.

3.1 Účel aplikácie

Účelom súčasnej aplikácie – DICOM Viewer – je analýza pohybu srdcového svalu, pomocou ktorej by bolo možné zistiť anomálie v jeho pohybe. Aplikácia umožňuje importovať súbory typu DICOM (popísaného v sekcii 3.2.1) obsahujúce snímky z magnetickej rezonancie, ktoré sú otagované SPAMM mriežkou.

Na importovaných snímkach môže používateľ vytvoriť mriežku, ktorej parametre môžu byť manuálne upravené. Táto mriežka by mala kopírovať mriežku vytvorenú SPAMM technikou. Štruktúra a parametre týchto mriežok sú neskôr spracované podprogramom `grid-tracker`, ktorého úlohou je zarovnanie používateľom vytvorených mriežok s mriežkami vytvorenými SPAMM technikou.

Po ich zarovnaní by malo byť možné pomocou techniky grafových rezov odsegmentovať srdečné komory a tým pádom zúžiť analýzu pohybu srdcového svalu len na tieto komory.

3.2 Použité technológie

Táto sekcia sa venuje popisu použitých technológií v súčasnej desktopovej aplikácii. Na základe zistenia, aké technológie a aplikačné závislosti aplikácia využíva, bude možné aplikáciu zostaviť a vyskúšať.

Nasledujúci prehľad použitých technológií je založený na dôslednom preskúmaní zdrojového kódu aplikácie, ktorý sa vyznačoval skoro neexistujúcou dokumentáciou.

3.2.1 DICOM

V súčasnosti sú snímky získané pomocou zobrazovacích techník v medicíne zväčša ukladané v archivačnom a komunikačnom systéme snímkov. Tento systém ukladá nielen snímkové dáta ale aj iné relevantné dáta podľa štandardu známym pod skratkou DICOM[1] (Digital Imaging and Communications in Medicine) [6] (vlastný preklad).

DICOM je medzinárodným štandardom pre komunikáciu a manažment informácií o medicínskych obrazových a k nim príslušných dátach. Definuje, ako majú byť takéto dáta spracovávané, ukladané, tlačené a prenášané medzi zariadeniami podporujúcimi príjem týchto dát [7] (vlastný preklad).

Začiatok vývoja DICOM štandardu sa datuje k prelomu 80. a 90. rokoch 20. storočia, kedy započala spolupráca medzi American College of Radiology a National Electrical Manufacturers Association. NEMA taktiež vlastní autorské práva k tomuto štandadu. Momentálne sa DICOM skladá z 22 nezávislých častí, z ktorých avšak nie všetky musia byť implementované zariadením podporujúcim tento štandard [8] (vlastný preklad).

[1] https://www.dicomstandard.org

Pre účely spracovania snímkových dát, DICOM štandard vo svojej 10. časti definuje dátovú štruktúru (formát) súboru, do ktorého sa tieto dáta ukladajú. Súbor, ktorý spĺňa podmienky 10. časti tohto štandardu, býva značený ako „DICOM Part 10“ súbor, inak známy ako DICOM súbor [6] (vlastný preklad).

Štruktúra tohto (binárneho) súboru je nasledovná – prvých 128 bajtov býva zväčša prázdnych (vyplnených 0). Ďalšie 4 bajty obsahujú reťazec „DICM“. Na základe týchto bajtov sa dá určiť, či sa jedná alebo nejedná o DICOM súbor. Ďalej nasleduje hlavička, ktorá je rozdelená na viacero skupín zoskupujúcich súvisiace atribúty. Konkrétne atribúty sa adresujú tagom - ten sa skladá z 8 čísiel v hexadecimálnom formáte. Prvé 4 čísla reprezentujú skupinu, v ktorej sa daný atribút nachádza. Posledné 4 čísla jednoznačne identifikujú konkrétny atribút v skupine [6] (vlastný preklad).

Ako príklad je uvedené získanie informácie o pacientovom veku - všetky informácie o pacientovi sa nachádzajú v skupine 0010. Pacientov vek v tejto skupine sa nachádza na pozícii 1010, tým pádom výsledný tag, pod ktorým nájdeme vek pacienta je (0010, 1010). Ku každému tagu je jednoznačne priradená reprezentácia jej hodnoty, ktorý určuje dátový typ, formát a dĺžku hodnoty daného atribútu [6] (vlastný preklad).

Po hlavičke nasleduje skupina 7FE0, ktorá už obsahuje dáta o samotných obrazových pixeloch [6] (vlastný preklad). Typ kódovania týchto dát určuje Transfer Syntax – ten udáva, akým spôsobom sú obrazové pixely zakódované. Transfer Syntax obsahuje taktiež informáciu, v akom poradí bajtov sú informácie zakódované (Little Endian vs Big Endian) a aká kompresia obrazových dát bola použitá [9] (vlastný preklad).

3.2.2 Qt

Súčasná aplikácia bola vyvinutá pomocou Qt[2] – cross-platformového frameworku určeného pre vytváranie aplikácií najmä v jazyku C++.[3] Aplikácie vyvinuté týmto frameworkom majú výhodu v tom, že sú spustiteľné na rôznych operačných systémoch s minimálnym počtom zmien v zdrojovom kóde [10] (vlastný preklad).

[2] https://www.qt.io/product/qt6
[3] https://isocpp.org

V súčasnosti (od roku 2014) zastrešuje vývoj tohto frameworku spoločnosť The Qt Company.

Výsadou Qt frameworku je rozdelenie jeho funkcionality do jednotlivých modulov. Pri následom vývoji aplikácie sa použijú len tie moduly, ktoré sú v danej aplikácii potrebné [10] (vlastný preklad).

Existujúca aplikácia využíva tento framework vo verzii 5.15, ktorá sa vyznačuje tým, že je verziou s dlhodobou podporou. Koniec podpory tejto verzie je naplánovaný na koniec mája 2023. Najnovšou verziou Qt frameworku je momentálne verzia 6.5, ktorá je taktiež verziou s dlhodobou podporou.

V súčasnej aplikácii sa používajú nasledovné moduly:

- Qt Core,
- Qt Widgets,
- Qt GUI a
- Qt Test.

Qt Core modul obsahuje najpoužívanejšie triedy ako napr. `QCoreApplication`, `QObject`, `QDebug` a iné. Nakoľko sú tieto triedy používané aj inými modulmi, je tento modul implicitne nalinkovaný Qt frameworkom pri budovaní aplikácie [11] (vlastný preklad).

Qt Widgets modul poskytuje UI elementy, ktoré sú určené pre vytváranie používateľského rozhrania. Tieto elementy môžu zobrazovať rozličné dáta, prijímať vstup z klávesnice, byť štylizované a zoskupované do rozličných usporiadaní [12] (vlastný preklad). Existujúca aplikácia používa z modulu napr. triedu `QMainWindow`, ktorá je zodpovedná za vykreslenie aplikačného okna a taktiež triedu `QGraphicsScene`, ktorá je zodpovedná za vykreslenie snímkov z magnetickej rezonancie v DICOM formáte.

Qt GUI modul obsahuje triedy určené pre zobrazovanie aplikačného okna a iného grafického obsahu s následnou obsluhou udalostí. Taktiež obsahuje triedy, ktoré sú zodpovedné za zobrazovanie 2D grafiky, fontov a typografie [13] (vlastný preklad). Súčasná aplikácia z tohto modulu používa napr. triedu `QImage`, ktorá obsahuje metódy pre priamy prístup k pixelom snímkov a ich manipuláciu.

Qt Test modul poskytuje rozličné triedy pre jednotkové testovanie Qt aplikácií a príslušných knižníc [14] (vlastný preklad) – v súčasnej aplikácii bol tento modul využitý pri testovaní grafického používateľského rozhrania a funkcionalít súčasnej aplikácie, ako napr. testovanie zmien v nastaveniach vykreslenej mriežky na obrázku z magnetickej rezonancie.

3.2.3 Qmake

Pre zjednodušenie písania Makefilov, ktoré definujú, ako má byť program skompilovaný, bol použitý nástroj Qmake.[4] Tento nástroj pochádza taktiež z dielne The Qt Company.

Qmake umožňuje vývojárom definovať vytvorenie Makefilu pomocou syntaxe definovaného programom Qmake [15] (vlastný preklad). Výsledkom tohto procesu je `.pro` súbor obsahujúci inštrukcie, ako daný Makefile vytvoriť.
Následne sa pomocou príkazu `qmake` s argumentom cesty k `.pro` súboru vytvorí `Makefile` súbor, pomocou ktorého je možné daný program skompilovať, čoho výsledkom je spustiteľný súbor aplikácie.

Pre súčasnú aplikáciu sa daný súbor volá `Cameo.pro` a nachádza sa v adresári `dicomViewer`.

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

3.2.4 DCMTK

DCMTK je knižnica, ktorá implementuje veľkú časť DICOM štandardu v jazykoch C a C++. Úlohou tejto knižnice je okrem skúmania DICOM súborov aj ich vytváranie a konverzia, manipulácia s pamäťovými médiami a odosielanie, či prijímanie obrazových súborov cez internetové pripojenie [16] (vlastný preklad).

Za jej vývojom stojí nemecká firma OFFIS, ktorá túto knižnicu vyvíja od roku 1993. V súčasnosti sa DCMTK knižnica používa v nemocniciach a rôznych spoločnostiach po celom svete, kde predstavuje softvérový základ pri rozličných výskumných projektoch, prototypoch a komerčných produktoch nevynímajúc [16] (vlastný preklad).

Súčasná aplikácia je kompatibilná s najnovšou verziou tejto knižnice, ktorou je verzia 3.6.7. DCMTK knižnica je v tejto aplikácii použitá pre získanie informácií z hlavičiek DICOM súborov, ako napr:

- údaje o pacientovi,
- údaje o snímke a
- údaje o sérií snímkov.

3.2.5 OpenMP

OpenMP[5] poskytuje rozhranie pre programovanie aplikácií (tzv. API), vďaka ktorému je možné vytvárať C,[6] C++ a Fortran[7] aplikácie využívajúce viac vlákien nad zdieľanou pamäťou [17].

Vývoj OpenMP v súčasnosti zastrešuje OpenMP Architecture Review Board [17].

[5] https://www.openmp.org
[6] https://www.open-std.org/jtc1/sc22/wg14/
[7] https://fortran-lang.org/en/

OpenMP funguje na báze direktív, pomocou ktorých sa jednotlivé časti programu dajú paralelizovať viacerými spôsobmi – a to paralelizáciou prevádzania jednotlivých úloh (tzv. funkčný paralelizmus – vhodný pre paralelizáciu rekurzívnych algoritmov, kde úloha $\equiv volanie funkcie$) alebo paralelizáciou dátovo nezávislých iterácií (tu sa jedná o tzv. iteračný dátový paralelizmus) [17].

Existujúca aplikácia využíva direktívy OpenMP pre paralelizáciu výpočetne náročnejších algoritmov. Súčasná verzia OpenMP, verzia 5.2, je plne kompatibilná s aktuálnym zdrojovým kódom aplikácie.

3.2.6 TNL

Template Numerical Library[8] je numerická knižnica, ktorá poskytuje rozličné dátové štruktúry, ktoré uľahčujú prácu s pamäťou a vývoj efektívnych numerických riešičov. Táto knižnica je implementovaná pomocou C++ s cieľom poskytnúť flexibilné a užívateľsky prívetivé rozhranie. TNL poskytuje natívnu podporu pre moderné hardwarové architektúry ako sú viacjadrové CPU, GPU a distribuované systémy, ktoré je možné spravovať cez jednotné rozhranie [18].

Vývoj TNL knižnice od jej počiatku riadi Tomáš Oberhuber z Katedry matematiky na FJFI ČVUT v spolupráci s Jakubom Klinkovským a Alešom Wodeckim.

V súčasnej aplikácii sú z TNL knižnice použité kolekcie ako napr. `String` pre manipuláciu s reťazcami a `Containers::Array`, `Containers::{Vector, StaticVector,MultiVector}`, čo sú šablóny pre reprezentáciu n-dimenzionálnych polí, ktoré abstrahujú manažment dát a exekúciu bežných operácií na rozličných hardvérových architektúrach.

Aplikácia z TNL knižnice taktiež používa `Solvers::ODE::Merson` – jedná sa o riešič používajúci Runge-Kutta-Merson metódu štvrtého rádu s adaptívnym výberom časového kroku, vďaka ktorému je možné získať približné riešenie diferenciálnych rovníc.

[8] https://tnl-project.org

Bohužiaľ, súčasná aplikácia nie je kompatibilná s najnovšími zdrojovými kódmi TNL knižnice – pre nájdenie posledného „dobrého stavu“ knižnice, t.j. stavu za ktorého bolo možné aplikáciu skompilovať, bol využitý nástroj `git bisect`. Tento nástroj našiel ako posledný „dobrý stav“ knižnice z 13.5.2021. Pri využití TNL knižnice zostavenej po tomto dátume nebolo možné súčasnú aplikáciu skompilovať.

3.3 Pomocné podprogramy

Súčasná aplikácia obsahuje a využíva nasledujúce 3 podprogramy:

- grid-tracker,
- local-variance a
- graph-cuts.

Prvý z týchto podprogramov – `grid-tracker` – je C++ podprogram určený pre sledovanie pohybu myokardu pomocou detekcie SPAMM mriežky z DICOM snímky a mriežky vytvorenej používateľom. Mriežku vytvorenú používateľom sa snaží zarovnať so SPAMM mriežkou pre každú snímku zo sekvencie snímkov.

Vstupom tohto programu sú nasledujúce parametre:

- `inputImageFileNames` – vektor reťazcov reprezentujúci cesty k súborom multivectorov (definované TNL knižnicou), ktoré obsahujú enkódované DICOM snímky,
- `inputGridFileNames` – vektor reťazcov reprezentujúci cesty k súborom používateľom definovaných mriežok uložených ako TNL textový multivector,
- `outputGridFileNames` (nepovinné) – vektor reťazcov reprezentujúce cesty k súborom spracovaných mriežok, ktoré budú uložené ako textový TNL multivector,
- `curvatureCoefficient` – závislosť vývoja mriežky od zakrivenia, vyššie číslo znamená vyššiu závislosť,

- `forceCoefficient` – závislosť mriežky od gradientu obrazu, vyššie číslo znamená vyššiu závislosť a
- `stopTime` – časový interval algoritmického výpočtu.

Pre daný výpočet `grid-tracker` využíva technológie TNL a OpenMP.

Jeho výstupom by mali byť predchádzajúce používateľom definované mriežky v textovom TNL multivektore upravené algoritmom tak, aby mriežky zodpovedali mriežkam definovanými SPAMM metódou.

Nasledujúce dva C++ podprogramy nie sú dôležité pre túto diplomovú prácu, avšak pre úplnosť je ich účel vysvetlený.

Úlohou `local-variance` podprogramu je aplikovanie filtra lokálnej variancie na danú snímku za účelom lepšej segmentácie srdcových komôr. Tento filter je implementovaný pomocou jednoduchého rozptylového filtru, ktorý vypočíta priemernú intenzitu pixelov vo štvorcovom okolí každého pixelu a túto hodnotu dosadí do výberového rozptylu, ktorý sa bude rovnať novej hodnote intenzity pixelu [19].

Podprogram `graph-cuts` slúži pre segmentáciu srdcových komôr, ktorý vychádza z predpokladu, že hranica medzi objektom a jeho pozadím sa nachádza v miestach nekonzistencie susedných pixelov snímky. Implementovaná metóda, ktorá dosiahla dobré výsledky segmentácie, je metóda grafových rezov, kombinujúca Fordov-Fulkersonovým algoritmom s Preflow-Push algoritmom [19].

3.4 Zostavenie aplikácie a jej spustenie

Pre potrebu popísania používateľského rozhrania je najprv žiaduce dosiahnuť zostavenie existujúcej aplikácie a jej následné spustenie. Po počiatočnej analýze zdrojového kódu, v ktorom bolo zistené, že aplikácia bola vyvíjaná pre Linuxové prostredie, bol vo virtuálnom prostredí nainštalovaný operačný systém Ubuntu 22.10.[9]

[9] https://ubuntu.com

Pre zostavenie súčasnej aplikácie je potrebné nainštalovať Qt framework spolu s nástrojom Qmake, nakoľko ich architektúra súčasnej aplikácia vyžaduje.

Inštalácia Qt frameworku a Qmake nástroja je možná pomocou inštalácie balíčka `qt5-default`.[10] Ten okrem iného obsahuje aplikáciu Qt Creator.[11]

Qt Creator je aplikácia, ktorá poskytuje prostredie pre integrovaný vývoj aplikácií postavených nad Qt frameworkom. Pomocou tejto aplikácie je možné vyvíjať, testovať, zostavovať, spúšťať a debugovať aplikácie postavené na tomto frameworku.

Okrem balíčku `qt5-default` je taktiež potrebné nainštalovať balíčky `cmake`[12] a `build-essential`,[13] čo sú balíčky poskytujúce ďalšie nástroje potrebné pre zostavenie aplikácie. Tieto balíčky už môžu byť súčasťou použitej linuxovej distribúcie, čo ale neplatilo v prípade použitia Ubuntu 22.10 ako operačného systému.

Nakoľko súčasná aplikácia využíva DCMTK knižnicu, ako bolo popísané v sekcii 3.2.4, je taktiež nutné nainštalovať nasledujúce balíčky: `dcmtk`[14] a `libdcmtk-dev`.[15]

Prvý z uvedených balíčkov nainštaluje samotnú DCMTK knižnicu, druhý obsahuje knižnice a hlavičkové súbory pre vývoj aplikácií používajúce túto knižnicu. Nakoľko sa v `Dicom.pro` súbore (ktorý je určený pre výsledné zostavenie aplikácie) nachádzajú cesty ku knižniciam z balíčku `libdcmtk-dev`, je aj tento balíček nutný nainštalovať pre neskoršie korektné spustenie súčasnej aplikácie.

Keďže `grid-tracker` podprogram obsahuje OpenMP direktívy pre paralelizáciu výpočetných algoritmov, je potrebné nainštalovať OpenMP prostredníctvom balíčku `libomp-dev`.[16]

[10] sudo apt install qt5-default
[11] https://doc.qt.io/qtcreator/
[12] sudo apt install cmake
[13] sudo apt install build-essential
[14] sudo apt install dcmtk
[15] sudo apt install libdcmtk-dev
[16] sudo apt install libomp-dev

TNL knižnica bude taktiež benefitovať z inštalácie tohto balíčka, nakoľko sama využíva OpenMP pre paralelizáciu algoritmov, čo znamená, že je potrebné tento balíček nainštalovať ešte pred samotnou inštaláciou TNL knižnice.

Ďalej je potrebné nainštalovať TNL knižnicu – pre jej inštaláciu je potrebné stiahnuť zdrojové kódy buď formou zabaleného zdrojového kódu v .zip balíčku, alebo naklonovaním repozitára pomocou programu `git`.[17] Keďže aplikácia je nekompatibilná s najnovšou verziou TNL knižnice, je potrebné stiahnuť alebo naklonovať repozitár so zdrojovými kódmi do dátumu 13.5.2021 (viď 3.2.6).

Pretože je nutné samotnú knižnicu zo zdrojových kódov zostaviť, je nevyhnutné mať nainštalovaný kompilátor podporovaný samotnou knižnicou.

Medzi podporované kompilátory sa radí GCC[18] kompilátor vo verzii 8.0 a vyššie alebo Clang[19] vo verzii 7.0 a vyššie. Prvý z nich je možné nainštalovať pomocou balíčku `gcc`[20] a druhý pomocou balíčku `clang`.[21]

Následne je potrebné exekuovať inštalačný skript `install`, ktorého účelom je nakonfigurovanie TNL knižnice a jej zostavenia. Pred exekuovaním samotného inštalačného skriptu `install` je potrebné zistiť, či sú nainštalované balíčky `doxygen`,[22] `matplotlib`[23] a `graphviz`.[24]

Tieto balíčky sa využívajú pri generovaní dokumentácie počas exekúcie `install` skriptu (pri defaultnej inštalácii). Ak sa niektorý z daných balíčkov nenachádza v systéme, je potrebné ho doinštalovať.

Po nainštalovaní všetkých potrebných balíčkov sa TNL knižnica zostaví spustením `install`[25] skriptu. Po jeho ukončení sa hlavičkové súbory TNL knižnice nachádzajú v priečinku `/usr/lib/aarch64-linux-gnu`.

[17] https://git-scm.com
[18] http://gcc.gnu.org
[19] https://clang.llvm.org
[20] sudo apt install gcc
[21] sudo apt install clang
[22] https://doxygen.nl/index.html
[23] https://matplotlib.org
[24] https://graphviz.org
[25] chmod +x install && ./install

Názov posledného priečinku sa môže líšiť, nakoľko je závislý na architektúre CPU, na ktorom sa TNL knižnica zostavuje. V tomto prípade bola knižnica zostavovaná na systéme Ubuntu 22.10 bežiacom nad CPU s architektúrou Aarch64 (Apple M1 Pro CPU).

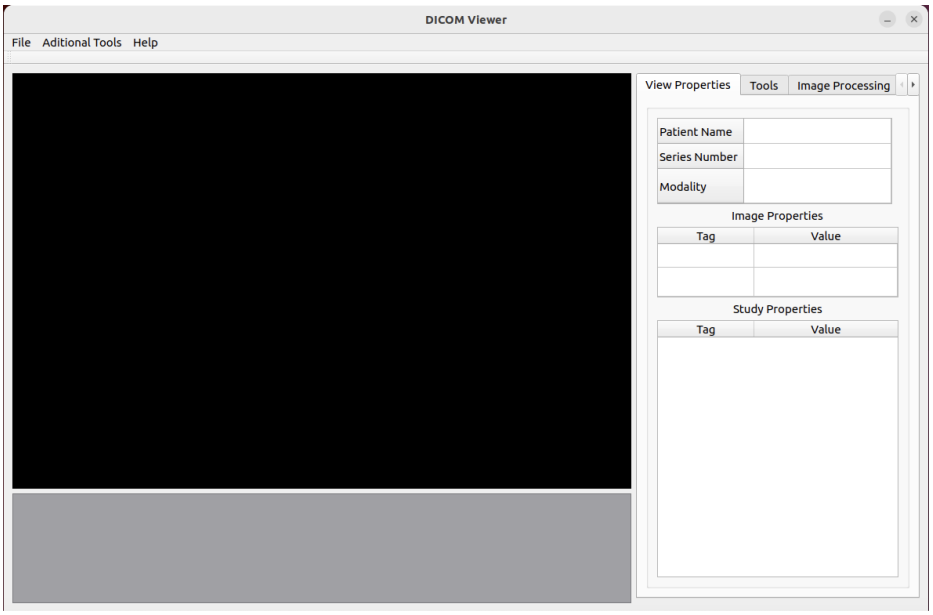
Ďalším krokom pre úspešné zostavenie aplikácie je definovanie cesty k hlavičkovým súborom TNL knižnice v samotnom `Cameo.pro` súbore. Po otvorení súboru `Cameo.pro` je potrebné nastaviť hodnotu premennej DCMTK_LIBS – tá obsahuje cesty k DCMTK knižniciam, ktoré súčasná aplikácia využíva. V tomto prípade ju bolo potrebné nastaviť na horeuvedený priečinok `/usr/lib/aarch64-linux-gnu` a následne daný súbor uložiť.

Po prevedení všetkých predchádzajúcich krokoch je možné spustiť nástroj `qmake` pomocou krokov popísaných v sekcii 3.2.3.
Jeho výsledkom je `Makefile` súbor, ktorý definuje kroky, ako zostaviť súčasnú aplikáciu. V tomto momente je už možné súčasnú aplikáciu skompilovať pomocou príkazu `make install`.

Po skompilovaní aplikácie sa v rovnakom priečinku objaví spustiteľný súbor `Cameo`, ktorému je potrebné nastaviť práva pre spustenie príkazom `chmod +x Cameo`. Následne je možné spustiť aplikáciu príkazom `./Cameo`.

3.5 Používateľské rozhranie

Po úspešnom spustení aplikácie sa používateľovi zobrazí jej hlavné okno, ktorého obsah je možné vidieť na nasledujúcom obrázku.



Obr. 3.1 Ukážka DICOM Viewer aplikácie po jej spustení

Používateľské rozhranie aplikácie je možné rozdeliť na nasledovné časti:

- aplikačné menu,
- ľavý postranný panel,
- centrálnu časť aplikácie a
- pravý postranný panel.

Na zobrazenom obrázku nie je možné vidieť ľavý postranný panel, nakoľko ten je po spustení aplikácie predvolene skrytý.

3.5.1 Aplikačné menu

V hornej časti okna aplikácie je zobrazené aplikačné menu. Toto menu pozostáva z troch hlavných možností, ktoré obsahujú viacero úrovní.

Jeho obsah je nasledovný:

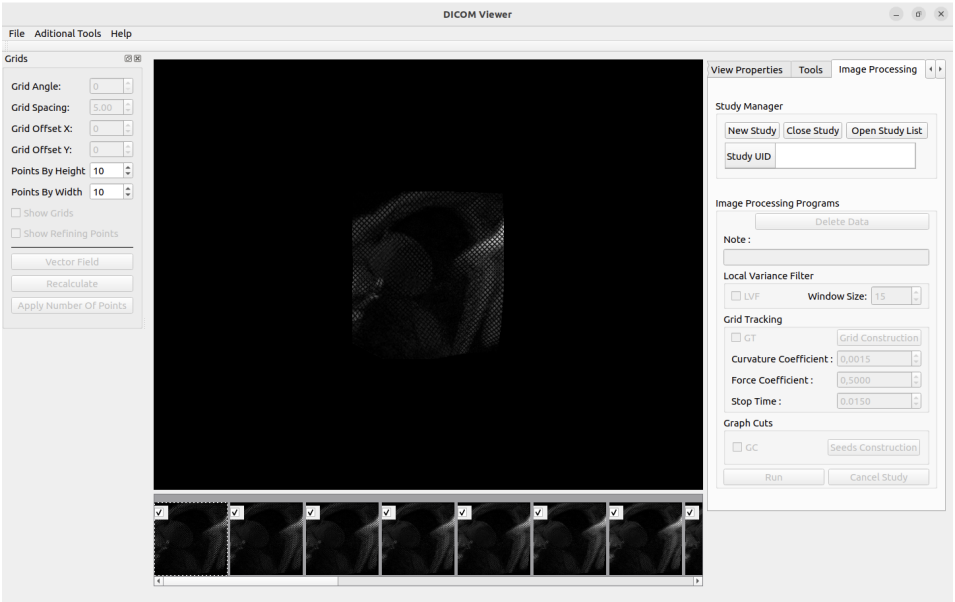
1. File
    a) Open – otvorí systémové okno pre výber priečinku s DICOM snímkami, ktoré sa majú importovať do aplikácie.
    b) Save Image
        i. Save View – uloží pohľad na aktuálnu snímku vo zvolenom formáte,
        ii. Save Scene – uloží scénu vo zvolenom formáte,
        iii. Save Area of Interes [sic] – uloží plochu záujmu vo zvolenom formáte.
    c) Save all selected
        i. Save View – uloží pohľad vybraných snímky vo zvolenom formáte,
        ii. Save Scene – uloží scénu vybraných snímkov vo zvolenom formáte,
        iii. Save Area of Interes [sic] – uloží plochu záujmu vybraných snímkov vo zvolenom formáte.
    d) Exit – ukončí aplikáciu.
2. Aditional [sic] Tools
    a) Grid Tools – otvorí ľavý postranný panel aplikácie s nastavením mriežky,
    b) Lvf Tools – otvorí ľavý postranný panel aplikácie s nastavením filtru lokálnej variácie,
    c) Graph Cuts Tools – otvorí ľavý postranný panel aplikácie s nastavením grafových rezov.
3. Help
    a) About – zobrazí informácie o DICOM Viewer aplikácii.

3.5.2 Import DICOM snímiek

Aplikácia sa po spustení nachádza v stave, v ktorom nie je možné s ňou interagovať. To je spôsobené tým, že do aplikácie je potrebné importovať aspoň jednu snímku v DICOM formáte.

Import snímiek je možné dosiahnuť zvolením možnosti File → *Open* z aplikačného menu. Následne sa otvorí systémové okno pre výber priečinka s DICOM snímkami, ktoré sa majú zobraziť v aplikácii. Bohužiaľ nie je možné zvoliť snímky jednotlivo z priečinku, čoho dôsledkom je načítanie všetkých snímiek z vybraného priečinku do aplikácie. Po jeho zvolení sa v aplikácii zobrazí prvá importovaná snímka.

Po výbere ľubovoľnej možnosti, ktoré ponúka Aditional [sic] Tools menu, sa taktiež zobrazí ľavý postranný panel v aplikácii, ako je možné vidieť na nasledujúcej snímke aplikácie. V tomto prípade bola zvolená možnosť „Grid Tools“.



Obr. 3.2 Zobrazenie prvej snímky v DICOM Viewer aplikácii

Zobrazením ľavého postranného panelu sa odhalí celková štruktúra používateľského rozhrania aplikácie.

3.5.3 Ľavý postranný panel

Obsah ľavého postranného panelu sa mení v závislosti na zvolenej možnosti z menu Aditional [sic] Tools. Momentálne je na snímke zobrazený obsah ľavého panelu po zvolení možnosti „Grid Tools“.

V ňom je možné nájsť nasledujúce možnosti:

- Grid Angle – uhol mriežky,
- Grid Spacing – rozpätie jednotlivých bodov,
- Grid Offset X – x pozícia od ľavého horného bodu,
- Grid Offset Y – invertovaná y pozícia od ľavého horného bodu,
- Points by Height – počet bodov na úsečku mriežky na výšku,
- Points By Width – počet bodov na úsečku mriežky na šírku,
- Show Grids – indikuje, či má byť zobrazená mriežka,
- Show Refining Points – indikuje, či majú byť zobrazené body, ktoré upresňujú pozíciu mriežky,
- Vector Field – spočíta rozdiel v pohybe mriežky medzi predchádzajúcou a aktuálnou snímkou,
- Recalculate – odošle dáta `grid-tracker` podprogramu pre opätovné zarovnanie mriežky voči SPAMM mriežke a
- Apply Number of Points – uloží mriežku ako textový TNL multivektor.

Ostatné dve možnosti z menu „Aditional [sic] Tools“ nie je potrebné pre účely tejto práce popisovať.

3.5.4 Centrálna plocha aplikácie

Obsahom centrálnej plochy, ktorá je dominantná v zobrazení aplikácie, je aktuálne vybraná DICOM snímka. Nad ňou môže byť tiež vykreslená používateľom definovaná mriežka.

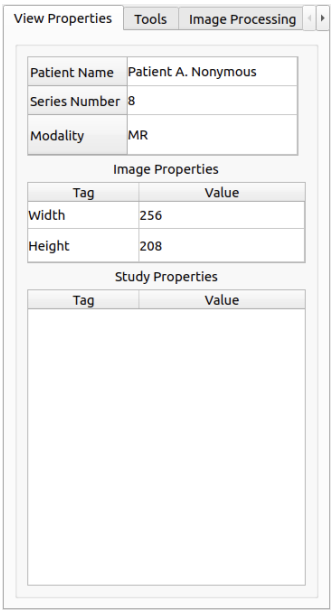
Pod touto plochou sú zobrazené náhľady všetkých snímiek, ktoré obsahujú zaškrtávacie políčko. Toto políčko reprezentuje možnosť, či má byť daná snímka spracovaná v rámci vybraného podprogramu.

3.5.5 Pravý postranný panel

Pravý postranný panel je rozdelený na nasledovné karty: „View Properties“, „Tools“, „Image Processing“ a „History“.

3.5.5.1 View Properties karta

Karta „View Properties“ nie je interaktívna – zobrazuje informácie ako meno pacienta nachádzajúceho sa na zobrazenej snímke, číslo série a modalitu snímiek. Taktiež je zobrazená výška a šírka aktuálne zobrazenej snímky.



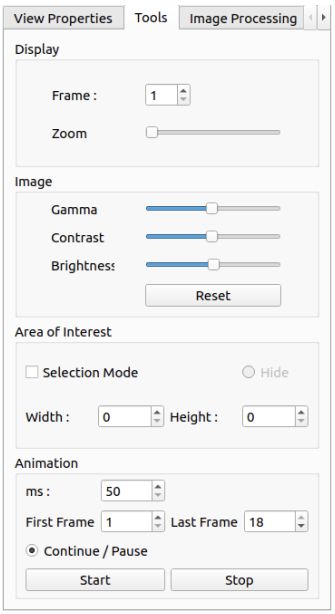
Obr. 3.3 Zobrazenie obsahu kariet pravého postranného panelu

3.5.5.2 Tools karta

Narozdiel od predchádzajúcej karty, „Tools“ karta obsahuje interaktívne prvky, ako napr. zobrazenie a zmena indexu zobrazenej snímky, či slider pre jej priblíženie. Snímke je taktiež možné zmeniť kontrast, jas a gammu pomocou sliderov.

Ďalej nasleduje sekcia pre nastavenie oblasti záujmu, pri ktorej je možné nastaviť jej zobrazenie alebo zmeniť jej výšku/šírku – oblasť záujmu je na základe týchto nastavení vykreslená nad snímkou.

Keďže aplikácia podporuje animáciu snímiek, je možné nastaviť jej rýchlosť v milisekundách (čo predstavuje čas, v rámci ktorého bude zobrazená jedna snímka), nastavenie počiatočnej snímky, od ktorej sa animácia spustí a poslednej snímky, po ktorú animácia bude spustená. Okrem nastavenia parametrov animácie nesmú chýbať tlačidlá pre spustenie a zastavenie animácie.

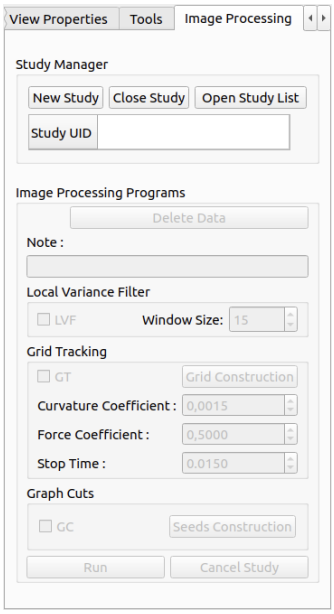


Obr. 3.4 Zobrazenie obsahu kariet pravého postranného panelu

3.5.5.3 Image Processing karta

Na „Image Processing“ karte sa nachádzajú tlačidlá ovládajúce manažéra štúdií. Jeho úlohou je zoskupovať rôzne štúdie, v rámci ktorých sa ukladajú parametre jej konfigurácie. Po kliknutí na tlačidlo „Open Study List“ sa zobrazí nové okno so všetkými štúdiami a ich parametrami. Pre interakciu s ostatnými poliami na tejto karte je potrebné najprv vytvoriť novú štúdiu kliknutím na tlačidlo „New Study“. Štúdiu je tiež možné ukončiť zvolením tlačidla „Close Study“. Každá štúdia je reprezentovaná jedinečným ID, ktoré sa skladá z dátumu a času jej vytvorenia.

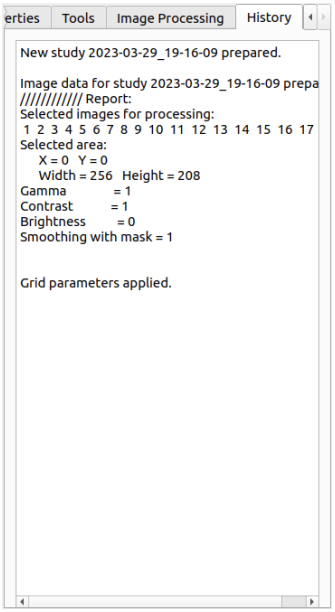
Vytvorením novej štúdie sa aktivujú polia „Image Processing Programs“ sekcie. Táto sekcia ponúka tri hlavné zaškrtávacie políčka – prvé reprezentuje aplikáciu filtra lokálnej variancie. Druhé z nich reprezentuje spustenie algoritmu pre zarovnanie vygenerovanej mriežky s mriežkou myokardu vytvorenou pomocou SPAMM techniky a tretie spustenie algoritmu segmentácie srdečných komôr pomocou grafových rezov. Zaškrtnutím daného políčka a kliknutím na tlačidlo „Run“ sa spustí algoritmus príslušný danému políčku. Pre „Grid Tracking“ algoritmus je v tejto sekcii možné definovať tri parametre, a to „Curvature Coefficient“, „Force Coefficient“ a „Stop Time“ (viď 3.3).



Obr. 3.5 Zobrazenie obsahu kariet pravého postranného panelu

3.5.5.4 History karta

Účelom poslednej karty „History“ je výpis rozličných záznamov pre informovanie používateľa o prebiehajúcich krokoch aplikácie.



Obr. 3.6 Zobrazenie obsahu kariet pravého postranného panelu

3.6 Testovanie a návrhy pre webovú aplikáciu

Účelom testovania súčasnej aplikácie je zoznámenie sa s aplikáciou a jej procesmi, ako aj overenie jej funkcionality a zistenia používateľského zážitku.

Počas samotného testovania bolo možné úspešne importovať DICOM súbory a zobraziť ich snímky. Snímky bolo taktiež možné animovať na základe nastavenia animácie, zmeniť im kontrast a jas, či ich priblížiť alebo oddialiť.

Následne bola odtestovaná funkcionalita týkajúca sa vykreslenia mriežky a jej úprav. Vykreslenie mriežky prebehlo bezchybne – avšak jej počiatočná poloha bola vždy fixne určená v ľavom hornom okraji snímky.

Toto správanie by sa dalo navrhnúť tak, aby bolo možné určiť počiatočné koordináty vytváranej mriežky jednoduchým kliknutím myši na miesto, kde by mala byť mriežka vykreslená.

Pri priblížení vykreslenej mriežky absentuje posun snímky samotným potiahnutím myši – posun snímky na ploche bolo možné len pomocou scrollbarov, ktoré boli ťažkopádnejšie na ovládanie.

Taktiež úprava polohy mriežky bola možná len pomocou stlačenia klávesy `Shift` a potiahnutím myši. Tento spôsob posunu avšak nie je používateľovi nikde odprezentovaný. Návrh používateľského rozhrania webovej aplikácie by mal preto uľahčiť prípadnú zmenu polohy snímky na ploche.

3.6.1 Problémy s `grid-tracker` podprogramom

Po úprave polohy mriežky nasledovalo spustenie samotného výpočtu zodpovedného za určenie súradníc bodov mriežok tak, aby zodpovedali vytvorenej mriežke pomocou SPAMM technológie. Pri spustení tohto algoritmu avšak aplikácia spadla. Nasledoval debugging aplikácie, ktorý potvrdil, že daný algoritmus nebol dokončený a prepojený so súčasnou aplikáciou.

Školiteľ bol následne s týmto problémom oboznámený. Po vzájomnej diskusii vzišlo k nasledujúcej dohode – možnosti prepojenia `grid-tracker` podprogramu s webovou aplikáciou budú zanalyzované a výsledné prepojenie navrhnuté, avšak následná implementácia prepojenia s týmto podprogramom sa neuskutoční, nakoľko jeho algoritmus bude potrebné upraviť tak aby nielen správne fungoval, ale navyše bol kompatibilný s najnovšou verziou TNL knižnice.

Webová aplikácia bude naďalej prijímať dáta potrebné pre samotný výpočet a odosielať výstup v štruktúrovanej podobe, avšak ten bude do určitej podoby reflektovať prijaté dáta, keďže dané prepojenie s `grid-tracker` podprogramom nebude implementované. Tento proces prijatia dát, výpočtu a ich odoslanie bude v nasledujúcich častiach spomínaný pod pojmom „SPAMM algoritmus“.

Kapitola 4

Analýza a návrh webovej aplikácie

Obsahom tejto kapitoly je počiatočná analýza mriežky, jej štruktúry a nastaviteľných parametrov. Následne sú zozbierané požiadavky kladené na webovú aplikáciu spolu s prípadmi použitia, ktoré opisujú, ako majú byť jednotlivé požiadavky implementované. Venovaná pozornosť je taktiež technológiam pre vývoj webových aplikácií, či analýzam architektúry a frameworkov pre tvorbu budúcej webovej aplikácie. Tiež nie je možné zabudnúť na analýzu spracovania MR snímiek a implementácie mriežky ako interaktívneho nástroja. Medzi posledné sekcie kapitoly patrí návrh používateľského rozhrania spolu s jej komunikáciou so serverom.

4.1 Analýza mriežky ako nástroja

Cieľom tejto sekcie je zhrnúť požiadavky kladené na štruktúru mriežky, ktorá sa bude vykreslovať nad zobrazenými DICOM snímkami. Obsahom tejto sekcie je aj zhrnutie nastaviteľných parametrov tejto mriežky používateľom.

4.1.1 Štruktúra mriežky

Mriežka pozostáva z horizontálnych a vertikálnych úsečok, ktoré sa navzájom pretínajú. Body, v ktorých sa úsečky pretínajú, nazveme „hlavnými“ bodmi. Tie viažu horizontálnu a vertikálnu úsečku prechádzajúcu týmto bodom.

To znamená, že posunutím tohto bodu by malo taktiež dojsť k úprave pozícií úsečiek tak, aby stále prechádzali cez presunutý bod.

Okrem spomenutých bodov by mala aplikácia vykresliť aj takzvané „refinement“ body na mriežke, ktoré by sa nachádzali medzi bodmi viažucími úsečky. Ich účel je presnejšie zarovnanie mriežky voči mriežke generovanej SPAMM technikou. V súčasnej aplikácii sa generujú tri „refinement“ body medzi každými dvoma „hlavnými“ bodmi.

Vykreslená mriežka by mala byť interaktívna, t.j. úprava jej celkovej polohy alebo polohy samotného bodu by malo byť možné pomocou pohybom myši.

4.1.2 Nastaviteľné parametre mriežky

Implementácia mriežky by mala obsahovať nastaviteľné parametre, pomocou ktorých by sa dala meniť jej vzhľad, resp. štruktúra.

Zoznam všetkých parametrov mriežky, ktoré by mali byť upraviteľné používateľom, je nasledovný:

- uhol mriežky,
- priestor medzi všetkými bodmi,
- X a Y offset mriežky počítaný z ľavého horného rohu snímky a
- počet horizontálnych a vertikálnych úsečiek.

Horeuvedené parametre by mali byť upraviteľné pomocou vstupu na klávesnici. Taktiež bude môcť byť upraviteľná poloha samotnej mriežky alebo jej akéhokoľvek bodu pomocou myši. Okrem toho by malo byť možné implementovať prepínač, ktorý zaistí zobrazenie „refinement“ bodov na vykreslenej mriežke a naopak.

Mriežka by mala na úpravy týchto parametrov reagovať jej prekreslením berúcim do úvahy zmenu daného parametru.

4.2 Analýza požiadaviek

Táto sekcia sa venuje analýze požiadaviek, ktoré sa delia na dve hlavné kategórie – funkčné a nefunkčné požiadavky. Ich realizácia je nutnosťou pre vytvorenie webovej aplikácie, ktorá bude obsahovať potrebnú funkcionalitu pre základ analýzy srdcového myokardu.

4.2.1 Funkčné požiadavky

Funkčné požiadavky sú požiadavky vymedzujúce rozsah funkcionality, ktorá by mala byť v danej aplikácii implementovaná.

4.2.1.1 FR1 – Spracovanie a zobrazenie MR snímiek

Do aplikácie by malo byť možné importovať snímky z magnetickej rezonancie vo formáte DICOM a tieto snímky taktiež zobraziť.

4.2.1.2 FR2 – Animácia MR snímiek

Aplikácia by mala umožniť animovať importované snímky pre jednoduchšiu analýzu pohybu myokardu. Parametre animácie ako jej rýchlosť a výber snímky, od/po ktorej/ktorú má animácia prebiehať, by mali byť upraviteľné, napr. pomocou číselného vstupu.

4.2.1.3 FR3 – Zobrazenie a interaktívna úprava mriežky

Implementovaná aplikácia by mala vedieť zobraziť mriežku nad snímkou z MR, ktorá by sa mala dať vygenerovať tlačidlom v používateľskom rozhraní. Taktiež je nutné implementovať nastaviteľné parametre mriežky, ktoré boli popísané v sekcii 4.1.2. Zmena týchto parametrov by mala byť ihneď viditeľná používateľom.

4.2.1.4 FR4 – Zadanie parametrov pre SPAMM algoritmus

Pre korektný výpočet súradníc bodov mriežok je nutné funkčnému SPAMM algoritmu podsunúť rozličné parametre. Ich hodnoty by sa mali dať určiť v aplikácií pre ich neskoršie použitie v tomto algoritme. Výpis týchto parametrov je možné nájsť v 3.3.

4.2.1.5 FR5 – Spustenie SPAMM algoritmu a zobrazenie jeho výsledkov

SPAMM algoritmus prijme potrebné dáta o všetkých importovaných snímkach a na nich definovaných mriežkach. Výstupom tohto algoritmu bude štruktúra dát, ktorá bude zodpovedať mriežkam s popisom súradníc bodov. Tento výstup bude následne nutné zobraziť v aplikácii.

4.2.2 Nefunkčné požiadavky

Požiadavky tohto typu síce nevymedzujú rozsah funkcionality danej aplikácie, avšak umožňujú určiť isté obmedzenia pre novú aplikáciu, ako napr. dôraz na podobu výslednej architektúry tejto aplikácie.

4.2.2.1 NF1 – Webová aplikácia

Prvou nefunkčnou požiadavkou je vytvorenie webovej aplikácie, ktorá by mala byť prístupná zo všetkých moderných webových prehliadačov. Pre lekárov výber tejto architektúry zjednoduší jej prístupnosť, nakoľko k takejto aplikácii bude možné pristupovať z rôznych zariadení a platforiem bez nutnosti inštalácie aplikácie.

4.2.2.2 NF2 – Používateľské rozhranie

Pre interakciu s aplikáciou je nutné navrhnúť a implementovať používateľské rozhranie, pomocou ktorého bude možné s aplikáciou interagovať. Lekári by preferovali používateľské rozhranie podobné iným aplikáciám z tejto oblasti.

4.2.2.3 NF3 – Ochrana pred únikom dát o pacientovi

Práca s osobnými dátami by mala byť do maximálnej možnej miere naprieč aplikáciou minimalizovaná, aby sa predišlo únikom citlivých údajov o pacientovi. Týka sa to najmä práce s DICOM súbormi, nakoľko tie obsahujú citlivé dáta o pacientovi.

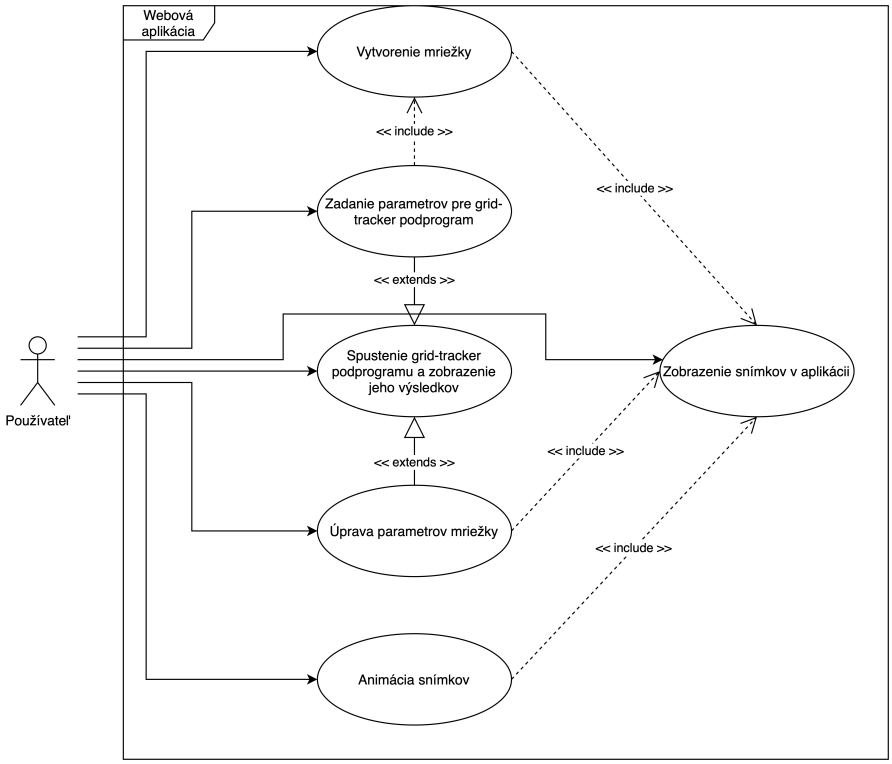
4.3 Používateľské role

V aplikácii sa bude nachádzať len jeden aktér – používateľ, rovnako ako v súčasnej aplikácii. Tomuto aktérovi by mala byť aplikácia sprístupnená bez rôznych funkčných obmedzení.

4.4 Prípady použitia

Nasledujúce prípady použitia reprezentujú rôzne činnosti, ktoré môže používateľ s aplikáciou vykonávať. Tieto prípady použitia sú popísané pomocou scenárov, ktoré vychádzajú z funkčných požiadaviek kladených na novú aplikáciu.

Pre lepšiu predstavu sú prípady použitia taktiež znázornené graficky pomocou nasledovného Use Case diagramu:



Obr. 4.1 Use case diagram

4.4.1 UC1 – Zobrazenie snímiek v aplikácii

Zobrazenie snímiek magnetickej rezonancie v DICOM formáte je jedným z esenciálnych funkčných požiadaviek – „FR1 – Spracovanie a zobrazenie MR snímiek“. Nasledovný scenár túto požiadavku realizuje.

Scenár:

1. Používateľ klikne na jedno z tlačidiel pre import DICOM snímiek do aplikácie.
2. Prehliadač zobrazí systémové okno, v ktorom si používateľ vyberie snímky, ktoré by chcel mať zobrazené v aplikácii.
3. Následne potvrdí import želaných snímiek.
4. Aplikácia automaticky vykreslí prvú importovanú snímku a taktiež zobrazí náhľady ostatných importovaných snímiek.
5. V prípade, že sa medzi zvolenými snímkami nachádza súbor, ktorý nekorešponduje so štruktúrou DICOM súboru, aplikácia zobrazí notifikáciu o neúspešnom zobrazení snímky.

4.4.2 UC2 – Animácia snímiek

Nasledovný scenár realizuje funkciu prehrania série snímiek ako animáciu, ako bolo popísané vo funkčnej požiadavke „FR2 – Animácia MR snímiek“. Okrem iného taktiež zahŕňa prípad „UC1 – Zobrazenie snímiek v aplikácii“.

Scenár:

1. UC1 – Zobrazenie snímiek v aplikácii.
2. Kliknutím na tlačidlo reprezentujúce štart animácie sa spustí animácia importovaných snímiek.
3. Kliknutím na tlačidlo reprezentujúce koniec animácie sa animácia skončí.

Alternatívny scenár:

2. Používateľ si nastaví rýchlosť animácie, index snímky, od ktorej má animácia začínať alebo index snímky, ktorou má animácia končiť.

3. Kliknutím na tlačidlo reprezentujúce štart animácie sa spustí animácia importovaných snímiek.

4. Kliknutím na tlačidlo reprezentujúce koniec animácie sa animácia skončí.

4.4.3 UC3 – Vytvorenie mriežky

Vytvorenie mriežky nad snímkou z MR je potrebné pre účely analýzu pohybu myokardu. Nasledujúci scenár čiastočne realizuje funkčnú požiadavku – „FR3 – Zobrazenie a interaktívna úprava mriežky“. Taktiež zahŕňa prípad použitia „UC1 – Zobrazenie snímiek v aplikácii“.

Scenár:

1. UC1 – Zobrazenie snímiek v aplikácii.

2. Používateľ klikne na tlačidlo „Create grid“.

3. Aplikácia zobrazí výzvu pre kliknutie na oblasť snímky, kde má byť mriežka vytvorená.

4. Používateľ klikne na oblasť snímky, kde chce vytvoriť mriežku.

5. Aplikácia vygeneruje mriežku s predvolenými nastaveniami a zobrazí ju na mieste predchádzajúceho kliknutia myši.

4.4.4 UC4 – Úprava parametrov mriežky

Medzi prípady použitia patrí aj úprava parametrov mriežky určenej pre analýzu pohybu srdcového svalu. Nakoľko je najprv potrebné mať mriežku pred jej úpravou vytvorenú, zahŕňa nasledovný scenár aj jej vytvorenie. Ten taktiež čiastočne realizuje funkčnú požiadavku „FR3 – Zobrazenie a interaktívna úprava mriežky“.

Scenár:

1. UC3 – Vytvorenie mriežky.
2. Používateľ upraví jeden alebo viacero parametrov uvedených v 4.1.2.
3. Aplikácia následne automaticky vykreslí mriežku na základe upravených parametrov.

4.4.5 UC5 – Zadanie parametrov pre SPAMM algoritmus

Pre spustenie algoritmu zodpovedného pre posun mriežky vytvorenej používateľom voči mriežke vygenerovanej SPAMM technikou je potrebné tomuto algoritmu poslať tri parametre definované v 3.5.5.3. Tieto parametre budú využité pri použití funkčného SPAMM algoritmu. Nasledujúci scenár tento prípad použitia realizuje spolu s funkčnou požiadavkou – „FR4 – Zadanie parametrov pre SPAMM algoritmus“.

Scenár:

1. UC1 – Zobrazenie snímiek v aplikácii.
2. Používateľ zadá číselné hodnoty parametrov „Curvature coefficient“, „Force coefficient“ a „Stop time“.

4.4.6 UC6 – Spustenie SPAMM algoritmu a zobrazenie jeho výsledkov

Spustenie algoritmu a zobrazenie jeho výsledku vyžaduje mať importované DICOM snímky spolu s používateľom vytvorenými mriežkami a ich modifikáciami.

To je dôvodom, prečo tento scenár použitia zahŕňa prípady „UC1 – Zobrazenie snímiek v aplikácii“, „UC3 – Vytvorenie mriežky“, „UC4 – Úprava parametrov mriežky“ a „UC5 – Zadanie parametrov pre SPAMM algoritmus“. Samotný scenár realizuje funkčnú požiadavku „FR5 – Spustenie SPAMM algoritmu a zobrazenie jeho výsledkov“.

Scenár:

1. UC1 – Zobrazenie snímiek v aplikácii.
2. UC3 – Vytvorenie mriežky.
3. UC4 – Úprava parametrov mriežky.
4. UC5 – Zadanie parametrov pre SPAMM algoritmus.
5. Používateľ kliknutím na tlačidlo „Compute“ spustí výpočet SPAMM algoritmu.
6. Po dokončení výpočtu aplikácia zobrazí mriežky upravené horeuvedeným algoritmom.

4.5 Technológie pre vývoj webovej aplikácie

V rámci tejto analýzy budú popísané technológie, ktoré budú použité pri vývoji webovej aplikácie. Konkrétne balíčky, príp. závislosti sa v tejto sekcii nachádzať nebudú – tie budú popísané v kapitole implementácie webovej aplikácie.

4.5.1 HTML5

Pre definovanie štruktúry webového dokumentu a jeho významu bude potrebné použiť značkovací jazyk HTML. Tento jazyk pozostáva zo série značiek (elementov) a k nim príslušných atribútov, pomocou ktorých je možné vytvorený obsah anotovať a významovo ho definovať. Týmto spôsobom je možné vytvoriť nadpisy, odstavce textu, číselné i nečíselné zoznamy, či importovať obrázky alebo sprostredkovať audio/video, atď.

Takto štruktúrovaný dokument definovaný pomocou jazyka HTML je možné zobraziť v ľubovoľnom webovom prehliadači. Webový prehliadač takýto dokument zanalyzuje a na základe použitých značiek vykreslí. Každá značka má definovaný predvolený štýl zobrazenia, ktorý sa môže líšiť od prehliadača k prehliadaču.

Nižšie je uvedený príklad základnej štruktúry HTML5 webového dokumentu:

```
<!doctype html5>
<html>
  <head></head>
  <body>
    <p>Hello world!</p>
  </body>
</html>
```

Značka `<!doctype>` definuje verziu použitého HTML dokumentu, čo je v tomto prípade HTML5. Ďalej nasleduje značka `<html>`, ktorej úloha je zoskupiť elementy `<head>` a `<body>`. V elemente `<head>` sa zvyčajne nachádzajú metadáta ako názov dokumentu, špecifikácia ďalších zdrojov pre načítanie v dokumente a iné. Na druhú stranu, element `<body>` zoskupuje obsah dokumentu, ktorý je zobrazený prehliadačom.

Prvá verzia tohto jazyka bola definovaná v roku 1993 samotným vynálezcom WWW, Timom Berners-Leeom. Momentálne najnovšou verziou HTML jazyka je tzv. HTML5 Living Standard,[26] vyvíjaný pracovnou skupinou WHATWG[27] [20] (vlastný preklad).

Najnovší štandard priniesol viacero nových značiek ako napr. `<audio>` pre prehrávanie audia, `<video>` pre prehrávanie videa, či `<picture>`, ktorá je určená pre definovanie viacero zdrojov pre obrázok. HTML5 štandard okrem značiek poskytuje niekoľko API, ktoré sú implementované webovými prehliadačmi. Pomocou nich je možné napr. geolokalizovať používateľa využitím HTML5 Geolocation API alebo vykreslovať grafiku použitím HTML5 Canvas API.

[26] https://html.spec.whatwg.org
[27] https://whatwg.org/

4.5.2 CSS 3

CSS[28] – z anglického Cascading Style Sheets – je jazyk popisujúci vzhľad použitých HTML5 elementov vo webovom dokumente. Tento jazyk definuje súbor pravidiel, ktoré môžu byť aplikované na jednotlivé elementy webového dokumentu, na základe ktorých sa mení vzhľad pravidlami ovplyvnených elementov.

Samotné pravidlo sa skladá zo selektora, ktorý definuje rozsah elementov, ktorých sa pravidlo týka. Nasleduje zoznam vlastností s ich hodnotami, ktoré majú byť aplikované na samotný selektor. Týmto spôsobom je možné definovať vzhľad nielen jedného, ale aj viacerých elementov vo webovom dokumente pomocou jedného pravidla [21] (vlastný preklad).

Nižšie je uvedený príklad pravidla, ktoré mení farbu textu vo všetkých elementoch `p` (`p` definuje odstavec textu) na červenú [21] (vlastný preklad):

```
p {
    color: red;
}
```

Zoskupené pravidlá sa väčšinou ukladajú do samostatného súboru s príponou `.css`. Tento súbor je následne nalinkovaný do HTML5 dokumentu pomocou značky `link`, ktorú prehliadač pri parsovaní dokumentu prečíta a následne aplikuje.

Definovanie samotného selektoru môže byť pre dané pravidlo sofistikovanejšie než ako bolo ukázané v príklade vyššie. Element môže byť špecifikovaný na základe jeho rôznych atribútov, ako ID, zoznam tried, či hodnotou jeho atribútu, atď.

Čo sa týka verzií CSS jazyka, nepoužívajú sa verzie ale tzv. levely. Prvým levelom bol CSS Level 1, ktorý sa stal odporúčanou špecifikáciou W3C konzorcia[29] v 1996. Tento level bol základom pre nasledujúce levely tohto jazyka [22] (vlastný preklad).

[28] https://developer.mozilla.org/en-US/docs/Web/CSS
[29] https://www.w3.org

V súčasnosti najnovší level CSS jazyka je CSS Level 3, v ktorom sa narozdiel od predchádzajúcich levelov jednotlivé časti jazyka delia na moduly, z ktorých každý môže mať level vyšší než level CSS jazyka. Z tohto dôvodu sa taktiež rozhodlo, že samotný level CSS jazyka sa už nebude zvyšovať [22] (vlastný preklad).

4.5.3 JavaScript

JavaScript je cross-platformový skriptovací programovací jazyk a treťou základnou technológiou pre vývoj webových stránok a aplikácií, po HTML a CSS. Používa sa pre implementovanie funkcionality, ktorú nie je možné dosiahnuť pomocou kombinácie HTML a CSS, ako napr. dynamická interakcia používateľa s webovou stránkou/aplikáciou, riešenie rôznych výpočetných úloh, odosielanie dát na server a prijímanie odpovede, a iné.

Samotný jazyk bol vytvorený Brendanom Eichom, pracujúcom vo firme Netscape, ktorá taktiež vyvíjala webový prehliadač – Netscape Navigator. JavaScript bol zahrnutý už vo verzii 2.0 tohto prehliadača, ktorý bol vydaný v roku 1995. Následne sa začal objavovať aj v iných prehliadačoch.

JavaScript nasleduje ECMAScript špecifikáciu,[30] vytvorenú organizáciou Ecma International. Tá kontinuálne vydáva každý rok novú štandardizovanú ECMAScript špecifikáciu, ktorá slúži ako predloha pre vytvorenie všeobecného skriptovacieho jazyka. JavaScript je v tomto prípade jazyk spĺňajúci tento štandard, nakoľko sa ním riadi a implementuje ho. Momentálne najnovšou verziou štandardu je jeho 14. edícia, ktorá pridala najmä nové metódy pracujúce s poľom (`Array`).

Čo sa týka vlastností samotného jazyka, JavaScript je dynamicky typovaným jazykom, čo znamená, že pri vytváraní premenných sa nedefinuje ich typ. To umožňuje do rovnakej premennej na jednom mieste uložiť číslo, a na inom zase reťazec. Taktiež sa jedná objektovo-orientovaný jazyk, kde dedičnosť je riešená mechanizmom prototypov – metódy a vlastnosti môžu byť za behu pridané do akéhokoľvek objektu [23] (vlastný preklad).

[30] https://tc39.es/ecma262/

JavaScript používa primárne jedno hlavné vlákno pre všetky svoje operácie, avšak je možné vytvoriť tzv. pracovné vlána pomocou Web Workers technológie. Nakoľko JavaScript podporuje OOP, imperatívny a deklaratívny štýl písania kódu, jedná sa o tzv. multi-paradigmový jazyk [23] (vlastný preklad).

Samotný kód je potrebné uložiť do súboru s koncovkou `.js`, aby bol rozpoznaný prehliadačom ako súbor obsahujúci JavaScript kód.

4.5.3.1 TypeScript

Pri písaní kódu v JavaScripte sa často môže stať, že programátor napíše nevalidný kód, avšak IDE ho žiadnym spôsobom na túto skutočnosť neupozorní. Táto situácia vzniká najmä kvôli tomu, že sa v JavaScript kóde nevyskytujú žiadne informácie o typoch premenných, argumentov funkcií a iné.

Pri menšom projekte je možné tieto problémy lepšie podchytiť, avšak pri vývoji robustnejšej aplikácie rôznymi ľuďmi je pravdepodobnejšie, že bude obsahovať chyby, na ktoré by mohli byť vývojári upozornení samotným IDE vopred ešte počas vývoja aplikácie.

Problém s neexistujúcimi typmi je možné vyriešiť pomocou písania kódu v jazyku TypeScript, vyvíjanom od jeho počiatku (2012) firmou Microsoft.[31] Ten pridáva podporu pre typy premenných, vstupných a výstupných argumentov funkcií či návratových hodnôt funkcií.

TS je nadmnožinou JavaScriptu, čo znamená, že validný JavaScript kód je taktiež validným TypeScript kódom. Taktiež je garantované, že JavaScript kód prevedený na TypeScript nezmení správanie kódu, čo znamená pre vývojárov jednoduchšiu migráciu z JavaScriptu do TypeScriptu [24] (vlastný preklad).

TypeScript v princípe funguje ako statický analyzátor kódu, ktorý analyzuje validitu napísaného kódu. V prípade, že kód nie je validný, alebo obsahuje chyby, TypeScript pomocou IDE upozorní vývojára na túto skutočnosť.
Pre samotnú analýzu kódu je nutné písať kód do súborov s koncovkou `.ts`.

[31] https://www.microsoft.com

Nižšie je uvedený príklad funkcie, ktorej argument má typ `number`. V prípade, že by bol argument nekompatibilného typu ako v uvedenom príklade, TS compiler pomocou IDE informuje vývojára o tejto skutočnosti. V prípade použitia JavaScriptu by IDE o tomto probléme vývojára neinformovalo.

```
1 function divideByThree(a: number): number {
2   return a / 3;
3 }
4
5 divideByThree('a');
6 // TS compiler v tomto prípade zobrazí na r. 5 nasledujúcu chybu:
7 // The left-hand side of an arithmetic operation
8 // must be of type 'any', 'number', 'bigint' or an enum type.
```

Nakoľko nie je možné TypeScript použiť priamo vo webovom prehliadači, je potrebné takýto kód konvertovať (transpilovať) do JavaScriptu. Podporu tejto funkcionality pridali vývojári TypeScriptu pomocou konzolového programu `tsc` [24] (vlastný preklad). Jeho vstupom sú TypeScript súbory, ktoré majú byť transformované do JavaScriptu. Výstupom sú už súbory v JavaScripte.

Počas transpilácie TypeScript kódu do JavaScriptu dochádza k vymazaniu všetkých informácií o typoch a iných konštruktoch nepodporovaných JavaScriptom, tak aby výsledný kód bol validným JavaScript kódom [24] (vlastný preklad).

Pre horeuvedené výhody tohto jazyka bude TypeScript použitý pri vývoji webovej aplikácie.

4.5.4 Node.js

Node.js je asynchrónne cross-platform runtime prostredie JavaScriptu, ktoré umožňuje vývojárom exekuovať JavaScript kód mimo prehliadača. Pomocou neho je možné vyvíjať nielen konzolové aplikácie, ale aj budovať škálovateľné webové služby [25] (vlastný preklad).

Node.js je open-source projektom a jeho vývoj zastrešuje nadácia OpenJS Foundation [25] (vlastný preklad).

Podobne ako JavaScript v prehliadači, Node.js používa jedno hlavné vlákno pre exekúciu kódu. Súčasne je nad Node.js vyvíjaných mnoho softvérových riešení (knižníc a frameworkov), ktoré sú dostupné vo forme balíčkov.

Tieto balíčky zvyknú byť dostupné v registri balíčkov pre Node.js – npm.[32] Pomocou rovnomenného konzolového programu je možné (globálne i lokálne) nainštalovať rôzne balíčky, ktoré môžu byť použité pri vývoji aplikácií.
Pre inštaláciu balíčka stačí exekuovať príkaz `npm install [-D] packagename`. Tieto balíčky je taktiež možné spravovať, aktualizovať na novšie verzie či odinštalovať. Počas inštalácie Node.js je automaticky nainštalovaný aj tento balíčkový manažér.

Okrem iného je vďaka Node.js možné používať len jeden programovací jazyk na vývoj oboch častí webových aplikácii – frontendu a backendu. Tento fakt predstavuje odpadnutie nutnosti ovládať ďalší programovací jazyk pre vývoj webových aplikácií.

Nakoľko je možné vyvíjať webovú aplikáciu v jednom jazyku, čo prináša komfort pre samotného vývojára, bude Node.js použitý na strane backendu pre komunikáciu s frontendom webovej aplikácie.

4.5.5 Docker

Docker je platforma určená pre vývoj a distribúciu aplikácií. Pomocou tejto platformy je možné separovať aplikáciu od infraštruktúry, čo umožňuje zrýchliť distribúciu softvéru [26] (vlastný preklad).

Docker poskytuje možnosť zabaliť aplikáciu a spustiť ju v izolovanom prostredí nazvanom „kontajner“. Tie sú vytvárané tak, aby obsahovali len nástroje potrebné pre beh aplikácie spolu s jej konfiguráciou, čo zrýchluje inicializáciu a spustenie kontajnerov. Kontajnery sú od seba predvolene izolované, avšak je možné dodatočne nastaviť sieťový interface pre komunikáciu medzi nimi [26] (vlastný preklad).

[32] https://npmjs.com

Vytvorenie kontajneru prebieha z objektu nazývanom „Image“. Image je inými slovami šablóna, ktorá definuje, ako má byť vytvorená zabalená aplikácia. Častokrát obsahuje príkazy, ktoré majú za úlohu nainštalovať aplikačné závislosti, nastaviť dodatočné bezpečnostné vlastnosti či otvoriť porty, cez ktoré je možné s danou aplikáciou komunikovať. Tieto príkazy sú následne uložené do súboru zvanom `Dockerfile`. Z jedného Docker Imagu je možné spustiť viacero kontajnerov, čím sa stáva škálovanie aplikácie ešte jednoduchším.

Novovytvorený image už väčšinou stavia na existujúcom image vytvoreným inou organizáciou, resp. vývojármi. Takéto image je možné nájsť v registri imageov – v Docker Hube. Pomocou tejto platformy je možné okrem sťahovania rôznych imageov taktiež zdieľať vlastný image s ostatnými.

Keďže použitie Dockeru prináša výhody ohľadom nasadenia aplikácie, bude táto technológia v tomto prípade využitá. Použitím Dockeru je taktiež možné vyhnúť sa prípadnou nemožnosťou zostavenia aplikácie, čoho príkladom môže byť problematické zostavenie súčasnej desktopovej aplikácie.

4.6 Analýza architektúry webovej aplikácie

Medzi prvými krokmi pred vývojom webovej aplikácie patrí analýza prípustných možností architektúry navrhovanej aplikácie. V tomto prípade analýza architektúry závisí najmä na prepojení webového rozhrania so súčasnou aplikáciou, a preto je potrebné najprv zanalyzovať dostupné možnosti tohto prepojenia.

Po porovnaní dostupných možností je nutné zvoliť jednu z nich. Následne bude možné pokračovať s analýzou technológií, ktoré budú použité pre vývoj aplikácie.

Čo sa týka prepojenia súčasnej aplikácie, resp. výpočetného podprogramu `grid-tracker` s novou webovou aplikáciou, existujú dve možnosti, ako by mohla daná integrácia prebehnúť.

Prvou možnosťou je využitie tzv. C++ addons[33] technológie, druhou možnosťou sa naskytuje využiť relatívne novú technológiu – WebAssembly.[34]

Nakoľko je potrebné pred samotným prepojením webovej aplikácie s `grid-tracker` podprogramom samotný podprogram upraviť, táto sekcia poslúži najmä autorom pokračujúcim vo vývoji webovej aplikácie.

4.6.1 C++ addons

C++ addons je technológia, ktorá poskytuje rozhranie medzi C/C++ knižnicami a JavaScriptom[35] [27] (vlastný preklad). Táto technológia je implementovaná v rámci Node.js.[36]

C++ addons umožňuje pristupovať k natívnym API operačného systému a taktiež pomáha integrovať C/C++ knižnice tretích strán pre ich priame použitie v Node.js. Doporučeným spôsobom písania takýchto addonov je pomocou technológie Node-API,[37] vďaka ktorej je možné vytvoriť Node-API addon v jazyku C [27] (vlastný preklad).

Pre písanie Node-API addonov v C++ musí byť použitý modul `node-addon-api`,[38] nakoľko ten obsahuje hlavičkové súbory v C++ [27] (vlastný preklad).

Výhodou použitia Node-API technológie je jej nemennosť v rámci rôznych verzií Node.js, čo zaručuje použitie skompilovaného addonu v rôznych verziách Node.js bez nutnosti jeho prekompilovania [27] (vlastný preklad).

Nástrojom pre zostavenie takéhoto modulu je build systém `node-gyp`.[39]
Ten používa `binding.gyp` súbor, ktorý špecifikuje konfiguráciu zostavenia modulu. Táto konfigurácia zahŕňa okrem iného aj cestu k zdrojovým `.cpp` a `.h` súborom [27] (vlastný preklad).

[33] https://nodejs.org/docs/latest-v18.x/api/addons.html
[34] https://webassembly.org
[35] https://developer.mozilla.org/en-US/docs/Web/JavaScript
[36] https://nodejs.org/en
[37] https://nodejs.org/docs/latest-v18.x/api/n-api.html
[38] https://github.com/nodejs/node-addon-api
[39] https://github.com/nodejs/node-gyp

Tie musia byť pred zostavením upravené tak, aby používali Node-API rozhranie. Tento krok zahŕňa vytvorenie metód, ktoré budú prijímať vstupné a vracať výstupné argumenty pretypované na Node-API typy.

Pred zostavením addonu je potrebné vygenerovať Makefile pre cieľový operačný systém pomocou príkazu `node-gyp configure`. Následne je možné addon zostaviť príkazom `node-gyp build`. Ten skompiluje súbory špecifikované v `binding.gyp` do jediného súboru s príponou `.node`. Ten je možné importovať ako modul do iného JavaScript modulu pomocou kľúčového slova `import`.[40] Po jeho importovaní je možné volať jeho metódy rovnakým spôsobom ako iné JS metódy [27] (vlastný preklad).

4.6.2 WebAssembly

WebAssembly je nový typ jazyku, ktorý je možné exekuovať vo všetkých moderných webových prehliadačoch. Jeho hlavnou výsadou je zvýšenie rýchlosti exekuovania kódu oproti JavaScriptu blížiaci sa k skoro natívnej exekúcii kódu naprogramovaného v jazykoch C, C++ alebo Rust[41] a iné. Je navrhnutý tak, aby mohol fungovať spoločne s JavaScriptom [28] (vlastný preklad).

Webová platforma sa dá vo všeobecnosti rozdeliť na dve časti:

- VM, pomocou ktorej sa exekuuje kód webovej aplikácie a
- webové API, ktoré je poskytnuté vývojárom pre kontrolu rozličných funkcionalít webového prehliadača, resp. zariadenia [28] (vlastný preklad).

Historicky VM umožňovala načítať len kód napísaný v JavaScripte. Avšak postupom času sa ukázalo, že JS nie je určený pre aplikácie, ktoré potrebujú väčší výpočetný výkon, ako sú napr. 3D hry, VR/AR, editácia videa či obrázkov a iné. WebAssembly bolo navrhnutý takým spôsobom, aby tieto problémy vyriešilo, čím by prinieslo prostriedky pre vývoj takýchto aplikácií [28] (vlastný preklad).

[40] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/import
[41] https://www.rust-lang.org

Pomocou WebAssembly JS API[42] je možné načítať WebAssembly moduly – čo sú moduly v binárnom formáte – do JavaScript aplikácie a zdieľať s touto aplikáciou funkcionalitu poskytovanú týmito modulmi [28] (vlastný preklad).

Možností, ako daný modul vytvoriť, je viacero:

- portovať C/C++ aplikáciu pomocou Emscripten[43] technológie,
- písať priamo vo WebAssembly,
- napísať aplikáciu v inom jazyku a kompilovať ju pomocou kompilátora podporujúci WebAssembly výstup, alebo
- použiť AssemblyScript,[44] ktorý je podobný TypeScript[45] jazyku a dá sa priamo skompilovať do WebAssembly [28] (vlastný preklad).

Nakoľko sa v tomto prípade jedná o C++ aplikáciu, bude bližšie analyzovaná prvá možnosť zo všetkých dostupných možností.

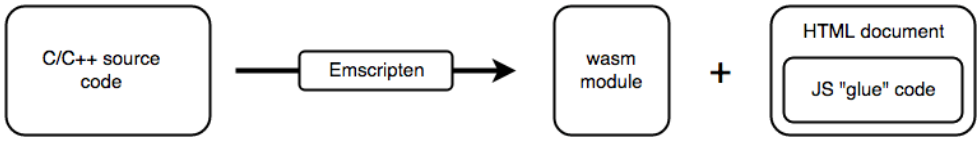
Najprv je potrebné stiahnuť a nainštalovať Emscripten kompilátor – ten umožní skompilovať program v C/C++ do modulu vo formáte `.wasm`, pomocou nástroja `em++`.

Nástroj `em++` je možné použiť nasledovným spôosobom:
`em++ sample.cpp -o sample.html`

Výstupom tohto príkazu sú tri súbory – `a.out.js`, `a.out.wasm` a `sample.html`. Prvý zo súborov tvorí JS kód, ktorého úloha je prepojiť vygenerovaný WASM modul (druhý súbor) s JS prostredím. Následne je možné tento modul spolu s JS kódom importovať do HTML[46] stránky, na ktorej sa daný modul exekuuje. Takto importovaný kód v HTML stránke je možné vidieť v súbore `sample.html` [29] (vlastný preklad).

[42] https://developer.mozilla.org/en-US/docs/WebAssembly/Using_the_JavaScript_API
[43] https://emscripten.org
[44] https://www.assemblyscript.org
[45] https://www.typescriptlang.org
[46] https://developer.mozilla.org/en-US/docs/Web/HTML

Celý proces je pre ilustráciu zobrazený pomocou diagramu nižšie:



Obr. 4.2 Od zdrojového kódu k .wasm modulu [30]

4.6.3 Výsledná voľba prepojenia

C++ addons technológia je implementovaná v Node.js, čo by v prípade zvolenia tejto technológie znamenalo zvolenie architektúry klient-server.

Klientom by bol v tomto prípade webový prehliadač, ktorý by posielal HTTP požiadavku serveru s potrebnými dátami pre `grid-tracker` podprogram. Po skončení výpočtu by server poslal odpoveď s informáciami o bodoch a úsečkách, ktoré by mali byť vykreslené na všetkých importovaných snímkach magnetickej rezonancie.

V prípade použitia technológie WebAssembly by všetky výpočetné operácie mohli byť implementované na úrovni klienta. Tým pádom by nebolo nutné posielať žiadne dáta serveru, čo hrá v prospech bezpečnosti.

Avšak, ako z analýzy `grid-tracker` podprogramu vyplýva, `grid-tracker` a TNL knižnica pre zrýchlenie výpočetných algoritmov používajú OpenMP technológiu. Bohužiaľ, WebAssembly túto technológiu nepodporuje, čo by v tomto prípade znamenalo, že by celý výpočet musel prebiehať jednovláknovo alebo byť refaktorovaný tak, aby používal viacero vlákien pomocou technológie WebAssembly threads [31] (vlastný preklad).

Čo sa týka C++ addons, tá OpenMP technológiu podporuje. Taktiež by bol v rámci tejto technológie nutný menší zásah do zdrojového kódu `grid-tracker` podprogramu, nakoľko by stačilo vytvoriť jednu wrapper funkciu v C++, ktorá by bola zodpovedná za konverziu dát do potrebného formátu a jeho výstup.

Taktiež nie je možné spoľahnúť sa na výkon zariadení, na ktorých by bežal `grid-tracker` pomocou WebAssembly. Nakoľko je tento algoritmus náročný na výpočet, bolo by potrebné zaistiť dostatočné výpočetné prostriedky pre každého lekára využívajúcim webovú aplikáciu.

Na základe týchto dôvodov bude lepšou a istejšou voľbou vybrať si technológiu C++ addons, s ktorou by mala byť implementácia prepojenia `grid-tracker` podprogramu a webovej aplikácie jednoduchšia a s podporou OpenMP technológie. Výpočty v rámci `grid-tracker` podprogramu by taktiež neboli závislé na dostupných výpočetných prostriedkov klienta ale serveru, čo umožňuje mať väčšiu kontrolu nad potrebným škálovaním výkonu pre `grid-tracker` podprogram.

4.7 Analýza frameworkov pre tvorbu webovej aplikácie

Webovú aplikáciu je možné od základov naprogramovať len pomocou vlastného kódu, avšak takýto vývoj by bol nie len zdĺhavejší, ale aj pracnejší, nakoľko by sa musel samotný vývojár zamerať na viac než len na implementáciu samotnej aplikácie. Riešením je použitie webového frameworku, ktorý vývojára od takejto práce odbremenia.

Pre vývoj webovej aplikácie by bolo vhodné využiť framework, ktorý je postavený nad Node.js technológiou, nakoľko by sa klientská a taktiež serverová časť aplikácie dala naprogramovať v jednom jazyku – JavaScripte. Plusom by v tomto prípade bolo, ak by daný framework natívne podporoval TypeScript, čo by mohlo zredukovať prípadné chyby v implementácii webovej aplikácie. Medzi ďalšími výhodami by patrilo použitie fullstack frameworku, vďaka ktorému by bolo potrebné orientovať sa len v jednom frameworku určenom pre obe časti webovej aplikácie – frontend aj backend.

V súčasnosti medzi najviac používané fullstackové frameworky, ktoré spĺňajú požiadavky uvedené vyššie, patria Nuxt.js[47] a Next.js.[48]

[47] https://nuxt.com
[48] https://nextjs.org

4.7.1 Nuxt.js

Nuxt.js je voľne dostupný open-source framework, pomocou ktorého je možné vytvárať fullstack webové aplikácie a stránky pomocou Vue.js.[49] Vue.js technológia bude popísaná v samostatnej podsekcii nižšie.

Nuxt.js ponúka automatický routing na základe štruktúry súborov v `/pages` zložke. Taktiež automaticky delí kód na menšie celky, čo môže pomôcť s prvým načítaním webovej aplikácie. Okrem renderovania obsahu až na klientovi je možné renderovať obsah už na serveri a takýto obsah poslať naspäť webovému prehliadaču. Pomocou automatických importov Vue.js komponentov nie je potrebné explicitne importovať použité Vue.js komponenty [32] (vlastný preklad).

Samotný framework je naprogramovaný v TypeScripte, čo znamená že je možné využívať type-hinty čo sa týka funkcionality Nuxt.js pri programovaní webovej aplikácie i bez nutnosti použitia TypeScriptu [32] (vlastný preklad).

Na pozadí používa Nuxt.js Nitro[50] server, ktorý generuje API endpointy na základe štruktúry súborov nachádzajúcich sa v zložke `server/api` [32] (vlastný preklad).

Zostavenie aplikácie je možné pomocou príkazu `nuxt build` – jeho výstupom je `.output` zložka obsahujúca minifikované súbory. Túto zložku je následne možné nasadiť na server podporujúci Node.js a zostavenú aplikáciu spustiť pomocou príkazu `node .output/server/index.mjs`.

4.7.1.1 Vue.js

Vue.js je JavaScript framework určený pre budovanie používateľského rozhrania postavený na štandardných technológiach ako HTML, CSS a JavaScript.

[49] https://vuejs.org
[50] https://nitro.unjs.io/

Tento framework poskytuje deklaratívny programovací model založený na znovupoužiteľných komponentoch, ktoré je možné použiť v rámci iných komponentov, čím pomáha zefektívniť proces vývoja znížením nutnosti použitia duplicitného kódu [33] (vlastný preklad).

Pod pojmom „komponent“ je možné predstaviť si samostatnú jednotku používateľského rozhrania (v HTML) s definovanými štýlmi (pomocou CSS) a stavom, ktorý je riadený pomocou JavaScriptu. Takto definovaný komponent je možné nazvať tzv. Single File Componentom (SFC).

Nasleduje príklad využitia Vue.js frameworku – pomocou vytvorenia SFC, ktorý v rámci jedného súboru kombinuje použitie HTML, CSS a JS ako v popise uvádzanom vyššie.

```
// ParagraphComponent.vue
<template>
  <p>{{ paragraphText }}</p>
</template>

<script setup lang='ts'>
import { computed, defineProps } from 'vue';
const props = defineProps({
  text: {
    type: String,
    default: '',
  },
});
const paragraphText = computed(() => {
  return props.text;
});
</script>

<style lang='scss' scoped>
p {
  color: red;
}
</style>
```

Uvedený príklad demonštruje dve hlavné funkcie Vue.js frameworku:

- deklaratívne vykreslovanie a
- reaktivitu [33] (vlastný preklad).

V prvom prípade sa jedná o rozšírenie štandardného HTML o template syntax – {{ }}, ktorý umožňuje dynamicky vykresliť obsah na základe JavaScript stavu.

V uvedenom príklade sa jedná o zobrazenie paragrafu, kde zobrazený text pochádza z premennej `paragraphText`. Táto premenná obsahuje `computed` funkciu, ktorá vracia hodnotu `props.text`.

Tá pochádza zo šablóny iného komponentu, kde bol tento komponent importovaný. Nasledujúci príklad zobrazuje šablónu tohto komponentu.

```
// ArticleComponent.vue
<template>
  <paragraph-component text="Example text">
</template>

<script setup lang='ts'>
import { ParagraphComponent } from './ParagraphComponent.vue';
</script>
```

V súbore `ArticleComponent.vue` bol importovaný komponent `ParagraphComponent.vue`, ktorý definuje atribút `text` a nastavuje ho na hodnotu „Example text“. Hodnota tejto premennej sa tým pádom spropaguje do `ParagraphComponent.vue` komponentu do kľúča `text` objektu `props` (`props.text`).

`props` je objekt, v ktorom je možné nájsť všetky takto definované `properties`. Ak by sa namiesto fixného textu v atribúte `text` nachádzala premenná, ktorá by zmenila hodnotu, funkcia `computed` zaistí, že sa jej vrátená hodnota (`paragraphText`) zmení na základe detekovanej zmeny jej hodnoty. Na základe tohto príkladu bola ukázaná sila reaktivity Vue.js frameworku.

Horeuvedeným spôsobom je možné modulárne vytvárať a zobrazovať rozličné komponenty podľa potreby. Taktiež je možné reagovať na rozličné eventy emitované prehliadačom, ako napr. na kliknutie myši na určitý element, posun po webovej stránke, atď.

Nakoľko webový prehliadač neumožňuje priamo importovať Vue.js komponenty, je nutné ich zostavením skonvertovať do JavaScriptu, napr. pomocou nástroja Vite[51] [33] (vlastný preklad).

Ten je nutné nainštalovať ako závislosť, napr. pomocou nástroja `npm`.
Potom je potrebné vytvoriť konfiguračný súbor, v ktorom sa definuje zostavovanie Vue.js komponentov a následne pomocou príkazu `vite build` je možné zostaviť Vue.js komponenty do `.js` súborov, ktoré môžu byť následne importované do HTML šablóny [33] (vlastný preklad).

4.7.2 Next.js

Next.js je voľne dostupným, taktiež open-source fullstack frameworkom podobným Nuxt.js. Avšak narozdiel od Nuxt.js, Next.js nepoužíva pre vytváranie znovupoužiteľných UI komponentov knižnicu Vue.js ale React.js.[52]

Next.js ponúka nástroje pre vytváranie API endpointov, ktoré musia byť vytvorené v zložke `pages/api`. Čo sa týka samotnej funkcionality, taktiež podporuje kompilovanie UI komponentov do spustiteľného JS kódu, jeho minifikovanie pre rýchlejší prenos dát medzi serverom a webovým prehliadačom, code-splitting (rozdelenie kódu pre zvýšenie výkonu aplikácie) až po server-side rendering (SSR, vykreslený obsah na serveri sa pošle klientovi). Samotný framework je naprogramovaný pomocou TypeScriptu, takže je možné využívať nápovedu pri programovaní webovej aplikácie pomocou tohto frameworku [34] (vlastný preklad).

Zostavenie aplikácie je možné príkazom `next build`. Tento príkaz vytvorí `.next` zložku obsahujúcu skompilovaný obsah aplikácie. Takto skompilovanú aplikáciu je možné spustiť príkazom `next start` [34] (vlastný preklad).

[51] https://vitejs.dev
[52] https://react.dev/

4.7.2.1 React.js

React.js je JavaScript framework, ktorý poskytuje možnosti pre budovanie používateľského rozhrania. Svojim účelom je podobný Vue.js frameworku a tiež patrí medzi open-source nástroj, ktorý je ale spravovaný spoločnosťou Meta[53] [35] (vlastný preklad).

React.js sa od Vue.js líši spôsobom definovania komponentu – nepoužíva SFC pre definovanie štruktúry komponentu, jeho štýlu a stavu ale tzv. JSX, ktorý umožňuje písať HTML v JavaScripte [35] (vlastný preklad).

JSX je oproti HTML striktnejší v tom, že:

- samotný komponent musí byť uzatvorený v značke,
- všetky značky musia mať uzatváraciu značku a
- atribúty značiek je potrebné písať v camelCase forme [36] (vlastný preklad).

Použitie CSS je v komponente možné pomocou importovania CSS stylesheetu napísaného pre daný komponent [37] (vlastný preklad), alebo importovania tzv. CSS modulu, ktorý je možné prepoužiť viacerými komponentmi [38] (vlastný preklad).

Taktiež je možné CSS definovať priamo v JS a ten naviazať priamo na element v komponente. CSS naviazané týmto spôsobom je možné dynamicky meniť v závislosti od vnútorného stavu komponentu.

Taémuto komponentu zostáva už len definovať jeho stav. Ten je možné vytvoriť pomocou funkcie `useState`, ktorého argumentom je inicializačná hodnota stavu. Táto funkcia vracia pole, kde na nultom indexe sa nachádza aktuálna hodnota daného stavu a na prvom indexe sa nachádza funkcia, ktorú je možné exekuovať pre aktualizáciu stavu [39] (vlastný preklad).

[53] https://about.meta.com/

Nasledujúci príklad zobrazuje použitie tejto funkcie:

```
// ParagraphComponent.js
import { useState } from 'react';

const [answer, setAnswer] = useState('');
console.log(answer); // výstupom na konzole bude prázdny reťazec
setAnswer('foo');
console.log(answer); // výstupom na konzole bude reťazec 'foo'
```

Pre porovnanie je na nasledujúcom príklade ukázaný rovnaký komponent ako v príklade pre Vue.js, avšak upravený pre React.js framework:

```
import 'paragraph.css';

export default function Paragraph({ paragraphText = '' }) {
  return (
    <>
      <p> {paragraphText} </p>
    </>
  );
}
```

V porovnaní s kódom pre Vue.js je horeuvedený kód kratší, nakoľko neobsahuje „boilerplate“ pre definovanie prijímaných properties ako vo Vue.js. Taktiež je automaticky zaistené prepojenie hodnoty `paragraphText` s vyrenderovaním jeho obsahu v `<p>` tagu. Pre fungovanie importovania horeuvedeného komponentu je potrebné funkciu, ktorá obsahuje definíciu React.js komponentu, exportovať. Táto povinnosť vo Vue.js odpadá.

Samozrejme je rozsah rozdielov väčší než tie uvedené v tejto práci. Účelom ukážky je informovať o základných rozdieloch vytvárania komponentov v oboch frameworkoch.

4.7.3 Výsledná voľba frameworku

Na základe doterajších skúseností s Vue.js frameworkom by mal byť vývoj aplikácie pomocou Nuxt.js frameworku pre autora plynulejší a menej problematický, keďže autor nemá skúsenosti s React.js a Nuxt.js frameworkom.

4.8 Analýza spracovania MR snímiek

Spracovávanie importovaných MR snímiek by malo čo v najväčšej miere prebiehať najmä na strane klienta – vo webovom prehliadači. Takto zvolený prístup zamedzí prípadnému útočníkovi preniknúť k snímkam a dátam o pacientoch, ktoré by v opačnom prípade museli byť uchovávané na strane servera.

Z uvedeného vyplýva, že pre implementáciu aplikácie bude potrebné nájsť JavaScript knižnicu resp. knižnice, ktoré sú schopné spracovať DICOM súbory v prehliadači.

Pod pojmom „spracovať“ je myslené: čítanie hlavičky DICOM súborov, zobrazenie snímiek nachádzajúcich sa v týchto súboroch, či možnosť tieto snímky modifikovať. Medzi ďalšie požiadavky kladené na takúto knižnicu patrí jej aktívny vývoj, dostupná dokumentácia a taktiež použiteľnosť knižnice pre produkčné nasadenie.

Bohužiaľ, všetky požiadavky kladené na hľadanú knižnicu nespĺňa ani jedna nájdená knižnica, ale výber viacero knižníc, kde každá z nich implementuje určitú časť požiadaviek a dokopy podmienky kladené na knižnicu vyššie, spĺňajú.

Jedná sa o nasledovné knižnice:

- Cornerstone Core,
- Cornerstone DICOM Image Loader a
- Dicom Parser.

4.8.1 Cornerstone Core

Cornerstone Core[54] je knižnica, ktorá má za úlohu zjednodušiť proces vývoja komplexnejších webových aplikácií, ktoré majú za úlohu zobrazovať snímky akéhokoľvek formátu, vrátane bežných medicínskych snímkových formátov. Taktiež poskytuje API, pomocou ktorého je možné zobrazovať DICOM snímky a samotné zobrazenie konfigurovať, ako napr. zvýšením alebo znížením intenzity jasi, priblížením a oddialením snímky, a iné [40] (vlastný preklad).

[54] https://github.com/cornerstonejs/cornerstone

Táto knižnica neimplementuje import DICOM súborov a ich spracovanie. Túto funkcionalitu deleguje na tzv. ImageLoaders. Tie po spracovaní DICOM súborov posunú DICOM dáta cez spoločné rozhranie Cornerstone Core knižnici, ktorá ich nakoniec vykreslí [40] (vlastný preklad).

Cieľom tohto prístupu Cornerstone Core knižnice je jej dôraz na minimalizmus a poskytnutie flexibility pri spracovávaní rôznych typov obrazových dát. Použitie Cornerstone Core knižnice pre vývoj špecializovaných aplikácií tohto typu je de-facto štandardom [40] (vlastný preklad).

V súčasnosti sa pripravuje nová „Cornerstone Core“ knižnica, ktorej názov sa zmení na „Cornerstone3D“.[55] Keďže je táto knižnica momentálne v beta verzii a stabilná verzia tejto knižnice ešte nebola vydaná, vývoj aplikácie bude postavený na doterajšej Cornerstone Core knižnici. Momentálne neexistuje alternatíva tejto knižnice, ktorá by sa špecifikovala na túto oblasť.

4.8.2 Cornerstone DICOM Image Loader

Cornerstone DICOM Image Loader[56] je tzv. ImageLoader, ktorý je zodpovedný za načítanie a spracovanie DICOM súborov. Použitie tejto knižnice je vynútené Cornerstone Core knižnicou. Táto knižnica podporuje nielen načítanie DICOM súborov cez HTTP protokol, ale aj z lokálneho súborového systému pomocou File API[57] implementovaného webovými prehliadačmi [41] (vlastný preklad).

Po načítaní DICOM súborov je ich parsovanie prenechané knižnici Dicom Parser. Nakoľko sa veľkosť týchto súborov môže pohybovať v rádoch megabajtov (MB), samotné parsovanie súborov prebieha pomocou využitia technológie Web Workers.[58] Pre začiatok priblížim technológiu Web Workers a následne knižnicu Dicom Parser.[59]

[55] https://github.com/cornerstonejs/cornerstone3D-beta
[56] https://github.com/cornerstonejs/cornerstone3D-beta/tree/main/packages/dicomImageLoader
[57] https://developer.mozilla.org/en-US/docs/Web/API/File_API
[58] https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API
[59] https://github.com/cornerstonejs/dicomParser

4.8.2.1 Web Workers

JavaScript je v prehliadači implementovaný ako jednovláknový jazyk využívajúci jedno hlavné vlákno a exekúcia skriptov tohto jazyka prebieha zvyčajne v tomto vlákne. Výpočetne náročné úlohy by avšak mohli vyústiť do zablokovania tohto vlákna, ktoré sa prejavuje nereagovaním prehliadača na rozličné používateľské akcie alebo nevykreslovaním aktualizácií na webovej stránke. Dôvodom zablokovania hlavného vlákna by v tomto prípade bolo využitie všetkých dostupných prostriedkov prioritne pre danú výpočetne náročnú úlohu.

Web Workers technológia je štandardom, ktorý je implementovaný a poskytovaný webovými prehliadačmi umožňujúci exekúciu takýchto úloh, ktoré by inak pri dlhšom spracovávaní mohli dané hlavné vlákno zablokovať.

Pomocou Web Workers je možné predísť zablokovaniu hlavného vlákna jednoduchým vytvorením nového pracovného vlákna pomocou konštruktu `new Worker(url)`, kde `url` je adresa skriptu, ktorý má bežať v novom pracovnom vlákne. Takéto pracovné vlákno môže exekuovať JS skript bez zablokovania hlavného vlákna, keďže je od neho nezávislé a taktiež komunikovať s hlavným vláknom [42] (vlastný preklad).

4.8.3 Dicom Parser

Dicom Parser je knižnica implementujúca parsovanie všetkých známych validných DICOM súborov. Knižnica je navrhnutá pre beh vo všetkých moderných HTML5 prehliadačoch a nie je závislá na žiadnej knižnici [43] (vlastný preklad).

Dicom Parser poskytuje globálny objekt `dicomParser`, ktorý obsahuje viacero metód, z ktorých je najzaujímavejšia metóda `parseDicom`. Argumentom tejto metódy je `Uint8Array` pole obsahujúce nespracovaný (raw) obsah DICOM súboru. Výsledkom volania tejto metódy spolu s `Uint8Array` poľom je `DataSet` objekt obsahujúci vyparsovaný obsah DICOM súboru.

Alternatívou Dicom Parser knižnice by mohla byť knižnica dcm.js,[60] avšak vývoj tejto knižnice nie je stále dokončený (nebola zatiaľ vydaná jej stabilná verzia) a sami vývojári varujú pred použitím tejto knižnice v produkčnom prostredí.[61]

4.8.4 Zhrnutie

Pre zhrnutie informácií v tejto sekcii – knižnica Cornerstone DICOM Image Loader využíva Web Workers pre vytvorenie nových pracovných vlákien, ktorých úloha je parsovanie DICOM súborov pomocou metódy `parseDicom` objektu `dicomParser` pochádzajúceho z Dicom Parser knižnice uvedenej vyššie. Metódou vrátený `DataSet` objekt je následne poslaný knižnici Cornerstone Core, ktorá sa postará o vykreslenie vyparsovanej DICOM snímky z tohto objektu.

4.9 Analýza možností implementácie mriežky

Pre implementáciu mriežky by bolo vhodné použiť knižnicu, pomocou ktorej by bolo možné danú mriežku vykresliť a upravovať podľa potrieb používateľa.

Rodina Cornerstone frameworkov ponúka pre tento účel knižnicu Cornerstone Tools,[62] ktorá asistuje nielen pri vytváraní rôznych anotácií pre DICOM snímky (zobrazených pomocou Cornerstone Core), ale aj pri ich segmentácii či rôznych meraní. Taktiež ponúka široký počet nástrojov, ktoré môžu dané snímky modifikovať alebo nad týmito snímkami vykreslovať rôzne informácie či lomené čiary.

4.9.1 Závislosti knižnice

Pre využitie tejto knižnice je potrebná nielen Cornerstone Core knižnica, nakoľko je s ňou úzko previazaná, ale aj knižnica Hammer.js[63] a Cornerstone Math.[64]

[60] https://github.com/dcmjs-org/dcmjs
[61] https://github.com/dcmjs-org/dcmjs
[62] https://github.com/cornerstonejs/cornerstoneTools
[63] https://github.com/hammerjs/hammer.js
[64] https://github.com/cornerstonejs/cornerstoneMath

Cornerstone Tools využíva Cornerstone Core knižnicu pre reagovanie na rôzne eventy, ktoré Conerstone Core knižnica emituje. Na základe týchto eventov môžu nástroje Cornerstone Tools knižnice meniť svoj stav.

Hammer.js knižnica implementuje podporu rozhrania založeného na dotyku namiesto myši. Túto knižnicu je potrebné importovať bez ohľadu na to, či sa plánujú využívať gestá na báze dotyku alebo nie, nakoľko niektoré nástroje sú od tejto knižnice závislé.

Cornerstone Math, ako už názov napovedá, poskytuje rôzne matematické operácie týkajúce sa prevažne vektorovej matematiky. Niektoré nástroje z Cornerstone Tools knižnice ju používajú napr. pre výpočet vzdialenosti medzi rôznymi bodmi.

4.9.2 Spôsoby implementácie mriežky

Nakoľko Cornerstone Tools neobsahuje mriežku ako nástroj, ktorý vie knižnica zobraziť a s ňou ďalej pracovať, bude nutné ju od základov implementovať.

Táto implementácia môže využívať API[65] poskytované knižnicou, pomocou ktorej by bolo možné danú mriežku implementovať bez zásahu do knižnice alebo bude nutné danú mriežku naprogramovať priamo do knižnice.

4.9.2.1 Implementácia pomocou Cornerstone Tools API

Pri implementácii mriežky pomocou dedikovaného API by nebolo potrebné udržiavať vlastnú kópiu Cornerstone Tools knižnice.

V tomto prípade by pre využitie rôznych funkcií implementovaných v samotnej knižnici bolo možné vyžadovanú funkcionalitu importovať pomocou metódy `importInternal(moduleName)`.

Nevýhodou tohto spôsobu je nedostatočná flexibilita spojená s nemožnosťou importovania všetkej funkcionality, ktorá by mohla byť pri vývoji potrebná. Ďalším negatívom zvolenia tohto spôsobu by bola nemožnosť upravenia akéhokoľvek kódu v Cornerstone Tools knižnici.

[65] https://tools.cornerstonejs.org/api/

4.9.2.2 Implementácia v rámci Cornerstone Tools knižnice

Na druhú stranu, ak by mala byť mriežka implementovaná priamo v Cornerstone Tools knižnici, odpadol by problém s importovaním teoreticky potrebnej funkcionality, nakoľko by sa dal importovať akýkoľvek modul knižnice priamo pomocou JS konštruktu `import`, bez nutnosti využitia `importInternal` metódy.

4.9.2.3 Výsledný výber spôsobu implementácie mriežky

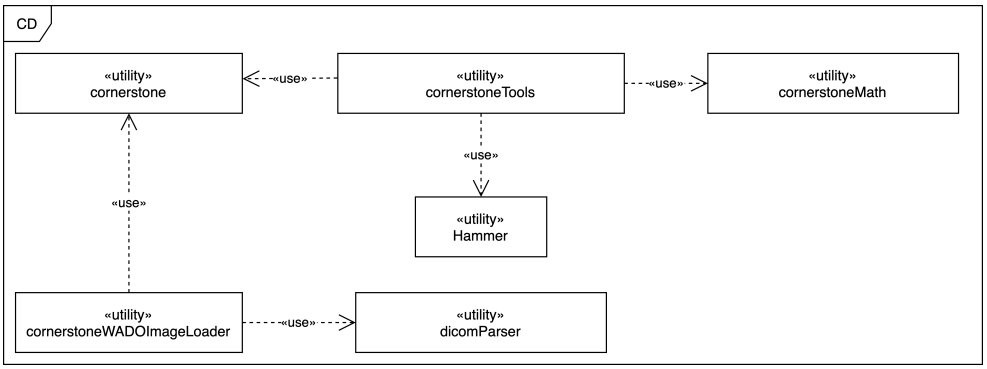
Nakoľko v tomto momente nie je jasné, či bude výsledná implementácia mriežky potrebovať zmenu niektorého zo súborov Cornerstone Tools knižnice, je vhodnejšie začať implementovať mriežku priamo v knižnici. Keď bude implementácia tejto mriežky dokončená, bude nutné posúdiť, či je možné celú funkcionalitu mriežky migrovať do riešenia využívajúceho iba dedikované API pre svoju funkcionalitu.

Ako pri Cornerstone Core, tak aj Cornerstone Tools knižnica bude mať čoskoro svojho nástupcu,[66] ktorá bude určená pre Cornerstone3D. Táto nová knižnica je momentálne v aktívnom vývoji a jej stabilná verzia rovnako ako v prípade Cornerstone3D nebola doteraz vydaná. To je dôvodom, prečo bude pri implementácii aplikácie použitá doterajšia verzia Cornerstone Tools knižnice.

[66] https://github.com/cornerstonejs/cornerstone3D-beta/tree/main/packages/tools

4.10 Zobrazenie závislostí medzi knižnicami

Pre lepšie znázornenie previazania jednotlivých knižníc uvedených v tejto sekcii bol vytvorený diagram tried, na ktorom sú zobrazené instancie tried a závislosti medzi nimi.



Obr. 4.3 Class diagram znázorňujúci závislosti medzi knižnicami

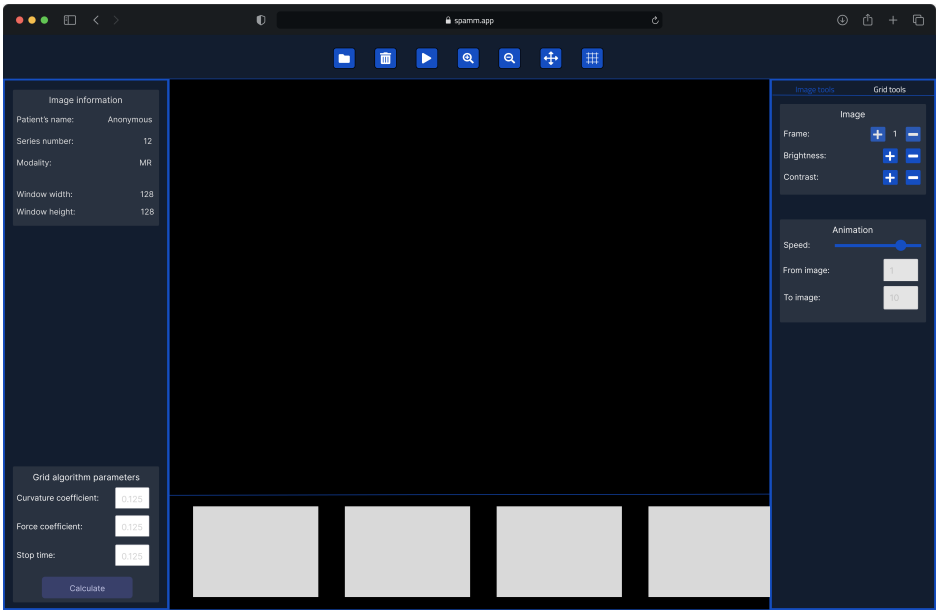
4.11 Návrh používateľského rozhrania

Pri návrhu používateľského rozhrania som sa inšpiroval UI doterajšej aplikácie, avšak s pár zmenami týkajúcimi sa štruktúry budúceho používateľského rozhrania.

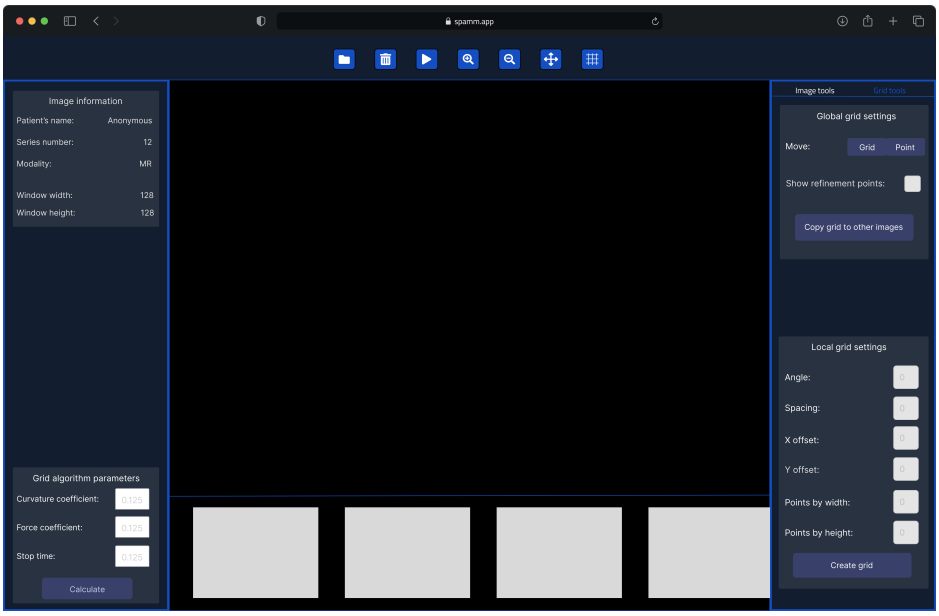
Tento návrh používateľského rozhrania som rozdelil na 4 hlavné časti:

- horný panel,
- ľavý postranný panel,
- centrálnu časť a
- pravý postranný panel.

Konečný návrh, ktorý bude slúžiť ako predloha pri vytváraní používateľského rozhrania webovej aplikácie, je priložený nižšie. Ten sa skladá z dvoch snímiek, nakoľko sa obsah pravého postranného panelu môže meniť na základe zvolenej karty.



Obr. 4.4 Návrh používateľského rozhrania - prvá časť



Obr. 4.5 Návrh používateľského rozhrania - druhá časť

Nasledujúce podsekcie sa budú venovať popisu jednotlivých častí s popisom jednotlivých možností zobrazených v daných častiach.

4.11.1 Horný panel

V hornom paneli sa nachádzajú tlačidlá, na ktoré bude možné kliknúť.

Prvé tlačidlo zľava bude slúžiť pre načítanie DICOM snímiek. Po kliknutí naň sa zobrazí systémové okno, kde si používateľ bude môcť vybrať DICOM snímky, ktoré sa majú importovať do aplikácie. Pri každej tejto akcii sa vymažú doteraz importované snímky a nahradia sa práve importovanými snímkami.

Akonáhle používateľ aplikácie klikne na druhé tlačidlo, vymažú sa všetky načítané snímky a aplikácia sa prepne do stavu pred načítaním snímiek do aplikácie.

Tretie tlačidlo bude určené pre spustenie animácie snímiek. Po kliknutí naň sa ikona zmení na „stop“ ikonu a tlačidlo zmení svoju funkciu – po opätovnom kliknutí sa animácia zastaví.

Pomocou štvrtého tlačidla bude možné priblížiť aktuálne zobrazenú DICOM snímku – piate tlačidlo ju oddiali.

Šieste tlačidlo umožní posunúť DICOM snímku v ľubovoľnom smere a siedmym tlačidlom bude možné presúvať vykreslenú mriežku alebo jej body v závislosti na aktuálnom nastavení tohto tlačidla.

4.11.2 Centrálna časť

V centrálnej časti aplikácie bude zobrazená aktuálne vybraná DICOM snímka. Pod touto snímkou budú zobrazené náhľady na všetky importované snímky, ktoré bude možné zobraziť kliknutím na ne. Okrem zobrazenia samotnej snímky bude možné vykresliť používateľom definovanú mriežku, s ktorou bude možné interagovať pomocou myši.

4.11.3 Ľavý postranný panel

Ľavý postranný panel bude obsahovať dve sekcie – „Image information“ a „Grid algorithm parameters“.

„Image information“ sekcia bude informovať používateľa o mene pacienta nachádzajúceho sa na snímke. Okrem mena bude zobrazené číslo série a modalita, ktorej hodnota by pri skenoch MR mala byť rovnomenná (rovnajúca sa hodnote „MR“). Pod týmito informáciami sa budú nachádzať informácie o výške a šírke zobrazenej snímky.

„Grid algorithm parameters“ bude slúžiť pre nastavenie parametrov SPAMM algoritmu. Tlačidlo „Compute“ bude predvolene vypnuté, dokým nebudú vytvorené mriežky na všetkých importovaných snímkach. V opačnom prípade bude možné kliknúť na toto tlačidlo, ktoré spracuje potrebné informácie z každej snímky a odošle požiadavku SPAMM algoritmu s potrebnými informáciami pre aktualizovanie súradníc bodov všetkých mriežok.

4.11.4 Pravý postranný panel

Pravý postranný panel bude rozdelený na dve karty – „Image tools“ a „Grid tools“. Pri kliknutí na jednu z možností sa obsah patriaci aktuálne zobrazenej možnosti skryje, aby bolo možné zobraziť obsah zvolenej karty.

4.11.4.1 Image tools karta

Na „Image tools“ karte bude možné zvoliť si snímku, ktorá sa má zobraziť. Okrem toho bude možné zmeniť jas a kontrast všetkých snímiek. Pod týmito nastaveniami bude možné upravovať nastavenia animácie snímiek, počínajúc nastavením rýchlosti animácie cez nastavenie počiatočnej a koncovej animovanej snímky.

4.11.4.2 Grid tools karta

„Grid tools“ karta bude obsahovať rozličné nastavenia mriežky. Tie sa rozdelia na globálne a lokálne nastavenia. Globálne nastavenia budú aplikované pre všetky vytvorené mriežky a lokálne nastavenia len na mriežku, ktorá je aktuálne zobrazená.

Globálne nastavenia mriežky

V globálnych nastaveniach bude možné nastaviť možnosť interakcie s mriežkou pomocou myši (nastavenie „Move“). Ak bude zapnutá možnosť „Grid“, potiahnutím bodu mriežky sa celá mriežka posunie o vektor posunu.

V prípade zvolenej možnosti „Point“ sa nebude posúvať celá mriežka, ale iba myšou zvolený bod mriežky.

Ďalšie nastavenie – „Show refinement points“ – bude slúžiť na dynamické pridanie, resp. odobranie „refinement“ bodov zo všetkých mriežok.
Tzv. „refinement“ body slúžia pre presnejšie zarovnanie používateľom vygenerovanej mriežky so SPAMM mriežkou vygenerovanou MR prístrojom.

Po klliknutí na tlačidlo „Copy grid to all images“ sa zobrazí modálne okno, ktoré upozorní používateľa o možnosti skopírovania aktuálne zobrazenej mriežky do všetkých importovaných snímiek s možnosťou prípadného vrátenia tohto kroku. Toto tlačidlo bude aktívne iba v prípade, že na aktuálnej snímke bude zobrazená mriežka.

Lokálne nastavenia mriežky

V lokálnych nastaveniach bude možné nastaviť rôzne parametre zobrazenej mriežky ako uhol, offset a iné. Pomocou tlačidla „Create grid“ bude možné vytvoriť mriežku s predvolenými nastaveniami. Po kliknutí naň sa tlačidlo zmení na „Remove grid“, ktorého funkcia bude spočívať vo vymazaní mriežky na zobrazenej snímke.

4.12 Návrh komunikácie webového rozhrania so serverom

Pre potrebu komunikácie medzi serverom a klientom bude nutné túto komunikáciu navrhnúť.

Serveru sa budú posielať dáta o používateľom definovaných mriežkach pre výpočet aktualizovaných súradníc bodov mriežok. Odpoveď klientovi by mala obsahovať aktualizované súradnice mriežok, ktoré webová aplikácia následne vykreslí.

V tomto prípade stačí, ak server bude poskytovať jeden REST API endpoint, ktorý potrebné dáta príjme, spracuje a odpoveď pošle klientovi vo dohodnutom formáte.

Detaily o tomto API endpointe sú nasledovné:

- pre komunikáciu s endpointom bude využitý HTTP protokol,
- endpoint bude dostupný na adrese `<hostname>/api/grid`,
- endpoint prijme dáta vtedy, ak daná požiadavka bude poslaná metódou POST a
- telo požiadavky a odpovede budú vo formáte JSON.

4.12.1 HTTP požiadavka

Telo požiadavky bude mať nasledovný formát, ktorý je pre popis typov premenných popísaný v TypeScripte:

```
{
  "data": [{
      "image": {
        "imageId": string;
        "imageData": Uint8Array;
      },
      "grid": {
        "includesRefinementPoints": boolean;
        "primaryLines": [{
            "points": [{
                "x": number;
                "y": number;
                "isCommonPoint": boolean;
              },
              ...
            ]
          },
          ...
        ]
      }
    }
  ]
}
```

V `data` poli sa budú nachádzať objekty, z ktorých každý bude reprezentovať entitu skladajúcu sa zo štruktúry snímky (`image`) a jej mriežky (`grid`).

Hodnota `image` bude objektom, ktorého obsahom bude `imageId` reprezentujúci ID snímky a `imageData`, ktorý bude obsahovať pole binárnych dát DICOM súboru.

Hodnota `grid` bude taktiež objekt obsahujúci dva kľúče – `primaryLines` a `includesRefinementPoints`.

`primaryLines` bude reprezentovať pole zvislých úsečiek idúcich zľava doprava. Obsahom každej takejto úsečky bude pole `points`, ktorého obsahom budú objekty reprezentujúce body na danej úsečke.

Každý bod sa bude skladať z troch kľúčov objektu: `x` – reprezentujúci x súradnicu bodu, `y` – reprezentujúci invertovanú y súradnicu bodu a príznak `isCommonPoint` značiaci, či je daný bod „common“ bodom (tzv. bod, v ktorom sa pretína vodorovná a zvislá úsečka mriežky) alebo „refinement“ bodom, pomocou ktorého je možné upresniť polohu mriežky.

`includesRefinementPoints` značí, či sa v poli `points` nachádzajú aj tzv. „refinement“ body .

Používateľ iniciuje poslanie dát v tejto štruktúre kliknutím na tlačidlo „Compute“.

4.12.1.1 Anonymizácia DICOM dát

Nakoľko obsahom požiadavky v rámci horeuvedeného API endpointu budú taktiež binárne dáta DICOM snímiek, bude nevyhnutné tieto snímky pred ich odoslaním na server anonymizovať takým spôsobom, aby nebolo možné spojiť snímky z magnetickej rezonancie s konkrétnym pacientom.

Podľa [6] je nutné anonymizovať všetky DICOM tagy nachádzajúce sa v skupinách „0008“ a „0010“. Skupina „0008“ obsahuje dáta ohľadom štúdie a skupina „0010“ obsahuje dáta o pacientovi.

Pre tento účel bude potrebné vytvoriť triedu reprezentujúcu DICOM anonymizér, ktorý bude implementovaný v JavaScripte na úrovni klienta. Táto trieda nahradí obsah tagov z horeuvedených skupín prázdnym reťazcom s dĺžkou daného tagu, aby nenastala inkonzistencia v DICOM dátach. Takto upravené dáta bude môcť byť bezpečne poslané na server.

4.12.2 HTTP odpoveď

Telo odpovede servera na požiadavku, ktorá bude obsahovať dáta o odoslaných mriežkach, bude v nasledujúcom formáte:

```
{
  "grids": [{
    "imageId": string;
    "primaryLines": [{
      "points": [{
        "x": number;
        "y": number;
        "isCommonPoint": boolean;
      },
      ...
      ]
    },
    ...
    ]
  },
  ...
  ]
}
```

Odpoveď bude obsahovať kľúč `grids`, ktorého obsahom budú objekty mriežok. Každý z týchto objektov bude obsahovať kľúč `imageId` a `primaryLines`.

Pomocou kľúča `imageId` bude možné spárovať objekt mriežky v odpovedi s mriežkou v aplikácii. Obsahom kľúča `primaryLines` bude pole zvislých úsečiek mriežky idúce zľava doprava.

Každá takáto úsečka bude obsahovať kľúč `points` reprezentujúci pole bodov v grafickej reprezentácii mriežky zhora nadol. Každý bod bude obsahovať svoje súradnice (x a y) a príznak `isCommonPoint`.

Kapitola 5

Implementácia

Táto kapitola sa zaoberá celým procesom implementácie webovej aplikácie, od poopisu počiatočnej štruktúry projektu pri jeho inicializácii, cez zoznam použitých balíčkov s ich popisom a inicializáciou. Taktiež je venovaná pozornosť štruktúre komponentov používateľského rozhrania a implementácii aplikácie podľa prípadov použití. Koniec kapitoly sa venuje následnému nasadeniu hotovej webovej aplikácie.

5.1 Počiatočná štruktúra projektu

Implementácia webovej aplikácie započala vytvorením základnej štruktúry projektu pomocou oficiálneho nástroja pre tento účel – `nuxi`.

Príkazom `npx nuxi init diploma-thesis` sa vytvorila zložka `diploma-thesis`, ktorej štruktúra bola nasledovná:

```
diploma-thesis ..... Adresár určený pre implementáciu aplikácie
├── README.md . Markdown súbor obsahujúci inštrukcie pre spustenie serveru
├── app.vue ..... Vue SFC súbor obsahujúci uvítaciu správu
├── nuxt.config.ts ..... Nuxt.js konfiguračný súbor
├── package.json .... npm súbor obsahujúci popis projektu a jeho detaily s
│   potrebnými balíčkami
├── public ..... Adresár určený pre verejne dostupný obsah
└── tsconfig.json .....Konfigurácia TypeScriptu
```

5.1.1 Obsah súborov projektu

Obsahom `app.vue` súboru je Vue.js SFC komponent, ktorý reprezentuje Nuxt.js uvítaciu správu, ktorá sa zobrazí pri návšteve indexovej stránky servera.

Čo sa týka obsahu `nuxt.config.ts` súboru, ten je spočiatku prázdny. Účelom tohto súboru je konfigurácia Nuxt.js frameworku.

Pomocou súboru `tsconfig.json` je možné nakonfigurovať TypeScript pre celý projekt. Jeho počiatočným obsahom je referencia na `tsconfig.json` súbor samotného Nuxt frameworku — čo znamená, že sa pre tento projekt aplikujú pravidlá TypeScriptu predvolené pre Nuxt.js projekt.

Jedným z najdôležitejších súborov v tejto štruktúre je `package.json`. Obsah tohto súboru definuje meno a detaily projektu spolu
s balíčkami, ktoré sú využité v rámci samotného projektu. Popisu použitým balíčkom sa venuje nasledujúca sekcia.

5.1.2 Spustenie Nuxt.js serveru

Pred spustením Nuxt.js serveru je potrebné doinštalovať projektové závislosti – balíčky, príkazom `npm install`. Následne je možné spustiť Nuxt.js server príkazom `npm run dev`. Spustenie serveru týmto príkazom aktivuje tzv. „file watcher“, ktorý je zodpovedný za skompilovanie aplikácie pri detekovaných zmenách v súboroch aplikácie.

V rámci spustenia tohto príkazu sa na konzolu vypíše adresu servera spolu s jej portom, na ktorej je možné pristúpiť k tomuto serveru. V tomto prípade

sa jednalo o adresu `http://localhost:3000/`.

Pre produkčné násadenie by mala byť Nuxt.js aplikácia najprv prejsť build procesom (pomocou príkazu `npm run build`), ktorý okrem iného minifikuje aplikačné súbory, čím zmenší prenášaný obsah aplikácie po sieti.

5.2 Použité balíčky

V rámci tejto sekcie sú uvedené všetky balíčky, ktoré sa použili pri vývoji webovej aplikácie. Čo sa týka balíčkov, tie je možné rozdeliť na balíčky určené pre produkčné a pre vývojárske prostredie.

Zmyslom tohto delenia je definovať, aké balíčky sa majú nachádzať v produkčnej zostave aplikácie. V prípade, že je balíček pridaný do sekcie vývojárskych balíčkov, nebude sa nachádzať v produkčnej zostave. Týmto krokom sa sleduje zníženie veľkosti a zrýchlenie produkčnej zostavy projektu.

5.2.1 Balíčky pre produkčné prostredie

`cornerstone-core` – Balíček poskytujúci Cornerstone Core knižnicu, ktorá je zodpovedná za zobrazenie DICOM snímiek v správnom formáte a jednoduchú manipuláciu s týmito snímkami.

`@tarotoma/cornerstone-tools` – Fork `cornerstone-tools` balíčku, ktorý obsahuje rôzne nástroje určené pre prácu s DICOM snímkami. V tomto forku bol pridaný Grid nástroj implementovaný v rámci tejto diplomovej práce. Samotné knižnica taktiež upravuje prehrávanie animácií a pridáva vlastné TypeScript definície pre metódy používané vo webovej aplikácii, čo umožňuje lepšie našeptávanie IDE pri práci s týmto balíčkom.

`cornerstone-math` – Tento balíček umožňuje použiť Cornerstone Math knižnicu v projekte. Tá slúži pre rôzne výpočty v oblasti vektorovej matematiky. V tomto prípade je táto knižnica potrebná len ako závislosť balíčka `@tarotoma/cornerstone-tools`.

`hammerjs` – Účelom knižnice Hammer.js je pridanie podpory multi-dotykových gést pri práci s webovou apllikáciou. Táto funkcionalita nebola vo webovej aplikácii využitá, avšak je nutnou závislosťou balíčka `@tarotoma/cornerstone-tools`.

`dicom-parser` – Dicom Parser je knižnica určená pre parsovanie DICOM súborov do štruktúrovaných objektov, s ktorými je možné ďalej pracovať.

`@cornerstonejs/dicom-image-loader` – Cornerstone DICOM Image Loader a.k.a CDIL knižnica je určená pre importovanie DICOM súborov do webovej aplikácie. Táto knižnica umožňuje využiť Web Workers technológiu pre parsovanie DICOM súborov pomocou Dicom Parser knižnice.

`pinia` – Pinia.js je knižnica určená pre tzv. state management aplikácie. Umožňuje ukladať stav aplikácie tak, aby bol dostupný v rámci všetkých Vue.js komponentov. Bez použitia tejto knižnice by zdieľanie stavových premenných Vue.js komponentov s inými komponentami bolo komplikovanejšie a implementačne náročnejšie.

Stav aplikácie je možné rozdeliť do rôznych samostatných častí pomocou modulárnych „Stores“. Tie môžu byť importované nezávisle na sebe. Knižnica je taktiež plne implementovaná pomocou TypeScriptu a ponúka podporu pre konzolu prehliadača na inšpekciu aktuálneho stavu aplikácie.

`@pinia/nuxt` – Tento balíček pridáva podporu pre integráciu Pinie do Nuxt.js aplikácie.

`vuestic-ui` – V rámci tohto projektu sa využíva Vuestic UI kit, čo je knižnica obsahujúca rôzne UI komponenty, ktoré môžu byť importované samotnou aplikáciou. Tento UI kit bol vybraný zo subjektívneho dôvodu, nakoľko ponúka komponenty s dizajnom vhodným pre ich aplikovanie do aplikácie určenej na medicínske účely.

`@vuestic/nuxt` – Ako už názov napovedá, tento balíček pridáva integráciu Vuestic UI kitu do Nuxt.js aplikácie tak, aby nebola potrebná jeho ďalšia konfigurácia.

`@fortawesome/free-solid-svg-icons` – Uvedený balíček obsahuje FontAwesome[67] ikony, z ktorých sú niektoré použité v aplikácii v hornom paneli aplikácie.

[67] https://fontawesome.com

`@fortawesome/vue-fontawesome` – Úlohou tohto balíčka je transformovať importované FontAwesome ikony do znovupoužiteľneho komponentu, ktorý je následne možné použiť v aplikácii a pomocou neho vyrenderovať požadované ikony.

`arraybuffer-encoding` – Tento balíček implementuje konverziu dát z `ArrayBuffer` do `base64` a naspäť. Jeho využitie je popísané neskôr v rámci kompresie odosielaných dát na server.

`crypto-random-string` – Uvedený balíček implementuje správnu podporu generovania kryptograficky silných reťazcov. Tieto reťazce sú využívané pri anonymizácií DICOM dát pred ich odoslaním na server tak, aby nebolo možné priradiť odoslanú snímku konkrétnemu pacientovi.

5.2.2 Balíčky pre vývojárske prostredie

`nuxt` – Inštalácia tohto balíčka je nutná pre vytvorenie a vývoj v rámci Nuxt.js projektu.

`eslint` – ESLint je open-source JavaScript nástroj, ktorý kontroluje kód a reportuje prípadné nezrovnalosti v ňom. Medzi tieto nezrovnalosti partrí nekonzistentné odsadenie kódu, či používanie jazykových konštruktov, ktoré nedodržiavajú určité jednotné pravidlá. Tieto pravidlá je možné konfigurovať, ale aj importovať z iných projektov. Konfigurácia pravidiel, parsera a iných konfiguračných možností je možné pomocou `.eslintrc.cjs` súboru umiestneného v projekte.

`@typescript-eslint/parser` – Nakoľko ESLint nepodporuje kontrolu TypeScript kódu, keďže používa Espree parser podporujúci len JavaScript, je nutné použiť tento balíček pre pridanie podpory kontroly Typescript kódu.

`@typescript-eslint/eslint-plugin` – Tento balíček pridáva do ESLintu pravidlá pre TypeScript kód. Použitie tohto balíčka je odporúčané pri použití predchádzajúceho balíčka.

`sass` – Sass je CSS preprocesor implementovaný v JavaScripte, ktorý umožňuje vytvárať modulárnejší CSS kód, ktorý je určený pre jednoduchšie opätovné použitie. Okrem iného umožňuje používať CSS premenné, implementuje dedičnosť či vnáranie CSS pravidiel.

`sass-loader` – Úlohou tohto balíčka je transpilovať Sass CSS kód do čistého CSS kódu, ktorý je podporovaný CSS prehliadačmi. Pri budovaní projektu je nutné akýkoľvek CSS kód vytvorený pomocou Sass transpilovať do CSS, nakoľko Sass nie je podporovaný webovými prehliadačmi.

`@types/{cornerstone-core, hammerjs, node}` – Tieto tri balíčky pridávajú TypeScript definície typov `cornerstone-core`, `hammerjs` a `node` knižníc. Pridaním týchto balíčkov umožňuje používať typehinty pri práci s týmito balíčkami.

`vue-tsc` – Tento balíček je wrapper nástroja `tsc`, pomocou ktorého je možné transpilovať TypeScript kód do JavaScriptu. Nakoľko sa v projekte využívajú Vue SFC komponenty a samotný `tsc` balíček nepodporuje kontrolu TypeScriptu v týchto komponentách, je potrebné použiť balíček `vue-tsc` miesto `tsc` pre pridanie tejto podpory.

5.3 Inicializácia vybraných knižníc

Biznis logika spojená s akýmkoľvek Cornerstone balíčkom a ich závislosťami bude zahrnutá v rámci súboru `functions/Cornerstone.ts`. V ňom budú tieto balíčky importované a inicializované. Medzi knižnice, ktoré je nutné pred ich použitím inicializovať, patria Cornerstone Tools a Cornerstone DICOM Image Loader (CDIL). Ich inicializácia spočíva v nalinkovaní externých knižníc, ktoré dané dve knižnice využívajú v rámci ich implementácie. Bez tohto nalinkovania by niektoré metódy oboch knižníc nemuseli korektne fungovať a mohli by vyhadzovať výnimky.

Inicializácia Cornerstone Tools knižnice zahŕňa volanie metódy `init` s objektom možností, ktorú túto knižnicu pripraví pre jej použitie. Inicializácia CDIL knižnice prebieha podobne, avšak miesto `init` používa `configure` metódiu s objektom definujúcim jej konfiguráciu.

```
// Cornerstone.ts
import cornerstone from 'cornerstone-core';
import cornerstoneMath from 'cornerstone-math';
import cornerstoneTools from 'cornerstone-tools';
import cornerstoneDICOMImageLoader from '@cornerstonejs/dicom-image-loader';
import dicomParser from 'dicom-parser';
import Hammer from 'hammerjs';

/**
 * Initialize cornerstone libraries
 */
export const initLibraries = (): void => {
  // Setup all required cornerstone-tools dependencies
  cornerstoneTools.external.cornerstone = cornerstone;
  cornerstoneTools.external.cornerstoneMath = cornerstoneMath;
  cornerstoneTools.external.Hammer = Hammer;
  cornerstoneTools.init({
    mouseEnabled: true,
    showSVGCursors: false,
  });

  // Setup all required cornerstone-wado-image-loader dependencies
  cornerstoneDICOMImageLoader.external.cornerstone = cornerstone;
  cornerstoneDICOMImageLoader.external.dicomParser = dicomParser;
  cornerstoneDICOMImageLoader.configure({
    useWebWorkers: true,
    decodeConfig: {
      convertFloatPixelDataToInt: false,
    },
  });

  const config = {
    maxWebWorkers: navigator.hardwareConcurrency || 1,
    startWebWorkersOnDemand: false,
    taskConfiguration: {
      decodeTask: {
        initializeCodecsOnStartup: true,
        strict: false,
      },
    },
  };
  cornerstoneDICOMImageLoader.webWorkerManager.initialize(config);
}
```

5.4 Štruktúra komponentov používateľského rozhrania

Po inicializácii projektu, inštalácii knižníc a ich inicializácii je nutné zamyslieť sa nad štruktúrou samotných Vue komponentov, ktoré budú definovať používateľské rozhranie webovej aplikácie.

Keďže z návrhu používateľského prostredia vyplýva, že rozhranie bude rozdelené na 4 hlavné časti, ponúka sa možnosť rozdeliť potrebné komponenty taktiež do 4 zložiek.

Tými budú `top-panel`, `left-panel`, `main-window` a `right-panel`. Komponenty použité v rámci jednej časti používateľského rozhrania budú spadať do danej zložky. Akýkoľvek komponent použitý vo viacerých častiach by mal byť umiestnený v zložke `general`. Pre každú časť tohto rozhrania by mal byť taktiež vytvorený jeden hlavný komponent reprezentujúci danú časť aplikácie.

Jedná sa o nasledujúce komponenty: `TheTopPanel.vue`, `TheLeftPanel.vue`, `TheMainWindow.vue` a `TheRightPanel.vue`.

Tie budú importované v jednom entry-point komponente, ktorý bude definovať hlavnú (a jedinú) stránku aplikácie. Tento komponent bude niesť názov `index.vue` a bude umiestnený v priečinku `pages`, aby Nuxt.js spároval zobrazenie tohto komponentu pri návšteve index stránky. V tomto komponente bude taktiež importovaná metóda `initLibraries` uvedená v predchádzajúcej časti kódu. Tá bude exekuovaná v rámci metódy `onMounted`, ktorá zaručuje exekúciu kódu po tom, čo je používateľské rozhranie súčasťou Document Object Modelu (DOM) webovej aplikácie.

Výsledná štruktúra pre definovanie častí UI a ich komponentov je nasledujúca:

- `diploma-thesis` ..... Adresár určený pre implementáciu aplikácie
  - `components` ..... Adresár určený pre komponenty
    - `top-panel` ..... Adresár pre komponenty použité v hornom paneli
    - `left-panel` ..... Adresár pre komponenty použité v ľavom paneli
    - `main-window` ..... Adresár pre komponenty použité v centrálnej časti aplikácie
    - `right-panel` ..... Adresár pre komponenty použité v pravom paneli
    - `general` ..... Adresár pre komponenty použité vo viacerých častiach aplikácie
    - `TheTopPanel.vue` ... Vue SFC komponent reprezentujúci horný panel
    - `TheLeftPanel.vue` ... Vue SFC komponent reprezentujúci ľavý panel
    - `TheMainWindow.vue` ... Vue SFC komponent reprezentujúci centrálnu časť aplikácie
    - `TheRightPanel.vue` Vue SFC komponent reprezentujúci pravý panel
  - `pages` ..... Adresár obsahujúci stránky aplikácie
    - `index.vue` ..... Vue SFC komponent reprezentujúci index stránku aplikácie

5.5 Implementácia prípadov použitia

Prípady použitia budú implementované podľa ich poradia, v akom boli uvedené.

5.5.1 UC1 – Zobrazenie snímiek v aplikácii

V rámci tohto prípadu použitia je nutné implementovať import snímiek a ich zobrazenie vo webovej aplikácii.

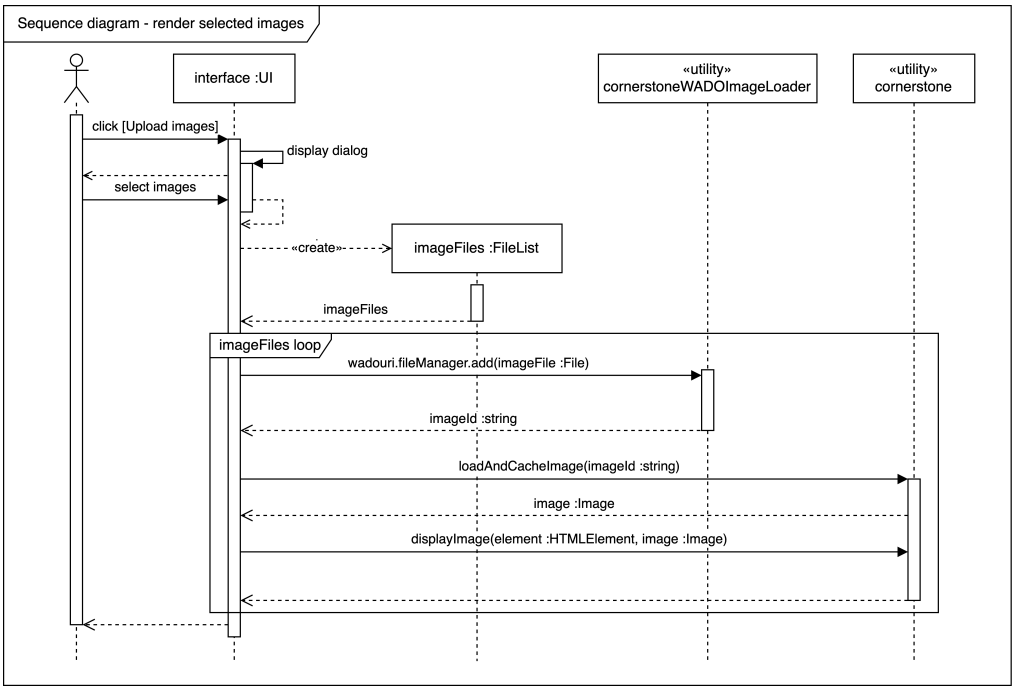
Pred importom snímiek je potrebné označiť vybraný HTML element za element zobrazujúci importované snímky z magnetickej rezonancie. Toto označenie spočíva v zavolaní metódy `cornerstone.enable(element: HTMLElement)` Cornerstone Core knižnice.

Samotný import DICOM snímiek sa dá realizovať pomocou tlačidla, ktoré bude naviazané na File API, ktoré je poskytované prehliadačmi a umožňuje načítať rôzne súbory do pamäti prehliadača.

Parsovanie importovaných DICOM snímiek je možné dosiahnuť pomocou CDIL knižnice. Následne sa použije Cornerstone Core knižnica pre uloženie týchto snímiek do internej cache. To zaručí, že sa pri neskoršom zobrazovaní snímiek nemusia tieto snímky znovu načítavať, čo zvyšuje používateľský komfort pri používaní aplikácie.

Následne je možné použiť statickú metódu `displayImage` Cornerstone Core knižnice pre zobrazenie MR snímky. Tá bude vykreslená vo zvolenom označenom HTML elemente.

Bližší pohľad na algoritmus zodpovedný za import snímiek, ich parsovanie až po výsledné zobrazenie je možné vidieť na nasledujúcom sekvenčnom diagrame.



Obr. 5.1 Sekvenčný diagram zobrazujúci import, parsovanie a zobrazenie DICOM snímiek

5.5.2 UC2 – Animácia snímiek

Knižnica Cornerstone Tools ponúka nástroj „PlayClip“, pomocou ktorého je možné importované snímky animovať.

Avšak po bližšej analýze tohto nástroja bolo zistené, že nie je konfigurovateľný do takej miery, ako vyžaduje alternatívny scenár tohto prípadu použitia. PlayClip nástroj neprijíma indexy snímiek ako argumenty, od akej, resp. po akú snímku sa animácia má prehrávať.

5.5.2.1 Vytvorenie a publikácia forku Cornerstone Tools knižnice

Táto skutočnosť viedla k potrebe vytvorenia vlastného forku Cornerstone Tools knižnice za účelom zmeny „PlayClip“ nástroja. Algoritmus tohto nástroja - metóda `playClip` – bol zmenený tak, aby prijímal chýbajúce argumenty a na ich základe animoval snímky. Pomocou metódy `stopClip` je možné prebiehajúcu animáciu zastaviť.

Takto upravená knižnica bola publikovaná do npm registru pod názvom `@tarotoma/cornerstone-tools`. Pre použitie upravenej knižnice v projekte je nutné zmazať existujúcu knižnicu a nahradiť ju forknutou verziou.

Na to, aby mohol „PlayClip“ nástroj korektne fungovať, je potrebné ho pridať spolu s nástrojom `stack` do tzv. „Stack stratégie manažovania stavu nástrojov“. K stavu nástrojov spravovaných touto stratégiou je možné pristupovať z akejkoľvek snímky. Predvolene je stav nástrojov oddelený pre každú snímku, nakoľko je pri väčšine nástrojov žiaduce mať zobrazené informácie viažuce sa k danej snímke.

Okrem úpravy „PlayClip“ nástroja boli taktiež vytvorené TypeScript definície typov pre metódy používané webovou aplikáciou.

5.5.3 UC3 – Vytvorenie mriežky

Vykreslenie mriežky nad zobrazenou snímkou patrí medzi hlavné funkčné požiadavky kladené na túto aplikáciu. Nakoľko Cornerstone Tools knižnica neposkytuje taký nástroj, bude nutné ho vytvoriť. Keďže je v projekte už použitý fork Cornerstone Tools knižnice, je priamočiarejšie implementovať tento nástroj rovno v rámci tohto forku.

Pred začatím implementácie nástroja mriežky je dôležité zoznámiť sa s anatómiou Cornerstone Tools nástrojov a jej API dokumentáciou.

5.5.3.1 Popis vybraných tried nástrojov a ich metód

Cornerstone Tools knižnica ponúka pre implementáciu anotačných nástrojov triedu `BaseAnnotationTool`, ktorá rozširuje základnú triedu nástrojov – `BaseTool`. Nástroje implementujúce triedu `BaseAnnotationTool` umožňujú vytvárať anotácie snímiek a náležite ich upravovať [44] (vlastný preklad).

`BaseTool` trieda

Táto trieda je zodpovedná za inicializáciu konfigurácie nástroja, dynamické aplikovanie dodatočnej funkcionality a virtuálne funkcie pre interakciu s nástrojom pomocou myši alebo dotyku [44] (vlastný preklad).

Poskytované virtuálne funkcie sú nasledovné:

- `preMouseDownCallback`,
- `postMouseDownCallback`,
- `preTouchStartCallback` a
- `postTouchStartCallback` [44] (vlastný preklad).

`preMouseDownCallback` je funkcia, ktorá je spustená pred tým, ako sa začne spracovávať `mousedown`[68] event emitovaný prehliadačom zo zvoleného elementu zobrazujúcom snímku. Predvolene táto callback funkcia nerobí nič [44] (vlastný preklad).

[68] https://developer.mozilla.org/en-US/docs/Web/API/Element/mousedown_event

`postMouseDownCallback` je funkcia, ktorá je spustená po spracovaní `mousedown` eventu taktiež emitovaného zo zvoleného elementu zobrazujúcom snímku. Predvolene taktiež nič nerobí [44] (vlastný preklad).

Posledné dve funkcie – `preTouchStartCallback` a `postTouchStartCallback` sú funkcie, ktoré sú funkcionalitou podobné funkciám `preMouseDownCallback` a `postMouseDownCallback`. Rozdiel medzi nimi je ten, že zatiaľ čo prvé dve reagujú na stlačenie tlačidla myši (`mousedown` event), posledné dve reagujú na dotyk dotykovej plochy (`touchstart`[69] event) [44] (vlastný preklad).

`BaseAnnotationTool` trieda

Táto trieda rozširuje počet virtuálnych metód o nasledujúce 4 metódy:

- `mouseMoveCallback`,
- `handleSelectedCallback`,
- `toolSelectedCallback` a
- `updateCachedStats` [44] (vlastný preklad).

`mouseMoveCallback` je funkcia, ktorá sa spustí pri detekcii `mousemove`[70] eventu. Ten je spúšťaný prehliadačom počas každého pohybu myšou [44] (vlastný preklad).

`handleSelectedCallback` je taktiež funkcia, ktorá sa avšak spúšťa po zvolení časti vykreslenej anotácie myšou alebo dotykom [44] (vlastný preklad).

`toolSelectedCallback` je funkcia, ktorá sa spustí pri zvolení daného nástroja [44] (vlastný preklad).

`updateCachedStats` funkcia je zodpovedná za aktualizáciou štatistík daného nástroja [44] (vlastný preklad).

[69] https://developer.mozilla.org/en-US/docs/Web/API/Element/touchstart_event
[70] https://developer.mozilla.org/en-US/docs/Web/API/Element/mousemove_event

Nakoľko sú všetky spomenuté metódy virtuálne, nemusia byť koniec koncov daným nástrojom implementované. Avšak `BaseAnnotationTool` trieda poskytuje aj abstraktné metódy, ktoré je nutné implementovať.

Ich zoznam je nasledovný:

- `createNewMeasurement`,
- `pointNearTool`,
- `distanceFromPoint` a
- `renderToolData` [44] (vlastný preklad).

`createNewMeasurement` metóda je zodpovedná za vytvorenie novej anotácie a jej uloženie do pamäte. Táto metóda sa exekuuje v rámci práve aktívneho nástroja ak žiaden nástroj neodpovedal na `mousedown` event [44] (vlastný preklad).

Metóda `pointNearTool` vráti pravdivú hodnotu, ak koordináty kliknutia myšou alebo dotyku na dotykovú plochu sa nachádzajú blízko anotácie nástroja. Ak áno, tak daná časť anotácie je poslaná ako argument `toolSelectedCallback` metódy [44] (vlastný preklad).

`distanceFromPoint` metóda vracia počet pixelov medzi zadanou súradnicou a najbližšou časťou anotácie [44] (vlastný preklad).

V rámci metódy `renderToolData` je nutné implementovať vykreslenie anotácie na snímku. Samotné vykreslenie sa dá dosiahnuť pomocou interných metód ako `drawJoinedLines` pre vykreslenie úsečiek spojených v určitých bodoch či `drawHandles` pre vykreslenie bodov, ktoré môžu byť presúvané po ploche snímky [44] (vlastný preklad).

5.5.3.2 Dostupné módy nástrojov

Cornerstone Tools nástroje sa môžu nachádzať v štyroch rôznych módoch, z ktorého je aplikovaný v čase vždy len jeden.

Zoznam týchto módov je nasledovný:

- aktívny mód,
- pasívny mód,
- zapnutý mód a
- vypnutý mód.

Ak je nástroj v aktívnom móde, je mu umožnené zobraziť jeho obsah a meniť svoj vnútorný stav. Nástroj v tomto stave môže tiež reagovať na interakciu používateľa s týmto nástrojom [45] (vlastný preklad).

Narozdiel od aktívneho módu, pasívny mód umožňuje všetko čo aktívny mód s tým rozdielom, že nie je možné vytvoriť nový stav nástroja (ako napr. vytvorenie novej zvislej úsečky mriežky), ale len manipulovať s jeho existujúcim stavom [45] (vlastný preklad).

Nástroje v zapnutom móde je možné vyrenderovať, avšak nereagujú na interakciu používateľa s takýmto nástrojom. Jedná sa vlastne o taký „read-only“ mód nástroja [45] (vlastný preklad).

Vypnutý mód nástroja narozdiel od všetkých uvedených módov neumožňuje s takýmto nástrojom interagovať a ani tento nástroj vykresliť. Tento mód je taktiež predvoleným módom pre každý nástroj [45] (vlastný preklad).

Zmena režimu je možná pomocou zavolania nasledovných metód:

- `cornerstoneTools.setToolActiveForElement(element: HTMLElement, toolName: string, options: object)` – volanie tejto metódy nastaví aktívny mód nástroja pre daný element,
- `cornerstoneTools.setToolPassiveForElement(element: HTMLElement, toolName: string, options: object)` – volanie tejto metódy nastaví pasívny mód nástroja pre daný element,
- `cornerstoneTools.setToolEnabledForElement(element: HTMLElement, toolName: string, options: object)` – volanie tejto metódy nastaví zapnutý mód nástroja pre daný element a

- `cornerstoneTools.setToolDisabledForElement(element: HTMLElement, toolName: string, options: object)` – volanie tejto metódy nastaví vypnutý mód nástroja pre daný element [45] (vlastný preklad).

Cornerstone Tools umožňuje danému nástroju reagovať na zmenu jeho módu pomocou nasledovných callback funkcií, ktoré je možné definovať v rámci triedy implementovaného nástroja:

- `activeCallback` – spustí sa po nastavení aktívneho módu,
- `passiveCallback` – spustí sa po nastavení pasívneho módu,
- `enabledCallback` – spustí sa po nastavení zapnutého módu a
- `disabledCallback` – spustí sa po nastavení vypnutého módu [45] (vlastný preklad).

5.5.3.3 Aktivácia vybraného nástroja

Použitie vybraného nástroja je možné až po jeho registrácii. Nasledujúca metóda `registerTool` v rámci súbora `functions/Cornerstone.ts` implementuje túto registráciu spolu s nastavením zapnutého módu registrovaného nástroja.

```
export const registerTool = (toolName: cornerstoneTools.ToolName): void => {
  const store = useGlobalStore() // store retrieval
  if (!store.mainImageContainer) {
    return
  }
  const fullToolName = toolName + 'Tool' as cornerstoneTools.FullToolName
  const tool = cornerstoneTools[fullToolName]
  cornerstoneTools.addToolForElement(store.mainImageContainer, tool)
  cornerstoneTools.setToolEnabledForElement(store.mainImageContainer, toolName)
}
```

5.5.3.4 Implementácia nástroja mriežky

Grid nástroj – nástroj mriežky – bol implementovaný pomocou vhodného doimplementovania abstraktných a virtuálnych funkcií `BaseTool` a `BaseAnnotationTool` tried uvedených v tejto práci.

V Cornerstone Tools knižnici je tento nástroj možný nájsť pod súborom `src/tools/annotation/GridTool.js`. Grid nástroj bol svojou implementá-

ciou inšpirovaný nástrojom `FreehandRoi`, ktorý umožňuje kresliť po snímku čiary rôznych dĺžok.

Taktiež boli v rámci Grid nástroja implementované pomocné metódy ako napr. metódy pre vytváranie vertikálnych úsečiek (`generateMainPrimaryLine`) a ich mazanie (`removeLastPrimaryLine`), metódy uľahčujúce prechádzanie týchto úsečiek (`getAllMainPrimaryLines`, `getPreviousPrimaryLines`, `getNextMainPrimaryLine`) či metódy určené pre generovanie pomocných úsečiek a ich bodov, na ktorých sa nachádzajú „refinement“ body (`generateSubsidiaryPrimaryLines` či `generateRefinementPointsOnCurrentPrimaryLines`).

Okrem horeuvedenej funkcionality je taktiež nutné myslieť na reprezentáciu mriežky ako stavu nástroja. Tomuto problému sa venuje nasledujúca podsekcia.

Reprezentácia a manažment stavu mriežky

Stav nástrojov je možné uložiť a získať pomocou metód `addToolState(element: HTMLElement, toolName: string)` a `getToolState(element: HTMLElement, toolName: string)` po ich importovaní zo súboru `stateManagement/toolState.js` nachádzajúceho sa v Cornerstone Tools knižnici.

Stav mriežky je reprezentovaný poľom obsahujúcim štruktúru vertikálnych úsečiek. Jedna takáto úsečka predstavuje tzv. measurement. V rámci measurementu sú uložené body tejto úsečky. Tie obsahujú x a y koordináty bodu spolu s príznakom, či je daný bod hlavným („common“) bodom alebo „refinement“ bodom.

Postupnosť uložených bodov predstavuje vertikálnu úsečku mriežky. Horizontálne úsečky budú automaticky dopočítané pri renderovaní mriežky pomocou susedných bodov priľahlých vertikálnych úsečiek.

Z horeuvedeného vyplýva, že sa pri volaní metódy `createNewMeasurement` x-krát vytvorí reprezentácia vertikálnej úsečky (measurement), ktoré budú v cykle ukladané do celkového stavu mriežky pomocou metódy `addToolState`.

Reprezentácia vertikálnej úsečky mriežky (tzv. „measurement“) bude vyzerať nasledovne:

```
const points = [
  {
    x: number;
    y: number;
    isCommonPoint: boolean;
  },
  ...,
  {
    x: number;
    y: number;
    isCommonPoint: boolean;
  }
];
```

Výstup metódy `getToolState` pre mriežku bude teda nasledujúci objekt:

```
{
  data: [{
    handles: {
      points: [
        {
          x: number;
          y: number;
          isCommonPoint: boolean;
        },
        ...,
        {
          x: number;
          y: number;
          isCommonPoint: boolean;
        }
      ]
    }
  },
  ...,
  ]
}
```

5.5.4 UC4 – Úprava parametrov mriežky

Webová aplikácia zobrazuje nielen mriežky vytvorené používateľom, ale aj mriežky prichádzajúce zo servera. Táto situácia znemožňuje si priebežne ukladať informácie o vytvorenej mriežke, nakoľko informácie o mriežke odoslanej zo serveru nebudú dostupné.

Na základe tejto skutočnosti je nutné parametre samotnej mriežky dopočítavať na základe uloženého stavu mriežky získaného pomocou už spomínanej metódy `getToolState`.

Nasledujúca sekcia sa venuje implementačným detailom týkajúcim sa zmeny konfiguráce mriežky používateľským vstupom a následnou komunikáciou aplikácie týchto zmien používateľskému rozhraniu aplikácie.

5.5.4.1 Zmena konfigurácie mriežky

Pre zmenu konfigurácie mriežky bolo nutné v jej triede vytvoriť setter metódy, ktorých úloha je úprava štruktúry mriežky podľa vstupu používateľa.

Zoznam implementovaných setter metód je nasledujúci:

- `angle(value: number)` – nastaví uhol mriežky
- `moveOneHandleOnly(value: boolean)` – nastaví mód pohybu mriežky
- `noOfPrimaryLines(value: number)` – nastaví počet vertikálnych úsečiek mriežky
- `noOfSecondaryLines(value: number)` – nastaví počet horizontálnych úsečiek mriežky
- `spacing(value: number)` – nastaví rozpätie medzi hlavnými bodmi mriežky
- `showRefinementPoints(value: boolean)` – vygeneruje/zmaže „refinement“ body

Webová aplikácia si priebežne ukladá referenciu na instanciu nástroja mriežky pomocou metódy `cornerstoneTools.getToolForElement)`, ktorej návratová hodnota je instancia nástroja. To umožňuje exekuovať implementované setter metódy nástroja mriežky priamo z Vue.js SFC komponentov.

5.5.4.2 Komunikácia medzi nástrojom mriežky a používateľským rozhraním

Pri akejkoľvek zmene v konfigurácii mriežky ako napr. zmena pozície bodu na mriežke, pridanie/odobratie vertikálnej/horizontálnej úsečky, úpravou konfigurácie mriežky alebo pri jej celkovom zmazaní je UI webovej aplikácie notifikované o zmene stavu mriežky príslušnej k zobrazenému snímku.

Táto funkcionalita je zaistená emitovaním eventov:

- `cornerstonetoolsmeasurementcompleted` a
- `cornerstonetoolsmeasurementremoved`

Event `cornerstonetoolsmeasurementcompleted` taktiež obsahuje aktuálny stav mriežky. Tieto eventy sú emitované elementom zobrazujúcim importované snímky z magnetickej rezonancie. Emitovanie týchto eventov je implementované v rámci nástroja mriežky.

Následne stačilo v komponente obsahujúcim element zobrazujúci importované snímky (`TheMainWindow.vue`) implementovať callback funkcie na horeuvedené eventy. Tieto callback funkcie sa následne postarajú o uloženie aktuálneho stavu mriežky pre zobrazenú snímku.

Taktiež bolo potrebné nastaviť callback funkciu pre `cornerstonenewimage` event, aby sa pri zmene zobrazenej snímky uložil aktuálny stav mriežky zobrazenej na danej snímke.

5.5.5 UC5 – Zadanie parametrov pre SPAMM algoritmus

Pre splnenie scenáru tohto prípadu použitia stačilo implementovať tri políčka pre zadanie čísiel. Ich hodnoty sa priebežne ukladajú do objektu, ku ktorému sa pristupuje pri posielaní dát na server za účelom výpočtu aktualizovaných koordinátov bodov mriežok.

5.5.6 UC6 – Spustenie SPAMM algoritmu a zobrazenie jeho výsledkov

Spustenie SPAMM algoritmu je implementované zozbieraním údajov o mriežkach a importovaných snímkoch tak, aby mohli byť tieto dáta odoslané spôsobom určeným podľa návrhu komunikácie so serverom, ktorý je možný nájsť v sekcii 4.12.1.

5.5.6.1 Vytvorenie API endpointu

Na to, aby klient mohol komunikovať so serverom, bolo potrebné vytvoriť API endpoint podľa jeho špecifikácie v 4.12.

Pre tieto účely bol vytvorený súbor `server/api/grid.post.ts`, ktorý obsahuje implementáciu daného endpointu. Telo požiadavky je daným endpointom validované – v prípade že nevyhovuje špecifikácii tela požiadavky, server vráti chybovú správu so správou, ktorá špecifikuje jej príčinu.

Ak telo požiadavky spĺňa dané náležitosti, je požiadavka naďalej spracovávaná. Následne je poslaná odpoveď klientovi na túto požiadavku s aktualizovanými údajmi mriežok. Webová aplikácie túto odpoveď spracuje a jej obsah vykreslí pomocou delegovania tejto funkcionality Grid nástroju.

5.5.6.2 Anonymizácia dát

Pre účely anonymizácie DICOM dát bola vytvorená trieda `DicomAnonymizer`, ktorá anonymizuje všetky nájdené tagy v rámci skupín „0008“ a „0010“.

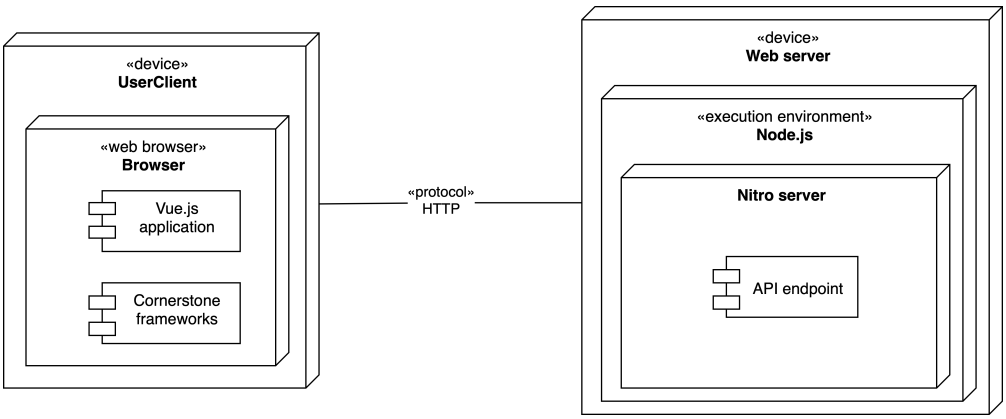
V rámci konštruktoru je predaný objekt s obsahom binárnych dát DICOM súboru. Implementovaná metóda `getAnonymizedImage` vracia požadovaný anonymizovaný DICOM súbor, ktorý je následne vhodný pre jeho zaslanie po sieti. Táto metóda je exekuovaná v rámci vytvárania tela požiadavky na horeuvedený endpoint.

5.6 Nasadenie aplikácie

Pre zrhnutie je na nasledujúcom diagrame nasadenia zobrazená výsledná štruktúra nasadenia implementovanej webovej aplikácie.

Jej frontendová časť beží u klienta – tá zahŕňa používateľské rozhranie implementované pomocou Vue.js aplikácie a manipuláciu s obrazovými dátami pomocou frameworkov rodiny Cornerstone.

V rámci backendu aplikácie beží Nitro server s implementáciou jediného API endpointu, ktorý bol uvedený vyššie.



Obr. 5.2 Diagram nasadenia zobrazujúci štruktúru webovej aplikácie

Výsledná aplikácia bola nasadená pomocou služby Vercel.[71] Táto služba poskytuje aj integráciu so službou GitHub, určenou pre vytváranie `git` repozitárov.

[71] https://vercel.com/

Keďže bola implementovaná webová aplikácia vyvíjaná s využitím GitHubu, bola táto služba prepojená s repozitárom obsahujúcim kód tejto aplikácie. Prepojenie git repozitára a služby Vercel umožnilo automatické publikovanie aplikácie, ktoré bolo spúšťané po každej implementovanej zmene aplikácie. Tento krok umožnil mať vždy nasadenú najnovšiu verziu aplikácie online.

Potreba nasadenia aplikácie taktiež vzišla z dôvodu odtestovania implementovanej webovej aplikácie testermi a ľuďmi z IKEMu.

Kapitola 6

Testovanie

Kapitola 7

Odporúčania pre ďalší vývoj webovej aplikácie

V budúcnosti by bolo vhodné zmigrovať webovú aplikáciu na novšie verzie knižníc z Cornerstone rodiny v prípade, že ich stabilné verzie budú dostupné. Jedná sa o knižnice Cornerstone Core, Cornerstone Tools a Cornerstone DICOM Image Loader.

Cornerstone Core knižnica by mala byť nahradená knižnicou Cornerstone3D, ktorej balíček je možné nájsť v registri npm pod názvom `@cornerstonejs/core`. Pre zjednodušenie migrácie vývojári Cornerstone Core knižnice taktiež uvoľnili sprievodcu migráciou[72] na túto knižnicu.

Balíček novej Cornerstone Tools knižnice je tiež už dostupný v npm registri, pod názvom `@cornerstonejs/tools`. Pred migráciou na novšiu verziu knižnice bude potrebné zanalyzovať, akým spôsobom bude importovaný implementovaný Grid nástroj slúžiaci pre vykreslenie a úpravu mriežky. Výstupom tejto analýzy by mali byť možnosti importovania tohto nástroja, z ktorých by mal byť jeden postup importovania vybraný a následne implementovaný.

Cornerstone DICOM Image Loader knižnica bude taktiež nahradená svojou novšou verziou – Streaming Image Volume Loader knižnicou. Tá podporuje zobrazenie snímku už počas jeho načítavania.

[72] https://www.cornerstonejs.org/docs/migrationGuides

Novú knižnicu je možné nájst v npm registri pod menom `@cornerstonejs/streaming-image-volume-loader`.

Nakoľko sú novšie verzie týchto knižníc naprogramované v TypeScripte, umožnia jednoduchšiu migráciu zo súčasnej implementácie. Väčšiu časovú záťaž v rámci migrácie spôsobí najmä zmigrovanie nástroja pre vykreslenie a úpravu mriežky.

Taktiež by bolo vhodné pokračovať na implementácii `grid-tracker` podprogramu a jeho komunikácii s webovou aplikáciou. Týmto krokom by sa dosiahol väčší potenciál webovej aplikácie, ktorá by následne mohla byť použitá pre detekciu anomálií v pohybe myokardu.

Kapitola 8

Záver

Bibliografia

1. MACKIEWICH, Blair. *Intracranial boundary detection and radio frequency correction in magnetic resonance images*. Burnaby, 1995. Dostupné tiež z: `https://summit.sfu.ca/item/6770`. Diplomová práca. Simon Fraser University.
2. LAM, Peter et al. What to know about MRI scans. In: Healthline Media. *Medical News Today* [online]. Júl 24, 2018 [cit. 2023-03-24]. Dostupné tiež z: `https://www.medicalnewstoday.com/articles/146309#what-is-an-mri-scan`.
3. MAZÁNKOVÁ, Jitka. *Kontrastní látky a jejich nežádoucí účinky*. Brno, 2011. Dostupné tiež z: `https://is.muni.cz/th/rsk3f/`. Bakalárska práca. Masarykova univerzita, Lekárska fakulta.
4. JÍRALOVÁ, Tereza. *MR - nové trendy a význam v moderní diagnostice*. České Budějovice, 2021. Dostupné tiež z: `https://theses.cz/id/pk75p4/`. Bakalárska práca. Jihočeská univerzita v Českých Budějovicích, Zdravotně sociální fakulta.
5. ELSTER, Allen D. What is SPAMM? In: *Myocardial tagging/SPAMM - Questions and Answers in MRI* [online]. Elster LLC, 2023 [cit. 2023-03-24]. Dostupné tiež z: `https://mriquestions.com/taggingspamm.html`.
6. VARMA, Dandu Ravi. Managing DICOM images: Tips and tricks for the radiologist. *Indian Journal of Radiology and Imaging*. 2012, roč. 22, č. 01, s. 4–13.

7. About DICOM format. In: *DICOM Library* [online]. DICOM Library, 2023 [cit. 2023-03-29]. Dostupné tiež z: `https://www.dicomlibrary.com/dicom/`.

8. History. In: *dicomstandard.org* [online]. The Medical Imaging Technology Association, [2023] [cit. 2023-03-29]. Dostupné tiež z: `https://www.dicomstandard.org/history`.

9. Transfer Syntax. In: *DICOM Library* [online]. DICOM Library, 2023 [cit. 2023-03-29]. Dostupné tiež z: `https://www.dicomlibrary.com/dicom/transfer-syntax/`.

10. BRYANT, David. About Qt. In: *Qt Wiki* [online]. The Qt Company, 18. júla 2022 [cit. 2023-03-29]. Dostupné tiež z: `https://wiki.qt.io/index.php?title=About_Qt`.

11. Qt Core 5.15.13. In: *Qt Documentation* [online]. The Qt Company, 2023 [cit. 2023-03-29]. Dostupné tiež z: `https://doc.qt.io/qt-5/qtcore-index.html`.

12. Qt Widgets 5.15.13. In: *Qt Documentation* [online]. The Qt Company, 2023 [cit. 2023-03-29]. Dostupné tiež z: `https://doc.qt.io/qt-5/qtwidgets-index.html`.

13. Qt GUI 5.15.13. In: *Qt Documentation* [online]. The Qt Company, 2023 [cit. 2023-03-29]. Dostupné tiež z: `https://doc.qt.io/qt-5/qtgui-index.html`.

14. Qt Test 5.15.13. In: *Qt Documentation* [online]. The Qt Company, 2023 [cit. 2023-03-29]. Dostupné tiež z: `https://doc.qt.io/qt-5/qttest-index.html`.

15. qmake Manual. In: *Qt Documentation* [online]. The Qt Company, 2023 [cit. 2023-03-29]. Dostupné tiež z: `https://doc.qt.io/qt-5/qmake-manual.html`.

16. Description. *dicom.offis.de* [online]. [2023] [cit. 2023-03-29]. Dostupné tiež z: `https://dicom.offis.de/dcmtk.php.en`.

17. LANGR, Daniel et al. *Úvod do OpenMP* [prednáška]. 2023. Dostupné tiež z: `https://courses.fit.cvut.cz/NI-PDP/media/lectures/NI-PDP-Prednaska02-OpenMP.pdf`. Praha: ČVUT v Praze, Fakulta informačních technologií, 27.02.2023.

18. *Template Numerical Library* [online]. 2023 [cit. 2023-03-29]. Dostupné tiež z: `https://tnl-project.org`.

19. KAFKA, Jiří. *Vývoj aplikace pro analýzu pohybu srdečních komor*. Praha, 2015. Diplomová práca. České vysoké učení technické v Praze, Fakulta jaderná a fyzikálně inženýrská.

20. HTML Living Standard. *Web Hypertext Application Technology Working Group (WHATWG)*. 2023 [cit. 2023-05-01]. Dostupné tiež z: `https://html.spec.whatwg.org`.

21. CSS basics. In: *MDN Web Docs* [online]. Mozilla Corporation, 2023 [cit. 2023-05-01]. Dostupné tiež z: `https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/CSS_basics`.

22. TAB ATKINS JR. Elika J. Etemad, Florian Rivoal. CSS Snapshot 2022. In: *World Wide Web Consortium (W3C)* [online]. World Wide Web Consortium, 22. November 2022 [cit. 2023-04-17]. Dostupné tiež z: `https://www.w3.org/TR/css-2023/`.

23. GUO, Shu-yu et al. ECMAScript® 2024 Language Specification. In: *TC39* [online]. Ecma International, 2023 [cit. 2023-04-16]. Dostupné tiež z: `https://tc39.es/ecma262/`.

24. TypeScript for the New Programmer. In: *TypeScript: JavaScript With Syntax For Types* [online]. Microsoft, Apr 12, 2023 [cit. 2023-04-16]. Dostupné tiež z: `https://www.typescriptlang.org/docs/handbook/typescript-from-scratch.html`.

25. About Node.js®. In: *Node.js* [online]. OpenJS Foundation, 2023 [cit. 2023-04-17]. Dostupné tiež z: `https://nodejs.org/en/about`.

26. STIJN, Sebastiaan van. Docker overview. In: *Docker documentation* [online]. Docker Inc., 2023 [cit. 2023-04-17]. Dostupné tiež z: `https://docs.docker.com/get-started/overview/`.

27. C++ addons v18.16.0. In: *Node.js* [online]. OpenJS Foundation, 2023 [cit. 2023-04-13]. Dostupné tiež z: `https://nodejs.org/docs/latest-v18.x/api/addons.html`.

28. MOZILLA.ORG CONTRIBUTORS. WebAssembly Concepts. In: *MDN Web Docs* [online]. Mozilla Corporation, Mar 23, 2023 [cit. 2023-04-13]. Dostupné tiež z: `https://developer.mozilla.org/en-US/docs/WebAssembly/Concepts`.

29. MOZILLA.ORG CONTRIBUTORS. Compiling a New C/C++ Module to WebAssembly. In: *MDN Web Docs* [online]. Mozilla Corporation, Mar 24, 2023 [cit. 2023-04-13]. Dostupné tiež z: `https://developer.mozilla.org/en-US/docs/WebAssembly/C_to_wasm`.

30. MOZILLA.ORG CONTRIBUTORS. Emscripten Diagram. In: *MDN Web Docs* [online]. Mozilla Corporation, Mar 23, 2023 [cit. 2023-04-13]. Dostupné tiež z: `https://developer.mozilla.org/en-US/docs/WebAssembly/Concepts/emscripten-diagram.png`.

31. STEPANYAN, Ingvar. Using WebAssembly threads from C, C++ and Rust. In: *web.dev* [online]. Júl 12, 2021 [cit. 2023-04-13]. Dostupné tiež z: `https://web.dev/webassembly-threads/`.

32. Introduction. In: *Nuxt: The Intuitive Web Framework* [online]. Nuxt, 2023 [cit. 2023-04-18]. Dostupné tiež z: `https://nuxt.com/docs/getting-started/introduction`.

33. Introduction. In: *Vue.js* [online]. 2023 [cit. 2023-04-18]. Dostupné tiež z: `https://vuejs.org/guide/introduction.html`.

34. What is Next.js? In: *Next.js by Vercel - The React Framework for the Web* [online]. Vercel, Inc., 2023 [cit. 2023-04-18]. Dostupné tiež z: `https://nextjs.org/learn/foundations/about-nextjs/what-is-nextjs`.

35. Describing the UI. In: *React* [online]. Meta Open Source, 2023 [cit. 2023-05-01]. Dostupné tiež z: `https://react.dev/learn/describing-the-ui`.

36. Writing Markup with JSX. In: *React* [online]. Meta Open Source, 2023 [cit. 2023-04-18]. Dostupné tiež z: `https://react.dev/learn/writing-markup-with-jsx`.

37. LLOBERA, Lewis. Adding a Stylesheet. In: *Create React App* [online]. Facebook, Inc., 2/13/2020 [cit. 2023-04-18]. Dostupné tiež z: `https://create-react-app.dev/docs/adding-a-stylesheet`.

38. SELBEKK, Kristofer Giltvedt. Adding a CSS Modules Stylesheet. In: *Create React App* [online]. Facebook, Inc., Oct 7, 2018 [cit. 2023-04-18]. Dostupné tiež z: `https://create-react-app.dev/docs/adding-a-css-modules-stylesheet`.

39. State: A Component's Memory. In: *React* [online]. Meta Open Source, 2023 [cit. 2023-04-18]. Dostupné tiež z: `https://react.dev/learn/state-a-components-memory`.

40. HAFEY, Chris. Cornerstone Core README. In: *GitHub* [online]. Sep 13, 2021 [cit. 2023-05-01]. Dostupné tiež z: `https://github.com/cornerstonejs/cornerstone`.

41. HAFEY, Chris. Cornerstone DICOM Image Loader README. In: *GitHub* [online]. Apr 1, 2014 [cit. 2023-05-01]. Dostupné tiež z: `https://github.com/cornerstonejs/cornerstone3D-beta/tree/main/packages/dicomImageLoader`.

42. MOZILLA.ORG CONTRIBUTORS. Using Web Workers. In: *MDN Web Docs* [online]. Mozilla Corporation, Mar 16, 2023 [cit. 2023-04-16]. Dostupné tiež z: `https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API/Using_web_workers`.

43. HAFEY, Chris. dicomParser README. In: *GitHub* [online]. Mar 29, 2014 [cit. 2023-05-01]. Dostupné tiež z: `https://github.com/cornerstonejs/dicomParser`.

44. BROWN, Danny. Tool Types. In: *tools.cornerstonejs.org* [online]. Sep 21, 2018 [cit. 2023-05-01]. Dostupné tiež z: `https://tools.cornerstonejs.org/tool-types/`.

45. BROWN, Danny. Anatomy of a Tool. In: *tools.cornerstonejs.org* [online]. Sep 21, 2018 [cit. 2023-05-01]. Dostupné tiež z: `https://tools.cornerstonejs.org/anatomy-of-a-tool/#modes`.

Dodatok A

Zoznam použitých skratiek

API Application Programming Interface

AR Augmentovaná realita

CSS Cascading Style Sheets

CT Počítačová tomografia

ČVUT České vysoké učení technické

DICOM Digital Imaging and Communications in Medicine

FID Free Induction Decay

FJFI Fakulta jaderná a fyzikálně inženýrská

GCC GNU Compiler Collection

HTML HyperText Markup Language

HTTP Hypertext Transfer Protocol

ID Identifikátor

IDE Integrated Development Environment

JS JavaScript

MB Megabajt

MR Magnetická rezonancia

NEMA National Electrical Manufacturers Association

SPAMM Spatial Modulation of Magnetization

TE Time to Echo

TNL Template Numerical Library

TR Repetition Time

UI User Interface

VM Virtuálny počítač

VR Virtuálna realita

W3C World Wide Web konzorcium

WADO Web Access to DICOM Objects

WWW World Wide Web

Dodatok **B**

Obsah priloženého média

- `src` ..... adresár so zdrojovými kódmi
  - `app` ..... adresár so zdrojovými kódmi aplikácie
  - `thesis` ..... adresár so zdrojovým kódom práce vo formáte LaTeX
- `text` ..... text práce
  - `DP_Tomáš_Taro_2023.pdf` ..... text práce vo formáte PDF