VŠB TECHNICKÁ UNIVERZITA OSTRAVA
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Nástroj pro anotaci medicínských snímků

Medical Images Annotation Tool

Sara Hýžů

Bakalářská práce

Vedoucí práce: Ing. Jan Gaura, Ph.D.

Ostrava, 2025

VŠB TECHNICKÁ UNIVERZITA OSTRAVA | FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Č.j.VSB/25/018502
vsboee250061C9

Zadání bakalářské práce

Student: **Sara Hýžů**

Studijní program: B0613A140014 Informatika

Téma: Nástroj pro anotaci medicínských snímků
Medical Images Annotation Tool

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem práce je implementovat multiplatformní aplikaci pro anotaci medicínských snímků v prostředí internetu a desktopu. Aplikace bude schopna anotovat jednovrstvá i vícevrstvá data. Pro zpracovnání a analýzu medicínckých snímků je nutno nejprve ve snímcích označit zájmová místa. Pro takové označení se používají anotační aplikace. Mnoho z nich však buď neumí označovat medicínská data nebo naopak není schopna načítat specifické medicínské snímky. Výsledná aplikace by měla tyto nedostatky odstranit.

Ve své práci proveďte:
1. Nastudujte dostupné knihovny pro načítání medicínských dat.
2. Naimplementujte vlastní aplikaci pro anotaci obrazů v prostředí desktopu a internetu (předpokládá se využití frameworku Tauri nebo Electron).
3. Svou implementaci náležitě otestujte.
4. Implementaci náležitě popište v textu práce a dosažené výsledky zdokumentujte v závěru práce.

Seznam doporučené odborné literatury:

[1] Tauri, https://tauri.app/
[2] Pydicom, https://pydicom.github.io/

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Jan Gaura, Ph.D.**

Datum zadání: 01.09.2024

Datum odevzdání: 30.04.2025

Garant studijního programu: doc. Mgr. Miloš Kudělka, Ph.D.

V IS EDISON zadáno: 24.02.2025 09:52:34

Abstrakt

Tato bakalářská práce se zabývá návrhem a implementací multiplatformní aplikace pro anotaci medicínských snímků, která je určena pro použití jak v prostředí desktopových operačních systémů (Windows, Linux, macOS), tak v běžných webových prohlížečích. Hlavním cílem bylo vytvořit intuitivní a výkonný nástroj, který umožní lékařům snadno a přesně označovat zájmové oblasti v medicínských snímcích ve formátu DICOM.

Aplikace byla vyvinuta pomocí moderního frameworku Tauri, kterýkterý kombinuje výkon nativních desktopových aplikací s flexibilitou webových technologií. Uživatelské rozhraní bylo vytvořeno pomocí frameworku Svelte a jazyka TypeScript.

Výsledná aplikace podporuje načítání DICOM snímků, práci s anotacemi včetně jejich exportu do formátu JSON, přizpůsobení rozhraní různým velikostem obrazovek a funguje i bez připojení k internetu. Důraz byl kladen nejen na uživatelskou přívětivost a vysokou přesnost anotací, ale také na spolehlivost, stabilitu a snadnou udržovatelnost kódu.

Klíčová slova

multiplatformní aplikace, anotace medicinských snímků, DICOM, Tauri

Abstract

This bachelor's thesis focuses on the design and implementation of a cross-platform application for annotating medical images, intended for use on both desktop operating systems (Windows, Linux, macOS) and in modern web browsers. The main goal was to develop an intuitive and powerful tool that enables doctors to easily and precisely annotate regions of interest in medical images in the DICOM format.

The application was developed using the modern Tauri framework, which combines the performance of native desktop applications with the flexibility of web technologies. The user interface is built with Svelte and TypeScript.

The resulting application supports loading DICOM images, working with annotations including exporting them to JSON format, responsive interface for different screen sizes, and functioning without an internet connection. Emphasis was placed not only on user-friendliness and high annotation accuracy, but also on reliability, stability, and ease of code maintainability.

Keywords

cross-platform application, anotation of medical images, DICOM, Tauri

Poděkování

Ráda bych na tomto místě poděkovala vedoucímu mé bakalářské práce Ing. Janu Gaurovi, Ph.D. za jeho cenné rady, trpělivost a ochotu mi pomoci. Dále bych chtěla poděkovat Bc. Barboře Kovalské, Bc. Martinu Korotwitschkovi, Bc. Petrovi Krautwurstovi, Evelyn Rei a Emerii Koudelkové za pomoc při testování aplikace a poskytnutí zpětné vazby.

Obsah

Seznam obrázků

Seznam ukázek kódu

Seznam tabulek

Kapitola 1

Úvod

Lékaři používají při procesu vyšetření a diagnostice pacientů různé typy přístrojů, které umožňují nahlédnout do lidského těla a poskytnout tak lékařům cenné informace. Ať už se jedná o výpočetní tomografii (CT), magnetickou rezonanci (MRI), pozitronovou emisní tomografii (PET) nebo ultrazvuk (UZ), všechny tyto přístroje generují snímky, které jsou následně vyhodnoceny lékařem. Při analýze těchto snímků je potřeba označit důležitá zájmová místa, která mohou být příznaky různých onemocnění nebo abnormalit. K tomu slouží anotační programy, které umožňují lékařům označit tato místa a přidat k nim poznámky, které umožní dalším lékařům se na snímku lépe orientovat.

Z pohledu lékařů je důležité, aby anotační aplikace umožnili označit oblast s vysokou přesností. Kvůli náročnému pracovnímu režimu a velkému množství snímků, které musí lékaři zpracovat, je dále taky potřeba, aby byla aplikace rychlá a snadná na použití. V neposlední řadě je podstatná konzistence zobrazení a anotací, aby lékaři mohli snadno porovnávat snímky mezi sebou a sdílet je s kolegy.

Z pohledu vývojáře je potřeba zajistit správné načítání a zobrazení snímků ve vysokém rozlišení a podporu pro různé formáty snímků, jako jsou například DICOM nebo JPEG, společně s podporou pro všechny typy přístrojů generujících tyto snímky (MRI, CT, UZ, atd). Taktéž je nutné dbát na intuitivnost uživatelského rozhraní, které bude vyhovovat potřebám odborníků.

V současné době existuje mnoho různých aplikací, které umožňují lékařům označovat tato místa. Většina z těchto aplikací je však uzpůsobena pouze pro určité typy snímků, neumožňuje označovat data, nebo nepodporuje vícevrstvá data. Tato práce si klade za cíl navrhnout univerzální multiplatformní aplikaci na anotaci medicínských snímků pro webové i desktop prostředí, která tyto nedostatky odstraní.

Kapitola 2

Analýza podobných řešení

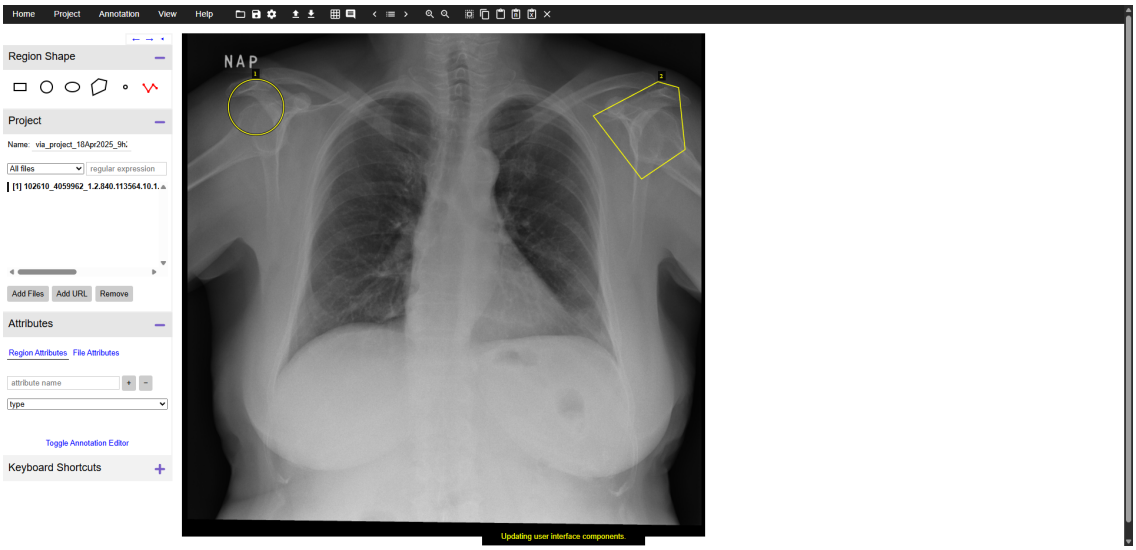
Jak bylo již v předchozí kapitole zmíněno, existuje již mnoho nástrojů pro práci s medicínskými snímky. Před návrhem vlastního řešení jsem se rozhodla prozkoumat jaká řešení jsou již dostupná a seznámit se s jejich výhodami, nevýhodami a s tím, jak moc odpovídají potřebám zmíněným v úvodu této práce.

Vyzkoušela jsem programy VGG Image Annotator, Computer Vision Annotation Tool a Radi-Ant DICOM Viewer a následně je zhodnotila na základě jak objektivních kritérií (podpora formátů, přesnost zobrazení, typy anotací) tak i subjektivních (snadnost použití, intuitivnost ovládání). Rozbory jednotlivých nástrojů jsou uvedeny níže.

2.1 VGG Image Annotator (VIA)

Jedná se o svobodně dostupný nástroj pro anotaci obrázků vyvinutý na Oxfordske univerzitě. Jeho hlavní výhodou je, že funguje přímo v prohlížeči bez potřeby instalace a je velmi jednoduchý na ovládání (obr. 2.1). Umožňuje anotaci obrázků pomocí různých typů anotací jako jsou obdélníky, elipsy, polygonální oblasti a body [1]. Jeho nevýhodou však je, že neumožňuje práci se soubory typu DICOM a tedy není vhodný pro medicínské snímky. Pohyb v obrázku je poměrně pomalý a při změně velikosti obrázku se vždy posune do levého horního rohu, což způsobuje neefektivní práci.

Tento program je vhodný pro obecnou anotaci obrázků a videí, ale není stavěný ani vhodný pro medicínské snímky. Snímky by bylo potřeba nejdříve převést do jiného formátu, což by bylo časově náročné. Zároveň práce v něm je sice poměrně jednoduchá, ale neefektivní a pomalá.



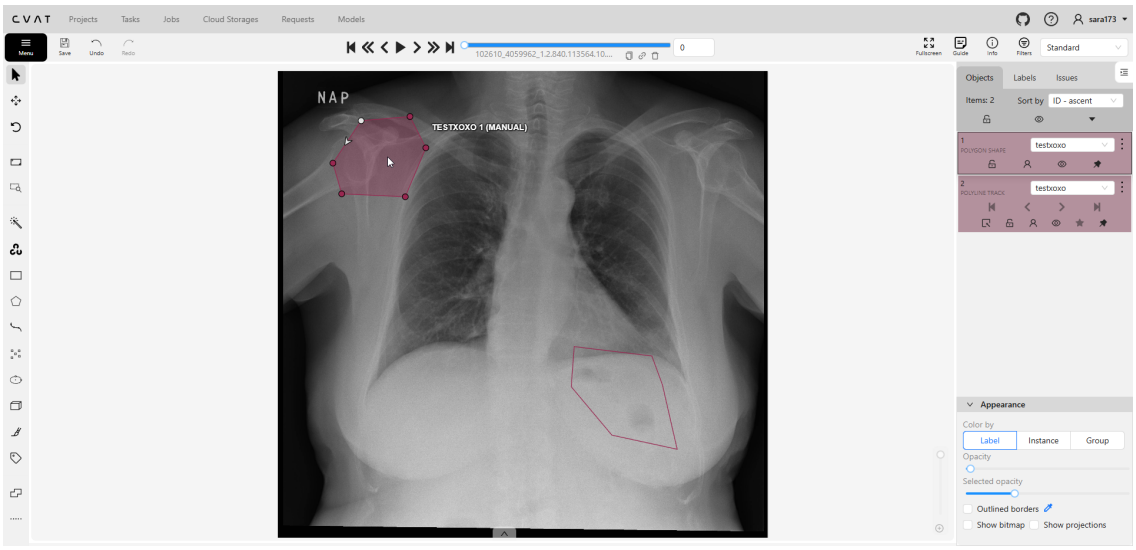
Obrázek 2.1: Ukázka programu VGG Image Annotator (VIA)

2.2 Computer Vision Annotation Tool (CVAT)

Jedná se o software pro anotaci obrazových dat, který je dostupný jak v komerční, tak ve svobodné komunitní verzi. Pro testování jsem použila komunitní verzi, která je dostupná zdarma a je možné ji nainstalovat na vlastní server nebo využít řešení běžící v prostředí cloudu [2].

Obrovskou výhodou tohoto nástroje jsou jeho schopnosti automatické segmentace a anotace, které jsou založeny na strojovém učení a umělé inteligenci. Editor anotací (obr. 2.2) je v tomto programu velmi přehledný a obsahuje mnoho různých typů anotací včetně polygonů, čar, bodů a obdélníků, ale také i 3D objektů. Co však musím této aplikaci vytknout je komplikovanější správa souborů, ve které je obtížnější se zprvu zorientovat a zdržuje celý proces anotace. Dále i přes to, že se aplikace na svém webu označuje za vhodnou pro medicínské snímky, tak nepodporuje formát DICOM [2].

Tento nástroj je k anotaci medicínských snímků použitelný, ale je potřeba nejdříve převést snímky do jiného formátu, což by bylo časově náročné. Zároveň práce v něm je sice poměrně jednoduchá, ale neefektivní a pomalá. Poskytuje však velmi pokročilé možnosti anotace a segmentace, které by mohly být užitečné pro pokročilejší uživatele a pro výzkumné účely.

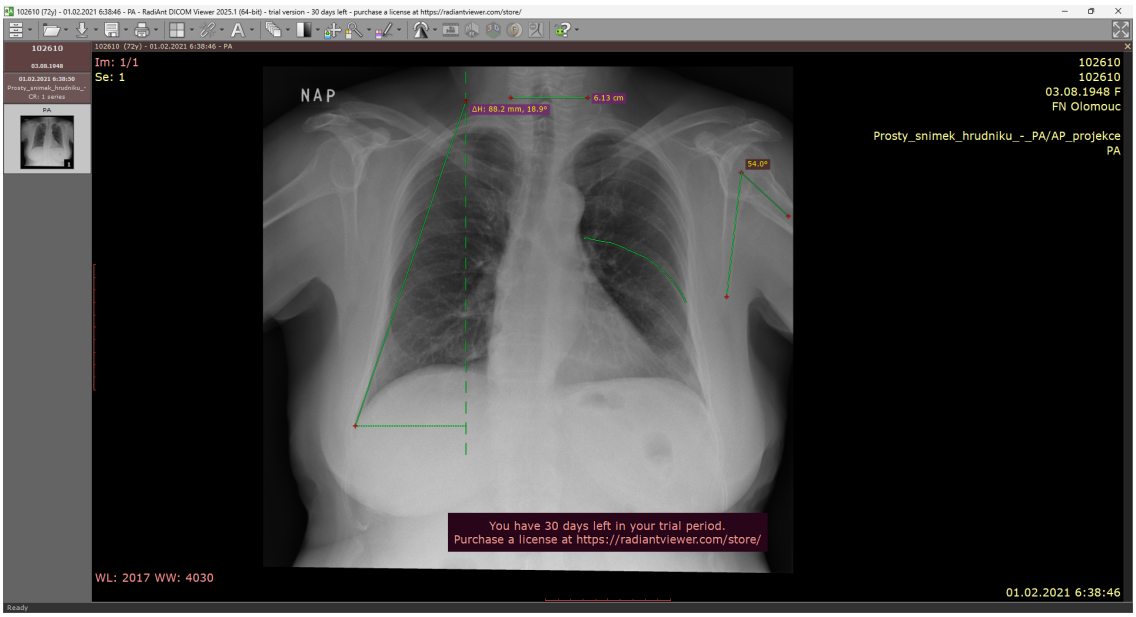


Obrázek 2.2: Ukázka programu Computer Vision Annotation Tool (CVAT)

2.3 RadiAnt DICOM Viewer

Jedná se o komerční program pro prohlížení a jednoduchou anotaci medicínských snímků. Program je možné nainstalovat pouze na operační systém Windows a je dostupný zdarma pouze v 30 denní zkušební verzi. Tento program je určený především pro lékaře a radiology, kteří potřebují rychle a efektivně prohlížet medicínské snímky a provádět jednoduché měření [3]. Je stavěný přímo na práci s DICOM soubory, a kromě anotací obsahuje také obrovskou škálu nástrojů jako pravítka, úhloměry a další. Ovládání programu (obr. 2.3) je zprvu dosti neintuitivní, ale po chvíli se na něj dá zvyknout a rychle se v něm orientovat.

Program tedy splňuje většinu kritérií, které jsem si stanovila, ale obsahuje nedostatky jako je nutnost instalace a chybějící podpora pro další platformy jako například Linux nebo MacOS. Pro práci s medicínskými snímky je však tento nástroj velmi dobrý a jeho možnosti anotace jsou velmi široké a měly by být pro většinu lékařů a radiologů dostačující.



Obrázek 2.3: Ukázka programu RadiAnt DICOM Viewer

Kapitola 3

Návrh aplikace

Tato část práce je věnována návrhu aplikace před započetím jejího vývoje. Je nezbytné si stanovit cíle a požadavky na aplikaci, které by měli být splněny. Dále je potřeba si ujasnit, jakým způsobem budou různí uživatelé s aplikaci pracovat a jaké funkce budou potřebovat. Požadavky jsou rozděleny do několika kategorií podle jejich zaměření a seřazeny podle důležitosti a priority. Součástí návrhu je taktéž výběr vhodných technologií, které budou při vývoji aplikace použity a návrh uživatelského rozhraní. Všechny tyto aspekty jsou důležité pro úspěšný vývoj aplikace a její následné použití.

3.1 Systémové požadavky

Systémové požadavky je možné rozdělit na funkční a nefunkční. Funkční požadavky popisují, jaké konkrétní funkce a schopnosti by měla aplikace mít. Nefunkční požadavky se zaměřují na vlastnosti aplikace, jako je výkon, spolehlivost, použitelnost a udržovatelnost. Některé požadavky vycházejí z analýzy podobných řešení, které byly popsány v předchozí kapitole, jiné byly odvozeny ze záměru aplikace a vcítění se do role uživatele.

3.1.1 Funkčnost

Program vyvíjený v rámci této práce bude sloužit k anotaci medicínských snímků. Je proto nezbytné, aby šlo načítat soubory ve formátu DICOM, který je standardem pro ukládání a přenos medicínských obrazových dat. Dále je potřeba, aby aplikace umožnila práci se samotnými anotacemi a výsledek práce uložit nebo exportovat do formátu, který je jednoduchý pro zpracování ostatními aplikacemi. V neposlední řadě by aplikace měla být multiplatformní a podporovat tak různé operační systémy. Z těchto informací vyplývá následující seznam funkčních požadavků:

- Aplikace musí podporovat načítání z formátu DICOM.
- Aplikace musí umožnit uživateli na snímku označit zájmová místa (zakroužkovat, zvýraznit).

- Aplikace musí umožnit uživateli přidat k označené oblasti další informace jako třeba popis, název, případně další.
- Aplikace musí podporovat platformy Windows, Linux, MacOS a webové prostředí.
- Aplikace musí obsahovat export do formátu JSON.
- Aplikace musí být schopna načíst exportovaná data.
- Aplikace musí podporovat jednovrstvá i vícevrstvá data.
- Aplikace musí umožnit uživateli vhodně manipulovat s obrazem (např. posun, přiblížení).

3.1.2 Vhodnost k použití

Program bude používán s největší pravděpodobností ve zdravotnických zařízeních, kde je potřeba pracovat rychle a především efektivně. Proto je důležité, aby aplikace byla snadno ovladatelná. Počítá se také s tím, že zdravotnická zařízení nemají vždy k dispozici nejnovější počítačové vybavení a kvalitní internet, proto je potřeba, aby se aplikace dokázala přizpůsobit různým velikostem monitorů a aby nebyla závislá na internetovém připojení.

- Aplikace by měla být intuitivní, aby jí dokázal obsluhovat i uživatel bez nutnosti zaškolení.
- Aplikace by měla být schopna se přizpůsobit různým velikostem monitorů.
- K použití aplikace by nemělo být potřeba internetové připojení.

3.1.3 Spolehlivost

Pořídit snímek za pomocí technologií jako je CT nebo MRI je časově náročné a drahé, proto aplikace nesmí za žádných okolností snímek poškodit nebo znehodnotit. Je také důležité, aby aplikace byla stabilní a nespadla v průběhu práce. Pokud by však k pádu přeci jen došlo, měla by být schopna obnovit poslední uloženou verzi rozpracovaného dokumentu.

- Aplikace nesmí zkreslovat data.
- Aplikace musí být stabilní a nesmí padat.
- Aplikace nesmí modifikovat originální soubor.
- Aplikace nesmí v případě chyby poškodit originální soubor.
- Aplikace by měla být schopna v případě pádu aplikace obnovit poslední uloženou verzi rozpracovaného dokumentu.

3.1.4 Výkon

Aplikace bude pracovat se snímky, které mohou být velmi detailní a ve vysokém rozlišení. Aplikace tak musí být schopna pracovat s velkými soubory, a to i na slabších počítačích.

- Aplikace by měla být schopna si poradit s velkými soubory.
- Aplikaci by měly být schopny spustit i slabší počítače.

3.1.5 Udržovatelnost

Z hlediska programátora je důležité, aby šla aplikace snadno rozšiřovat, v tomto případě především o nové formáty souborů a nové typy anotací. Dále je taky podstatné, aby se tyto změny dostaly k uživatelům co nejdříve a nejjednodušeji.

Programy pro anotace jsou používány jak v malých zdravotnických zařízeních, které mnohdy nemají vlastní IT oddělení, tak i ve velkých nemocnicích, kde se nachází velké množství počítačů, což komplikuje údržbu. Proto by bylo vhodné, aby aplikace dokázala fungovat i bez pravidelné údržby ze strany uživatele a sama se aktualizovat.

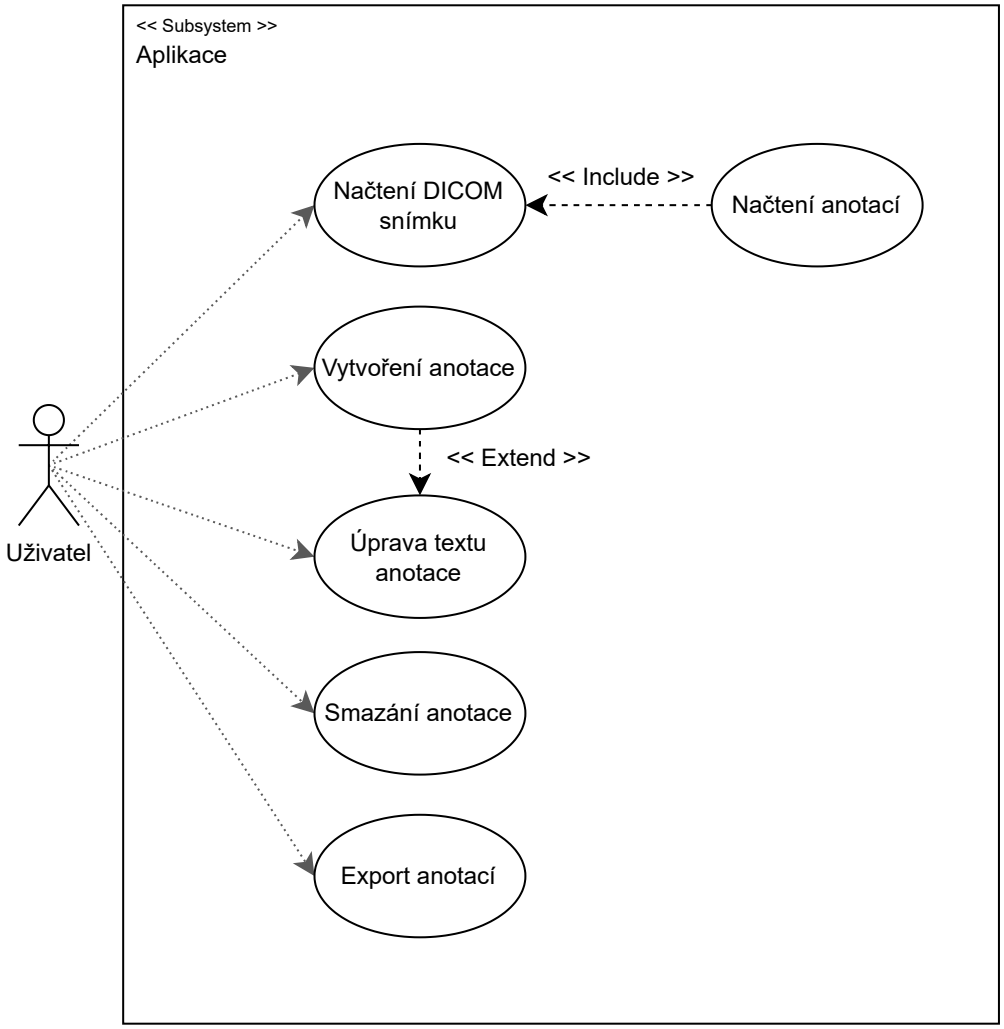
- Aplikace musí být snadno rozšiřitelná o nové formáty souborů.
- Aplikace musí být snadno rozšiřitelná o nové typy anotací.
- Aplikace by měla být schopna fungovat bez pravidelné údržby ze strany uživatele.
- Aplikace by měla být schopna se sama aktualizovat.

3.2 Charakteristika aktérů a prostředí

Jelikož aplikace bude využívána pouze pro práci s lokálními soubory, nebude obsahovat žádný mechanismus pro přihlašování a správu účtů. V aplikaci bude figurovat pouze jeden uživatel, který bude mít přístup k veškerým funkcím aplikace. Tento uživatel bude mít možnost načítat soubory z disku, provádět anotace a exportovat výsledky.

3.3 Use case diagram

Níže uvedený use case diagram na obr. 3.1 ilustruje, jaké akce může uživatel provádět v rámci aplikace. Jednotlivá použití (use case) jsou znázorněna jako ovály, které jsou propojeny s uživatelským aktérem pomocí čar. Uživatel může provádět různé akce, jako je načítání DICOM snímku, vytváření anotací, úprava textu anotace, mazání anotace a export anotací. Každá z těchto akcí je podrobně popsána v následujících sekcích.



Obrázek 3.1: Use case diagram

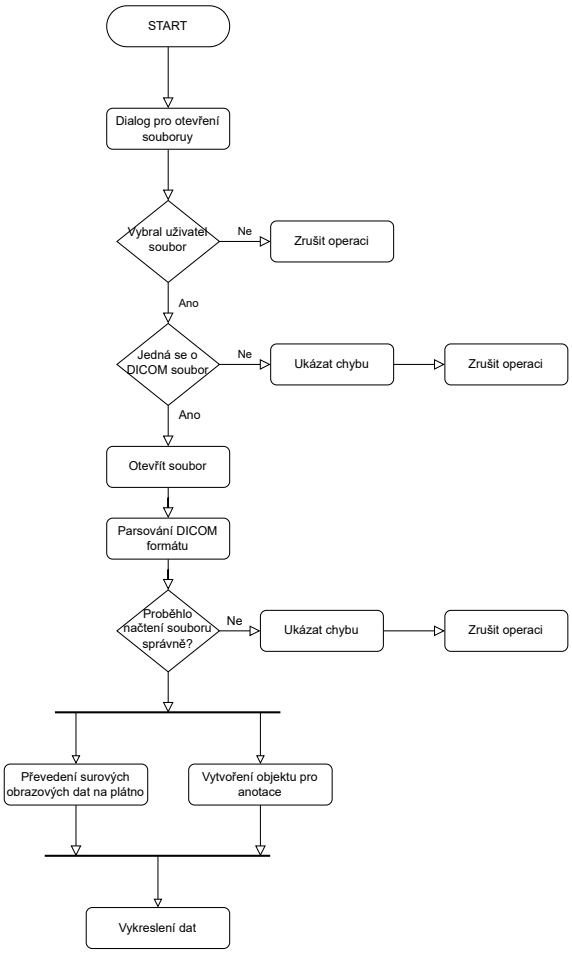
3.3.1 Use Case: Načtení DICOM snímku

Hlavní scénář

Scénář zahájí uživatel kliknutím na tlačítko pro otevření souboru. Tím se otevře dialogové okno, které uživatele vyzve k výběru souboru. Uživatel vybere požadovaný soubor a potvrdí. Program ověří, že se jedná o platný DICOM soubor a převede ho z binárního formátu na jednotlivé DICOM štítky a následně z něj extrahuje surová obrazová data a nutná potřebná metadata. Surová data se převedou do běžného obrazového formátu, který je možné zobrazit v prohlížeči a v paměti se vytvoří

prázdný objekt pro anotace. Nakonec se uživateli vykreslí na obrazovku editor s požadovaným snímkem. Aktivitní diagram tohoto scénáře je vyobrazen na obr. 3.2.

- Uživatel klikne na tlačítko pro otevření souboru.
- Otevře se dialogové okno pro výběr souboru.
- Uživatel vybere požadovaný soubor a potvrdí.
- Soubor se převede z binárního formátu na jednotlivé DICOM štítky.
- Z převedeného souboru se extrahují surová obrazová data a nutná potřebná metadata.
- Surová obrazová data se převedou do běžného obrazového formátu.
- V paměti se vytvoří prázdný objekt pro anotace.
- Data se vykreslí uživateli na obrazovku.



Obrázek 3.2: Aktivitní diagram pro načtení DICOM snímku

3.3.2 Use Case: Načtení existujících anotací

Hlavní scénář

Načítání existujících anotací rozšiřuje načítání DICOM snímku. Uživatel klikne na tlačítko pro otevření souboru a následně se otevře dialogové okno pro výběr souboru. Uživatel vybere soubor s anotacemi a následně potvrdí otevření souboru. Program ověří, že k souboru existuje i příslušný DICOM soubor a následně převede soubor z binárního formátu na jednotlivé DICOM štítky. Poté se uživatel zeptá, zda chce soubor upravovat, nebo vytvořit kopii. Obrazová data ze souboru se převedou do formátu, který je možné zobrazit v prohlížeči. Soubor s anotacemi se načte z JSON souboru a uživateli se vykreslí na obrazovku editor s požadovaným snímkem a existujícími anotacemi.

- Uživatel klikne na tlačítko pro otevření souboru.
- Otevře se dialogové okno pro výběr souboru.
- Uživatel vybere soubor s anotacemi a potvrdí.
- Ověří se, že existuje DICOM soubor, ke kterému anotace patří.
- Soubor se převede z binárního formátu na jednotlivé DICOM štítky.
- Program se zeptá uživatele, zda chce soubor upravovat, nebo vytvořit kopii.
- Obrazová data ze souboru se převedou do formátu, který je možné zobrazit v prohlížeči.
- Soubor s anotacemi se načte z JSON souboru.
- Data se vykreslí uživateli na obrazovku.

3.3.3 Use Case: Vytvoření anotací

Hlavní scénář

Uživatel vybere v menu nástroj pro vytvoření anotace a následně pomocí podržení levého tlačítka a následným tažením myši vybere oblast, kterou chce označit. Jakmile uživatel uvolní tlačítko myši, vytvoří se pro danou oblast nová anotace. Na základě tvaru oblasti se určí typ anotace (otevřená, uzavřená). Z důvodu optimalizace dojde k výpočtu zjednodušeného tvaru anotace. Nakonec se pole pro úpravu textu anotace se stane aktivní, aby uživatel mohl rovnou anotaci upravit.

- Uživatel vybere nástroj pro vytvoření anotace.
- Uživatel s pomocí myši vybere na obrázku oblast.
- Jakmile uživatel uvolní myš, vytvoří se pro danou oblast nová anotace.
- Určí se typ anotace (otevřená, uzavřená) na základě tvaru oblasti.
- Z důvodu optimalizace dojde k výpočtu zjednodušeného tvaru anotace.
- Pole pro úpravu textu anotace se stane aktivní, aby uživatel mohl rovnou anotaci upravit.

3.3.4 Use Case: Úprava textu anotace

Hlavní scénář

Uživatel klikne na anotaci v seznamu anotací, která se následně přepne do editačního režimu a

umožní uživateli upravovat její text a případné další vlastnosti. Uživatel provede jím požadované změny, jejíž náhled se popisuje v živém čase na snímek. V poslední fázi uživatel změnu potvrdí kliknutím na tlačítko nebo stisknutím klávesy Enter.

- Uživatel vybere anotaci v seznamu anotací a klikne na ni.
- Anotace se přepne do editačního režimu a umožní uživateli upravovat její text a případné další vlastnosti.
- Uživatel změnu potvrdí kliknutím na tlačítko nebo stisknutím klávesy Enter.

Alternativní scénář
Uživatel na snímku dvojitě klikne na anotaci, kterou si přeje upravit. Tím se anotace přepne do editačního režimu a umožní uživateli upravovat její text a případné další vlastnosti. Společně s tím se anotace zvýrazní v seznamu anotací a pokud se momentálně nachází mimo obrazovku, posune se do popředí, aby jí uživatel viděl. Uživatel provede jím požadované změny, jejíž náhled se popisuje v živém čase na snímek. V poslední fázi uživatel změnu potvrdí kliknutím na tlačítko nebo stisknutím klávesy Enter.

- Uživatel dvojitě klikne na snímku na požadovanou anotaci.
- Anotace se přepne do editačního režimu a umožní uživateli upravovat její text a případné další vlastnosti.
- V seznamu anotací se vybraná anotace posune do popředí, aby jí uživatel viděl.
- Uživatel změnu potvrdí kliknutím na tlačítko nebo stisknutím klávesy Enter.

Zrušení editace
Kdykoliv v průběhu editace anotace může uživatel změnu odmítnout stisknutím tlačítka nebo klávesou Esc. V takovém případě se změny nahradí původními daty.

- Uživatel při editaci změnu odmítne stisknutím tlačítka nebo klávesou Esc.
- Změny se nahradí původními daty.

3.3.5 Use Case: Smazání anotace

Hlavní scénář
Uživatel klikne na tlačítko smazání u požadované anotace v seznamu anotací. Tím se otevře dialogové okno, které uživatele vyzve k potvrzení akce. Pokud uživatel akci potvrdí, dojde k odstranění anotace.

- Uživatel klikne na tlačítko smazání u požadované anotace v seznamu anotací.
- Vyskočí dialogové okno, které uživatele vyzve k potvrzení akce.
- Uživatel potvrdí smazání anotace.
- Dojde k odstranění anotace.

Alternativní scénář
Uživatel vybere nástroj guma, který slouží k odstranění anotací. Uživatel klikne na jednu nebo více anotací, které chce odstranit. Jakmile uživatel uvolní tlačítko myši, dojde k odstranění vybraných anotací.

- Uživatel vybere nástroj guma.
- Uživatel klikne na jednu nebo více anotací, které chce odstranit.
- Jakmile uživatel uvolní tlačítko myši, dojde k odstranění vybraných anotací.

3.3.6 Use Case: Export anotací

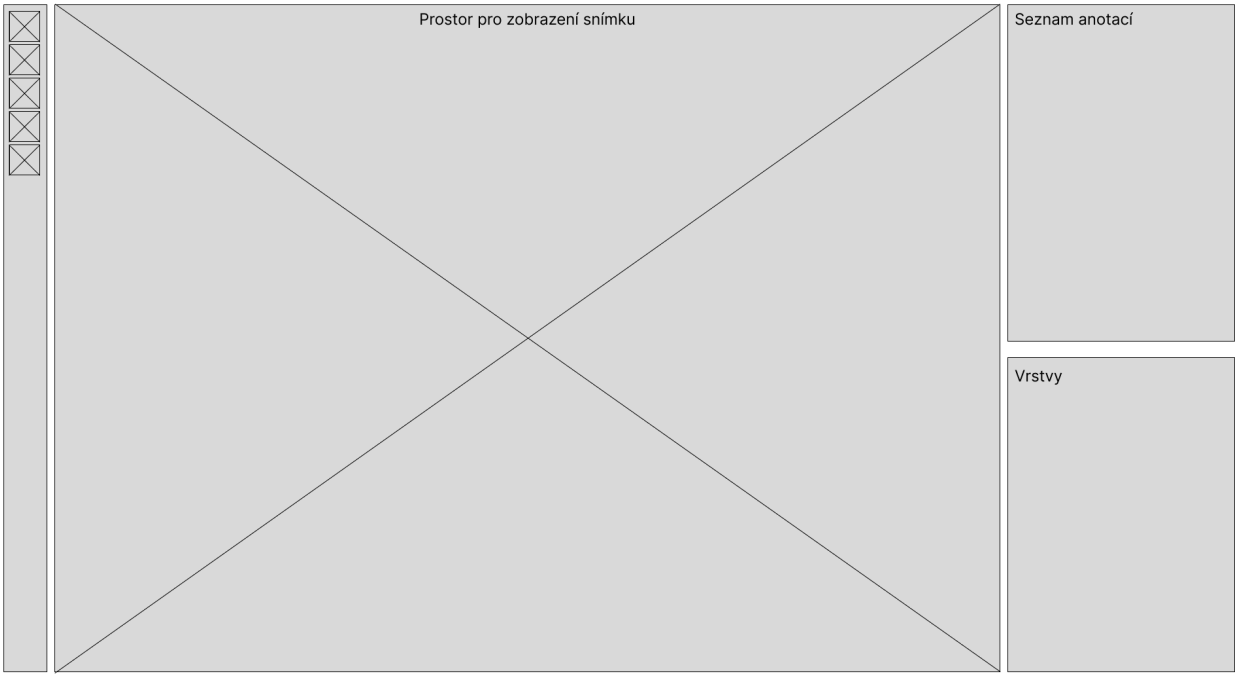
Hlavní scénář
Uživatel zahájí export kliknutím na tlačítko export. Následně program otevře dialogové okno, které uživatele vyzve k vybrání umístění, kam soubor uložit. Ve výchozím nastavení je umístění nastaveno na složku, kde se nachází původní soubor. Uživatel vybere umístění a potvrdí. Následně program otevře dialogové okno, které uživatele vyzve k vybrání formátu exportu (soubor typu JSON / obrázek). Uživatel vybere formát a potvrdí. Nakonec se exportované soubory uloží na disk.

- Uživatel klikne na tlačítko export.
- Program vyzve uživatele k vybrání umístění, kam soubor uložit (výchozí umístění je do složky k původnímu souboru).
- Program vyzve uživatele k vybrání formátu exportu (soubor typu JSON / obrázek).
- Exportované soubory se uloží na disk.

3.4 Návrh uživatelského rozhraní

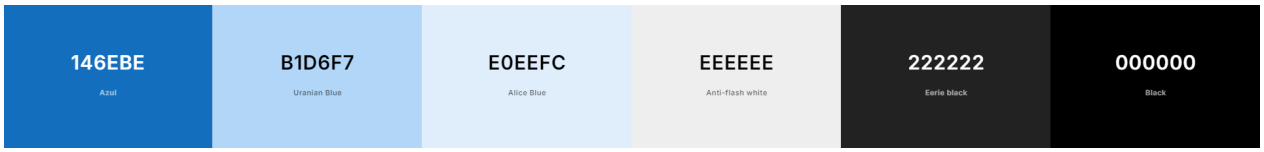
Návrh uživatelského rozhraní je důležitou součástí vývoje aplikace, jelikož ovlivňuje uživatelskou zkušenost a celkovou použitelnost aplikace. Pro návrh uživatelského rozhraní jsem v rámci této práce použila nástroj Figma, který umožňuje snadno vytvářet návrhy a prototypy uživatelského rozhraní. Tento nástroj je velmi užitečný pro rychlé iterace a testování různých návrhů, což je v procesu návrhu velmi důležité.

Prvním krokem při návrhu uživatelského rozhraní bylo vytvoření drátového modelu (wireframe) aplikace. Tento model představuje základní strukturu aplikace a slouží jako výchozí bod pro další návrh. Při návrhu jsem kladla důraz primárně na co největší jednoduchost a na intuitivní ovládání. Rozhodla jsem se inspirovat rozložením většiny grafických programů na práci s obrázky. Na levou stranu jsem umístila vertikální lištu nástrojů, které budou reprezentovány ikonami. Panel se seznamem anotací a panel pro přepínaní vrstev jsem se rozhodla umístit vpravo. Uprostřed se pak nachází plocha se samotným snímkem. Tento prvotní návrh je zobrazen na obrázku 3.3.



Obrázek 3.3: Prvotní návrh rozložení uživatelského rozhraní

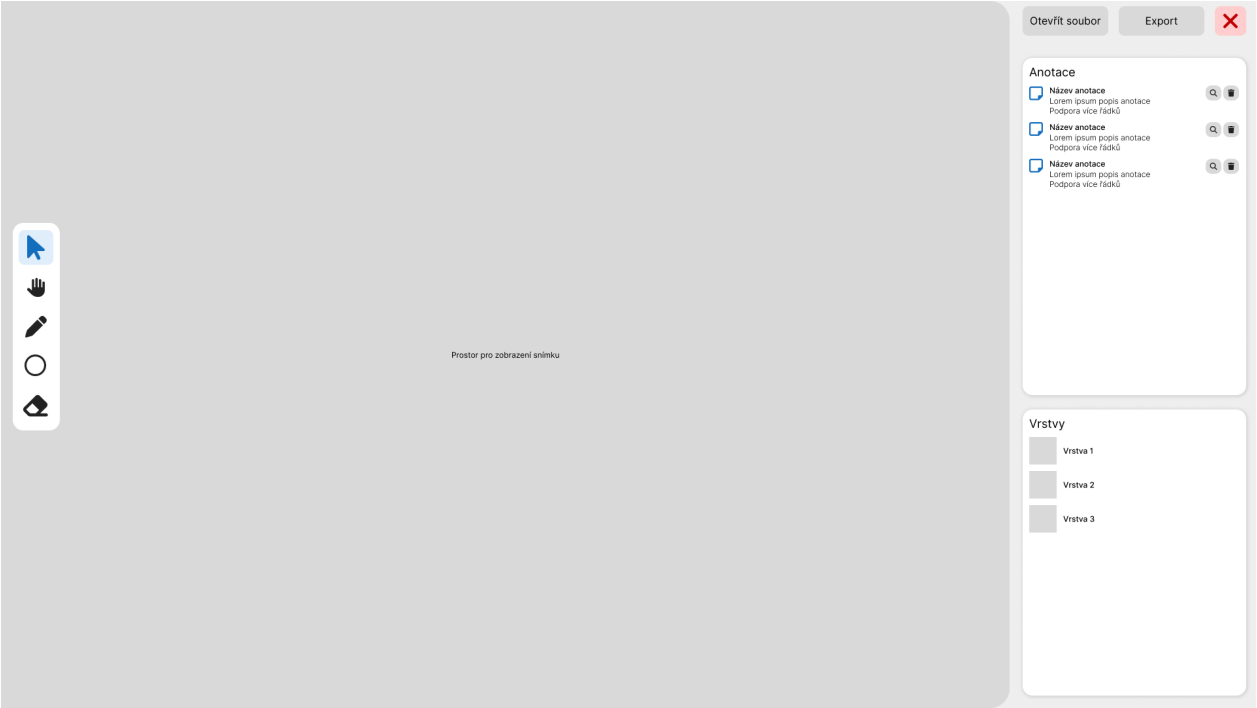
Ze začátku jsem uvažovala o použití červeno-bílé barevné palety, ale ve výsledku jsem se rozhodla vyměnit červenou barvu za modrou, jelikož působí více profesionálně a více se hodí k nemocničnímu prostředí. Modrá barva je taktéž spojena s klidem, důvěrou a produktivitou. Výsledná barevná paleta je zobrazena na obrázku 3.4.



Obrázek 3.4: Barevná paleta uživatelského rozhraní

Po několika iteracích návrhu uživatelského rozhraní jsem se nakonec dostala k finálnímu vzhledu, se kterým jsem byla spokojená (obrázek 3.5). Oproti původnímu návrhu jsem vertikálně zmenšila prostor pro lištu nástrojů, jelikož nástrojů bude ve výsledné aplikaci pouze pět, a to nástroj pro interakci, pohyb, kreslení anotací, tvary a guma.

Vpravo nahoře se nachází akce pro otevření souboru, export anotací a zavření aktuálně otevřeného souboru. Pod těmito akcemi se nachází seznam anotací, přičemž každá anotace má název, popis, tlačítko na smazání a tlačítko které anotaci najde na snímku. Pod anotacemi se nachází seznam vrstev, pomocí kterého je možné jednotlivé vrstvy přepínat.



Obrázek 3.5: Finální návrh uživatelského rozhraní

Kapitola 4

Výběr vhodných technologií

Pro tento projekt byly vybrány technologie Tauri, TypeScript, Svelte a fomrát DICOM. Bylo potřeba zvolit vhodný framework pro tvorbu multiplatformních aplikací, přičemž jsem volila mezi Tauri a Electron. I přesto, že jsou oba tyto frameworky postaveny na podobném principu, mají mezi sebou zásadní rozdíly.

4.1 Srovnání technologií Electron a Tauri

Electron, stejně jako Tauri, slouží k vývoji multiplatformních aplikací za pomocí primárně webových technologií. Obě tyto technologie umožňují vývojářům vytvářet aplikace, které běží na různých operačních systémech, aniž by bylo nutné psát nativní kód pro každou platformu zvlášť.

Electron byl představen v roce 2013 a od té doby se stal jedním z nejpopulárnějších frameworků pro vývoj desktopových aplikací. Jeho hlavní výhodou je jednoduchost a široká podpora díky velké komunitě. Existuje kolem něj rozsáhlý ekosystém doplňků, návodů a je velmi dobře zdokumentovaný. Jedná se o svobodný software, který je ovšem vyvíjen velkou organizací (GitHub), takže je možné očekávat i další budoucí vývoj, stabilitu a důkladné testování. Je využíván velkou spoustou známých aplikací jako například Discord, Dropbox, GitHub Desktop, Slack, Skype, VS Code nebo třeba Microsoft Teams [4, 5]. Naopak jeho zásadní nevýhodou jsou vysoké systémové nároky. To je způsobeno tím, že Electron aplikace obsahují zabudovaný prohlížeč Chromium, což zvyšuje velikost aplikace a nároky na systémové prostředky. To může být problém zejména pro uživatele s nižším výkonem počítače nebo pro mobilní zařízení.

Tauri je modernější alternativa, která byla představena v roce 2022 a zaměřuje se především na minimalizaci velikosti aplikací a efektivní využití systémových zdrojů. Toho se snaží dosáhnout tím, že aplikace neobsahují přibalený vlastní prohlížeč, ale využívají nativní prohlížeč, který je již zabudovaný v operačním systému uživatele. Dále umožňuje psaní nízko úrovňového kódu v jazyce Rust, což umožňuje vykonávat složitější operace efektivněji [6]. Nevýhodou této technologie je ovšem

menší podpora a komunita, což vede k menšímu počtu dostupných knihoven, návodů a vést k potřebě psát více vlastního kódu.

V rámci rozhodování jsem vytvořila jednoduchou aplikaci v obou těchto technologiích, abych si mohla vyzkoušet, jak se s nimi pracuje. Zjistila jsem, že Electron je mnohem lépe zdokumentovaný a má více dostupných knihoven, což usnadňuje vývoj. Na druhou stranu Tauri je mnohem jednodušší pro prvotní nastavení, a i přes to, že psaní samotného kódu mi komplikovala potřeba pracovat s mentálním modelem ve dvou různých jazycích naráz (JavaScript a Rust), tak jsem se nakonec rozhodla pro Tauri. Důvodem byla především možnost využít nízko úrovňového jazyka pro složitější výpočetní operace, což působilo jako výhoda, kterou je možné využít při zpracování obrazových dat v rámci tohoto projektu.

4.2 Tauri

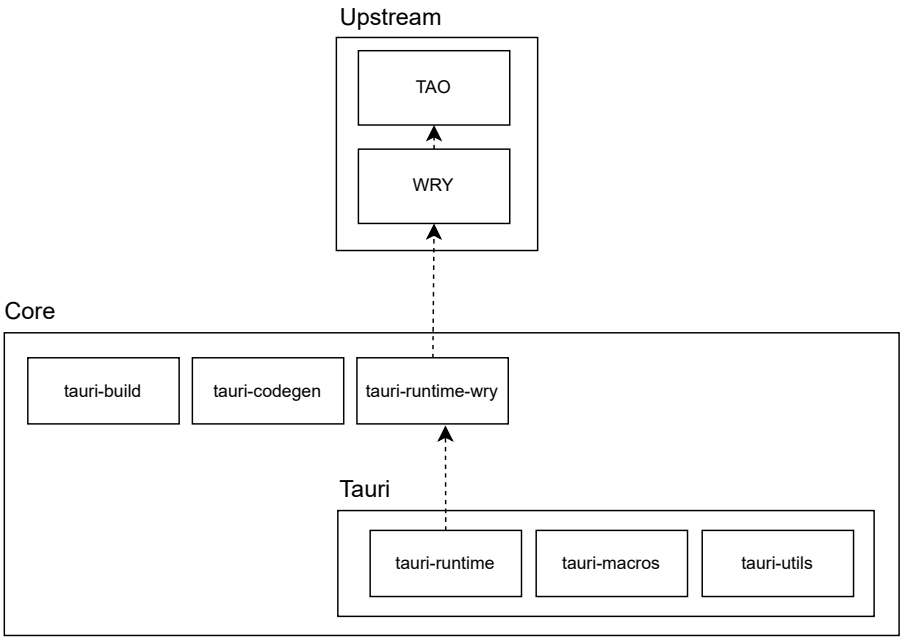
Jak již bylo zmíněno, Tauri je framework pro vytváření multiplatformních aplikací pro platformy Windows, Linux, MacOS, Android a iOS za pomocí webových technologií. I přes to, že se jedná o relativně nový framework, získal podporu od několika velkých technologických firem, mezi které patří mimo jiné Amazon, 1Password a Cloudflare [6].

Výhoda multiplatformních aplikací spočívá v tom, že je možné napsat kód pouze jednou a ten pak spustit na různých operačních systémech bez nutnosti provádět jakékoliv úpravy. To šetří čas a náklady na vývoj, jelikož není potřeba psát a udržovat kód pro každou platformu zvlášť.

Tauri se snaží multiplatformního chování dosáhnout tím, že rozdělí aplikaci na dvě části. První částí je jádro napsané v jazyce Rust, které zajišťuje komunikaci mezi uživatelským rozhraním a operačním systémem, umožňuje přístup k nativním funkcím a jako je například souborový systém, notifikace nebo přístup k hardwaru. Taktéž umožňuje přidávat vlastní funkce napsané v jazyce Rust, které lze volat z uživatelského rozhraní.

Druhou částí je uživatelské rozhraní založené na webových technologiích, jako HTML, CSS a JavaScript, které se zobrazuje v okně zabudovaného prohlížeče, který je součástí operačního systému. Komunikace mezi těmito částmi je zajištěná s pomocí knihoven frameworku, které tuto komunikaci zapouzdřují a zjednodušují.

Tento přístup umožňuje vývojářům snadno vytvářet aplikace, které mají jednotný vzhled a chování napříč různými systémy, aniž by museli psát nativní kód pro každou platformu zvlášť. Tauri také podporuje různé webové frameworky, jako jsou React, Vue nebo Svelte, což usnadňuje integraci s existujícími projekty.



Obrázek 4.1: Zjednodušená architektura Tauri frameworku [7]

Na výše uvedeném obrázku 4.1 je vyobrazená zjednodušená architektura Tauri frameworku, který se skládá z jádra (Core), které se stará celkově o generování aplikace a její běh a z „Upstream" části, která se stará o vykreslení aplikace a komunikaci s uživatelským rozhraním. Obě tyto části jsou rozděleny do více menších balíčků, které se starají o jednotlivé úkoly. Mezi tyto balíčky patří:

TAO Vytváří jednotné rozhraní pro práci s okny a jejich událostmi napříč různými platformami.

WRY Vytváří jednotnou abstrakční vrstvu nad WebWiew různých operačních systémů. Vybírá nejvhodnější implementaci pro aktuální platformu a stará se o její inicializaci, správu a interakce s uživatelem.

tauri-runtime Obsahuje samotné běhové prostředí Tauri, které je zodpovědné za chod celé aplikace a komunikaci jednotlivých částí.

tauri-build Při sestavení aplikace nahrazuje a aplikuje makra, která se starají o kompatibilitu s různými platformami.

tauri-codegen Stará se o generování potřebného kódu pro běh aplikace, kompresi jednotlivých zdrojů (včetně ikon) a zpracovává konfigurační soubor Tauri aplikace.

tauri-runtime-wry Zodpovídá za správu oken, vykreslování aplikace a komunikaci s WRY.

tauri-macros Spolupracuje s *tauri-codegen* a vkládá do něj makra.

tauri-utils Obsahuje části kódu, které Tauri používá opakovaně napříč různými balíčky. Mimo jiné zodpovídá za zpracování konfiguračních souborů, detekování platformy a správu zabezpečení aplikace.

4.3 Výběr frameworku pro uživatelské rozhraní

Tauri umožňuje pro vývoj uživatelského rozhraní použít libovolný webový front-end framework. Webový framework je sada nástrojů a knihoven, které poskytují vývojářům vyšší úroveň abstrakce nad HTML, JavaScript a CSS kódem. Typicky obsahují šablonovací jazyk, který umožňuje používat proměnné a funkce přímo v pseudo-HTML kódu, což usnadňuje vytváření dynamických uživatelských rozhraní a funkce pro správu vnitřního stavu aplikace, což je důležité pro reaktivní aplikace, které reagují na uživatelské akce a mění svůj vzhled a chování v závislosti na těchto akcích.

Existuje celá řada různých frameworků, které se liší svojí architekturou, syntaxi a funkcemi. Mezi nejznámější, ale zároveň nejstarší patří React, Vue a Angular. Za posledná desetiletí se však objevilo velké množství nových frameworků, které se snaží zjednodušit vývoj a odstranit některé nedostatky těchto starších frameworků. Mezi tyto nové frameworky patří například Svelte, Solid, Preact, Qwik, Stencil, Lit a mnoho dalších. Způsobů, jakým lze v dnešní době vytvořit webovou aplikaci je tedy mnoho a výběr toho správného frameworku se může zdát jako velmi obtížný úkol.

Při výběru jsem se snažila zohlednit především svoji vlastní zkušenost s daným frameworkem, jeho popularitu a podporu komunity, dostupnost knihoven a návodů a také jeho výkon a velikost výsledné aplikace. Výběr byl nakonec zúžen na dva frameworky – React a Svelte. Oba tyto frameworky stojí na principu komponent. Komponenta je malá část uživatelského rozhraní, která má vlastní logiku a vzhled. Komponenty lze kombinovat a vytvářet tak složitější uživatelská rozhraní. Tím však jejich podobnost končí.

React definuje každou komponentu jako funkci, která vrací svoji šablonu v jazyce JSX. Správa stavu komponenty je zajištěna pomocí takzvaných „hooks", což jsou speciální funkce, které umožňují komponentě reagovat na změny v datech a aktualizovat svůj vzhled. Dále React používá virtuální DOM, což je zjednodušená reprezentace skutečného DOM v paměti. Změny se provádějí nejprve v tomto virtuálním DOM a poté se synchronizují se skutečným DOM a tak minimalizuje počet operací nad skutečným DOM, které jsou časově více náročné [5].

Svelte naopak definuje komponentu jako jeden soubor, který se skládá ze tří částí - šablony, stylů a logiky. Pro šablony využívá svůj vlastní šablonovací jazyk, který vychází z HTML5 a rozšiřuje jej o další pokročilou syntaxi, jako například podmínky, cykly a použití komponent. Správa stavu komponenty je zajištěna pomocí běžných proměnných, u kterých Svelte automaticky sleduje změny a zajišťuje dynamické chování zbytku aplikace. Na rozdíl od Reactu však Svelte nevyužívá virtuální DOM, ale místo toho v čase kompilace generuje optimalizovaný kód, který přímo manipuluje se skutečným DOM [8].

Na základě svých zkušeností, ale také osobních preferencí jsem se rozhodla v tomto projektu využít Svelte. Důvodem byla především jeho jednoduchost a přehlednost kódu.

4.4 TypeScript

Při vývoji aplikace ve frameworku Svelte je možné využít programovacího jazyka JavaScript, nebo jazyka TypeScript. JavaScript je skriptovací jazyk, který byl poprvé představen v roce 1995 jakožto jazyk pro skriptování webových stránek. V současné době je jedním z nejpoužívanějších jazyků na světě a je podporován většinou moderních webových prohlížečů a je možné jej používat i na serveru.

TypeScript je nadmnožinou jazyka JavaScript, což znamená, že každý platný kód napsaný v jazyce JavaScript je zároveň platným kódem napsaným v jazyce TypeScript, do kterého se také překládá. Byl vydán v roce 2012 s cílem odstranit některé nedostatky jazyka JavaScript a přidat do něj další funkce, které usnadňují vývoj větších a složitějších aplikací [9]. Mezi tyto funkce patří především podpora pro datové typy, díky kterým je možné lépe definovat strukturu dat a odhalit tak chyby již při překladu kódu, a ne až při jeho spuštění. Jeho nevýhodou je nutnost překladu kódu, složitější syntaxe a výsledný kód je o něco delší ve srovnání s jazykem JavaScript.

Při volbě mezi těmito dvěma jazyky jsem se rozhodla pro TypeScript. Volba byla založena především jeho schopností odhalit chyby již při překladu a větší přehlednosti kódu.

4.5 DICOM formát

DICOM (Digital Imaging and Communications in Medicine) je standard, který definuje jednotný formát pro uchování medicínských snímků z radiologie, CT, MRI či ultrazvuku a souvisejících metadat k těmto snímkům. Poprvé byl vytvořen roku 1993 a kromě samotného formátu souborů taktéž definuje jakým způsobem se data mají přenášet po síti či offline a jak se mají zobrazovat na výsledném zařízení.

Jedná se o binární formát, který se skládá ze štítků (tag). Každý štítek je identifikován dvěma čísly – skupinou a elementem. Obě tato čísla jsou kladná a zabírají 2 bajty. Následuje datový typ, délka a samotná data, která kromě obrázku obsahují také nejrůznější informace o pacientovi, aby nemohlo dojít k záměně souborů. Každý DICOM soubor může mít pouze jeden štítek obsahující data obrázku, nicméně tento štítek může pojmout několik obrázků zároveň [10, 11].

Kapitola 5

Implementace aplikace

Implementace je proces, při kterém se návrh aplikace převádí do funkčního kódu. Tento proces zahrnuje psaní kódu, testování a ladění aplikace. V této kapitole jsou rozepsány některé důležité kroky při implementaci, stejně tak podproblémy, které bylo třeba vyřešit a použité technologické postupy.

5.1 Založení projektu

Implementace jakékoliv aplikace začíná založením projektu. Postup se liší na základě zvoleného vývojového prostředí, programovacího jazyka a dalších použitých technologií. V tomto kroku dojde k vytvoření základní struktury projektu, která obsahuje všechny potřebné soubory, složky a nastavení pro správné sestavení aplikace a její spuštění. Při použití nestandardní kombinace technologií se může jednat o složitý a časově náročný proces, to naštěstí není případ této aplikace, jelikož Tauri poskytuje nástroj na vytvoření kostry projektu, který je kompatibilní s ostatními technologiemi zvolenými pro vývoj této aplikace, přesněji Svelte a TypeScript.

Před založením Tauri projektu je ovšem do počítače potřeba nainstalovat překladač jazyka Rust[1] a program NodeJS[2], který slouží jako běhové prostředí pro JavaScript a TypeScript. Po instalaci tohoto nezbytného programového vybavení je možné založit samotný projekt. To je možné vytvořením nové složky pro daný projekt a následným spuštěním instalačního příkazu Tauri.

```
npm create tauri-app@latest
```

Tento příkaz otevře v příkazové řádce průvodce vytvořením projektu, který se programátora zeptá na několik otázek ohledně projektu, jako název, identifikátor, jaký jazyk má být použitý na front-endu, jaký použít správce balíčků a jaký front-end framework. Nakonec, po zodpovězení všech otázek, dojde k inicializaci a ve složce se vytvoří základní struktura projektu. Následně je potřeba

[1] https://tauri.app/
[2] https://nodejs.org/en

ještě použít příkaz `npm install` k doinstalování potřebných JavaScript knihoven. Projekt je tímto založen a připraven.

5.2 Načítání DICOM souborů

Mojí původní myšlenkou bylo vyřešit načítání a parsování DICOM souborů v jazyce Rust a následně načtený obrázek poslat na front-end z důvodu rychlosti a efektivity. Od toho postupu jsem však upustila, jelikož aplikace musí být schopná běžet ve webovém prohlížeči a v takovém případě není dostupná Rust část Tauri aplikace. Celé načítání souborů je tedy řešeno na front-endu za pomocí jazyka TypeScript.

Po nastudování, jak se pracuje s DICOM soubory a jaké jsou dostupné knihovny na parsování, jsem zvolila knihovnu Cornerstone.js dicomParser[3] neboť je nenáročná a podporuje načítání ze streamů. Jedinou nevýhodou této knihovny je neintuitivní přístup k jednotlivým hodnotám v souboru, jelikož pro přístup k hodnotě je třeba znát její datový typ a zapsat identifikátor štítku v hexadecimální podobě (ukázka kódu 5.1).

[3] https://github.com/cornerstonejs/dicomParser

```
import * as dicomParser from 'dicom-parser';

// Štítky dicom souborů zapsány v enum pro větší přehlednost
enum DicomTag {
  IMAGE_WIDTH = 'x00280011',
  IMAGE_HEIGHT = 'x00280010',
  IMAGE_DEPTH = 'x00280100',
  PIXEL_DATA = 'x7fe00010',
}

// Následující funkce bere na vstupu soubor a vrací objekt s informacemi potřebnými k vykreslení obrázku
async function loadDicomFile(file: File): Promise<DicomImage> {
  // Načtení obsahu souboru do bufferu a následně do pole bajtů
  const arrayBuffer = await file.arrayBuffer();
  const byteArray = new Uint8Array(arrayBuffer);

  // Použití knihovny dicom-parser pro zpracování DICOM souboru
  const dataSet = dicomParser.parseDicom(byteArray);

  // Získání informací ze souboru
  const imageWidth = dataSet.uint16(DicomTag.IMAGE_WIDTH); // Šířka obrázku
  const imageHeight = dataSet.uint16(DicomTag.IMAGE_HEIGHT); // Výška obrázku
  const imageDepth = dataSet.uint16(DicomTag.IMAGE_DEPTH); // Bitová hloubka obrázku

  // Získání odkazu na obrazová data
  const pixelDataElement = dataSet.elements[DicomTag.PIXEL_DATA];

  // Převední obrazových dat na pole pixelů
  const pixelData = new Uint16Array(dataSet.byteArray.buffer, pixelDataElement.dataOffset,
      pixelDataElement.length / 2);

  // Sestrojení objektu s informacemi o obrázku
  const dicomImage = {
    width: imageWidth ?? 0,
    height: imageHeight ?? 0,
    depth: imageDepth ?? 16,
    pixelData: pixelData,
    fileName: file.name,
  };

  return dicomImage;
}
```

Ukázka kódu 5.1: Načítání DICOM souboru

Po načtení souboru je ještě potřeba převést obrazová data do formátu, který je možné vykreslit v prohlížeči. Toho jsem dosáhla transformací prostřednictvím vytvoření plátna (HTML canvas elementu), vykreslením jednotlivých pixelů na plátno a následným převedením elementu na PNG s pomocí zabudované funkce prohlížeče. Tento postup šetří výkon procesoru, jelikož mají prohlížeče zabudovanou podporu pro PNG obrázky a pro jejich vykreslení použije grafickou kartu místo hlavního vlákna aplikace. Níže uvedený kód 5.2 ukazuje, jak probíhá normalizace obrazových dat a jejich následné vykreslení na canvas.

```
1  export function renderDicomToCanvas(ctx: CanvasRenderingContext2D,dicom: DicomImage, invert:
       boolean = false, depthMinMax: { min: number, max: number} | undefined = undefined) {
2    const imageData = ctx.createImageData(ctx.canvas.width, ctx.canvas.height);
3    const data = imageData.data;
4
5    const minMaxValues = depthMinMax ?? dicomGetMinMax(dicom);
6
7    for (let i = 0; i < dicom.pixelData.length; i++) {
8      const j = i * 4;
9
10     // Convert original pixel depth to 8 bit grayscale
11     const converted = changeDepth(dicom.pixelData[i] - minMaxValues.min, minMaxValues.max -
           minMaxValues.min, 255);
12
13     if (invert) {
14       data[j + 0] = 255 - converted; // R
15       data[j + 1] = 255 - converted; // G
16       data[j + 2] = 255 - converted; // B
17       data[j + 3] = 255; // A
18       continue;
19     }
20
21     data[j + 0] = converted; // R
22     data[j + 1] = converted; // G
23     data[j + 2] = converted; // B
24     data[j + 3] = 255; // A
25   }
26   ctx.putImageData(imageData, 0, 0);
27 }
```

Ukázka kódu 5.2: Vykreselní DICOM dat

5.3 Vykreslování snímku a anotací

Většina UI aplikace je vytvořena za pomocí kombinace HTML a CSS a o samotné vykreslování se tak stará prohlížeč automaticky. Jedinou výjimkou je plocha pro samotný snímek a jeho anotace,

která je vykreslována na plátno za pomocí Malířova algoritmu. Při vykreslování se používají tři typy souřadnicových systémů:

- **Souřadnice na snímku** - Reprezentují pozici bodu na snímku v pixelech. Začínají v levém horním roku a jsou v nich uloženy souřadnice anotací.

- **Souřadnice na plátně** - Jedná se o desetinné číslo v intervalu $\langle 0;1 \rangle$, které určuje pozici bodu na plátně. S tímto prostorem pracují jednotlivé nástroje, mezi které patří kreslení anotací, pohyb a přiblížení. Při transformaci z prostoru obrázku na tento prostor první vydělíme šířku a výšku obrázku šířkou a výškou plátna, poté vynásobíme přiblížením plátna, a nakonec přičteme posunutí obrázku na plátně.

- **Souřadnice na obrazovce** - Souřadnice, které se používají pro finální vykreslení na obrazovku. Transformace z prostoru plátna zahrnuje vynásobení souřadnice šířkou a výškou plátna a přenásobení přiblížením.

Prvním krokem je vyčištění plátna a vykreslení obrázku. Tento krok je jednoduchý, jelikož je DICOM snímek v této fázi již připraven z načítání do formátu, který můžeme předat standardní funkci `ctx.drawImage` pro vykreslení na plátno. Jediné, co je tak třeba řešit je provedení transformace levé horní a pravé dolní souřadnice obrázku do prostoru obrazovky.

Následuje vykreslení anotací. Každá anotace je reprezentována uspořádaným seznamem bodů. Před vykreslením se z důvodů optimalizace první zkontroluje, zda není anotace mimo obrazovku a v takovém případě je přeskočena. Samotné vykreslení probíhá na hlavním vlákně aplikace za pomocí `ctx.beginPath`, což není nejefektivnější způsob, nicméně je dostatečný pro potřeby této aplikace.

Posledním krokem je vykreslení překrytí nástrojů. Do této vrstvy patří například nástroj pro vytváření anotací, který vykresluje aktuální anotaci v průběhu kreslení.

5.4 Multiplatformní přístup k souborovému systému

Přístup k souborům je v anotační aplikaci nezbytný pro načítání a ukládání DICOM souborů, export anotací a ukládání konfigurace. Tauri naštěstí poskytuje velmi jednoduché rozhraní pro práci se soubory. Není tak problém přistupovat k souborům na disku přímo z TypeScript části aplikace. Problém však nastává ve chvíli, kdy je aplikace otevřená v prostředí prohlížeče. V takovém případě není toto rozhraní dostupné a je nutné použít jiný způsob pro práci se soubory. K řešení tohoto problému jsem využila návrhový vzor Adaptér, který umožňuje vytvořit rozhraní, které je nezávislé na konkrétní implementaci. V tomto konkrétním případě je tedy možné vytvořit rozhraní pro práci se soubory, které bude implementováno pomocí Tauri v desktopové verzi a pomocí HTML5 File API v prohlížeči. Program po spuštění zjistí, ve kterém se nachází prostředí a podle toho zvolí kterou implementaci adaptéru použije (zdrojový kód 5.3).

```
import { FileSystemAdapterBrowser } from "./file-system/fs-adapter-browser";
import { FileSystemAdapterDesktop } from "./file-system/fs-adapter-desktop";

export enum AppRuntime {
  Browser = "browser",
  Desktop = "desktop",
};

export class Adapters {
  static get runtime (): AppRuntime {
    // @ts-ignore
    const isTauri = window.__TAURI__ !== undefined;
    return isTauri ? AppRuntime.Desktop : AppRuntime.Browser;
  }

  static get fileSystem () {
    return {
      [AppRuntime.Browser]: FileSystemAdapterBrowser,
      [AppRuntime.Desktop]: FileSystemAdapterDesktop,
    }[Adapters.runtime];

    throw new Error('File system adapter not implemented for runtime');
  }
};
```

Ukázka kódu 5.3: Detekce prostředí a načtení adaptérů

Níže uvedený kód 5.4 obsahuje obecnou definici rozhraní adaptéru pro práci se soubory. Součástí rozhraní je metoda `save` pro zápis textových souborů na disk a metoda `openFileDialog` pro výběr a načtení souborů z disku.

```
export type FileData = {
  byteArray: Uint8Array;
  fileName: string;
};

export type FileSystemAdapter = {
  save: (data: string, path: string) => Promise<void>;
  openFileDialog: () => Promise<FileData | null>;
};
```

Ukázka kódu 5.4: Rozhraní adaptéru souborového systému

Implementace tohoto adaptéru pro desktop verzi aplikace (kód 5.5) využívá pro práci se soubory rozhraní Tauri, které umožňuje aplikaci přímý přístup k souborům na disku a práci s nimi. Imple-

mentace pro prohlížeč (kód 5.6) naopak využívá formulářových HTML elementů, jelikož prohlížeče z bezpečnostních důvodů přímý přístup k souborům neumožnují.

```
import { fs } from "@tauri-apps/api";
import { open } from "@tauri-apps/api/dialog";
import type { FileSystemAdapter } from "./fs-adapter";

export const FileSystemAdapterDesktop: FileSystemAdapter = {
  save: async (data: string, path: string) => {
    await fs.writeTextFile(path, data);
  },

  openFileDialog: async () => {
    // Otevření dialogu pro výběr souboru
    const dialogResponse = await open({
      multiple: false,
      title: 'Otevřít soubor k anotaci',
    });

    if (!dialogResponse) {
      return null;
    }

    // Načtení souboru jako pole bytů
    const selectedFile = Array.isArray(dialogResponse) ? dialogResponse[0] : dialogResponse;
    const byteArray = await fs.readBinaryFile(selectedFile);

    return {
      byteArray,
      fileName: selectedFile,
    }
  },
};
```

Ukázka kódu 5.5: Implementace adaptéru souborového systému pro desktopovou verzi aplikace

```typescript
import type { FileSystemAdapter } from "./fs-adapter";

export const FileSystemAdapterBrowser: FileSystemAdapter = {
  save: async (data: string, path: string, mimeType = 'application/json') => {
    // Vytvoření blob objektu a jeho URL pro stažení
    const blob = new Blob([data], { type: mimeType });
    const url = URL.createObjectURL(blob);
    const fileName = path.split('/').pop() || 'file.json';

    // Vytvoření elementu odkazu pro stažení
    const a = document.createElement('a');
    a.href = url;
    a.download = fileName;
    document.body.appendChild(a);

    // Kliknutí na odkaz pro stažení
    a.click();

    // Úklid
    URL.revokeObjectURL(url);
    document.body.removeChild(a);
  },

  openFileDialog: () => {
    return new Promise((resolve) => {
      // Vytvoření input elementu pro výběr souboru
      const input = document.createElement('input');
      input.type = 'file';

      input.oninput = async () => {
        // Po výběru souboru
        if (!input.files) {
          resolve(null);
          return;
        }

        // Načtení souboru jako pole bytů
        const file = input.files[0];
        const arrayBuffer = await file.arrayBuffer();
        const byteArray = new Uint8Array(arrayBuffer);
        const fileName = file.name;

        // Vrácení výsledku
        resolve({
          byteArray,
          fileName,
        });
        return;
      };

      input.onblur = () => {
        resolve(null);
      };

      // Otevření dialogu pro výběr souboru
      input.click();
    });
  }
};
```

Ukázka kódu 5.6: Implementace adaptéru souborového systému pro web verzi aplikace

5.5 Výstupní soubor

Anotace, které uživatel v programu provádí je potřeba někam uložit. Exportovaná data by měla být čitelná pro ostatní programy a zároveň by měla být snadno rozšiřitelná o další typy anotací.

Z tohoto důvodu jsem zvolila formát JSON, který je široce podporován a je možné jej snadno číst i zapisovat. Nevýhodou tohoto formátu ovšem je, že není vhodný pro zápis obrazových dat, proto je potřeba je první převést do textové podoby (například Base64) a až poté je možné je uložit do JSON souboru. Převod velkého objemu dat do textové podoby může být však časově náročný a neefektivní, proto jsem od ukládání všech obrazových dat do exportovaného souboru upustila a místo toho se do JSON souboru ukládají pouze metadata a reference na soubor s obrazovými daty, ke kterému se anotace vztahují. Tyto dva soubory jsou pak společně uloženy ve stejné složce, aby k nim bylo možné snadno přistupovat. (Sidecar files)

Vnitřní struktura JSON souboru obsahuje atribut `sourceFile` s názvem DICOM souboru, ke kterému se anotace vztahují. Dále se v něm nachází pole `annotations` ve kterém jsou uloženy jednotlivé anotace. Každá anotace se skládá z názvu (`label`), popisu (`description`), typu (`type`), čísla vrstvy (`frame`) a pole `points` uchovávajícího jednotlivé body anotace. Každý bod je složen z `x` a `y` souřadnice, které odpovídají pozici bodu v obrázku.

```
{
  "sourceFile": "142704_4012344_1.2.840.113564.10.1.283943762525608172611531861591518342106662.dcm",
  "annotations": [
    {
      "label": "Zlomenina",
      "points": [
        { "x": 1999.5044598612492, "y": 1281.4667988107037 },
        { "x": 1999.5044598612492, "y": 1444.9950445986126 },
        { "x": 2002.4777006937566, "y": 1450.9415262636273 },
        { "x": 2005.4509415262637, "y": 1459.8612487611497 },
        { "x": 2008.4241823587715, "y": 1465.8077304261644 },
        { "x": 2011.397423191279, "y": 1468.780971258672 }
      ],
      "type": 1,
      "frame": 0
    },
    {
      "label": "Anomalie",
      "points": [
        { "x": 2793.359762140734, "y": 228.93954410307236 },
        { "x": 2597.125867195243, "y": 231.91278493557974 },
        { "x": 2525.768087215065, "y": 252.72547076313182 },
        { "x": 2843.9048562933604, "y": 258.6719524281467 },
        { "x": 2840.9316154608523, "y": 258.6719524281467 }
      ],
      "type": 2,
      "frame": 0
    }
  ]
}
```

Ukázka kódu 5.7: Ukázka výstupního souboru

5.6 Automatické aktualizace aplikace

V dnešní době je pro každou aplikaci nezbytné, aby se dokázala v případě vydání nové verze aktualizovat. Automatické aktualizace zajišťují, že uživatelé mají vždy přístup k nejnovějším funkcím a bezpečnostním opravám, aniž by museli ručně stahovat a instalovat nové verze. Proces vydávání aktualizací aplikace lze rozdělit na tři části:

- **Sestavení aplikace** – Krok ve kterém dojde k vytvoření spustitelného souboru pro všechny podporované platformy a vygenerování instalačního balíčku.
- **Distribuce aplikace** – Proces, při kterém je nová verze a její metadata nahrána na server a uživatelé jsou informováni o dostupnosti aktualizace.
- **Instalace aktualizace** – Proces, při kterém dojde k nahrazení staré verze novou verzí na straně uživatele.

5.6.1 Sestavení aplikace

Samotné sestavení Tauri aplikace pro aktuální platformu se provádí přes příkaz `npm tauri build`, nicméně to samo o sobě nestačí na automatizaci celého procesu. Při sestavení je žádoucí minimalizovat riziko lidské chyby a zajistit, že proběhne vždy se stejnými parametry a pro všechny platformy. Pro tento účel je vhodné použít nástroj pro automatizaci sestavení. V rámci této práce jsem zvolila nástroj GitHub Actions[4], jelikož je součástí platformy GitHub[5], kterou používám pro správu zdrojového kódu tohoto projektu.

GitHub Actions umožňuje vytvořit a spouštět skripty pro automatizaci různých činností, jako je například testování kódu nebo nasazení aplikace. Celé sestavení pro všechny platformy i následující kroky je definováno v rámci jednoho souboru a můžeme být následně kdykoliv provedeno pouhým kliknutím na tlačítko v rozhraní platformy GitHub.

5.6.2 Distribuce aplikace

Po sestavení je potřeba novou verzi aplikace dostat k uživateli. Existuje několik způsobů, jak toho lze dosáhnout. Nejčastěji se používá distribuce přes webový server, kde jsou uloženy instalační balíčky a soubor s neměnnou URL obsahující informace o nejnovější verzi aplikace. Uživatelská aplikace pak pravidelně kontroluje tuto URL a v případě, že je dostupná nová verze, tak uživateli se zobrazí notifikace o dostupnosti aktualizace.

[4] https://github.com/features/actions
[5] https://github.com/

K dosažení této funkcionality není nutné provozovat vlastní server, jelikož GitHub poskytuje nástroj Github Releases[6], který umožňuje hostovat instalační balíčky a metadata o nových verzích. Zároveň je velmi snadné propojit tento nástroj s GitHub Actions a vytvořit tak kompletní řešení pro automatizaci distribuce.

5.6.3 Instalace aktualizace

Posledním krokem při aktualizaci je, aby samotná aplikace na straně uživatele zaznamenala, že vyšla nová verze a nainstalovala ji. Aplikace kontroluje při spuštění URL s informacemi o dostupných verzí a v případě, že je dostupná nová verze, tak se uživateli zobrazí notifikace o dostupnosti aktualizace. Po potvrzení uživatelem se stáhne instalační balíček a spustí se instalace nové verze. Po dokončení instalace se aplikace restartuje a uživatel pokračuje v práci s novou verzí.

Tauri má v sobě tuto výše popsanou funkcionalitu zabudovanou a není tedy nutné ji implementovat ručně. Do konfiguračního souboru Tauri stačí předat na jaké URL má aplikace hledat informace o verzích a povolit automatické aktualizace. Výhodou využití zabudované funkcionality je zároveň i to, že Tauri aktualizace podepisuje a kontroluje, zda je podepsaná i nová verze, což zvyšuje bezpečnost a brání možným útokům.

[6] https://docs.github.com/en/repositories/releasing-projects-on-github/about-releases

Kapitola 6

Testování aplikace

Pro ověření kvality a funkčnosti aplikace jsem aplikaci rozeslala týmu dobrovolníků, kteří aplikaci řádně otestovali a poskytli k ní svojí zpětnou vazbu. Testování probíhalo na platformách Windows, Linux a prohlížečích Google Chrome a Mozilla Firefox. Testovací skupina byla složena z 5 členů, kteří první aplikaci otestovali bez bližšího seznámení s aplikací kvůli ověření intuitivnosti a poté se seznámili s aplikací a testovali ji podrobněji.

Zpětná vazba byla převážně velmi pozitivní. Testování potvrdilo, že je aplikace stabilní, konzistentně funguje na všech testovaných platformách a je intuitivní. Většina uživatelů byla schopna aplikaci používat bez předchozího vysvětlení, jak s aplikací pracovat. Nicméně testování odhalilo i několik zásadních problémů, které bylo nutno vyřešit. Jednalo se především o problémy s chybějícími omezeními uživatelských vstupů, které způsobovaly neočekávané chování aplikace a drobné nedostatky v uživatelském rozhraní. Níže je uveden seznam nejvýznamnějších problémů a jejich řešení.

6.1 Omezení délky textů

Při testování aplikace bylo zjištěno, že uživatelé byli schopni zadat do textových polí libovolně dlouhé texty, což způsobovalo v krajních případech pád aplikace. I přesto, že k pádům docházelo při zadání hodnoty v řádech tisíců znaků, které se v praxi nevyskytují, bylo nutné tuto chybu ošetřit a to omezením povolené délky vstupních polích a to na 25 znaků u názvu anotace a 500 znaků u popisu anotace. Toto omezení bylo implementováno pomocí HTML atributu `maxlength`.

6.2 Klávesové zkratky

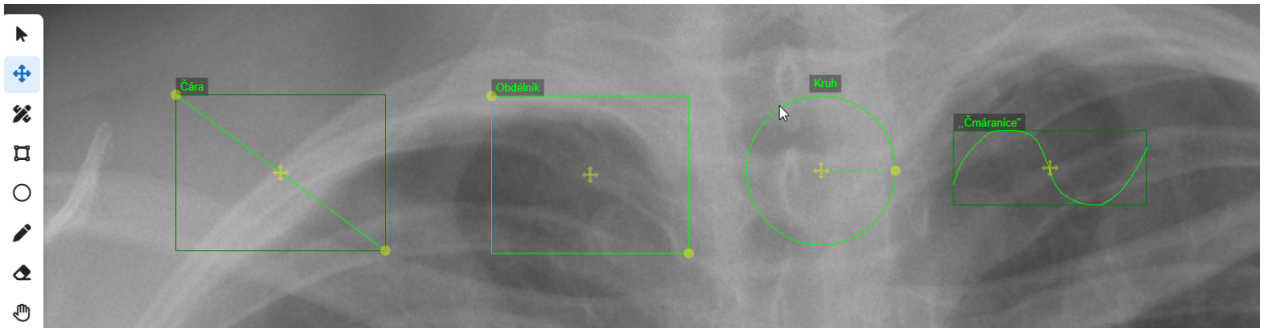
Uživatelé, kteří aplikaci testovali si stěžovali na absenci klávesových zkratek a to zejména pro přepínání mezi jednotlivými nástroji a anotacemi. Absence těchto zkratek byla označena za omezující, až frustrující. V reakci na tento podnět jsem přidala klávesové zkratky, které jsou uvedeny v tabulce 6.1.

Klávesa	Akce
1	Nástroj výběr
2	Nástroj úpravy
3	Nástroj čtverec
4	Nástroj kruh
5	Nástroj kreslení
6	Nástroj guma
7	Nástroj pohyb
Page up	Přepnout vrstvu snímku nahoru
Page down	Přepnout vrstvu snímku dolů
Ctrl + S	Uložit anotace

Tabulka 6.1: Seznam klávesových zkratek

6.3 Úprava pozice anotací

Uživatelé by uvítali možnost přesouvat již vytvořené anotace, případně lehce upravit jejich velikost. Z tohoto důvodu jsem do aplikace přidala nástroj na úpravu pozice anotací, který zobrazí editační body anotací, které je možné posouvat a tím upravovat umístnění a velikost jednotlivých anotací. Úprava anotací je vyobrazena na obr. 6.1.



Obrázek 6.1: Ukázka nástroje pro úpravu pozice a tvaru anotací

6.4 Orientace na snímku

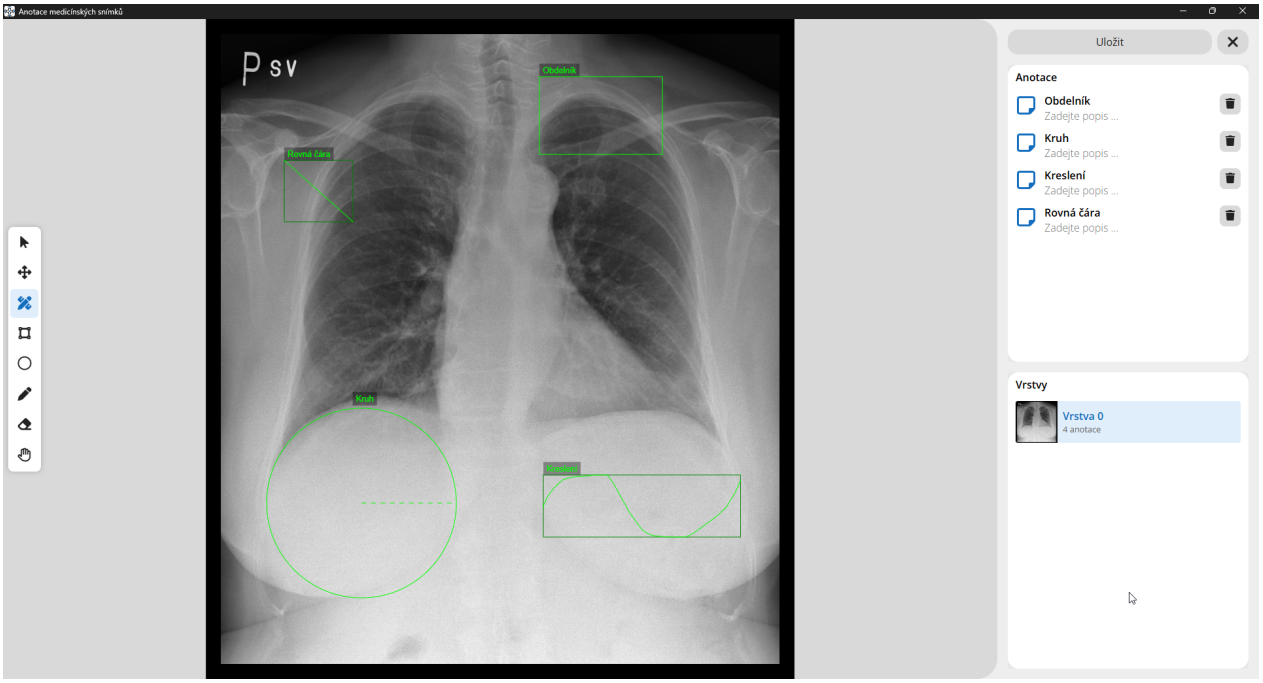
Při testování aplikace bylo zjištěno, že uživatelé měli problémy s orientací na snímku při různých úrovních přiblížení a taktéž, že je možné úplně ztratit snímek z obrazovky. Pro zlepšení navigace na snímku jsem omezila, jak moc lze snímek přiblížit, jelikož se při přílišném přiblížení nástroj pro kreslení a nástroj pro posun chovali nepředvídatelně a sekaně. Dále jsem přidala ukazatel přiblížení, který zobrazuje aktuální úroveň přiblížení a umožňuje rychle vrátit snímek do původního stavu. V

neposlední řadě jsem zamezila tomu, aby šel snímek posunout mimo obrazovku a přidáno tlačítko na jeho resetování.

Kapitola 7

Výstupní program

Výstupem implementační části této práce je multiplatformní nástroj pro anotaci medicínských snímků, který splňuje všechny požadavky uvedené v kapitole 3.1. Aplikace byla vyvinuta pomocí moderních technologií Tauri, Svelte a TypeScript. Aplikace je schopna pracovat s DICOM snímky a umožňuje uživatelům vytvářet, upravovat a mazat anotace. Ukázka hotové aplikace je uvedena na obrázku 7.1. Aplikace je plně funkční a stabilní, což bylo ověřeno testováním na různých platformách.



Obrázek 7.1: Ukázka hotové aplikace

Kapitola 8

Prostor pro další rozvoj

V této kapitole bych se chtěla zaměřit na teoretické možnosti dalšího rozšíření aplikace o nové funkce, kterým se tato práce nevěnovala. Popisuji zde několik konceptů, které by mohly zvýšit užitečnost a efektivitu aplikace.

8.1 Práce se vzdáleným úložištěm

Zdravotní zařízení s větším počtem lékařů a pacientů mohou mít potřebu ukládat snímky na vzdáleném serveru nebo v cloudu. Tato funkcionalita by umožnila lékařům přistupovat k datům z více zařízení a z různých míst. Aplikace by mohla podporovat otevírání a ukládání dat z nejčastěji používaných cloudových úložišť jako je Google Cloud, Amazon S3 nebo Microsoft Azure, nejčastěji používaných protokolů jako je FTP, HTTPS a speciálních lékařských programů, které používají sítové úložiště.

8.2 Detekce objektů na snímku

Do aplikace by bylo vhodné zabudovat funkci automatické detekce objektů na snímku pro ulehčení práce lékařům. Detekce by mohla být implementována například pomocí neuronových sítí, které by byly schopny přidat návrhy na anotace na základě analýzy snímku.

8.3 Lepší zařazení do pracovního procesu

Aplikace by mohla být napojená na další systémy, které lékaři používají a usnadnit tak proces anotace snímků. Jako příklad uvedu scénář, kdy lékař pořídí RTG snímek a po jeho pořízení se automaticky otevře anotační aplikace s tímto snímkem. Po dokončení anotací by se snímek uložil, zavřela by se anotační aplikace a spustila se další aplikace, která by s anotacemi dále pracovala.

Kapitola 9

Závěr

Cílem této práce bylo implementovat multiplatformní aplikaci pro anotaci medicínských snímků v prostředí internetu a desktopu, která by usnadnila lékařům práci s medicínskými snímky za pomocí jednoduchého uživatelského rozhraní.

Během analýzy existujících řešení jsem zjistila, že mnoho z dostupných nástrojů nesplňuje požadavky na univerzálnost, jednoduchost a podporu potřebných formátů, zejména formátu DICOM. Na základě zjištění jsem navrhla a vyvinula aplikaci, která se snaží odstranit tyto nedostatky. Implementace proběhla v jazyce TypeScript za použití frameworku Tauri a Svelte.

V rámci vývoje jsem věnována pozornost tomu, aby bylo rozhraní aplikace co nejvíce uživatelsky přívětivé. Do aplikace jsem implementovala načítání DICOM souborů, anotace snímků a export do běžných formátů. Kromě toho jsem aplikaci optimalizovala pro běh na různých platformách včetně webového prostředí.

Výsledná aplikace byla řádně otestována za pomoci dobrovolníků, kteří poskytli cennou zpětnou vazbu. Na základě jejich podnětů jsem provedla úpravy, jako například přidání klávesových zkratek, vylepšení ovládání aplikace a opraveny zásadní chyby.

Aplikace v současné podobě splňuje stanovené cíle, nicméně existuje prostor pro další rozvoj, například integrace s cloudovými úložišti, detekce objektů na snímku nebo lepší začlenění do pracovního procesu lékařů.

Literatura

1. DUTTA, A.; GUPTA, A.; ZISSERMANN, A. *VGG Image Annotator (VIA)* [http://www.robots.ox.ac.uk/ vgg/software/via/]. 2016. Version: 2.0.12, Accessed: 17. 4. 2025.
2. CVAT. *Computer Vision Annotation Tool*. 2024. Dostupné také z: `https://www.cvat.ai/`.
3. VIEWER, RadiAnt DICOM. *RadiAnt DICOM Viewer*. 2024. Dostupné také z: `https://www.radiantviewer.com/`.
4. *Electron*. 2023. Dostupné také z: `https://www.electronjs.org/`.
5. *Introduction to Electron*. 2024. Dostupné také z: `https://www.electronjs.org/docs/latest/`.
6. *Tauri*. [B.r.]. Dostupné také z: `https://tauri.app/`.
7. *Tauri Architecture*. 2025. Dostupné také z: `https://v2.tauri.app/concept/architecture/`.
8. *Svelte*. 2024. Dostupné také z: `https://svelte.dev/`.
9. TYPESCRIPT. *TypeScript Documentation*. 2024. Dostupné také z: `https://www.typescriptlang.org/docs/handbook/typescript-from-scratch.html`.
10. *DICOM*. 2024. Dostupné také z: `https://en.wikipedia.org/wiki/DICOM`.
11. *About DICOM*. 2024. Dostupné také z: `https://www.dicomstandard.org/about`.