UNIVERZITA PARDUBICE

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

BAKALÁŘSKÁ PRÁCE

2024 Maksym Hriha

Univerzita Pardubice

Fakulta elektrotechniky a informatiky

Webová aplikace pro správu multimediálních dat pro trénování neuronových sítí

Bakalářská práce

2024 Maksym Hriha

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: 2023/2024

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Maksym Hriha**
Osobní číslo:	**I21155**
Studijní program:	**B0688A140009 Informační technologie**
Téma práce:	**Webová aplikace pro správu multimediálních dat pro trénování neuronových sítí**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování

Cílem této práce je vytvoření webového aplikace, která umožní správu multimediální dat, která budou sloužit pro trénování konvolučních neuronových sítí. Aplikace bude realizována ve formě kontejnerů a bude spolupracovat s dalšími mikroslužbami.

Rozsah pracovní zprávy: **30**
Rozsah grafických prací:
Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

Joseph Ingeno, Software Architect's Handbook, Birmingham Packt Publishing Ltd., ISBN 978-1-78862--406-0
Eric J. Braude, Michael E. Bernstein, Software Engeneering Modern Approaches, Boston University, Metropolitan College, ISBN 978-1-4786-3230-6
NIEDERST ROBBINS, Jennifer. Learning Web design: a beginner's guide to HTML, CSS, JavaScript, and web graphics. 4th ed. Sebastopol: O'Reilly, c2012. ISBN 978-1-449-31927-4.
ROSENFELD, Louis a MORVILLE, Peter. Information architecture for the world wide web. Cambridge: O'Reilly, 1998. ISBN 1-56592-282-4.

Vedoucí bakalářské práce: **Ing. Martin Pozdílek, Ph.D.**
Katedra informačních technologií

Datum zadání bakalářské práce: **15. prosince 2023**
Termín odevzdání bakalářské práce: **10. května 2024**

L.S.

Ing. Zdeněk Němec, Ph.D. v.r.
děkan

Ing. Jan Panuš, Ph.D. v.r.
vedoucí katedry

V Pardubicích dne 28. února 2024

Prohlašuji:

Práci s názvem *Webová aplikace pro správu multimediálních dat pro trénování neuronových sítí* jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 7/2019 Pravidla pro odevzdávání, zveřejňování a formální úpravu závěrečných prací, ve znění pozdějších dodatků, bude práce zveřejněna prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 13.08.2024

Maksym Hriha v.r.

PODĚKOVÁNÍ

Rád bych vyjádřil svou hlubokou vděčnost svému vedoucímu práce, panu Ing. Martinovi Pozdílkovi, Ph.D., za jeho odborné vedení, trpělivost a cenné rady, které mi poskytl během psaní této bakalářské práce.

ANOTACE

Tato bakalářská práce se zabývá návrhem a implementací webové aplikace pro správu multimediálních dat, která slouží jako nástroj pro trénování neuronových sítí. V rámci práce byl vytvořen frontend aplikace s využitím moderních technologií jako jsou React, Tailwind CSS a Axios. Aplikace nabízí uživatelsky přívětivé rozhraní pro přihlašování, stejně jako možnost zakládat a spravovat projekty pro práci s obrazovými daty. Klíčovým aspektem je poskytnutí efektivního a intuitivního nástroje, který umožňuje uživatelům snadno nahrávat, třídit a připravovat data pro procesy strojového učení.

KLÍČOVÁ SLOVA

webové aplikace, frontend, React, Tailwind CSS, Axios Api, komponenta

TITLE

Web application for managing multimedia data for training neural networks

ANNOTATION

This bachelor thesis focuses on the design and implementation of a web application for managing multimedia data, which serves as a tool for training neural networks. The frontend of the application was developed using modern technologies such as React, Tailwind CSS, and Axios. The application provides a user-friendly interface for logging in, as well as the ability to establish and manage projects for working with image data. A key aspect is the provision of an efficient and intuitive tool that allows users to easily upload, categorize, and prepare data for machine learning processes.

KEYWORDS

web applications, frontend, React, Tailwind CSS, Axios API, component

OBSAH

SEZNAM ILUSTRACÍ A TABULEK

SEZNAM ZDROJOVÝCH KÓDŮ

SEZNAM ZKRATEK A ZNAČEK

API	Application Programming Interface
BMP	Bitmap
CRA	Create React App
CSS	Cascading Style Sheets
DICOM	Digital Imaging and Communications in Medicine
DOM	Document Object Model
HEIC	High Efficiency Image Format
HTML	HyperText Markup Language
JPEG	Joint Photographic Experts Group
JSON	JavaScript Object Notation
JSX	JavaScript XML
JWT	JSON Web Token
PNG	Portable Network Graphics
SOAP	Simple Object Access Protocol
SPA	Singlea Page Application
XML	Extensible Markup Language

ÚVOD

Ve světě neustálého technologického pokroku a digitalizace se správa multimediálních dat stává klíčovým prvkem ve výzkumu a vývoji umělé inteligence. Schopnost efektivně řídit a manipulovat s těmito daty je nezbytná pro trénování neuronových sítí, které jsou základem mnoha současných inovací. Tato bakalářská práce představuje návrh a vývoj webové aplikace, která slouží jako platforma pro správu a organizaci multimediálních dat s cílem usnadnit procesy trénování a vývoje neuronových sítí.

Cílem této práce je poskytnout detailní pohled na frontendovou část aplikace, včetně rozhodovacích procesů, které vedly k výběru použitých technologií. Hlavní roli hraje React, populární JavaScriptová knihovna, která umožňuje vytvářet interaktivní uživatelské rozhraní. Spolu s Tailwind CSS, který nabízí efektivní stylování, a axios API pro správu HTTP požadavků tvoří základ pro snadnou a přístupnou práci s obrazovými daty. Dále práce zkoumá různé knihovny, které byly integrovány do aplikace s cílem rozšířit její funkcionalitu a zlepšit uživatelský zážitek.

Motivací pro výběr tématu byl stále rostoucí význam efektivního a intuitivního přístupu k práci s velkým objemem dat a potřeba zjednodušení procesu přípravy dat pro strojové učení. Při tvorbě aplikace bylo klíčové zvážit aktuální potřeby výzkumných a vývojových týmů a poskytnout nástroj, který by tyto potřeby nejen splnil, ale i předčil v efektivitě a uživatelské přívětivosti.

1 TEORETICKÁ ČÁST

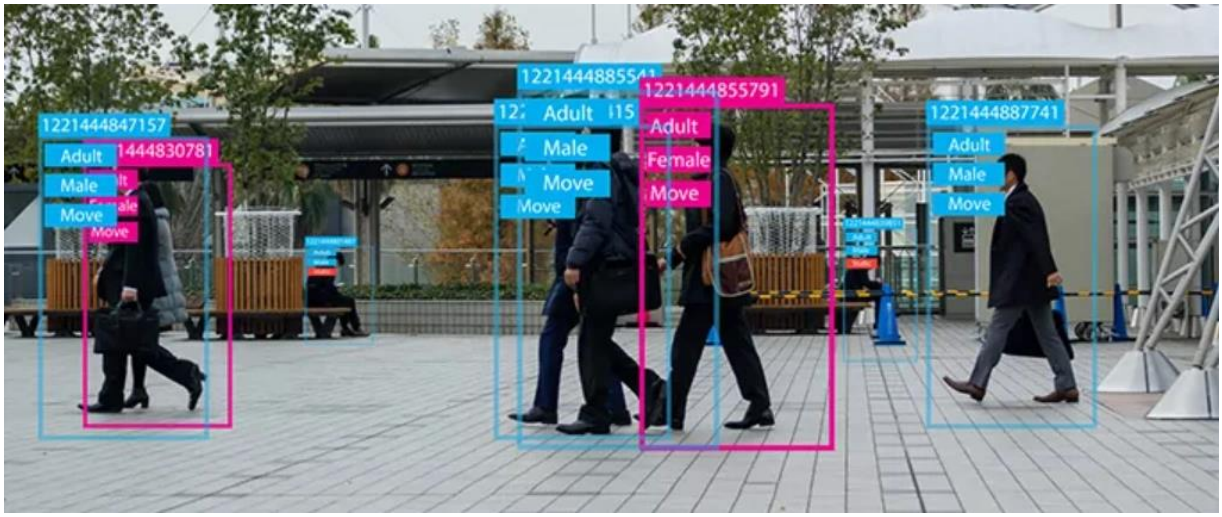
1.1 Problém anotace obrazu pro detekci a klasifikaci objektů

1.1.1 Úvod do problematiky

Anotace obrazu je klíčovým prvkem v procesu přípravy trénovacích dat pro úlohy strojového učení, zejména v oblastech, jako je detekce a klasifikace objektů. Tento proces zahrnuje ruční označování objektů na obrázcích, což umožňuje algoritmům strojového učení naučit se identifikovat a rozlišovat různé objekty na nových, dosud neviděných snímcích. Správně provedená anotace je naprosto nezbytná pro úspěch aplikací, jako jsou autonomní vozidla, systémy rozpoznávání obličejů nebo monitorovací systémy, které se spoléhají na přesné rozpoznávání objektů. Anotace obrazu tak není jen technickou formalitou, ale kritickým krokem, který má přímý dopad na kvalitu a spolehlivost vyvíjených modelů.

1.1.2 Význam přesné anotace

Přesnost anotací má zásadní význam pro úspěch a spolehlivost modelů strojového učení. Jakékoli nepřesnosti v anotacích mohou vést k nesprávnému učení modelů, což má za následek chybné rozpoznávání nebo klasifikaci objektů v praxi. Například v oblasti autonomního řízení by nepřesná anotace mohla vést k nesprávnému rozpoznání překážky na silnici, což by mělo vážné důsledky. Naopak, přesně a konzistentně anotovaná data výrazně zvyšují pravděpodobnost, že modely budou schopny nejen správně interpretovat nové obrázky, ale také generalizovat své znalosti na různé nové situace. Tato schopnost je klíčová pro bezpečné a efektivní nasazení technologií založených na umělé inteligenci v reálném světě. [1]



Obrázek 1: Přesnost anotace (zdroj: [1])

1.1.3 Výzvy v anotaci obrazu

Anotace obrazu představuje několik významných technických a logistických výzev, které je třeba při přípravě trénovacích dat překonat:

- **Konzistence anotací:** Zajištění konzistentních anotací napříč rozsáhlými datasety je nezbytné pro udržení vysoké kvality trénovacích dat. Nekonzistence může vést k nesprávnému učení modelů a tím i ke snížení jejich přesnosti.
- **Variabilita v datech:** Anotace musí pokrýt širokou škálu variací objektů, aby modely mohly generalizovat a efektivně rozpoznávat objekty v různých reálných situacích. To zahrnuje různé úhly pohledu, osvětlení, pozadí a další faktory, které mohou ovlivnit vzhled objektů.
- **Objem dat:** S rostoucím objemem obrazových dat roste i složitost anotace. Tento proces může být časově náročný a vyžaduje značné lidské zdroje, zvláště pokud jde o velké datasety, které jsou nutné pro moderní modely hlubokého učení. [2]
- **Kontrola kvality:** Pravidelná kontrola kvality anotací je nezbytná k zajištění jejich přesnosti a konzistence. Tato kontrola zahrnuje pečlivé řízení celého anotačního procesu a často vyžaduje revizi a úpravy, aby byly splněny požadované standardy kvality. [3]
- **Subjektivita anotace:** Ruční anotace mohou být ovlivněny subjektivním pohledem anotátorů, což může vést k rozdílům v interpretaci objektů a jejich hranic. Tento problém lze zmírnit použitím více anotátorů a následným sloučením jejich anotací. [3]

Tyto výzvy zdůrazňují, jak komplexní může být proces anotace obrazů, a ukazují na potřebu pečlivého plánování, řízení a kontroly, aby byly dosaženy vysoké standardy kvality, které jsou nezbytné pro úspěšné nasazení modelů strojového učení.

1.2 Existující řešení

Na internetu je dostupná řada nástrojů pro anotaci obrazu, které slouží k různým účelům od manuální anotace až po poloautomatizované systémy. Tyto nástroje se liší v uživatelské přívětivosti, funkcionalitě a dostupnosti pro širokou veřejnost.

- **LabelImg**: Jedná se o open-source nástroj, který poskytuje grafické uživatelské rozhraní pro anotaci obrázků ve formátech PASCAL VOC a YOLO. I přes své silné stránky v jednoduchosti a přístupnosti je primárně určen pro manuální anotaci, což může být časově náročné pro velké datasety. [4]
- **Labelbox**: Tento nástroj je naopak zaměřen na týmovou spolupráci a efektivitu, nabízí cloudové řešení s funkcemi AI asistované anotace. Labelbox je ideální pro komerční a velká data projekty, ale jeho použití vyžaduje placené předplatné, což může být bariéra pro jednotlivce nebo akademické uživatele. [5]
- **VGG Image Annotator** (VIA): VIA poskytuje flexibilní a snadno přizpůsobitelné prostředí pro anotaci obrazu, videa i zvuku, a to jak online, tak offline. Jeho open-source licencování a absence potřeby serverové backendové infrastruktury činí VIA vhodným pro výzkumné účely a projekty s omezeným rozpočtem. [5]
- **RectLabel:** Tento nástroj je specifický pro anotaci dat ve formátu YOLO a COCO. RectLabel je dostupný pro macOS a poskytuje jednoduché uživatelské rozhraní, které je ideální pro rychlé a efektivní vytváření anotací. Je oblíbený mezi uživateli, kteří preferují nativní aplikace a potřebují rychlé a snadné řešení pro menší projekty.
- **CVAT (Computer Vision Annotation Tool):** CVAT je další open-source nástroj, který je vysoce přizpůsobitelný a podporuje jak obrázky, tak videa. Nabízí možnosti automatizace pomocí předtrénovaných modelů, které mohou značně urychlit proces anotace. CVAT je vhodný pro velké průmyslové projekty i pro akademické výzkumy a je využíván v komunitách pro anotaci rozsáhlých datasetů. [4]

Většina dostupných nástrojů vyžaduje určitou úroveň technické zdatnosti a mnohé z nich jsou optimalizovány pro specifické úlohy, ne pro univerzální použití. To může být limitující pro uživatele, kteří hledají jednoduché a přímé řešení pro anotace obrazu. Například, i když nástroje jako Labelbox nabízí pokročilé funkce, mohou být příliš složité a drahé pro hobby projekty nebo malé týmy. Naopak, nástroje jako VIA nebo RectLabel mohou nabídnout dostatečné možnosti pro menší projekty, ale nemusí vyhovovat potřebám rozsáhlejších a komplexnějších úloh.

1.3 Architektura webových aplikací a API

1.3.1 Architektura webových aplikací

Architektura webových aplikací představuje základní strukturu, která definuje, jak různé softwarové komponenty, jako jsou databáze, aplikace a middleware, vzájemně komunikují a spolupracují. Správně navržená architektura zajišťuje, že data jsou efektivně a bezpečně přenášena prostřednictvím protokolu HTTP mezi klientskou částí (frontend) a backendovými servery. Architektura také garantuje, že uživatelské požadavky jsou zpracovány korektně, přičemž zajišťuje autorizaci, autentizaci a správu dat.

Výběr vhodné architektury je klíčový pro růst společnosti, její spolehlivost a schopnost vyhovět budoucím IT potřebám. Proto je důležité pochopit jednotlivé komponenty, které tvoří architekturu webových aplikací.

Komponenty architektury webové aplikace

Typická architektura webové aplikace se skládá ze tří hlavních komponent [6]:

1. **Webový prohlížeč (Web Browser):** Webový prohlížeč představuje klientskou část nebo frontend komponentu, která interaguje s uživatelem, přijímá vstupy a spravuje prezentační logiku. Tato komponenta zajišťuje validaci uživatelských vstupů, pokud je to potřeba, a kontroluje interakce uživatele s aplikací.

2. **Webový server (Web Server):** Webový server, známý také jako backend nebo serverová komponenta, zpracovává logiku pro aplikace a zajišťuje zpracování uživatelských požadavků. Směruje tyto požadavky na správné komponenty a řídí operace celé aplikace. Server může přijímat požadavky z různých klientských zařízení.

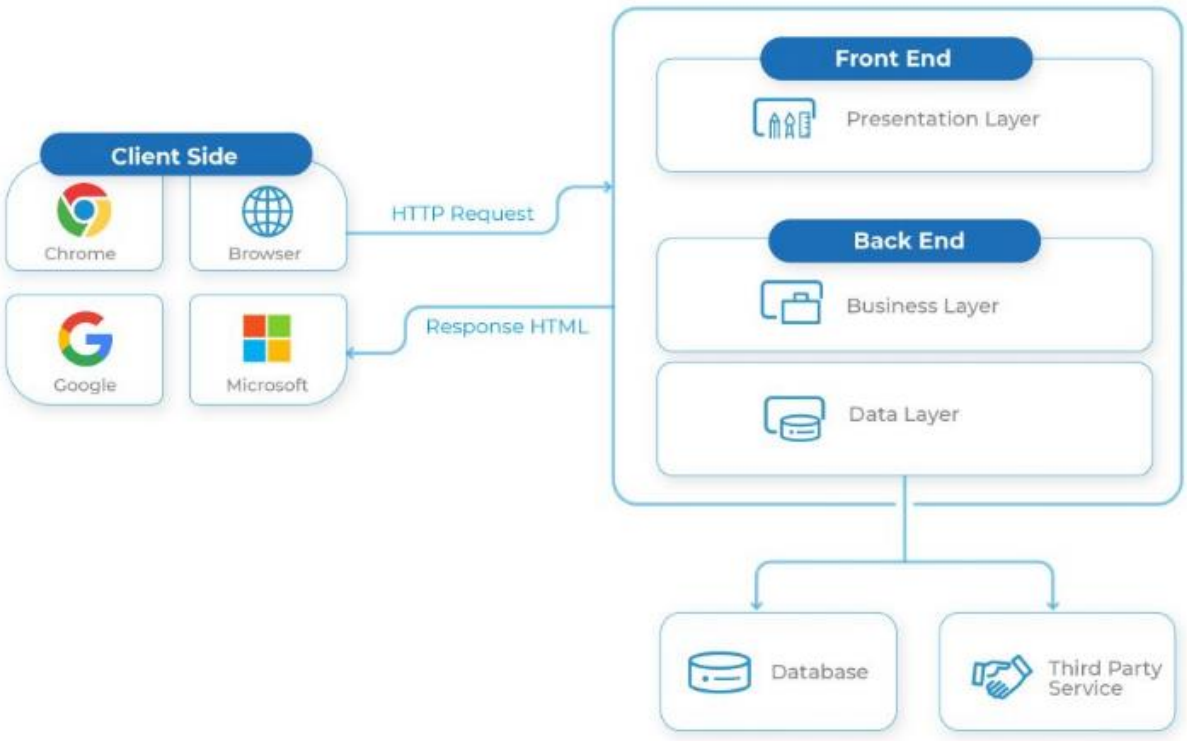
3. **Databázový server (Database Server):** Databázový server poskytuje potřebná data pro aplikaci a spravuje úkoly související s daty. V případě vícevrstvé architektury může databázový server spravovat obchodní logiku pomocí uložených procedur.

Vrstvy moderní architektury webových aplikací

Moderní architektura webových aplikací je navržena tak, aby byla škálovatelná, bezpečná a efektivní. Vrstvená architektura umožňuje snadnou identifikaci jednotlivých komponent a provádění změn v určité vrstvě bez vlivu na celou aplikaci. To výrazně usnadňuje psaní, ladění, správu a opětovné použití kódu.

Tři hlavní vrstvy moderní architektury webových aplikací jsou:

- **Prezentační vrstva / Klientská vrstva (Presentation Layer / Client Layer):** Tato vrstva zahrnuje klientskou část aplikace, která interaguje s uživatelem prostřednictvím webového prohlížeče. Kód této vrstvy běží v prohlížeči, přijímá požadavky uživatele a poskytuje mu odpovídající informace. V této vrstvě se nachází UI/UX design, dashboardy, notifikace, konfigurační nastavení, layout a interaktivní prvky.
- **Aplikační vrstva / Vrstva logiky pro aplikace (Application Layer / Business Logic Layer):** Tato vrstva zajišťuje zpracování logiky pro aplikace a komunikaci mezi prezentační vrstvou a datovou vrstvou. Obsahuje servery, databáze, webové služby a další prvky nezbytné pro fungování aplikace.
- **Datová vrstva (Data Layer):** Tato vrstva spravuje databáze a data potřebná pro aplikaci. Zajišťuje integritu dat, bezpečnost a efektivní přístup k datům.



Obrázek 2: Moderní architektura webových aplikací (zdroj: [6])

Třívrstvá architektura je více zabezpečená, protože klient přímo nepřistupuje k datům. Nasazení aplikačních serverů na více strojích poskytuje vyšší škálovatelnost, lepší výkon a lepší

znovupoužitelnost. Tento model umožňuje snadné řízení změn a úpravy jednotlivých vrstev bez narušení ostatních komponent.

Typy architektur webových aplikací

Architektura webových aplikací se může lišit podle způsobu vývoje a nasazení softwaru, a proto ji lze rozdělit do několika kategorií [7]. Každá z těchto kategorií má své vlastní charakteristiky, výhody i nevýhody, které ji předurčují k použití v různých typech projektů.

Monolitická architektura představuje tradiční přístup k vývoji softwaru, kde je celý systém vyvíjen jako jeden celek. V tomto modelu jsou všechny komponenty aplikace úzce propojené a závislé na sobě navzájem. Jakákoli změna nebo aktualizace určité funkce vyžaduje přepracování a kompilaci celého kódu, což může být časově náročné a komplikované. Tento přístup je vhodný zejména pro menší projekty s omezeným rozpočtem, kde je potřeba jednoduchého nasazení. Nicméně, jak aplikace roste, stává se její správa složitější a její škálování může být problémem.

Na druhé straně **architektura mikroservisů** řeší problémy spojené s monolitickým přístupem tím, že rozděluje aplikaci na volně propojené, nezávislé služby, které spolu komunikují prostřednictvím RESTful API. Každý mikroservis operuje s vlastní databází a konkrétní obchodní logikou, což umožňuje jednodušší vývoj, nasazení a škálování těchto služeb. Tento model je ideální pro složitější a vysoce škálovatelné aplikace. Na druhou stranu, správa a nasazení velkého množství mikroservisů může být komplikované a vyžaduje pečlivou organizaci.

Další možností je využití **kontejnerové technologie**, která se skvěle hodí pro nasazení mikroservisů. Kontejner poskytuje lehké a izolované běhové prostředí pro aplikaci, které může běžet na fyzickém nebo virtuálním stroji. To umožňuje efektivnější využití zdrojů, protože více komponent aplikace může běžet v jednom virtuálním prostředí. Technologie Docker je v tomto ohledu nejpopulárnější a nabízí bohatý ekosystém s rozsáhlou podporou komunity.

Nakonec, **serverless architektura** přináší inovativní model pro vývoj softwarových aplikací, kde poskytovatel infrastruktury zajišťuje správu zdrojů, zatímco vývojář platí pouze za skutečně využitou infrastrukturu. Tento přístup je ideální pro aplikace s proměnlivým zatížením, jako jsou multimediální zpracování, živé přenosy, chatboty nebo IoT. U mikroservisních aplikací lze serverless computing realizovat pomocí služeb jako AWS Lambda, API Gateway a API Step Functions, což zjednodušuje správu zdrojů a snižuje náklady.

1.3.2 API

Programové rozhraní aplikací, známé také pod zkratkou API (Application Programming Interface), je koncept, který umožňuje různým aplikacím komunikovat mezi sebou. API slouží jako prostředník, který umožňuje přístup k určitým datům a funkcím softwaru, a to prostřednictvím definovaných protokolů, nástrojů a podprogramů.

API není technologií jako takovou, ale spíše metodou, která usnadňuje integraci a komunikaci mezi různými softwarovými komponentami. Například když se uživatel přihlásí do aplikace, tato aplikace prostřednictvím API komunikuje se servery, aby získala uživatelské údaje a předala je zpět do uživatelského rozhraní.

Webové API je specifický druh API, který je přístupný přes webový protokol HTTP. Může být vybudováno pomocí různých technologií, jako jsou .NET nebo Java. API výrazně usnadňuje vývoj aplikací, protože umožňuje využívat existující funkce a snižuje tak náklady na vývoj a zkracuje čas potřebný k uvedení na trh.

Existuje několik typů API [8]:

- **RESTful API:** Representational State Transfer API je nejrozšířenějším typem API. Používá lehký formát JSON a je známé svou škálovatelností, spolehlivostí a rychlým výkonem.
- **SOAP:** Simple Object Access Protocol používá XML pro přenos dat a vyžaduje více šířky pásma a pokročilou bezpečnost.
- **XML-RPC:** Extensible Markup Language - Remote Procedure Calls využívá specifický formát XML pro přenos dat.
- **JSON-RPC:** Tento typ API používá formát JSON pro přenos dat.

API umožňuje vývojářům přístup k funkcím a datům jiných aplikací, což zvyšuje produktivitu, zlepšuje konektivitu v rámci ekosystému a zlepšuje uživatelskou zkušenost.

1.4 Cíle projektu

Cíle tohoto projektu jsou navrženy tak, aby splnily hlavní potřeby při správě multimediálních dat pro trénování neuronových sítí. Projekt se zaměřuje na vytvoření uživatelsky přívětivé a efektivní platformy, která usnadní práci s obrazovými daty a zajistí jejich snadnou správu. Níže jsou uvedeny hlavní cíle projektu.

1. Vytvořit intuitivní a efektivní platformu pro správu multimediálních dat určených k trénování neuronových sítí.
2. Zajistit podporu pro širokou škálu obrazových formátů, což umožní flexibilní práci s různými typy dat.
3. Umožnit snadný import, export a verzování datasetů pro efektivní správu a opakované využití dat.
4. Vyvinout nástroje pro hromadnou manipulaci s daty, které výrazně urychlí pracovní procesy a usnadní správu velkých objemů dat.
5. Zajistit bezpečný přístup k systému pomocí uživatelských účtů, včetně možnosti správy uživatelských dat a hesel.
6. Poskytnout uživatelsky přívětivé rozhraní, které umožní snadnou navigaci a efektivní práci i pro méně technicky zdatné uživatele.

1.5 Požadavky na aplikaci

Tato kapitola popisuje požadavky na aplikaci, které jsou klíčové pro její funkčnost a uživatelskou přívětivost. Požadavky jsou rozděleny do dvou hlavních kategorií: funkční požadavky, které se zaměřují na konkrétní funkce aplikace, a nefunkční požadavky, které stanovují obecné vlastnosti systému, jako je bezpečnost, výkon a škálovatelnost. Tyto požadavky zajišťují, že aplikace splní očekávání uživatelů a bude schopna efektivně podporovat různé pracovní procesy spojené se správou multimediálních dat.

1.5.1 Funkční požadavky (FR)

1. Projekt
    - FR1: Uživatelé mohou vytvářet projekty s jménem a timestampem, které lze kdykoliv přejmenovat, mazat nebo kopírovat.
    - FR2: Po přihlášení uživatel vidí seznam všech projektů, které může třídit a vyhledávat.
2. Dataset
    - FR3: Uživatelé mohou vytvářet, přejmenovávat, mazat a kopírovat datasety, které lze exportovat do ZIP formátu a zpětně importovat.
    - FR4: Datasety v projektu lze zobrazit v tabulkovém seznamu nebo jako náhledy, s možností třídění a vyhledávání.

3. Obrázky
    - FR5: Uživatelé mohou hromadně přidávat obrázky různých formátů, které lze třídit, filtrovat a označovat kategoriemi.
    - FR6: Kliknutím na náhled obrázku se zobrazí jeho plná velikost, kde je možné navigovat mezi obrázky a provádět anotace.
    - FR7: Obrázky lze zvětšovat a označovat objekty pro detailní prohlížení a anotace.

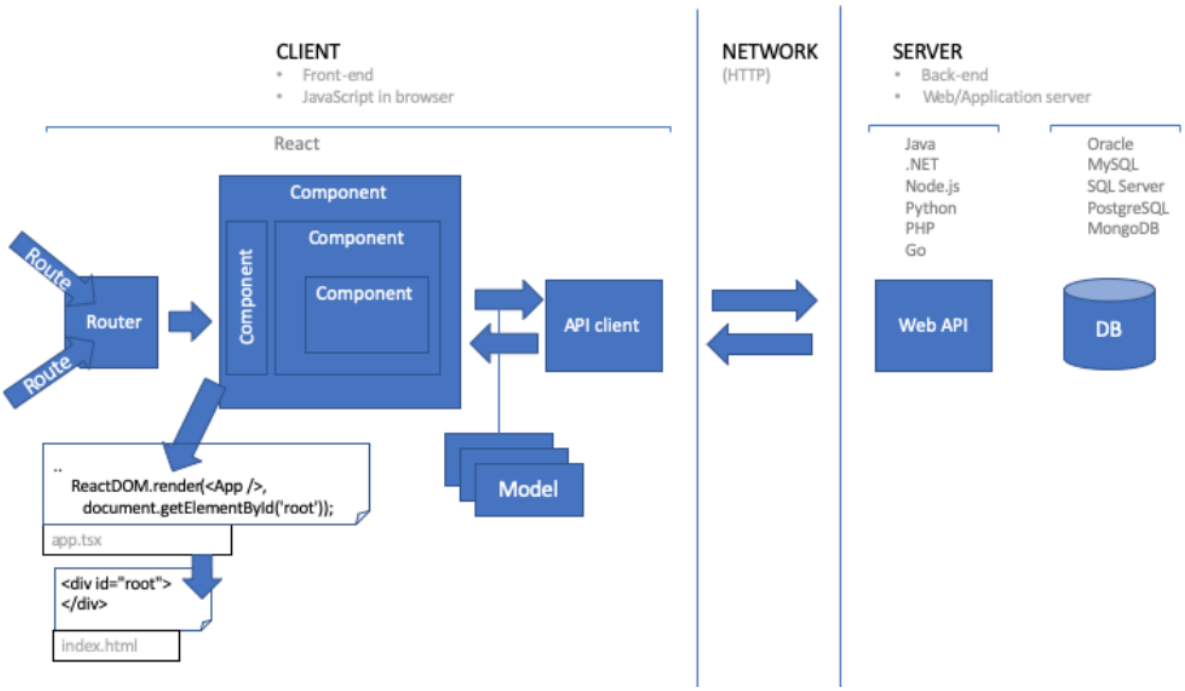
1.5.2 Nefunkční požadavky (NF)

- NF1: Systém musí zajistit bezpečný přístup pro ověřené uživatele pomocí hashovaných hesel.
- NF2: Aplikace musí být uživatelsky přívětivá a intuitivní na ovládání.
- NF3: Systém musí efektivně zvládat práci s velkými objemy dat bez zpomalení.
- NF4: Aplikace musí být kompatibilní s různými operačními systémy a prohlížeči.
- NF5: Systém musí být škálovatelný, aby umožňoval rozšiřování a růst počtu uživatelů a dat.

1.6 Fronend Frameworky

1.6.1 React

React.js je populární open-source JavaScriptová knihovna, kterou vyvinula společnost Facebook, a která se široce používá pro vytváření uživatelských rozhraní, zejména pro jednostránkové aplikace (SPA). React je navržen tak, aby usnadnil tvorbu komplexních uživatelských rozhraní pomocí komponentového přístupu. Každé uživatelské rozhraní je v Reactu rozděleno na menší, znovupoužitelné komponenty, které lze snadno spravovat a kombinovat. Tento modulární přístup nejen zlepšuje organizaci a udržovatelnost kódu, ale také umožňuje rychlejší vývoj a snadnější ladění. React také nabízí jednosměrný tok dat, který zjednodušuje sledování změn stavu aplikace a usnadňuje správu uživatelských interakcí. Díky těmto vlastnostem je React ideální volbou pro vývoj moderních, dynamických webových aplikací. [9]



Obrázek 3: Architektura Reactu (zdroj: [9])

Adresářová struktura React

Při vytváření nové React aplikace je nejjednodušší cestou použití nástroje Create React App (CRA), který automaticky vygeneruje základní adresářovou strukturu a všechny potřebné soubory pro spuštění aplikace. Tento nástroj nastaví projekt tak, aby obsahoval všechny nezbytné závislosti, konfigurační soubory a předpřipravené skripty pro běžné úkoly, jako je vývoj, testování a produkční nasazení. [10]

- *public/*: Tento adresář obsahuje statické soubory, které jsou dostupné přímo z webového serveru. Mezi tyto soubory patří například index.html, do kterého je vkládána React aplikace, dále ikony, obrázky, nebo jiné statické zdroje.
- *src/*: Hlavní adresář pro zdrojový kód aplikace. Obsahuje všechny soubory, které tvoří samotnou aplikaci.
    - *index.js*: Vstupní bod aplikace, kde se React aplikace připojuje k HTML stránce.
    - *App.js*: Hlavní komponenta aplikace, která často slouží jako kořen pro všechny ostatní komponenty.
    - *components/*: Podadresář, který obsahuje jednotlivé React komponenty, které jsou znovupoužitelné části uživatelského rozhraní.
    - *assets/*: (Volitelný) Adresář pro obrázky, fonty, nebo jiné statické zdroje, které jsou specifické pro aplikaci.
    - *styles/*: (Volitelný) Adresář pro soubory se styly, jako je CSS nebo SCSS, které definují vzhled aplikace.
    - *utils/*: (Volitelný) Adresář pro utility nebo pomocné funkce, které jsou používány napříč různými komponentami.
- *node_modules/*: Tento adresář obsahuje všechny závislosti projektu, které jsou instalovány pomocí npm nebo yarn. Není obvykle ručně spravován a je generován při instalaci závislostí.
- *package.json*: Konfigurační soubor, který obsahuje informace o projektu, včetně seznamu závislostí, skriptů pro spuštění a build aplikace, a dalších metadat.
- *package-lock.json*: Soubor, který zamyká verze závislostí, aby bylo zajištěno, že všichni vývojáři používají stejné verze knihoven.

Komponentová architektura

React je postaven na konceptu komponentové architektury. Komponenty jsou základními stavebními bloky každé React aplikace. Každá komponenta je samostatná a opakovaně použitelná část uživatelského rozhraní, která může obsahovat vlastní logiku a stav. Komponenty lze snadno kombinovat a hierarchicky organizovat, což umožňuje vytvářet složité uživatelské rozhraní z menších, modulárních částí. Tento přístup vede k lepší organizaci kódu, snazší údržbě a znovupoužitelnosti komponent v různých částech aplikace.

Virtuální DOM

React využívá koncept virtuálního DOMu (Document Object Model), což je lehká a efektivní reprezentace skutečného DOMu v paměti. Když se ve stavu aplikace něco změní, React aktualizuje virtuální DOM a porovná jej s předchozím stavem. Tento proces se nazývá **reconciliation**. Na základě této porovnání pak React efektivně aplikuje pouze nezbytné změny na skutečný DOM. Tím se minimalizuje počet operací s DOMem, což zlepšuje výkon aplikace, zejména u komplexních uživatelských rozhraní. [11]

Životní cyklus component

React komponenty procházejí životním cyklem, který zahrnuje různé fáze od vytvoření komponenty až po její odstranění z DOMu. Tento životní cyklus zahrnuje několik klíčových metod, které lze použít k provádění specifických úkolů, jako je inicializace stavu, získávání dat nebo uvolňování zdrojů v určitých okamžicích životního cyklu komponenty.

Hlavní fáze životního cyklu třídy komponenty zahrnují:

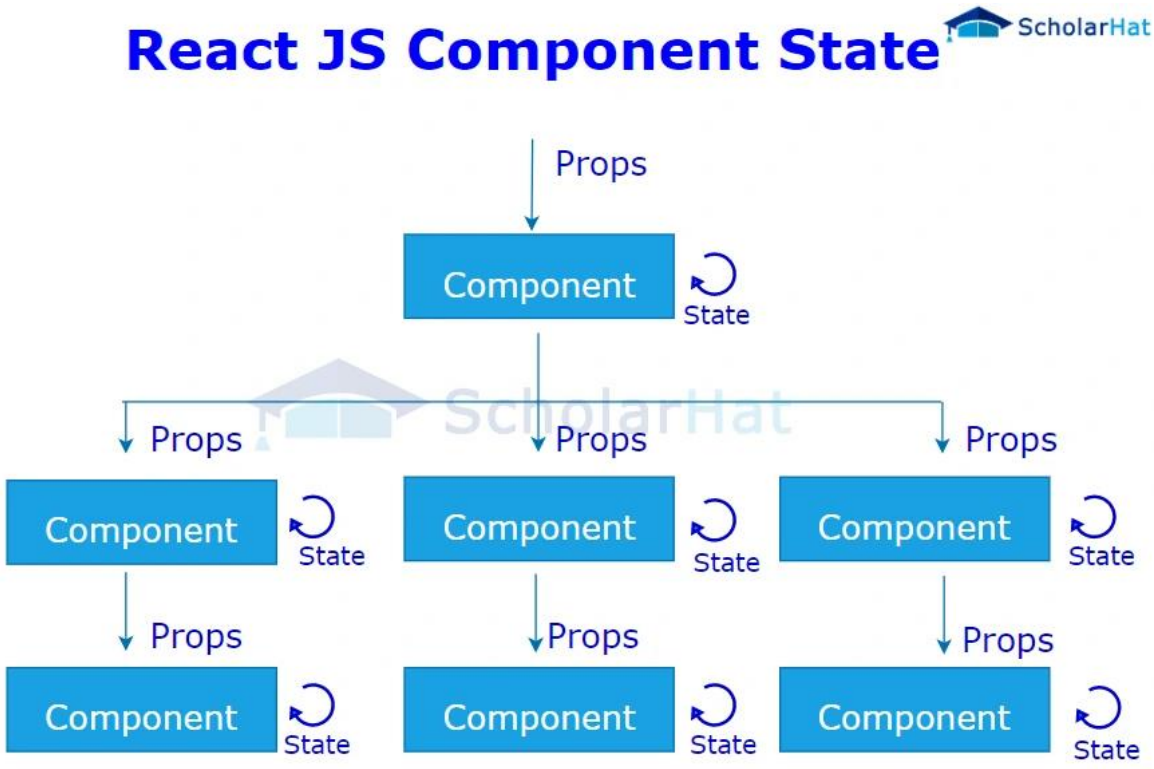
- **constructor()**: Tato funkce je první, která se volá při vytvoření komponenty. Používá se k inicializaci stavu komponenty a ke svázání (binding) událostních handlerů.
- **componentDidMount()**: Tato metoda je volána po renderování komponenty do DOMu. Obvykle se používá k získání dat z API nebo k navázání předplatného na externí služby.
- **shouldComponentUpdate(nextProps, nextState)**: Tato metoda je volána těsně před aktualizací komponenty. Umožňuje komponentě rozhodnout, zda se má znovu vykreslit na základě změn v props nebo stavu.

- **componentDidUpdate(prevProps, prevState)**: Tato metoda se volá po aktualizaci komponenty. Používá se k provádění následných akcí, jako je manipulace s DOMem nebo volání dalších API.
- **componentWillUnmount()**: Tato metoda se volá těsně před odstraněním komponenty z DOMu. Používá se k odstranění jakýchkoli zdrojů nebo event listenerů, které komponenta vytvořila.

Porozumění životnímu cyklu komponent je zásadní pro efektivní správu stavu, zpracování asynchronních úloh a optimalizaci výkonu v React aplikacích. [12]

Jednosměrný tok dat (One-Way Data Binding) a Props

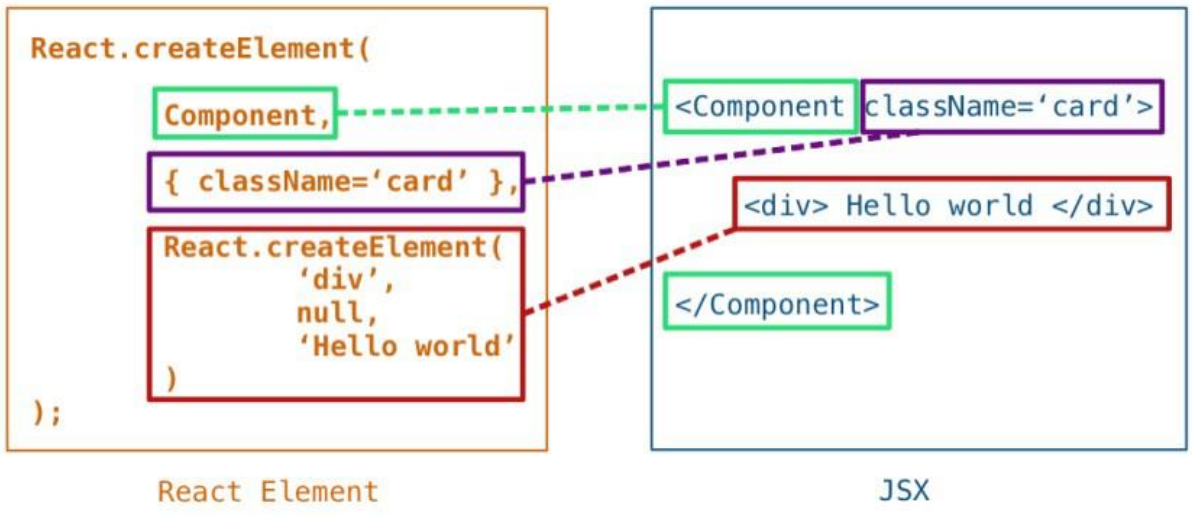
Jednosměrný tok dat je princip, na kterém je React založen. Data v aplikaci proudí jedním směrem – od rodičovské komponenty k dětským komponentám prostřednictvím **props**. Props jsou vlastnosti komponenty, které jsou předávány z rodičovské komponenty a umožňují dětským komponentám přijímat data a volat funkce. Props jsou neměnné, což znamená, že komponenta, která je přijímá, je nemůže přímo měnit. Tento tok dat usnadňuje sledování stavu aplikace a zajišťuje, že změny v jedné části kódu neovlivní jiné části nepředvídatelným způsobem.



Obrázek 4: Stav komponent React (zdroj: [12])

JSX – JavaScript XML

JSX je speciální syntaxe používaná v Reactu, která umožňuje psát HTML podobný kód přímo v JavaScriptu. JSX usnadňuje vývojáře vytvářet složité uživatelské rozhraní kombinováním HTML a JavaScriptu do jednoho souboru. Tento kód je při překladu přeměněn na volání React.createElement(), což umožňuje vytváření a manipulaci s DOMem v JavaScriptu. Použití JSX zlepšuje čitelnost a usnadňuje práci s komponentami a dynamickými daty.



Obrázek 5: React Element – JSX (zdroj: [13])

1.6.2 Alternativní Frameworky

Při výběru vhodného frameworku pro vývoj webových aplikací existuje několik populárních alternativ k Reactu. Každý z těchto frameworků nabízí své specifické výhody a funkce, které mohou být vhodné pro různé typy projektů. Mezi nejčastěji zvažované alternativy patří Angular, Vue, Svelte a Ember.

Angular je robustní framework vyvinutý společností Google, který je často využíván pro vývoj rozsáhlých enterprise aplikací. Nabízí dvoucestné vázání dat, což umožňuje synchronizaci dat mezi modelem a uživatelským rozhraním bez potřeby manuálního zásahu. Angular je známý svou komplexní architekturou, která zahrnuje vše od šablon přes služby až po testování, což může být výhodné pro větší týmy, ale také vyžaduje vyšší náročnost na učení. [14]

Vue.js je lehký a flexibilní framework, který se vyznačuje jednoduchostí a snadnou integrací do existujících projektů. Vue kombinuje nejlepší vlastnosti Reactu a Angularu a je oblíbený mezi vývojáři, kteří hledají rychlý start a jednoduchou syntaxi. Nabízí také dvoucestné vázání

dat a podporu pro server-side rendering, což z něj činí silnou volbu pro různé typy aplikací. [14]

Svelte je relativně nový framework, který se odlišuje od ostatních tím, že při kompilaci převádí komponenty na vysoce optimalizovaný JavaScript, který nepotřebuje virtuální DOM. Výsledkem je velmi malá velikost aplikací a vysoký výkon, což činí Svelte atraktivní volbou pro projekty, kde je rychlost a efektivita kritická. [14]

Ember.js je framework zaměřený na produktivitu vývojářů, který poskytuje mnoho zabudovaných funkcí a doporučených postupů. Ember je často využíván pro vývoj ambiciózních webových aplikací a má silnou komunitu a nástroje pro podporu složitých projektů. [14]

Pro lepší porozumění rozdílům mezi těmito frameworky a jejich vhodností pro různé typy projektů jsem našel a následně přepracoval tabulku srovnání. Tato tabulka obsahuje klíčové vlastnosti, jako je velikost, podpora server-side renderingu, architektura a populární použití, což poskytuje jasný přehled o tom, jak se jednotlivé frameworky liší a které mohou být nejvhodnější pro konkrétní scénáře.

Tabulka 1: Porovnání frameworku. Zdroj [14]

Funkce	React	Angular	Vue	Svelte	Ember
Virtuální DOM	Ano	Ne	Ano	Ne	Ano
Dvoucestné vázání dat	Ne	Ano	Ano	Ne	Ano
Komponentová architektura	Ano	Ano	Ano	Ano	Ano (volitelně)
Velikost	Malá (43,9 KB)	Velká (2,1 MB)	Malá (80 KB)	Malá (10,5 KB)	Velká (1,3 MB)
Server-Side Rendering	Ne (lze přidat pomocí Next.js)	Ne (lze přidat pomocí Angular Universal)	Ano (ale ne tak efektivní jako Next.js)	Ne (lze přidat pomocí Sapper)	Ne (lze přidat pomocí FastBoot nebo Glimmer.js)

Routing	React Router	Angular Router	Vue Router	Svelte Routing	Ember Router
Populární použití (např.)	Facebook, Instagram, Netflix, Airbnb, Dropbox	Google, Microsoft, IBM, Cisco	Alibaba Group, Xiaomi, Baidu, Xiaomi Youpin	Microsoft Teams, Discord	LinkedIn.com, Yahoo.com

1.6.3 Tailwind CSS

Tailwind CSS je utility-first CSS framework, který poskytuje nízkoúrovňové CSS třídy pro vytváření vlastních designů bez nutnosti psaní vlastního CSS kódu. Tailwind CSS byl zvolen pro stylování uživatelského rozhraní z následujících důvodů [15]:

- **Rychlost vývoje**: Díky předdefinovaným CSS třídám umožňuje Tailwind rychlou tvorbu stylů přímo v HTML nebo JSX kódu, což zkracuje čas potřebný na vývoj. Tento přístup také zjednodušuje údržbu kódu, protože všechny styly jsou viditelné přímo v komponentách.
- **Flexibilita a přizpůsobení**: Tailwind CSS poskytuje širokou škálu tříd, které lze snadno kombinovat a přizpůsobovat potřebám projektu. Umožňuje také definování vlastních barevných palet, typografie a dalších aspektů designu, což přináší vysokou míru flexibility.
- **Responsivní design**: Tailwind má zabudovanou podporu pro tvorbu responzivních designů pomocí specifických tříd, které usnadňují přizpůsobení rozvržení aplikace pro různá zařízení.

Princip fungování

Tailwind CSS funguje na principu rychlého vytváření uživatelského rozhraní pomocí předdefinovaných tříd, které jsou aplikovány přímo na HTML elementy. Tento přístup umožňuje vyhnout se psaní vlastního CSS od nuly. Místo toho vývojář kombinuje utility třídy přímo v HTML, čímž dosahuje požadovaného vzhledu a chování elementů. Každá třída v Tailwind CSS odpovídá jedné konkrétní CSS vlastnosti, například „p-4“ nastaví padding na

1rem, text-center zarovná text na střed, nebo „bg-blue-500“ nastaví pozadí na konkrétní modrou barvu. [16]

Tailwind CSS umožňuje vývojářům rychle iterovat na designu tím, že poskytuje flexibilní a snadno přizpůsobitelné styly bez nutnosti vytvářet složité a specifické CSS selektory. Tento přístup také přispívá k udržování konzistence stylů napříč celou aplikací, protože všechny styly jsou aplikovány prostřednictvím jednotných tříd.

Tailwind nabízí také možnost konfigurovat si vlastní sadu utility tříd, což znamená, že si můžete přizpůsobit framework podle potřeb projektu. Kromě toho je možné využívat různé varianty stylů, jako jsou responzivní design, hover efekty nebo dark mode, které jsou integrovány přímo do utility tříd. [17]

V této aplikaci byl Tailwind CSS použit pro tvorbu uživatelského rozhraní, včetně layoutů, tlačítek, formulářů a tabulek. Tailwind umožnil vytvoření moderního a přehledného designu s minimálním úsilím a bez nutnosti vytvářet vlastní CSS styly od základu.

2 PRAKTICKÁ ČÁST

V této části bude popsáno, jaké technologie byly použity při vývoji aplikace, jak aplikace funguje prostřednictvím ukázek jejího ovládání, a jak byla implementována její hlavní funkcionalita. Bude také uvedeno, jaká rozhodnutí a postupy vedly k vytvoření finální podoby aplikace.

2.1 Použité technologie

V praktické části vývoje aplikace byly využity následující klíčové technologie, které usnadnily a urychlily celý vývojový proces.

2.1.1 Visual Studio Code

Visual Studio Code (VS Code) je výkonný a oblíbený textový editor vyvinutý společností Microsoft, který byl použit pro napsání frontendové části této aplikace. VS Code je známý svou flexibilitou a rozsáhlou podporou rozšíření a pluginů, což z něj činí ideální nástroj pro vývoj moderních webových aplikací.

Jednou z hlavních výhod VS Code je jeho podpora pro různé programovací jazyky a vývojové nástroje, což umožňuje snadnou integraci s technologiemi, jako je React. VS Code poskytuje vývojářům robustní sadu nástrojů, včetně integrovaného terminálu, ladicího nástroje (debugger) a podpory pro Git, což výrazně zefektivňuje vývojový proces. Další výhodou je široká škála dostupných rozšíření, která umožňují přizpůsobení editoru konkrétním potřebám projektu, například automatické doplňování kódu, zvýrazňování syntaxe a podpora pro linting, což pomáhá udržovat kvalitu kódu.[18]

2.1.2 Postman

Postman je populární nástroj pro testování API, který umožňuje vývojářům vytvářet a testovat HTTP požadavky. Umožňuje snadno sestavit a poslat požadavky na server, zobrazit odpovědi a zpracovat data. V našem projektu byl Postman využit pro testování backendových služeb, ověřování správnosti API endpointů a zajištění, že komunikace mezi frontendem a backendem funguje podle očekávání. Tento nástroj rovněž umožnil simulaci různých scénářů, což přispělo k rychlejšímu odhalování a řešení problémů během vývoje.[19]

2.1.3 React Vite

Vite je moderní buildovací nástroj, který byl použit jako základní stavební nástroj pro náš React projekt. Vite je navržen tak, aby poskytoval rychlý a efektivní vývojový proces, a to díky svým schopnostem okamžitého načítání modifikovaného kódu a optimalizace pro produkční

nasazení. Vite podporuje moderní standardy JavaScriptu a CSS, což umožňuje využívat nejnovější funkce a postupy při vývoji. Díky Vite bylo možné dosáhnout rychlého sestavení a minimalizace velikosti produkčních balíčků, což přispělo k lepšímu výkonu a uživatelskému zážitku z naší aplikace. [20]

2.1.4 Různé knihovny pro aplikaci

V průběhu vývoje aplikace byly použity různé JavaScriptové knihovny, které pomohly rozšířit funkcionalitu a zjednodušit vývoj. Mezi hlavní knihovny patří:

- React Router: Použitý pro správu navigace v aplikaci. Umožňuje vytváření dynamických a responzivních tras, které zajišťují plynulý přechod mezi různými obrazovkami aplikace. [21]
- Axios: Knihovna pro zpracování HTTP požadavků, která byla využita pro komunikaci mezi frontendem a backendem. Axios usnadňuje práci s asynchronními požadavky a poskytuje jednoduché rozhraní pro práci s API. [21]
- React Hooks: Hooks jsou klíčovou součástí moderního vývoje v Reactu, umožňující správu stavu a vedlejších efektů v komponentách bez nutnosti používat třídy. Použití hooků jako useState, useEffect, nebo useContext výrazně zjednodušuje práci s komponentami a zlepšuje čitelnost kódu.

Použití těchto knihoven výrazně zkrátilo dobu vývoje a umožnilo zaměřit se na implementaci klíčových funkcionalit aplikace.

2.2 Ovládání a navigace v aplikaci

2.2.1 Realizace navigace

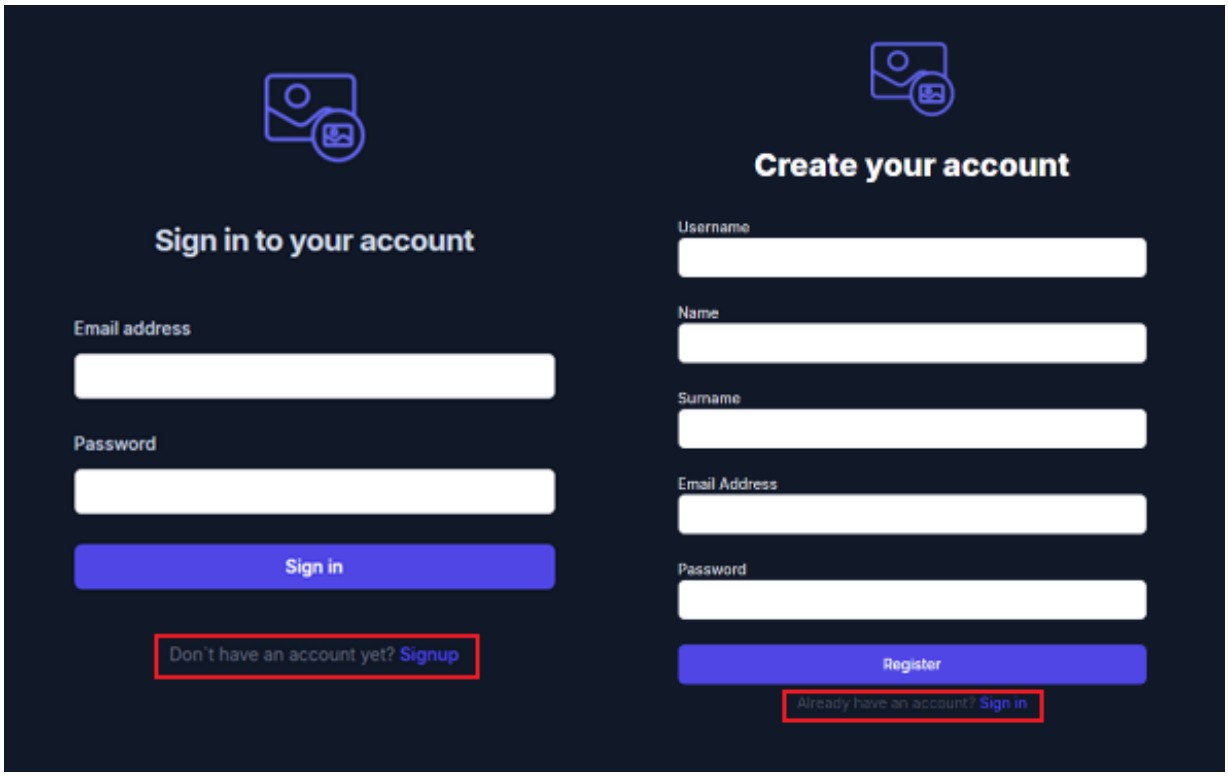
Navigace v aplikaci je zajištěna pomocí knihovny React Router, která umožňuje dynamické směrování mezi různými stránkami aplikace. React Router poskytuje uživatelsky přívětivé rozhraní, které zajišťuje plynulé přechody mezi jednotlivými obrazovkami bez nutnosti znovu načítat celou stránku. Každá hlavní část aplikace, jako je domovská stránka, stránka pro správu uživatelských dat, nebo stránka s nástroji pro anotaci obrazů, má svou vlastní trasu (route), která je definována v souboru pro konfiguraci navigace.

Uživatelé mohou mezi jednotlivými stránkami přecházet pomocí navigačního menu, které je umístěno v levém postranním panelu (sidebar) a je vždy viditelné. Tento sidebar obsahuje odkazy na hlavní části aplikace a umožňuje rychlý a snadný přístup k požadovaným funkcím.

Díky stálé dostupnosti tohoto navigačního panelu mohou uživatelé snadno procházet aplikaci, aniž by museli ztrácet čas hledáním konkrétních sekcí.

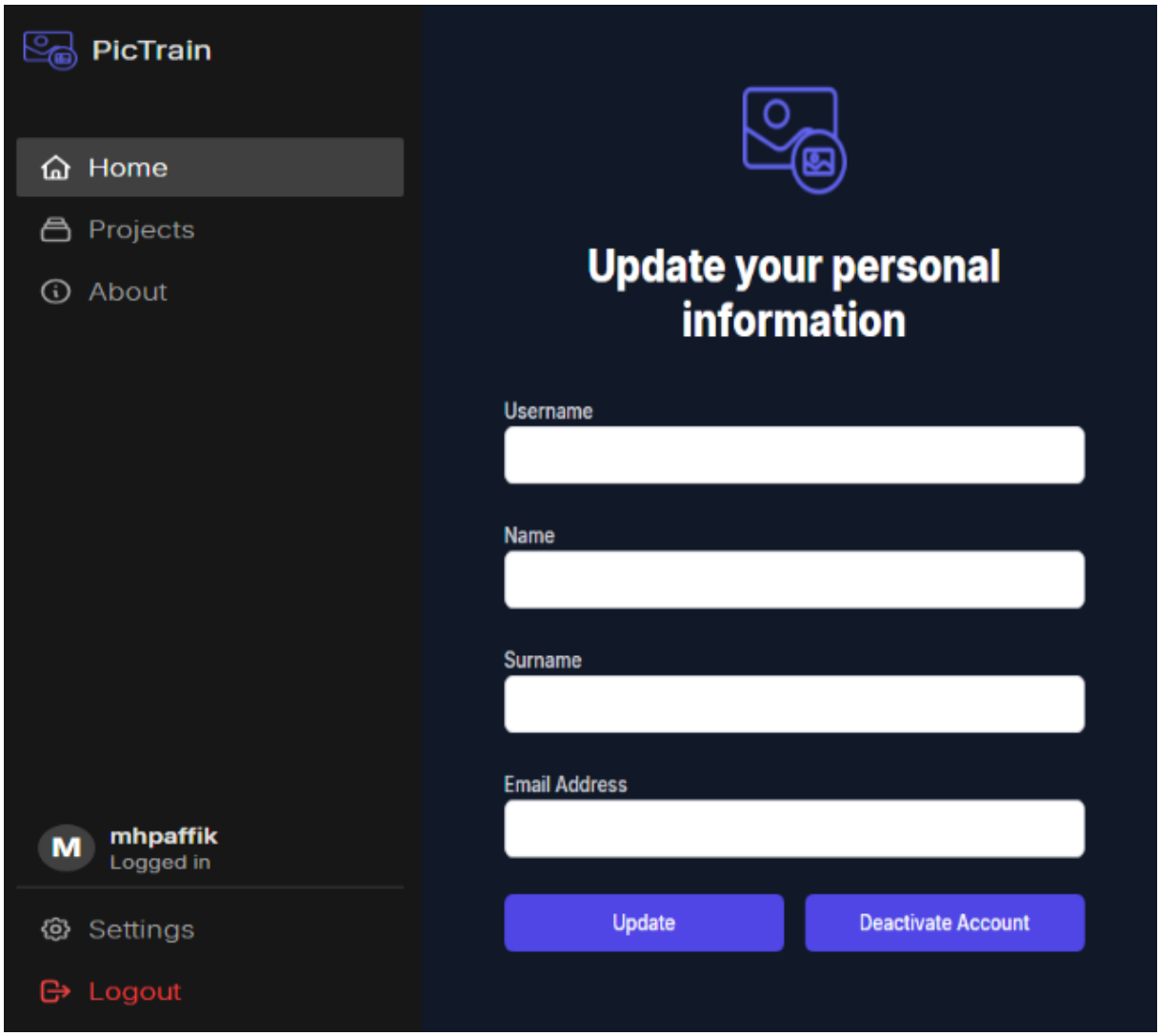
2.2.2 Ovládání aplikace

V následující části budou představeny jednotlivé obrazovky aplikace prostřednictvím obrázků. Každá obrazovka bude doplněna o podrobný popis jejího ovládání a navigace. Tento popis se zaměří na to, jakým způsobem mohou uživatelé využívat různé funkce, jak se orientovat v uživatelském rozhraní a jak efektivně pracovat s aplikací.



Obrázek 6: Navigace mezi registrací a přihlášením (zdroj: vlastní)

Na obrázku 6. jsou zobrazeny dvě stránky aplikace, které slouží pro přihlašování a registraci uživatele. Levá stránka je určena pro přihlášení (Sign in), kde uživatel zadává svou e-mailovou adresu a heslo. Pravá stránka slouží pro registraci nového uživatele (Sign up), kde jsou vyžadovány údaje jako uživatelské jméno, jméno, příjmení, e-mailová adresa a heslo. Pomocí červeně označených odkazů „Sign up“ na přihlašovací stránce a „Sign in“ na registrační stránce mohou uživatelé cyklicky přepínat mezi těmito dvěma stránkami. Tento mechanismus umožňuje snadný přístup k registraci, pokud uživatel ještě nemá účet, nebo k přihlášení, pokud už je registrován.



Obrázek 7: Navigace v sidebaru a stránku settings (zdroj: vlastní)

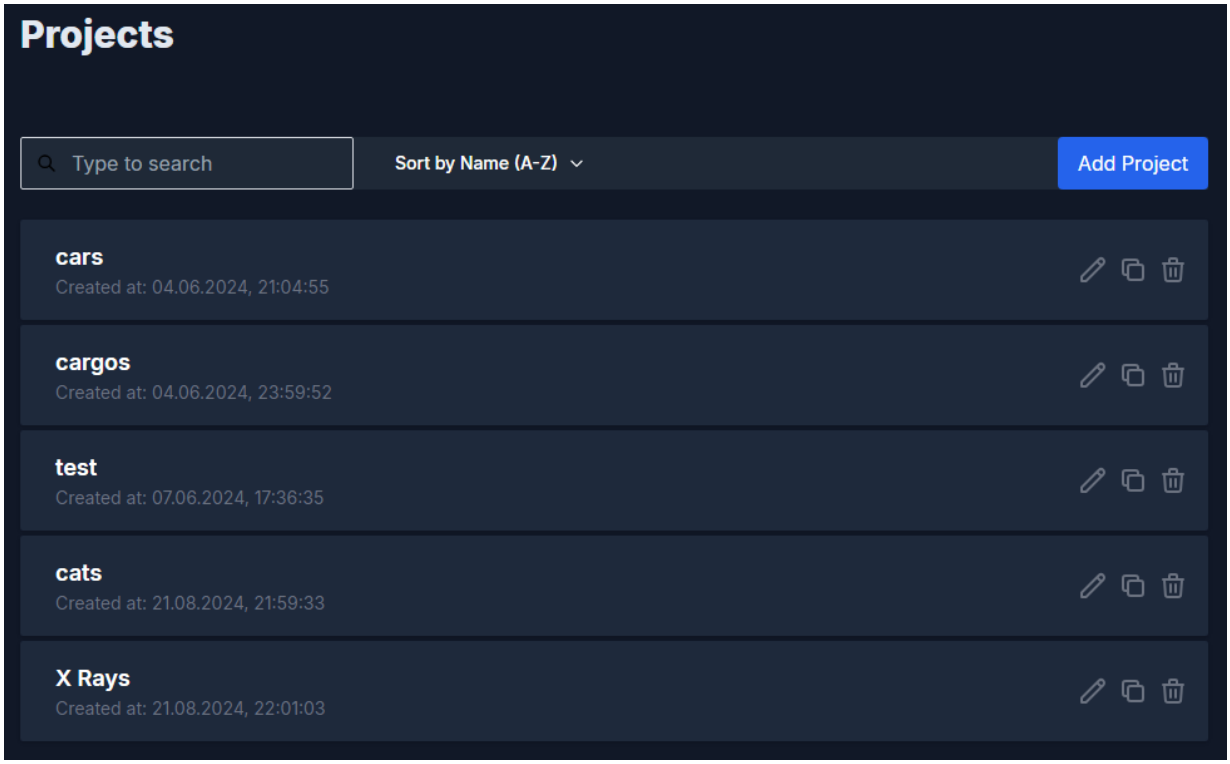
Na obrázku 7. je zobrazena stránka aplikace s levým postranním panelem (sidebar) a otevřenou stránkou „Settings“ (Nastavení), kde může uživatel aktualizovat své osobní údaje, jako je uživatelské jméno, jméno, příjmení a e-mailová adresa. Aplikace je navržena jako jednostránková aplikace (SPA), což znamená, že navigace mezi různými sekcemi aplikace probíhá bez nutnosti obnovování celé stránky. Všechny změny obsahu jsou prováděny dynamicky na stejné stránce.

Funkce sidebaru:

- **Home:** Kliknutím na tuto položku se uživatel vrátí na domovskou stránku aplikace, která může obsahovat přehled hlavních funkcí nebo rychlý přístup k projektům.

- **Projects:** Tato položka umožňuje přístup k seznamu projektů, na kterých uživatel pracuje, nebo může umožňovat vytváření nových projektů.
- **About:** Poskytuje informace o aplikaci nebo vývojovém týmu, který aplikaci vytvořil.
- **Settings:** Tato položka, která je právě aktivní, umožňuje uživateli spravovat jeho osobní údaje. Uživatel může aktualizovat své údaje nebo deaktivovat svůj účet.
- **Logout:** Kliknutím na tuto položku se uživatel odhlásí z aplikace.

Každá z těchto položek je v sidebaru neustále viditelná, což uživatelům umožňuje snadnou a rychlou navigaci mezi různými částmi aplikace. Po kliknutí na libovolnou položku v sidebaru se obsah hlavního panelu aplikace dynamicky změní, aniž by došlo k načítání nové stránky, což poskytuje plynulý a rychlý uživatelský zážitek. Stránka „Settings", stejně jako všechny ostatní části aplikace, je součástí tohoto interaktivního prostředí, kde lze snadno upravit osobní nastavení přímo na obrazovce.

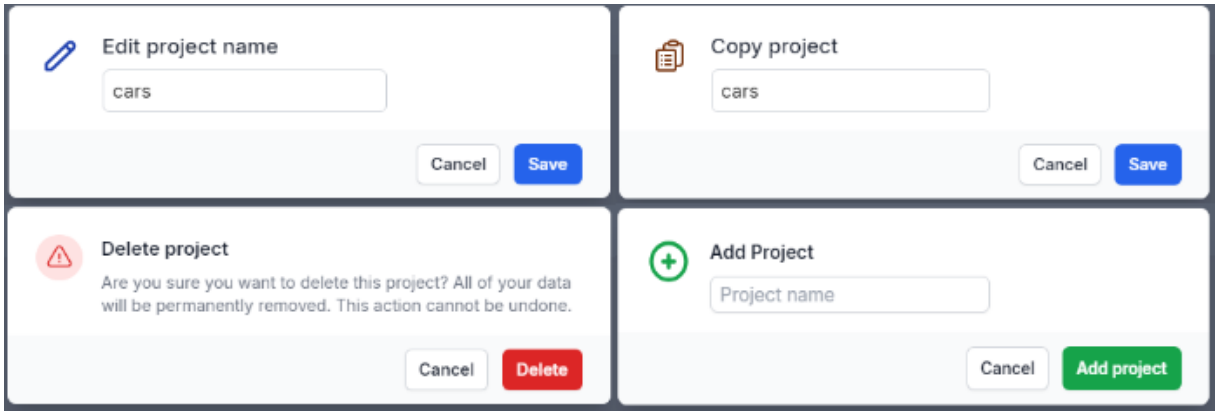


Obrázek 8: Navigace a správa projektů (zdroj: vlastní)

Na obrázku 8. je zobrazena hlavní stránka aplikace, která zobrazuje seznam všech projektů. Každý projekt je uveden ve vlastním řádku, který obsahuje název projektu a datum a čas jeho vytvoření (timestamp). Stránka je navržena tak, aby uživatelům umožnila snadnou správu projektů a rychlou orientaci mezi nimi.

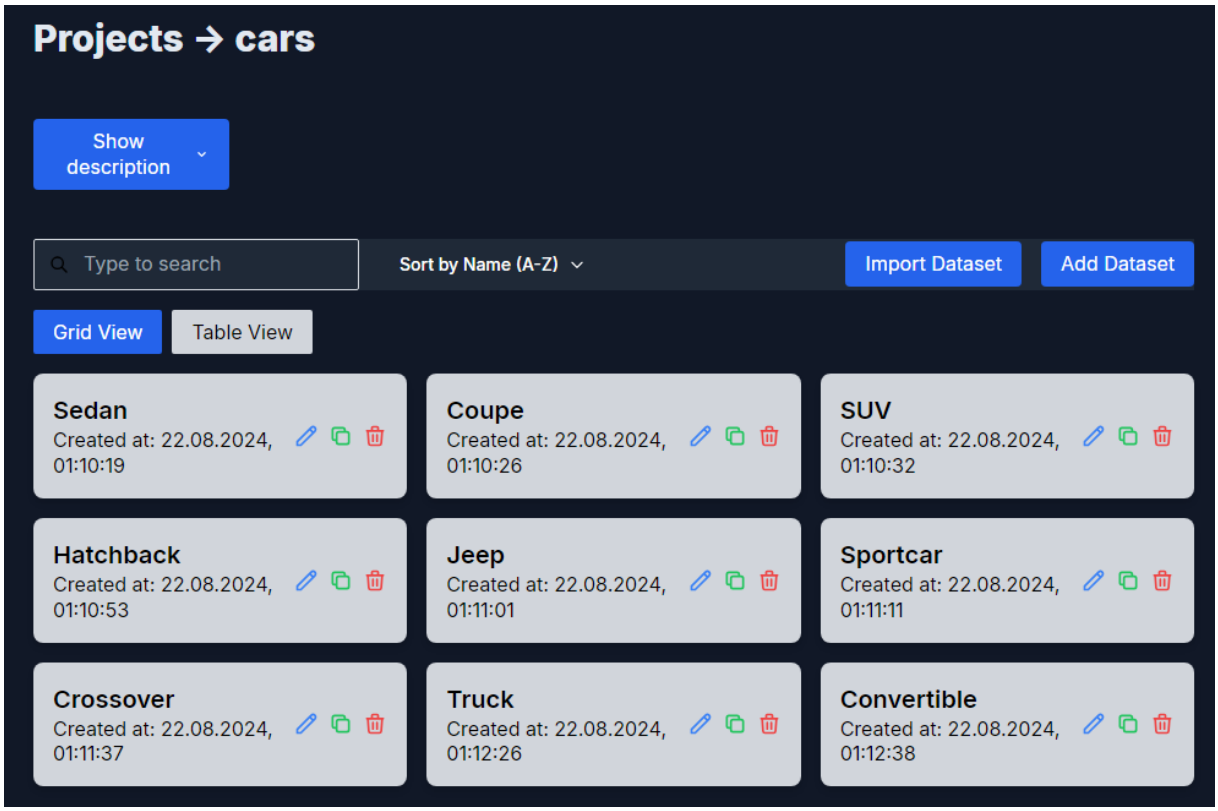
V horní části stránky se nachází vyhledávací pole **(**Type to search**)**, které umožňuje dynamické vyhledávání. Uživatelé mohou začít psát název projektu, a seznam se automaticky filtruje podle zadaných klíčových slov. Vedle vyhledávacího pole je umístěno rozbalovací menu pro řazení projektů. Uživatelé mohou třídit projekty podle jména (abecedně A-Z nebo Z-A) nebo podle časového razítka vytvoření (od nejstaršího po nejnovější a naopak). Tato funkce usnadňuje organizaci a rychlé nalezení konkrétního projektu.

V pravém horním rohu stránky se nachází tlačítko „Add Project“, které umožňuje uživatelům přidat nový projekt do seznamu. Každý projekt má vedle svého názvu tři ikony, které umožňují editaci, kopírování a mazání projektu. Tyto funkce budou podrobněji popsány na následujícím obrázku.



Obrázek 9: Navigace a správa projektů – Modální okna pro správu projektů (zdroj: vlastní)

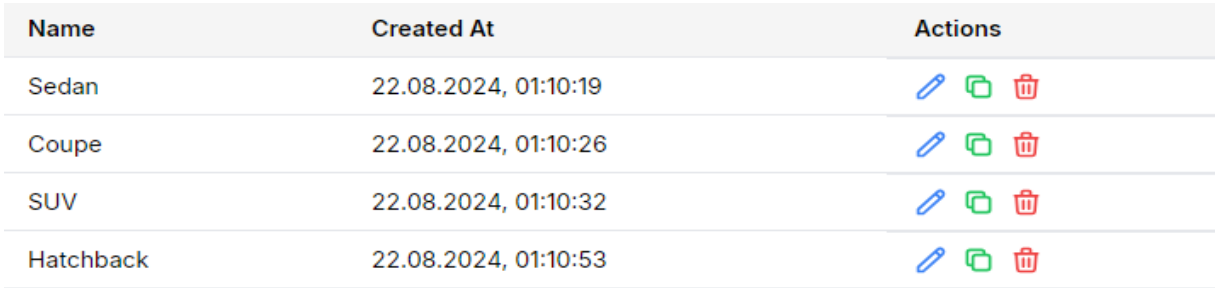
Tento obrázek 9. zobrazuje modální okna, která se otevřou při kliknutí na tlačítka pro úpravu, kopírování, mazání nebo přidání projektu. Každé okno umožňuje provést konkrétní akci spojenou se správou projektů. Modální okna se automaticky zavřou, pokud uživatel klikne mimo okno nebo zvolí možnost „Cancel“, což umožňuje snadné zrušení akce.



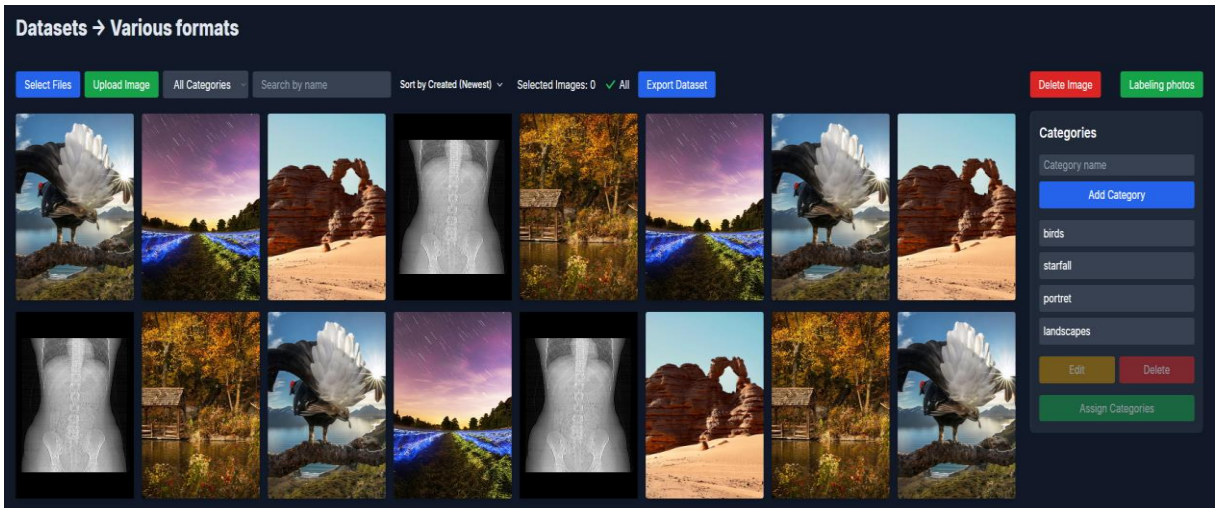
Obrázek 10: Navigace a správa datasetů v projektu (zdroj: vlastní)

Na tomto obrázku 10. je zobrazen seznam datasetů obsažených v konkrétním projektu, v tomto případě projektu „cars“. Po kliknutí na název projektu se otevře tato stránka, kde mohou uživatelé spravovat jednotlivé datasety. Správa datasetů je velmi podobná správě projektů – uživatelé mohou dataset upravit, kopírovat nebo smazat pomocí ikon vedle každého datasetu.

Kromě toho je zde možnost přidání nového datasetu pomocí tlačítka „Add Dataset“ nebo importování existujícího datasetu pomocí tlačítka „Import Dataset“. Při použití funkce importu je vyžadováno nahrání ZIP archivu, který obsahuje metadata a obrázky, jež budou součástí datasetu. Tato funkce umožňuje snadno integrovat externí datasety do projektu. Uživatelé mohou také přepínat mezi zobrazením mřížky (Grid View) a tabulkovým zobrazením (Table View) pro lepší přehlednost.



Obrázek 11: Tabulkové zobrazení datasetu (zdroj: vlastní)



Obrázek 12: Navigace a správa obrázků v datasetu (zdroj: vlastní)

Na tomto obrázku 12. je zobrazen seznam obrázků patřících k vybranému datasetu, který se otevře po kliknutí na konkrétní dataset. Aplikace podporuje širokou škálu formátů obrázků, včetně běžných formátů jako „jpg“ a „png“, ale také méně obvyklých formátů, jako je „bmp“ (Bitmap), „heic“ (High Efficiency Image Format), který je známý svou efektivní kompresí, a „dicom“ (Digital Imaging and Communications in Medicine), který se často používá v lékařské zobrazovací technice. Uživatelé mohou snadno spravovat obrázky a přiřazovat je k různým kategoriím. Zmíněné funkce správy obrázků a kategorií budou detailněji popsány v následujících obrázcích.

Obrázek 13: Navigace a správa obrázků – Výběr, nahrávání, filtrování a export (zdroj: vlastní)

Tento obrázek 13. zobrazuje nástroje pro správu obrázků v rámci datasetu. Následuje podrobný popis jednotlivých funkcí, které jsou dostupné pro efektivní práci s obrázky:

1. **Výběr souborů a nahrávání obrázků:**
    - Select Files**:** Uživatel může vybrat jeden nebo více obrázků z podporovaných formátů („jpg”, „png”, „bmp”, „heic”, „dicom”) na svém zařízení. Tyto obrázky jsou poté připraveny k nahrání do datasetu.
    - Upload Image**:** Po výběru souborů uživatel klikne na toto tlačítko, aby nahrál vybrané obrázky do seznamu datasetu.

2. **Filtrování podle kategorie:**
    - All Categories**:** Tento rozbalovací seznam umožňuje uživateli filtrovat zobrazené obrázky podle jejich přiřazených kategorií. Uživatel může zvolit konkrétní kategorii a zobrazit pouze obrázky, které do ní patří.
3. **Dynamické vyhledávání:**
    - Search by name**:** Toto pole umožňuje uživatelům dynamicky vyhledávat obrázky podle názvu. Jakmile uživatel začne psát, seznam se automaticky filtruje podle zadaných klíčových slov.
4. **Řazení obrázků:**
    - Sort by Created (Newest)**:** Uživatel může třídit obrázky podle různých kritérií, jako je čas vytvoření (od nejnovějších po nejstarší) nebo jiné dostupné možnosti řazení.
5. **Počet vybraných obrázků a výběr všech:**
    - Selected Images**:** Zobrazuje aktuální počet vybraných obrázků. Uživatel může vybrat obrázky jednotlivě nebo použít možnost „All" pro vybrání všech obrázků zobrazených na aktuální stránce.
6. **Export datasetu:**
    - Export Dataset**:** Toto tlačítko umožňuje uživatelům exportovat celý dataset do ZIP archivu. Tento archiv obsahuje všechny obrázky v datasetu, včetně jejich metadat a kategorií.

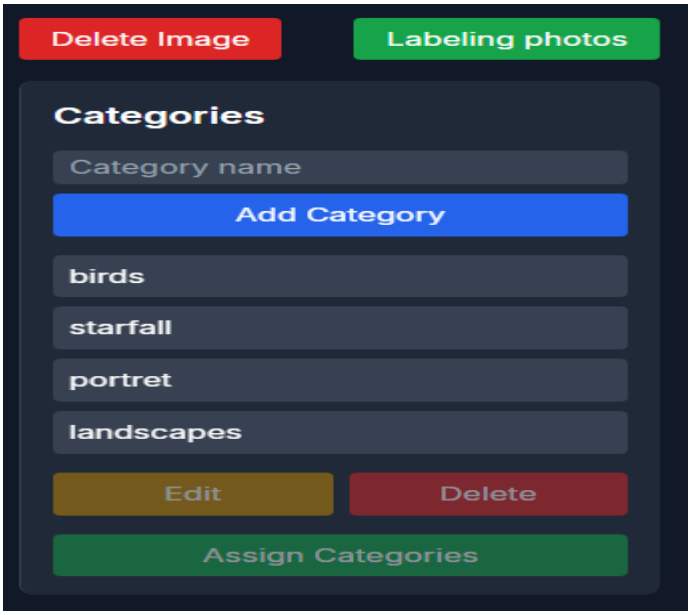
Tyto nástroje poskytují uživatelům komplexní kontrolu nad obsahem datasetu, usnadňují organizaci a manipulaci s obrázky a zajišťují rychlý a efektivní pracovní postup.

Obrázek 14: Navigace a správa označených obrázků v datasetu (zdroj: vlastní)

Na tomto obrázku 14. je znázorněno, jak vypadá označení obrázků v datasetu. Po označení (zvýraznění modrým rámečkem) mohou uživatelé s těmito obrázky dále pracovat. Konkrétně mohou:

- **Přiřadit kategorie:** Označeným obrázkům lze přiřadit konkrétní kategorie, což usnadňuje jejich organizaci a pozdější filtrování. Tento krok je také zásadní pro klasifikační anotace, které jsou nezbytné při trénování neuronových sítí.
- **Mazat obrázky:** Uživatel může smazat označené obrázky ze seznamu datasetu.
- **Označovat objekty pomocí Annotation Tool:** Uživatelé mohou na označených obrázcích identifikovat a anotovat objekty pomocí nástroje pro anotaci, což je klíčová funkce pro přípravu dat pro trénování neuronových sítí.

Tento systém označování umožňuje efektivní správu a manipulaci s obrázky v datasetu.



Obrázek 15:Navigace a správa kategorií obrázků (zdroj: vlastní)

Tento obrázek 15. zobrazuje rozhraní pro správu kategorií obrázků v datasetu. Aby mohl uživatel provést jakoukoli akci týkající se kategorií, musí nejprve vybrat konkrétní kategorii ze seznamu a případně označit jeden nebo více obrázků.

Pro přidání nové kategorie stačí zadat název do textového pole a kliknout na tlačítko „Add Category". Pokud uživatel chce upravit nebo odstranit existující kategorii, musí ji nejprve vybrat a poté použít tlačítka „Edit" pro editaci nebo „Delete" pro smazání.

Pokud chce uživatel přiřadit kategorii k obrázkům, musí nejprve označit požadované obrázky a vybrat kategorii ze seznamu. Poté může kliknout na tlačítko „Assign Categories", čímž přiřadí zvolenou kategorii k vybraným obrázkům. Důležité je zmínit, že každý obrázek může mít přiřazeno více kategorií současně, což umožňuje flexibilnější organizaci a vyhledávání v datasetu.

Toto rozhraní poskytuje uživatelům snadnou možnost organizace obrázků v rámci datasetu, což zlepšuje přehlednost a usnadňuje následnou práci s daty.

2.3 Implementace

V této kapitole bude popsána implementace aplikace, přičemž budou představeny jednotlivé části zdrojového kódu, které byly klíčové pro dosažení požadované funkcionality. Dále budou popsány problémy, které se vyskytly během vývoje, a způsoby, jak byly tyto problémy vyřešeny.

2.3.1 Základní architektura aplikace

Aplikace je rozdělena na dvě hlavní části: frontendovou část (client) a backendovou část (server). Frontendová část je implementována v Reactu, který poskytuje dynamické a interaktivní uživatelské rozhraní. Pro komunikaci mezi frontendem a backendem je využívána knihovna Axios, která usnadňuje práci s HTTP požadavky. Backendová část je postavena na frameworku Express.js, který běží na Node.js a poskytuje robustní API endpointy pro komunikaci s databází a zpracování požadavků.

Backendová část zajišťuje správu uživatelských dat, zpracování požadavků z frontendové části a komunikaci s databází. API endpointy jsou chráněny middlewarem, který ověřuje autentizaci pomocí JSON Web Tokens (JWT).

```
const token = localStorage.getItem("token");
await axios.post(
  `http://localhost:8080/api/images/assign-category`,
  {
    categoryId: selectedCategory._id,
    imageIds: selectedImages,
  },
  {
    headers: {
      Authorization: `Bearer ${token}`,
    },
  }
);
```

Zdrojový kód 1: Client - HTTP požadavek pro přiřazení kategorie (zdroj: vlastní)

Tento zdrojový kód 1. ukazuje, jak frontendová část aplikace odesílá HTTP POST požadavek na server pomocí knihovny Axios. Tento požadavek slouží k přiřazení kategorie k vybraným obrázkům. Požadavek obsahuje ID kategorie a ID obrázků, které mají být aktualizovány. Autentizační token je připojen k požadavku v záhlaví (header) pro ověření uživatele.

Kód získává token uložený v prohlížeči (localStorage) a připojuje jej k záhlaví HTTP požadavku jako součást autorizace. Následně odesílá data na specifikovaný API endpoint, kde jsou zpracována.

```
app.post("/api/images/assign-category", authenticateToken, async (req, res) => {
  const { categoryId, imageIds } = req.body;

  try {
    const images = await Image.updateMany(
      { _id: { $in: imageIds } },
      { $addToSet: { categories: categoryId } }
    );
    res.status(200).json({ message: "Category assigned successfully" });
  } catch (error) {
    console.error("Error assigning category to images:", error);
    res.status(500).json({ message: "Error assigning category to images" });
  }
});
```

Zdrojový kód 2: Server - API endpoint pro přiřazení kategorie obrázkům (zdroj: vlastní)

Tento kód 2. představuje API endpoint na straně serveru, který zpracovává požadavek na přiřazení kategorie k obrázkům. Endpoint je chráněn middlewarem pro autentizaci (viz následující obrázek). Pokud jsou všechny údaje správně předány, server aktualizuje databázi, přidá ID kategorie do vybraných obrázků a vrátí úspěšnou odpověď. V případě chyby server vrátí odpověď s chybovým hlášením.

Endpoint přijímá categoryId a imageIds z těla požadavku, poté pomocí metody updateMany aktualizuje záznamy v databázi přidáním ID kategorie k odpovídajícím obrázkům. Úspěšná nebo chybová odpověď je následně vrácena klientovi.

```
const authenticateToken = (req, res, next) => {
  const authHeader = req.headers["authorization"];
  const token = authHeader && authHeader.split(" ")[1];

  if (token == null) {
    console.log("No token provided");
    return res.sendStatus(401); //  401 (Unauthorized)
  }

  jwt.verify(token, process.env.JWT_SECRET, (err, user) => {
    if (err) {
      console.log("Token verification failed:", err);
      return res.sendStatus(403); //403 (Forbidden)
    }
    req.user = user;
    next();
  });
};
```

Zdrojový kód 3: Server - Middleware pro autentizaci (zdroj: vlastní)

Tento kód 3. slouží k ověření autentizačního tokenu, který je součástí HTTP požadavků. Pokud token není poskytnut nebo je neplatný, middleware vrátí chybu 401 (Unauthorized) nebo 403 (Forbidden). Pokud je token platný, middleware extrahuje uživatelské údaje z tokenu a umožní pokračování požadavku k dalšímu zpracování.

Middleware nejprve extrahuje token z hlavičky požadavku, poté jej ověřuje pomocí knihovny JWT (JsonWebToken). V případě úspěšného ověření přidá uživatele do objektu req, což umožňuje přístup k těmto údajům v následujících middlewarech nebo endpointu.

2.3.2 Klíčové části aplikace a jejich implementace

V této části budou představeny konkrétní úseky kódu, které zajišťují klíčové funkcionality aplikace. Každý kódový blok bude následován vysvětlením, jak daný kód přispívá k celkovému fungování aplikace.

```
useEffect(() => {
  let sortedProjects = [...projects];

  if (sortOrder === "nameAsc") {
    sortedProjects.sort((a, b) => a.name?.localeCompare(b.name));
  } else if (sortOrder === "nameDesc") {
    sortedProjects.sort((a, b) => b.name?.localeCompare(a.name));
  } else if (sortOrder === "createdOldest") {
    sortedProjects.sort(
      (a, b) => new Date(a.createdAt) - new Date(b.createdAt)
    );
  } else if (sortOrder === "createdNewest") {
    sortedProjects.sort(
      (a, b) => new Date(b.createdAt) - new Date(a.createdAt)
    );
  }

  const filtered = sortedProjects.filter((project) =>
    project.name?.toLowerCase().includes(searchItem.toLowerCase())
  );

  setFilteredProjects(filtered);
}, [searchItem, sortOrder, projects]);
```

Zdrojový kód 4: Client - Filtrování a řazení projektů (zdroj: vlastní)

Tento kód je součástí frontendové logiky aplikace a zajišťuje filtrování a řazení projektů podle různých kritérií. Kód využívá React hook useEffect, který se spustí při změně některé ze závislostí (v tomto případě searchItem, sortOrder, nebo projects).

- **Řazení projektů:** Podle zvoleného řazení (sortOrder) jsou projekty seřazeny buď podle názvu (vzestupně nebo sestupně), nebo podle data vytvoření (od nejstaršího po nejnovější a naopak). Toto řazení se provádí pomocí JavaScriptové metody sort().
- **Filtrování projektů:** Po seřazení jsou projekty filtrovány na základě hledaného výrazu (searchItem). Filtrování probíhá tak, že se hledaný výraz porovná s názvem každého projektu, přičemž se ignorují rozdíly v malých a velkých písmenech.
- **Aktualizace stavu:** Výsledné seřazené a filtrované projekty jsou uloženy do stavu komponenty pomocí setFilteredProjects(filtered), což způsobí překreslování komponenty a aktualizaci zobrazených projektů.

Kód zajišťuje, že uživatelé mohou efektivně vyhledávat a řadit projekty, čímž se usnadňuje práce s velkým množstvím dat.

```
const Pagination = ({ pageSize, totalImages, paginate, currentPage }) => {
  const pageNumbers = [];

  for (let i = 1; i <= Math.ceil(totalImages / pageSize); i++) {
    pageNumbers.push(i);
  }

  return (
    <nav className="flex justify-center mt-4">
      <ul className="inline-flex items-center -space-x-px">
        {pageNumbers.map((number) => (
          <li key={number}>
            <button
              onClick={() => paginate(number)}
              className={`px-3 py-2 border rounded-md ${
                number === currentPage
                  ? "bg-blue-600 text-white"
                  : "bg-gray-200 text-black"
              }`}
            >
              {number}
            </button>
          </li>
        ))}
      </ul>
    </nav>
  );
};
```

Zdrojový kód 5: Client - Implementace stránkování (Pagination) (zdroj: vlastní)

Komponenta Pagination 5. zajišťuje stránkování dat v aplikaci, což umožňuje uživatelům přehledně procházet velké množství informací po jednotlivých stránkách.

- **Výpočet počtu stránek:** Na základě celkového počtu obrázků (totalImages) a velikosti stránky (pageSize) se vypočítá celkový počet stránek. Následně se vygeneruje pole pageNumbers, které obsahuje čísla všech stránek.
- **Generování tlačítek:** V navigačním prvku <nav> se vytváří seznam (<ul>) tlačítek pro každou stránku. Každé tlačítko (<button>) odpovídá jedné stránce a umožňuje uživateli přechod na ni.
- **Aktuální stránka:** Tlačítko odpovídající aktuální stránce (currentPage) je zvýrazněno jiným barevným schématem pomocí dynamicky nastavované třídy (className), což pomáhá uživateli snadno rozpoznat, na které stránce se právě nachází.
- **Volání funkce paginate:** Kliknutím na tlačítko se volá funkce paginate(number), která zajišťuje přepnutí na zvolenou stránku a aktualizaci zobrazených dat.

Stránkování umožňuje efektivní práci s daty, usnadňuje uživatelům orientaci a zajišťuje přehlednost při práci s rozsáhlými datasetty.

```
<Transition.Root show={active} as={Fragment}>
  <Dialog
    as="div"
    className="relative z-10"
    initialFocus={cancelButtonRef}
    onClose={() => setActive(false)}
  >
    <Transition.Child
      as={Fragment}
      enter="ease-out duration-300"
      enterFrom="opacity-0"
      enterTo="opacity-100"
      leave="ease-in duration-200"
      leaveFrom="opacity-100"
      leaveTo="opacity-0"
    >
      <div className="fixed inset-0 bg-gray-500 bg-opacity-75 transition-opacity" />
    </Transition.Child>

    <div
      className="fixed inset-0 z-10 overflow-y-auto"
      onClick={(e) => e.stopPropagation()}
    >
```

Zdrojový kód 6: Client - Implementace modálního okna (zdroj: vlastní)

Tato část kódu 6. představuje implementaci modálního okna v aplikaci. Modální okno je často používáno pro zobrazení dialogů, formulářů nebo jiných interaktivních prvků, které vyžadují, aby uživatel provedl nějakou akci, než bude moci pokračovat.

- **Transition.Root a Transition.Child:** Tyto komponenty zajišťují plynulé přechody při zobrazení a skrytí modálního okna. Transition.Root obaluje celý modální dialog a určuje, kdy by měl být zobrazen (show={active}). Uvnitř něj Transition.Child definuje animace pro zobrazení a skrytí modálního okna, například změny opacity (průhlednosti) s různou délkou trvání.
- **Dialog:** Tento prvek představuje samotné modální okno. Je zde uvedeno, že dialog by se měl zaměřit na konkrétní prvek (initialFocus) při otevření, a že po zavření modálního okna by se mělo volat setActive(false) pro skrytí dialogu.
- **Vrstva pozadí:** V rámci Transition.Child je definována poloprůhledná vrstva pozadí, která se zobrazí za modálním oknem, aby vizuálně oddělila dialog od zbytku stránky a zaměřila pozornost uživatele na samotné okno.

- **Ochrana proti kliknutí:** Kód také obsahuje funkci onClick={(e) => e.stopPropagation()}, která zajišťuje, že kliknutí uvnitř modálního okna nezpůsobí zavření dialogu kliknutím na pozadí.

Tento kód zajišťuje elegantní a plynulé zobrazení modálního okna, které je uživatelsky přívětivé a funkční.

```
const __filename = fileURLToPath(import.meta.url);
const __dirname = path.dirname(__filename);

const imageDirectory = path.join(__dirname, "public/images");

if (!fs.existsSync(imageDirectory)) {
  fs.mkdirSync(imageDirectory, { recursive: true });
}

const storage = multer.diskStorage({
  destination: (req, file, cb) => {
    cb(null, imageDirectory);
  },
  filename: (req, file, cb) => {
    const uniqueSuffix = Date.now() + "-" + Math.round(Math.random() * 1e9);
    cb(null, `${file.fieldname}-${uniqueSuffix}-${file.originalname}`);
  },
});
const upload = multer({ storage: storage });
```

Zdrojový kód 7: Server - Ukládání nahraných obrázků (zdroj: vlastní)

Tento kód 7. na straně serveru zajišťuje ukládání nahraných obrázků do specifického adresáře pomocí knihovny Multer, která je často používána v Node.js aplikacích pro zpracování multipart/form-data, tedy dat zahrnujících soubory. Nejprve se určuje cesta k adresáři, kam budou obrázky ukládány (imageDirectory). Pokud tento adresář neexistuje, kód jej automaticky vytvoří.

Multer je nakonfigurován pro ukládání souborů na disk pomocí diskStorage. Tato funkce umožňuje specifikovat, kam se soubory uloží a jak se budou jmenovat. V tomto případě se jméno souboru skládá z původního názvu a unikátního příponu (uniqueSuffix), který zajišťuje, že každý soubor bude mít jedinečný název.

Nakonec je Multer inicializován s tímto nastavením úložiště a připraven k použití při zpracování nahraných souborů. Tento kód zajišťuje, že všechny nahrané obrázky jsou správně uloženy na serveru, a to s jedinečnými názvy, což zabraňuje přepisování existujících souborů.

```
const zipFileName = `dataset_${id}.zip`;
const output = fs.createWriteStream(zipFileName);
const archive = archiver("zip", { zlib: { level: 9 } });

output.on("close", () => {
  res.download(zipFileName, (err) => {
    if (err) {
      console.error("Error downloading zip file:", err);
    }
    fs.unlinkSync(zipFileName);
  });
});

archive.on("error", (err) => {
  throw err;
});

archive.pipe(output);
```

Zdrojový kód 8: Server - export datasetu do ZIP archivu (zdroj: vlastní)

Tento kód 8. ukazuje proces exportu datasetu do ZIP archivu. Vytváří název souboru na základě ID datasetu a poté inicializuje stream pro zápis tohoto ZIP souboru. Pomocí knihovny archiver je definována úroveň komprese, kterou bude ZIP archiv používat. Jakmile je export dokončen, soubor je stažen a následně odstraněn z úložiště. Ošetření chyb je zajištěno pomocí několika bloků pro zachycení případných chyb během procesu komprese a stahování. Tento přístup zajišťuje, že soubor je bezpečně vytvořen a zpřístupněn uživateli ke stažení, a zároveň minimalizuje zbytečné využívání diskového prostoru tím, že soubor po stažení odstraní.

```
const metadata = {
  name: dataset.name,
  createdAt: dataset.createdAt,
  project: dataset.project,
  images: dataset.images.map((image) => ({
    name: path.basename(image.url),
    url: image.url,
    format: image.format,
    createdAt: image.createdAt,
    categories: image.categories,
  })),
};
archive.append(JSON.stringify(metadata, null, 2), {
  name: "metadata.json",
});

for (const image of dataset.images) {
  const imagePath = image.url;
  archive.file(imagePath, { name: `images/${path.basename(imagePath)}` });
}

await archive.finalize();
```

Zdrojový kód 9: Server - tvorba metadat a přidání obrázků do ZIP archivu (zdroj: vlastní):

Tento kód 9. je zodpovědný za vytváření metadatového souboru a přidávání obrazových souborů do ZIP archivu během exportu datasetu. Nejprve se vytváří objekt metadata, který obsahuje základní informace o datasetu, jako je jeho název, datum vytvoření, projekt a informace o jednotlivých obrázcích (včetně názvu, formátu, cesty URL, data vytvoření a přiřazených kategorií). Tento objekt je následně serializován do formátu JSON a přidán do ZIP archivu jako soubor metadata.json.

Poté kód prochází všechny obrázky v datasetu a přidává je do archivu. Každý obraz je uložen do složky images/ s odpovídajícím názvem souboru. Na závěr se archiv uzavře voláním metody finalize(), což signalizuje dokončení procesu přidávání souborů do archivu.

```
"name": "Various formats",
"createdAt": "2024-06-28T16:02:32.432Z",
"project": "666329034760be877e37a87e",
"images": [
  {
    "name": "files-1722459384011-581542376-sample_640426.bmp",
    "url": "public/images/files-1722459384011-581542376-sample_640426.bmp",
    "format": "bmp",
    "createdAt": "2024-07-31T20:56:24.036Z",
    "categories": []
  },
```

Zdrojový kód 10: Výsledek - vygenerovaný soubor metadat metadata.json (zdroj: vlastní)

Tento obrázek 10. ukazuje výsledek generovaného metadatového souboru metadata.json, který je součástí exportovaného ZIP archivu. Metadata zahrnují název datasetu, datum jeho vytvoření, identifikátor projektu a seznam obrázků, které jsou součástí datasetu. Každý obrázek má v metadatech uložené informace, jako je název souboru, cesta k obrázku, formát souboru, datum vytvoření a přiřazené kategorie. Tento strukturovaný formát umožňuje snadné zpracování a interpretaci dat, což je užitečné při pozdějším použití nebo analýze datasetu.

```
const directory = await unzipper.Open.file(file.path);
const metadataFile = directory.files.find(
  (d) => d.path === "metadata.json"
);

const metadataStream = await metadataFile.stream();
const metadata = JSON.parse(await streamToString(metadataStream));

const dataset = new Dataset({
  name: metadata.name,
  createdAt: metadata.createdAt,
  project: metadata.project,
});
await dataset.save();
```

Zdrojový kód 11: Server - import datasetu z ZIP archivu (zdroj: vlastní)

Tento kód 11. ukazuje proces importu datasetu z ZIP archivu zpět do systému. Nejprve se otevře ZIP soubor pomocí knihovny unzipper, a poté se vyhledá soubor metadata.json, který obsahuje informace o datasetu. Tento soubor je následně převeden do JSON formátu a zpracován tak, aby bylo možné vytvořit nový dataset v systému. Po zpracování metadat je dataset uložen do databáze. Tento postup umožňuje snadný import a obnovu datasetů včetně všech souvisejících informací.

```
for (const imageMeta of metadata.images) {
  const imageFile = directory.files.find((d) => {
    const fileName = decodeURIComponent(d.path.replace("images/", ""));
    return fileName === imageMeta.name;
  });

  if (!imageFile) {
    console.warn(
      `Image file ${imageMeta.name} not found in the uploaded zip`
    );
    continue;
  }

  const imageStream = await imageFile.stream();
  const imagePath = `public/images/${imageMeta.name}`;
  const writeStream = fs.createWriteStream(imagePath);
  imageStream.pipe(writeStream);
```

Zdrojový kód 12: Server - zpracování obrázků z importovaného datasetu (zdroj: vlastní)

Tento kód 12. pokračuje v procesu importu datasetu z ZIP archivu a zaměřuje se na zpracování jednotlivých obrázků, které jsou součástí datasetu. Pro každý obrázek v metadatech se nejprve vyhledá odpovídající soubor v archivu. Pokud není soubor nalezen, systém zobrazí varování a pokračuje ke zpracování dalších obrázků. Pokud je soubor nalezen, je otevřen jeho stream a vytvořen nový soubor na specifikované cestě v systému. Tento soubor je poté naplněn daty z původního souboru pomocí streamování. Tento postup zajišťuje, že všechny obrázky z datasetu jsou správně extrahovány a uloženy na správné místo v systému.

2.3.3 Řešení problémů při implementaci

Implementace práce s různými obrazovými formáty byla jedním z klíčových aspektů vývoje této aplikace. Zvláštní výzvou se stala práce s DICOM formátem, který je široce používán v lékařské zobrazovací technice. Tento formát je značně odlišný od běžnějších formátů jako JPG nebo PNG, a vyžaduje speciální přístup k manipulaci a zpracování. Při práci s DICOM formátem se objevila řada technických problémů, které zahrnovaly potřebu importu několika specializovaných knihoven a hlubší pochopení struktury tohoto formátu. Řešení těchto

problémů si vyžádalo prohlédnutí mnoha tematických stránek a dokumentací, což nakonec vedlo k úspěšné integraci tohoto formátu do aplikace. Následující sekce poskytne podrobný přehled o tom, jak byly tyto výzvy překonány a jakým způsobem byla implementace DICOM formátu řešena.

```
import * as dicomParser from "dicom-parser";
import cornerstone from "cornerstone-core";
import cornerstoneWADOImageLoader from "cornerstone-wado-image-loader";
import cornerstoneMath from "cornerstone-math";
import cornerstoneTools from "cornerstone-tools";

cornerstoneWADOImageLoader.external.cornerstone = cornerstone;
cornerstoneWADOImageLoader.external.dicomParser = dicomParser;
cornerstoneWADOImageLoader.external.cornerstoneMath = cornerstoneMath;
cornerstoneTools.external.cornerstone = cornerstone;
```

Zdrojový kód 13: Client - import a nastavení knihoven pro práci s DICOM formátem (zdroj: vlastní)

Tento kód 13. ukazuje import nezbytných knihoven pro práci s DICOM formátem v aplikaci. Byly použity knihovny jako dicom-parser, cornerstone-core, cornerstone-wado-image-loader, cornerstone-math a cornerstone-tools [22] [23]. Tyto knihovny jsou klíčové pro zpracování a zobrazování DICOM souborů. Kód dále ukazuje, jak jsou tyto knihovny integrovány do prostředí aplikace pomocí externích referencí, což umožňuje jejich vzájemnou spolupráci při zpracování DICOM dat. Tento postup byl nezbytný pro správné načítání a manipulaci s DICOM obrazy v rámci aplikace.

```
useEffect(() => {
  if (image.format === "dicom") {
    const imageId = `wadouri:http://localhost:8080/${image.url}`;
    const element = document.getElementById(`dicomImage-${image._id}`);
    if (element) {
      cornerstone.enable(element);
      cornerstone
        .loadImage(imageId)
        .then((image) => {
          cornerstone.displayImage(element, image);
        })
        .catch((error) => {
          console.error("Error displaying DICOM image:", error);
        });
    }
  }
}, [image]);
```

Zdrojový kód 14: Client - zpracování a zobrazení DICOM souboru pomocí Cornerstone (zdroj: vlastní)

Tento kódový úryvek 14. ukazuje, jak se zpracovávají a zobrazují DICOM soubory v rámci aplikace pomocí knihovny Cornerstone. Pomocí efektu useEffect se zajišťuje, že při každé změně obrázku formátu DICOM dojde k jeho načtení a zobrazení. Nejdříve se vytvoří identifikátor obrázku na základě URL, následně se pomocí tohoto ID vyhledá HTML element, který bude sloužit pro zobrazení obrázku. Pokud je element nalezen, aktivuje se pro něj Cornerstone a obraz se načte a zobrazí. V případě, že dojde k chybě při načítání nebo zobrazování obrázku, je tato chyba zachycena a vypsána do konzole.

Pokud je obrázek ve formátu DICOM, vytvoří se v uživatelském rozhraní HTML element (div) s unikátním identifikátorem. Tento element slouží jako cíl pro vykreslení DICOM obrázku pomocí knihovny Cornerstone. Element je nastaven tak, aby zabíral 100 % šířky a výšky kontejneru, což zajišťuje, že obrázek bude vykreslen ve správném rozměru.

Tento kód umožňuje správné zpracování a vizualizaci lékařských obrazových dat ve formátu DICOM přímo v prohlížeči.

```
const uploadedImages = await Promise.all(
  req.files.map(async (file, index) => {
    const format =
      file.mimetype.includes("dicom") ||
      file.mimetype === "application/octet-stream"
        ? "dicom"
        : file.mimetype.split("/")[1];
    const name = originalNames[index];
    const image = new Image({
      name,
      url: `public/images/${file.filename}`,
      dataset: datasetId,
      format: format,
    });
    await image.save();
    dataset.images.push(image._id);
    return image;
  })
);
```

Zdrojový kód 15: Server - zpracování a ukládání nahraných souborů s detekcí DICOM formátu (zdroj: vlastní)

Tento kód 15. ukazuje proces zpracování a uložení nahraných souborů na serveru, přičemž se zvláštní pozornost věnuje rozpoznávání DICOM souborů. Každý soubor z nahraných souborů (req.files) je procházen a zpracováván asynchronně pomocí Promise.all, což umožňuje paralelní zpracování všech souborů.

Kód nejprve identifikuje formát souboru na základě jeho MIME typu. Pokud MIME typ obsahuje „dicom“ nebo se rovná application/octet-stream (což je často používáno pro DICOM soubory), formát je označen jako „dicom“. V opačném případě se formát určí z přípony souboru.

Následně je vytvořen nový objekt Image, který obsahuje název, URL, identifikátor datasetu a určený formát souboru. Tento objekt je uložen do databáze a jeho identifikátor je přidán do seznamu obrázků v rámci datasetu. Na závěr je obraz vrácen jako výsledek.

2.3.4 Seznam všech použitých externích knihoven

Tato kapitola obsahuje seznam všech externích knihoven, které byly použity při vývoji aplikace, zahrnující jak backendovou, tak frontendovou část. Každá knihovna je uvedena s krátkým popisem jejího účelu a specifikací, kde byla v rámci projektu použita.

Backend:

- **express:** Použito pro vytvoření a správu serveru. Express je minimalistický webový framework pro Node.js, který poskytuje robustní sadu funkcí pro tvorbu webových a mobilních aplikací.
- **dotenv:** Slouží k načítání environmentálních proměnných z .env souboru do process.env. Tento nástroj je užitečný pro správu konfigurace aplikace, jako jsou hesla, klíče API a další citlivé údaje.
- **cors:** Middleware pro umožnění CORS (Cross-Origin Resource Sharing) v aplikaci, což umožňuje klientským aplikacím přistupovat k prostředkům na jiných doménách.
- **bcrypt:** Použito pro hashování hesel uživatelů, což zajišťuje bezpečné uložení hesel v databázi. Bcrypt poskytuje bezpečný a osvědčený algoritmus pro šifrování hesel.
- **jsonwebtoken:** Použito pro generování a ověřování JWT (JSON Web Token) pro autentizaci a autorizaci uživatelů. JWT umožňuje bezpečnou komunikaci mezi klientem a serverem bez nutnosti uložení uživatelských dat na straně serveru.
- **multer:** Middleware pro zpracování multipart/form-data, což je hlavní formát pro nahrávání souborů v aplikaci. Multer umožňuje snadnou práci s nahranými soubory, jako jsou obrázky nebo dokumenty.

- **path:** Vestavěný modul Node.js, který umožňuje práci s cestami k souborům a adresářům. Používá se pro tvorbu a manipulaci s cestami v rámci souborového systému aplikace.
- **fileURLToPath:** Vestavěná funkce Node.js, která převádí URL (používané například při práci s moduly ECMAScript) na cestu k souboru. Používá se pro určení absolutní cesty k souboru na základě URL.
- **fs:** Vestavěný modul Node.js pro práci se souborovým systémem. Používá se k čtení, zápisu a manipulaci se soubory a adresáři.
- **archiver:** Použito pro vytváření ZIP archivů, které umožňují kompresi a balení souborů a adresářů do jednoho komprimovaného souboru. Tento nástroj byl využit pro export datasetů.
- **unzipper:** Použito pro rozbalování ZIP archivů, což umožňuje snadné rozbalení a zpracování nahraných datasetů. Tato knihovna byla využita pro import datasetů.

Frontend:

- **react:** Knihovna JavaScriptu pro vytváření uživatelských rozhraní. React umožňuje efektivní tvorbu dynamických a interaktivních webových aplikací pomocí komponentového přístupu.
- **react-router-dom:** Použito pro správu směrování v aplikaci. Tato knihovna umožňuje navigaci mezi různými stránkami a komponentami v React aplikaci.
- **axios:** Knihovna pro provádění HTTP požadavků z klientské strany. Axios byl použit pro komunikaci mezi frontendem a backendem aplikace, například pro odesílání a přijímání dat.
- **cornerstone-core:** Knihovna pro zobrazování lékařských obrazových formátů DICOM přímo v prohlížeči. Byla použita ve frontendové části aplikace pro práci s DICOM soubory.
- **cornerstone-tools:** Rozšíření pro Cornerstone, které přidává nástroje pro anotace a manipulaci s DICOM obrázky.
- **dicom-parser:** Knihovna pro analýzu a zpracování DICOM souborů, která byla využita pro čtení metadat a extrakci informací z DICOM souborů ve frontendové části aplikace.

- **react-icons:** Knihovna obsahující sadu populárních ikon, jako jsou HiX a HiCheck, které byly využity pro vizuální zobrazení různých akcí v uživatelském rozhraní.
- **flowbite-react:** Komponenty pro React založené na TailwindCSS, které byly použity pro vytvoření responzivního uživatelského rozhraní, včetně prvků jako jsou dropdown menu.
- **heic2any:** Knihovna umožňující převod obrazových souborů ve formátu HEIC na jiné formáty (například JPG nebo PNG), což zajišťuje kompatibilitu s různými prohlížeči.
- **ReactLoading:** Knihovna pro zobrazení načítacích animací během zpracování požadavků nebo při načítání obsahu.
- **sonner:** Použito pro zobrazování notifikací v uživatelském rozhraní, což zlepšuje interaktivitu a poskytuje uživatelům okamžitou zpětnou vazbu.
- **file-saver:** Knihovna umožňující uživatelům stahovat soubory přímo z prohlížeče, což bylo využito například při exportu datasetů.
- **@headlessui/react:** Knihovna poskytující přístup k předdefinovaným přístupným komponentám Reactu, jako jsou dialogová okna a přechody, které zajišťují responzivní a přístupné uživatelské rozhraní.

ZÁVĚR

Tato práce se zaměřila na vývoj webové aplikace pro správu multimediálních dat, která je určena pro přípravu dat pro trénování neuronových sítí. Hlavním cílem bylo vytvořit systém, který umožní efektivní správu a zpracování obrazových dat, včetně specifických formátů, jako je DICOM, které jsou klíčové pro oblasti jako lékařství.

V teoretické části práce byl poskytnut přehled moderních přístupů k architektuře webových aplikací. Byly rozebrány klíčové koncepty architektur, jako jsou monolitická a mikroservisní architektura, a také důležitost správné implementace API pro efektivní komunikaci mezi systémovými komponentami. Tento teoretický základ poskytl pevný rámec pro následnou praktickou realizaci.

Praktická část se zaměřila na konkrétní implementaci webové aplikace s využitím moderních technologií, jako jsou React a Node.js. Během vývoje bylo nutné řešit různé technické výzvy, včetně integrace a zpracování obrazových formátů, jako je DICOM, což bylo technicky náročné. Nicméně díky zvoleným technologiím se podařilo vytvořit aplikaci, která je uživatelsky přívětivá, výkonná a škálovatelná.

Přínos této práce spočívá především v možnosti praktického využití aplikace pro různé úlohy, jako je analýza a anotace dat pro účely strojového učení. Aplikace je navržena tak, aby byla snadno rozšiřitelná, což umožňuje její přizpůsobení specifickým potřebám v různých odvětvích, jako je zdravotnictví, bezpečnostní systémy a další oblasti vyžadující práci s velkými objemy obrazových dat.

Z hlediska budoucího vývoje nabízí aplikace mnoho možností. Patří sem například rozšíření podpory dalších obrazových formátů a metadat, což by umožnilo obsluhovat širší spektrum uživatelů a aplikací. Další rozvoj může zahrnovat integraci pokročilých algoritmů strojového učení přímo do aplikace, což by umožnilo provádět pokročilé analýzy a predikce přímo v rámci systému.

Závěrem lze říci, že se podařilo úspěšně splnit stanovené požadavky a cíle. Aplikace byla navržena a implementována tak, aby efektivně podporovala správu a zpracování multimediálních dat, což přispívá k její použitelnosti v různých oblastech. Projekt poskytl pevný základ pro další rozvoj a vylepšování, a aplikace splňuje nároky na efektivní správu dat, což umožňuje její praktické využití v širokém spektru aplikací.

POUŽITÁ LITERATURA

[1] Guide to Image Annotation: Techniques, Tools, and Best Practices. *Labellerr.com* [online]. [cit. 2024-08-15]. Dostupné z: https://www.labellerr.com/blog/guide-to-image-annotation-techniques-tools-and-best-practices/

[2] WESTON, Jason; BENGIO, Samy a USUNIER, Nicolas. Large scale image annotation: learning to rank with joint word-image embeddings. Online. *Machine learning*. 2010, roč. 81, č. 1, s. 21-35. ISSN 0885-6125. Dostupné z: https://doi.org/10.1007/s10994-010-5198-3. [cit. 2024-04-18].

[3] Image Annotation: Challenges & Their Solutions. *Labellerr.com* [online]. 2023 [cit. 2024-08-15]. Dostupné z: https://www.labellerr.com/blog/challenges-and-solutions-in-image-annotation/

[4] Image Annotation Tools. *Medium.com* [online]. 2021 [cit. 2024-08-15]. Dostupné z: https://medium.com/data-folks-indonesia/5-best-free-image-annotation-tools-80919a4e49a8

[5] Best Image Annotation Tools for Computer Vision. *Encord.com* [online]. 2024 [cit. 2024-08-15]. Dostupné z: https://encord.com/blog/best-image-annotation-tools/

[6] Web Application Architecture 2024. *Clickittech.com* [online]. 2024 [cit. 2024-08-24]. Dostupné z: https://www.clickittech.com/devops/web-application-architecture/

[7] What Are the Main Types of Web Application Architecture? *Hapy.co* [online]. 2024 [cit. 2024-08-24]. Dostupné z: https://hapy.co/journal/web-application-architecture/

[8] What is an API? *Ibm.com* [online]. 2024 [cit. 2024-08-24]. Dostupné z: https://www.ibm.com/topics/api

[9] React Architecture. *Handsonreact.com* [online]. 2023 [cit. 2024-08-26]. Dostupné z: https://handsonreact.com/docs/architecture

[10] React folder structure. *Create-react-app.dev* [online]. 2020 [cit. 2024-08-15]. Dostupné z: https://create-react-app.dev/docs/folder-structure

[11] What is the Virtual DOM in React? *Freecodecamp.org* [online]. [cit. 2024-08-13]. Dostupné z: https://www.freecodecamp.org/news/what-is-the-virtual-dom-in-react/

[12] React JS Components. *ScholarHat.com* [online]. 2024 [cit. 2024-08-17]. Dostupné z: https://www.scholarhat.com/tutorial/react/react-js-components

[13] React JSX in Depth. *Geeksforgeeks.org* [online]. 2021 [cit. 2024-08-26]. Dostupné z: https://www.geeksforgeeks.org/react-jsx-in-depth/

[14] Best JavaScript Frameworks in 2024 (Extensive Comparison). *Prefender.io* [online]. 2023 [cit. 2024-08-17]. Dostupné z: https://prerender.io/blog/best-javascript-frameworks-pros-cons-and-statistics/

[15] Tailwind CSS: Pros and Cons. *Medium.com* [online]. 2023 [cit. 2024-08-13]. Dostupné z: https://medium.com/readytowork-org/tailwind-css-pros-and-cons-f1a8fdb1fb47

[16] Tailwind CSS under the hood. *Dev.to* [online]. 2023 [cit. 2024-08-17]. Dostupné z: https://dev.to/m4xshen/tailwind-css-under-the-hood-m1o

[17] Utility-First Fundamentals. *Tailwindcss.com* [online]. 2024 [cit. 2024-08-17]. Dostupné z: https://tailwindcss.com/docs/utility-first

[18] Why Visual Studio Code? *Visualstudio.com* [online]. 2023 [cit. 2024-08-20]. Dostupné z: https://code.visualstudio.com/docs/editor/whyvscode

[19]What is Postman? *Postman.com* [online]. 2024 [cit. 2024-08-20]. Dostupné z: https://www.postman.com/product/what-is-postman/

[20]Why Vite? *Vitejs.dev* [online]. 2022 [cit. 2024-08-20]. Dostupné z: https://vitejs.dev/guide/why.html

[21] Libraries You Should Know if You Build with React. *Medium.com* [online]. 2024 [cit. 2024-08-20]. Dostupné z: https://medium.com/@khushi1399gupta/15-libraries-you-should-know-if-you-build-with-react-088d7bb110e8

[22] Cornerstone-tools. *Github.com* [online]. 2021 [cit. 2024-08-24]. Dostupné z: https://github.com/cornerstonejs/cornerstoneTools

[23] Dicom Parser. *Github.com* [online]. 2021 [cit. 2024-08-24]. Dostupné z: https://github.com/cornerstonejs/dicomParser

SEZNAM PŘÍLOH

Příloha A – WEBOVÁ APLIKACE - PicTrain

Archiv PicTrain.zip obsahuje:

1) Zdrojový kód aplikace
2) Grafické uživatelské rozhraní
3) Pokyny pro spuštění webové aplikace.