VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

UŽIVATELSKÉ ROZHRANÍ PRO ACTIVE LEARNING DETEKCE A KLASIFIKACE V OBRAZE

GUI FOR ACTIVE LEARNING OF IMAGE DETECTION AND CLASSIFICATION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **TOMÁŠ BUREŠ**
AUTHOR

VEDOUCÍ PRÁCE **Ing. JAROSLAV ROZMAN, Ph.D.**
SUPERVISOR

BRNO 2020

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav inteligentních systémů (UITS) Akademický rok 2019/2020

Zadání bakalářské práce

Student: **Bureš Tomáš**
Program: Informační technologie
Název: **Uživatelské rozhraní pro Active Learning detekce a klasifikace v obraze**
GUI for Active Learning of Image Detection and Classification
Kategorie: Zpracování obrazu

Zadání:

1. Seznamte se s konceptem aktivního učení v neuronových sítích. Soustřeďte se především na práci s obrazem.
2. Navrhněte uživatelské rozhraní aplikace pro aktivní učení neuronových sítí, vypracujte use-case a diagram tříd pomocí UML.
3. Implementujte jednoduché rozhraní pro výběr dat v rámci aktivního učení.
4. Implementujte možnost složitější anotace dat než je pouze obdélník - polygon, popřípadě pomocí štětce.
5. Aplikaci implementujte po dohodě s vedoucím buď jako webovou aplikaci (HTML, JavaScript/TypeScript, Python), nebo desktopovou aplikaci na bázi Electron frameworku (JavaScript/TypeScript).
6. Rozšiřte Vaši implementaci o výpis a vizualizaci vyhodnocení modelu během aktivního učení.

Literatura:

- Kovashka, Adriana et al. Crowdsourcing in Computer Vision.
- Dudley, John J. a Per Ola Kristensson. A Review of User Interface Design for Interactive Machine Learning.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Rozman Jaroslav, Ing., Ph.D.**
Konzultant: Svoboda Pavel, Ing., Ph.D., UITS FIT VUT
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1. listopadu 2019
Datum odevzdání: 14. května 2020
Datum schválení: 31. října 2019

Abstrakt

Při aktivním učení doménový expert nemusí anotovat veškerá data, ale pouze ta, která umožní model trénovat postupně. Příkladem aktivního učení je například detekce a následné odstranění špatných anotací. Dalším příkladem je detekce a rozšíření trénovacích dat, na kterých model selhává. Součástí práce je popis knihoven, frameworků a programů, které lze pro aktivní učení integrovat. Hlavní částí je návrh a popis uživatelského rozhraní webové aplikace pro aktivní učení. Aplikace umožňuje uživateli prohlížet dataset, řadit anotace a obrázky na základě vícero kritérií a upravovat anotace generované modelem aktivního učení. Grafické uživatelské rozhraní aplikace bylo implementováno s použitím frameworku Vue.js a knihovny Paper.js. V závěru práce je diskutována funkčnost a možnosti budoucícho rozšíření.

Abstract

With active learning, domain expert doesn't need to annotate the whole dataset, but only those which will allow incremental training of a given model. An example of active learning could be detection and removal of wrong annotations. Another example is detection and expansion of training data which model fails to predict. Description of libraries, frameworks and programs which can be used to integrate with active learning is included in this work. The main part of this work is the design and description of a user interface for active learning. The application allows user to browse dataset, sort annotations and images by multiple criteria and modify annotations generated by active learning model. The application's graphical user interface was implemented with the Vue.js framework and Paper.js library. In conclusion, functionality and future application expansion are discussed.

Klíčová slova

webová aplikace, gui pro aktivní učení, COCO dataset, Vue.js, Paper.js

Keywords

web application, gui for active learning, COCO dataset, Vue.js, Paper.js

Citace

BUREŠ, Tomáš. *Uživatelské rozhraní pro Active Learning detekce a klasifikace v obraze.* Brno, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Jaroslav Rozman, Ph.D.

Uživatelské rozhraní pro Active Learning detekce a klasifikace v obraze

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Pavla Svobody, Ph.D. jako konzultanta a Ing. Jaroslava Rozmana, Ph.D. jako vedoucího práce. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Tomáš Bureš
28. května 2020

Poděkování

Rád bych poděkoval Ing. Jaroslavu Rozmanovi, Ph.D. za vedení má bakalářské práce a především mému konzultantovi panu Ing. Pavlu Svobodovi, Ph.D. za cenné rady a věcné připomínky při řešení práce.

Obsah

Kapitola 1

Úvod

Hlavním cílem mé práce je vytvoření aplikace pro prohlížení a úpravy anotací v datasetu. Tato aplikace může sloužit jak pro obyčejné prohlížení datasetu ve formátu COCO, případně výsledků učení modelu pro detekci a klasifikaci v obraze, ale také pro výběr dat a úpravy jednotlivých anotací.

S rozvojem strojového učení se aktivní učení jeví být stále důležitější při trénování detekčních a klasifikačních modelů pro práci s obrazem. Na rozdíl od ostatních metod strojového učení totiž dokáže výrazně snížit množství dat potřebných ke trénování modelu a tím i jeho cenu. Vhodnými nástroji pro použití v kombinaci s aktivním učením tedy jsou aplikace umožňující zobrazovat dataset, vybírat data a anotovat je.

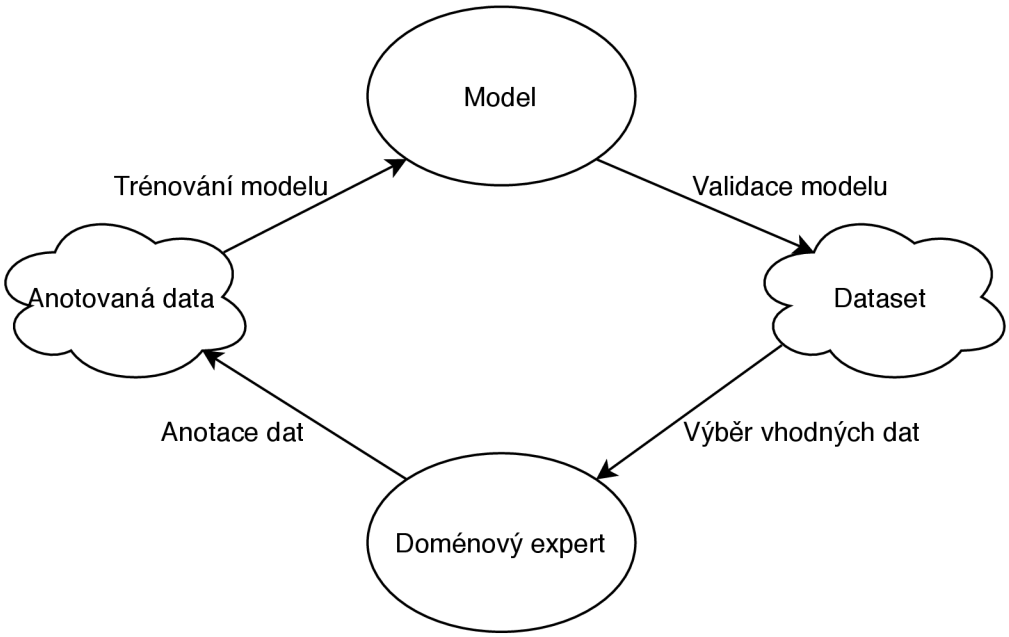
Aplikace poběží v prohlížeči a bude tak dostupná na širší škále zařízení, bez ohledu na operační systém. Umožní uživateli nahrát vlastní dataset upravovat anotace, řadit je a odstraňovat podle potřeby. Následně umožní dataset exportovat do JSON souboru k použití v kombinaci s modelem aktivního učení.

Kapitola 2 popisuje koncept aktivního učení, vysvětluje princip a význam jeho použití. Kapitola 3 se věnuje představení některých v současnosti používanách aplikací a nástrojů pro anotaci dat, jejich srovnání a poukázání na jejich nedostatky. V kapitole 4 jsou vysvětleny některé ze základních pojmů pro vytváření webových stránek, představeny frameworky a knihovny pro implementaci webových uživatelských rozhraní. Dále pak se kapitola zabývá vektorovými grafickými knihovnami, návrhem aplikace a výběrem konkrétních technologií. Nakonec, kapitola 5 přináší přiblížení zajímavých částí implementace uživatelského rozhraní a způsobu uložení dat aplikace.

Kapitola 2

Aktivní učení

Aktivní učení (učení s učitelem) je metoda strojového učení, vyžadující množství anotovaných dat. Model strojového učení v tomto případě podle daných kritérií vybírá data vhodná pro trénování. Aktivní učení je pak iterativní proces, při kterém doménový expert anotuje vhodná data, model se na těchto datech natrénuje, dojde ke zpracování datasetu daným modelem a doménový expert na základě daného kritéria vybere příklady dat, na kterých opraví predikce modelu a vrací se opět na začátek. Tento proces můžeme vidět na obrázku 2.1.



Obrázek 2.1: Cyklus aktivního učení

Takto natrénovaný model je potom schopen podávat lepší výsledky i při učení na menším množství dat. Kritérií pro výběr vhodných dat může být hned několik. Při zpracování datasetu model přiřazuje jednotlivým anotacím tzv. *score* jenž určuje míru s jakou si je model „jistý“ svojí predikcí. Jedno z kritérií výběru dat tedy vybírá právě ty anotace, kdy se *score* nachází blízko hranice potřebné pro správné vyhodnocení anotace. Dalším kritériem pak může být případ anotace u níž si model sice je jistý, ovšem zařadil anotaci do špatné třídy.

Modely strojového učení, převážně pak neuronové sítě, obvykle vyžadují značné množství trénovacích dat pro dosažení co nejlepších výsledků. V mnoha oblastech je získání označených vzorků vhodných k učení modelu snadné, například ve spojení se sociálními sítěmi, kdy uživatelé sami svojí aktivitou poskytují dostatečné množství dat. Anotace v tomto případě mohou být například jednotliví uživatelé označení na fotkách. Na druhou stranu existují oblasti v nichž je získání kvalitních vzorků obtížné nebo velmi drahé. Mezi tyto oblasti patří i *Detekce a klasifikace v obraze*. Zde kromě získání datasetu je třeba aby lidský anotátor vyznačil obrysy každého detekovaného objektu a určil jeho kategorii. Tato činnost je nejen časově, ale i finančně velmi náročná. Aktivní učení tento problém řeší optimalizací učícího procesu dotazováním většinou lidského anotátora na neoznačené vzorky, vybrané tak aby při jejich využití model učinil co největší pokrok [18].

V případě detekce obrazu pojem aktivní učení může také zahrnovat opravy anotací vyhodnocených učícím se modelem. Takovýto proces pak probíhá tak, že model se učí na vhodně vybraných datech a vrací výsledky svého vyhodnocování. Lidský „učitel“ pak může tyto anotace následně upravovat, tak aby více odpovídaly realitě, například přesněji kopírovaly obrys objektu.

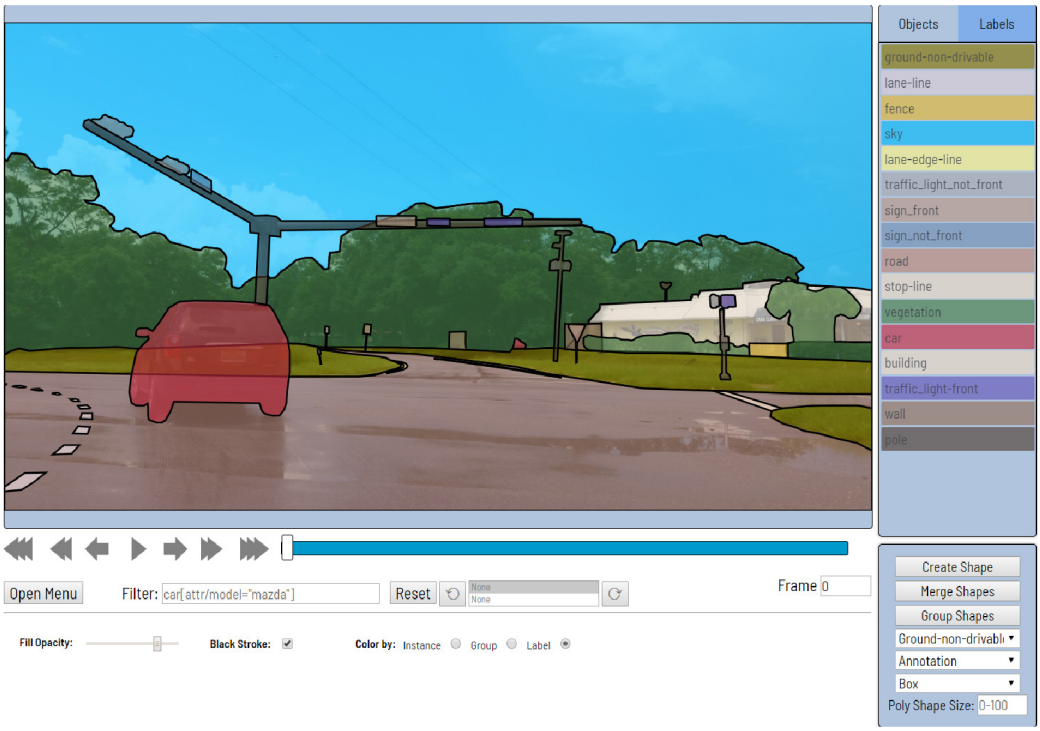
Kapitola 3

Přehled nástrojů pro aktivní učení

Tato kapitola se zaměřuje na existující nástroje pro anotaci obrazových dat. Přibližuje technologie v nichž jsou nástroje implementovány a také zaměření konkrétních nástrojů.

3.1 Computer Vision Annotation Tool (CVAT)

CVAT[1] je online interaktivní nástroj pro anotaci video a obrazových dat poskytovaný zdarma. Nástroj byl vytvořen společností Intel jako webová aplikace zaměřená na anotaci dat určených pro strojové vidění (computer vision). Aplikace jako taková vyla vytvořena v JavaScriptu, HTML, CSS a Pythonu, respektive jeho webového frameworku Django [7].



Obrázek 3.1: Ukázka uživatelského rozhraní aplikace CVAT

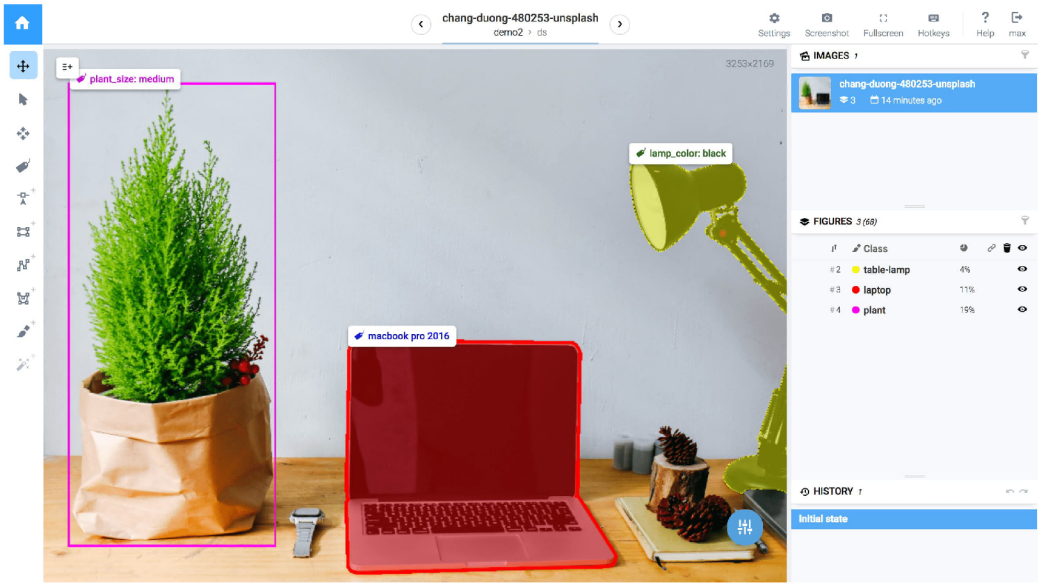
[1] https://github.com/opencv/cvat

Aplikace byla navržena aby uživatelům poskytovala sadu vhodných nástrojů pro anotaci digitálních obrázků a videí. Podporuje mnoho požadavků potřebných pro aktivní strojové učení týkající se detekce objektů, klasifikace obrazu a segmentace obrazu. Umožňuje uživatelům anotovat obrázky čtyřmi typy tvarů: bounding box, polygon, polyline (lomená čára) a bod. Dále pak CVAT má vlastnosti usnadňující typické anotační úlohy, například množství automatických nástrojů, vizuální nastavení, zkratky, filtry a další [7]. Co je také pro anotaci obrázků důležité, umožňuje také přibližovat (zoom).

Umožňuje týmovou spolupráci, stejně jako práci jednotlivců. Uživatel například může vytvořit veřejný úkol a rozdělit práci mezi další uživatele.

3.2 Supervisely

Supervisely[2] je webová aplikace pro anotaci a správu dat vytvořená společností Deep Systems. Aplikace byla implementována s použitím JavaScriptu, HTML a Pythonu. Slouží mimo jiné také pro tvorbu obrazových a video datasetů pro modely strojového učení. Aplikace také poskytuje nástroje pro trénování modelů strojového učení a pokračování v jejich zlepšování za pomocí lidského učitele.



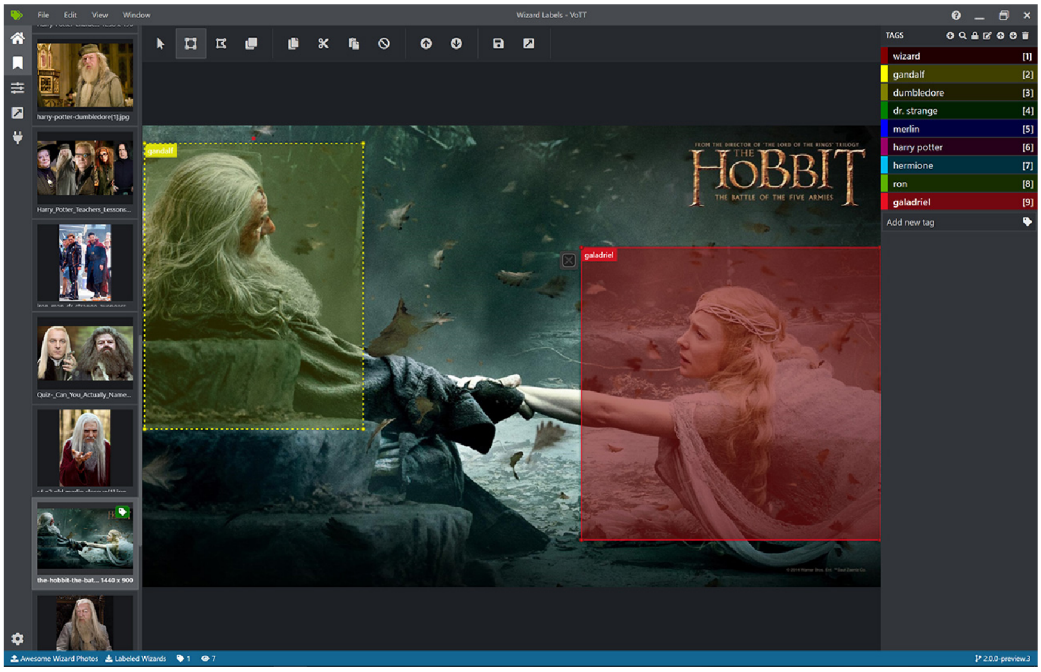
Obrázek 3.2: Ukázka uživatelského rozhraní aplikace Supervisely

Supervisely podporuje celý cyklus výzkumu a vývoje, včetě zmiňovaného anotování dat. Z nástrojů poskytovaných pro anotaci obrazových dat je třeba zmínit ty, využívající tvarů polygon, umožňující definovat i "díry"v objektu. Dále pak bounding box a štětec v kombinaci s gumou (eraser) pro opravy. Pravěpodobně nejvýznamějším nástrojem poskytnutým v rámci aplikace Supervisely je takzvané SmartTool, umožňující anotaci dat za pomocí předtrénovaných modelů neuronových sítí. V takovémto případě stačí za pomocí bounding boxu pouze definovat oblast, ve které si uživatel přeje provést anotaci a aplikace anotaci sama provede. Uživatel má potom možnost tuto anotaci případně opravit nebo upravit [20].

[2]https://supervise.ly/

3.3 Visual Object Tagging Tool (VoTT)

VoTT[3] je open-source aplikace vyvinutá společností Microsoft pro anotaci obrázků a videa, zaměřená primárně na druhé zmíněné. Jako webová aplikace byla implementována v HTML a TypeScriptu za použití knihoven React.js a Redux. V případě její desktopové verze byl pro implementaci navíc použit framework Electron. Výhoda desktopové verze spočívá v možnosti přistupovat do lokálního systému souborů, zatímco u webové verze aplikace data musejí být jedřív nahrána do některého z podporovaných cloudových úložišť. Stejně pak u následného exportu dat aplikace, ve webové verzi musí k exportu dojít přes rozhraní cloudového úložiště.



Obrázek 3.3: Ukázka uživatelského rozhraní aplikace VoTT

I s ohledem na fakt, že VoTT je aplikace primárně určená pro anotaci videa, jediným tvarem pro anotaci je bounding box. Aplikace dále potom umožňuje poloautomatické označování a sledování objektů ve videu za pomocí CAMSHIFT sledovacího algoritmu. Anotovaná data je pak možno exportovat do formátu CNTK (The Microsoft Coginitive Toolkit) pro trénování CNTK detekčních modelů. Jednou z dalších funkcí je také možnost validace natrénovaného CNTK modelu pro detekci na videích pro generování silnějších modelů [22].

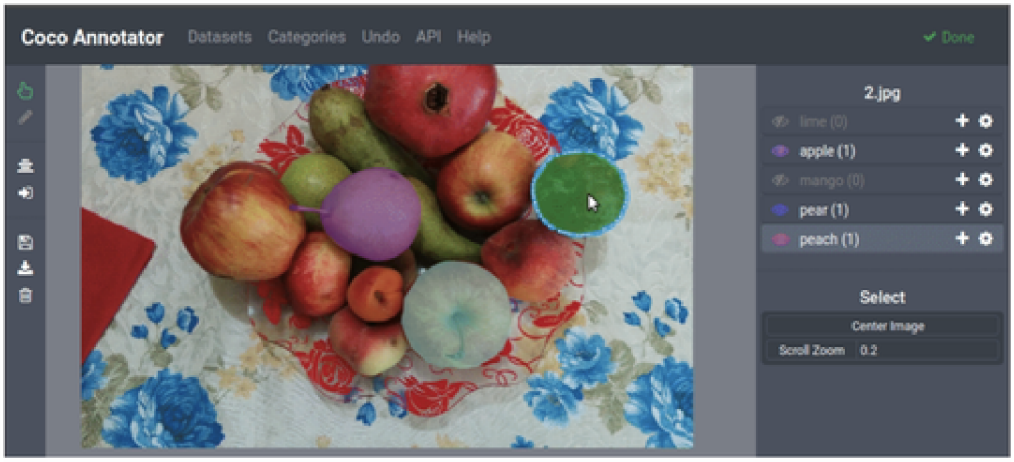
The Microsoft Cognitive Toolkit (CNTK)

The Microsoft Cognitive Toolkit[4] je open-source sada nástrojů pro distribuované hluboké učení v komerční sféře. Popisuje neuronové sítě jako sérii výpočetních kroků přes graf. CNTK umožňuje uživatelům jednoduše kombinovat populární modely strojového učení, jako jsou dopředné hluboké neuronové sítě, konvoluční neuronové sítě a rekurentní neuronové sítě. Implementuje stochastické učení se setupným gradientem s automatickou diferenciací a paralelizací napříč několika grafickými kartami a servery [5].

[3] https://github.com/microsoft/VoTT
[4] https://github.com/microsoft/CNTK

3.4 COCO Annotator

COCO Annotator[5] je webový nástroj pro anotaci obrázků COCO datasetu. Byl navržen pro všestranost a efektivní anotování a popisování obrázků, tvorbu tréningových dat pro obrazovou lokalizaci a detekci objektů. Poskytuje mnoho různých vlastností včetně možnosti označovat jednotlivé segmenty obrazu (nebo části segmentů), sledovat instance objektů s nepropojenými viditelnými částmi a efektivně ukládat a exportovat anotace v COCO formátu [6]. Frontend aplikace byl vytvořen v JavaScriptu s použitím frameworku Vue.js a knihoven Axios, PaperJS a Bootstrap. Backend je implementován v Pythonu a frameworku Flask a databázového systému MongoDB.



Obrázek 3.4: Ukázka uživatelského rozhraní aplikace COCO Annotator

Stejně jako mnoho dalších aplikací podobného zaměření, poskytuje i COCO Annotator mnoho nástrojů pro anotaci dat, mezi ty základní patří nástroj pro výběr a úpravu anotací, polygon, štětec a guma. Dále pak obsahuje pokročilejší nástroje, jako je flood fill (magic wand), což je algoritmus používaný pro výběr oblasti spojené s daným pixelem a expandujícím dále do do větší vzdálenosti od daného pixelu dokud další pixely tuto podobnost nezačnou postrádat. Další pak je nástroj DEXTR (Deep Extreme Cut), využívající extrémních bodů konkrétní anotace (nejvíce vlevo, nejvíce vpravo, nahoře, dole) k dosažení precizní segmentce objektu v obraze. Tohoto docílí přidáním dalšího kanálu do obrázku na vstupu konvoluční neuronové sítě obsahující Gaussovu funkci v každém z extrémních bodů. Konvoluční neuronová síť se pak učí transformovat tuto informaci v segmentaci objektu mezi extrémními body [13]. Nakonec COCO Annotator obsahuje také nástroj pro přímé zapojení učícího se modelu (annotate tool). Tento nástroj umožňuje do procesu anotace zapojit přímo učený (nebo i jakýkoliv jiný) model, vyžádat si oanotování obrázku a urychlit tak výsledný anotační proces.

[5]https://github.com/jsbroks/coco-annotator

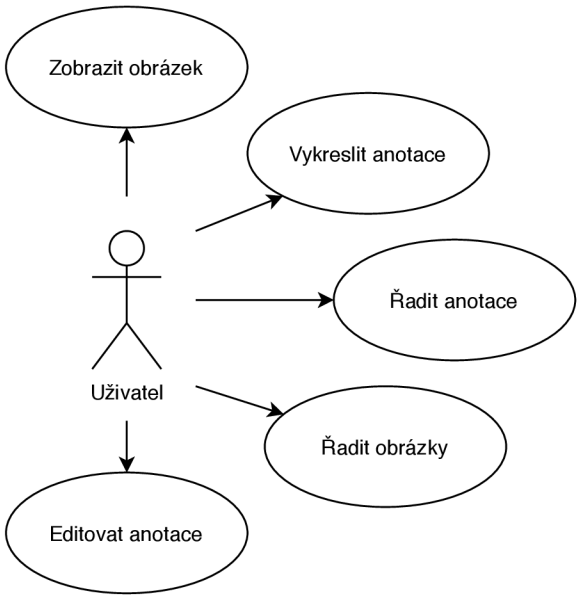
3.5 Srovnání nástrojů

Ačkoliv je v dnešní době na aktivní učení kladen čím dál větší důraz ve smyslu možností dalšího posunu, neexistuje příliš velké množství nástrojů jež by umožňovaly pohodlně anotovat data. Většina v současnosti vyvíjených nástrojů se soustřední na použití vlastních předtrénovaných modelů strojového učení k anotaci vloženého datasetu. Možnost manuálně vybírat a anotovat data je pak jen v podstatě funkce navíc, ne však zamýšlená pro použití v kombinaci s aktivním učením. Zmiňovaným nástrojům často také chybí některé z funkcionalit důležitých pro samotnou anotaci. VoTT například neumožňuje přibližování (zoom) a neobsahuje ani žádaný pokročilejší nástroj pro anotaci. CVAT naproti tomu obsahuje zcela dostatečné množství nástrojů pro anotaci, jeho uživatelské rozhraní je však poměrně komplikované a i relativně jdenoduché úlohy vyžadují od uživatele nepoměrně mnoho práce. Pro někoho taktéž může být nevýhodou, že oficiálně funguje pouze v prohlížeči Google Chrome. Supervisely nechybí žádná důležitá funkcionalita pro anotování dat, dokonce poskytuje velké množství funkcí. Oproti ostatním ovšem nabízí mnoho funkcionalit, jenž pro některé uživatele mohou být zbytečné a aplikace je pak až příliš komplexní. Občas také mívá problémy s výkonností při přepínání mezi obrázky. COCO Annotator splňuje sice požadavky, jeho zásadními nedostatky jsou však časté problémy s výkonem (zamrzání) a také nemožnost použití vlastního datasetu. COCO Annotator totiž dokáže pracovat pouze s obrázky obsaženými v COCO datasetu.

Kapitola 4

Návrh aplikace

Tato kapitola se zabývá technologiemi používanými pro tvorbu webových uživatelských rozhraní, jako jsou jazyky, knihovny a frameworky. Dále pak grafickými knihovnami pro vektorovou grafiku a animace na webu a krom toho také technologiemi používanými pro ukládání dat na straně klienta (ve webovém prolížeči). Není však jejich encyklopedickým výčtem, slouží převážně pro zasvěcení čtenáře do problematiky. Dále je pak kapitola věnována návrhu aplikace, výběru technologií a jejich následného použití při implementaci aplikace.

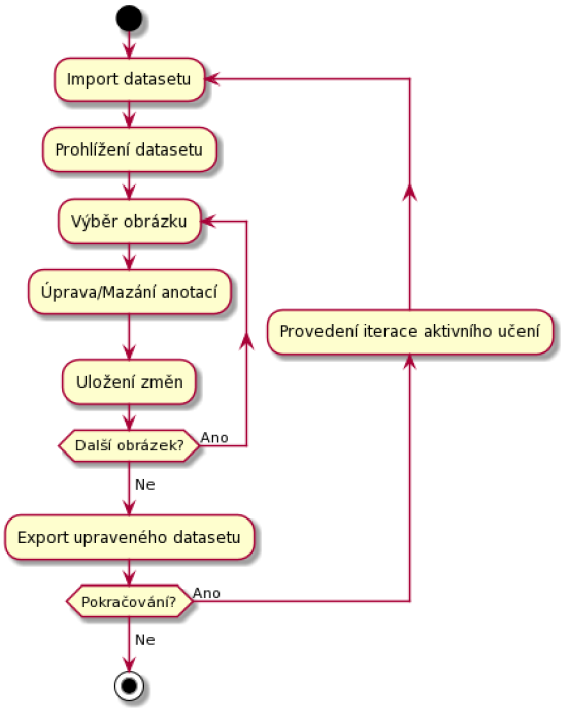


Obrázek 4.1: Use case diagram

V diagramu 4.1 jsou zachyceny základní požadavky kladené na funkčnost výsledné aplikace. Z jednotlivých případů užití je patrné, že mimo samotné vykreslování a uživatelskou interakci bude také nutno zajistit uložení dat, a to jak samotných obrázků, tak informací o jejich anotacích.

Pro správu verzí bude pužit sysém GIT a zdrojový kód aplikace bude veřejně dotupný na serveru GitHub[1].

[1] https://github.com/tomas-bures/active-learning-gui



Obrázek 4.2: Diagram použití aplikace

V diagramu 4.2 lze vidět typický způsob použití aplikace v kombinaci s modelem aktivního učení. Z diagramu je patrné, že před samotným použitím je požadován import dat vygenerovaných modelem. Následně uživatel provede potřebné úpravy, data opět exportuje a předloží modelu k provedení další iterace aktivního učení.

4.1 Přehled vhodných knihoven a frameworků

Pro definici vzhledu webových stránek se dnes používá HTML ve spojení s CSS a JavaScript obstarává funkcionalitu, animace a další. Tato kapitola popisuje technologie pro tvorbu webových aplikací, přesněji pak jejich uživatelských rozhraní. Dále se potom věnuje grafickým javascriptovým knihovnám a frameworkům používaných pro tvorbu webových uživatelských rozhraní.

Základním stavebním prvkem webových stránek je jazyk HTML. Jazyk využívá tzv. značek pro definici struktury a obsahu stránky, jež vzniká skládáním jednotlivých elementů, jimž lze definovat další atributy. Webový prohlížeč pak HTML zpracovává a skládá z něj DOM (Document Object Model).

DOM je objektově orientovaná reprezentace XML nebo HTML dokumentu. Je to rozhraní umožňující přístup či modifikaci obsahu, struktury nebo stylu dokumentu či jeho částí. DOM reprezentuje stránku způsobem, jakým programy mohou měnit strukturu dokumentu, jeho styl, nebo obsah. Ve většině moderních webových prohlížečů je implementován standard W3C DOM nebo WHATWG DOM, což umožňuje výrazně omezit HTML kód specifický pro jednotlivé prohlížeče. Na druhou stranu však mnoho těchto prohlížečů standard dále rozšiřuje, takže tyto standardy musejí být brány s jistou rezervou [8].

Další jaykem, bez nějž by moderní webové stránky nemohly fungovat je JavaScript, což je multiplatformní, objektově orientovaný, událostmi řízený a dynamicky typovaný programovací jazyk nejvíce známý pro svoje využití při tvorbě skriptů webových stránek. Je ovšem používán také v dalších prostředích, jako je Node.js (JavaScript na serveru) nebo Adobe Acrobat. Na webu primárně slouží k interakci s uživatelem a stává se tak hlavním nástrojem pro vylepšení uživatelského zážitku. Vzhledem k jeho značné polularitě a rychlému vývoji vznikají také další jazyky, které z něj přímo vycházejí a dále ho vylepšují, například TypeScript.

TypeScript je *open-source* programovací jazyk, fungující jako nadstavba na Javascriptem, který rozšiřuje o statické typování, třídy, moduly a další atributy známé z objektově orientovaného programování. Při jeho použití se používá tzv. *transpiler* pro překlad TypeScriptu do čistého Javascriptu.

Pro implementaci uživatelských rozhraní webových aplikací je v současné době pravděpodobně nejvhodnějším jazykem Javascript. Dnes existuje široká škála knihoven a frameworků, jenž usnadňují použití Javascriptu právě pro tyto účely. Tato podkapitola pak popisuje některé z nejpoužívanějších knihoven a frameworků pro tvorbu grafických uživatelských rozhraní v Javascriptu.

Základním rozdílem mezi knihovnou a frameworkem je ten, že knihovna je souborem funkcionalit obsažených v balení, jenž programátor může dle libosti, buď jako celek nebo pouze jeho části, používat ke zjednodušení některých činností bez nutnosti znovu implementovat funkcionalitu v knihovně obsaženou. Bavíme-li se o frameworku, pak máme na mysli jakousi šablonu, v rámci níž je výsledná aplikace implementována. Taková aplikace je potom většinou svoji architekturou přímo podřízena architektuře frameworku. Framework jako takový pak může být souborem knihoven a dalších funkcionalit jenž tvoří funkční celek, přičemž programátor úpravami částí toho celku vyvíjí výslednou aplikaci.

React.js

React.js v současnosti udává trend mezi knihovnami pro tvorbu grafických uživatelských rozhraní. Princip staví na komponentovém přístupu, což jsou znovupoužitelné HTML elementy se zapouzdřenou funkcionalitou spravující vlastní vnitřní stav. Skládáním těchto komponent vzniká komplexní uživatelské rozhraní.

Vzhledem k tomu, že React.js reprezentuje pouze *View* v architektuře MVC (Model View Controller), používá se často v kombinaci s dalšími knihovnami zajišťujícími správu globálního stavu nebo dalšími pro vykreslování na straně serveru (Server-side rendering). Vykreslování na straně serveru spočívá v generování stránky na serveru a jejího odeslání v odpovědi na požadavek. Tento přístup také zajišťuje, že stránka je přístupnější pro vyhledávače. Při běžném chování totiž React.js vykresluje každou stránku až na straně klienta.

Knihovna React.js byla představena společností Facebook a je vyvíjena jako *open-source* projekt.

Virtual DOM

React.js, ale i jiné knihovny, či frameworky používají tzv. virtuální DOM (Document Object Model). Klasický DOM je možno dynamicky upravovat a měnit tím vzhled stránky bez nutnosti jejího znovunačtení. Tyto úpravy jsou však náročné na výkon a u složitějších uživatelských rozhraní s vysokým počtem elementů mohou tyto úpravy vést ke zpomalení celé stránky [2]. Na každou takovou úpravu jsou napojeny interní činnosti prohlížeče

jako přepočet CSS pravidel a úprava souřadnic elementů, což je hlavním důvodem tohoto zpomalení [27].

Virtual DOM je kopií klasického DOMu, uloženou v paměti. React.js díky této kopii může pomocí pokročilých algoritmů porovnávat Virtual DOM s reálným DOMem a zajistit tak minimální potřebné množství úprav v reálném DOMu [2]. Kromě zvýšení rychlosti, Virtual DOM umožňuje deklarativní přístup React.js knihovny. Stačí tedy definovat v jakém stavu reálný DOM chceme a React.js zajistí zbytek [21].

Angular

Angular je framework implementovaný v TypeScriptu společností Google. Stejně jako knihovna React.js staví na komponentovém přístupu, na rozdíl od něj však v architektuře MVC zastává všechny tři jeho složky. Byl navržen tak, aby umožňoval oddělovat zobrazovací logiku od aplikační logiky. Základem každé komponenty je programová třída, framework pak přímo staví na základech objektově orientovaného programování (dále jen OOP). K této třídě pak naváže vlastní HTML šablonu a dokonce i CSS pomocí tzv. direktiv. Tímto způsobem umožňuje rozdělit webovou stránku na samostatné celky a poskládat ji pomocí principů OOP. Funkcionalitu komponent nepřímo související s *View* zajišťují *služby*, jenž mohou být vloženy do komponenty jako závislosti (Dependency injection). To také zajišťuje modularitu a znovupoužitelnost kódu [1]. Angular pro detekci změn používá tzv. dirty-checking, který kontroluje zda se nová hodnota proměnné liší od staré. Pokud ke změně došlo, Angular vyhodnotí, zda je skutečně potřeba změnit DOM a případně tuto změnu provede. Pro předávání dat z části Model do View je použit tzv. *two-way data binding*. Což zajišťuje, že když se změní hodnota v Model, změní se i ve View a naopak. Narozdíl od *one-way data binding*, které zajišťuje pouze propagaci změnu v Model do View.

Vue.js

Vue.js je označován za tzv. progresivní webový framework, což znamená, že umožňuje webovou aplikaci rozdělit na části a ty pak vyvíjet nezávisle na sobě. Stejně jako Angular i Vue.js reprezentuje všechny tři části architektury MVC a stejně jako obě výše zmíněné technologie, staví na komponentovém přístupu, kdy se výsledné grafické uživatelské rozhraní sestavuje skládáním jednotlivých komponent. Podobně jako React.js taktéž využívá Virtual DOM popsaný výše v této kapitole. I přesto, že se Vue.js zaměřuje primárně na grafické elementy stránky, pokročilé funkce jako je routování, správa stavu nebo nástroje pro výsledné sestavování stránek jsou dostupné skrze oficiálně podporované a udržované podpůrné knihovny a balíčky [23]. Framework jako takový pak také umožňuje vykreslování na straně serveru pro lepší indexování stránek ve vyhledávačích [17]. Jistou zajímavostí je pak také fakt, že Vue.js lze použít i jako knihovnu nalinkováním do HTML souboru. Přičemž v takovémto případě je možné vyvíjet webovou stránku bez frameworku a Vue.js použít jen pro vytvoření určitých komponent.

Vue.js bylo vytvořeno bývalým programátorem firmy Google jménem Evan You a představeno společností Vue. V současnosti je dále udržováno a vyvíjeno jeho tvůrcem a dalšími členy týmu z různých společností jako je Netlify nebo Netguru.

Polymer.js

Polymer.js je javascriptová knihovna sloužící podobně jako React.js ke tvorbě nezávislých komponent s vlastním vnitřním stavem a logikou. Na rozdíl od výše zmíněných technologií

však využívá Web Components API (Application Programming Interface). V MVC architektuře reprezentuje View, podobně jako React.js. Z tohoto vyplývá nutnost kombinovat knihovnu s dalšími frameworky či knihovnami. Jistou výhodou je možnost kombinovat Polymer.js pro vytváření komponent s dalšími frameworky či knihovnami jako je Angular pro zajištění další funkcionality [15].

Polymer.js byl stejně jako Angular představen společností Google.

Web Components

Web Components je soubor různých technologií umožňující tvorbu vlastních znovupoužitelných elementů se zapouzdřenou funkcionalitou. Hlavní myšlenkou je možnost znovupoužívání HTML elementů napříč různými webovými stránkami. Tohoto je dosaženo za použití tří hlavních technologií: vlastní elementy (Custom elements), Shadow DOM a HTML šablony (HTML templates). Základní přístup obvykle pro tvorbu Web Components vypadá takto [24]:

1. Vytvoření třídy nebo funkce ve které je specifikována funkcionalita
2. Registrace nového elementu
3. Připojení Shadow DOM k elementu. Přidání dalších komponent jež jsou součástí právě vytvářené (child components), naslouchání událostem (event listeners) a dalších do Shadow DOM za použití stejných metod jako při použití klasického DOMu
4. Je-li to vyžadováno, definice HTML šablony
5. Využití vlastního elementu kdekoliv na stránce stejným způsobem jako kterýkoliv jiný HTML element

Hlavním rozdílem mezi Web Components a komponentami ve výše zmíněných frameworcích a knihovnách je ten, že Web Components jsou součástí standardu a jako takové jsou přímo implementovány v prohlížečích, zatímco komponenty ve Vue.js, Angularu nebo React.js je třeba při sestavování aplikace *přeložit* do standardních elementů HTML.

Shadow DOM

Shadow DOM je sada javascriptových API pro připojení zapouzdřeného stromu DOMu k elementu, jenž je vykreslen odděleně od hlavního dokumentového DOMu, a ovládání jeho funkcionality. Tímto způsobem mohou být vlastnosti elementu na venek skryté, takže mohou být implementovány a stylovány bez obav z kolize s dalšími částmi dokumentu [24].

Electron

Electron je na rozdíl od výše zmiňovaných frameworkem pro tvorbu nativních desktopových aplikací, ovšem využívá k tomu webových technologiií, jako jsou JavaScript, HTML a CSS. Jedná se o open-source framework vyvinutý společností GitHub. Pro svoji funkčnost používá *Chromium rendering engine* (vykreslovací engine prohlížeče Google Chrome) a Node.js běhové prostředí a byl napsán za použití jazyků C++, JavaScript, Objective-C++, Python a Objective-C. Aplikace mohou být vyvíjeny multiplatformně pro operační systémy Windows, Linux a macOS.

Electron aplikace sestávají z hlavního procesu, jenž obstarává aplikační logiku a může spouštět vykreslovací procesy, které pak vykreslují grafické uživatellské rozhraní jako webovou stránku za pomocí již zmíněného *Chromium* vykreslovacího enginu. Každá takováto stránka má svůj vlastní vykreslovací proces. Na rozdíl od webových stránek fungujících ve webových prohlížečích, však stránky v Electron aplikacích mohou přes Node.js API přistupovat k lokálním zdrojům, například lokálnímu systému souborů [9]. Toto je pak jejich hlavní výhoda oproti klasickým webovým applikacím. Na druhou stranu ovšem spolu s klasickými webovými stránkami sdílí jejich hlavní omezení, zvláště co se týká výkonu a jejich zranitelnosti vůči potenciálním útokům.

Paper.js

Paper.js je open-source vektorová grafická knihovna implementovaná v JavaScriptu a bežící nad HTML5 canvasem. Byla vytvořena programátory Juerg Lehni a Jonathan Puckey v roce 2011. Knihovna používá tzv. graf scény (scene graph) neboli vlastní variantu DOMu a nabízí mnoho funkcionalit k tvorbě a práci s vektorovou grafikou a beziérovými křivkami. Manipulace a vykreslování grafických objektů je automatické a optimalizované, což umožňuje konstruovat a modifikovat objekty a styly zatímco vnitřní funkcionalitu obstarává Paper.js. Psaní kódu je možné přímo v JavaScriptu nebo lze využít tzv. PaperSript, jednoduché rozšíření JavaScriptu, umožňující vykonávání skriptů odděleně od globálního *scope*. Vícero skriptů pak může současně běžet na stránce, přičemž každý pak má vlastní *scope*, zatímco sdílí kód knihovny.

P5.js

P5.js je javascriptová open-source knihovna pro kreativní grafické programování vyvinutá prográmatorem Lauren McCarthy v roce 2013. Ačkoliv je založená na základě technologie Processing, nejedná se jen o její emulaci nebo port. Stejně jako Paper.js používá HTML5 canvas, ovšem na rozdíl od Paper.js, se neomezuje pouze na něj. Díky dalším přidaným knihovnám může P5.js interagovat i s dalšími HTML objeky, jako jsou texty, vstupy, video, webkamera a zvuky [14]. Na rozdíl od již zmiňovaného Paper.js aplikace využívající P5.js mají zcela odlišnou architekturu. Zatímco v Paper.js je možné každý objekt definovat zvlášt, držet v paměti a dále ho modifikovat a interagovat s ním (každý objekt má například možnost definovat vlastní funkce pro interakci s myší) v P5.js je vše jeden tzv. *sketch* do nějž se definuje veškerá funkcionalita a vzhled. Uživatelská interakce s jednotlivými prvky je tak složitější než v Paper.js.

Processing

Processing je open-source grafická knihovna pro tvorbu a učení programování skrze vizuální kontext vytvořená programátory Casey Reas a Ben Fry v roce 2001. Využívá programovací jazyk Java s některými zjednodušeními a naopak také s přidanými funkcemi pro matematické operace. Vznikly také další jeho implementace, či emulace v jiných jazycích jako je Processing.js (JavaScript), Processing.py (Python), P5.js a další.

Fabric.js

Fabric.js je open-source javascriptová grafická knihovna využívající HTML5 canvas, vytvořená programátory Juriy Zaytsev, Stefan Kienzle a Andrea Bogazzi v roce 2010. Podobně

jako Paper.js využívá objektový model (varianta DOMu) a obsahuje také SVG parser, umožňující převod SVG do canvas a naopak. Podobně jako Paper.js také umoňuje samostatnou manipulaci každého objektu uvnitř canvasu.

Three.js

Three.js je open-source javascriptová knihovna a API pro tvorbu a zobrazování 3D počítačové grafiky ve webových prohlížečích. Knihovna byla poprvé představena v roce 2010 programátorem Ricardo Cabello. Na rozdíl od výše zmíněných knihoven, Three.js využívá k vykreslování obsahu místo canvasu WebGL. Toto umožňuje v knihovně vytvářet komplexní 3D grafické prvky a animace s vyšším výkonem a menším úsilím než v canvasu. Novější verze knihovny ovšem podporují také 2D canvas, SVG a CSS3D vykreslování.

WebGL

WebGL (Web Graphics Library) je javascriptové API pro vykreslování 2D a 3D grafiky v jakémkoliv kompatibilním prohlížeči bez použití doplňků (plugin). Dosahuje toho představením API, jenž je přizpůsobeno OpenGL ES 2.0, které může být použito uvnitř HTML5 *canvas* elementů. Toto přizpůsobení zajišťuje, že API může využít hardwarovou grafickou akceleraci poskytnutou zařízením uživatele [26].

Pixi.js

Pixi.js je javascriptová 2D knihovna starající se o vykreslování obsahu za pomocí WebGL, podobně jako Three.js. Hlavním cílem Pixi.js je poskytovat využití hardwarové akcelerace k vykreslování animací i bez znalosti samotného WebGL. Pomáhá zobrazovat, animovat a spravovat interaktivní grafické prvky (sprity) pro relativně snadné vytváření her, ale i dalších druhů aplikací. Podobně jako další výše zmíněné knihovny využívá graf scény (vlastní verze DOM), umožňující vytváření kompletních hierarchiií zanořených spritů, stejně jako připojení událostí myši přímo ke spritům.

4.2 Technologie pro ukládání dat na straně klienta

Moderní webové prohlížeče podporují relativně velké množství způsobů jak webové stránky mohou ukládat potřebná data do počítače uživatele. Toto jim umožňuje tyto data dlouhodobě uchovávat pro uložení stránek nebo dokumentů pro offline použití, uložení specifických nastavení uživatele a dalších [4].

Cookies

Jako cookie se v protokolu HTTP uznačuje malé mnnožství dat, jenž server odesílá prohlížeči, který je uloží na počítači uživatele. Při každé další návštěvě téhož serveru pak prohlížeč tato data posílá zpět serveru. Typicky se používá k určení zda požadavek přišel ze stejného prohlížeče, takže například uživatel může zůstat přihlášený i po opuštění stránky. Slouží k zapamatování stavu v bezestavovém protokolu HTTP.

Jejich tři hlavní důvody použití jsou:

- Správa sezení (session) - uložení přihlášní uživatele, nákupního košíku, score her, atd.
- Perzonalizace - uživatelská nastavení, témata (themes)

- Sledování - zanamenávání a analýza chování uživatele

Dříve cookies sloužily jako jediná možnost uložení dat na straně klienta. Dnes však jejich význam upadá a kvůli malé kapacitě a faktu, že jsou zasílány s každým požadavkem, jsou postupně nahrazovány modernějšími způsoby uložní dat.

HTTP

HTTP je internetový protokol, využívaný při komunikaci mezi klientem a serverem. Komunikace pomocí HTTP protokolu se skládá z požadavku a příslušné odpovědi. Obvykle, ne však vždy, probíhá komunikace na portu 80. Protokol využívá TCP, zajišťující spolehlivé doručení [2]. V dnešní době se stává téměř samozřejmostí použití zabezpečené varianty HTTPS. Data odeslaná pomocí protokolu HTTPS jsou zabezpečena prostřednictvím protokolu TLS (Transport Layer Security), jenž typicky poskytuje autentizaci serveru pomocí certifikátu, který musí být vydán ověřenou certifikační autoritou [11]. Protokol TLS pak zajišťuje šifrování dat pro ochranu před odposlechem a integritu dat - nelze data během přenosu změni ani poškodit, ani by to nebylo zjištěno a ověření s použitím již zmíněných certifikátů.

Web Storage

Web Storage rozhraní poskytuje mechanismy, kterými mohou prohlížeče ukládat data ve formátu klíč/hodnota. Umožňuje s pomocí jednoduché syntaxe ukládat malé datové položky sestávající z jména (klíče) a související hodnoty. Hodí se převážně pro uložení jednoduchých dat jako například jméno uživatele, informace zda-li je přihlášen, jakou barvu použít v konkrétním stylu a podobně [25]. V tomto směru tedy dokáže plně nahradit již zmíněné cookies. Nehodí se však příliš pro ukládání velkého množství dat. Tento problém řeší IndexedDB.

Samotné Web Storage pak používá dva mechanizmy ukládání dat:

- sessionStorage - udržuje data po dobu celého sezení (session), včetně znovunačtení a obnovení stránky. Tyto data jsou smazána, dojde-li k vypnutí prohlížeče nebo zavření záložky. Výhodou je, že data se nezasílají na server a kapacita pro uložení, větší než u cookie, může být až 5MB.

- localStorage - ukládá data stejným způsobem, ovšem tyto zůstávají zachována i po zavření prohlížeče. Jediným způsobem jak je smazat je skrze JavaScript nebo vymazáním cache prohlížeče/lokálně uložených dat [25]. Maximální kapacita takového úložiště je závislá na konkrétním webovém prohlížeči.

IndexedDB

IndexedDB je nízkoúrovňové rozhraní pro úložiště na straně klienta, umožňující ukládat značné množství strukturovaných dat včetně souborů. Toto rozhraní užívá indexování pro dosažení vysoké rychlosti vyhledávání v datech. IndexedDB je transakční databázový systém podobný SQL databázovým systémům. Narozdíl od SQL databázových systémů využívajících tabulky s pevně danými sloupci, IndexedDB je javascriptová objektově orientovaná databáze. IndexedDB umožňuje ukládání a získávání objektů, indexovaných jejich klíčem, jež jsou podporovány tzv. *structured clone algorithm*. Stačí jen specifikovat schéma databáze, otevřít spojení s databází a následně je možno vkládat, upravovat nebo mazat data v sérii transakcí. Operace spojené s IndexedDB probíhají asynchroně, takže nedochází k

blokování běhu aplikací. I přesto, že IndexedDB původně obsahovala jak synchronní tak asynchronní API, synchronní bylo ze specifikace odstraněno, jelikož nebylo jisté, zda-li je vůbec potřeba [12].

I když je IndexedDB API mocným nástrojem pro ukládání dat, v mnoha případech je jeho použití relativně komplikované. Proto se většinou nepoužívá ve své základní formě, ale ve formě knihoven jako je localForage, Dexie.js nebo PouchDB, jenž činí používání IndexedDB výrazně přívětivější k programátorům.

Ačkoliv je možné v IndexedDB ukládání relativně velkého množství dat, je třeba v tomto směru postupovat opatrně a mít na paměti, že kvóty pro maximální velikost databáze se liší v každém prohlížeči a závisí na velikosti úložiště na němž je prohlížeč nainstalovaný.

- Google Chrome - umožňuje použít až 60% kapacity disku [3], přičemž jakmile dojde místo, Chrome smaže databázi vybranou LRU (Least Recently Used) algoritmu [16].
- Mozilla Firefox - celkem umožňuje použít až 50% kapacity disku, ovšem s tím, že pro jednu databázi je možno použít maximálně 20% z tohoto místa. Tedy pro jednu databázi 10% kapacity disku, ovšem maximálně 2GB [10]. Když místo dojde, stejně jako Chrome smaže databázi vybranou LRU [16].

Structured clone algorithm

Structured clone algorithm kopíruje komplexní javascriptové objekty. Je používán interně k přenosu dat mezi tzv. *Workers* (Web Worker), ukládání objektů do IndexedDB nebo kopírování objektů do jiných API. Klonuje rekurzivním procházením vstupním objektem zatímco udržuje *mapu* navštívených referencí aby zabránil nekonečnému cyklickému procházení [19].

Structured clone algorithm nepodporuje:

- Funkce - pokus o duplikaci vyvolá výjimku
- DOM uzly - pokus o duplikaci vyvolá výjimku
- Property descriptors - gettery, settery, a další nebudou duplikovány
- Řetěz prototypů (Prototype chain) nebude duplikován

Úložiště pro dataset

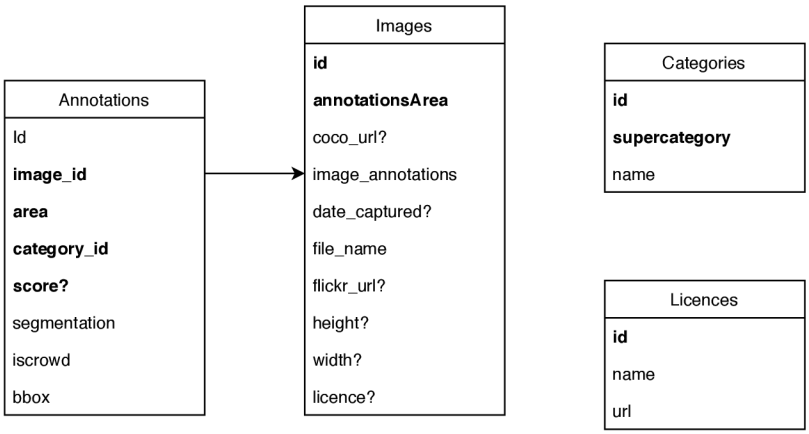
Vzhledem k tomu, že aplikace implementovaná v rámci této práce musí být schopna zobrazovat a dále manipulovat s množstvím obrázků různých formátů a velikostí, je třeba zvolit vhodné úložiště pro jejich uložení v rámci aplikace, respektive tak, aby k nim aplikace měla přístup. Implementovaná aplikace poběží na webu, je tedy zřejmé, že nebude mít přístup k lokálnímu souborovému systému na počítači klienta. Vzniká tak potřeba využití úložiště na nějž je možno nahrávat vlastní obrázky datasetu a dále k nim pak přistupovat.

Způsobů jak toto řešit může být mnoho. Může se jednat o vlastní databázi implementovanou na backendu aplikace, nebo o použití obrázků veřejně dostupných na webu. Potenciálně by také bylo možno použít technologii IndexedDB popisovanou v kapitole 4.2, ovšem v takovém případě je třeba počítat s omezeními z toho plynoucími. Z důvodu jednoduchosti použití, stejně tak jako rychlosti s jakou je možno tuto technologii nasadit byla pro úložiště obrázků zvolena služba Firebase Storage[2]. Služba poskytuje cloudové úložiště souborů, ve

[2] https://firebase.google.com/

verzi zdarma, až do celkové velikosti 1GB. Toto omezení je sice pro potřeby aplikace jako takové nedostačující, ovšem pro potřeby vývoje demostraci funkcí postačí.

Funkcionalita aplikace je ovšem založena především na datech o anotacích, jenž do aplikace musí uživatel nahrát. Jako vstup v tomto smyslu slouží JSON soubor obsahující potřebné informace o datasetu, především o obrázcích a anotacích. Vzhledem k tomu, že anotací i obrázků mohou v datasetu být řádově desítky až stovky tisíc, je třeba vhodným způsobem uložit informace obsažené v daném JSON souboru. Jedním ze způsobů uložení daného souboru by mohla být opět zvolena služba Firebase Storage. Problém s tímto způsobem řešení je však fakt, že aplikace by při každém spuštění musela soubor načíst a držet v paměti, což by vzhledem k potenciální velikosti souboru mohlo vést ke špatnému uživatelskému zážitku. Držet obsah souboru v paměti (proměnné) by pak dále negativně ovlivňovalo výkonnost aplikace, nemluvě o faktu, že data ze souboru musejí být přístupná napříč vícero komponentami aplikace, což vzhledem k využití frameworku Vue.js přináší nutnost použít k tomuto účelu globální úložiště stavu. Pokusy o seřazení i velmi malých datasetů ovšem oproti stejné akci prováděné v lokální proměnné, přináší zpomalení až 40x. Z tohoto důvodu jsem tento způsob řešení zavrhl. Naopak jako velmi rychlé a efektivní řešení se ukazuje použití technologie IndexedDB popsané v kapitole 4.2. S ohledem na jistou komplikovanost API, pro zprostředkování práce s IndexedDB byla zvolena knihovna Dexie.js[3].



Obrázek 4.3: Schéma databáze

V obrázku 4.3 je možno vidět schéma implementované databáze. Jak již bylo popsáno výše, nejedná se o relační databázi, takže mezi tabulkami nejsou žádné vazby. Vazba mezi tabulkou *Annotations* a *Images* je zde znázorněna pouze k lepšímu pochopení, protože tabulka *Images* ve sloupci *image_annotations* obsahuje kopie řádků z tabulky *Annotations* náležící k danému obrázku dle jeho *id*. Tučně vyznačené sloupce u jednotlivých tabulek označují indexované položky. A položky s otazníkem pak značí nepovinné sloupce. Tabulka *Licences* je nepovinná úplně.

Vzhledem k použití technologií jako je IndexedDB ale i další, navrhovaná aplikace pravděpodobně nebude schopna správně fungovat na zastaralých prohlížečích nepodporujících tyto funkcionality. Pro správnou funkci aplikace tedy pro uživatele bude vhodné použití nejnovější verze prohlížečů Google Chrome nebo Mozilla Firefox, což ovšem s ohledem k jejich zastoupení na trhu a cílové skupině uživatelů není zásadním omezením.

[3]https://dexie.org/

4.3 Návrh grafického uživatelského rozhraní

Aplikace bude implementována jako *Single Page Application* (dále jen SPA). Při návrhu grafického uživatelského rozhraní je třeba si uvědomit jaké jsou nejdůležitější požadavky na funkce aplikace. Tím prvním je možnost zobrazit si obrázek v plné velikosti spolu s jeho anotacemi a možnost tyto anotace editovat. Druhým hlavním požadavkem na aplikaci je možnost procházet a zobrazit si náhledy všech obrázků datasetu.

Výběr technologie pro tvorbu uživatelského rozhraní

V otázce výběru vhodného frameworku, či kihovny je třeba si položit zásadní otázku, zda výsledná aplikace má fungovat na webu, nebo jako nativní dektopová aplikace. Ačkoliv každý přístup s sebou nese určité výhody a nevýhody, mezi hlavní důvody proč zvolit nativní aplikaci patří hlavně vyšší rychlost, přístup do lokálního systému souborů. Webové aplikace na druhou stranu, ovšem není třeba instalovat a jsou přístupné odkudkoliv. A i když dnes dokáží webové aplikace fungovat i v offline režimu, většinou vyžadují stálé internetové připojení. Z důvodu multiplatformnosti a jednoduché přístupnosti bylo rozhodnuto, že aplikace implementovaná v rámci této práce bude běžet na webu. Vzhledem k zaměření aplikace, jako pracovního nástroje, nutnost připojení k internetu nepředstavuje zásadní nedostatek.

V sekci 4.1 byly popsány některé z v současnosti nejpoužívanějších frameworků a knihoven pro návrh a implementaci webových grafických uživatelských rozhraní. Tato kapitola více popisuje rozdíly mezi nimi a v závěru odůvodnění výběru frameworku pro implementaci. Pro začátek je třeba si uvědomit hlavní rozdíl mezi zmiňovanými knihovnami a frameworky. Zatímco knihovny React.js a Polymer.js se soustředí pouze na rozhraní aplikace a další funkcionalitu je třeba řešit použitím dalších knihoven, frameworky Angular a Vue.js poskytují kompletní řešení včetně správy globálního stavu nebo routování. Na druhou stranu však tyto frameworky mohou poskytovat značné množství funkcionalit pro potřeby aplikace nadbytečných. Dalším aspektem rozhodování také může být velikost komunity vývojářů kolem každé technologie, u zástupců s větší komunitou lze očekávat snadnější řešení problémů se kterými si programátor neví rady, jelikož je mnohem pravděpodobnější, že už stejný problém někdo jiný řešil. V neposlední řadě je jistě důležitým argumentem velikost samotné knihovny či frameworku, což ovlivňuje množství dat jež, uživatel aplikace musí ze serveru stáhnout aby aplikace mohla fungovat.

Nejstarší technologií mezi zmiňovanými je Angular, jehož první verze byla představena společností Google v roce 2010, tehdy pod názvem Angular.js. Od té doby ovšem prošel velmi výrazným vývojem jenž se vyznačoval, pro vývojáře značně problematickými změnami, jenž způsobovaly nefunkčnost kódu napsaného v předchozích verzích frameworku. Tato absence zpětné kompatibility se během vývoje a přechodu na nové verze frameworku mnohokrát opakovala, což mělo neblahý vliv na popularitu technologie jako takové. Angular taktéž na rozdíl od React.js nebo Vue.js nepoužívá Virtual DOM, což jej činí pomalejší při implementaci aplikací vyžadující vysokou rychlost vykreslování pro lepší interaktibilitu s uživatelem. Vzhledem k velké komplexnosti framwerku se lépe hodí převážné pro velké a robustní aplikace, informační systémy, atd.

React.js naproti tomu využívá technologii Virtual DOM, což jej činí výrazně rychlejším než jen konkurenční Angular. Ve srovnání s ním se však jedná pouze o knihovnu, obstarávající pouze vykreslování grafického uživatelského rozhraní. Což znamená, že bude potřeba použít také další knihovny pro další funkce. Na rozdíl od Vue.js i Angularu, také React.js

nepoužívá šablony pro definici vzhledu komponenty. Zhledem k tomu, že se jedná o knihovnu, je možné React.js přidat do již rozpracovaného projektu, což například u Angular možné není.

Vue.js je nejmladší ze zmiňovaných, staví však na zkušenostech autora s používáním Angular i React.js a dále koncepty v nich použité vylepšuje. I přestože je Vue.js nejmladším představeným frameworkem v popularitě již předehnal Angular a i když na React.js to stále nestačí, jistě to vypovídá o kvalitě technologie. Mezi hlavní výhody Vue.js oproti ostatním, mimo ty zmiňované, patří také fakt, že je relativně snadný na naučení, zejména pro programátory se zkušenostmi s Angular nebo React.js.

Hlavní motivace pro použití Polymer.js je hlavně fakt, že využívá relativně nového standardu Web Components, což jej činí dobrou volbou do budoucna, ovšem v současnosti neposkytuje žádné zásadní výhody oproti výše rozebíraným technologií. Jeho nízká popularita mezi uživateli také, přináší potenciální ztížení řešení problémů, kvůli absenci komunity, jenž by podobné záležitosti řešila.

Tabulka 4.1: Popularita technologií na serveru GitHub k datu 25.5.2020

	Angular	React.js	Vue.js	Polymer.js
Sledující	3.2k	6.7k	6.2k	969
Hvězdičky	61.4k	149k	165k	21.4k
Větvení	16.7k	29k	24.9k	2k
Přispěvatelé	1119	1386	293	141

Z tabulky 4.1 je zřejmá vysoká popularita Vue.js a to i přes fakt, že ve srovnání s Angular a React.js má výrazně menší počet přispěvatelů. Menší počet přispěvatelů ovšem může být způsoben halvně faktem, že Vue.js téměř úplně ve svém vývoji spoléhá na open-source komunitu, zatímco za Angular a React.js stojí velké společnosti jako je Google a Facebook. U Polymer.js můžeme vidět, že popularitou zdaleka nedosahuje ostatních technologií, což je ovšem způsobeno převážně jeho zcela jiným zaměřením. Vzhledem k tomuto faktu, Polymer.js příliš nevyhovuje požadavkům pro implementaci aplikace řešené v této práci. Dále tedy bude za vhodné kandidáty považovat pouze Angular, Ract.js a Vue.js.

Tabulka 4.2: Srovnání některých vlastností technologií pro web GUI

	Angular	React.js	Vue.js
Virtual DOM	Ne	Ano	Ano
Velikost balíčku	247kB	41kB	92kB
Jazyk	TypeScript	JavaScript	JavaScript
Typ technologie	Framework	Knihovna	Framework
Role v MVC	Všechny	View	Všechny

V tabulce 4.2 je očekávaná velikost balíčku s knihovnou/frameworkem. Je důležité si uvědomit, že u Angular a Vue.js je velikost závislá na množství použitých funkcí v aplikaci. Dle očekávání je však React.js jakožto knihovna pouze pro definici uživatelských rozhraní nejmenší ze zmíněných, ovšem jak již bylo zmíněno, ve většině případů je k React.js potřeba přibalit další knihovny pro další funkcionalitu. Velikost balíčku Angular odráží jeho komplexnost a zacílení převážně pro psaní velkých a robustních aplikací. Při použití Angular, je

také třeba využít jazyku TypeScript a to i přesto, že React.js i Vue.js TypeScript jakožto rozšíření JavaScriptu taktéž podporují, je možné u nich použít i obyčejný JavaScript.

Pro implementaci aplikace popisované v této práci jsem se rozhodl použít frameworku Vue.js pro jeho relativní jednoduchost a zároveň komplexnost ve srovnání s React.js. Angular pro svoji obsáhlost a poskytování mnoha funkcionalit v aplikaci nepotřebných, také jako pro absenci Virtual DOM jsem se rozhodl nepoužít a to i přes plán využití TypeScriptu pro implementaci. React.js jsem zase vyloučil kvůli jeho omezeného zaměření pouze na grafickou část aplikace a tedy nutnosti použít dalších knihoven, jenž v případě Vue.js jsou součástí frameworku.

Výběr vektorové grafické knihovny

Tato kapitola se zabývá popsání rozdílů mezi knihovnami popsanými v části **??** a v závěru odůvodněním pro výběr jedné z nich pro implementaci anotátoru v aplikaci obsažené v této práci. I když každá z knihoven byla vytvořena s mírně odlišným záměrem, mají často mnoho podobných rysů, jenž jejich výsledný výběr znesnadňují. Prvním měřítkem podle nějž lze tyto knihovny posuzovat, jezpůsob jakým pracují se samotnou scénou. Většina zmíněných požívá vlastní graf scény (scene graph), což však neplatí pro P5.js, jehož použití je dosti podobné používání samotného canvasu, kdy je celý sketch (canvas, i když P5.js se neomezuje jen na něj) je řízen jako celek. Posloupností příkazů je možno definovat výsledný obraz, pomocí dalších funkcí přidávat interaktibilitu. Ovšem objekty obsažené ve scéně nelze nijak jednoduše rozlišovat ani jim přidávat reakce na vstup uživatele. Například pro kliknutí myši je tedy možno definovat jen jednu globální funkci, uvnitř níž je pak třeba rozlišit kam přesně uživatel kliknul (do kterého objektu) a podle toho zvolit odpovídající akci. Pro potřeby implementované aplikace také bude zásadní aby knihovna podporovala používání obrázků, umožňovala je otáčet a přibližovat/oddalovat (zoom) a hýbat s nimi. Absence grafu scény u P5.js s sebou v tomto případě přímo nese komplikace, například při otáčení obrázku, kdy dojde k otočení celé scény a při snaze hýbat obrázkem posunováním myší je pak třeba manuálně transformovat souřadnice pro určení polohy obrázku. S ohledem na tyto vlastnosti je zřejmé, že P5.js je vhodné spíše na umělecky zaměřené animace a grafiku, nevyžadující precizní uivatelskou interakci.

Tabulka 4.3: Srovnání důležitých vlastností grafických knihoven

	Paper.js	P5.js	Fabric.js	Three.js	Pixi.js
Graf scény (DOM)	Ano	Ne	Ano	Ano	Ano
2D/3D	2D	2D	2D	3D	2D
WebGL	Ne	Ne	Ne	Ano	Ano
Podporuje obrázky	Ano	Ano	Ano	Ano	Ano
Zoom	Ano	Ano	Ano	Ano	Ano

V tabulce 4.3 je možno vidět pro implementovanou aplikaci podstatné vlastnosti porovnávaných knihoven. Three.js se od ostatních odlišuje hlavně tím, že je primárně zaměřeno na 3D grafiku, což pro potřeby implementované aplikace není zásadní, dokonce ani příliš vhodné. Z tohoto důvodu se tedy Three.js příliš pro výslednou implementaci nehodí. Z knihoven využívající WebGL tedy zbývá Pixi.js. Je třeba si ovšem uvědomit, že Pixi.js je v zásadě vykreslovací engine nad WebGL, z tohoto důvodu poskytuje funkcionalitu jenž svými možnostmi přesahuje potřeby implementované aplikace. Použití WebGL v tomto pří-

padě taktéž nepřináší zásadní výhody oproti canvasu u Paper.js nebo Fabric.js, vzhledem k tomu, že implementovaná aplikace nebude náročná na animace.

Tabulka 4.4: Popularita grafických knihoven na serveru GitHub k datu 25.5.2020

	Paper.js	P5.js	Fabric.js	Three.js	Pixi.js
Sledující	425	488	471	2.5k	1.1k
Hvězdičky	11.6k	13.2k	15.9k	60.8k	29.6k
Větvení	1.1k	2.1k	2.5k	23.7k	4.1k
Přispěvatelé	70	353	215	1.2k	323

Stejně jako v případě frameworků a knihoven pro tvorbu grafických uživatelských rozhraní i u grafických knihoven je třeba vzít do úvahy velikost komunity, jenž může usnadňovat hledání řešení různých potenciálních problémů. Popularita knihovny také může do jisté, i když omezené míry, vypovídat o její výkonnosti, množství chyb a celkové kvalitě řešení. Z tabulky 4.4 je zjevná popularita knihovny Three.js, následovaná Pixi.js a Fabric.js. I přes fakt, že ze srovnání je zřejmé, že se Paper.js těší nejmenší popularitě mezi srovnávanými, rozhodl jsem se ji upřednostnit před Fabric.js převážně na základě osobní preference vzniklé při vyzkoušení základních operací pro manipulaci s obrázky a vykreslení polygonů do nich. Knihovna Paper.js pak také byla použita i v implementaci aplikace COCO Annotator jíž se implementovaná aplikace inspiruje, je tedy zřejmé, že jejím použitím lze docílit kvalitního výsledku.

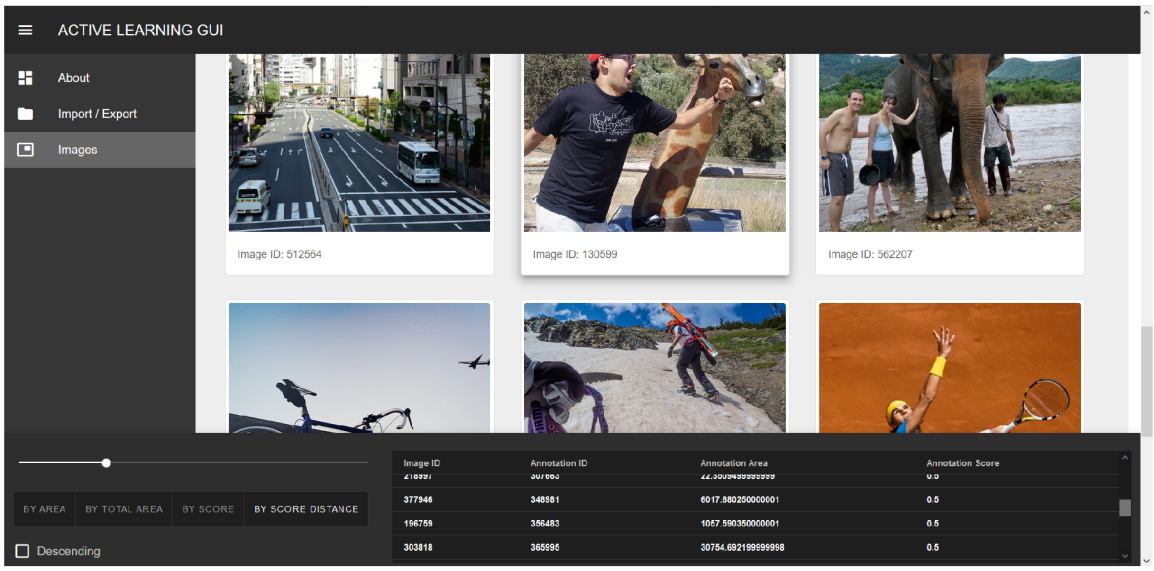
Kapitola 5

Implementace aplikace

V této kapitole jsou rozebírány zajímavé části implementace navržené aplikace. Hlavní část kapitoly se zabývá na uložením anotací a další prací s nimi, jako je řazení, dále pak zobrazování náhledů celého datasetu a zajímavými částmi uživatelského rozhraní.

5.1 Prohlížení datasetu

Pro prohlížení datasetu v aplikaci slouží komponenta *ImageGallery* jenž za pomoci Vue.js direktivy *v-for* v cyklu vykresluje množství instancí komponenty *Picture* obsahující samotný náhled obrázku. Zmíněný cyklus vykresluje náhledy nacházející se ve vlastnosti koponenty *ImageGallery images*. Vlastnost *images* je v tomto případě pole objektů do nějž jsou uloženy seřazené obrázky načtené z databáze. U každého obrázku se následně použije hodnota sloupce *file_name* pro získání URL z Firebase Storage a jeho následnému použití jako *src* atributu (tzv. prop) komponenty *Picture*.



Obrázek 5.1: Ukázka grafického rozhraní aplikace, při prohlížení datasetu

S ohledem na značné množství načítaných obrázků však není možno načíst všechny seřazené obrázky naráz. Toto by pravděpodobně způsobilo zamrznutí stránky a obecně by to přispívalo k negativnímu uživatelskému zážitku. K řešení této výzvy lze obecně přistupovat

vícero způsoby z nichž asi nejpoužívanější jsou rozdělení obsahu na stránky, mezi kterými se uživatel přepíná, když dojde ke konci té současné. Druhým, elegentnějším je použití techniky známé jako *lazy loading*, již sám v aplikaci používám. Smyslem této techniky je načítání obsahu po částech a detekce, kdy uživatel došel na konec zobrazované části. Ve chvíli, kdy je uživatel na konci, dojde k načtení další části dat a jejímu přidání na konec zrovna zobrazené. V implementované aplikaci je tohoto dosaženo s použitím Vue.js direktivy *vue-infinite-scroll*[1]. Tato direktiva umožňuje zavolání metody či funkce, jakmile se na obrazovce objeví konec elementu k němuž je tato direktiva připojena. V aplikaci to tedy vypadá tak, že při načtení komponenty *ImageGallery* dojde k zavolání metody loadFirstPage, jenž do vlastnosti *images* načte prvních 20 obrázků dle dané metody řazení. V cykly vykreslované náhledy jsou pak obaleny HTML elementem *div* jenž k sobě připojuje zmiňovanou direktivu. Jakmile užvatel odscrolluje nakonec, zavolá se metoda *loadMore*, jenž do vlastnosti *images* přidá dalších 20 obrázků. Reaktivní systém Vue.js nakonec vyvolá dokreslení dalších náhledů na obrazovce.

5.2 Editace anotací

Pro samotný editor anotací implementovaný v komponentě *ImageAnnotator* je hlavní částí samotný canvas v němž se odehrává většina uživatelské interakce. Veškerou práci s canvasem v implementaci obstarává knihovna Paper.js, jenž při inicializaci komponenty do canvasu vykreslí požadovaný obrázek a následně přes něj i samotné polygony anotací. Při přepnutí na jiný obrázek dojde ke zrušení současné instance komponenty *ImageAnnotator* a vytvoření nové pro nový obrázek. Pravěpodobně z důvodu provádení množství asynchronních operací při inicializaci komponenty a Paper.js knihovny však při prvním otevření editoru po načtení stránky (případně po refresh) funkcionality implementované s použitím Paper.js nepodaří správně inicializovat a nedojde tak k vykreslení anotací, případně jen části z nich. Uživatel pak musí přepnou na jiný obrázek nebo do *ImageGallery* a následně se k požadovanému obrázku vrátit. Tuto chybu se ovšem zatím nepodařilo vyřešit.

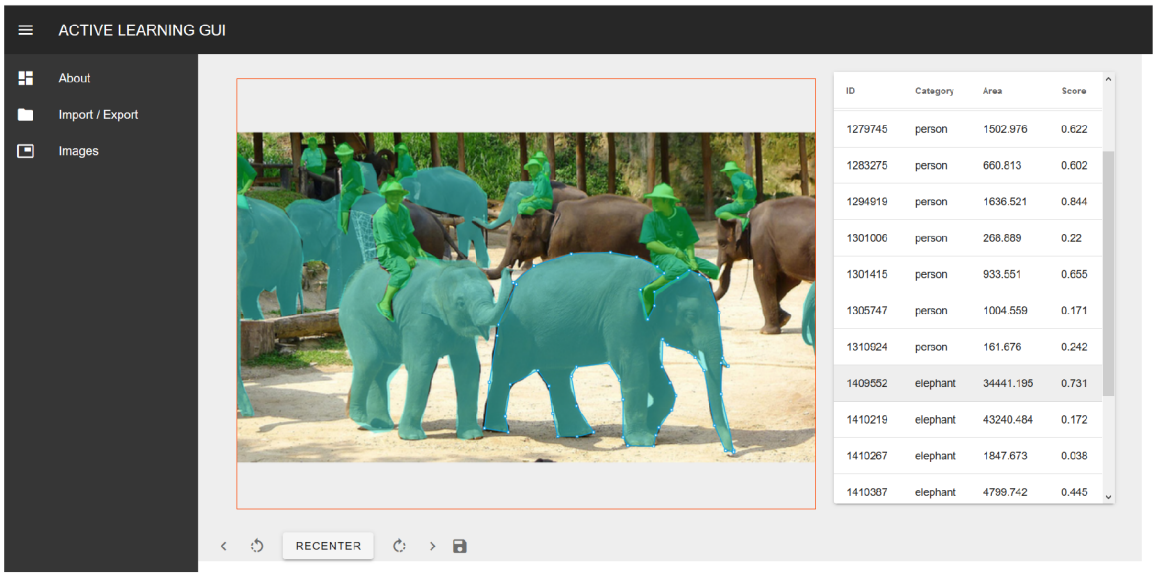
Ačkoliv COCO dataset umožňuje definování anotace také pomocí masky označující jednotlivé pixely v anotaci obsažené. Tyto anotace jsou v JSON souboru s anotacemi označené parametrem *iscrowd = 1*. Současná verze aplikace ovšem nepodporuje vykreslování tohoto typu anotací. Jedinou možností zobrazování anotací je vykreslení pomocí polygonu spolu s bounding boxem anotace. Polygony lze dále upravovat manipulací s jednotlivými body, odebíráním bodů a přidáváním bodů. Celou anotaci je také možno posunout. Za pomocí manipulace s rohy bounding boxu, lze také anotaci otáčet, případně celou zvětšit či zmenšit (scale) a také zcela smazat.

Detekce kliknutí na jednotlivé body provádí knihovna Paper.js s tím, že není třeba kliknout přímo na daný bod, ale je zde nastavena jistá míra tolerance. Tento způsob řešení ovšem u velmi malých anotací přináší problematické chování, jelikož jednotlivé body mohou být k sobě tak blízko, že s připočítáním nastavené tolerance, nedokáže Paper.js vždy přesně detekovat uživatelem zamýšlený bod.

Jednotlivé anotace jsou barevně odlišeny podle jejich atributu *category*. V modulu *categoryColors.ts* je mapa používaná k přiřazení barvy k *id* každé z 91 kategorií podporovaných COCO datasetem. Je však třeba si uvědomit, že dataset importovaný do aplikace uživatelem nemusí nutně obsahovat všech 91 kategorií, ovšem zvolený způsob implementace zajistí, že každá kategie se bude barevně zobrazovat konzistentním způsobem. Jistou nevýhodou

[1] https://github.com/ElemeFE/vue-infinite-scroll

tohoto řešení však může být fakt, že daná anotace nemusí být vždy dobře viditelná, jelikož má stejnou nebo podobnou barvu jaká je na obrázku za ní.



Obrázek 5.2: Ukázka rozhraní editoru anotací

Jak lze vidět na obrázku 5.2, kromě samotného canvasu pro práci s anotacemi, obsahuje editor anotací také tabulku se seznamem anotací obsažených v obrázku. Přejetím kurzoru přes řádky tabulky, případně kliknutím na ně, je možno označovat jednotlivé anotace.

Ukládání provedených změn by se dalo řešit různými způsoby, jako je automatické uložení po provedení každé změny nebo automatické uložení po opuštění daného obrázku. Nakonec jsem se ovšem rozhodl, nechat na uživateli, zda chce změnu uložit nebo ne. Uložení změn tedy uživatel provádí kliknutím na tlačítko k tomu určené.

5.3 Uložení anotací

Jak již bylo nastíněno v kapitole 4, zásadní problémem je uložení obsahu načítaného JSON souboru a další práce s tímto obsahem. K vyřešení toho problému byla využita technologie IndexedDB s níž se veškerá interakce provádí skrze knihovnu Dexie.js. Původně jsem taktéž zvažoval knihovnu localForage, ovšem z důvodu rychlosti Dexie.js jsem nakonec zvolil ji.

Tabulka 5.1: Srovnání rychlosti knihoven rychlosti Dexie.js a localForage. Převzato z https://biancadanforth.github.io/git/2017/08/11/storage-one-of-the-final-frontiers.html

Method	**localForage (ms)**	**Dexie (ms)**	**Winner**
clearAllDatabaseFromBefore	50	3	Dexie
setAll	6554	2265	Dexie
getAll	2491	120	Dexie
setSingle	6	3	Dexie
setManySinglesBlocking	14611	15640	localforage
setManySinglesUnblocking	579	3003	localforage
getRandomSet	8354	1149	Dexie
getMostRecentSite	2704	14	Dexie
getLastThreeSites	2420	5	Dexie
getLastThreeDaysVisibleSites	2732	541	Dexie

Databáze byla implementována podle návrhu z kapitoly 4.2. Vstupní JSON soubor je tedy po nahrání převeden do IndexedDB databáze. Nad níž je pak možno provádět všechny další manipulace s daty, převážně pak řazení. Pro správné použití databáze je třeba si uvědomit, že i když lze řazení obrázků či anotací provádět přímo v javascriptovém kódu, řazení za pomocí indexovaných položek je výrazně rychlejší. Vyplatí se tak načtení a přidání anotací patřících ke každému obrázku do sloupce *image_annotations* a jeho použití jako indexované položky i přes nutnost manuálně znovu načíst upravené anotace při každé změně provedené na anotacích konkrétního obrázku. I když Dexie.js podporuje řazení i neindexovaných položek, ve výsledku se jedná pouze o seřazení Javascriptem a nevyplatí se proto používat. K výhodám řazení za pomocí indexovaných sloupců patří také to, že výsledky se cachují a při dalším vyvolání seřazení uživatelem není třeba samotné seřazení znovu provádět.

Aplikace podporuje čtyři různé metody řazení. První z nich je řazení podle celkové plochy anotací na obrázku. V takovém případě jako jediném řazení probíhá nad tabulkou *images* a pro řazení se používá právě výše zmiňovaný sloupec *annotationsArea*. Dalšími způsoby řazení jsou možnost řadit podle plochy každé jednotlivé anotace, popřípadě podle jejího score. V obou příkladech řazení probíhá nad tabulkou *Annotations* s využitím indexovaných sloupců *area*, respektive *score*. Poslední metoda řazení spočívá v řazení dle vzdálenosti od zadané hodnoty score (v současné verzi aplikace omezeno na vzdálenost od hodnoty 0.5). Řazení v tomto případě opět probíhá na tabulkou *Annotations*, ovšem od přechozích se liší v tom, že není možno použít řazení za pomocí indexovaných položek. S ohledem na fakt, že se zadaná hodnota dynamicky mění v závislosti na požadavku uživatele, není možné přidat do tabulky *Annotations* indexovaný sloupec, jenž by obsahoval vzdálenost od score. Vyžadovalo by to totiž při každé změně této hodnoty, upravení řádků v tomto sloupci napříč celou tabulkou *Annotations* a následně upravení všech řádků *image_annotations* v tabulce *Images*. Z tohoto důvodu, se řazení provádí za pomocí JavaScriptu a seřazené položky se následně uloží do proměnné v modulu *storage.ts* v níž zůstává dokud je potřeba. Jak bylo vysvětleno v části 4.2, sloupec score je v tabulce *Annotations* nepovinný, aplikace tedy automaticky zablokuje možnosti řazení s využitím score pokud při importu datasetu zaznamenala alespoň jednu anotaci bez atributu score. Je samozřejmě otázkou do jaké míry je toto optimální řešení, v budoucích verzích aplikace by se tedy toto chování pravděpodobně mohlo změnit, například vynecháním pouze anotací bez score.

5.4 Celková funkčnost aplikace

Jak již tedy bylo nastíněno aplikace používá obrázky uložené ve službě Firebase Storage. K jejich použití tedy musí být před samotným procesem úpravy anotací obrázky v tomto úložišti nejdříve nahrány. To lze udělat jak přímo přes webové rozhraní Firebase Storage, i když to se mně osobně při nahrávání většího množství obrázků zároveň příliš neosvědčilo. Často tento proces totiž může vést až k zamrznutí okna prohlížeče. Samotná aplikace umožňuje nahrávat obrázky do Firebase Storage také, je ovšem třeba počítat s tím, že tento proces při nahrávání většího množství obrázků může být značně časově náročný a s aplikací není v tutéž chvíli rozumné dále pracovat a to i přes fakt, že proces nahrávání probíhá asynchroně. Aplikace také nekontroluje zbývající volné místo v úložišti, takže je třeba počítat s tím, že když místo dojde, aplikace na to nijak neupozorní. Vzhledem k tomu, že takové chování není příliš uspokojivé, jako budoucí možnost dalšího rozvoje se počítá s použitím jiného způsobu uložení obrázků. Využití Firebase Storage bylo zvoleno hlavně pro možnost snadného začlenění do projektu a jednoduchého použití ve srovnání například s implementací vlastní databáze. Hlavním smyslem úložiště tedy v tuto chvíli je usnadnění vývoje, testování a také demonstraci možností aplikace.

Aplikace v současné verzi umožňuje použití v kombinaci s modelem aktivího učení jak bylo nastíněno v digramu 4.2. Import a export dat uživatel musí provádět manuálně. Kromě mazání anotací aplikace umožňuje odstraňovat i celé obrázky. Při jejich odstranění však nedochází k odstranění souboru obrázku z Firebase Storage ale pouze z lokální databáze zrovna zpracovávaného datasetu. Je tak vyřazen z procesu aktivního učení, ale je možné jej znovu použít v kombinaci s jiným datasetem.

Jako další možnosti rozvoje se nabízí také vykreslování anotací definovaných za pomocí masky a přidání funkcí pro zlepšení práce s editorem anotací, například možností skrýt vybrané anotace, případně měnit jejich průhlednost. Dále pak optimalizace výkonnosti aplikace v některých případech jejího použití nebo odstranění známých chyb. V neposlední řadě samozřejmě také vylepšení uživatelského rozhraní, například zvětšení canvasu v editoru anotací pro lepší práci s velkými obrázky.

Kapitola 6

Závěr

Cílem práce bylo seznámit se s konceptem aktivního učení v neoronových sítích a vytvořit funkční grafické uživatelské rozhraní pro anotaci obrazových dat. Výsledné uživatelské rozhraní a funkcionalitu jsem se rozhodl implementovat jako webovou aplikaci s využitím moderních javascriptových frameworků a knihoven.

Na samém začátku bylo provedeno seznámení se s aktivním učením s důrazem především na práci s obrazem. Dále bylo třeba prozkoumat současné možnosti týkající se návrhu a implementace webových grafických uživatelských rozhraní a vybrat technologii nejvhodnější pro implementaci. Dále pak bylo třeba nastudovat specifikaci COCO datasetu.

Vzledem k požadavků na funkcionalitu aplikace bylo třeba také vybrat a nastudovat vhodnou knihovnu pro práci s vektorovou grafikou na webu. Pro tento účel byla zvolena knihovna Paper.js s pomocí níž byla implementována funkcionalita pro vykreslování a manipulaci s polygony a bounding boxy jednotlivých anotací. S ohledem na požadavky aplikace však také bylo třeba nastudovat možnosti vhodné pro ukládání dat na straně klienta a zpracování velkého množství obrázků a anotací.

Na základě průzkumu byl vytvořen návrh aplikace implementující požadované funkcionality prohlížení datasetu, výběru dat implementovaného možností odstraňovat anotace či obrázky, možností provádět úpravy anotací, zobrazovat score jednotlivých anotací a řadit podle něj.

Hlavní částí práce byl návrh a implementace samotného grafického rozhraní a možností uživatelské interakce, ovšem největší výzvou se ukázala být samotná práce s daty, jejich uložením a manipulací s nimi. Vzledem ke komplexnosti této části implementace se mi bohužel nepodařilo implementovat některou funkcionalitu, jenž sice není nezbytná, ovšem udělala by používání aplikace výrazně příjemnějším. V tomto směru tedy spolu s využitím vhodnějšího úložiště pro soubory obrázků a odstranění občasných chyb při vykreslování anotací vidím hlavní možnosti dalšího rozvoje aplikace.

Pro někoho limitujícím faktorem by mohl být fakt, že aplikace vzhledem k použití některých moderních technologií a jejích potenciálně značných požadavcích na výkon nepočítá s fungováním ve starých verzích webových prohlížečů. S ohledem na cílovou skupinu uživatelů pro mě toto ovšem nikdy nebyla priorita. Pro ukázku je aplikace připravena na adrese https://active-learning-gui.firebaseapp.com/.

Aplikace má tedy značný potenciál pro další rozvoj, ať už se jedná o výše zmíněné záležitosti, tak ve smyslu přidávání dalších funkcionalit, jako jsou další nástroje pro anotaci, možnost spolupráce v týmu, nebo s využitím vhodného úložiště souborů obrázků také možnosti fungovat zcela offline.

Literatura

[1] Google. *Introduction to Angular concepts* [online]. [cit. 2020-05-01]. Dostupné z: https://angular.io/guide/architecture.

[2] Bednář, R. *Služba pro zjištění zranitelností knihoven použitých na webových stránkách.* Brno, CZ, 2019. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Dostupné z: https://www.fit.vut.cz/study/thesis/22139/.

[3] LePage, P. *Storage for the web* [online]. [cit. 2020-05-10]. Dostupné z: https://web.dev/storage-for-the-web/.

[4] MDN contributors. *Client-side storage* [online]. [cit. 2020-05-04]. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Client-side_web_APIs/Client-side_storage.

[5] Bornstein, A. *The Microsoft Cognitive Toolkit* [online]. 2017 [cit. 2020-05-12]. Dostupné z: https://docs.microsoft.com/en-us/cognitive-toolkit/.

[6] Brooks, J. *COCO Annotator* [online]. 2019 [cit. 2020-05-15]. Dostupné z: https://github.com/jsbroks/coco-annotator/wiki.

[7] Sekachev, B., Zhavoronkov, A. a Manovich, N. *Computer Vision Annotation Tool: A Universal Approach to Data Annotation* [online]. 2019 [cit. 2020-05-12]. Dostupné z: https://software.intel.com/content/www/us/en/develop/articles/computer-vision-annotation-tool-a-universal-approach-to-data-annotation.html.

[8] MDN contributors. *Introduction to the DOM* [online]. [cit. 2020-05-03]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction.

[9] Electron. *Electron Application Architecture* [online]. [cit. 2020-05-15]. Dostupné z: https://www.electronjs.org/docs/tutorial/application-architecture.

[10] MDN contributors. *Browser storage limits and eviction criteria* [online]. [cit. 2020-05-10]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API/Browser_storage_limits_and_eviction_criteria#Storage_limits.

[11] Google. *Zabezpečení webu protokolem HTTPS* [online]. [cit. 2020-05-05]. Dostupné z: https://support.google.com/webmasters/answer/6073543?hl=cs.

[12] MDN contributors. *IndexedDB API* [online]. [cit. 2020-05-06]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API.

[13] Maninis, K.-K., Caelles, S., Pont Tuset, J. a Van Gool, L. Deep Extreme Cut: From Extreme Points to Object Segmentation. In: *Computer Vision and Pattern Recognition (CVPR)*. 2018.

[14] Verma, U. *P5.js overview* [online]. 2019 [cit. 2020-05-15]. Dostupné z: https://github.com/processing/p5.js/wiki/p5.js-overview.

[15] Wijk, L. van. *Polymer is dead, long live Web Components!* [online]. [cit. 2020-05-02]. Dostupné z: https://craftsmen.nl/polymer-is-dead-long-live-web-components/.

[16] Camden, R. *IndexedDB and Limits* [online]. [cit. 2020-05-10]. Dostupné z: https://www.raymondcamden.com/2015/04/17/indexeddb-and-limits.

[17] Koďousková, B. *Vue JS: výhody, nevýhody a možnosti využití* [online]. [cit. 2020-05-01]. Dostupné z: https://www.rascasone.com/cs/blog/co-je-framework-vuejs.

[18] Settles, B. *Active learning literature survey*. 2010.

[19] MDN contributors. *The structured clone algorithm* [online]. [cit. 2020-05-10]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API/Structured_clone_algorithm.

[20] Supervise.ly. *Advanced annotation tools in Deep Learning: training data for computer vision with Supervisely* [online]. 2018 [cit. 2020-05-12]. Dostupné z: https://hackernoon.com/%EF%B8%8F-advanced-annotation-tools-in-deep-learning-training-data-for-computer-vision-with-supervisely-847f8699a9cb.

[21] Facebook. *Virtual DOM and Internals* [online]. [cit. 2020-04-30]. Dostupné z: https://reactjs.org/docs/faq-internals.html#what-is-the-virtual-dom.

[22] Bornstein, A. *End to End Object Detection in a Box* [online]. 2017 [cit. 2020-05-12]. Dostupné z: https://devblogs.microsoft.com/cse/2017/04/10/end-end-object-detection-box/.

[23] Vue. *Introduction* [online]. [cit. 2020-05-01]. Dostupné z: https://vuejs.org/v2/guide/.

[24] MDN contributors. *Web Components* [online]. [cit. 2020-05-02]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/Web_Components.

[25] MDN contributors. *Web Storage API* [online]. [cit. 2020-05-06]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API.

[26] MDN contributors. *WebGL: 2D and 3D graphics for the web* [online]. [cit. 2020-05-15]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API.

[27] Zakas, N. C. *High Performance JavaScript - Build Faster Web Application Interfaces*. O'Reilly,, 2010. ISBN 978-0-596-80279-0.

Příloha A

Obsah přiloženého paměťového média

- `/app/` - zdojové kódy implementované aplikace
- `/tex/` - zdrojové kódy textu práce
- `/pdf/` - text práce ve formátu pdf

Příloha B

Zprovoznění aplikace v lokálním prostředí

Pro zprovoznění aplikace v lokálním prostředí je potřeba provést následující kroky.

B.1 Prerekvizity

Na zařízení je třeba mít instalovány tyto programy:

- `Node` ve verzi >= 10
- Balíčkovací systém `npm` ve verzi >= 6

Je třeba také mít založený projekt ve službě Firebase

B.2 Instalace aplikace

Po přepnutí do kořenového adresáře projektu (adresář obsahující soubor `package.json`) je třeba spustit příkaz:

- `npm install`

Po skončení instalace závislostí je třeba pro připojení aplikace ke službě Firebase v kořenovém adresáři vytvořit soubor `.env`. V tomto souboru je potom nutné definovat tyto proměnné:

- `VUE_APP_FIREBASE_API_KEY`
- `VUE_APP_FIREBASE_AUTH_DOMAIN`
- `VUE_APP_FIREBASE_DB_URL`
- `VUE_APP_FIREBASE_PROJECT_ID`
- `VUE_APP_FIREBASE_STORAGE_BUCKET`
- `VUE_APP_FIREBASE_MESSAGING_SENDER_ID`
- `VUE_APP_FIREBASE_APP_ID`
- `VUE_APP_FIREBASE_MEASUREMENT_ID`

Do proměnných je třeba přiřadit informace získané z projektu ve Firebase Console.

B.3 Spuštění aplikace

Pro spuštění aplikace je potřeba spustit lokální server, spustitelný například příkazem:

- `npm run serve`

Toto krom spuštění aplikace umožňuje i provádět změny ve zdrojovém kódu, přičemž tyto změny se automaticky projeví v běžící aplikaci.

B.4 Kompilace aplikace

Kompilace zdrojových kódů se provádí příkazem:

- `npm run build`

Výsledek kompilace se potom nachází v adresáři `dist`