VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

ZNOVUPOUŽITELNÝ 2D EDITOR PRO WEBOVÉ APLIKACE

REUSABLE 2D EDITOR FOR WEB APPLICATIONS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. MARTIN SCHNEIDER**
AUTHOR

VEDOUCÍ PRÁCE **prof. Ing. ADAM HEROUT, Ph.D.**
SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání diplomové práce

148741

Ústav: Ústav počítačové grafiky a multimédií (UPGM)
Student: **Schneider Martin, Bc.**
Program: Informační technologie a umělá inteligence
Specializace: Vývoj aplikací

Název: **Znovupoužitelný 2D editor pro webové aplikace**
Kategorie: Uživatelská rozhraní
Akademický rok: 2022/23

Zadání:

1. Seznamte se s problematikou vývoje frontendových aplikací; zaměřte se na editaci 2D grafiky a zobrazování obrázků.
2. Analyzujte potřeby na editaci anotací a zobrazování výsledků v úlohách počítačového vidění.
3. Prototypujte dílčí prvky řešeného editoru, testujte je s uživateli a iterativně je vylepšujte.
4. Integrujte dílčí prvky řešeného editoru do vyvíjené komponenty.
5. Testujte vyvíjené řešení s uživateli a iterativně je vylepšujte k dokonalosti.
6. Dokumentujte vytvořenou komponentu a vytvořte aplikaci pro demonstraci jejího nasazení. Zvažte možnosti demonstrování jejího použití ve spojení s různými frontend/backend frameworky.
7. Zhodnoťte dosažené výsledky a navrhněte možnosti pokračování projektu; vytvořte plakátek a krátké video pro prezentování projektu.

Literatura:

- Kelvin Sung et al.: Build Your Own 2D Game Engine and Create Great Web Games (Using HTML5, JavaScript, and WebGL2), Apress Berkeley, CA, 2022
- Foley et al.: Computer Graphics: Principles and Practice, Addison-Wesley, 3rd edition, 2013
- Dan Vanderkam: Effective TypeScript, O'Reilly Media, 2019
- Tidwell et al.: Designing Interfaces: Patterns for Effective Interaction Design, O'Reilly, 2020
- Steve Krug: Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability, ISBN: 978-0321965516
- Steve Krug: Rocket Surgery Made Easy: The Do-It-Yourself Guide to Finding and Fixing Usability, ISBN: 978-0321657299
- Joel Marsh: UX for Beginners: A Crash Course in 100 Short Lessons, O'Reilly 2016

Při obhajobě semestrální části projektu je požadováno:
body 1. a 2., značné rozpracování bodů 3. a 4.

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

Vedoucí práce: **Herout Adam, prof. Ing., Ph.D.**
Vedoucí ústavu: Černocký Jan, prof. Dr. Ing.
Datum zadání: 1.11.2022
Termín pro odevzdání: 17.5.2023
Datum schválení: 31.10.2022

Abstrakt

Cílem této práce je realizovat 2D editor jako knihovnu pro webové aplikace společně s testovací aplikací. Knihovna primárně cílí na prostředí segmentačních neuronových sítí a vzdáleného použití s optimalizacemi na přenos dat. Nejprve se čtenář seznámí s problematikou vývoje frontendových aplikací se zaměřením na editaci 2D grafiky a zobrazování obrazových dat. Dále je uvedena analýza potřeb na 2D editor s využitím pro editaci anotací a zobrazování výsledků v úlohách počítačového vidění. Výsledek analýzy je posléze převeden na návrh systému, který je následně implementován a otestován.

Abstract

The goal of this work is to implement a 2D editor as a library for web applications together with a test application. The library primarily targets segmentation neural network and remote use environments with optimizations for data transfer. First, the reader is introduced to frontend application development with a focus on 2D graphics editing and image data display. Next, an analysis of the needs for a 2D editor with applications to annotation editing and displaying results in computer vision tasks is presented. The result of the analysis is then converted into a system design, which is subsequently implemented and tested.

Klíčová slova

2D editor, rastr, vektor, webová aplikace, zpracování obrazu, dlaždicování obrazu, anotátor, anotační nástroj, segmentační neuronová síť, JavaScript, TypeScript, React

Keywords

2D editor, raster, vector, web application, image processing, image tiling, annotator, annotation tool, segmentation neural network, JavaScript, TypeScript, React

Citace

SCHNEIDER, Martin. *Znovupoužitelný 2D editor pro webové aplikace*. Brno, 2023. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce prof. Ing. Adam Herout, Ph.D.

Znovupoužitelný 2D editor pro webové aplikace

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana prof. Ing. Adama Herouta, Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Martin Schneider
6. května 2023

Poděkování

Děkuji svému vedoucímu, panu prof. Ing. Adamu Heroutovi, Ph.D., za aktivní přístup k vedení mé práce a přínosné rady. Dále děkuji všem, kteří se podíleli na testování a připomínkování výsledků mé práce.

Obsah

Kapitola 1

Úvod

Tato diplomová práce se zabývá návrhem a implementací znovupoužitelného 2D editoru pro webové aplikace, který by umožňoval snadnou a intuitivní tvorbu grafických prvků a objektů pro webové aplikace. Cílem práce je navrhnout a implementovat funkční 2D editor, který by primárně sloužil k použití v oblasti segmentačních neuronových sítí. Součástí procesů zmíněných sítí je anotování dat a monitorování výstupu neuronových sítí.

Anotace 2D obrazových dat je důležitá pro řadu oborů, které zahrnují analýzu obrazu a videa, jako je počítačové vidění, strojové učení, robotika a autonomní systémy. 2D anotace zahrnuje proces přidávání štítků, značek nebo ohraničujících rámečků do 2D obrázků nebo videí za účelem identifikace a lokalizace objektů zájmu. Tento proces obvykle provádějí lidé ručně nebo pomocí příslušných softwarových nástrojů. Význam 2D anotace spočívá v tom, že poskytuje způsob, jak trénovat a zlepšovat přesnost algoritmů počítačového vidění. Označováním objektů na snímcích nebo videích se mohou modely strojového učení naučit rozpoznávat a klasifikovat tyto objekty, což jim umožní provádět úlohy, jako je detekce, sledování a segmentace objektů. V odvětvích, jako je zdravotnictví, automobilový průmysl a bezpečnost, je přesná detekce a sledování objektů zásadní pro zajištění bezpečnosti a prevenci nehod. V oborech, jako je zábava a reklama, se 2D anotace používá pro vytváření realistických vizuálních efektů a animací. Celkově je 2D anotace základním krokem ve vývoji algoritmů počítačového vidění a hraje klíčovou roli v tom, aby stroje mohly „vidět“ svět kolem sebe a komunikovat s ním.

Anotace může vytvářet velký počet anotátorů s různými zařízeními z různých míst na světě. Monitorování výstupů neuronových sítí může také probíhat vzdáleně. Díky těmto faktorům je knihovna určena pro webové aplikace, díky čemuž je zajištěno jednoduché nasazení. K webovým aplikacím může být přistupováno i pomocí pomalého připojení nebo připojení účtovaného podle objemu dat, a proto je jednou z klíčových vlastností této knihovny také optimalizace úspory přenosu dat. Vzhledem ke komplexnosti editoru a povahy použití, kdy bude sloužit pro neustálé monitorování dat z neuronové sítě a dlouhodobé anotování, jsou součástí této práce i testy zaměřené na úniky paměti. Vrstvy anotací dat i výstupy neuronových sítí mohou být buď ve vektorovém nebo rastrovém formátu a ve více vrstvách, proto knihovna umožňuje kombinaci těchto přístupů.

Výstupem této diplomové práce je funkční a otestovaná knihovna 2D editoru, společně s ukázkovou webovou aplikací, serverovou aplikací jakožto zdroje obrazových dat a testovací prostředí pro monitorování využití paměti aplikace. Práce postupně probíhá od analýzy požadavků a potřeb uživatelů viz kapitola 2, přes návrh a specifikaci editoru, jenž jsou popsány v kapitole 3, až po jeho implementaci a testování, kterými se zabývá kapitola 4.

V závěrečné kapitole 5 je shrnutí a závěrečné hodnocení celkového návrhu a implementace editoru a případné doporučení pro další rozvoj a rozšíření editoru.

Kapitola 2

Analýza požadavků na anotační 2D editor a dostupných technologií

2.1 Anotace v kontextu segmentačních neuronových sítí

Segmentační neuronové sítě jsou typem hlubokých neuronových sítí, které se používají pro segmentaci obrazu. Segmentace obrazu znamená přiřazení každému pixelu v obraze určité třídy, což umožňuje rozlišovat různé části obrazu a provádět různé úkoly, jako je detekce objektů, rozpoznávání defektů a obličejů a dalších. Více informací k segmentačním neuronovým sítím lze nalézt v knize o segmentaci obrazu [3]. Anotace dat jsou nezbytné pro trénování segmentačních neuronových sítí. Existují dva hlavní způsoby, jak anotovat data pro segmentační neuronovou síť: vektorové anotace a rastrové anotace. Více informací o obrazových anotacích píše Govindaraj Sureshkumar v knize [6].

2.1.1 Rastrové anotace

Rastrové anotace jsou druhem anotace, která se používá pro segmentaci obrazu na základě pixelů. Tento typ anotace se používá pro anotaci objektů nebo oblastí, které nemají jasně definované hrany, jako jsou například defekty dílů, mraky nebo stromy. Rastrové anotace jsou snadno vytvořitelné a zpracovatelné pomocí běžných nástrojů pro úpravu obrázků, jako je například Photoshop nebo GIMP. Anotace jsou velmi přesné, je však nutné ukládat velké množství dat.

Pixelová anotace spočívá v tom, že každý pixel v obrázku je přiřazen k určité třídě (např. pozadí, objekt 1, objekt 2, atd.). Toto je nejpřímější způsob anotace dat pro segmentační neuronové sítě. Pro každý obrázek v trénovací množině je tedy nutné anotovat každý pixel. Pixelová anotace může být použita pro trénování sítě, která dokáže segmentovat obrázky do jednotlivých tříd.

Masková anotace je jednodušší varianta pixelové anotace. V této anotaci jsou všechny pixely, které představují objekt, vyznačeny stejnou barvou. To znamená, že masková anotace vytvoří binární obrázek, který má hodnotu 0 pro pozadí a hodnotu 1 pro objekt. Tento způsob anotace je také vhodný pro trénování segmentačních neuronových sítí.

2.1.2 Vektorové anotace

Vektorové anotace jsou kódovány jako soubory vektorových grafik, které obsahují matematické popisy bodů a křivek v obraze. Tento typ anotace se používá zejména pro anotaci

objektů, které jsou dobře definované a mají jasně viditelné hrany, jako jsou budovy, silnice, auta atd. Vektorové anotace také zaujímají malý prostor na disku, což znamená, že jsou snadno přenositelné a uložitelné.

Polygonová anotace se používá k označení oblastí na obrázku, které představují jednotlivé objekty. Pro každý objekt na obrázku je nakreslen polygon, který obklopuje daný objekt. Tento polygon obsahuje několik bodů a může mít různé tvary. Pro každý polygon je poté přiřazena třída, kterou reprezentuje. Tato data jsou poté použita pro trénování sítě, která dokáže segmentovat obrázky do jednotlivých objektů.

Bounding box anotace se používá k označení oblasti, kterou objekt obsadí. Obvykle se používají čtyřúhelníky, které jsou kresleny kolem objektu a definují jeho polohu na obrázku. V této anotaci není nutné označovat každý pixel, což zjednodušuje proces anotace dat. Pro každý bounding box je poté přiřazena třída, kterou objekt reprezentuje. Tato data jsou poté použita pro trénování sítě, která dokáže detekovat a lokalizovat objekty na obrázku.

Kombinovaná anotace kombinuje různé způsoby anotace dat, aby se získala více informací pro trénování segmentačních neuronových sítí. Například může být použita pixelová anotace pro segmentaci obrazu a bounding box anotace pro detekci a lokalizaci objektů.

2.2 Funkční požadavky na knihovnu editoru

Kreslení v 2D počítačové grafice probíhá buď vektorově nebo rastrově. Pro segmentační anotace obrazových dat se využívá obou typů těchto nástrojů. Tak jako obecné editory, tak i v oblasti anotací je běžná práce s vrstvami, které mohou značit různé typy anotací jako například typ defektu výrobku. Požadavkem tedy je, aby knihovna podporovala rastrový i vektorový režim v současně zobrazených vrstvách.

Požadavky na editor z pohledu zobrazování dat

Neuronové sítě mohou produkovat velké množství obrazových data, které může být žádoucí monitorovat. Obrazová data mohou při velkých rozměrech velmi nabývat na datové náročnosti. V případě vzdáleného monitorování výstupů neuronových sítí vytváří tato skutečnost problém se zatížením datového provozu, obzvláště u připojení účtovaných podle objemu přenesených dat. Komprese dat v případě požadavku na bezztrátovost obrazu není příliš vysoká, proto je třeba stahovat pouze potřebná data. Dalším požadavkem je i vysoká stabilita, protože při nasazení editoru v produkčním prostředí může aplikace běžet i několik let a během tak dlouhého časového intervalu zpracovat obrovské množství dat. Je tedy nutné editor otestovat na případné úniky paměti.

Požadavky na editor z pohledu anotací dat

Pro učení segmentačních neuronových sítí je často potřeba velké množství anotovaných dat. Kvůli požadavku na velké množství dat je vhodné, aby se procesu anotování účastnilo více uživatelů. Velké množství uživatelů má však velkou škálu různých platforem, kde aplikaci budou používat. Díky tomu je nutné vytvořit multiplatformní řešení s co nejjednodušším nasazením a následnou podporou.

2.3 Průzkum konkurenčních 2D editorů a anotačních aplikací

Existuje velké množství online kreslících nástrojů. Dostupné nástroje se však zaměřují spíše na vytváření obrázků pro estetické účely. Obecným problémem je také antialiasing, který je u webové grafiky vynucen a je nutné implementovat tradiční grafické algoritmy[1].

Hlavní motivací vzniku této práce je absence obdobného nástroje jak proprietárního, tak open-source. Dle průzkumu ve společnosti dodávající řešení pro inspekci průmyslových výrobků pomocí segmentačních neuronových sítí, jsou datové sada pro učení sítí anotovány pomocí nástroje GIMP. Tento způsob bude nahrazen právě komponentou vzniklou z této diplomové práce.

SuperAnnotate[2]

SuperAnnotate je proprietární cloudová platforma pro anotaci obrazů a dat. Tato služba umožňuje uživatelům nahrát, sdílet a anotovat různé typy dat, jako jsou obrázky, videa, zvuky a texty. SuperAnnotate nabízí širokou škálu nástrojů pro anotaci, včetně možnosti označování a segmentace objektů, zvýrazňování textu, kruhových a obdélníkových značek a mnoha dalších. Platforma disponuje možností spolupráce v reálném čase. SuperAnnotate také nabízí řadu pokročilých funkcí, jako je například automatické generování anotačních úkolů, sledování pokroku anotace a export dat ve formátech, jako je COCO, Pascal VOC a mnoho dalších.

Protože nástroj podporuje pouze vektorový mód anotací a není open-source, tak není v této práci dále využit. Uživatelské rozhraní nástroje viz snímek obrazovky nástroje 2.1.

Polygon-RNN++[3]

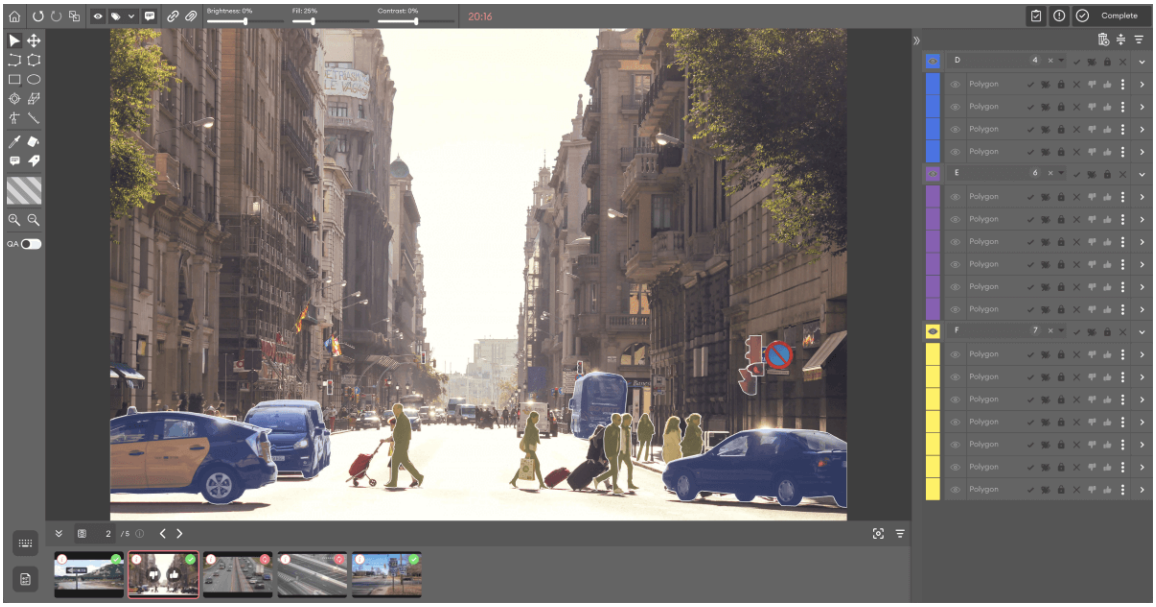
PolygonRNN++ je nástroj pro automatické generování vektorových obrazů pomocí neuronových sítí. Zdrojové kódy jsou k dispozici na vyžádání pro akademické účely. Komerční účely jsou řešeny individuálně. Jedná se o rozšíření původního nástroje PolygonRNN, který byl vyvinutý v roce 2018 v rámci výzkumného projektu na univerzitě v Torontu. Hlavním cílem PolygonRNN++ je generovat kvalitní vektorové obrazy s minimálním množstvím interakce od uživatele. Tento nástroj se zaměřuje na tvorbu jednoduchých vektorových obrazů, jako jsou například ikony, loga, nápisy a jednoduché obrázky. PolygonRNN++ využívá neuronovou síť k předvídání souřadnic vektorů, které jsou potřebné k vytvoření vektorového obrazu. Uživatel začíná kreslením hrubých náčrtů obrazu pomocí myši nebo stylusu a nástroj následně predikuje další body vektoru, které mají být kresleny. Uživatel může kdykoli přerušit generování a upravovat již nakreslené body.

Z důvodu částečné uzavřenosti se tímto nástrojem práce dále nezabývá. Je však možné uvažovat nad rozšířením editoru pomocí neuronových sítí pro předvídání anotací uživatele. Uživatelské rozhraní nástroje viz snímek obrazovky nástroje 2.2.

[1] `https://medium.com/@kozo002/how-to-draw-without-antialiasing-on-html5-canvas-cf13294a8e58`

[2] `https://superannotate.com/`

[3] `https://github.com/fidler-lab/polyrnn-pp-pytorch`

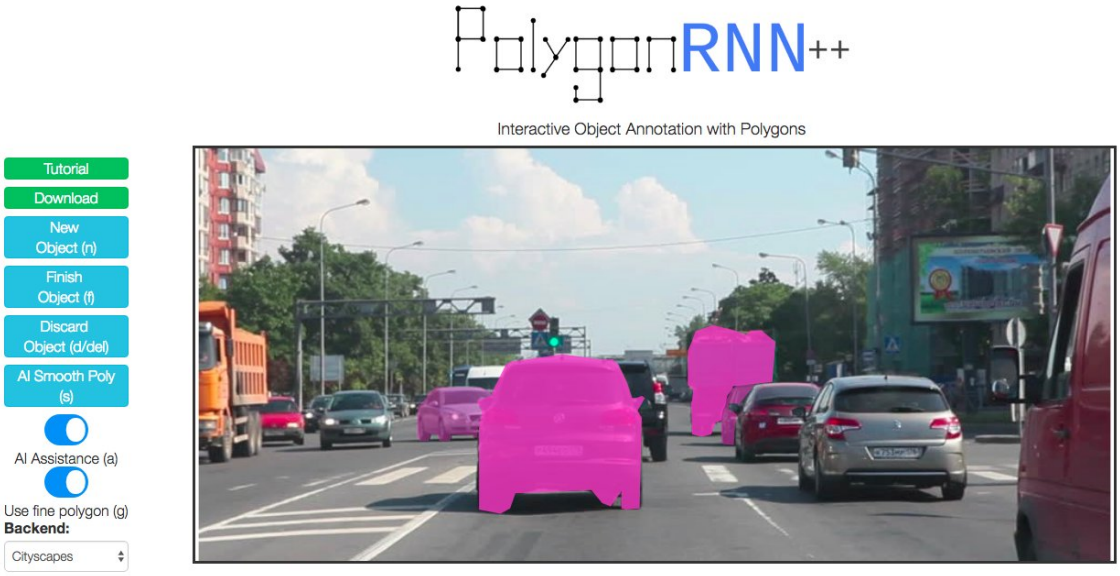


Obrázek 2.1: Snímek obrazovky aplikačního rozhraní anotačního nástroje SuperAnnotate. Nástroj v levém menu disponuje vektorovými nástroji pro kreslení lomených čar a jednoduchých vektorových tvarů. Umožňuje také třídění jednotlivých anotací do skupin v pravém menu, které jsou reprezentovány barvou a názvem. Jednotlivá anotovaná data lze přepínat ve spodní liště, kde jsou i jejich náhledy.

Label Studio

Label Studio je open-source nástroj pro anotaci dat vyvinutý společností Heartex. Label Studio umožňuje uživatelům vytvářet anotační projekty a přizpůsobovat je svým potřebám, jako jsou například kategorizace textu, rozpoznávání entit, klasifikace obrázků a mnoho dalšího. Uživatelé mohou také vybrat z různých typů anotací, včetně jednoduchého textového označování, segmentace, vztahů a mnoho dalšího. V Label Studiu mohou uživatelé také spolupracovat a sdílet projekty s týmem, což je užitečné pro velké projekty s velkým objemem dat. Uživatelé mohou také použít nástroje pro validaci anotací a využívat automatizovanou anotaci k urychlení procesu.

Samotný anotační nástroj pro segmentaci obrazových dat však disponuje pouze vektorovým režimem a postrádá některé funkcionality zefektivňující anotování dat, jako například ovládání zoomu pomocí kolečka myši. Samotné anotování pomocí lomených čar je velmi pokročilé a funkcionalita nástroje editoru této diplomové práce je jím silně inspirována. Segmentační anotační nástroj Label Studia byl prezentován třem uživatelům, z nichž ani jeden nebyl schopen vytvořit lomenou čáru během prvních patnácti minut, kvůli nepřehlednému UI pro výběr aktuálně kreslené anotace. Label Studio však nad rámec anotačního nástroje disponuje informačním systémem pro anotovaná data, čímž se tato práce nezabývá. Bylo by však možné tyto dvě řešení propojit, využitím informačního systému Label Studia. Uživatelské rozhraní nástroje viz snímek obrazovky nástroje 2.3.



Obrázek 2.2: Snímek obrazovky rozhraní anotačního nástroje Polygon-RNN++. Uživatelské rozhraní je minimalistické pro demonstrační účely a zobrazuje použití automatického označení objektů vygenerovaného neuronovou sítí. Vzniklé anotace jsou reprezentovány pomocí uzavřených lomených čar s růžovou výplní.

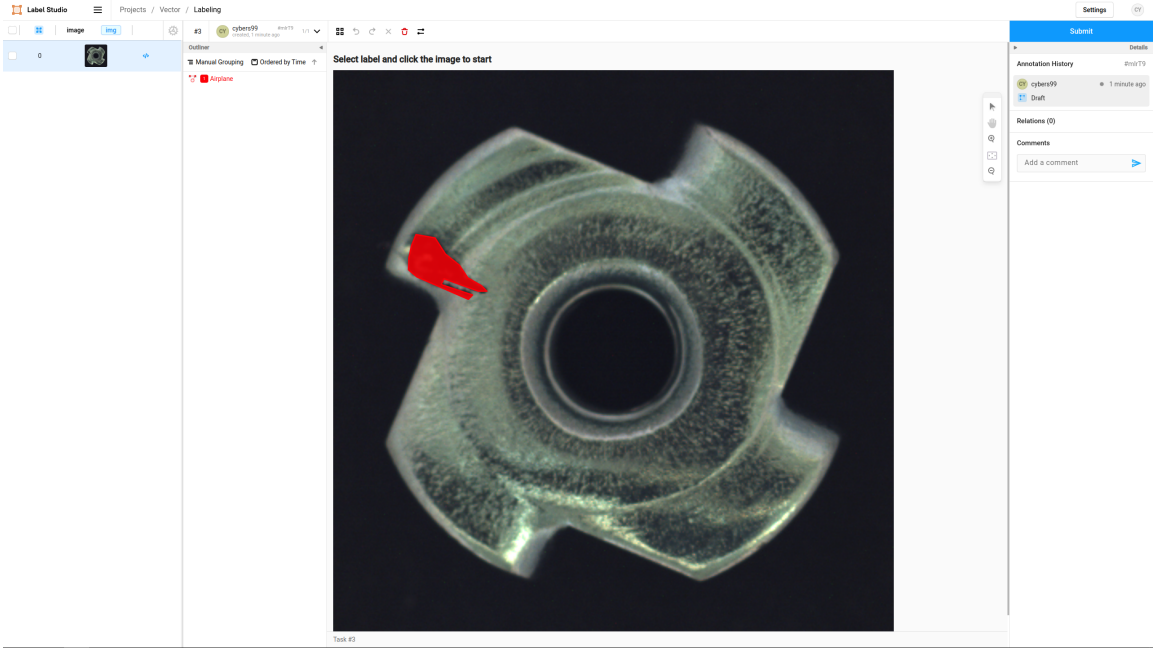
2.4 Dostupné technologie pro vývoj grafické webové aplikace

V této kapitole se seznámíme s dostupnými technologiemi pro implementaci 2D editoru jako webové aplikace. Představíme si jejich vlastnosti, výhody a nevýhody. Prozkoumáme technologie pro zobrazování 2D grafiky a práci s ní a technologie pro vývoj aplikací na straně klienta i serveru.

Jazyky pro vývoj webových aplikací

V dnešní době je prakticky standardem vyvíjet webové aplikace v jazyce JavaScript, případně jeho typovanou variantou TypeScript. Pro využití jiných jazyků je možné použít WebAssembly, případně využít renderování stránek na straně serveru. **JavaScript** je programovací jazyk, který se používá pro vývoj webových aplikací. Je to objektově orientovaný jazyk, což znamená, že se skládá z objektů, které obsahují vlastnosti a metody. JavaScript se často používá spolu s HTML a CSS pro tvorbu interaktivních webových stránek, například pro vkládání dynamických efektů nebo pro vytváření webových aplikací. JavaScript se často používá pro vývoj webových aplikací s využitím frameworků jako například Angular, Vue nebo React. Moderní přístup k JavaScriptu popisuje Mark Myers v knize [4].

TypeScript je superset jazyka JavaScript, což znamená, že obsahuje všechny funkce a možnosti JavaScriptu a navíc přidává některé další vlastnosti. Typescript je staticky typovaný jazyk, což znamená, že proměnné a funkce mají předem určený datový typ. To umožňuje vývojářům předem určit, jakého datového typu se mají hodnoty v kódu používat, a tím snížit možnost výskytu chyb při tvorbě kódu. TypeScript se často používá stejně jako Javascript pro vývoj webových aplikací s využitím frameworků. Podrobnější informace lze nalézt v knize pojednávající o efektivitě TypeScriptu [8].



Obrázek 2.3: Snímek obrazovky rozhraní anotačního nástroje Label Studio. V levém menu je zobrazen seznam datových záznamů k anotaci. V pravém menu lze získat informace o historii anotování podle jednotlivých uživatelů a jejich komentáře. Jediným nástrojem pro zaznačování anotací v tomto režimu je vytváření polygonů po kliknutí. Výběr třídy anotace je kvůli špatnému návrhu UI skryt pod obrázkem a lze jej zobrazit pomocí posunutí obrazovky.

Frameworky pro vývoj webových aplikací

Frameworky pro vývoj moderních webových aplikací jsou knihovny nebo sady nástrojů, které poskytují strukturu a nástroje pro vývoj webových aplikací. Tyto frameworky zjednodušují vývoj tím, že poskytují jednotný způsob práce se sítí, databází, uživatelským rozhraním a dalšími aspekty vývoje webových aplikací. Populární JavaScriptové frameworky pro vývoj webových aplikací jsou např. React, Angular a Vue.js. Tyto frameworky poskytují komponentový přístup k vývoji uživatelského rozhraní, což umožňuje snadné znovupoužití kódu a snižuje složitost vývoje. V každém případě, volba frameworku závisí na specifických požadavcích aplikace a preferencích vývojáře. Důležité je, aby framework poskytoval dostatek podpory a dokumentace, aby bylo možné rychle a efektivně vyvíjet aplikaci. Více informací obsahuje kniha zaměřená na JavaScriptové frameworky a moderní vývoj webu [7]

Angular je open source framework pro vývoj webových aplikací, který je založený na jazyce TypeScript. Používá se pro vytváření moderních a rychle se rozvíjejících webových aplikací s použitím komponentního designu. Angular poskytuje širokou škálu nástrojů a knihoven pro práci s daty, animacemi, formuláři a dalšími aspekty webových aplikací. Jeho vývoj a správa jsou zajišťovány skupinou vývojářů společnosti Google a jeho komunitou uživatelů.

Vue.js je open source framework pro vývoj webových aplikací, který se zaměřuje na snadnou a rychlou tvorbu interaktivních uživatelských rozhraní. Využívá se především pro vývoj aplikací s jednoduchou logikou a menšími požadavky na rychlost, ale lze jej použít i pro vývoj větších a složitějších aplikací. Vue.js se vyznačuje lehkou a přehlednou syntaxí a

poskytuje širokou škálu nástrojů pro práci s daty, animacemi a dalšími aspekty webových aplikací. Jeho vývoj a správa jsou zajišťovány skupinou vývojářů a jeho komunitou uživatelů.

React je open source JavaScript knihovna pro vývoj uživatelských rozhraní. Používá se především pro vývoj webových aplikací s velkými požadavky na rychlost a efektivitu. React se zaměřuje na tvorbu komponentního designu a umožňuje vývojářům snadno vytvářet a spravovat interaktivní uživatelská rozhraní. React poskytuje nástroje pro práci s daty a událostmi a umožňuje snadnou integraci s dalšími knihovnami a frameworky. Jeho vývoj a správa jsou zajišťovány společností Facebook a jeho komunitou uživatelů.

Technologie pro práci s grafikou ve webovém prostředí

Klíčovým prvkem editoru je zpracování obrazu kombinující vektorové a rastrové prostředí spolu s pokročilými transformacemi obrazu. Průzkum se zabývá technologiemi těchto oblastí. Editor má společné vlastnosti s herním průmyslem, ohledně kterého je možné čerpat relevantní informace v knize o tvorbě webových her [5].

WebGL (Web Graphics Library) je knihovna pro vykreslování 3D i 2D grafiky ve webových prohlížečích. WebGL je založen na OpenGL, což je standardní API pro vykreslování 3D grafiky, a umožňuje vývojářům vytvářet interaktivní 3D a 2D aplikace a hry pomocí JavaScriptu. WebGL poskytuje nástroje pro vykreslování 3D a 2D objektů, animací a efektů a umožňuje snadnou integraci s dalšími webovými technologiemi, jako je HTML5 a CSS. WebGL je podporován většinou současných webových prohlížečů a jeho vývoj a správa jsou zajišťovány komunitou vývojářů.

HTML canvas je element HTML, který umožňuje vývojářům vykreslovat grafiku pomocí JavaScriptu. Canvas se skládá z plochy, na kterou lze kreslit pomocí různých nástrojů, jako jsou tahy, obrázky a barvy. Canvas umožňuje vytváření interaktivních obrázků a animací a je často používán pro vývoj her nebo vizualizačních nástrojů. Canvas může být také použit pro zpracování a zobrazování velkých množství dat nebo pro tvorbu grafických efektů na webových stránkách.

Fabric.js je open source JavaScript knihovna pro tvorbu interaktivních grafických aplikací. Fabric.js se zaměřuje na snadnou tvorbu a úpravu objektů v canvasu, což je element HTML pro vykreslování grafiky pomocí JavaScriptu. Fabric.js poskytuje širokou škálu nástrojů pro práci s objekty, jako jsou tahy, obrázky, texty a animace, a umožňuje vývojářům snadno tvořit interaktivní aplikace a hry. Fabric.js je podporován většinou současných webových prohlížečů a jeho vývoj a správa jsou zajišťovány jeho komunitou uživatelů.

Konva.js je open source JavaScript knihovna pro tvorbu interaktivních grafických aplikací. Konva.js se zaměřuje na snadnou tvorbu a úpravu objektů v canvasu, což je element HTML pro vykreslování grafiky pomocí JavaScriptu. Konva.js poskytuje širokou škálu nástrojů pro práci s objekty, jako jsou tahy, obrázky, texty a animace, a umožňuje vývojářům snadno tvořit interaktivní aplikace a hry. Konva.js je podporován většinou současných webových prohlížečů a jeho vývoj a správa jsou zajišťovány jeho komunitou uživatelů.

OpenCV (Open Source Computer Vision) je platformově nezávislá knihovna volně dostupných softwarových nástrojů pro práci s obrazem a videem. Byla vytvořena pro vývoj aplikací z oblasti počítačového vidění, které se zaměřují na zpracování digitálních obrazů a videí za účelem získání informací z nich. OpenCV obsahuje širokou škálu algoritmů pro práci s obrazem, včetně detekce tváří, rozpoznávání objektů, sledování pohybu, extrakci vlastností obrazu a mnoho dalšího. Tyto algoritmy jsou implementovány v jazyce C++ a jsou dostupné jako knihovna pro vývojáře, kteří chtějí tyto funkce integrovat do svých aplikací. Je široce

používána ve výzkumu a průmyslu a je jednou z nejpopulárnějších knihoven pro počítačové vidění.

Technologie pro server aplikaci

Existuje velké množství různých technologií pro zpracování požadavků na straně serveru. Práce se zabývá primárně knihovnou, proto je uveden pouze omezený výběr.

Node.js je open-source, cross-platform runtime prostředí, které je navržené pro spouštění JavaScriptu na serveru. Využívá výkonný JavaScriptový engine V8 od společnosti Google a umožňuje vývojářům psát server-side aplikace v JavaScriptu. Node.js je často používán pro webové aplikace, ale může být použit pro jakýkoli typ aplikace, který vyžaduje spolehlivé a rychlé načítání dat. Díky svému modulárnímu designu a rozsáhlé komunitě kontribucí je Node.js také velmi flexibilní a může být snadno rozšířen pomocí externích balíčků.

Python je programovací jazyk, který byl vytvořen v roce 1989 Guido van Rossumem. Jeho hlavní výhodou je přehlednost a snadnost použití, což ho činí vhodným pro začátečníky i pokročilé programátory. Python je také velmi flexibilní a podporuje mnoho různých programovacích paradigmat, včetně objektově orientovaného, funkčního a imperativního programování. Kromě toho má Python širokou škálu knihoven a modulů, které umožňují snadnou integraci s jinými aplikacemi a službami. To z něj činí vhodný nástroj pro mnoho různých úkolů, včetně webového vývoje, analýzy dat, automatizace úloh a mnoho dalšího. Python je také velmi populární a má velkou komunitu nadšenců, kteří se aktivně podílejí na vývoji a rozšiřování jeho možností. To zajišťuje, že Python je stále aktuální a inovativní nástroj pro programování.

Technologie pro zpracování obrazu v datovém skladu

Existuje široká škála technologií pro zpracování obrazu na straně serveru pro různé jazyky. Práce se zabývá primárně knihovnou, proto jsou uvedeny pouze technologie pro výše popsané jazyky JavaScript a Python.

Sharp je knihovna pro Node.js, která umožňuje snadné zpracování a manipulaci s obrázky. Umožňuje například načítání obrázků, úpravu jejich velikosti, přidávání filtrů nebo změnu formátu. Je to výkonný nástroj pro práci s obrázky v Node.js a může se hodit při vývoji webových aplikací nebo jiných projektů, které pracují s obrázky.

Pillow je knihovna pro Python, která umožňuje práci s obrázky a grafikou. Poskytuje širokou škálu funkcí pro manipulaci s obrázky, včetně ořezávání, otáčení, úpravy velikosti a mnoha dalších. Tato knihovna je navržena tak, aby byla snadno použitelná a je založena na populární knihovně Python Imaging Library (PIL), která již několik let poskytuje podobné funkce. Pillow je k dispozici jako volně dostupný software pod licencí MIT a je široce používán v mnoha různých oblastech, jako je webový vývoj, vývoj her a vědecká práce s obrázky.

Kapitola 3

Návrh editoru, demonstrační aplikace a datového skladu

Kapitola se zabývá následujícím využitím knihovny editoru. Knihovnu editoru je možné využít ve vlastní aplikaci, která může být například jednoduchým grafickým editorem nebo pokročilým informačním systémem spravujícím uživatele, datové sady a anotace dat provedené uživateli. Cílem této práce je vytvoření knihovny editoru, proto obsáhlý systém umožňující správu dat a uživatelů nebude implementován, ale bude nastíněn diagramem a popisem případů užití. Implementace takového systému je navíc časově velmi náročná a liší se vždy dle konkrétních požadavků, jenž může být i integrace do současného informačního systému. Pro účely testování a demonstrace funkcionalit knihovny bude však navrhnut a implementován demonstrační systém skládající se z demonstrační aplikace využívající knihovnu editoru a demonstračního datového skladu.

3.1 Návrh případů užití systému předpokládaného využití knihovny editoru

Systém obsluhují dva aktéři: anotátor a technolog, který zároveň může vykonávat akce anotátora. Technolog nejprve vytvoří uživatele, datové sady, nahraje data do datových sad a přiřadí uživatele datovým sadám podle toho, jaké datové sady mají anotovat. Uživatelé si poté zobrazí seznam jim přiřazených datových sad a rezervují si anotaci z některé z nich. Rezervace anotace způsobí, že jiný uživatel nemůže nad tímto konkrétním datovým záznamem provádět anotaci. Pokud si uživatel rezervaci anotace rozmyslí, může ji zrušit. Po dokončení anotace odesílá svá data pomocí akce dokončení anotace. Technolog poté zobrazí datové sady, z nichž vybere jednu, zobrazí její anotace a může je buď potvrdit jako validní nebo je invalidovat, což způsobí, že datový podklad pro anotaci se opět uvolní pro rezervaci. Pro manažerské účely je možné zobrazit statistiky jednotlivých uživatelů, které udávají počet úspěšných a neúspěšných anotací a celkový čas strávený anotováním v určeném časovém intervalu. Přehled všech případů užití viz diagram případů užití 3.1.

3.2 Použité technologie a zdůvodnění jejich výběru

Tato sekce se zabývá výběrem z dostupných technologií pro implementaci 2D editoru jako webové aplikace. Nyní se zaměříme na ty, které jsou pro implementaci našeho editoru nej-

vhodnější. Podrobněji se seznámíme s jednotlivými technologiemi a uvedeme důvody jejich výběru pro naši webovou aplikaci.

Jazyk pro vývoj knihovny a ukázkové aplikace

Základem pro implementaci bude TypeScript. TypeScript je superset JavaScriptu, což znamená, že všechny funkce JavaScriptu jsou také dostupné v TypeScriptu. Použití TypeScriptu může pomoci vylepšit kvalitu, čitelnost a předvídatelnost kódu, protože TypeScript nabízí silnou typovou kontrolu. Hlavní součást této diplomové práce – knihovna, je znovupoužitelným kódem, proto je jazyk s typovou kontrolou vhodný.

Framework pro vývoj knihovny a ukázkové aplikace

Pro usnadnění vývoje bude využit framework React. Oproti čistému JavaScriptu/TypeScriptu nabízí životní cyklus jednotlivých komponent, což umožňuje jednoduchou efektivní správu zdrojů. React toho dosahuje pomocí virtuálního stromu DOM, kterým ostatní popisované frameworky nedisponují. Navíc oproti ostatním frameworkům je v době vzniku této práce několikanásobně používanější než Angular a Vue dohromady.

Technologie pro práci s grafikou ve webovém prostředí

Pro výslednou knihovnu je potřeba funkce zoom a práce s vrstvami, které Konva.js přímo nabízí narozdíl od Fabric.js. Navíc je navržena na výkon, což dokládají zátěžové testy přímo na stránkách s dokumentací knihovny[1].

Vzhledem k tomu, že Konva.js podporuje pouze vektorovou grafiku, bude pro implementaci rastrových vrstev využito HTML canvas elementů.

WebGL nebylo zvažováno, protože jeho API je velice nízkoúrovňové a realizace práce by se prodloužila o velké množství času.

Technologie pro server aplikaci

Pro implementaci ukázkové server aplikace bude použit Node.js, protože využívá JavaScript jako výsledná knihovna této práce. Aplikace je ukázková a jednoúčelová, proto výběr technologie nehraje velkou roli.

Technologie pro zpracování obrazu v server aplikaci

Knihovna Sharp bude použita pro svou možnost vytvářet výřezy a ty případně podvzorkovávat, což bude nutné pro implementaci funkce zoom s dynamickým škálováním a mezipamětí pro úsporu přenosu dat. Tato knihovna byla zvolena, protože disponuje potřebnými funkcemi pro vytváření škálovaných dlaždic a je to nejpoužívanější knihovna pro zpracování obrazu v rámci repozitářů npm.

3.3 Architektura výsledného demonstračního systému

Výsledný demonstrační systém se sestává z ukázkové aplikace, která využívá knihovnu editoru. Knihovna je samostatným balíčkem, který je v aplikaci použit pomocí balíčkovacího systému. Knihovna editoru pomocí nativních JavaScript funkcí žádá obrazová data dle

[1] `https://konvajs.org/docs/sandbox/Drag_and_Drop_Stress_Test.html`

zadaných parametrů od aplikace, která komunikuje s datovým skladem pomocí HTTP protokolu a předává získaná data zpět knihovně. Parametrizace umožňuje získávat škálované výřezy obrazových dat. Náhled celé architektury viz diagram 3.2.

3.4 Funkce editoru

Plátno editoru se sestává z libovolného počtu vrstev daných typů. Velikost vrstev je společná pro všechny vrstvy a lze ji nastavit. Očekávané použití je získání informace o rozlišení externích rastrových vrstev z datového skladu a nastavení těchto rozměrů jako globálního rozměru všech vrstev. Vrstvám je možné měnit jejich pořadí, průhlednost a případně je úplně skrýt. Jednotlivé typy vrstev jsou reprezentovány jako různé komponenty, podle toho, jestli jsou vytvořené, stažené z datového skladu a dále podle toho, jestli jsou rastrové nebo vektorové. Zoom i posuv plátna je pro všechny vrstvy společný, přičemž jednotlivé typy vrstev mohou mít pro zoom vlastní logiku.

Externí rastrové vrstvy

Externí rastrové vrstvy jsou vzdáleně načítaná obrazová data. Tyto vrstvy mohou v oblasti neuronových sítí nabývat velkých rozměrů, což způsobuje přenos objemných dat. Pro dosažení požadované úspory dat je získávání dat optimalizováno pomocí rozdělení na dlaždice, které se ukládají u klienta a v případě požadavku na stejnou dlaždici stejné vrstvy je použita tato záloha. Pro vrstvy, které jsou reprezentovány jako stupně šedi je možné nastavit přebarvení. Praktickým příkladem použití je zobrazení tepelné mapy pro inferenční vrstvy neuronových sítí. Příklady přebarvených výstupů neuronových sítí viz mřížka příkladů 3.3. Díky tomuto mapování jsou pro člověka obvykle lépe viditelné extrémy a šum výstupních dat inference. Některé vrstvy mohou být použity i jako barevné masky. Pro lepší přehlednost je možné tyto vrstvy šrafovat se zadanými parametry.

Dlaždicové rozdělení externí rastrové vrstvy s mezipamětí

Externí rastrová vrstva je při získávání z datového skladu dělena do čtvercových dlaždic, které jsou po stažení uloženy v paměti prohlížeče. Získávají se pouze dlaždice, které zasahují do zobrazeného výřezu vrstev. V případě, že byla dlaždice v minulosti již získána, tak se získá z lokální paměti. Počet dlaždic na šířku a výšku je dán dynamicky podle úrovně přiblížení a zadaných parametrů.

Externí vektorové vrstvy

Tyto vrstvy mohou sloužit pro případné rozšíření informační hodnoty výsledků neuronové sítě, přičemž není nutné získávat rastrovou vrstvu s plným rozlišením. Vrstvy jsou stahovány ve formátu SVG, který je velmi rozšířený ve světě vektorové 2D grafiky. Prvky SVG je možné za běhu upravovat, díky čemuž je možné upravit data pro konkrétní nasazení aplikace, aniž by bylo třeba modifikovat výstup neuronové sítě. Při implementaci vlastní aplikace je díky standardizovanému formátu snadné vytvořit vlastní logiku parsování elementů a změny jejich konfigurace v aplikačním rozhraní.

Interní rastrové vrstvy

Vytvořené rastrové vrstvy umožňují kreslení a mazání na úrovni jednotlivých pixelů. Tento přístup je vhodný pro přesné a detailní anotace. Požadovanými nástroji jsou štětec, který má nastavitelný rozměr, tvar, průhlednost a barvu, kyblík poskytující funkci záplavového vyplňování dle zadané barvy a průhlednosti, mazání s nastavitelným rozměrem a tvarem a nástrojem pro označení a vymazání oblasti.

Interní vektorové vrstvy

Vytvořené vektorové vrstvy mohou sloužit pro méně detailní a rychlejší anotace. Do těchto vrstev je možné kreslit pomocí nástroje pro vytváření lomených čar (pro složitější tvary) a jednoduchých přímek, kruhů a čtyřúhelníků (například jako bounding box). Všechny tyto nástroje mají nastavitelnou šířku, barvu a průhlednost. Každý prvek je možné posouvat, případně odstranit. Nástroj pro vytváření lomených čar navíc obsahuje editační režim, který umožňuje přesouvat, mazat a přidávat body do editované lomené čáry. Vrstvu je také možné převést na vytvořenou rastrovou vrstvu.

Mřížka

Pro účely detailního anotovaní editor disponuje mřížkou ohraničující jednotlivé pixely rastrových vrstev. Mřížka jakožto vestavěná vektorová vrstva se zobrazí po dosažení určitého přiblížení dle parametru. Mřížce je možné nastavit rozměry a barvu pomocí parametrů komponenty editoru.

Kurzor

Další vestavěnou pomocnou vrstvou je rastrová vrstva pro kurzor při vybraném nástroji pro kreslení do vytvořené rastrové vrstvy. Tato vrstva zobrazuje teoretickou stopu štětce.

Práce s bloby

Po každé změně ve vytvořené rastrové vrstvě jsou na této vrstvě nalezeny bloby a označeny novým ID. Bloby jsou nacházeny jako spojité části obrazových dat se stejnou barvou. Jednotlivé barvy jsou specifikovány parametrem editoru.

3.5 Demonstrační systém editoru

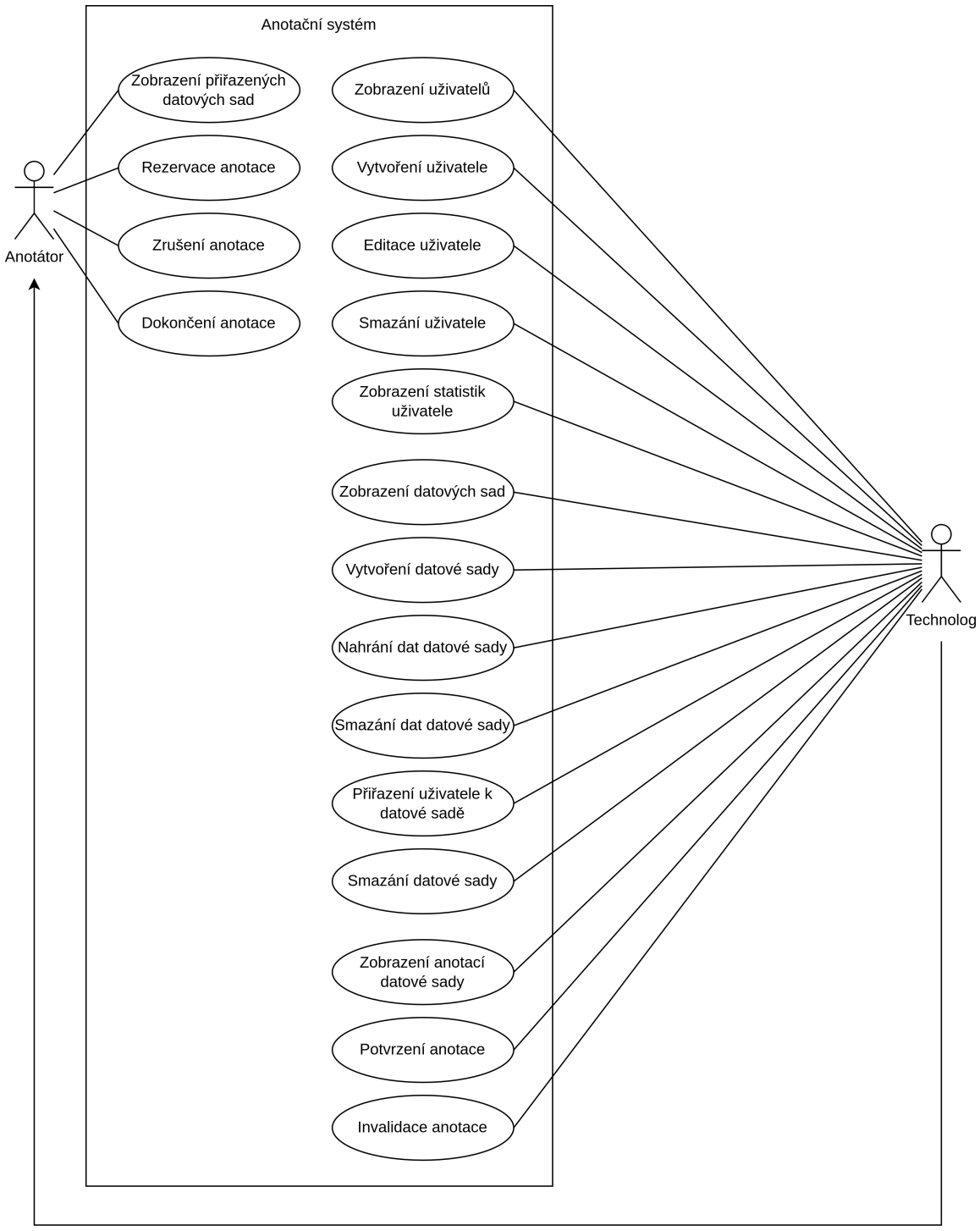
Pro účely prezentace a ladění knihovny editoru slouží demonstrační systém zahrnující tuto knihovnu. Skládá se ze dvou částí, a to webové aplikace jakožto uživatelského rozhraní a datového skladu pro získávání externích vrstev.

Demonstrační webová aplikace

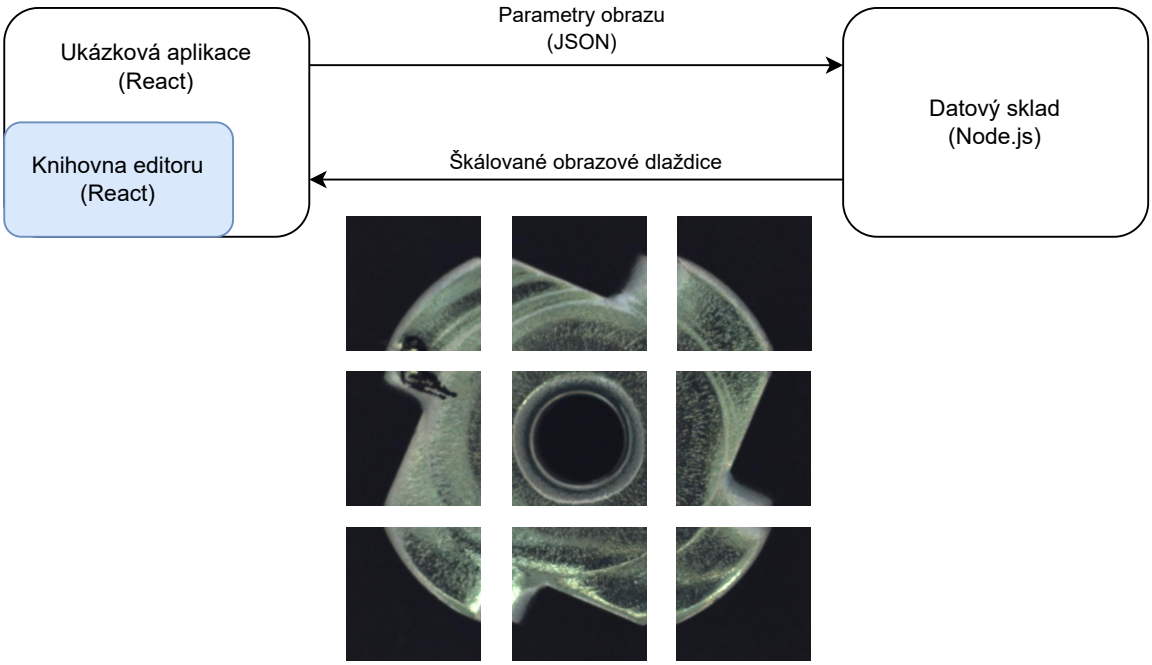
Testovací aplikace reprezentuje možný způsob využití knihovny editoru. Aplikace implementuje rozhraní mezi uživatelem a knihovnou a slouží jako prostředník pro komunikaci editoru s datovým skladem. Je zaměřena více na prezentační a vývojové účely, proto není zahrnuta optimalizace uživatelského zážitku.

Demonstrační zdroj dat

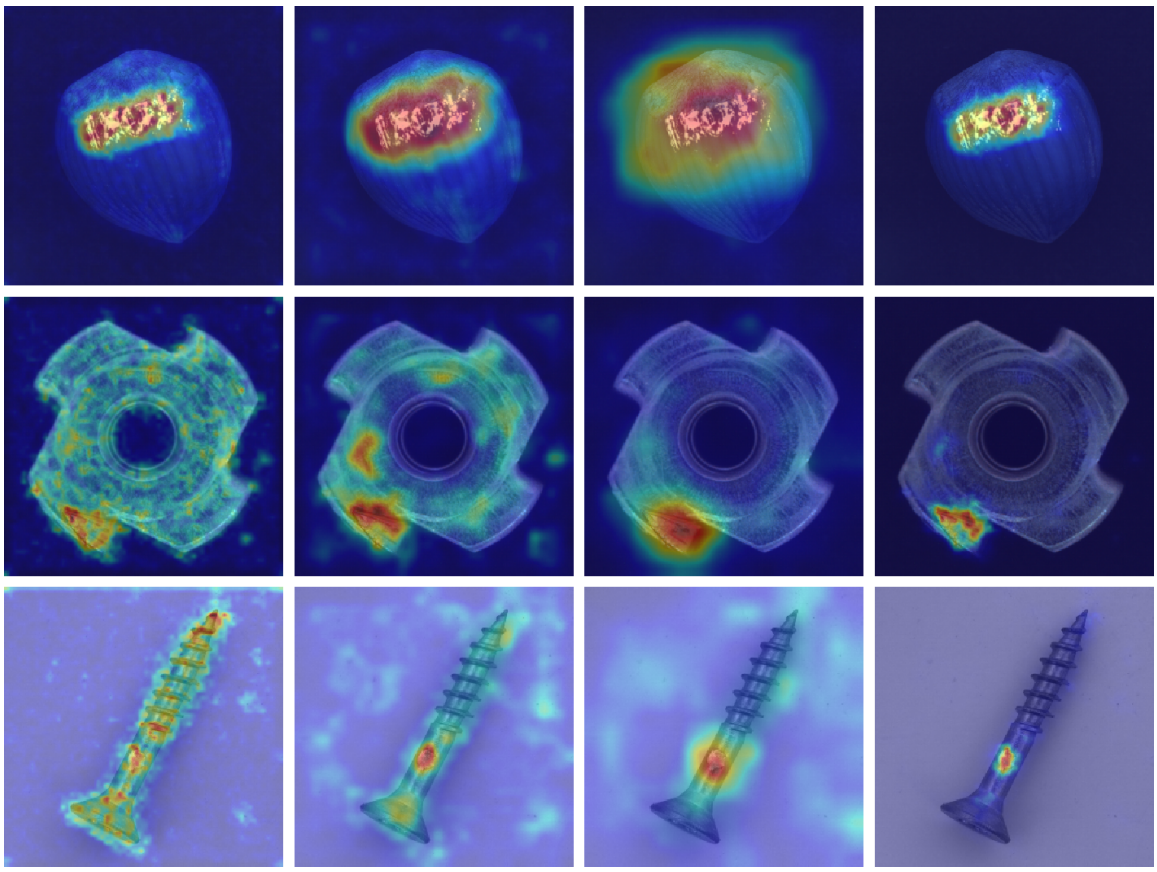
Získávání externích vrstev jak rastrových, tak vektorových probíhá z externí aplikace. Datový sklad implementuje rozhraní nutné pro správnou funkci obou typů vrstev. Rastrová data jsou poskytována jako škálované výřezy obrázků a vektorová data ve formátu SVG reprezentovaná jako textový řetězec.



Obrázek 3.1: Diagram případů užití systému předpokládaného využití knihovny editoru. Zahrnuje případy použití pro systém, který zahrnuje správu uživatelů, datových sad, anotování dat anotátory a kontroly jejich práce technologem.

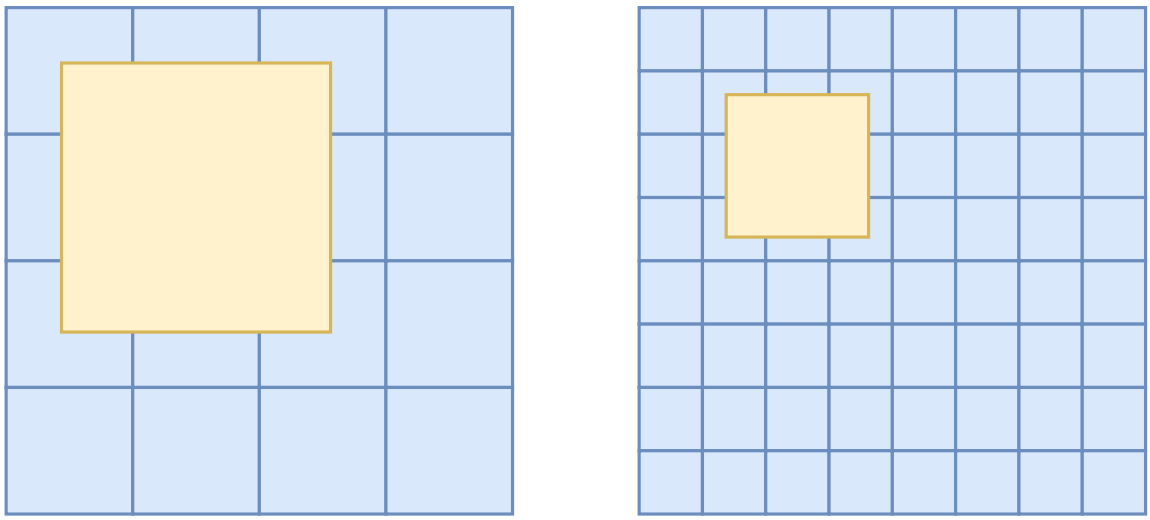


Obrázek 3.2: Diagram architektury výsledného demonstračního systému. Diagram se skládá ze dvou bloků, kde levý blok reprezentuje frontend aplikaci a pravý blok backend aplikaci. Frontend posílá backendu požadavky na získání obrazových dat ve formátu JSON a backend odpovídá obrazovými daty zpracovanými jako škálované dlaždice dle získaných parametrů.



Obrázek 3.3: Příklady přebarvení inference neuronové sítě tepelnou mapou demonstrující snadnou rozpoznatelnost šumu a extrémů výstupu neuronové sítě. Výsledky inference jsou převzaty z článku konference The British Machine Vision Conference [9]

.



Obrázek 3.4: Diagram rozdělení externí rastrové vrstvy do dlaždic při různých úrovních přiblížení. Čím větší je úroveň přiblížení, tím více dlaždic z původního obrázku vznikne. Modré čtverce symbolizují dlaždice a žlutý obdélník je oblast zobrazeného výřezu vrstev.

Kapitola 4

Implementace editoru, demonstrační aplikace a datového skladu

Tato kapitola se zabývá implementací celého systému, jeho sestavením a jeho následném testování. Knihovna editoru i testovací frontend aplikace je implementována v jazyce TypeScript s využitím frameworku React a komunikuje synchronně s datovým skladem pomocí knihovny `axios` založené na protokolu HTTP. Implementační framework pro datový sklad byl zvolena Node.js, protože pro něj lze taktéž psát v jazyce TypeScript. Vizualizační aplikace pro zobrazení výsledků testů na úniky paměti je psána v čistém JavaScriptu, kvůli své jednoduchosti. Díky tomu není třeba pro aplikaci vytvářet prostředí pro převod z jazyka TypeScript do jazyka JavaScript.

4.1 Implementace a sestavení knihovny editoru

Knihovna je implementována jako balíček, který je po sestavení možné nainstalovat pomocí balíčkovacích nástrojů jako je npm nebo yarn. Výsledný balíček je sestaven pomocí balíčku `parcel`, který je ve vývojářských závislostech společně s jeho rozšířeními `@parcel/packager-ts` a `@parcel/transformer-typescript-types` pro propagování typů v rámci jazyka TypeScript i po sestavení. Výsledný balíček exportuje TypeScript typy, React komponenty a funkce, které budou popsány v samostatné sekci s jejich detailním rozhraním. Veškeré snímky obrazovky demonstrační aplikace obsahují buď vlastní data nebo data z článků společnosti MVTec [1, 2].

4.1.1 Architektura knihovny editoru

Knihovna exportuje samotnou komponentu s plátnem **AnnotationCanvas**. Tuto komponentu je nutné obalit providery pro kontexty. Je tedy možné vytvořit více komponent s plátnem sdílející některé nebo všechny z kontextů. **LayersProvider** slouží pro uchovávání aktuálního stavu vrstev a poskytuje rozhraní pro manipulaci s nimi. Obsahuje též historii úprav vrstev a rozhraní pro vrácení akcí zpět, případně zrušení vrácení akce. Při sdílení více komponentami budou všechna plátna vykreslovat stejný obraz, který však může být různě posunut, škálován či přiblížen. **ToolProvider** uchovává a poskytuje informace o aktuálně zvoleném nástroji a jeho parametrech. Sdílení mezi více komponentami způsobí

jednotné přepínání nástrojů pro všechny tyto komponenty. **ImageCacheProvider** slouží jako prostředník pro komunikaci s datovým skladem. Získává škálované dlaždice externích rastrových vrstev, které ukládá do paměti prohlížeče, aby je nebylo třeba znovu získávat z datového skladu. Tento provider dává největší smysl sdílet napříč celou aplikací pro maximalizaci úspory přenosu dat.

4.1.2 Implementace knihovny editoru

Použité externí knihovny:

- **@techstark/opencv-js** – 4.7.0-release.1 – knihovna pro práci s počítačovým viděním převedená do JavaScriptu. Využita pro hledání blobů.
- **konva** – 8.4.2 – knihovna pro práci s grafikou ve webovém prostředí. Využita pro vytvoření plátna a jejími transformacemi. Přímo implementuje vytvořené vektorové vrstvy a pomáhá se zobrazováním ostatních vrstev.
- **localforage** – 1.10.0 – knihovna vytvářející slovníkovou abstrakci nad IndexedDB úložištěm prohlížeče. Využita pro ukládání historie úprav vrstev a dlaždic získaných z datového skladu.
- **lodash** – 4.17.21 – knihovna poskytující pomocné funkce pro JavaScript. Využita pro funkci hluboké kopie objektů.
- **react** – 18.2.0
- **react-dom** – 18.2.0
- **react-konva** – 18.2.5 – knihovna poskytující abstrakce knihovny Konva.js pro framework React. Použita pro většinu manipulace s objekty knihovny Konva.js.

AnnotationCanvas komponenta

Tato komponenta obsahuje hierarchii jednotlivých vrstev a manipulaci s nimi jako s celkem v rámci plátna. Vnořené komponenty reprezentující jednotlivé vrstvy nejsou v rámci aplikace, kde je knihovna použita viditelné. Implementace jednotlivých vrstev bude popsána samostatně dále.

Parametry:

- **onPointerMove** – nepovinný parametr – funkce zpětného volání, která je volána vždy po pohybu kurzoru po plátně s předanými souřadnicemi kurzoru relativně k rastrové vrstvě.
- **onZoomChange** – nepovinný parametr – funkce zpětného volání, která je volána vždy po změně přiblížení plátna s předáním měřítka a posunu ve výšce a šířce od levého horního okraje plátna.
- **gridEnabled** – nepovinný parametr – implicitně true – je to pravdivostní hodnota, která přepíná zobrazování vektorové mřížky kolem rastru po dosažení přiblížení plátna na parametr gridScale.
- **gridScale** – nepovinný parametr – implicitně 6 – číslo, které udává práh pro zobrazení vektorové mřížky kolem rastru. Pokud je aktuální přiblížení plátna větší nebo rovno této hodnotě a parametr gridEnabled je nastaven na pravdu, je mřížka zobrazena.

- **gridLine** – nepovinný parametr – implicitně { stroke: "white", strokeWidth: 0.05 } – objekt, který definuje vlastnosti čar vektorové mřížky kolem rastru. Vlastnosti čar jsou definovány podle knihovny Konva.js, viz její dokumentace[1].
- **disableZoom** – nepovinný parametr – implicitně false – pravdivostní hodnota, která přepíná možnost změny měřítka a pozice plátna pomocí akce scrollování.
- **blobColors** – nepovinný parametr – pole objektů, které definují pomocí číselné hodnoty barevných složek červené, zelené a modré které barvy blobů mají být v rastrových vrstvách hledány a označovány.

Responzivita je zajištěna tím, že komponenta při inicializaci nastaví pozorovatele změny rozměrů na okno editoru. Při každé změně rozměrů jsou přepočítány parametry aktuálního přiblížení a plátno je v okně vycentrováno.

Získávání koordinátů zobrazeného výřezu vrstev slouží pro další použití při optimalizaci datové náročnosti a optimalizaci vektorového vykreslování. Po každé změně pozice a úrovně přiblížení plátna jsou vypočteny koordináty rastrové oblasti v okně editoru. Výpočet zahrnuje situace, kdy pohled není částečně na vrstvách, je celý vně vrstev, je celý mezi hranicemi vrstev nebo přetéká všechny hranice vrstev. Vizualizace některých těchto případů viz diagram 4.1.

Změna přiblížení je řešena pro akci scrollování i nástroj přiblížení a oddálení společnou funkcí applyZoom, která jako parametr získává směr přiblížení a číselnou hodnotu mocnosti změny přiblížení. Tato funkce získává pozici kurzoru a provádí změnu přiblížení relativně vůči jeho pozici. Je limitována funkcí stageBound tak, aby v případě, že je plátno menší než okno editoru, bylo plátno vycentrováno a zároveň oddálení nemohlo dosáhnout úrovně, kdy by všechny rozměry plátna byly menší než rozměry okna – nevznikne bílé ohraničení kolem plátna v obou osách. Pro dotyková zařízení s funkcí pro snímání více dotyků v jeden okamžik je implementována funkce pinch, která je taktéž limitována funkcí stageBound.

Rastrové bloby jsou hledány po každé manipulaci s aktuálně vybranou rastrovou vrstvou. Pomocí funkce getContoursBoundingBoxes jsou získány ohraničující boxy všech blobů a do jejich středu jsou vektorově vykresleny popisky značící jejich barvu a pořadové číslo.

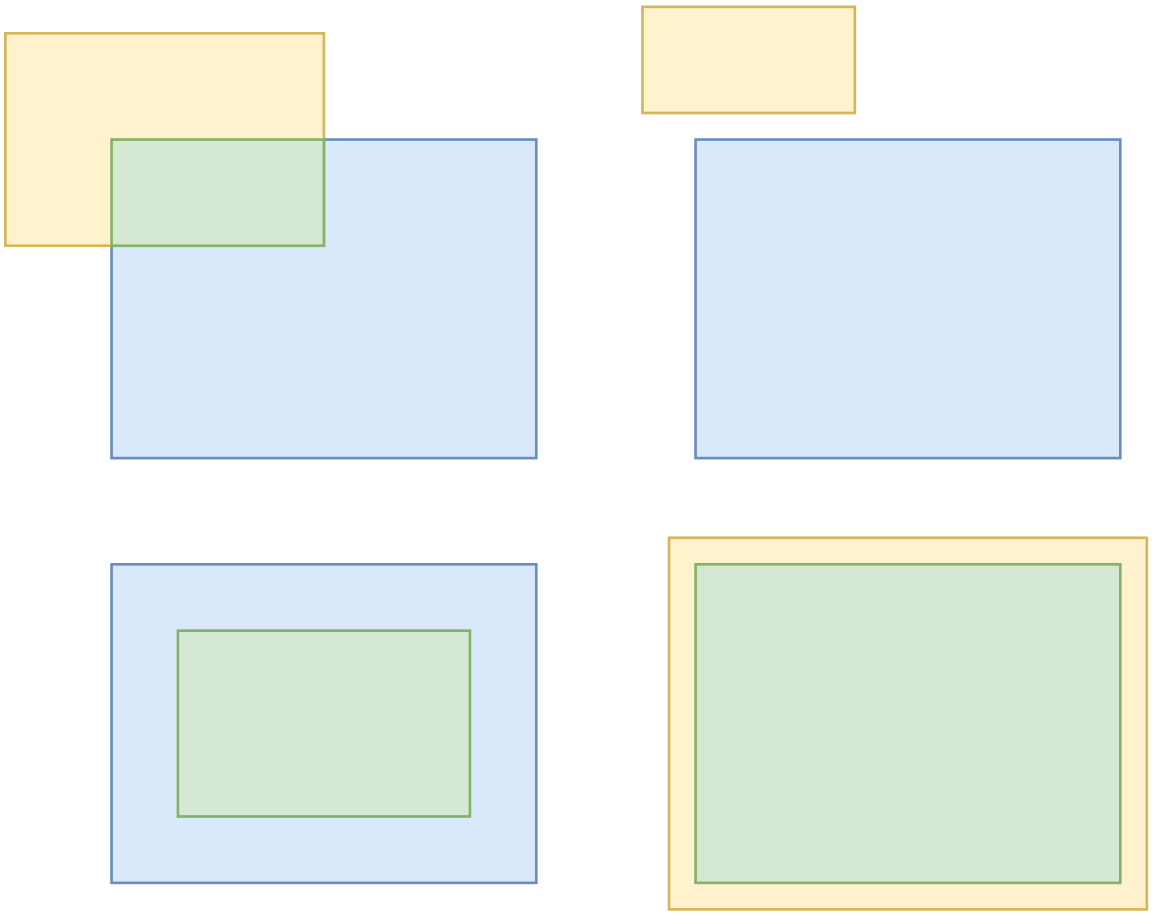
Posuv po plátně je řešen změnou posunu ve výšce a šířce od levého horního okraje plátna a je limitován funkcí stageBound tak jako u změny přiblížení.

Externí rastrové vrstvy

Externí rastrové vrstvy jsou vzdáleně načítaná obrazová data pomocí funkce předané knihovně editoru od aplikace, která ji využívá. Pro dosažení požadované úspory dat je získávání dat optimalizováno pomocí rozdělení na dlaždice s využitím mezipaměti. Rozměry dlaždic jsou ve stejném poměru stran a pokud není rozměr vrstev dělitelný rozměrem dlaždice, tak již přímo knihovna pro krajní dlaždice na dolní a pravé straně volá předanou funkci pro získávání dlaždic s ořezem odpovídajícímu rozměrům vrstev. Vizualizace výběru dlaždic podle zobrazeného výřezu vrstev viz diagram 4.2.

Pomyslná mřížka dlaždic je dělena pomocí zadaných parametrů downloadedRasterLevelSize a downloadedRasterMinTilesCount v LayersProvider. První z parametrů značí velikost intervalu na úrovni přiblížení, pro který platí jedna úroveň dlaždic. Poslední úroveň je na intervalu 1.0 – hodnota parametru až 1.0. Druhý parametr udává granularitu mřížky,

[1] `https://konvajs.org/api/Konva.Line.html`

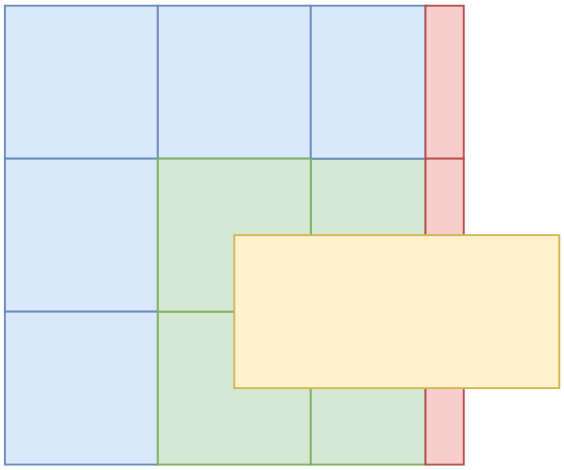


Obrázek 4.1: Diagram příkladů zobrazeného výřezu vrstev, kde modrý obdélník symbolizuje zobrazované vrstvy, žlutý obdélník symbolizuje okno editoru a zelený obdélník je počítaným výřezem. Získaný výřez je poté použit pro výpočet pozic konkrétních dlaždic, které je nutné načíst, aby byl obraz v okně editoru kompletní. Výřez je také použit pro optimalizaci vykreslování vektorové mřížky.

reprezentovanou minimálním počtem dlaždic, který je vykreslen v rámci okna editoru. Dlaždice jsou vždy buď ve vyšším nebo stejném rozlišení jako rozlišení aktuálně zobrazeného pohledu. Po dosažení stejné úrovně rozlišení displeje jako zdrojového obrázku se již další úroveň nevytváří. Další úrovně se nevytváří i opačným směrem, pokud je dosažena úroveň, kdy je v okně editoru zobrazen celý obrázek.

Každé získávání dlaždic po změně posunu plátna nebo změně jeho přiblížení je provedeno přes takzvanou debounce funkce s odkladem na jednu vteřinu. Dlaždice se získávají z ImageCacheProvider, který se stará o ukládání dlaždic do mezipaměti.

Dlaždice ve stupni šedi je možné po získání šrafovat nebo přebarvit podle zadaných parametrů hatching a heatmap. Parametr hatching obsahuje dvě hodnoty, které udávají šířku mezery a šířku vykreslené části. Parametr heatmap obsahuje pole 255 polí s jednotlivými barevnými složkami ve formátu RGB. Hodnota pixelu zdrojového obrázku je svou hodnotou mapována na toto pole a tím vzniká barevný obrázek. Každému pixelu je také nastavena hodnota průhlednosti na hodnotu původní úrovně hodnoty stupně šedi. Příklad přebarvení tepelnou mapou jet 4.4 viz snímek obrazovky 4.3.



Obrázek 4.2: Diagram výběru dlaždic podle zobrazeného výřezu vrstev, kde modré čtverce a obdélníky symbolizují dlaždice nebo jejich části pokrývající vrstvy, zelené čtverce a obdélníky reprezentují získávané dlaždice, červené obdélníky symbolizují přesah dlaždic přes vrstvy, který je ořezán a žlutý obdélník je oblast zobrazeného výřezu vrstev.

Externí vektorové vrstvy

Tyto vrstvy jsou získávány vzdáleně jako strom prvků ve formátu SVG. Knihovna editoru tyto prvky pouze vykreslí do plátna. Manipulaci s prvky je možné provádět externě v rámci implementující aplikace.

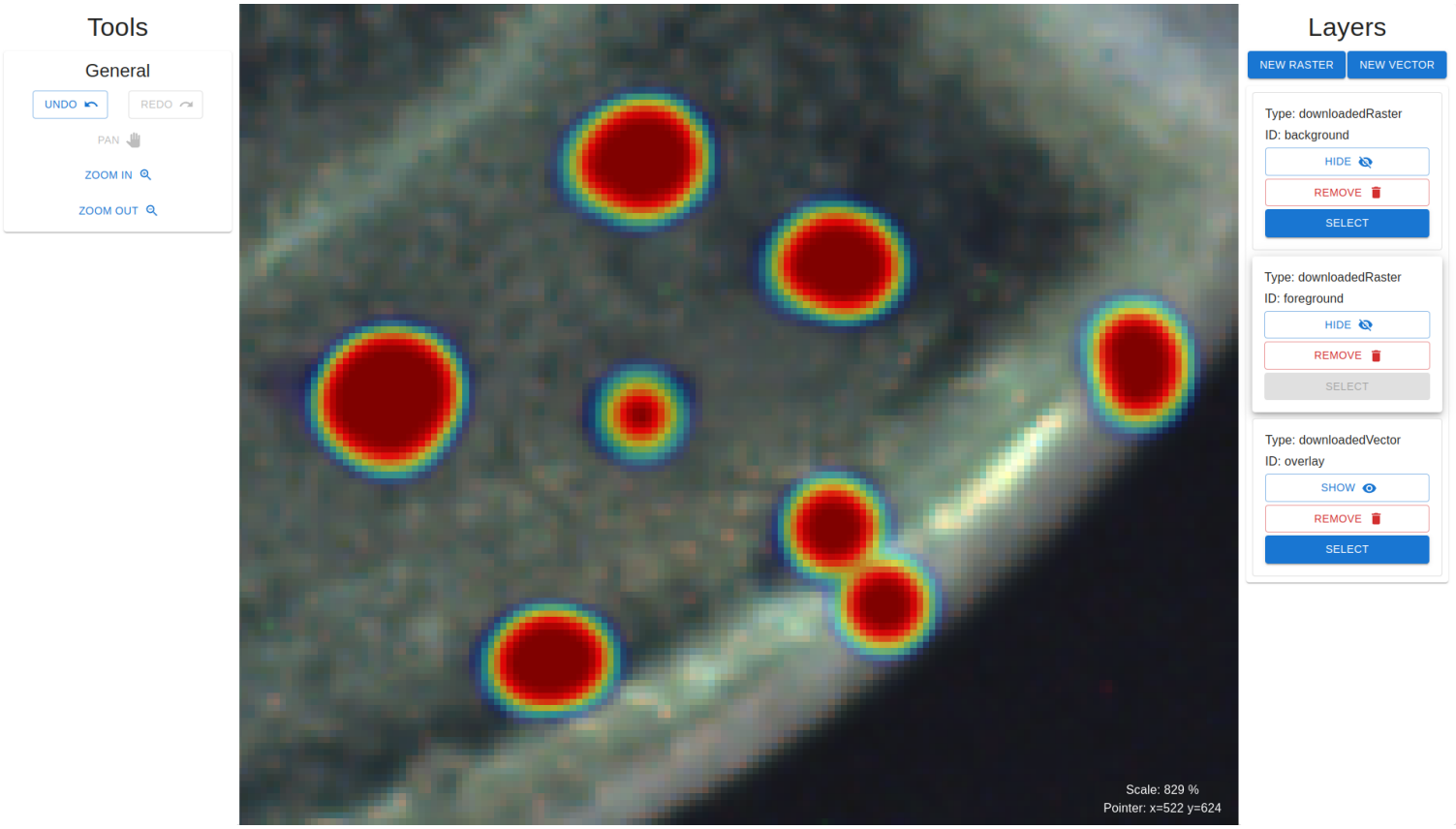
Interní rastrové vrstvy

Vytvořené rastrové vrstvy umožňují kreslení a mazání a jsou implementovány pomocí HTML canvas. Implementovanými nástroji jsou štětec, který má nastavitelný rozměr, tvar, průhlednost a barvu, kyblík poskytující funkci záplavového vyplňování dle zadané barvy a průhlednosti, mazání s nastavitelným rozměrem a tvarem a nástrojem pro označení a vymazání oblasti.

Vestavěné kreslení HTML canvas neumožňuje kreslit bez vyhlazování viz snímek obrazovky 4.5 a v případě použití průhlednosti se průhlednost sčítá viz snímek obrazovky 4.6, proto je implementováno vlastní kreslení pomocí Bresenhamova algoritmu pro kružnici a Bresenhamova algoritmu pro úsečku. Při tažení je mezi zachycenými body vykreslena úsečka, jejíž body jsou vykresleny jako kružnice. Pro optimalizaci rychlosti vykreslování plných kružnic jsou jednotlivé body kružnice nahrazeny čarami o horizontální délce kružnice v aktuální pozici. Pro čtvercový tvar štětce je využito přímo funkce vykreslení čtverce o velikosti nástroje.

Pro nástroj „kyblík“ je implementován záplavový algoritmus expandující do čtyřokolí. Demonstrace kroku algoritmu viz diagram 4.7. Pro optimalizaci výkonu jsou HTML canvas data, která jsou reprezentována jako pole 8bitových hodnot obsahující jednotlivé barevné kanály převedeny na pole 32bitových hodnot, které reprezentují všechny složky jednoho pixelu.

Nástroj pro vymazání oblasti využívá pomocné rastrové vrstvy kurzoru, která je popsána v sekci 4.1.2. Nástroj v této vrstvě vykresluje obdélník vyznačující oblast pro vymazání.



Obrázek 4.3: Snímek obrazovky demonstrační aplikace s nastaveným přebarvením vrstvy ve stupních šedi pomocí tepelné mapy.

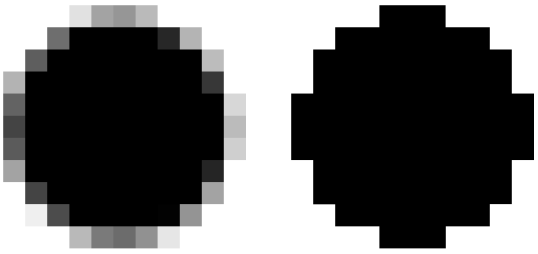
Po dokončení tahu pouze vymaže z vybrané vrstvy označený obdélník pomocí vestavěné funkce HTML canvas.

Interní vektorové vrstvy

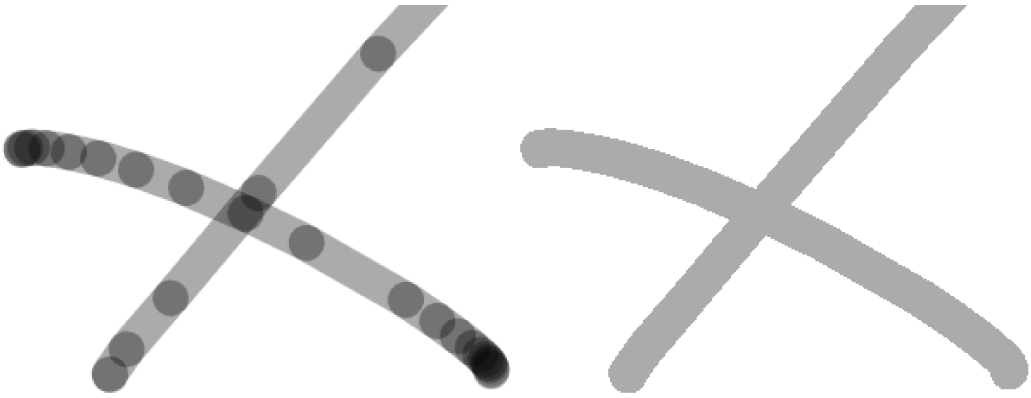
Do vytvořené vektorové vrstvy je možné kreslit pomocí lomených čar, kruhů a čtyřúhelníků, kdy všechny tyto nástroje mají nastavitelnou šířku, barvu a průhlednost. Jednotlivé objekty je možné posouvat a mazat pomocí samostatných nástrojů.

Kreslení lomených čar probíhá s pomocnou strukturou, která reprezentuje dočasnou lomenou čáru. Klikáním do plátna se vytváří nové body, které je možné poté přesouvat, případně dvojím kliknutím smazat. Po kliknutí na výchozí bod je lomená čára uzavřena a vložena do vybrané vrstvy. Lomenou čáru je možné po vytvoření stejným nástrojem znovu

Obrázek 4.4: Tepelná mapa použitá pro přebarvení vrstvy ve stupních šedi na snímku obrazovky 4.3.



Obrázek 4.5: Ukázka HTML canvas kreslení s nežádoucím antialiasingem vlevo. Požadovaný výstup je uveden vpravo.



Obrázek 4.6: Ukázka HTML canvas kreslení s nežádoucím aditivním kreslením vlevo. Požadovaný výstup je uveden vpravo.

vybrat a mazat nebo přemisťovat její body. Příklad editace i vytváření nové lomené čáry viz snímek obrazovky 4.8.

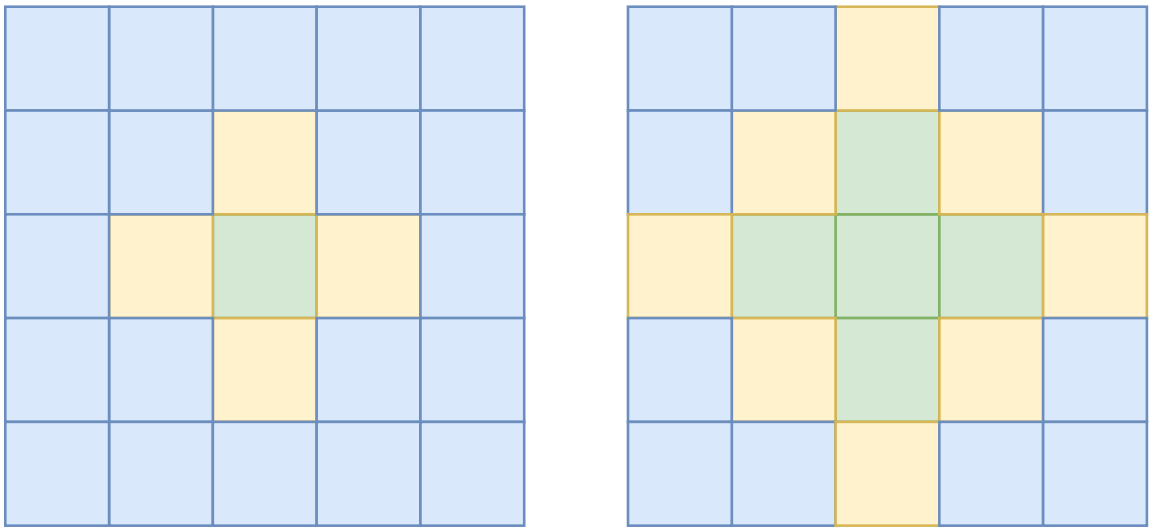
Ostatní prvky jsou kresleny tažením, kdy body prvků jsou dány začátkem a koncem tažení. U kružnice je prvním bodem její střed a druhým bod ležící na kružnici. Všechny tyto nástroje jsou omezeny funkcí, pro zamezení kreslení mimo plátno.

Rastrová vrstva s mřížkou

Při dosažení určitého přiblížení je zobrazena mřížka vektorová vrstva ohraničující jednotlivé pixely rastrových vrstev viz snímek obrazovky 4.9. Pro optimalizaci rychlosti vykreslení jsou rozměry mřížky omezeny pouze na zobrazený výřez vrstev. Vizualizace získávání výřezu viz diagram 4.2. Získaný výřez je dále omezen hranicemi plátna. Tloušťku čar mřížky a styl jejich vykreslování je možné parametrizovat přes parametry komponenty AnnotationCanvas.

Rastrová vrstva s kurzorem

Pokud je vybrán rastrový nástroj, zobrazí se vrstva vykreslující kurzor, která vykresluje jeho teoretickou stopu pod kurzorem. Tato vrstva je jako ostatní rastrové vrstvy reprezentována pomocí HTML canvas a podobně jako při kreslení do vytvořených rastrových vrstev je využito Bresenhamova algoritmu pro kružnici, případně vykreslení čtverce pro čtvercový štětec.



Obrázek 4.7: Diagram kroku záplavového algoritmu pro čtyřokolí, kde modré čtverce symbolizují pixely k obarvení, zelené čtverce reprezentují obarvené pixely, žluté čtverce jsou aktuálně obarvované pixely.

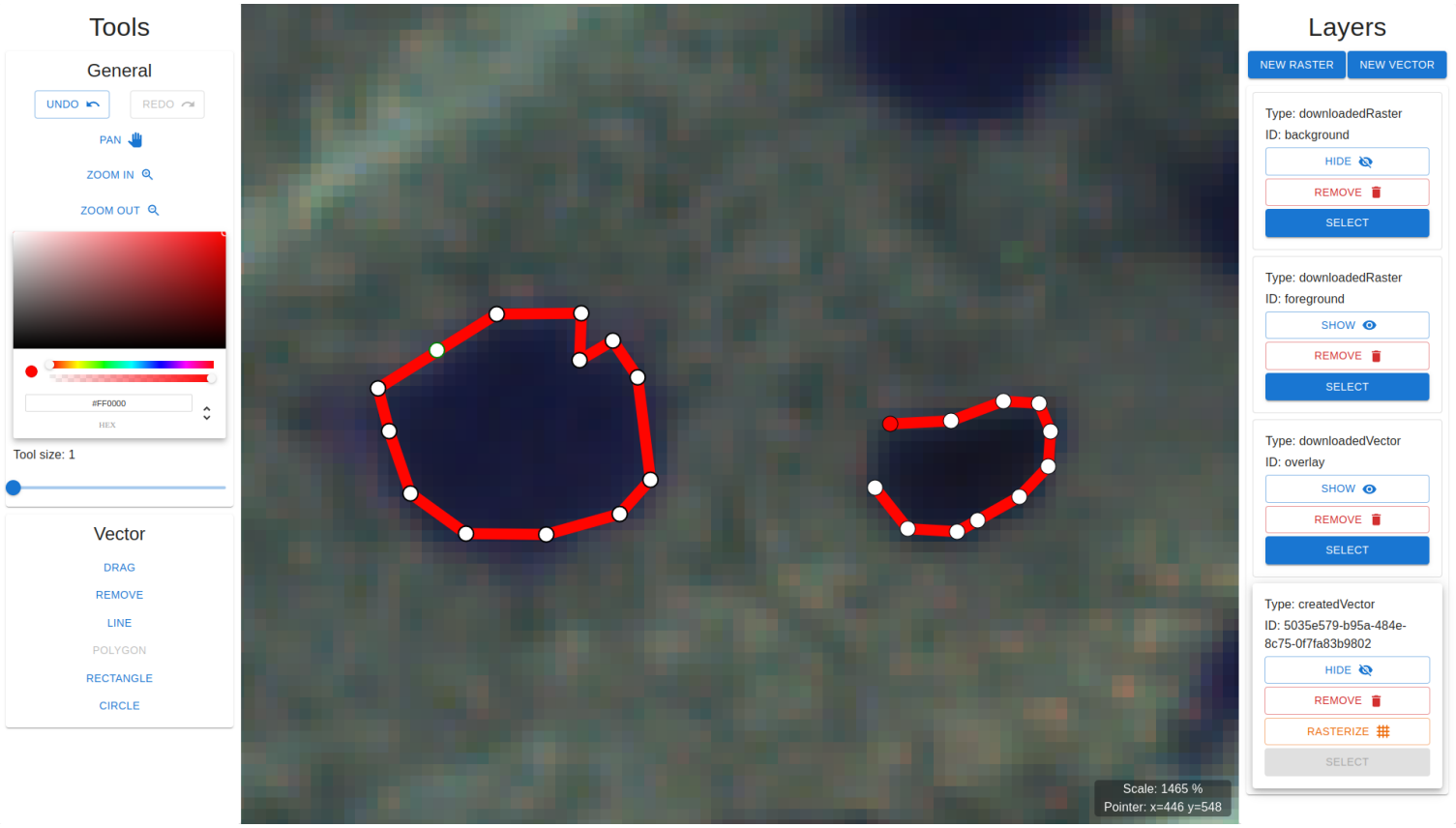
Označování blobů

Bloby jsou spojité části obrazových dat se stejnou barvou. Po každé změně ve vytvořené rastrové vrstvě jsou na této vrstvě pomocí knihovny OpenCV nalezeny kontury podle barev zadaných do parametru komponenty AnnotationCanvas. Kontury jsou získány funkcí findContours s parametrem RETR_CCOMP, který způsobí, že vrácenou strukturu kontur lze iterovat pouze po vnějších konturách, které reprezentují bloby viz dokumentace OpenCV[2]. Z vyfiltrovaných kontur jsou získány jejich bounding boxy, které jsou označeny novým ID a jsou vykresleny v samostatné vektorové vrstvě s popiskem obsahujícím jejich ID a barvu. Příklad označených blobů v implementované aplikaci viz snímek obrazovky 4.10.

LayersProvider

Provider pro vrstvy uchovává aktuální stav vrstev a poskytuje rozhraní pro manipulaci s nimi. Implementuje také manipulaci s historií úprav vrstev. Historie vrstev je reprezentována polem záznamů, ve kterých je uvedeno, jestli se jedná o vytvoření, editaci, smazání nebo rasterizaci dané vrstvy. Příklad převodu interní vektorové vrstvy na interní rastrovou vrstvu viz snímek obrazovky 4.11. Původní stav aplikace před rasterizací lze vidět na snímku obrazovky 4.8. Záznam dále obsahuje stav vrstvy po provedení změny, aby mohla být operace vrácena nebo zpětně provedena. Tato data jsou kvůli své velké datové náročnosti uchovávána v úložišti IndexedDB, se kterým je komunikováno pomocí abstrakce knihovny `localForage`. Každý záznam by měl obsahovat stav před i po vykonání operace, aby bylo možné provádět vrácení operace obousměrně. To by však vedlo k vytváření duplicitních dat, proto jsou uchovávány pouze stavy po provedení operace a stav předchozí operace je získáván z předchozí akce pro danou vrstvu. Pokud vrstva nemá předchozí akci, je uvedena do jejího iniciálního stavu. Po každé změně v historie jsou také přepočteny hodnoty undoAvailable a redoAvailable poskytované rozhraním.

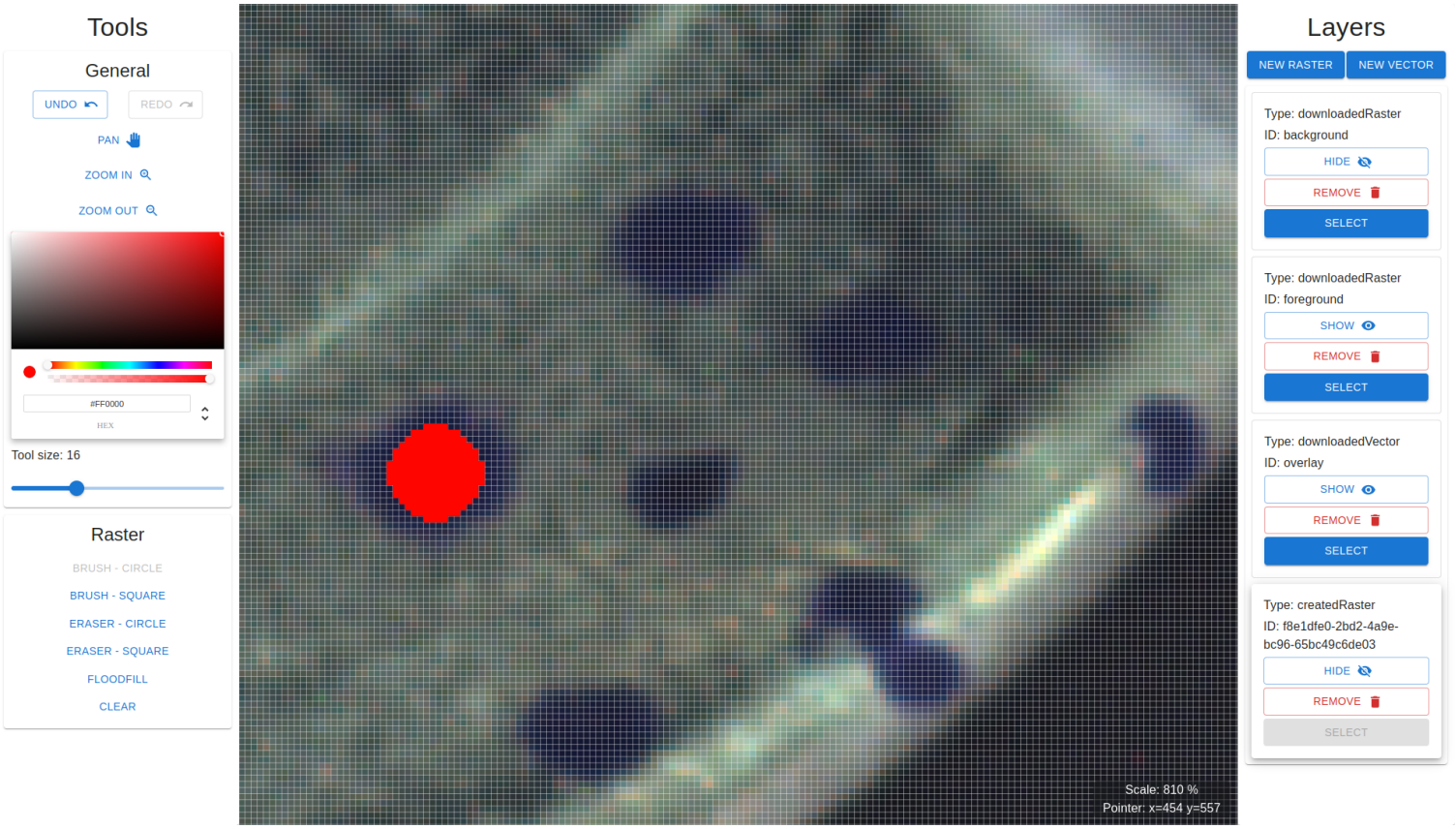
[2] `https://docs.opencv.org/4.x/d9/d8b/tutorial_py_contours_hierarchy.html`



Obrázek 4.8: Snímek obrazovky demonstrační aplikace s vlevo zobrazeným editačním módem lomené čáry, kde zeleně ohraničený bílý kruh označuje přidání nového bodu v poloze kurzoru a černě ohraničené bílé kruhy značí existující body, které je možné přesouvat a mazat. V pravé části je vytvářena lomená čára nová. Červený kruh je počátečním bodem, kde je možné lomenou čáru ukončit a bílé kruhy značí již vytvořené body, které je možné přesunout nebo smazat.

Parametry:

- **rasterWidth** – povinný parametr – číselná hodnota, která udává šířku všech vrstev, podle rozlišení rastrových vrstev.
- **rasterHeight** – povinný parametr – číselná hodnota, která udává výšku všech vrstev, podle rozlišení rastrových vrstev.
- **layers** – nepovinný parametr – pole vrstev editoru. V případě využití tohoto parametru je nutné předat i setter přes parametr setLayers. Detailní popis struktury vrstev je v sekci 4.1.2.
- **setLayers** – nepovinný parametr – setter pro pole vrstev. V případě jeho využití je nutné předat i samotný stav skrze parametr layers.
- **defaultLayers** – nepovinný parametr – pole výchozích vrstev. V případě využití tohoto parametru jsou ignorovány parametry layers a setLayers a stav pro vrstvy je vytvořen samotným providerem s touto výchozí hodnotou. Detailní popis struktury vrstev je v sekci 4.1.2.



Obrázek 4.9: Snímek obrazovky demonstrační aplikace se zobrazenou vektorovou mřížku okolo jednotlivých pixelů rastrových vrstev pro detailní rastrové anotace a kurzorem rastrového aktuálně vybraného štětce.

- **downloadedRasterLevelSize** – nepovinný parametr – implicitně 0.25 – číselná hodnota, podle níž je určována aktuální úroveň dlaždicové mřížky. Určuje interval úrovně přiblížení plátna, který platí pro jednu úroveň mřížky.
- **downloadedRasterMinTilesCount** – nepovinný parametr – implicitně 3 – číselná hodnota, která určuje minimální počet dlaždic na kratším z rozměrů aktuálního výřezu plátna zobrazeném v okně editoru.
- **downloadedRasterDrawAtOnce** – nepovinný parametr – implicitně true – pravdivostní hodnota, určující, zdali se mají dlaždice vykreslovat ihned po načtení z datového skladu nebo až naráz jako jeden obrázek po dokončení získávání všech relevantních dlaždic.

Rozhraní:

- **rasterWidth** – číselná hodnota, která udává šířku všech vrstev, podle rozlišení rastrových vrstev. Je nutné ji zadat při inicializaci provideru parametrem.
- **rasterHeight** – číselná hodnota, která udává výšku všech vrstev, podle rozlišení rastrových vrstev. Je nutné ji zadat při inicializaci provideru parametrem.
- **layers** – pole vrstev editoru. Detailní popis struktury vrstev se nachází v sekci 4.1.2.

Obrázek 4.10: Snímek obrazovky demonstrační aplikace s detekovanými bloby červenou barvou s jejich textovým označením barvy a ID.

- **setLayers** – setter pro pole vrstev.
- **selectedLayer** – číselná hodnota udávající aktuální vybranou vrstvu.
- **setSelectedLayer** – setter pro změnu aktuálně vybrané vrstvy.
- **selectedLayerType** – pomocná hodnota udavající typ aktuálně vybrané vrstvy. Hodnotu lze využít pro zobrazení relevantních nástrojů pro daný typ vrstvy.
- **downloadedRasterLevelSize** – číselná hodnota, podle nějž je určována aktuální úroveň dlaždicové mřížky.
- **downloadedRasterDrawAtOnce** – pravdivostní hodnota, určující, zdali se mají dlaždice vykreslovat ihned po načtení z datového skladu nebo až naráz jako jeden obrázek po dokončení získávání všech relevantních dlaždic.
- **createLayer** – funkce pro vytvoření nové vrstvy. Předaná vrstva musí být kompatibilní se strukturou vrstev 4.1.2. Nová vrstva je vždy vytvořena v popředí.
- **setLayerByIndex** – funkce, která přepíše vrstvu na indexu v poli layers předanou vrstvou.
- **setLayerById** – funkce pro přepis vrstvy. Vrstva pro přepsání je nalezena podle ID, které má i předaná vrstva.

Obrázek 4.11: Snímek obrazovky demonstrační aplikace po převodu interní vektorové vrstvy na interní rastrovou vrstvu.

- **removeLayer** – funkce pro smazání vrstvy z pole layers na zadaném indexu.
- **removeLayerById** – funkce pro smazání vrstvy, která je nalezena podle zadaného ID.
- **rasterizeLayer** – funkce pro rasterizaci vytvořené vektorové vrstvy. Vektorová vrstva se musí nacházet na zadaném indexu v poli layers. Původní vrstva je smazána a místo ní vznikne vytvořená rastrová vrstva.
- **historyPush** – funkce, která je interně volána při manipulaci s některou z vrstev. Uchová uvnitř provideru záznam o provedené akci a informace nutné pro vrácení akce.
- **undo** – funkce vyvolávající zrušení poslední akce v historie a posuv indexu aktuálního stavu historie.
- **redo** – funkce vyvolávající zpětné provedení poslední zrušené akce v historie a posuv indexu aktuálního stavu historie.
- **undoAvailable** – pravdivostní hodnota značící, zdali je možné provést zrušení akce.
- **redoAvailable** – pravdivostní hodnota značící, zdali je možné provést zpětné provedení poslední zrušené akce.

Datová struktura vrstev editoru

Datová struktura externí rastrové vrstvy

```
DownloadedRasterLayer = {
  id: any;
  type: "createdVector";
  visible: boolean;
  opacity: number;
  image?: Blob;
  getImage?: (
    x: number,
    y: number,
    width: number,
    height: number,
    viewWidth: number,
    viewHeight: number,
    signal?: AbortSignal,
  ) => Promise<Blob | null>;
  heatmap?: number[][];
  hatching?: {
    blankWidth: number;
    maskWidth: number;
  };
}
```

Datová struktura externí vektorové vrstvy

```
DownloadedVectorLayer = {
  id: any;
  type: "createdVector";
  visible: boolean;
  opacity: number;
  data: string;
}
```

Datová struktura interní rastrové vrstvy

```
CreatedRasterLayer = {
  id: any;
  type: "createdRaster";
  visible: boolean;
  opacity: number;
  data: {
    canvas?: HTMLCanvasElement;
    ctx?: CanvasRenderingContext2D;
    layer?: Konva.Layer;
    initImage?: Blob;
    actualImage?: Blob;
```

```
    actualImageStorageId?: string;
  }
}
```

Datová struktura interní vektorové vrstvy

```
CreatedVectorLayer = {
  id: any;
  type: "createdVector";
  visible: boolean;
  opacity: number;
  data: {
    elements: (
      | CreatedVectorLayerLine
      | CreatedVectorLayerRectangle
      | CreatedVectorLayerCircle
    )[];
    initElements?: (
      | CreatedVectorLayerLine
      | CreatedVectorLayerRectangle
      | CreatedVectorLayerCircle
    )[];
    layer?: Konva.Layer;
  }
}
```

Datové struktury prvků interní vektorové vrstvy

```
CreatedVectorLayerLine = {
  type: ElementType.Line;
  x: number;
  y: number;
  stroke: string;
  strokeWidth: number;
  opacity: number;
  closed?: boolean;
  points: number[];
  lineCap: LineCap;
  lineJoin: LineJoin;
}

CreatedVectorLayerRectangle = {
  type: ElementType.Rectangle;
  x: number;
  y: number;
  stroke: string;
  strokeWidth: number;
  opacity: number;
```

```
  width: number;
  height: number;
}

CreatedVectorLayerCircle = {
  type: ElementType.Circle;
  x: number;
  y: number;
  stroke: string;
  strokeWidth: number;
  opacity: number;
  radius: number;
}
```

ToolProvider

Provider pro nástroje neposkytuje žádnou parametrizaci. Drží informace o aktuálně vybraném nástroji a jeho nastavení. Poskytuje však rozhraní, ze kterého je tyto informace možné vyčíst nebo je změnit.

Rozhraní:

- **selectedTool** – hodnota výčtového typu reprezentující aktuálně zvolený nástroj. Hodnoty výčtového typu nástrojů viz figura 4.1.2.
- **setSelectedTool** – setter pro aktuálně zvolený nástroj.
- **drawColor** – objekt obsahující vlastnosti pro RGBA kanály aktuálně zvolené barvy nástroje.
- **setDrawColor** – setter pro objekt reprezentující vybranou bravu pro nástroj.
- **toolSize** – číselná hodnota udávající velikost nástroje.
- **setToolSize** – setter číselné hodnoty velikosti nástroje.
- **brushShape** – hodnota výčtového typu reprezentující tvar štětce pro kreslení do vytvořených rastrových vrstev. Výčtový typ obsahuje hodnoty pro kruh a čtverec.
- **setBrushShape** – setter aktuálně zvoleného tvaru štětce pro kreslení do vytvořených rastrových vrstev.

ImageCacheProvider

Provider pro získávání obrazových dat získává předanou funkcí data z datového skladu pomocí funkce getTile, kterou poskytuje přes své rozhraní.

Funkce getTile vytvoří klíč pro mezipamět ve formátu:

```
cacheId + "," + x + "," + y + "," + width + "," + height +
"," + viewWidth + "," + viewHeight
```

```
Tool {
  // General
  Move = "move",
  ZoomIn = "zoomIn",
  ZoomOut = "zoomOut",
  // Vector
  Drag = "drag",
  Remove = "remove",
  Rectangle = "rectangle",
  Circle = "circle",
  Line = "line",
  Polygon = "polygon",
  // Raster
  Brush = "brush",
  Eraser = "eraser",
  Floodfill = "floodfill",
  Clear = "clear",
}
```

Obrázek 4.12: Výčet nástrojů pro manipulaci s vrstvami

tímto klíčem se dotazuje prohlížečového úložiště IndexedDB, pomocí abstrakce knihovny `localForage`. Pokud uspěje, rovnou vrací lokálně získanou dlaždici. v případě neúspěchu volá předanou funkci getImage, které předává své parametry pro výřez: x, y, width, height, viewWidth, viewHeight a signal. Po získání dat, pokud nebyl požadavek přerušen zapisuje tato data do mezipaměti s vytvořeným klíčem a data vrací.

Parametry:

- **getImage** – povinný parametr – funkce pro získání obrazových dat z datového skladu.
- **x** – povinný parametr – číselná hodnota udávající souřadnici na ose x počátku výřezu zdrojového obrázku.
- **y** – povinný parametr – číselná hodnota udávající souřadnici na ose y počátku výřezu zdrojového obrázku.
- **width** – povinný parametr – číselná hodnota udávající šířku výřezu směrem doprava od počáteční souřadnice x.
- **height** – povinný parametr – číselná hodnota udávající výšku výřezu směrem dolů od počáteční souřadnice y.
- **viewWidth** – povinný parametr – šířka aktuálně zobrazeného výřezu dlaždice v okně editoru. Udává škálování zdrojového obrázku.
- **viewHeight** – povinný parametr – výška aktuálně zobrazeného výřezu dlaždice v okně editoru. Udává škálování zdrojového obrázku.
- **signal** – nepovinný parametr – instance objektu AbortSignal, jenž umožňuje předčasně ukončit načítání dat, například po změně posunu nebo přiblížení plátna.

- **cacheId** – nepovinný parametr – číselná hodnota nebo textový řetězec, udávající ID vrstvy pro využití na ukládání do slovníkové mezipaměti.

4.2 Implementace a sestavení demonstrační webové aplikace

Aplikace je implementována společně s knihovnou editoru v jednom balíčku. Aplikace je sestavena pomocí balíčku `parcel`, který je ve vývojářských závislostech společně s jeho rozšířeními `@parcel/packager-ts` a `@parcel/transformer-typescript-types` pro propagování typů v rámci jazyka TypeScript i po sestavení. Tato aplikace slouží primárně pro testování a demonstraci funkcionality knihovny.

Použité externí knihovny:

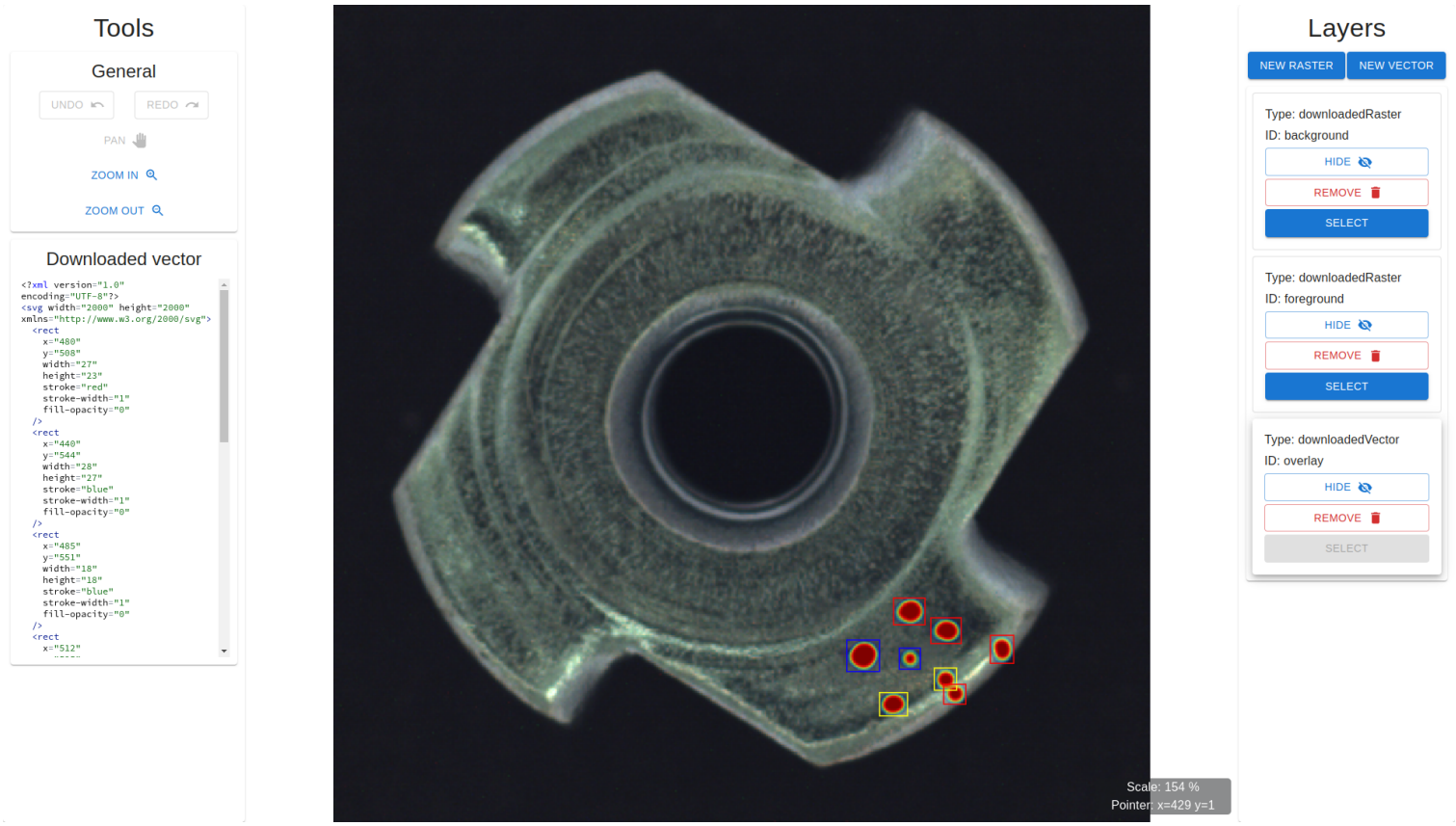
- **react** – 18.2.0
- **react-dom** – 18.2.0
- **@fontsource/roboto** – 4.5.8 – knihovna poskytující písmo Roboto, které je základním písmem v Material UI designu.
- **@mui/material** – 5.11.16 – knihovna obsahující sadu komponent pro vizualizaci vstupů a výstupů v aplikaci. Komponenty dodržují konvence Material UI designu.
- **@mui/icons-material** – 5.11.16 – knihovna obsahující sadu ikon Material UI designu. Je použita pro rozšíření vizuálních prvků aplikace.
- **@emotion/styled** – 11.10.6 – knihovna rozšiřující možnosti stylování nad rámec CSS. Je požadavkem knihovny `@mui/material`.
- **@emotion/react** – 11.10.6 – knihovna rozšiřující možnosti stylování v rámci JSX elementů používaných v React aplikacích. Je požadavkem knihovny `@mui/material`.
- **axios** – 1.3.4 – knihovna pro práci s HTTP komunikací. Oproti fetch API disponuje pokročilejší funkcionalitou. Použita pro komunikaci s datovým skladem.
- **react-ace** – 10.1.0 – knihovna exportující komponentu textového editoru se zvýrazňováním syntaxe a našeptáváním pro zdrojové kódy. Aplikace ji využívá k editaci externích vektorových vrstev ve formátu SVG.
- **react-beautiful-dnd** – 13.1.1 – knihovna poskytující komponenty k vytvoření seznamu s možností změny pořadí pomocí tažení jednotlivých prvků. Použita pro implementaci hierarchie vrstev.
- **react-color** – 2.19.3 – knihovna obsahující komponenty pro výběr barvy a průhlednosti. Využita pro parametrizaci nástrojů
- **uuid** – 9.0.0 – knihovna poskytující funkci pro generování unikátního ID pro nově vytvořené vrstvy.

Demonstrační aplikace se skládá z komponenty obsahující nástrojovou lištu pro výběr a manipulaci s nástroji editoru, komponenty s lištou pro manipulaci s vrstvami a importované komponenty z knihovny editoru. Součástí jsou i některé varianty tepelných map ve formátu požadovaném knihovnou editoru.

Aplikace nejprve z datového skladu načte rozlišení rastrových vrstev, které předává provideru LayersProvider z knihovny editoru. Dále jí předává struktury dvou externích rastrových vrstev, z nichž jedna je v jednokanálové reprezentaci stupňů šedi s parametrem pro přebarvení tepelnou mapou jet. Struktury obsahují funkce podle rozhraní editoru knihovny s využitím funkcí `axios` knihovny pro provádění HTTP požadavků na demonstrační datový sklad. Demonstrační externí vektorovou vrstvu získává aplikace z datového skladu sama a předává ji jako textový řetězec. URL datového skladu je možné definovat v souboru *frontendConfig.ts*. Aplikace také využívá zpětných k získání pozice kurzoru a aktuální úrovně přiblížení a získané hodnoty vypisuje v překryvu nad komponentou editoru.

Nástrojová komponenta demonstrační aplikace

Nástrojová lišta se skládá z obecných nástrojů pro manipulaci s editorem, nástrojů pro manipulaci s interními vektorovými vrstvami, nástrojů pro manipulaci s interními rastrovými vrstvami a oknem editoru kódu knihovny `react-ace` pro editaci externí vektorové vrstvy ve formátu SVG viz snímek obrazovky 4.13. Obecné nástroje jsou zobrazeny vždy a nástroje pro konkrétní typ vrstvy se zobrazují dynamicky podle aktuálně vybrané vrstvy, která je získávána z provideru LayersProvider.



Obrázek 4.13: Snímek obrazovky demonstrační aplikace s aktivní externí vektorovou vrstvou a jejím editorem zobrazeným v nástrojové liště vlevo.

Obecné nástroje zahrnují funkce undo a redo pro pohyb v rámci historie úprav vrstev a jsou získávány z provideru ToolProvideru, stejně jako příznaky informující o jejich do-

stupnosti. Pro manipulaci s posunem plátna tažením slouží nástroj pan. Je možné využít i nástroje pro přiblížení a oddálení na hlavní akci kurzoru pro dotykové zařízení bez podpory snímání více dotyků zároveň.

Nástroje pro editaci interních rastrových vrstev obsahují komponentu pro výběr barvy a průhlednosti z knihovny `react-color` a posuvník pro změnu velikosti nástroje. Samotný výběr nástrojů se skládá ze všech rastrových nástrojů z knihovny editoru i s kombinacemi jejich tvarů. Aktuálně vybraný nástroj je získáván z provideru ToolProvider, jako i funkce pro jeho změnu.

Komponenta pro manipulaci s vrstvami demonstrační aplikace

Lišta pro manipulaci s vrstvami nabízí akce pro vytvoření nové interní vektorové nebo rastrové vrstvy s náhodným ID. Dále zobrazuje údaje o vrstvách aktuálně vykreslených do komponenty editoru. Prvky obsahující údaje a tlačítka s relevantními funkcemi pro daný typ vrstvy jsou součástí vertikálního seznamu implementovaného pomocí knihovny `react-beautiful-dnd`, díky čemuž je možné měnit pořadí těchto vrstev pomocí tažení. Změna pořadí vyvolá v editoru překreslení vrstev v pořadí jako mají v tomto seznamu.

Každý prvek reprezentující vrstvu zobrazuje její typ a ID. Společnými akcemi všech vrstev je jejich skrytí případně opětovné zobrazení, smazání a nastavení jako aktuální vrstvy určené k editaci. Vytvořené vektorové vrstvy mají jako jediné navíc možnost je převést na vytvořenou rastrovou vrstvu, kdy tato akce je vratná pouze pomocí akce undo.

4.3 Implementace a sestavení demonstračního datového skladu

Použité externí knihovny:

- **express** – 4.18.2 – webový framework využitý s rozšiřujícími knihovnami pro zpracovávání HTTP požadavků.
- **body-parser** – 1.20.1 – middleware využitý v instanci frameworku Express.js pro parsování JSON formátu v těle požadavků.
- **cors** – 2.8.5 – middleware využitý v instanci frameworku Express.js, který umožňuje nastavení CORS.
- **sharp** – 0.31.1 – knihovna, kterou jsou načítána a zpracovávána obrazová data dle zadaných parametrů.

Demonstrační datový sklad je minimalistickou verzí možné implementace. Asynchronně načítá obrazová data uložená v souborovém systému, provádí nad nimi transformace a tato data odesílá klientovi. Poskytuje také informace o rozlišení rastrových dat.

Aplikační rozhraní:

- **/svg** – HTTP GET – tento endpoint vrací textový řetězec obsahující demonstrační vektorový obrázek ve formátu SVG.
- **/image** – HTTP GET – tento endpoint vrací objekt s vlastnostmi width a height, reprezentující rozlišení rastrových dat.
- **/image1** – HTTP POST – parametrizovatelný endpoint pro získání první demonstrační rastrové vrstvy. Popis parametrizace koncového bodu viz sekce 4.3.

- **/image2** – HTTP POST – parametrizovatelný endpoint pro získání druhé demonstrační rastrové vrstvy. Popis parametrizace koncového bodu viz sekce 4.3.

Zpracování rastrových obrazových dat na datovém skladu

Požadavek na rastrová data je parametrizovatelný objektem, který může mít vnořené objekty extract a resize. Objekt extract vybírá bod v obrázku podle zadaných souřadnic x a y a vytvoří výřez o šířce width směrem doprava a o výšce height směrem dolů. Pokud je zadán i objekt resize, tak je následně získaný výřez škálován na zadané rozlišení se šířkou viewWidth a výškou viewHeight.

Typová anotace parametrizačního objektu:

```
ImageRequest = {
    extract?: {
      x: number;
      y: number;
      width: number;
      height: number;
    };
    resize?: {
      viewWidth: number;
      viewHeight: number;
    };
  }
```

4.4 Testování chyb a použitelnosti výsledného systému

Výsledný systém byl testován průběžně s přibývající funkcionalitou. Testování vzhledem k povaze systému probíhalo manuálně, protože vytvořit automatizované testy pro grafický editor je implementačně velmi náročné. Pro usnadnění testování bylo využito nástroje React Developer Tools a vývojářské konzole prohlížeče Chromium. Systém byl po dokončení několikrát otestován jako celek vlastním manuálním testováním a testováním vybraných uživatelů.

4.4.1 Testování na úniky paměti

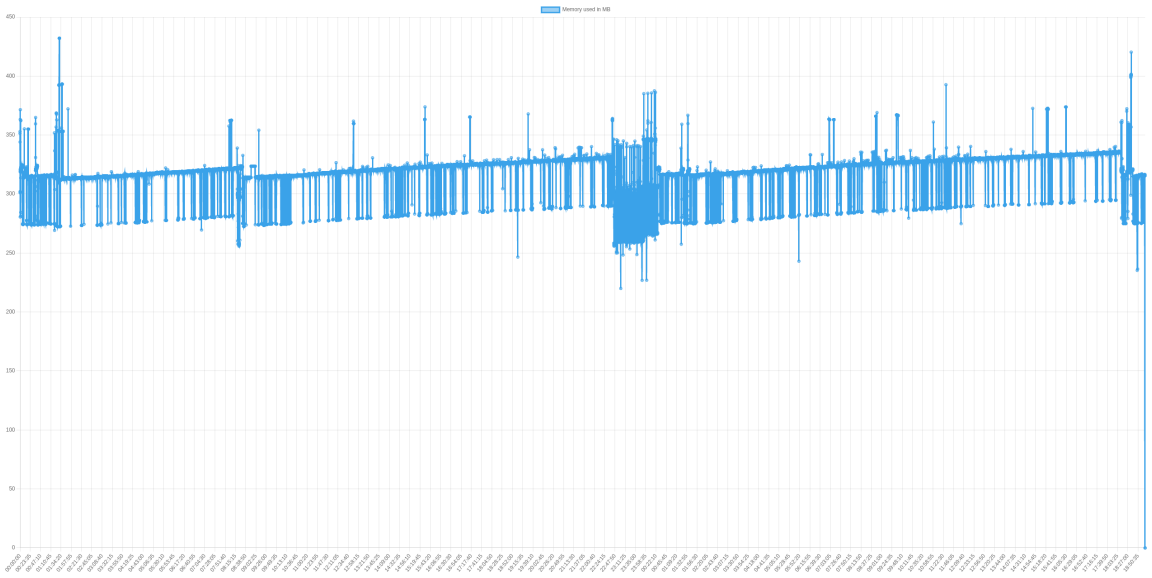
Protože je knihovna určena do produkčního prostředí s předpokládaným dlouhodobým během je nutné editor řádně otestovat na úniky paměti. Pro účely tohoto testování obsahuje testovací aplikace editoru parametr memoryLeakTest s pravdivostní hodnotou v úložišti prohlížeče `localStorage`. Po nastavení tohoto parametru na hodnotu „true“ bude testovací aplikace střídat mezi dvěma externími rastrovými vrstvami, kterým bude přiřazovat vždy unikátní ID, aby byl simulován reálný provoz, kdy přichází nové unikátní vrstvy.

Pro samotné měření byl implementován shellový skript *measureMemory.sh*, který bere jako jediný argument ID procesu, který má být měřen. Tento skript vytvoří virtuální prostředí pro Python a nainstaluje do něj balíček pro měření využité paměti procesem ps_mem, jehož dokumentace je umístěna v repozitáři knihovny[3]. Při spuštění tohoto balíčku mu je předáno ID procesu zadaného při spuštění shellového skriptu, zadán interval pro měření 5

[3] `https://github.com/pixelb/ps_mem`

vteřin a nastaven kompaktní výstup, který je přesměrován do souboru *./memoryFrontend/measurements.txt*.

Data získaná z nástroje `ps_mem` jsou dále zpracovávána vlastní JavaScriptovou aplikací využívající grafovou knihovnu Chart.js, jejíž dokumentace je umístěna na webové prezentaci knihovny[4]. Vizualizační aplikace načítá v intervalu soubor *measurements.txt* a získané hodnoty vykresluje do grafu v MB. Běh o délce 42 hodin 50 minut a 35 sekund je vizualizován na grafu 4.14. Z grafu je patrné, že paměť se dočasně zvyšuje, ale v určitých intervalech je vždy garbage collectorem vyčištěna.



Obrázek 4.14: Graf měření spotřebované paměti procesem testovací aplikace při střídání externích rastrových vrstev s unikátním ID v intervalu jedné vteřiny.

4.4.2 Uživatelské testování funkcionality demonstrační aplikace

Uživatelské testování je důležitou součástí vývoje softwaru, protože umožňuje ověřit, zda je produkt použitelný, efektivní a splňuje potřeby uživatelů. Tato sekce se zaměřuje na popis uživatelského testování funkcionality demonstračního systému vytvořené jako součást diplomové práce.

Testování systému bylo při dosažení milníků stavu spustitelného prototypu provedeno ve třech iteracích s pomocí kvantitativního a kvalitativního přístupu. Testovací skupina se skládala z osmi účastníků, kteří byli zvoleni z akademického prostředí a z odborníků na manuální testování webových aplikací. Jedním z účastníku byl i odborník na oblast strojového učení se specializací na počítačové vidění. Testování probíhalo vzdáleně přes virtuální privátní síť, což umožnilo aplikaci testovat ve vlastním prostředí účastníků. Zároveň účastníci sdíleli svou obrazovku, díky čemuž bylo možné sledovat jejich práci s aplikací. Účastníkům byl poskytnut úvodní materiál a pokyny k testování. Účastníci byli požádáni, aby prováděli anotační úkony jak ve vektorovém, tak v rastrovém režimu. Při procesu anotování popisovali své myšlenkové pochody, které byly vyhodnocovány a případně zapisovány. Odborníkem na neuronové sítě byla navíc testována i zobrazovací část systému, kdy manipuloval s de-

[4] `https://www.chartjs.org/`

monstračním datovým skladem, kde vytvářel různé kombinace zobrazovaných vrstev, jako byly například masky zón, podkladové fotky průmyslového produktu, rastrový výstup inference neuronové sítě s nastaveným přebarvením a vektorových vrstev, ohraničující defekty nalezené inferencí neuronové sítě.

Na základě výsledků testování byly v každé iteraci testování veškeré nalezené chyby systému průběžně opravovány a byly evidovány požadavky na rozšiřující funkcionalitu, které byly buď zapracovány do další iterace nebo zařazeny do sekce s návrhy pro další vývoj 5.2. Mezi zapracovanými požadavky byly například:

- funkce pro převod interní vektorové vrstvy na interní rastrovou verzi,
- pokročilejší funkcionalita vytváření lomených čar,
- podpora dotykových zařízení,
- pinch akce pro změnu úrovně přiblížení na vícedotykových zařízeních,
- zobrazování polohy kurzoru a úrovně přiblížení,
- úprava rychlosti změny přiblížení.

Finální iterace testování ukázala, že aplikace splňuje potřeby uživatelů. Účastníci neměli problémy s ovládáním aplikace a dokázali úspěšně provádět zadané úkoly. Účastníci ohodnotili aplikaci jako snadno použitelnou a intuitivní. Byli spokojeni s výkonem aplikace a rychlostí odezvy. Celkově lze říci, že uživatelské testování bylo úspěšné a poskytlo užitečnou zpětnou vazbu pro vylepšení aplikace.

Kapitola 5

Závěr

5.1 Shrnutí a závěrečné hodnocení návrhu a implementace

Anotační nástroje pro 2D obrazová data jsou důležité pro řadu úloh, zejména v aplikacích počítačového vidění a zpracování obrazu. Mnoho aplikací počítačového vidění, jako je detekce objektů nebo segmentace obrazu, se spoléhá na algoritmy strojového učení, které je třeba trénovat pomocí anotovaných dat.

Nástroje pro 2D anotaci umožňují lidem anotovat obrázky pomocí ohraničujících rámečků, segmentačních masek, klíčových bodů nebo jiných anotací a poskytnout tak potřebná označená data pro trénování těchto algoritmů. 2D anotační nástroje se používají také ke kontrole přesnosti strojově generovaných anotací. Ručním ověřováním a opravováním anotací mohou uživatelé zajistit, že algoritmy generují přesné výsledky. Výzkumní pracovníci používají 2D anotační nástroje k anotování snímků pro své výzkumné projekty. To může pomoci při pochopení různých problémů souvisejících s obrázky a při vývoji inovativních řešení pro jejich řešení. 2D anotační nástroje lze také integrovat s modely strojového učení a automatizovat tak proces anotace. To pomáhá snížit čas a náklady spojené s ručním anotováním a umožňuje výzkumným pracovníkům zpracovávat velké množství dat. Celkově zastávají 2D anotační nástroje důležitou roli v počítačovém vidění a zpracování obrazu, neboť usnadňují vytváření anotovaných dat pro algoritmy strojového učení, kontrolu kvality, výzkum a vývoj a automatizaci.

Cílem práce bylo analyzovat trh s dostupnými 2D editory grafiky pro webové aplikace a webové anotační nástroje. Analyzovat potřeby uživatelů těchto řešení, implementovat knihovnu 2D editoru grafiky společně s demonstračním prostředím a výsledný systém řádně otestovat.

Při průzkumu trhu bylo zjištěno, že není dostupný žádný nástroj umožňující kombinaci vektorové a rastrové grafiky. U rastrových nástrojů je také rozšířený problém nativního API prohlížeče pro kreslení, a to nemožnost kreslit bez antialiasingu. Dostupné nástroje jsou také často placené nebo nemají zveřejněné zdrojové kódy.

Potřeby uživatelů byly částečně přejímány z existujících nástrojů a byly také po celou dobu práce konzultovány s externí firmou zabývající se počítačovým viděním v oblasti průmyslu. Vývoj díky tomu proběhl agilní metodou, díky čemuž se podařilo pokrýt veškerou mandatorní funkcionalitu. Negativním efektem tohoto přístupu však byla nutnost kód po dokončení finálního prototypu celý refaktorovat.

Výsledná implementace knihovny editoru nyní běží v produkčním prostředí v rámci vlastní aplikace již zmíněné externí firmy a je plánováno její uvolnění do veřejných repozitářů, což může rozšířit její funkcionalitu, případně vylepšit stávající rozhraní.

5.2 Návrhy na další vývoj

V průběhu analýzy požadavků, i při návrhu, implementaci a testování byly zaznamenávány možné funkcionality knihovny, které však kvůli své nízké důležitosti nebyly implementovány. Vzhledem k tomu, že je knihovna produkčně využívána budou pravděpodobně tyto návrhy přibývat a k jejich implementaci by mohlo pomoci zveřejnění knihovna do veřejných repozitářů.

Aktuální výčet plánovaných funkcionalit editoru:

- Parametrizovatelný nástroj „magic wand“ pro automatický výběr oblasti s podobnou barvou. Jeho cílem je zrychlení vytváření anotací.
- Podpora zobrazování videa. Účelem je zobrazovat živý proud obrazových dat z kamer a možnost přes něj zobrazovat masky a další informace.
- Mód knihovny pro statické zobrazení zadaného výřezu, jehož účelem by bylo nastavit si pevně zadaný výřez vrstev, kdy by nebylo třeba při zobrazování výstupů z neuronových sítí nebo kamer načítat zbytečná data z nezájmových oblastí.
- Množinové operace s bloby. Pro větší flexibilitu editoru pro účely anotací by se mohly mezi bloby různých barev provádět množinové operace.
- Editační mód interní vektorové vrstvy i pro čtverce, kružnice a jednoduché čáry.
- Rozšíření parametrizace existujících prvků editoru, jako například koeficient pro rychlost změny přiblížení, barva a průhlednost výplně vytvořených vektorových prvků, přizpůsobení designu nástroje pro vytváření a editaci lomených čar a možnost změnit popisky blobů.

Literatura

[1] Bergmann, P., Batzner, K., Fauser, M., Sattlegger, D. a Steger, C. The MVTec Anomaly Detection Dataset: A Comprehensive Real-World Dataset for Unsupervised Anomaly Detection. *International Journal of Computer Vision*. 2021.

[2] Bergmann, P., Fauser, M., Sattlegger, D. a Steger, C. MVTec AD – A Comprehensive Real-World Dataset for Unsupervised Anomaly Detection. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019.

[3] Lei, T. a Nandi, A. K. *Image Segmentation: Principles, Techniques, and Applications*. 1. vyd. New Jersey: John Wiley, 2023. ISBN 9781119859000.

[4] Myers, M. *A smarter way to learn JavaScript*. 1. vyd. North Charleston: Createspace Independent Publishing Platform, 2014. ISBN 1497408180.

[5] Sung, K., Pavleas, J., Munson, M. a Jason Pace, J. *Build your own 2D game engine and create great web games*. 2. vyd. Berlin: APress, prosinec 2021. ISBN 1484209532.

[6] Sureshkumar, G. *Image Annotation and Applications*. 1. vyd. Chisinau: LAP LAMBERT Academic Publishing, 2016. ISBN 9781119859000.

[7] Uzayr, S. B., Cloud, N. a Ambler, T. *JavaScript frameworks for modern web development*. 2. vyd. Berlin: APress, 2019. ISBN 1484249941.

[8] Vanderkam, D. *Effective TypeScript*. 1. vyd. Sebastopol: O'Reilly Media, 2019. ISBN 1492053740.

[9] Wang, G., Han, S., Ding, E. a Huang, D. Student-Teacher Feature Pyramid Matching for Anomaly Detection. In: The British Machine Vision Conference. 2021.