FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Systém záznamu pohybu očí pro účely studií použitelnosti
Student:	Bc. Michal Mroček
Vedoucí:	Ing. Josef Pavlíček, Ph.D.
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce zimního semestru 2021/22

Pokyny pro vypracování

Cílem práce je vytvořit webovou aplikaci umožňující záznam pohybu očí po obrazovce. Tento systém bude veřejně dostupný a instalovaný na serveru portal.uspin.cz. Postupujte dle následující metodiky:
- Nastudujte současnou technologii možností záznamu pohybu očí s EDGE camer umístěných na monitorech přenosných pocítačů,
- navrhněte vhodnou technoligii pro vývoj takové aplikace,
- aplikaci naprogramujte,
- aplikaci otestujte,
- uveďte do použitelného provozu na zmíněném serveru,
- definujte závěry.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 5. září 2020

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

Systém záznamu pohybu očí pro účely studií použitelnosti

Bc. Michal Mroček

Katedra softwarového inženýrství

Vedoucí práce: Ing. Josef Pavlíček, Ph.D.

5. ledna 2021

Poděkování

Děkuji vedoucímu práce Ing. Josefu Pavlíčkovi, Ph.D. za pomoc při jejím vytváření. Dále všem testerům, kteří mi věnovali svůj čas, a pomohli výslednou aplikaci odladit. Děkuji i své rodině a přátelům za podporu během studia.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 5. ledna 2021 .....

České vysoké učení technické v Praze

Fakulta informačních technologií

Odkaz na tuto práci

Mroček, Michal. *Systém záznamu pohybu očí pro účely studií použitelnosti.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2021.

Abstrakt

Tato práce obsahuje popis softwarového procesu, jehož výsledkem je webová aplikace. Ta zaznamenává data o pohybu očí po obrazovce a v reálném čase z nich vytváří teplotní mapy. Data umí exportovat do obrázku nebo strojově čitelné podoby. S pomocí zabezpečeného administračního rozhraní umožňuje vytvářet sady dat, které lze distribuovat mezi další uživatele. Součástí práce je analýza stávajících řešení, návrh systému, popis implementace, vyhodnocení testování a návrhy na budoucí vylepšení.

Klíčová slova detekce pohledu, sledování pohybu očí, teplotní mapa, JavaScript, Angular, Docker

Abstract

This master thesis contains a description of the software process, which results in a web application. Application tracks eye movements and estimates where the user gazes whilst generating heatmaps. Data can be exported to an image or machine-readable form. With help of secured admin section allows to create data bundles which can be distributed to users. This thesis also includes analysis of current solutions, design of a new system, its implementation description, evaluation of testing results and proposals for further development.

Keywords eye gaze, eye tracking, heatmap, JavaScript, Angular, Docker

Obsah

Seznam obrázků

Seznam ukázek kódu

Seznam tabulek

Úvod

Počítačové vidění je jedním z vědních oborů zabývajícím se zpracováním digitálního obrazu. Algoritmy postupně extrahují data z reálného světa a převádí je do strojové podoby s cílem nalézt regiony obsahující specifické vlastnosti [1]. Díky počítačovému vidění lze jednoduše automatizovat úkoly, které by jinak musel vykonávat člověk.

V této práci se počítačové vidění uplatňuje při hledání lidské tváře, očí, zornice a jiných částí obličeje. Pomocí dalších vstupních dat – jako je například pohyb kurzoru – systém vytváří odhad, kam se uživatel přibližně dívá.

S použitím dat o pohybu dále pracuje a vytváří teplotní mapy, které zvýrazňují uživatelsky nejpoutavější regiony. Tato technika se využívá v oblasti uživatelského testování – testerovi je představena množina obrázků, videí, web nebo jiná aplikace a zjišťuje se, jak s nimi interaguje [2]. Pozorováním chování uživatelů dochází k odhalení chyb, nedodělků nebo i neintuitivních elementů v systému.

Cílem této práce je vytvořit aplikaci, která bude umět zaznamenávat data o pohybu očí a generovat teplotní mapy. Do budoucna by měla být připravena i k provádění uživatelských testování jiných systémů.

Kapitola 1

Analýza

Prvním krok procesu vývoje spočívá v analýze. V této kapitole se budu věnovat základní teorii stojící za úspěšnou detekcí pohledu – konkrétně rozpadu algoritmu na jednotlivé části a jejich základním popisem. V dalším kroku zanalyzuji stávající řešení. Na základě dostupnosti hardwaru a softwaru vyberu technologii pro vývoj aplikace. Dále vydefinuji její rozsah pomocí funkčních požadavků, nefunkčních požadavků a případů užití. S ohledem na požadovanou funkcionalitu představím stavový diagram reprezentující životní cyklus aplikace a nakonec navrhnu drátěný model.

1.1 Teorie detekce pohledu

Princip rozpoznání lze rozdělit do několika fází:

- detekce očí,
- nalezení zornice,
- konstrukce 3D vektoru reprezentující směr pohledu,
- projekce vektoru a namapování na cílovou plochu.

1.1.1 Detekce očí

Samotná detekce očí by se dala kategorizovat jako podúlohou rozpoznání, jestli se na vstupním obraze vůbec nachází obličej. Při detekci se spoléháme na základní rysy lidského obličeje. Například že obočí je na černobílé fotografii tmavší než oblast očí, nos je o něco světlejší než líce, stejně tak rty jsou tmavější než okolní oblast úst.

(a) (b) (c) (d)

Obrázek 1.1: Příklady Haarových vzorců

1.1.1.1 Haar vlastnosti

S využitím těchto předpokladů můžeme na obraz umístit obdélník o předem definovaných rozměrech. Rozdělit jej na dvě, tři nebo čtyři stejné regiony. V každém z nich určit, kolik tmavých nebo světlých pixelů obsahuje a konečně zamítnout nebo potvrdit hypotézu, jestli se v daném regionu může nacházet daná část lidského obličeje.

S touto myšlenkou přišel v roce 1909 Alfrédem Haar, dnes ji v anglické literatuře lze najít pod pojmem „Haar features“. V ideálním případě máme na vstupu černobílý obraz, pixel je buď černý nebo bílý. V případě stupně šedi nelze přímo rozpoznat černé a bílé pixely. Pro pixel ve 2D prostoru na pozici i, j počítáme tedy s normalizovanou hodnotou $n_{i,j}$ v intervalu $\langle 0, 1\rangle$. Nula přísluší bílé barvě, jednička černé. S cílem rozdělit pixely do dvou množin je nutné určit hraniční hodnotu, od které bude pixel pokládán za černý. Pro obdélník typu (a) na obrázku 1.1 o rozměrech N, M a hraniční hodnotu t lze výslednou hodnotu r vypočítat:

$$r = \frac{2}{N * M} \sum_{i=1}^{N} \sum_{j=1}^{M/2} n_{i,j} - \frac{2}{N * M} \sum_{i=1}^{N} \sum_{j=M/2+1}^{M} n_{i,j}$$

V ideálním případě se počet černých pixelů v horní polovině nasčítá přesně na půlku obsahu obdélníku, tedy na $\frac{N*M}{2}$. Ve spodní půlce budou jen bílé, takže součet bude 0. Detekce je 100% úspěšná, pokud je výsledná hodnota r rovna 1.

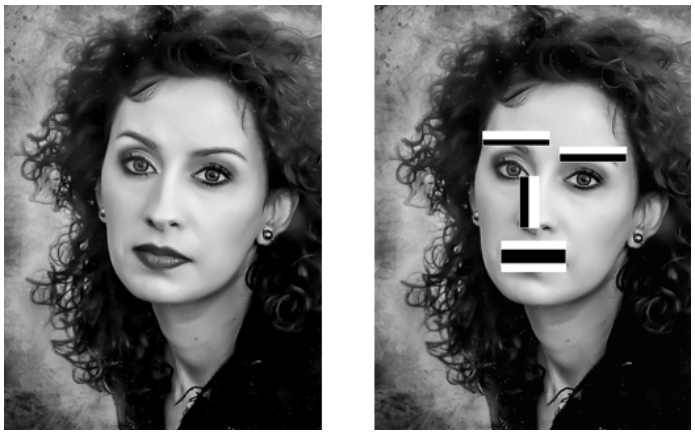
S ohledem na povahu obrázků je na místě určit si další hodnotu, opět v intervalu $\langle 0, 1\rangle$. Pokud výsledné r bude vyšší, budeme pokládat vyhledávání za úspěšné.

```
for all (x, y) such that 1 ≤ x ≤ 24 and 1 ≤ y ≤ 24 do
    for all (w, h) such that x+h-1 ≤ 24 and y+2w-1 ≤ 24 do
        S1 = sum of white pixels in [x,x+h-1][y,y+w-1]
        S2 = sum of black pixels in [x,x+h-1][y+w,y+2w-1]
        r(x, y, w, h)= S1-S2
    end for
end for
```

Kód 1.1: Ukázka počítání hodnot r pro obrázek o rozměrech 24x24 [4]

1.1.1.2 Viola-Jones algoritmus

Viola-Jones algoritmus funguje na principu vyhledávaní Haar oblastí na vstupním obrazu. Pokud se nacházejí na správných místech, lze předpokládat, že se v daném rámci nachází lidský obličej. Příklad možných oblastí lze pozorovat na obrázku 1.2.



Obrázek 1.2: Ukázka možných Haar oblastí na fotografii obličeje (vpravo) [3]

Na vstupu o rozměru 24x24 pixelů a vyhledávání vzoru 1.1 (a) lze vytvořit až 43 200 různých obdélníků. Předpokládejme obdélník o rozměrech (w, h) s pravým horním rohem na souřadnicích (i, j). Všechny možné iterace lze vyjádřit následujícím pseudokódem 1.1.

Vezeme-li v potaz další možné tři vzory, dostáváme se na celkový počet 162 336 možností. Ten lze snížit pomocí algoritmu Adabost. Kompletní optimalizací se zabývá Wang Y. [4].

1.1.2 Nalezení zornice

Po nalezení regionů, kde se vyskytují oči, je dalším úkolem algoritmu identifikace zornice. Ta je zpravidla nejtmavší částí oka. Této skutečnosti využívá program od Antoine i a [5]. Pomocí vytrénovaného modelu nejdříve identifikuje regiony, kde se nachází oči. Obrázky převede do stupně šedi a postupně aplikuje filtry dostupné v OpenCV knihovně s cílem zvětšit a dále identifikovat region, kde se nachází zornice. Jednou z fází je takzvaný thresholding. Ten přebarvuje pixely na bílo, pokud nedosahují určité úrovně jasu, v opačném případě jsou obarveny černou barvou. Špatné nastavení hraniční hodnoty vede ke ztrátě informace o pozici čočky. Algoritmus se tak musí pro každý nový vstup překalibrovat. Autor využil skutečnosti, že čočka zaujímá určitou část obrázku. Empiricky zvolil 48 % plochy. Pro každý vstup testuje různé vstupy pro thresholding s cílem se nejvíce přiblížit této hodnotě. V dalším kroku hledá na obrázku celistvé regiony černých pixelů. Vybere ten s největší plochou, vypočítá jednotlivé momenty a určí geometrický střed. Ten je pokládán ze střed zornice.

Ukázka vstupního souboru a výsledků po aplikaci transformací s různými hodnotami prahové hodnoty t jsou vykresleny na obrázku 1.3.

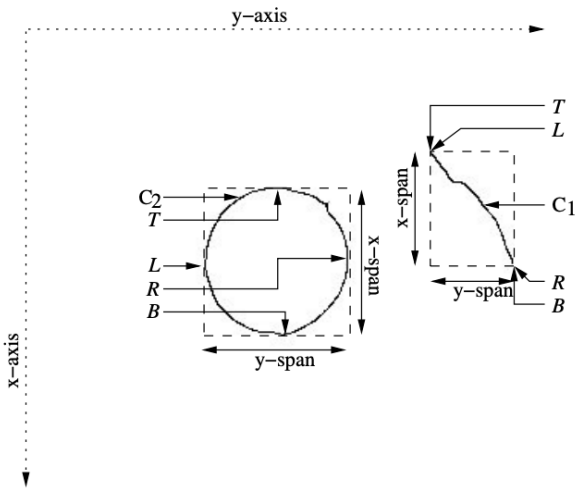
(a) Originál (b) $t = 20$ (c) $t = 30$ (d) $t = 40$ (e) $t = 50$

Obrázek 1.3: Různé nastavení prahové hodnoty

Pokročilejší algoritmy hledající zornici, využívají skutečnosti, že její tvar je elipsovitý. Algoritmus navržený v roce 2007 [6] nejdříve obrázek zmenší, provede sadu transformací pro zvýšení kontrastu a identifikuje jednotlivé shluky pixelů. Ty nazývá komponenty. Pro každou nalezne 4 souřadnice reprezentující extrémy v rámci komponenty – L (levou), T (horní), R (pravou) a B (spodní). Zprůměrováním L,R a T,B komponent odhadne střed zornice. Poloměr dopočte tak, aby se do vzniklého kruhu vešlo co nejvíce pixelů z komponenty. Nakonec je vybrána komponenta, která má v tomto směru nejlepší skóre. Její střed a poloměr je přemapován na vstupní obrázek v původním rozlišení.

1.1.3 Konstrukce 3D vektoru

Jakmile známe pozici zornice, tak dalším úkolem je nalezení dvou 3D vektorů reprezentující směr pohledu každého z ok. Jedna ze základních myšlenek je



Obrázek 1.4: Znázornění namapování jednotlivých L,T,R,B bodů na dvě komponenty

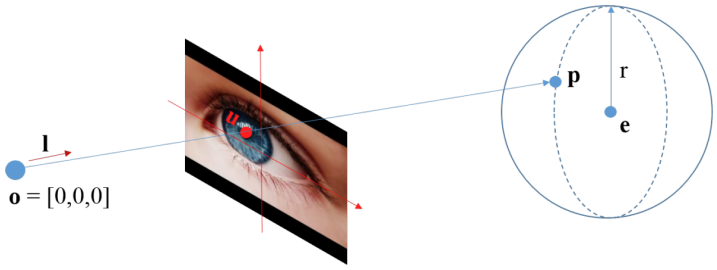
popsána v publikaci „Eye Gaze Tracking Using an RGBD Camera: a Comparison with an RGB Solution“ [7]. Opírá se o fakta, že střed hlavy je vždy vůči očím fixní, stejně tak se nemění rozměry očí. Neznámou v závislosti na hardwaru může být přesná pozice jednotlivých bodů v 3D prostoru. Respektive jaká je vzdálenost od středu oka k místu, kde se nachází zornice. Směr pohledu dále ovlivňují i vertikální a horizontální úhly natočení hlavy.

Jeden z programů zabývající se touto problematikou se nazývá eye-gaze [8]. Je publikován pod MIT licencí a napsán v jazyce C++. Při odhadování vektoru spoléhá na fakta, že kraje očí jsou vůči zornici taktéž neměnné. Střed oka je středem úsečky s krajními body v levé a pravé straně oka. Následný vektor spojující tento střed a souřadnice zornice určuje, kam se člověk dívá.

Obrázek 1.5 zobrazuje lidské oko jako kouli se středem e, poloměrem r a zornicí p na jejím povrchu. Oko je snímáno kamerou na pozici o. Bod u reprezentuje 3D souřadnici čočky. Směr pohledu lze vyčíst z vektoru $\overrightarrow{ep}$.

1.1.4 Projekce na cílovou plochu

Finálním krokem je převod aktuální hodnoty 3D vektoru na 2D souřadnice relativní vůči plátnu cílové plochy, v tomto případě monitoru. Na řadu přichází kalibrace. Ta je potřeba vždy, protože není znám předpis pro převod mezi optickou a vizuální osou oka. Minimálně je potřeba nasnímat 4 body repre-



Obrázek 1.5: Vytvoření 3D vektoru reprezentující směr pohledu [7]

zentující rohy obrazovky. Microsoft a WebGazer.js jich pro kalibraci využívá 9 [7],[9], GazeCloudAPI knihovna 33 [10].

Díky kalibračním datům lze zpřesňovat i odhad směru pohledu. V systému Kinect autoři empiricky zvolili poloměr oční bulvy 12 mm [7] a dále s využitím algoritmu COBYLA a naměřených dat odhad dále zpřesňují. Ke zpřesnění celkového odhadu se využívá i skutečnost, že se uživatel dívá na kurzor, když s ním pohybuje. V rámci vývoje knihovny WebGazer.js autoři porovnali několik metod projekce na cílovou plochu. Jako základní použili model založený na lineární regresi. Porovnávali vzdálenost v pixelech mezi reálným místem, kam se testovací subjekt díval a odhadnutou polohou. V případě modelu lineární regrese dosahovala průměrná chyba 256.9 pixelů se směrodatnou odchylkou 75. O něco lepší výsledek zaznamenali při využití hřebenové regrese (chyba 232.4, směrodatná odchylka 92.3). Jako nejlepší se jevila hřebenová regrese s využitím dat o pohybu kurzoru – chyba 174.9 pixelů, směrodatná odchylka 91.6.

1.2 Stávající řešení pro skenování pohledu

1.2.1 Tobii Pro

Aktuálně jedeno z nejpokročilejších řešení v oblasti sledování pohledu nabízí firma Tobii Pro. Pro využití jejího softwaru je potřeba nakoupit i jejich hardware. Díky SDK se lze napojit přímo na zdrojová data, a vyvinout vlastní aplikaci (například s využitím PyGaze, viz níže).

Samotný hardware je hůře dostupný zejména kvůli své ceně. Společnost neuvádí oficiální cenovky. K dispozici jsou pouze kontaktní formuláře pro zájemce. Na internetu pak lze dohledat, že hardware pro vědecké účely lze pořídit v ce-

nové relaci od 1 000 do 10 000 amerických dolarů za kus [11]. Samotná veřejná dokumentace je značně omezená a nelze přesně zhodnotit, do jaké míry by šel tento produkt použít pro vyvinutí cílového produktu.

1.2.2 GazeCloudAPI

GazeCloudAPI je javascriptová knihovna, která je dostupná na Githubu [10]. Využívá kameru počítače a vyžaduje kalibraci. V prvotní fázi kalibrace je sejmuto 17 bodů. Uživatel nesmí hýbat hlavou, pouze očima. V druhé fázi je uživatel vyzván, aby zafixoval pohled a hýbal pouze hlavou. Dalších 16 bodů je sejmuto, z toho poslední 4 na černém podkladu (všechny předchozí byly na bílém). Na webu gazerecorder.com je dostupná ukázková verze aplikace. Za vstup lze zvolit sadu obrázků z počítače a nastavit dobu, po jakou budou zobrazovány. V rámci beta testu je umožněno načíst i webovou stránku. Po dokončení snímání aplikace vygeneruje jedno video. Exportovat lze pouze videozáznam, na kterém jde vidět, jak uživatel interaguje s webem. Znázorněno je i generování teplotní mapy.

Aplikace funguje spolehlivě. Dokonce i v případě, že uživatel má brýle, v jejichž sklech jsou vidět odlesky okolí.

Nevýhodou řešení je výsledná prezentace dat. Vše je zkombinováno do jednoho videa. Aplikace nerozlišuje teplotní mapu pro různé zdrojové soubory. Výsledná data nejsou strojově čitelná. Součástí bohužel není licence, na webu autora je pouze formulář pro registraci, kde se musí uvést důvod využití knihovny.

Knihovna funguje k dnešnímu dni v 95 procentech prohlížečů [12]. Nejcitelnější problémy mohou v případě jejího nasazení pocítit uživatelé Internet Exploreru. Kvůli nepodpoře Stream API ze strany prohlížeče ji nelze použít.

1.2.3 WebGazer.js

Další javascriptová knihovna WebGazer.js je vyvíjena na americké Brown University. K dispozici je pod GPLv3 licencí. Což v případě jejího využití implikuje, že nově vytvořený zdrojový kód musí být publikován pod stejnou licencí [13]. Knihovna sama zaznamenává pohyb očí a události jako kliky myší. Na základě nich neustále zpřesňuje generované odhady. Pro integraci je potřeba pouze knihovnu spustit a řídit ji v rámci životního cyklu prohlížeče.

2D souřadnice popisující přibližné místo na monitoru s časovou značkou jsou předávány od spuštění a prvního kliku myši.

Stejně jako GazeCloudAPI i tato knihovna závisí na Stream API a lze ji použít v circa 95 procentech prohlížečů.

1.2.4 EyeQuant

Společnost EyeQuant za pomocí svého softwaru generuje teplotní mapy, ale bez účasti lidí. V rámci vývoje svého produktu najali 500 lidí [14], nasnímali jejich reakce na prezentované reklamy a vytrénovali AI. Vytvořili vlastní metodiku, pomocí které jim přiřazují skóre na stupnici od 0 do 100. Ostatní firmy tak pomocí jejich produktu mohou odhadnout, jestli jejich reklamní materiál má potenciál uspět. Zároveň díky teplotní mapě lze i odhadnout, jaká část upoutá největší pozornost. Software nelze jednoduše vyzkoušet, je vyžadováno schválení žádosti autory.

1.2.5 PyGaze

Open source knihovna PyGaze umožňuje přístup k SDK různých výrobců hardwaru pro sledování pohybu očí [15]. Je napsána v jazyce Python, což do jisté míry znemožňuje jednoduchou instalaci a spuštění pro běžné uživatele. Díky snadné rozšiřitelnosti lze napojit vlastní hardware – tedy například kameru notebooku. Kalibrace a konečný odhad, kam se uživatel dívá, je funkcionalita, která knihovnou není podporována a musí být doprogramována.

1.2.6 Ostatní možnosti

Snaha rozpoznat, kam se lidé dívají, není trendem posledních let, a proto existuje velká řada knihoven i univerzitních projektů, které se touto problematikou zabývají. V roce 2008 vznikla pod hlavičkou Cambridgeské univerzity a společnosti Samsung C++ knihovna **OpenGazer** [16]. Poslední dostupná aktualizace je z roku 2010 [17]. Na webu je dále napsáno, že další měla vyjít v roce 2012.

Dalším z univerzitních projektů je Open Gaze And Mouse Analyzer neboli **OGAMA**. Vznikl v roce 2010 a od roku 2016 je neudržovaný. Zdrojový kód je napsaný v jazyce .NET a je publikován pod GPL licencí. Aplikace umožňuje sledovat pohyb myši a dále odhaduje, kam se uživatel dívá na monitor. Stejně jako PyGaze předchozí řešení umožňuje integraci snímačů třetích stran.

1.3 Výběr technologie

Aktuálně se na trhu nachází sada knihoven v různých programovacích jazycích. Veřejně dostupný produkt, který by splňoval základní požadavky kladené na cílovou aplikaci, neexistuje. S pomocí některých knihoven, které jsou dostupné v jazycích C++, Python, JavaScript, jej lze vyvinout s nižším úsilím. S ohledem na univerzálnost C++ lze zvažovat i naprogramování aplikace v dalších jazycích jako:

- Java – pomocí knihovny JavaCPP lze volat C++ kód přímo z Javy,
- Kotlin – zdrojový kód se kompiluje do bytekódu Javy,
- .NET – C++ kód lze volat pomocí C++ Interop.

Limitací může být hardware, zejména jeho cenová dostupnost. V případě využití komerčních řešení se může vyšplhat až do několika tisíců dolarů. Aplikaci lze vyvinout jen s použitím běžných kamer, které jsou součástí notebooků nebo se dají zakoupit od několika stovek korun. Důsledkem této volby bude pravděpodobně ztráta přesnosti u naměřených hodnot. Další nevýhoda spočívá v dynamičnosti hardwaru. Každá kamera produkuje jinak kvalitní výstup o jiném rozlišení. Nižší rozlišení vstupního obrazu bude mít za následek i zhoršení podmínek detekce směru pohledu. Samotné výhody a nevýhody shrnuje tabulka 1.1.

	Běžně dostupné kamery	Speciální hardware
Výhody	Cena Dostupnost Instalace a použití Podpora přímo v OS Zastupitelnost	Přesnost Kvalita vstupu
Nevýhody	Přesnost Variabilní kvalita dle modelu	Cena Dostupnost Instalace a použití Vendor lock

Tabulka 1.1: Srovnání výhod a nevýhod běžně dostupných a speciálních kamer

S ohledem na převažující výhody běžně dostupných kamer se spíše kloním k jejich využití. V dalším kroku je potřeba zvolit programovací jazyk. Pro maximalizaci uživatelského komfortu jsem se rozhodl pro JavaScript. Díky

němu půjde výslednou aplikaci spustit ve webovém prohlížeči bez nutnosti instalace dodatečného software. Ostatní jazyky jako Java, Python nebo .NET vyžadují pro spuštění dodatečný software [18]. V případě C++ by se kód musel kód kompilovat zvlášť pro každý operační systém a architekturu CPU. Webovou JavaScript aplikaci lze navíc pomocí knihovny Electron [19] přetvořit do samostatně funkční aplikace. Podporovány jsou systémy Windows, Linux a macOS.

1.4 Renderování dat

Pro plnou funkčnost aplikace je nutné umět vykreslit obrázky, ostatní webové stránky a PDF soubory. Webová aplikace ale tento úkol v některých případech nedokáže zvládnout bez pomoci backendu.

1.4.1 Vykreslování obrázků

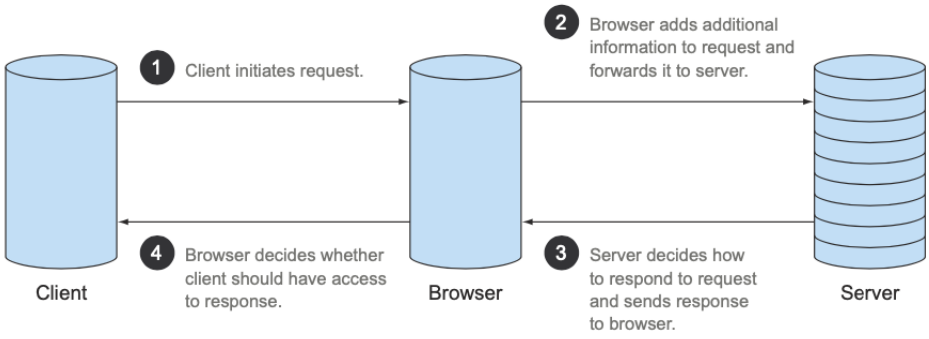
Vykreslování obrázků v prohlížečích je nativní funkcí. Problém může nastat u zdrojů uložených na cizích serverech, respektive doménách. V rámci ochrany serverů existuje mechanismus CORS [20]. Primárně zabraňuje načítání dat, pokud cílový server neudělí webu přístup. Implementován v 98 procentech prohlížečů, proto s ním musím počítat. Pokrývá celou škálu REST metod, v tomto případě se ale stačí zaměřit na `GET` dotazy.

Prohlížeč při požadavku na cizí doménu automaticky připojí hlavičku s klíčem `Origin` a hodnotou obsahující schéma, hosta a doménu aktuální stránky. Server hlavičku načte a ověří, jestli tato doména má mít přístup k požadovaným datům. V odpovědi vrátí hlavičku `Access-Control-Allow-Origin`, možné její hodnoty jsou:

- „*“ v případě, kdy má ke zdroji přístup kdokoliv,
- stejná hodnota, jaká byla v příchozím dotazu v hlavičce `Origin` – tím signalizuje povolení přístupu pro danou doménu,
- a konečně cokoliv jiného – značí odepření přístupu.

Pokud hlavička není vůbec přítomna, tak prohlížeče automaticky pokládají odpověď serveru za zamítavou.

Ve snaze obejít tento CORS mechanizmus je potřeba na serveru implementovat logiku, která bude požadovaná data přeposílat. Server tento problém



Obrázek 1.6: Životní cyklus CORS požadavku [20]

odstraní, protože CORS jsou vynucovány prohlížeči, viz obrázek 1.6.

1.4.2 Vykreslování webu

O něco komplikovanější je vykreslování cizích webů. Pro tento účet vznikl HTML tag `iframe`. Načtení jednoho webu v rámci druhého představuje ale závažný bezpečností risk. Externí web může pomocí JavaScriptu například odposlouchávat, kam uživatel kliká, nebo jaké mačká klávesy. Provádět lze i tzv. XSS útoky [21]. Webové stránky se pomocí hlavičky `X-Frame-Options` můžou bránit a zakázat prohlížeči je vůbec vykreslit mimo jejich doménu [22].

Jediným možným řešením je načtení webové stránky v prohlížeči, vytvoření jejího snímku a vykreslení na cílovém webu. Tohoto cíle lze dosáhnout s javascriptovou knihovnou Puppeteer. K dispozici je i knihovna Playwright, která má širší podporu programovacích jazyků. Lze použít Python, JavaScript, .NET nebo Javu.

Obě zmíněné dokáží simulovat webový prohlížeč – tedy načíst jakoukoliv webovou stránku a HTML výstup vykreslit do souboru.

1.4.3 Technologie serveru

Největší omezení na výběr použité technologie má dostupnost knihoven pro vykreslení webu. Mezi další potřebné schopnosti patří možnost připojit se k databázi a přeposílat data z jiných zdrojů. Tyto požadavky splňuje http.server v jazyce Python, Node.js pro JavaScript nebo například Ktor pro Kotlin. S ohledem na fakt, že webová aplikace bude napsána v JavaScriptu, volím i zde stejný programovací jazyk.

1.5 Funkční požadavky – aplikace

Aplikace obsahuje dvě části – veřejně dostupnou, kam se může dostat kdokoliv, a zabezpečenou. Pro přístup do zabezpečené části je potřeba znát přihlašovací jméno a heslo.

1.5.1 Veřejná sekce

Nahrání vstupu Uživatel má možnost si definovat vstupní obrázek, který bude zobrazen jako pozadí celé aplikace. Nahrát lze soubor z lokálního úložiště, vložit URL adresu obrázku nebo URL adresu webové stránky. Vložit lze taktéž i PDF soubor, který aplikace musí umět vykreslit.

Uživatelem definované sady dat Pohyb očí lze sledovat i na množině obrázků, které jsou součástí jedné sady dat. Každá sada je identifikována jedinečným klíčem, který se musí v aplikaci zadat.

Detekce nepřítomnosti obličeje Na základě dat z kamery aplikace umí rozpoznat přítomnost lidské tváře. V případě absence tuto informaci komunikuje uživateli.

Kalibrace Pro zpřesnění odhadu, kam se uživatel dívá, obsahuje aplikace kalibrační funkcionality. Před každým skenováním nebo při změně rozlišení vykreslovacího plátna musí dojít ke kalibraci.

Zaznamenávání pohybu očí Aplikace dokáže v reálném čase zaznamenávat, kam se uživatel přibližně dívá. Poloha je zaznamenávána jako 2D bod s počátkem v levém horním rohu monitoru.

Pozastavení skenování V průběhu sezení lze skenování kdykoliv přesušit. Data o pohybu očí nebudou nadále zaznamenávána. Na žádost uživatele lze proces ukončit nebo v něm pokračovat.

Generování teplotní mapy Z dat o sledování pohledu aplikace umí vytvořit teplotní mapu. Tu vykreslí přes zdrojový obrázek, na který se uživatel dívá.

Export výstupu Webová aplikace dokáže na žádost uživatele vygenerovat PNG soubor, který obsahuje vstupní obrázek jako podklad. Přes něj je vykreslena teplotní mapa. Data od pohybu lze vyexportovat i do CSV souboru. Součástí jsou souřadnice relativní vůči levému hornímu rohu obrázky a časová značka.

1.5.2 Zabezpečená sekce

Přihlášení administrátora Aplikace obsahuje chráněnou část, do které je potřeba se přihlásit pomocí uživatelského jména a hesla.

Zobrazení sad dat Administrátor vidí všechny sady dat. Má možnost si jednotlivé záznamy procházet po stránkách.

Filtrace dat Zobrazená data lze filtrovat dle klíče, který identifikuje sadu, nebo podle jména sady.

Správa sad dat Administrátor má možnost vytvořit novou sadu dat, upravovat nebo smazat existující.

Odhlášení Administrátor se může kdykoliv ze systému odhlásit.

1.6 Funkční požadavky – server

Renderování webu Server umí na základě URL adresy vyrenderovat webovou stránku do obrázku je formátu JPEG.

Renderování obrázku Server funguje jako proxy a umí načíst obrázky z internetu a dále je přeposlat frontendu.

Renderování PDF Server umí překonvertovat PDF soubor do JPEG souboru.

Správa sad dat Na základě REST standardu server poskytuje rozhraní pro vytváření, čtení, změnu a mazání uživatelsky vytvořených množin obrázků.

1.7 Nefunkční požadavky

Tato sekce popisuje nefunkční požadavky pro aplikaci i server.

Dostupnost Výsledná aplikace i server budou dle zadání této práce veřejně dostupné na síti internet.

Okamžité vykreslování Aplikace musí změny v datech promítat do uživatelského rozhraní okamžitě.

Škálovatelnost Server i aplikace musí být vyvinuty s ohledem na horizontální nebo vertikální škálovatelnost.

Rozšiřitelnost Zdrojový kód obou programů musí být napsán s ohledem na rozšiřitelnost – ideálně musí být pro každou určen jednotný styl, jak integrovat nové funkcionality.

Kompatibilita Aplikace musí být plně funkční v prohlížečích Opera 40, Google Chrome 53, Safari 11, Mozilla Firefox 36, Microsoft Edge 12 a novějších.

Interoperabilita Oba systémy musí být schopné komunikovat mezi sebou.

Bezpečnost Veškerá komunikace po síti internet musí být šifrována a zabezpečena minimálně protokolem TLS 1.2.

1.8 Případy užití

1.8.1 Aktoři

V systému existují dvě role – uživatel a administrátor. Dle nich jsou následné případy užití rozděleny do podsekcí 1.8.2 a 1.8.3.

1.8.2 Veřejná sekce

Tato sekce obsahuje veškeré případy užití, které může vykonávat běžný uživatel systému. Autorizace není vyžadována. Napojení případů užití na funkční požadavky je shrnuto v tabulce 1.2.

Zadání kódu sady

1. Uživatel otevře aplikaci.
2. Uživatel klepne na tlačítko „Zadat kód“.
3. Uživatel vloží kód.

4. Uživatel potvrdí svou volbu.

Nahrání obrázku nebo PDF souboru z lokálního úložiště

1. Uživatel otevře aplikaci.
2. Uživatel klepne na tlačítko „Nahrát soubor“.
3. Uživatel zvolí soubor, který chce nahrát.
4. Uživatel potvrdí svou volbu.

Nahrání obrázku nebo PDF souboru z internetu

1. Uživatel otevře aplikaci.
2. Uživatel klepne na tlačítko „Nahrát soubor“.
3. Uživatel překlikne na záložku „Vložit URL obrázku“.
4. Uživatel vloží URL adresu do textového pole.
5. Uživatel potvrdí svou volbu.

Nahrání webové stránky

1. Uživatel otevře aplikaci.
2. Uživatel klepne na tlačítko „Nahrát webovou stránku“.
3. Uživatel vloží URL adresu do textového pole.
4. Uživatel potvrdí svou volbu.

Skenování pohledu

1. Uživatel povolí přístup ke své kameře.
2. Uživatel se posadí tak, aby jeho obličej byl přibližně ve středu snímaného obrazu.
3. Uživatel provede kalibraci.
4. Uživatel potvrdí začátek skenování.
5. Uživatel se dívá na prezentované obrázky.
6. Uživatel může nepovinně pozastavit skenování a případě jej ukončit nebo v něm dále pokračovat.
7. Uživatel ukončí skenování.

Export dat

1. Uživatel provede kroky případu užití *Skenování pohledu*.
2. Po ukončení skenování si uživatel stáhne teplotní mapu se zdrojovým obrázkem jako podkladem, bez něj nebo strojově čitelná data.

Případy užití / Funkční požadavky	Nahrání vstupu	Uživatelem definované sady dat	Detekce nepřítomnosti obličeje	Kalibrace	Zaznamenávání pohybu očí	Pozastavení skenování	Generování teplotní mapy	Export výstupu
Zadání kódu sady		×						
Nahrání z lokálního úložiště	×							
Nahrání z internetu	×							
Nahrání webové stránky	×							
Skenování pohledu			×	×	×	×	×	
Export dat								×

Tabulka 1.2: Tabulka pokrytí funkčních požadavků veřejné sekce případy užití

1.8.3 Zabezpečená sekce

Do zabezpečené sekce se dostane pouze administrátor. Základní predispozicí je úspěšná autorizace dle případu užití „Přihlášení do systému“. Tabulka 1.3 zobrazuje pokrytí jednotlivých funkčních požadavků případy užití.

Přihlášení do systému

1. Uživatel otevře webový portál pro administrátory.
2. Uživatel zadá uživatelské jméno a heslo.
   a) Zadané údaje jsou správné a uživateli se zobrazí administrační rozhraní.
   b) Údaje nejsou správné, uživatel pokračuje krokem 2.

Zobrazení sad dat

1. Administrátor se přihlásí do systému.
2. Administrátorovi se zobrazí tabulka obsahující jednotlivé záznamy.

Procházení záznamů

1. Administrátor následuje kroky případu užití *Zobrazení sad dat*.
   a) Administrátor klepne na tlačítko „Další“. Aplikace zobrazí další kolekci záznamů.
   b) Administrátor klepne na tlačítko „Předchozí“. Aplikace zobrazí předchozí kolekci záznamů.

Vyhledávání v záznamech

1. Administrátor následuje kroky případu užití *Zobrazení sad dat*.
2. Administrátor změní text ve vyhledávacím poli.
3. Aplikace načte nové výsledky.

Vytvoření nové sady dat

1. Administrátor následuje kroky případu užití *Zobrazení sad dat*.
2. Administrátor klepne na tlačítko „Přidat záznam“.
3. Administrátor vyplní kód a název sady dat.
4. Administrátor přidá minimálně jeden záznam do sady. Záznam obsahuje název a URL adresu.
5. Administrátor klepne na tlačítko „Uložit“.

Upravení sady dat

1. Administrátor následuje kroky případu užití *Zobrazení sad dat*.
2. Administrátor najde záznam, který chce upravit a klepne na něj.
3. Administrátor může upravit název sady, přidávat, měnit nebo upravovat jednotlivé záznamy. U každého záznamu může měnit název a URL adresu.
4. Administrátor klepne na tlačítko „Uložit“.

Smazání záznamu

1. Administrátor následuje kroky případu užití *Zobrazení sad dat*.
2. Administrátor najde záznam, který chce smazat a klepne na něj.
3. V detailu záznamu administrátor klepne na ikonu tří teček.
4. Aplikace zobrazí rozhraní prvek, ve kterém administrátor musí potvrdit smazání položky.

Odhlášení

1. Administrátor následuje kroky případu užití *Zobrazení sad dat*.
2. Administrátor klepne na tlačítko „Odhlásit“.
3. Aplikace vymaže veškerá lokální data identifikující administrátora a přesměruje jej na přihlašovací obrazovku.

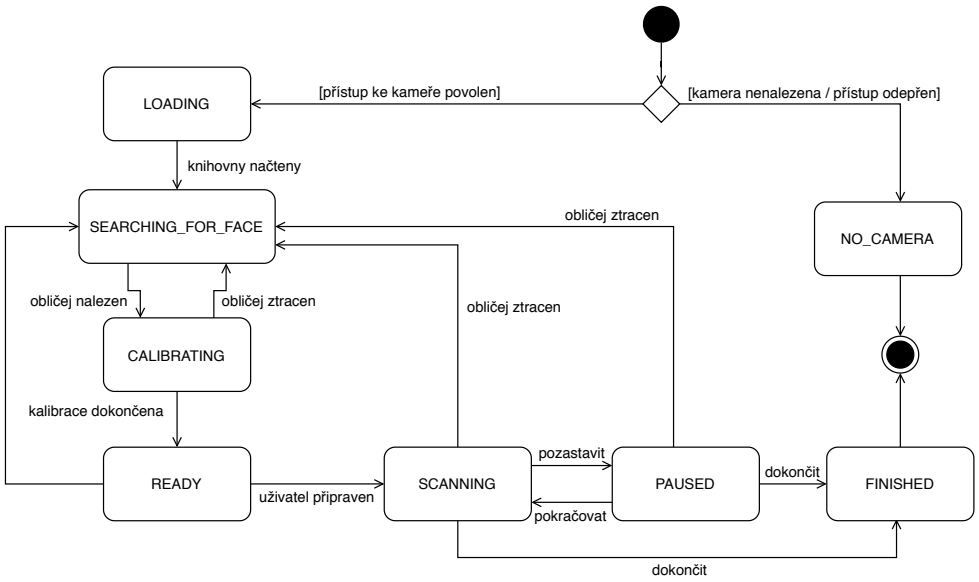
Případy užití / Funkční požadavky	Přihlášení administrátora	Zobrazená sad dat	Filtrace dat	Správa sad dat	Odhlášení
Přihlášení do systému	×				
Zobrazení sad dat		×			
Vyhledávání v záznamech			×		
Vytvoření nové sady dat				×	
Upravení sady dat				×	
Smazání záznamu				×	
Odhlášení					×

Tabulka 1.3: Tabulka pokrytí funkčních požadavků zabezpečené sekce případy užití

1.9 Životní cyklus aplikace

V rámci procesu snímání pohledu uživatele aplikace prochází sadou stavů, které popisuje stavový diagram 1.7 vytvořený dle UML 2.5.1 [23]. Po zadání

vstupních dat dojde k ověření přítomnosti zařízení schopného snímat obraz. V případě potřeby k žádosti o přístup k němu. Pokud uživatel přístup zamítne nebo zařízení není k dispozici, tak dojde k ukončení celého procesu. Přístup k datům z kamery je nutný.



Obrázek 1.7: Diagram životního cyklu aplikace při skenování pohledu

Loading Stav, ve kterém se aplikace nachází, když načítá externí data pro rozpoznaní obličeje.

Searching for face Jakmile jsou data načtena, je potřeba detekovat přítomnost tváře. Do tohoto stavu se dá opět dostat ze všech následujících stavů v případě, že tvář bude ztracena. Po nalezení se uživatel přesouvá do stavu „Calibrating“, protože nelze navázat na data z případných předchozích kalibrací. Zejména kvůli možné změně pozice, nebo pozorovacích úhlů uživatele.

Calibrating Kalibrace je nezbytný krok před zaznamenáváním dat. Zajišťuje zvýšení přesnosti výstupu. Po nakalibrování následuje stav „Ready“.

Ready V tomto kroku je vše připraveno. Uživatel pouze musí odsouhlasit, že je i on připraven započít proces snímání pohledu.

Scanning Aplikace kontinuálně snímá pohled uživatele na obrazovku a data zaznamenává do paměti. Na žádost uživatele se proces ukončí a stav se změní

na „Finished“. Stejně tak lze proces pozastavit, v tom případě následuje stav „Paused“.

Paused Mezistav, který dočasně pozastaví skenování. To lze dále dokončit nebo v něm pokračovat. Uživatel musí stále sedět před počítačem. V opačném případě se aplikace automaticky přesune do stavu „Searching for face“.

Finished Skenování je ukončeno a uživateli je nabídnut export dat dle funkčních požadavků 1.5. Stav je terminální.

1.10 Drátěný model

V konečné fázi mé analytické práce jsem vytvořil sadu drátěných modelů (wireframů). Mým cílem bylo vizualizovat systém, a pokud možno jednotně uchopit průchod jeho životním cyklem. Veškeré obrázky v této sekci jsem vytvořil v nástroji Balsamiq[1].

Po načtení aplikace se uživateli zobrazí základní rozcestník (obrázek 1.8) s třemi možnosti. Každá z nich vede na sledování pohledu. Liší se pouze ve způsobu získání vstupních dat. Po klepnutí na jakékoliv tlačítko se zobrazí dialogové okno 1.9. V něm bude uživatel moci vyplnit vše potřebné. V případě

- načtení webu to je URL adresa,
- nahrání souboru se jedná o vložení URL adresy nebo vybrání obrázku z úložiště a
- u možnosti zadání kódu jej uživatel vloží do textového pole.

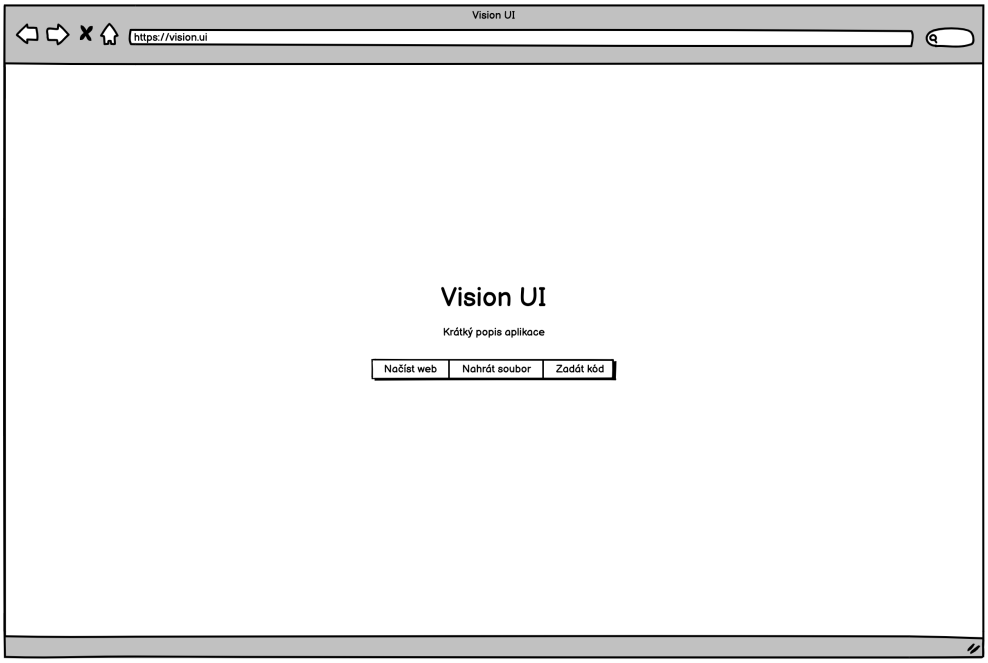
Následně se zobrazí obrazovka 1.10. Na její levé straně se nachází panel s jednotlivými obrázky, mezi nimiž jde přepínat. Pokud je součástí sady pouze jeden obrázek, tak se tento panel ani nezobrazí a aplikace využije pro záznam šířku celé obrazovky.

Pro navigaci mezi jednotlivými stavy aplikace (viz sekce 1.9) slouží dynamická sekce ukotvená v horní části aplikace. Celý proces skenování může uživatel ukončit (resp. pozastavit) tlačítky v pravém horním rohu. Po dokončení se zobrazí dialogové okno. V něm půjde spustit export dat.

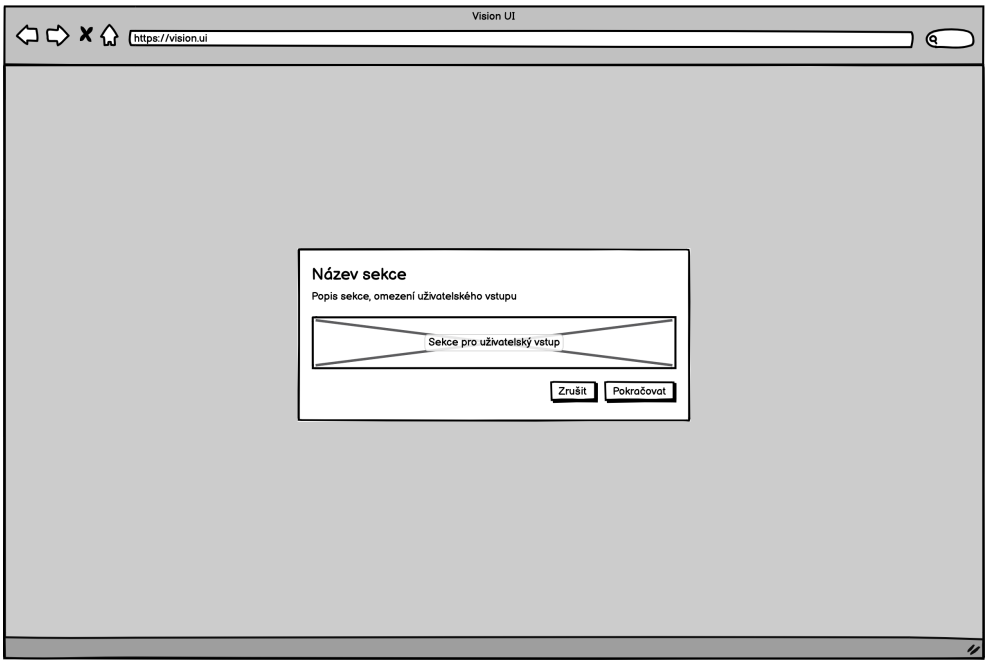
[1] www.balsamiq.com

Po přihlášení do administračního rozhraní 1.11 uvidí administrátor jednu tabulku. Ta obsahuje všechny sady dat a základní informace o nich. Pomocí vyhledávání lze data dodatečně filtrovat. Aplikace podporuje i stránkování. V detailu sady dat ji lze upravovat nebo smazat.

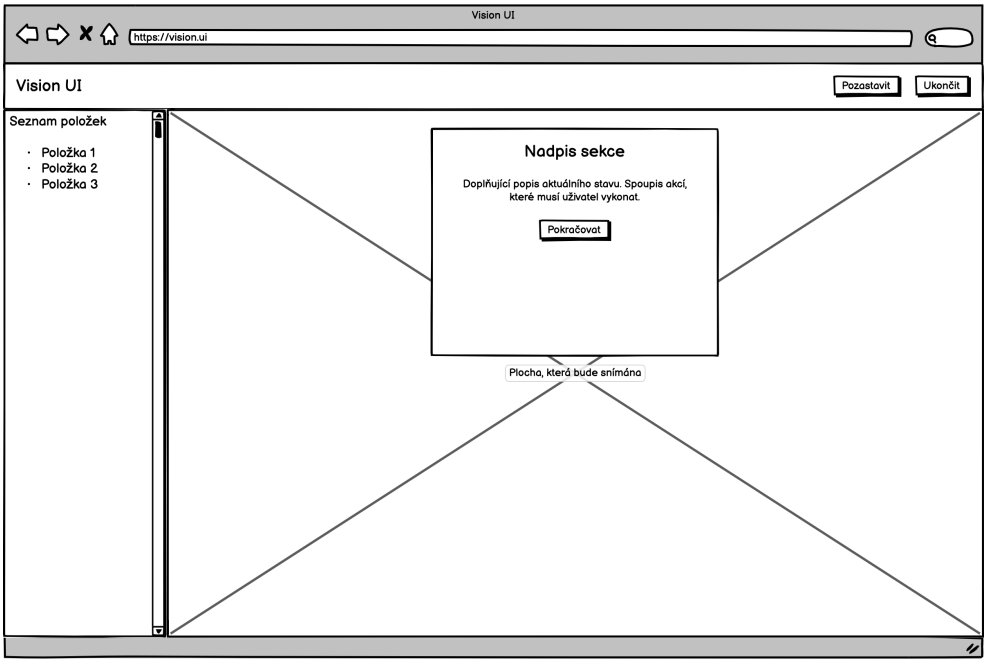
Všechny drátěné modely jsem vložil do přílohy B této práce.



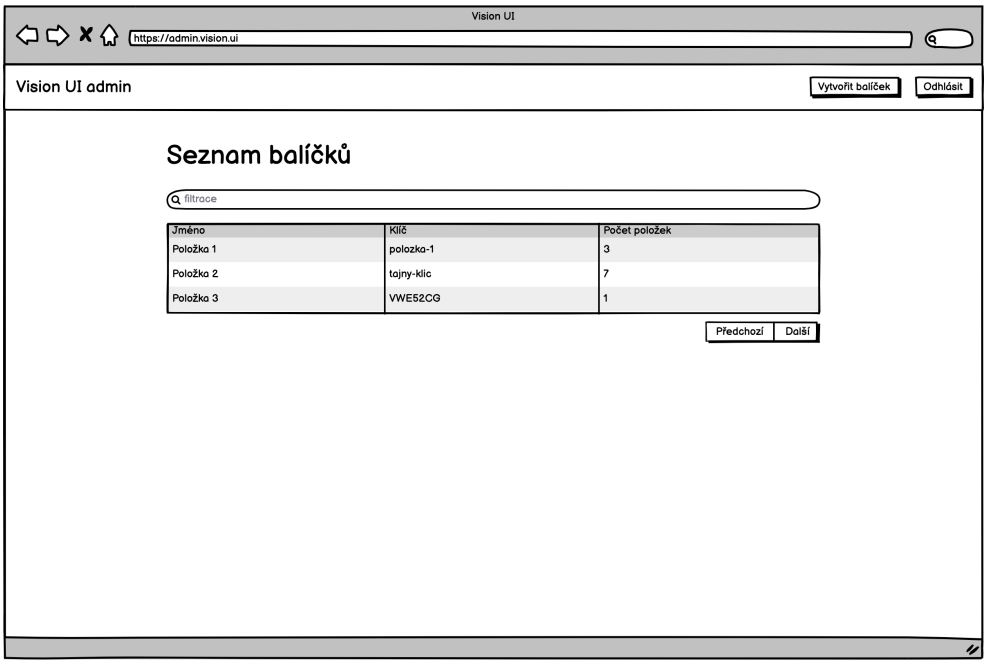
Obrázek 1.8: Hlavní stránka – rozcestník



Obrázek 1.9: Hlavní stránka – vložení vstupních dat



Obrázek 1.10: Wireframe skenování pohledu – rozložení prvků



Obrázek 1.11: Wireframe administrace aplikace

Kapitola 2

Návrh

V této kapitole se blíže věnuji jazyku JavaScript a knihovnám, které použiji pro vývoj aplikace. Při navrhování rozlišuji dva systémy – frontend a backend. S cílem zachovat data i po restartu aplikací dále navrhuji využít databázi. Konkrétně se zabývám výběrem její technologie. V další části řeším model komunikace mezi těmito dvěma systémy a bezpečnost přenosu dat. Na závěr se v této kapitole zaměřím na použité barvy v designu aplikace.

2.1 JavaScript

JavaScript je implementací standardu s názvem „ECMAScript“ spravovaným ECMA International. Jedná se o konsorcium dotované společnostmi Apple, Facebook, Google, Intel, Microsoft a dalšími [24]. První verze ES byla vydána v roce 1997, postupně do dnešního data autoři publikovali dalších 10 verzí. V rámci vývoje nastal největší zlom v roce 2015, roce vydání šesté verze s názvem ES6 nebo ES2015 [25]. Od tohoto milníku vychází nová verze každým rokem.

Co se týče podpory ze strany prohlížečů, ES5 standard splňuje i Internet Explorer 10. Vezmeme-li v potaz aktuální rozšířenost jednotlivých prohlížečů a jejich verzí, tak při kompilaci do ES5 by aplikace mohlo používat statisticky přes 97 procent uživatelů internetu. V případě ES6 se toto číslo snižuje na necelých 95 procent.

```
// (a) JavaScript
function getPassword(clearTextPassword) {
    if (clearTextPassword) {
        return 'password';
    }
    return '********';
}

let password = getPassword('false'); // "password"

// (b) TypeScript
function getPassword(clearTextPassword: boolean) : string {
    if (clearTextPassword) {
        return 'password';
    }
    return '********';
}

let password = getPassword('false'); // throws: error TS2345:
// Argument of type '"false"' is not assignable to parameter
// of type 'boolean'.
```

Kód 2.1: JavaScript vs. TypeScript [26]

2.1.1 TypeScript

TypeScript je open-source rozšířením JavaScriptu, které vyvíjí Microsoft. Mezi hlavní přidané funkce patří typovost a možnost striktně popisovat atributy objektů. Jelikož se jedná o rozšíření, tak jakýkoliv kód napsaný v JavaScriptu lze zkompilovat TypeScript kompilátorem. Veškerá přidaná typovost se během kompilace ztrácí. Proto se TS spíše využívá pro zlepšení čitelnosti kódu a do jisté míry i dokumentace. Vynucováním striktní typovosti nenutí programátora se učit, jak se JavaScript může chovat v mezních situacích. Jedním z příkladů může být případ, kdy řetězec s hodnotou `false` JavaScript umí vyhodnotit v „if“ konstruktu. TypeScript v tomto případě kód 2.1 nezkompiluje.

2.1.2 Node.js

Další z řady open-source knihoven patřící do JavaScript ekosystému je Node.js. Konkrétně se jedná o běhové prostředí (runtime) pro JavaScript založené na JavaScript enginu V8 [27]. Ten je vyvíjen Googlem a zdrojový kód je volně dostupný. Node.js umožňuje spouštět aplikace napsané v JavaScriptu a dnes je základním kamenem každého backendu napsaném v tomto jazyce. Mezi základní vlastnosti Node.js aplikací patří tyto dvě.

Běh řízený událostmi Srdcem aplikace je jeden kanál, do kterého postupně přicházejí požadavky na zpracování. Tento architektonický styl není nic nového, a využívají ho například i Windows Forms aplikace.

Hlavní vlákno Programátor nemá přímo přístup k vláknům. Veškeré funkce jsou volány na jednom (hlavním) vlákně. Autoři tuto volbu odůvodňují snahou eliminovat uváznutí a zlepšit (horizontální) škálovatelnost aplikací [28]. Node.js podporuje i asynchronní operace (například I/O). V implementaci se uplatňují funkce vyšších řádů a návrhový vzor „strategy“, resp. jeho speciální případ zvaný „callback“. Volané funkci se předá další funkce, která je vykonána po dokončení asynchronní operace s výsledkem.

Samotného asynchronního programování lze docílit pomocí dvou klíčových slov `async` a `await`. Await není blokující, tzn. program je schopen vykonávat jiné operace na stejném vlákně při čekání na dokončení té asynchronní. Node.js od programátora vyžaduje základní znalost asynchronního programování. Díky tomu lze optimalizovat zpracování dat, a tím pádem i rychlost odpovědi na dotaz.

2.1.3 Node package manager

NPM je software pro správu Node.js balíčků. Jedná se o vůbec největší registr softwaru obsahující přes 800 000 záznamů [29]. Používá se především pro šíření open-source softwaru. Pro přidání a stažení nové závislosti je potřeba příkazová řádka. Závislosti se dělí do produkčních a vývojářských. Příkladem vývojářských mohou být třeba pomocné soubory pro TypeScript. Ty slouží vývojáři k identifikaci typovosti funkcí, tříd a dalších části poskytovaných knihovnou a nepřikládají se do zkompilovaných produkčních souborů.

V každém Node.js projektu NPM vytváří adresář `node_modules`, kam ukládá zdrojové soubory závislostí. Tento adresář se zpravidla vyjímá, pokud se zdrojové kódy sdílejí mezi uživateli. Knihovny se dají stáhnout dodatečně pomocí příkazu `npm install`, navíc změna verze závislosti by se měla propisovat do zdrojových kódů minimálně.

Pro správu závislostí a jejich verzí existuje soubor `package.json` v kořenovém adresáři projektu. Jeho obsah je ve formátu JSON a obsahuje základní informace o aplikaci – její jméno, verzi, skripty a zmiňované závislosti. Kompletní seznam atributů je k dispozici na docs.npmjs.com.

2.2 Frontend – použité technologie

Možností, jak vyvinout frontend, je nespočet. Základem pro všechna řešení je znalost HTML a CSS. S pomocí HTML se skládají jednotlivé bloky UI a CSS slouží k jejich dodatečnému stylování. JavaScript, který se vykonává v prohlížeči, se používá k dynamické změně UI na základě stavových komponent.

2.2.1 Angular

Aplikaci lze vyvinout bez použití dodatečných knihoven nebo komponent, ale pro usnadnění práce jsem se rozhodl použít framework Angular. Jedná se o open-source projekt vyvíjený společností Google publikovaný pod MIT licencí. Pro svůj běh vyžaduje TypeScript, což je dle mého názoru velká výhoda. Dále nastavuje pravidla, jak psát jednotlivé části aplikace, a tím přímo vynucuje architektonický styl MVC. Ten je založen na třech hlavních komponentách:

- model – objekt reprezentující aktuální stav komponenty,
- view – výsledná podoba modelu, v našem případě jeho reprezentace v HTML a CSS,
- controller – prostředník mezi view a modelem. Obsahuje logiku, která způsobuje změnu modelu, a díky tomu i překreslení view.

2.2.1.1 Module

Základní stavební jednotkou v Angularu je NgModule (modul). Aplikace má vždy minimálně jeden, který se konvenčně jmenuje `AppModule`. Moduly umožňují aplikaci dělit do několika částí a následně mezi nimi vytvářet závislosti. Často používaným modulem je `RouterModule`, který za základě aktuální cesty v URL adrese umí vybrat komponentu, která vykreslí UI.

U modulu je důležitý dekorátor `NgModule`. Do jeho atributů se vkládají metadata, ze kterých lze vyčíst provázanost systému. Mezi hlavní patří:

- „declarations“ – pole vlastních komponent a direktiv, které se v modulu využívají,
- „imports“ – seznam jiných modulů potřebných pro spuštění toho aktuálního,

- „exports“ – kolekce deklarací, které by měly být dostupné v komponentách cizích modulů a
- „bootstrap“ – hlavní view celé aplikace, definuje se jen na úrovní hlavní komponenty a je jedno.

2.2.1.2 Component

Component (komponenta) generuje uživatelské rozhraní na základě aktuálního stavu. Využívá data binding – techniku, která propojuje producenta s konzumenty a zajišťuje správnou synchronizaci dat. Pomocí data bindingu a dodatečného zpracování ze strany Angularu lze HTML šablony rozšířit o další syntax. Ten se využívá k:

- přegenerování výsledného HTML kódu dle stavu modelů (property binding),
- ad hoc napojení na akce, které uživatel vykonává v UI (event binding).

Komponenty lze do sebe zanořovat. Díky této vlastnosti lze systém rozdělit na nejmenší atomické části, přičemž každá řeší jednu specifickou úlohu. Tato vlastnost také podporuje znovupoužitelnost.

2.2.1.3 Service

Service (služba) je stavební jednotka, která negeneruje UI. Primárně slouží pro implementaci logiky, která může být sdílená napříč komponentami. Data produkovaná službou můžou vést k překreslení UI. Potřeba je ale komponenty, která se na ně napojí, a dále zpracuje. Framework umí zajistit, že v celé aplikaci bude existovat pouze jedna instance dané služby, proto se hodí i pro ukládání dat do mezipaměti.

2.2.1.4 Dependency Injection

Vkládání závislostí je jedna z technik, která umožňuje separací závislostí. Angular uplatňuje dependency injection při propojování služeb s dalšími částmi systému. Potřeba je pouze službě přidat dekorátor `@Injectable`. Pro využití služby se jen přidá jako atribut do primárního konstruktoru třídy. Při sestavování vytvoří kompilátor strom závislostí a ověří, jestli jde všechny požadované třídy nainicializovat. Program se nezkompiluje, pokud je zjištěna některá třída, která nepůjde za běhu vytvořit. Po spuštění jsou instance tříd vytvářeny dle potřeby (lazy inicializce).

2.2.2 Další software

Pro zjednodušení vývoje využiji sadu knihoven.

Angular Pro implementaci využiji i další dodatečné knihovny pro Angular. Mezi ně patří například **forms** pro validaci obsahu formulářů, **material** pro uživatelské rozhraní (viz sekce 2.7) nebo **router** pro snadnější navigaci v aplikaci.

heatmap.js Knihovna pro generování teplotních map. Na základě seznamu bodů a intenzity vykreslí na plátno (HTML canvas) výsledek. Dostupná je volně pro jakékoliv použití.

rxjs RxJS je knihovna umožňující reaktivní programování v JavaScriptu. Využiji ji hlavně k lepší propagaci změn v datech do komponent. Publikována je pod Apache 2.0 licencí.

WebGazer.js Knihovna umožňující odhadovat, kam se uživatel dívá na obrazovku dle pohybu kurzoru a klikání. Detailněji je popsána v sekci 1.2.3.

2.3 Backend – použité technologie

Pro implementaci lze přímo použít Node.js, kterému jsem se věnoval v sekci 2.1.2. Na webových stránkách Node.js je k dispozici ukázka, jak implementovat základní verzi, která vypíše „Hello World“. Celkově má 14 řádků. Z kódu 2.2 lze pozorovat hned několik nedostatků. Existuje jedno centrální místo, kde se řeší příchozí požadavky. V něm je potřeba implementovat rozlišení metody, ověřit cestu a podle ní načíst tělo nebo parametry požadavku.

2.3.1 Express

Řešení problémů nabízí framework Express postavený nad Node.js. Pomocí třídy `Router` se vytváří sady cest, u kterých se aplikuje sdílená logika při jejich zpracování. Příkladem může být třeba autorizovaná a veřejně dostupná část rozhraní. Jednotlivé cesty se registrují zvlášť a nepovinně i navazují na specifickou HTTP metodu, pomocí speciální signatury lze části v cestě mapovat na dynamické řetězce.

Express zpracovává požadavek postupně dle návrhového vzoru „chain of responsibility“. JavaScript zároveň dovoluje do objektů přidávat atributy dyna-

```
const http = require('http');

const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello World');
});

server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

Kód 2.2: Kód pro spuštění serveru v Node.js

micky. Spojením těchto dvou vlastností umožňuje psát dodatečné funkce, které budou přidávat mezivýsledky k objektu požadavku. Příkladem je knihovna multer, která je popsána níže.

2.3.2 Další software

V kombinaci s Express frameworkem použiji i tyto knihovny.

cors Obsahuje implementaci CORS standardu pro Node.js. Bližší informace jsem již popsal v sekci 1.4.1.

dotenv Knihovna umožňující simulaci systémových proměnných. Zdrojem je `.env` soubor v kořenovém adresáři repozitáře. Lze tak použít různé nastavení pro vývojové a produkční instance serveru.

express-basic-auth Přidává podporu Basic Authorization pro Express. Detailnějšímu popisu této problematiky se věnuji v kapitole 2.6.

multer Knihovna, která přečte tělo požadavku, pokud se jedná o oktetový stream. Přetransformuje jej do souborů a připojí je jako atributy k požadavku.

pdf-image Vykreslování PDF souborů nemusí být přímo podporováno prohlížeči. Proto na serveru využiji tuto knihovnu, která PDF soubor umí převést do obrázku.

pg Knihovna umožňující připojení se k PostgreSQL databázi a provádění CRUD operací nad ní.

puppeteer Dle analýzy v sekci 1.4.2 je potřeba podpora vykreslování webu. Tato knihovna řeší daný požadavek, proto ji využiji v souladu s její Apache 2.0 licencí.

Knihovna	Licence
cors	MIT
dotenv	BSD-2-Clause
express-basic-auth	MIT
multer	MIT
pdf-image	MIT
pg	MIT
puppeteer	Apache 2.0

Tabulka 2.1: Licence použitých server knihoven

Přehled licencí, pod kterými jsou knihovny publikovány, je v tabulce 2.1. Využití jednotlivých knihoven nijak neomezí výslednou aplikaci, dokonce půjde publikovat i na komerční bázi.

2.3.3 Architektura serveru

Framework Express na rozdíl od Angularu nijak nepředepisuje, jak se postavit k návrhu systému. Při zpracování dat je vždy na vstupu požadavek od klienta a výstup se zapisuje do odpovědi.

Po zajištění přehlednosti rozděluji třídy do třech kategorií, resp. vrstev dle jejich zodpovědnosti:

- prezenční – interaguje s třídy Express frameworku,
- aplikační – zpracovává jednotlivé požadavky z prezenční vrstvy, komunikuje s datovou,
- datová – stará se o změnu stavu dat (CRUD operace).

Tato architektura se též nazývá třívrstvá. Jednotlivé vrstvy od sebe oddělují použité knihovny ve snaze snížit jejich provázanost celým systémem. Třídy Express frameworku se vyskytují pouze v prezenční vrstvě. V této vrstvě dále probíhá transformace dat do modelů, které slouží jako vstup pro zpracování

v aplikační vrstvě. Ta je vyhodnotí a vrátí výstupní model. Model prezenční vrstva opět přetransformuje do jiné podoby vhodné pro Express.

Základní jednotkou v aplikační vrstvě je „use-case“. Jedná se o izolovaný úkon, který má jeden vstupní model a jeden výstupní model. Use-case může být závislý na jiných use-casech a exekuovat je. Například aktualizace dat se může skládat ze dvou operací: smazání a vytvoření dat. Každý use-case má přístup do datové vrstvy. Načítá z ní data a dle povahy požadavku je mění.

Datová vrstva poskytuje funkce pro vytvoření, čtení, úpravu a smazání entit. Entita je jednotka, která reprezentuje určitý model existující v systému. Přistupovat do datové vrstvy umí jen třídy aplikační vrstvy.

2.4 Databáze

Databáze je přímo svázána s datovou vrstvu. Server s databází komunikuje pomocí ovladače, který se doinstalovává separátně. Seznam všech podporovaných je k dispozici na webu expressjs.com. Obsah by se dal rozdělit na dvě části – SQL (MySQL, Oracle, PostgreSQL, ...) a NoSQL (MongoDB, Neo4j, ...) databáze.

Hlavní rozdíly mezi SQL a NoSQL databázemi popisuje tabulka 2.2, přičemž obsahuje zkratky ACID a BASE. Akronym ACID popisuje čtyři vlastnosti – změna v databázi buď proběhne celá nebo žádná, databáze je vždy v konzistentním stavu dle pravidel, změny jsou viditelné až po jejich kompletním uložení a jsou trvalé. BASE je o něco benevolentnější. Vyžaduje konzistentní stav někdy po dokončení transakce, databáze nemusí být schopna vyřídit požadavek a může se měnit asynchronně pro zajištění dodatečné konzistence.

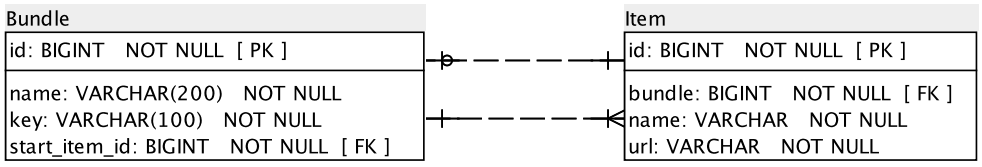
SQL	NoSQL
Typu „ACID“.	Typu „BASE“ nebo i „ACID“
Databáze má předem dané schéma.	Dynamické nebo neznámé schéma.
Lepší vertikální škálovatelnost.	Lepší horizontální škálovatelnost.
Data jsou uložena v tabulkách.	Data jsou uložena jako dokumenty, grafy, páry „klíč - hodnota“, ...

Tabulka 2.2: Rozdíly mezi SQL a NoSQL databázemi

Nakonec jsem se kvůli garanci spojenou s ACID pravidly rozhodl pro SQL databázi. Konkrétně pro PostgreSQL, zejména kvůli jeho otevřenému zdrojovému kódu.

2.4.1 Databázové schéma

V databázi budou existovat dvě entity. Entita „Bundle“ reprezentuje sadu dat obsahující jednu nebo více položek. Každá z nich má vlastní pojmenování a URL adresu, kde se obrázek nachází. Zároveň každý „Bundle“ má přesně jeden „Item“, který je označen jako výchozí. Tento příznak značí, že bude použit jako první při skenování.



Obrázek 2.1: Schéma databáze serveru

2.5 Propojení backendu a frontendu

Obě dvě hlavní části této práce vyvinu nezávisle a je potřeba najít způsob, jak si budou moci předávat data. V tomto případě se přímo nabízí architektonický styl „klient-server“. Klientem se stane prohlížeč uživatele, ve kterém poběží javascriptová aplikace. Serverem může být jedna nebo vícero instancí backendu.

Komunikaci mezi nimi zajistí předem definované rozhraní neboli API. To poskytne sadu funkcí, které zajistí interakci mezi jednotlivými programy. Výměna dat bude probíhat v souladu s REST architekturou, kterou vydefinoval Roy Thomas Fielding v roce 2000 ve své disertační práci [30]. Na serveru tak vytvořím REST API, ke kterému se pomocí HTTP protokolu napojím.

2.5.1 API

API bude obsahovat několik zdrojů, každý z nich jednoznačně definují HTTP metoda požadavku a cesta v URI. Dle specifikace v knize „REST API Design Rulebook“ [31] se budu držet následujících pravidel při přiřazování HTTP metod k funkcím, které na ně budou navázány:

- **GET** – načte data odpovídající požadavku,
- **POST** – vytvoří nový záznam dle těla požadavku,
- **PUT** – upraví nebo vytvoří nový záznam dle těla požadavku,
- **DELETE** – smaže existující záznam, pokud existuje.

```
{
  "id": 473,
  "title": "John's store",
  "nationality": null,
  "public": true,
  "phones": [ "+420555444333", "+420777999222" ],
  "owner": {
    "firstName" : "John",
    "lastName": "Doe"
  }
}
```

Kód 2.3: Ukázka dat v JSON formátu

2.5.1.1 JSON

JavaScript Object Notation neboli JSON je open-source formát pro přenos textových dat. Jednotlivé atributy kóduje do párů „klíč“, „hodnota“. Klíčem je vždy textový řetězec. Hodnotou může být boolean, číslo, řetězec, pole, JSON objekt nebo `null`, viz ukázka 2.3.

Společně s XML se dnes JSON používá pro přenos textových dat po síti. Nativně je podporován i javascriptem, proto jsem se rozhodl pro jeho použití v těle požadavků na server a odpovědí serveru.

2.5.1.2 Návrh API

Následující část zobrazuje HTTP metodu, relativní cestu a popis funkce, kterou vykonávají. Části cest ve složených závorkách reprezentují parametry, které se mění dynamicky. V případě úspěchu je implicitně návratovou hodnotou kód HTTP status 200 (pokud není specifikováno jinak).

GET /screenshot?url={url}&width={width} Vytvoří snímek webu, který je dostupný pod URL adresou `{url}`. Šířka obrázku bude `{width}` pixelů. Výstup server vrátí v binární podobě a formátu JPG.

GET /image?url={url} Načte data ze zdroje reprezentovaným URL adresou `{url}`. V případě, že se jedná o obrázek, tak jej vrátí v odpovědi.

POST /pdf Vyhledá v požadavku, který musí být ve formátu „form-data“, soubor s názvem „pdf“. V případě přítomnosti jej převede do JPEG obrázku.

Ten vrátí v odpovědi. Pokud požadavek parametr neobsahuje, vrátí HTTP status kód 400.

GET /login Ověří, jestli přihlašovací údaje uživatele jsou správné pro přístup do administrace. V pozitivním případě vrátí HTTP kód 204, jinak 401.

GET /bundle?limit={limit}&offset={offset}&query={query} Načte všechny sady dat. Výsledky jsou omezeny třemi parametry. Jejich maximální počet ovlivňuje `{limit}`, `{offset}` určuje pořadí prvního navráceného záznamu z celkové kolekce. Konečně `{query}` je textový řetězec, který musí obsahovat klíč nebo název sady.

GET /bundle/{key} Načte sadu dat reprezentovanou jedinečným klíčem `{key}`. Pokud neexistuje vrátí HTTP status kód 404.

POST /bundle Vytvoří novou sadu dat dle požadavku a vrátí HTTP kód 201. V těle se nachází nově vytvořený objekt Pokud je alespoň jeden parametr nevalidní, server vrátí HTTP kód 400.

PUT /bundle/{id} Upraví již existující sadu, kterou reprezentuje identifikátor `{id}`. Tělo požadavku je identické jako v případě „POST /bundle“ požadavku. Stejně tak propagace chyb.

DELETE /bundle/{id} Smaže sadu identifikovanou identifikátorem `{id}` a vrátí HTTP kód 204. Pokud neexistuje, tak vrátí kód 404.

Kompletní specifikaci s ukázkami těl požadavků i odpovědí a možných chybových kódů jsem sepsal podle OpenAPI 3.0 specifikace. Zdrojový soubor ve formátu YAML a zkompilovaný v HTML jsou součástí kódu serveru. Na přiloženém médiu se nachází ve složce `/src/impl/vision-be/spec`.

2.6 Zabezpečení

Jelikož aplikace obsahuje administrační rozhraní, tak je potřeba zajistit, aby přístup do něj byl řízený. Standard, který by šlo využít, je OAuth 2.0. Díky němu by šel pomocí parametru „scope“ jednoduše řídit přístup k jednotlivými funkcionalitám [32]. Díky JWT přístupovým tokenům omezit délku platnosti přístupu a jednoduše ověřovat jejich autenticitu [33].

S ohledem na velikost administrované části jsem se rozhodl uchýlit k jednodušímu řešení. Implementaci OAuth 2.0 vidím ale jako jeden z dalších kroků při vývoji systému v budoucnosti. Pro autentizaci administrátora naimplementuji Basic access authentication (dále BAA).

2.6.1 Base64

Base64 je kódování, které se využívá pro implementaci BAA. Vstupní binární proud dat zpracovává pro 3bajtových kusech. Každý z nich zakóduje do 4 znaků z ASCII abecedy [34]. Jelikož 3 bajty obsahují 24 bitů, tak každých 6 bitů převede na jeden znak. Pro jednoznačnou transformaci je potřeba 64 možných výstupních znaků, protože počet možných kombinací 0 a 1 v řetězci o délce šesti znaků je 2^6.

V případě, že délka vstupu není zrovna násobkem 24, je možno data zarovnat. Dle specifikace jsou na konec doplněny nuly tak, aby délka byla násobkem 8. K dosažení úplného zarovnání (tedy délky, která je násobkem 24) je na konec přidán jednou nebo dvakrát speciální ASCII znak. Pro odstranění zarovnání je nejdříve nutné odebrat z konce všechny výskyty speciálního znaku. Doplněné nuly nelze vždy odstranit jednoznačně.

Tabulka 2.3 zobrazuje postupný převod binárních dat do Base64 kódování. V prvním sloupci jsou data shlukována do oktetů, v druhém do 6bitových skupin s přidanou výplní (modře), nevyužitá místa jsou znázorněna zeleně. Ve třetím sloupci jsou data převedena do desítkové soustavy. a konečně ve čtvrtém jsou převedena dle RFC 4648 na znaky, nevyužitá místa jsou nahrazena speciálním znakem. Což je „=“ dle specifikace. První řádek znázorňuje převod 24bitového vstupu, bez nutnosti přidání výplně. Druhý a třetí řádek zobrazuje, jak dochází k jeho doplnění. V případě třetího a čtvrtého řádku lze pozorovat kolizi kódování, která může nastat, pokud vstup končí nulami.

Base64 kódování existuje ve několika verzích a liší se hlavně v použitých znacích, protože některé (například lomítko „/“) mohou narušovat strukturu URL adres. V případě BAA je aplikována sada znaků definována v RFC 4648, kterou lze použít i pro URL adresy, výplň na 3 bajty je povinná.

2.6.2 Basic access authentication

Basic access authentication umožňuje bezpečnou autentizaci uživatele za předpokladu, že komunikace mezi uživatelem a serverem je šifrovaná [35]. Standard

Binární data	Data s výplní	Desítková soustava	Base64
10011001 10011001 10011001	100110 011001 100110 011001	38 25 38 25	mZmZ
10011001 10011001	100110 011001 100100 xxxxxx	38 25 36 xx	mZk=
10011001	100110 010000 xxxxxx xxxxxx	38 16 xx xx	mQ==
10011001 00	100110 010000 xxxxxx xxxxxx	38 16 xx xx	mQ==

Tabulka 2.3: Převod binárních dat do Base64

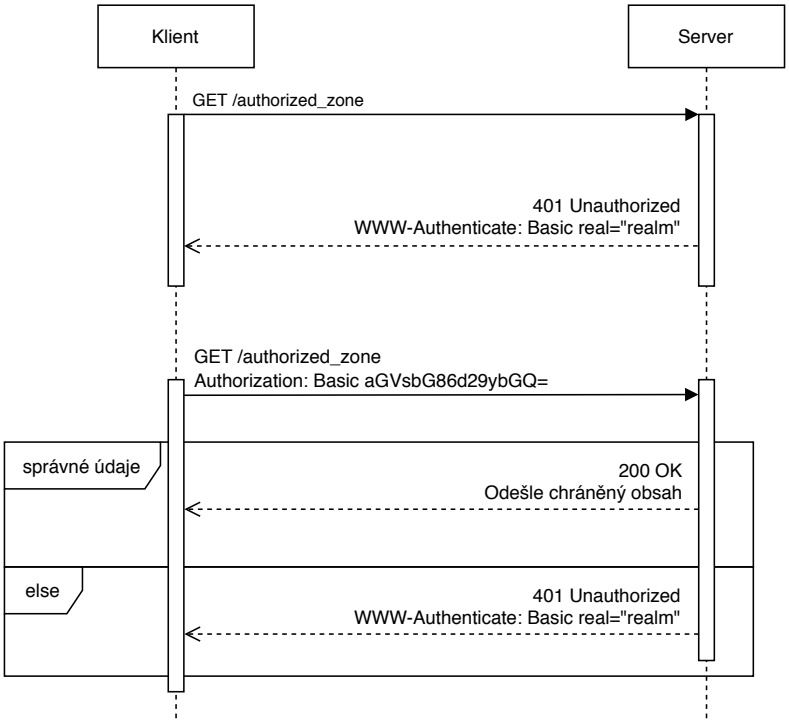
```
encoded = Base64.encode(username + ":" + password)
```

Kód 2.4: Zakódování uživatelského jména a hesla dle BAA

totiž přenáší dvojici parametrů – uživatelské jméno a heslo v nešifrované podobě. Pro další část předpokládám, že uživatelské jméno je reprezentováno proměnnou `username` a heslo proměnnou `password`.

Pro jejich zakódování se využívá Base64. Základní predispozicí je, že vstup musí být oktetový řetězec. Tím se odstraňuje možný problém nejednoznačnosti při dekódování, protože padding lze odstranit jednoznačně (viz řádky 2 a 3 v tabulce 2.3). Zakódovaní proběhne dle pseudokódu 2.4. Výsledná hodnota je předána v požadavku na server v hlavičce s klíčem `Authorization` a hodnotou `Basic encoded`. Jelikož uživatelské jméno a heslo se spojují dvojtečkou, je nutno zajistit, aby se tento symbol nevyskytoval v uživatelském jméně.

Sekvenční diagram 2.2 zobrazuje, jak může vypadat komunikace mezi serverem a klientem. Při dotazu na chráněný obsah pod cestou `/autorized_zone` server přístup zamítne a připojí hlavičku `WWW-Authenticate`. Ta klientovi říká, že musí dojít k autentizaci pomocí BAA. Parametr `realm` je dle specifikace povinný a označuje název chráněné zóny [35]. Pokud je klientem webový prohlížeč, tak uživateli zobrazí dialogové okno pro vyplnění jména a hesla a následně jej zakóduje dle 2.4 a odešle. Server ověří dané údaje a přístup udělí nebo zamítne.



Obrázek 2.2: Autentizace uživatele pomocí BAA

2.7 Vzhled aplikace

Pro zrychlení implementace uživatelského rozhraní jsem se rozhodl použít knihovnu obsahující Material design prvky od autorů Angularu.

Součástí specifikace Material designu je i návod na řešení konfliktů, které mohou vzniknout při zadávaní uživatelského vstupu. Například při validaci uživatelského vstupu Material design nařizuje zobrazovat chybu přímo v místě, kde vznikla. Do jisté míry se tak opírá o heuristické desatero Jakoba Nielsena [36].

Material design je založen i na sadě uživatelských studií. Například na přelomu let 2016-2017 a vzorku 600 lidí inženýři Googlu studovali ideální vzhled textových polí [37]. Uživatelé očekávají, že tyto prvky budou vždy graficky ohraničené, zakulacené a s popiskem vně boxu, do kterého se text zadává.

Na základě WCAG 2.1 [38] specifikace a požadavků Material designu jsem vybral čtyři barvy. Ty slouží jako základ barevné palety používané napříč celou aplikací. Pro zajištění dobré čitelnosti všechny tři základní (primární, akcent a varování) splňují kontrastní poměr definovaný WCAG AA standardem. Jed-

notlivé barvy, jejich hexadecimální a pantone reprezentace jsou uvedeny na obrázku 2.3.

Obrázek 2.3: Barevná paleta aplikace

Kapitola 3

Implementace

V této kapitole se věnuji procesu realizace aplikace. Pro vývoj jsem použil následující podpůrný software:

- IntelliJ IDEA 2020.3 – integrované vývojové prostředí od společnosti JetBrains. V tomto programu jsem napsal veškerý zdrojový kód;
- pgAdmin 4.28 – nástroj umožňující připojení k PostgreSQL databázi a provádění operací nad ní;
- prohlížeče Safari, Opera a Chrome – pro základní otestování aplikace během vývoje.

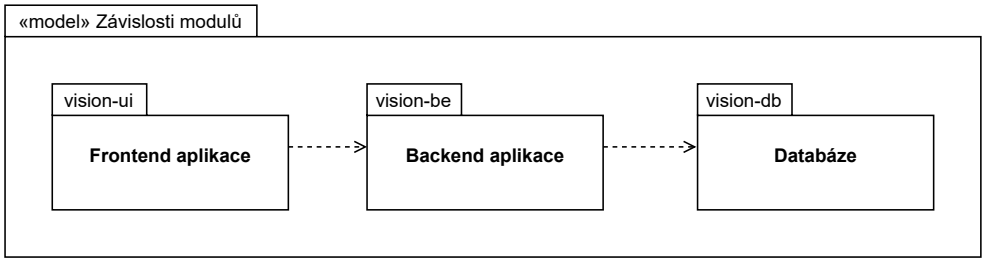
Celý projekt jsem rozdělil do tří modulů:

- „vision-ui“ – frontend aplikace, která generuje uživatelské rozhraní,
- „vision-be“ – backend aplikace poskytující data pro frontend, dále komunikuje s databází,
- „vision-db“ – modul zajišťující perzistentní úložiště.

Každý z nich jde spustit a aktualizovat separátně. Pro plnou funkcionalitu systému je potřeba, aby běžely všechny tři. Závislosti mezi nimi popisuje diagram 3.1.

3.1 Vlastní řešení

Na úplném začátku vývoje jsem se rozhodl, že si vyzkouším napsat vlastní algoritmus na rozpoznávání, kam se uživatel dívá. Vycházel z jednotlivých fází



Obrázek 3.1: Závislosti jednotlivých modulů

rozpoznávání, které jsem popsal v sekci 1.1.

V prvním kroku jsem použil knihovnu face-api.js pro nalezení oblastí, kde se nachází oči. Výstupem při pozitivním nálezu bylo pole 68 bodů. S pomocí dokumentace knihovny jsem pole překonvertoval na dvě další – každé obsahovalo souřadnice jednoho oka. Z bodů jsem vyextrahoval minima a maxima os x a y a vytvořil obdélník. Vně každého se tak nacházelo celé jedno oko. Výšku obdélníku jsem zvětšil o osm a šířku snížil o šest pixelů. Tento krok jsem provedl, protože knihovna měla občas tendenci uřezávat vrchní a spodní části oka. Dále jsem se držel teze, že zornice je spíše ve středu oka, a proto mohu odřezat jeho krajní části.

3.1.1 Nalezení zornice

Oba výřezy jsem poupravil pomocí grafických filtrů dostupných v knihovně OpenCV. Jejich výběr a pořadí aplikace značně ovlivnil kód Antoine Laméa [5]. Vstup jsem nejdříve převedl do stupně šedi, ekvalizoval histogram, použil bilaterální filtr, morfologickou erozi a konečně ještě jednou ekvalizaci histogramu. Postupné změny způsobené aplikací jednotlivých kroků lze pozorovat na obrázku 3.2.

(a) Originál (b) Stupně šedi (c) Ekvalizace hist. (d) Bilaterální filtr

(e) Eroze (f) Thresholding (g) Výstup

Obrázek 3.2: Výstupy algoritmu, který hledá zornici, po aplikaci dané operace

Upravený obrázek jsem následně použil pro kalibraci thresholdingu. V rámci kalibračního procesu jsem se vždy snažil najít ideální prahovou hodnotu, aby na obrázku zůstalo 40 procent černých (a 60 procent bílých) pixelů. Čtyřicet procent jsem zvolil empiricky a během vývoje několikrát upravoval, aby odhad fungoval nejlépe v možných scénářích.

Ke kalibraci docházelo s každým novým snímkem, pokud po aplikaci thresholdingu s posledním validním parametrem nezůstalo na obrázku 30-50 procent černých pixelů. Tento krok jsem zvolil zejména pro zrychlení algoritmu.

Vstupem dalšího kroku byl obrázek s 30-50 procenty černých pixelů. V něm jsem nalezl největší spojitou oblast černých pixelů (na obrázku 3.2 (g) je vyznačena červeně) a určil její geometrický střed. Tento bod (zelená část na 3.2 (g)) jsem považoval za střed zornice.

3.1.2 Konstrukce a projekce 3D vektoru

V dalších krocích jsem počítal s dvěma hodnotami, každá z nich určovala odchylku aktuálních souřadnic od středu oka. Střed oka jsem určil jako průměr x a y souřadnic obdélníku, ve kterém se oko nachází. Pro kalibraci jsem vyžadoval čtyři body – každý byl umístěn v jednom rohu monitoru. Při odhadu, kam se uživatel dívá, jsem zkonstruoval dvě lineární rovnice. Výstupem každé z nich bylo číslo, ideálně v intervalu od 0 do 1. Nula značila, že se uživatel dívá na levý (resp. horní) kraj monitoru, jednička pravý (resp. dolní) kraj.

Každou z hodnot jsem vynásobil šířkou (nebo výškou) kreslícího plátna, a tím dostal souřadnice místa, kam se uživatel zřejmě dívá.

3.1.3 Testování

Vytvořený program jsem testoval pouze na svém počítači (MacBook Pro, macOS 11.0.1, procesor Intel I7-8569U, 16 GB RAM) v prohlížeči Opera verze 72.0.3815.400. Celý běh algoritmu trval přibližně 87 milisekund, systém tak zvládal zpracovat 12 snímků za sekundu, přičemž samotné nalezení 68 bodů identifikujících tvář trvalo v průměru 63 milisekund.

Program dokázal úspěšně nalézt region, kde se nacházely oči, a také identifikovat zornici. Problémem byla kvalita dat, protože vstupní proud dat byl o rozměrech 224x244 pixelů. Sebemenší nepřesnost v případě projekce na monitor s 2K rozlišením způsobila odchylku několik pixelů. Zároveň můj jednoduchý

```
1  webgazer
2    .setRegression('ridge')
3    .setGazeListener(function(data, clock) {
4      component.checkForFace();
5
6      if (data !== null &&
7          component.currentState === ScanningState.SCANNING) {
8        const x = data.x + component.scrollOffset.x;
9        const y = data.y + component.scrollOffset.y;
10       component.cache.add(new Point(x, y));
11     }
12 }).begin();
```

Kód 3.1: Inicializace knihovny Webgazer.js

přístup, co se týče projekce na cílovou plochu (použití lineárních funkcí, využití pouze 4 kalibračních bodů), jen dále zvyšoval nepřesnost výstupu. Vcelku korektního chování jsem dosáhl jen při ideálních světelných podmínkách, a proto jsem se rozhodl jít jinou cestou.

Zdrojové kódy tohoto projektu jsem přiložil k této práci.

3.2 Webgazer.js

Po neúspěchu implementovat vlastní řešení jsem se rozhodl využít knihovnu Webgazer.js. S cílem ji co nejvíce izolovat od zbylého systému jsem pro ni vytvořil vlastní Angular komponentu. Dle stavu systému (viz sekce 1.9) se Webgazer.js zapíná a vypíná. Kód 3.1 reprezentuje proces inicializace. Pro zapnutí je potřeba pouze zavolat metodu `enable()`. Knihovně lze pak předat jednu „listener“ metodu, kterou bude periodicky volat, kdykoliv budou k dispozici nová data.

Po obdržení nových dat, se snažím inicializovat proces ověření přítomnosti obličeje (pokud není inicializován). Kód a dokumentace metody `checkForFace()` je k dispozici v přiložených zdrojových kódech. V dalších krocích k předpověděném bodu přičtu pixely zbývající k dosažení vrcholu obrazovky (řádky 8 a 9). Výsledný bod uložím do dočasného úložiště v paměti.

Díky této implementaci lze knihovnu jednoduše odstranit – stačí smazat komponentu a vytvořit novou, která bude do dočasné paměti ukládat nové body.

3.3 Zpracování dat

Získaná data aplikace následně transformuje do různých forem. Celá procedura se dělí na tři kroky – filtraci, shlukování a vykreslení.

3.3.1 Filtrace

Surová data aplikace musí vykreslit na kreslící plátno. Vstupem algoritmu je kolekce bodů. Každý bod reprezentují jeho 2D souřadnice, časové razítko a identifikátor obrázku, který byl v té době zobrazen jako podklad. Aplikace data před vykreslením nejdříve přefiltruje. Aby byl bod použit, je potřeba, aby splňoval tyto dvě podmínky:

- v době uložení musel být zobrazen stejný podkladový obrázek jako teď,
- souřadnice bodu musí být v oblasti, ve níž je vykreslen podklad.

3.3.2 Shlukování

Jelikož samotné zaznamenávání není přesné na pixel, tak aplikace po přefiltrování data ještě shlukuje. Ve výchozím nastavení se plátno rozdělí do několika regionů, každý má rozměr 20x20 pixelů. Aplikace ze vstupních souřadnic vypočítá výstupní a vytvoří nový seznam bodů. U každého výstupního si drží počet původních bodů, které do něj spadají. V posledním kroku nalezne oblast s nejvíce body. Výstupem je třída `HeatmapModel` obsahující pole bodů s četností a číslo reprezentující maximální četnost.

3.3.3 Vykreslení teplotní mapy

Celý tento proces má na starosti metoda `toHeatMap`, jíž vstupem je pole bodů, obdélník obsahující absolutní souřadnice aktuálně zobrazeného obrázku, jeho identifikátor a škálovací faktor. Komponenta vykreslující data pak jen sleduje změnu uložených bodů, vyvolá transformaci bodů do požadovaného modelu a předá je knihovně heatmap.js. Kód popisující tuto operaci je zachycen v ukázce 3.2.

3.4 Export

Aplikace nabízí export dat do dvou formátů – obrázku s teplotní mapou a textového souboru.

```
// Transformace v~UI vrstvě
// Zdrojový soubor: gaze.component.ts
this.cache.points.subscribe((it) => {
  if (this.heatmapLib !== null && this.canvasRect !== null) {
    const heatmapModel = this.cache.toHeatMap(it, this.canvasRect);

    this.heatmapLib.setData(heatmapModel);
    this.heatmapLib.setDataMax(heatmapModel.max);
  }
});

// Metoda provádějící transformaci
// Zdrojový soubor: cache.service.ts
toHeatMap(
  data: Array<CachePoint>, // uložené body
  clipBounds: Rect | null, // region, který nás zajímá
  requestId: number | null = this.scanService.requestId, // id obrázku
  downscale: number = this.DOWNSCALE_RATIO // 20 ve výchozím nastavení
): HeatmapModel { ... }
```

Kód 3.2: Kód zajišťující vykreslení teplotní mapy

Export obrázku Princip popsaný v předchozí části 3.3 je aplikován i v případě exportu dat do obrázku. Aplikace v tomto případě iteruje přes všechny soubory, které jsou součástí skenované sady. Každý obrázek dynamicky vykreslí, překryje jej plátnem (HTML canvas element), vygeneruje teplotní mapu, oba tyto elementy sloučí a překonvertuje do obrázku.

Export strojově čitelných dat U exportu do strojově čitelných dat je proces omezen pouze na filtraci dat. Shlukování by v tomto případě znamenalo ztrátu přesnosti. Po filtraci aplikace data překonvertuje do řetězce. Jeden záznam reprezentuje jeden řádek ve výstupním textovém souboru. Jednotlivé údaje jsou odděleny středníkem – jedná se o CSV formát. Pozice elementů na každém řádku je pevně daná a odpovídá tabulce 3.1.

Index	Typ	Jednotka	Popis	Příklad
0	celé číslo	ms	UNIX časová značka	1608397575951
1	celé číslo	px	X souřadnice	13
2	celé číslo	px	Y souřadnice	53

Tabulka 3.1: Exportované položky do CSV souboru

```
1  const app = express()
2  const publicRouter = express.Router()
3  const privateRouter = express.Router()
4  const auth = basicAuth({
5      users: {
6          'admin': <string> process.env.AUTH_PASSWORD,
7      }
8  })
9  publicRouter.get('/screenshot', getScreenshot)
10 publicRouter.get('/image', getImage)
11 publicRouter.get('/bundle/:key', getBundle)
12 publicRouter.post('/pdf', upload.single('pdf'), postPdf)
13
14 privateRouter.get('/login', login)
15 privateRouter.get('/bundle', getAllBundles)
16 privateRouter.post('/bundle', createBundle)
17 privateRouter.put('/bundle/:bundleId', updateBundle)
18 privateRouter.delete('/bundle/:bundleId', deleteBundle)
19
20 app.use('/', publicRouter)
21 app.use('/', auth, privateRouter)
22 app.listen(process.env.PORT, () => { ... })
```

Kód 3.3: Integrace REST API na straně backendu

3.5 Propojení backendu a frontendu

V této části navazuji na sekci 2.5, kde jsem navrhl styl komunikace mezi aplikacemi. Zároveň ukazuji, jak jsem implementoval vrstvu zabezpečení popsanou v sekci 2.6.

3.5.1 Backend

Na straně backendu integrace zahrnovala vytvoření dvou tříd typu `Router`. Instance s názvem `publicRouter` v ukázce kódu 3.3 vyřizuje požadavky bez nutnosti provést autorizaci. Druhá instance – `privateRouter` – vyžaduje BAA. V kódu lze tento požadavek pozorovat na řádku 21, kde dochází k registraci routeru do aplikace. Důležitý je druhý argument funkce `use` s názvem `auth`. Díky němu aplikace nejdříve zkontroluje přítomnost autorizace a v případě úspěšného ověření předá řízení routeru. Momentálně existuje pouze jeden uživatelský účet s přihlašovacím jménem „admin“ a heslem, které se nastavuje jako proměnná prostředí. Routery pak dle cesty v URI adrese volají funkce, které požadavky zpracovávají.

3.5.2 Frontend

Základní stavební třídou na straně klienta je třída `HttpClient`, která je součástí balíku „common/http“ Angularu. Pomocí funkcí jako jsou get, post, put, delete... umožňuje volat různé HTTP metody. Odpovědi v JSON formátu umí automaticky mapovat na JavaScript objekty. Výstupem je vždy instance třídy `Observable`. Případné chybové stavy tak lze řešit pomocí funkce `catchError`.

Pro doplnění autorizační hlavičky se využívá návrhový vzor „interceptor“. Před odesláním požadavku na server aplikace zavolá metodu `intercept` všech zaregistrovaných interceptorů. V ní lze odchozí data pozměnit. Logika je tak na jednom místě a nemusí se doplňovat pro každý požadavek. Hlavička se generuje pomocí funkce `btoa`, která převádí řetězec do Base64 řetězce a je součástí základních funkcí, kterými JavaScript disponuje. Všechny tyto kroky lze pozorovat na výňatku kódu aplikace 3.4.

V případě kladné odpovědi serveru si aplikace přihlašovací údaje uloží do paměti a využívá je, dokud nedojde k zavření okna nebo manuálnímu odhlášení.

3.6 Nasazení

Jednotlivé části lze nasadit separátně a díky balíčkovacímu systému NPM stačí mít pouze vlastní zdrojové soubory aplikace. Potřebné knihovny se stáhnou dodatečně. Jak backend, tak frontend lze spustit pomocí dvojice příkazů `npm install` a `npm run`. Backend je závislý na PostgreSQL databázi, kterou je potřeba nainstalovat separátně.

3.6.1 Docker

Pro odstranění možných problémů, které by mohly vzniknout na cílovém zařízení, doporučuji využít nástroj Docker. Docker je open-source nástroj vyvinutý pro zjednodušení vytváření, spouštění a nasazení aplikací [39]. Základem je soubor s názvem „Dockerfile“. Do něj se pomocí příkazů zadává, jak vytvořit Docker obraz. Umožněno je i dědění – jako základ lze použit již předpřipravený Docker obraz a pomocí Dockerfile do něj přidat další soubory nebo dále konfigurovat. Z Docker obrazu se nakonec inicializuje Docker kontejner. Je to izolovaná jednotka běžící na cílovém zařízení [40]. Dokáže využívat pouze zdroje, které byly součástí obrazu při jeho vytváření.

```
// Převod přihlašovacích údajů do Base64
get baseAuthHeader(): string {
  if (this.credentials != null) {
    return btoa(
    `${this.credentials.username}:${this.credentials.password}`
    );
  } else {
    return null;
  }
}

// Doplnění hlavičky, pokud existují přihlašovací údaje
intercept(
  req: HttpRequest<any>,
  next: HttpHandler
): Observable<HttpEvent<any>> {
    const header = baseAuthHeader;
    if (header !== null) {
      req = req.clone({
        setHeaders: {Authorization: `Basic ${header}`}
      });

      return next.handle(req);
    } else {
      return next.handle(req);
    }
  }

// Stažení balíku dat
getBundle(key: string): Observable<Bundle> {
  return this.http.get<Bundle>(
    this.path(`bundle/${key}`),
    {responseType: 'json'}
  );
}
```

Kód 3.4: Integrace REST API na straně frontendu

Mezi hlavní tři výhody, které přináší Docker, patří izolace prostředí, sdílené úložiště a možnost omezit prostředky.

Izolace prostředí Docker kontejner dokáže přistupovat pouze k souborům, které jsou jeho součástí. Veškeré závislosti tak musí být nainstalovány při jeho inicializaci. Díky této vlastnosti lze vždy zajistit požadované verze knihoven. Řeší se tak možný problém, kdy každá aplikace vyžaduje jinou verzi a nelze mít na počítači nainstalovány obě.

Sdílené úložiště Docker umožňuje vytvářet sdílená úložiště, kam kontejnery mohou ukládat data vytvořená za jejich běhu. Díky úložišti, které v systému existuje nezávisle na kontejneru, lze kontejner například aktualizovat a nedojde ke ztrátě dat. V případě této práce je obsah databáze uložen právě v tomto úložišti. Po restartu aplikace tak nedojde ke ztrátě dat.

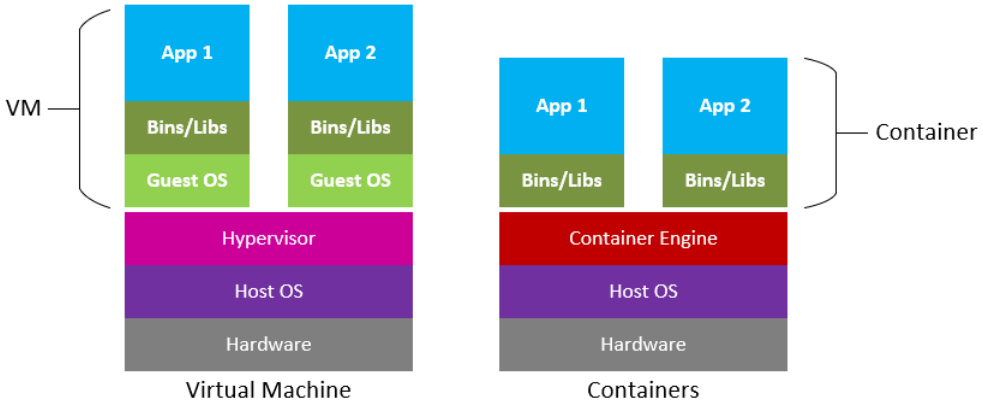
Omezení prostředků Každý stroj má omezené prostředky, ať už se to týká velikosti vyrovnávací paměti, úložiště nebo výpočetního výkonu. Docker je umožňuje každému kontejneru omezit. Díky tomu lze například zajistit, aby jeden z kontejnerů nespotřebovával celý výkon stroje. Špatné nastavení (například nastavení příliš malé velikosti RAM) může ale vést k degradaci výkonu celého systému.

Tyto tři výhody lze získat i pomocí virtualizace operačního systému. Soudě dle obrázku 3.3 je potřeba navíc hypervizor – software pro virtualizaci hardwaru počítače umožňující spuštění vícero operačních systémů. Docker je jednodušší, a také co se týče využití prostředků šetrnější, tím pádem i vyžaduje méně energie.

Všechny součásti systému obsahují vlastní Dockerfile, a lze je nasadit pomocí tohoto nástroje. V případě JavaScript aplikací jsem použil jako základ obraz `node`. Ten obsahuje vše potřebné pro vytvoření aplikací pomocí nástroje NPM. Pro databázi jsem zvolil obraz `postgres`. Pro každou část jsem vytvořil Dockerfile a obrazy rozšířil o zdrojové soubory aplikací a potřebné konfigurační proměnné.

3.6.2 Docker Compose

Docker Compse je nástroj pro vytváření a spouštění aplikací využívající Docker. Pomocí něj lze spravovat několik Docker kontejnerů zároveň a definovat mezi



Obrázek 3.3: Porovnání architektury virtuálního stroje a Dockeru [41]

nimi závislosti. S ohledem na rozvrstvení celkového řešení do několika repositářů a Docker kontejnerů je ideálním doplňkem, který usnadní správu nasazeného řešení jako celku.

Základem řešení je jeden „docker-compose.yml“ soubor, ve kterém se vydefinují jednotlivé služby a jejich specifika. Každá služba má v rámci konfiguračního souboru vlastní název. Jednou z hlavních předností je automaticky vytvořená interní síť.

Interní síť Docker Compose automaticky vytváří interní síť mezi jednotlivými službami, které spravuje. V případě nutnosti komunikace mezi nimi je nutno použít v konfiguračních souborech jejich názvy, nikoliv IP adresy. Z vnější sítě není nic přístupné, pokud se nespecifikuje alespoň jeden výstupní port. Dochází tak ke kompletní izolaci, a tedy i možnému zvýšení zabezpečení, protože některé části výsledného systému nemusí být přímo dosažitelné.

V interní síti má každá služba svou sadu portů, které lze využít ke komunikaci. Docker Compose dále umožňuje zpřístupnění služeb do vnější sítě. Výhodou je, že v každé síti lze mít službu dostupnou na jiném portu. Služby stačí jednou propojit ve vnitřní síti a není třeba řešit, jak se k nim bude přistupovat z vnější sítě. S ohledem na variabilní dostupnost portů na každém koncovém zařízení je rekonfigurace velmi jednoduchá.

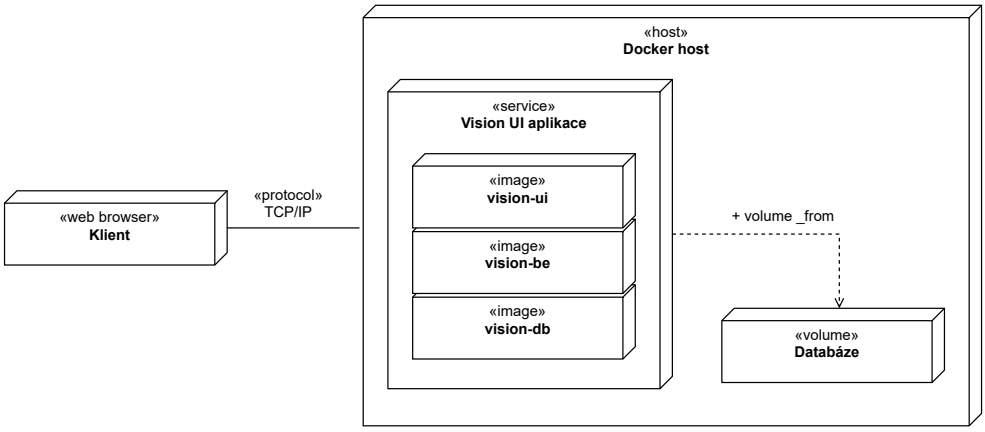
Závislosti Mezi jednotlivými službami lze vytvářet závislosti. Ty definují, v jakém pořadí dojde ke spouštění a ukončování jednotlivých služeb. Nevýhodou je, že Docker Compose nedokáže zajistit dostupnost jednotlivých slu-

žeb. Jinými slovy Docker Compose nedokáže detekovat, že služba je interně plně inicializována (například, že databázový stroj kompletně naběhl). Dokáže pouze rozpoznat, jestli služba běží nebo ne.

3.6.2.1 Výsledný Docker Compose

Výsledný Docker Compose soubor 3.5 obsahuje tři služby (vision-be, vision-ui a vision-db). U každé služby je pomocí parametru `build.context` specifikována složka, ve které se nachází konfigurační Docker soubor a jeho název (`build.dockerfile`). Na řádku 8 lze vidět závislost backendu na databázi. Každá služba je dostupná i ve vnější síti. V případě databáze dochází k mapování portu 5432 vnitřní sítě na port 6768 vnější sítě. Její obsah se na řádku 29 mapuje do na cestu `./vision-db/data` relativní vůči lokaci Docker Compose souboru. Každý z obrazů má limitovanou maximální velikost vyrovnávací paměti na 1 GB a 512 MB respektive (řádky 12, 22 a 32).

Všechny služby lze spustit pomocí příkazu `docker-compose up`. Kompletní informace se nachází v souboru `/src/impl/README.md`, který je součástí přiloženého média. Obrázek 3.4 reprezentuje diagram nasazení v případě využití této konfigurace a nástroje Docker Compose.



Obrázek 3.4: Diagram nasazení aplikace

3.6.3 Git

Git je open-source nástroj pro správu verzí souborů. Umožňuje v průběhu času vytvářet záchytné body (commity). Díky nim lze porovnávat změny ve sledovaných souborech nebo obnovovat jejich starší verze. Každý záchytný bod má svůj jedinečný hexadecimální identifikátor (commit hash). Git dále umož-

```yaml
version: '3.8'
services:
  vision-be:
    container_name: vision-be
    build:
      context: vision-be
      dockerfile: Dockerfile
    depends_on:
      - vision-db
    ports:
      - '127.0.0.1:1111:1111'
    mem_limit: 1G
  vision-ui:
    container_name: vision-ui
    build:
      context: vision-ui
      dockerfile: Dockerfile
    depends_on:
      - vision-be
    ports:
      - '127.0.0.1:80:80'
    mem_limit: 1G
  vision-db:
    container_name: vision-db
    build:
      context: vision-db
      dockerfile: Dockerfile
    ports:
      - '127.0.0.1:6768:5432'
    volumes:
      - './vision-db/data:/var/lib/postgresql/data'
    mem_limit: 512M
```

Kód 3.5: Výsledný Docker compose

ňuje větvení projektu – v pojmenovaných větvích (branches) lze držet různé podoby souborů. Umožňuje tak vícero vývojářům pracovat na různých funkcionalitách paralelně. Nad větvemi lze vystavovat další větve – nepřímo tak podporuje agilní řízení projektu [42]. Vývojáři dle požadavků mohou plynule vyvíjet jiné části systému. Po schválení dokončení implementace se větve nakonec v rámci softwarového procesu sloučí do jedné (merge) a vyřeší se možné konflikty mezi pozměněnými verzemi souborů.

Git v základu podporuje modularizaci projektu [43], kterou jsem využil. Pomocí sady příkazů `git submodule` jsem definoval závislosti na ostatních repo-

zitářích. Závislost je přímo navázaná na konkrétní commit – změny se nepropisují automaticky a vždy je potřeba vytvořit nový záchytný bod v rodičovském repozitáři. Tato vlastnost umožňuje lépe držet konzistenci díla jako celku.

Výsledný repozitář „vision“ obsahuje vazbu na další dva „vision-ui“ a „vision-be“. Pro inicializaci je potřeba mít na cílovém zařízení nainstalován Git a stáhnout repozitář pomocí příkazu `git clone`. V kořenovém adresáři je soubor `README.md` jehož součástí je kompletní návod popisující, jak:

- získat zbylé zdrojové soubory,
- pozměnit URL adresy jednotlivých serverů,
- přemapovat porty v rámci docker compse sítě,
- upravit přístupová hesla k administraci a databázi
- a vše spustit pomocí Docker compose.

Repozitář je v souladu s GPLv3 licencí, která je vynucena knihovnou WebGazer.js, dostupný na BitBucket serveru[2]. Pro stažení a okamžité spuštění na lokálním stroji jsou potřeba pouze čtyři příkazy uvedené v ukázce kódu 3.6. Kompletní návod jsem uvedl v instalační příručce v příloze D.

```
git clone https://bitbucket.org/mroczis/vision.git
cd vision
git submodule update --init --recursive
docker-compose up
```

Kód 3.6: První spuštění aplikace

3.6.4 Nasazení na produkci

Aplikace je připravena na nasazení na produkční server dle zadání této práce. Bohužel kvůli dočasné nedostupnosti serveru v době, kdy jsem tuto práci dokončil, k nasazení nedošlo. Po konzultaci se zadavatelem jsme se rozhodli, že produkční instanci aplikace zprovozníme dodatečně.

[2] https://bitbucket.org/mroczis/vision/

Kapitola 4

Testování

V této kapitole se věnuji testování obou vzniklých systémů. Pro zaručení co nejlepší funkčnosti systému jsem provedl testy obou dvou aplikací. Pro každou jsem ale zvolil jinou strategii.

4.1 Backend

Jelikož backend přijímá požadavky v JSON formátu, tak se pro testování nabízí hned několik možností. Jednou z nich je testování pomocí externích nástrojů – server se spustí v produkčním módu a posílají se na něj požadavky. Jednotlivé změny se kontrolují v databázi. Tento způsob testování má ale spíš více nevýhod než výhod, proto jsem se rozhodl pro automatizované testy.

Využil jsem nástrojů „chai“ a „supertest“. Chai je takzvaná „assertion library“, tedy knihovna vyvinutá speciálně pro účel ověřování hodnot v proměnných. V testech ji využívám, abych zjistil, jestli se po provedení operace nachází v proměnné hodnota, kterou očekávám. Knihovna „supertest“ umožňuje volat jednotlivá rozhraní na serveru za pomocí HTTP protokolu.

Jelikož jsem server rozdělil do tří vrstev, tak každou z nich jsem i nezávisle otestoval.

Testování serveru Běžný testovací scénář začíná exekucí alespoň jednoho požadavku na server pomocí knihovny supertest. V této vrstvě jsem testoval převody parametrů v URL adrese, JSON výstup nebo správnost reakce serveru na různé hodnoty autorizační hlavičky.

Testování uživatelských případů V případě této vrstvy jsem vytvořil přímo vstupní data ve formě javascriptových objektů a ověřil jednotlivé výstupy, jimiž byly také objekty.

Testování databáze U databáze jsem ověřil správnost SQL dotazů a převod výstupu databázového stroje do javascriptového objektu. Jeden z testů například ověřuje vložení a smazání záznamu. Zobrazen je v kódu 4.1.

```
describe('BundleDao tests', () => {
    it('Create and delete item', async () => {
        const dao = new BundleDao()
        const bundle = await dao.insert("name", "key", null)

        expect(bundle.id).to.be.greaterThan(0)
        expect(bundle.name).to.equal("name")
        expect(bundle.key).to.equal("key")
        expect(bundle.startItemId).to.equal(null)

        const deleted = await dao.delete(bundle.id)
        expect(deleted).to.be.equal(true)
    })
})
```

Kód 4.1: Ukázka testu databáze

4.2 Frontend

4.2.1 Heuristická analýza

Po dokončení implementace jsem provedl heuristickou analýzu. Založil jsem ji na 10 Nielsonových pravidlech [36], které jsem se v rámci návrhu aplikace snažil dodržet.

4.2.1.1 Nielsonova kritéria

V této části se věnuji jednotlivým kritériím, a jak jsem se je snažil splnit během vývoje.

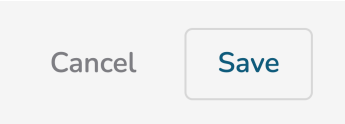
Viditelnost stavu systému U akcí, které zpravidla trvají delší dobu, jsem se vždy snažil zobrazit UI prvek signalizující nahrávání. Příkladem je načítání potřebných externích souborů pro rozpoznávání obličeje. V rámci procesu zaznamenávání pohybu očí uživatele vždy informuji o aktuálním kroku (hledání obličeje, kalibrace, ...).

Spojení mezi systémem a reálným světem V aplikaci jsem omezil odborné termíny, jednou z výjimek je například pojem „URL adresa“, který je odbornější.

Uživatelská kontrola a svoboda Veškeré akce lze zrušit. Skenování může uživatel opustit klepnutím na křížek v levém horním rohu. V dialogových oknech je tlačítko na jejich zavření.

Konzistence a standardizace Aplikace používá dva řezy dvou fontů. Hlavními barvami systému jsou šedá a modrá. Chyby jsou vykreslovány červeně – což je nejčastěji volená varianta pro tyto stavy.

Prevence chyb Primární akce jsou zobrazeny vždy v pravém rohu a jsou barevně zvýrazněny (viz obrázek 4.1). U polí limituji délku vstupu a validuji je okamžitě, jakmile uživatel začne vyplňovat jiné.



Obrázek 4.1: Vzhled tlačítek – primární (vpravo) a sekundární (vlevo) akce

Rozpoznání místo vzpomínání Ke každému stavu jsem vždy uvedl krátký popisek, který pomůže uživateli se zorientovat. Pro průchod systémem dle mého názoru stačí pouze číst zobrazené instrukce.

Flexibilní a efektivní použití Do systému jsem neimplementoval speciální klávesové zkratky. Prohlížeče nativně podporují přesun mezi vstupními poli pomocí tabulátoru.

Estetický a minimalistický design V aplikaci jsem se snažil zobrazovat jen informace relevantní k aktuálnímu stavu. Pokud vícero akcí vede do stejného stavu, snažím se je vždy oddělit na nezávislé podstránky.

Pomoc uživatelů poznat, pochopit a vzpamatovat se z chyb U vstupních polí vždy zobrazuji, co do nich uživatel musí vložit. Dle povahy dat neomezuji formát vstupu, ale validuji jej ve vhodnou chvíli – například URL adresy. Chybu zobrazuji u daného pole, viz obrázek 4.2.

Obrázek 4.2: Komunikace chyby uživateli

Nápověda a návody Do aplikace jsem nezahrnul dodatečnou nápovědu nebo návody kromě krátkých popisků ke každé sekci.

4.2.1.2 Dodatečná kritéria

V rámci analýzy jsem dále požádal participanty, aby při hodnocení vzali v potaz další kritéria, která nejsou součásti Nielsonova desatera. Definoval jsem je následovně.

Kontrast Mezi jednotlivými prvky systému by měl být dostatečný kontrast (například mezi podkladem a textu na něm). Neaktivní položky jsou zpravidla méně viditelné, vždy by ale měly být jednoznačně rozpoznatelné i v tomto stavu.

Čitelnost Nezávisle na kontrastu by všechny prvky měly mít dostatečnou velikost, aby se s nimi dalo snadno integrovat. Stejně je nutné, aby text šel bez problému přečíst ze vzdálenosti 50 cm měřené od očí uživatele k monitoru.

4.2.1.3 Výsledky

O vytvoření heuristické analýzy jsem požádal své dva kolegy, kteří se zaměřují na návrh uživatelského rozhraní. Každý nalezený problém měli ohodnotit na stupnici od 1 do 5 dle následující legendy:

- 1 – možné vylepšení vedoucí k lepší použitelnosti,
- 2 – zanedbatelná chyba neomezující práci,
- 3 – nepříjemnost způsobující horší uživatelskou zkušenost,
- 4 – problém omezující práci,
- 5 – problém způsobující nemožnost práce s aplikací.

Připomínky, návrhy řešení a podniknuté kroky zobrazují následující tabulky. Každá přísluší jednomu nalezenému problému.

Problém	Formuláře nelze odeslat pomocí klávesy „Enter“.
Návrh řešení	Implementovat reakci na danou klávesnici, simulovat kliknutí na pozitivní tlačítko.
Priorita	4
Řešení	Implementováno.

Problém	Při vložení URL adresy je vždy nutné začít vstup slovy „http://“ a „www“.
Návrh řešení	Doplňovat je v aplikační logice, pokud je uživatel nezadá.
Priorita	3
Řešení	Aplikace automaticky doplňuje prefix „https://“ v případě potřeby. Z povahy fungování webu nedoplňuje „www“.

Problém	Pro nahrání obrázku je potřeba klepnout na ikonu, klik na pole nefunguje.
Návrh řešení	Umožnit klik na pole.
Priorita	3
Řešení	Po kliknutí na pole se vykoná stejná akce jako při klepnutí na ikonu.

Problém	Pole pro vložení URL adresy neobsahují nápovědu vysvětlující, co je URL adresa.
Návrh řešení	Doplnit vysvětlení s ukázkami možných vstupů.
Priorita	3
Řešení	Doplněna do pravého rohu ikona s kontextovou nápovědou.

Problém	Z jednoho obrázku by mělo jít přejít na druhý. Navigace by neměla probíhat exkluzivně přes boční menu.
Návrh řešení	Rozšířit aplikaci o tuto funkcionalitu.
Priorita	3
Řešení	Prozatím odloženo, viz sekce 5.

Problém	Tlačítka „Pozastavit“ a „Ukončit“ se zobrazují ve všech fázích skenování.
Návrh řešení	Schovat je nebo zakázat (a graficky odlišit jejich stav).
Priorita	2
Řešení	Obě tlačítka se zobrazují pouze, když jsou relevantní vůči stavu aplikace.

4.2.2 Uživatelské testování

Po zapracování připomínek, které vyplynuly z heuristické analýzy, jsem dal aplikaci otestovat dalším šesti uživatelům. Prerekvizitou výběru byla neznalost principů návrhu uživatelského rozhraní. Samotná technika testování byla značeně proměnná – účastnili se rodinní příslušníci, kteří byli se mnou v jedné místnosti. Dále jsem testy prováděl i vzdáleně. S testerem jsem si vždy zavolal, požádal ho o nasdílení obrazovky a postupně četl jednotlivé instrukce. Před počátkem testu jsem každého účastníka požádal o vyplnění dotazníku, ve kterém jsem se ptal na věk, zkušenosti s počítači a chytrými telefony. Zároveň jsem je seznámil se základní myšlenkou systému a nutnosti povolit přístup ke kameře.

4.2.2.1 Scénáře

S uživateli jsem testoval dva různé scénáře – první uvedený se vztahuje na běžné použití systému. Popis scénáře, zejména druhý krok, je mírně vágní. Mým cílem bylo si ověřit, jestli uživatelé dokáží bez problému povolit přístup ke kameře, nakalibrovat systém a nasbírat nějaká data. První tři kroky jsem jim přečetl najednou.

- Předpokládejte, že chcete zjistit, na jaká místa se nejčastěji díváte na webu „seznam.cz“. V aplikaci tak načtěte tento web.
- Projděte celým procesem sledování pohledu.
- Jakmile budete mít pocit, že vás už nic nezajímá, ukončete proces.
- Nestahuje výsledná data.
- Nyní proveďte skenování předpřipravené sady. Její kód je „test“.
- Po dokončení stáhněte obrázek s teplotní mapou a surová data.

Druhý scénář zahrnuje základní administrační úkony.

1. Přihlaste se do systému pomocí uživatelského jména „test“ a hesla „test“.
2. Najděte všechny záznamy, jejichž jméno obsahuje text „auto“.
3. Upravte jednu z těchto položek tím, že odeberte jeden obrázek ze sady a přidáte nový. Nově přidaný obrázek zvolte jako výchozí.
4. Smažte položku, která je identifikována klíčem „test-12“.
5. Odhlaste se ze systému.

4.2.2.2 Výsledky

Během testování aplikace jsem si dělal poznámky o krocích uživatelů. Mezi nejzajímavější pokládám následující fakta.

- Uživatelé měli tendenci nečíst doplňkový text k jednotlivým úkonům, v případě potřeby spíše využívali kontextovou nápovědu u jednotlivých vstupních polí.
- Uživatelům chvíli trvalo, než se zorientovali mezi třemi tlačítky na hlavní obrazovce aplikace. Neměli problém ale zvolit korektní akci.
- Uživatelé vnímali červená tlačítka jako destruktivní nebo ukončovací akce, neměli problém najít tlačítko pro ukončení skenování.
- Kalibrační fáze uživatele většinou zarazila a chvíli jim trvalo, než si uvědomili, co je potřeba udělat. V druhém kole už neměli problém.
- Testeři neuměli přecházet mezi různými obrázky jedné skenovací sady.
- Při stahování výsledků uživatelé občas intuitivně klepli na tlačítko pro stažení výsledků a nezaškrtli, že chtějí uložit i surová data.
- Vyhledávání v administraci používali intuitivně. Nechybělo jim potvrzovací tlačítko.
- Při mazání sady dat testerům chvíli trvalo, než akci našli. Testeři pokládající se za zkušenější uživatele chytrých telefonů automaticky zkusili hledat pod ikonou tří teček.

Ve zpětné vazbě testerů se vyskytlo pár připomínek.

- Automatické zobrazování teplotní mapy je spíše rušivé. Dle testerů upoutává pozornost, a tím pádem i zkresluje nasbíraná data.
- Načtení dat při změně textu ve vyhledávacím poli bez nutnosti potvrdit změnu je příjemné.
- Tlačítko na smazání sady by mělo být viditelné na první pohled.

4.2.2.3 Zhodnocení testování

V rámci uživatelského testování bylo pro mne klíčové zjistit, jestli uživatelé aplikaci vůbec dokáží používat. Výsledky mne utvrdily, že aplikace je jako celek použitelná. V jednom bodě se shoduje heuristická analýza i uživatelské

testování – a to sice, že sady dat nejsou v aktuálním stavu moc uživatelsky přívětivé.

Potěšila mne zpětná vazba na jednoduchost používání vyhledávání a zároveň i stížnost na horší dohledatelnost funkce smazání. Když jsem systém navrhoval, tak jsem se ji snažil částečně schovat. Pro smazání musí uživatel klepnout na ikonu tří teček, a potom vybrat možnost „Smazat“. Akce je záměrně dvoukroková, protože je destruktivní a nenávratná.

Kapitola 5

Další kroky

Ať už z výsledku testování nebo samotného stavu aplikace plyne sada dalších požadavků. Ty bude vhodné zapracovat v dalších fázích vývoje.

Autorizace Aktuálně využívaný Basic access authentication sice vystačí jako základní úroveň zabezpečení. V aktuální implementaci ale nenabízí základní flexibilitu, na kterou jsou uživatele zvyklí – jako je možnost obnovit zapomenuté heslo nebo registrace. Sám bych rád v aplikaci jednou viděl autorizaci založenou na OAuth 2.0 standardu.

Nativní podpora PDF Aplikace pro vykreslení PDF nyní využívá server, který soubor převede do obrázku. V dalších fázích by z hlediska optimalizace bylo vhodné přesunout vykreslování přímo do webové aplikace.

Průchod obrázky Aktuálně je možná navigace mezi obrázky jedné sady pomocí bočního menu. Tato implementace je dle testů uživatelský nepřívětivá. V budoucnu by se hodilo přidat ke každému obrázku aktivní oblast. Po kliku do ní by se zobrazil jiný obrázek.

Uložení výsledků Do dokončení procesu skenování má uživatel možnost si stáhnout výsledek. Ten musí manuálně předat cílové straně. Jelikož už teď existuje backend, tak by se surová data mohla odesílat přímo na něj. Teplotní mapy by se z nich daly generovat dodatečně na vyžádání.

Přesnější validace Při úpravě sad dat aplikace ověřuje jen správnost URL adresy, ale nekontroluje, jestli se na druhé straně doopravdy vyskytuje ob-

rázek. S cílem omezit možné chyby by se aplikace mohla tuto funkcionalitu mohla naučit.

Rozpoznání vstupu Aplikace obsahuje tři různé sekce, kam může uživatel vložit textový vstup – URL adresu obrázku, webu nebo kód. Podle odpovědi cílového serveru by bylo možné rozpoznat, jestli se pod URL adresou nachází obrázek nebo ne. Podle toho automaticky zvolit způsob vykreslování. Tato funkcionalita by zejména zlepšila uživatelský komfort.

Závěr

Hlavním cílem této práce bylo vytvořit webovou aplikaci umožňující záznam pohybu očí po obrazovce. Motivací zadavatele byla primárně neexistence veřejně dostupného řešení, které by touto funkcionalitou disponovalo.

V úvodu práce jsem se věnoval teorii stojící za nalezení lidské tváře a očí, dále mapování získaných dat na cílovou plochu. Prozkoumal jsem podobná komerční řešení a veřejně dostupné knihovny. Následně jsem vydefinoval funkční požadavky, nefunkční požadavky a uživatelské případy. Během analýzy vyvstaly i mimo zadání další požadavky jako potřeba vykreslovat externí webové stránky nebo obrázky. Kvůli nim jsem musel nakonec naprogramovat podpůrnou serverovou aplikaci.

V návrhové části jsem prozkoumal dostupné možnosti, jak takovou aplikaci vyvinout rychleji za pomocí knihoven třetích stran a prostudoval jejich licenční podmínky. Navrhl jsem podobu API a vydefinoval standard komunikace mezi klientem a serverem. V neposlední řadě jsem se věnoval i volbě barev, aby byl v aplikaci dodržen dostatečný kontrast.

Na počátku vývoje webové aplikace jsem se snažil vytvořit vlastní algoritmus pro identifikaci místa, kam se uživatel právě dívá. Po neúspěchu jsem se rozhodl použít veřejně dostupnou knihovnu. S pomocí ní a webové kamery počítače se mi nakonec podařilo pohyb očí zaznamenávat.

Ve výsledné aplikaci si uživatelé mohou zvolit ze čtyř zdrojů dat – lokálního souboru, obrázku na internetu, předpřipravené sady obrázků a nebo zadat URL adresu webu, který se vykreslí. Výsledná data si mohou vyexportovat

do obrázku nebo strojově čitelného formátu. Zabezpečená administrační část aplikace dále obsahuje i správu těchto sad obrázků.

Nakonec jsem aplikaci připravil pro jednoduché nasazení s využitím nástroje Docker Compose. Dále jsem ji i otestoval – automatizovaně, heuristicky a s potencionálními reálnými uživateli.

První verze aplikace je dokončena a lze ji dle mého názoru využít na základní otestování jiných uživatelských rozhraní. Jelikož během vývoje a testování vznikly další požadavky, tak je potřeba se připravit na další iteraci vývoje. Doufám, že aplikace v budoucnu poslouží širší komunitě a budu se moci podílet na jejím vývoji. Zároveň jsem rád, že jsem se díky této práci naučil programovat v jazyce JavaScript a aplikoval nabyté znalosti z předmětu „Návrh uživatelských rozhraní“.

Literatura

[1] A Gentle Introduction to Computer Vision. červenec 2019, [cit. 2021-01-01]. Dostupné z: https://machinelearningmastery.com/what-is-computer-vision/

[2] Pavlickova, P.; Pavlicek, J.: Business Process Models (BPMN and DEMO Notation) - Usability Study. In *Enterprise and Organizational Modeling and Simulation*, editace R. Pergl; E. Babkin; R. Lock; P. Malyzhenkov; V. Merunka, Cham: Springer International Publishing, 2019, ISBN 978-3-030-35646-0, s. 167–174, [cit. 2021-01-03].

[3] Adakane, D.: What are Haar Features used in Face Detection? 2019, [cit. 2020-11-28]. Dostupné z: https://medium.com/analytics-vidhya/what-is-haar-features-used-in-face-detection-a7e531c8332b

[4] Wang, Y.-Q.: An Analysis of the Viola-Jones Face Detection Algorithm. *Image Processing On Line*, ročník 4, 2014: s. 128–148, doi:10.5201/ipol.2014.104, [cit. 2020-11-28]. Dostupné z: https://doi.org/10.5201/ipol.2014.104

[5] Lamé, A.: Gaze Tracking. [cit. 2020-11-28]. Dostupné z: https://github.com/antoinelame/GazeTracking

[6] Dey, S.; Samanta, D.: An Efficient Approach for Pupil Detection in Iris Images. In *15th International Conference on Advanced Computing and Communications (ADCOM 2007)*, 2007, s. 382–389, doi:10.1109/ADCOM.2007.79, [cit. 2020-11-24].

[7] Xiong, X.; Cai, Q.; Liu, Z.; aj.: Eye Gaze Tracking Using an RGBD Camera: A Comparison with an RGB Solution. In *The 4th International Workshop on Pervasive Eye Tracking and Mobile Eye-Based Interaction (PETMEI 2014)*, ACM - Association for Computing Machinery, září 2014, [cit. 2020-11-24].

[8] Vision, C.; Group, I.: eye-gaze. 2015, [cit. 2020-11-24]. Dostupné z: `https://github.com/iitmcvg/eye-gaze`

[9] Alexandra Papoutsaki, J. L. N. D. J. H. J. H., Patsorn Sangkloy: WebGazer Demo. [cit. 2020-11-25]. Dostupné z: `https://webgazer.cs.brown.edu/calibration.html?`

[10] szydej: GazeCloud, Real-Time online Eye-Tracking. [cit. 2020-11-23]. Dostupné z: `https://github.com/szydej/GazeCloud`

[11] Bryn Farnsworth, P.: Eye Tracker Prices – An Overview of 20+ Eye Trackers. [cit. 2020-11-23]. Dostupné z: `https://imotions.com/blog/eye-tracker-prices/`

[12] Deveria, A.: Can I use... Support tables for HTML5, CSS3, etc. [cit. 2020-11-29]. Dostupné z: `https://caniuse.com/stream`

[13] The GNU General Public License v3.0 - GNU Project - Free Software Foundation. [cit. 2020-11-23]. Dostupné z: `https://www.gnu.org/licenses/gpl-3.0.html`

[14] Diaz, J.: This AI Knows When Your Graphic Design Is Good (Or Bad). [cit. 2020-11-23]. Dostupné z: `https://www.fastcompany.com/90171173/eyequant`

[15] Dalmaijer, M. S. . V. d. S. S., E.S.: About - PyGaze. [cit. 2020-11-23]. Dostupné z: `http://www.pygaze.org/about`

[16] Zieliński, P.: Opengazer: open-source gaze tracker for ordinary webcams. 2008, [cit. 2020-11-23]. Dostupné z: `http://www.inference.org.uk/opengazer`

[17] opengazer: OpenGazer: GitHub. 2010, [cit. 2020-11-23]. Dostupné z: `https://github.com/opengazer/OpenGazer`

[18] Tyson, M.: What is the JRE? Introduction to the Java Runtime Environment. 2020, [cit. 2020-11-26]. Dostupné z:

https://www.infoworld.com/article/3304858/what-is-the-jre-introduction-to-the-java-runtime-environment.html

[19] Electron | Build cross-platform desktop apps with JavaScript, HTML, and CSS. [cit. 2020-11-26]. Dostupné z: https://www.electronjs.org

[20] Hossain, M.: *CORS in Action: Creating and consuming cross-origin APIs*. Manning Publications, 2014, ISBN 9781617291821.

[21] Selim, H.; Tayeb, S.; Kim, Y.; aj.: Vulnerability Analysis of Iframe Attacks on Websites. srpen 2016, s. 1–6, doi:10.1145/2955129.2955180, [cit. 2020-11-30].

[22] Ross, D.; Gondrom, T.: HTTP Header Field X-Frame-Options. RFC 7034, RFC Editor, říjen 2013.

[23] Cook, S.; Bock, C.; Rivett, P.; aj.: Unified Modeling Language (UML) Version 2.5.1. Standard, Object Management Group (OMG), prosinec 2017. Dostupné z: https://www.omg.org/spec/UML/2.5.1

[24] General Assembly. červen 2020, [cit. 2020-12-07]. Dostupné z: https://www.ecma-international.org/memento/ga.htm

[25] Uluca, D.: TypeScript vs ES6 vs ES2015. říjen 2016, [cit. 2020-12-07]. Dostupné z: https://www.excella.com/insights/typescript-vs-es6-vs-es2015

[26] Nance, J.: TypeScript vs. JavaScript: Should You Migrate Your Project to TypeScript? září 2017, [cit. 2020-12-07]. Dostupné z: https://stackify.com/typescript-vs-javascript-migrate/

[27] Syed, B. A.: *Beginning Node.js*. New York, NY: Apress, 2014, ISBN 9781484201886, 281 s., oCLC: ocn899226851, [cit. 2020-12-07].

[28] Rogers, M.: About Node.js®. leden 2020, [cit. 2020-12-07]. Dostupné z: https://nodejs.org/en/about/

[29] What is npm? [cit. 2020-12-07]. Dostupné z: https://www.w3schools.com/whatis/whatis_npm.asp

[30] Fielding, R. T.; Taylor, R. N.: *Architectural Styles and the Design of Network-Based Software Architectures*. Dizertační práce, 2000, aAI9980887, [cit. 2020-12-12].

[31] Masse, M.: *REST API design rulebook*. Sebastopol, CA: O'Reilly, 2012, ISBN 9781449319908, [cit. 2020-12-12].

[32] Hardt, D.: The OAuth 2.0 Authorization Framework. RFC 6749, RFC Editor, říjen 2012, [cit. 2020-12-04]. Dostupné z: `http://www.rfc-editor.org/rfc/rfc6749.txt`

[33] Jones, M.; Bradley, J.; Sakimura, N.: JSON Web Token (JWT). RFC 7519, RFC Editor, květen 2015, [cit. 2020-12-04]. Dostupné z: `http://www.rfc-editor.org/rfc/rfc7519.txt`

[34] Josefsson, S.: The Base16, Base32, and Base64 Data Encodings. RFC 4648, RFC Editor, říjen 2006, [cit. 2020-12-04]. Dostupné z: `http://www.rfc-editor.org/rfc/rfc4648.txt`

[35] Reschke, J.: The 'Basic' HTTP Authentication Scheme. RFC 7617, RFC Editor, září 2015, [cit. 2020-12-04]. Dostupné z: `http://www.rfc-editor.org/rfc/rfc7617.txt`

[36] Nielsen, J.: 10 Usability Heuristics for User Interface Design. listopad 2020, [cit. 2020-12-22]. Dostupné z: `https://www.nngroup.com/articles/ten-usability-heuristics/`

[37] Zaraysky, S.: The Evolution of Material Design's Text Fields. listopad 2019, [cit. 2020-12-22]. Dostupné z: `https://medium.com/google-design/the-evolution-of-material-designs-text-fields-603688b3fe03`

[38] Web Content Accessibility Guidelines 2.0, W3C World Wide Web Consortium Recommendation 2.1 July 2018. červen 2018, [cit. 2020-12-22]. Dostupné z: `https://www.w3.org/TR/2018/REC-WCAG21-20180605/`

[39] Vaughan-Nichols, S. J.: What is Docker and why is it so darn popular? březen 2018, [cit. 2020-12-08]. Dostupné z: `https://www.zdnet.com/article/what-is-docker-and-why-is-it-so-darn-popular/`

[40] Docker Inc.: Docker overview. prosinec 2020, [cit. 2020-12-08]. Dostupné z: `https://docs.docker.com/get-started/overview/`

[41] Haley, J.: Demystifying containers, Docker, and Kubernetes. červenec 2019, [cit. 2020-12-07]. Dostupné z: `https://cloudblogs.microsoft.com/opensource/2019/07/15/how-to-get-started-containers-docker-kubernetes/`

[42] Daly, L.: Git makes software development, well, easier. [cit. 2020-12-07]. Dostupné z: `https://www.atlassian.com/agile/software-development/git`

[43] Chacon, S.; Straub, B.: *Pro Git*. USA: Apress, druhé vydání, 2014, ISBN 1484200772, [cit. 2020-12-07].

Příloha A

Seznam použitých zkratek

ACID Atomicity, consistency, isolation, durability

AI Artificial intelligence

API Application Programming Interfaces

BASE Basically available, soft state, eventual consistency

BAA Basic access authentication

CSS Cascading Style Sheets

CORS Cross-Origin Resource Sharing

CPU Central processing unit

CSV Comma-separated values

ES ECMAScript

HTML Hyper Text Markup Language

I/O Input/output

JPEG Joint Photographic Experts Group

NPM Node package manager

OS Operating system

PDF Portable Document Format

PNG Portable Network Graphics

TS TypeScript

URL Uniform Resource Locator

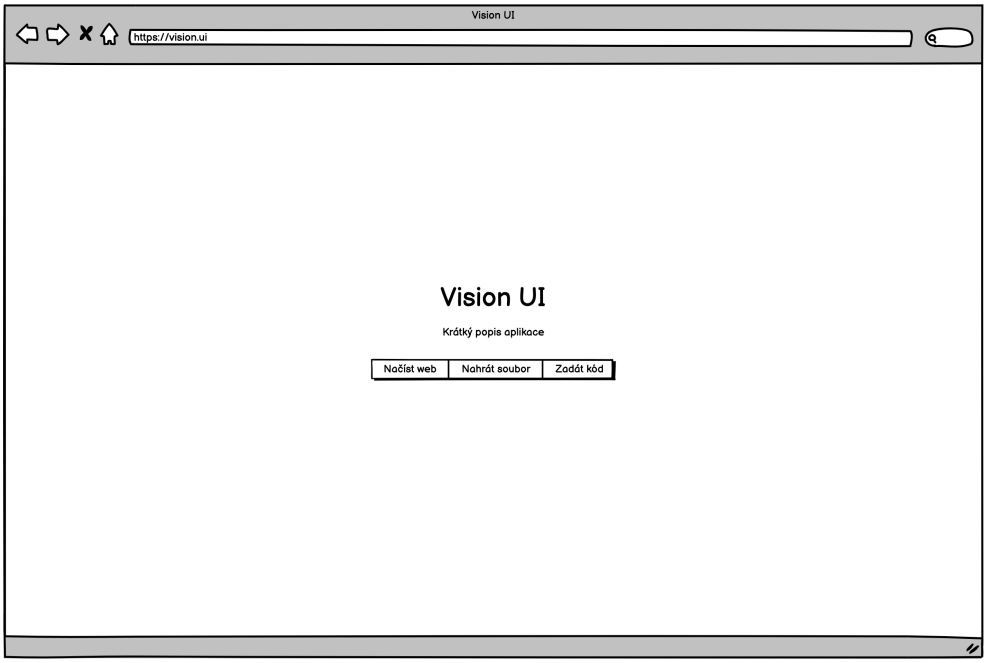
VM Virtual Machine

YAML YAML Ain't Markup Language

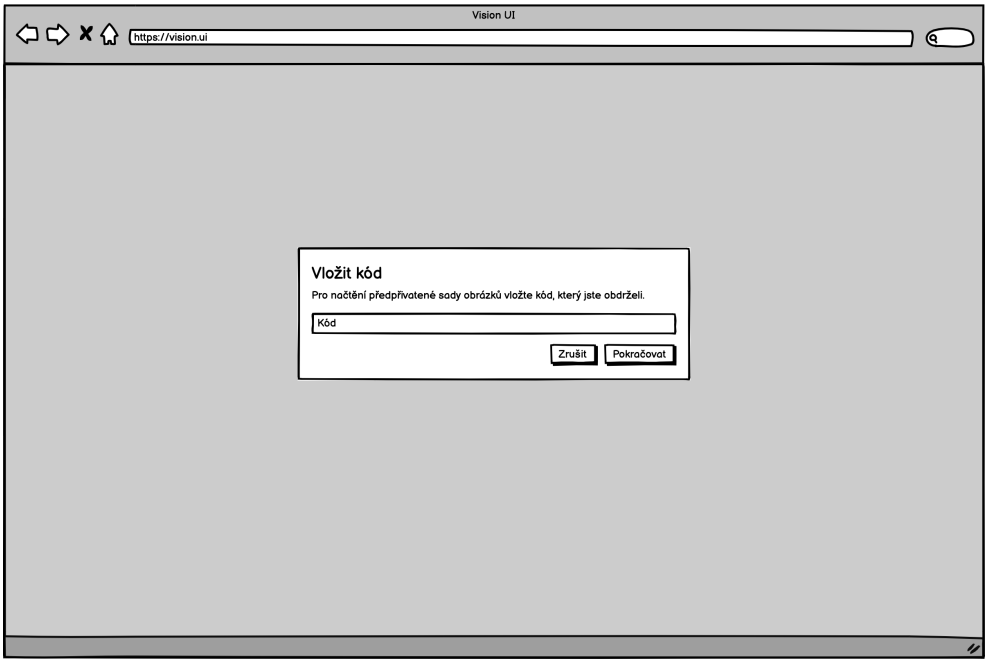
Příloha B

Drátěný model

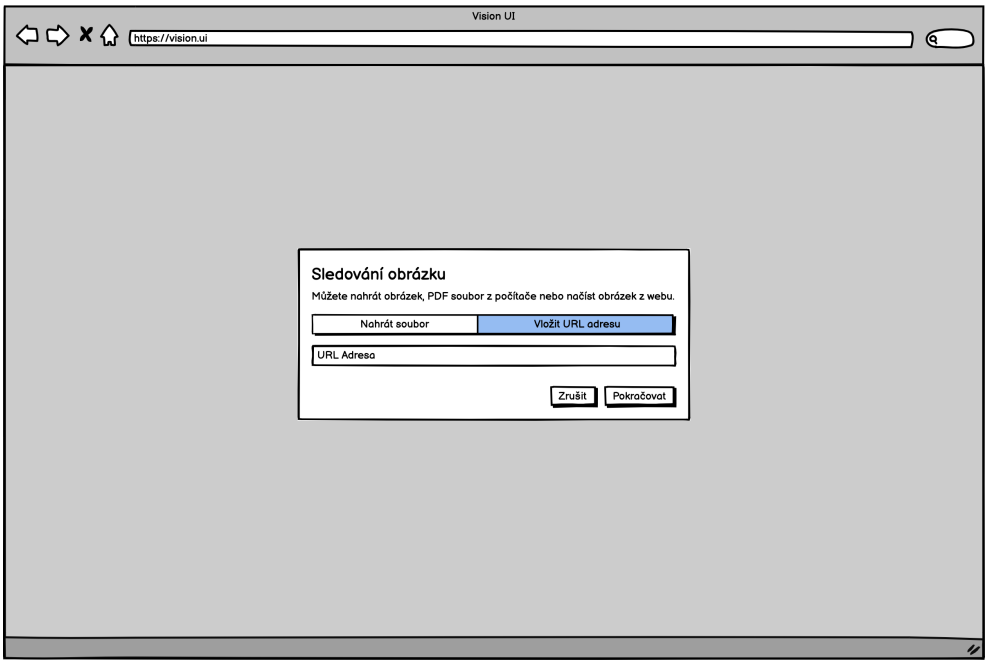
Tato příloha obsahuje veškeré drátěné modely, které jsem během analytické fáze vytvořil.



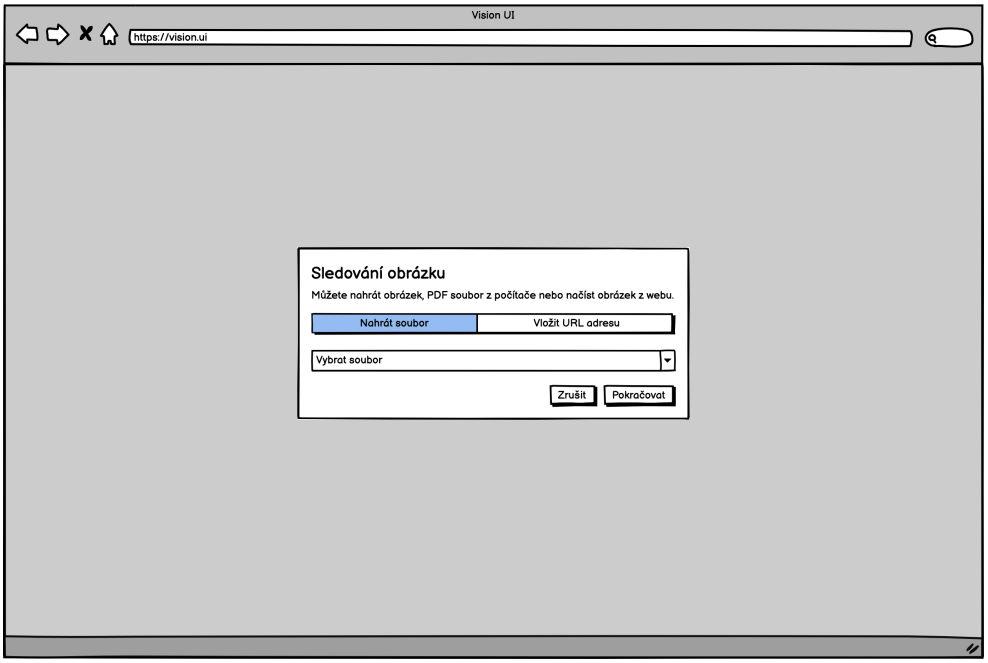
Obrázek B.1: Hlavní stránka – rozcestník



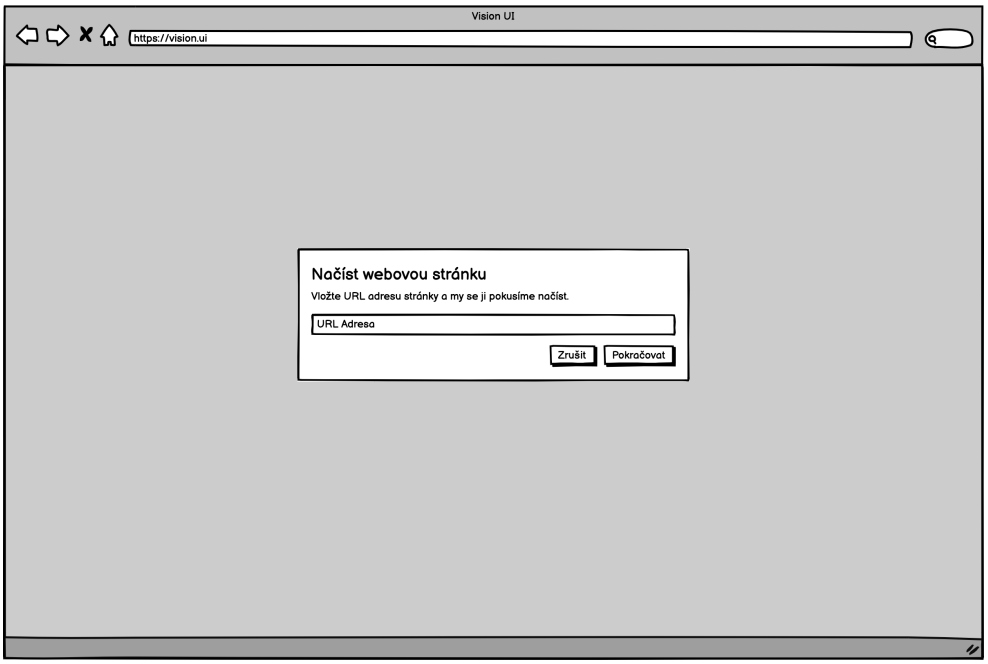
Obrázek B.2: Vložení kódu pro načtení sady dat



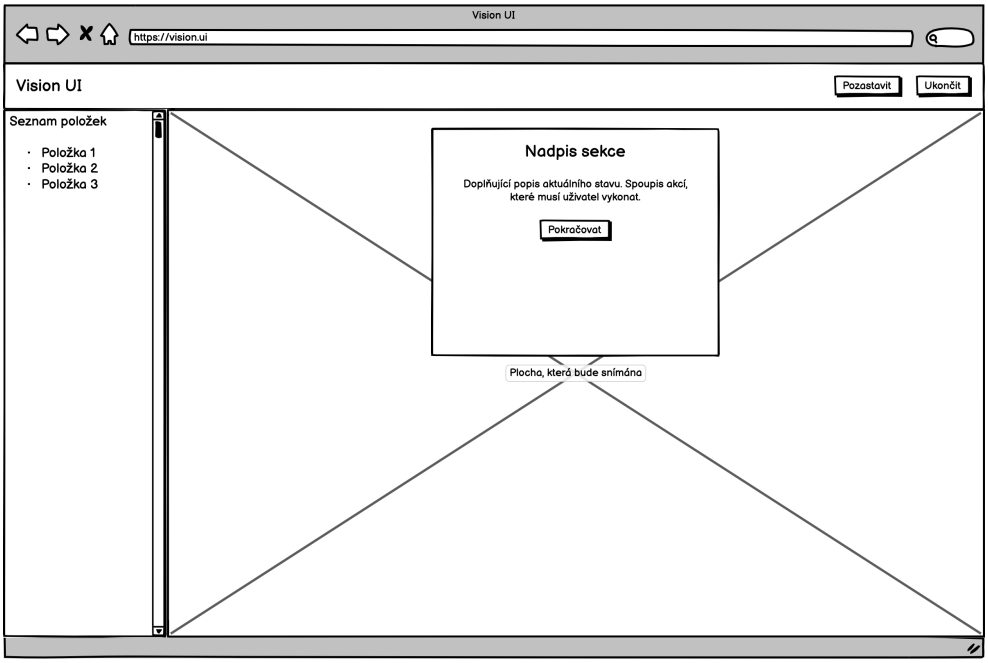
Obrázek B.3: Vložení URL adresy obrázku



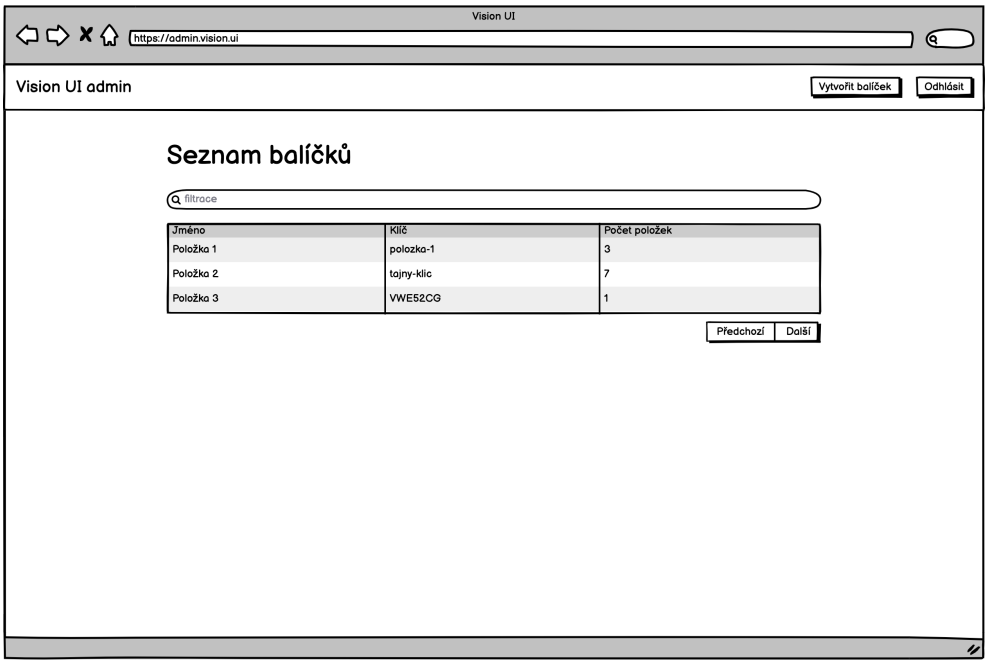
Obrázek B.4: Vložení souboru z počítače



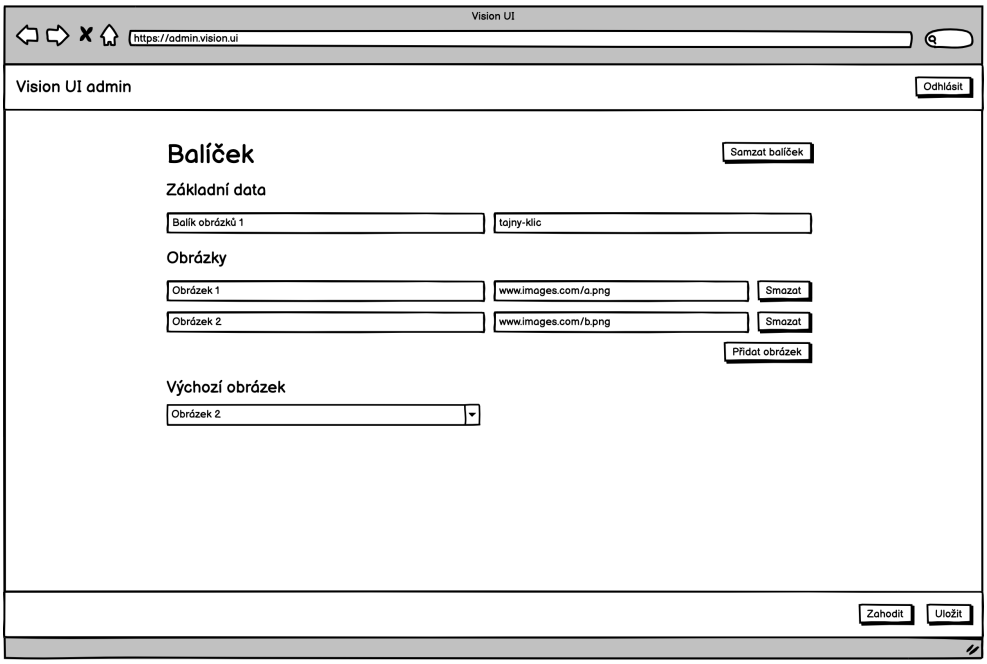
Obrázek B.5: Vložení URL adresy webové stránky



Obrázek B.6: Skenování pohledu – rozložení prvků



Obrázek B.7: Administrace aplikace – přehled sad

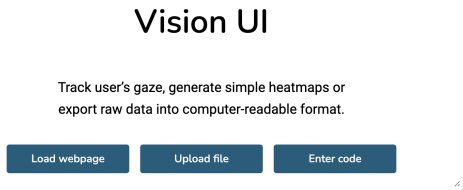


Obrázek B.8: Administrace aplikace – úprava sady

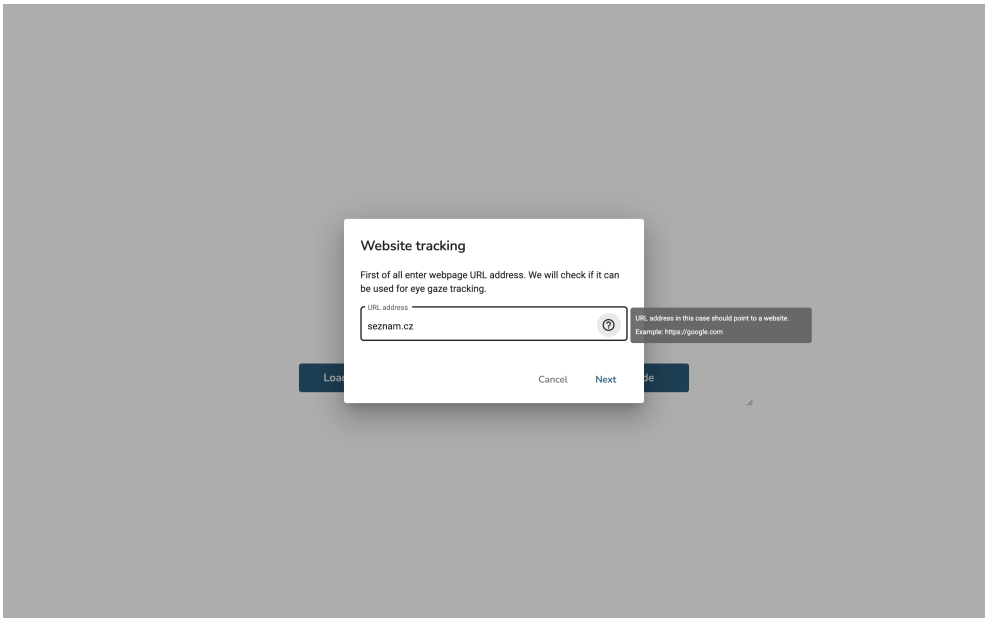
Příloha C

Výsledná aplikace

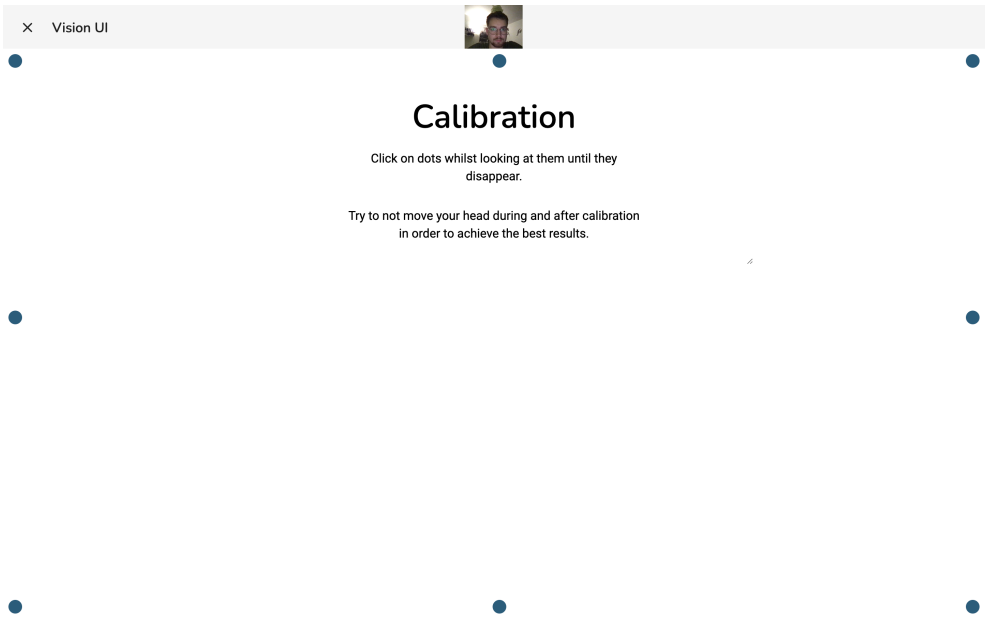
V této příloze naleznete snímky obrazovky, na kterých je výsledná aplikace.



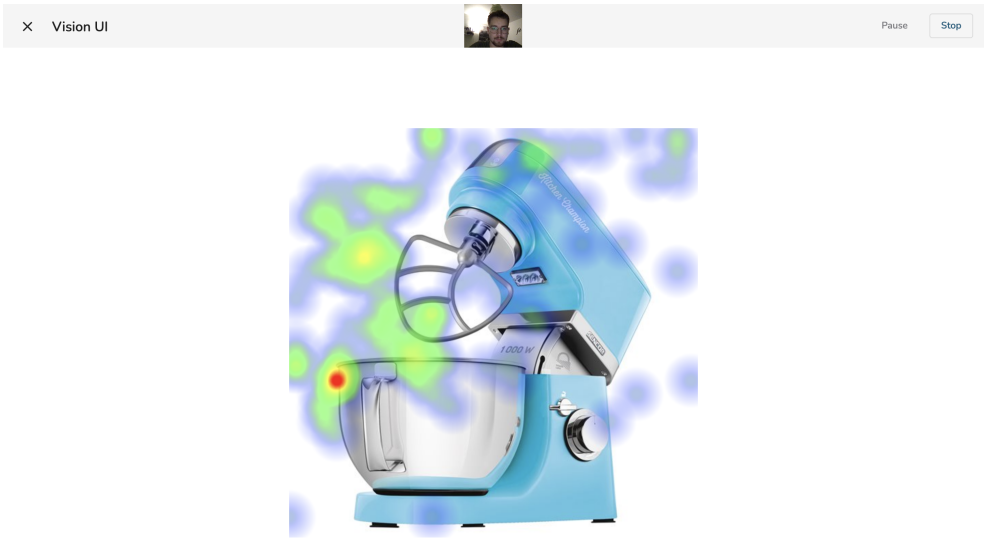
Obrázek C.1: Hlavní stránka – rozcestník



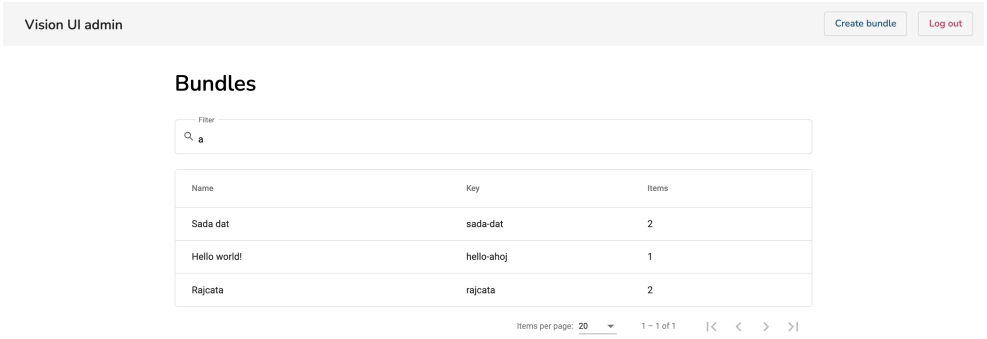
Obrázek C.2: Vložení URL adresy webu a zobrazení nápovědy



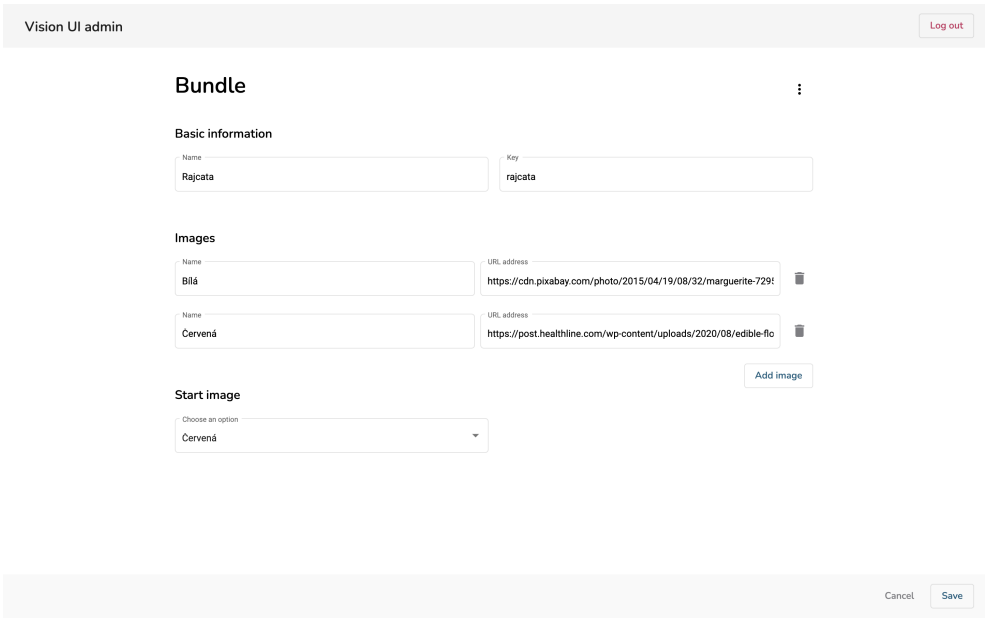
Obrázek C.3: Kalibrace systému záznamu pohybu očí



Obrázek C.4: Průběh záznamu pohybu očí a generování teplotní mapy



Obrázek C.5: Zobrazení vytvořených sad dat s aktivní filtrací



Obrázek C.6: Editace jednoho záznamu

Příloha D

Instalační příručka

Pro instalaci je vyžadován operační systém Windows, Linux nebo macOS. Dále jsou potřebné zdrojové kódy, jež jsou součástí přiloženého média k této práci nebo je lze získat z internetu. Mezi nutné nástroje pro zprovoznění serveru patří Docker, Docker Compose a Git.

Pro stažení souborů ze sítě internet je potřeba do příkazové řádky zadat příkaz: `git clone https://bitbucket.org/mroczis/vision.git`.

Vznikne tak složka `vision`, do které je potřeba přejít pomocí příkazu: `cd vision`.

Pro získání kompletních zdrojových kódů je potřeba si stáhnout submoduly kořenového repositáře. Pokud tuto akci provádíte poprvé, je potřeba zadat příkaz:
`git submodule update --init --recursive`.

V případě aktualizace již existujících souborů je nutné použít příkaz: `git submodule update --recursive --remote`.

Složka bude poté obsahovat tři adresáře. Každý reprezentuje jednu část celku:

- `vision-be` – Backend server (express-js),
- `vision-ui` – Frontend server (angular),
- `vision-db` – PostgreSQL databáze.

Všechny tři lze spusit pomocí Docker Compose. Pro spuštění je potřeba v terminálu napsat:

`docker-compose up`.

V případě potřeby lze přidat ještě přepínač `-d`, aby se proces spustil asynchronně a běžel dále i po odhlášeni uživatele z příkazové řádky. Obrazy lze znovu vytvořit přidáním přepínače `--build`.

Výchozí mapování portů ven z Docker sítě je:

- `vision-be` – 1111,
- `vision-ui` – 80,
- `vision-db` – 6768.

Při spuštění na lokální síti lze k aplikaci přistupovat na adrese `http://localhost`.

Příloha E

Obsah přiloženého média

- `readme.txt` ..... stručný popis obsahu média
- `src`
  - `impl` ..... zdrojové kódy implementace
    - `vision-be` ..... zdrojové kódy backendu
    - `vision-db` ..... zdrojové kódy databáze
    - `vision-ui` ..... zdrojové kódy frontendu
  - `impl-gaze` ..... zdrojové kódy vlastního algoritmu pro rozpoznávání
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
    - `images` ..... obrázky použité v práci
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF