VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

WEBOVÝ NÁSTROJ PRO PŘEHRÁVÁNÍ A ANOTACI BAGU

WEB TOOL FOR PLAYING AND ANOTATION OF BAG FILES

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. MARTIN OMACHT**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. MICHAL KAPINUS**
SUPERVISOR

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2020/2021

Zadání diplomové práce

Student: **Omacht Martin, Bc.**
Program: Informační technologie a umělá inteligence
Specializace: Vývoj aplikací
Název: **Webový nástroj pro přehrávání a anotaci BAGu**
Web Tool for Playing and Anotation of BAG Files
Kategorie: Uživatelská rozhraní
Zadání:

1. Seznamte se s formátem BAG, používaným v robotickém operačním systému pro ukládání multimediálních dat. Prostudujte moderní přístupy návrhu a realizace uživatelských rozhraní pomocí webových technologií.
2. Navrhněte systém umožňující nahrát a zpracovat BAG soubory s důrazem na modularitu a rozšiřitelnost.
3. Navrhněte uživatelské rozhraní přehrávače, umožňující efektivně zobrazovat 2D a 3D data uložená v BAG souboru, včetně časové synchronizace jednotlivých dat. Navrhněte potřebné datové struktury popisující zájmové části multimediálních dat a uživatelské rozhraní umožňující data anotovat.
4. Implementujte navržený systém včetně uživatelského rozhraní.
5. Naplňte systém vhodnými daty a demonstrujte funkčnost řešení. Proveďte experimenty na uživatelskou použitelnost a technickou efektivitu řešení. Komentujte výsledky experimentů a diskutujte možnosti dalšího vývoje.
6. Vytvořte plakát nebo krátké video prezentující klíčové výsledky vašeho řešení.

Literatura:

- Dle pokynu vedoucího.

Při obhajobě semestrální části projektu je požadováno:

- Body 1, 2, 3 a částečně bod 4.

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

Vedoucí práce: **Kapinus Michal, Ing.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 30. července 2021
Datum schválení: 30. října 2020

Abstrakt

Tato práce se zabývá návrhem a implementací webové aplikace pro přehrávání a anotaci multimediálních dat ze souborů ve formátu BAG. Vytvořené anotace jsou určeny pro trénování umělé inteligence do robotických systémů. Výsledná aplikace je implementována s pomocí knihovny React pro tvorbu uživatelského rozhraní a frameworku Django pro REST API na straně serveru. Celá aplikace je kontejnerizovaná pomocí nástroje Docker. V práci je popsán formát BAG, rozebrány moderní knihovny pro tvorbu webových aplikací a představeny existující řešení pro anotaci dat nebo přehrávání BAG souborů. Dále je představen návrh aplikace a popsána její implementace. V rámci práce také vznikla samostatná knihovna `rosbag_pyreader`, která slouží pro čtení BAG souborů s náhodným přístupem. Výsledná aplikace umožňuje zobrazovat a anotovat 2D a 3D data z více robotických senzorů zároveň a anotace následně exportovat ve formátu JSON.

Abstract

This work deals with the design and implementation of a web application for playback and annotation of multimedia data from files in BAG format. The created annotations are intended for training artificial intelligence in robotic systems. The resulting application is implemented with the help of React library for creating the user interface and framework Django for the server-side REST API. The entire application is containerized using Docker. This work describes the BAG format, analyzes modern libraries for creating web applications and introduces existing solutions for data annotation or playback of BAG files. Furthermore, the design of the application is presented and its implementation is described. A separate library `rosbag_pyreader`, which is used to read BAG files with random access, was created as part of this work. The resulting application allows user to display and annotate 2D and 3D data from multiple robotic sensors at the same time and then export the annotations in JSON format.

Klíčová slova

ROS, BAG soubory, anotace, React, Redux, Docker, webová aplikace, TypeScript, Python, Django, REST API

Keywords

ROS, BAG files, annotation, React, Redux, Docker, web application, TypeScript, Python, Django, REST API

Citace

OMACHT, Martin. *Webový nástroj pro přehrávání a anotaci BAGu.* Brno, 2021. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Michal Kapinus

Webový nástroj pro přehrávání a anotaci BAGu

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Michala Kapinuse. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Martin Omacht
29. července 2021

Poděkování

Děkuji vedoucímu diplomové práce Ing. Michalovi Kapinusovi za odbornou pomoc, nápady a připomínky poskytnuté během řešení této práce.

Obsah

Kapitola 1

Úvod

V dnešní době se spoustu lidí a firem zabývá umělou inteligencí, například pro autonomní vozidla, roboty a tak podobně. K řízení těchto robotických systémů často využívají systém ROS (Robot Operating System). Ten poskytuje abstrakci nad hardwarovými zařízeními, jako jsou senzory, motory a tak dále, pomocí systému zasílání zpráv. Celý systém tak spolu komunikuje zprávami na tak zvaných tématech (angl. *topics*). Obsahem zpráv můžou být příkazy, kalibrační nebo diagnostická data, ale hlavně i data ze senzorů, jako jsou například kamery nebo LiDARy. Tok těchto zpráv lze nahrávat do souborů ve formátu BAG a následně přehrávat.

Při učení umělé inteligence se nejčastěji využívá tak zvaného učení s učitelem, kdy algoritmus pro učení má k dispozici trénovací datovou sadu s již označenými správnými výstupy. Značení těchto výstupů se nazývá anotace. V případě, že vstupem jsou například obrazová data, anotace pak může být například obdélníkem vyznačená osoba v daném obrázku. Pro naučení umělé inteligence je většinou potřeba velké množství anotovaných dat, které zpravidla musí vytvářet člověk. Používají se k tomu data z těch samých senzorů, které pak využívá umělá inteligence. V případě, že se bude používat systém ROS, je vhodné anotovat data právě z BAG souborů. Zatím však neexistují žádné nástroje, které by anotaci BAG souborů podporovaly. Dosavadní způsob anotace spočívá v exportování jednotlivých dat z BAG souboru do samostatných souborů, které jsou následně anotovány každý zvlášť v běžných anotačních nástrojích pro anotaci obrazových a 3D dat. Nástroj, jenž by umožňoval anotovat více zdrojů dat z jednoho BAG souboru zároveň, by tak ulehčil práci, jelikož tyto data spolu většinou souvisí (např. kamera snímá stejnou oblast, jako LiDAR). Uživatel tak může z kamerového záznamu vidět, jaké data se nachází v 3D datech, ve kterých většinou není zrovna jednoduché se vyznat, protože se jedná o pouhé mračno bodů v prostoru.

Návrh a implementace takového nástroje je cílem této diplomové práce. Nástroj byl implementován jako webová aplikace s použitím moderní knihovny React, pro vytvoření uživatelského rozhraní, a frameworku Django na straně serveru, se kterým klientská aplikace komunikuje pomocí REST API. Nástroj umožňuje zobrazení více zdrojů dat najednou a poskytuje nástroje pro jejich anotaci. Důraz byl kladen na rozšiřitelnost a modularitu, aby bylo v budoucnu možné aplikaci rozšířit o zobrazování dalších typů dat a nových druhů anotací.

V kapitole 2 si přiblížíme formát BAG a systém ROS. Poté si do detailu popíšeme strukturu tohoto formátu. Jelikož nástroj je implementován jako webová aplikace, popíšeme si v kapitole 3 moderní přístupy pro jejich vytváření. V rámci ní si také představíme JavaScriptové knihovny React, Angular a Vue.js. Následuje kapitola o analýze požadavků na nástroj a o jeho návrhu. Zde si probereme jaké jsou na nástroj kladeny požadavky, po-

díváme se na již existující řešení, kterými se můžeme inspirovat a následně si představíme návrh. V kapitole 5 pak bude popsána implementace klíčových částí projektu. Na závěr, v kapitole 6, je popsáno, jak probíhalo uživatelské testování a jeho výsledky.

Kapitola 2

Formát BAG

Formát BAG je souborový formát používaný v systému ROS (Robot Operating System) pro ukládání zpráv [4]. Tyto soubory jsou typicky vytvářeny nástrojem `rosbag`, který naslouchá jednomu nebo více tzv. tématům (angl. *topics*) a zprávy, jenž takto přijímá, ukládá serializované do souboru BAG [4]. Tyto zprávy mohou pocházet například ze senzorů robotického systému a jejich obsahem mohou být obrazová data, data z LiDARu, kalibrační data, informace o pohybu robota a tak dále. Tyto data mohou být objemná a bývá jich velké množství, což znamená, že velikost BAG souborů se typicky pohybuje v řádu jednotek až desítek GB, pro delší záznamy to mohou být i stovky GB[1].

Systém ROS je využíván robotických systémech, jako jsou například autonomní vozidla, robotická ramena a tak podobně. Tyto systémy často využívají různé senzory k mapování okolního prostředí, k detekci překážek nebo objektů atd. K tomu se často využívá strojové učení, jenž pro učení většinou vyžaduje anotované data. V případě použití ROS systému, jsou data ukládaná právě do BAG souborů. Prozatím však neexistuje žádný nástroj, který by umožňoval anotovat přímo BAG soubory, kromě velice jednoduchých nástrojů s velmi omezenou sadou funkcí pro anotaci (blíže popsané v sekci 4.2). Z toho důvodu se k anotaci využívají anotační nástroje, které pracují se soubory v běžných formátech (JPEG, MP4, PCD...). Před anotací se proto musí data z BAG souboru nejdříve exportovat do těchto formátů a anotovat zvlášť. Běžné anotační nástroje většinou také neposkytují synchronní zobrazení více různých dat, které právě BAG soubor poskytuje. Nástroj, jenž by umožňoval anotovat více zdrojů dat z jednoho BAG souboru zároveň, by tak ulehčil práci, jelikož tyto data spolu většinou souvisí (např. kamera snímá stejnou oblast, jako LiDAR). Uživatel tak může z kamerového záznamu vidět, jaké data se nachází v 3D datech, ve kterých většinou není zrovna jednoduché se vyznat, protože se jedná o pouhé mračno bodů v prostoru.

2.1 Robot Operating System

Než si rozebereme strukturu formátu BAG, přiblížíme si více systém ROS, co poskytuje, jak funguje a jak tento formát využívá. Z názvu Robotic Operating System může vyplývat, že se jedná o samostatný operační systém, jako je například Linux nebo Windows, což ale není pravda. Jedná se o open source framework, který poskytuje sadu nástrojů, knihoven a konvencí pro zjednodušení vytváření komplexního a robustního robotického chování napříč širokým spektrem robotických platforem [11]. Operačním systémem je to nazýváno, protože

[1] Toto lze vypozorovat např. zde https://marvhub.com/#/collection/bags

poskytuje abstrakci nad hardwarovými zařízeními (senzory, motory atp.) pomocí systému zasílání zpráv.

Základní komponenty ROS systému tvoří infrastruktura pro komunikaci, knihovny a nástroje specifické pro robotiku a silné vývojářské nástroje. Pro komunikaci poskytuje na nejnižší úrovni rozhraní pro posílání zpráv, které umožňuje mezi-procesovou komunikaci (tzv. *middleware*)[12]. Pro zasílání zpráv nabízí anonymní mechanizmus zveřejni/odebírej (angl. *publish/subscribe*) [12]. Díky tohoto mechanizmu lze data jednoduše zachytit, uložit do BAG souboru a následně je později znovu zveřejnit z tohoto souboru. Pro robotiku ROS poskytuje například standardní definice zpráv pro nejčastější případy užití, knihovnu `tf`[2] zaznamenávající pozice veškerých komponent robotického systému, jazyk `URDF` pro popis fyzických vlastností robota nebo standardizovaný způsob produkování, sbírání a agregaci diagnostických dat o robotovi [12]. Vývoj robotického systému ulehčí více než 45 nástrojů s rozhraním příkazové řádky. Pro vizualizaci pak poslouží nástroj `rviz`, jenž umožňuje 3D vizualizaci dat z mnoha senzorů i samotného robota popsaného již zmíněným jazykem `URDF` [12]. Dále také framework `rqt` založený na knihovně `Qt`, jenž umožňuje vytvářet vlastní grafická rozhraní pro vašeho robota [12].

ROS podporuje několik programovacích jazyků: C++, Python, Octave a LISP. Aby toho docílil, používá jednoduchý jazyk pro definici rozhraní IDL (*Interface Definition Language*) pro popis zpráv posílaných mezi moduly [16]. IDL používá krátké textové soubory pro popis jednotlivých položek zprávy a umožňuje kompozici zpráv [16]. Příklad definice zprávy lze vidět v 2.3.

Systém ROS lze také integrovat s jinými frameworky a knihovnami. Dle [13] mezi ně patří:

- Gazebo - simulátor robotů v komplexních vnitřních i venkovních prostředích [14].
- OpenCV - knihovna algoritmů pro počítačové vidění a strojové učení [15].
- PCL - knihovna pro zpracování n-dimenzionálních mračen bodů a 3D geometrie [18].
- MoveIt - framework pro plánování pohybu robota [20].
- ROS-industrial - open source projekt rozšiřující ROS do automatizace a robotiky výrobního průmyslu [17].

2.2 Struktura formátu BAG

V této sekci si podrobně popíšeme formát BAG. Pokud není uvedeno jinak, informace v této sekci vychází z dokumentace systému ROS [9]. Jak již bylo zmíněno, tento formát slouží pro uchování ROS zpráv. Soubory tohoto formátu mají příponu `.bag`. Existuje několik verzí: 1.0, 1.1, 1.2 a 2.0. Formát verze 2.0 však vyšel již s ROS verzí 1.1.5, která vyšla koncem roku 2010 [10] a od té doby se formát nezměnil. Předchozí verze se tudíž nepoužívaly dlouhou dobu, takže podpora těchto starých verzí v anotačním nástroji nemá příliš velký význam.

Oproti předchozím verzím přináší verze 2.0 uložení zpráv po kouscích (angl. *chunks*), které lze individuálně komprimovat a přitom zachovávají možnost náhodného přístupu ke zprávám. Dále přidává index vyšší úrovně, který umožňuje nástrojům rychle sesbírat statistiky o souboru. V neposlední řadě je formát také kompaktnější.

[2] http://wiki.ros.org/tf

Základní struktura formátu

Základní struktura formátu je velice jednoduchá. Jako první se v souboru nachází řetězec ukončený znakem nového řádku. Tento řetězec označuje verzi BAG formátu, kterou tento soubor používá. Pro verzi 2.0 je tento řetězec `#ROSBAG V2.0`. Za znakem konce řádku se pak nachází jednotlivé záznamy v binárním formátu. Výpis 2.1 tuto strukturu znázorňuje. Veškeré hodnoty formátu jsou uložené ve formátu *little-endian*.

```
#ROSBAG V2.0
<zaznam 1><zaznam 2>...<zaznam N>
```

Výpis 2.1: Základní struktura BAG formátu verze 2.0

Záznamy

Jak bylo naznačeno výše, soubory BAG se skládají ze záznamů. Těch je několik typů, které budou popsány níže, avšak všechny mají stejnou základní strukturu. Jak je znázorněno ve výpisu 2.2, záznam se skládá z délky hlavičky, což je celé číslo o délce 4 bajty, následuje hlavička o této délce. Za hlavičkou následují data, jejichž délka je definována stejným způsobem.

```
<delka hlavicky><hlavicka><delka dat><data>
```

Výpis 2.2: Základní struktura záznamu BAG souboru

Hlavička záznamu obsahuje položky s názvem a hodnotou oddělenými znakem '='. Každá položka začíná celým číslem o velikosti 4 bajty, jenž určuje délku této položky v bajtech. Za tímto číslem se nachází název položky. Ten může obsahovat jakýkoliv tisknutelný ASCII znak, kromě znaku '=', kterým je ukončen. Za tímto znakem se nachází hodnota položky, jejíž formát se liší podle významu dané položky. Délka položky je dána velikostí názvu, hodnoty i znaku '=' v bajtech. Položky se v hlavičce mohou nacházet v libovolném pořadí.

Každý záznam má povinnou položku s názvem „op“, jejíž hodnota je 1 bajt bez znaménka. Tento bajt určuje typ záznamu. Jaké hodnoty korespondují s jakými typy záznamů lze vidět v tabulce 2.1.

Hodnota	Typ záznamu
0x02	Message data
0x03	Bag header
0x04	Index data
0x05	Chunk
0x06	Chunk info
0x07	Connection

Tabulka 2.1: Tabulka hodnot položky „op“ (hexadecimálně)

Bag header

Záznam typu *Bag header* se vyskytuje v souboru pouze jednou a to jako první záznam. Garantované položky hlavičky lze vidět v tabulce 2.2. Obsahuje základní informace o souboru,

jako jsou počty záznamů typu *Chunk* a počet unikátních spojení (v souboru se může nacházet více stejných záznamů typu *Connection*). Položka `index_pos` je užitečná v případě, že ze souboru nepotřebujeme číst data, ale pouze metadata, která se typicky nachází až za všemi *Chunk* záznamy. Hodnota této položky udává pozici v bajtech od začátku souboru, kde se nachází první záznam za záznamy typu *Chunk*. Říkejme části souboru označované touto položkou „indexová“. V datové sekci záznamu se nachází pouze výplň v podobě ASCII znaků mezery takové velikosti, aby celý záznam měl velikost 4096 bajtů. Tato výplň slouží k tomu, aby se později daly do hlavičky přidat další informace.

Název	Popis	Formát	Délka
`index_pos`	Pozice prvního záznamu po záznamech typu *Chunk*	`long integer`	8 bajtů
`conn_count`	Počet unikátních spojení v souboru	`integer`	4 bajty
`chunk_count`	Počet záznamů typu *Chunk* v souboru	`integer`	4 bajty

Tabulka 2.2: Tabulka garantovaných položek v hlavičce záznamu *Bag header*

Chunk

Záznamy *Chunk* se nachází hned za prvním záznamem *Bag header*. Tyto záznamy shlukují zprávy do větších celků, které lze komprimovat. Garantované položky v hlavičce *Chunk* záznamu jsou uvedené v tabulce 2.3. Položka `compression` nabývá hodnot „none“ značící žádnou kompresi, „bz2“ značící kompresi typu Bzip2 a „lz4“ značící kompresi LZ4. V datové sekci záznamu se nachází záznamy typu *Connection* a *Message data* komprimované touto metodou.

Název	Popis	Formát	Délka
`compression`	Typ komprese dat	`string`	proměnná
`size`	Velikost záznamu bez komprese v bajtech	`integer`	4 bajty

Tabulka 2.3: Tabulka garantovaných položek v hlavičce záznamu *Chunk*

Connection

Tyto záznamy se mohou nacházet jak v datové sekci záznamu *Chunk*, tak i samostatně v indexové části souboru (viz podsekce Bag header výše). V indexové části slouží k tomu, aby se dalo jednoduše zjistit, jaké typy zpráv a jaké témata se v souboru nacházejí, aniž by se musely tyto informace vyhledávat v záznamech *Chunk*, což by zahrnovalo i čtení potenciálně velkého množství dat. Záznamy *Connection* uchovávají informace o komunikačním spojení, ze kterého se zprávy získávaly. Každá zpráva pochází z jednoho z uvedených spojení a každá zpráva z jednoho spojení má stejný formát.

Garantované položky v hlavičce záznamu *Connection* uvádí tabulka 2.4. Jelikož se záznamy o spojení mohou v souboru vyskytnout duplicitně, každé spojení má své unikátní ID uvedené v položce `conn`. Položka `topic` uvádí, na kterém téma jsou zprávy *uloženy* (zprávy mohou být publikovány na jiném téma než jsou pak uloženy v BAG souboru).

Název	Popis	Formát	Délka
`conn`	Unikátní ID spojení	`integer`	4 bajty
`topic`	Téma, na kterém jsou zprávy uloženy	`string`	proměnná

Tabulka 2.4: Tabulka garantovaných položek v hlavičce záznamu *Connection*

V datové sekci záznamu se nachází hlavička spojení, která je ve stejném formátu jako hlavička záznamu. Povinně obsahuje následující položky typu `string`:

- `topic` - téma, na které se odběratel připojil,
- `md5sum` - kontrolní součet md5 typu zprávy,
- `type` - typ zprávy,
- `message_definition` - plný text definice zprávy včetně všech podtypů (v jazyce IDL) [3].

Nepovinně pak obsahují položky `callerid` obsahující název uzlu, jenž data odesílá, a `latching` označující jestli je publikující v módu *latching*, což znamená, že novým odběratelům zašle poslední publikovanou zprávu [3].

Message data

Záznamy typu *Message data* se nachází pouze v datové části záznamu *Chunk* a obsahují data jednotlivých přijatých zpráv. Hlavička obsahuje položky definované v tabulce 2.5. V datové části záznamu se nachází serializovaná zpráva. Způsob serializace je blíže popsaný v sekci 2.3.

Název	Popis	Formát	Délka
`conn`	ID spojení, ze kterého zpráva pochází	`integer`	4 bajty
`time`	Čas přijmutí zprávy	`long integer`	8 bajtů

Tabulka 2.5: Tabulka garantovaných položek v hlavičce záznamu *Message data*

Index data

Pokud je soubor BAG indexovaný, za každým *Chunk* záznamem se nachází záznam typu *Index data*, pro každé spojení jenž se nachází v tomto *Chunk* záznamu. Záznam *Index data* obsahuje pozice jednotlivých záznamů typu *Message data* v dekomprimované datové části.

V hlavičce záznamu se nachází položky definované v tabulce 2.6. Položka `ver` označuje verzi tohoto záznamu. Aktuální verze je 1. Položka `conn` určuje ID spojení, ze kterého byly přijaty zprávy indexované tímto záznamem, a položka `count` udává jejich počet.

V záznamu s verzí 1 obsahuje datová část po sobě jdoucí dvojice o délce 12 bajtů, kde prvních 8 bajtů udává čas, kdy byla zpráva přijata a zbylé 4 bajty označují pozici v bajtech relativní vůči začátku datové části předcházejícího *Chunk* záznamu, na které se nachází záznam *Message data* pro tuto zprávu. Počet těchto dvojic je udán v položce hlavičky s názvem `count`.

Název	Popis	Formát	Délka
`ver`	Verze záznamu Index data	`integer`	4 bajty
`conn`	ID spojení	`integer`	4 bajty
`count`	Počet zpráv, jenž přišlo ze spojení `conn` v předcházejícím záznamu *Chunk*	`integer`	4 bajty

Tabulka 2.6: Tabulka garantovaných položek v hlavičce záznamu *Index data*

Chunk info

Záznamy typu *Chunk info* se typicky nacházejí na samotném konci souboru v jeho indexové části (viz podsekce Bag header výše). Těchto záznamů by mělo být v souboru stejný počet jako záznamů typu *Chunk*. Jedná se o index vyššího řádu, kde každý záznam obsahuje pozici jednoho z *Chunk* záznamů a další souhrnné informace o datech v něm.

V hlavičce se nachází položky, jenž jsou v tabulce 2.7. Položka `ver` udává verzi záznamu. I zde je aktuální verze 1. Pro tuto verzi se v datové části nachází po sobě jdoucí dvojice o délce 8 bajtů, kde první 4 bajty udávají ID spojení a druhé 4 bajty počet zpráv, jenž bylo přijato na tomto spojení v daném *Chunk* záznamu. Počet těchto dvojic udává položka hlavičky s názvem `count`.

Název	Popis	Formát	Délka
`ver`	Verze záznamu *Chunk info*	`integer`	4 bajty
`chunk_pos`	Pozice záznamu *Chunk* v souboru	`long integer`	8 bajtů
`start_time`	Čas přijmutí nejstarší zprávy v záznamu *Chunk*	`long integer`	8 bajtů
`end_time`	Čas přijmutí nejnovější zprávy v záznamu *Chunk*	`long integer`	8 bajtů
`count`	Počet spojení v *Chunk* záznamu	`integer`	4 bajty

Tabulka 2.7: Tabulka garantovaných položek v hlavičce záznamu *Chunk info*

2.3 Definice zpráv

Informace v této sekci vycházejí z dokumentace systému ROS [2]. Jak již bylo zmíněno v sekci 2.1, pro komunikaci se v systému ROS využívají zprávy. Formát těchto zpráv je definován pomocí jednoduchého jazyka IDL (Interface Definition Language). Z definice zprávy v tomto jazyce lze jednoduše vygenerovat zdrojový kód pro serializaci a deserializaci zpráv tohoto typu v různých cílových programovacích jazycích (např. C++ nebo Python). Díky tohoto jednoduchého jazyka mohou ROS balíčky definovat vlastní typy zpráv. Definice zpráv jsou uloženy v souborech s příponou „`.msg`“ ve složce `msg` ROS balíčku. Název takto definovaného typu zprávy se skládá z názvu balíčku (např. `std_msgs`), lomítka a názvu souboru s definicí zprávy bez přípony „`.msg`“ (např. `std_msgs/String`). Tento název můžeme využít v jiné definici zprávy jako datový typ jedné z položek zprávy. Pokud se definice zprávy nachází ve stejném balíčku, můžeme se na ni odkazovat pouze názvem souboru bez přípony. Jedinou výjimkou tohoto pravidla, je speciální typ zprávy `Header` z balíčku `std_msgs` (více

Název	Způsob serializace
`bool`	unsigned 8-bit int
`int8`	signed 8-bit int
`uint8`	unsigned 8-bit int
`int16`	signed 16-bit int
`uint16`	unsigned 16-bit int
`int32`	signed 32-bit int
`uint32`	unsigned 32-bit int
`int64`	signed 64-bit int
`uint64`	unsigned 64-bit int
`float32`	32-bit IEEE float
`float64`	64-bit IEEE float
`string`	ASCII string
`time`	secs/nsecs unsigned 32-bit ints
`duration`	secs/nsecs signed 32-bit ints

Tabulka 2.8: Vestavěné typy položek a jejich způsob serializace

o tomto typu v sekci 2.4), u nějž, pokud se nachází jako první položka v definici zprávy, se také nemusí uvádět název balíčku.

Definice zprávy se skládá ze dvou částí: konstanty a položky. Konstanty slouží například k definování užitečných hodnot pro interpretaci některé z položek. Položky pak definují formát dat posílaných ve zprávě.

Jazyk IDL je velice jednoduchý. Jedná se o seznam konstant a položek oddělenými znakem nového řádku. Definice položky se skládá z typu a názvu. Typem může být jeden z vestavěných typů (viz tabulka 2.8) nebo jiná definice zprávy (např. `geometry_msgs/Pose`). Položky mohou být i pole fixní nebo variabilní délky. V takovém případě jsou za typem položky uvedeny hranaté závorky a pokud je mezi nimi uvedeno číslo, jedná se o pole fixní délky a toto číslo udává jeho délku. Pole variabilní délky jsou při serializaci prefixována 32bitovým celým číslem bez znaménka, jenž udává jeho délku. Stejným způsobem je serializován i typ `string`.

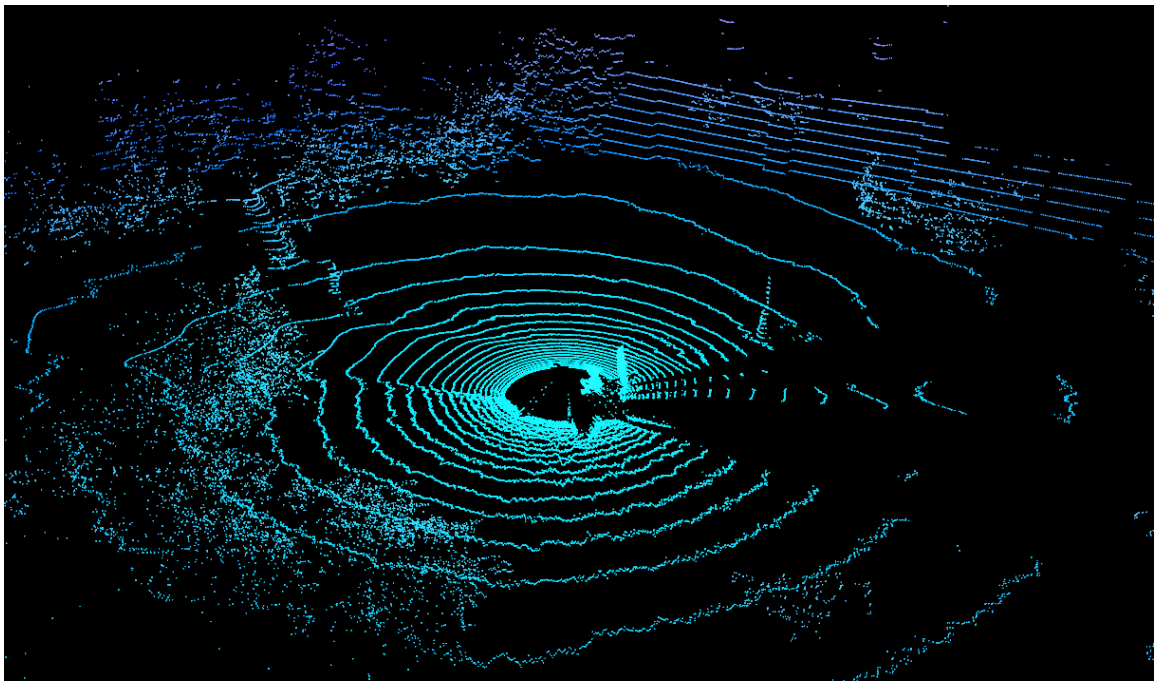
```
# Priklad definice zpravy
# Komentare jsou uvozeny znakem '#'
# Ukazka konstant
int32 KONST=-98
string RETEZEC=Retezec muze obsahovat i~mezery

# Polozky zpravy
Header header # Specialni hlavicka obsahujici, mimo jine, casovou znacku
float64[9] matrix # Pole fixni delky
uint32[] elevations # Pole variabilni delky
sensor_msgs/Image image
sensor_msgs/Temperature[] temperatures
```

Výpis 2.3: Ukázka definice zprávy

Konstanty jsou definovány stejným způsobem jako položky, avšak za názvem konstanty je znak „=“ a za tímto znakem se nachází hodnota konstanty. Typ konstanty však může být pouze jeden z vestavěných typů (viz tabulka 2.8), kromě typů `time` a `duration`. Celočíselné hodnoty konstant mohou být uvedené pouze v dekadické soustavě. Ukázku definice zprávy lze vidět ve výpisu 2.3.

2.4 Typy zpráv

Systém ROS definuje řadu zpráv pro běžné použití[3]. Tyto zprávy jsou kategorizované do balíčků. Jedná se například o zprávy týkající se geometrie z balíčku `geometry_msgs`, kde se nachází definice zpráv pro zasílání informací o pozicích různých částí robota, jejich akcelerace, setrvačnost a podobně, včetně základních geometrických útvarů a datových struktur, ze kterých se tyto zprávy skládají (polygon, bod, vektor, matice, kvaternion, atd.). Pro použití v této diplomové práci jsou nejzajímavější zprávy v balíčku `sensor_msgs`, jenž definuje zprávy pro běžně používané senzory. Patří mezi ně například *CompressedImage*, *Image*, *LaserScan*, *PointCloud2* a další.



Obrázek 2.1: Vizualizace mračna bodů. 3D skener je umístěn uprostřed a jednotlivé body představují jednotlivá měření laserem.

Zprávy typu *Image* obsahují výšku a šířku obrázku, použité kódování jednotlivých pixelů (`yuv422`, `bgr8`, `rgb8` atd.), příznak určující, zda jsou bajty v pořadí *little-endian* nebo *big-endian*, počet bajtů jednoho řádku pixelů a jako poslední je pole bajtů s daty jednotlivých pixelů. *CompressedImage* obsahuje pouze formát (jpeg, png nebo tiff) a pole bajtů komprimovaných obrazových dat.

LaserScan zprávy obsahují data z laserových skenerů, které snímají pouze v jedné rovině. Zpráva obsahuje pole vzdáleností naměřených tímto skenerem v určité kruhové výseči, která

[3] https://github.com/ros2/common_interfaces

je dána položkami `angle_min` a `angle_max`. Zprávy typu *MultiEchoLaserScan* jsou velmi podobné, akorát v jednom měření může být naměřeno více vzdáleností (skener vyšle svazek paprsků a v případě, že narazí na hranu, se každý odrazí z jiné vzdálenosti).

PointCloud2 slouží pro mračna bodů, které jsou většinou vytvářeny pomocí 3D skenerů, 3D kamer nebo stereo kamer. Na obrázku 2.1 lze vidět, jak tyto data vypadají po vykreslení v 3D prostoru. Zpráva obsahuje pole bajtů s jednotlivými body i informace o tom, z jakých položek se každý bod skládá. Nemusí se skládat pouze ze souřadnic, ale také mohou obsahovat intenzitu, normály a tak dále.

Kapitola 3

Moderní technologie pro tvorbu webových aplikací

Pro frontend webových stránek se standardně využívají jazyky HTML5, CSS3 a JavaScript (ECMAScript), jenž jsou podporovány všemi moderními prohlížeči. Tyto jazyky v základu možná stačí pro relativně jednoduché webové stránky (ale i pro ty se dnes často využívají preprocesory jako je např. SASS[1]), avšak pro komplexní a dynamické webové aplikace se dnes využívají moderní knihovny pro jazyk JavaScript umožňující tvořit uživatelská rozhraní deklarativně a rozdělovat je na uzavřené, znovupoužitelné komponenty. Mezi nejpopulárnější patří React, Angular a Vue.js.

U webových uživatelských rozhraní je však problém s používáním moderních technologií. Nejnovější verze webových prohlížečů sice podporují nejnovějších verze jazyků a standardů, ale spousta uživatelů má zastaralé verze prohlížečů nebo dokonce již nepodporované prohlížeče (Internet Explorer). Z tohoto důvodu nelze přímo využívat nejnovější verze jazyků pro tvorbu webových aplikací.

Pokud chceme v JavaScriptu využít funkci z novější verze, stačí využít tzv. *polyfill* této funkce. To znamená, že jestliže prohlížeč danou funkci nepodporuje, nahradí se vlastní implementací. V případě, že chceme využít syntaxi z novější verze jazyka, potřebujeme již náš zdrojový kód přeložit do nižší verze jazyka. K tomu slouží nástroj Babel[2].

Další z nástrojů pro moderní vývoj webových aplikací je Webpack[3]. Jedná se o balíčkovací nástroj, který umožňuje například sloučit více JavaScriptových souborů do jednoho. Díky tomu můžeme náš kód rozdělit do více souborů pro lepší přehlednost a klientovi odesílat tyto soubory zabalené do jednoho souboru, což je optimálnější. V případě, že výsledný balíček začne být příliš veliký, je možné ho rozdělit na více částí (tzv. *chunks*), které si klient vyžádá, až budou potřeba. Dále také umožňuje výsledný kód minimalizovat, spustit vývojářský server, jenž automaticky při změně zdrojového souboru aktualizuje výsledek v prohlížeči, a spoustu dalších funkcí, jenž jsou dodávány pomocí pluginů.

3.1 React

React je knihovna vyvíjena převážně společností Facebook, který ji používá ve svých produktech (Facebook, Instagram atd.) [5]. Zaměřuje se hlavně na vytváření uživatelských

[1] https://sass-lang.com/
[2] https://babeljs.io/
[3] https://webpack.js.org/

rozhraní a neposkytuje například prostředky pro směrování, získávání dat z API, dělení projektu na moduly a další. Tyto funkce jsou dodávány knihovnami.

Komponenty

Základním stavebním kamenem nejen v Reactu, ale i v ostatních podobných knihovnách, jsou komponenty. Komponenty jsou znovupoužitelné a nezávislé části uživatelského rozhraní. Můžeme nad nimi uvažovat jako o JavaScriptových funkcích, které přijímají libovolný vstup (tzv. *props*) a vrací React elementy popisující co se má zobrazit na obrazovce [7]. React elementem může být buď HTML element nebo jiná React komponenta. Komponenty tedy tvoří stromovou strukturu, stejně jako HTML elementy v DOM[4].

V Reactu můžou být komponenty definovány buď jako třídy, anebo funkce (tzv. funkcionální komponenty). Ve starších verzích Reactu, se funkcionální komponenty používaly pouze pro jednoduché části uživatelského rozhraní, jenž si nepotřebovaly uchovávat vnitřní stav. Od verze 16.8 se však zavedly takzvané *hooks*, které přidávají možnost uchovávání stavu ve funkcionálních komponentách a zároveň také způsob jak velice dobře sdílet funkcionalitu vyžadující stav mezi komponentami vytvářením vlastních *hook* funkcí.

Hlavním pravidlem Reactu je, že komponenty se musí chovat jako čistá funkce (anglicky *pure function*) vzhledem ke svým vstupům (*props*) [7]. To znamená, že je nesmí měnit. Pokud by na vstupu byl například objekt a komponenta by upravila jeho vlastnost, propsala by se tato změna i do nadřazené komponenty, která jí tento vstup předala. V takovém případě, už by se komponenta nechovala jako čistá funkce, ale měla by vedlejší účinky, což by vedlo k velké nepředvídatelnosti.

Uživatelské rozhraní je však velice dynamické a proto by nám komponenty pouze se vstupy nestačily [7]. Z tohoto důvodu můžou mít komponenty ještě stav (*state*). Stav může ovlivňovat pouze danou komponentu nebo její potomky [8]. Díky tomu je mezi komponentami pouze jednosměrný tok dat, a to shora dolů [8] (neboli od kořene k listovým komponentám). Když se tedy změní stav v jedné z komponent, stačí překreslit pouze podstrom daný touto komponentou. Abychom nemuseli manuálně určovat, které komponenty se mají překreslit, React toto detekuje sám. Musí se však dodržet jedno pravidlo: stav se nesmí modifikovat přímo [8]. Místo toho musíme část stavu, kterou chceme upravit, zkopírovat a až následně změnit. Poté se nový stav nastaví pomocí funkce `setState()`. Toto je jednou z nevýhod Reactu, jelikož složitější stav se zanořenými objekty může být komplikované a nepřehledné měnit, kvůli nutnosti kopírování.

JSX

React také přišel s rozšířením JavaScript syntaxe nazvaným JSX. Toto rozšíření umožňuje využívat syntaxi podobnou XML přímo v JavaScriptu. Toho se využívá při definování výstupu komponent. Můžeme totiž popisovat vzhled komponent stejně, jako kdybychom psali HTML, ale přímo v jazyce JavaScript. Využití syntaxe JSX ve funkcionální React komponentách můžete vidět ve výpisu 3.1. Jak lze v ukázce vidět, můžeme v JSX výrazech využívat i jakékoliv JavaScriptové výrazy uzavřením do složených závorek. V komponentě `DisplayName` z ukázky vypisujeme atribut `name`, který je komponentě předaný jako atribut komponentou `Container`.

[4] Document Obejct Model

Podpůrné knihovny

Jelikož React se zaměřuje pouze na reaktivní vykreslování uživatelského rozhraní, funkce jako směrování, správu globálního stavu a tak dále, je potřeba doplnit knihovnami. Mezi nejpopulárnější knihovny, jenž se využívají spolu s Reactem, patří:

- *React Router* — knihovna přidávající podporu pro URL směrování, vhodná pro vytváření SPA[5].
- *Redux* — knihovna pro centralizované uchovávání a správu vnitřního stavu aplikace.
- *MobX* — alternativa ke knihovně Redux využívající vzoru *observer*.

```
function DisplayName(props) {
    return (
        <div>
            <p>{props.name}</p>
        </div>
    );
}

function Container(props) {
    return (
        <div>
            <DisplayName name="George" />
        </div>
    );
}
```

Výpis 3.1: Použití syntaxe JSX v React kompnentách

3.2 Angular

Další z knihoven pro tvorbu moderních webových aplikací je Angular. Tato knihovna vychází z knihovny AngularJS, která se dnes již nevyvíjí. Za vývojem této knihovny stojí společnost Google [5]. Pro vývoj v Angularu se většinou používá jazyk TypeScript, což je jazyk, který JavaScript obohacuje o typovou kontrolu. Není to však podmínkou a lze používat i klasicky JavaScript nebo jazyk Dart, avšak většina dokumentace a návodů je v jazyce TypeScript.

Významným rozdílem oproti Reactu je, že Angular již v základu obsahuje všechny potřebné součásti pro vývoj dynamických webových aplikací. Například `RouterModule` nabízí směrování URL, `HttpClientModule` obsahuje potřebné třídy a funkce pro stahování dat z API, balíček `@angular/localize` zase přidává možnost multijazyčného obsahu. Také je k dispozici nástroj Angular CLI, jenž poskytuje řadu užitečných příkazů, kterými lze generovat nový projekt nebo například nové moduly, komponenty, služby a tak dále.

Komponenty v Angularu se většinou nachází v samostatných složkách, jelikož se skládají z několika souborů. Tyto soubory mají stejné jméno (např. `header.component`) a liší se

[5] Single Page Application

pouze příponou. Soubor s příponou `.ts` obsahuje třídu definující danou komponentu a její logiku. Soubor s příponou `.html` obsahuje HTML šablonu komponenty se speciální přidanou syntaxí pro zobrazení dat z třídy této komponenty. Dále má komponenta soubor s CSS styly (lze použít i preprocesory SASS, LESS nebo Stylus) s příponou dle použitého jazyku. A v neposlední řadě může obsahovat ještě soubor s příponou `.spec.ts`, který obsahuje jednotkové testy pro danou komponentu.

3.3 Vue.js

Poslední knihovnu, kterou si zde popíšeme je Vue.js. Jedná se o nejmladší knihovnu ze zde uvedených, ale rychle se stává jednou z nejpopulárnějších [5]. Vue.js se snaží odstranit největší neduhy knihoven React a Angular a z obou knihoven se snaží převzít to nejlepší. Stejně jako React se Vue.js zaměřuje pouze na základní funkce pro tvorbu uživatelských rozhraní a ostatní funkce jako směrování a globální správa stavu je nechána na knihovnách, které si každý může doinstalovat podle potřeby.

Z Angularu knihovna přebírá například rozdělení komponent na logiku, šablonu a styly. Avšak místo rozdělování těchto částí do zvláštních souborů, jsou všechny tyto části v jednom souboru. Ukázková komponenta ve Vue.js je k dispozici ve výpisu 3.2. Syntaxe šablon je taky nápadně podobná té z knihovny Angular.

```
<template>
    <p>Hello {{ name }}!</p>
</template>

<script>
export default {
    data() {
        return {
            name: 'World'
        }
    }
}
</script>

<style scoped>
p {
    color: red;
    font-size: 30px;
}
</style>
```

Výpis 3.2: Ukázková komponenta ve Vue.js

Kapitola 4

Analýza požadavků a návrh

V této kapitole se zabývá analýzou požadavků na aplikaci a návrhem. Nejdříve jsou rozebrány požadavky a existující řešení, kterými se lze inspirovat při návrhu. Dále je popsán výběr architektury aplikace a technologií pro její implementaci. Také je navrhnut způsob zpracování BAG souboru po nahrání na server, ER diagram a uživatelské rozhraní.

4.1 Požadavky na aplikaci

Cílem této práce je vytvořit webový nástroj pro přehrávání a anotaci multimediálních dat uložených ve formátu BAG. Nástroje pro přehrávání BAG souborů již existují (např. RVIZ nebo Webviz), ale neumožňují tyto soubory anotovat pro využití ve strojovém učení. Toto by měl být hlavní přínos této diplomové práce.

Další z požadavků je modularita nástroje. Jelikož existuje velké množství robotických senzorů a každý z nich může mít vlastní definice ROS zpráv, nemůže nástroj jednoduše podporovat zobrazení všech těchto typů zpráv. Je tedy důležité připravit systém takovým způsobem, aby bylo možné jednoduše rozšířit možnosti nástroje o nové typy senzorů nebo anotací.

Jednou z výhod anotace BAG souboru je synchronní přehrávání několika typů zpráv najednou, jenž poskytne širší kontext pro anotaci. Jednotlivé senzory také mohou být kalibrované a lze tedy i transformovat anotace mezi např. 3D LiDARem a kamerou. Uživatel by v takovém případě nemusel anotovat více typů dat zvlášť. Synchronní přehrávání a transformace mezi kalibrovanými senzory by tedy byly užitečnými funkcemi nástroje.

Jednotlivé zprávy z různých spojení však nemusí být v BAG souboru synchronizovány (záleží, jestli je synchronizace nastavena při nahrávání zpráv). Pro přehrávání a anotaci je však vhodné mít zprávy synchronizované. Proto by nástroj mohl nabízet synchronizaci souboru po jeho nahrání a nebo provádět synchronizaci až při čtení dat.

Co se týče anotace, nástroj by měl podporovat hlavně anotaci 2D a 3D dat. Obrazová data z kamer a 3D bodová mračna generované LiDARy, 3D kamerami typu Kinect, stereo kamerami atp., patří k asi k nejdůležitějším typům dat pro anotaci. Tyto typy senzorů jsou hojně využívány v autonomních vozidlech, kde se často používá také právě strojové učení.

Jelikož se budou anotovat převážně souvislá data, měl by nástroj umožňovat pohodlnou anotaci pohybujících se objektů. Toho lze docílit například interpolací anotací mezi klíčovými snímky nebo automatickým sledováním objektu (tzv. *object tracking*).

Každý pro anotaci také používá jiné štítky (angl. *labels*), které mohou mít různé parametry a hodnoty. Proto nástroj nemůže poskytovat pevně danou sadu štítků, ale musí umožňovat definici vlastních štítků, pokud možno co nejflexibilnější.

Aby byly vytvořené anotace užitečné, musí nástroj umožnit jejich export do souboru, jenž půjde následně jednoduše využít pro strojové učení. Vhodné by bylo zahrnout i import anotací, v případě přenášení anotací mezi systémy nebo automatickým vygenerováním anotací pomocí nějakého externího nástroje.

Anotace dat bývá často také velmi pracná záležitost a často se na ní podílí více lidí. Z tohoto důvodu, by bylo dobré, kdyby nástroj podporoval správu a kolaboraci uživatelů. Tudíž i správu rolí a oprávnění jednotlivých uživatelů, případně jejich rozdělení do týmu, či přiřazení práce. Data nahraná do systému by mohlo být vhodné taky řadit do projektů a tím mezi sebou sdílet různá nastavení.

4.2 Existující řešení

Kvalitní nástroj, který by umožňoval anotovat BAG soubory, momentálně neexistuje. Jediné dohledatelné nástroje, jenž by pracovaly přímo s BAG soubory, jsou `rosbag_annotator`[1] a `qml-rosbag-annotator`[2]. Jedná se o velmi jednoduché nástroje poskytující pouze omezené možnosti anotování. První z uvedených umožňuje anotovat obrázky pouze jednoduchým označením aktuální časové značky jedním z předdefinovaných ID (0-4) a tyto anotace následně uloží do výsledného textového souboru, který obsahuje dvojice časových značek a přiřazených ID. Druhý nástroj poskytuje i grafické uživatelské rozhraní, avšak anotace snímku probíhá pouhým přiřazením hodnoty v jednom z předdefinovaných datových typů (Bool, Int, Double, String, IntArray a DoubleArray). Anotace následně umožňuje uložit přímo do stejného nebo do vlastního BAG souboru.

Jelikož se nejedná o plnohodnotné anotační nástroje, inspiraci je vhodnější čerpat z klasických anotačních nástrojů, které pracují s multimédii v rozšířenějších formátech než je BAG. Takovýchto nástrojů existuje velké množství, a proto si zde uvedeme jenom několik nejzajímavějších.

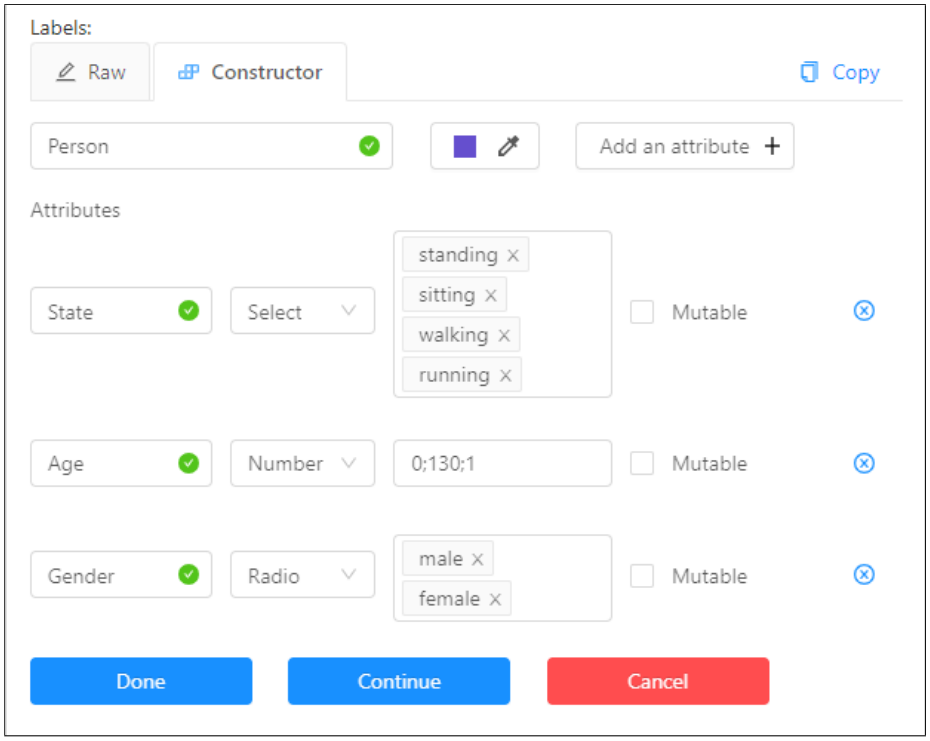
Computer Vision Annotation Tool (CVAT)

CVAT je open source anotační nástroj vyvinut vývojáři z firmy Intel v rámci sady nástrojů OpenVINO. Tento webový nástroj je dostupný v omezené podobě na doméně `https://cvat.org` nebo jej lze používat lokálně pomocí nástroje Docker. Nástroj využívá moderní technologie v podobě Rest API vytvořeného ve frameworku Django a fronendu vytvořeném v knihovně React.

CVAT podporuje anotaci obrázků a videí. Nástroj poskytuje několik geometrických tvarů pro anotaci objektů v obrázku či snímku videa: obdélník, polygon, lomená čára, množina bodů a kvádr. Každý tento tvar je přiřazen k určitému štítku (angl. *label*) a lze mu nastavit různé vlastnosti (např. jestli je mimo záběr nebo jestli je překryt jiným objektem). U videa lze tyto tvary interpolovat mezi klíčovými snímky. Pro anotaci lze také využít předem připravené modely umělé inteligence, které umí automaticky detekovat určité objekty nebo sledovat pohyb objektu ve videu. Takto vytvořené anotace pak stačí pouze ručně zkontrolovat a doladit.

[1] `https://github.com/dsgou/rosbag_annotator`
[2] `https://github.com/chili-epfl/qml-rosbag-annotator`

Štítky lze definovat vlastní pro každý projekt. Štítku lze nastavit název, barvu a atributy. Atribut se pak skládá z názvu, typu vstupního prvku (Text, Checkbox, Select, Radio, Number), definice přípustných hodnot (závisí na typu vstupního prvku) a označení jestli je atribut modifikovatelný mezi snímky nebo zůstává konstantní. Snímek tohoto konstruktoru štítků lze vidět na obrázku 4.1. Tento způsob zaručuje vysokou flexibilitu.



Obrázek 4.1: Snímek konstruktoru štítku z nástroje CVAT. Na obrázku lze vidět vytvořený štítek „Person“ s fialovou barvou, který má tři atributy: *State*, *Age*, *Gender*. Každému atributu lze nastavit typ vstupního pole (*Select*, *Number*, *Radio*, *Text*, *Checkbox*) a podle toho pak obor hodnot. Při zaškrtnutí položky „Mutable“ bude možné hodnotu atributu měnit mezi jednotlivými snímky.

Anotace lze dělit do projektů a ty lze dále dělit na úkoly (angl. *tasks*). Každý úkol se pak dá přiřadit jednomu uživateli, který bude anotovat danou část dat. Úkoly lze vytvářet i samostatně mimo projekt. Takovému úkolu je možno definovat vlastní štítky, stejně jako pro projekt. V projektu však nelze omezit typy anotací a vždy jsou k dispozici všechny tvary.

Pro import a export anotací, nástroj poskytuje mimo svůj vlastní formát i řadu dalších formátů, jenž používají jiné anotační nástroje nebo nástroje pro strojové učení. Patří mezi ně PASCAL VOC, Datumaro (pouze export), YOLO, MS COCO Object Detection, TFrecord, MOT a LabelMe 3.0.

V neposlední řadě tento nástroj také umožňuje sbírat data o anotování jednotlivých úloh (např. kolik času daný uživatel strávil anotací určitého úkolu) a tyto data pak zobrazuje pomocí nástroje Kibana a Elasticsearch.

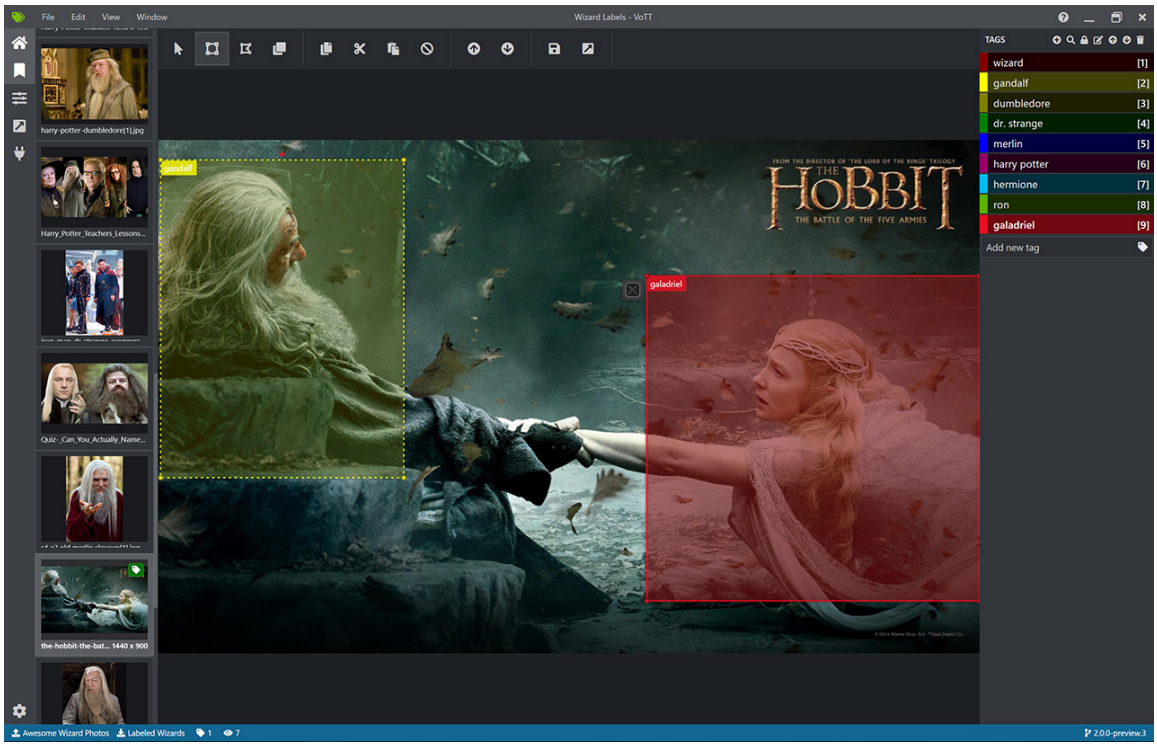
Tento nástroj byl vyvinut pro a používá ho profesionální anotační tým, tudíž se jedná o praxí otestovaný nástroj a myslím si, že je to velmi dobrý zdroj pro inspiraci.

Visual Object Tagging Tool (VoTT)

Jedná se o open source nástroj od firmy Microsoft pro anotování obrázků a videí, vytvořený v jazyce TypeScript s knihovnami React a Redux. Lze jej spustit jak ve webovém prohlížeči, tak i jako desktop aplikaci pomocí frameworku Electron. Tento nástroj však nepoužívá architekturu klient-server jako CVAT, ale jedná se pouze o klienta, který pracuje s lokálními soubory nebo s cloudovým uložištěm (momentálně podporuje pouze Azure Blob Storage[3] a Bing Image Search[4]). S lokálními soubory navíc může pracovat pouze pokud je spuštěn jako desktopová aplikace, jelikož prohlížeč neumožňuje Javascriptovým aplikacím přistupovat k lokálnímu souborovému systému z bezpečnostních důvodů.

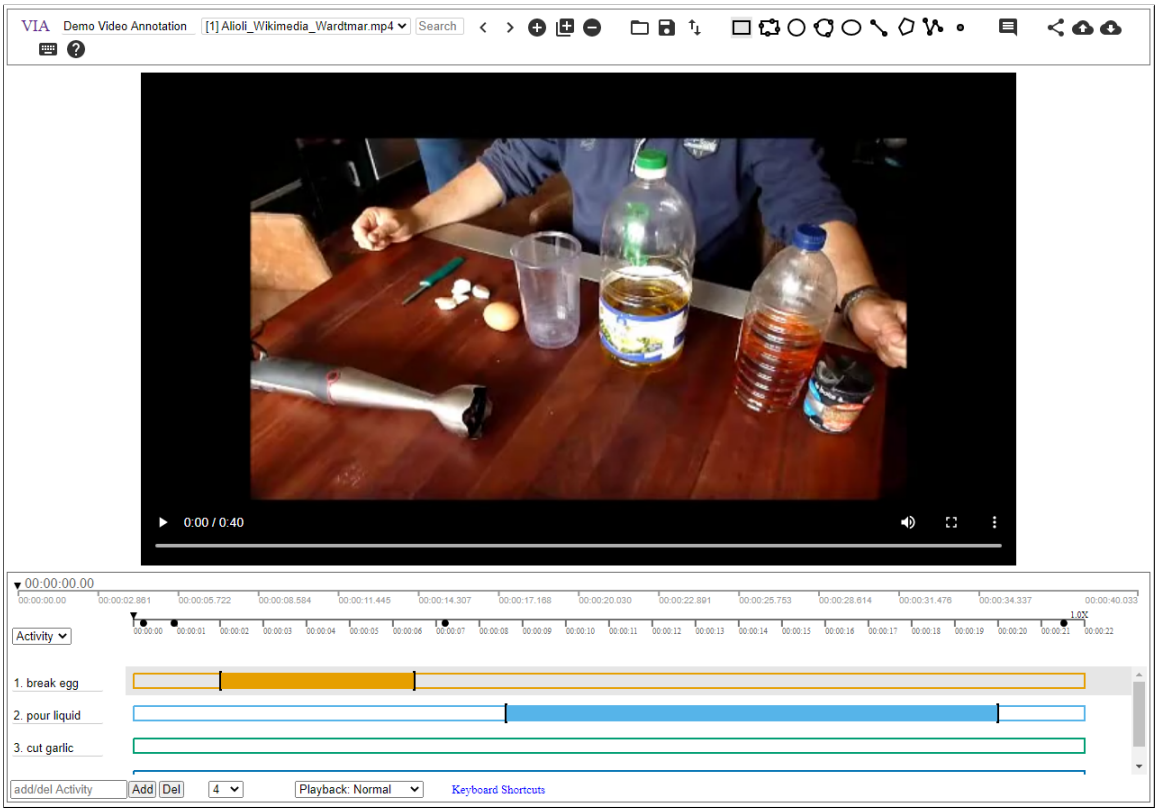
Anotace dat začíná vytvořením projektu, kterému se, mimo jiné, nastaví zdrojové a cílové uložiště dat (lokální nebo cloudové), kolik snímků za sekundu se bude ze zdrojových videí extrahovat pro anotaci a seznam štítků pro anotování dat (pouze barva a název). Každý projekt má také svůj bezpečnostní token, který se používá k zašifrování citlivých hodnot nastavení projektu (např. API klíč).

Nástroj pro anotování obrázků a snímků videa poskytuje pouze dvě možnosti: obdélník a polygon. K těmto tvarům se přiřazují štítky definované v projektu. U videí tyto tvary nelze ani interpolovat mezi klíčovými snímky, jako u nástroje CVAT, a je nutné je pro každý snímek vytvářet znovu nebo zkopírovat z jiného snímku. Snímek anotačního prostředí lze vidět na obrázku 4.2.



Obrázek 4.2: Snímek z anotačního prostředí nástroje VoTT. Zdroj: https://github.com/microsoft/VoTT.

[3] https://docs.microsoft.com/en-us/azure/storage/blobs/storage-blobs-introduction
[4] https://www.microsoft.com/en-us/bing/apis/bing-image-search-api



Obrázek 4.3: Snímek anotace segmentů videa pomocí nástroje VIA. Ve spodní části, pod přehrávačem videa, lze vidět časovou osu, kde lze anotovat aktivity ve videu. Vyplněné části barevných časových os označují, kde ve videu se anotace nachází. Ve vrchním panelu se pak nacházejí mimo jiné i nástroje pro anotaci objektů ve videu.

VoTT také nabízí automatickou anotaci obrázků pomocí natrénovaného modelu. Model se spouští buď na vyžádání nebo automaticky při zobrazení obrázku. Nástroj nabízí pouze jeden natrénovaný model, lze však přidat i vlastní modely.

Anotace lze exportovat ve formátu VoTT (soubory ve formátu JSON), CSV, Azure Custom Vision Service, Microsoft Cognitive Toolkit (CNTK), Pascal VOC a TFRecords. Avšak importovat anotace není možné. Při exportu se dá zvolit, jestli exportovat pouze anotované data, navštívené data, nebo všechny data.

VGG Image Annotator (VIA)

Další z anotačních nástrojů, které stojí za zmínku, je open source nástroj VIA vyvinutý na Oxfordské univerzitě skupinou Visual Geometry Group[5] (VGG). Tento nástroj je vytvořen v čistém HTML, Javascriptu a CSS, bez externích knihoven a je distribuován jako samostatný HTML soubor, který lze zobrazit ve většině moderních prohlížečích.

VIA podporuje anotaci obrázků, videí, zvuku i titulků. My se zde však zaměříme pouze na obrázky a videa. Pro anotaci obrázků, či snímků videa, je k dispozici obdélník, kružnice, elipsa, čára, polygon, lomená čára a bod. Kromě anotace regionů ve snímcích videa však tento nástroj umožňuje anotovat i segmenty videa, které označují např. aktivitu, kterou

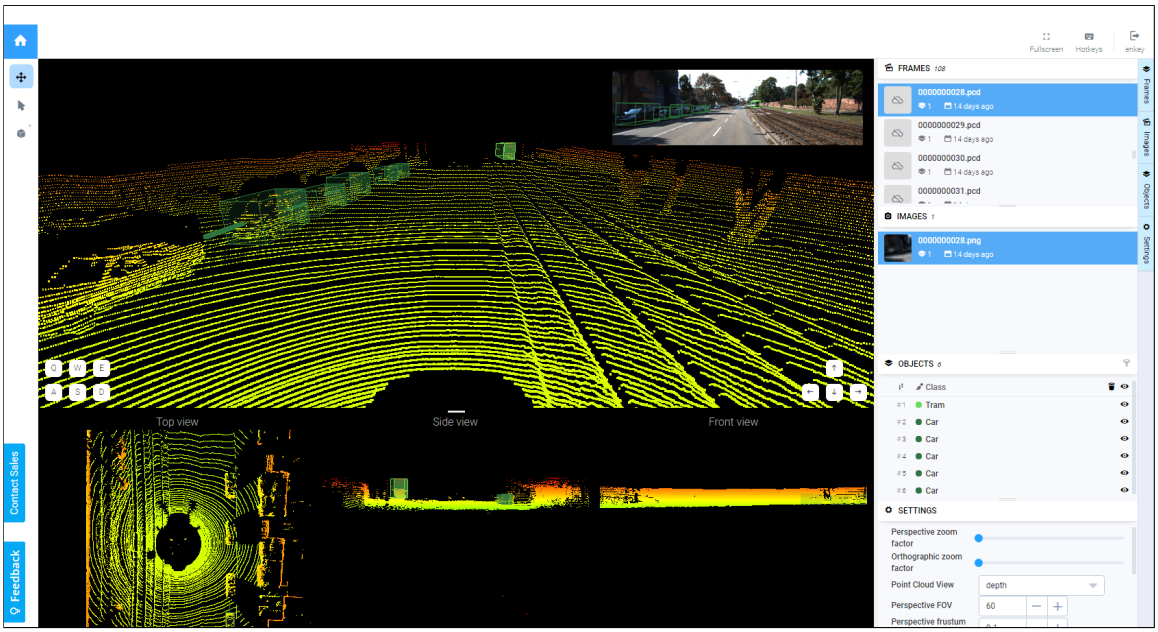
[5] https://www.robots.ox.ac.uk/~vgg/

osoba v daném segmentu videa provádí. Toto lze vidět na obrázku 4.3. Anotace lze exportovat pouze ve formátu CSV a importovat lze pouze sdílené VIA projekty přes unikátní ID nebo projektový soubor ve formátu JSON.

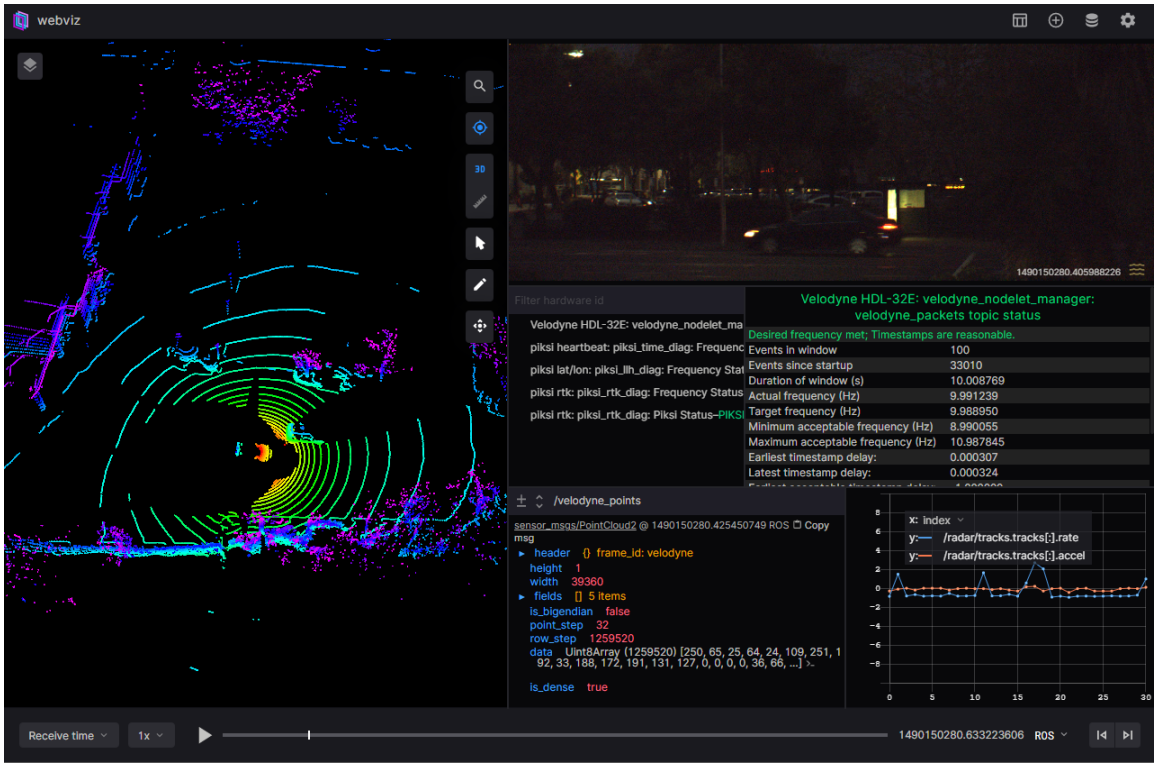
Supervisely

Pro anotaci 3D dat existují převážně komerční produkty. Jedním z takových je nástroj Supervisely. Tento nástroj podporuje jak 2D anotaci obrázků a videí, tak 3D anotaci point cloud dat, na kterou se zde zaměříme. Nástroj je webový. Je k dispozici veřejná instance, ale lze si ji hostovat i lokálně. Limitovaná verze pro nekomerční účely je k dispozici zdarma. Supervisely má také API, které umožňuje tento nástroj rozšiřovat o další funkce.

Nástroj podporuje formáty PCD a KITTI včetně synchronního zobrazení videa z kalibrované kamery. Anotace lze následně exportovat ve formátu JSON. Anotovat lze však pouze pomocí kvádrů, kterým lze přiřadit klasifikační třídu. Jak jde vidět na obrázku 4.4, na anotační obrazovce je hlavním prvkem 3D zobrazení mračna bodů, kde ve vrchní části perspektivní zobrazení scény, kde lze libovolně pohybovat scénou. Ve spodní části jsou pak tři ortogonální zobrazení: pohled shora, pohled z boku a pohled zepředu. Tyto pohledy slouží pro snazší umisťování anotačních kvádrů do scény. V pravém horním rohu je pak synchronizovaný náhled z kalibrované kamery, pokud je k dipozici. V tomto náhledu se také zobrazují anotace z 3D scény. 3D zobrazení se dá různě konfigurovat. Například lze změnit způsob obarvení bodů: podle výškové souřadnice, podle vzdálenosti od senzoru nebo obarvení jednou barvou. Lze také upravit počet zobrazovaných bodů, upravit jejich velikost rozsah, od jaké a do jaké vzdálenosti od senzoru se budou body zobrazovat.



Obrázek 4.4: Snímek anotačního prostředí nástroje Supervisely. V levé části se nachází lišta s nástroji pro anotaci a v pravém panelu je seznam snímků a anotovaných objektů. Uprostřed je zobrazení 3D dat z různých pohledů včetně záznamu z kamery v pravém horním rohu.



Obrázek 4.5: Snímek z nástroje Webviz.

Pointly

Pointly je poslední anotační nástroj, který si zde popíšeme. Jedná se o komerční nástroj pro anotaci 3D point cloud dat. Tento nástroj je především specializován na sémantickou segmentaci, podporuje však pouze soubory typu LAS/LAZ. Verze zdarma není k dispozici, pouze omezené demo.

Anotace probíhá vybráním jedné z klasifikačních tříd a následným označením bodů, které do této třídy spadají, pomocí polygonů nebo kvádrů. Body lze také označit pouhým kliknutím na jeden bod a umělá inteligence, zabudovaná v nástroji, označí i okolní body, které vyhodnotí jako související s objektem. Takto lze jednoduchým způsobem vybírat velké plochy bodů.

Webviz

Nakonec se podíváme na nástroj Webviz. Tento nástroj neslouží pro anotaci dat, ale pro přehrávání BAG souborů. Jedná se open source nástroj vyvinut firmou Cruise, která tento software využívá k vizualizaci dat získaných z jejich autonomních vozidel. Nástroj umožňuje přehrávat data z lokálního BAG souboru, streamovat z cloudu nebo přes WebSocket získávat data živě přímo z robota. Jedná se o čistě klientskou webovou aplikaci vytvořenou v knihovně React. V rámci toho projektu vzniklo několik užitečných knihoven, jako je například WorldView[6]. Tato knihovna poskytuje jednoduché renderování 2D a 3D scén pomocí knihovny React a regl, jenž může být velmi užitečná při zpracovávání této diplomové práce.

[6] https://webviz.io/worldview/

Jak lze vidět na obrázku 4.5, hlavní okno aplikace je, kromě záhlaví, tvořeno ovládacími prvky pro přehrávání obsahu BAG souboru, které lze nalézt ve spodní části obrazovky. Nachází se zde přepínaní mezi časovými razítky, jenž by měly být použity pro přehrávání obsahu. Na výběr je čas přijetí zprávy nebo čas uvedený v hlavičce zprávy. Dále je zde výběr rychlosti přehrávání, spuštění/pauza přehrávání, časová osa pro rychlou orientaci a ovládání přehrávání, časové razítko aktuálního snímku s možností přepnutí jeho formátu zobrazení a jako poslední jsou tlačítka pro zobrazení předchozího a následujícího snímku. Ve zbylé části obrazovky se nachází panely pro zobrazení dat v různých formátech. Umístění, velikost i počet panelů lze libovolně upravit a toto rozmístění lze importovat i exportovat.

Nástroj nabízí řadu panelů, každý určený pro jiný způsob zobrazení dat nebo diagnostiku nástroje, či publikování nových zpráv do BAG souboru. Pro zobrazování dat jsou nejužitečnější panely pro zobrazení 3D dat, obrázků, grafů nebo tabulek. Každému panelu lze určit, jaké zprávy má zobrazovat, zvolením jednoho z témat ze zdrojového souboru. Avšak některé panely umí zobrazovat pouze určité typy zpráv, například *3D panel* nebo panel, pro zobrazení obrázků.

4.3 Výběr architektury

I když se má jednat o webovou aplikaci, nemusí to nutně znamenat, že aplikace bude mít klient-server architekturu. Pokud aplikace nepotřebuje sdílet data mezi několika klienty, může běžet pouze lokálně u klienta v prohlížeči bez serveru. V této sekci si probereme výhody a nevýhody obou těchto řešení.

Nejprve se podíváme na architekturu klient-server. V této architektuře je jeden centralizovaný server (může být rozdělen na více serverů pro rozdělení zátěže, ale to zde neuvažujme), který obsluhuje všechny klienty. Z tohoto serveru si klienti stahují data, které prezentují uživateli. Jelikož je server centralizovaný, mohou tak uživatelé přistupovat ke stejným datům. Výhody této architektury pro námi vyvíjený nástroj jsou následující:

- Uživatelé mohou spolupracovat na anotaci souborů.
- Uživatelé mohou přistoupit ke svým datům z více zařízení.
- Méně zátěže pro klienta (co se týče zpracování a čtení BAG souborů).
- V případě výpadku na straně klienta zůstanou data uložená na serveru (nebo dojde k minimální ztrátě)

Naopak mezi nevýhody se řadí tyto věci:

- BAG soubory mohou být obrovské (i stovky GB) a jejich nahrávání na server může být dlouhé.
- Potenciálně velká zátěž na server kvůli zpracování velkých BAG souborů.
- Nutno ověřovat práva uživatelů a jejich případnou souběžnou úpravu anotací.

Pokud bychom udělali aplikaci pouze klientskou, odpadla by nutnost vývoje serverové části a přineslo by to několik výhod:

- Soubory se nemusí nahrávat na server, ale mohou se číst přímo z klientského zařízení.

- Jednodušší na implementaci než klient-server.
- Funkční i bez připojení k síti.

Nevýhody jsou však velké:

- Zpracování velkých souborů v prohlížeči může být problematické.
- Prohlížeče mají omezený přístup k lokálním souborům klienta.
- Data jsou pouze lokální, tzn. že spolupráce uživatelů na anotaci není zabudovaná přímo v aplikaci.
- Ztráta neuložené práce při zavření či znovu načtení aplikace.

Jak lze vidět, každé řešení má svoje výhody, avšak velkou výhodou architektury klient-server je právě možnost spolupráce. Dat pro anotaci bývá často velké množství a tudíž je potřeba tento úkol rozdělit mezi více lidí. Z výše popsaných důvodů byla zvolena architektura klient-server.

Jelikož na klientské části bude použita jedna z moderních knihoven pro tvorbu webových aplikací (viz kapitola 3), které komunikují se serverem asynchronně, bude nejlepší server implementovat jako REST API. To nám přinese další výhodu — externí nástroje budou moci toto API také využívat například pro získání anotací, dat nebo automaticky nahrávat BAG soubory na server.

Mít však jeden veřejný server pro používání této aplikace by nemělo příliš smysl, jelikož by takový server měl velké požadavky na úložný prostor vzhledem k velikosti BAG souborů a také na síť pro přenos těchto dat. Aplikace bude koncipována spíše tak, že si ji každá organizace bude pro své účely hostovat na vlastním serveru. Tím pádem bude mít organizace kontrolu nad bezpečností svých dat a pokud budou přistupovat k serveru po lokální síti, bude nahrávání souborů na server velice rychlé. Jednotlivci si pak mohou server rozjet přímo na svých lokálních počítačích bez specializovaného serveru. Kontejnerizací aplikace pomocí nástroje Docker by pak spuštění serveru měla velice ulehčit.

4.4 Výběr technologií

Když máme vybranou architekturu aplikace, můžeme se pustit do výběru technologií pro implementaci. Jelikož jsme vybrali architekturu klient-server, musíme vybrat technologie nejen pro klientskou část, ale i pro serverovou část, která obsahuje i databázi. Rozebereme si tedy výběr tří hlavních technologií: knihovny pro tvorbu uživatelského rozhraní, aplikačního rámce pro tvorbu REST API a výběr databáze. Nakonec si ve zkratce shrneme různé podpůrné knihovny a nástroje, které nám pomohou ve vývoji.

Uživatelské rozhraní

Knihovny pro vývoj moderních webových aplikací byly představeny v kapitole 3. Na výběr tedy máme z knihoven Angular, React a Vue.js. Jelikož mají všechny velmi podobný princip, kterým je rozdělení uživatelského rozdělení na uzavřené a znovupoužitelné komponenty, je potřeba vybírat vhodného kandidáta podle vyspělosti, dostupnosti externích knihoven a rychlosti.

Angular se jeví jako dobrý kandidát, jelikož je velice vyspělý, existuje pro něj velké množství knihoven a i když jeho předchůdce, AngularJS, rychlostí zaostával za knihovnou React, nynější verze jsou již s touto knihovnou porovnatelné. Nevýhodou Angularu ale může být jeho strmá křivka učení.

Vue.js je z těchto knihoven nejmladší, a proto je také nejméně vyspělé a mohl by být problém s dostupností kvalitních externích knihoven. Co se týče rychlosti, je srovnatelná s knihovnou React, v některých ohledech i rychlejší.

Knihovna React je velice vyspělá a rozšířená, tudíž pro ni existuje řada kvalitních externích knihoven. Co je zde však nejdůležitější, jak již bylo uvedeno v sekci 4.2, nástroj Webviz tuto technologii také využívá a poskytuje několik React knihoven, které by mohly být velice užitečné pro tento projekt. Toto byl hlavní důvod, proč byl React pro implementaci nakonec vybrán.

REST API

Pro vývoj REST API není obecně potřeba nějaký specializovaný framework, avšak některé z nich poskytují lepší vývojářské prostředky než ostatní. Abychom zbytečně neomezovali platformy, které server bude podporovat, budeme vybírat z frameworků postavených na skriptovacích jazycích PHP a Python. Ty jsou momentálně jedny z nejrozšířenějších pro tyto účely a podporují platformy Windows, Linux i macOS. Výběr proběhl z těchto nejpopulárnějších frameworků: Symfony (PHP), Django (Python) a Flask (Python).

Framework Django zvítězil kvůli kompletní sadě funkcí, jako je například zabudovaná a přizpůsobitelná administrace databázových dat, ale také díky skvělé knihovně *Django REST framework*. Tato knihovna umožňuje jednoduše vytvářet REST API a poskytuje užitečné nástroj pro vývoj, jako je třeba automaticky generované webové rozhraní pro API, pomocí kterého je možné API jednoduše testovat nebo používat bez rozhraní klientské aplikace. Další výhodou frameworku Django je, že je v jazyce Python, jelikož systém ROS poskytuje řadu knihoven právě v tomto jazyce a tudíž je bude možné využít na serverové straně aplikace.

Databáze

U databázových technologií probíhal výběr mezi MySQL (MariaDB[7]) a PostgreSQL. Jedná se o nejrozšířenější databáze co do počtu běžících instancí a obě mají otevřený zdrojový kód. MySQL je nejpoužívanější databází pro menší projekty, jelikož je rychlá a spolehlivá. PostgreSQL vyniká v rychlosti u složitějších dotazů, ale hlavně podporuje *Multi-version-concurrency control* (MVCC), což se stará o souběžný přístup do databáze. To by mohlo být užitečné pro asynchronní zpracování BAG souboru na serveru (více viz podsekce Zpracování BAG souborů níže). Z tohoto důvodu byla vybrána databáze PostgreSQL.

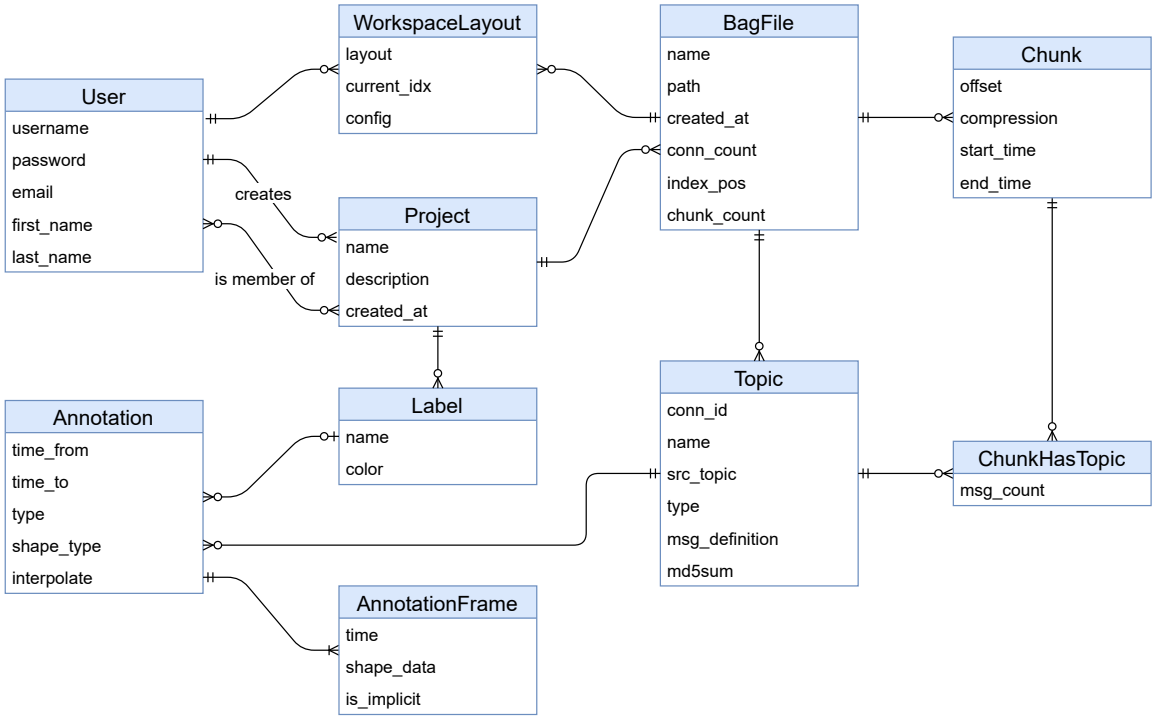
4.5 Zpracování BAG souborů

Důležitým aspektem nástroje bude práce s BAG soubory na straně serveru. Jelikož server bude klientům zprostředkovávat data z těchto souborů nahraných na server, musí mít možnost rychle najít konkrétní zprávu nebo rozsah zpráv v souboru. Soubory BAG sice obsahují indexaci pro náhodný přístup *Chunk* záznamům a volitelně také indexaci jednotlivých zpráv

[7] Vychází z databáze MySQL

v každém *Chunk* záznamu, to však stále vyžaduje tento index ze souboru přečíst při každém požadavku na data z něj. Řešením tohoto problému je uložit si pozice *Chunk* záznamů a informace o jejich obsahu do databáze. Při požadavku na zprávu pak stačí najít v databázi pozici *Chunk* záznamu, který ji obsahuje, z této pozice v souboru přečíst data daného *Chunk* záznamu a v nich vyhledat sekvenčně konkrétní zprávu. Pokud by i toto sekvenční vyhledávání bylo příliš pomalé, lze v budoucnu přidat do databáze i pozice jednotlivých zpráv v *Chunk* záznamu. Problémem však je, že žádné dostupné knihovny pro čtení BAG souboru neumožňují přímo přečíst konkrétní záznam na zadané pozici v souboru. Proto bude potřeba implementovat vlastní knihovnu v jazyce Python.

4.6 ER diagram



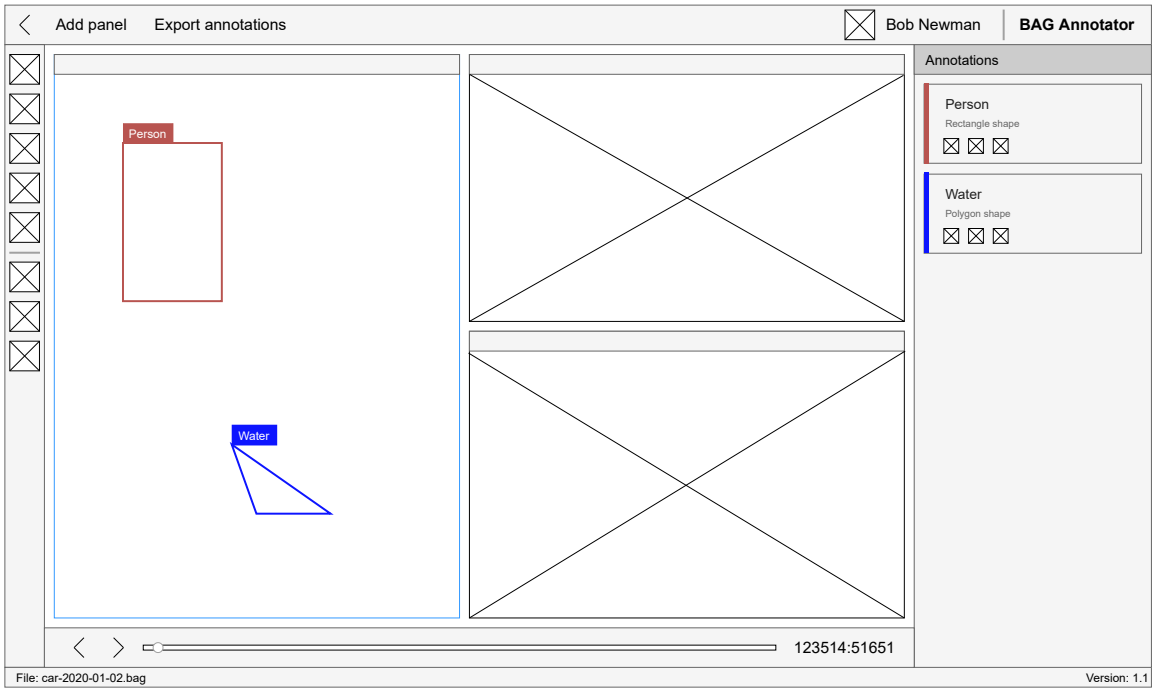
Obrázek 4.6: ER diagram aplikace.

Na základě analýzy požadavků vznikl ER diagram, který lze vidět na obrázku 4.6. Entita *User* představuje uživatele aplikace. Autentizaci, role a práva uživatelů řeší v Djangu balíček `django.contrib.auth`, proto entity související s touto funkčností nejsou v diagramu uvedeny (jsou generovány automaticky). Entita *Project* rozděluje anotaci do souborů pro jejich jednodušší správu a kategorizaci. Projekt má informaci o tom, který uživatel jej založil a kdo jsou jeho členové. Pouze členové a zakladatel mají přístup k projektu. Projekt je dále v relaci s entitami *Label* a *BagFile*. Entita *BagFile* představuje nahraný BAG soubor. Ten je přiřazen k jednomu projektu a entity *Topic* a *Chunk* obsahují metadata o souboru — tedy jaké témata se v něm vyskytují a pozice jednotlivých *Chunk* záznamů. Entita *ChunkHasTopic* určuje počty zpráv na jednotlivých tématech v daném *Chunk* záznamu. *Annotation* představuje anotaci jednoho objektu. Tato anotace může mít rozsah přes více než jednu zprávu, což určují atributy `time_from` a `time_to`. *AnnotationFrame* pak bude obsahovat

klíčové snímky anotace, mezi kterýma se bude interpolovat. Zpráva, které klíčový snímek náleží je identifikována atributem `time` a relací s entitou *Topic* přes entitu *Annotation*. Tvar anotace je uložený v klíčovém snímku v atributu `shape_data` ve formátu JSON, což umožní jednoduše rozšířit aplikaci o nové tvary. Tvar anotace je určen atributem `shape_type` v entitě *Annotation*. Jedna zpráva může mít více anotací. Každá anotace by měla mít přiřazený *Label* (nemusí ho však mít hned při vytvoření). Jelikož data budou zobrazována v panelech, které si každý uživatel může přizpůsobit, dle své potřeby, je potřeba stav rozložení také ukládat v databázi. K tomu slouží entita *WorkspaceLayout*. Ta uchovává nejen rozložení panelů, ale i aktuální pozici v souboru nebo různé nastavení. Položky `layout` a `config` jsou ve formátu JSON, jelikož jejich struktura je dynamická.

Návrh uživatelského rozhraní

Nejdůležitější částí uživatelského rozhraní je přehrávač, který bude sloužit k anotaci BAG souboru. Návrh tohoto rozhraní můžete vidět na obrázku 4.7. V horní části obrazovky se nachází šipka zpět, kterou opustíme aktuální obrazovku a vrátíme se zpět na obrazovku s projektem. Vpravo od šipky se nachází hlavní nabídka této obrazovky. Zde se budou nacházet položky, jenž se týkají celého přehrávače (např. přidání nového panelu, export anotací atd.). Na pravé straně záhlaví se pak nachází logo nástroje a jméno aktuálně přihlášeného uživatele. Na spodní straně obrazovky se nachází stavová lišta, která informuje o aktuálním stavu aplikace a poskytuje dodatečné informace.



Obrázek 4.7: Wireframe návrh uživatelského rozhraní pro anotaci.

Zbytek obrazovky tvoří hlavní část a tou je samotný přehrávač. Nalevo se nachází lišta nástrojů, kde budou umístěny nástroje pro ovládání a vytváření anotací. Jako první jsou globální nástroje a pod nimi, oddělené horizontální čarou, se nachází kontextové nástroje, které se mohou měnit v závislosti na aktuálně vybraném panelu zobrazení. Panely zobrazení

se nachází uprostřed obrazovky, jejich rozložení je přizpůsobitelné, včetně jejich množství a typů. Každý typ panelu bude podporovat určité typy zpráv a bude je zobrazovat určitým způsobem. Z tohoto plyne, že nemusí umět podporovat všechny typy anotací (např. 2D zobrazení nebude podporovat 3D anotace a naopak). Panel tedy bude definovat i podporované typy nástrojů a právě ty se budou zobrazovat v kontextové části lišty nástrojů.

Panel, který je aktuálně zvolený, bude zvýrazněn (na obrázku 4.7 světle modrým okrajem) a k němu budou na boční liště vpravo zobrazeny bližší informace o anotacích. V této boční liště se nachází seznam všech anotovaných objektů na daném panelu. U každého objektu je uveden přiřazený *label* (např. Person, Water), tvar, jenž objekt vyznačuje, a několik tlačítek s rychlými akcemi.

Pod panely zobrazení se nachází ještě lišta s ovládacími prvky přehrávače, kde se nachází tlačítka pro skok o snímek vzad, spuštění/pauza přehrávání a skok o snímek vpřed. Uprostřed se nachází časová osa zpráv v souboru, pomocí které se můžeme rychle přesunout na určitý čas. Napravo se pak nachází časové razítko aktuálně zobrazených zpráv.

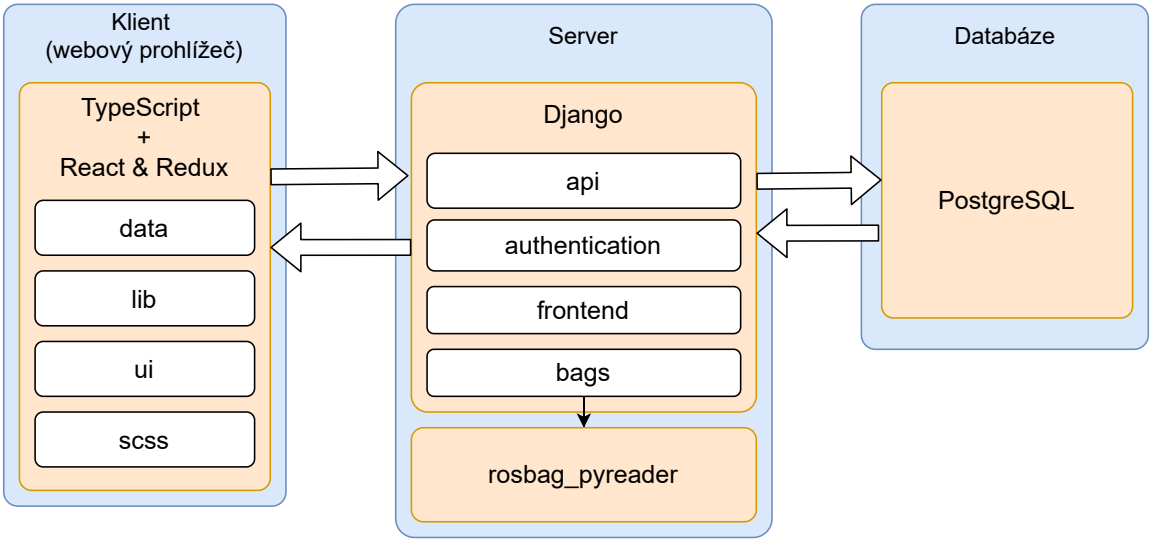
Kapitola 5

Implementace

V této kapitole se budeme zabývat implementací jednotlivých částí nástroje, který byl pojmenován BAG Annotator. Nejprve si rozebereme architekturu aplikace... na nahrávání BAG souborů, kde je potřeba řešit problém s nahráváním velkých souborů, a následně si rozebereme, jak se budou soubory zpracovávat pro rychlé čtení zpráv.

5.1 Architektura

Aplikace je rozdělena na serverovou a klientskou část. Serverová část je implementována jako REST API v jazyce Python 3 za pomocí frameworku Django s knihovnou Django REST framework. Pro čtení dat z BAG souborů slouží vlastní samostatná knihovna `rosbag-pyreader`. Klientská část využívá knihovnu React a celá aplikace je kontejnerizována pomocí nástroje Docker pro jednoduché spouštění. Strukturu celé aplikace lze vidět na obrázku 5.1.



Obrázek 5.1: Struktura aplikace

Django umožňuje rozdělit backendovou část na oddělené znovupoužitelné části, tzv. *aplikace*. Nástroj BAG Annotator je rozdělen na 4 aplikace: `api`, `authentication`, `bags` a `frontend`. Aplikace `api` obsahuje REST API rozhraní pro klientskou část a veškeré databázové modely. `Authentication` se stará o autentizaci uživatele. Poskytuje API end-

pointy pro přihlášení a odhlášení uživatele a získání aktuálně přihlášeného uživatele. Pro čtení a zpracování BAG souborů slouží aplikace `bags`. Využívá k tomu již zmíněnou knihovnu `rosbag-pyreader`. Poslední je aplikace `frontend`. Ta se stará o servírování souboru `index.html`, který vznikne sestavením frontendu, jehož zdrojové kódy se také nacházejí v této aplikaci.

Frontendová část aplikace byla vygenerována pomocí nástroje *Create React App*, který vygeneruje základní složkovou strukturu a konfigurační soubory pro pohodlné vyvíjení *single-page* aplikace[1] v Reactu. Výhodou použití tohoto nástroje je, že se postará o nastavení všech běžně používaných knihoven a pluginů pro Webpack, jenž slouží pro sestavení projektu, a také ulehčuje aktualizaci knihoven v budoucnu. Nevýhodou však je, že tyto konfigurační soubory jsou vývojáři skryté v knihovně a nelze je měnit. Zpřístupnit je lze pouze příkazem `npm run eject`, který konfigurační soubory exportuje z knihovny do vašeho zdrojového kódu. Tato akce je však nenávratná a přijdeme tímto o možnost jednoduché aktualizace knihoven. Jedna z možností, jak toto obejít, je například knihovna `craco`, umožňující měnit vnitřní konfiguraci knihovny *Create React App*. To umožnilo vytvoření TypeScript aliasů pro import souborů, pokročilejší nastavení proxy pro vývojový server a použití knihovny `typings-for-css-modules-loader`, která nahrazuje knihovnu `css-loader` a umožňuje využívat CSS moduly s jazykem TypeScript. CSS moduly slouží k lokálnímu stylování React komponent. Umožňují styly shlukovat do modulů, což jsou soubory s příponou `*.module.scss` a ty následně importovat do zdrojových kódů komponent. Ukázku použití lze vidět ve výpisu 5.1. Pro každou CSS třídu v modulu se vytvoří při kompilaci unikátní název, aby tyto styly neovlivňovaly zbytek stránky. Díky toho jsou styly lépe udržovatelné a snadno dohledatelné, jelikož jsou úzce vázány na jejich použití.

```
// soubor Example.module.scss
.wrapper {
    padding: 10px;
}

.inner {
    color: red;
}

// soubor Example.tsx
import styles from './Example.module.scss';

export default function Example() {
    return <div className={styles.wrapper}>
        <div className={styles.inner}>
            ...
        </div>
    </div>;
}
```

Výpis 5.1: Ukázka využití CSS modulů

[1] více viz https://developer.mozilla.org/en-US/docs/Glossary/SPA

Kód frontendové části je dále členěn do čtyř hlavních složek:

- `data` – třídy a funkce pro správu interního stavu aplikace a komunikace s REST API,
- `lib` – vlastní malé knihovny (ChunkReader a WorldviewTransforms),
- `ui` – React komponenty a pomocné funkce pro vykreslování uživatelského rozhraní,
- `scss` – zdrojové kódy globálních stylů v jazyce SCSS.

Ve složce `data` jsou třídy a funkce dále děleny do složek, podle domény (například `annotation`, `auth`, `bagfile`, `user` atd.). Ty pak obsahují mimo jiné definice modelů, služby nebo části Redux storu (viz sekce 5.1). Složka `ui` se dále dělí na podsložky `components` obsahující globální znovupoužitelné komponenty, `hooks`, kde jsou různé vlastní React hooky, `pages`, s komponentami pro jednotlivé stránky aplikace, a `panels` pro jednotlivé panely sloužící k zobrazování a anotaci dat z BAG souborů.

Struktura REST API

Jak již bylo zmíněno, REST API je implementováno knihovnou Django REST framework. Tato knihovna přidává speciální pohledy, které na rozdíl od klasických Django pohledů vracejících vyrenderované HTML šablony, vracejí strukturovaná data. Ty mohou stejně jako v klasickém Djangu pocházet z ORM modelů nebo i z externích zdrojů (v našem případě z BAG souborů), avšak většinou ještě projdou přes tzv. *serializery*, jenž data transformují do výsledné struktury, nebo naopak transformují přijaté data z těla HTTP požadavku na databázový model. Výsledná strukturovaná data jsou formátována pomocí rendererů do JSONu nebo XML.

Jelikož v REST API máme většinou pro každou entitu několik standardních operací (vytvoření, získání, aktualizace, mazání atd.), poskytuje knihovna Django REST framework nad tímto abstrakci v podobě *viewsets*. To jsou obecné sady pohledů zabalené do abstraktních tříd, které implementují zmíněné standardní operace a stačí pouze specifikovat `queryset`, což vlastně určuje, nad jakými daty má pohled pracovat, a `serializer_class` určující *serializer* pro tyto data. Na základě těchto standardních sad pohledů, byla stanovena konvence struktury API endpointů vyobrazena v tabulce 5.1.

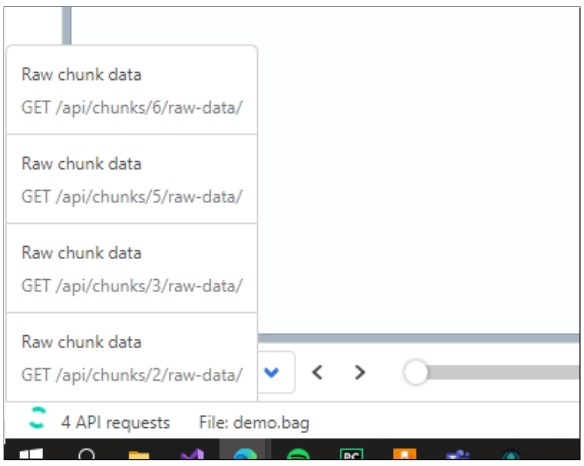
Metoda	URL	Popis
GET	`/<resource>/`	Získání kolekce všech položek
POST	`/<resource>/`	Vytvoření nové položky
kterákoliv	`/<resource>/<action>/`	Akce nad celou kolekcí položek
GET	`/<resource>/<id>/`	Získání konkrétní položky podle ID
PUT	`/<resource>/<id>/`	Aktualizování hodnot celé položky
PATCH	`/<resource>/<id>/`	Částečná aktualizace hodnot položky
DELETE	`/<resource>/<id>/`	Smazání položky
kterákoliv	`/<resource>/<id>/<action>/`	Akce nad konkrétní položkou
GET	`/<resource>/<id>/<collection>/`	Získání kolekce položek v relaci

Tabulka 5.1: Konvence REST API struktury

Napojení frontendu na API

Díky zavedené konvenci ve struktuře REST API, bylo možno i na fronendu vytvořit generické třídy pro zasílání požadavků na API, čímž odpadne repetitivní psaní kódu pro každou sadu endpointů.

Nejdříve byla vytvořena třída `ApiClient` obalující knihovnu `axios` pro posílání asynchronních požadavků na API. Poskytuje jednodušší rozhraní pro zasílání požadavků a v Redux storu udržuje aktuální seznam nedokončených požadavků. Tento seznam a aktuální počet probíhajících požadavků je v uživatelském rozhraní zobrazen ve spodní liště v levé části obrazovky, jak lze vidět na obrázku 5.2.



Obrázek 5.2: Zobrazení aktuálně probíhajících API requestů v uživatelském rozrhraní (celý seznam se zobrazí až po kliknutí na „4 API requests“)

Třídu `ApiClent` pak využívají tzv. služby, což jsou třídy, které jsou registrovány do registru `ServiceRegistry` a následně zpřístupněny React komponentám pomocí knihovny `react-service-container`. Ta používá React Context pro distribucí služeb stylem *dependency injection*, kdy React komponenty nepoužívají služby přímo, ale vyžádají si její instanci právě z React Contextu pomocí hooku `useService`, jenž je také z této knihovny. To mezi komponentami a službami vytvoří pouze volnou vazbu (angl. *loose coupling*), což je lepší například pro testování komponent, kdy můžeme jednoduše službu nahradit za pouhý *mock* této služby.

`ModelService` je abstraktní třída, kterou rozšiřují služby podporující standardní operace nad daty, jako je vytváření, mazání, získání položky a tak dále. Tato třída obsahuje standardní rozhraní pro provádění těchto operací skrze REST API. Při rozšíření této třídy stačí pouze v konstruktoru předat nadřazené třídě schémata návratových hodnot z API, jenž jsou definována pomocí knihovny `yup`. Tyto schémata pak slouží k transformaci dat z API na objekt, který je v souladu s daným schématem. Užitečné například pro automatický převod data z řetězce na objekt typu `Date`. Schémata se využívají také k validaci formulářů, jelikož kromě typu hodnot lze definovat i omezení (maximální délka řetězce, číslo větší než 0 atd.) a knihovna `Formik`, využívaná k vytváření formulářů, nativně `yup` pro validaci podporuje.

Správa stavu klientské aplikace

V Reactu je uživatelské rozhraní závislé pouze na vnitřním stavu aplikace. Tímto stavem se myslí například data načtená z API, data popisující stav uživatelského rozhraní (jestli je tooltip zobrazen, jestli modální okno viditelné a tak dále). React umožňuje tento stav ukládat lokálně v komponentách nebo globálněji v *Context*. Ten lze vytvořit funkcí `React.createContext`, což vytvoří objekt obsahující komponenty `Provider` a `Consumer`. Při použití komponenty `Provider` se jí předá aktuální hodnota, kterou má poskytovat, a všechny komponenty, jež jsou jejími potomky, mají přístup k této hodnotě s použitím komponenty `Consumer` nebo za použití hooku `useContext`. To je výhodné, pokud tyto data používá více vzdálených komponent, kde bychom při předávání stejných dat přes *props* museli data předávat přes velké množství jiných komponent, které tyto data nepotřebují a pouze je předávají dál.

Dalším způsobem uchovávání stavu aplikace, je použití externích knihoven. Mezi dvě nejznámější patří Redux a MobX. Knihovna MobX využívá návrhový vzor *Observer*. Umožňuje vytvářet třídy, které uchovávají určitý stav a definují akce, kterými lze tento stav měnit. Jednotlivé komponenty, jež potřebují tento stav, se obalí funkcí `observer`. Komponenta takto může naslouchat na změny stavu (při jeho změně se přerenderuje). Jelikož je stav oddělen od komponent, může jej využívat kterákoliv komponenta bez ohledu na pozici ve stromu vyrenderovaných komponent.

Knihovna Redux přináší koncept jednoho globálního stavu, tzv. *store*. Tento stav lze měnit pouze pomocí akcí. Akce je objekt, který obsahuje položku `type`, což je unikátní řetězec označující typ akce, a `payload` – data přibalená k akci. Akce jsou vytvářeny funkcemi, tak zvané *action creators*. Dále jsou předány do tak zvaného kořenového *reduceru*. To je *pure*[2] funkce, která bere jako parametr aktuální stav a danou akci, a vrátí nový stav se změněnými hodnotami. Reducerů je většinou celá hierarchie, kde se každý reducer stará pouze o malou část stavu. Komponenty pak získávají data ze *storu* pomocí HOC[3] komponenty `connect` nebo hooku `useSelector`, kterému je předán tak zvaný *selector*, což je funkce co bere jako parametr aktuální stav a vrací nějakou jeho část.

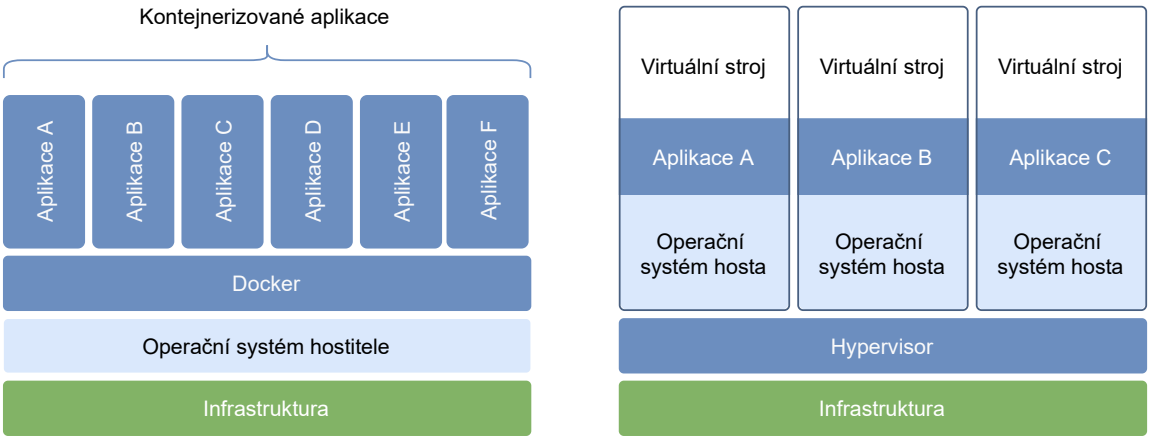
Už z tohoto popisu je možná jasné, že používání této knihovny vyžaduje dost „*boilerplate*“ kódu, zvlášť pokud chcete používat typové kontroly TypeScriptu. Tohle je jedna z nevýhod Reduxu. Proto existuje oficiální knihovna Redux Toolkit, která se snaží zmenšit množství tohoto kódu pomocí sady užitečných funkcí pro snadnější vytváření akcí a reducerů.

Po zkušenostech z předchozích projektů, je správa stavu v aplikaci BAG Annotator řešena kompromisem mezi lokálním uchováváním stavu v komponentách a ukládáním celého stavu v globálním Redux *storu*. Stav, který jsou používaná pouze jednou komponentou nebo menším množství blízkých komponent je ukládán lokálně, a pouze stav globálního charakteru (aktuálně přihlášený uživatel, seznam právě probíhajících požadavků na API, data z BAG souboru a tak dále) je ukládán v globálním Redux *storu*. Použit byl také React *Context* a to převážně na globálnější uchování stavu, který nelze serializovat (jelikož data v Redux *storu* by měla být serializovatelná), což jsou například instance objektů s metodami, nebo také pro stav, jenž je využíván komponentami v různě vzdálených úrovních jedné větve renderovaného stromu komponent, avšak je lokální pouze pro tuto větev.

[2] *Pure function* je funkce, jejíž návratová hodnota závisí pouze na jejích parametrech
[3] Higher-Order Component https://reactjs.org/docs/higher-order-components.html

Kontejnerizace aplikace

Aby bylo možné aplikaci jednoduše spustit na svém počítači nebo serveru, je kontejnerizovaná pomocí nástroje Docker. Kontejner je standardní jednotka softwaru, která zahrnuje kód a veškeré jeho závislosti, aby aplikace běžela rychle a spolehlivě v různých počítačových prostředích [6]. Uživatel si tedy nemusí instalovat veškeré závislosti a nástroje pro zprovoznění vlastního serveru s aplikací, ale pouze stačí mít nainstalovaný nástroj Docker. Zjednoduší se také distribuce aplikace, protože nemusíme zajišťovat, aby aplikace fungovala na různých operačních systémech. Na rozdíl od virtuálních strojů jsou kontejnery abstrakcí na úrovni aplikací a neobsahují operační systém [6]. Proto jsou také mnohem menší a na jednom operačním systému jich může běžet více [6]. Porovnání konterjnerizovaných aplikací a aplikací bežících ve virtuálních strojích lze vidět na obrázku 5.3.



Obrázek 5.3: Schéma kontejnerizovaných aplikací (vlevo) a virtuálních strojů (vpravo). Převzato a přeloženo z [6]

.

Aplikace využívá nástroje Docker Compose, která umožňuje aplikaci rozdělit na více služeb (kontejnerů) a ty společně propojit a ovládat příkazem `docker-compose`. Diagram rozdělení aplikace na služby lze vidět na obrázku 5.4. Služba `nginx` používá aplikaci Nginx jako webový server, který klientovi servíruje statické soubory (JS, CSS, obrázky atd.) v případě že cesta URL požadavku začíná řetězcem „/static“, jinak požadavky přepošle WSGI[4] serveru Gunicorn ve službě `django`. Gunicorn poskytuje standardní rozhraní web serveru a zprostředkovává požadavky na Django aplikaci. Django pak komunikuje s PostgreSQL databází, jenž běží ve službě `db`.

5.2 Autentizace

Autentizace uživatele je identifikace autentizačních údajů, se kterými byl požadavek proveden [1] a následné spárování s konkrétním uživatelem, který požadavek vykonal. To je důležité pro kontrolu přístupu k REST API rozhraní aplikace. Django REST framework nabízí několik způsobů autentizace uživatele [1]:

- HTTP Basic Authentication – jednoduchá autentizace, kde každý požadavek obsahuje jméno a heslo uživatele,

[4] Web Server Gateway Interface https://www.python.org/dev/peps/pep-3333/

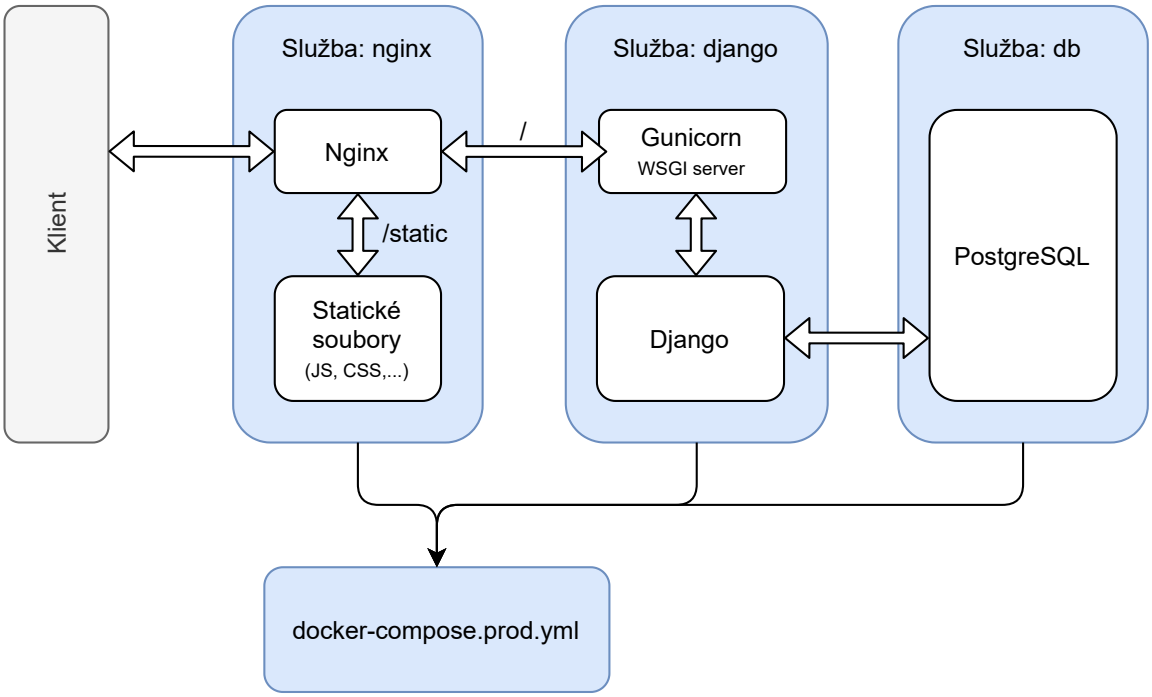
- Autentizace na základě tokenů – každý uživatel má svůj token, který je pak vkládán do HTTP hlavičky `Authorization` v každém požadavku na API,
- Autentizace přes *session* – server si po přihlášení uživatele uloží jeho informace do *session* a v odpovědi pošle cookie s identifikátorem, pomocí kterého následně páruje požadavek s přihlášeným uživatelem,
- Remote user autentizace – deleguje autentizaci webovému serveru.

Pro webovou aplikaci je nejvýhodnější autentizace pomocí *session*. Tato metoda funguje stejným způsobem jako na klasických webových stránkách, kdy se po přihlášení u klienta automaticky uloží cookie s identifikátorem pro *session* a také se následně automaticky posílá spolu s požadavky na API.

Implementace této autentizace je na serverové části jednoduchá. Do souboru pro konfiguraci frameworku Django (`settings.py`) stačí přidat do konstanty `REST_FRAMEWORK` třídu, která autentizaci přes *session* zajišťuje (viz výpis 5.2).

Poté stačí už jen udělat API endpointy pro přihlášení a odhlášení uživatele za pomocí klasických Django funkcí `authenticate`, `login` a `logout`. Na co je však potřeba si dát pozor u tohoto typu autentizace jsou CSRF tokeny [1]. „Nebezpečné“ HTTP metody (PUT, PATCH, POST, DELETE atd.) totiž v takovém případě vyžadují, aby v požadavku byl validní CSRF token. Proto byl vytvořen ještě endpoint pro jeho získání. Jedná se o jednoduchý pohled, který nevrací žádné data, ale pouze v odpovědi pošle cookie s hodnotou tohoto tokenu, čehož je docíleno použitím dekorátoru `@ensure_csrf_cookie` z frameworku Django.

Na frontedu je potřeba řešit hlavně detekci nepřihlášeného uživatele. Při načtení stránky se nejdříve pošle požadavek na API endpoint pro získání informací o aktuálně přihlášeném



Obrázek 5.4: Rozdělení aplikace na služby pomocí Dockeru Compose

uživateli (email, jméno, oprávnění, ...), které se uloží do Redux *storu*. Pokud tento nebo jakýkoliv jiný požadavek skončí s HTTP status kódem 401, aplikace uživatele přesměruje na přihlašovací stránku. Jestliže je tedy uživatel neaktivní, má aplikaci otevřenou a jeho *session* vyprší, jakýkoliv další požadavek vrátí chybový stav s kódem 401 a bude také přesměrován na přihlašovací stránku, aby se znovu přihlásil.

```
REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': (
        'authentication.authenticator.SessionAuthentication',
        ...
    ),
    ...
}
```

Výpis 5.2: Přidání autentizační třídy v nastavení frameworku Django.

5.3 Nahrávání a zpracování BAG souborů

V této sekci bude popsáno, jak bylo řešeno nahrávání BAG souborů, kde bylo potřeba řešit jejich velkou velikost, a také jejich následné zpracování pro rychlejší přístup k datům.

Nahrávání

Nahrávání velkých souborů přes protokol HTTP může být problematické. Server může mít limit pro velikost nahrávaných souborů nebo se při výpadku spojení nahrávání přeruší a musíme začít znovu. Řešením tohoto problému, je nahrávat soubor po menších částech.

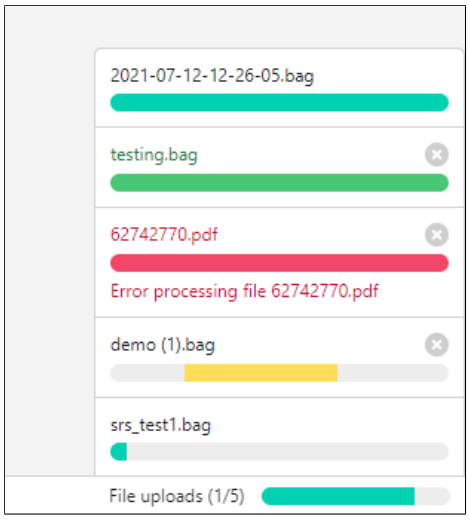
Princip tohoto řešení je jednoduchý. Na straně klienta soubor, který budeme chtít nahrát, rozdělíme na části o předem dané velikosti. Tyto části postupně odesíláme na server, který je u sebe spojuje dohromady do fyzického souboru na disku a identifikátor a stav nahrávání si ukládá v databázi. Díky toho, když se dočasně přeruší síťové spojení, můžeme po jeho obnovení pokračovat v odesílání dalších částí souboru (u klienta však nesmíme ztratit identifikátor nahrávání). Limit velikosti nahrávaných souborů tímto také obejdeme, jelikož soubor rozdělíme na části, které jsou menší než tento limit.

Pro Django existuje knihovna `django-chunked-upload`[5], která řeší serverovou část tohoto úkolu. Na klientské straně využívá knihovnu `jQuery-File-Upload`[6], jenž je postavená na knihovně jQuery[7]. Jelikož však používáme jako knihovnu React, je zbytečné přidávat závislost na knihovnu jQuery, což je docela velká knihovna, kvůli jedné funkci. Z tohoto důvodu byla zvolena vlastní implementace v Reactu, nejedná se však o nic složitého.

Uživatel může BAG soubor nahrát kliknutím na tlačítko „+ Add“ na stránce s detailem projektu. Otevře se modální okno, kde uživatel může vybrat soubory pro nahrání na server. Vstupní pole bylo vytvořeno pomocí knihovny `react-dropzone`[8], aby bylo možné soubory nahrát i pouhým přetažením z prohlížeče souborů (tzv. *drag and drop*). Po kliknutí na tlačítko *Upload* se vybrané soubory předají komponentě `BagFileUploadProvider`, což je komponenta obalující React *Context* globální úrovně (je velice blízko kořenové komponentě

[5] https://github.com/juliomalegria/django-chunked-upload
[6] https://github.com/blueimp/jQuery-File-Upload
[7] https://jquery.com/
[8] https://react-dropzone.js.org/

a tudíž všechny komponenty mají tento kontext k dispozici a je k dispozici na všech stránkách aplikace) a stará se o nahrání těchto souborů na server po částech. Na globální úrovni je to řešeno proto, aby uživatel mohl během nahrávání přejít na jiné stránky v aplikaci, aniž by se nahrávání přerušilo. Aktuální stav nahrávání lze vidět na pravé straně spodní lišty, aby měl uživatel stále přehled o tom, jestli se soubory nahrávají. Snímek uživatelského rozhraní je na obrázku 5.5. Pokud se uživatel pokusí během nahrávání ze stránky odejít (zavře okno nebo znovu načte celou stránku), aplikace jej upozorní na případnou ztrátu dat a vybídne uživatele ke zrušení této akce. Po úspěšném nahrání se soubor začne zpracovávat, což lze v uživatelském rozhraní také vidět na stejném místě jako nahrávání. Teprve až je soubor úspěšně zpracován, se přidá záznam do databázové tabulky BagFile a soubor je považován jako úspěšně nahraný na server.



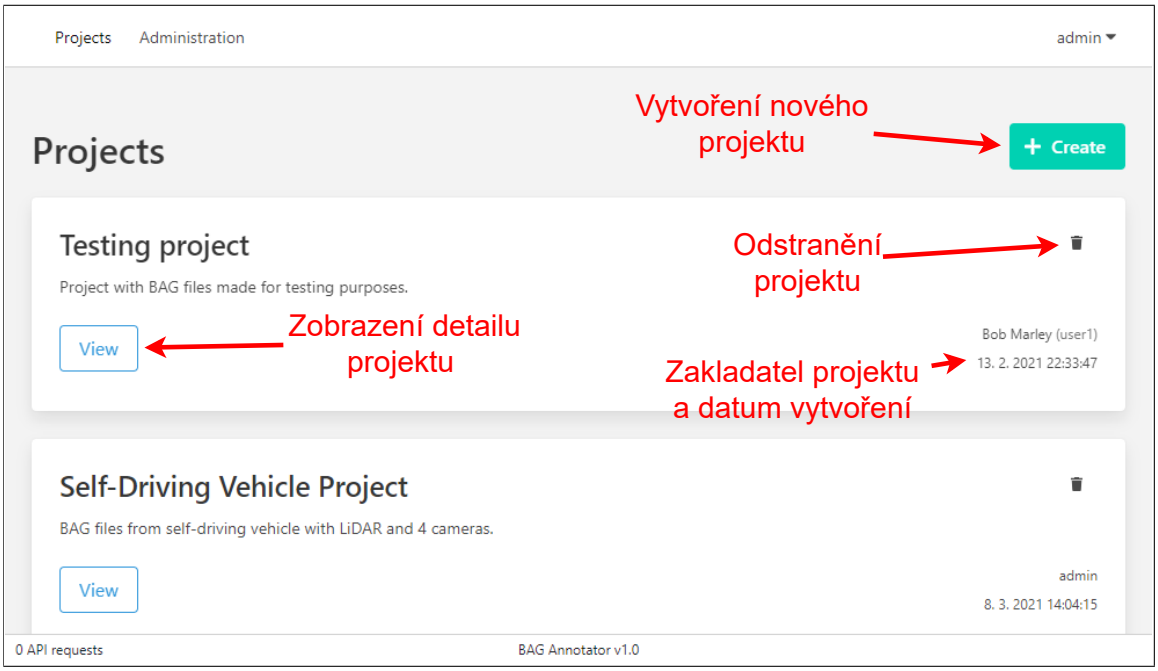
Obrázek 5.5: Zobrazení stavu nahrávaných souborů ve spodní liště. Barva ukazatele postupu indikuje aktuální stav nahrávání. Tyrkysová barva označuje právě nahrávané soubory, žlutá zpracovávané soubory, zelená úspěšně dokončené nahrávání a červená chybu během nahrávání nebo zpracování souboru.

Zpracování

Jak již bylo naznačeno v sekci 4.5, aby bylo možné rychle číst zprávy s náhodným přístupem, je nutné z BAG souboru po jeho nahrání přečíst pozice jednotlivých *Chunk* záznamů a ty si uložit do databáze. Dostupné knihovny toto však neumožňují a také neumožňují náhodný přístup bez čtení indexů ze souboru. Z tohoto důvodu bylo potřeba implementovat vlastní řešení.

Byla tedy vytvořena samostatná knihovna v jazyce Python s názvem `rosbag-pyreader`. Hlavní třídou této knihovny je třída `RosbagReader`. Ta bere jako parametr konstruktoru objekt typu `BinaryIO` (tedy například otevřený soubor v binárním režimu). Při vytvoření nové instance se ověří, že soubor je typu BAG verze 2.0 přečtením prvního řádku.

Dále třída `reader.RosbagReader` poskytuje dvě důležité metody. První je metoda `read_metadata()`, která přečte všechny potřebné metadata ze souboru — záznamy *Bag header*, *Connection* a *Chunk info*. Tyto metadata si pak můžeme uložit do databáze pro pozdější použití. Během čtení záznamů typu *Connection* dochází také k parsování definice zprávy, která se následně využívá k parsování obsahu zpráv do struktury, pomocí třídy



Obrázek 5.6: Hlavní obrazovka aplikace se seznamem projektů přihlášeného uživatele.

`messages.DefinitionParser`. Všechny tyto metadata se z metody `read_metadata()` vrací v instanci třídy `reader.BagMetadata`.

Druhá důležitá metoda třídy `reader.RosbagReader` je `read_messages_from_chunk()`. Ta přijímá parametr `metadata` typu `reader.BagMetadata`, parametr `chunk_info` typu `records.ChunkInfo` a volitelně parametr `topics`, kterým můžeme filtrovat zprávy z určitých témat (pokud tento parametr nezadáme, metoda vrací zprávy ze všech témat). Tato metoda slouží jako generátor[9] a lze ji tedy přímo použít v cyklu jako iterátor. Postupně čte záznamy v datové části daného záznamu typu *Chunk* a v případě, že se jedná o záznam typu *Message data*, rozparsuje obsah zprávy v datové části tohoto záznamu a vrátí jej pomocí klíčového slova `yield`.

Díky těmto dvěma metodám můžeme náhodně přistupovat k jednotlivým záznamům typu *Chunk* bez zbytečného čtení všech metadat. Stačí po nahrání souboru na server využít metodu `read_metadata()` třídy `reader.RosbagReader` a vrácené metadata si uložit do databáze (viz entity *BagFile*, *Chunk* a *Topic* v diagramu na obrázku 4.6). Při REST API požadavku na určitou zprávu pak stačí vyhledat v databázi záznam *Chunk*, v němž se nachází, získat všechny záznamy *Topic* daného souboru a následně využít metodu `read_messages_from_chunk()` pro přečtení zpráv z nalezeného *Chunk* záznamu, mezi kterýma pak stačí vyhledat požadovanou zprávu.

5.4 Projekty

Nahrané BAG soubory jsou organizovány do projektů. Každý projekt má svoji sadu štítků a své členy, kteří mají k projektu přístup. Ke správě slouží jednoduché uživatelské rozhraní, jenž bude v této sekci představeno.

[9] https://wiki.python.org/moin/Generators

Seznam projektů

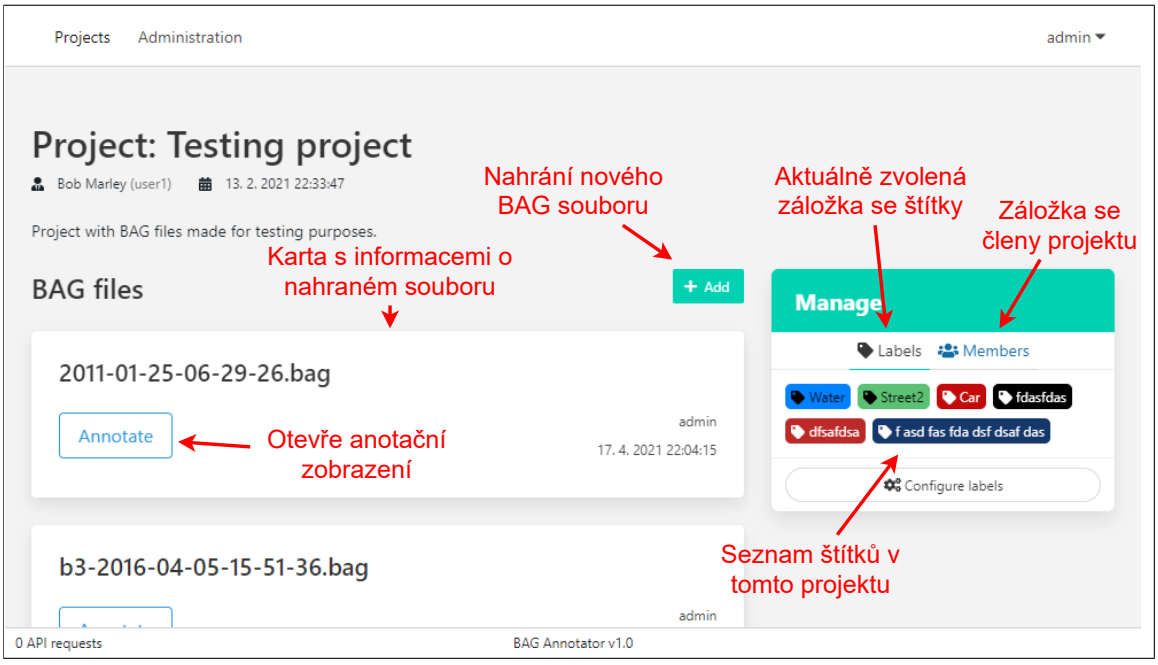
První stránka, kterou uživatel po přihlášení uvidí, je *Projects*. Snímek obrazovky této stránky lze vidět na obrázku 5.6. Uživatel zde vidí všechny projekty, jenž vytvořil, nebo jejich členem. Pokud je uživatel *superuser*, vidí všechny projekty v systému. Na této stránce se nachází také tlačítko pro vytvoření nového projektu, v kartách jednotlivých projektů se nachází tlačítko s ikonkou popelnice pro odstranění projektu. Před smazáním projektu se aplikace uživatele zeptá, zda jej chce opravdu smazat, jelikož se jedná o nevratnou destruktivní akci. Kliknutím na tlačítko *View* se přejde na detail projektu.

Detail projektu

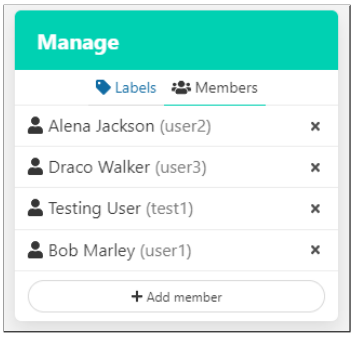
Stránka s detailem projektu obsahuje seznam BAG souborů. Jak lze vidět na obrázku 5.7, nachází se zde i tlačítko *Add* pro nahrání nového souboru. Na pravé straně se je panel se dvěma záložkami: *Labels* a *Members*. První z nich obsahuje seznam štítků daného projektu s tlačítkem *Configure labels*, které otevře stránku pro přidávání, mazání a editaci štítků. V druhé záložce, *Members*, je seznam členů projektu (viz obrázek 5.8). Tlačítkem s ikonkou křížku lze člena z projektu odstranit a tlačítkem *Add member* na spodní straně panelu lze člena přidat. Otevře se modální okno se seznamem uživatelů, kteří dosud nejsou členové, a kliknutím na jednoho z nich se potvrdí výběr. Tlačítko *Annotate* u BAG souboru jej otevře v anotačním zobrazení (viz sekce 5.6).

Nastavení štítků

Každý projekt má vlastní sadu štítků, kterou si lze nastavit na stránce ukázané na obrázku 5.9. Nachází se zde formulář pro vytvoření nového štítku, který obsahuje vstupní pole

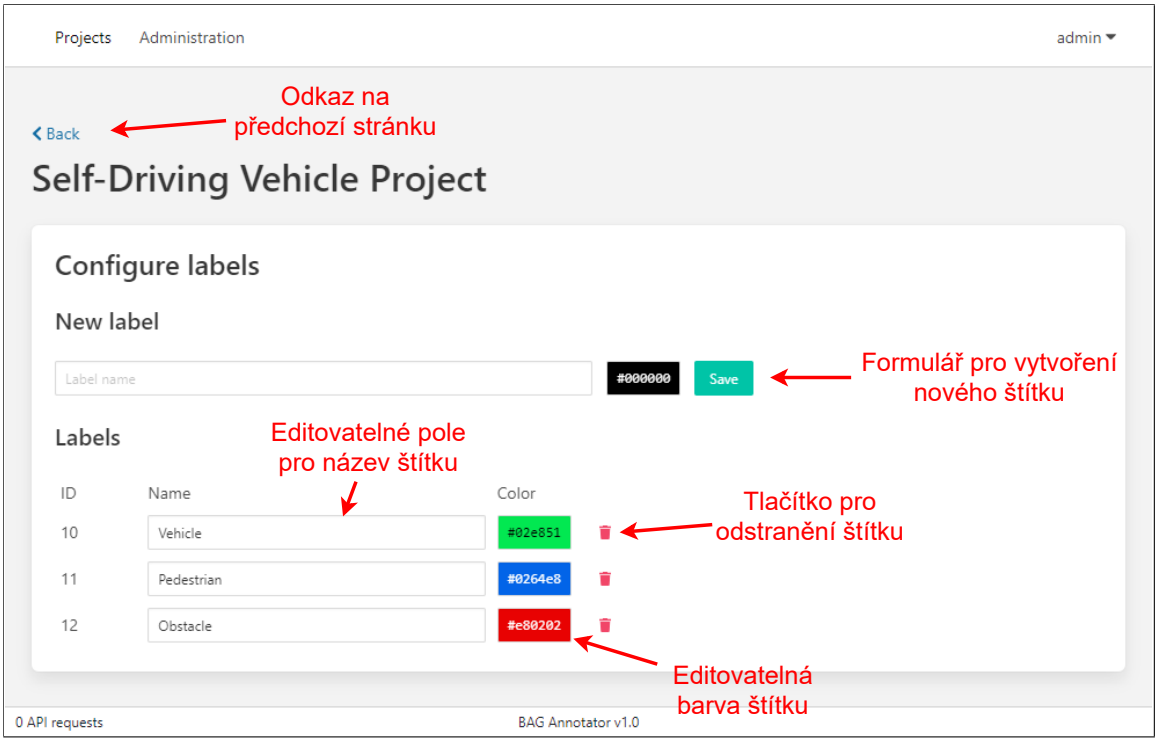


Obrázek 5.7: Detail projektu se seznamem nahraných BAG souborů.



Obrázek 5.8: Záložka se správou členů projektu.

pro název a vstupní pole pro výběr barvy. Výběr barvy je řešen knihovnou `react-color`[10], jenž poskytuje sadu různých komponent pro její jednoduchý výběr pomocí posuvníků a barevné palety. Pod tímto formulářem je seznam vytvořených štítků, kde je lze přímo i editovat. Změny jsou ukládány přímo při změně a není nutno je nijak ukládat. Každý štítek má u sebe i tlačítko s ikonkou popelnice pro vymazání, avšak vymazat jdou pouze ty, které ještě nebyly využity pro anotaci.



Obrázek 5.9: Stránka pro konfiguraci štítků konkrétního projektu.

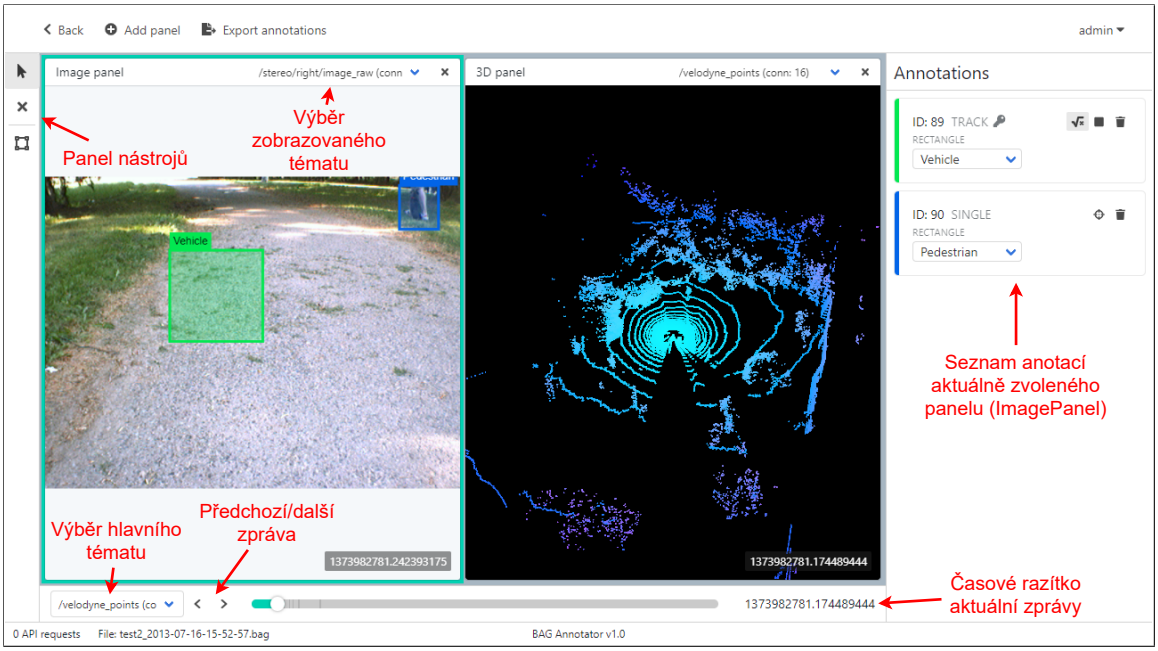
[10] https://casesandberg.github.io/react-color/

5.5 Administrace

Uživatelé s tzv. *staff* statusem mají přístup do administrace přes odkaz *Administration* v horní navigační liště aplikace. Tato administrace je poloautomaticky generována frameworkem Django. Využívá k tomu definice databázových modelů, pomocí kterých generuje formuláře pro vytváření nebo editaci záznamů. Slouží převážně znalejším a povolaným uživatelům k provádění úkonů, jenž nejsou obsaženy v uživatelském rozhraní aplikace. Například k vytváření nových uživatelů, přesouvání BAG souborů mezi projekty, editace záznamů nebo jejich mazání. Konkrétně jsou zde zpřístupněné projekty, štítky, BAG soubory, uživatelé, a skupiny. Skupiny slouží k seskupování práv uživatelů, tyto práva jsou však pouze pro administraci. Ostatní modely, jenž obsahují informace o jednotlivých BAG souborech a anotacích, nejsou v administraci zpřístupněné, jelikož jejich ruční editace nemá smysl.

5.6 Anotační zobrazení

Anotační zobrazení slouží pro zobrazení a následnou anotaci dat z vybraného BAG souboru. Toto zobrazení lze vidět na obrázku 5.10. V levé části je lišta nástrojů, kde je na výběr jako první výchozí nástroj *Pointer*. Dále je nástroj pro mazání anotací a pod ním jsou nástroje pro vytváření anotací, které závisí na tom, jestli je vybrán *Image panel* nebo *3D panel*. Aktuálně se zde nachází pouze nástroj pro vytváření obdélníkových anotací pro 2D a kvádrů pro 3D. Na pravé straně je boční panel se seznamem anotací ve zvoleném panelu, kde je možné anotacím přiřazovat štítek, přepnout anotace na režim manuálního sledování objektu nebo anotace mazat. Uprostřed obrazovky jsou panely, pro zobrazení a anotaci dat.



Obrázek 5.10: Stránka pro přehrávání a anotaci dat z BAG souboru.

Pro navigaci v aktuálně otevřeném BAG souboru slouží ovládací prvky na spodní straně stránky. Jako první vlevo je prvek pro zvolení hlavního tématu, kterým se navigace bude řídit (více viz podsekce *Synchronizace zpráv* v sekci 5.7 níže). Prvek umožňuje vybrat jakékoliv téma z BAG souboru. Napravo od něj se nachází dvě tlačítka s šipkami vlevo a vpravo.

Ty slouží k přesunu na předchozí, respektive další zprávu v hlavním tématu. Dále se zde nachází posuvník, pomocí kterého se lze rychle přesouvat na konkrétní místo v souboru a také poskytuje rychlý přehled toho, kde se uživatel v souboru nachází. Pozadí v posuvníku s tmavší šedou barvou ukazuje, jaké části souboru jsou aktuálně načtené. Posledním prvkem úplně vpravo je časová značka aktuální vybrané zprávy z hlavního tématu.

Systém panelů

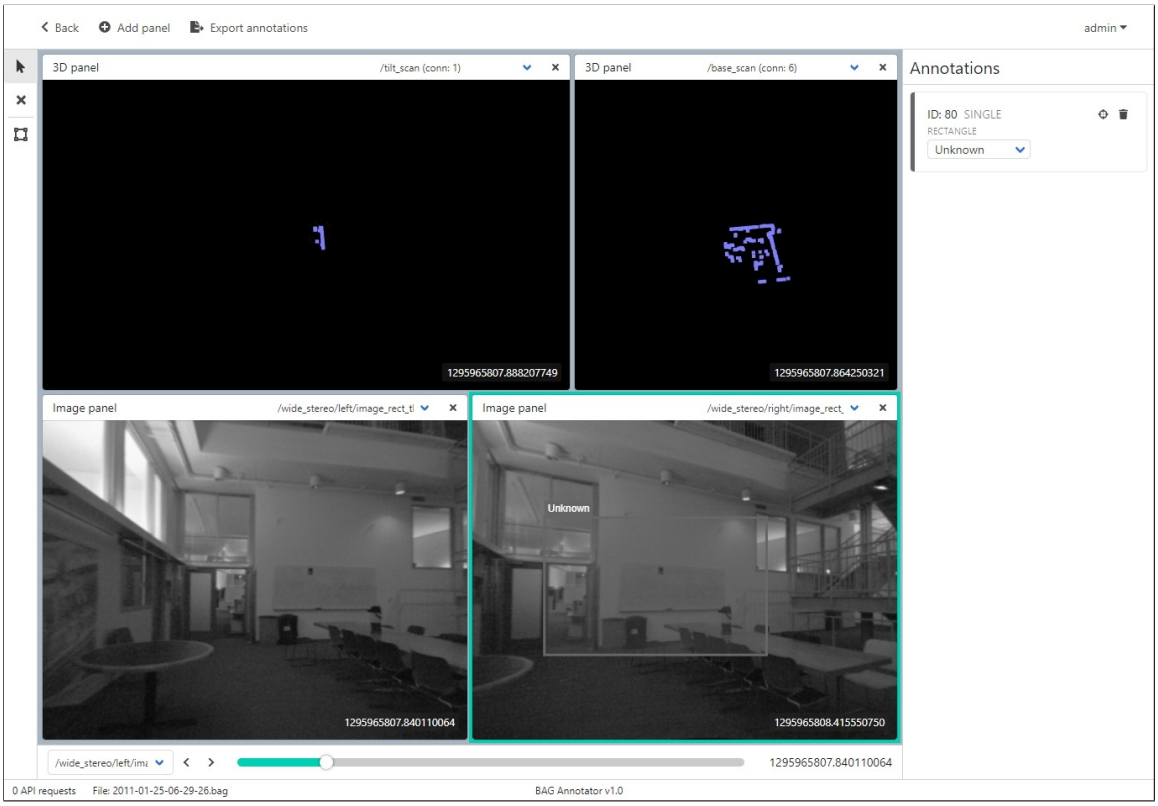
Jelikož BAG soubory mohou obsahovat data například z více kamer nebo 3D senzorů, měl by mít uživatel možnost si zobrazit všechny potřebné data najednou. To je také hlavní výhoda anotace přímo BAG souborů, oproti exportování jednotlivých témat do souborů a anotace každých dat zvlášť. Proto byl vytvořen systém panelů za pomocí knihovny `react-mosaic`[11], která se stará o jejich rozložení.

Systém panelů je navržen rozšiřitelně, aby bylo možné v budoucnu přidávat další typy. Panel je komponenta, jenž má *props* typu `PanelProps` a jako kořenovou komponentu renderuje `PanelView`, která se stará o propojení s knihovnou `react-mosaic` a vykreslení vrchního panelu s názvem, výběrem téma pro zobrazení a tlačítkem pro odstranění panelu a také časového razítka aktuální zprávy v pravém dolním rohu (lze vidět na obrázku 5.10). Dále pro panel musí být vytvořen objekt typu `PanelInfo`, jenž obsahuje informace o panelu. Jeho unikátní ID, zobrazitelný název, popis, reference na komponentu a seznam podporovaných typů zpráv. Následně je tento objekt přidán do registru panelů pomocí statické metody `registerPanel` třídy `PanelRegistry`. V souboru frontend/react/src/ui/panels/hooks.tsx je pak řada užitečných *hooků* pro panely, jako je například `useCurrentMessage`, pomocí kterého lze jednoduše získat aktuální zprávu pro zobrazení a informaci o tom, jestli se ještě načítá. Každý panel taky může definovat seznam nástrojů, které jsou pro něj specifické. Tento seznam předá jako *prop* s název *tools* již zmíněné komponentě `PanelView`. Tyto nástroje se pak při vybrání tohoto panelu zobrazí v boční liště s nástroji. Seznam nástrojů se skládá z položek, jenž obsahují unikátní ID nástroje, ikonu pro zobrazení v panelu nástrojů a text nápovědy. Aktuálně vybraný nástroj lze pak získat *hookem* `useCurrentTool` z již zmíněného souboru.

Knihovna `react-mosaic` umožňuje panely dynamicky vytvářet, přesouvat potáhnutím myši a mazat. Ukázka rozložení panelů je vidět na obrázku 5.11. Stav rozložení panelů je binární strom, jehož nelistové uzly rozdělují rozložení na dvě části a listové uzly jsou unikátní ID jednotlivých panelů. Toto ID je odlišné od dříve zmíněného ID v objektu `PanelInfo` a dále bude označováno jako ViewID. Strom je ukládán v Redux `storu` spolu s mapou jednotlivých ViewID na ID typu panelu, který se zde má zobrazit. Také je zde uložen objekt, jenž mapuje aktuální konfiguraci zobrazeného panelu na jeho ViewID. V této konfiguraci je momentálně uloženo pouze aktuálně vybrané téma, které má panel zobrazovat, ale v budoucnu by zde mohlo přibýt i další nastavení. Poslední položkou, co se pro rozložení panelů ukládá, je `nextId`, která slouží jako čítač a při přidání nového panelu se hodnota této položky použije jako jeho ViewID a následně se inkrementuje. Tím je zajištěna unikátnost těchto identifikátorů.

Aby si uživatel nemusel panely při každém otevření souboru znovu nastavovat, je tento stav ukládán při každé změně přes REST API do databáze. Informace o rozložení jsou ukládány do tabulky *WorkspaceLayout* (viz ER diagram 4.6) společně s aktuálním indexem zobrazené zprávy a zvoleným hlavním téma (*master topic*). Díky tomu při znovu načtení stránky se uživateli načte anotační zobrazení ve stejném stavu, v jakém ho zanechal, a ne-

[11] https://github.com/nomcopter/react-mosaic



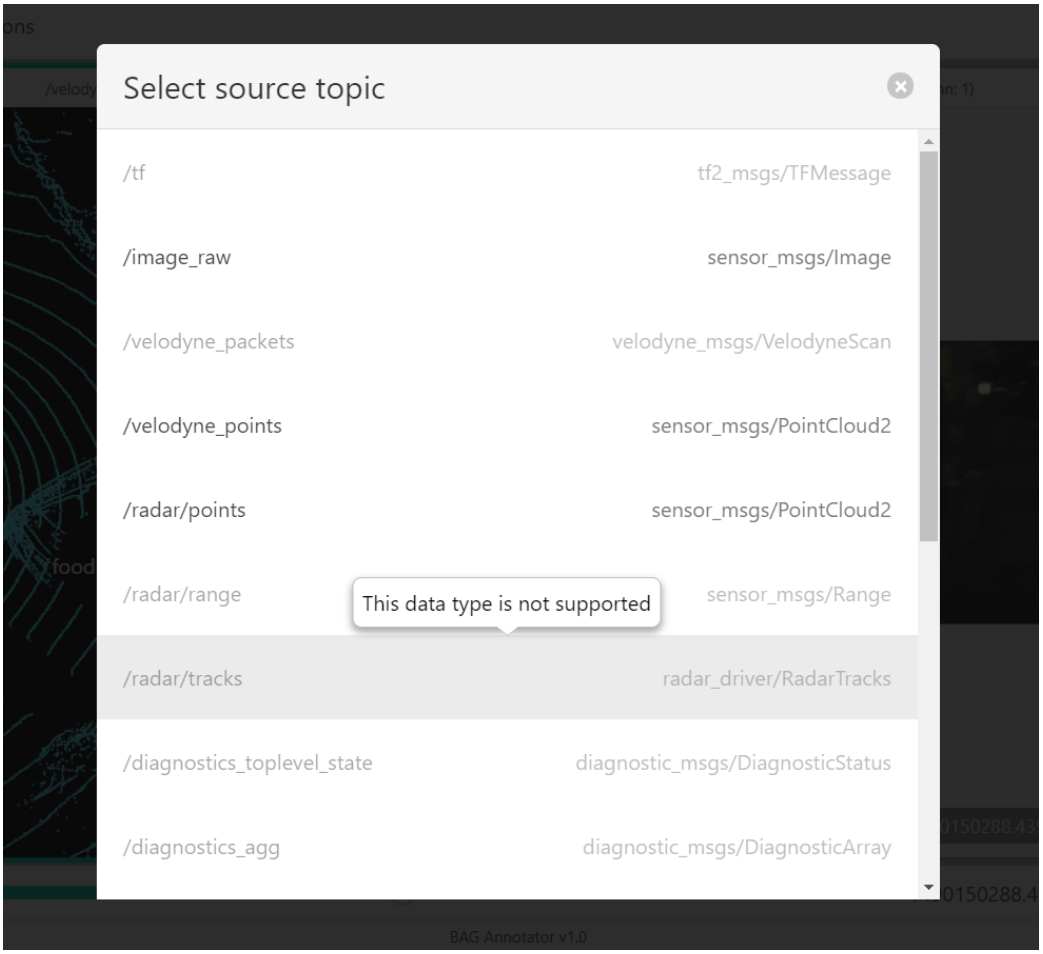
Obrázek 5.11: Rozložení panelů v anotačním zobrazení

musí hledat, na které zprávě skončil. Proto je stav individuálně ukládaný pro každého uživatele zvlášť.

Při prvním zobrazení určitého BAG souboru se aplikace sama pokusí dynamicky nastavit anotační prostředí podle dostupných typů dat. Pokud soubor obsahuje obrazová data, vytvoří *Image panel* s tímto tématem. Následně se pokusí najít vhodné téma s 3D daty. Nejprve se podívá, jestli soubor obsahuje data typu *PointCloud*, pokud ne, tak se pokusí najít data typu *LaserScan*. Jestli nějaké vhodné téma najde, vytvoří pro něj *3D panel*. Nakonec se nastaví hlavní téma, kde se primárně vezme to s 3D daty, sekundárně s obrazovými daty.

Přidat nový panel v anotačním zobrazení lze pomocí tlačítka *Add panel* v hlavním menu v horní části stránky. Po kliknutí na toto tlačítko, se otevře modální okno se seznamem všech témat spolu s datovým typem v aktuálně otevřeném BAG souboru (viz obrázek 5.12). Témata s datovým typem, který není podporován (v registru panelů není panel, který by tento typ uměl zobrazit), jsou vyznačena světlejším textem, nejde na ně kliknout a při najetím kurzoru myši se objeví popisek „*This data type is not supported,*“ nebo-li že tento datový typ není podporován.

Po vybrání jednoho z podporovaných témat se otevře další modální okno se seznamem panelů podporujících datový typ vybraného tématu. To lze vidět na obrázku 5.13. V seznamu je zobrazen název panelu a jeho popis. Tyto položky jsou definované v objektu typu `PanelInfo` daného panelu. Z obrázku 5.13 může být taky patrné, že je na výběr pouze jeden panel. To je z důvodu, že v aplikaci není registrován žádný jiný panel, který by daný typ podporoval, ale protože je systém panelů rozšiřitelný, mohlo by v budoucnu být panelů



Obrázek 5.12: Modální okno s výběrem téma pro nový panel.

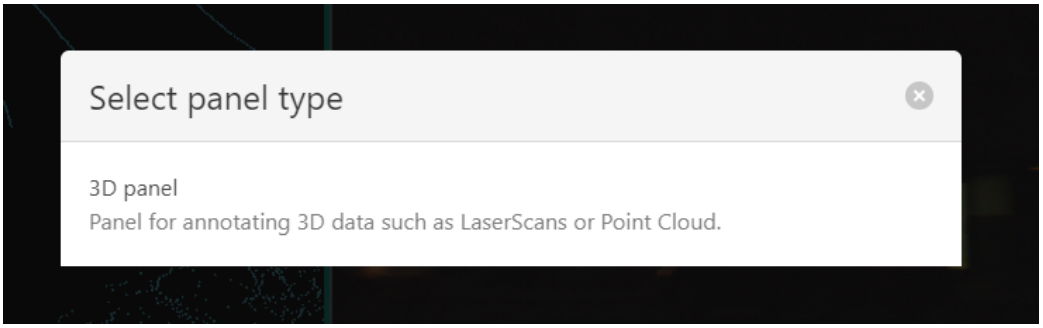
na výběr více. Například pro vývojářské účely byl vytvořen ještě panel *MessageDataPanel*, jenž umožňuje zobrazit strukturovaně data zprávy pomocí knihovny `react-json-view`[12], a ten podporuje všechny datové typy. Ve výchozím stavu však tento panel není přidán do registru (lze změnit v konfiguraci aplikace v souboru `Config.tsx`), aby běžné uživatele nepletl, jelikož není určen k anotaci.

Vybráním jedné z možností se přidá nový panel do rozložení a to tak, že se vytvoří nový kořen binárního stromu reprezentující rozložení panelu, původní kořen se dá jako levý potomek a nově přidaný panel se vloží jako pravý potomek. To znamená, že původní rozložení se horizontálně zmenší na polovinu a na druhou polovinu se vloží nový panel.

5.7 Přehrávání dat z BAG souboru

Před implementací anotací je potřeba zobrazit data z BAG souboru. K tomu bylo využito částí kódu z aplikace Webviz, která byla zmíněna už v sekci 4.2 o existujících nástrojích. Tato aplikace je totiž také psána v Reactu, je *open-source* (licence Apache 2.0) a slouží právě k přehrávání dat z BAG souborů. Z této aplikace byl využit kód pro zobrazování zpráv typu

[12] https://github.com/mac-s-g/react-json-view



Obrázek 5.13: Modální okno s výběrem typu panelu.

`Image` (konkrétně funkce na převádění dat ze zprávy na objekt typu `ImageData`), `LaserScan` a `PointCloud2`.

Získávání dat z API

Jelikož kvůli velikosti BAG souborů nelze načíst všechna data najednou, je potřeba data načítat postupně podle potřeby. Prvotní plán byl, že frontend bude na server posílat požadavky na určitý rozsah dat daným časovými značkami zpráv a server vrátí data přesně v tomhle rozsahu, i když by při hledání zpráv v *Chunk* záznamech mohl zbytečně číst zprávy, které do rozsahu nespadají. Po prozkoumání několika BAG souborů bylo zjištěno, že *Chunk* záznamy zprávy rozdělují do příhodně velkých celků, tudíž by bylo efektivnější načítat data na frontendu po těchto celcích. Nedocházelo by tak ke čtení dat, které nakonec nebudou využity. Implementace API endpointu je pak také jednodušší, jelikož stačí z databáze získat informace o daném *Chunk* záznamu, převést tyto informace na vhodné objekty z knihovny *rosbag-pyreader* a použít je pro přečtení všech zpráv *Chunk* záznamu z BAG souboru.

Binární data ze zpráv však po rozparsování narostly na velikosti a místo např. 1,9 MB binárních dat se na frontend posílá 4,2 MB dat v JSONu. Pokud máme k serveru rychlé připojení, tak to není až tak velký problém, protože se však jedná o poměrně velké množství dat, které je potřeba přečíst ze souboru, rozparsovat na strukturovaná data, poté opět serializovat do JSONu (přitom data projdou přes různý *middleware* knihovny Django REST framework, takže tento krok zabere hodně času) a nakonec ještě dekódovat z JSONu do JavaScript objektů na frontendu, může požadavek na ně zabrat třeba i skoro minutu. To je velmi vysoký čas na načtení jednoho *Chunk* záznamu, jenž obsahuje například 0,5 sekund nahraných dat. Proto se později přešlo na frontendové parsování zpráv. To znamená, že se na serveru pouze přečtou binární data *Chunk* záznamu, pokud jsou komprimovaná, tak se dekomprimují (jelikož se mi nepodařilo zprovoznit žádnou knihovnu pro dekomprimaci Bzip2 formátu na frontendu), a pošlou se v odpovědi jako řetězec ve formátu base64. Tímto odpadne zbytečná serializace na straně serveru, která zabírala dlouhý čas. Parsování zpráv z *Chunk* záznamu a následně i dat ze samotných zpráv, jenž nezabírá tolik času, se tak přesune na frontend. Za tímto účelem měla být použita knihovna `rosbag`, která také vznikla v rámci aplikace Webviz. Bohužel se však tuto knihovnu nepodařilo v projektu zprovoznit, nejspíš kvůli knihovně Flow, kterou využívá. Jelikož má knihovna také otevřený kód, byly některé její užitečné části převzaty a přepsány do jazyku TypeScript. Část kódu byla také přenesena z vlastní Python knihovny `rosbag-pyreader`, aby byl zachován formát výstupních dat a nebylo potřeba upravovat značnou část frontendu, který byl v té době už téměř

hotový. Díky této optimalizaci se snížila doba zpracování požadavku na serveru z desítek sekund na řádově stovky milisekund.

Parsování zpráv na frontendu zabere, v závislosti na velikosti dat, něco kolem půl sekundy. Jelikož však JavaScript běží synchronně s uživatelským rozhraním, způsobí takto dlouhý výpočet jeho zaseknutí. Půl sekundy se nemusí zdát jako moc, avšak v uživatelském rozhraní je i takto krátké zaseknutí velmi poznat. Pokus o řešení tohoto problému pomocí Web Workerů[13] selhal, jelikož přenášení velkého množství dat mezi vlákny trvalo mnohem déle, než samotné parsování, a uživatelské rozhraní se tak zasekávalo ještě více. Existují sice efektivnější metody přenosu velkých dat do jiného vlákna, než využívá použitá knihovna `workerize-loader`, avšak na experimentování s touto optimalizací již nezbyl čas.

Aby bylo možné data nějakým způsobem indexovat a tedy se v nich i posouvat dopředu nebo dozadu, je nutno vědět, jaké zprávy se v kterém *Chunk* záznamu nachází. Proto je v anotačním zobrazení potřeba nejdříve načíst informace o všech *Chunk* záznamech a časová razítka všech zpráv v každém z nich, tříděných podle tématu. Index zprávy je tedy dán tématem, na kterém se zpráva nachází, indexem *Chunk* záznamu a indexem zprávy v rámci tohoto záznamu. Takovýto komplexní index není příliš vhodný pro ovládací prvky, jako je posuvník, kde je potřeba mít jako index pouhé číslo. Proto se v některých částech aplikace používá ještě globální index, který je dán jako počet všech zpráv před danou zprávou na daném tématu, bez ohledu na *Chunk* záznam, ve kterém se nachází. Mezi těmito index je potřeba umět převádět, což se dělá pomocí sumy prefixů. Tzn. že pokud chceme převést index zprávy na globální index, spočteme pro každý *Chunk* záznam globální index jeho první zprávy na daném téma tak, že tuto hodnotu vezmeme z předchozího *Chunk* záznamu a přičteme k ní počet zpráv na daném téma také z předchozího záznamu. Poté stačí vzít tento spočtený globální index z *Chunk* záznamu, do kterého zpráva spadá a přičíst k němu index zprávy v rámci tohoto záznamu.

Převod z globálního indexu na index zprávy je o něco komplikovanější. Také se spočítá suma prefixů, následně binárním vyhledáváním nalezneme, do kterého *Chunk* záznamu globální index spadá. Jelikož však *Chunk* záznamy nemusí obsahovat žádnou zprávu na daném tématu, musíme je přeskočit, abychom našli záznam, který hledanou zprávu opravdu obsahuje. Když už máme vyhledaný správný záznam, stačí odečíst od globálního indexu globální index první zprávy v záznamu, jenž byl spočítán sumou prefixů. Pak už máme všechny potřebné informace, což je index *Chunk* záznamu a index zprávy v rámci něj.

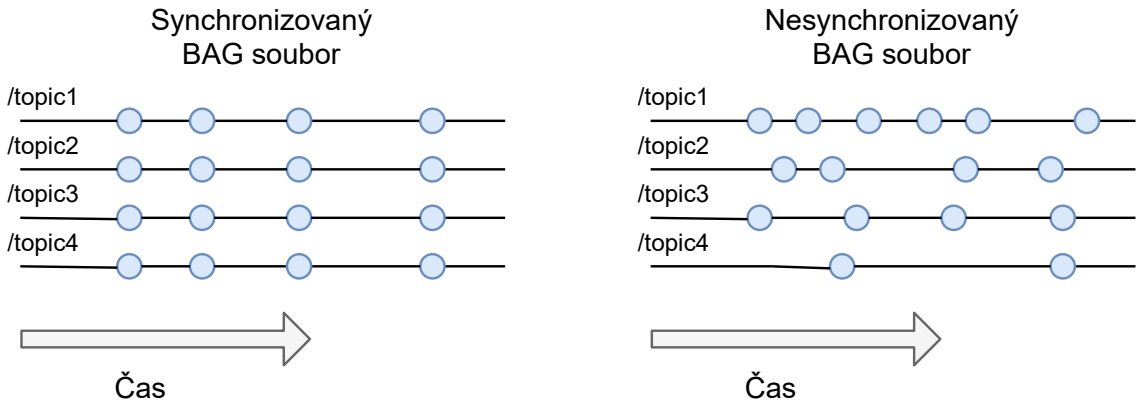
Synchronizace zpráv

Jelikož zprávy ROS systému chodí z různých senzorů, které nemusí být synchronizované, a každý z nich může mít jinou frekvenci snímaní dat, jsou i zprávy v BAG souborech nesesynchronizované. Systém ROS poskytuje knihovnu `message_filters`, která umožňuje zprávy následně i během jejich sběru synchronizovat. Nabízí dva způsoby synchronizace: *ExactTime*, kde zůstanou pouze zprávy se stejným časovým razítkem v hlavičce, a *ApproximateTime*, který zprávy i s různými časovými razítky pomocí adaptivního algoritmu[14]. Rozdíl mezi synchronizovaným a nesynchronizovaným BAG souborem lze vidět na obrázku 5.14.

Aplikace by však měla umět pracovat i s nesesynchronizovanými BAG soubory. Bohužel knihovnu `message_filters` pro tento účel není možné využít a po konzultaci s vedoucím práce byl zvolen způsob synchronizace, kde si uživatel vybere tzv. *master topic*, neboli

[13] Pomocí Web Workerů je možné spouštět kód na pozadí v separátním vlákně (více viz https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API/Using_web_workers)

[14] více viz http://wiki.ros.org/message_filters/ApproximateTime



Obrázek 5.14: Synchronizovaný BAG soubor versus nesynchronizovaný. Modré puntíky naznačují jednotlivé zprávy.

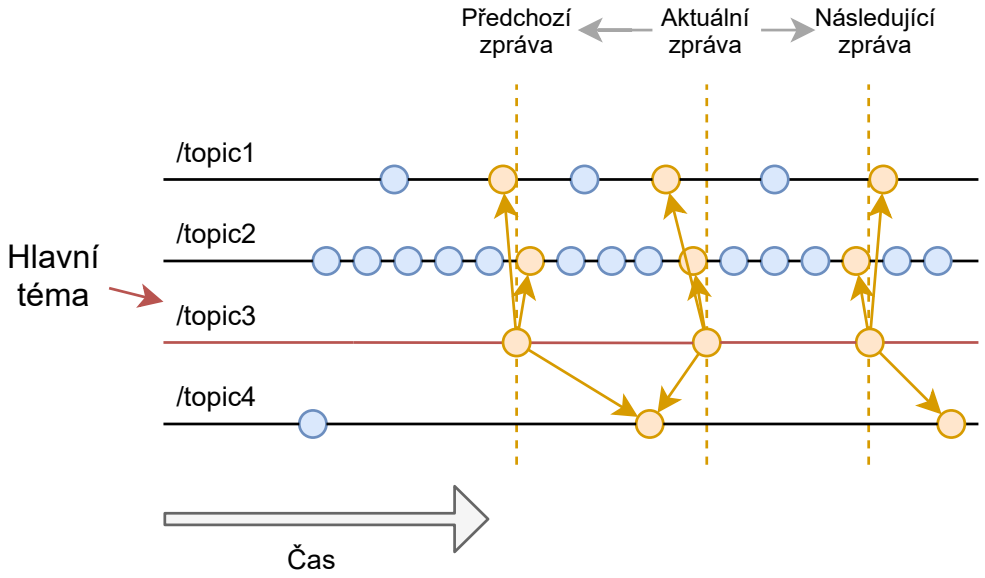
hlavní téma, které bude určovat navigaci v souboru (bude určovat časovou osu pro ovládací prvky) a ostatní zobrazené témata se budou synchronizovat k němu. Diagram naznačující, jak hlavní téma ovlivňuje navigaci a synchronizaci, lze vidět na obrázku 5.15. Z diagramu lze vypozorovat, že některé zprávy uživatel nemusí vidět vůbec a některé může vidět dokonce vícekrát. Synchronizace probíhá vždy dynamicky podle toho, z kterých témat jsou zprávy potřeba. Nejdříve se zjistí časové razítko aktuálně vybrané zprávy z hlavního tématu a následně se najde zpráva s nejbližším časovým časovým razítkem na tématu, jenž chceme synchronizovat. Abychom zprávu nemuseli hledat ve všech *Chunk* záznamech, nastaví se maximální povolený rozdíl mezi časovými značkami (aktuálně například 5 sekund), a pak stačí hledat pouze v záznamech obsahující zprávy z okolí daného tímto rozdílem. Nejbližší zpráva se pak vyhledá upraveným binárním vyhledáváním, které v bodě, kdy se vyhledává v rozsahu dvou prvků, vrátí ten, který je blíž hledané hodnotě.

Přednačítání a uvolňování dat

I přes optimalizace, popsané v předchozí podsekci, trvá načítání dat poměrně dlouho (jednotky sekund), zvlášť pokud k serveru nemáme rychlé připojení (může se prodloužit na desítky sekund). Proto je vhodné data načítat předem, aby uživatel při přechodu na další zprávu nemusel čekat na její načtení.

Prvotní implementace byla naivní a načítala pevný počet *Chunk* záznamů dopředu (testováno na počtu 3). To se však ukázalo jako neefektivní způsob, jelikož velká data, jako jsou například data typu *Point Cloud*, mají většinou nízkou frekvenci zpráv a často se stává, že mezi dvěma sousedícími zprávami bylo několik *Chunk* záznamů, která žádné zprávy tohoto typu neobsahovala. Pokud má uživatel zobrazených více dat z různých témat, která nejsou synchronizovaná, může se také stát, že se zprávy nachází v jiných *Chunk* záznamech. Kvůli těmto problém se velice často stávalo, že předem načtená data neobsahovala potřebná data a při přechodu na další zprávu se data teprve začaly načítat. Toto by se dalo vyřešit přednačítáním více *Chunk* záznamů, avšak i to není spolehlivé řešení pro všechny případy a nevyřeší to zbytečné a časově drahé načítání dat, která neobsahují užitečné zprávy.

Proto se později přešlo na implementaci, která načítá pouze *Chunk* záznamy, jež budou opravdu potřeba. Algoritmus funguje tak, že se vezme aktuální index zprávy na daném téma a najde se index následující zprávy. Poté se zjistí časové razítko této zprávy a najdou



Obrázek 5.15: Synchronizace zpráv k hlavnímu tématu. Oranžové šipky ukazují, které zprávy budou zobrazené spolu se zprávou z hlavního tématu.

se nejbližší zprávy na aktuálně zobrazených tématech (stejným způsobem, jako probíhá synchronizace zpráv popsána v podsekci výše). Pro všechny tyto zprávy se následně zjistí ID jejich *Chunk* záznamu, který se má načíst. Všechny takto vyhledané záznamy se načtou, pokud ještě nejsou načtené nebo se aktuálně nenačítají. Jelikož si takto dopředu zjistíme, které zprávy budou jako další v pořadí zobrazeny, budeme načítat *Chunk* záznamy, které jsou opravdu potřeba, a pokud se stihnou načíst, než uživatel přejde na další zprávu, tak už se nebude muset na žádné načítání dat čekat. To by v případě anotace dat neměl být problém, jelikož dokončení anotace jednoho snímku nějaký čas zabere a mezitím má aplikace čas načíst další data.

Jak již bylo zmíněno, kvůli velikosti BAG souborů nemůžeme mít celý soubor najednou načtený u klienta. Postupné načítání dat však tento problém úplně nevyřeší, jelikož pokud by uživatel postupně procházel soubor, po čase by mu došla paměť RAM. Proto je data potřeba i postupně uvolňovat. K tomu bylo vytvořené rozhraní *ChunkUnloadingPolicy*, které definuje následující metody:

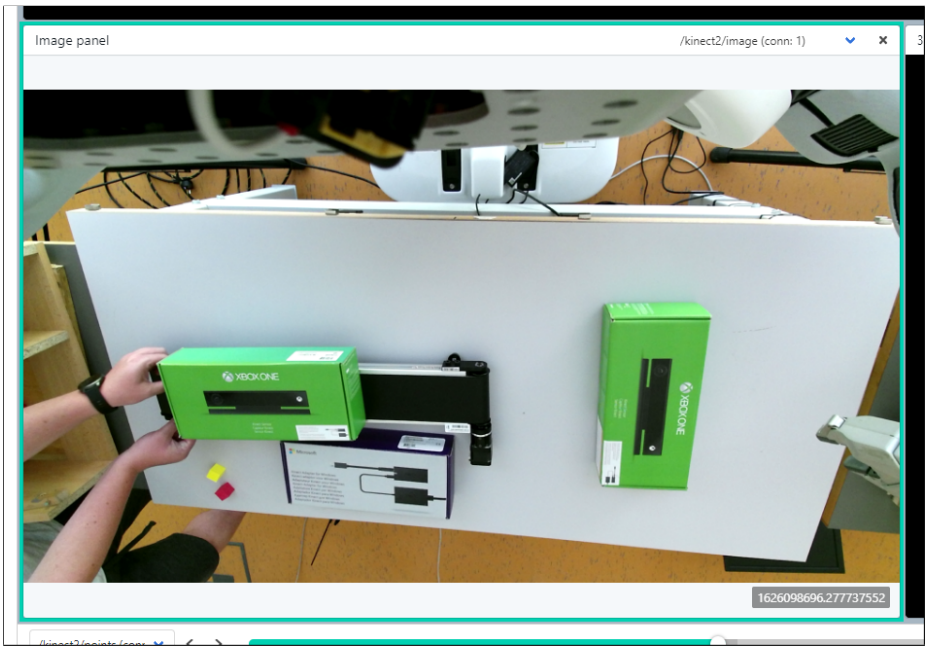
- `getChunksToUnload` – vrátí seznam ID *Chunk* záznamů pro uvolnění,
- `onChunkLoaded` – tato metoda je zavolána při načtení určitého *Chunk* záznamu, jehož ID je předáno parametrem,
- `onChunkAccess` – tato metoda je zavolána při zobrazení zprávy z určitého *Chunk* záznamu, jehož ID je předámo parametrem.

Konkrétní implementace je pak dosazena do konfigurace aplikace v souboru `Config.ts`. Tuto instanci pak využívá komponenta `ChunkUnloader`, která při změně v počtu načtených *Chunk* záznamů zavolá metodu `getChunksToUnload` a vrácené záznamy uvolní z paměti. Zatím jedinou implementací tohoto rozhraní je třída `LruChunkUnloadingPolicy`, jenž implementuje algoritmus LRU (*Least Recently Used*). Funguje tak, že při načtení *Chunk* záznamu si jeho ID uloží na konec pole a při přístupu na něj ho z tohoto pole odstraní

a vloží opět na konec. Takto budou záznamy seřazeny podle toho, kdy byly naposledy použity, s tím, že na konci pole je ten nejpozději použitý. V metodě `getChunksToUnload` tedy nejdříve zkontroluje jestli počet načtených *Chunk* záznamů přesáhl daný limit (předán parametrem konstruktoru) a pokud ano, tak z pole odstraní určitý počet záznamů (tento počet je také předán parametrem konstruktoru) a odstraněné položky z metody vrátí.

Zobrazování obrazových dat

Pro zobrazení obrazových dat slouží *Image panel*. Používá pro vykreslení obrázku a anotací knihovnu `react-konva`[15], která usnadňuje práci s HTML5 Canvas, zvláště v oblasti interakce s vykreslenými objekty. Obrazová data ze zpráv se nejdříve převedou na objekt, který lze vykreslit, což je například `ImageBitmap`. Tento kód pro převod byl převzat z již zmíněného nástroje Webviz. Následně stačilo tento objekt předat komponentě `Image` z knihovny `react-konva`, která jej vykreslí. Ukázku panelu lze vidět na obrázku 5.16.



Obrázek 5.16: Zobrazený obrázek v *Image panelu* (zelený okraj naznačuje, že je panel aktuálně vybraný)

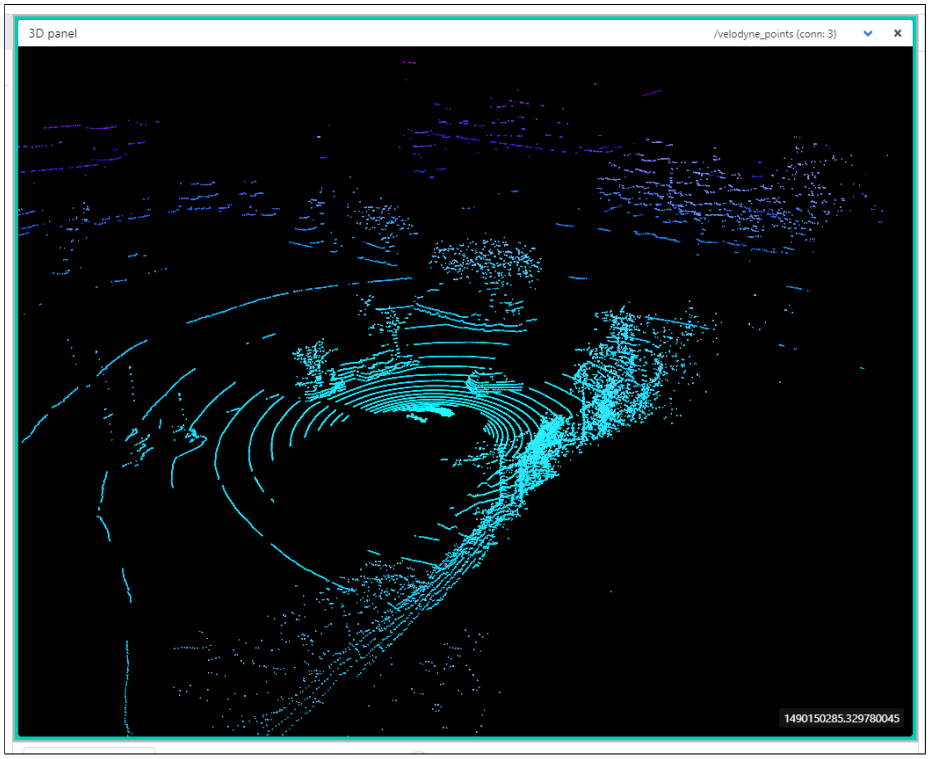
Panel také automaticky vykreslený obraz vycentruje a upraví přiblížení obrázku tak, aby využil co nejvíce dostupného místa. Byla také přidána podpora přiblížení a oddálení pomocí kolečka myši, což je důležité pro anotaci menších objektů. Obraz jde samozřejmě také posouvat potažením myši, což je už implementované v knihovně `react-konva` a pro zapnutí této funkce stačí nastavit *prop* `draggable={true}` komponentě `Stage`.

Zobrazování 3D dat

Pro zobrazení dat typu `sensor_msgs/PointCloud2` nebo `sensor_msgs/LaserScan` slouží *3D panel*. I pro tento panel bylo využito kódu z nástroje Webviz. Konkrétně knihovna `regl-worldview`, jenž obaluje knihovnu `regl`, pro vykreslování 3D scén pomocí WebGL,

[15] https://github.com/konvajs/react-konva

do komponent pro snadnější použití s knihovnou React. Také umožňuje vykreslovat různé základní objekty, jako jsou čáry, šipky, válce, text a tak podobně, čehož bylo využito při vytváření anotací. Dále byly převzaty komponenty `LaserScans` a `PointClouds`, pro vykreslování dat ze zpráv pomocí zmíněné knihovny `regl-worldview`, a pomocné funkce k těmto komponentám. Tyto komponenty vykreslují data pomocí vertex shaderu a fragment shaderu. Kód převzatých komponent musel být přizpůsoben aplikaci a upraven tak, aby fungoval v jazyce TypeScript. U zpráv typu *Point Cloud* jsou data ještě před vykreslením transformována na vhodnou strukturu, což však u zpráv s velkým množstvím bodů může trvat až například sekundu. To opět není příliš optimální pro uživatelské rozhraní, které se na tuto dobu zasekne. Bohužel toto nejde jednoduše vyřešit pomocí Web Workerů, protože je zde zase problém s přenášením velkého množství dat mezi vlákny.



Obrázek 5.17: Zobrazená data v *3D panelu* s obarvenými body

Jednotlivé body v *Point Cloud* zprávách je potřeba také obarvit podle vzdálenosti od počátku a jejich výšky, aby se ve vykreslené scéně dalo lépe vyznat. To bylo implementováno tak, že se při zmíněné transformaci dat zjistí největší horizontální souřadnice (X nebo Y) a také nejnižší bod a nejvyšší bod. Poté se ve vertex shaderu spočítá barva pro každý bod na základě souřadnic a extrémů zjištěných při transformaci a to tak, že se spočítá vzdálenost bodu od počátku a podělí se hodnotou největší horizontální souřadnice a tato hodnota se omezí na rozsah 0 až 1. To ovlivňuje zelenou složku barvy. Červenou složku barvy ovlivňuje výška, která se spočítá podělením výškové souřadnice bodu rozdílem maximální výšky a minimální výšky. Modrá složka barvy má vždy hodnotu 1. Výsledek lze vidět na obrázku 5.17.

5.8 Anotace

Tato sekce se zabývá implementací anotací. Jelikož druhů anotací je hodně (např. obdélník, polygon, body atd.), jsou udělány rozšiřitelně. Každý druh anotace se skládá z unikátního identifikátoru (řetězec), komponenty, která ji zobrazuje, a nástroje pro její vytváření. Tento nástroj je také komponentou, která na základě uživatelského vstupu anotaci vytváří a také renderuje náhled anotace během vytváření. Anotace musí být registrovány ve třídě *AnnotationTypeRegistry* specifikováním identifikátoru, zobrazitelného názvu, komponenty pro zobrazení anotace a identifikátoru a komponenty nástroje pro její vytváření.

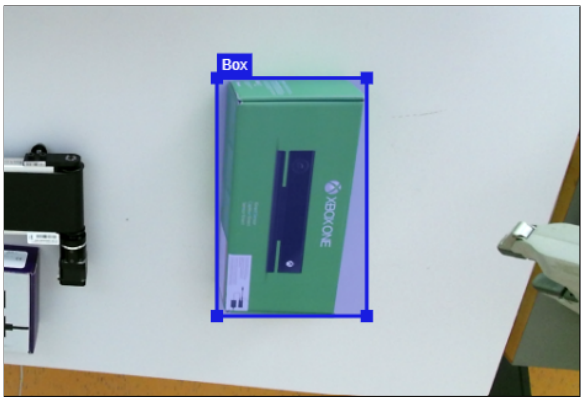
Anotace se načítají pouze pro aktuální zobrazené zprávy a všechny změny se automaticky ukládají na pozadí. Uživatel tedy nemusí explicitně ukládat provedené změny a nemusí se bát, že ztratí svoji práci například při výpadku proudu. Nevýhodou však je, že se anotace načítají až při zobrazení zprávy, a vzniká zde prodleva, kdy uživatel při přechodu na zprávu nevidí její anotace dokud se nenačtou ze serveru. V budoucnu by bylo vhodné anotace načítat předem, podobně jako data z BAG souboru.

Protože jsou anotace ukládány do globálního Redux *storu*, jsou data, definující pozici a tvar anotace, zrcadleny i do lokálního stavu komponenty pro zobrazení, aby při editaci nedocházelo k příliš častému aktualizování Redux *storu*, což by mohlo způsobit zpomalení uživatelského rozhraní. Proto se při editaci anotace mění pouze lokální stav a globální se aktualizuje pouze až poté, co uplyne 200 milisekund od poslední změny lokálního stavu. Ve stejnou chvíli se změny v anotaci uloží i na server.

Anotace jsou také zobrazovány v bočním panelu anotačního zobrazení, jak lze vidět na obrázku 5.10. V tomto panelu jim lze přiřazovat štítky, které určují o jaký objekt se jedná a podle něj mají také anotace barvu. Pokud anotace nemá přiřazený štítek, je barva anotace šedá a název štítku *Unknown*. Barva uživateli pomáhá rychleji se orientovat v tom, který objekt anotace označuje. Při najetí myší na některou z anotací v bočním panelu se anotace zvýrazní nejen zde, ale i v příslušném panelu, aby bylo poznat, kde se nachází.

2D anotace

V *Image panelu* jsou anotace vykreslovány stejně, jako obrazová data z BAG souboru, pomocí knihovny `react-konva`. Tato knihovna umožňuje vykreslovat ve více vrstvách pomocí komponenty *Layer*. Vrstva je samostatný HTML5 Canvas a tudíž se vykresluje každá zvlášť. Vykreslování obrazových dat a anotací probíhá v samostatných vrstvách, jelikož anotace se překreslují mnohem častěji a není při tom třeba překreslovat obraz na pozadí.



Obrázek 5.18: Obdélníková anotace v *Image panelu* ve zvýrazněném stavu.

Pro 2D data existuje v aplikaci prozatím jenom jedna anotace a tou je obdélníková. Lze ji vidět na obrázku 5.18. Zobrazení této anotace zprostředkovává komponenta *RectAnnotationView*. Ta využívá komponentu *Group* knihovny `react-konva` pro seskupení všech vykreslených komponent, v rámci této anotace. V této skupině lze pak manipulovat se všemi podřazenými objekty zároveň. To se používá hlavně pro posouvání anotace potáhnutím myši nebo pro mazání anotace, jelikož je možno zpracovávat i události pro celou skupinu. Tloušťka čar anotace a velikost textů pro vykreslení anotací je nezávislá na přiblížení, aby při anotaci menších objektů nebránila anotace uživateli ve výhledu.

Byla také vytvořena pomocná komponenta *ControlPoint*, jenž usnadňuje vykreslování a pohyb kontrolních bodů pro editaci tvaru anotace. U obdélníkové anotace se tyto kontrolní body zobrazují v rozích při zvýraznění anotace (najetím myši na anotaci přímo v *Image panelu* nebo v bočním seznamu anotací) a lze je vidět na obrázku 5.18. Potáhnutím těchto bodů lze upravovat velikost obdélníku. Posunout anotaci lze kliknutím a táhnutím anotace kdekoliv, kromě kontrolních bodů.

Další pomocná komponenta je *LabelView*, jenž slouží pro zobrazení názvu přiřazeného štítku. Na obrázku 5.18 se jedná o nápis „Box“. Aby byl nápis dobře vidět na pozadí v barvě štítku, je potřeba vybrat kontrastní barvu. K tomu byla využita knihovna `invert-color`, která podle dané barvy pozadí umí zvolit barvu textu (černou nebo bílou), tak aby byl čitelný.

Vytváření obdélníkových anotací má na starosti komponenta *RectTool*. Využívá *hooku* `useHandlers`, který předané obslužné funkce zaregistruje pro příslušné události objektu *stage* a před voláním obslužné funkce přepočítá pozici myši do souřadnicového systému obrázku za použití transformační matice. Tento objekt se získá z komponenty *Stage*, jenž se stará o vykreslování i zachytávání událostí ze všech vrstev, pomocí reference `ref`[16]. Reference je pak propagována do podřízených komponent kontextem *ImageViewContext*. *Hooku* `useHandlers` lze předat tři obslužné funkce:

- `onCreatePoint` — volána při kliknutí levým tlačítkem myši,
- `onEndCreation` — volána při kliknutí pravým tlačítkem myši,
- `onMouseMove` — volána při pohybu myši.

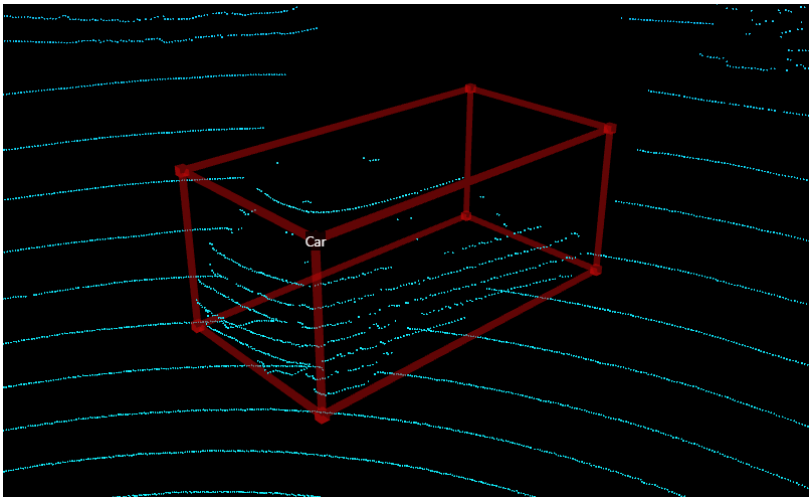
Obdélníková anotace je určena dvěma body. Při prvním volání `onCreatePoint` se pozice myši uloží jako první bod obdélníku. Při volání `onMouseMove` se aktualizuje druhý bod, aby bylo možné vykreslovat během vytváření náhled anotace. Druhým voláním `onCreatePoint` se vytváření obdélníku dokončí, aktuální pozice myši se uloží jako druhý bod a anotace se vytvoří voláním API endpointu. Nakonec se změní aktuální nástroj zpátky na výchozí *Pointer*. Volání obslužné funkce `onEndCreation`, zresetuje vytváření anotace.

3D anotace

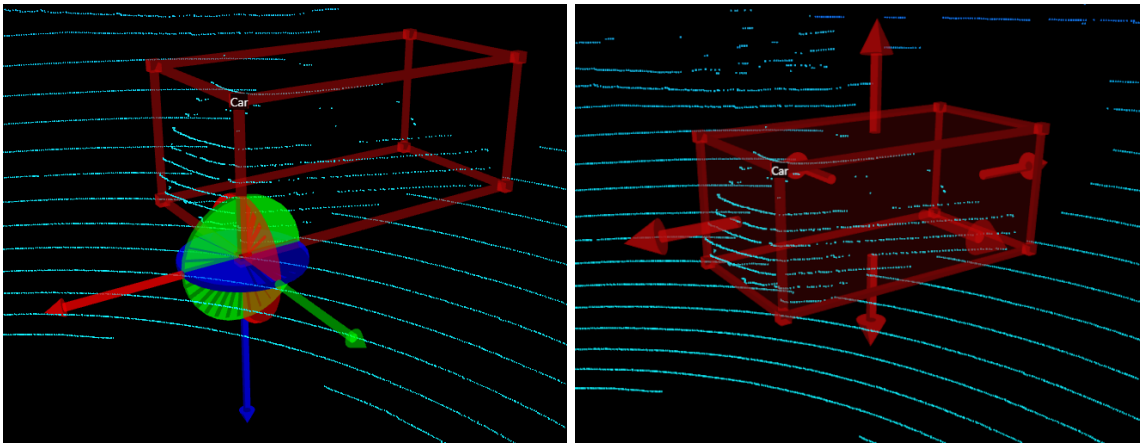
V *3D panelu* je implementována anotace s tvarem kvádru. Lze ji vidět na obrázku 5.19. Kliknutím na anotaci se vybere a zobrazí se ovládací prvky pro pohyb a otáčení anotací, což je jeden ze stavů, ve kterém se vybraná anotace může nacházet. Do druhého stavu se přejde opětovným kliknutím na vybranou anotaci a v tomto stavu se zobrazí šipky kolmé k jednotlivým stěnám kvádru. Pomocí těchto šipek, lze stěny kvádru posouvat v daném směru a tím zmenšovat, či zvětšovat anotaci. Oba tyto stavy je vidět na obrázku 5.20. Každé

[16] https://reactjs.org/docs/refs-and-the-dom.html

další kliknutí na anotaci přepíná mezi stavem pro úpravu pozice a pro úpravu velikosti. Výběr anotace lze zrušit kliknutím mimo ní.



Obrázek 5.19: 3D anotace ve tvaru kvádru.

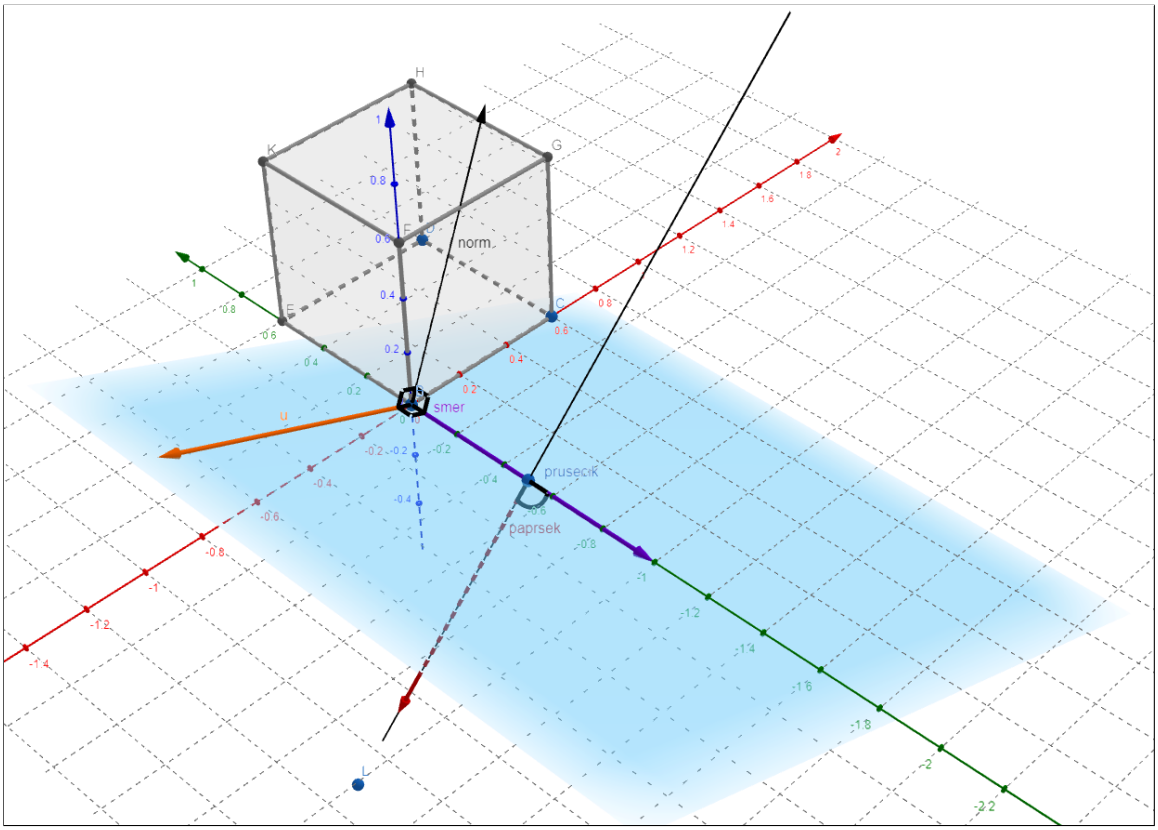


Obrázek 5.20: 3D anotace ve stavu úpravy pozice nalevo a ve stavu úpravy velikosti napravo.

Anotací se pohybuje pomocí barevných šipek (viz obrázek 5.20), kde zelenou šipkou se pohybuje po směru lokální osy X, červenou po směru lokální osy Y a modrou po směru lokální osy Z. Takto je pohyb anotace předvídatelný, jelikož kdyby se anotací pohybovalo podobně, jako ve 2D (kliknutím a tažením myši kdekoliv na anotaci), bylo by těžké určit, po které rovině má uživatel v úmyslu anotaci posunout.

Rotace anotace probíhá podobně, jako její pohyb, pomocí barevných kotoučů (opět viz obrázek 5.20). Stejně tak barvy souvisí s lokální osou, po které se má anotace otočit. Kliknutím na jeden z kotoučů se vybere osa otáčení a následné tažení anotaci postupně otáčí. Úhel otočení je závislý na vzdáleností od počátečního bodu táhnutí a také jeho směru.

Anotace jsou stejně jako 3D data vykreslovány knihovnou `regl-worldview`. Ta však nepodporuje seskupování vykreslených objektů, jako je tomu u knihovny `react-konva`. Proto byla vytvořena knihovna *WorldviewTransforms*, která přidává podporu hierarchie transformací pro knihovnu `regl-worldview`, a její implementace je popsána v sekci 5.9.



Obrázek 5.21: Geometrické zobrazení roviny (světle modrá plocha) vůči paprsku myši (červený vektor *paprsek*). Krychle naznačuje anotaci, fialový vektor *smer* je vektor ukazující směr šipky.

Hierarchie transformací je využita v komponentě *CuboidAnnotationView*, která vykresluje anotaci ve tvaru kvádru. Díky toho může být složena z více různých 3D objektů, kterým stačí zadat pozici a orientaci relativně vůči lokálnímu souřadnicovému systému anotace.

Pro manipulaci s anotací se používá táhnutí myší. Knihovna `regl-worldview` sice podporuje události myši na jednotlivých vykreslených objektech a dokonce převádí 2D souřadnice myši na 3D paprsek, který reprezentuje její pozici v prostoru, avšak události pro táhnutí myši nepodporuje. Proto bylo potřeba tuto funkci implementovat.

Při posunutí anotace uživatel nejdříve najede myší na šipku a stiskne levé tlačítko myši. To vyvolá událost `onMouseDown` a v obsluze této události se prvně zjistí rovina, ve které bude tažení myši probíhat, a následně její průsečík s paprskem myši. Tento průsečík udává počáteční bod táhnutí a je uchován v lokálním stavu spolu s rovinou a vektorem šipky, který bude udávat směr pohybu s anotací. Zmíněnou rovinu lze vidět v grafu 5.21. Je reprezentována bodem, ze kterého šipka vychází (v grafu je to počátek souřadnicového systému) a normálovým vektorem (v grafu vektor *norm*). Je vypočítána vektorovým součinem vektoru *paprsek* a vektoru *smer*, což nám dá vektor u kolmý k obou těmto vektorům. Pro vektor u a vektor šipky *smer* opět spočítáme vektorový součin, čímž získáme normálový vektor *norm* k požadované rovině.

Následně uživatel myší táhne, stále se stisknutým levým tlačítkem, což vyvolává události `onMouseMove`. V obsluze události se opět spočítá průsečík uložené roviny a aktuálního paprsku myši. Od tohoto průsečíku se odečte počáteční průsečík, který je uložen v lokálním

stavu, čímž je získán vektor aktuálního tahu myší, označme jej *tah*. Skalárním součinem vektoru *tah* a jednotkového vektoru *smer* (viz graf 5.21) a následným vynásobením jednotkového vektoru *smer* tímto skalárním součinem, získáme vektor udávající délku tahu ve směru šipky. Tento vektor uložíme do lokálního stavu a při vykreslování anotace tento vektor připočteme k aktuální pozici, aby uživatel viděl její pohyb ihned během tahu myší. Reálná pozice anotace se aktualizuje teprve až uživatel pustí levé tlačítko myši.

Stejným způsobem funguje změna velikosti anotace. U rotace je rozdíl v tom, že se nepoužívá paprsek myši ani rovina, ale pracuje se pouze s 2D souřadnicemi myši. Na začátku se uloží počáteční poloha myši na obrazovce a při táhnutí se spočítá 2D vektor tahu odečtením počáteční polohy a aktuální polohy myši. Úhel otočení je pak spočítán součtem složek X a Y tohoto vektoru a podělením konstantou 200. Tento výpočet úhlu umožní zápornou rotaci a uživatel si může vybrat jestli bude táhnout horizontálně nebo vertikálně. Kdybychom místo toho použili délku vektoru, úhel by byl vždycky kladný a uživatel by mohl anotaci otáčet pouze na jednu stranu.

Vytváření anotace má na starost komponenta *CuboidTool*. Anotaci lze vytvořit buď pouhým kliknutím, kdy kvádr bude mít danou výchozí velikost, nebo potáhnutím myši. Při potáhnutí myši se uživateli zobrazí podstava kvádru, pro náhled jeho velikosti a otočení. Uživatel tak určí jednu hranu kvádru a zbytek rozměrů se vypočítá ve výchozím poměru k této hraně. To slouží k tomu, aby uživatel mohl určit počáteční otočení kvádru a jeho přibližnou velikost. Při vytváření 3D anotací je však problém se zjištění přesné polohy, kde uživatel chce anotaci umístit. Protože souřadnice myši jsou pouze 2D, chybí nám informace o „hloubce“ kliknutí. Místo konkrétního bodu, musíme pracovat s celou přímkou, která reprezentuje pozici myši (tzv. paprsek).

Jedním řešením je, že by uživatel musel klikat na konkrétní vykreslené body reprezentující data. Ačkoliv knihovna `regl-worldview` podporuje události na vlastních 3D objektech, kvůli množství bodů v *Point Cloud* datech jejich zpracování aplikaci velmi zpomalovalo. Také by to nemuselo být intuitivní pro uživatele, protože by nemohl začít vytvářet anotaci kdekoliv, ale pouze kliknutím na konkrétní bod.

Další možné řešení by bylo data rozdělit do datové struktury *Quadtree*, ve které by se pak podle paprsku vyhledaly okolní body a z nich se určila nejpravděpodobnější *hloubka* kliknutí. Sestavení stromové struktury *Quadtree* z velkého množství dat by však nejspíš bylo také časově náročné. Z tohoto důvodu, i kvůli složitosti implementace a nejistých výsledků, se toto řešení zavrhnulo. Nakonec bylo zvoleno nejjednodušší řešení, kdy se anotace umístí do průsečíku paprsku a roviny dané osami X a Y (tedy bod na paprsku s výškovou souřadnicí Z rovnající se nule). To je ve většině případů dostačující, jelikož umístění senzoru je v počátku souřadnicového systému, a pokud senzor snímá horizontálně, je i většina objektů v podobné výšce.

Manuální sledování objektu

Jelikož nástroj bude většinou sloužit k anotací záznamů, kde se objekty po scéně pohybují a není příliš velký rozdíl mezi předchozí a následující zprávou, byl implementován také mód manuálního sledování objektu, kdy se anotace z jedné zprávy promítne do všech následujících zpráv a uživateli pak již jenom stačí provést malé úpravy v každém dalším snímku, aby anotace kopírovala pohyb objektu.

Každá anotace je po vytvoření typu *SINGLE* — vyskytuje se pouze v jedné zprávě. Tlačítkem s ikonou zaměřovače v kartě anotace (viz obrázek 5.22) lze anotaci přepnout na typ *TRACK*. To způsobí, že se vytvoří implicitní klíčový snímek anotace na poslední zprávě

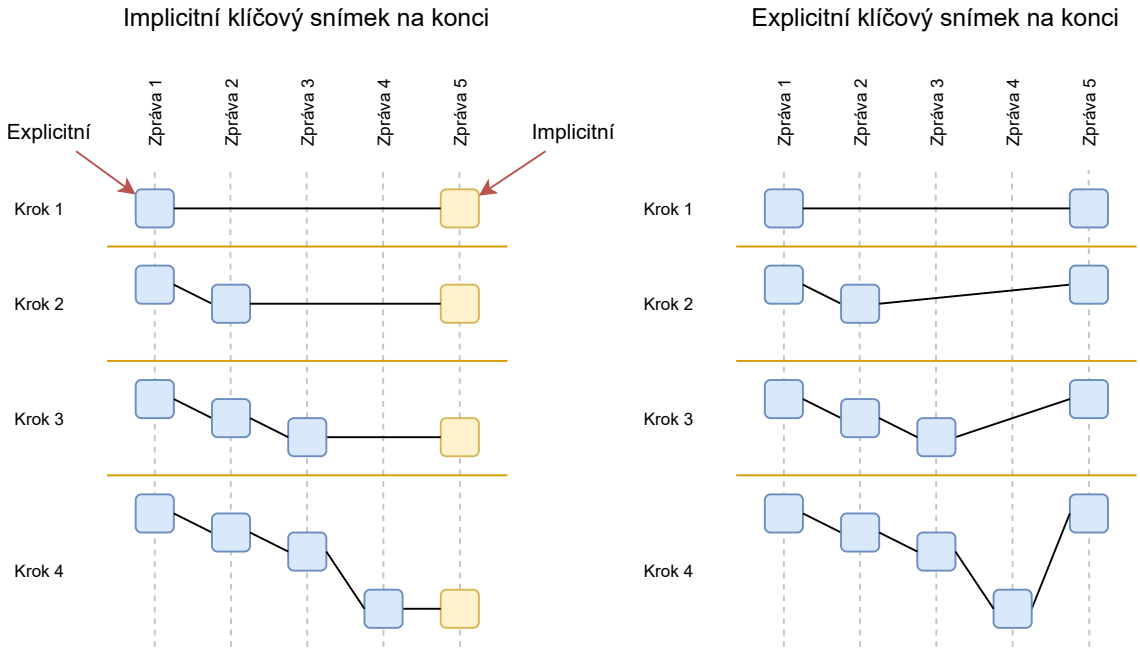
na daném téma a adekvátně se upraví i časový rozsah dané anotace. Tvar a pozice anotace se pak na jednotlivých zprávách v daném časovém rozmezí interpolují mezi nejbližším předcházejícím a následujícím klíčovým snímkem. Klíčový snímek se automaticky vytvoří při úpravě jakékoliv interpolované anotace.

Obrázek 5.22: Karta anotace s typem *SINGLE* v pravém panelu anotační obrazovky.

Implicitní klíčový snímek je speciální v tom, že při úpravě tvaru či pozice klíčového snímku, jenž právě předchází implicitnímu, se tyto změny propíšou i do něj. Kdyby tomu tak nebylo a uživatel by změnil typ anotace na sledování objektu a následně pokračoval po jednotlivých zprávách dále a postupně upravoval anotaci tak, aby sledovaný objekt pokrývala, anotace by se vždy při přechodu na další snímek opět interpolovala k poslednímu klíčovému snímku, který by však byl stále na originální pozici. Toto je znázorněno na diagramu 5.23. Implicitní klíčový snímek se tedy chová, jako kdyby neexistoval, avšak usnadní nám to mírně implementaci, jelikož můžeme vždy spoléhat na to, že na konci anotace je vždy klíčový snímek. Při úpravě anotace však musíme vždy zkontrolovat, jestli není potřeba upravit i implicitní klíčový snímek. Při úpravě implicitního klíčového snímku se z něj stane explicitní a již se žádné změny do něj nepropisují.

Interpolace mezi snímky je ve výchozím stavu zapnutá, avšak dá se také pro danou anotaci vypnout a v takovém případě se pozice a tvar anotace mezi klíčovými snímky nebude přepočítávat, ale vezmou se hodnoty z nejbližšího předcházejícího klíčového snímku. Přepínač tohoto stavu se nachází v kartě s anotací v pravém panelu (viz snímek 5.24). Toto je připraveno pro typy anotací, které by interpolaci nemuseli podporovat (například interpolace polygonu s proměnlivým počtem vrcholů je poměrně složitá na implementaci), a v takovém případě by interpolace byla ve výchozím stavu vypnutá a nešla zapnout. Může se to také hodit v případě, že by uživateli vadilo, že úprava anotace ovlivní předchozí neklíčové snímky, které již navštívil a nebylo je potřeba upravovat.

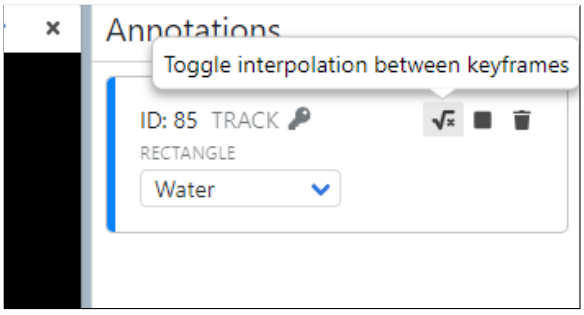
Znalejším uživatelům, kteří ví, že se nějaký objekt bude pohybovat po nějakou dobu přibližně lineárně, pomůže interpolace tento pohyb anotovat, aniž by musel uživatel jít snímek po snímku a v každém anotaci upravit, aby kopírovala pohyb objektu. Stačí najít přímo konec tohoto pohybu a vytvořit klíčový snímek až v této zprávě. Ulehčí si tak práci s anotací objektu po celou jeho dráhu pohybu, jelikož interpolace zajistí přepočet pozice a tvaru anotace ve zprávách mezi klíčovými snímky. Je však potřeba interpolovanou anotaci trochu zkontrolovat a místy poupravit, jelikož pohyb objektů není většinou perfektně lineární. Interpolace se ale také hodí i pro anotaci nesynchronizovaných dat. Taková data totiž mívají různou frekvenci zpráv a pokud nastavíme jako hlavní téma takové, že bude mít nižší frekvenci (často například data typu *PointCloud*) a budeme anotovat téma s vyšší frekvencí zpráv (např. obrazová data z kamery v 60FPS), tak při navigaci po snímcích bu-



Obrázek 5.23: Rozdíl mezi explicitním a implicitní klíčovým snímkem na konci anotace. Jednotlivé kroky ukazují, co se děje s implicitním klíčovým snímkem během toho, co uživatel postupně upravuje anotaci. Čáry mezi snímky naznačují interpolaci mezi nimi.

deme přeskakovat spoustu zpráv z tohoto tématu (viz diagram 5.15). Díky interpolaci se však anotace přepočítá i do zpráv, které uživatel neuvidí.

Sledování objektu lze také ukončit. To se hodí pro případ, kdy sledovaný objekt zmizí ze záběru senzoru a už jej tedy nepotřebujeme dále anotovat. Ukončení se provádí kliknutím na čtvercovou ikonu v kartě anotace, což lze také vidět na obrázku 5.24. Po kliknutí se do databáze vloží klíčový snímek v čase ukončení, smažou se všechny klíčové snímky s časem pozdějším než čas ukončení a upraví se časové rozmezí anotace. To však znamená, že anotace zmizí až na další zprávě.



Obrázek 5.24: Karta anotace s typem *TRACK* v pravém panelu anotační obrazovky. Vyznačen přepínač pro zapnutí/vypnutí interpolace. Ikonka klíče naznačuje, že se jedná o klíčový snímek.

Backend

Aby aplikace mohla podporovat sledování objektu, musíme anotace ukládat do dvou databázových tabulek: *Annotation* a *AnnotationFrame* (viz diagram 4.6). Tabulka *Annotation* obsahuje informace, které se mezi jednotlivými zprávami nemění:

- `type` — typ anotace. Může být:
  - *SINGLE* — anotace se vyskytuje pouze v jedné zprávě,
  - *TRACK* — anotace se vyskytuje v rozsahu zpráv (sledování objektu).
- `label` — přiřazený štítek,
- `topic` — téma, kterému anotace náleží,
- `shape_type` — řetězec identifikující typ tvaru anotace (např. obdélník, kvádr, polygon atd.),
- `interpolate` — příznak určující, zda se má tvar anotace interpolovat mezi klíčovými snímky,
- `time_from` — časové razítko zprávy, na které anotace začíná,
- `time_to` — časové razítko zprávy, na které anotace končí.

AnnotationFrame pak obsahuje klíčové snímky, v nichž jsou data pro konkrétní zprávu. V případě, že anotace je typu *SINGLE*, tak má pouze jeden klíčový snímek. Pokud je typu *TRACK* (mód sledování objektu), tak je v tabulce *AnnotationFrame* i více klíčových snímků (viz podsekce Sledování objektu). Záznam tabulky obsahuje:

- `time` — časové razítko zprávy, na které se tento snímek anotace nachází,
- `shape_data` — data definující tvar a pozici anotace ve formátu JSON,
- `is_implicit` — příznak určující, zda je daný klíčový snímek implicitní (více viz podsekce Manuální sledování objektu).

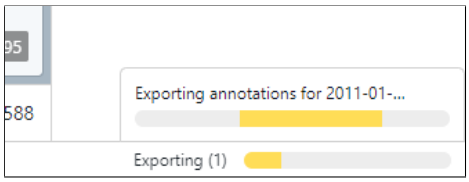
Při dotazu na seznam anotací pro určité téma v daném čase se na backendu najdou všechny záznamy *Annotation*, které jsou z daného tématu a existují v daném čase (na základě `time_form` a `time_to`). Následně se pro každý tento záznam zavolá pomocná funkce `get_or_interpolate_frame`, jenž pro daný čas buď najde klíčový snímek, pokud v daném čase existuje, nebo najde nejbližší klíčový snímek před a za tímto časem a lineárně interpoluje tvar anotace mezi nimi. Jestliže pro daný typ anotace není definována interpolační funkce, nebo položka `interpolate` je *false*, tak se místo interpolace vezme tvar anotace z nejbližšího klíčového snímku před daným časem. Interpolační funkce se definují v souboru `settings.py` s Django konfigurací v položce `SHAPE_INTERPOLATORS`, což je slovník, kde klíčem jsou identifikátory tvarů anotací a hodnoty jsou funkce, jenž berou jako první dva parametry položky `shape_data` nejbližších klíčových snímků a jako třetí parametr číslo mezi 0 a 1 určující o kolik se má interpolovat.

Při aktualizaci anotace je na backendu potřeba řešit několik situací. Pokud se změní položka `type` ze *SINGLE* na *TRACK*, je potřeba najít časové razítko poslední zprávy na daném tématu a následně vytvořit implicitní klíčový snímek s tímto časovým razítkem.

V opačném případě, kdy se `type` změní z *TRACK* na *SINGLE*, stačí pouze smazat všechny klíčové snímky kromě počátečního. Pokud se při aktualizaci změní některá z položek v *AnnotationFrame* mohou nastat dvě situace: jedná se o interpolovaný snímek nebo o klíčový snímek. V případě, že je interpolovaný, vytvoří se v daném čase nový klíčový snímek, jinak se pouze aktualizuje již existující. V obou případech je ještě potřeba aktualizovat implicitní klíčový snímek, jestliže existuje a nachází se hned za aktuálním klíčovým snímkem. Jeho aktualizace spočívá ve zkopírování tvaru anotace z aktuálního snímku do implicitního.

Export anotací

Anotace lze exportovat tlačítkem *Export annotations* v horním menu anotačního zobrazení. Po stisknutí tohoto tlačítka, podobně jako při nahrávání souborů, se objeví ve spodní liště ukazatel probíhajících exportů a kliknutím lze zobrazit celý seznam, což jde vidět na obrázku 5.25. Také je možné anotační zobrazení opustit a navigovat kdekoliv po aplikaci aniž by se musel bát, že se exporty zruší. Pokud by se v průběhu exportů pokusil aplikaci opustit úplně, prohlížeč uživatele upozorní, že může dojít ke ztrátě dat.



Obrázek 5.25: Ukazatel probíhajících exportů ve spodní liště aplikace.

Export anotací může trvat dlouhý čas, jelikož pro anotace typu *TRACK* je potřeba vygenerovat a případně interpolovat záznamy `AnnotationFrame` pro všechny zprávy mezi klíčovými snímky. K tomu je opět využitá funkce `get_or_interpolate_frame`, která je volána s časovými značkami všech zpráv, které se nachází v rozmezí `time_from` a `time_to` dané anotace. Výsledný export je ve formátu JSON a ukázku jeho struktury lze vidět ve výpisu 5.3. Kromě anotací obsahuje také informace o BAG souboru, kde jsou mimo jiné uvedeny i všechny štítky a témata.

```
{
    "annotations": [
        {
            "id": 101,
            "label": "Water",
            "connId": 1,
            "frames": [
                {
                    "id": 163,
                    "time": "1490150285.34015687600",
                    "shapeData": {
                        "dim": {
                            "width": 264.75037821482607,
                            "height": 146.142208774584
                        },
                        "pos": {
                            "x": 368.53252647503786,
```

```
                        "y": 208.3449319213314
                    }
                },
                "keyframe": true
            }
        ],
        "timeFrom": "1490150285.34015687600",
        "timeTo": "1490150285.34015687600",
        "type": "SINGLE",
        "shapeType": "ANNOTATION_RECT",
        "interpolated": true
    }
],
"bag": {
    "id": 87,
    "name": "demo.bag",
    "labels": [
        { "id": 4, "name": "Water", "color": "#0081ff" },
        { "id": 5, "name": "Car", "color": "#c30a0a" }
    ],
    "topics": [
        {
            "id": 243,
            "connId": 3,
            "name": "/velodyne_points",
            "srcTopic": "/velodyne_points",
            "type": "sensor_msgs/PointCloud2",
            "msgDefinition": {
                ...
            },
            "md5sum": "2d3a8cd499b9b4a0249fb98fd05cfa48",
            "totalMsgs": 30
        }
    ]
}
}
```

Výpis 5.3: Ukázka struktury exportu anotací

5.9 WorldviewTransforms

Knihovna `regl-worldview` pro vykreslování 3D objektů nepodporuje hierarchií transformací, aby bylo možné vykreslené objekty seskupovat a určovat jejich pozici a otočení relativně vůči nadřazené komponentě, stejně jako v knihovně `react-konva` pro 2D zobrazení. U objektů z této knihovny jde pouze určit tzv. pózu, která obsahuje vektor udávající pozici a kvaternion určující rotaci. Proto byla vytvořena knihovna *WorldviewTransforms*, jenž přidává podporu hierarchických transformací do knihovny `regl-worldview`. Nepoužívá však klasickou hierarchii transformačních matic. Při pokusu použít transformační matice pomocí

knihovny `math3d`, bylo zjištěno, že tyto transformace nesedí s těmi používanými v knihovně `regl-worldview` a výsledné pozice objektů nebyly správné. Další problém byl, že pro vykreslení 3D objektů je potřeba znát nejen jejich pozici, ale i natočení, a proto je v hierarchii potřeba uchovávat i kvaternion s celkovým otočením objektu. K sestavení hierarchie se využívá React *Context* nazvaný *TransformContext*. Jeho hodnota je objekt, ve kterém se nachází dvě funkce a kvaternion:

- `transform` — otočí a posune vektor podle dané pózy,
- `inverseTransform` — inverzní funkce k `transform`,
- `orientation` — aktuální otočení objektu (kvaternion).

Každá komponenta, která chce být prvkem hierarchie, využije *hook* `useTransform`, jemuž předá svoji pózu. Tento *hook* nejdříve získá transformační funkce nejbližší nadřazené komponenty pomocí kontextu *TransformContext*. To zajišťuje samotný React, jelikož při použití funkce `useContext` se v hierarchii komponent najde nejbližší *Provider* daného kontextu a v případě, že neexistuje, se vrátí výchozí hodnota. Výchozí hodnotou pro *TransformContext* jsou transformační funkce, které vstup nijak nemění, a kvaternion, jenž vektor nijak neotáčí. Toto zajistí, že komponenta v hierarchii transformací se nemusí starat o to, jestli je jejím kořenem nebo ne. Poté co `useTransform` získá transformace z *TransformContext*, vytvoří své lokální transformace na základě předané pózy. Návratová hodnota tohoto *hooku* jsou pak opět dvě funkce a kvaternion pro *TranformContext*, které zahrnují lokální transformace. Jak je tato nová hodnota vytvořena ukazuje výpis 5.4. Komponenta následně vyrenderuje *Provider* pro *TransformContext*, jenž bude poskytovat návratovou hodnotu `useTransform` podřízeným prvkům hierarchie.

```
function useTransform(pose: Pose): TransformContextValue {
    // Ziskani nadrazenych transformaci
    const parentTransforms = useContext(TransformContext);
    // Vytvoreni lokalnich transformaci
    const transform = createTransform(pose);
    const invTransform = createInverseTransform(pose);

    return {
        // Spojeni transformaci
        transform: point => parentTransforms.transform(transform(point)),
        inverseTransform: point =>
            invTransform(parentTransforms.inverseTransform(point)),
        // Nasobeni kvaternionu spoji jejich rotace do jednoho kvaternionu
        orientation: multiplyQuaternions(
            parentTransforms.orientation,
            pose.orientation
        ),
    };
}
```

Výpis 5.4: Spojení nadřazených a lokálních transformací v *hooku* `useTransform`

Jelikož však knihovna `regl-worldview` s touto hierarchií neumí pracovat, je potřeba její komponenty pro vykreslování 3D objektů (*Cubes*, *Arrows*, *Lines* atd.) obalit do komponent, které hierarchii využijí. Proto knihovna *WorldviewTransforms* poskytuje komponenty

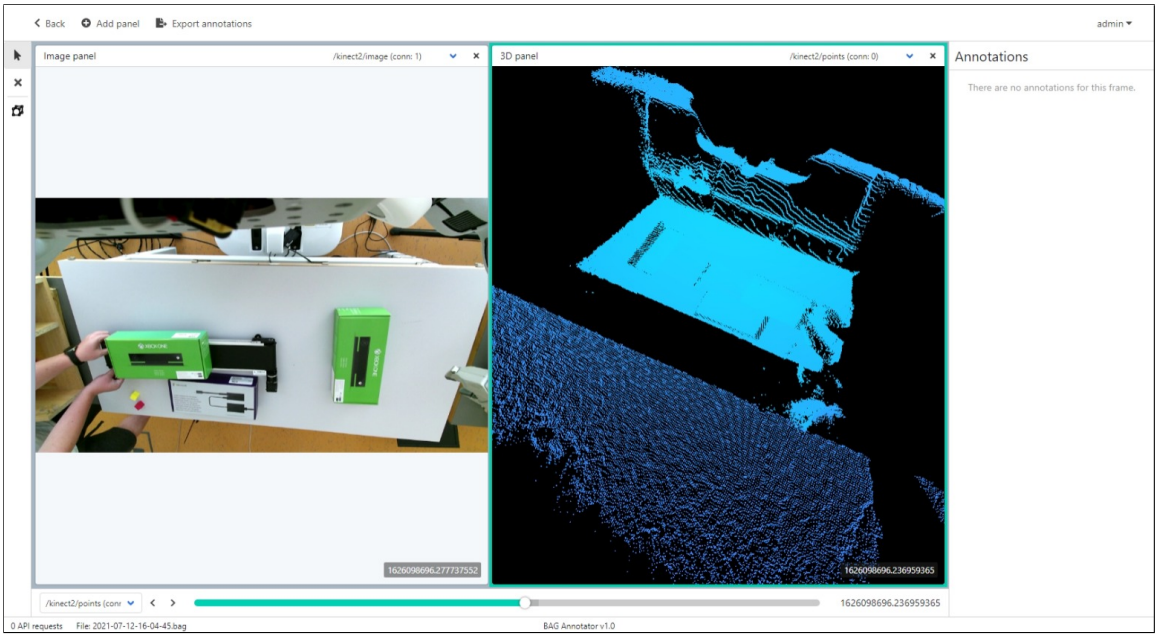
s prefixem „T“ (*TCubes*, *TArrows*, *TLines* atp.), které z *TransformContext* získají transformační funkce a otočení, a před vykreslením původních komponent transformují potřebné vstupy tak, aby se objekty vykreslily správně.

Kapitola 6

Uživatelské testování

Jelikož anotace dat je pracná, je důležité aby uživatelské rozhraní umožňovalo anotace efektivně vytvářet a nezdržovalo uživatele při práci. Vývojář s uživatelským rozhraním pracuje velmi často a ví, jak funguje aplikace uvnitř, tudíž nedokáže dobře posoudit intuitivnost rozhraní pro uživatele, který aplikaci nezná. Proto je nutné nástroj otestovat na skupině takovýchto lidí a vyhodnotit tak efektivnost a intuitivnost uživatelského rozhraní pro nového uživatele.

Hlavní část aplikace je stránka pro přehrávání a anotaci dat z BAG souboru, z tohoto důvodu se při testování budeme soustředit právě na ni. Uživatelé, jenž budou aplikaci testovat, nemají zkušenosti s anotací dat ani s BAG soubory. Pro testování tedy byla zvolena metoda, kde na začátku uživatele seznámíme s problematikou a stručně předvedeme co aplikace umí, aby získal nějakou představu o tom, co se po něm bude chtít, a až následně bude uživatel provádět samostatný úkol. Tímto tyto uživatele přiblížíme reálným uživatelům, kteří už něco o anotaci většinou ví, nebo jsou s nástrojem na začátku zaučeni.



Obrázek 6.1: Ukázka dat z testovacího souboru použitého k uživatelskému testování.

Samostatné úkoly budou probíhat na testovacím souboru, který vznikl přímo pro tyto účely. V souboru jsou obrazové i 3D data ze zařízení Kinect, které je umístěno nad stolem, kde leží krabice a s jednou se během záznamu pohybuje, aby bylo možné otestovat sledování objektu. Ukázku testovacích dat lze vidět na obrázku 6.1.

Po testování uživatel vyplní krátký dotazník, pro měření použitelnosti aplikace. K tomu je použit dotazník SUS (*System Usability Scale*). John Brooke publikoval tento dotazník v roce 1986 a od té doby se stal průmyslovým standardem [19]. Používá se nejenom pro testování softwaru, ale i hardwaru, telefonů nebo dokonce Zlatých stránek [19]. Dotazník se skládá z deseti položek, kde každá položka je nějaké tvrzení o systému a uživatel hodnotí, jestli s ním souhlasí nebo nesouhlasí. Tvrzení se hodnotí škálou od 1 do 5 [19]. 1 znamená „vůbec nesouhlasím“ a 5 znamená „naprosto souhlasím“ [19]. Položky dotazníku jsou následující:

1. Rád bych systém používal opakovaně.
2. Systém je zbytečně složitý.
3. Systém se snadno používá.
4. Potřeboval bych pomoc člověka z technické podpory, abych mohl systém používat.
5. Různé funkce systému jsou do něj dobře začleněny.
6. Systém je příliš nekonzistentní.
7. Řekl(a) bych, že většina lidí se se systémem naučí pracovat rychle.
8. Systém je příliš neohrabaný.
9. Při práci se systémem se cítím jistě.
10. Musel jsem se hodně naučit, než jsem se systémem dokázal pracovat.

Odpovědi se následně zpracují a výsledkem pro odpovědi každého uživatele je jedno skóre mezi 0 a 100 [19]. Ačkoliv to rozsah naznačuje, nejedná se o procentuální škálu [19]. Skóre je následně nejlepší převést ještě na percentil nebo známku (A až F) pomocí grafu 6.2, který byl sestaven z 500 studií [19]. Percentil tedy udává, kolik produktů z těchto studií mělo horší použitelnost než aktuálně testovaný produkt.

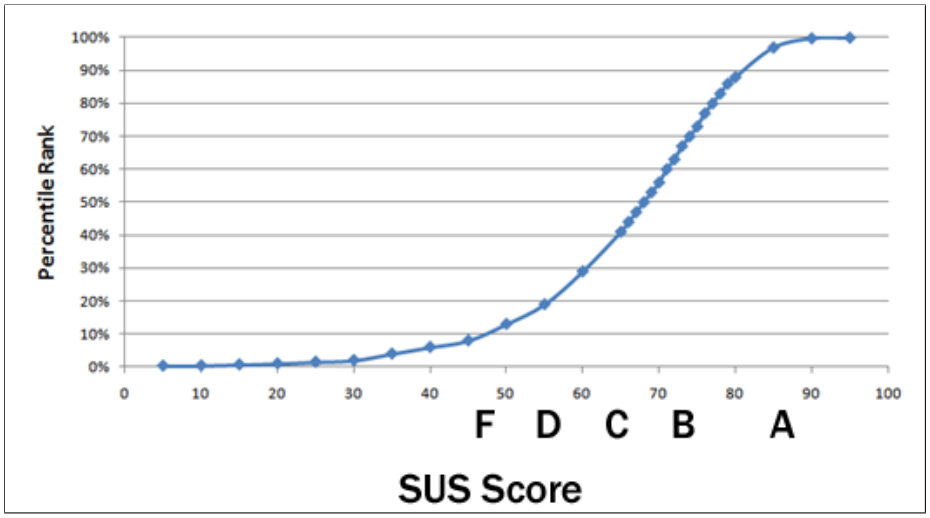
6.1 Průběh testování

Na začátku testování je uživatel seznámen s jeho průběhem. Nejdříve je mu vysvětleno, že není předmětem testování on, ale aplikace, a proto se nemusí bát, že by něco udělal či řekl špatně. Také je poučen, že by měl během samostatného úkolu přemýšlet nahlas (tzv. *Think-aloud* protokol). Následně je uživateli vysvětleno k čemu aplikace slouží a proběhne krátká demonstrace toho, jak se s ní pracuje. Během této demonstrace je předvedena práce s panely pro zobrazení dat, vytváření a manipulace 2D i 3D anotací včetně sledování objektu, navigace mezi snímky a tak dále. Po tomto seznámení s aplikací jsou uživateli zadány postupně úkoly, při kterých je pozorováno, s čím si uživatel neví rady, co ho frustruje, nebo co nemůže najít. Úkoly pro uživatele jsou následující:

1. Najděte soubor `testing.bag` v projektu *Usability testing* a otevřete jej pro anotaci.

2. Přidejte panel s daty z kamery.
3. Přejděte přibližně do poloviny souboru.
4. Anotujte zelené krabice v 3D zobrazení i v obrazových datech. Využijte sledování objektu.
5. Ukončete sledování objektu.
6. Odstraňte jednu z anotací v každém panelu.
7. Exportujte anotace.

Po ukončení testování je uživateli dán k vyplnění SUS dotazník, který vyplní v klidu sám, a ohodnotí pomocí něj svůj dojem z používání aplikace. Poté následuje volná diskuze s uživatelem, kde je možné se ho doptat na to, co se mu líbilo nebo nelíbilo, případně na jeho nápady na vylepšení uživatelského rozhraní.



Obrázek 6.2: Graf zobrazující asociaci mezi SUS skóre a percentilem [19]

6.2 Vyhodnocení výsledků

Aplikace byla testována šesti lidmi studujících nebo pracujících v oblasti IT, nikdo však neměl zkušenosti s BAG soubory nebo anotací dat. Polovina testů proběhla osobně a polovina online. Aplikace byla dostupná na veřejném serveru a účastníci ji tak mohli testovat na svých zařízeních.

Jako první se podíváme na výsledky SUS dotazníku. Odpovědi byly vyhodnoceny podle [19] a to tak, že od hodnoty odpovědi lichých položek byla odečtena 1 a od hodnota odpovědi sudých položek byla odečtena od čísla 5. Toto nám převede všechny odpovědi na rozsah 0 až 4. Odpovědi v tomto rozsahu pak sečteme pro každého uživatele zvlášť a vynásobíme hodnotou 2,5, aby výsledné skóre bylo v rozsahu 0 - 100. Pomocí grafu na obrázku 6.2 se skóre převede na percentil. Z grafu lze vidět, že průměrné skóre je přibližně 68. Vyhodnocené výsledky můžete vidět v tabulce 6.1.

	SUS skóre	Percentil	Známka
Uživatel 1	62,5	37	C
Uživatel 2	60,0	29	C
Uživatel 3	60,0	29	C
Uživatel 4	57,5	24	D
Uživatel 5	65,0	40	C
Uživatel 6	55,0	19	D
Průměr	60,0	29,6	C

Tabulka 6.1: Tabulka s vyhodnocenými odpověďmi z dotazníku SUS.

Jak lze vidět, použitelnost aplikace byla průměrně hodnocena skórem 60, což je podprůměrný výsledek. V tabulce 6.2, kde je průměrné skóre pro jednotlivá tvrzení z dotazníku, lze vidět, že hlavní problém aplikace je, že se uživatelé necítí jistě při jejím používání. Toto pramenilo z absence možnosti vrátit se o krok zpět při anotaci. Uživatelé se báli, že když při pohybu s anotací udělají chybu, tak ji nebudou moct vrátit jednoduše zpět. S tím šla ruku v ruce složitější manipulace s anotacemi ve *3D panelu*. Uživatelé se snažili posouvat anotace stejně, jako v *Image panelu*, kliknutím kdekoliv na anotaci a tažením, místo toho však hýbali kamerou, což vedlo ke zmatenosti a frustraci. S orientací v 3D datech měli uživatelé obecně problém, ale ukázalo se, že možnost zobrazit si k 3D datům zároveň i obraz z kamery, jim v orientaci pomohla. Co také pomohlo, ačkoliv na to sami nepřišli, bylo zobrazení 3D dat z více úhlů zároveň. Toho lze docílit přidáním více panelů, kde každý zobrazuje stejná data, a v každém panelu si uživatel nastaví pohled z jiné strany (např. shora a z boku). Horší orientaci v prostoru také způsobily testovací data, které byly vzhůru nohama (takto data přišla již ze senzoru Kinect) a pohyb kamery, jenž je implementován knihovnou `regl-worldview`, byl při pohledu ze spodní strany invertovaný a nedal se příliš dobře používat. Řešením tohoto by mohlo být přidání globální transformace dat do nastavení panelu, kde by si uživatel mohl data vertikálně či horizontálně převrátit nebo otočit.

Druhý nejhorší výsledek mělo tvrzení, že by uživatel potřeboval pomoc člověka z technické podpory, aby mohl systém používat. To není až tak překvapivé, jelikož aplikace je šitá na míru pro práci s BAG soubory, které nejsou obecně známé, a lidé nemají dobrou představu o tom, jak tyto soubory fungují. Některé funkce aplikace, jako je například přidávání nového panelu, kde si uživatel musí vybrat, které téma chce zobrazit, uživatelé nechápou, protože názvy témat a jejich datových typů jsou pro ně neznámé a často nepochopitelné. Řešením by mohlo být přidání pomocného textu nebo ikonky k názvu tématu, jenž bude pro běžného člověka přívětivější (např. přidání označení *Image data*, *3D data* atp.).

Třetí nejhorší výsledek mělo hodnocení konzistentnosti aplikace. Tohle zapříčinily převážně rozdíly v manipulaci s anotacemi ve 2D a 3D prostoru. Ve 2D se obdélníková anotace vytváří klikáním, což bylo uděláno pro budoucí konzistenci. Kdyby se totiž v budoucnu přidaly například polygonové anotace, které se lépe vytvářejí klikáním, a vytváření obdélníkové anotace by bylo řešeno táhnutím myší, bylo by těžké konzistenci zachovat. Vytváření kvádrové anotace ve 3D však bylo uděláno právě táhnutím myši, jelikož tato funkcionalita byla implementována již pro manipulaci s anotací a dala se jednoduše znovu použít. Tímto vznikla nekonzistence mezi vytvářením 2D a 3D anotací. Z testování se nakonec ukázalo, že uživatelé preferují tažení myší. Bylo to vždy první co zkusili a také to vyplynulo z ná-

Tvrzení	**Průměr**
Rád bych systém používal opakovaně.	2,5
Systém je zbytečně složitý.	3,3
Systém se snadno používá.	2,3
Potřeboval bych pomoc člověka z technické podpory, abych mohl systém používat.	1,5
Různé funkce systému jsou do něj dobře začleněny.	3,3
Systém je příliš nekonzistentní.	1,8
Řekl(a) bych, že většina lidí se se systémem naučí pracovat rychle.	3,0
Systém je příliš neohrabaný.	2,6
Při práci se systémem se cítím jistě.	1,3
Musel jsem se hodně naučit, než jsem se systémem dokázal pracovat.	2,1

Tabulka 6.2: Průměrné skóre v rozsahu 0 - 4 pro jednotlivá tvrzení.

sledné diskuze. Další problémy z konzistencí se už týkaly menších věcí. Jednalo se o použití ikony s křížkem pro nástroj na mazání anotací v levém panelu a ikony odpadkového koše pro tu samou funkci v kartě s anotací v pravém panelu a *Image panel* měl funkci zoomu implementovanou opačně oproti *3D panelu*.

Další věc co uživatelům dělalo problém, bylo ukončení sledování objektu. Jelikož při zmáčknutí tlačítka s touto funkcí se na aktuální zprávu nastaví pro danou anotaci pouze klíčový snímek a ukončení sledování se reálně promítne až na další zprávě, pro uživatele se na aktuální obrazovce nic nezmění a to je pro něj matoucí. Implementovat tuto funkci tak, aby sledování skončilo již na aktuálním snímku a anotace z něj zmizela, jak někteří uživatelé navrhovali, není úplně jednoduché a přináší řadu dalších problémů, a proto byl tento problém prozatím vyřešen vyskakovacím upozorněním, že akce proběhla úspěšně a že anotace zmizí až na následující zprávě. Ideálním řešením, jenž bylo také některými uživateli navrhnuto, by asi bylo, kdyby tlačítko pro ukončení sledování se po kliknutí změnilo zpátky na tlačítko pro zapnutí sledování objektu a uživatel jím mohl sledování opět obnovit. Pro tuto implementaci však již nezbyl čas a byla ponechána pro budoucí vývoj.

Během testování se také narazilo na chybu v prohlížeči Firefox, kde se v 3D zobrazení náhodně přestávaly zobrazovat některé tvary vykreslované knihovnou `regl-worldview`, jenž byly použity pro anotace. Při zkoumání této chyby, se přišlo na to, že se nejspíš jedná o chybu v této knihovně a nelze ji jednoduše vyřešit. Řešením by mohlo být vlastní implementace vykreslování těchto tvarů pomocí komponenty *Triangles*, jenž se zdá být funkční.

Uživatele během testování také mátlo fungování panelu nástrojů pro anotaci na levé straně obrazovky. Dostupné nástroje se mění podle aktuálně zvoleného panelu, čehož si však uživatelé nevšimli a snažili se použít nástroj pro vytváření obdélníkových anotací v *3D panelu*, což nejde. Většinou si ani nevšimli, že se panely při kliknutí označují obrysem. Jako řešení uživatelé navrhovali mít všechny nástroje v panelu zobrazené najednou a pouze

v případě, že nástroj v panelu nejde použít, se zobrazí nějaká informační hláška, nebo by každý panel měl svůj vlastní panel nástrojů s nástroji, které podporuje.

Na základě testování byly provedeny následující změny:

- Bylo přidáno označení typu dat při výběru tématu pro zobrazení v novém panelu, pro lepší přehled.
- Ikonka nástroje pro mazání byla změněna z křížku na popelnici, pro lepší konzistenci.
- Vytváření obdélníkové anotace bylo změněno na metodu s tažením myši místo klikání.
- Při zastavení sledování objektu vyskočí zpráva informující o vykonané akci a informací o tom, že anotace zmizí až na dalším snímku.
- Přibližování a oddalování v *Image panelu* a *3D panelu* kolečkem myši bylo sjednoceno.

6.3 Budoucí vývoj

V rámci dalšího vývoje aplikace je potřeba rozšířit sadu anotačních nástrojů. Například o nástroje pro vytváření polygonů, sady bodů pro vyznačení orientačních bodů v obličeji nebo nástroje pro segmentaci dat. Díky navrhnutému rozhraní, které bylo popsáno v sekci 5.8, lze nové druhy anotací do aplikace implementovat poměrně jednoduše.

Dále je potřeba optimalizovat parsování dat na frontendu, aby nezasekávalo uživatelské rozhraní (viz sekce 5.7). Jedna z možností je posílat data do Web Workerů po menších částech, aby mělo uživatelské rozhraní čas se mezitím překreslit. Další možností by mohlo být využití objektu *SharedArrayBuffer*[1], jenž umožňuje sdílet paměť mezi vlákny.

Snížení spotřeby paměti RAM je také jedna z věcí, na které je potřeba se zaměřit. Jeden ze způsobů, jak toho docílit, by mohlo být ukládání *Chunk* záznamů na frontendu v původní binární formě a konkrétní zprávy by se parsovaly, až když by bylo potřeba. Zprávy v binární formě totiž zabírají podstatně méně místa, než když jsou převedeny na objekty a pole jazyka JavaScript.

Dalším možným rozšířením aplikace by mohlo být přidání podobného konstruktoru štítků, jako je v aplikaci CVAT zmíněné v sekci 4.2. Uživatel by si tak mohl definovat vlastní atributy štítků, pomocí kterých by bylo možné k anotacím přidávat další informace, jako například věk osoby.

Automatická transformace anotací mezi 2D a 3D daty v případě, že soubor obsahuje kalibrované senzory, je jedna z dalších věcí co by mohla být do aplikace přidána. Její implementace byla zvažována již v rámci diplomové práce, avšak kvůli komplexitě této funkce a samotné aplikace, byla ponechána do budoucího vývoje. Náročnost řešení spočívá v nutnosti implementovat knihovnu `tf2`[2] v jazyce TypeScript pro použití na frontendu a také v návrhu uživatelského rozhraní, které umožní intuitivní a pohodlné používání této funkce.

V neposlední řadě by aplikace mohla být rozšířena o poloautomatickou anotaci pomocí předem natrénovaných modelů pro detekci běžných objektů, jako například lidí, tváří, vozidel atd. Daly by se využít kupříkladu modely Mask R-CNN[3] nebo YOLO v3[4]. Uživatel by pak nemusel všechny objekty anotovat ručně, ale mohl si data nechat anotovat těmito modely a následně anotace pouze zrevidovat a upravit podle potřeby.

[1] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/SharedArrayBuffer

[2] http://wiki.ros.org/tf2

[3] https://github.com/matterport/Mask_RCNN

[4] https://github.com/ultralytics/yolov3

Kapitola 7

Závěr

V rámci této diplomové práce byla navržena a implementována webová aplikace pro přehrávání a anotaci multimediálních dat ze souborů ve formátu BAG používaných v systému ROS, který slouží k řízení robotických systémů. Formát BAG mimo jiné obsahuje data ze senzorů, jako jsou LiDARy, kamery, 3D kamery a tak dále. Aplikace umožňuje zobrazit tyto data v panelech, jejichž rozložení si uživatel může přizpůsobit (může přidávat nové panely, přesouvat je nebo zmenšovat a zvětšovat). Možnost souběžného zobrazení a anotace více dat najednou uživateli usnadní práci.

Nejdříve bylo potřeba nastudovat formát BAG a jak je definována struktura jednotlivých zpráv, které jsou v něm uložené. Dále byly prozkoumány nejpoužívanější knihovny pro vytváření moderních uživatelských rozhraní webových aplikací: React, Vue.js a Angular. U každé byly popsány její přednosti a nevýhody. V rámci práce byly také rozebrány existující řešení, jako jsou CVAT, VoTT nebo Supervisely, které slouží k anotaci 2D nebo 3D dat uložených v klasických souborových formátech (MP4, PNG, PCL atd.), nebo také Webviz, jenž slouží k přehrávání dat z BAG souborů, avšak neumožňuje jejich anotaci. V rámci návrhu aplikace proběhl výběr architektury a následně technologií pro implementaci klientské a serverové části.

Jelikož klientská aplikace načítá data ze serveru po malých částech, je potřeba BAG soubory číst s náhodným přístupem. Existující knihovny však náhodný přístup neumožňují, a proto byla navrhnuta vlastní knihovna `rosbag_pyreader` v jazyce Python, jež umožňuje ze souboru přečíst metadata, pomocí kterých lze k datům následně přistupovat náhodným způsobem. Proto při nahrání souboru na server je potřeba jej ještě zpracovat. To znamená přečíst ze souboru metadata a ty si následně uložit do databáze. Při požadavku na konkrétní data pak stačí z uložených metadat zjistit, kde se data v souboru nachází a rovnou je přečíst a vrátit v odpovědi.

Výsledná webová aplikace umí efektivně zobrazovat 2D a 3D data pomocí systému panelů, který je zároveň rozšiřitelný pro nové typy dat. Zobrazené zprávy z více různých témat jsou časové synchronizované a lze je anotovat. Aplikace také umožňuje seskupovat BAG soubory do projektů, pro lepší organizaci. V rámci projektu pak lze definovat vlastní sadu štítků, které pak lze použít pro anotaci dat v souborech v daném projektu. Aplikace umožňuje vytvořené anotace přepnout do režimu manuálního sledování objektu, kde se daná anotace promítne do všech následujících zpráv a uživatel ji může postupně posouvat dle pohybu objektu. Při posunu anotace během sledování objektu se vytváří klíčové snímky a ve snímcích mezi těmito klíčovými se anotace lineárně interpolují. Uživatel tak nemusí upravovat anotaci na každém snímku, ale pouze tam, kde se mění pohyb objektu.

Aplikace byla nakonec testována na uživatelskou použitelnost, kde byly zjištěny nedostatky v oblasti konzistence a intuitivnosti některých částí uživatelského rozhraní. Na základě zpětné vazby uživatelů, kteří aplikaci testovali, bylo implementováno několik změn, aby se tyto nedostatky pokud možno odstranily.

Ačkoliv je aplikace použitelná již teď, představuje díky rozšiřitelnosti dobrý potenciál i pro budoucí vývoj. Aplikaci lze jednoduše rozšířit o nové typy anotací nebo o nové panely pro zobrazování nového typu dat. Dále by se dala přidat podpora pro transformaci anotací mezi 2D a 3D daty nebo poloautomatická anotace pomocí předem natrénovaných modelů.

Literatura

[1] Christie, T. *Authentication*. 2021. [Online; navštíveno 15. 7. 2021]. Dostupné z: https://www.django-rest-framework.org/api-guide/authentication/.

[2] Conley, K., Faust, J., Hendrix, A. et al. *ROS/Connection Header*. 2019. [Online; navštíveno 27. 12. 2020]. Dostupné z: http://wiki.ros.org/msg.

[3] Conley, K., Field, T., Alexander, B. et al. *ROS/Connection Header*. 2011. [Online; navštíveno 26. 12. 2020]. Dostupné z: http://wiki.ros.org/ROS/Connection%20Header.

[4] Conley, K., Leibs, J. et al. *Bags - ROS Wiki*. 2020. [Online; navštíveno 15. 1. 2021]. Dostupné z: http://wiki.ros.org/Bags.

[5] Daityari, S. *Angular vs React vs Vue: Which Framework to Choose in 2021*. 2021. [Online; navštíveno 28. 7. 2021]. Dostupné z: https://www.codeinwp.com/blog/angular-vs-vue-vs-react/.

[6] Docker. *What is a Container? | App Containerization | Docker*. 2021. [Online; navštíveno 15. 7. 2021]. Dostupné z: https://www.docker.com/resources/what-container.

[7] Facebook Inc.. *Components and Props - React*. 2021. [Online; navštíveno 13. 1. 2021]. Dostupné z: https://reactjs.org/docs/components-and-props.html.

[8] Facebook Inc.. *State and Lifecycle - React*. 2021. [Online; navštíveno 13. 1. 2021]. Dostupné z: https://reactjs.org/docs/state-and-lifecycle.html.

[9] Field, T. a Foote, T. *Bags/Format/2.0*. 2013. [Online; navštíveno 26. 12. 2020]. Dostupné z: http://wiki.ros.org/Bags/Format/2.0.

[10] Gerkey, B., Conley, K., Leibs, J., Thomas, D. et al. *ROS/ChangeList/1.1*. 2013. [Online; navštíveno 26. 12. 2020]. Dostupné z: https://wiki.ros.org/ROS/ChangeList/1.1##A1.1.5_.282010-05-12.29.

[11] Open Robotics. *About ROS*. [Online; navštíveno 26. 12. 2020]. Dostupné z: https://www.ros.org/about-ros/.

[12] Open Robotics. *Core Components*. [Online; navštíveno 26. 12. 2020]. Dostupné z: https://www.ros.org/core-components/.

[13] Open Robotics. *Integration with Other Libraries*. [Online; navštíveno 26. 12. 2020]. Dostupné z: https://www.ros.org/integration/.

[14] Open Source Robotics Foundation. *Gazebo*. 2014. [Online; navštíveno 26. 12. 2020]. Dostupné z: http://gazebosim.org/.

[15] OpenCV team. *About - OpenCV*. 2020. [Online; navštíveno 26. 12. 2020]. Dostupné z: https://opencv.org/about/.

[16] Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T. et al. ROS: an open-source Robot Operating System. In: Kobe, Japan. *ICRA workshop on open source software*. 2009, sv. 3, 3.2, s. 5.

[17] ROS-Industrial. *Description — ROS-Industrial*. [Online; navštíveno 26. 12. 2020]. Dostupné z: https://rosindustrial.org/about/description.

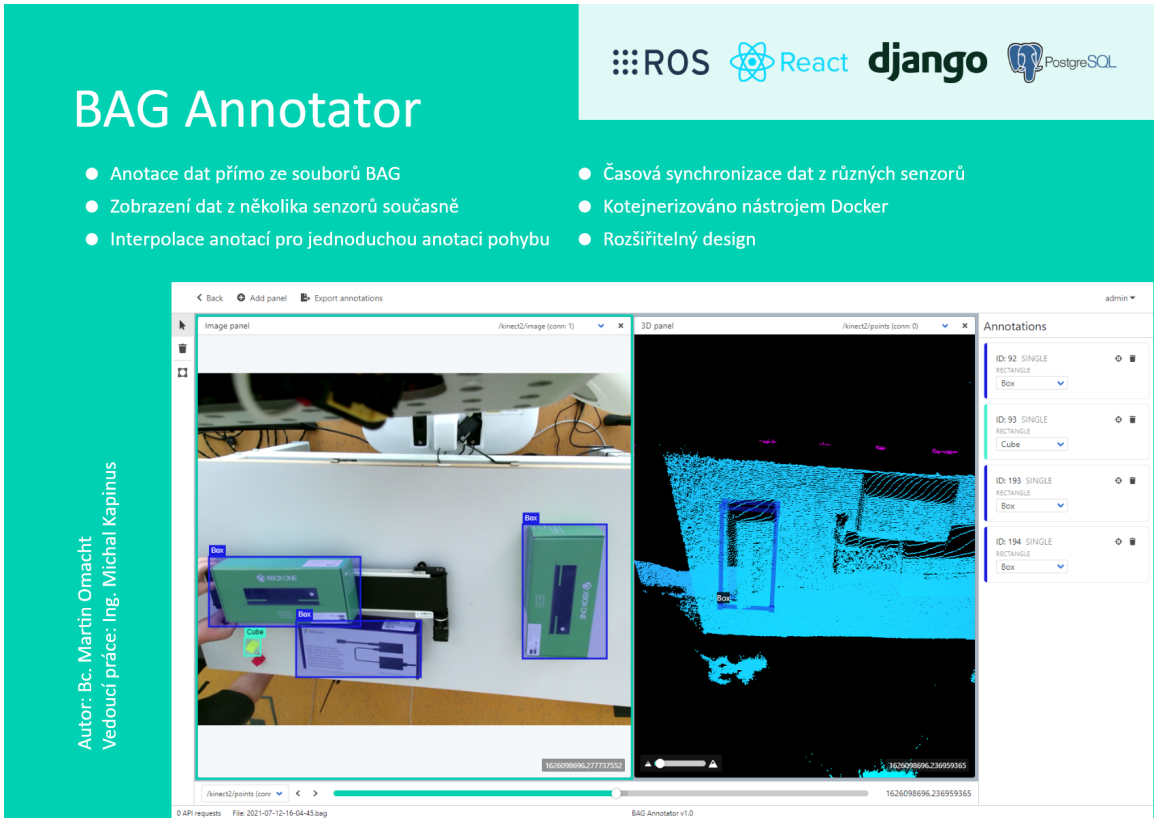
[18] Rusu, R. B. a Cousins, S. 3D is here: Point Cloud Library (PCL). In: *IEEE International Conference on Robotics and Automation (ICRA)*. Shanghai, China: [b.n.], May 9-13 2011.

[19] Sauro, J. *Measuring Usability with the System Usability Scale (SUS)*. 2011. [Online; navštíveno 19. 7. 2021]. Dostupné z: https://measuringu.com/sus/.

[20] Sucan, I. A. a Chitta, S. *MoveIt*. [Online; navštíveno 26. 12. 2020]. Dostupné z: https://moveit.ros.org/.

Příloha A

Plakát



Obrázek A.1: Plakát nástroje BAG Annotator.

Příloha B

Obsah přiloženého paměťového média

- `/data` — Ukázkové BAG soubory.
- `/doc` — Zdrojové kód technické zprávy.
- `/src` — Zdrojové kódy webové aplikace.
- `/README.txt` — Textový soubor s informacemi o obsahu paměťového média a způsobu sestavení a spuštění webové aplikace.
- `/plakat.png` — Plakát aplikace BAG Annotator.
- `/xomach00.pdf` — Technická zpráva ve formátu PDF.