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

DOPLNENIE TRAJEKTÓRIE LETU LIETADLA O VHF KOMUNIKÁCIU S PREPISOM

ADDING FLIGHT TRAJECTORY OF VHF COMMUNICATION WITH TRANSCRIPTION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **ADAM RAJKO**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. IGOR SZŐKE, Ph.D.**
SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

146281

Ústav:	Ústav počítačové grafiky a multimédií (UPGM)
Student:	**Rajko Adam**
Program:	Informační technologie
Specializace:	Informační technologie
Název:	**Doplnění trajektorie letu letadla o VHF komunikaci s přepisem**
Kategorie:	Zpracování signálů
Akademický rok:	2022/23

Zadání:

1. Seznamte se s výsledky projektu ATCO2, mapovými frameworky, základy letecké komunikace, VHF, ADS-B.
2. Najděte a nastudujte dostupné API pro získávání jak hlasové komunikace, tak geografické pozice letadla.
3. Vytvořte koncept služby vizualizující pozici letadla doplněnou o hlasovou komunikaci pilot-věž.
4. Otestujte službu na několika uživatelích a získejte zpětnou vazbu. Službu vylepšete.
5. Zhodnoťte výsledky a navrhněte směry dalšího vývoje.
6. Vyrobte A2 plakátek a cca 30 vteřinové video prezentující výsledky vaší práce.

Literatura:

- ATCO2 projekt, http://atco2.org
- OpenSky Network, http://opensky-network.org
- Dále dle pokynů vedoucího

Při obhajobě semestrální části projektu je požadováno:
Body 1 až 3 ze zadání.

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

Vedoucí práce:	**Szőke Igor, Ing., Ph.D.**
Vedoucí ústavu:	Černocký Jan, prof. Dr. Ing.
Datum zadání:	1.11.2022
Termín pro odevzdání:	10.5.2023
Datum schválení:	31.10.2022

Abstrakt

Cieľom tohto projektu je vytvorenie aplikácie pre doplnenie rečovej komunikácie medzi pilotom a riadením leteckej prevádzy (ATC - Air Traffic Control) do trajektórie letu lietadla. Inšpiráciou pre túto prácu je projekt ATCO2. Na základe naštudovaných mapových frameworkov a služieb poskytujúcich informácie o pozíciach lietadiel bol vykonaný návrh aplikácie. Na základe návrhu a použitých technológií sa podarilo aplikáciu implementovať. Výsledná funkcionalita aplikácie bola otestovaná na užívateľoch použitím dotazníka.

Abstract

The aim of this project is to create an application for adding speech communication between the pilot and ATC (Air Traffic Control) to the flight trajectory of the aircraft. The inspiration for this work is the ATCO2 project. Based on the studied mapping frameworks and services providing aircraft position information, an application design has been performed. Based on the design and used technologies, the application was successfully implemented. The resulting functionality of the application was tested on users using a questionnaire.

Kľúčové slová

Flask, API, UI, UX, použiteľnosť, React, riadenie letovej prevádzky, webová aplikácia, Bootstrap, testovanie funkcionality

Keywords

Flask, API, UI, UX, usability, React, air traffic control, web application, Bootstrap, functionality testing

Citácia

RAJKO, Adam. *Doplnenie trajektórie letu lietadla o VHF komunikáciu s prepisom.* Brno, 2023. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Igor Szőke, Ph.D.

Doplnenie trajektórie letu lietadla o VHF komunikáciu s prepisom

Prehlásenie

Prehlasujem, že som túto bakalárku prácu vypracoval samostane pod vedením pána Ing. Igora Szőkeho, Ph.D. Uviedol som všetky literárne pramene, publikácie a ďalšie zdroje, z ktorých som čerpal.

.....
Adam Rajko
9. mája 2023

Poďakovanie

Ďakujem svojmu vedúcemu práce pánovi Ing. Igorovi Szőkemu, Ph.D. za konzultácie, odborné rady a veľkú trpezlivosť.

Obsah

Kapitola 1

Úvod

Cieľom tohto projektu je vytvorenie aplikácie pre doplnenie rečovej komunikácie medzi pilotom a riadením leteckej prevádzky (ATC - Air Traffic Control) do trajektórie letu lietadla. Konkrétne je komunikácia zachytávaná na VHF (very high frequency). Inšpiráciou pre túto prácu je projekt ATCO2.

Projekt ATCO2[1] sa zameriava na vývoj platformy pre zber, organizáciu a predbežné spracovanie údajov o riadení letovej prevádzky (hlasovej komunikácie). V projekte ATCO2 sú pomocou strojového učenia ziskané prepisy z komunikácie a pre lepšiu vizualizáciu výsledku práce by bola vhodná webová aplikácia s interaktívnou mapou - čo je cieľom tejto práce.

Prepisy k nahrávkam sú poskytované anotačnou službou SpokenData[2] firmy ReplayWell, ktoré sú získavané prostredníctvom API danej služby.

Nasledujúca kapitola tejto práce sa zaoberá základnými informáciami pre lepšie pochopenie problematiky, prečo boli isté riešenia prevedené istým spôsobom a potrebné informácie pre pochopenie diagramov použitých v kapitolách nižšie.

Tretia kapitola sa zaoberá s technológiami, ktoré boli použité pri implementácií tejto aplikácie a prečo bola daná technológia zvolená.

Štvrtá kapitola sa zaoberá získavaním dát potrebných na vytvorenie aplikácie. Taktiež sa v tejto kapitole nachádza, ako sú dané dáta následne spracované po ich získaní.

V piatej kapitole je popísaná implementácia aplikácie, kde je vysvetlený spôsob fungovania backendu, ako aj frontendu do potrebnej hĺbky pre prípadné pokračovanie v práci alebo jej úpravu.

Šiesta kapitola sa zaoberá testovaním výslednej aplikácie vo forme dotazníka. V tejto kapitole sú popísané upravené funkcionality na základe spätnej odozvy a možnosti na ďalší vývoj aplikácie.

V závere práce sú zrekapitulované a zhodnotené dosiahnuté výsledky a taktiež stručne zhrnuté možné vylepšenia výsledku práce a jej možné smerovanie do budúcna.

[1] https://www.atco2.org/
[2] https://www.spokendata.com/

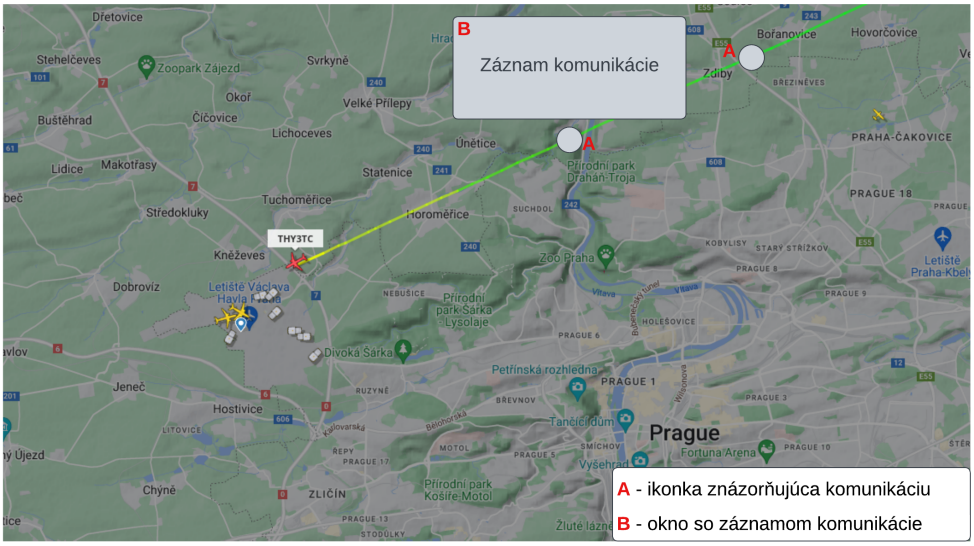
Kapitola 2

Základné informácie

Pre lepšie pochopenie problematiky, ktorá je popísaná hlbšie v ďalších kapitolách je potrebné vedieť na základe akých dát sa vychádzalo a aké boli použité technológie pre zber týchto dát. Taktiež aké prostriedky boli použité ďalej v práci pre spracovanie dát a vývoj.

2.1 Uchopenie problematiky

Výsledkom práce by mal byť v jednoduchosti ozvučený flightradar24[1], kde sa do trajektórie letu doplnia body, v ktorých bola zachytená komunikácia.



Obr. 2.1: Doplnená snímka z flightradar24 o návrh zobrazenie záznamov k trase letu

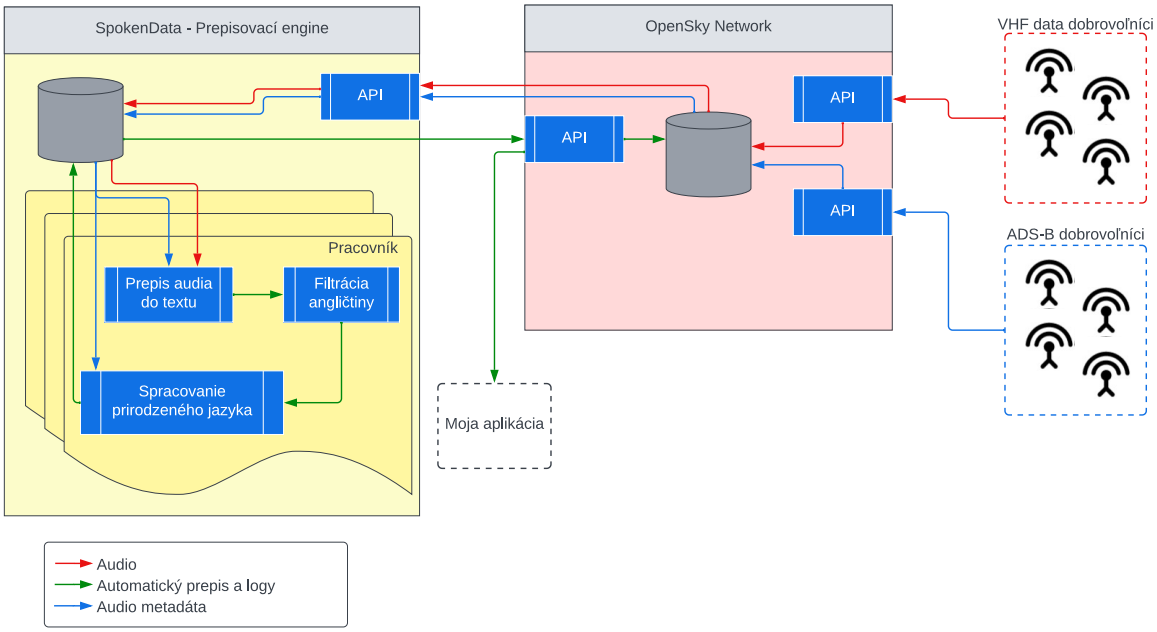
Na obrázku vyššie 2.1 je znázorená myšlienka, ako by sa mohla zobrazovať komunikácia v trase letu.

Na trasovanie letu je potrebné cyklicky získavať informáciu o aktuálnej polohe lietadla. Na vyriešenie tohto problému je možné použiť dostupné služby, ktoré poskytujú API na

[1] https://www.flightradar24.com/about

získanie aktuálneho stavu lietadla. API na tento účel poskytuje aj flightradar, ale aj napriek vyššiemu pokrytiu bolo použité API od OpenSky Networku. Na základe žiadosti od vedúceho práce vzhľadom na prepojenie služieb OpenSky Network a SpokenData.

Pre získanie komunikácie exitujú tiež rôzne služby, ako napríklad liveatc[2], ktorá poskytuje stream komunikácie. Služba ale nemôže byť použitá v aplikáciách tretích strán. Vzhľadom na žiadost veducého práce a prepojeniu služby SpokenData s OpenSky Network sa nahrávky a ich prepis získavajú z tejto služby.



Obr. 2.2: Komunikácia medzi SpokenData a OpenSky Network (poskytnuté ako interná správa z ATCO2 projektu)

Na obrázku vyššie 2.2 je znázornená komunikácie medzi OpenSky Network a službou SpokenData ako aj pripojenie mojej aplikácie na API od OpenSky Network.

Na priradenie komunikácie k letu sa porovnáva volací znak letu získaný z OpenSky Network s automaticky detekovaným volacím znakom zo služby SpokenData. Vzhľadom na to, že volací znak sa nepodarí vždy dokonale detekovať alebo je zvuk zašumený má za následok neúplné doplnenie záznamov k danému letu.

ATCO2

Informácie z tejto sekcie sú z [1]. Projekt sa zameriava na vývoj jedinečnej platformy na zber, organizáciu a predbežné spracovanie údajov riadenia letovej prevádzky (hlasová komunikácia) z vzdušného priestoru. Cieľom projektu je zohľadniť hlasovú komunikáciu v reálnom čase medzi riadiacimi letovej prevádzky a pilotmi, ktorá je dostupná buď priamo prostredníctvom verejne prístupných rádiofrekvenčných kanálov, alebo nepriamo od poskytovateľov leteckých navigačných služieb (ANSP). Okrem hlasovej komunikácie sa budú využívať aj kontextové informácie vo forme metadát (údaje z dohľadu).

Výsledkom tohto projektu sú datasety, ktoré sú určené na vývoj a vyhodnotenie automatického rozpoznávania hovorenej reči.

[2] https://www.liveatc.net

OpenSky Network

OpenSky Network[3] je nezisková organizácia, ktorej cieľom je zvýšiť bezpečnosť, spoľahlivosť a efektívnosť letovej prevádzky tým, že verejnosti ponúkne otvorený prístup k reálnym údajom z riadenia letovej prevádzky. Dáta sa získavajú od príjmačov dobrovoľníkov (feeders), po celom svete. Po spracovaní dát sú ďalej poskytované veronosti pomocou API ktoré je možné použiť na vývoj aplikácií, ktoré napríklad monitorujú stav vzdušného priestoru.

Hlavnými technológiami, ktoré používa sieť OpenSky, sú Automatic Dependent Surveillance Broadcast (ADS-B) 2.2 a Mode S 2.2.

2.2 Základy leteckej komunikácie

ATCO2 aj OpenSky Network používajú technológie alebo sa zaoberajú s istými zavedenými pravidlami v leteckej komunikácií. Komunikácia je nevyhnuteľná súčasť letectva, ktorá pomáha k zaisteniu bezpečnosti a efektívnosti letov.

Rádiová (hlasová) komunikácia

Informácie z tejto sekcie boli čerpané z [11][8]. Rádiová komunikácia je najbežnejšia forma komunikácie v letectve a je používaná z mnohých dôvodov, ako napríklad komunikácia s dopravou vzdušného priestoru (Air traffic control - ATC), inými lietadlami a pozemným personálom. Rádiová komunikácia používaná v letectve je často vedená pomocou VHF (Very high frequency) rádií, ktoré operujú v rozsahu 118-136 MHz.

VHF

VHF komunikačné systémy sú široko používané pre udržovanie kontaktu medzi zemou a lietadlom. Tie využívajú takzvaný "Line of sight" prenos, čo je v rozsahu okolo 50 kilometrov pre lietadlo, ktoré operuje vo výsške 300 metrov nad zemou alebo 220 kilometrov pre lietadlo operujúce vo výške 3000 metrov.

Callsign

Volacia značka (callsign) lietadla je skupina alfanumerických znakov, ktoré slúžia na identifikáciu lietadla pri komunikácií vzduch-zem.

Typ (a) Znaky zodpovedajúce registračnej značke lietadla (napr. ABCDE). Názov výrobcu lietadla alebo modelu môže byť použitý ako predpona (napr. Airbus ABCDE).

Typ (b) Telefónne označenie prevádzkovateľa lietadla, za ktorým nasledujú posledné štyri znaky registračného označenia lietadla (napr. Rushair BCDE).

Typ (c) Telefónne označenie prevádzkovateľa lietadla, po ktorom nasleduje identifikácia letu (napr. Rushair 1234).

Príklady volacích znakov			
	Typ (a)	Typ (b)	Typ (c)
Celý callsign	Delta Air Lines 123	FedEx 30	Air Ambulance 10
Skrátený callsign	DAL123	FDX30	-

[3] https://opensky-network.org/about/about-us

Identifikácia letu môže byť “verejné” číslo letu, ktoré sa používa pri predaji leteniek a odbavovaní lietadla alebo to môže byť alternatívny jedinečný alfanumerický reťazec.

Taktiež "telefónne označenie agentúry prevádzkujúcej lietadlo"je v prípade leteckých spoločností označenie spoločnosti, pre ktorú sa let prevádzkuje, čo nemusí byť prevádzkovateľ lietadla.

Pri nadväzovaní komunikácie sa musí použiť úplný volací znak.

Po úspešnom nadviazaní komunikácie sa môžu používať skrátené volacie znaky, pokiaľ nedôjde k zámene. Lietadlo však musí používať svoj plný volací znak, kým pozemná stanica nepoužije skrátený volací znak.

Pre volacie znaky sa môžu používať len skratky uvedené v tabuľke vyššie, ktoré sa líšia podľa typu letu.

Typ (a) skratka obsahuje prvý znak registračného čísla lietadla a aspoň posledné dva znaky úplného volacieho znaku. Prípadne sa prvý znak môže nahradiť názvom výrobcu alebo modelu lietadla.

Typ (b) skratka obsahuje telefónne označenie leteckej spoločnosti prevádzkujúcej lietadlo, za ktorým nasledujú aspoň posledné dva znaky úplného volacieho znaku.

Typ (c) nie je k dispozícii žiadna skrátená forma volacieho znaku.

Dátová komunikácia

Informácie z tejto sekcie boli čerpané z [9][10]. Je to systém, ktorý dovoľuje lietadlu vymieňať dáta s ostatnými lietadlami a s pozemnými systémami používaním digitálnej komunikácie. Táto komunikácia môže obsahovať veci ako plány letu, informácie o počasí a správy medzi pilotami a ATC.

Automatic Dependent Surveillance Broadcast (ADS-B)

ADS-B je monitorovacia technika, ktorú používajú lietadlá alebo letiskové vozidlá na vysielanie svojej identity, polohy a ďalších informácií získaných z palubných systémov. Signály ADS-B Out sa môžu používať na sledovanie na zemi alebo inými lietadlami na uľahčenie informovanosti o situácii vo vzdušnej doprave, odstupu, separácie a separovania prostredníctvom ADS-B In.

ADS-B je automatický a závisí od palubných systémov lietadla a vysielané údaje nie sú smerované na konkrétny prijímač alebo príjemcu.

Pozemné rozdeľovanie prevádzky závisí od toho či sú lietadlá vybavené systémom ADS-B Out zatiaľ, čo samostatné rozdeľovanie vo vzduchu si vyžaduje systém ADS-B In a účinné zobrazovanie informácií o prevádzke.

Mode S

Režim S je proces sekundárneho monitorovacieho radaru, ktorý umožňuje selektívne vypočúvanie (monitorovanie) lietadla podľa jedinečnej 24-bitovej adresy priradenej každému lietadlu.

Režim S vo svojej základnej podobe je už mnoho rokov štandardizovaný ICAO. Využíva pozemné dotazovače a vzdušné transpondéry.

Vzdušné transpondéry sa využívajú na poskytovanie údajov o nadmorskej výške a identifikácii, pričom automatické závislé sledovanie vysielania (ADS-B) pridáva globálne navigačné údaje, ktoré sa zvyčajne získavajú z prijímača globálneho pozičného systému (GPS).

Polohové a identifikačné údaje poskytované vysielaním režimu S/ADS-B sú dostupné pilotom a riadiacim letovej prevádzky.

Údaje v režime S/ADS-B sa rýchlo aktualizujú, sú veľmi presné a poskytujú pilotom a riadiacim letovej prevádzky spoločné povedomie o vzdušnej situácii pre vyššiu bezpečnosť, kapacitu a efektivitu. Ďalej môže poskytnúť nákladovo efektívne riešenie pre pokrytie sledovania v neradarovom vzdušnom priestore.

2.3 ER diagram

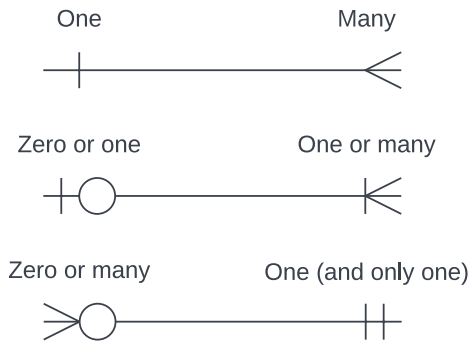
Pre znázornenie databázy, ktorá je potrebná na ukladanie starších letov sa v kapitole o implementácií využil ERD (Entity-Relationship diagram). Informácie pre túto sekciu boli čerpané z [7].

ERD je grafická reprezentácia entít a ich vzťahov medzi nimi v databáze. Používajú sa na modelovanie a dizajnovanie relačných databáz. Táto reprazentácia poskytuje vysoko úrovňový pohľad na schému databázy.

Entita je objekt alebo koncept v reálnom svete, ktorý je reprezentovaný pomocou tabuľky v databáze, napríklad lietadlo, alebo let.

Súbor entít je kolekcia podobných alebo súvisiacich entít v databáze. Inými slovami, množina entít je skupina objektov alebo pojmov, ktoré majú spoločné atribúty a ktoré možno reprezentovať ako tabuľku v relačnej databáze.

Vzťah je spojenie alebo asociácia medzi dvoma, alebo viacerými entitami v databáze. Vzťahy sa používajú na reprezentáciu spôsobu, akým sú entity navzájom prepojené. Napríklad lietadlo môže mať viac letov alebo let môže obsahovať viac časových záznamov.



Obr. 2.3: Zápis vzťahov medzi entitami

Atribúty sú vlastnosti alebo charakteristiky jednotlivých entít. Napríklad let môže mať atribúty dátum, čas alebo trvanie.

2.4 Diagram toku dát

Pre znázornenie implmentácie algoritmov na spracovanie dát sa použil diagram toku dát. Informácie pre túto sekciu boli získané z [6].

DFD (Data flow diagram) je grafické znázornenie systému alebo procesu, ktoré ukazuje, ako ním údaje prúdia a ako sa transformujú. DFD má rôzne komponenty, ktoré pomáhajú pri vizualizácii systému.

Procesy
Je súbor činností, ktoré transformujú vstupné údaje na výstupné údaje. Procesy sú v DFD znázornené ako obdĺžniky alebo kruhy. Proces sa nazve tak aby bol v niekoľkých slovách alebo frázach, ktoré vystihujú jeho podstatu.

Úložiská údajov
V úložisku, sú uložené údaje na budúce použitie, ktoré sa používajú alebo vytvárajú v procese. Keď údaje prúdia do úložiska, označuje sa to ako vkladanie údajov alebo aktualizácia údajov, a keď údaje prúdia z úložiska, označuje sa to ako čítanie údajov.

Dátové toky
Dátový tok medzi rôznymi časťami systému sa označuje ako výmena informácií. Pre znázornenie tohto toku sa používajú šípky. Na identifikáciu presúvaných informácií by sa mal toku priradiť zmysluplný názov. V danom toku by sa mal prenášať len jeden druh informácií. Smer toku symbolizuje šípka, ktorá môže byť alternatívne obojsmerná.

Externé entity
Externá entita je termín používaný na označenie entity, ktorá nie je súčasťou modelovaného systému, ale je s ním v interakcii. Príkladmi externých entít môže byť databáza alebo API. Modelovaný systém je schopný komunikovať aj s externou entitou, čo je ďalší dôležitý aspekt interakcie.

Obr. 2.4: Symboly použité v DFD

2.5 DBSCAN zhlukovanie

Pre optimalizavnie frontendovej časti aplikácie sa použilo zhlukovanie. Informácie pre túto sekciu boli čerpané z [3].

Zhlukovanie je spôsob zoskupenia súboru dátových bodov tak, aby sa podobné dátové body zoskupili. Preto algoritmy zhlukovania hľadajú podobnosti alebo rozdielnosti medzi dátovými bodmi. Zhlukovanie je metóda učenia bez učiteľa, kde algoritmus sa snaží nájsť základnú štruktúru údajov.

Techniky založené na rozdelení a hierarchickom zhlukovaní sú vysoko efektívne pri zhlukoch normálneho tvaru. Pri zhlukoch ľubovoľného tvaru alebo pri zisťovaní odľahlých hodnôt sú však efektívnejšie techniky založené na hustote.

DBSCAN (Density-Based Spacial Clustering of Applications with Noise) je zhlukovanie založené na hustote. Zoskupuje body, ktoré sú tesne pri sebe v oblastiach s vysokou hustotou, pričom sa ignorujú riedke oblasti. Tento algoritmus obsahuje dva veľmi dôležité parametre, a to epsilon a minimálny počet vzorkov.

Epsilon
: Je polomer, v ktorom algoritmus hľadá susedné body. Určuje prah blízkosti pre zhlukovanie. Body v tejto vzdialenosti od seba sa považujú za súčasť toho istého zhluku.

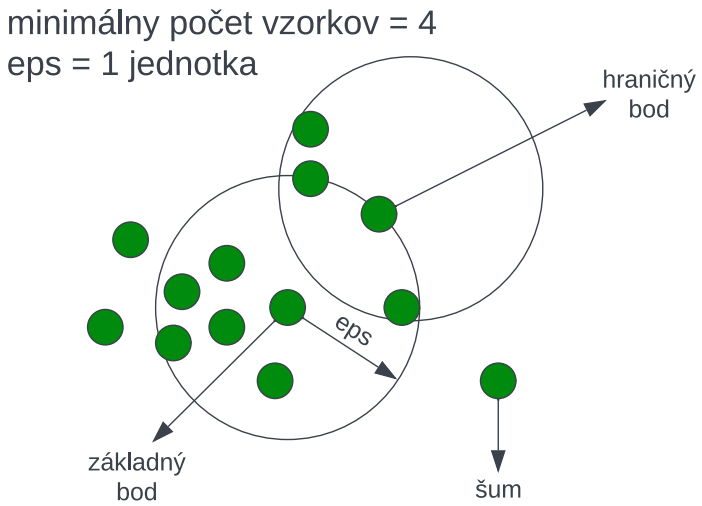
Minimálny počet vzorkov
: Je to minimálny počet bodov, ktoré sa musia nachádzať v polomere epsilon, aby sa vytvorila hustá oblasť. Ak je počet bodov v rámci polomeru menší, bod sa považuje za odľahlý.

Algoritmus zahŕňa tri typy dátových bodov, a to:

Základný bod sa klasifikuje ako jadrový bod, ak má minimálny počet bodov v polomere epsilon.

Hraničný bod je bod, ktorý má menej ako minimálny počet bodov v polomere epsilon, ale je v susedstve jadrového bodu.

Šum je bod, ktorý nepatrí do žiadnej z uvedených kategórií, sa klasifikuje ako šum alebo odľahlý bod.



Obr. 2.5: Príklad DBSCANu, prekreslené z [4]

2.6 Kontajnery

Pre jednoduché nasadenie aplikácie do produkcie sa využíva technológia kontajnerov. Informácie z tejto sekcie boli získané z [5].

Kontajnery sú samostatné a prenosné softvérové jednotky, ktoré balia aplikačný kód spolu s jeho závislosťami a knižnicami. Toto balenie umožňuje spustiť kód kdekoľvek, či už na počítači, v tradičnom IT alebo v cloude. Na dosiahnutie tohto cieľa kontajnery využívajú typ virtualizácie operačného systému (OS), ktorý využíva funkcie jadra OS na izoláciu procesov a riadenie množstva CPU, pamäte a disku, ku ktorým môžu tieto procesy pristupovať.

V porovnaní s virtuálnymi počítačmi sú kontajnery malé, rýchle a prenosné, pretože nevyžadujú hosťujúci OS v každom prípade. Namiesto toho využívajú funkcie a prostriedky hostiteľského OS, vďaka čomu sú efektívnejšie a ľahšie.

Kapitola 3

Použité technológie

3.1 Backend

Backend je potrebný na trasovanie letu lietadla, spracovanie záznamov a poskytovanie API na frontend aplikácie.

Backend aplikácie je napísaný v programovacom jazyku Python[1], ako bolo požadované v zadaní. Je to vysoko úrovňový objektovo orientovaný jazyk, ktorý vďaka veľkému množstvu knižníc umožňuje vytváranie komplexných aplikácií.

requests

Pre elegantné a jednoduché vytváranie požiadaviek na API slúži knižnica `requests`[2]. Poskytuje hlavne množstvo metód overovania ako sú napríklad OAuth a Digest, ktoré sú podstané na prístup k službám ktoré poskytújú dáta pre tento projekt pomocou API. Obsahuje taktiež metódy na spracovanie rôznych typov údajov ako je napríklad JSON.

jmespath

Pre jednoduché vyhľadávanie požadovaných údajov z dát vo formáte JSON slúži knižnica `jmespath`[3]. Umožňuje deklaratívne určiť spôsob akým sa budú jednotlivé prvky extrahovať z dokumentu `JSON`. Využíva sa v časti projektu kde je potrebné získať dáta z komplexných odpovedí z API vo formáte JSON.

Extrahovanie požadovaných dát z JSON dokumentu, ktoré môžu byť dokonca vnorené v zozname, umožňuje pomocou zadania cesty vo forme regexu. Pekný bonus tejto knižnice je, že na rozdiel od iných v prípade ak sa prvok nenašiel tak jednoducho vráti `None`.

Flask

Pre poskytovanie dát na frontend aplikácie slúži webový framework `Flask`[4]. Poskytuje jednoduchý a zároveň výkonný spôsob ako vytvoriť webovú aplikáciu. Tento framework je odľahčený a vysoko prispôsobivý, ktorý poskytuje len základné funkcie potrebné na vytváranie webových aplikácií.

[1] https://www.python.org/about/
[2] https://requests.readthedocs.io/en/latest/
[3] https://github.com/jmespath/jmespath.py
[4] https://flask.palletsprojects.com/en/latest/

V projekte sa používa ako na poskytovanie API pre frontend aplikácie tak aj na poskytovanie frontendu. Umožňuje celkom príjemné riešenie na zobrazenie optimalizovanej produkčnej webovej aplikácie vo frameworku React 3.2.

Vďaka jeho jednoduchosti, je dokonalý pre potreby tohto projektu.

SQLAlchemy

Pre vyšší level abstrakcie a modularitu sa využíva knižnica `SQLAlchemy`[5], ktorá umožňuje pracovať s databázou objektovo orientovane. To uľahčuje vytvárať komplexné dopyty a vykonávať operácie bez použitia nízko úrovňového SQL kódu. Taktiež je priamo integrovaná s viacermými knižnicami, v tomto prípade s frameworkom Flask, ktorý sa využíva na poskytovanie API.

Jedná sa o ORM (Object-Relational Mapping), kde jednotlivé tabuľky sú reprezentované v pythone ako objekty, nad ktorými je možné vykonávať CRUD operácie.

scikit-learn

Na vytváranie zhlukov v pythone pomocou algoritmu DBSCAN sa používa knižnica `skicit-learn`[6], ktorá obsahuje implementáciu tohto algroritmu. Knižnica je jednoduchá na použitie a konzistentná v rámci použitia medzi modulami implemnetovanými v tejto knižnici.

pytz

Na výpočet časovej zóny a posielanie časových údajov aj na iné platformy slúži knižnica `pytz`[7]. Dôvod použitia je vďaka podpore väčšiny časových zón a možnosťou ho priamo používať so vstavanou knižnicou `datetime`.

3.2 Frontend

Frontend aplikácie poskytuje užívateľské prostredie a je napísaný v programovacom jazyku TypeScript[8], keďže je vyvýjaný vo frameworku React 3.2.

React

Na vytváranie webových užívateľských rozhraní je celkom populárna `JavaScript` knižnica `React`[9], vytvorená spoločnosťou Facebook. Vďaka rozsiahlemu množstvu nástrojov a knižníc, je pomerne rýchle vytvárať užívateľské rozhrania. Umožňuje opakovane vytvárať použiteľné komponenty, ktoré možno kombinovať a vytvárať tak komplexné užívateľské rozhranie.

Knižnucu som si vybral hlavne kvôli jednoduchosti a modulárnosti. Pre vykreslovanie dynamicky sa meniacich elementov je možné použiť `hooks`, ktoré uľahčujú písanie stručnejšieho a opakovane použiteľného kódu.

V kóde sa hlavne využívajú funkcie `useState` a `useEffect`.

[5] https://www.sqlalchemy.org/
[6] https://scikit-learn.org/stable/
[7] https://pythonhosted.org/pytz/
[8] https://www.typescriptlang.org/
[9] https://react.dev/

`useState`
: Umožnuje pridať hodnotu do funkčného komponentu. Ako argument berie počiatočnú hodnotu a vráti pole s dvoma prvkami a to aktuálnu hodnotu stavu a funkciu s ktorou je možné tento stav aktualizovať.

```
const [value, setValue] = useState<number>(0);
```

Výpis 3.1: Príklad použitia `useState`

`useEffect`
: Umožňuje pridať takzvaný životný cyklus do funkcionálneho komponentu. Môže sa spustiť na začiatku použitia, aktualizácií alebo pred ukončením komponentu. Ako prvý argument je funkcia, ktorá sa zavolá po vykreslení komponentu. Druhý argument je pole závislostí, kde keď sa zmení stav nejakej závislosti sa vykoná funkcia z prvého argumentu.

```
useEffect(() => {
  // some action
}, [value]);
```

Výpis 3.2: Príklad použitia `useEffect`

Leaflet

Pre vytvorenie mapy bola použitá JavaScript knižnica Leaflet[10], ktorá umožňuje vytváranie mobilných webových aplikácií s interaktívnymi mapami. Poskytuje jednoduchý framework s možnosťami, ako sú napríklad markeri, vyskakovacie okná a interakcie ako napríklad priblíženie.

Jedna z hlavných výhod leafletu je jednoduchosť použitia, intuitívne API a flexibilita. K týmto vlastnostiam pomáha aj široké spektrum pluginov a rozšírení, ktoré pomáhajú k prispôsobeniu mapy k vlastným potrebám.

Jedeným z použitých rozšírení je `leaflet-rotatedmarker`, čo pridáva nový atribút k markerom. Tento atribút umožňuje nastaviť uhol pod akým bude marker vykreslený.

Bootstrap

Bootstrap[11] je populárny framework pre vytváranie rezponzívneho diznajnu webovej stránky. Poskytuje už predom vytvorené príklady, ktoré je možno použiť a upraviť podľa vlastnej potreby.

Hlavný dôvod výberu tohto frameworku je, že exituje knižnica implemtovaná v reacte, ktorá poskytuje jednotlivé HTML štruktúry ako komponenty.

[10] https://leafletjs.com/
[11] https://getbootstrap.com/

wavesurfer.js

Pre krajšie zobrazenie nahrávky, kde sa priamo zobrazí tvar vlny existuje open-source knižnica `wavesurfer.js`[12] s podporou pre mnohé audio formáty. Poskytuje širokú funkcionalitu, ako napríklad prispôosobivosť tvaru vlny, anotácie alebo vytváranie takzvaných regiónov.

Regióny sú celkom užitočné a riešia problém viacerých hovorcov v jednej nahrávke, kde je ich možné vizuálne rozlíšiť.

react-bs-datatable

Knižnica `react-bs-datatable`[13], jednoduchým spôsobom dokáže zobraziť dáta vo forme tabuľky. Možnoť jednoducho upraviť mnou potrebnú funkcionalitu a pomerne dobrej dokumentácií, bola jednou z mála, ktoré boli použiteľné.

MUI

Na filtrovanie letov so záznamom je potrebné zadať časové rozmedzie a vzhľadom na to, že základné formuláre v HTML na dátum a čas veľmi nespĺňajú požiadavky dizajnu na dnešné pomery je použitá knižnica MUI[14]. Konkrétne sa z tejto knižnice používajú iba formunáre na čas a dátum. Poskytuje vysokú prispôsobiteľnosť a vytváranie vlatných štýlov čo robí implementáciu k použitému dizajnu projektu jednoduchou.

d3-scale-chromatic

Knižnica `d3-scale-chromatic`[15] poskytuje sekvenčné a divergentné a kategorické farebné schémy. Pomocou tejto knižnice je možné si namapovať istý interval na získanie farby z vybranej farebnej schémy.

V prípade implementácie aplikácie sa to dá dokonale použiť na mapovanie výšky v istom bode letu. Umožňuje to farebne rozlíšiť výšku lietadla v istom bode letu, takže pre lepšie vizualizovanie trasy sa jednotlivé priamky vykreslia s farbou na základe tejto výšky.

3.3 Kontajnerizácia

Kontajnery sú dokonalým riešením pre vytvorenie prenosnej aplikácie, ktorá poskytuje spôsob, ako zabaliť a izolovať softvérévové závislosti do jedného balíka. Poskytuje konzistentné prostredie nezávisle od infraštruktúry.

Dockerfile

V súbore `Dockerfile`[16] sa definovaním setu inštrukcií vytvára obraz (image), ktorý sa používa na vytvorenie kontajnera. Medzi inštrukcie patria kopírovanie, inštalovanie závislostí a konfigurácia kontajnera.

[12] https://wavesurfer-js.org/
[13] https://github.com/imballinst/react-bs-datatable
[14] https://mui.com/
[15] https://github.com/d3/d3-scale-chromatic
[16] https://docs.docker.com/engine/reference/builder/

Manažovanie kontajnerov

Súbor `compose.yml`[17] sa používa na manažovanie viacerých kontajnerov ako jednu aplikáciu. Čo je vhodné pre prípad projektu, kde je potrebné prepojiť kontajner pre databázu s kontajnerom pre aplikáciu.

[17] https://docs.docker.com/compose/gettingstarted/

Kapitola 4

Získavanie dát a ich spracovanie

V tejto kapitole je popísané aké boli použité API na získavanie informáccií relevatných pre aktuálne lety.

4.1 Aktuálny stav letu

Na zobrazenie letov je potrebné poznať ich pozíciu, smer a iné parametre.

OpenSky Network API

Na diagrame 2.2 je znázornené, ako prebieha zber dát od feedorov do OpenSky Network a jeho komunikácia so službou SpokenData. V časti diagramu je taktiež znázornené spracovanie dát z OpenSky Network, kde pomocou API sa zasielajú údaje na spracovanie a taktiež získajú spracované údaje. Na diagrame je znázornené pripojenie mojej aplikácie na API OpenSky Network.

Na získanie aktuálneho stavu letu je použité API od OpenSky Network, ktoré poskytuje informácie nie len o práve prebiehajúcich letoch, ale aj o minulých letoch pre konkrétne letiská v istom časovom intervale a iné. Pre tento problém som si, ale zvolil možnosť získavania aktuálneho stavu jednotlivých letov, vzhľadom na to, že ostatné možnosti API neboli spoľahlivé sto percent času a mali isté obmedzenia.

OpenSky Network poskytuje informácie o aktuálnom lete vo forme stavového vektoru, ktorý obsahuje dáta z ADS-B a Mode S správ. Aktuálny stav letu je zhrnutie všetkých informácii o sledovaní v istom časovom bode.

```
$ curl -s "https://opensky-network.org/api/states/all" | python
   ↪  -m json.tool
```

Výpis 4.1: Volanie API na získanie stavov letov

```
{
  "time": 1681648556,
  "states": [
    [
      ...,
      "SWR831R ",
      "Switzerland",
      1681648556,
      ...
    ],...
  ]
}
```

Výpis 4.2: Príklad odpovede z OpenSky Network API

Odpoveď tohto volania je vo formáte `JSON`, kde je zaujímavá časť s informáciami o letoch, ako sú jednoznačná identifikácia lietadla `ICAO24`, volací znak letu, pozícia lietadla, čas kedy boli tieto údaje zaznamenané, výška v akej sa lietadlo nachádza a iné. Na základe týchto informácií je možné jednotlivé lety trasovať.

Na získanie týchto informácií slúži modul `api/opensky.py`, ktorý obsahuje triedu pre vytváranie API volaní a následne ich ukladanie do objektov typu `StateVector` v správnom formáte. Konkrétne sa využíva metóda `OpenSkyApi.get_all_state_vectors`, ktorá vráti zoznam týchto objektov.

```
class OpenSkyApi:
    def get_all_state_vectors(
        self,
        icao24: Optional[list[str]] = None,
    ) -> Optional[list[StateVector]]:
        ...

api = OpenSkyApi()
state_vectors = api.get_all_state_vectors()
```

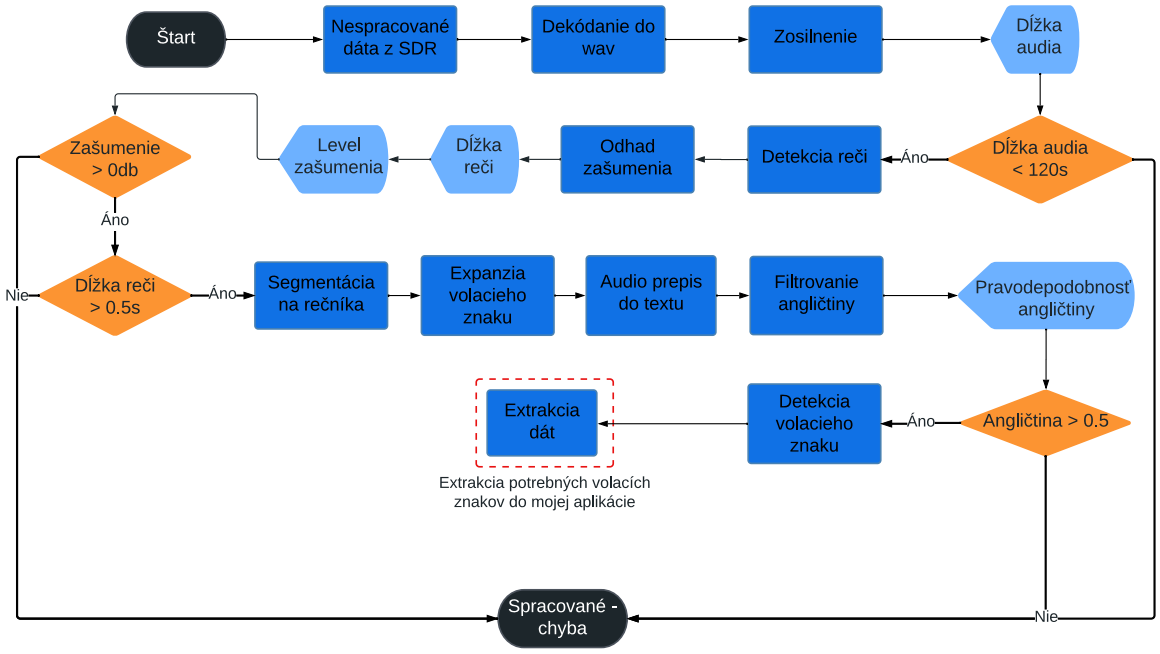
Výpis 4.3: Príklad použitia OpenSkyApi

Spracovanie dát z OpenSky Network API

Získané dáta vo formáte `JSON`, je možné previesť do štruktúr používaných v pythone pomocou metódy `json` v rámci modulu `requests`, ktorý sa v tomto príapde používa na vytváranie `GET` volaní. Odpoveď je spracovaná v cykle, ktorý formátuje dáta a vytvorí nový `StateVector` a priradí ho do výsledného zoznamu.

4.2 Získavanie nahrávok s prepismi

ATCO2 a SpokenData API



Obr. 4.1: Tok spracovania nahrávok v ATCO2 a prepojenie toku na moju aplikáciu (poskytnuté ako interná správa z ATCO2 projektu)

Na obrázku vyššie 4.1, je znázornené spracovanie zvuku v ATCO2 pipeline a následné napojenie mojej aplikácie, odkiaľ sa získavá prepis nahrávky.

Služba SpokenData poskytuje API na získavanie týchto prepisov. Obsahuje informácie o aktuálnom stave jednotlichých prácach na prekladoch, kedy bola komunikácia zachytená, z akého letiska bola komunikácia zachytená, odkaz na záznam komunikácie a jej prepis.

```
{
  ...,
  "speaker_tags": [
    ...,
    {
      "id": "D",
      "label": "RYR53C"
    }
  ],...,
  "segments": [
    {
      "start": 0.13,
      "end": 4.44,
      ...
      "words": [
        {
          "label": "Ryanair",
          "start": 0.13,
          "end": 1.12,
          ...
        },...
      ]
    },...
  ]
}
```

Výpis 4.4: Príklad prepisu nahrávky zo SokenData

V príklade vyššie 4.2, je uvedený príklad prepisu nahrávky. Tento prepis sa skladá z troch hlavných častí.

Hovoriaci

Pod kľúčom `speaker_tags`, sa nachádzajú detekovaní hovoriaci pre jednotlivé segmenty záznamu. Veža môže v rámci jednej nahrávky komunikovať s viac, ako jedným pilotom a pre rozoznanie kto komunikuje v akom čase nahrávky slúži segmentácia. Na základe tejto informácie je možné zistiť kto komunikuje a aké volacie znaky sa v nahrávke vyskytujú.

Bohužiaľ detekcia hovoriacich nie je dokonalá a niektoré segmenty sú označené, ako neznáme.

Segmenty

Jednotlivé segmenty je možné nájsť pod kľúčom `segments`, kde sú podstatné informácie v akom čase sa segment začína, končí a zoznam detekovaných slov.

Na základe času začatia a ukončenia segmentu, je ich následne možné zvýrazniť v užívateľskom prostredí.

Slová

Jednotlivé slová je možné nájsť v sekciách, kde na základe časového údaju začatia a ukončenia je možné ich namapovať na časy v nahrávke. Tým je možné vytvoriť animáciu zvýrazňovania slov.

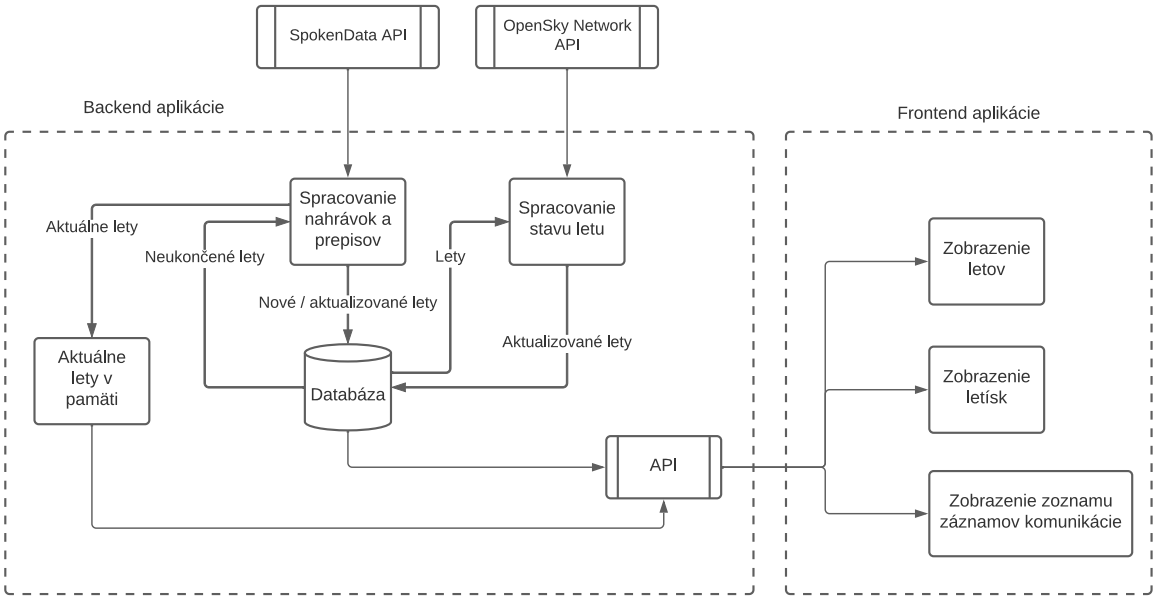
Pre zasielanie požiadavkou na službu SpokenData slúži modul `api/spokendata.py` v ktorom sa nachádza trieda pre vytváranie API volaní a ich následné spracovanie do typov, ktoré je možné používať v pythone. Pri API volaní je možné špecifikovať `limit`, teda počet záznamov, ktorý sa obdrží a taktiež `offset`, od koľkého záznamu sa začne aplikovať parameter `limit`.

```
class SpokenDataApi:
    @__check_params(["limit", "offset"])
    def get_jobs(self, **kwargs) -> dict[str, Any]:
        ...

api = SpokenDataApi()
jobs = api.get_jobs(limit=10, offset=30)
```

Výpis 4.5: Príklad použitia SpokenData API

Kapitola 5

Implementácia



Obr. 5.1: Schéma aplikácie

Celkovú štruktúru programu by som rozdelil do štyroch častí.

- Spravovanie letov s informáciami z OpenSky API
- Prideľovanie záznamov k letom s dátami zo SpokenData
- Poskytovanie API pomocou frameworku Flask
- Frontendová časť napísaná pomocou frameworku React

V prvom rade je dôležité, aby časť, ktorá poskytuje užívateľské rozhranie bola neustále dostupná a nebola prerušovaná prípadnými spracovavaniami dát z API. Preto som sa rozhodol tieto časti implementovať ako nezávislé vlákna pomocou knižnice `threading`.

```
class OpenSkyThread(Thread):
    def __init__(self) -> None:
        Thread.__init__(self)
        ... # custom attributes

    def run():
        # custom implementation on how to run the thread
        ...
```

Výpis 5.1: Príklad implementácie vlastného vlákna ako triedu

Tejto funkcionality je docielené tak, že časti pre spracovanie sú implementované v triedach, ktoré následne dedia z triedy `Thread` z modulu `threading`. Tieto tiredy implementujú vlastnú funkcionalitu potrebnú na spracovanie daných údajov ako aj prepísanie metódy `run`. Táto metóda obsahuje kód, ktorý cyklicky spúšťa metódu, ktorá slúži na spracovanie dát a ich následné aktualizovanie do databázy. Týmto sa zabezpečuje zapúzdrenie kódu, čo potom umožňuje jednoduchšie manažovanie a udržovanie kódu. Príklad tejto implementácie je uvedený na kóde vyššie 5.

Samozrejme celok tvoria aj pomocné moduly pre manipulaćiu s databázou, poskytovanie dát z vyššie uvedených API a iné.

5.1 Backend

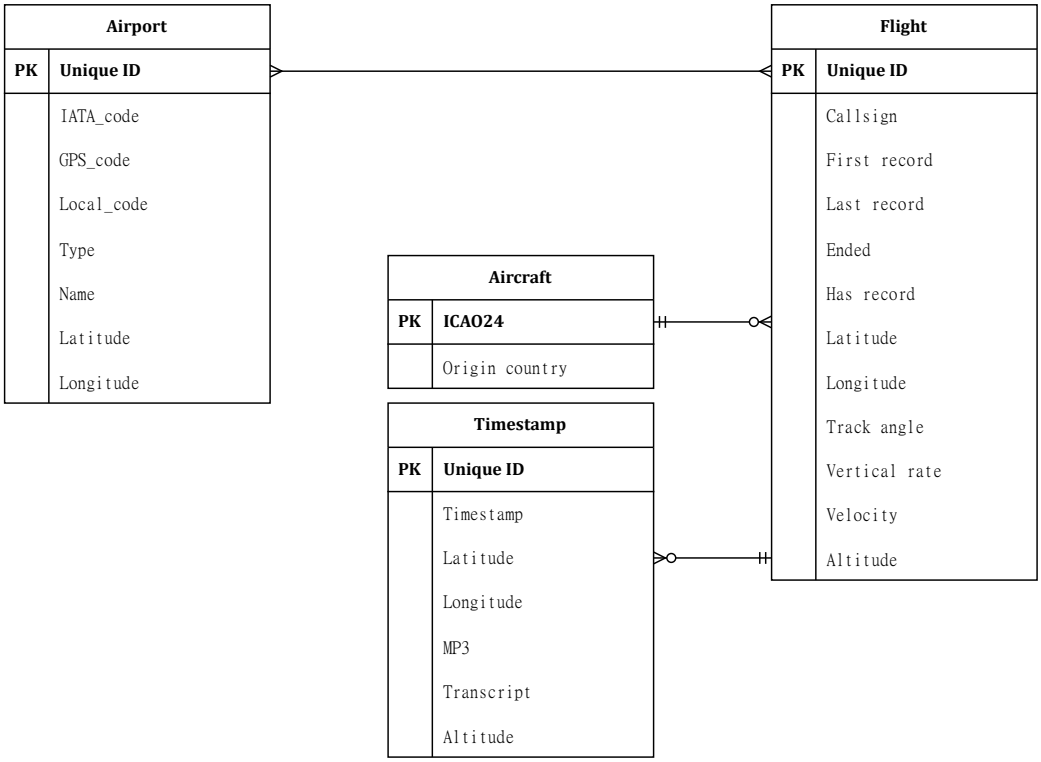
Backendová časť aplikácie je implementovaná v priečinku `backend/`. Cesty v tejto sekcií budú relatívne k danému priečinku.

Databáza

Databáza slúži na ukldanie záznamov jednotlivých letov, aby si užívateľ vedel prezerať lety, ktoré už boli ukončené.

Pre tento účel využívam MySQL databázu, keďže pri prvotnej implementácií s NoSQL databázou nastal problém s celkovou veľkosťou, keďže sa taktiež ukladajú k letu jednotlivé časové záznamy ďalej len timestamps.

Reprezentácia pomocou ER diagramu



Obr. 5.2: ER diagram

V diagrame vyššie 5.2 sa používa na vyjadrenie vzťahov takzvaný `crow's foot` zápis. Význam tohto zápisu bol spomenutý vyššie v teoretickej časti 2.3.

Reprezentácia v kóde

```python
class Airport(Base):
    __tablename__ = "airport"

    # primary key
    id: Mapped[int] = mapped_column(primary_key=True)

    # columns
    iata_code: Mapped[str] = mapped_column(String(10), nullable=
        ↪ True)
    ...
    latitude: Mapped[float] = mapped_column(nullable=False)
    longitude: Mapped[float] = mapped_column(nullable=False)

    # relationship between Airport and Flight model
    detected_flights: Mapped[list["Flight"]] = relationship(
        secondary=association_table, cascade="all, delete"
    )
```

Výpis 5.2: Ukážka implementácie tabuľky použitím SQLAlchemy

Vo vyššie uvedenom príklade 5.1 je reprezentovaná tabuľka s letiskami, s použitím ORM. Trieda `Airport` je mapovaná na tabuľku `airport` v databáze, čo je špecifikované atribútom `__tablename__`. Atribút `id` reprezentuje primárny kľúč, s použítím argumentu vo `primary_key=True` vo funkcíí `mapped_column`. Jednotlivé stĺpce v tabuľke sú namapované obdobne na typy ktoré sa používajú v pythone. Vzťah medzi tabuľkami je definovaný pomocou atribútu `detected_flights`, v tomto prípade sa jedná o vzťah `many-to-many`, pomocou spojovacej tabuľky `association_table`.

Jednotlivé modely sa nachádzajú v module `database/models.py`.

Vytváranie dotazov

Modul `database/session.py` slúži na vytváranie prepojenia s databázou a jednotlivých dotazov, ktorý obsahuje pomocnú triedu `Session` na vytvorenie novej relácie s databázou a vytváranie dotazov.

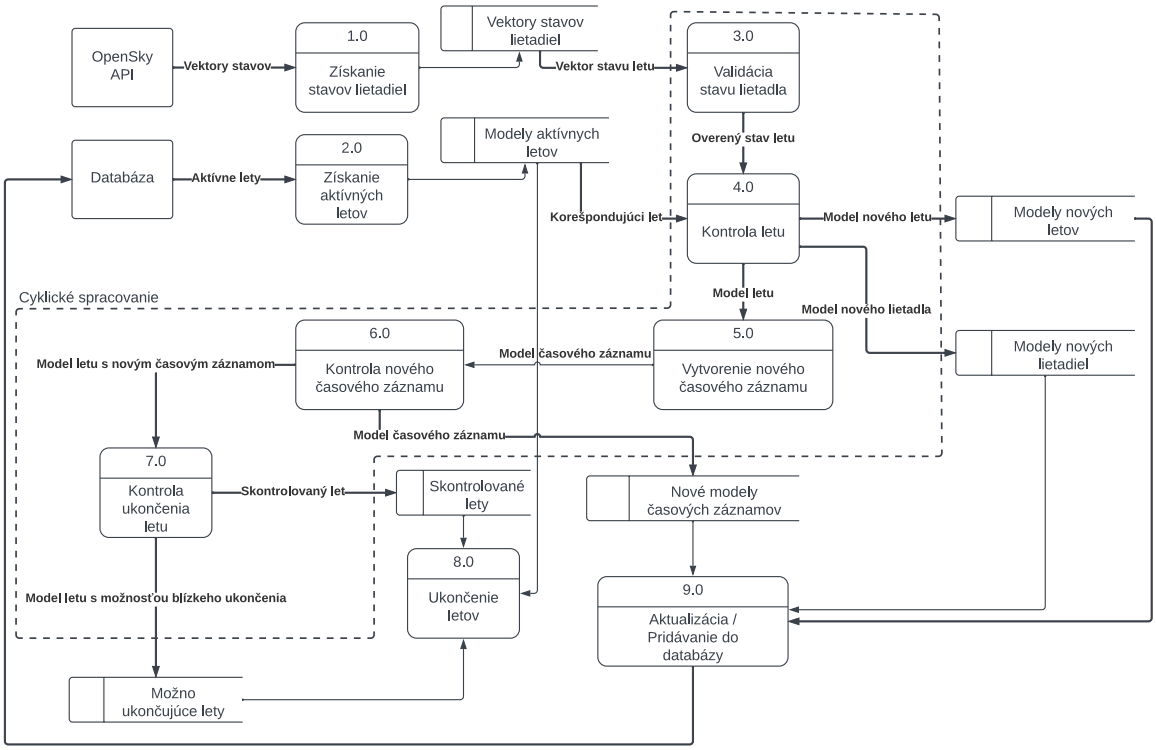
```python
class Session:
    # create thread safe session
    def __init__(self) -> None:
        self.session = Scoped_session()

    # query all rows in airport table
    def get_airports(self) -> list[Airport]:
        return self.session.query(Airport).all()

session = Session()
airports = session.get_airports()
```

Výpis 5.3: Ukážka dotazu s použitím SQLAlchemy

Vo vyššie uvedenom kóde trieda `Session` vytvára bezpečnú reláciu v rámci vlákna v magickej metóde `__init__`. Taktiež implementuje metódu `get_airports`, pre vytvorenie dotazu na získanie všetkých riadkov v tabuľke `airports`, ako objekty typu `Airport`.

Spracovanie aktuálnych letov

V súbore `threads/opensky.py` je implementované spracovanie letov, kde kód na spracovanie je v triede `OpenSkyThread` a je spúšťaná ako samostatné vlákno.



Obr. 5.3: Diagram toku dát pre spracovanie letov

Diagram 5.3 je podrobnejšie popísaný v texte nižšie.

Získanie dát o letoch z API a databázy (1, 2)

Na začiatku získajú dáta z OpenSkyApi uložené v lokálnej premennej `state_vector`. Ak je odpoveď validná, sa následne získajú modely aktívne modely letov z databázy a uložia do atribútu triedy `__flights`.

Validácia údajov o lete (3)

Pre ďalšie spracovanie je dôležité, aby sa pracovalo s validnými údajmi. Konkrétne lety, ktoré majú nevalidný tvar callsignu, chýbajúce údaje o pozícií alebo nemajú časový údaj kedy boli tieto údaje získané sa nebudú používať. To znamená, že daná iterácia pre ďalšie spracovanie sa preskočí.

Kontrola či sa jedná o nové lety (4)

Táto časť je implemtovaná v metóde `create_flight_check`, ktorá berie ako parameter `state_vector`, z ktorého sa získajú údaje potrebné na kontrolu, či sa jedná o nový let alebo nie. V každom prípade vracia model letu, buď nového alebo existujúceho v atribúte `__flights`. Ak sa jedná o nový let tak ho pridá do zoznamu nových letov `add_flights`. V prípade, že sa jedná aj o nové lietadlo tak sa pridá do zoznamu nových lietadiel do atribútu `add_aircrafts`.

Kontrola nových časových údajov (5, 6)

Z údajov v `state_vector` sa vytvorí nový model `Timestamp`, ktorému je pridelené `id` z modelu letu ktoré bolo získané v časti vyššie ako cudzí kľúč. V metóde `last_contact_check` sa timestamp sa pridá do zoznamu nových timestampov `add_timestamps`, ak spĺňa isté požiadavky. Ako napríklad novšieho času získania informácií, ako bol posledný v danom lete, prejdenie istej vzdialenosti a validitov. Validita v tomto prípade kontroluje, či lietadlo mohlo za rozdiel nového a posledného času prejsť danú vzdialenosť. Toto bolo nutné implementovať, keďže dáta neboli vždy validné hlavne v okamihu pristávania.

Kontrola možného ukončenia letu (7)

V tejto časti kódu sa v metóde `possible_ending_check` kontroluje, či lietadlo možno pristáva. Bohužiaľ dáta z OpenSky Network neobsahujú údaj či sa let ukončil alebo, či lietadlo pristáva. Takže na základe klesania lietadla a jeho aktuálnej rýchlosti sa zisťuje či sa let ukončuje. Do atribútu `possibly_ended_flights` sa v prípade možného ukončenia letu pridá model konkrétneho letu.

Ukončenie letov (8)

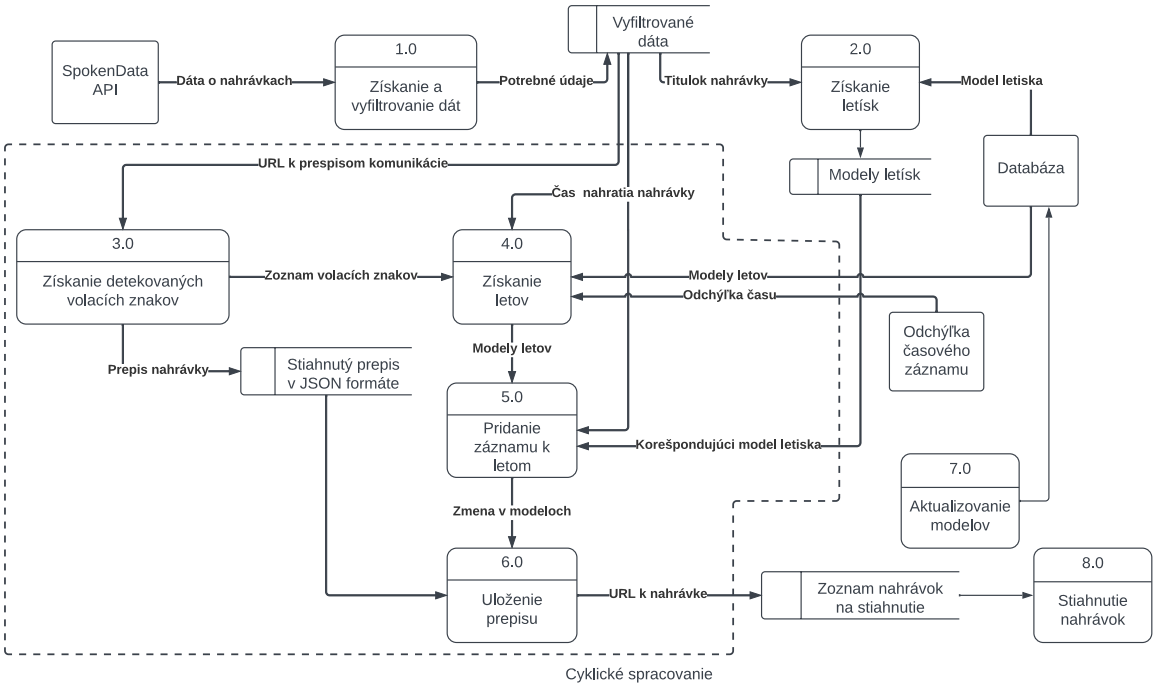
V metóde `remove_flights_check` prebieha ukončovanie letov. Kontrolujú sa iba lety, ktoré neboli skontrolované to znamená, že o nich neboli získané žiadne údaje. Let je ukončený iba v prípade, že sa nachádza v zozname `possibly_ended_flights` a posledný údaj o ňom je starši ako 2 minúty. Alebo ak poslendný časový údaj o lete je starší ako 5 hodín.

Aktualizácia databázy (9)

Následne pomocou metódy `update_flights_db_state` sú nové modely pridané do databázy. Modely, ktoré boli zmenené, sú taktiež aktualizované v databáze.

Spracovanie záznamov

Jednotlivé záznamy komunikácie sú hlavnou záležitosťou tohto projektu, ich spracovanie je implementované v súbore `threads/spokendata.py`, v triede `SpokenDataThread` spúšťané ako samostatné vlákno.



Obr. 5.4: Diagram toku dát pre spracovanie záznamov komunikácie

Získanie dát o záznamoch (1)

Na začiatku sa získajú údaje zo SpokenData pomocou metódy `get_all_valid_jobs`, kde metóda vracia zoznam štvoríc, a to titulok, čas nahrávky, url k prepisu, url k zvukovému záznamu.

Získanie modelov letísk (2)

Následne sa z jednotlivých titulkov extrahujú kódy letísk a na základe kódov sa získajú jednotlivé modely letísk z databázy.

Získanie detekovaných volacích značiek (3)

Pre získanie volacích znakov slúži metóda `get_matched_callsigns`, ktorá berie ako argument url k prepisu danej nahrávky a vracia dvojicu obsah prepisu a zoznam detekovaných volacích značiek. Obsah prepisu obsahuje dáta aké boli zachytené volacie značky v nahrávke. V prípade, že znaky neboli nájdené tak sa iterácia ďalej nevykonáva a pokračuje sa s ďalšou nahrávkou.

Získanie letov v intervale (4)

Akonáhle je známy zoznam volacích značiek, je možné z databázy získať konkrétne lety s rovnakými volacími značkami. Táto implementácia je v metóde `get_flights_in_range`,

ktorá berie ako argumenty zoznam volacích značiek, čas, v ktorom bola nahrávka získaná a interval. Interval určuje okolie okolo času nahratia nahrávky a na základe tohto intervalu sa získajú iba lety, ktoré majú timestamp v tomto intervale, keďže lety môžu mať rovnaké volacie značky. Metóda vracia zoznam letov, ktoré majú volacie znaky zo zoznamu získaných znakov a majú časový údaj v danom intervale.

Pridávanie záznamu k letu a uloženie prepisu (5, 6)

Pre pridanie záznamu k letu slúži metóda `new_timestamp_record`, v ktorej sa nájde najbližší timestamp k času kedy bola nahrávka nahratá a pridá potrebné informácie k modelu. Taktiež pridá let k letisku, kde bola nahrávka získaná.

Ak sa pridali informácie k modelu letu, tak sa uloží prepis a pridá URL na stiahnutie nahrávky k zoznamu `mp3_download_data`, kde jednotlivé URL sa po ukončení cyklu stiahnu.

Aktualizácia databázy a stiahnutie nahrávok (7, 8)

Po ukončení iterácie, v ktorej sa prideľovali nahrávky k letom sa aktualizované lety nahrajú do databázy. Následne sa stiahnu nahrávky, pre zrýchlenie tohto úkonu sa sťahovanie deje vo viacerých procesoch.

Optimalizácie

Zdieľaná premenná medzi vláknami

Pre optimalizáciu, aby sa predišlo k neustálemu vytáraniu dotazov do databázy na aktívne lety zo strany frontendu aplikácie, sa vytvárajú zhluky 5.1 jednotlivých letov a sú následne uložené do zdieľanej premennej medzi vláknami `flights`.

Odstránenie starých letov

Keďže držanie letov ktoré sú ukončené a nemajú nahrávky nemá zmysel ďalej držať v databáze tak sa odstránia tak sa odstránia s nejakých časoým odstupom. Taktiež lety s nahrávkami staršie, ako špecifikovaný počet dní sa odstránia, kvôli tomu aby sa uvoľnilo miesto na disku.

Poskytovanie API a frontendu aplikácie

Aby bolo možné posielať dáta na frontend aplikácie je použitý framework Flask. V súbore `flask_app/__init__.py`, sa nachádza počiatočná konfigurácia webu, ako aj nastavenie URL ciest.

Funkcie pre poskytovanie API

Funkcie pre poskytovanie API sa nachádzajú v súbore `flask_app/flask_api/api/api.py`.

```
@api.route("/flight", methods=["GET"])
def get_flight() -> tuple[Response, int]:
    flight = session.get_flight(int(request.args["id"]))
    return (
        jsonify({"flight": get_flight_info(flight)[0]}),
        200,
    )
```

Výpis 5.4: Príklad implementácie poskytovania dát

Aby sa funkcia používala na spracovávanie požiadavkov na špecifické URL, je dekorovaná dekorátorom `route`. V dekorátori je možné špecifikovať url tejto tejto funkcie a aké druhy požiadavkov je schopná spracovať.

Vytváranie zhlukov

Táto časť slúži hlavne na optimalizáciu zobrazovania vo frontend-ovej časti. Ale taktiež to poskytuje oveľa prehľadnejší spôsob zobrazenia letov na mape. Keďže zhlukovanie je náročné na výpočet, je implementované na strane serveru aplikácie. Princíp je pomerne jednoduchý, na základe predom určených prilížení sa vypočítajú zhluky na každý jeden.

Pre vytváranie týchto zhlukov bol najviac optimálny algoritmus DBSCAN 2.5.

```
eps = 0.1 * 2 ** (-0.7 * zoom) # optimal eps based on the zoom
    ↪ level
clustering = DBSCAN(
    eps=eps, min_samples=2, algorithm="ball_tree", metric="
        ↪ haversine"
).fit(points_rad)

# labeled clusters
labels = clustering.labels_
```

Výpis 5.5: Príklad použitia algoritmu DBSCAN v pythone

Zhluky sa vytvárajú zvlášť pre každé jedno priblíženie, kde údaj o priblížení je predávaný ako argument `zoom` do funkcie `get_cluser`. Epsilon som určil postupným skúšaním pridania argumentu `zoom` do rovnice tak, aby nevznikali veľké skoky medzi jednotlivými zhlukmi. Premenná `points_rad` obsahuje pozície lietadiel v radiánoch. Z pozícií lietadiel v jednotlivých zhlukoch sa spraví priemer, aby sa určila pozícia clusteru. Daná funkcia vracia zoznam zhlukov ako objekty `Cluster`, kde daný objekt je možné reprezentovať ako `JSON`.

```
{
  cluster: 0,
  position: [36.561, -91.772],
  data: [
    angle: 54.5,
    has_record: False,
    icao24:"06a12d",
    id: 727241,
    position: [31.462, -93.336]
  ],...
}
```

Výpis 5.6: Reprezentácia objektu `Cluster` ako JSON

5.2 Frontend

Prvotné rozhodnutie na vytváranie užíteľského rozhrania bolo použiť priamo Flask so štýlovaním pomocou frameworku `bootstrap` a využitím leafletu, čo sa nakoniec nestalo. Keďže kód začínal byť veľmi neprehľadný a bolo ho skoro nemožné rozumne štrukturovať. Kvôli tomu som sa rozhodol použiť react v kombinácií s leafletom, čo mi umožnilo štrukturovať kód oveľa lepšie.

Testovanie použiteľnosti aplikácie bolo v pláne spraviť pomocou dotazníka. Vzhľadom na to, že nie každý užívateľ vlastní počítač, sa aplikácia prispôsobila aj na mobilné zariadenia.

Súbory frontend-ovej časti sa nachádzajú v priečinku `frontend/src/`. Jednotlivé cesty v tejto sekcií budú relatívne k tomuto priečinku.

Vykreslovanie na mapu

Pre väčší prehľad a celkovou rozšíriteľnosťou aplikácie sa komponenty nachádzajú v priečinku `components/`. Jednotlivé komponenty sú potom ešte v ďalších priečinkoch aby sa prípadne k nim mohli komfortne a jednoducho vytvárať vlastné štýly alebo súvisiace komponenty.

```
export const Map = ({ children }: MapProps) => {
  return (
    <>
      <MapContainer>
        ...
        { children }
      </MapContainer>
    </>
  );
}
```

Výpis 5.7: Priklad funkcionálneho komponentu mapy

Hlavná komponenta `Map` do ktorej sa potom následne pridávajú zvyšné komponenty, je vykreslená na hlavnej stránke, ktorá je implementovaná v `pages/home/Home.tsx`. Kompomeny, ktoré vykresľujú značky, vyskakovacie okná a iné elementy na mapu sú následne použité ako tzv. `children` pre komponentu `Map`, sa v podstate vykreslia do tela komponenty.

Interaktívna časť užívateľského rozhrania pozostáva z dvoch pohľadov na mapu.

Pohľad na lety

Pohľad je implementovaný v komponente `live-flights/Flights.tsx`, ktorá potrebuje isté vstupné údaje na vykreslovanie. Ako je napríklad aktuálne priblíženie, hranice aktuálne zobrazenej časti na mape a iné.

Aktuálne informácie o lietadlách, ako sú zhluky, v ktorých sa nachádzajú, ich pozície, `id` letu a iné, sa využíva `useState`. Hlavná časť je získavanie aktuálnych letov z backendu aplikácie, čo je umožnené pomocou API spomenutého vyššie v tejto kapitole. Jednotlivé funkcie na vytváranie týchto volaní sú implementované v súbore `assets/util/fligths.tsx`.

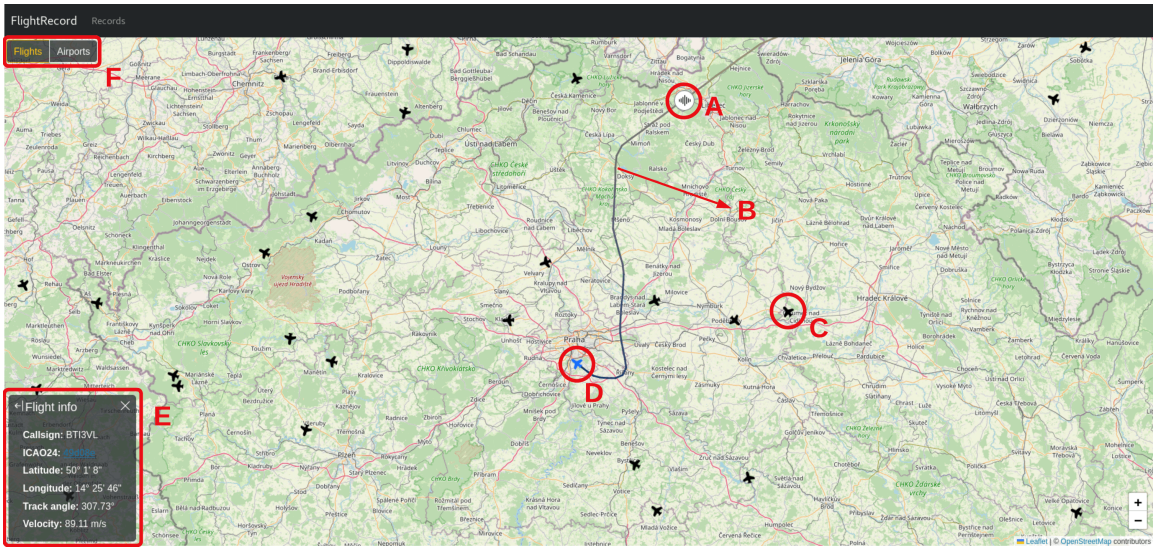
```
import { getFlightsFromAPI } from "../../assets/util/flights";

// create request
let req = getFlightsFromAPI(zoom);
// make request and handle response
req.request((response) => setClusters(response.clusters));
```

Výpis 5.8: Príklad získania zhlukov z API

V príklade vyššie 5.2 sa získajú informácie o zhlukoch z backendu aplikácie pomocou funkcie `getFlightsFromAPI`, ktorá berie ako argument aktuálne priblíženie. Keďže požiadavok sa vykonáva asynchrónne následné spracovanie odpovedi sa do funkcie ako druhý argument používa funkcia, ktorá sa má vykonať po ukončení komunikácie s backend-om aplikácie, takzvaný callback. V tomto prípade sa odpoveď zapíše do premennej `clusters` pomocou funkcie `setClusters`.

Na základe obsahu premennej `clusters` sa vykreslia jednotlivé markeri zhlukov alebo lietadiel. Obsah tejto premennej sa obnovuje každých niekoľko sekúnd.

Vzhľadom na to, že jednotlivé markeri sa vykreslia ako `DOM` elementy, čo je pomerne náročné, sa pre optimalizovanie tohto problému vykreslia iba markeri, ktoré sa nachádzajú v aktuálnych hraniciach zobrazenia.



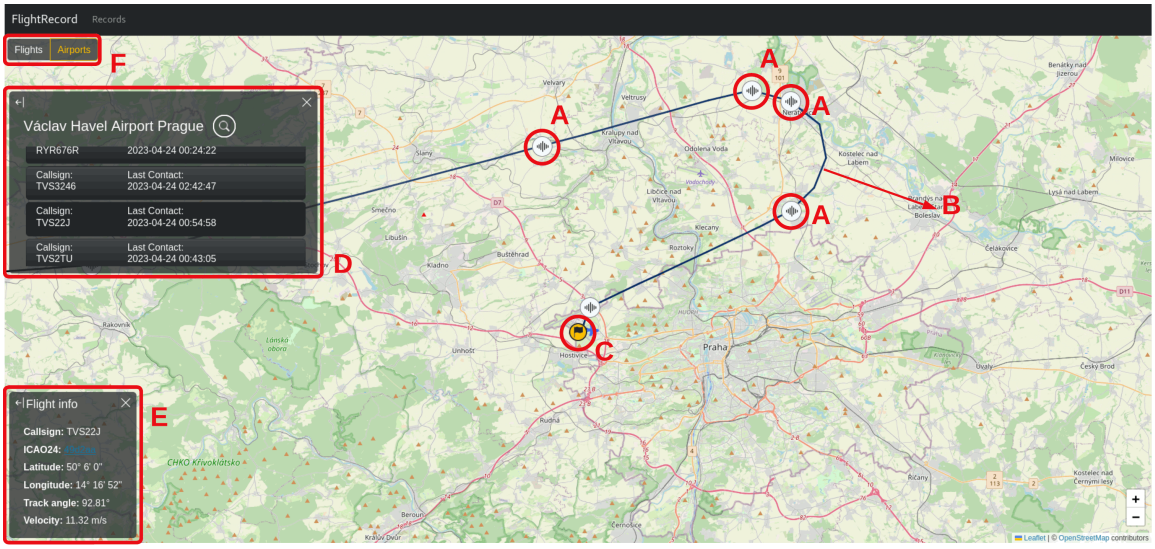
Obr. 5.5: Ukážka pohľadu na zobrazenie letov: **A** - ikonka záznamu, **B** - trajektória letu, **C** - ikonka lietadla, **D** - ikonka vybraného lietadla, **E** - základné informácie o lietadle, **F** - prepínanie medzi zobrazeniami

Pohľad na letiská

Implementácia pohľadu sa nachádza v komponente `airports/Airports.tsx`, ktorý vykreslí letiská, ktoré obsahujú aspoň jeden záznam komunikácie s lietadlom. Na získanie letísk slúži funkcia `getAirports`, ktorá vráti objekt na vytvorenie požiadavku. Na spracovanie výsledku požiadavku slúži argument funkcie, ktorý berie funckiu, ktorá sa vykoná akonáhle sa získa výsledok.

Informácie o letiskách sa nachádzajú v premennej `airports`, využívajúca `useState`.

Po kliknutí na marker letiska sa užívateľovi poskytne možnosť prehľadávať lety, ktoré boli zaznamenané v bočnom okne. Lety patriace k danému letisku sú získané pomocou funkcie `getDetectedFlights`. Jednotlivé lety je možné taktiež filtrovať od istého času, napríklad ak užívateľ si želá vidieť iba lety v istom dni stačí zadať iba deň vo formulári na filtrovanie. Lety je možné taktiež filtrovať v istom časovom intervale.



Obr. 5.6: Ukážka pohľadu na zobrazenie letísk s výberom letu: **A** - ikonka záznamu, **B** - trajektória letu, **C** - ikonka vybraného letiska, **D** - okno pre výber letu, **E** - základné informácie o lietadle, **F** - prepínanie medzi zobrazeniami

Interakcia s komponentami

Ak si užívateľ želá vidieť nahrávky pre daný let na mape tak po kliknutí na marker lietadla alebo po klinkutí na konkrétny let v zozname v pohľade letísk sa mu zobrazí cesta letu na mape. Na ceste letu sú vyznačené miesta ikonkou, kde bola zachytená komunikácia pilota s vežou. Po kliknutí na ikonku sa užívateľovi zobrazí vyskakovacie okienko, na ktorom je čas kedy bola komunikácia zachytená, volacie znaky letu a zvukový záznam.

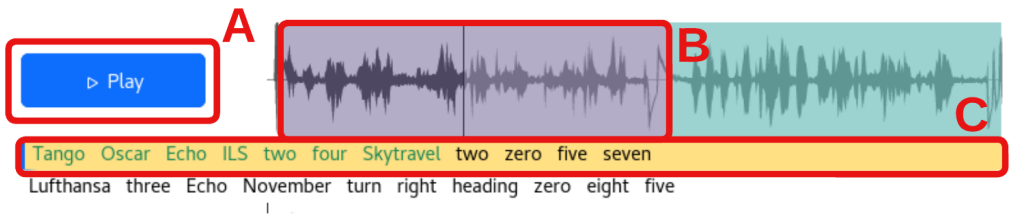
Pod nahrávkou sa nachádza prepis komunikácie, ktorý je prípadne farebne rozdelený na viac sekcií. Tieto sekcie sa vyskytujú iba v prípade, keď sa v nahrávke komunikuje s viacerými pilotmi alebo ak sa strieda komunikácia s vežou.

Pri prehrávaní sa farebne odlišujú slová, ktoré sa aktuálne vyslovujú v nahrávke kvôli lepšiemu pôžitku pre používanie. Taktiež je farbene odlíšené, aká sekcia sa akurát prehráva na upriamenie pozornosti užívateľa.

Komponenta nahrávky

Ako bolo spomenté vyššie na zobrazenie nahrávok sa používa knižnica `wavesurfer.js` 3.2.

Komponenta je implementovaná v súbore `waveform/WaveForm.tsx`, kde ako argumenty berie url k nahrávke, prepis danej nahrávky a logickú hodnotu či sa má nahrávka prehrávať po načítaní. Komponenta obsahuje tlačítko na spustenie prehrávania a jej zastavenie, kde pri prehrávaní sa podľa času upravuje štýl prepisu, ako je zvýraznenie hovoriacej sekcie a zvýraznenie slov, ktoré sa vyslovujú.



Obr. 5.7: Komponenta záznamu: **A** - tlačítko na ovládanie prehrávania, **B** - sekcia hovoriaceho, **C** - prepis sekcie hovoriaceho

5.3 Profiling

Pre optimalizovanie výslednej backendovej časti aplikácie sa pomocou profilingu vedeli detekovať pomalé časti kódu a prípadne zrýchliť ak to bolo možné. Pre zistenie času vykonávania jednotlivých funkcií slúži súbor `backend/profiling_decorators.py`, kde sa nachádzajú dekorátory pre časovanie funkcií, ale aj časovanie funkcií v cykloch.

5.4 Nasadenie

Pre jednoduchosť nasadenia aplikácie sa používa technológia kontajnerizácie 3.3. Na nastavnie portu, ktorý bude aplikácia používať je potrebné priradiť číslo portu do environmentálnej premennej `FLASK_PORT`. Aplikácia podporuje viac takýchto premenných ako napríklad pre automatickú detekciu verejnej IP adresy alebo nastavenie cesty k súboru s autentizačnými údajmi. Tieto premnné je možné definovať v súbore `deploy.env`.

Na vytvorenie obrazov a kontajnerov je potrebné mať nainštalovanú platformu na manažovanie kontajnerov ako je napríklad `docker` alebo `podman`. Na nasadenie aplikácie stačí jednoducho zadať príkaz `podman-compose up -d`.

Kapitola 6

Testovanie

Testovanie UX (User Experience) prebiehalo formou dotazníka, kde užívatelia dostali isté úlohy, ktoré mali zistiť užívateľskú prívetivosť aplikácie. Kombináciou prvkov kvalitatívnych a kvantitatívnych metodológií sa hľadalo komplexné pochopenie používateľskej skúsenosti.

Dotazník obsahuje aspekty kvalitatívneho testovania, kde užívateľ má možnosť sa vyjadriť k jednotlivým funkcionalitám. Taktiež obsahuje aspekty kvantitatívneho testovania, kde sa hodnotí funkcionalita na škále od 1 do 5 a úspešnosť vykonanie úloh.

6.1 Metodológie

Informácie pre túto sekciu boli čerpané z [2].

Kvalitatívne testovanie

Kvalitatívne údaje ponúkajú priame posúdenie použiteľnosti systému. Užívateľa pri riešení jednotlivých úloh pozorujú výskumníci, ktorí vyvodia problematické aspekty návrhu ako aj funkčné aspekty. Jedná sa o priamu interakciu s užívateľom, kde môže dostávať doplňujúce otázky.

Kvantitatívne testovanie

Kvantitatívne údaje ponúkajú nepriame posúdenie použiteľnosti návrhu. Môžu byť založené na výkone používateľov pri plnení danej úlohy (napr. miera úspešnosti) alebo môžu odrážať vnímanie použiteľnosti účastníkmi (napr. hodnotenie spokojnosti). Kvantitatívne metriky sú jednoducho čísla, a ako také sa môžu ťažko interpretovať, ak chýba referenčný bod.

Kvantitatívne údaje môžu síce povedať, že návrh nemusí byť použiteľný v porovnaní s referenčným bodom. Bohužia nepoukazujú na to, s akými problémami sa používatelia stretli alebo aké zmeny v návrhu urobiť.

6.2 Dotazník

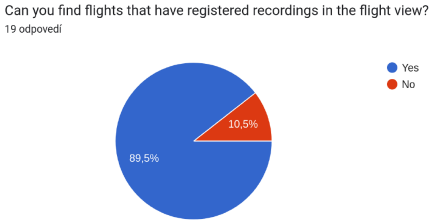
Dotazníka sa celkovo zúčastnilo 19 ľudí. V sekciách nižšie sú graficky zobrazené odpovede od užívateľov a čo sa zmenilo na základe spätnej odozvy, prípadne, čo by malo byť predmetom ďalšieho vývoja.

Informácie o použitom zariadení a prehliadači

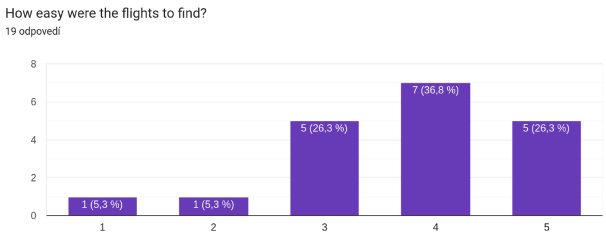
Pre efektívne zúženie oblasti, kde sa mohli prípadné chyby vyskytnúť dokáže byť otázka na druh použitého prehliadača veľmi účinná. Tak isto, ako aj na akom type zariadenia sa aplikácia použila.

Navigácia a jednoduchosť používania

Ďalšia sekcia dotazníka sa hlavne zaoberala na zistenie užívateľskej prívetivosti aplikácie, ktorá pozostávala z úloh na použitie funkcionality, ktorú má aplikácia poskytovať. Otázky sú v nasledovnej forme, napríklad nech užívateľ nájde let s nahrávkou. Užívateľ označí či sa mu podarilo úlohu splniť a následne vyplní obtiažnosť splnenia tejto úlohy.



(a) Hľadanie letov s nahrávkou na mape

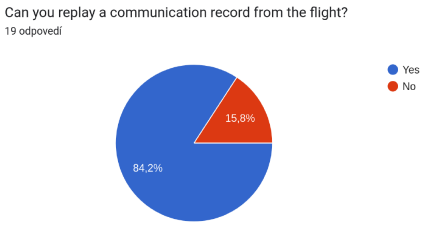


(b) Jednoduchosť funkcionality

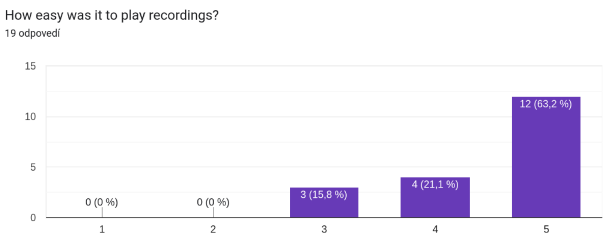
Obr. 6.1: Spätná odozva na funkcionalitu hľadania letov

Vo vyššie uvedených grafoch 6.1 je vidno, že väčšina užívateľov nemala problém nájsť lety s nahrávkami ale stále ostalo miesto na zlepšenie.

Na zlepšenie tejto funkcionality sa farebne odlišujú aj zhluky lietadiel na mape, v ktorých sa nachádza aktuálne let s nahrávkou. V prípade kliknutia na zhluk sa užívateľovi zmení priblíženie a pozícia na let, ktorý obsahuje nahrávku. Tým sa eleminuje potreba vyhľadávania.



(a) Prehrávanie nahrávok v trase letu

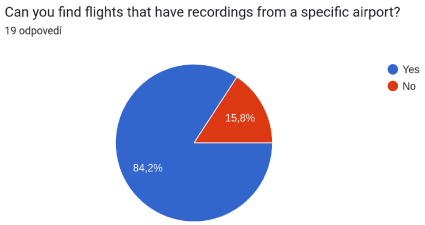


(b) Jednoduchosť funkcionality

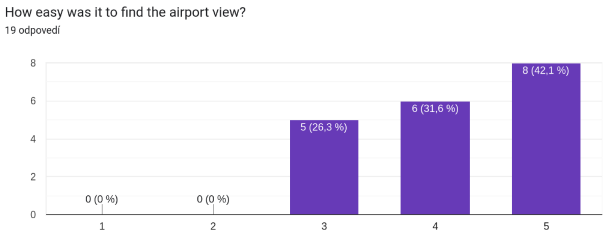
Obr. 6.2: Spätná odozva na funkionalitu prehrávania nahrávok v trase letu

Ako je vidno na grafoch vyššie 6.2, väčšina užívateľov nemala problém prehrávať nahrávky zobrazené na trase letu.

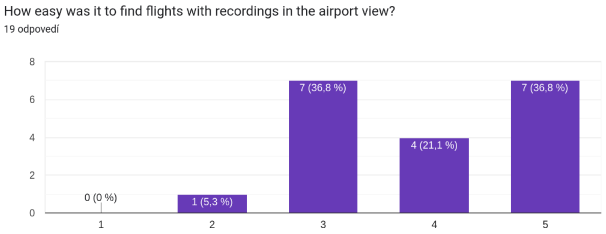
Pre zlepšenie nájdenia nahrávok, sa pridala funkcionalita, kde po kliknutí na lietadlo s nahrávkmi sa zmení pozícia zobrazenia na mape na konkrétnu nahrávku. To eliminuje potrebu hýbania sa po mape a zmeny priblíženia.



(a) Vyhľadávanie letov podľa letiska



(b) Jednoduchosť funkcionality prepnutia pohľadu na letiská

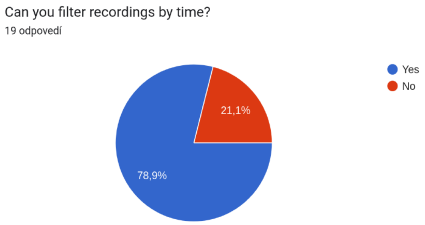


(c) Jednoduchosť funkcionality nájdenia letov v letisku

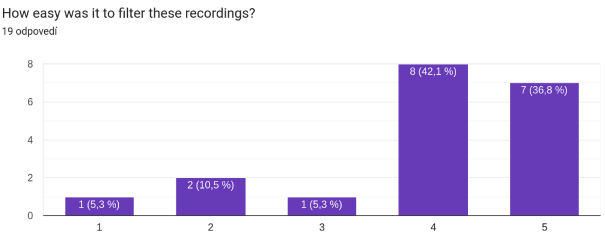
Obr. 6.3: Spätná odozva na funcionalitu prepnutia pohľadu a vyhľadávanie letov v letisku

Z grafov vyššie 6.3 je možné usúdiť, že implementácia prepnutia pohľadu na zobrazenie letísk a zobrazenie letísk s nahrávkami nie je celkom dokonalá.

Z toho dôvodu sa prepínač spravil viac výraznejší, ako bol predtým, zmenou farby a obraničenia. Marker ktorý označoval letisko na mape sa spravil výraznejším a po zakliknutí zmení dizajn, aby užívateľ vedel aj vizuálne zistiť v akom letisku momentálne vyhľadáva lety.



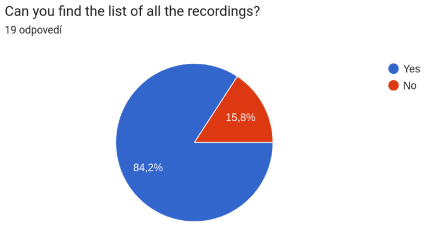
(a) Filtrovania letov podľa času



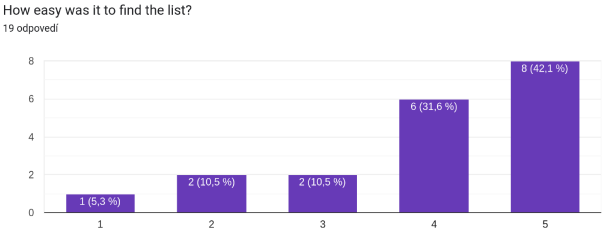
(b) Jednoduchosť funkcionality

Obr. 6.4: Spätná odozva na filtrovanie letov podľa času

Niektorý užívateľia mali problém s nájdením funkcionality filtrovania 6.4 a preto sa pomocou animácie upriamila pozornosť na ikonku filtrovania.



(a) Zoznam všetkych nahrávok



(b) Jednoduchosť funkcionality

Obr. 6.5: Spätná odozva na zoznam všetkých nahrávok

Funkcionalita

Sekcia o funkcionalite aplikácie v dotazníku slúži na získanie spätnej odozvy od užívateľa k jednotlivým funkcionalitám ako celku. Aká funkcionalita mu prišla príliš zložitá, čo mu chýbalo v aktuálnej funkcionalite a prípadne, čo by pridal.

Na základe spätnej odozvy bolo možné opraviť niektoré funkcie aplikácie, ako napríklad filtrovanie letov pre isté prehliadače. Táto sekcia poskytla skvelé nápady na zlepšenie doterajšej funkcionality.

Odozva od užívateľov k funkcionalire je prílohách A.1 A.2.

Celkový pocit z aplikácie

Na koniec sa mohli užívatelia vyjadriť k celkovej implementácií aplikácie. Podľa poskytnutých odpovedí sa na aplikácií našli isté nedokonalosti, ako napríklad veľkosť dát, ktoré sa prenášajú v istom časovom intervale.

Táto časť bola asi najviac informatívna, čo sa týka budúceho pokračovania na projekte.

Odozva od užívateľov k aplikácií ako celku je v prílohe A.3.

6.3 Vyhliadky do bucúcnosti pre projekt

Na základe spätnej odozvy od užívateľov sa navrhli nasledovné vylepšenia.

Pridanie legendy
Bolo by skvelé mať aj vysvetlenie farebného rozdelenia ikoniek lietadiel na mape. To by mohlo lepšie vysvetliť ako aj funkcionalitu aplikácie tak, aj uľahčiť orientáciu na mape.

About
Pridanie zvlášť stránky na vysvetlenie aplikácie pre lepšie pochopenie projektu, kde by sa mohli spomenúť aj použité technológie.

Bolo by dobré užívateľom oznámiť prečo exitujú nahrávky iba na určitú dobu, prečo nie sú poskytnuté bližšie informácie k letom alebo kedy prebieha aktualizácia zoznamu letov s nahrávkou v pohľade na letiská.

Veľkosť prenášaných dát
Bolo by dobré taktiež spraviť aplikáciu viac optimálnu, čo sa týka záťaže na sieť. Zasielať napríklad iba lety ktoré sa aktualizovali alebo iba lety, ktoré by sa užívateľovi zobrazili na základe jeho aktuálneho priblíženia a oblasti.

Zmena zobrazovania zhlukov
Aktuálne riešenie zhlukov, nie je veľmi prívetivé pre užívateľov, keďže prvotný zámer bol hlavne optimalizovať aplikáciu, aby bola schopná zobrazovať jednotlivé lietadlá aj na menej výkonných zariadeniach.

Filtrovanie nahrávok podľa hovoriaceho
V niektorých prípadoch sa stáva, že v nahrávke veža komunikuje s viacerými pilotmi. Nahrávka sa priraďuje k trase letu, podľa výskytu volacieho znaku a nie vždy je dokonale zachytený volací znak letu. Takže let poskytuje aj údaje nie k nemu relevantné.

Kapitola 7

Záver

Cieľom tohto projektu je vytvorenie aplikácie pre doplnenie rečovej komunikácie medzi pilotom a riadením leteckej prevádzky (ATC - Air Traffic Control) do trajektórie letu lietadla. Konkrétne je komunikácia zachytávaná na VHF (very high frequency). Inšpiráciou pre túto prácu je projekt ATCO2.

Pre trasovanie letov som si musel naštudovať dostupné služby, ktoré poskytujú API na tento účel, kde projekt OpenSky Network dokonale spĺňal požiadavky. Taktiež som si musel naštudovať API od služby SpokenData firmy ReplayWell, ktorá poskytuje nahrávky z komunikácie a jej prepis. Ďalej bolo potrebné naštudovať potrebné technológie na reprezetovanie dát na mape vo webovej aplikácií, ako aj technológie na spracovanie a ukladanie dát zo služieb spomenutých vyššie.

Na základe porovania dostupných technológií pre spracovanie dát a vyvtáranie užívateľského rozhrania boli vybrané tie najvhodnejšie. Pre urýchlenie vývoja boli k technológiam použité taktiež knižnice tretích strán.

Nabudnuté znalosti boli následne použité na návrh funkcionality aplikácie a následnú implementáciu algoritmov na spracovanie dát a webovej aplikácie pre zobrazenie týchto dát.

Testovanie funkcionality aplikácie prebiehalo vo forme online dotazníka, ktorého sa zúčastnilo 19 ľudí. Na základe výsledkov z dotazníka boli opravené isté chyby v aplikácií a získané nové poznatky na vylepšenie.

Do budúcna je potrebné upraviť istú funcionalitu, optimalizovať zasielanie dát a zmeniť zobrazovanie zhlukov.

Vďaka tejto práci som nadobudol veľa cenných znalostí, ktoré sa budem môcť zužitkovať vo svojom ďalšom pôsobení v nie len oblasti informatiky. Najväčším prínosom pre mňa bola možnosť skĺbiť frontend s backedom aplikácie a jej následné nasadenie pomocou technológie kontajnerizácie.

O prácu je prejavený záujem zo strany OpenSky Network, kde by mala byť práca nasadená do prevádzky ako ukážka.

Literatúra

[1] ATCO2. *ATCO2* [online]. ATCO2 [cit. 2023-04-25]. Dostupné z: https://www.atco2.org/.

[2] Budiu, R. Quantitative vs. Qualitative Usability Testing. [online]. Nielsen Norman Group. October 2017, [cit. 2023-05-08]. Dostupné z: https://www.nngroup.com/articles/quant-vs-qual/.

[3] Deng, D. DBSCAN Clustering Algorithm Based on Density. In: *2020 7th International Forum on Electrical Engineering and Automation (IFEEA)*. Sep. 2020, s. 949–953 [cit. 2023-05-08]. DOI: 10.1109/IFEEA51475.2020.00199. Dostupné z: https://ieeexplore.ieee.org/document/9356727.

[4] Dey, D. *DBSCAN Clustering in ML | Density based clustering* [online]. geeksforgeeks, 11. January 2023 [cit. 2023-04-18]. Dostupné z: https://www.geeksforgeeks.org/dbscan-clustering-in-ml-density-based-clustering/.

[5] IBM. *What are containers?* [online]. IBM [cit. 2023-04-27]. Dostupné z: https://www.ibm.com/topics/containers.

[6] Lucidchart. *What is a Data Flow Diagram* [online]. Lucidchart [cit. 2023-04-18]. Dostupné z: https://www.lucidchart.com/pages/data-flow-diagram.

[7] Lucidchart. *What is an Entity Relationship Diagram (ERD)?* [online]. Lucidchart [cit. 2023-04-18]. Dostupné z: https://www.lucidchart.com/pages/er-diagrams.

[8] SKYbrary. Aircraft Call-sign. [online]. SKYbrary. [cit. 2023-01-03]. Dostupné z: https://www.skybrary.aero/articles/aircraft-call-sign.

[9] SKYbrary. Automatic Dependent Surveillance Broadcast (ADS-B). [online]. SKYbrary. [cit. 2023-01-03]. Dostupné z: https://skybrary.aero/articles/automatic-dependent-surveillance-broadcast-ads-b.

[10] SKYbrary. Mode S. [online]. SKYbrary. [cit. 2023-01-03]. Dostupné z: https://www.skybrary.aero/articles/mode-s.

[11] Sparks, J. Aircraft communications. [online]. aviationpros. October 1998, [cit. 2023-01-03]. Dostupné z: https://www.aviationpros.com/home/article/10389097/aircraft-communications.

Príloha A

Odpovede od užívateľov

A.1 Obtiažnosť používania funkcionality

Sometimes, the displayed planes are not stable, for example I see some specific plane, then I zoom in and the plane disappear and few other planes appear at the zoomed area (maybe one of them is my desired plane, but it's a bit confusing).

barvy na mapě quick filtrovat jde jen čas podle stáří newest/oldest není zcela zřejmé že search okno je jen pro airplane callsign nelze kopírovat z popup oken text

the flight clustering algorithm is not that great (lots of empty space when zooming out), but that's probably not the focus of your thesis

Pohled se při většině akcí resetuje, tzn. uzavření pop-upu se záznamy resetuje zobrazení v Records, návrat do mapy zase vždy resetuje pohled mapy. Filtrování podle data nefunguje.

A.2 Pridanie funkcionality

Slovenský/český jazyk

Semantic representation of the utterances.

legenda pro mapu, co která barva znamená nejde date range filtr airport codes bych přidal drop down selection filter option

Filter by airport, help when accessing the app for the first time and most importantly sound to stop playing once I click another one and when I close the window with recordings

would be nice to be able to see more details about a flight, like the aircraft model, departure and destination airport, flight time, altitude chart, etc... or at least a link to 3rd party site which can show such data. right now it's not even possible to select (to copy & paste) the callsign.

Small map legend would be halpful (e.g. red/black airplane, recording, airport), filtering by date for Flights.

Jednotlivé záznamy by měly mít časové značky. Infobox Flight Info by mohl být větší, aby se nemuselo scrollovat. Altitude se v letectví udává vždy ve stopách.

* Departure - Arrival airports in status box of a flight in Flight view (if info available). * The "red"planes could have a little icon with a "speakerïn a corner. * I was able to play to recordings at the same time. I'd suggest to force-stop playing the 1st recording when starting playing the 2nd.

A.3 Spätná odozva k UI

What is the use exact case?

aircraft icon color coding scheme is not explained anywhere i can find (what does it mean when a particular flight is red?); it seems a bit heavy to poll >600kB of flight data every 5 seconds (regardless of the zoom level, download takes anywhere from half a second to 3 seconds!) – can the server send just the diff over a websocket subscription, with the full snapshot sent as the initial message?; related to the previous point, most of the time the server fails to send any flight data (overloaded?) but the response code is still 200.

Viz předchozí odpovědi, plus v Records jaksi chybí nejnovější záznamy, není zřejmé, kdy se seznam aktualizuje a proč tam chybí cokoli staršího, než 1 den. Výslovnost některých číslic je v letectví záměrně jiná, např. 9 se vyslovuje 'niner'. Ale to je detail. Celkově to není špatné, UX chce určitě doladit.

The app is very nice. It is fascinating to see the communication linked to a particular position of the airplane on its trajectory and the automatic transcript. The other flight monitoring apps don't have this functionality.

Maybe one suggestion to the visual design of the app. The "red"planes could have a "golden"color like the other items already existing in the UI.

Príloha B

Obsah priloženého média

```
/
├── flight-record
├── plagat.pdf
├── README.md
├── src-latex
├── technicka-sprava.pdf
└── video.mp4
```

Zložka `flight-record`, obsahuje zdrojový kód aplikácie, súbor `plagat.pdf` je plagát formátu A2 prezentujúci prácu, súbor `README.md` obsahuje návod na použitie aplikácie, zložka `src-latex` obsahuje zdrojový kód technickej správy, súbor `technicka-sprava.pdf` je preložená technická správa vo formáte PDF a `video.mp4` je súbor s krátkym videom, ukazujúce výsledok práce.