Vysoká škola ekonomická v Praze

Fakulta informatiky a statistiky


# Tvorba modelu strojového učení pro rozpoznání záměru z rádiové komunikace ATC

DIPLOMOVÁ PRÁCE

Studijní program: Informační systémy a technologie

Specializace: Vývoj informačních systémů

Autor: Bc. Michal Marhan

Vedoucí diplomové práce: Ing. Jan Mittner, Ph.D.

Konzultanti diplomové práce: Ing. David Muschalik a Ing. Jan Stádník

Praha, duben 2024

## Poděkování

Chci poděkovat vedoucímu práce Ing. Janu Mittnerovi, Ph.D. za dlouhodobou spolupráci a entusiasmus v oblasti strojového učení a létání. Zároveň mým dlouholetým kamarádům Ing. Davidu Muschalikovi a Ing. Janu Stádníkovi za odborné konzultace v oblasti řízení letového provozu a pochopení letecké radiokomunikace.

## Abstrakt

Tato diplomová práce se zaměřuje na vývoj modelu strojového učení pro převod řeči na text specificky upravený pro českou leteckou komunikaci, který dosahuje vyšší přesnosti než obecné modely. Výsledky ukazují, že navržený model strojového učení pomocí trénování modelu Whisper Large v3 od OpenAI dosahuje mnohem nižší slovní chybovosti (WER 9 %). Vzhledem k chybějícím volně dostupným datasetům pro učení práce dále obsahuje návrh a realizaci rozsáhlé architektury pro automatický sběr, ukládání a zpracování rádiové komunikace v zóně letového provozu na českých letištích. Práce poskytuje detailní popis hardwarové části architektury pro spolehlivý příjem letecké komunikace s minimální úrovní šumu. Softwarová část popisuje a implementuje škálovatelnou architekturu pro nasazení zařízení internetu věcí s možností vzdálené správy napříč českými letišti včetně backendové části pro sběr a zpracování velkého množství leteckých nahrávek. Následně práce navrhuje ověření konceptu pro automatickou extrakci důležitých informací, jako jsou volací značky, letové dráhy a záměr pilota, z přepsané nahrávky pomocí velkých jazykových modelů. Během celého projektu práce využívá metodiky MMSP pro vývoj softwarové části projektu a CRISP-DM zaměřeným na porozumění, modelování a zpracování dat.



## Klíčová slova

řeč na text, whisper, hugging face, letištní provozní zóna, letecká komunikace, strojové učení, internet věcí, azure speech, softwarově definované rádio, sdr, pytorch, docker, letecké pásmo, velké jazykové modely, gpt

## Abstract

Thesis focuses on the development of a machine learning model for converting speech to text, specifically tailored for Czech aviation communication, achieving higher accuracy than general models. The results show that the designed machine learning model, using OpenAI's Whisper Large v3 training model, achieves much lower word error rate (WER 9%). Given the lack of freely available datasets for training, the thesis also includes the design and implementation of an extensive architecture for automatic collection, storage, and processing of radio communication in the aerodrome traffic zone at Czech airports. The work contains a detailed description of the hardware part of the architecture for reliable reception of aviation communication with minimal noise level. The software part describes and implements a scalable architecture for deploying Internet of Things devices with remote management capabilities across Czech airports, including the backend for collecting and processing a large volume of aviation recordings. Subsequently, the thesis proposes a concept verification for automatic extraction of important information such as call signs, flight paths, and pilot intentions from the transcribed recordings using large language models. For the entire project, the thesis utilizes MMSP methodology for developing the software part of the project and CRISP-DM focused on understanding, modeling, and processing data.

## Keywords

speech to text, whisper, hugging face, atz, atc, machine learning, internet of things, air traffic communication, azure speech, sdr, software defined radio, pytorch, docker, airband, llm, large language models, gpt

# Obsah

Úvod ..... 12
1 Stav poznání ..... 13
2 Cíl práce a aplikace metodik na projekt ..... 14
2.1 CRISP-DM ..... 14
2.2 MMSP ..... 15
2.3 UML a Dokumentace ..... 16
2.4 Postup práce a definice dílčích cílů ..... 17
2.4.1 Iterace 1 – Nahrávání a sběr dat ATC ..... 17
2.4.2 Iterace 2 – Zpracování a příprava testovacích dat ..... 18
2.4.3 Iterace 3 – Průběžné trénování a implementace modelů ..... 18
2.4.4 Iterace 4 - Celkové vyhodnocení modelů a implementace ..... 18
3 Nahrávání a sběr dat ATC ..... 19
3.1 Specifikace leteckého rádia ..... 19
3.1.1 Letecké pásmo Airband ..... 19
3.1.2 AM Modulace ..... 19
3.1.3 Nové kanálové rozestupy 8,33 kHz ..... 20
3.2 Optimální hardwarová architektura klientského zařízení ..... 21
3.2.1 SDR Rádio ..... 21
3.2.2 Anténa a koaxiální vedení ..... 22
3.2.3 FM filtry, signálové propusti, zesilovače nízkého šumu ..... 25
3.2.4 Eliminace elektromagnetického šumu ..... 25
3.2.5 Zvolené řešení pro HW architekturu ..... 26
3.3 Softwarová architektura klientského zařízení ..... 27
3.3.1 Operační systém pro platformu internetu věcí ..... 28
3.3.2 Proces nasazení a provozu softwaru ..... 30
3.4 Popis softwaru pro automatizovaný sběr letecké komunikace ..... 31
3.4.1 Nahrávaný formát zvuku ..... 31
3.4.2 RTLSDR-Airband ..... 32
3.4.3 Aplikace pro zpracování audia ..... 35
3.4.4 Úložiště pro nahrávky a první iterace serveru ..... 39
3.5 Nasazení architektury na první letiště ..... 42
3.6 Shrnutí fáze konstrukce a zavedení první iterace ..... 44
4 Analýza a příprava dat ..... 47

4.1 Struktura letecké komunikace ..... 47
4.1.1 Rozdíly mezi ATZ a CTR ..... 47
4.1.2 Volací značka letiště ..... 48
4.1.3 Volací znak letadla ..... 48
4.1.4 Letištní dráha ..... 49
4.1.5 Letištní okruh ..... 50
4.1.6 Výška a QNH ..... 51
4.1.7 Záměr pilota ..... 52
4.2 Návrh datového modelu ATZ komunikace ..... 53
4.3 Návrh a implementace přepisu do aplikace ..... 54
4.3.1 Funkční a UX/UI požadavky ..... 54
4.3.2 Technické řešení frontendu ..... 56
4.4 Zavedení druhé iterace ..... 62
5 Výběr, trénování, vyhodnocení modelu strojového učení ..... 65
5.1 Analýza dostupných modelů ..... 65
5.1.1 Kritéria výběru modelu ..... 65
5.1.2 Kritéria vyhodnocování modelů ..... 65
5.2 Azure Custom Speech To Text ..... 66
5.2.1 Možnosti trénování Azure Custom Speech ..... 67
5.3 OpenAI Whisper Large ..... 67
5.3.1 Koncepty fungování modelu Whisper ..... 67
5.3.2 SpecAugment ..... 69
5.3.3 Možnosti trénování Whisper ..... 70
5.4 Export trénovacích dat ..... 71
5.5 Trénování Azure Custom Speech ..... 73
5.6 Trénování Open AI Whisper ..... 74
5.6.1 Příprava datasetu ..... 74
5.6.2 Příprava trénovacího prostředí ..... 75
5.6.3 Trénovací skript ..... 77
5.7 Výsledky trénování modelů ..... 84
5.7.1 Azure Speech ..... 85
5.7.2 OpenAI Whisper Large v3 ..... 86
6 Implementace a integrace modelu ..... 88
6.1 Integrace Azure Speech ..... 89
6.2 Integrace OpenAI Whisper ..... 91

6.2.1 Vytvoření Whisper API ..... 91
6.2.2 Implementace služby do backendu ..... 93
6.3 Extrakce metadat záměru pomocí LLM ..... 95
6.4 UI implementace přepisu ..... 99
Závěr ..... 100
Použitá literatura ..... 101
Přílohy ..... I
Příloha A: Diagram nasazení finální architektury ..... II

# Seznam obrázků

Obrázek 1 Životní cyklus procesu CRISP-DM (překlad autora) ..... 15
Obrázek 2 Role a jejich zaměření dle MMSP ..... 16
Obrázek 3 AM (amplitudová) a FM (frekvenční) modulace ..... 20
Obrázek 4 Ukázka nasazení antény Diamond D777 (archiv autora) ..... 24
Obrázek 5 Použití feritového jádra pro odstranění RFI šumu (archiv autora). Aplikace SDR++ ..... 26
Obrázek 6 Schéma zapojení klientského zařízení ..... 27
Obrázek 7 Schéma operačního systému Balena OS (zdroj balena.io) ..... 29
Obrázek 8 Ukázka hlavního panelu IoT platformy Balena Cloud ..... 29
Obrázek 9 UML Aktivity diagram procesu nasazení SW komponent ..... 31
Obrázek 10 UML Aktivity diagram zachycující logiku klientské aplikace ..... 38
Obrázek 11 Sekvenční diagram klientské aplikace ..... 39
Obrázek 12 Ukázka webového rozhraní objektového úložiště ..... 40
Obrázek 13 Prvotní verze datového modelu ..... 40
Obrázek 14 UML balíčkový diagram cibulové aplikační architektury backendu ..... 42
Obrázek 15 První nasazení prototypu v živém provozu (archiv autora) ..... 43
Obrázek 16 Kontrola nahraného audio souboru v přehrávači ..... 44
Obrázek 17 Výškový profil antény (215 m.n.m.) a letiště LKMB (260 m.n.m.) ..... 45
Obrázek 18 Ověření kvality zapojení přes server/klient SDR++ ..... 45
Obrázek 19 Ukázka letištních drah na LKMB 04/22 a 16/34 (zdroj: VFR příručka rlp.cz) 50
Obrázek 20 Letištní okruh (levý) v ATZ (tvorba autora) ..... 51
Obrázek 21 Finální datový model nahrávek ..... 54
Obrázek 22 UX drátový návrh (Wireframe) pro přepis nahrávky ..... 55
Obrázek 23 UX drátový model seznamu nahrávek ..... 56
Obrázek 24 Dodatečná obrazovka pro přehled aktivity ..... 63
Obrázek 25 Magický kvadrant s lídry v oblasti poskytovatelů cloudových služeb (Gartner, 2023) ..... 66
Obrázek 26 Ukázka spektrogramu vygenerovaného z audia ..... 68
Obrázek 27 Obecná architektura modelu Whisper (zdroj: OpenAI)[47] ..... 69
Obrázek 28 Příklad aplikace SpecAugment na Spektrogramu ..... 70
Obrázek 29 Ukázka rozhraní pro export nejnovějšího leteckého datasetu ..... 71
Obrázek 30 Azure Speech – nahrání vlastního datasetu (archiv autora) ..... 73
Obrázek 31 Ukázka nahraných přepisů v Azure Speech Studio ..... 74
Obrázek 32 Ukázka UI rozhraní pro přepis a záměr pomocí strojového modelu ..... 99
Obrázek 33 UML Diagram nasazení celé architektury ..... II

# Seznam výpisů programového kódu

Výpis 3.1 Souborová struktura RTLSDR-Airband projektu ..... 33
Výpis 3.2 Konfigurační soubor RTLSDR-Airband airspyhf.example (vlastní zpracování) 34
Výpis 3.3 Struktura klientské aplikace pro zpracování nahrávek..... 37
Výpis 3.4 Ukázka testu klientské aplikace pomocí ffmpeg ..... 42
Výpis 4.1 Ukázka použití Templ komponenty..... 57
Výpis 4.2 Ukázka použití HTMX..... 58
Výpis 4.3 Zkrácená verze nativní komponenty audio přehrávače..... 59
Výpis 4.4 Použití vlastní komponenty fl-player v HTML ..... 61
Výpis 4.5 Produkční Docker Compose soubor backendu ve druhé iteraci..... 62
Výpis 5.1 Export .zip s daty pro učení modelu v Go..... 72
Výpis 5.2 Výsledná struktura exportu .zip leteckého datasetu..... 73
Výpis 5.3 Ukázka souboru s přepisem..... 73
Výpis 5.4 Třída vlastního leteckého datasetu pro Hugging Face..... 74
Výpis 5.5 Příprava trénovacího prostředí v Linuxu pro PyTorch s CUDA ..... 76
Výpis 5.6 Skript pro start PyTorch Docker kontejneru..... 76
Výpis 5.7 Python závislosti pro trénování ..... 77
Výpis 5.8 Trénovací skript – parametry..... 77
Výpis 5.9 Trénovací skript – načtení vlastního datasetu..... 78
Výpis 5.10 Trénovací skript – inicializace Whisper Processoru ..... 78
Výpis 5.11 Trénovací skript – Inicializace modelu s konfigurací..... 78
Výpis 5.12 Trénovací skript – Předzpracování datasetu pro Whisper..... 79
Výpis 5.13 Trénovací skript – Spuštění předzpracování datasetu pro Whisper..... 79
Výpis 5.14 Trénovací skript – Výpočet chyby WER ..... 80
Výpis 5.15 Trénovací skript – DataCollator třída..... 81
Výpis 5.16 Trénovací skript – zahájení trénování strojového modelu ..... 82
Výpis 6.1 Implementace rozhraní služby pro přepis nahrávky v backendu ..... 88
Výpis 6.2 Implementace Azure Speech klienta do backendu ..... 89
Výpis 6.3 Vytvoření lokální REST API Whisper služby v Pythonu..... 91
Výpis 6.4 Implementace klienta pro Whisper API do Go backendu ..... 93
Výpis 6.5 Definice rozhraní pro službu záměru ..... 96
Výpis 6.6 Ukázka vygenerovaného promptu pro GPT model..... 98

# Seznam zkratek

| | |
|---|---|
| ATC | Air Traffic Communication |
| ATZ | Air Traffic Zone |
| IoT | Internet of Things |
| ADBS-B | Automatic Dependent Surveillance–Broadcast |
| NATO | North Atlantic Treaty Organization |
| SDR | Software Defined Radio |
| ML | Machine Learning |
| API | Application Programming Interface |
| VKV | Velmi krátké vlny |
| VHF | Very High Frequency |
| AM | Amplitudová modulace |
| FM | Frekvenční modulace |
| ICAO | International Civil Aviation Organization |
| UML | Unified Modeling Language |
| VFR | Visual Flight Rules (let za viditelnosti) |
| LLM | Large Language Model |

# Úvod

Vysílače ADS-B, pomocí kterých letadla vysílají volně dostupné údaje o svém letu, zapříčinily vznik služeb jako FlightRadar, FlightAware nebo OpenSky, které sdružují globální síť amatérských přijímačů těchto signálů. Platformy agregují a dlouhodobě ukládají přijímaná data, která lze použít pro akademický výzkum nebo zpravodajství z otevřených zdrojů. [1]

Dalším zdrojem informací o letu je komunikace pilota s věží letového provozu pomocí analogové rádiové komunikace na vlnách VKV. [2] Pilot je v komunikaci povinen udávat svůj aktuální záměr (přistání, vzlet, přiblížení, dráha) a ohlásit identifikační číslo svého letadla. [3] Na českých letištích se pro komunikaci používá oficiálně čeština. Jedná se o strukturovanou řeč, kde je množina používaných slov omezena ve srovnání se standardním dialogem mezi dvěma osobami. Strukturovanou frazeologii u nás definuje Řízení letového provozu České republiky. [4][5]

Moderní technologie, jako objektová úložiště, koncová zařízení internetu věcí nyní dávají příležitost i jedinci k vytvoření architektury pro sběr, ukládání a zpracování velkého množství dat v reálném čase po celém světě. To umožní vytvoření vlastního datasetu s českou letištní komunikací pro trénování vlastního strojového modelu. [6]

Aktuální pokročilé modely strojového učení pro převod řeči na text umožňují dotrénování ve specifické doménové oblasti i na nižším objemu dat, který je možné rozumně získat vlastními silami. [7] Cílem práce je vytvoření modelu strojového učení pro převod řeči na text letecké komunikace, který je přesnější než obecné modely s potenciálem pro vytvoření platformy pro automatický přepis letištní komunikace a rozpoznání záměru pilota v letištní zóně.

# 1 Stav poznání

Aktuální technologie pro sledování informací o stavu letu vychází hlavně z technologie ADS-B/Mode S. Letadlo určuje svou polohu pomocí satelitní navigace (GPS, GLONASS, Galileo) a pravidelně vysílá svou polohu a další související data, jako rychlost letadla, jeho výšku, což umožňuje jeho sledování. Tyto informace mohou být přijímány pozemními nebo satelitními přijímači leteckého řízení provozu jako náhrada za sekundární radarový dohled (SSR). Na rozdíl od SSR ADS-B nevyžaduje k aktivaci svých vysílání žádný dotazovací signál ze země nebo jiných letadel. ADS-B mohou také přijímat ostatní blízká letadla (nebo drony) s vybavením "ADS-B In", aby poskytovala povědomí o situaci v leteckém provozu a podporovala samostatné rozmisťování. ADS-B je "automatické" v tom, že k vyvolání jeho vysílání nevyžaduje žádný vstup pilota nebo vnějšího zdroje. Je "závislé" na tom, že spoléhá na data z navigačního systému letadla, aby poskytlo vysílaná data. [1]

Jedná se o mezinárodně uznávaný standard ratifikovaný organizací ICAO.

ADS-B ale již neposkytne aktuální stav letu při přistávání nebo vzlétání. ADS-B ani není povinný u některých typů letadel. [8]

Komunikace pilota s věží letového provozu pomocí rádiové komunikace je tak někdy jediným volně dostupným zdrojem dat o aktuálním záměru pilota. Data (záznamy komunikace) pro česká letiště, kromě Letiště Praha a Brno, nejsou aktuálně volně dostupná, jelikož se nejedná o data přirozeně se vyskytující ve strojově zpracovatelném formátu, ale o lidskou řeč. Po sběru těchto dat je tedy následně potřeba převod na strojová data. [9] Lidský dialog je možné zpracovávat metodami strojového učení.

Model strojového učení se zaměřuje na česká letiště, kde probíhá komunikace pilota s věží v českém jazyce. Cílem není vytvořit sadu modelů nebo univerzální model pro různé jazyky a dialekty. Trénovací data budou sbírána na českých letištích.

Aktuální rešerše již ukazuje průzkum stejné problematiky této výseče reality. Kateřina Žmolíková (2016) ve své práci, která se zaměřila na tvorbu modelu rozpoznávání řeči pro leteckou komunikaci, konstatovala chybějící trénovací sadu s leteckou komunikací. Pokusila se tedy vytvořit model na základě datasetu, který se co nejlépe letecké komunikaci podobal. Výsledná chybovost modelu dosahovala 29.5 % WER[1]. [9] Tedy téměř tři slova z deseti byla chybně přepsaná. Bylo ukázáno, že převod komunikace ATC na text je technicky proveditelný, avšak s vyšší mírou chybovosti při použití generických modelů převodu řeči na text. Tedy těmi, které nebyly doučené a zpřesněné pomocí trénovacích dat zaměřených právě na ATC komunikaci. Aktuální rešerše také nepotvrdila žádný volně dostupný trénovací dataset. Tato práce se tedy zaměří i na samotný proces získání vlastních dat.

[1] Word Error Rate

# 2 Cíl práce a aplikace metodik na projekt

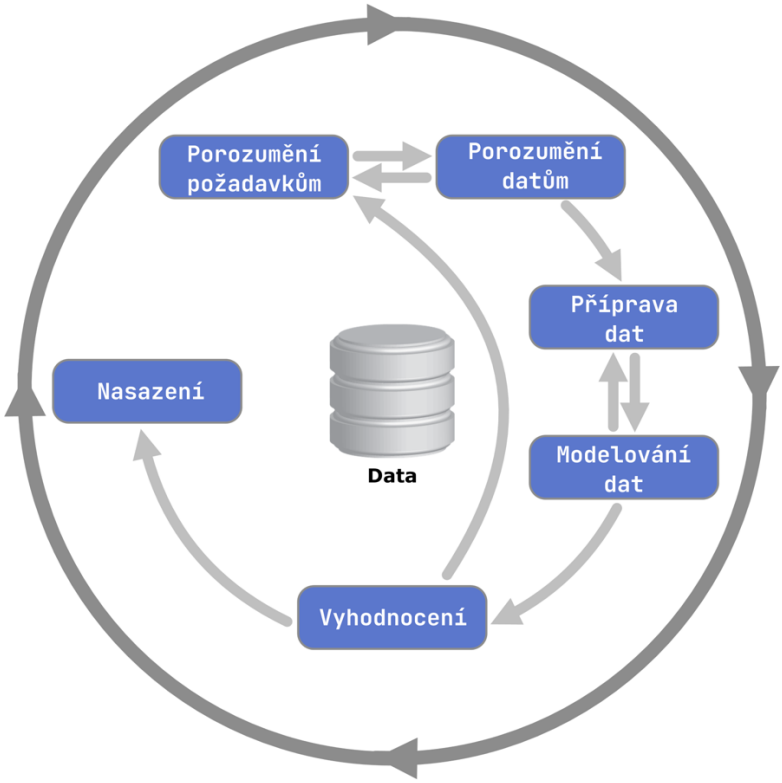
Tato diplomová práce si klade za cíl vytvořit a implementovat softwarové řešení skládající se z optimalizovaného modelu strojového učení a kompletní architektury skládající se z backendové a frontendové části, které společně umožní efektivní převod řeči na text pro rozpoznání záměru pilota z letecké rádiové komunikace (dále ATC) v letištní zóně českých letišť (dále ATZ). Cílem je zobrazit výsledky tohoto převodu v téměř reálném čase prostřednictvím frontendového rozhraní.

K dosažení těchto cílů bude v práci využito kombinace dvou metodik: CRISP-DM, zaměřené na data a jejich analýzu, a MMSP, která se soustředí na správu menších softwarových projektů. Tyto metodiky budou aplikovány tak, aby bylo možné efektivně řešit jednotlivé dílčí úkoly projektu a zajistit jeho úspěšné dokončení. Zároveň obě metodiky doplní průběžná dokumentace pomocí UML, standardizovaného jazyka pro modelování a dokumentaci nejen softwarových produktů. [10 s. 1] [11 s. 1]

## 2.1 CRISP-DM

CRISP-DM, neboli Cross-Industry Standard Process for Data Mining, je metodika pokrývající celý proces úloh a projektů zaměřující se na těžbu a zpracování dat. Tato metodika se zaměřuje na porozumění požadavkům, pochopení dat, přípravu dat, modelování, hodnocení a nasazení. Její výhodou je, že je nezávislá na konkrétních nástrojích, technologiích či odvětvích. Hodí se tedy jak pro klasické datové pumpy, tak při tvorbě modelů strojového učení. Filozofie jednotlivých fází této metodiky budou průběžně naplňovány v jednotlivých iteracích projektu. [12]

- Porozumění požadavkům
- Porozumění datům
- Modelování dat
- Vyhodnocení výsledků
- Využití výsledků




Obrázek 1 Životní cyklus procesu CRISP-DM (překlad autora)

## 2.2 MMSP

Metodika MMSP (Metodika malých softwarových projektů) je zaměřená na efektivní vývoj malých softwarových projektů pomocí agilních metod a přístupů. Tato metodika je flexibilní, umožňuje rychlé reakce na změny a přináší výsledky v krátkém čase. Proto je ideální na přípravu podpůrné aplikace pro využití natrénovaného modelu strojového učení, která bude zobrazovat historii letecké komunikace na daném letišti. Jednotlivé fáze životního cyklu MMSP se budou odrážet v iteracích projektu.

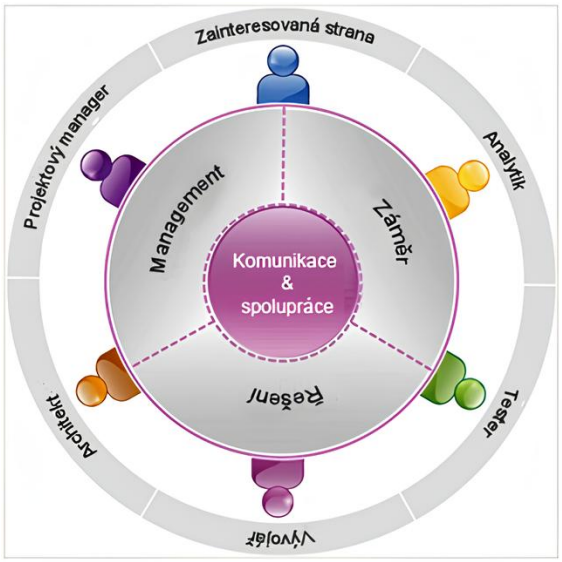
- Fáze Zahájení
- Fáze Rozpracování
- Fáze Konstrukce
- Fáze Zavedení

MMSP definuje také užitečné role, kterými se snaží charakterizovat jednotlivé členy zapojené do procesu. A poté definovat funkce a schopnosti, které má daný člen s definovanou rolí vykonávat.

- Vývojář, Tester, Analytik, Architekt, Projektový manažer

Vzhledem k projektu, kterým je diplomová práce, spadají role na autora. Ovšem jsou zde 2 role, které budou pokrývat i další členové. Role analytika a testera budou také pokrývat 2

experti pro oblast řízení letového provozu a provoz letecké techniky. Role testera je zde spíš role uživatelská, kde tito členové zabrání návrhu řešení, které by bylo zbytečné a nesplňovalo požadavky konečného uživatele – pilota nebo řídícího letového provozu.




Obrázek 2 Role a jejich zaměření dle MMSP

## 2.3 UML a Dokumentace

Při aplikování obou metodik je nezbytné důkladně dokumentovat architektonické přístupy, které byly zvoleny pro řešení problému. Práce má být zaznamenána tak, aby mohla být replikovatelná.

Metodika MMSP zakládá na pravidlu „Dokumentovat co nejpozději“, aby se zabránilo vytvoření virtuální reality, která vůbec neodpovídá stavu reálného světa. Pozdější dokumentace zachytí to, co bylo skutečně vytvořeno a co je stabilní. Na druhou stranu ale také definuje princip „Nepřetržité dokumentace“, kde by se mělo dokumentovat v průběhu celého životního cyklu produktu. [13] Kombinací těchto dvou principů se jeví jako nejlepší čas pro dokumentaci konec každé iterace projektu. Ta totiž přináší výsledek ve stabilní formě. Výhodou bude znázornění vždy malé výseče reality pro dobré pochopení problému. A v konečné fázi dojde k vytvoření vyšší úrovně dokumentace, která zachytí celou architekturu holisticky.

Kvalitní dokumentací bude dosaženo pomocí diagramů z jazyka UML (Unified Modeling Language). Tento jazyk je široce používán pro vizualizaci, specifikaci, konstrukci a dokumentaci artefaktů softwarového systému.

Pro tento konkrétní projekt budou nejvhodnější modely nasazení, modely komponent a komunikační modely. Ty umožní detailně popsat a vizualizovat strukturu a komunikaci mezi jednotlivými částmi systému.

## 2.4 Postup práce a definice dílčích cílů

Dílčí cíle jsou rozděleny dle filozofie MMSP metodiky, která podporuje flexibilitu a adaptibilitu vývojového procesu. Ta je nezbytná při výzkumném procesu, který pracuje s novými daty. Využitím iterativního návrhu je práce rozdělena do postupných iterací, které postupně navazují na zjištěné poznatky z předešlých iterací a generují funkční inkrementální přírůstky. Rozsah jednotlivých iterací bude plánován na dobu kolem 4 týdnů. Následující kapitola je návrhem technické vize dle MMSP.

Následující dílčí cíle jsou seřazeny vodopádovým návrhem. Tedy nejdříve identifikace, analýza, implementace, vyhodnocení.

DC1 – Identifikovat přístupy k automatickému sběru dat z koncových zařízení
DC2 – Identifikovat dostupné knihovny a nástroje pro trénování modelů strojového učení pro převod řeči na text
DC3 – Identifikovat přístupy vhodného návrhu frontendové architektury pro ruční zpracování dat i zobrazení výsledků strojového modelu v téměř reálném čase
DC4 – Analyzovat frazeologii letecké komunikace a identifikovat klíčové informace
DC5 – Navrhnout architekturu pro automatizovaný sběr surových dat z českých rádiových věží a ruční přepis pro získání trénovací množiny dat
DC6 – Implementovat architekturu pro automatizovaný sběr surových dat
DC7 – Implementovat architekturu pro ruční přepis a klasifikování trénovacích dat
DC8 – Implementovat model pro převod řeči na text pomocí aktuálně dostupných řešení
DC9 – Navrhnout architekturu pro živý přepis a dolování záměru pilota s frontendovým rozhraním
DC9 – Implementovat architekturu pro téměř živý přepis
DC10 – Vyhodnotit model pomocí vytvořeného prototypu a celkové řešení

### 2.4.1 Iterace 1 – Nahrávání a sběr dat ATC

Ačkoliv obdržení dat je v CRISP-DM metodice na druhém místě pro pochopení dat, tato práce bude v první iteraci řešit sběr dat. Datům ještě není porozuměno, ale již je známo, že jejich zdrojem jsou letištní nahrávky. První iterace se bude soustředit na úspěšnou identifikaci, návrh a implementaci architektury pro nahrávání a sběr dat rádiové letištní komunikace ATC. Bude obsahovat cíle DC1, DC5 a DC6.

Tento přístup byl zvolen i z důvodu, že žádná volně dostupná sada audia z českých letišť aktuálně neexistuje (nepočítaje Prahu Ruzyň, která má odlišný charakter dat). Datovou sadu si bude nutné pracně získat.

Výsledkem této iterace bude funkční automatické nahrávání komunikace ve vybrané letištní zóně. Díky tomu vznikne surový set dat připravených k dalšímu zpracování.

### 2.4.2 Iterace 2 – Zpracování a příprava testovacích dat

Druhá iterace analyzuje leteckou frazeologii z předpisů Řízení letového provozu České republiky a definuje klíčové informace, které má být schopen model strojového učení spolehlivě přepsat a extrahovat. To jsou informace, které potřebuje pro své rozhodování řídící letového provozu nebo pilot pohybující se v letištní zóně. [4]

Dle metodiky MMSP a CRISP-DM je kritické správné porozumění zkoumané oblasti dat. Tedy problémům, které řešíme, a pochopení potřeb uživatelů (pilotů a řídících letového provozu). V této iteraci budou hlavními rolemi analytická a uživatelská. Budou provedeny konzultace s Ing. Janem Stádníkem pro oblast řízení letového provozu a Davidem Muschalikem, inženýrem pro oblast leteckého provozu, provoz letecké techniky a profesionálním pilotem.

Výstupem konzultací bude připravený datový model letecké komunikace v ATZ. Poté bude navrženo rozhraní pro co nejvíce efektivní ruční štítkování dat. Výsledkem iterace bude funkční prostředí (frontendová aplikace s databází) pro poslech, přepis a štítkování leteckých záznamů.

Plně oštítkované záznamy pak budou tvořit připravené artefakty pro trénování strojového modelu. Iterace 2 tak naplní dílčí cíle DC3, DC4 a DC7.

### 2.4.3 Iterace 3 – Průběžné trénování a implementace modelů

Třetí iterace identifikuje aktuálně dostupné nejvyspělejší modely pro převod řeči na text v českém jazyce, které mohou být doučeny na vlastním datasetu. Práce chce ukázat různorodost, a proto bude záměrně vybráno jedno proprietární řešení a druhé jako volně dostupný model. Obě řešení budou doučena na sebraných datech a porovnána.

U volně dostupného modelu, kde je očekávána vyšší míra kastomizace, bude provedeno více iterací trénování pomocí různé konfigurace hyperparametrů ve snaze co nejvíce snížit chybovost modelu. Iterace také identifikuje metriky, pomocí kterých se bude průběžně vyhodnocovat úspěšnost modelů.

Při trénování modelů se iterace také zaměří na ukázkovou implementaci obou modelů do zbytku softwarového řešení. Výsledkem iterace bude z pohledu uživatele funkční aplikace, kde lze vidět všechny nahrávky z letištní zóny s možností automatického přepisu a extrakce záměru.

### 2.4.4 Iterace 4 - Celkové vyhodnocení modelů a implementace

Poslední iterace vezme již hotový produkt běžící na produkčním prostředí a zpracuje veškerá posbíraná data a navrhne další doporučení. Zároveň agreguje dokumentace z jednotlivých iterací a vytvoří holistický přehled celého řešení pomocí UML diagramů. Iterace 4 naplní cíle DC10.

# 3 Nahrávání a sběr dat ATC

## 3.1 Specifikace leteckého rádia

Následující část analyzuje specifikace leteckého rádiového vysílání pro úspěšný návrh přijímacího hardwaru. Po analýze vyšly tyto body jako důležité:

- Vyhrazené pásmo VKV frekvencí od 118 do 137 MHz
- Velmi krátké vlny počítají s propagací v přímé viditelnosti. Přijímač by měl být co nejvýše.
- Používá AM modulaci, která je náchylná na elektromagnetický šum
- Rozestup kanálů je 8,33 kHz, je vhodné korektně přepočítat frekvenci letiště

### 3.1.1 Letecké pásmo Airband

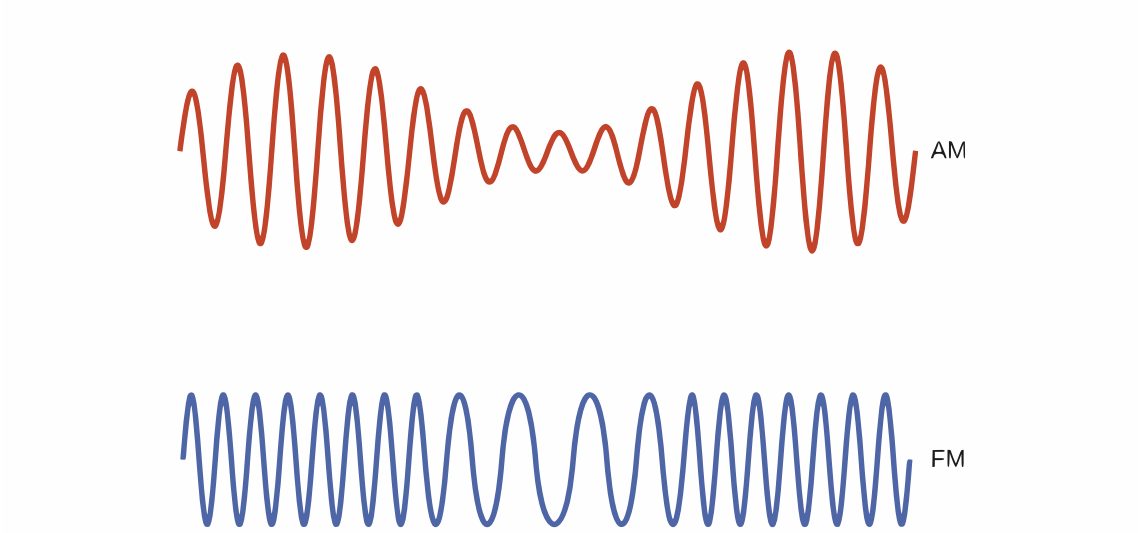
Zdrojem dat je rádiová letištní komunikace. Operuje v tzv. leteckém pásmu (Airband), což je rozmezí velmi krátkých vln VKV, anglicky VHF. Jedná se o rozsah 118-137 MHz, hned nad veřejným pásmem FM stanic. Pásmo je mezinárodně uznávané. Oficiální příručka mezinárodní letecké organizace pro civilní letectví ICAO[2] zmiňuje existenci pásma již od roku 1947.[14 kap. 7] Řízení letového provozu České republiky tento rozsah také přejímá a veškeré aktuálně používané kmitočty všech letišť jsou dostupné v příručce pravidel pro let za viditelnosti (VFR). [15]

Letištní rádiová komunikace je veřejná a nezašifrovaná, tudíž je příjem umožněn všem. Nesmí se však do tohoto prostoru v žádném případě vysílat, pokud se nejedná o pilota nebo provozovatele kmitočtu (letiště). [16]

### 3.1.2 AM Modulace

Rádiová komunikace mezi věží a pilotem probíhá v leteckém pásmu s amplitudovou modulací AM. V této práci není třeba řešit její tzv. demodulaci, tedy převod zpátky na audio. Na to jsou již dostupné hotové softwarové implementace. Je ale dobré vědět její specifika. Zatímco u FM se při kódování zvuku do signálu mění frekvence, u AM se informace přenáší změnou amplitudy. A právě to zapříčiňuje její větší šum oproti FM. Elektrické signály snadněji ovlivní amplitudu. [17 s. 315] Jelikož světlo je tvořeno také elektromagnetickými vlnami, nabízí se analogie ke světlu (které je jen frekvenčně dost výše). AM komunikuje změnou jasu světla. Mnoho věcí může blokovat světlo, čímž se může zdát méně jasné, než se očekávalo, zatímco FM komunikuje změnou barvy. Mnoho věcí může světlo ztlumit, ale málo věcí může snadno změnit jeho barvu.

[2] ICAO – International Civil Aviation Organization




Obrázek 3 AM (amplitudová) a FM (frekvenční) modulace

Bude třeba navrhnout hardwarové řešení, které je minimálně náchylné na okolní elektromagnetický šum a ideálně emituje samo o sobě šumu co nejméně.

### 3.1.3 Nové kanálové rozestupy 8,33 kHz

Od roku 2017 došlo v Evropě, včetně České republiky, k zásadní změně v alokaci kmitočtů pro leteckou VHF komunikaci. Rozestupy mezi jednotlivými kmitočty byly sníženy z původních 25 kHz na 8,33 kHz. Tato změna zvyšuje počet dostupných kanálů pro vysílání a příjem v daném pásu, a to přidáním dvou dodatečných kanálů pro každých 25 kHz, což ztrojnásobí kapacitu frekvenčního spektra. Před touto změnou používání širších rozestupů způsobovalo, že méně zatížená, neřízená letiště často sdílela stejné frekvence. To vedlo k situacím, kdy piloti v jedné letištní zóně mohli nechtěně zachytit komunikaci z jiného, vzdáleného letiště, což mohlo vést k záměně a potenciálnímu riziku kolize. [18]

Na praktickém příkladu, pokud byl původní vysílací kmitočet 118,025 MHz určen pro kanál s šířkou 25 kHz, může být nyní rozdělen na tři nové 8,33 kHz kanály s vysílacími frekvencemi 118,0167 MHz, 118,0250 MHz a 118,0333 MHz. Vzhledem k desetinnému rozvoji čísla 8,3333 kHz byly pro zjednodušení komunikace a minimalizaci chyb vysílací frekvence zaokrouhleny na čtyři desetinná místa, což usnadňuje jejich zobrazení a nastavení v radiostanicích.

Zásadním aspektem této změny je zavedení specifických kanálových čísel pro každou frekvenci, aby se snížila potenciální chybovost mezi kontrolory a piloty. Tato čísla kanálů jsou navržena tak, aby co nejvíce připomínala frekvenci, ale pro zjednodušení komunikace ve vzduchu a snížení počtu číslic, které je nutné zadat do palubního rádia, se používají pouze tři desetinná místa. Tím se podstatně zjednodušuje vyslovování frekvence a snižuje se počet číslic, které piloti musí zadat do svých radiostanic.

Zařízení pro příjem 8,33 kHz je navrženo tak, že při zadání čísla kanálu dojde automaticky k naladění na skutečnou vysílací frekvenci, což minimalizuje prostor pro chyby. Zařízení pro příjem 8,33 kHz umí naladit i kanál na 25 kHz rozestupu při použití původní frekvence. Jako na tabulce níže.

Tabulka 3.1 - Ukázka mapování 8,33 kHz kanálových rozestupů

| **Původní 25 kHz rádio** | **Nové 8,33 kHz rádio** | | | | |
|---|---|---|---|---|---|
| | Displej rádia | | | Skutečná rádio frekvence | |
| Frekvence | Displej rádia | 25 kHz frekvence | 8,33 kHz kanál | Frekvence (kHz) | Rozestup (kHz) |
| 118,000 | 118,000 | 118,000 | | 118,0000 | 25 |
| | 118,005 | | 118,005 | 118,0000 | 8,33 |
| | 118,010 | | 118,010 | 118,0083 | 8,33 |
| | 118,015 | | 118,015 | 118,0167 | 8,33 |
| 118,025 | 118,025 | 118,025 | | 118,0250 | 25 |

V oficiálních příručkách se vyskytují zaokrouhlené frekvence. Ty se také zobrazují na displejích leteckých radií v kokpitu a ve věži. Nově je tedy třeba dbát na správné naladění skutečné frekvence při stavbě vlastního řešení. Eurocontrol, což je evropská mezinárodní organizace pro rozvoj postupů a systémů pro plynulé řízení letového provozu, společně s ICAO vydala příručku, kde je tato implementace podrobněji rozepsána. [19]

## 3.2 Optimální hardwarová architektura klientského zařízení

### 3.2.1 SDR Rádio

Klasické rádio má v sobě elektroniku, která provede demodulaci přímo na úrovni hardwaru. Například převod z AM/FM na analogový audio signál, který poté pošle do reproduktorů. Tohle je jednoduché a praktické pro příjem nebo vysílání bez dalšího zpracování. Stejným způsobem fungují všechna spotřebitelská rádia do domácností. Pro další zpracování by bylo ovšem nutné výstup rádia napojit například na zvukovou kartu a stejně provést konverzi do digitálního formátu. Konverzní krok by byl jen oddálen. Zároveň takové rádio nejde rozumně ovládat na dálku, ladění se provádí ručním nastavením frekvence. Tento přístup není praktický.

Pro automatizovaný příjem rádiového vysílání, kde hlavní účel je jeho pozdější digitální zpracování (přepis řeči na text, dolování záměru), je jediným rozumným všeobecně přijímaným přístupem softwarově definované rádio. Dále jen SDR. Jedná se o druh zařízení, které se přes software naladí na zadanou frekvenci a poté jen produkuje surová data v podobě digitální reprezentace rádiového signálu. Hned po prvním kroku se jedná o čistě digitální signál.

Tento formát se nazývá I/Q data. Protože celý proces je od začátku již digitální, má své maximální „rozlišení", tedy kolik těchto bodů z analogového signálu převede do digitálního. Tomuto obvodu se říká analogový na digitální konvertor, dále jen ADC[3]. ADC hraje klíčovou roli v kvalitě signálu zachyceného SDR. Bitové rozlišení ADC ovlivňuje dynamický rozsah a schopnost zařízení rozlišit slabé signály v přítomnosti silnějších signálů. Vyšší bitové rozlišení ADC, například 12-bit a více, poskytuje větší přesnost a citlivost, což je zásadní pro aplikace jako přepis lidské řeči na text.

Výhodou softwarového přístupu ve formě SDR je implementace demodulace čistě pomocí softwarových algoritmů, které zpracuje procesor. Se stejným SDR je tedy možné přijímat jakýkoliv signál a algoritmem demodulovat AM/FM nebo jakýkoliv jiný druh dat včetně digitálních. Například i snímky z meteorologických satelitů na orbitě nebo i již zmíněný digitální ADS-B u letadel. Kombinace SDR přijímačů a softwarových dekodérů pro ADS-B například pohání celosvětové portály jako FlightRadar24.

Po ADC je dalším důležitým parametrem rozsah pásma, které SDR dokáže v jeden moment zachytávat. Jedná se o další výhodu SDR oproti klasickému rádiu. Některé SDR, jako například AirSpy R2, dokážou nahrávat až pásmo široké 10 MHz najednou. Tohle je vhodné pro letiště, která provozují několik kmitočtů najednou. Například letiště Praha Ruzyně, které provozuje kmitočty 118,310 MHz až 134,560 MHz, by tak pokryla jen 2 SDR. [20] U menších neřízených letišť, která jsou cílem této práce, se nachází jen jedna frekvence. Tímto se potřebný rozsah SDR pro projekt rovná rozsahu jednoho letištního kmitočtu – 8,33 kHz. Většina SDR umí více než 1 MHz.

Cílem je dostat co nejvíce kvalitní signál na vstup SDR. Zbytek procesu je již digitální.

### 3.2.2 Anténa a koaxiální vedení

SDR přijímač je potřeba připojit k anténě. Dobře zvolená anténa a její umístění je nejdůležitější část celé SDR sestavy. Z antény proudí veškerý signál, který se SDR snaží dekódovat. Kvalita příjmu na dané frekvenci je definována tzv. ziskem antény v decibelech – dBi. Tato hodnota je k nalezení v technických specifikacích u výrobců antén.

Optimální délku antény a její zisk je možné spočítat i teoreticky. Správná délka antény je totiž spjatá se samotnou fyzikální podstatou přijímané frekvence rádiových signálů. Anténa má být stejně dlouhá jako vlnová délka přijímané frekvence. Tu lze vypočítat jako rychlost

[3] Analog to Digital Converter

propagace rádiového signálu v médiu a dělením přijímané frekvence. Rádiový signál je elektro-magnetické vlnění, tedy má stejnou rychlost jako světlo. 299 792 458 m/s.

$$Vlnová\ délka\ (m) = \frac{300}{130} \approx 2{,}3\ m$$

Následující vzorec ukazuje, že pro efektivní příjem na frekvenci 130 MHz (přibližný střed leteckého pásma 118-137 MHz) by byla potřeba anténa o délce 2,3 metru.

Tůma (2019) ve své práci doporučuje pro příjem civilního leteckého pásma anténu D-777 od japonského výrobce Diamond. Jedná se o anténu vyladěnou právě pro civilní letecké pásmo. Její zisk pro 120 MHz je 3,4 dBi. [21]

Anténu je poté nutné připojit k přijímači SDR. Na propojení se používá koaxiální kabel. Ten má jako jádro měděný drát, po kterém putuje signál. Druh/kvalita kabelu výrazně ovlivní útlum signálu. Kabel s menším průměrem bude mít větší ztrátu signálu na metr než kabel s širším průměrem. Ztrát 3 dB se rovná ztrátě signálu na polovinu výkonu. Parametry kabelu jsou udávané v technických specifikacích výrobce. Pro práci byl použitý kabel typu H155. S průměrem 5,3 mm má útlum 0,45 dB na 5 metrů kabelu.

Obrázek 4 Ukázka nasazení antény Diamond D777 (archiv autora)

### 3.2.3 FM filtry, signálové propusti, zesilovače nízkého šumu

Jak bylo zmíněno v předchozí části, ADC převodník uvnitř SDR má zhoršený výkon v případě, že je v okolí přítomnost mnohem silnějších signálů než těch, které se snaží zachytit. Nejčastějším a naprosto běžným jevem jsou komerční rozhlasové FM stanice. Jejich vysílací výkon bývá natolik silný, že pokud se nachází ve stejném městě jako letištní zóna i vysílač rozhlasové stanice, znemožní méně kvalitnímu přijímači spolehlivý příjem leteckého pásma. I vyšší bitové rozlišení ADC nemusí stačit.

Tato překážka se řeší dvěma způsoby. Použitím filtrů nebo pásmové propusti.

Pásmový filtr (anglicky band-stop nebo také notch filter) výrazně utlumí signál, který se nachází v rozmezí pásmového filtru. Nejpoužívanějšími jsou FM pásmové filtry. Ty blokují rozsah komerčních rozhlasových stanic 88-108 MHz. [17 kap. 2]

Druhým řešením jsou pásmové propusti. Ty fungují na inverzním principu. Pustí pouze rozsah, pro který jsou vyrobené, a utlumí vše ostatní. Jedná se o lepší řešení v případě, že je cílem příjmu SDR jen konkrétní, neměnný rozsah pásma. Tedy i v tomto projektu, kdy je cílem příjem jen leteckého pásma. Konkrétním produktem může být například japonská pásmová propust pro letecké pásmo AOR ABF128. [22]

Třetím běžně používaným prvkem je zesilovač nízkého šumu, tzv LNA[4]. Ten se hodí jen v případě, že délka koaxiálního kabelu od antény k SDR přijímači je natolik velká, že by zapříčinila výrazný pokles signálu. [23 s. 153–154]

FM filtry nebo signálové propusti se dávají zásadně co nejblíže anténě, před LNA. Tedy Anténa->FM Filtr nebo signálová propust -> koaxiální kabel -> SDR přijímač. Nejdřív je totiž žádoucí eliminovat hned u vstupu veškeré nežádoucí signály a poté zesílit ty slabé, které zbývají. [17 kap. 2]

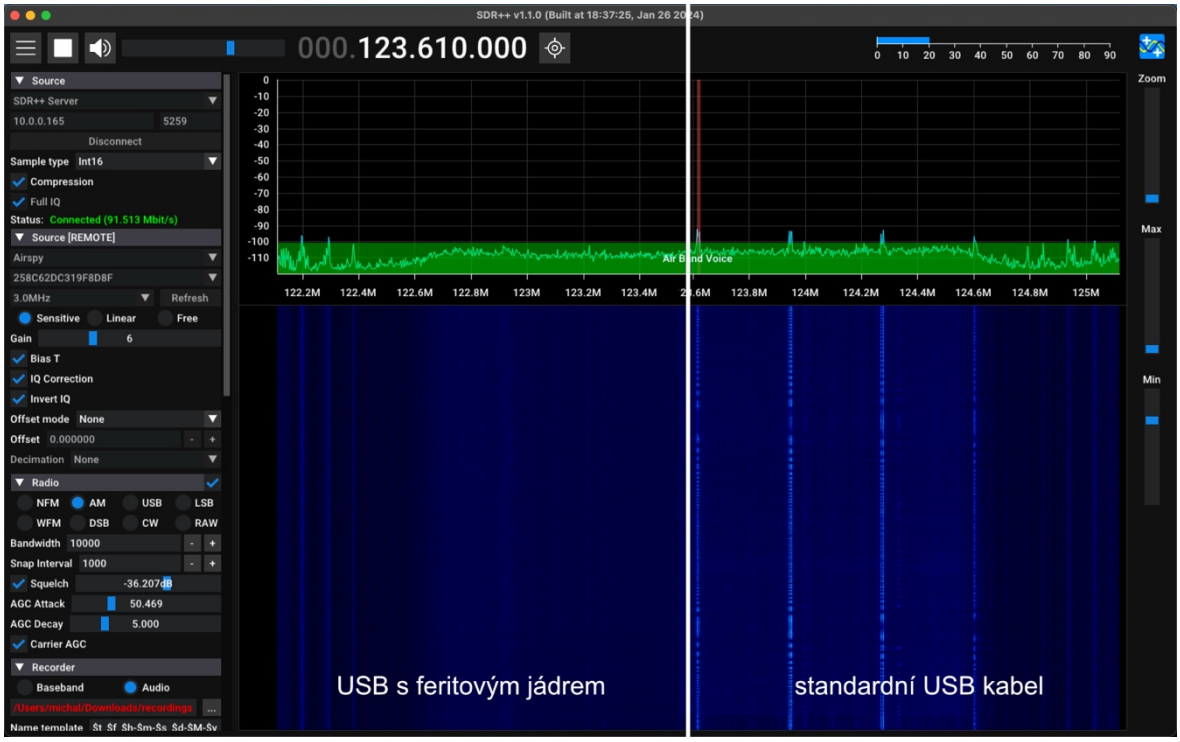
### 3.2.4 Eliminace elektromagnetického šumu

Přijímač SDR konvertuje na vstupu analogový signál do digitální formy. V kapitole 3.1.2 bylo zmíněno, že AM signál je více náchylný na elektromagnetický šum. Je potřeba dostatečně ochránit vstupní obvod SDR přijímače od okolního elektromagnetického šumu (dále jen RFI[5]). Kvalitní SDR přijímače jsou vloženy buď do kovového pouzdra, které slouží jako stínění, nebo implementují jiné techniky stínění přímo na desce tištěného spoje.

Silným emitentem RFI šumu nemusí být jen vzdálenější okolní prostředí, ale přímo počítač, do kterého je SDR přijímač zapojen. I při dobrém stínění může RFI šum doputovat přes uzemnění USB kabelu až dovnitř přijímače. Velice používaným řešením jsou feritová jádra, která se umístí hned na vnější plášť USB kabelu, kterým je SDR připojeno k počítači. Během

[4] Low Noise Amplifier
[5] Radio Frequency Interference

analýzy byla zjištěna velká hladina RFI šumu u jednodeskového počítače Raspberry Pi 4. Feritová jádra úspěšně eliminovala nežádoucí šum.




Obrázek 5 Použití feritového jádra pro odstranění RFI šumu (archiv autora). Aplikace SDR++

### 3.2.5 Zvolené řešení pro HW architekturu

Z analýzy vyplývají následující technické požadavky pro SDR přijímač:

- Rozlišení ADC alespoň 12-bit
- Dobrý příjem pro rozsah leteckého pásma 118-137 MHz
- Šířka pásma alespoň 8,33 kHz (splňuje v zásadě každé SDR)
- Integrovaná nebo dodatečná propust pro letadlové pásmo nebo FM filtr pro omezení přehlušení ADC
- Responzivní automatické nastavení zisku AGC
- Navíc podpora operačního systému linux, jelikož je to očekáváné běhové softwarové prostředí pro klientské zařízení

A pro zbytek zapojení:

- Koaxiální kabel H155 o maximální délce 5 metrů
- USB kabel s feritovými jádry
- Anténa umístěna co nejvýše
- Celá sestava musí být v oblasti ATZ

Počítač je v tomto případě irelevantní. Na provoz a demodulaci signálu z jednoho kmitočtu stačí i starší jednodesková platforma. Výhodou je zvolit řešení, které má nízké energetické

nároky. Například Raspberry Pi 4, kde je spotřeba kolem 5 wattů. Je dobré ale ověřit, zda zařízení neemituje příliš RFI šumu.


Obrázek 6 Schéma zapojení klientského zařízení

## 3.3 Softwarová architektura klientského zařízení

Druhou půlku koncepce klientského zařízení tvoří jeho softwarová část. V počáteční technické vizi bylo definované zařízení, které umí nahrávat komunikaci v letištní zóně. Zařízení tedy může být umístěné fyzicky na letišti nebo poblíž letištní zóny kdekoliv po České republice. Z toho vyplývají zvýšené nároky na stabilitu a vzdálenou správu řešení.

Architektura musí splňovat následující funkční i nefunkční požadavky:

1. Softwarová architektura musí podporovat rychlé iterace formou aktualizací na dálku[6]
2. Vzdálená správa softwaru musí fungovat, i když je zařízení umístěné uvnitř vzdálené interní sítě (NAT)
3. Architektura včetně operačního systému musí být navržena tak, aby se co nejvíce zabránilo dostat operační systém do nekonzistentního stavu, který již neumožní vzdálenou správu
4. Klientská část musí komunikovat se serverovou částí zabezpečeně s HTTPS
5. Architektura musí umožňovat vzdálené aktualizace firmwaru SDR zařízení
6. Software běžící na klientském zařízení bude tzv. lehký klient. Zpracování dat bude probíhat na serverové části
7. Součástí architektury musí být i možnost připojit vzdálené SDR zařízení na lokální počítač pro vzdálenou diagnostiku

Samotný software pro příjem a zpracování letecké komunikace musí umět:

1. Zpracovat a demodulovat AM signál z připojeného SDR zařízení
2. Mít konfigurovatelné parametry, jako frekvenci na poslech a ID letiště
3. Automaticky detekovat, kdy probíhá aktivita na letištní frekvenci. Nenahrává tichá místa
4. Nahrávat audio v dostatečné kvalitě vhodné pro zpracování strojovými modely
5. Nahrávky nejsou ukládány lokálně, ale do cloudového úložiště

[6] Over The Air Update

6. Nahrávky jsou v průběhu nahrávání ukládány pouze do RAM paměti, aby se maximalizovala životnost disku nebo SD karty v případě nasazení na jednodeskovém počítači
7. Nahrávky se posílají ihned po skončení komunikace
8. Software je rigidní v případě, že dojde k chvilkovému přerušení připojení k internetu nebo výpadku serverové části. V takovém případě jsou nahrávky uchovávány lokálně a periodicky se pokouší nahrávat
9. Software má ochrany, které omezí falešně pozitivní detekci komunikace
10. Software má definovaný horní limit délky nahrávky pro zabránění zaplnění cloudového úložiště a RAM při chybovém stavu

### 3.3.1 Operační systém pro platformu internetu věcí

Povaha klientského zařízení pro nahrávání letištní komunikace spadá do kategorie internetu věcí. Jsou tímto definována zařízení vybavená senzory nebo jiným hardware pro sběr dat a síťovou konektivitou pro výměnu těchto informací. [24] Byla provedena rešerše na dostupné platformy pro provoz s následujícími hledanými výrazy:

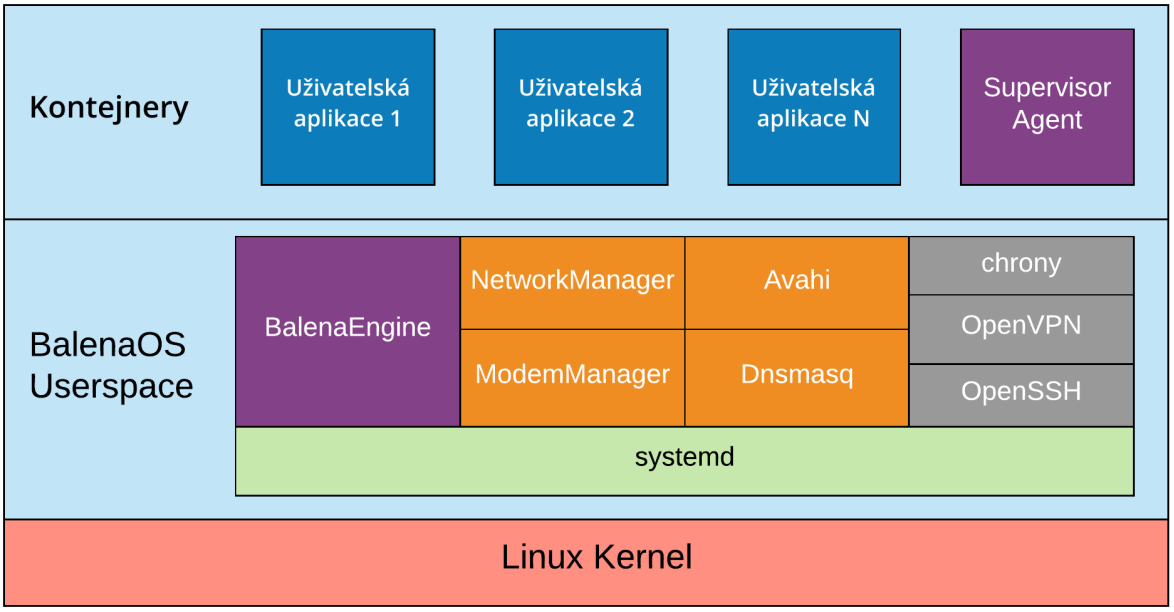
- „IoT internet of things management platform “
- „Management and deployment for IoT infrastructure “

Byla vybrána platforma od Balena Cloud, protože splňovala veškeré architektonické požadavky. Jedná se platformu pro nasazení a kompletní správu vzdálených zařízení. Součástí řešení je linuxový operační systém, který je plně doručován v režimu software jako služba (SaaS)[7]. Uživatel neřeší správu a konfiguraci operačního systému, aktualizace jsou prováděné automaticky nebo na vyžádání.

Uživatel nasazuje software pouze prostřednictvím kontejnerů. Díky kontejnerové technologii software běží odděleně se všemi jeho závislostmi ve vlastním prostředí. [25] Díky tomu je jednoduché provádět rychlé iterace. Software se nasadí jako kompletní balíček. To zaručuje, že nedojde ke konfliktu závislostí nebo jiné chybě, která dostane software nebo rovnou operační systém do nekonzistentního stavu.

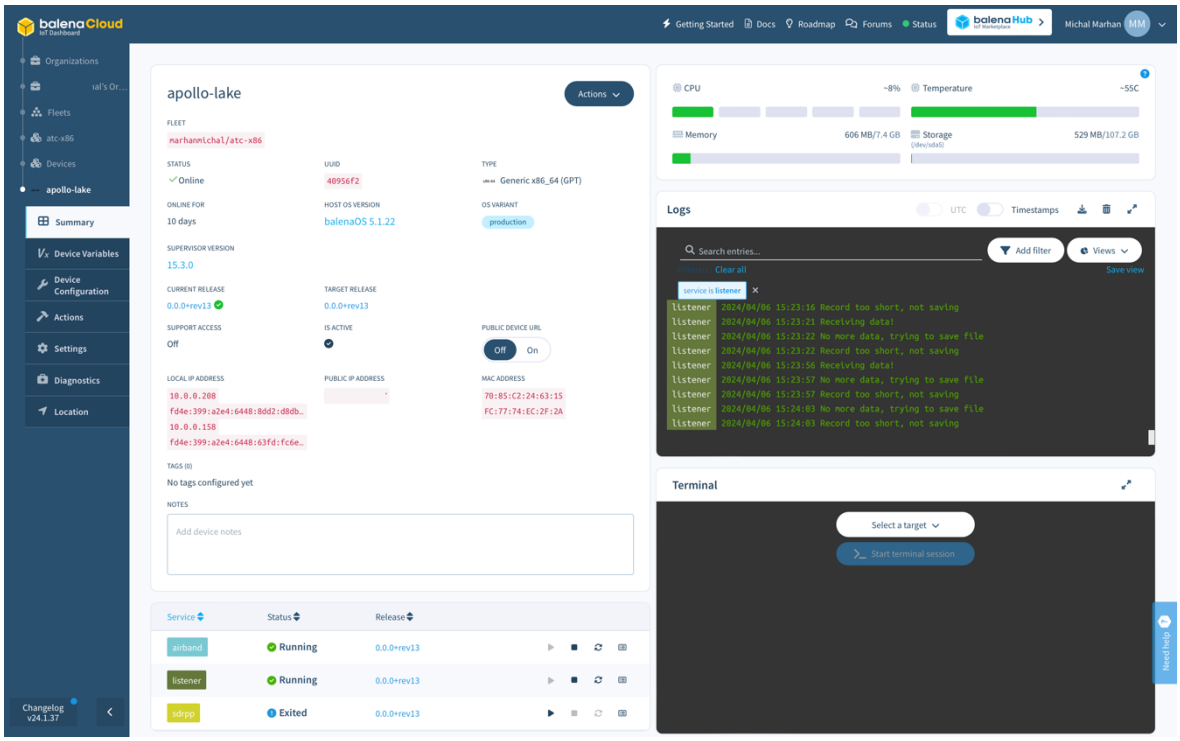
Zároveň platforma automaticky zajišťuje možnost vzdálené správy pomocí vytvoření vlastního VPN tunelu mezi servery Balena Cloud a koncovým zařízením. Uživatel má tímto možnost kompletní správy včetně přístupu k terminálu. Odpadá tím potřeba veřejné IP adresy, konfigurace firewallů a přesměrování portů. To je kritický požadavek, který umožní nasadit zařízení na jakémkoliv letišti.

[7] Software as a service




Obrázek 7 Schéma operačního systému Balena OS (zdroj balena.io)

Na obrázku výše je vidět diagram nasazovaného operačního systému Balena OS do klientského zařízení. Jedná se o minimální operační systém, který má jen nezbytné komponenty pro běh kontejnerového prostředí a síťové knihovny pro navázání vzdáleného VPN as SSH spojení.




Obrázek 8 Ukázka hlavního panelu IoT platformy Balena Cloud

Na obrázku výše je vidět aktuální prostředí Balena platformy. Klientské zařízení je online, je možné se k němu dostat pomocí VPN, které zprostředkovává Balena. Jsou vidět i aktuální výstupy z běžících aplikací (Docker kontejnerů) a možnost vytvořit SSH terminálové spojení pro ruční správu kontejnerů. Vlevo dole jsou vidět 3 nasazené kontejnery.

Počáteční analýza koncepce ukazuje, že výběrem platformy Balena Cloud budou splněny architektonické požadavky 1, 2 a 3.

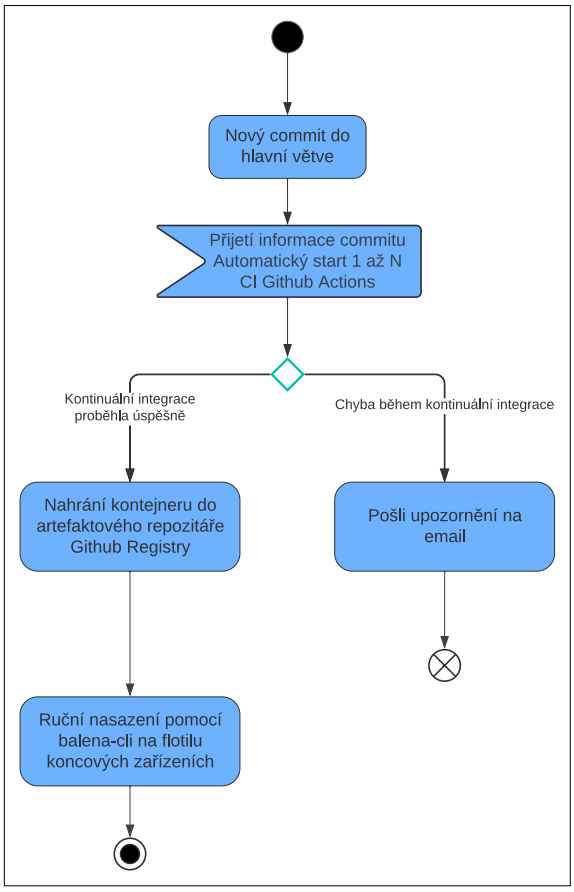
### 3.3.2 Proces nasazení a provozu softwaru

Metodika MMSP definuje i potřebu sestavení konfiguračního plánu, což je artefakt, který definuje postupy, jak se software bude verzovat, kam se bude ukládat, jak bude probíhat sestavení softwaru (tzv. buildu) a jeho nasazení do produkce. [13]

Veškerý vývoj softwaru bude probíhat v repozitářích pomocí verzovacího systému Git. Konkrétně jeho implementace ve formě SaaS služby GitHubu. Git zajišťuje bezpečné verzování každé změny v softwaru. Vývojář má na své pracovní stanici k dispozici lokální kopii projektu a poté nahrává změny pomocí tzv. commitů. Každá změna je navíc přirozeně dokumentována pomocí zpráv.

Pro zajištění konzistence v jednotlivých sestavených verzích softwaru a pro urychlení procesu sestavení a nasazení bude pro každou softwarovou komponentu celé architektury implementován systém pro kontinuální integraci na platformě Github Actions. Každý commit projde pokusem o úspěšné sestavení softwaru. V případě úspěšného sestavení bude vytvořen artefakt v podobě Docker kontejneru, a to vždy pro architektury linux/arm64 a linux/x86_64, aby byla zajištěna kompatibilita s oběma architekturami.

Kontejnerový artefakt je poté uložen do repozitáře Github Container Artifact registry. Tím je připraven na nasazení do produkce. Nebude se implementovat kontinuální nasazení, aby bylo možné ručně ovlivnit dobu nasazení nové verze softwaru.




Obrázek 9 UML Aktivity diagram procesu nasazení SW komponent

## 3.4 Popis softwaru pro automatizovaný sběr letecké komunikace

Architektura softwaru pro automatizované nahrávání letecké komunikace se skládá ze 2 částí ve vlastních kontejnerech.

### 3.4.1 Nahrávaný formát zvuku

Výsledek celého automatizovaného procesu má být dle funkčních požadavků z kapitoly 3.3 nahrávka ve formátu a dostatečné kvalitě, která je vhodná při práci se strojovými modely pro převod řeči na text.

Nyquistův teorém, pojmenovaný po švédsko-americkém inženýrovi Harry Nyquistovi, je jedním z principů v teorii digitálního zpracování signálu. Stanovuje, že pro úspěšnou digitalizaci analogového signálu musí být vzorkovací frekvence (sample rate) alespoň dvojnásobkem jeho maximální frekvence. Tím se zachytí téměř všechny informace v analogovém signálu bez ztráty. V kontextu nahrávání lidské komunikace, kde frekvence lidského hlasu obvykle nepřesahuje 8 kHz, podle teorému vychází vzorkovací frekvence 16 kHz jako optimální. Teorém je používán už od začátků telefonní komunikace. [26] Vzorkovací frekvence znamená, kolikrát za vteřinu se změří hodnota analogového signálu.

Bitová hloubka je dalším parametrem digitálních formátů zvuku. Ta zase říká, jaký rozsah lze znamenat při změření analogového signálu. Například při ukládání do hloubky 16bitového integeru je možné uložit informaci o výšce amplitudy od -32 768 až do +32 767. Tedy celkově 65 536 hodnot. Ještě větší hloubkou je použití typu float32 s plovoucí desetinnou čárkou. Ten již pro speech to text nepřináší žádné výrazné zlepšení za cenu dvojnásobného místa na disku.

Dalším aspektem formátu je použití nebo nepoužití komprese. Při poslechu digitální hudby se dnes můžeme setkat s velice rozšířeným formátem MP3. Ten je jedním z příkladů ztrátové komprese. Ztrátová komprese se snaží minimalizovat velikost souboru tím, že z dat záměrně vypouští informace, které mají na kvalitu nejmenší vliv. Při poslechu hudby z formátu se ztrátovou kompresí většina uživatelů nepozná rozdíl v kvalitě. Ale modely strojového učení, které převádějí řeč na text, nemají stále tak vynikající rozlišovací schopnost jako lidské ucho a mozek. Jak ve své práci ukázal Kang (2022), při porovnávání modelu řeči na text s použitím různé úrovně komprese docházelo k výraznému zhoršení účinnosti modelu. [27 s. 3] I oficiální dokumentace Google Speech To Text zmiňuje, že použití kompresních formátů může vést k výrazné redukci přesnosti modelu. [28]

Proto bude použitý bezztrátový formát WAV. Alternativou je použití bezztrátové komprese na zmenšení souboru se zachováním všech dat. Například FLAC. Nicméně u většiny modelů je WAV podporován přímo bez dodatečného zpracování, protože má v sobě kodek PCM, což je nejjednodušší forma reprezentace audio dat.

Definice formátu zvuku vhodného pro převod řeči na text je tedy:

- 16 kHz vzorkovací frekvence
- WAV formát s PCM kodekem
- 16bitová hloubka

### 3.4.2 RTLSDR-Airband

První částí softwarové architektury je aplikace RTLSDR-Airband zabalená do vlastního kontejneru s přibalenými ovladači, které umožní funkčnost SDR AirSpy zařízení.

RTLSDR-Airband je svobodný software, který umožňuje příjem analogových rádiových kanálů a jejich převod do podoby audio streamů. Tyto streamy je možné dále směrovat do různých výstupů. Pro tento projekt je důležitá podpora nekomprimovaného audio streamu ve formě UDP paketů. [29] Nekomprimované audio je ve formě 8 nebo 16 kHz vzorkovací frekvence s PCM kodekem a float 32 bitové hloubky, kterou lze poté zpracovat do 16 bitové.

Původně program podporoval pouze SDR zařízení typu Realtek DVB-T dongle, což se odráží i v názvu projektu. Avšak s příchodem knihovny Soapy SDR, která je neutrální vůči výrobcům SDR zařízení, se podpora rozšířila i na další typy rádií včetně zařízení od firmy AirSpy, která byla vybrána v předchozí kapitole.

Aplikace dále podporuje jedním konfiguračním parametrem i automatický squelch. Squelch je funkcí, která slouží k potlačení šumu v rádiovém přijímači v době, kdy nejsou přijímány žádné užitečné signály. Jedná se o druh prahu šumu, který automaticky ztlumí výstup zvuku

z rádia, dokud síla přijímaného signálu nedosáhne určité úrovně, což indikuje, že je přítomen užitečný signál (někdo stisknul vysílačku). RTLSDR-Airband implementuje automatický squelch. Při spuštění během prvních pár vteřin zjistí hladinu šumu a hranici squelch nastaví lehce nad tuto úroveň.

U této aplikace je potřeba pouze konfigurace bez úpravy zdrojového kódu. Aplikace byla kontejnerizovaná pomocí Dockerfile skriptu. V něm se automaticky stáhne nejnovější verze kódu aplikace a přibalí se veškeré potřebné ovladače pro funkčnost AirSpy SDR včetně adaptéru SoapySDR. Pro aplikaci byl vytvořen i dynamický konfigurační soubor. Ten má v sobě proměnnou pro změnu frekvence, která se načte z běhových proměnných (environment variables).

Výpis 3.1 Souborová struktura RTLSDR-Airband projektu

```
|-rtl_airband_airspy.example
|-Dockerfile
|-rtl_airband.example
|-rtl_airband_airspyhf.example
|-entrypoint.sh
```

Výše uvedená struktura ukazuje výčet všech zdrojových souborů pro úspěšné sestavení customizovaného kontejneru s RTLSDR-Airband.

- .example jsou šablony konfiguračních souborů. V projektu byly prototypovány na RTL-SDR, AirSpy Mini a AirSpy HF+
- Entrypoint.sh je skript, který se spustí při každém kontejneru. Zkopíruje šablony konfiguračního souboru a vloží do nich proměnné, které byly zadány při vytvoření kontejneru
- Dockerfile obsahuje skript pro sestavení kontejnerového artefaktu

Výpis 3.2 Konfigurační soubor RTLSDR-Airband airspyhf.example (vlastní zpracování)

```
fft_size = 512;
devices: (
  {
    type = "soapysdr";
    device_string = "driver=airspyhf,device_id=0";
    mode = "scan";
    channels:
    (
      {
        freqs = ( #FREQUENCY );
        bandwidth = ( 9000 );
        outputs: (
          {
            type = "udp_stream";
            dest_address = "127.0.0.1";
            dest_port = 8001;
            continuous = false;
          }
        );
      }
    )
  });
```

- FFT_SIZE je parametr určující počet bodů použitých při transformaci signálu z časové domény do frekvenční domény. Velikost FFT ovlivňuje rozlišení frekvenčního spektra, tedy jak detailně můžeme rozlišit jednotlivé frekvence v rámci signálu. Dle oficiální dokumentace je vhodné ji měnit v případě, že je demodulované audio zkreslené. U AirSpy HF+ bylo vyzkoušeno jak 512, tak 256 bez žádné změny v kvalitě
- TYPE definuje, zda se má použít pro připojení zařízení nativní implementace pro zařízení značky RTL-SDR, nebo ostatní zařízení přes adaptérovou knihovnu SoapySDR.
- DEVICE_STRING obsahuje filtry, které aplikace přepošle do SoapySDR knihovny pro vyhledání správného zařízení. Zde tedy říká: „Připoj první nalezené zařízení s ovladačem AirspyHF.“
- MODE
- CHANNELS
- FREQS – obsahuje pole frekvencí, které má aplikace neustále cyklovat a zjišťovat, zda je na nich přítomný squelch – tedy aktivní rádiová komunikace.
- BANDWIDTH – jedná se o volitelnou položku, lze s ní omezit nahrávaný rozsah
- OUTPUTS – zde jsou definované výstupy, kam má aplikace posílat demodulovaný zvuk. Bude posílat UDP audio stream na portu 8001. Continuous = false zajišťuje, že v případě neaktivního squelch stream přestane vysílat.

### 3.4.3 Aplikace pro zpracování audia

Druhou součástí je aplikace vlastního zpracování. Její role je poslouchat vysílající UDP audio stream z RTLSDR-Airband aplikace, zpracovat stream, vytvořit z něho čistě v paměti RAM .wav soubor a ten poslat do cloudového úložiště. Při úspěšném nahrání aplikace upozorní backendový server na aktivitu na letišti s identifikátorem nahraného souboru. Nahráním souborů přímo do úložiště, a ne přes backendový server se ušetří na datovém toku backendového serveru v případě asymetrického internetového připojení.

První prototyp aplikace byl vytvořen v Pythonu, po zabalení do kontejneru měla výsledná aplikace velikost přes 800 MB. Na pouhé zpracování UDP streamu se jedná o nepřípustnou velikost. Druhá verze aplikace byla napsána v Go. Jedná se o jazyk s výbornou podporou kompilace napříč architekturami, platformami. Je typově bezpečný a generuje malé binární spustitelné soubory. Celá aplikace se i včetně kontejneru vešla do 20 MB.

Plný zdrojový kód je k dispozici v přílohách. Zde je ukázána zjednodušená koncepce hlavního cyklu. Ten otevře naslouchání na UDP portu, který je stejný, jako má druhá RTLSDR-Airband aplikace. IP adresa je v tomto případě localhost, tedy stejného počítače. Následně je vytvořen nekonečný cyklus, který čeká na příchozí UDP pakety. V případě, že dojde k časovému limitu proměnné UDP_TIMEOUT_SEC bez žádných nově přijatých paketů, je smyčka resetována. Tím se řeší funkční požadavky architektury 3,7 a 10 v kapitole 3.3. Tímto je docíleno, že v případě zachycení nějakého audia z aplikace RTLSDR-Airband a následně tichého místa, které je delší než výše proměnná, se audio nahrávka uzavře, zpracuje a pošle ze zařízení na server.

Po prototypování se nastavila 1 vteřina, jelikož piloti mezi sebou odpovídají velice rychle. A v případě delšího časového limitu docházelo ke spojení více dialogů z různých letadel k sobě.

```
func main() {
  addr := net.UDPAddr{
    Port: UDP_PORT,
    IP:   net.ParseIP(UDP_ADDR),
  }
  conn, err := net.ListenUDP("udp", &addr)
  printError("Error opening UDP", err, true)
  defer conn.Close()

  buffer := make([]int, 0)
  tmp := make([]byte, SAMPLE_RATE) // Maximum UDP packet size
  voiceStartTime := time.Time{}
  log.Println("Started Listening for data")

  for {
    conn.SetReadDeadline(time.Now().Add(UDP_TIMEOUT_SEC * time.Second))
    n, _, err := conn.ReadFromUDP(tmp)

    // No error means receiving UDP data
```

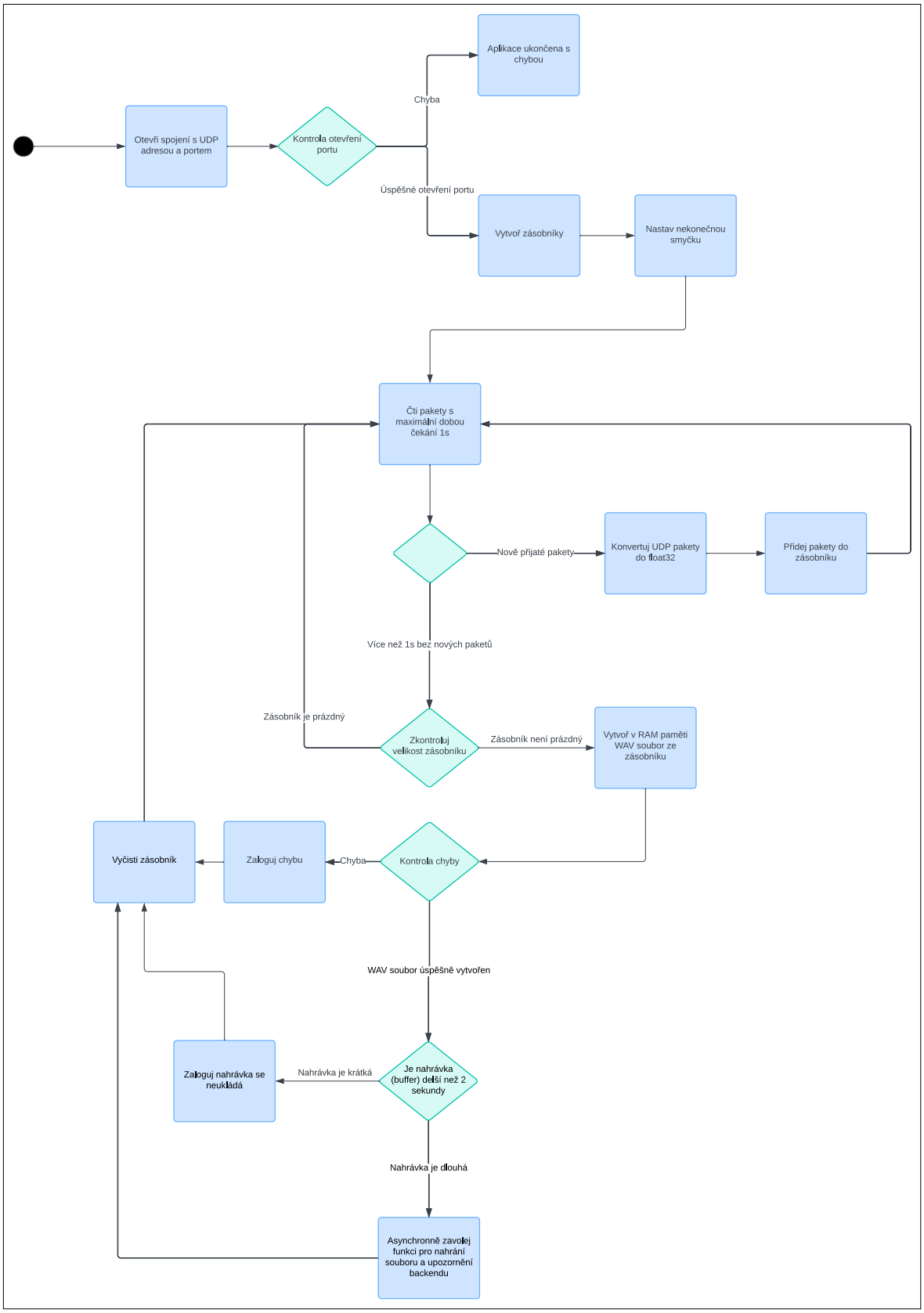
```
    if err == nil {
      // n/4 because 4 bytes = 32 bits = float32
      floats := make([]float32, n/4)
      reader := bytes.NewReader(tmp[:n])
      if err := binary.Read(reader, binary.LittleEndian, &floats); err != nil {
        printError("Error converting UDP data to float32", err, false)
        continue
      }
      val := *convertFS16(&floats)
      buffer = append(buffer, val...)
    }

    // Error means UDP timeout
    if err != nil && len(buffer) > 0 {

      log.Println("No more data, trying to save file")
      reader, err := createWAV(buffer)

      if err != nil {
        printError("Error creating WAV", err, false)
      } else if len(buffer) > (RECORD_MIN_SEC * SAMPLE_RATE) {
        go cloud.HandleRecord(voiceStartTime, reader)
      } else {
        log.Println("Record too short, not saving")
      }

      // To clear buffer and start time
      buffer = buffer[:0]
      voiceStartTime = time.Time{}
    }
  }
}
```

Jelikož knihovna na vytvoření WAV souboru vyžadovala rozhraní Go pro WriteSeeker, které pracuje se soubory, byla použita jednoduchá implementace stejného rozhraní čistě v RAM, aby nedocházelo k žádnému ukládání souborů na disk dle požadavků 10 pro klientský software v kapitole 3.3 [30].

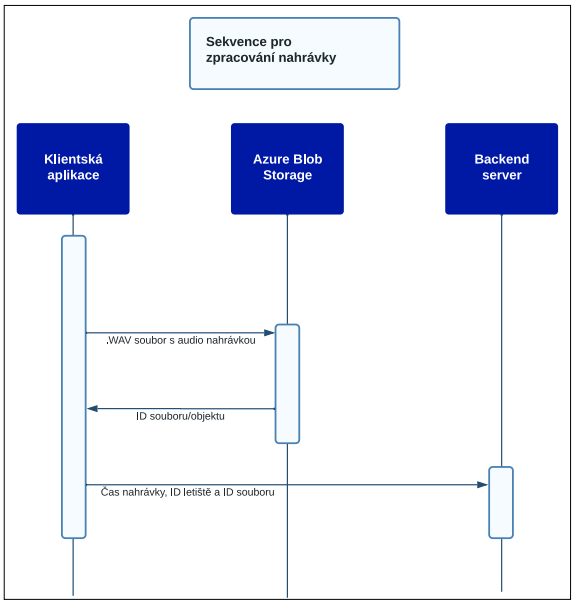
Kromě již výše popsané hlavní smyčky v main.go je ještě v aplikaci v separátních souborech implementována logika pro zpracování souboru čistě v paměti (writeseeker), balíček *cloud,* který jen implementuje kód pro nahrání souborů do cloudového úložiště a zaslání informace na backendový server. A balíček *retry*, který implementuje logiku pro opakování pokusů nahrání se vzrůstající prodlevou.

Výpis 3.3 Struktura klientské aplikace pro zpracování nahrávek

```
|-retry
 | |-retry.go
 |-Dockerfile
 |-writerseeker.go
 |-.env
 |-cloud
 | |-records.go
 |-.env.example
 |-main.go
```




Obrázek 10 UML Aktivity diagram zachycující logiku klientské aplikace




Obrázek 11 Sekvenční diagram klientské aplikace

Kompletní diagram nasazení, který zachycuje celou architekturu práce včetně klientského zařízení je k dispozici v příloze.

### 3.4.4 Úložiště pro nahrávky a první iterace serveru

Po zpracování oběma aplikacemi je potřeba audio uložit do bezpečného místa. Tím se vyznačuje systém, kde je zajištěna jednak trvalost uložení, ale také škálovatelnost, která umožňuje efektivní přístup k rostoucímu množství dat. Vzhledem k těmto požadavkům bylo pro uchování nahrávek zvoleno objektové úložiště, jež se těmito prvky vyznačuje.

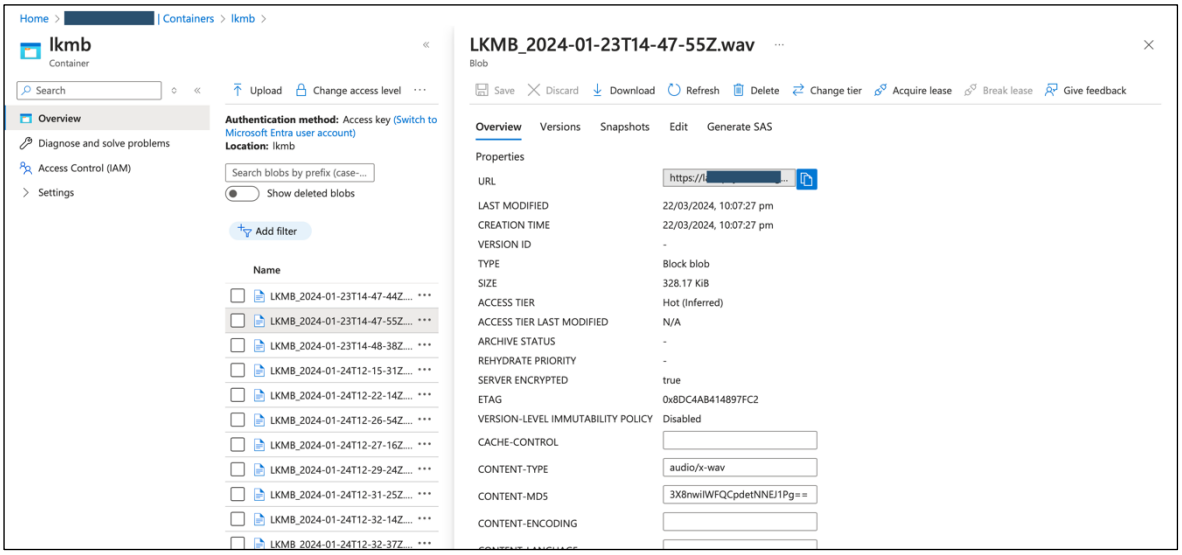
Objektové úložiště, odlišné od tradičního souborového systému, ukládá data v ploché struktuře bez adresářové hierarchie. Soubory mohou být jednoduše uloženy a replikovány napříč servery. Každý nahraný soubor má své jedinečné unikátní ID (GUID), se kterým se lze na soubor odkazovat. Také lze využít i jméno samotného souboru. To je totiž také neměnné. Proto samotné úložiště nelze využít jako databázi pro vyhledávání souborů. V průběhu času se mění byznysové požadavky, které by k souborům chtěly uchovávat další metadata pro jejich třídění a vyhledávání.

To vede k výběru konvence zvané Úložiště adresované podle obsahu (CAS[8]), je metoda ukládání informací, která umožňuje jejich vyhledávání na základě obsahu, nikoli názvu nebo umístění. Tento přístup znamená, že každý soubor (blob) je jednoznačně identifikován svým unikátním ID a na toto ID je odkazováno například z databáze, kde jsou uchovány veškeré dodatečné informace o nahrávce.

[8] Content-addressable storage - CAS

Protože nahrávky letištní komunikace jsou neměnné, je tohle vhodný přístup k jejich archivaci. Budou se totiž jen měnit či přidávat informace zjištěné z těchto nahrávek, ale samotné audio nahrávky zůstanou fixní.

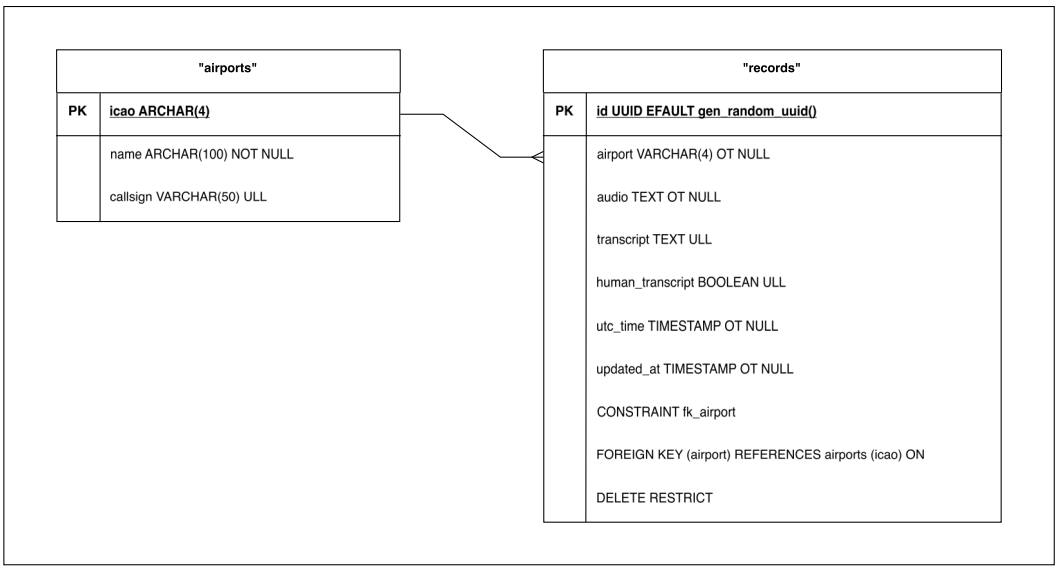
Pro objektové úložiště byl vybrán Azure Cloud Storage, jelikož nabízí 5 GB objektového úložiště zdarma a pro studentské projekty v rámci předplatného Azure for Students nabízí i další prostředky na jeden rok. V budoucnu je možné nahrávky zmigrovat kamkoliv jinam. Každé letiště má svůj vlastní kontejner pro základní organizaci dat.




Obrázek 12 Ukázka webového rozhraní objektového úložiště

## Databáze

Souběžně s úložištěm byla vytvořena první verze relační databáze. Fáze porozumění dat v tomto projektu přichází až v druhé iteraci, proto byl v tomto bodě vytvořen datový model, který má nezbytné minimum potřebné pro uložení nahrávky.




Obrázek 13 Prvotní verze datového modelu

Z konceptuálního schématu je patrné, že každá nahrávka obsahuje:

Interní unikátní ID – Oproti klasickému číselnému inkrementu byl zvolen unikátně generovaný GUID. Je to s ohledem na plánovaný technický koncept na začátku projektu MMSP. Plánuje se, že nahrávky budou štítkované uživateli a bude k nim různě přistupováno. GUID je bezpečnější, protože se může použít rovnou v URL adrese v budoucí aplikaci pro práci se záznamy.

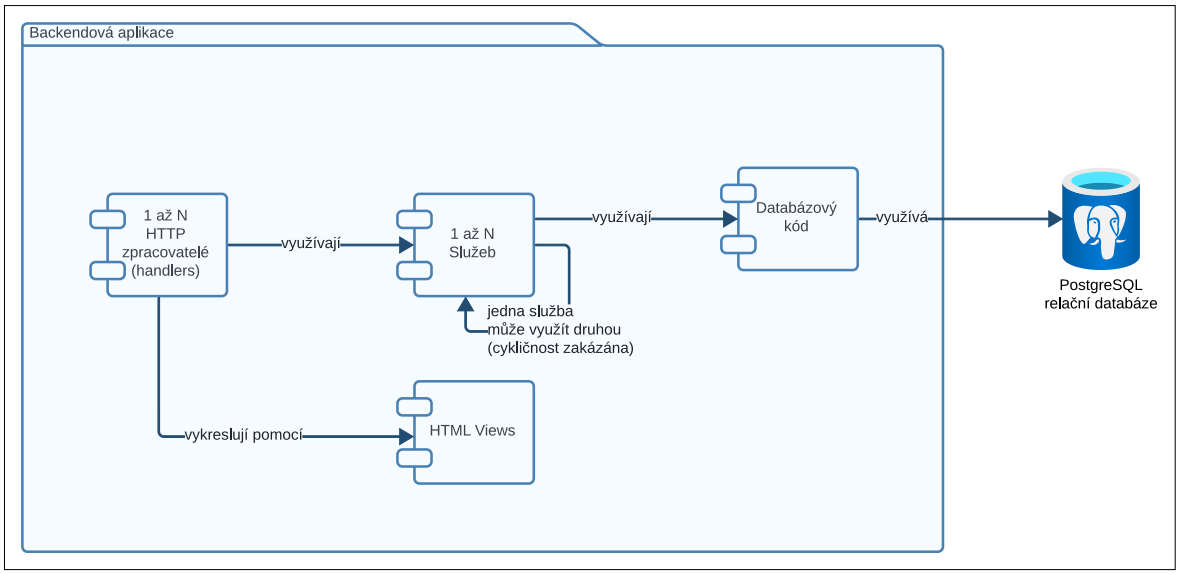
Letiště – Zde se jedná jen o čtyř písemné ID letiště. Organizace ICAO zajišťuje, že názvy letišť jsou vždy unikátní, a proto je možné využít název letiště jako unikátní ID, tedy například „LKPR“ nebo „LKMB“ – Letiště Mladá Boleslav.

Fyzická vrstva systému řízení báze dat je PostgreSQL. Je to díky jeho podpoře moderních datových typů, všeobecné použitelnosti a svobodné licenci. Pro první iteraci byla spuštěna instance PostgreSQL také na Azure serveru se službou Azure Database for PostgreSQL. Ale právě výběr PostgreSQL místo proprietární Azure Database umožňuje budoucí migraci. [31]

Backend pro klientské zařízení ze stejných důvodů jako v kapitole výše (viz. 3.4.3), tedy přenositelnosti, rychlosti, byl zvolen jazyk Go pro implementaci backendu. Tento jazyk má bohatou standardní knihovnu, která se zaměřuje právě na rychlou tvorbu mikroslužeb a backendů. Bez dodatečných závislostí je jednoduché vytvořit backendové API pro klientské zařízení.

S ohledem na celkovou koncepci projektu byl zvolen cibulový model architektury, který umožní lehkou škálovatelnost v dalších iteracích. V cibulovém modelu je aplikace rozdělena do vrstev, kde nižší vrstva neví o té vyšší. To zajišťuje volné propojení. V rámci aplikace jsou definovány obslužné moduly (handlers), které zpracovávají HTTP požadavky (vstup/výstup). Tyto obslužné moduly využívají služby (services), jež implementují byznys logiku, funkcionality a komunikaci s externími API. Služby mají možnost vzájemné komunikace. Následně mohou služby využívat databázovou vrstvu, která je odpovědná za manipulaci s daty – jejich ukládání dovnitř a ven.

Backendový server byl hostován na vlastní architektuře. Doména byla obalena proxy serverem s HTTPS certifikátem pro splnění požadavku v kapitole 3.3.




Obrázek 14 UML balíčkový diagram cibulové aplikační architektury backendu

## 3.5 Nasazení architektury na první letiště

Před prvním pokusem nahrát reálný letištní provoz bylo vhodné provést simulované testy funkčnosti co největší části řetězce. Samotný příjem AM ověřit nejde bez simulování AM signálu. Test se tedy zaměřil na zbytek. Klientská aplikace pro zpracování UDP streamu, cloudové úložiště a první iterace backendového serveru.

Bylo pouze potřeba simulovat stejný UDP datový tok, který bude produkovat RTLSDR-Airband. Na to posloužila aplikace FFmpeg, která byla puštěná na stejném počítači. FFmpeg je knihovna pro dekódování, enkódování a převod všemožných audio a video formátů.

Pokud chcete streamovat libovolný audio soubor přes UDP a konvertovat ho do tohoto formátu, můžete použít následující příkaz ffmpeg:

Následující ukázka kódu bere jako vstup .mp3 soubor a:

- ACODEC převádí ho do PCM audio kodeku s bitovou hloubkou float 32
- AR s vzorkovací frekvencí 16 kHz
- AC nastavuje počet kanálů na 1 (mono)
- F a jako souborový formát čistý audio proud float 32 little endian
- Výstup není do souboru, ale do UDP proudu 127.0.0.1:8001

Výpis 3.4 Ukázka testu klientské aplikace pomocí ffmpeg

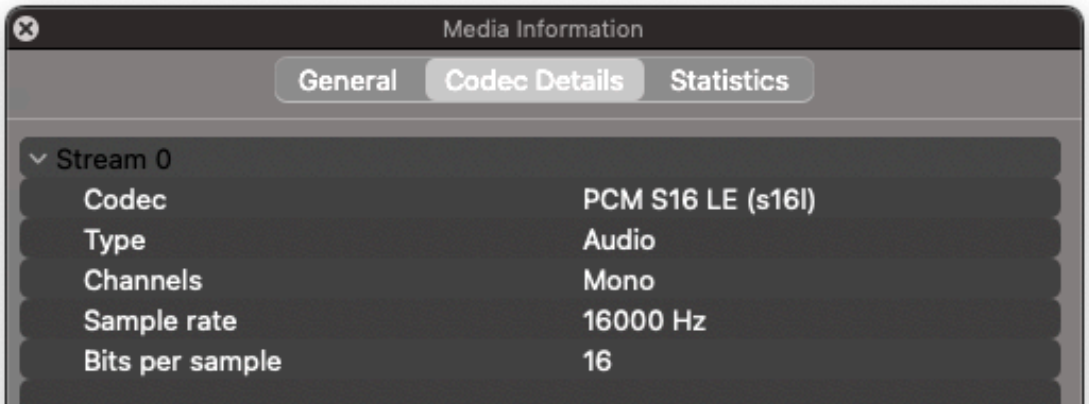
```
ffmpeg -re -i input.mp3 -acodec pcm_f32le -ar 16000 -ac 1 -f f32le udp://127.0.0.1:8001
```

První místo, kde byla architektura vyzkoušena s reálným letištním provozem, bylo Letiště Václava Havla v Praze Ruzyni (LKPR).

Obrázek 15 První nasazení prototypu v živém provozu (archiv autora)

Cílem testovací fáze bylo ověření následujících scénářů:

1. Aplikace RTLSDR-Airband korektně zachytí, demoduluje AM signál a audio začne posílat pomocí UDP proudu do vnitřní sítě zařízení
2. Aplikace pro zpracování audia zachytí UDP přenos
3. Aplikace pro zpracování audia korektně vytvoří .wav audio ve formátu 16 kHz, int 16 bitovou hloubkou a PCM kodekem
4. Soubor se úspěšně pošle do cloudového objektového úložiště
5. Aplikace korektně upozorní backend server na nový záznam, který si uloží do databáze




Obrázek 16 Kontrola nahraného audio souboru v přehrávači

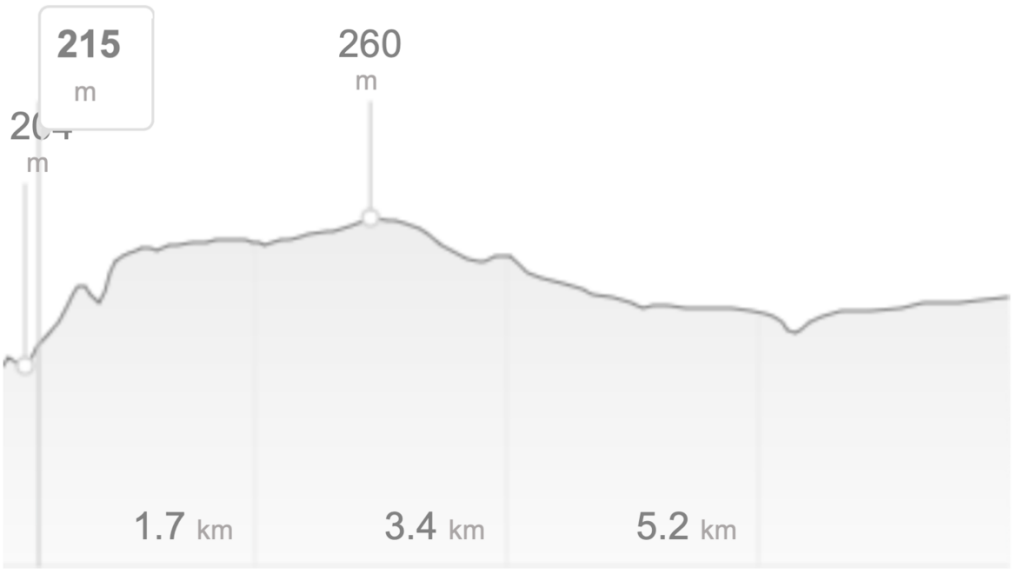
Všechny scénáře byly úspěšné. Nahrané audio mělo zpočátku vyšší šum, který byl odladěn v průběhu zavedení první iterace.

## 3.6 Shrnutí fáze konstrukce a zavedení první iterace

Po úspěšné demonstraci prototypu bylo vybráno finální místo pro sbírání dat. Jelikož je cílem práce sběr nahrávek v českém jazyce, bylo vybráno nejbližší letiště u autora práce.

Nasazení proběhlo uvnitř ATZ letové zóny letiště Mladá Boleslav LKMB.

Jak bylo zmíněno v kapitole o leteckém pásmu 3.1.1, příjem je nejlepší v přímé viditelnosti. Letiště by nemělo mít větší převýšení než místo s příjmovým zařízením. První lokalita v domě splňuje vzdálenost, nachází se totiž přímo uvnitř letištní zóny, avšak převýšení je větší než optimální. Nachází se totiž níže než letiště. Sestava však byla vyzkoušena i na tomto místě a během 2 týdnů zachytila dostatečné množství nahrávek pilotů ve zřetelné kvalitě.




Obrázek 17 Výškový profil antény (215 m.n.m.) a letiště LKMB (260 m.n.m.)

### Ověření správného zapojení a elektromagnetického šumu

Pro ověření, že SDR rádio nepřijímá nevyžádaný šum, případně, že je frekvence správně naladěná, byl pro nasazení v první iteraci také připraven Docker kontejner s aplikací SDR++, která díky architektuře server-klient dokáže na zařízení, ke kterému je připojena anténa a SDR, spustit server a poté na jakémkoliv jiném počítači spustit klientskou aplikaci s GUI, která umožňuje vizualizaci přijímaného signálu. Po otestování byla na zařízení přes Balena Dashboard aplikace vypnuta a místo ní nastartován kontejner s RTLSDR-Airband.


Obrázek 18 Ověření kvality zapojení přes server/klient SDR++

Definice Docker kontejneru (Dockerfile) je přiložena v elektronické příloze.

### Nároky na úložiště

Po dokončení první iterace již bylo možné vyčíslit průběžnou aktivitu a velikost sebraných nahrávek. Bylo očekáváné, že s příchodem letecké sezóny se náročnost zvýší. Stále se však

pohybuje v nízkých jednotkách gigibajtů měsíčně, a to v měsících letecké sezóny. Průměrná délka nahrávky je mezi 7 a 8 vteřinami.

| Měsíc | Velikost uložených záznamů během měsíce | Počet nahrávek |
|---|---|---|
| Únor | 352 MiB | 1 694 |
| Březen | 711 MiB | 3 671 |
| Duben[9] | 1,2 GiB | 5 880 |

### Úprava konfiguračních parametrů

Při poslouchání a štítkování nahrávek v druhé fázi byl zpětně upraven konfigurační parametr (viz. 3.4.3) pro časový limit čekání na další příchozí audio před ukončením nahrávky z 5 na 1 vteřinu. Jelikož se s delším limitem spojovaly nahrávky od více pilotů dohromady.

Je tedy potřeba navrhnout a zprovoznit koncové zařízení internetu věcí (tzv. Edge IoT), které bude umístěno na jednotlivých letištích. Zařízení musí neustále zachytávat rádiové frekvence letiště a v případě probíhající komunikace (pilot nebo operátor řízení provozu na daném letišti) vytvořit nahrávku a uložit ji na vzdálený sběrný server.

[9] Duben byl vyhodnocen na základě trendu k 25.4. 2024

# 4 Analýza a příprava dat

Dle metodiky CRISP-DM je vhodné provést analýzu obsahu a vyvinout vhodný datový model. Tento proces, zahrnutý ve fázi *Porozumění datům* metodiky, zahrnuje rozpoznání a klasifikaci informací v nahrávkách, jako jsou letištní dráhy, záměry pilotů, barometrické tlaky (QNH) a volací znaky. Pochopení významu dat umožní data transformovat na získané informace. [11 s. 20–22]

Výstupem fáze je vytvoření datového modelu letištní nahrávky a rozšíření serverové aplikace na webové rozhraní, které umožní ruční přepis a klasifikování nahrávek.

## 4.1 Struktura letecké komunikace

Výhoda letecké komunikace při strojovém zpracování dat je, že se jedná o strukturovanou řeč, která má jasně danou frazeologii. Pravidla české komunikace jsou definována ve předpisu letecké frazeologie Řízení letového provozu České republiky. [4]

V této části byla využita dle MMSP role analytiků. Ing. David Muschalik, profesionální pilot a inženýr pro oblast leteckého provozu a Ing. Jan Stádník pro oblast řízení letového provozu. Jejich úkolem bylo definovat seznam, jaké informace můžeme získat z běžné letecké komunikace na menších letištích.

### 4.1.1 Rozdíly mezi ATZ a CTR

První zásadní informací bylo rozlišit mezi pojmy ATZ a CTR[10].

CTR je tzv. Řízený okrsek. U něho si můžeme laicky představit velká letiště, kde létají komerční lety, například Letiště Václava Havla v Praze Ruzyni. Letadla musí mít pro vstup letové povolení, mít schválený let. Povolení pro vstup do CTR je plně v kompetenci řídícího. Například v Brně do CTR běžně lítají i kluzáky a balóny. Pilot bez příslušného povolení do prostoru nesmí vstoupit. Komunikace je zde tímto o dost více regulovaná a je preferována angličtina. [15]

Na druhou stranu ATZ[11] je tzv. Provozní letištní zóna. Piloti se při vstupu do ATZ musí oznámit na místním kmitočtu a postupně udávat své záměry. Pokud se nikdo nehlásí zpět, vysílají tzv. „naslepo“, protože v ATZ nemusí být přítomná obsluha služby. V tomto případě většina ohlásí vstup a výstup z ATZ. V případě, že mezitím do ATZ přiletí další provoz, ohlásí svoji současnou polohu. Komunikaci nikdo nekontroluje a většinou ani nenahrává. Provoz bývá o dost nižší. Stane se, že si na frekvenci piloti povídají mezi sebou, i když dle předpisu

[10] Control Zone
[11] Air Traffic Zone

by se měly používat pouze standardizované fráze nebo otevřená řeč. Z osobní zkušenosti Davida Muschalika však stále dodržují a oznamují klíčové momenty pro zachování bezpečnosti provozu.

### 4.1.2 Volací značka letiště

Začátkem každého hlášení by měla být volací znaky letiště. Většinou se jedná o název letiště, případně jeho zkráceninu. A následně název služby. U neřízených se většinou jedná o službu RADIO nebo méně INFO. Tedy „Boleslav RADIO“, „Liberec RADIO“ a podobně. Řízené se dělí na APPROACH, RADAR, TOWER, GROUND, DELIVERY atd.

Jedná se o kontrolu, že je zbylá část zprávy určená pro správný okruh. Pro případ chybného naladění kmitočtu pilota nebo v minulosti kdy ke sdílení frekvencí napříč letišti, jak bylo zmíněno v Kapitole 3.1.3. Zároveň má volací značka i funkci adresní. Pilot vysílá všesměrově a všichni na kmitočtu zprávu slyší. Volací značkou se cílí na příjemce, aby zpozornil.

### 4.1.3 Volací znak letadla

Po volací značce letiště se pilot identifikuje volacím znakem svého letadla. Ten vychází z hláskování volacího znaku ve formě NATO abecedy. Tu definuje již zmíněné ICAO (Mezinárodní organizace pro civilní letectví). Jedná se o mezinárodně uznávaný standard pro jasné a nepochybné hláskování slov, zejména ve ztíženém prostředí, kdy v komunikaci může být přítomna vyšší hladina šumu (viz. AM modulace Kapitola 3.1.2).

Tato abeceda přiřazuje každému písmenu anglické abecedy specifické slovo, například „A“ jako „Alpha“, „B“ jako „Bravo“, „C“ jako „Charlie“ atd., což zabraňuje záměnám písmen, která mohou znít podobně. Termíny jako „Alpha“, „Oscar“ jsou vybrány záměrně kvůli jejich velké fonetické rozdílnosti. Tím i v případě, že je jen slyšitelná půlka slova, je stále možné pro lidský mozek domyslet zbytek „olf“- Golf, „trot“ – „Foxtrot“, … Stejný výsledek snížení chybovosti to přinese i do modelů přepisu řeči na text.

Ve volacích znacích se mohou vyskytnou i číslice. Ty jsou vždy řečené každou číslicí zvlášť.

Česká letadla mají na začátku vždy označení OK následované zbytkem. Například tedy *„OK-MMC“* bude *„Oscar Kilo Mike Mike Charlie“*.

Ing. Jan Stádník podotýká, že je možné i zkrácení volací značky. Oficiálně tímto odpovídá řídící věž pilotovi v příkazových/potvrzovacích instrukcích. Avšak v případě ATZ se může stát, že pilot vysloví zkrácenou volací značku. Zkrácený formát je složen z prvního písmene a posledních dvou znaků. V předchozím příkladu by se jednalo o OK-MMC jako *„Oscar Mike Charlie“*. [4 s. 31]

David Muschalik poté uvedl z praxe i neoficiální zkrácenou formu v případě 3 stejných znaků za sebou. *„OK-DDD“* tedy může znít *„Oscar Kilo Triple Delta“*.

Okrajovým případem může být použití slov výrobce letecké techniky. Jedná se zejména o případy přeletů armádní vojenské techniky, která se ozývá jen výrobcem a číslem. Například

*„Robinson čtyři“*. Ale v ATZ tak někdy označí i pilot své letadlo, které není v leteckém rejstříku. (Jedná se o ultralehký letoun). Například *„bristell osm sedm“*.

Posledním speciálním případem mohou být ještě vrtulníky letecké záchranné složky, které se oslovují slovem „Kryštof“ a číslem. Každý kraj má většinou jedno číslo. [32]

### 4.1.4 Letištní dráha

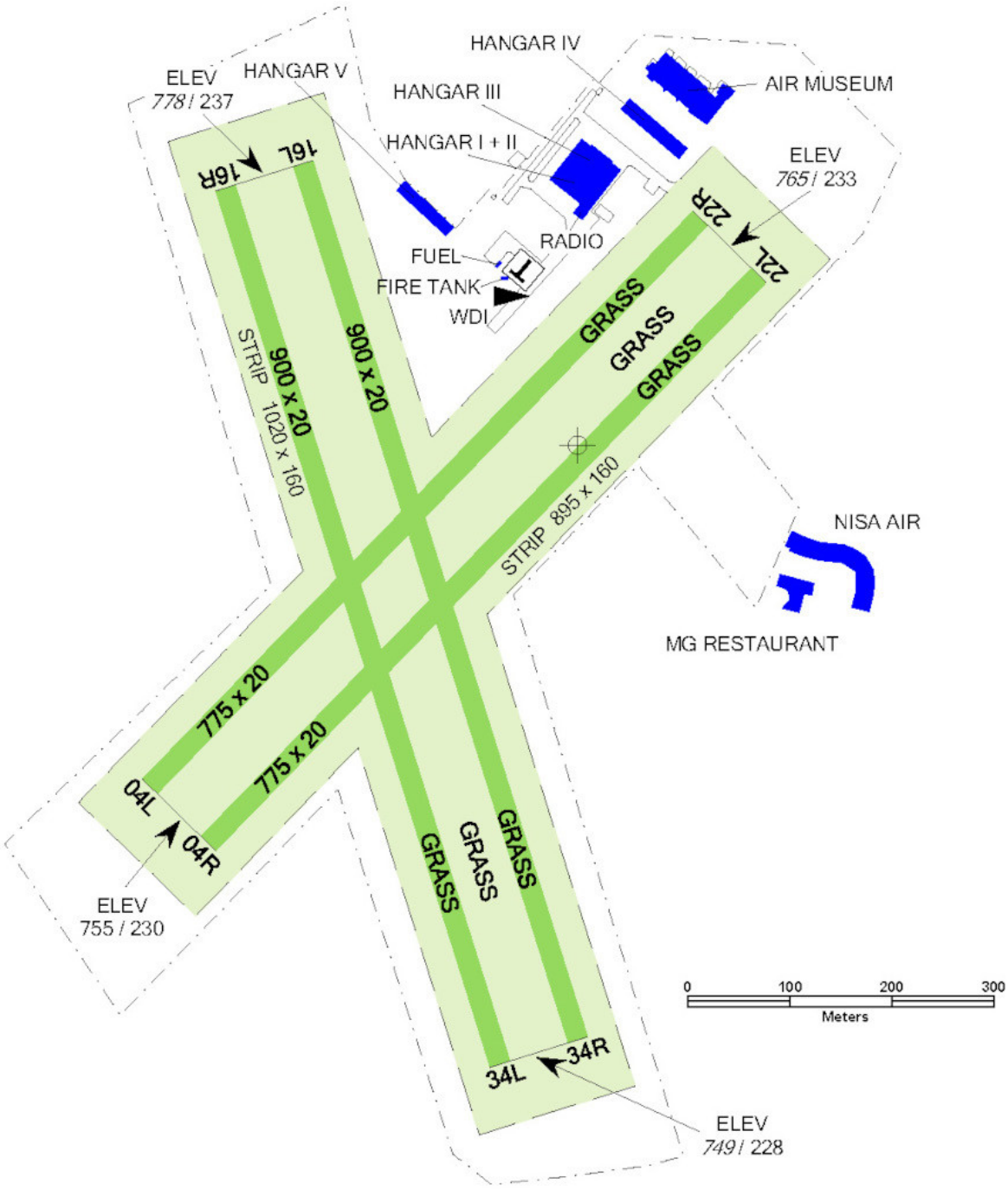
V komunikaci se lze setkat se samotnou vzletovou a přistávací dráhu (anglicky runway) a letištním okruhem (traffic pattern).

Vzletová a přistávací dráha je specificky upravený povrch na letišti, určený pro vzlety a přistání letadel. Letištní okruh pak představuje definovaný vzor letu, který piloti sledují při vzletu, přiblížení a přistání na letišti. Tento umožňuje systematické a bezpečné zařazování letadel do sekvenčního pořadí pro přistání nebo vzlet.

V komunikaci se tedy najednou používá informace jak o letištní dráze, tak v jakém bodě okruhu této dráhy se pilot právě nachází.

Seznam letištních drah je vždy publikovaný v příručce letového provozu pro každé letiště [2]. Dle Ing. Davida Muschalika si pilot vždy před cestou zjistí dostupné letištní dráhy. Dráha se vyznačuje číslem, které je zkráceným azimutem její orientace od severu. Tedy dráha 22 – frazeologicky „dva dva“ je orientována přibližně 220 stupňů od severu. To pomáhá pilotům ihned se orientovat na letištní dráze a okruhu.

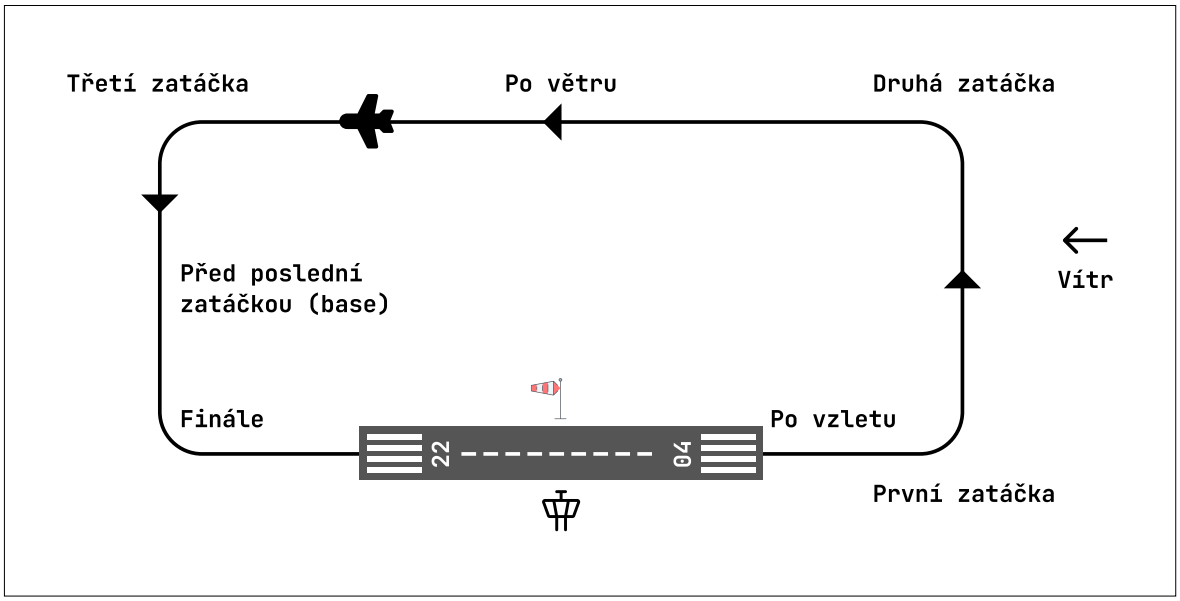
Typicky se setkáme se stejnou dráhou, která je označená dvěma opačnými čísly, neboť dráha může být použitá pro vzlet a přistání z opačných směrů. Letiště s dráhou 22 bude mít pravděpodobně i dráhu 04. (22 po odečtení 18 jako 180 stupňů).




Obrázek 19 Ukázka letištních drah na LKMB 04/22 a 16/34 (zdroj: VFR příručka rlp.cz)

### 4.1.5 Letištní okruh

Jak bylo zmíněno v předchozí kapitole 4.1.4 , letištní okruh je pak obloukovitá dráha, která slouží k úspěšnému zařazení se do fronty a přípravy na přistání nebo vzlet. V případě, že již pilot vstoupil do ATZ, oznamuje, na jaký okruh se chce připojit.




Obrázek 20 Letištní okruh (levý) v ATZ (tvorba autora)

Pro ATZ byly zahrnuty následující části okruhu:

- Po vzletu (departure)
- První okruhová zatáčka (first turn)
- Druhá okruhová zatáčka (second turn)
- Po větru (downwind)
- Třetí okruhová zatáčka (third turn)
- Před poslední zatáčkou (base leg)
- Finále (final)

Při příletu do ATZ nebo po vzletu před první zatáčkou se pilot dle seznamu preferovaných nebo dokonce přikázaných okruhů ve VFR příručce rozhodne, zda poletí okruh po směru nebo proti směru hodinových ručiček. To se oficiálně nazývá levý a pravý okruh (podle toho, kam by pilot zatočil v první okruhové zatáčce).

Příklady:

- *„Vstupuji do vaší ATZ a zařazuji se do první zatáčky pravého dráhy dva dva.“*
- *„Po vzletu a opouštím vaši ATZ ze druhé zatáčky levého dráhy dva dva levá.“*
- *„Finále dva dva plné přistání.“*

### 4.1.6 Výška a QNH

V případě vstupu do ATZ nebo při změně nadmořské výšky (altitude) pilot udává do komunikace i svoji výšku. Nadmořská výška je udávána vždy ve stopách.

Avšak tlak vzduchu na povrchu země není vždy stejný, a tak je nutné výškoměr nastavit na aktuální tlak na daném místě či alespoň na společnou základnu s ostatními letadly. Proto

mají výškoměry možnost nastavení tlaku – QNH[12]. Díky tomu výškoměr ukazuje spolehlivější hodnotu nadmořské výšky nad úrovní moře. Absolutně přesnou výšku neukazuje, jelikož závisí i na teplotě přístroje, jeho interní chybě, polohové odchylce.

Technicky se jedná o atmosférický tlak přepočtený na úroveň moře podle standardní atmosféry a je vyjádřen v hektopascalech (hPa). [2 s. 18] Aktuální změřené QNH může pilot obdržet při komunikaci s letištní věží. V případě, že na rádiu nikdo není, použije pilot QNH daného letiště, které je dostupné v příručce. [2]

Může dojít k situaci, kdy dva piloti mají výškoměr nastavený s různým QNH. I když jsou tak ve stejné skutečné výšce, jejich výškoměry ukazují rozdílnou výšku a může dojít ke kolizi. To se ale samozřejmě neděje a piloti mimo ATZ létají na oblastním QNH pro celou zemi, které je dostupné na informační službě Praha INFO. Uvnitř ATZ pak na již zmíněném QNH pro dané letiště.

Proto by piloti ihned za výškou měli udat i své aktuálně nastavené QNH. To umožní dispečerovi i dalším pilotům v oblasti lepší koordinaci.

Nastavení výškoměru poté pilot udává za výškou vyslovováním každé číslice odděleně. [4 s. 24]. I při české komunikace někdy pilot použije anglicismus *„altitude“* místo *„výška“*.

Příklady:

- *„Vstupuji do vaší ATZ výška dva tisíce feetů a QNH jedna nula jedna pět.“*
- *„Opouštím vaši ATZ altitude tisíc pět set QNH jedna nula jedna nula.“*

### 4.1.7 Záměr pilota

Celým důvodem vysílání pilota je nějaký záměr, který se pilot chystá provést. Po jeho identifikaci tedy oznámí na kmitočtu svůj nejbližší záměr, který se chystá provést.

Analytik Ing. David Muschalik identifikoval prvotní sadu záměrů. Po poslechu

- Vstup do ATZ
- Opuštění ATZ
- Průlet nad ATZ
- Průlet nad letištěm
- Přistání
- Vzlet

Během poslechu a ručního štítkování 600 nahrávek byly ještě přidány dva další záměry:

- Touch And Go (přistání a okamžitý vzlet zpět)

[12] QNH není akronym. Atmospheric Pressure (Q) at Nautical Height (NH)

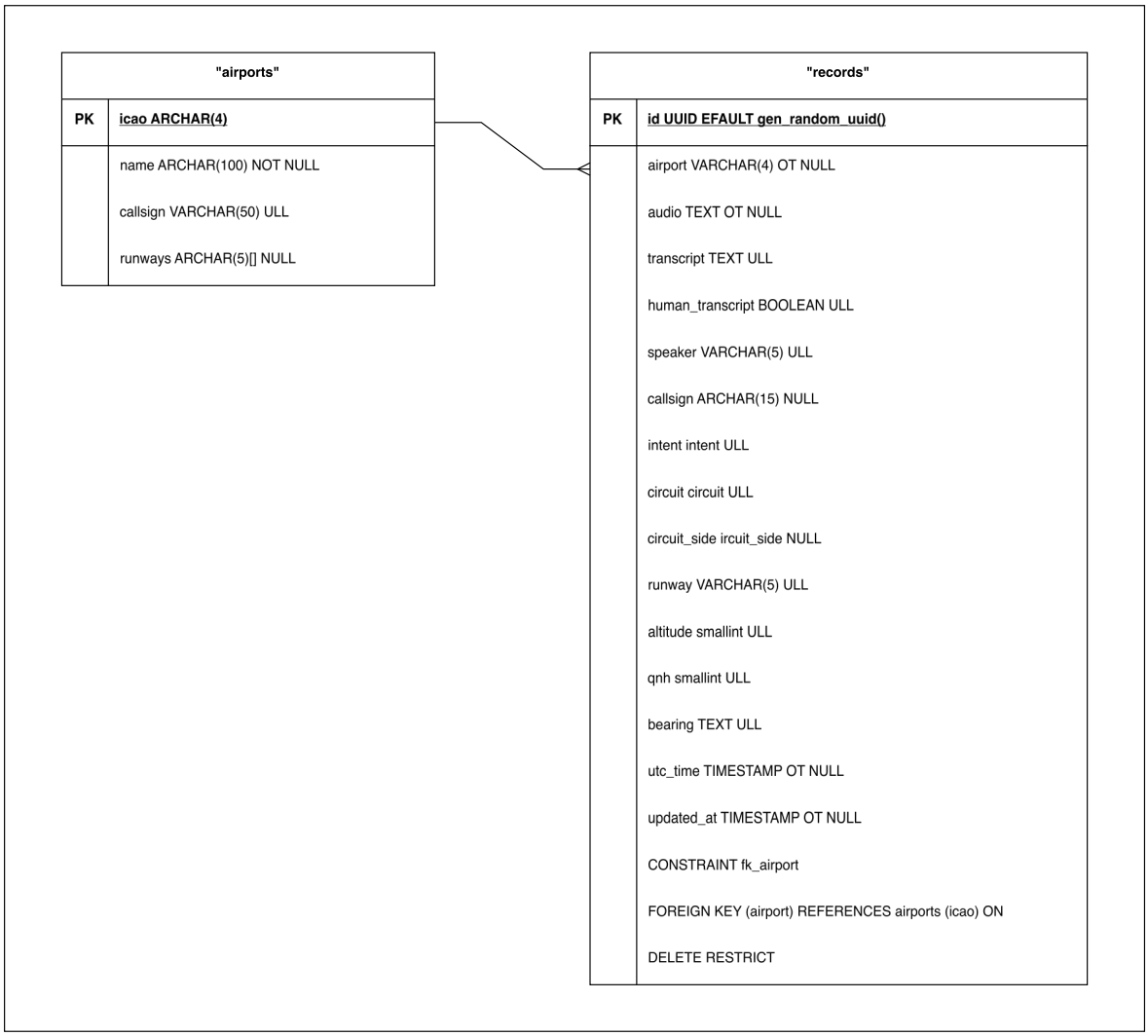
## 4.2 Návrh datového modelu ATZ komunikace

Z analýzy předchozí kapitoly byl rozšířen relační model databáze nahrávek. Relační model byl přímo napsán v SQL kódu pro systém řízení báze dat PostgreSQL.

V modelu jsou přítomny 2 entity. Letiště obsahuje svůj čtyřmístný kód ICAO. Protože je kód vždy unikátní, byl zároveň použitý i jako primární klíč letiště. Dále obsahuje svůj volací znak, jako například „Boleslav RADIO“ a pole se seznamem dostupných letištních drah. Tedy například [“22L”, “22R”, “04”, “34”].

Tabulka záznamů (“records”) poté obsahuje:

- AIRPORT – Cizí klíč letiště (ICAO)
- AUDIO – Název souboru v objektovém úložišti (např. „LKMB_2024-04-29T15-56-54Z.wav)
- TRANSCRIPT – Přepis nahrávky
- HUMAN_TRANSCRIPT – Ano/Ne zda se jedná o ruční přepis
- SPEAKER – Zda mluvil pilot nebo řídící (pilot/atc)
- CALLSIGN – Volací značka (např. „OKMMC“)
- INTENT – Záměr pilota. Jedná se o enum. Výčet hodnot
- CIRCUIT – Poloha na letištním okruhu. Výčet hodnot
- CIRCUIT_SIDE – Strana letištního okruhu (left/right)
- RUNWAY – Letištní dráha (22L, 14, 04…)
- ALTITUDE – Výška ve stopách
- QNH – Aktuální nastavení QNH baroměru pilota
- BEARING – Směr (sever, východ, severovýchod…)
- UTC_TIME a UPDATED_AT – Časové značky, kdy byla nahrávka pořízena (její počátek) a kdy byla naposledy změněna




Obrázek 21 Finální datový model nahrávek

## 4.3 Návrh a implementace přepisu do aplikace

### 4.3.1 Funkční a UX/UI požadavky

Po vytvoření datového modelu bylo součástí iterace i přidání webového rozhraní do backendové části architektury. Funkcionalita měla maximálně podpořit co nejrychlejší ruční štítkování a přepis záznamů. Autor práce využil své UX zaměření v roli analytika MMSP. Funkční a uživatelské požadavky byly definované:

- Možnost zobrazení přehledu všech nahrávek a úpravy jakékoliv nahrávky
- Dedikovaná sekce pro uživatele, kteří budou provádět přepis. Ta automaticky najde nejstarší nepřepsanou nahrávku k přepisu. Při odeslání ručního přepisu se ihned načte další nepřepsaná nahrávka v pořadí.
- Zcela customizovaný JavaScriptový audio přehrávač pro podporu napříč prohlížeči.
    - Možnost přehrání nahrávky se zpomalenou rychlostí
- Našeptávač pro nejvíce pravděpodobná slova

    - NATO abeceda
    - Dráhy, které jsou relevantní k dané nahrávce (dle letiště); „22“ bude po stisknutí TABulátoru převedeno na „dva dva“
    - Části letového okruhu
    - Nápověda se zobrazí vedle kurzoru textu
    - Při potvrzení TABulátorem se nápověda potvrdí
- Možnost smazání nahrávky v případě, že je neslyšitelná
- Základní předvyplňovač (autofill) štítků na bázi shody slov v poli přepisu
    - Tedy NATO abeceda a čísla vyskytující se hned po NATO abecedě budou rovnou vyplněna v poli pro volací značku
    - Slovní čísla („jedna“, „nula“, „dva tisíce“…) budou převedena na čísla v případě, že se před nimi nachází slovo „výška“ nebo „qnh“, a budou automaticky vyplněna


Obrázek 22 UX drátový návrh (Wireframe) pro přepis nahrávky

Obrázek výše zahrnuje návrh uživatelského rozhraní pro přepis nahrávky. Uživatel může nahrávku pustit pomocí customizovaného audio přehrávače. V případě, že je rychlost mluvení pilota rychlejší, může uživatel zpomalit přehrávání na 0,75x rychlosti. Nahrávka běží v nekonečné smyčce. Seznam drah je načítán podle daného letiště nahrávky. Na obrázku je právě zachycen našeptávač, který uživateli radí slovo „golf“ po zadání znaků „go“.


Obrázek 23 UX drátový model seznamu nahrávek

### 4.3.2 Technické řešení frontendu

Většina interaktivity uživatele ve webové aplikaci (frontendu) se pohybuje nad operacemi CRUD. Tedy: vytvoř, čti, aktualizuj a smaž. [7 s. 308]. Proto bylo přistoupeno k serverovému vykreslování aplikace. Namísto klientských knihoven, jako React, bylo použité jednoduché serverové vykreslování statického HTML na každý požadavek uživatele. Oficiální vývojářský blog Googlu tuto metodu definuje jako „Renderování na straně serveru“, tzv. SSR[13]. Výhodou je o dost jednodušší řetězec úkonů, a i rychlejší zdánlivý výkon aplikace. Vzhledem k tomu, že se pracuje s živými daty, je vždy vyžadováno připojení k internetu, a tudíž odpadají výhody, které bývají spojené s webovými aplikacemi orientovanými více na klientskou stranu. [33]

Typické zpracování požadavku v Reactu:

1. Uživatel provede akci
2. Vyšle se požadavek na server
3. Operace/Databáze

[13] Server-Side Rendering

4. Server vygeneruje a vrátí JSON
5. React zpracuje JSON do objektu
6. React porovná aktuální interní stav aplikace se zpracovaným objektem
7. React zanese změny do virtuálního HTML DOMu
8. React vygeneruje a nahradí HTML
9. Prohlížeč zobrazí výsledné HTML

Zpracování pomocí renderování na straně serveru:

1. Uživatel provede akci
2. Vyšle se požadavek na server
3. Operace/Databáze
4. Server vygeneruje a vrátí HTML
5. Prohlížeč zobrazí/vymění HTML.

Serverové vykreslení probíhá přímo na stejném backendu z minulé iterace, který byl nyní obohacen o tuto funkcionalitu. Jedná se o backend napsaný v Go jazyce. Pro generování HTML kódu byla použitá knihovna Templ. Jedná se o parser, který umožňuje psát webové komponenty v jazyce Go. Vývojář používá přímé HTML s několika základními Go konstrukty (podmínky, iterace). A Templ knihovna poté vygeneruje typově bezpečné Go funkce. Tím se zajištuje zvýšená stabilita softwaru, protože je odhaleno více statických chyb. Například v momentě, kdy je v Go zavolána funkce pro generování šablony, je zaručeno, že má šablona všechny proměnné, které potřebuje k vykreslení. [34]

Následující výpis kódu zobrazuje syntaxi knihovny Templ pro vygenerování nastylované customizované HTML komponenty pro zadání textu. Po spuštění příkazu *templ generate* v příkazové řádce se ve stejné složce, kde je umístěn kód, vygeneruje soubor s Go funkcí se stejným názvem, kterým je poté možné zavolat v kódu při generování HTML.

Výpis 4.1 Ukázka použití Templ komponenty

```
templ TextArea(label string, name string, defaultVal string, attrs templ.Attributes) {
  <label class="flex flex-col gap-1">
    <span class="text-gray-400">{ label }</span>
    <textarea type="text" name={ name } { attrs... }>{ defaultVal }</textarea>
  </label>
}
```

Webová aplikace ale bude vyžadovat nějakou míru lokální klientské interaktivity, která bude probíhat přímo na straně prohlížeče. Jak bylo definováno v kapitole požadavků 4.3.1. Interaktivita je dosažena dvěma způsoby, kterými je statické HTML obohaceno:

- HTMX
- Nativní Webové komponenty pomocí knihovny Lit

### HTMX

HTMX je minimalistická JavaScriptová knihovna, jejímž cílem je podpořit hypermédia jako zdroj stavu webové aplikace. Což znamená, že celý stav aplikace je tvořen HTML. Server s prohlížečem si nevyměňují JSON a podobná data, která musí poté prohlížeč knihovnami dále zpracovat do HTML podoby. HTMX pouze přidává pár užitečných atributů do HTML tagů, které doplňují nativní HTML.

Například každý element může poslat jakýkoliv GET/POST/PUT/DELETE požadavek. A jeho odpověď ve formátu HTML poté vložit do jakéhokoliv elementu. Nebo formuláře díky HTMX umí poslat i PUT a DELETE požadavek kromě výchozích GET/POST. [35]

Těmito metodami HTMX umožňuje se mnohem blíže přiblížit uživatelské zkušenosti podobné z velké webové aplikace psané frontendovým frameworkem jako React, Svelte, SolidJS, Vue. Ale se zachováním jednoduchosti a principů „hypermedia jako zdroj stavu webové aplikace“. [36]

Následující kód ukazuje použití HTMX pro provedení GET požadavku na klasickém HTML tlačítku. Odpovědí je HTML dialog, který je pomocí HTMX připojen na konec těla HTML (hx-swap a hx-target).

Výpis 4.2 Ukázka použití HTMX

```
<button hx-get="/transcribe/form/92034c32" hx-swap="beforeend" hx-target="body"
type="button">Opravit nahrávku</button>
```

### Nativní webové komponenty s Lit

Pro případy, kdy HTMX nestačí (tedy načtení HTML ze serveru a jeho výměna uvnitř stránky nebo zaslání jiného než GET a POST požadavku), budou vytvořeny vlastní nativní webové komponenty.

Tento přístup bude potřeba pro funkční požadavky výše v kapitole 4.3.1. A to zejména vlastní audio přehrávač, našeptávač pro přepis letecké komunikace a automatický vyplňovač štítků u leteckých záznamů.

Po úspěchu frontendových Javascriptových frameworků jako React, Vue, Svelte byl později vytvořen webovým konsorciem standard, jak definovat a používat vlastní webové komponenty pomocí nativního Javascriptu. Oficiální dokumentace Mozilla foundation zmiňuje, že tento standard umožňuje definovat v Javascriptu veškerou funkcionalitu, styly, vzhled a značku nové komponenty a poté stačí jen přiložit Javascriptový soubor do stránky a komponentu využívat jako jakýkoliv jiný element dle její definované značky. [37]

Pro zlepšení práce při vývoji byla použitá minimalistická knihovna *Lit*, která pouze tuto nativní funkcionalitu obaluje do vyšší abstrakce, která eliminuje určité repetitivní kroky. [38]

Následující výpis ukazuje zjednodušenou definici Nativní webové komponenty <fl-audio>, která implementuje vlastní audio přehrávač s možností zpomalení audia a stejného vzhledu napříč prohlížeči. Celá komponenta je k dispozici v elektronické příloze práce. Ve výpisu je vidět definice atributů, kterými může být komponenta konfigurována při jejím použití v HTML, stavové proměnné, názvy metod, které komponenta implementuje, a HTML kostru.

Výpis 4.3 Zkrácená verze nativní komponenty audio přehrávače

```
import { LitElement, html } from "lit";
import { customElement, property, state } from "lit/decorators.js";

@customElement("fl-player")
export class AudioPlayer extends LitElement {
  @property({ type: String }) src?: string;
  @property({ type: Boolean }) loop?: boolean;
  @property({ type: Boolean }) slider?: boolean;
  @property({ type: Boolean }) speed?: boolean;

  @state() protected isPaused = true;
  @state() protected isSlowed = false;
  @state() protected volume = 100;
  @state() protected seek = 0;
  // Safari has a bug. It borks audio.duration when in loop mode
  @state() protected duration = 10;
  // Request Animation Frame holder
  @state() protected raf: number | null = null;

  @state() private audio: HTMLAudioElement;

  constructor() {
    super();
    this.audio = new Audio();
    this.audio.preload = "none";
  }

  firstUpdated() {
    this.preparePlayer();
  }

  disconnectedCallback(): void {
    super.disconnectedCallback();
    this.audio.pause();
    cancelAnimationFrame(this.raf!);
  }
  render() {
```

```
  let playIcon;
  if (this.isPaused) {
    playIcon = html`<i class="material-symbols-sharp">play_arrow</i>`;
  } else {
    playIcon = html`<i class="material-symbols-sharp">pause</i>`;
  }

  let speedIcon;
  if (this.isSlowed) {
    speedIcon = html`<i class="material-symbols-sharp text-yellow-300"
      >speed_0_75</i
    >`;
  } else {
    speedIcon = html`<i class="material-symbols-sharp">speed_0_75</i>`;
  }

  let slider = html`
    <input
      @change=${this.handleSeekFinish}
      @input=${this.handleSeeking}
      type="range"
      id="seek-slider"
      .max="${this.duration}"
      .value="${this.seek}"
      step="0.01"
      class="h-2 bg-gray-200 appearance-none cursor-pointer dark:bg-gray-600 relative"
    />
  `;
  return html`
    <div class="flex flex-row items-center gap-1">
      <button @click=${this.handlePlayPause} class="inline-flex">
        ${playIcon}
      </button>
      <span id="current-time" class="time"
        >${this.formatTime(this.seek)}</span
      >
      ${this.slider ? slider : ""}
      <span id="duration" class="time">0:00</span>
      <button @click=${this.handleSlowdown} class="inline-flex">
        ${this.speed ? speedIcon : ""}
      </button>
    </div>
  `;
}
```

```
  protected createRenderRoot(): HTMLElement | DocumentFragment {
    return this;
  }

  private handlePlayPause(e: Event) {  }

  private handleSeekFinish({ target }: { target: HTMLInputElement }) {}

  private handleSeeking({ target }: { target: HTMLInputElement }) {  }

  private handleSlowdown({ target }: { target: HTMLInputElement }) {  }

  private handleMute(e: Event) {
    this.audio.muted = !this.audio.muted;
  }
  private whilePlaying = () => {  };

  private formatTime(secs: number) {  }

  private showRangeProgress(rangeInput: HTMLInputElement) {  }

  preparePlayer() {
    const root = this.renderRoot;
    const audio = this.audio;
    audio.loop = this.loop || false;
    const durationContainer = root.querySelector("#duration")!;
    audio.src = this.src || "";
    const displayBufferedAmount = () => {    };
    const displayAndSet = () => {    };
    audio.addEventListener("progress", displayBufferedAmount);
  }
}
```

Výsledná komponenta se nyní při vložení Javascriptu na stránku může použít bez další potřebné konfigurace a nastavování na jakékoliv stránce, v jakékoliv aplikaci. Stačí ji jen vložit jako HTML značku, případně s volitelnými atributy. V následujícím výpisu kódu je komponenta použitá na stránce aplikace pro přepis nahrávek. Je u ní zapnutý posouvač místa času, je zapnutá smyčka a tlačítko pro změnu rychlosti je viditelné.

Výpis 4.4 Použití vlastní komponenty fl-player v HTML

```
<fl-player src={ record.AudioURL } slider loop speed></fl-player>
```

Takto bylo implementováno 7 vlastních komponent:

- <fl-player> Vlastní audio přehrávač
- <fl-autocomplete> Našeptávač pro přepis letištní komunikace
- <fl-autofill> Vyplňovač štítků letištní komunikace
- <fl-toast> Notifikační oznámení
- <fl-copy-button> Tlačítko pro zkopírování odkazu nahrávky do schránky
- <fl-dialog-controller> Obslužný JS pro ovládání HTML5 dialogové komponenty
- <fl-collapsible> Obalová komponenta pro rozbalení/zabalení jakéhokoliv elementu
- <fl-confirm> Potvrzovací okno Ano/Ne

## 4.4 Zavedení druhé iterace

Backend byl publikován dle procesu nasazení v kapitole 3.3.2. Na kontinuální integraci v Github Actions byl sestaven nový Docker kontejner. Ten byl poté nastartován pomocí Docker compose scriptu. V něm jsou dodané proměnné, které aplikace nyní v druhé iteraci očekávala. Databáze zároveň byla přesunuta z Azure prostředí do lokálního jako vedlejší Docker kontejner. Ve výpisu níže označená jako služba *db*.

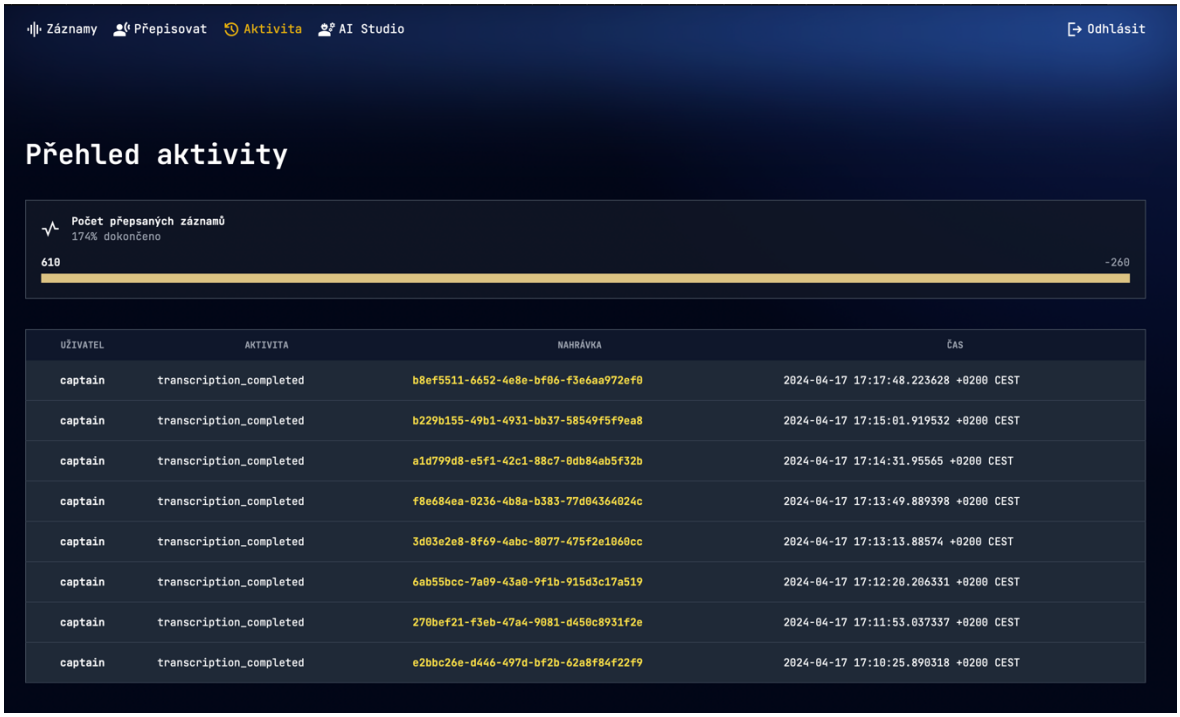
- Proměnné DB_ obsahují přihlašovací údaje do databáze *db*
- JWT_SECRET může být náhodný řetězec, který slouží jako privátní klíč pro generování přihlašovacích tokenů
- AZURE_CONNECTION obsahuje řetězec pro Azure Blob úložiště

Výpis 4.5 Produkční Docker Compose soubor backendu ve druhé iteraci

```
name: flightlog
services:
  server:
    image: ghcr.io/marhycz/flightlog-server:main
    ports:
      - 1323:1323
    environment:
      - DB_HOST=db
      - DB_PORT=5432
      - DB_USER=xxxxxx
      - DB_PASSWORD=xxxxxx
      - DB_NAME=flightlog
      - JWT_SECRET=xxxxxx
      - AZURE_CONNECTION=DefaultEndpointsProtocol=https;.......
      - TZ=Europe/Prague
    volumes:
      - /home/user/docker/storage/flightlog:/cache
    depends_on:
      - db
    labels:
      - "traefik.enable=true"
      - "traefik.http.routers.freshpoint.rule=Host(`flightlog.xxxxxx.xxxxxx.cz`)"
```

```
  db:
    image: postgres:alpine
    restart: always
    user: postgres
    volumes:
      - db-prod:/var/lib/postgresql/data
    environment:
      - POSTGRES_DB=flightlog
      - POSTGRES_PASSWORD=xxxxxx
      - POSTGRES_USER=xxxxxx
    ports:
      - 5400:5432
    healthcheck:
      test: [ "CMD", "pg_isready" ]
      interval: 10s
      timeout: 5s
      retries: 5
volumes:
   db-prod:
```

V průběhu zavedení druhé iterace byla ještě přidána do projektu funkce s přehledem aktivity. Ta zobrazovala poslední změněné nahrávky za 24 hodin pro kontrolu.




Obrázek 24 Dodatečná obrazovka pro přehled aktivity

Tímto byla druhá iterace uzavřena. Letecké nahrávky mohly být nyní přepisovány a štítkovány do datového modelu a paralelně mohla startovat třetí iterace pro analýzu a trénování strojového modelu.

# 5 Výběr, trénování, vyhodnocení modelu strojového učení

Ve třetí iteraci je nyní díky této práci připravena získaná množina dat pro trénování vlastního modelu řeči na text. Hypotézou je, že při dotrénování modelu na doménově specifické oblasti (letecké komunikaci) se jeho přesnost zvýší natolik, aby bylo možné spolehlivě extrahovat automaticky důležitá data definovaná v leteckém datovém modelu v kapitole 4.1 a 4.2.

Třetí iterace bude znovu dle MMSP probíhat zahájením a rozpracováním fáze ve formě analýzy dostupných modelů (definice kritérií a následný výběr). Ve fázi konstrukce se exportují připravená data přímo ve formátu pro daný model a spustí se jeho trénování. Na konci každého trénování se provede vyhodnocení kvality natrénovaného modelu.

## 5.1 Analýza dostupných modelů

V úvodu práce bylo zmíněno, že pro zachování diverzifikace za účelem získání více vědeckých výsledků budou použity modely jak z řady komerčních/uzavřených (tzv. closed-source), tak volných modelů, které jsou otevřené a se svobodnou licencí. Při vyhodnocení modelů může zároveň práce identifikovat, zda mají komerční modely navrch.

### 5.1.1 Kritéria výběru modelu

Pro tuto práci musí model splňovat následující funkční kritéria:

- Vícejazyčný režim včetně češtiny – jeho výchozí natrénovaný stav již musí obsahovat korpus pro přepis více jazyků včetně češtiny
- Možnost dotrénování – model musí mít k dispozici knihovny či jiný software, který umožní jeho dotrénování na vlastních datech za účelem zpřesnění přepisu

### 5.1.2 Kritéria vyhodnocování modelů

Pro vyhodnocení úspěšnosti modelů byla vybrána standardní metrika WER[14]. Tedy „Poměr chybných slov“. [39]

$$WER = \frac{počet\ substitucí + počet\ smazaných\ slov + počet\ chyb}{celkový\ počet\ slov}$$

[14] Word Error Rate

Výsledná hodnota je poté vyjadřována v procentech. Nižší hodnota znamená nižší chybovost modelu, a tudíž lepší výsledek.

## 5.2 Azure Custom Speech To Text

Kandidáty pro komerční modely řeči na text byla zvolena řešení lídrů z oblasti poskytovatelů cloudových služeb, jak je definoval Gartner ve svém magickém kvadrantu pro rok 2023. [40]

- Microsoft se svým Azure
- Google s Google Cloud
- Amazon s Amazon Web Services


Obrázek 25 Magický kvadrant s lídry v oblasti poskytovatelů cloudových služeb (Gartner, 2023)

Amazon dle jejich oficiální dokumentace nyní nepodporuje trénování modelu nad vlastními daty pro český jazyk. [41]

Google Cloud Speech-to-Text podporuje češtinu, ale dotrénování modelu probíhá pouze s textovými daty. Tedy je možné do modelu dodat frekventovaná slova, výrazy, které se v komunikaci častěji vyskytují, ale není možné model dotrénovat včetně vlastních audio dat s jejich přepisy. [42]

Vybraným vzorkem pro komerční model řeči na text je nakonec Azure Speech. Jedná se o službu součástí cloudového řešení zvaného Azure provozovaného společností Microsoft. Pro akademické účely studentům vysokých škol zároveň nabízí prostředky ve formě 100 dolarů na vyzkoušení jejich služeb. Služba převod řeči na text Azure Speech podporuje dotrénování modelu na vlastních audio nahrávkách s přepisy včetně češtiny. [31]

Jelikož se jedná o software jako služba (SaaS), je zprovoznění a konfigurace jednodušší. Pro trénování vlastního modelu služba očekává nahrání .zip souboru, ve kterém jsou v jednotlivých souborech .wav audio nahrávky, a poté jakýkoliv 1 textový soubor, který obsahuje 2 sloupce oddělené tabulátorem. A to název audio souboru a jeho textový přepis.

### 5.2.1 Možnosti trénování Azure Custom Speech

Oficiální dokumentace Azure Speech publikuje seznam podporovaných možností doménového trénování dle jazyka. [43]

V českém jazyce umožňuje trénování pomocí textových přepisů a kombinace audia + přepisů.

V režimu doménového trénování modelu pouze s přepisy se nevkládají audio nahrávky, ale pouze textové ukázky. Tak jak vypadá běžná konverzace v dané doménové oblasti. Model se neučí z nového audia, ale pouze nastaví své vnitřní váhy více na slova, která se častěji v konkrétní oblasti vyskytují.

Druhou metodou je trénování pomocí kombinace audio nahrávek s přepisy, které je preferované a vede dle dokumentace k mnohem lepším výsledkům.

Obě metody trénování lze konfigurovat Váhou, což je jediný konfigurační parametr u trénování Azure modelů, který umožňuje od 1 do 100 definovat, jak moc se má model přiklánět k trénovacím datům. Čím větší číslo, tím více bude model upřednostňovat slova z trénovacího datasetu, a bude tím méně generický.

## 5.3 OpenAI Whisper Large

Pro výběr zástupce z řady otevřených modelů byla pro analýzu využita data z organizace Hugging Face. Tato organizace se zaměřuje na tvorbu svobodných podpůrných softwarových knihoven pro trénování a provozování modelů strojového učení. Zároveň umožňuje, podobně jako GitHub se softwarem, publikaci modelů, datasetů, jejich vyhodnocování a diskuzi nad nimi. [44]

Hugging Face zároveň publikuje veřejnou tabulku ve formě výkonnosti modelů napříč sadou volně dostupných datasetů, které obsahují štítkované nahrávky lidské řeči. [45] Například Common Voice od Mozilla Foundation, který obsahuje téměř 20 000 hodin přepsané řeči napříč 120 jazyky. [46]

Po odfiltrování modelů, které jsou zaměřené pouze na anglický nebo jiný jazyk (což je dle Hugging Face aktuálně 5 variací stejného modelu od NVIDIA), byl na prvním místě model Whisper Large v3 od firmy OpenAI.

OpenAI je známá v oblasti velkých jazykových modelů, které pohání službu ChatGPT. Avšak firma také poskytuje služby pro převod textu na řeč a řeči na text. A zatímco nejnovější LLM modely a modely textu na řeč nejsou svobodně publikované, jejich strojový model řeči na text je volně dostupný pod názvem Whisper se svobodnou MIT licencí. [47]

### 5.3.1 Koncepty fungování modelu Whisper

Model Whisper od OpenAI byl vyvinut s cílem vytvořit co nejvíce různorodého modelu s minimální nutnou mírou přizpůsobení na konkrétní dataset nebo doménovou oblast.

Tohoto cíle se snažili lidé z OpenAI dosáhnout pomocí trénování na obrovském korpusu dat. Whisper je natrénován na 680 000 hodinách štítkovaných audio záznamů přes 97 různých jazyků. Výsledkem je model v jeho největší variantě „large“ s 1,6 miliardou parametrů.

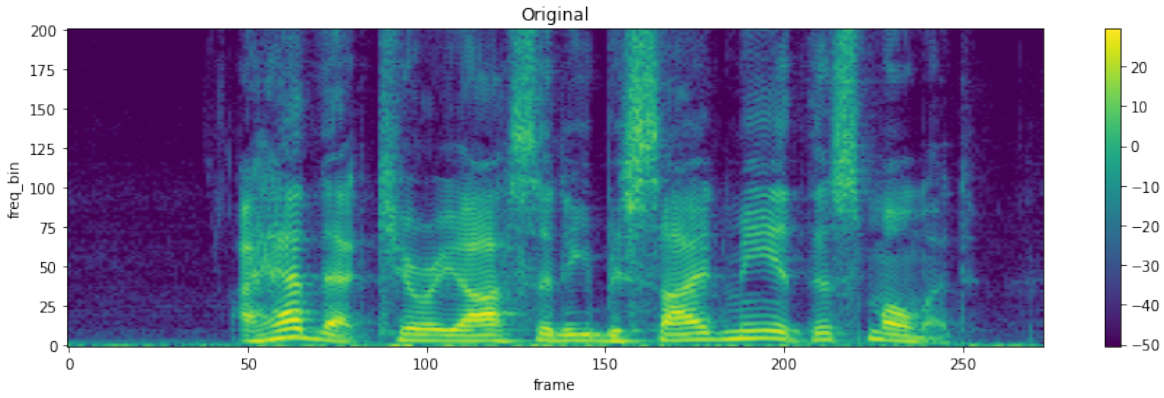
Model je schopen pracovat i bez nutnosti dalšího přizpůsobování pro specifické úlohy, ale stále je možné ho dotrénovat. Whisper funguje na dvou hlavních konceptech. Architektuře transformátorů a spektrogramech.

### Architektura transformátorů

Transformátor je typ strojového učení, který se používá k analýze sekvencí dat. Tím může být právě text nebo zvuk. U Whisperu se používá transformátorový model dvou částí: Enkodér-Dekodér. Enkodér přijímá vstupní data a transformuje je na jeho vnitřní reprezentaci. Ta se snaží uchovat ty nejdůležitější informace ze vstupních dat. Dekodér poté tuto reprezentaci používá ke generování výstupů (zde textového přepisu řeči), které jsou co nejvíce pravděpodobné při přijatém vstupu z výstupu enkodéru. [47]

### Spektrogramy

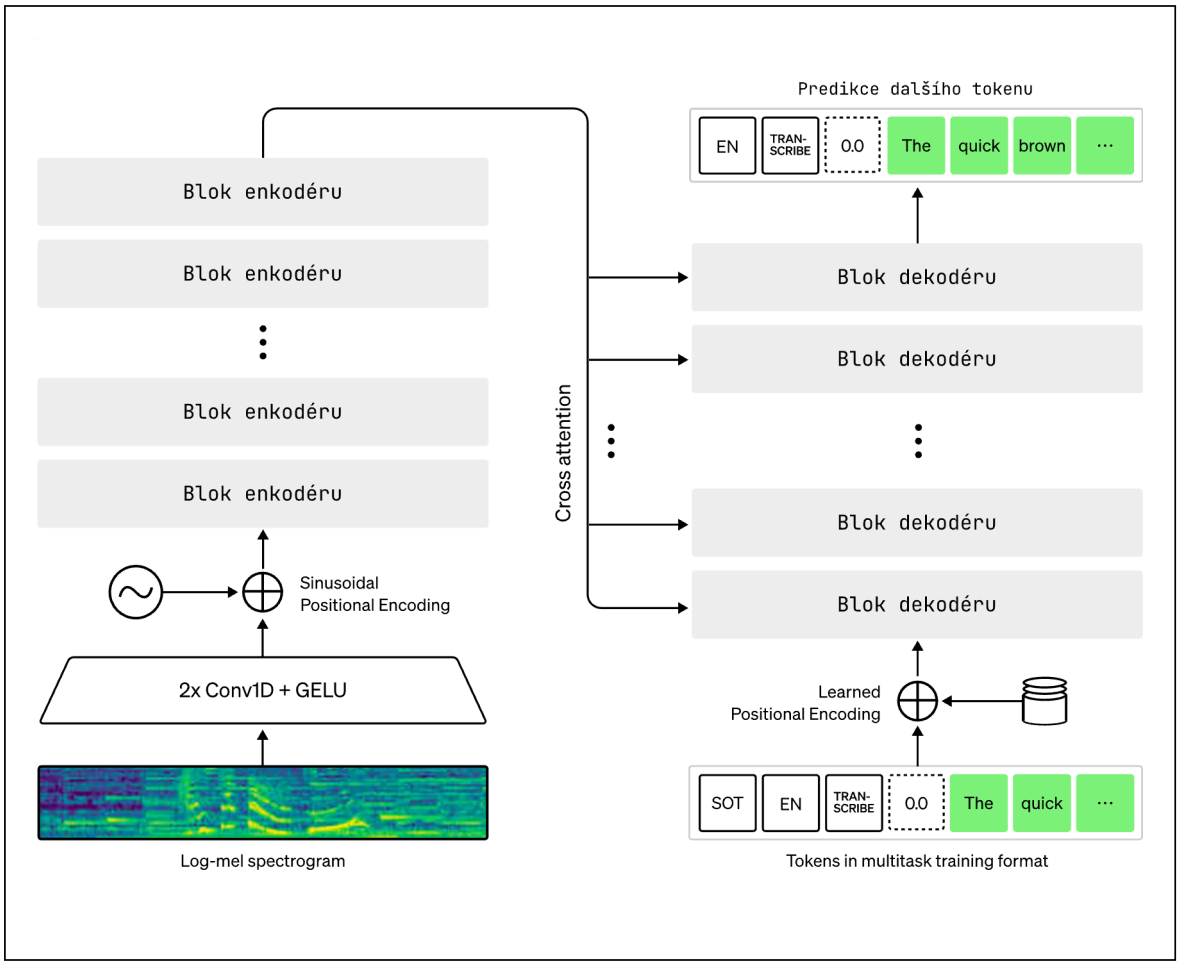
Když Whisper pracuje se zvukovým signálem, převádí nejprve zvuk do formy, která se nazývá spektrogram. Spektrogram je vizuální reprezentace frekvencí (výšek tónů), které se ve zvuku vyskytují v průběhu času. Na ose x spektrogramu je čas a na ose y jsou frekvence. Barvy na spektrogramu ukazují intenzitu různých frekvencí v různých časech. [47]




Obrázek 26 Ukázka spektrogramu vygenerovaného z audia

### Výsledný proces Whisper modelu

Whisper nejprve vezme zvukový záznam, jako je záznam komunikace pilota s věží, a převede ho na spektrogram. Tento spektrogram pak projde enkodérem, který vyhodnotí a zakóduje z něho důležité informace. Informace jsou následně předány do dekodéru, který interpretuje zakódované informace tak, že z nich vygeneruje text odpovídající slyšeným slovům.



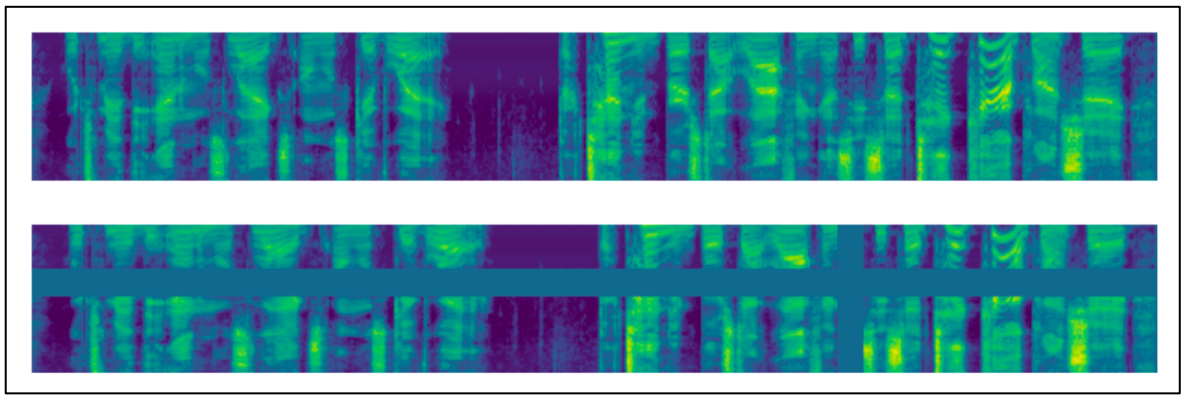

Obrázek 27 Obecná architektura modelu Whisper (zdroj: OpenAI)[47]

### 5.3.2 SpecAugment

Při rešerši trénování modelů pro práci byla také nalezena jedna z optimalizačních metod pro trénovaní modelů přepisu řeči na text zvaná SpecAugment. Tato technika se aplikuje přímo na logaritmické mel-spektrogramy vstupního audia, které se zpracovávají jako by šlo o obraz.

SpecAugment zahrnuje tři druhy deformací spektrogramu: deformace v čase (time warping), kde se časová osa náhodně posouvá, a maskování frekvencí a časových úseků, při kterém se určité bloky frekvencí nebo časových kroků "zakryjí". Tato jednoduchá, ale účinná metoda umožňuje modelům učit se rozpoznávat řeč v různých ztížených akustických podmínkách. [48]

Touto metodou se má zvýšit odolnost modelů vůči různým ztíženým podmínkám. Avšak letecká komunikace již sama o sobě bývá mnohdy šumivá, a ne zcela srozumitelná. Trénování modelu tedy bude probíhat jak s použitím SpecAugmentu, tak bez. Knihovna Transformers od Hugging Face již totiž implementuje základní logiku SpecAugmentu.



Obrázek 28 Příklad aplikace SpecAugment na Spektrogramu

Na obrázku výše je vidět princip aplikace SpecAugment metody. Lehce se zdeformují a zakryjí části spektrogramu, takže model se učí za ztížených podmínek.

### 5.3.3 Možnosti trénování Whisper

Whisper model umožňuje trénování pouze s textovými přepisy pomocí trénování jen dekodérové části. Ale tato metoda se nedoporučuje. Jelikož je Whisper ve výchozím stavu robustní s 96 jazyky a 680,000 hodinami lidské řeči, nemá problém s textovou reprezentací. I doménově specifická slova má již ve svém interním korpusu. Trénování jen dekodérové části tak může rozbít všechny natrénované schopnosti modelu. [49, 50]

Pro Whisper model se tedy doporučuje trénování hlavně pomocí kombinace audio + přepisů, nebo použití tzv. promptu.

Whisper umožňuje na začátek dekodérové části přidat textovou ukázku, která může pomoci při predikci dalších tokenů, které budou následovat (tedy přepisu audio části). To pomůže modelu se lépe orientovat v obsahu a struktuře rozhovoru. Tedy pro situace, kdy dochází k jazykovým nebo dialektickým nejasnostem. Prompt u Whisperu je jednodušší formou jazykového modelu ve srovnání s OpenAI GPT-3 a GPT-4 modely. Zadání ve stylu „Jedná se o leteckou komunikaci, upřednostni NATO abecedu“, nebude fungovat. Avšak prompt, který obsahuje ukázku komunikace nebo slova, která se v přepisu běžně vyskytují, pomáhá. Tedy například „Boleslav RADIO Alpha Beta Charlie Delta Gamma Oscar Kilo Charlie dráha dva dva levá výška tisíc osm set“ ukáže, že má model upřednostnit daná slova a číslice přepisovat v textovém formátu. Prompt by měl být ideálně dynamický dle dané nahrávky (tedy vložit do promptu volací značku daného letiště, dráhy a podobně).

Ve výzkumu firmy MediaTek se pokusili o stejnou metodu na anglickém datasetu letecké komunikace. Použití trénování dekodérové části jen s přepisy vedlo nakonec ke zlepšení z 74 % WER na 51 %. Při použití promptu z 74 % na 56 %. A při doménovém trénování s použitím audia z 74 % na 24 %. [50] V této práci se z důvodu komplexnosti přistoupilo rovnou na trénování pomocí preferovaného audia + přepisu.

## 5.4 Export trénovacích dat

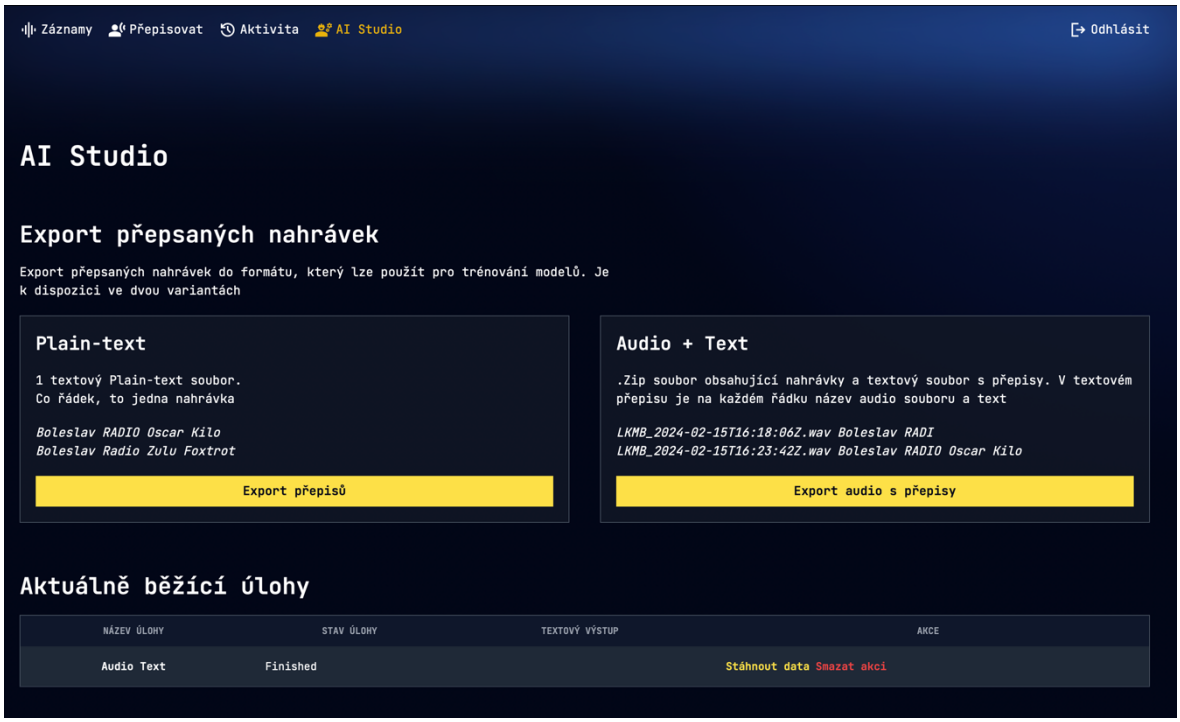
První část konstrukční fáze v této iteraci bylo vytvoření exportovacího modulu do backendové aplikace projektu.

Požadavky na tuto funkcionalitu byly následující:

- Vyfiltrují se veškeré nahrávky s příznakem, že jsou přepsané člověkem
- Z cloudového úložiště se všechny tyto nahrávky stáhnout do dočasného úložiště
- Nahrávky se náhodně oddělí v poměru 90/10. (Trénovací/Testovací)[51 s. 98]
- Vytvoří se textový soubor, kde každý řádek bude obsahovat:
    - Název audio nahrávky, tabulátor a poté přepis dané nahrávky
- Výsledek se zabalí do .zip souboru, který je poté možné vyzvednout.

Pro splnění funkčních požadavků byly do backendové aplikace implementovány další 3 služby. Ty jsou součástí zdrojového kódu:

- Základní podpora prací, které mohou běžet na pozadí. O to se stará služba *worker*.
- Služba cache, která umožní dočasně uložit soubory na disk.
- Služba jobs, která obsahuje jednotlivé skripty pro práce na pozadí, včetně exportu datasetu pro učení modelu.




Obrázek 29 Ukázka rozhraní pro export nejnovějšího leteckého datasetu

Následující ukázka kódu zachycuje logiku v backendové aplikaci (psané v Go) pro extrakci všech přepsaných nahrávek a vytvoření výsledného .zip souboru. Výpis je záměrně zkrácen o chybové stavy pro lepší čitelnost.

Výpis 5.1 Export .zip s daty pro učení modelu v Go

```
func ExportAudioWithTranscriptions(dir string, r *record.RecordService) (path string, err
error) {
	records, err := r.ListRecords(record.ListRecordsParams{
		Page:               0,
		PerPage:            500,
		HumanTranscription: true,
	})
	archive, err := os.CreateTemp(dir, "audio_transcriptions*.zip")
	defer archive.Close()

	zipWriter := zip.NewWriter(archive)
	train, test := splitRecords(records)

	saveRecordsToZip(train, "train/", zipWriter, r.AudioRepo)
	saveRecordsToZip(test, "test/", zipWriter, r.AudioRepo)
	zipWriter.Close()
	return archive.Name(), nil
}

func saveRecordsToZip(records []record.Record, dir string, zipWriter *zip.Writer, a
*audiostore.AudioRepository) error {
	txtOutput := ""
	for _, rec := range records {
		txtOutput += fmt.Sprintf("%s\t%s\n", rec.Audio, rec.Transcript.String)

		// Add audio file to zip
		audio, err := a.LoadAudio(rec.Airport, rec.Audio)
		audioWriter, err := zipWriter.Create(dir + rec.Audio)
		_, err = audioWriter.Write(audio)
	}
	txtWriter, err := zipWriter.Create(dir + "transcriptions.txt")
	_, err = io.WriteString(txtWriter, txtOutput)
	return nil
}

func splitRecords(records []record.Record) (train, test []record.Record) {
	for _, rec := range records {
		if rand.Intn(100) < 90 {
			train = append(train, rec)
		} else {
			test = append(test, rec)
		}
	}
	return
```

```
}
```

Výsledkem je .zip soubor, který obsahuje složky *„train“* a *„test“* s poměrem 90/10.

Výpis 5.2 Výsledná struktura exportu .zip leteckého datasetu

```
|-test
 | |-transcriptions.txt
 | |-LKMB_2024-03-15T07-59-54Z.wav
 | |……
 | |-LKMB_2024-02-17T12-55-57Z.wav
|-train
 | |-transcriptions.txt
 | |-LKMB_2024-03-04T14-56-23Z.wav
 | |……
 | |-LKMB_2024-03-20T11-47-48Z.wav
```

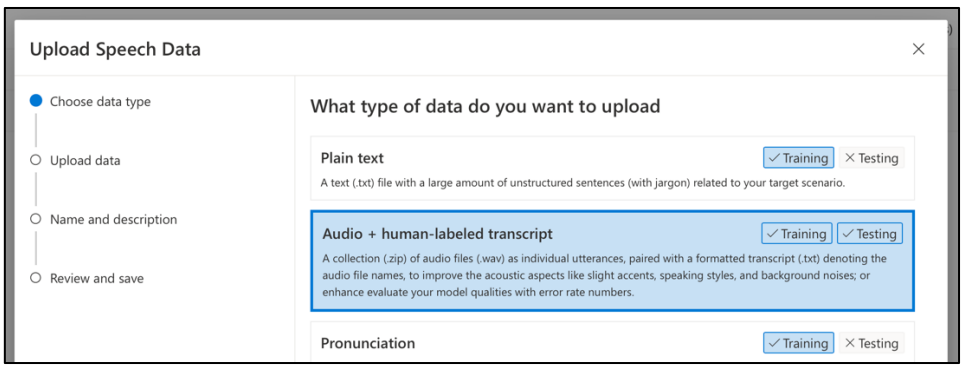
Výpis 5.3 Ukázka souboru s přepisem

```
LKMB_2024-03-17T05-48-26Z.wav        Boleslav RADIO November dva dva opouštím ATZ na
severu
```

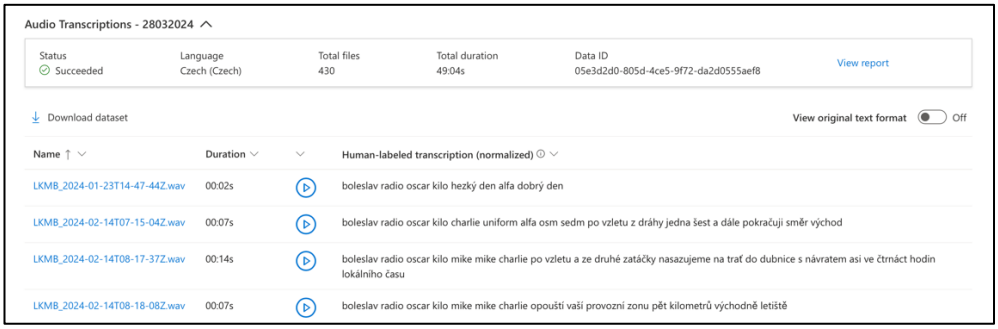
Ve výpisu výše je ukázka jednoho řádku ze souboru *transcriptions.txt*. Tabulátorem oddělený název souboru a vedle jeho přepis.

## 5.5 Trénování Azure Custom Speech

Trénování modelu v Azure bylo jednodušší, protože se jedná o software dodávaný jako služba. V portálu Speech Studio stačí vytvořit nový projekt typu „Custom Speech“. Poté nahrát testovací a trénovací data. Azure očekává separátní zip soubory, takže jen stačilo separátně zazipovat složky *„train“* a „test“ z backendového exportu.




Obrázek 30 Azure Speech – nahrání vlastního datasetu (archiv autora)




Obrázek 31 Ukázka nahraných přepisů v Azure Speech Studio

Poté již jen stačí spustit trénování nad tímto datasetem. Jediný konfigurační parametr, který Azure Custom Speech nabízí, je váha vlastního datasetu od 0 do 100.

## 5.6 Trénování Open AI Whisper

Pro trénování modelu Whisper byly použité knihovny od Hugging Face pro trénování sekvenčních modelů. Ty jsou dostupné v jazyce Python, takže celá část práce s modelem strojového učení probíhala v Python prostředí.

Hlavním modulem je knihovna Transformers, která nabízí většinu nástrojů pro trénování a inferenci (provoz) sekvenčních modelů. Jelikož jsou knihovny stále pod aktivním vývojem, ukázkové skripty pro trénování obsahovaly nesrovnalosti s konfigurací Whisper modelu pro trénování modelu pro jiné jazyky než angličtinu. Autor práce tedy zároveň přispěl s opravou do této knihovny na GitHubu. [52]

### 5.6.1 Příprava datasetu

Dataset exportovaný z backendové aplikace bylo potřeba načíst do formátu, který očekává knihovna Transformers. K tomu slouží knihovna Audio od Hugging Face. Stačilo pouze vytvořit list objektů s dvěma proměnnými „audio“, který obsahuje datový proud audia a „sentence“ obsahující textový přepis dané nahrávky.

Výpis 5.4 Třída vlastního leteckého datasetu pro Hugging Face

```
import os
from datasets import Dataset, DatasetDict, Audio, Value, Features

class FlightlogDataset:
    def __init__(self):
        """
        dataset_path: str, path to the dataset directory containing 'train' and 'test'
subdirectories.
        """
        self.dataset_path = os.path.dirname(os.path.abspath(__file__))
```

```
    def _load_data(self, split):
        """
        Load data for a given split ('train' or 'test').
        Directly include the 'audio' field as a dictionary with 'path' to the audio file.
        """
        data_path = os.path.join(self.dataset_path, split, "transcriptions.txt")
        examples = []
        with open(data_path, "r", encoding="utf-8") as f:
            for line in f:
                filename, transcription = line.strip().split("\t", 1)
                audio_path = os.path.join(self.dataset_path, split, filename)
                examples.append({"audio": audio_path, "sentence": transcription})
        return examples

    def load_dataset(self, split: str) -> Dataset:
        """
        Load Hugging Face dataset ('train' or 'test').
        """
        examples = self._load_data(split)

        return Dataset.from_dict(
            {
                "audio": [example["audio"] for example in examples],
                "sentence": [example["sentence"] for example in examples],
            },
            Features(
                {
                    "audio": Audio(sampling_rate=16000),
                    "sentence": Value("string"),
                }
            ),
        )
```

### 5.6.2 Příprava trénovacího prostředí

Model byl trénován na stroji s 64 GB RAM, GPU Nvidia RTX 4070 Ti s 12 GB VRAM a operačním systémem Fedora Workstation 40.

Při trénování modelu Whisper Large s metodou DeepSpeed bylo využito 48 GB RAM a 9,5 GB VRAM.

#### Nvidia NGC s PyTorch

Knihovny od Hugging Face běží nad Python frameworkem PyTorch. Pro akceleraci na GPU využívá PyTorch knihovnu CUDA (Nvidia) nebo ROCm (AMD). Jelikož zprovoznění

akcelerace vyžaduje instalaci mnoha závislostí v různých specifických verzích, vydává Nvidia své Docker kontejnery pod názvem Nvidia NGC. Tyto kontejnery mají připravené PyTorch prostředí s CUDA akcelerací. [53]

Předpokladem byl pouze Docker na hostitelském Linux systému, poslední verze CUDA ovladače a malá podpůrná knihovna od Nvidia pro kontejnerové prostředí zvaná Nvidia Container Toolkit. [54]

Následující výpis kódu zachycuje bash skript pro přípravu trénovacího prostředí v této práci, které běželo nad Fedora Workstation 40.

Výpis 5.5 Příprava trénovacího prostředí v Linuxu pro PyTorch s CUDA

```
# Aktualizace OS a balíčků
sudo dnf update --refresh

# Instalace proprietárních CUDA ovladačů
sudo dnf install xorg-x11-drv-nvidia-cuda

# Docker
sudo dnf install docker
sudo usermod -aG user docker
# Install Nvidia container toolkit to run deepspeed container
curl -s -L https://nvidia.github.io/libnvidia-container/stable/rpm/nvidia-container-
toolkit.repo |   sudo tee /etc/yum.repos.d/nvidia-container-toolkit.repo
sudo dnf install -y nvidia-container-toolkit

# Aktivace Container Toolkit
sudo nvidia-ctk runtime configure --runtime=docker
sudo systemctl restart docker
```

### Spuštění kontejneru

Po instalaci dockeru a Nvidia závislostí již stačí pustit Docker NGC kontejner.

Výpis 5.6 Skript pro start PyTorch Docker kontejneru

```
#!/bin/bash
docker run -it \
--name pytorch \
--gpus all \
--security-opt=label=disable \
--ipc=host --ulimit memlock=-1 --ulimit stack=67108864 \
--net=host \
-v ${PWD}:/workspace:Z \
nvcr.io/nvidia/pytorch:24.03-py3 /bin/bash
```

Výpis bash skriptu výše startuje Docker kontejner s vlastním názvem “pytorch”.

- “gpus all” do kontejneru připojuje veškeré grafické karty
- “ipc” a “ulimit” jsou doporučená nastavení od Nvidia, která se vypíšou do logu při spuštění kontejneru bez těchto parametrů
- “net host” jen zpřístupní porty uvnitř kontejneru v síti LAN. Pro pozdější provoz modelu
- “v” parameter vloží aktuální složku, ve které je script spuštěn

### DeepSpeed

Pro efektivní trénovaní strojových modelů se většinou využívá akcelerace pomocí grafického adaptéru (GPU). Avšak problém nastává v momentě, kdy velké modely, jako například Whisper Large, potřebují pro svůj běh desítky GB video paměti (VRAM). I silná a výkonná grafická karta model nespustí, pokud se nevejde celý do paměti. Tato limitace byla prolomena výzkumným projektem DeepSpeed z laboratoří Microsoftu. Jedná se o optimalizační knihovnu pro PyTorch, která implementuje mnoho metod pro snížení hardwarových nároků. Jedním z nich je využití klasické paměti RAM jako odkladného úložiště (offload). To umožňuje provoz mnohem větších modelů i se spotřebitelskými grafickými kartami. [55]

Ve spuštěném kontejneru stačí knihovnu doinstalovat přes správce Python balíčků pip. Všechny závislosti a jejich instalaci včetně DeepSpeed ukazuje následující výpis.

Výpis 5.7 Python závislosti pro trénování

```
pip install -r requirements.txt

requirements.txt :
# Torch je již nainstalovaný uvnitř Nvidia NGC kontejneru
datasets[audio]
transformers
evaluate
jiwer
deepspeed
accelerate
```

## 5.6.3 Trénovací skript

Níže je postupně popsaný trénovací skript. Chybí pouze část s importy závislostí, která není potřebná pro vysvětlení. Kompletní skript je k dispozici v elektronické příloze.

Výpis 5.8 Trénovací skript – parametry

```
@dataclass
class args:
    apply_spec_augment = True
    model_name = "openai/whisper-large-v3"
```

```
    language = "czech"
    task = "transcribe"
    max_input_length = 30.0  # seconds
    do_normalize_eval = True
```

Na začátku jsou definované některé parametry, které se v kódu opakují. Zda se má aktivovat metoda SpecAugment, název repozitáře výchozího modelu z Hugging Face. Jazyk a úloha (Whisper umí i překládat cokoliv do angličtiny, zde je pouze přepis). Whisper funguje na 30 vteřinových kouscích audia. A poslední parametr mění, zda se má pro výpočet WER použít normalizovaný text. Ideální je učit Whisper předvídat i kapitálky (např. „ATZ“ místo „atz“). Ovšem při vyhodnocování modelu by se nemělo brát jako chybu, pokud model správně text přepíše jen s chybnými kapitálkami.

Výpis 5.9 Trénovací skript – načtení vlastního datasetu

```
dataset = DatasetDict()
dataset["train"] = FlightlogDataset().load_dataset(split="train")
dataset["test"] = FlightlogDataset().load_dataset(split="test")
```

Prvním krokem je načtení vlastního datasetu. Výpis kódu definice datasetu je výše v kapitole 5.6.1. Zde se vytvoří nový slovník *DatasetDict* z knihovny Hugging Face a z vlastního datasetu se načtou trénovací a testovací audio ukázky.

Výpis 5.10 Trénovací skript – inicializace Whisper Processoru

```
processor = WhisperProcessor.from_pretrained(
    args.model_name, language=args.language, task=args.task
)
```

Hned v dalším kroku se inicializuje instance WhisperProcessoru. Jedná se třídu, která sdružuje dva objekty dohromady pro lepší práci. WhisperTokenizer a WhisperFeatureExtractor. Ty mohou být načteny i separátně. Tyto nástroje připraví vstupní audio.

Výpis 5.11 Trénovací skript – Inicializace modelu s konfigurací

```
model_config = WhisperConfig.from_pretrained(args.model_name)

model_config.update(
    {
        "apply_spec_augment": args.apply_spec_augment,
        "use_cache": False,
    }
)

model = WhisperForConditionalGeneration.from_pretrained(
```

```
    args.model_name, config=model_config
)

model.generation_config.language = "cs"
model.generation_config.task = args.task
model.generation_config.forced_decoder_ids = None

forward_attention_mask = (
    getattr(model_config, "apply_spec_augment", False)
    and getattr(model_config, "mask_time_prob", 0) > 0
)
```

Následně se načte výchozí konfigurace Whisper modelu WhisperConfig. Výchozí konfigurace se aktualizuje podle toho, zda se má aktivovat metoda SpecAugment. Poté se konfigurace přiřadí k načtenému modelu.

Výpis 5.12 Trénovací skript – Předzpracování datasetu pro Whisper

```
def prepare_dataset(batch):
    sample = batch["audio"]

    # compute log-Mel input features from input audio array
    inputs = processor.feature_extractor(
        sample["array"],
        sampling_rate=sample["sampling_rate"],
        return_attention_mask=forward_attention_mask,
    )
    batch["input_features"] = inputs.get("input_features")[0]

    # compute input length of audio sample in seconds
    batch["input_length"] = len(sample["array"]) / sample["sampling_rate"]

    if forward_attention_mask:
        batch["attention_mask"] = inputs.get("attention_mask")[0]

    # encode target text to label ids
    batch["labels"] = processor.tokenizer(batch["sentence"]).input_ids
    return batch
```

Výpis 5.13 Trénovací skript – Spuštění předzpracování datasetu pro Whisper

```
dataset = dataset.map(
    prepare_dataset,
    remove_columns=next(iter(dataset.values())).column_names,
    num_proc=1,
```

```
)


def is_audio_in_length_range(length):
    return length < args.max_input_length


dataset["train"] = dataset["train"].filter(
    is_audio_in_length_range,
    input_columns=["input_length"],
)
```

Po jejím definování se funkce nad datasetem spustí. Tím se pro každou nahrávku přidá sloupec s tokeny a spektrogramem. Zároveň se odeberou původní sloupce s přepisem a audiem, které již nejsou potřeba.

Výpis 5.14 Trénovací skript – Výpočet chyby WER

```
metric = evaluate.load("wer")

normalizer = BasicTextNormalizer()

def compute_metrics(pred):
    pred_ids = pred.predictions
    label_ids = pred.label_ids

    # replace -100 with the pad_token_id
    label_ids[label_ids == -100] = processor.tokenizer.pad_token_id

    pred_str = processor.tokenizer.batch_decode(pred_ids, skip_special_tokens=True)
    # we do not want to group tokens when computing the metrics
    label_str = processor.tokenizer.batch_decode(label_ids, skip_special_tokens=True)

    if args.do_normalize_eval:
        pred_str = [normalizer(pred) for pred in pred_str]
        label_str = [normalizer(label) for label in label_str]
        # filtering step to only evaluate the samples that correspond to non-zero
references:
        pred_str = [pred_str[i] for i in range(len(pred_str)) if len(label_str[i]) > 0]
        label_str = [
            label_str[i] for i in range(len(label_str)) if len(label_str[i]) > 0
        ]

    wer = metric.compute(predictions=pred_str, references=label_str)
    return {"wer": wer}
```

Pro ověření modelu se načte z knihovny *evaluate* funkce pro výpočet WER. Zároveň se načte z knihovny Whisper *normalizer*, který provede převedení původního, i modelem přepsaného textu na stejný formát – převede čísla na text, velká písmena na malá.

Model bude hodnocen jako úspěšný i v případě, že odhadne špatně kapitálky nebo rovnou napíše číslo číselným než slovním zápisem. Tyto případy jsou jen estetické a na význam přepisu komunikace nemají dopad. Nebudou tedy upřednostňované pro model. Toto jde vypínat a zapínat argumentem *do_normalize_eval* na začátku trénovacího skriptu.

Výpis 5.15 Trénovací skript – DataCollator třída

```
@dataclass
class DataCollatorSpeechSeq2SeqWithPadding:
    processor: WhisperProcessor
    decoder_start_token_id: int
    forward_attention_mask: bool

    def __call__(
        self, features: List[Dict[str, Union[List[int], torch.Tensor]]]
    ) -> Dict[str, torch.Tensor]:
        # split inputs and labels since they have to be of different lengths and need
different padding methods
        # first treat the audio inputs by simply returning torch tensors
        input_features = [
            {"input_features": feature["input_features"]} for feature in features
        ]
        batch = self.processor.feature_extractor.pad(
            input_features, return_tensors="pt"
        )

        # get the tokenized label sequences
        label_features = [{"input_ids": feature["labels"]} for feature in features]

        if self.forward_attention_mask:
            batch["attention_mask"] = torch.LongTensor(
                [feature["attention_mask"] for feature in features]
            )

        # pad the labels to max length
        labels_batch = self.processor.tokenizer.pad(label_features, return_tensors="pt")

        # replace padding with -100 to ignore loss correctly
        labels = labels_batch["input_ids"].masked_fill(
            labels_batch.attention_mask.ne(1), -100
        )
```

```
        # if bos token is appended in previous tokenization step,
        # cut bos token here as it's append later anyways
        if (labels[:, 0] == self.decoder_start_token_id).all().cpu().item():
            labels = labels[:, 1:]

        batch["labels"] = labels

        return batch
```

Posledním krokem před spuštěním trénování je definice tzv. DataCollatoru. Je to poslední krok zpracování dat pro sekvenční Whisper model. Jedná se o ukázkovou implementaci převzatou z Hugging Face. Pro rychlejší trénování modelu je běžné, že se několik nahrávek dává do jedné dávky trénování. V tomto případě je ale potřeba, aby všechny přepisy nahrávek (v jejich tokenové formě) měly stejnou délku.

Tato třída proces automatizuje. Vezme zpracovaný dataset, spektrogramy nechá být, ale zpracuje tokeny tak, aby měly všechny stejnou délku. V případě nutnosti je vyplní prázdnými tokeny (padding).

Výpis 5.16 Trénovací skript – zahájení trénování strojového modelu

```
training_args = Seq2SeqTrainingArguments(
    deepspeed="ds_config.json",
    output_dir="./flightlog-model",
    per_device_train_batch_size=16,
    gradient_accumulation_steps=2,  # increase by 2x for every 2x decrease in batch size
    learning_rate=5e-6,  # 1e-5 default
    warmup_steps=500,
    max_steps=7000,
    gradient_checkpointing=True,
    evaluation_strategy="steps",
    per_device_eval_batch_size=8,
    predict_with_generate=True,
    generation_max_length=225,
    save_steps=300,
    eval_steps=300,
    logging_steps=25,
    load_best_model_at_end=True,
    metric_for_best_model="wer",
    greater_is_better=False,
    push_to_hub=False,
    bf16=True,
)

transformers.logging.set_verbosity_info()
```

```
data_collator = DataCollatorSpeechSeq2SeqWithPadding(
    processor=processor,
    decoder_start_token_id=model.config.decoder_start_token_id,
    forward_attention_mask=forward_attention_mask,
)

trainer = Seq2SeqTrainer(
    args=training_args,
    model=model,
    train_dataset=dataset["train"],
    eval_dataset=dataset["test"],
    data_collator=data_collator,
    compute_metrics=compute_metrics,
    tokenizer=processor.feature_extractor,
)

processor.save_pretrained(training_args.output_dir)
print(trainer.evaluate())
trainer.train()
```

Poslední část skriptu již spouští trénovací algoritmus. Seq2SeqTrainingArguments jsou argumenty pro trénovací knihovnu Hugging Face.

- DEEPSPEED definuje konfigurační soubor pro optimalizační DeepSpeed knihovnu (dostupný v elektronické příloze)
- OUTPUT_DIR je cesta, kam bude natrénovaný model uložen
- PER_DEVICE_TRAIN_BATCH_SIZE – počet nahrávek zpracovaných najednou na GPU. V případě DeepSpeed a RTX 4070 Ti byl maximum 16. (mělo by se jednat o násobky 2)
- GRADIENT_ACCUMULATION_STEPS – v případě, že batch size je nižší, je potřeba zvýšit akumulaci. Pro každé snížení BATCH_SIZE dvěma by se akumulační kroky měly zvýšit na dvojnásobek
- LEARNING_RATE – Rychlost učení modelu. Dle oficiálního Whisper dokumentu by měla být maximální rychlost 1e-5. (0.00001). Vzhledem k malému datasetu byla snížena na polovinu této rychlosti
- WARMUP_STEPS – Počet kroků, než rychlost trénování nabude maximální rychlosti (viz. LEARNING_RATE). Pomáhá modelu se stabilizovat.
- MAX_STEPS – Maximální počet tréninkových kroků. V případě, že se model bude stále zlepšovat, je možné trénování protáhnout dále
- GRADIENT_CHECKPOINTING – Metoda pro snížení paměťových nároků za cenu zvýšení výpočetních nároků. Nezbytné pro komerční GPU.
- EVALUATION_STRATEGY – zde je definováno, že k vyhodnocování modelu bude probíhat průběžně při dosažení N kroků.

- PER_DEVICE_EVAL_BATCH_SIZE – kolik nahrávek se má najednou přepsat při vyhodnocování modelu (také úměrné dle video paměti GPU)
- PREDICT_WITH_GENERATE – Použije funkci *generate* pro predikci výstupů modelu během evaluace, což je typické pro sekvenční modely včetně Whisperu
- GENERATION_MAX_LENGTH – Maximální délka generovaného textu Whisper modelem při predikci
- SAVE_STEPS – Průběžné ukládání modelu každých N kroků
- EVAL_STEPS – Průběžná validace modelu každých N kroků
- LOGGING_STEPS – Každých N kroků se na standardní výstup terminálu vypíše aktuální stav (trénovací ztráta, epocha atp.)
- LOAD_BEST_MODEL – Hugging Face knihovna může vybrat nejlepší ze všech průběžně uložených modelů a vytvořit na něj zkratku v hlavní složce modelu
- METRIC_FOR_BEST_MODEL – Dle jaké metriky vybrat nejlepší model. Zde se definuje předem vytvořená WER
- GREATER_IS_BETTER – Zda hodnoty, které metrika vyhodnocení vrací, jsou lepší, pokud se zvyšují. V případě WER je lepší hodnota nižší, tudíž false.
- PUSH_TO_HUB – Zda se má model po dokončení tréninku nahrát na HuggingFace repozitář
- BF16 – V případě novější generace grafických karet lze využít podporu nového datového typu plovoucí čáry BF16. Snižuje paměťové nároky při zachování téměř stejné spolehlivosti modelu.

Zároveň je potřeba získat WER úspěšnost modelu ve výchozím stavu před trénováním. Proto je zavolána samostatně funkce *print(trainer.evalute())* před zahájením trénování *trainer.train()*. V případě chyby s použitím DeepSpeed knihovny je vhodné provést tuto evaluaci pouze na CPU (*use_cpu=True* a odebrat *deepspeed* v *training_args*) bez DeepSpeed knihovny. Poté řádek s *trainer.evaluate()* odebrat a následně již znovu trénovat s GPU a s použitím DeepSpeed.

## 5.7 Výsledky trénování modelů

Po dokončení trénování na stejném trénovacím a validačním datasetu u Azure Custom Speech a OpenAI Whisper modelu se výrazně zlepšily výkony obou modelů v doménové oblasti letecké komunikace.

430 nahrávek čítající 49 minut přepsané letecké komunikace a 41 validačních nahrávek obsahujících 5 minut komunikace, kterou modely ještě předtím neviděly.

Ve výchozím stavu bez trénování mají modely od Azure a OpenAI podobný výkon v oblasti letecké komunikace okolo 50 % WER. To ale znamená, že průměrně každé druhé slovo je špatné oproti původnímu přepisu (buď přidané slovo, které nebylo řečeno, špatně přepsané, nebo vynechané).

### 5.7.1 Azure Speech

Tabulka 5.1 Vyhodnocení modelu Azure Speech

| Model | Konfigurace | WER (%) |
|---|---|---|
| Azure Speech | Výchozí model | 53,13 |
| Azure Speech | Text – váha 30 | 35,19 |
| Azure Speech | Text – váha 70 | 34,21 |
| Azure Speech | Audio – váha 30 | 13,35 |
| Azure Speech | Audio – váha 50 | 12,66 |
| **Azure Speech** | **Audio – váha 70** | **12,10** |
| Azure Speech | Audio – váha 90 | 12,52 |

Prvním pokusem bylo vyzkoušení trénování Azure Speech modelu pouze s pomocí textových přepisů. Trénování této verze modelu bylo velice rychlé, při 430 textových přepisech bylo trénování hotové do 2 minut. Pouhý text vedl k velkému zlepšení. Z 53 % WER na 34 %.

Při použití trénování s kombinací audio nahrávek a vlastního přepisu se však výkon modelu výrazně zvýšil. WER tedy z původních 53 % dosahovalo nyní na 12-13 % dle váhy. Detekce volacích znaků NATO abecedy byla bez chyby.

Zvýšení váhy pro trénovací dataset z původních 30 % na 70 % mělo pozitivní vliv na snížení chybovosti. Avšak další zvýšení již vedlo k přeučení modelu (tzv. over-fitting) a zhoršilo jeho schopnost generalizace na validačním datasetu.

### 5.7.2 OpenAI Whisper Large v3

Výchozí výkon Whisper modelu byl na validačním datasetu mírně lepší než u Azure Speech. Avšak stále se jednalo o chybovost v řádu 50 % WER.

Iterace 1. až 3. verze modelu zde nejsou zmíněny, jelikož se jednalo o špatně nakonfigurovaný referenční trénovací skript od Hugging Face Transformers knihovny v případě vícejazyčného trénování Whisperu. Model nebyl správně natvrdo přpnutý do českého jazyka s pevně nastavenou úlohou pro přepis (druhá je překlad do angličtiny).

Tato práce přispěla k opravě na GitHubu projektu. [52] A změna je reflektována v popisovaném skriptu v kapitole výše 5.6.3

| Model | Konfigurace | WER (%) | Checkpoint modelu |
|---|---|---|---|
| Whisper Large v3 | Výchozí model | 50,62 [a] | N/A |
| Whisper Large v3 | Model v4 | 12,79 | 1000 |
| Whisper Large v3 | Model v5 | **9,04** | 3600 |
| Whisper Large v3 | Model v6 | 10,57 | 2800 |
| Whisper Large v3 | Model v7 | **9,04** | 2000 |

*Pozn:*[a] Byla navíc provedena normalizace českých čísel

Model v4 byla první korektní konfigurace trénování. Model dosáhnul WER 12,79 %, tedy obdobnému jako u Azure Speech.

```
Seq2SeqTrainingArguments:
bf16=True
learning_rate=1e-5

class args:
apply_spec_augment = False
```

Model v5 měl stejné argumenty jako v4, kromě aktivace SpecAugment metody, která nyní byla povolená. To vedlo k výraznému zlepšení z WER 12,79 % na 9,04 %.

```
Seq2SeqTrainingArguments:
bf16=True
learning_rate=1e-5

class args:
apply_spec_augment = True
```

Iterace modelu v6 pouze přepnula režim přesnosti plovoucí desetinné čárky. Pokus o využití staršího datového typu fp16 namísto bf16. WER se zhoršilo na 10,57 %.

U sedmé iterace modelu v7 se vrátil datový typ zpět na bf16 a byla otestována pomalejší rychlost učení modelu o polovinu pomalejší hodnotou 5e-6 (0,000005) než původní. To nemělo žádný vliv na WER, ale model ke stejnému WER konvergoval za kratší jednotku času.

# 6 Implementace a integrace modelu

Po úspěšném natrénování se druhá půlka čtvrté iterace projektu soustředila na implementaci služeb do zbytku architektury backendové aplikace, které mohou naučené modely zavolat Tak, aby bylo možné na požadavek ihned do modelu přeposlat nahrávku a obdržet zpět textový přepis.

Funkční požadavky implementace:

- Vzhledem k dostupnosti obou modelů by služba měla nejdříve zkusit nahrávku přepsat pomocí lokálně nasazeného Whisper modelu. V případě nedostupnosti modelu se použije cloudová služba Azure Speech
- V případě existence skóre u výsledku přepisu (jak si je model jistý svým přepisem) se toto skóre uloží společně s přepisem pro snadnější vyhledání problematických nahrávek

Výpis 6.1 Implementace rozhraní služby pro přepis nahrávky v backendu

```
type Transcript struct {
	Text       string
	Confidence float32
}
type TranscribeService interface {
	TranscribeAudio(adata []byte) (Transcript, error)
}

type UnifiedTranscribeService struct {
	azure AzureTranscribeService
	local LocalTranscribeService
}

func NewUnifiedTranscribeService() *UnifiedTranscribeService {
	return &UnifiedTranscribeService{
		azure: *NewAzureTranscribeService(),
		local: *NewLocalTranscribeService(),
	}
}

func (t *UnifiedTranscribeService) TranscribeAudio(adata []byte) (Transcript, error) {
	transcript, err := t.local.TranscribeAudio(adata)
	if err == nil { return transcript, nil }
	return t.azure.TranscribeAudio(adata)
}
```

Výpis výše ukazuje definici rozhraní s metodami, jež by měly implementovat objekty pro přepis z Azure a Whisper modelu. Také je poté definován typ spojené služby *UnifiedTranscribeService*, která vytvoří instance služeb pro připojení k Azure i Whisper modelu. Následně funkce pro přepis této spojené služby nejdříve zavolá preferovaný lokálně nasazený Whisper model a v případě nedostupnosti využije Azure službu.

## 6.1 Integrace Azure Speech

Azure Speech má k dispozici vývojářské knihovny (SDK) pro implementaci do vlastní aplikace. V případě jazyku Go, ve kterém je backend napsán, má ale toto SDK jako závislost i některý kód jazyka C. To vyžaduje při sestavení buildu aplikace aktivaci CGO, která umožňuje volat funkce z C. Aplikace by již nebyla staticky kompilována a vyžadovalo by se využití jiného základního kontejneru než Alpine Linux pro sestavení backendu. Vzhledem k využití jen jedné funkce z celého SDK byla ručně implementována služba pro přepis audio souboru dle oficiální REST API dokumentace. [56]

Výpis 6.2 Implementace Azure Speech klienta do backendu

```
type AzureTranscribeService struct {
	resourceKey string
	endpoint    string
}

// https://learn.microsoft.com/en-us/azure/ai-services/speech-service/rest-speech-to-text-short
func NewAzureTranscribeService() *AzureTranscribeService {
	region := os.Getenv("AZURE_SPEECH_REGION")
	baseUrl, err := url.Parse(fmt.Sprintf("https://%s.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1", region))
	if err != nil {
		log.Println("Cannot parse Azure Base URL for speech", err)
	}

	params := url.Values{}
	params.Add("language", "cs-CZ")
	params.Add("format", "detailed")
	params.Add("cid", os.Getenv("AZURE_SPEECH_ENDPOINT_ID"))
	baseUrl.RawQuery = params.Encode()

	return &AzureTranscribeService{
		resourceKey: os.Getenv("AZURE_SPEECH_RESOURCE_KEY"),
		endpoint:    baseUrl.String(),
	}
}
```

```
func (t *AzureTranscribeService) TranscribeAudio(adata []byte) (Transcript, error) {
	body := &bytes.Buffer{}
	writer := multipart.NewWriter(body)
	part, err := writer.CreateFormFile("file", "audio.wav")
	if err != nil {
		return Transcript{}, err
	}
	if _, err = part.Write(adata); err != nil {
		return Transcript{}, err
	}
	if err := writer.Close(); err != nil {
		return Transcript{}, err
	}

	req, err := http.NewRequest("POST", t.endpoint, body)
	if err != nil {
		return Transcript{}, err
	}

	req.Header.Add("Ocp-Apim-Subscription-Key", t.resourceKey)
	req.Header.Add("Content-Type", "audio/wav; codecs=audio/pcm; samplerate=16000")
	req.Header.Add("Accept", "application/json")

	client := &http.Client{}
	resp, err := client.Do(req)
	if err != nil {
		return Transcript{}, err
	}
	defer resp.Body.Close()

	responseBody, err := io.ReadAll(resp.Body)
	if err != nil {
		return Transcript{}, err
	}
	log.Println(string(responseBody))
	type response struct {
		RecognitionStatus string `json:"RecognitionStatus"`
		Offset            int    `json:"Offset"`
		Duration          int    `json:"Duration"`
		DisplayText       string `json:"DisplayText"`
		NBest             []struct {
			Confidence float32 `json:"Confidence"`
			Lexical    string  `json:"Lexical"`
			ITN        string  `json:"ITN"`
```

```
                        MaskedITN  string  `json:"MaskedITN"`
                        Display    string  `json:"Display"`
                } `json:"NBest"`
        }
        var r response
        err = json.Unmarshal(responseBody, &r)
        if err != nil {
                return Transcript{}, err
        }
        return Transcript{
                Text:       r.NBest[0].Lexical,
                Confidence: r.NBest[0].Confidence,
        }, err
}
```

Výpis výše ukazuje implementaci Azure Speech klienta, který zároveň implementuje rozhraní TranscribeService. Funkce vezme data a zabalí je jako přílohu HTTP formuláře. (FormFile). Nastaví korektní URL parametry a HTTP hlavičky (cílový jazyk překladu, vzorkovací frekvence audia, očekávaný výstup) a požadavek pošle. Jako odpověď přichází JSON s přepisem a dalšími metadaty. Z něho funkce použije lexikální přepis (kterým Azure definuje přepis obsahující vše ve slovní podobě včetně čísel) a míru jistoty modelu s danou nahrávkou.

## 6.2 Integrace OpenAI Whisper

U Whisper modelu je implementace složitější ve dvou částech. Nejprve je nutné naučený Whisper model z minulé kapitoly nasadit na nějakém stroji a poté ho zpřístupnit přes API rozhraní k backendové aplikaci.

Jelikož Hugging Face obsahuje knihovny nejen na trénovaní, ale provoz modelu (inference) bylo implementováno jednoduché HTTP API rozhraní obalující tuto knihovnu. Tím bylo možné spustit model na jiném výkonnějším stroji, než na kterém je provozován backend. Případně lze později díky této architektuře model hostovat na některé cloudové službě s dostatečnými výpočetními prostředky.

### 6.2.1 Vytvoření Whisper API

Výpis 6.3 Vytvoření lokální REST API Whisper služby v Pythonu

```
@dataclass
class args:
    model_path = "./flightlog-model-v9"
    language = "czech"
    task = "transcribe"
```

```
processor = WhisperProcessor.from_pretrained(
    args.model_path, language=args.language, task=args.task
)

device = "cuda:0" if torch.cuda.is_available() else "cpu"
torch_dtype = torch.float16 if torch.cuda.is_available() else torch.float32

model = WhisperForConditionalGeneration.from_pretrained(
    args.model_path,
    torch_dtype=torch_dtype,
    use_safetensors=True,
)
model.generation_config.language = "cs"
model.generation_config.task = args.task
model.generation_config.forced_decoder_ids = None

model = model.to(device)  # CUDA

app = Flask(__name__)

@app.route("/", methods=["POST"])
def transcribe_audio():
    if "file" not in request.files:
        return jsonify({"error": "No file part"}), 400
    file = request.files.get("file")
    if file == None:
        return jsonify({"error": "No selected file"}), 400

    audio_file = io.BytesIO(file.stream.read())

    array, _ = sf.read(audio_file)
    input_features = processor.feature_extractor(
        array, return_tensors="pt", sampling_rate=16000
    ).input_features
    input_features = input_features.to(dtype=torch_dtype, device=device)  # CUDA
    generation_output = model.generate(
        input_features,
        output_scores=True,
        return_dict_in_generate=True,
    )
    predicted_ids = generation_output.sequences
    scores = generation_output.scores
    print(scores)
```

```
    # Compute average log probability
    avg_log_prob = np.mean([score.max(dim=-1).values.mean().item() for score in scores])
    confidence = 1 / (1 + np.exp(-avg_log_prob * 0.1))
    print(confidence)
    transcription = processor.tokenizer.batch_decode(
        predicted_ids, skip_special_tokens=True
    )
    response = {"Text": transcription[0], "Confidence": confidence}
    return jsonify(response)


if __name__ == "__main__":
    app.run(debug=True, port=5000, host="0.0.0.0")
```

Výpis kódu výše zachycuje celý jednosouborový Python skript pro vytvoření jednoduchého REST API endpointu. K tomu využívá knihovnu Python knihovnu Flask, která umožňuje rychlou tvorbu REST API. API očekává HTTP POST požadavek se souborem s názvem „file“. Ten poté načte jako pole bajtů rovnou do paměti. Knihovna soundfile poté z .wav souboru načte pole surových audio dat pomocí funkce sf.read(). Data se následně převedou na tokeny (input_features) a poté vloží do strojového modelu. Nezapomíná se na případné vložení dat do GPU místo CPU, pokud se používá akcelerace modelu přes grafickou jednotku.

Whisper model poté vrátí predikované tokeny a skóre. Celkové skóre přepsané nahrávky je poté vypočítáno jako maximum logaritmické pravděpodobnosti, což odpovídá logaritmické pravděpodobnosti nejpravděpodobnějšího tokenu v daném kroku generace. Průměr těchto maximálních hodnot přes všechny kroky generace je pak převeden do intervalu 0 až 1. Tím se získá velice hrubý odhad jistoty modelu, který ale stačí pro pozdější filtrování nahrávek s nižší mírou jistoty. Tokeny se zase převedou na text pomocí Whisper Tokenizeru.

### 6.2.2 Implementace služby do backendu

Výpis 6.4 Implementace klienta pro Whisper API do Go backendu

```
type LocalTranscribeService struct {
          endpoint string
}

func NewLocalTranscribeService() *LocalTranscribeService {
          endpoint := os.Getenv("LOCAL_SPEECH_ENDPOINT")
          endpoint = fmt.Sprintf("http://%s", endpoint)
          return &LocalTranscribeService{endpoint: endpoint}
}

func (t *LocalTranscribeService) TranscribeAudio(adata []byte) (Transcript, error) {
          // Prepare a form
```

```
	var requestBody bytes.Buffer
	multiPartW := multipart.NewWriter(&requestBody)

	// Add the file
	fileW, err := multiPartW.CreateFormFile("file", "transcribe.wav")
	if err != nil {
		return Transcript{}, fmt.Errorf("error creating form file: %w", err)
	}
	if _, err = fileW.Write(adata); err != nil {
		return Transcript{}, err
	}
	if err := multiPartW.Close(); err != nil {
		return Transcript{}, err
	}

	// Submit form to the handler
	req, err := http.NewRequest("POST", t.endpoint, &requestBody)
	if err != nil {
		return Transcript{}, fmt.Errorf("error creating request: %w", err)
	}
	req.Header.Set("Content-Type", multiPartW.FormDataContentType())
	req.Header.Add("Accept", "application/json; charset=utf-8")

	// Do the request
	client := &http.Client{}
	res, err := client.Do(req)
	if err != nil {
		return Transcript{}, fmt.Errorf("error sending request: %w", err)
	}
	defer req.Body.Close()

	if res.StatusCode != http.StatusOK {
		return Transcript{}, fmt.Errorf("server error: %s", res.Status)
	}

	responseBody, err := io.ReadAll(res.Body)
	log.Println(string(responseBody))
	if err != nil {
		return Transcript{}, fmt.Errorf("error reading response body: %w", err)
	}

	type response struct {
		Text       string  `json:"Text"`
		Confidence float32 `json:"Confidence"`
	}
```

```
        var r response
        err = json.Unmarshal(responseBody, &r)
        if err != nil {
                return Transcript{}, err
        }
        return Transcript(r), nil
}
```

Výpis výše ukazuje implementaci služby v backendové části aplikace volající hostované Whisper API. Pole bajtů zabalí do HTTP POST formulářového souboru. Poté nastaví HTTP hlavičky definující, že je součástí požadavku právě souborová příloha a že odpověď očekává ve formátu JSON. Po odeslání požadavku se pokusí odpověď parsovat a vrátit jako výstup funkce. Formát odpovědi je zde stejný jako v Python skriptu výše.

## 6.3 Extrakce metadat záměru pomocí LLM

Architektura řešení nyní umí nahrát a přepsat letištní komunikaci. Posledním krokem je pokus o extrakci metadat obsahující záměr pilota v nahrávce.

Jelikož se jedná o úlohu s problémem porozumění textu a extrakce informací, bylo jako řešení zvoleno využití velkých jazykových modelů LLM[15]. Díky vysoké schopnosti porozumění informací bez dodatečného trénování byl zvolen model GPT-4 od firmy OpenAI. [57]

Funkční požadavky pro implementaci:

- Jazykový model musí být schopen spolehlivě vrátit odpověď ve strojově zpracovatelném formátu. Například JSON
- Každá nahrávka zaslaná do modelu bude mít dodatečné informace vztahující se k danému letišti jako dráhy, směry a statické informace jako seznam záměrů, letových okruhů
- Cílem je, aby model vrátil datové schéma podobné jako navržený datový model v kapitole 4.2

OpenAI má oficiálně zdokumentované své publikované REST API [58]. Před implementací byla provedena rychlá rešerše na Googlu a Githubu pro knihovnu implementující toto REST API v jazyce Go. Byla nalezena knihovna *go-openai*, která byla využita pro rychlejší implementaci. [59]

[15] Large Language Model

Implementace probíhala nejdříve definováním rozhraní pro službu k extrakci záměru a datovým typem Záměr (Intent). Rozhraní implementuje jedinou metodu pro extrakci záměru. Jako parametry se vkládají informace o letišti a samotná nahrávka.

Výpis 6.5 Definice rozhraní pro službu záměru

```
type Intent struct {
	Callsign    string
	Intent      string
	Circuit     string
	CircuitSide string
	Runway      string
	Altitude    int
	Qnh         int
	Bearing     string
}

type IntentService interface {
	ExtractIntent(rec record.Record, airport record.Airport) (Intent, error)
}
```

```
type GPTIntentService struct {
	client *openai.Client
}

func NewGPTIntentService() *GPTIntentService {
	key := os.Getenv("OPENAI_API_KEY")
	client := openai.NewClient(key)
	return &GPTIntentService{
		client: client,
	}
}

func (g *GPTIntentService) ExtractIntent(rec record.Record, airport record.Airport)
(Intent, error) {
	resp, err := g.client.CreateChatCompletion(context.Background(),
		openai.ChatCompletionRequest{
			Model: openai.GPT4TurboPreview,
			Messages: []openai.ChatCompletionMessage{
				{
					Role:    openai.ChatMessageRoleUser,
					Content: g.generatePromt(rec, airport),
				},
			},
```

```
                        ResponseFormat: &openai.ChatCompletionResponseFormat{Type:
"json_object"},
                },
        )
        if err != nil {
                return Intent{}, err
        }
        var intent Intent
        if err = json.Unmarshal([]byte(resp.Choices[0].Message.Content), &intent); err !=
nil {
                return Intent{}, err
        }
        log.Println(intent)
        return intent, nil
}

func (g *GPTIntentService) generatePromt(rec record.Record, airport record.Airport) string
{
        transcription := strings.ReplaceAll(rec.Transcript.String,
strings.ToLower(airport.Callsign), "")

        taskIntro := "Extract from air traffic transcript. Output JSON data only."
        schema := `All Fields optional: callsign (e.g. OKA123), intent, circuit, circuit-
side, runway, qnh (int), bearing, altitude (int). Translate using provided maps.`

        // Suggestions
        // Because GPT returns intents in Czech, we NEED to use inverted maps
        intentsJSON, _ := json.Marshal(record.TextToIntentsCZ)
        circuitJSON, _ := json.Marshal(record.TextToCircuitCZ)
        runwaysJSON, _ := json.Marshal(airport.Runways)
        circuitSideJSON, _ := json.Marshal(record.TextToCircuitSideCZ)
        bearingsJSON, _ := json.Marshal(record.Bearings)

        mapsInstruction := fmt.Sprintf("Maps: intent=%s, circuit=%s, runway=%s, circuit-
side=%s, bearing=%s",
                string(intentsJSON), string(circuitJSON), string(runwaysJSON),
string(circuitSideJSON), string(bearingsJSON))

        prompt := fmt.Sprintf("%s\n%s\n%s\n%s", taskIntro, schema, transcription,
mapsInstruction)
        return prompt
}
```

Výpis výše ukazuje celý kód implementace GPT rozpoznání záměru. Nejdříve je vytvořena instance klienta. Metoda *ExtractIntent* zavolá GPT-4 API v režimu JSON odpovědi (model je tedy kontrolován, že vrací pouze JSON). Výsledek se pokusí namapovat do datového typu záměru (*Intent*).

Interní funkce generatePrompt poté obsahuje celou logiku pro vygenerování textového řetězce s příkazem do GPT modelu. První věta „Extract from air traffic transcript. Output JSON data only" dává instrukce pro extrakci informací, definuje, že se jedná o oblast leteckého provozu. Zároveň obsahuje příkaz pro výstup do JSON. Jedná se o povinnou věc, kterou doporučuje oficiální API dokumentace i v případě využití modelu s režimem JSON odpovědi. Bez tohoto příkazu model někdy nemusí JSON vrátit.

Poté je definováno textově JSON schéma odpovědi. Jednotlivé názvy JSON klíčů a případně u QNH a výšky, že chceme vrátit číselný datový typ. U volací značky je zároveň ukázán příklad, aby docházelo ke zkrácení.

Poté se sestaví mapa, která slouží pro jazykový model jako pomůcka. Obsahuje seznamy přípustných hodnot pro každý atribut v odpovědi. V první verzi byla vytvořena mapa pomocí již existujících číselníků v backendu. Ale existující číselníky měly mapování z datové hodnoty na daný jazykový překlad (používá se v UI frontendové části aplikace). Tedy například „*atz-exit:opuštění atz*". To ale vedlo model k výstupu překladu než samotného klíče. Mapy jsou tedy při použití invertovány a model vrací správný výstup.

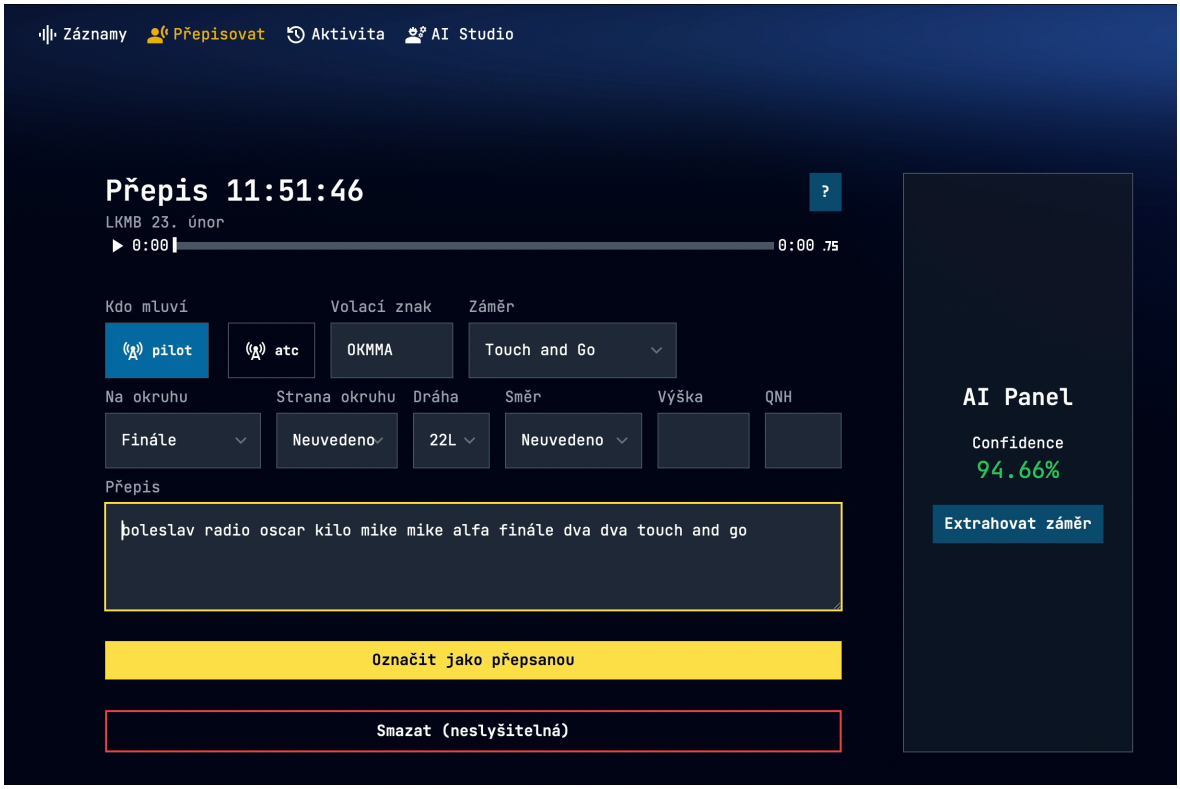
Výpis 6.6 Ukázka vygenerovaného promptu pro GPT model

```
Extract from air traffic transcript. Output JSON data only.

All Fields optional: callsign (e.g. OKA123), intent, circuit, circuit-side, runway, qnh
(int), bearing, altitude (int). Translate using provided maps.

india sedm čtyři po větru levého pro dráhu dva dva

Maps: intent={"opuštění atz":"atz-exit","průlet nad atz":"atz-flyby","průlet nad
letištěm":"airport-flyby","přistání":"landing","touch and go":"touch-and-go","vstup do
atz":"atz-entry","vzlet":"take-off"}, circuit={"druhá zatáčka":"second-
turn","finále":"final","po vzletu":"departure","po větru":"downwind","první
zatáčka":"first-turn","před poslední zatáčkou":"base","třetí zatáčka":"third-turn"},
runway=["04L","22R","04R","22L","16L","34R","16R","34L","22","16","04","34"], circuit-
side={"levého":"left","pravého":"right"}, bearing=["north","north-east","east","south-
east","south","south-west","west","north-west"]
```

Ve výpisu výše je ukázka vygenerovaného promptu pro jednu nahrávku. Pro lepší čitelnost byly vytvořeny mezery. Ovšem v reálném provozu je pro minimalizaci tokenů (a snížení ceny) vše minifikováno.

## 6.4 UI implementace přepisu

Implementované služby byly zpřístupněny přes webové UI serverové aplikace v obrazovce pro přepis nahrávky. V případě, že je již nahrávka přepsána člověkem, se možnost AI přepisu a extrakce záměru nezobrazí. Uživatel má možnost přepis opravit a poté uložit. Uložením se nahrávka označí jako zpracovaná člověkem a může být použitá na další iterace trénování modelu.




Obrázek 32 Ukázka UI rozhraní pro přepis a záměr pomocí strojového modelu

# Závěr

Cílem diplomové práce bylo vytvořit model strojového učení, který dokáže spolehlivě přepsat českou letištní komunikaci a poté z ní případně extrahovat důležité informace v podobě štítků reprezentujících záměr pilota. Protože nebyl k dispozici žádný veřejně dostupný dataset této české komunikace v dostatečné kvalitě pro trénování modelu strojového učení, musela být do práce zahrnuta i příprava vlastního datasetu. Tedy nahrávání, ruční přepis, štítkování a samotné porozumění letecké komunikaci.

Ve výsledku tato diplomová práce poskytuje kompletní návrh a ukázkovou implementaci rozsáhlé architektury včetně hardwaru i softwaru pro zcela automatické nahrávání, sběr a zpracování komunikace v letecké oblasti. Návrh architektury je škálovatelný a je možné ji implementovat napříč sítí českých letišť. V průběhu práce byla produkční verze nasazena v letové zóně Mladé Boleslavi (LKMB).

Následně diplomová práce obsahuje dostatečně detailní popis frazeologie a struktury letecké komunikace pilotů v podobě použitelného datového modelu, který se hodí na ruční i automatické štítkování sebraných nahrávek. Tento popis byl konzultován i s profesionálními piloty na validaci teoretických předpisů a reálného leteckého provozu.

Poté jsou v práci natrénovány 2 modely strojového učení. Jeden ze zástupce proprietárních cloudových poskytovatelů (Microsoft Azure) a druhý svobodně dostupný Whisper Large v3 od firmy OpenAI, aktuálně nejlepší vícejazyčný model dle benchmarků provedených společností Hugging Face. Během implementace trénování diplomová práce přispěla i do projektu Hugging Face s opravou učících skriptů pro Whisper ve vícejazyčném režimu.

Výsledkem je reprodukovatelný model převodu české letecké řeči na text, který místo standardní slovní chybovosti WER okolo 50 % snižuje chybovost na 9 % na stejném validačním datasetu. To potvrzuje hypotézu v úvodu práce, že aktuální vyspělost generických modelů pro převod řeči na text je již natolik vysoká, že na zpřesnění pro konkrétní doménovou oblast stačí již menší audio dataset v řádu hodin.

Součástí je i ukázková implementace propojení modelu s již předem vytvořenou architekturu pro sběr a zpracování nahrávek. Tím je vytvořen systém, který může automaticky nebo na vyžádání přepisovat probíhající letecký provoz v téměř reálném čase.

Práce může být nadále rozšířena a optimalizována. Jednak může být architektura přizpůsobena pro zcela reálný přepis komunikace se zpožděním do jednotek sekund. Model přepisu řeči na text může být znovu natrénován na řádově větším datasetu a porovnán, zda vede k nějakému měřitelnému zlepšení. Zároveň může být vytvořen customizovaný model pro extrakci štítků z přepsané komunikace, který může nabídnout ještě vyšší míru přesnosti.

# Použitá literatura


[1] ASSOCIATION FOR COMPUTING MACHINERY, INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, ASSOCIATION FOR COMPUTING MACHINERY, IEEE COMPUTER SOCIETY, a CARL VON OSSIETZKY UNIVERSITÄT OLDENBURG, ed. *Demonstration abstract: OpenSky: a large-scale ADS-B sensor network for research*. Piscataway, NJ: IEEE, 2014. ISBN 978-1-4799-3146-0.

[2] ŘÍZENÍ LETOVÉHO PROVOZU ČESKÉ REPUBLIKY, S. P. *VFR příručka - Česká republika* [online]. [vid. 2024-04-03]. Dostupné z: https://aim.rlp.cz/vfrmanual/actual/gen_1_cz.html

[3] ŘÍZENÍ LETOVÉHO PROVOZU ČESKÉ REPUBLIKY, S. P. *Všeobecná pravidla a postupy* [online]. Dostupné z: https://aim.rlp.cz/ais_data/aip/data/valid/e1-1.pdf

[4] ŘÍZENÍ LETOVÉHO PROVOZU ČESKÉ REPUBLIKY, S. P. Letecká frazeologie a terminologie pro poskytování letových provozních služeb a provádění letů. nedatováno.

[5] ŘÍZENÍ LETOVÉHO PROVOZU ČESKÉ REPUBLIKY, S. P. *Letecký předpis letové provozní služby* [online]. Dostupné z: https://aim.rlp.cz/predpisy/predpisy/dokumenty/L/L-11/data/print/L11_cely.pdf

[6] FACTOR, M., K. METH, D. NAOR, O. RODEH a J. SATRAN. Object Storage: The Future Building Block for Storage Systems A Position Paper. In: *2005 IEEE International Symposium on Mass Storage Systems and Technology*: *2005 IEEE International Symposium on Mass Storage Systems and Technology* [online]. Sardinia, Italy: IEEE, 2005, s. 119–123 [vid. 2024-04-03]. ISBN 978-0-7803-9228-1. Dostupné z: doi:10.1109/LGDI.2005.1612479

[7] BRUCKNER, Tomáš, VOŘÍŠEK, JIŘÍ, a BUCHALCEVOVÁ, ALENA. *Tvorba informačních systémů: principy, metodiky, architektury*. Praha: Grada, 2012. ISBN 978-80-247-4153-6.

[8] FEITOSA, Antonio, Paulyne JUCÁ, Márcio MAIA, Miguel CASTRO a Arthur CALLADO. Low-cost ADS-B Collectors for Distributed Aircraft Surveillance. In: *EATIS '18: Euro American Conference on Telematics and Information Systems*: *Proceedings of the Euro American Conference on Telematics and Information Systems* [online]. Fortaleza Brazil: ACM, 2018, s. 1–8 [vid. 2024-04-03]. ISBN 978-1-4503-6572-7. Dostupné z: doi:10.1145/3293614.3293622

[9] ŽMOLÍKOVÁ, Kateřina. *Rozpoznávání řeči pro leteckou komunikaci* [online]. B.m., 2016 [vid. 2024-04-03]. Bakalářská práce. Vysoké učení technické v Brně. Dostupné z: https://theses.cz/id/hlokak/

[10] Unified Modeling Language, v2.5.1. *Unified Modeling Language*. nedatováno.

[11] CHAPMAN, Pete. *CRISP-DM 1.0: Step-by-step Data Mining Guide* [online]. B.m.: SPSS, 2000. Dostupné z: https://www.ibm.com/docs/it/SS3RA7_18.3.0/pdf/ModelerCRISPDM.pdf

[12] *M505 : Metodika CRISP–DM* [online]. [vid. 2024-04-04]. Dostupné z: https://mbi.vse.cz/public/cs/obj/METHOD-113

[13] ING. REJNKOVÁ, Petra. *Metodika MMSP* [online]. [vid. 2024-04-04]. Dostupné z: https://spicenter.vse.cz/wp-content/uploads/metodiky/publikovana_MMSP/index.htm

[14] INTERNATIONAL CIVIL AVIATION ORGANIZATION. *Handbook on Radio Frequency Spectrum Requirements for Civil Aviation* [online]. 2021. Dostupné z: https://www.icao.int/safety/FSMP/Documents/Doc9718/Doc.9718%20Vol.%20I%20(AdvanceUneditedVersion%202021).pdf

[15] ŘÍZENÍ LETOVÉHO PROVOZU ČESKÉ REPUBLIKY, S. P. *Provozní kmitočty - VFR příručka* [online]. [vid. 2024-04-05]. Dostupné z: https://aim.rlp.cz/vfrmanual/actual/gen_3_cz.html

[16] *Amatérská radiokomunikační služba | Český telekomunikační úřad* [online]. [vid. 2024-04-05]. Dostupné z: https://ctu.gov.cz/amaterska-radiokomunikacni-sluzba

[17] SILVER, H. Ward. *Ham Radio For Dummies*. 1st edition. B.m.: For Dummies, 2004. ISBN 978-0-7645-5987-7.

[18] AEROWEB.CZ. *Třetí vlna přeladění na 8,33 kHz již za měsíc - Aeroweb.cz* [online]. [vid. 2024-04-05]. Dostupné z: https://www.aeroweb.cz/clanky/5546-treti-vlna-preladeni-na-8-33-khz-jiz-za-mesic

[19] *8.33 kHz implementation support | EUROCONTROL* [online]. 1. červenec 2017 [vid. 2024-04-05]. Dostupné z: https://www.eurocontrol.int/function/833-khz-implementation-support

[20] *LKPR - Praha, VFR příručka - Česká republika* [online]. [vid. 2024-04-05]. Dostupné z: https://aim.rlp.cz/vfrmanual/actual/lkpr_text_cz.html

[21] TŮMA, Tomáš. *Návrh a realizace VKV přijímacího systému na bázi SDR* [online]. B.m., 2019. Bakalářská práce. b.n. Dostupné z: https://dspace.cvut.cz/handle/10467/85366

[22] *ABF128 | ACCESSORIES | AOR,LTD. Authority On Radio Communications* [online]. [vid. 2024-04-06]. Dostupné z: http://www.aorja.com/accessories/abf128.html

[23] INC, ARRL. *ARRL Ham Radio License Manual 5th Edition – Complete Study Guide with Question Pool to Pass the Technician Class Amateur Radio Exam*. B.m.: ARRL The National Association for Amateur Radio®, 2022. ISBN 978-1-62595-155-7.

[24] INTERNET SOCIETY. *The Internet of Things (IoT): An Overview* [online]. 2015 [vid. 2020-11-01]. Dostupné z: https://www.internetsociety.org/wp-content/uploads/2017/08/ISOC-IoT-Overview-20151221-en.pdf

[25] THE LINUX FOUNDATION. *Open Container Initiative* [online]. 2020 [vid. 2020-11-14]. Dostupné z: https://opencontainers.org/

[26] SHANNON, C. E. A mathematical theory of communication. *The Bell System Technical Journal* [online]. 1948, **27**(3), 379–423. ISSN 0005-8580. Dostupné z: doi:10.1002/j.1538-7305.1948.tb01338.x

[27] KANG, Tianqu, Anh-Dung DINH, Binghong WANG, Tianyuan DU, Yijia CHEN a Kevin CHAU. *Optimization of a Real-Time Wavelet-Based Algorithm for Improving Speech Intelligibility*. 2022.

[28] Introduction to audio encoding for Speech-to-Text | Cloud Speech-to-Text Documentation. *Google Cloud* [online]. [vid. 2024-04-06]. Dostupné z: https://cloud.google.com/speech-to-text/docs/encoding

[29] CHARLIE-FOXTROT. *charlie-foxtrot/RTLSDR-Airband* [online]. C++. 6. duben 2024 [vid. 2024-04-06]. Dostupné z: https://github.com/charlie-foxtrot/RTLSDR-Airband

[30] writerseeker/writerseeker.go at master · orcaman/writerseeker. *GitHub* [online]. [vid. 2024-04-07]. Dostupné z: https://github.com/orcaman/writerseeker/blob/master/writerseeker.go

[31] *Azure Database for PostgreSQL | Microsoft Azure* [online]. [vid. 2024-04-07]. Dostupné z: https://azure.microsoft.com/en-us/products/postgresql

[32] *Kryštof 01* [online]. 2023 [vid. 2024-04-13]. Dostupné z: https://cs.wikipedia.org/w/index.php?title=Kry%C5%A1tof_01&oldid=22295244

[33] Rendering on the Web | Articles. *web.dev* [online]. [vid. 2024-04-18]. Dostupné z: https://web.dev/articles/rendering-on-the-web

[34] *Components | templ docs* [online]. [vid. 2024-04-18]. Dostupné z: https://templ.guide/core-concepts/components

[35] *</> htmx ~ Reference* [online]. [vid. 2024-04-18]. Dostupné z: https://htmx.org/reference/

[36] *</> htmx - high power tools for html* [online]. [vid. 2024-04-16]. Dostupné z: https://htmx.org/

[37] *Web Components - Web APIs | MDN* [online]. 9. únor 2024 [vid. 2024-04-18]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Web_components

[38] Lit. *lit.dev* [online]. [vid. 2024-04-18]. Dostupné z: https://lit.dev/

[39] LAKSHMANAN, Valliappa, Sara ROBINSON a Michael MUNN. *Machine learning design patterns: solutions to common challenges in data preparation, model building, and MLOps*. First edition. Beijing Boston Farnham: O'Reilly, 2020. ISBN 978-1-09-811578-4.

[40] *Gartner Reprint* [online]. [vid. 2024-04-19]. Dostupné z: https://www.gartner.com/doc/reprints?id=1-2ES4ML14&ct=230823&st=sb

[41] *Supported languages and language-specific features - Amazon Transcribe* [online]. [vid. 2024-04-19]. Dostupné z: https://docs.aws.amazon.com/transcribe/latest/dg/supported-languages.html

[42] Improve transcription results with model adaptation | Cloud Speech-to-Text V2 documentation. *Google Cloud* [online]. [vid. 2024-04-19]. Dostupné z: https://cloud.google.com/speech-to-text/v2/docs/adaptation-model

[43] ERIC-URBAN. *Language support - Speech service - Azure AI services* [online]. 6. únor 2024 [vid. 2024-04-19]. Dostupné z: https://learn.microsoft.com/en-us/azure/ai-services/speech-service/language-support

[44] *Terms of Service – Hugging Face* [online]. [vid. 2024-04-19]. Dostupné z: https://huggingface.co/terms-of-service

[45] *Open ASR Leaderboard - a Hugging Face Space by hf-audio* [online]. [vid. 2024-04-18]. Dostupné z: https://huggingface.co/spaces/hf-audio/open_asr_leaderboard

[46] *mozilla-foundation/common_voice_16_1 · Datasets at Hugging Face* [online]. [vid. 2024-04-20]. Dostupné z: https://huggingface.co/datasets/mozilla-foundation/common_voice_16_1

[47] *openai/whisper* [online]. Python. B.m.: OpenAI. 20. duben 2024 [vid. 2024-04-20]. Dostupné z: https://github.com/openai/whisper

[48] PARK, Daniel S., William CHAN, Yu ZHANG, Chung-Cheng CHIU, Barret ZOPH, Ekin D. CUBUK a Quoc V. LE. SpecAugment: A Simple Data Augmentation Method for Automatic Speech Recognition. In: *Interspeech 2019* [online]. 2019, s. 2613–2617 [vid. 2024-04-21]. Dostupné z: doi:10.21437/Interspeech.2019-2680

[49] *openai/whisper · How to fine tune the model* [online]. 24. září 2022 [vid. 2024-04-26]. Dostupné z: https://huggingface.co/spaces/openai/whisper/discussions/6

[50] LIAO, Feng-Ting, Yung-Chieh CHAN, Yi-Chang CHEN, Chan-Jan HSU a Da-shan SHIU. *Zero-shot Domain-sensitive Speech Recognition with Prompt-conditioning Fine-tuning* [online]. B.m.: arXiv. 5. říjen 2023 [vid. 2024-04-26]. Dostupné z: http://arxiv.org/abs/2307.10274. arXiv:2307.10274 [cs, eess]

[51] CHOLLET, François. *Deep learning v jazyku Python: knihovny Keras, Tensorflow*. B.m.: Grada Publishing, a.s., 2019. ISBN 978-80-247-3100-1.

[52] *Discrepancies between whisper training script and notebook · Issue #29973 · huggingface/transformers* [online]. [vid. 2024-04-20]. Dostupné z: https://github.com/huggingface/transformers/issues/29973

[53] PyTorch | NVIDIA NGC. *NVIDIA NGC Catalog* [online]. [vid. 2024-04-21]. Dostupné z: https://catalog.ngc.nvidia.com/orgs/nvidia/containers/pytorch

[54] *Installing the NVIDIA Container Toolkit — NVIDIA Container Toolkit 1.14.5 documentation* [online]. [vid. 2024-04-21]. Dostupné z: https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/latest/install-guide.html#installation

[55] Latest News. *DeepSpeed* [online]. [vid. 2024-04-21]. Dostupné z: https://www.deepspeed.ai/

[56] ERIC-URBAN. *Speech to text REST API for short audio - Speech service - Azure AI services* [online]. 22. leden 2024 [vid. 2024-04-27]. Dostupné z: https://learn.microsoft.com/en-us/azure/ai-services/speech-service/rest-speech-to-text-short

[57] OPENAI. *GPT-4 Technical Report* [online]. B.m.: arXiv. 4. březen 2024 [vid. 2024-04-27]. Dostupné z: https://arxiv.org/abs/2303.08774

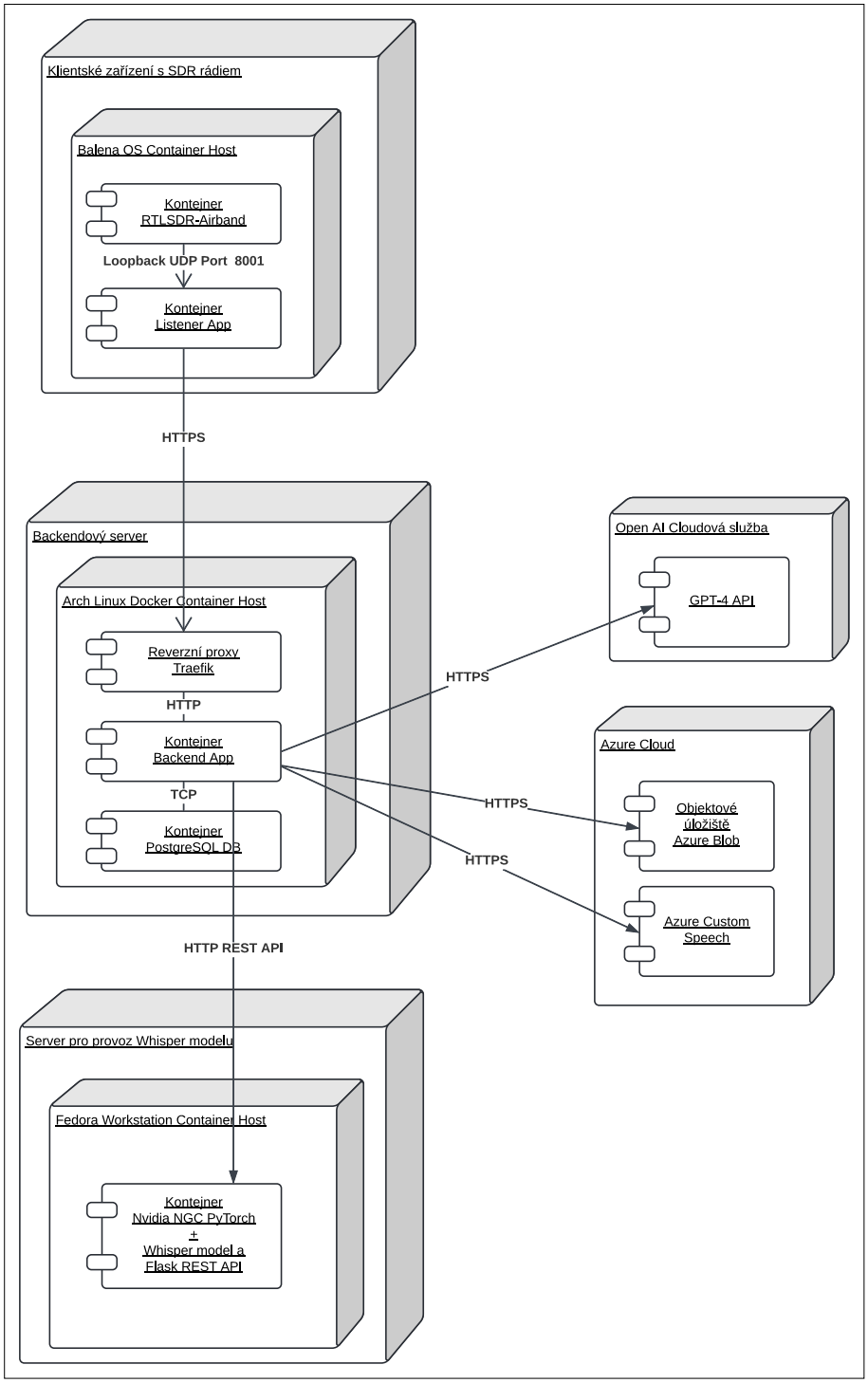
[58] *OpenAI Platform* [online]. [vid. 2024-04-27]. Dostupné z: https://platform.openai.com

[59] BARANOV, Alexander. *sashabaranov/go-openai* [online]. Go. 27. duben 2024 [vid. 2024-04-27]. Dostupné z: https://github.com/sashabaranov/go-openai

# Přílohy

Dokumentační artefakty byly postupně přikládané na relevantních místech v práci. Vzhledem k velikosti projektu jsou zbylé přílohy se zdrojovými kódy vloženy jako elektronická příloha.

# Příloha A: Diagram nasazení finální architektury




Obrázek 33 UML Diagram nasazení celé architektury

UML diagram nasazení zobrazuje finální stav nasazené produkční architektury řešení.