Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

# Bakalářská práce

# Software pro mobilního klienta a pro operátorskou konzoli systému PocketEAR

Plzeň 2018 František Pártl

**Místo této strany bude zadání práce.**

# Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 1. května 2018

František Pártl

Rád bych poděkoval Ing. Kamilu Ekšteinovi, Ph.D. za cenné rady, věcné připomínky a vstřícnost při konzultacích a vypracování bakalářské práce.

# Abstract

This bachelor thesis deals with the development of the PocketEAR system which consists of two parts. These are the client application used to collect and parameterize general acoustic signals using the MFCC method and the web operator interface which allows system operators to classify unrecognized signals for further learning of the background recognizer. This recognizer is not subject of this work.



# Abstrakt

Tato bakalářská práce se zabývá vývojem systému PocketEAR, který sestává ze dvou částí. Těmi jsou klientská aplikace sloužící pro sběr a parametrizaci obecných akustických signálů užitím metody mel-frekvenčních kepstrálních koeficientů (MFCC) a webového operátorského rozhraní, které umožňuje operátorům systému klasifikovat nerozpoznané signály za účelem dalšího učení rozpoznávače na pozadí. Zmíněný rozpoznávač není předmětem této práce.

# Obsah

# 1 Úvod

V současné době probíhá na Katedře informatiky a výpočetní techniky Fakulty aplikovaných věd ZČU v Plzni vývoj rozpoznávače, který klasifikuje akustické signály na základě analýzy mel-frekvenčních kepstrálních koeficientů. Pro kompletní využití této inteligence je třeba vytvořit infrastrukturu, kterou bude možné vyměňovat data mezi rozpoznávačem a mobilní aplikací koncového uživatele. Další plánovanou částí této infrastruktury je operátorské stanoviště, které přijímá audio záznamy, jejichž obsah nebylo možné rozpoznávačem stanovit. Posádkou tohoto stanoviště je skupina operátorů, kteří určí obsah zmíněného nerozpoznaného zvuku, čímž se rozpoznávač bude dále učit. Tento celistvý systém byl označen jako *PocketEAR* (česky „kapesní ucho“).

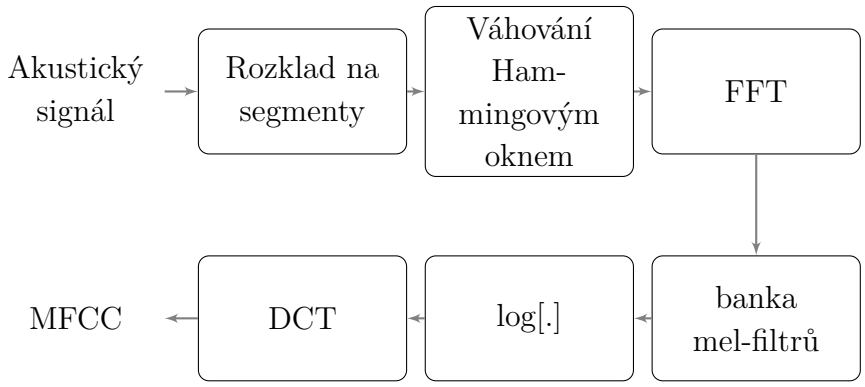
Úvahami o operátorské konzoli vznikají nové otázky týkající se možností distribuce objemných zdrojových souborů jednotlivých audio záznamů. Při návrhu řešení této úlohy se objevila ambiciózní myšlenka rekonstrukce originálních audio nahrávek z daných mel-frekvenčních kepstrálních koeficientů.

Funkční systém, přesněji mobilní aplikace, by pak mohla napomáhat například sluchově postiženým jedincům při orientaci v blízkém okolí. Dále by mohla varovat před hrozícím nebezpečím, tj. například projíždějícím automobilem, sirénou, houkáním, pískáním a podobně.

# 2 Mel-frekvenční kepstrální koeficienty

MFCC je metoda parametrizace akustického signálu, která patří do oblasti homoformní analýzy, jež je ze skupiny postupů nelineárního zpracování signálů využívajících principu superpozice. MFCC využívá zpracování akustického signálu jak v čase, tak i ve frekvenční oblasti. Popisuje přitom spektrální vlastnosti zmíněného signálu, konkrétně odhad výkonové spektrální hustoty. Tato analýza je postavena na tzv. melovské stupnici, která svým logaritmickým průběhem napodobuje citlivost lidského ucha při vnímání vlnění různých frekvencí.[29]

Schéma 2.1 ukazuje postup výpočtu MFCC. Významu jednotlivých bloků se věnují následující podkapitoly.

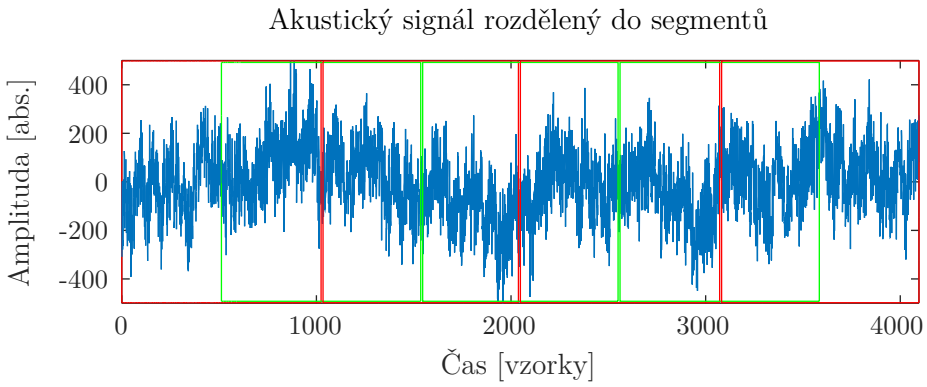



Obrázek 2.1: Schéma výpočtu MFCC.

## 2.1 Rozklad signálu na segmenty

Akustický signál je v čase velice proměnlivá veličina, čili pro usnadnění jeho analýzy je rozdělen do menších celků, kde je již kvazistacionární a jeho spektrální analýza má smysl. V metodě MFCC je signál rozložen do krátkých segmentů. Pro tyto segmenty probíhá pozdější výpočet mel-frekvenčních koeficientů odděleně. Platí tedy, že pro každý segment je vypočten unikátní vektor koeficientů.

Segmenty mají délku obecně od 10 do 50 ms. Optimální hodnota je závislá na typu rozpoznávaného zvuku, tj. jiná délka bude volena pro rozpoznávání




Obrázek 2.2: Akustický signál rozdělený do segmentů.

řeči a jiná pro určení druhu hudebního nástroje.

Obrázek 2.2 ilustruje 4096 vzorků akustického signálu, který je rozdělen do segmentů o velikosti 1024 vzorků. Hodnota překryvu jednotlivých oken je 50 %.

## 2.2 Váhování Hammingovým oknem

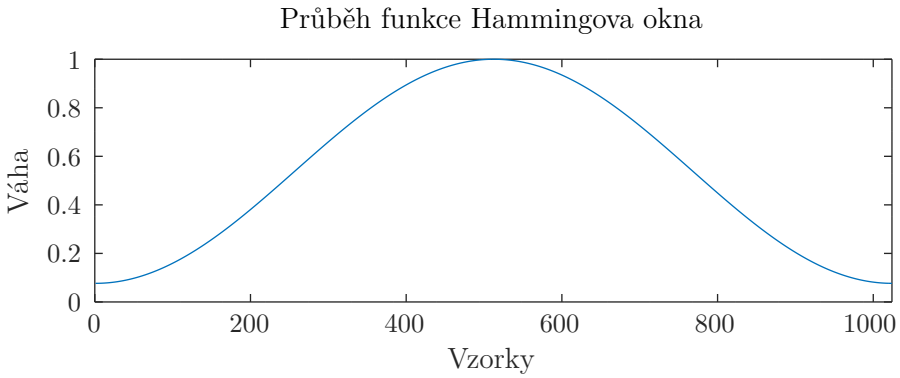
Součástí MFCC je výpočet frekvenčního spektra krátkého segmentu akustického signálu. Po rozdělení signálu do segmentů tyto segmenty nemají charakter akustického signálu, protože jejich střední hodnota není blízká nule a na krajích mohou vznikat skoky (kolem segmentu uvažujeme nulovou energii). Při aplikaci Fourierovy transformace by se tak mohly projevit energie ve vysokých frekvencích, které analyzovaný zvuk ve skutečnosti neobsahuje.

Hammingovo okno je jedním z mnoha přístupů, jak potlačit vzorky na krajích segmentu a přitom nijak neovlivnit hladkost signálu. Okno je definováno vztahem

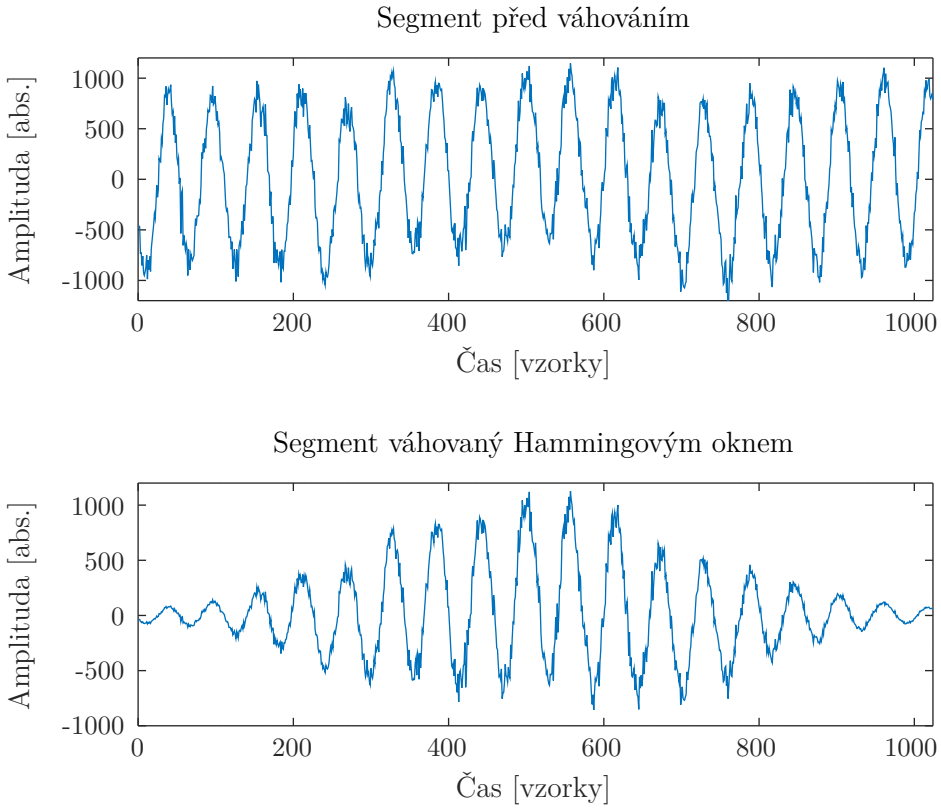
$$\omega(n) = 0.54 - 0.46 \cos\left(\frac{2\pi n}{N-1}\right),$$

kde $N$ je počet vzorků okna.

Samotné váhování probíhá jako jednoduchá lineární kombinace segmentu vstupního signálu a Hammingova okna. Průběh Hammingova okna je zobrazen na obrázku 2.3, váhování vstupního segmentu pak popisuje obrázek 2.4.




Obrázek 2.3: Průběh funkce Hammingova okna.




Obrázek 2.4: Průběh váhování segmentu signálu.

## 2.3 Diskrétní Fourierova transformace

Z váhovaného segmentu akustického signálu je potřeba vypočítat odhad jeho výkonové spektrální hustoty. K tomu slouží DFT (Discrete Fourier Transform), která je odvozena od Fourierových řad, jejichž myšlenkou je, že jakýkoli průběh signálu se dá rozdělit na nekonečnou řadu harmonických (sinusových) průběhů o různé frekvenci, fázi a amplitudě [29].

Předpis diskrétní Fourierovy transformace popisuje vztah 2.1.

$$X[k] = \sum_{n=0}^{N-1} x(n) e^{\frac{-j2\pi nk}{N}}, k = 0, 1, ..., N-1 \tag{2.1}$$

Výsledkem DFT je obecně vektor komplexních čísel nesoucích informaci o energii a fázi, ze kterého je třeba určit úrovně energetického zastoupení jednotlivých frekvenčních pásem, tj. stanovit velikosti komplexních čísel. Tento krok se provede užitím Euklidovské normy definované vztahem 2.2.

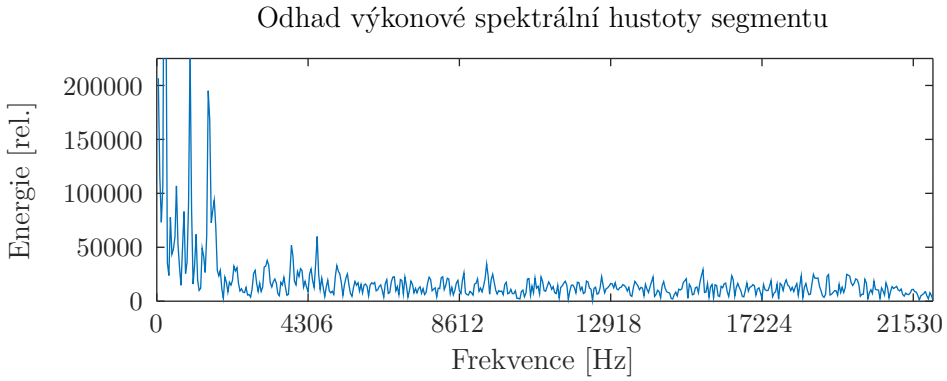
$$||X[k]|| = \sqrt{Re\{X[k]\}^2 + Im\{X[k]\}^2}, k = 0, 1, ..., N-1 \tag{2.2}$$

Výsledkem této operace je konečný odhad výkonové spektrální hustoty.

Pro výpočet všech $N-1$ koeficientů potřebuje DFT provést $N^2$ sčítání a také $N^2$ násobení komplexních čísel. Kvůli této vysoké výpočetní náročnosti vytvořili Cooley a Tukey algoritmus FFT (Fast Fourier Transform). Jedná se – zjednodušeně řečeno – o urychlení diskrétní Fourierovy transformace, využívající periodicity bázových funkcí, díky čemuž lze koeficienty získané v průběhu výpočtu využít v pozdějších fázích výpočtu, aniž by se musely opětovně vypočítávat. Kritériem použití tohoto algoritmu je nutnost omezit délku vstupního segmentu $N = 2^m$, kde $m \in \mathbb{N}$. Toto omezení lze vyřešit doplněním vstupního vektoru nulami. Užitím metody FFT je výpočetní složitost snížena na $\frac{mN}{2}$ komplexních násobení a $mN$ sčítání [32]. Výsledný vektor pak obsahuje teoreticky $N$ komplexních čísel. Pro výpočet FFT byla ale využita knihovna *KissFFT*, která vrací vektory o velikost $\frac{N}{2} + 1$.

## 2.4 Parametrizace melovskou bankou filtrů

Melovská banka filtrů se využívá k parametrizaci odhadu výkonové spektrální hustoty segmentu akustického signálu. Tato banka využívá podobnosti se zpracováním zvuku lidským sluchem. Frekvenční rozlišovací schopnost sluchu nelineárně klesá s rostoucí frekvencí, proto je melovská banka navržena tak, aby simulovala vnímání signálu lidským sluchem deformací frekvenční




Obrázek 2.5: Odhad výkonové spektrální hustoty segmentu.

osy [28]. Všechny trojúhelníkové filtry se z 50 % překrývají, tj. konec trojúhelníku je v bodě, kde má další trojúhelník maximum.

Výslednou množinu filtrů ilustruje obrázek 2.6. Tato banka byla vytvořena pomocí následujícího postupu:

1. Frekvenční spektrum zaznamenávaného akustického signálu je převedeno do melovské stupnice pomocí vztahu 2.3.

$$Mel(f) = 2595 \log_{10}\left(1 + \frac{f}{700}\right) \tag{2.3}$$

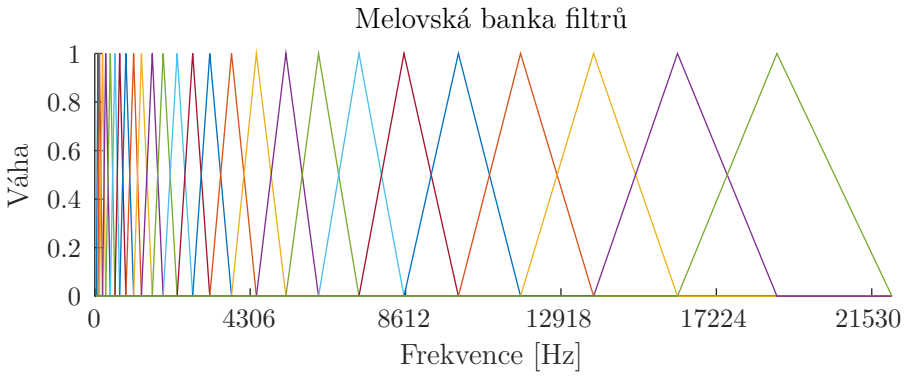
2. V takto vytvořené melovské stupnici je rovnoměrně rozprostřeno $M+2$ bodů, kde $M$ označuje požadovaný počet filtrů. Je třeba přičíst dva další body, které reprezentují začátek prvního trojúhelníku a konec posledního.

3. Vytvořené body se následně převedou zpět do lineárního frekvenčního spektra dosazením do vztahu 2.4.

$$f(m) = 700(10^{\frac{m}{2595}} - 1) \tag{2.4}$$

4. Nad definovanými body se vytvoří trojúhelníkové filtry pomocí předpisu 2.5.

$$H_m(k) = \begin{cases} 0 & k < f(m-1) \\ \dfrac{k - f(m-1)}{f(m) - f(m-1)} & f(m-1) \leq k < f(m) \\ 1 & k = f(m) \\ \dfrac{f(m+1) - k}{f(m+1) - f(m)} & f(m) < k \leq f(m+1) \\ 0 & k > f(m-1) \end{cases} \tag{2.5}$$




Obrázek 2.6: Banka melovských filtrů.

Následná parametrizace probíhá jako skalární součin vektoru váhových koeficientů a jednotlivých filtrů (viz 2.6). Při užití $M$ filtrů tedy vzniká $M$ kladných reálných hodnot, které nazýváme melovskými koeficienty. Tyto koeficienty jsou ale vzájemně korelované, a tak pro rozpoznávání nejsou příliš vhodné a častěji se převádějí na kepstrální koeficienty [28].

$$S_{mel}[m] = \sum_{k=0}^{N-1} |X[k]H_m(k)| \tag{2.6}$$

## 2.5 Převod na melovské kepstrální koeficienty

Předpokládá se, že parametrizovaný signál je vytvořen konvolucí užitečného a šumového signálu, který je třeba odstranit. Na melovské koeficienty je tedy nejdříve aplikován přirozený logaritmus, který signál převede na aditivní. Ten je následně možné odečíst. Logaritmus je následován inverzní Fourierovou transformací. Protože jsou ale na vstupu této transformace jenom nezáporné reálné hodnoty, je možné Frourierovu transformaci nahradit výpočetně méně náročnější diskrétní kosinovou transformací (DCT – *Discrete Cosine Transformation*) [28]. Převod melovských koeficientů na kepstrální definuje vztah 2.7.

$$cc[m] = \beta_L(m) \sum_{l=1}^{L} \log\left(S_{mel}[l]\right) \cos\left[\frac{m\pi}{L}\left(l - \frac{1}{2}\right)\right], \tag{2.7}$$

kde $L$ je počet použitých filtrů, $S_{mel}[l]$ je l-tý melovský koeficient a $\beta_L(m)$ označuje normalizační faktor definovaný vztahem 2.8.

$$\beta_L(m) = \begin{cases} \sqrt{\frac{1}{L}} & m = 0 \\ \sqrt{\frac{2}{L}} & m > 0 \end{cases} \tag{2.8}$$

## 2.6 Rekonstrukce zvuku z MFCC

Tato myšlenka je z pohledu celého systému PocketEAR velice zajímavá, jelikož možnost rekonstrukce akustického signálu reprezentovaného vektorem jeho mel-frekvenčních kepstrálních koeficietů by výrazně snížila objem komunikace mezi mobilní aplikací a rozpoznávačem.

Problémem rekonstrukce je fakt, že během parametrizace akustického signálu došlo k několika následujícím ztrátovým operacím:

1. Výsledek DFT, tj. vektor komplexních váhových koeficientů, byl pomocí Euklidovské normy převeden na vektor velikostí těchto komplexních koeficietů. To způsobilo ztrátu informace o fázi rekonstruovaného akustického signálu.

2. Odhad výkonové spektrální hustoty je obecně parametrizován například pomocí $N$ kepstrálních koeficientů. V případě rozpoznávání zvuku je ale dostačující menší počet než $N$. Některé koeficiety jsou tedy zahozeny.

## 2.7 Rekonstrukce váh filtrů melovské banky

Jedná se o zpětný převod kepstrálních koeficientů na jejich melovské ekvivalenty. Problémem této rekonstrukce je již zmíněný fakt, že některé z kepstrálních koeficientů byly zahozeny. Cílem tohoto kroku je tedy obnovit vektor $S_{mel}$ z vektoru $\boldsymbol{cc}$.

Ze vztahu 2.7 je patrné, že nejdříve je třeba kepstrální koeficienty zpětně vydělit normalizačním faktorem (viz 2.8):

$$Y[m] = \frac{cc[m]}{\beta_L(m)}, \tag{2.9}$$

kde $m = 0, ..., K-1$ ($K$ označuje počet dostupných kepstrálních koeficientů). Zahozené koeficienty se nahradí hodnotou 0 až na velikost vektoru L,

tj. počet použitých trojúhelníkových filtrů.

Dalším krokem je inverze diskrétní kosinové transformace definované následujícím vztahem:

$$Z[m] = \frac{2}{L}\left(\frac{1}{2}Y[0] + \sum_{l=1}^{L-1} Y[l]\cos\left[\frac{\pi l}{L}\left(m+\frac{1}{2}\right)\right]\right), \tag{2.10}$$

kde $m = 0, ..., L-1$ a L označuje počet použitých filtrů.

Nakonec je třeba na vzniklé hodnoty aplikovat exponencionální funkci jako inverzi k funkci logaritmické.

$$S_{mel}[m] = e^{Z[m]}, \tag{2.11}$$

kde $m = 0, ..., L-1$ a L označuje počet použitých filtrů.

## 2.8 Rekonstrukce odhadu výkonové spektrální hustoty segmentu

Při tomto kroku rekonstrukce se využívá přímé úměry, která platí mezi hodnotou melovských koeficientů a úrovní energie v odhadu výkonové spektrální hustoty. Při výpočtu se tedy využívá *trojčlenky*, a to podle následujícího postupu:

1. Jednotlivé složky výsledného rekostruovaného vektoru se nastaví na hodnotu 0.

2. U vektorů jednotlivých filtrů se stanoví součet jejich složek.

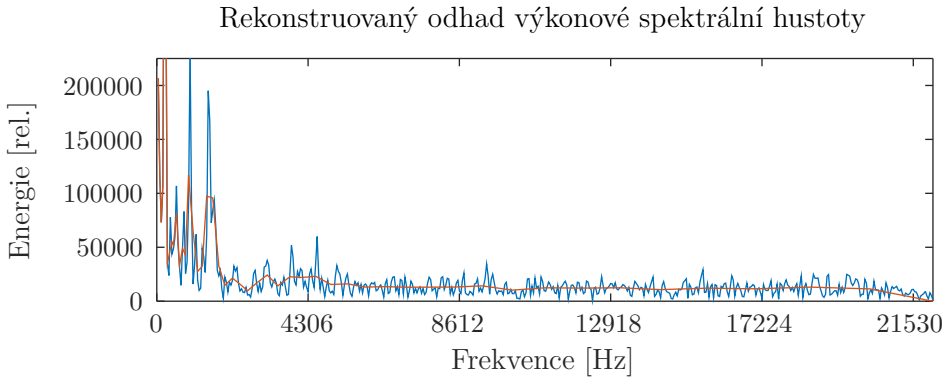
$$R[l] = \sum_{i=0}^{N-1} |H_l[i]| \tag{2.12}$$

3. Konkrétní hodnota rekonstruovaného vektoru se pak rovná její předchozí hodnotě, ke které je přičtena míra zastoupení v rámci daného filtru. Výpočet je následující:

$$X[k] = X[k] + S_{mel}[l]\left(\frac{H_l[k]}{R[l]}\right), \tag{2.13}$$

   kde $k = 0, ..., N-1$ a tento krok je potřeba opakovat pro všechna $l = 0, ..., L-1$.

Zrekonstruovaný odhad spektrální výkonové hustoty je zobrazen na obrázku 2.7. Tato konkrétní rekonstrukce byla provedena ze 48 mel-frekvenčních kepstrálních koeficientů a stejného počtu trojúhelníkových filtrů, tj. uvažujeme všechny koeficienty.




Obrázek 2.7: Rekonstruovaný odhad výkonové spektrální hustoty segmentu.

## 2.9 Inverzní Fourierova transformace

K diskrétní Fourierově transformaci existuje předpis její inverze daný vztahem 2.14.

$$x[k] = \frac{1}{N} \sum_{n=0}^{N-1} X[n] e^{\frac{j2\pi nk}{N}}, k = 0, 1, ..., N-1 \tag{2.14}$$

Tato transformace očekává na vstupu vektor komplexních koeficientů. Při parametrizaci byla ale tato komplexní čísla nahrazena jejich velikostmi, čímž se ztrácí informace o fázi rekonstruovaného signálu. Z tohoto důvodu byl experimentálně využit a pro potřeby této práce upraven algoritmus *Iterative Inverse Short-time Fourier Transform Magnitude Algorithm* [30], který je popsán následovně:

1. Na vstupu je očekáván odhad výkonové spektrální hustoty rekonstruovaného segmentu ($|X|$) a počet iterací algoritmu.
2. Výstupem je pak segment akustického signálu ($x$).
3. Na počátku algoritmu uvažujme, že $x$ bude bílý šum.
4. Dále se provede DFT signálu $x$. Výsledek této transformace označme $Y$.
5. Výsledný vektor komplexních váhových koeficientů ($Y$) je převeden do polárních souřadnic.
6. U jednotlivých prvků vektoru $Y$ se nahradí jednotlivé magnitudy, a to magnitudami z vektoru $|X|$. Úhel $\omega$ zůstává nezměněn.
7. Vektor $Y$ je opět převeden do kartézských souřadnic.

8. Vektor $x$ je nahrazen inverzí DFT vektoru $Y$.

9. Algoritmus pokračuje opětovně bodem 4 až do stanoveného počtu iterací.

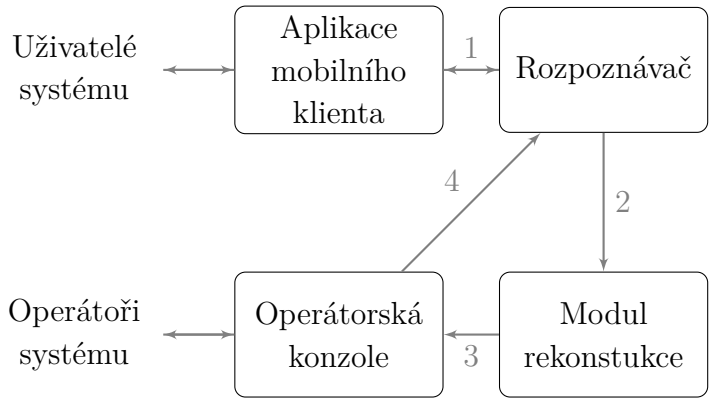
## 2.10 Sestavení akustického signálu

Po inverzi diskrétní Fourierovy transformace, která je popsána v sekci 2.9, je třeba překrývající se segmenty sestavit do cílového časového průběhu rekonstruovaného akustického signálu. Z experimentů vyplývá, že je vhodné překrývající se vzorky sousedních segmentů průměrovat, aby se eliminovaly možné nespojitosti druhého druhu, které by se později negativně projevily při přehrávání audio záznamu.

# 3 Architektura systému

## 3.1 Původní architektura

Původní architektura systému PocketEAR byla postavena na myšlence rekonstrukce akustického signálu z daných vektorů mel-frekvenčních kepstrálních koeficientů. Systém by tak měl následující strukturu:




Obrázek 3.1: Schéma původní architektury systému PocketEAR.

Jednotlivé relace chápeme následovně:

1. Aplikace mobilního klienta zaznamenává akustický signál pomocí mikrofonu zařízení, parametrizuje jej metodou MFCC a výsledné koeficienty odesílá rozpoznávači, od něhož může, ale nemusí, přijít odpověď.

2. Rozpoznávač pomocí hluboké neuronové sítě klasifikuje parametrizovaný audio záznam. Pokud uspěje, odešle třídu zvuku zpět ke klientské aplikaci. V případě neúspěchu předá koeficienty k rekonstrukci původního signálu pro potřeby operátorů systému. Tato rekonstrukce by kvůli vysoké výpočetní náročnosti probíhala v nočních hodinách, čímž se vylučuje odpověď operátora v reálném čase.

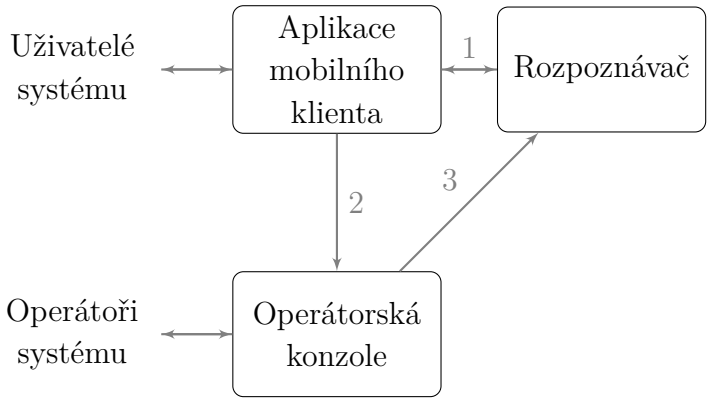
3. Rekonstruované úseky audio záznamu jsou předány operátorské konzoli, která je cyklicky přidružuje operátorům ke klasifikaci.

4. Po tom, co operátor určí obsah daného záznamu, je tato informace přenesena zpět k rozpoznávači, čímž se rozpoznávač dále učí.

Hlavní výhodou této architektury je bezesporu minimální objem dat v rámci relace 1, jejíž fyzickou implementací je ve většině případů komerční mobilní síť s omezenou rychlostí přenosu dat v oblastech s horším pokrytím.

## 3.2 Implementovaná architektura systému

Během práce se ukázalo, že rekonstrukce akustického signálu z mel-frekvenčních koeficientů je možná, ale pro potřeby rozpoznávání obecného zvuku dává velice zkreslené výsledky (více viz sekce 2.7). Z tohoto důvodu bylo nutné architekturu systému následovně upravit:




Obrázek 3.2: Schéma výsledné architektury systému PocketEAR.

Význam jednotlivých relací je následující:

1. Stejný význam jako v původní architektuře (viz obrázek 3.1).
2. V případě, že rozpoznávač nedokáže určit povahu zaznamenaného signálu, odešle směrem ke klientovi požadavek o poskytnutí neparametrizovaných vzorků nerozpoznaného úseku záznamu.
3. Stejný význam jako relace 4 v původní architektuře (viz obrázek 3.1).

## 3.3 Výhody a nevýhody výsledné architektury

Velkou nevýhodou tohoto návrhu je zvýšený objem datového přenosu směrem od klientské aplikace, která pro komunikaci využívá hlavně mobilní sítě. Přenosová rychlost takové sítě může být silně degradována v závislosti na odlehlosti lokality uživatele systému a samozřejmě i dostupné službě.

Výhodou oproti původnímu návrhu je výrazné snížení výpočetních nároků díky vyloučení modulu rekonstrukce, což potenciálně otevírá možnost klasifikace audio záznamu operátorem v reálném čase.

# 4 Vývoj knihovny `libpe`

V návaznosti na předchozí studium metody mel-frekvenčních kepstrálních koeficientů byla pro potřeby systému PocketEAR vytvořena knihovna `libpe`, která obsahuje veškeré prostředky pro parametrizaci akustického signálu a jeho zpětnou rekonstrukci.

## 4.1 Užité technologie

Kvůli značným výpočetním nárokům zejména v oblasti rekonstrukce jednotlivých segmentů z MFCC byl přirozenou volbou nějaký z nízkoúrovňových jazyků (například *C* nebo *C++*). Z důvodu plánované implementace klientské aplikace s využitím knihovny *Qt* byl kvůli následnému snadnému spojení obou projektů využit právě jazyk *C++* a zmíněná knihovna.

### 4.1.1 *Qt*

*Qt*[3], čti `/'kju:t/`, je multiplatformní framework s otevřeným zdrojovým kódem napsaný v jazyce *C++*. Je vyvíjen norskou společností *The Qt Company*. Kromě širokých možností tvorby grafických rozhraní, které budou popsány v sekci 5.1, nabízí framework obdobnou funkcionalitu a implementované datové struktury jako knihovna *STL (Standard Template Library)*.

## 4.2 Struktura knihovny

Obsahem knihovny je 11 tříd, zdrojové soubory knihovny *KissFFT*[7], která je potřebná pro výpočet Rychlé Fourierovy transformace, a jeden konfigurační soubor.

### 4.2.1 Konfigurační soubor

V tomto souboru se nalézají instrukce potřebné ke správnému použití knihovny v jiných projektech. Hlavní částí konfiguračního souboru jsou definované konstanty, které ukazují veškeré možné nastavení jednotlivých tříd. Všechny třídy používají pro inicializaci svých instancí parametry konstruktoru, které nejsou nijak závislé na zmíněných konstantách, avšak jejich použití jako parametrů jednotlivých konstruktorů se doporučuje kvůli lepší správě kódu.

Důležitými konstantami jsou například:

- `AUDIO_CODEC` – metoda kódování vzorků zvuku,
- `SEGMENT_SIZE` – počet vzorků jednotlivých segmentů akustického signálu,
- `OVERLAP` – počet vzorků, jimiž se dva sousední segmenty překrývají,
- `NUM_FILTERS` – počet trojúhelníkových filtrů pro výpočet MFCC,
- `MFCC_COUNT` – počet mel-frekvenčních kepstrálních koeficientů z jednoho segmentu,
- `ESPD_RECOVERY_ITERS` – počet iterací algoritmu inverzní Fourierovy transformace popsaném v sekci 2.9.

### 4.2.2 Důležité implementované třídy

#### `AudioWindower`

Třída představuje vstupně-výstupní zařízení, do kterého putují data ze zvukového vstupu. Data následně pomocí cyklického bufferu rozkládá na segmenty o daném počtu vzorků a velikosti překryvu.

#### `WindowFunction`

Jedná se o virtuální třídu, která představuje obecnou váhovací funkci. Na vstupu této třídy jsou jednotlivé segmenty, které třída podle svých implementací váhuje a výsledek vrátí. Knihovna `libpe` obsahuje jednu konkrétní implementaci této třídy, kterou je `HammingWindow`.

#### `HammingWindow`

Je konkrétní implementací obecné váhovací funkce `WindowFunction`. Váhovacím oknem v této implementaci je Hammingovo okno blíže popsané v sekci 2.2.

#### `FFT`

Třída, která je postavena nad knihovnou *KissFFT*[7] (*Keep it simple, stupid FFT*), což je velice malá a efektivní knihovna s otevřeným zdrojovým kódem. Při konstrukci instance třídy `FFT` je požadováno zadání velikosti vstupních segmentů. Pokud mají vstupní segmenty menší počet vzorků, jsou doplněny

nulami, aby počet prvků vstupního segmentu byl nejnižší další hodnotou posloupnosti $a_n = 2^n$, kde $n \in \mathbb{N}$ (více viz sekce 2.3).

#### `MFCC`

Obsahuje metody pro výpočet mel-frekvenčních kepstrálních koeficientů z daných odhadů výkonové spektrální hustoty jednotlivých segmentů. Prvním krokem výpočtu MFCC z odhadu výkonové spektrální hustoty je filtrace pomocí trojúhelníkových filtrů, jejichž tvorba je úzce spjata s třídou `MelFilterBank`. Dalším krokem je výpočet hodnoty logaritmu vypočtených koeficientů a jejich převod na kepstrální užitím DCT, která je součástí třídy.

#### `MelFilterBank`

Dle zadané velikosti vstupních odhadů výkonové spektrální hustoty třída vytváří, uchovává a uvolňuje hodnoty jednotlivých trojúhelníkových filtrů, které jsou následně využity ve třídě `MFCC`.

#### `ESPDRecover`

Představuje první krok při rekonstrukci akustického signálu z mel-frekvenčních kepstrálních koeficientů. Podle daných MFCC obnovuje odhad výkonové spektrální hustoty segmentu, a to díky pouhé znalosti celkové energie jednotlivých filtrů a trojčlenky.

#### `SegmentRecover`

Metody této třídy představují upravený algoritmus *Iterative Inverse Short-time Fourier Transform Magnitude Algorithm*, který je blíže popsán v sekci 2.9.

#### `AudioComposer`

Přijímá rekonstruované segmenty akustického segmentu, které podle daného překryvu spojuje v celistvý akustický signál. Doporučenou možností této třídy je průměrování překrývajících se vzorků sousedních segmentů, což podle provedených experimentů snižuje počet nespojitostí druhého druhu v průběhu signálu.

## 4.3 Ukázka rekonstrukce

Pro ukázku možností knihovny `libpe` byla vytvořena aplikace ukázky rekonstrukce, která se stala hlavním nástrojem pro rozhodování mezi původní a nově upravenou architekturou celého systému (viz kapitola 3).

### 4.3.1 Výsledky hodnocení kvality rekonstrukce

Jedním z klíčových kroků při vývoji architektury systému PocketEAR bylo hodnocení kvality rekonstrukce nezaujatými posluchači. Výsledky těchto hodnocení naleznete v tabulce 4.1, kde jednotlivá kritéria byla hodnocena na stupnici od 0 do 10 a jejich význam je následující:

- **Srozumitelnost**

  Jaká je zřetelnost v případě záznamu lidského hlasu, kdy bodové ohodnocení je následující:

  0–3 znamená absolutní nesrozumitelnost obsaženého slova,

  4–6 značí nemožnost určení některých pasáží textu,

  7–10 znamená celkové porozumění rekonstruované řeči.

- **Kvalita**

  Pokud je zaznamenaný zvuk obecný (například zvíře, přírodní úkaz, stroj atd.), toto kritérium uvádí, jak moc je zvuk rozpoznatelný lidským uchem. Stupnice bodového ohodnocení je:

  0–3 totální nemožnost určení významu zvuku,

  4–6 obsah zvuku je velmi nejasný s větším počtem možných odpovědí uživatele,

  7–10 z rekonstruovaného zvuku je jasně zřejmý jeho obsah.

- **Čistota zvuku**

  Při jakémkoli sémantickém obsahu záznamu uživatel stanovil, v jaké míře rekonstruovaný zvuk obsahuje různé kazy, jako je například „pískání“, „cvakání“, „šumění“ apod. Bodová stupnice tohoto kritéria je stanovena následovně:

  0–3 zvuk je víceméně tvořen pouze posloupností výše uvedených kazů,

  4–6 v záznamu se vyskytují kazy, které ale neimplikují neúspěšnost určení obsahu zvuku,

7–10 zvuk neobsahuje žádné výrazné kazy.

| **Dotázaný** | **Srozumitelnost** | **Kvalita** | **Čistota zvuku** |
|---|---|---|---|
| 1 | 8 | 3 | 5 |
| 2 | 7 | 2 | 3 |
| 3 | 8 | 2 | 4 |
| 4 | 6 | 1 | 3 |
| 5 | 7 | 2 | 2 |
| 6 | 6 | 3 | 4 |
| 7 | 8 | 2 | 5 |
| 8 | 8 | 4 | 6 |
| 9 | 7 | 3 | 5 |
| 10 | 7 | 3 | 4 |

Tabulka 4.1: Tabulka výsledků hodnocení kvality rekonstrukce.

Celková hodnocení v jednotlivých kategoriích byla stanovena aritmetickým průměrem, jehož hodnoty jsou následující:

- **Srozumitelnost = 7,2**, tj. celkové porozumění rozpoznávané řeči,
- **Kvalita = 2,5**, tj. totální nemožnost určení obsahu zvuku,
- **Čistota zvuku = 4,1**, tj. v záznamu se vyskytují kazy, které ale neimplikují neúspěšnost určení obsahu zvuku.

Z výsledného hodnocení vyplývá, že rekonstrukční algoritmus není schopen dostatečně rekonstruovat obecný zvuk (hlavně z důvodu ztráty informace o fázi originálního signálu ve vztahu 2.2), ale je použitelný například v oblasti rozpoznávání řeči s použitím metody MFCC.

# 5 Klientská aplikace

## 5.1 Užité technologie

Kvůli potřebám multiplatformního řešení a předchozí implementaci potřebné knihovny `libpe` byl pro tvorbu aplikace užit jazyk *C++* a framework *Qt*. Z hlediska tvorby grafického rozhraní nabízí *Qt* následující dva možné přístupy.

- ***Qt Widgets***

  Modul, který obsahuje klasické desktopové grafické komponenty [21] napsané v jazyce *C++* pro tvorbu okenních aplikací. Má podobnou strukturu a styl použití jako jmenný prostor *System.Windows.Forms* [26] frameworku *.NET*. Nicméně použití této sady pro mobilní aplikace se nedoporučuje, a to jak z praktického, tak estetického hlediska.

- ***Qt Quick***

  Sada grafických komponent [15], která používá deklarativní jazyk *QML* [14] (Qt Meta-object Language) založený na jazyce *Javascript*. Jedná se o nové pojetí tvorby grafických rozhraní. Zdrojový kód napsaný v jazyce *QML* je přeložen za běhu aplikace. Tento přístup je navržen hlavně pro tvorbu mobilních aplikací, ale lze jej využít i pro desktop. Na rozdíl od modulu *Qt Widgets* nabízí například návrh vlastního vzhledu ovládacích prvků, animace a nejrůznější efekty. Jediným drobným problémem je právě ona komunikace mezi programem *C++* na pozadí a rozhraním popsaným *QML*.

## 5.2 Struktura aplikace

Při vývoji aplikace byla využita architektura MVC, jejíž základní myšlenkou je oddělení logiky od výstupu[9]. Logickou část aplikace tvoří program napsaný v jazyce *C++* a výstup pak zmíněný *QML* engine.

Aplikace je rozvržena do tří hlavních částí.

- **Hlavní kontroler**

  Propojuje všechny dílčí části, zabývá se spouštěním a bezpečným ukončením aplikace a obsahuje objekt, ve kterém je spuštěn *QML* kód na popředí.

- **Zpracování zvuku**

  Spouští a ukončuje všechna explicitně definovaná vlákna (viz sekce 5.3.1), spravuje nahrávání zvuku, jeho distribuci (jak v parametrizované, tak originální formě), konverzi do formátu *MP3* apod. Pro podrobnější popis navštivte sekci 5.3.

- **Síťové rozhraní**

  Sestává z tříd, které zprostředkovávají komunikaci mezi aplikací a serverem operátorské konzole a rozpoznávače. Detailnější popis naleznete v sekci 5.4.

## 5.3 Zpracování zvuku

Aplikace je z pohledu zpracování zvuku postavena na knihovně `libpe`, jejíž součásti kvůli zvýšení výkonu aplikace paralelizuje (viz sekce 5.3.1). Následující třídy se významným dílem podílejí na zpracování zvuku.

#### `PEAudioProcessor`

Metody této třídy řídí celou práci se zvukem. Vytvářejí instance tříd knihovny `libpe`, spouští a ukončuje vlákna zmíněná v sekci 5.3.1 a distribuuje veškeré požadavky a odpovědi mezi hlavním kontrolerem a objekty správy zvuku.

#### `SegmentSeries`

Přestavuje sérii segmentů akustického signálu (popř. vektorů mel-frekvenčních kepstrálních koeficientů), která tvoří jeden celek odesílaný rozpoznávači. Tato série segmentů získává v rámci spuštění aplikace jednoznačný 32-bitový číselný identifikátor, podle kterého jsou pak spravovány zdrojové soubory odpovídajících nahrávek.

#### `RecordWorker`

Třída, která reprezentuje pracovní vlákno (v terminologii frameworku *Qt* označeno jako *worker*[22] (pracovník)). Úlohou této třídy je záznam akustického signálu, tvorba segmentů a jejich následné váhování.

#### `MFCCWorker`

Stejně jako třída `RecordWorker` je tato pracovníkem v odděleném vláknu. Účelem této třídy je výpočet mel-frekvenčních kepstrálních koeficientů z da-

ných sérií segmentů. Tato třída zastává roli konzumenta v úloze *producent-konzument*, která je řešena v sekci 5.3.2.

#### `AudioFileStorage`

Úlohou této třídy je správa audio záznamů odpovídajících jednotlivým sériím segmentů. Správou je míněno vytváření a odstraňování záznamů jak ve formě *PCM* vzorků, tak formátu *MP3*. Pro potřeby této úlohy byla využita třída `QTemporaryDir` [19], která zabezpečuje tvorbu dočasné složky v dočasných adresářích konkrétního operačního systému. Takto vytvořený dočasný adresář je automaticky odstraněn ve chvíli destrukce instance třídy `QTemporaryDir` nebo samotným operačním systémem v případě selhání aplikace.

#### `MP3Compressor`

Účelem třídy je převod vzorků akustického signálu do souboru formátu *MP3*, a to pouze v případě, kdy rozpoznávač není schopen rozpoznat obsah záznamu dané série, a je ji tedy třeba odeslat na server operátorské konzole k analýze. Pro samotnou konverzi byla využita knihovna *LAME* [8] verze 3.100 s otevřeným zdrojovým kódem.

### 5.3.1 Paralelizace výpočtů

Kvůli vysoké výpočetní náročnosti záznamu akustického signálu, jeho parametrizaci metodou mel-frekvenčních kepstrálních koeficientů a případnému převodu zdrojových *PCM* dat jednotlivých záznamů do *MP3* formátu byla aplikace z pohledu práce se zvukem paralelizována do těchto čtyř explicitně vytvořených vláken:

1. Záznam akustického signálu, tvorba segmentů a jejich váhování užitím Hammingova okna.
2. Výpočet odhadů výkonové spektrální hustoty segmentů a mel-frekvenčních kepstrálních koeficientů.
3. Správa, tj. zápis a odstraňování, zdrojových souborů nahrávek.
4. Převod nahrávek z originální *PCM* formy do formátu *MP3*.

Aplikace je dále tvořena hlavním vláknem, ve kterém pracuje například grafické rozhraní. Další vlákna jsou implicitně vytvářena dle potřeb mechanismu *signal-slot* [17] knihovny *Qt*.

### 5.3.2 Řešení komunikace mezi vlákny

Většina takové komunikace je v prostředí frameworku *Qt* řešena mechanismem *signal-slot*. Pokud jsou metody jiných vláken invokovány pomocí tohoto mechanismu a nastavení `Qt::QueuedConnection`, jejich fyzické vykonávání probíhá právě ve volaném vlákně.

Jediný řešený problém nastává při výměně dat mezi vlákny záznamu zvuku a výpočtu *MFCC*. Kvůli vysoké výpočetní náročnosti metody *MFCC* je možné zaplnění fronty zpráv (segmentů akustického signálu) mechanismu *signal-slot*, což vede ke zhroucení celé aplikace. Z tohoto důvodu byla v tomto bodě implementována třída `SharedQueue`, která je popsána níže.

#### `SharedQueue`

Jedná se o třídu, která pomocí *monitoru* (viz přednášky *KIV/ZOS*[27]) vytvořeného ze synchronizačních primitiv `QMutex`[13] a `QWaitCondition`[20] realizuje frontu mezi producentem (vláknem záznamu zvuku) a konzumentem (výpočtem MFCC). Na rozdíl od fronty zpráv mechanismu *signal-slot* je tak možné neaktivní vlákna uspávat, řídit počet čekajících segmentů ve frontě a popřípadě bezpečně ukončit program při nedostatečném výpočetním výkonu.

## 5.4 Síťová rozhraní

Podmínkou plné funkčnosti aplikace je síťové připojení jak k serveru rozpoznávače, tak operátorské konzoli. Pokud tedy selže spojení alespoň k jednomu z těchto serverů, spuštění aplikace bude zakázáno a případné běžící nahrávání zastaveno. Mechanismy pro připojení a výměnu dat obsahují třídy `ManagerConnection` a `RecognizerConnection`, nastavení připojení pak udržuje a distribuuje třída `ConnectionConfig` podle návrhového vzoru *Observer* [31]. Zmíněné třídy jsou detailně popsány níže.

#### `Connection`

Virtuální třída, která předepisuje funkcionalitu obecného síťového připojení. Implementuje konstruktor, který automaticky nastaví příjem aktuálního nastavení z instance třídy `ConnectionConfig`. Při určení nových parametrů připojení ve třídě `ConnectionConfig` se v konkrétní implementaci virtuální třídy `Connection` zavolá metoda, která odpojí stávající připojení a vytvoří nové. Dále třída obsahuje stavové proměnné metody pro stanovení stavu konkrétního připojení.

#### `ConnectionConfig`

Třída zastupuje úlohu *observera* ve stejnojmenném návrhovém vzoru. Instance této třídy jsou přímo dostupné z prostředí, ve kterém běží popředí aplikace. Úpravou hodnot v grafickém rozhraní se tedy nezaobírá globální kontroler, nýbrž rovnou metody této třídy, které validují zadané IP adresy a porty a distribuují nová nastavení připojením. Uložené hodnoty jsou udržovány v dočasných souborech, jejichž správa je v kompetenci třídy `QSettings` [16].

#### `ManagerConnection`

Jedná se o jednu z konkrétních implementací virtuální třídy `Connection`, která je postavena na *HTTP* protokolu. Jelikož je tento aplikační protokol bezestavový, aplikace se o výpadku serveru operátorské konzole dozví až ve chvíli neúspěšného odeslání nového nerozpoznaného záznamu. Pro komunikaci byla využita třída *QNetworkAccessManager* [12] frameworku *Qt*. Podrobný popis komunikačního protokolu naleznete v sekci 6.4.1.

#### `RecognizerConnection`

Třída je druhou implementací virtuální třídy `Connection`. Obsahuje prostředky pro komunikaci mezi mobilní aplikací a serverem rozpoznávače. Komunikace je postavena na bitově orientovaném protokolu (viz 5.4.1), který pracuje na transportním protokolu `TCP` implementovaném ve třídě frameworku *Qt* – `QTcpSocket` [18].

### 5.4.1 Aplikační protokol klient – rozpoznávač

Pro komunikaci mezi klientskou aplikací a serverem rozpoznávače byl vytvořen bitově orientovaný aplikační protokol postavený na transportním protokolu `TCP`[1]. U dále uvedených formátů přenášených požadavků prosím chápejte položky v závorkách <> jako mandatorní a [] jako fakultativní.

Obecný tvar požadavku je definován následovně:

```
PE<1B-typ-požadavku>[data]
```

[1] `TCP` je spojovaný, spolehlivý protokol transportní vrstvy *ISO/OSI* modelu.

### Požadavky aplikace rozpoznávači

1. **Úvodní *handshake***

   `PE0`

   Účelem požadavku je ověření, že druhou stranou nově vytvořeného spojení je skutečně server rozpoznávače. Validní odpovědí na tento požadavek je hlášení o stavu (*status*).

2. **Odeslání MFCC dané série (*sendMFCC*)**

   `PE1<4B-id-série><4B-počet-bytů-dat><data>`

   Ve formě tohoto požadavku klientská aplikace odesílá samotné mel-frekvenční kepstrální koeficienty rozpoznávači za účelem započetí určování jejich obsahu.

### Požadavky rozpoznávače aplikaci

1. **Aktuální *status* rozpoznávače**

   `PES<1B-status>`

   Pomocí tohoto požadavku informuje rozpoznávač aplikaci o svém stavu. V současné verzi zná komunikační protokol pouze stav `0`, tj. vše v pořádku (OK). Pokud aplikace přijme jinou hodnotu než `0`, zahlásí chybu a přeruší záznam zvuku.

2. **Kladná odpověď rozpoznávače (*recognized*)**

   `PER<4B-id-série><řetězec-kategorie>/<řetězec-typ>`

   Požadavek představuje kladnou odpověď serveru, která obsahuje třídu a typ rozpoznaného zvuku. V návaznosti na tento typ odpovědi klientská aplikace provede odstranění souboru nahrávky dané série.

3. **Záporná odpověď rozponávače (*not-recognized*)**

   `PEN<4B-id-série>`

   Touto odpovědí dává rozpoznávač najevo nemožnost rozlišení jakéhokoli zvuku v nahrávce. Mobilní aplikace tedy na základě tohoto požadavku spustí kompresi zdrojových *PCM* dat zvukového záznamu do formátu *MP3*, který odešle operátorské konzoli ke stanovení zvuků.

# 6 Operátorská konzole

## 6.1 Užité technologie

Zejména dle předchozích zkušeností autora práce (předmět *KIV/WEB*[25]) byla pro implementaci operátorské konzole vybrána rodina svobodného softwaru LAMP[1].

### *Apache*

*Apache* [1] je projekt skupiny dobrovolníků po celém světě se snahou vytvořit robustní, funkční a volně dostupnou implementaci serveru *HTTP* s otevřeným zdrojovým kódem. Tento projekt je součástí *Apache Software Foundation*.

### *MySQL*

*MySQL* [10] je velice rychlý a spolehlivý systém řízení báze dat s otevřeným zdrojovým kódem vyvíjený společností Oracle®. Pro správu využívá relační model dat.

### *PHP*

PHP [11] je široce užívaný skriptovací jazyk s otevřeným zdrojovým kódem, který je využívaný zejména pro tvorbu dynamických webových rozhraní.

### *jQuery*

Knihovna napsaná v jazyce *Javascript* pro usnadnění práce programátora na straně webové aplikace[6]. Obsahuje i prostředky pro asynchronní dotazování serveru pomocí technologie *AJAX*[4].

### *Bootstrap*

Jedná se o knihovnu s otevřeným zdrojovým kódem pro rychlou a intuitivní tvorbu responzivních webových rozhraní užitím jazyků *HTML*, *CSS* a *Javascript*[5]. Nejnovější verzí je *Bootstrap v4.1*.

[1] Zkratka, která označuje sadu svobodného softwaru *Linux™*, *Apache*, *MySQL* a *PHP*.

***Twig***

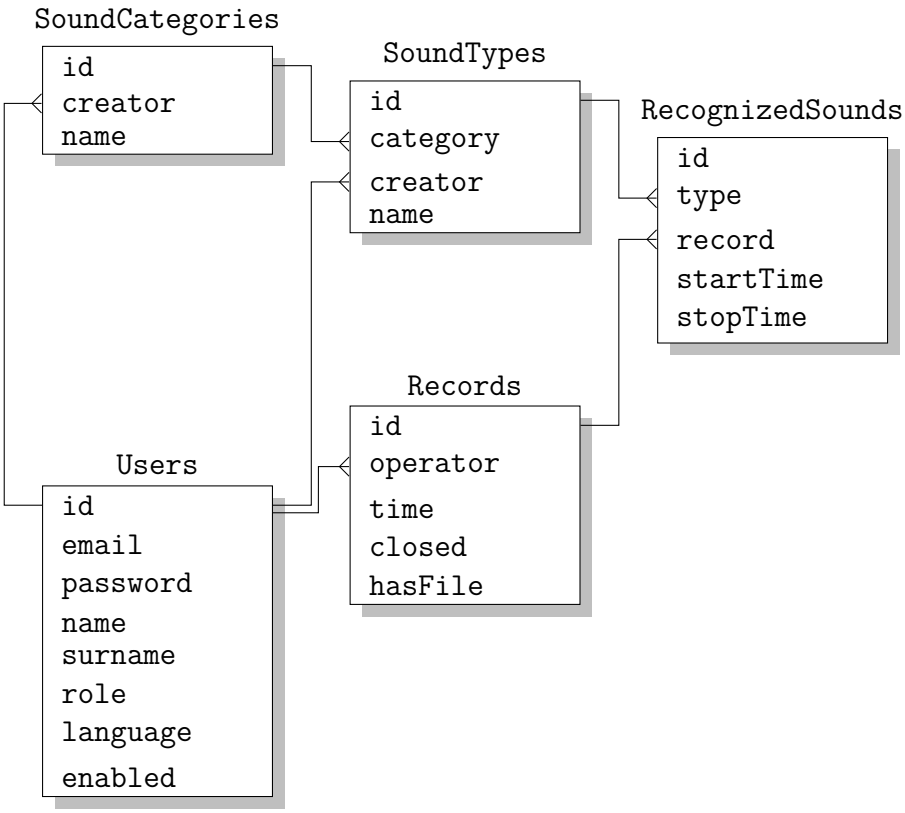
Flexibilní, rychlý a bezpečný šablonovací systém napsaný v jazyce *PHP* s otevřeným zdrojovým kódem [23].

***Wavesurfer.js***

Je nástroj pro vizualizaci průběhu akustického signálu audio souborů [24].

## 6.2 Databázový systém

### 6.2.1 ER diagram




Obrázek 6.1: ER diagram implementované databáze.

### 6.2.2 Popis tabulek

Všechny tabulky obsahují atribut `id`, který představuje primární klíč jednoznačně identifikující jednotlivé záznamy dané tabulky.

#### `Users`

Tabulka obsahuje všechny evidované uživatele operátorské konzole. Jsou zde zapsáni jak administrátoři, tak operátoři. Mezi těmito rolemi rozlišuje enumerace `role`. Uživatele lze dále jednoznačně identifikovat pomocí atributu `email`. Účty uživatelů jsou chráněny heslem, které je uloženo v atributu `password`, a to ve formě otisku *SHA-256*. Před samotným hashováním jsou hesla konkatenována s obskurními řetězci `PassSalt` a `PassPepper`, což znemožňuje jejich odcizení při úniku dat z tabulky.

#### `Records`

Obsahem této tabulky jsou všechny přijaté nahrávky nerozpoznaných zvuků od klientských aplikací. Při přijetí je záznam zapsán do tabulky společně s identifikátorem operátora, který zvuk analyzuje (viz 6.4.2). Atribut `closed` říká, zda je záznam uzavřen, tj. operátor stanovil všechny obsažené zvuky a nahrávka je připravena pro odeslání k rozpoznávači.

#### `RecognizedSounds`

Obsahuje typy zvuků (`type`), které byly určeny operátory v konkrétních záznamech (`record`) v přesně vymezeném časovém intervalu (`startTime` a `stopTime`) v milisekundách.

#### `SoundTypes`

Eviduje všechny známé typy zvuků, které dle potřeb vytvořili operátoři konzole (`creator`). Typy jsou pomocí atributu `category` rozděleny do kategorií.

#### `SoundCategories`

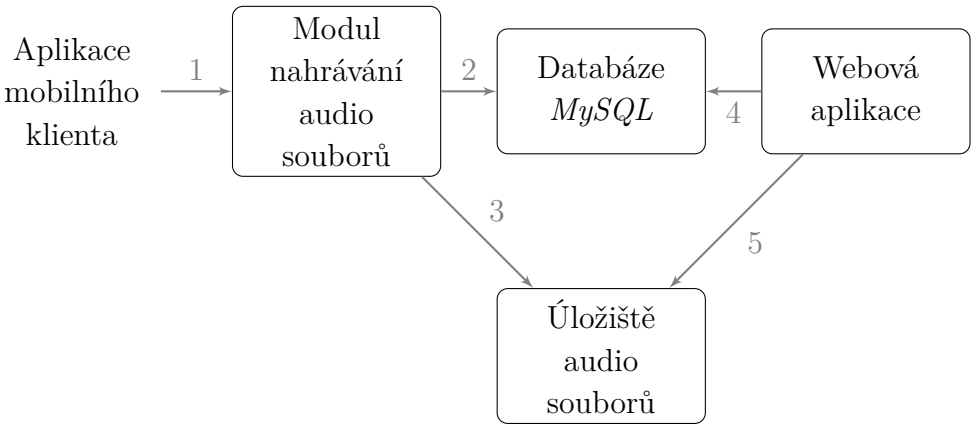
Vyjadřuje výčet všech známých kategorií zvuků, které byly vytvořeny konkrétními operátory (`creator`).

## 6.3 Struktura operátorské konzole

Operátorská konzole byla rozdělena do dvou na sobě nezávislých modulů, kterými jsou *Modul nahrávání audio souborů* a samotná *Webová aplikace*. Více o těchto modulech viz sekce 6.4 a 6.5.

Schéma 6.2 popisuje strukturu celé operátorské konzole. Jednotlivé body ve schématu pak mají následující význam:

1. Mobilní aplikace pomocí aplikačního protokolu *HTTP* nahrává audio soubory ve formátu *MP3*.
2. Modul zapíše nově příchozí nahrávku do databáze a přidělí ji operátorovi podle vztahu 6.1.
3. Audio soubor je zapsán do úložiště, jehož umístění je definováno v konfiguračním souboru.
4. Operátoři skrze webové rozhraní přistupují k databázi uložených nahrávek, již označených zvuků apod.
5. Skrze webové rozhraní si operátoři stahují zdrojové soubory nahrávek pro potřeby dalšího označování zvuků.




Obrázek 6.2: Struktura operátorské konzole systému PocketEAR.

## 6.4 Modul nahrávání audio souborů

Úlohou této části operátorské konzole je příjem a evidence nově příchozích audio nahrávek z klientských aplikací a jejich následná evidence v operátorské konzoli. Pro příjem audio záznamů byla využita zjednodušená architektura *REST* (*REpresentational State Transfer*) založená na *HTTP* protokolu a webovém serveru *Apache*.

### 6.4.1 Struktura komunikačního protokolu

Mobilní aplikace komunikuje s modulem pomocí *HTTP* protokolu. Toto řešení bylo zvoleno pro svou jednoduchost a snadnou implementaci v prostředí jazyka *PHP*. Jednotlivé požadavky jsou v předem stanovené struktuře přenášeny na server metodou `POST`.

**Požadavky aplikace modulu**

- `handshake`
  Požadavek je odesílán pro ověření, zda se skutečně jedná o server operátorské konzole. V závislosti na odpovědi pak mobilní aplikace rozhodne, zda je spojení připraveno.

- `audio`
  Tímto požadavkem mobilní aplikace upozorňuje modul na nově příchozí soubor, který je pak očekáván pod označením `audioFile`.

**Odpovědi modulu aplikaci**

- `pe_uploader_ok`
  Tento požadavek reprezentuje kladnou odpověď serveru, která je v současné verzi protokolu užita pouze jako validní odpověď na požadavek `handshake`.

- `<název_souboru>`
  Při úspěšném přijetí a zavedení nové nahrávky operátorskou konzolí je název přijatého souboru odeslán zpět klientské aplikaci. Na to aplikace reaguje odstraněním audio souboru z paměti mobilního zařízení.

- `pe_uploader_error`
  Touto odpovědí operátorská konzole ohlašuje blíže nespecifikovanou chybu, na kterou mobilní aplikace reaguje zastavením nahrávání. Kompletní původ a obsah chyby je pak podrobněji popsán v logovacím souboru, jehož umístění je dáno konfiguračním souborem.

### 6.4.2 Mechanismus přijetí nového audio souboru

Při přijetí nové nahrávky modul provede následující akce:

1. Zkontroluje správnost přijatého souboru a v případě, že soubor neodpovídá kritériím, odstraní jej a klientské aplikaci odešle zpět chybovou odpověď `pe_uploader_error`.

2. Spustí transakci, která nejdříve vloží novou nahrávku do databáze, dle nově přiděleného primárního klíče určí operátora, kterému bude nahrávka přidělena ke zpracování, a nakonec přesune soubor nahrávky do adresáře definovaného, jehož umístění je definováno konfiguračním souborem. V případě jakékoli chyby je databáze uvedena zpět do konzistentního stavu.

#### Kontrola příchozích souborů

Všechny příchozí audio soubory musí před přijetím operátorskou konzolí projít následující množinou kontrol:

1. Kontrola `MIME`[2] typu přijatého souboru.
2. Ověření velikosti souboru dle dané minimální a maximální hodnoty.

#### Přidělování nahrávek operátorům

Dle požadavků zadavatele není vhodné, aby jedna přijatá nahrávka byla dostupná více operátorům. Z toho důvodu jsou nahrávky přidělovány cyklicky podle vztahu

$$F_u = I_r \bmod U, \tag{6.1}$$

kde $F_u$ je identifikátor přidruženého operátora, $I_r$ je identifikátor nově přidaného audio záznamu a U je počet existujících a neblokovaných operátorů konzole.

## 6.5 Webová aplikace

### 6.5.1 Architektura aplikace

Webová aplikace je postavena na upravené architektuře MVC. Pro tvorbu uživatelského rozhraní byl použit šablonovací systém *Twig*[23], který vykresluje výsledky kořenového skriptu `index.php`, který je při doporučeném nastavení webového serveru jediným přístupovým bodem. Aplikace je rozdělena do sekcí, jejichž kontrolery jsou napsány v jazyce *PHP*. V těchto kontrolerech spočívá právě ona úprava MVC architektury. Kontrolery sekcí totiž generují kompletní *HTML* kód výsledných sekcí, který šablonovací systém *Twig* již dále neupravuje.

### 6.5.2 Role uživatelů

Uživatelé operátorské konzole jsou rozděleni do následujících tří skupin.

- **Návštěvník** (`Visitor`)

  Výchozí role nově příchozího uživatele. Má velmi omezená přístupová práva, konkrétně se může pouze přihlásit (sekce `login`) nebo změnit jazyk (`changeLang`).

[2]Multipurpose Internet Mail Extensions – rozšíření elektronické pošty, které dovoluje odesílat data v binární podobě.

- **Administrátor** (`Admin`)

  Hlavní úlohou administrátorů je registrace nových operátorů a sledování průběžného stavu operátorské konzole, tj. kolik je operátorů, evidovaných nahrávek, typů a kategorií zvuků apod.

- **Operátor** (`Operator`)

  Jedná se o poučeného uživatele, jehož úkolem je analýza a stanovení typu nerozpoznaných zvuků v daných nahrávkách. Tito operátoři jsou registrováni pouze administrátorem. Jeho dalším úkolem je případná tvorba nových kategorií a typů zvuku, což vyžaduje patřičnou zodpovědnost kvůli možnosti existence významově duplicitních záznamů. Problém duplicit je programem operátorské konzole řešen pouze ze syntaktického hlediska. Pokud již v databázi operátorské konzole existuje typ zvuku „falling leaves“, pak nový typ „Falling Leaves“ nelze vytvořit. Ovšem tvorbě typu „fall of leaves“ software konzole nezabrání.

### 6.5.3 Řízení přístupu k sekcím

V příloze C.3.1 je důrazně doporučeno, aby kořenový adresář webového hostitele byl nastaven až na složku `public`. Je tak zaručeno omezení přístupu k sekcím jenom uživatelům s rolí, kterým je sekce přístupná dle konfiguračního souboru (`AllowedSections`). Pokud tedy uživatelům s rolí *Návštěvník* odeberete právo k přístupu do sekce `login`, přihlášení dalších uživatelů již nebude možné. Nutno podotknout, že konkrétní sekce již nekontrolují roli uživatele, který k ní přistupuje.

# 7 Závěr

Na základě studia metody MFCC byla vytvořena a implementována knihovna `libpe`, která obsahuje veškeré potřebné nástroje pro tvorbu krátkodobých segmentů z daného akustického signálu s volitelnou velikostí a překryvem, váhování segmentů užitím Hammingova okna s možností snadné implementace dalších typů oken (Hannovo, Blackmanovo atd.), výpočet odhadu výkonové spektrální hustoty obecně velkých segmentů akustického signálu založený na knihovně *KissFFT* a výpočet mel-frekvenčních kepstrálních koeficientů z těchto odhadů.

Nedílnou součástí knihovny `libpe` je i vyvinutý algoritmus rekonstrukce původního akustického signálu, který je založený na upraveném algoritmu *Iterative Inverse Short-time Fourier Transform Magnitude Algorithm*[30]. Výsledná kvalita rekonstruovaného signálu je podle nezávislých testů dostačující pro porozumění mluvené řeči. Výsledky těchto testů jsou k zhlédnutí v sekci 4.3.1. V budoucnu je plánováno další rozšiřování tohoto algoritmu pomocí autokorelačních technik při zpětném spojování segmentů akustického signálu.

Tato knihovna má v současném konceptu systému PocketEAR ještě minimálně jedno neimplementované uplatnění v podobě aplikace, která parametrizuje zvuky označené uživateli operátorské konzole pro potřeby dalšího učení rozpoznávače.

Podle zadání byla vytvořena aplikace mobilního klienta, která prostřednictvím knihovny `libpe` parametrizuje akustický signál z mikrofonu zařízení. Výsledné mel-frekvenční kepstrální koeficienty odesílá serveru rozpoznávače. V závislosti na odpovědi pak vypisuje výsledky nebo provádí kompresi zaznamenaných *PCM* dat do souborů formátu *MP3*, které předává serveru operátorské konzole.

Poslední implementovanou součástí systému je operátorská konzole, která přijímá nerozpoznané záznamy ke klasifikaci skupinou operátorů. Na základě těchto klasifikací pak probíhá další učení rozpoznávače. V budoucnu je plánována implementace klasifikace nerozpoznaných zvuků v reálném čase. Při neúspěchu rozpoznávače by tak uživatel dostal pohotovou odpověď od lidského operátora.

# Literatura


[1] *Apache* [online]. [cit. 2018-04-14]. Dostupné z: https://httpd.apache.org/.

[2] *GNU Octave* [online]. [cit. 2018-04-14]. Dostupné z: https://www.gnu.org/software/octave/.

[3] *Qt* [online]. [cit. 2018-04-14]. Dostupné z: https://www.qt.io/.

[4] *AJAX* [online]. [cit. 2018-04-14]. Dostupné z: https://www.w3schools.com/js/js_ajax_intro.asp.

[5] *Bootstrap* [online]. [cit. 2018-04-14]. Dostupné z: https://getbootstrap.com/.

[6] *jQuery* [online]. [cit. 2018-04-14]. Dostupné z: https://www.w3schools.com/jquery/jquery_intro.asp.

[7] *Kiss FFT* [online]. [cit. 2018-04-14]. Dostupné z: https://sourceforge.net/projects/kissfft/.

[8] *Lame* [online]. [cit. 2018-04-14]. Dostupné z: http://lame.sourceforge.net/index.php.

[9] *Úvod do MVC architektury* [online]. [cit. 2018-04-14]. Dostupné z: https://www.itnetwork.cz/csharp/asp-net/mvc/asp-dot-net-uvod-do-mvc-architektury.

[10] *MySQL* [online]. [cit. 2018-04-14]. Dostupné z: https://www.mysql.com/.

[11] *PHP* [online]. [cit. 2018-04-14]. Dostupné z: http://www.php.net/.

[12] *QNetworkAccessManager* [online]. [cit. 2018-04-14]. Dostupné z: http://doc.qt.io/qt-5/qnetworkaccessmanager.html.

[13] *QMutex* [online]. [cit. 2018-04-14]. Dostupné z: http://doc.qt.io/qt-5/qmutex.html.

[14] *Qt QML* [online]. [cit. 2018-04-14]. Dostupné z: https://doc.qt.io/qt-5.10/qmlapplications.html.

[15] *Qt Quick* [online]. [cit. 2018-04-14]. Dostupné z: https://doc.qt.io/qt-5.10/qtquick-index.html.

[16] *QSettings* [online]. [cit. 2018-04-14]. Dostupné z: http://doc.qt.io/qt-5/qsettings.html.

[17] *Signals & Slots* [online]. [cit. 2018-04-14]. Dostupné z: http://doc.qt.io/archives/qt-4.8/signalsandslots.html.

[18] *QTcpSocket* [online]. [cit. 2018-04-14]. Dostupné z: http://doc.qt.io/qt-5/qtcpsocket.html.

[19] *QTemporaryDir* [online]. [cit. 2018-04-14]. Dostupné z: http://doc.qt.io/qt-5/qtemporarydir.html.

[20] *QWaitCondition* [online]. [cit. 2018-04-14]. Dostupné z: http://doc.qt.io/archives/qt-4.8/qwaitcondition.html.

[21] *Qt Widgets* [online]. [cit. 2018-04-14]. Dostupné z: https://doc.qt.io/qt-5.10/qtwidgets-index.html.

[22] *Qt Threading Basics* [online]. [cit. 2018-04-14]. Dostupné z: https://doc.qt.io/qt-5.10/thread-basics.html.

[23] *Twig* [online]. [cit. 2018-04-14]. Dostupné z: https://twig.symfony.com/.

[24] *wavesurfer.js* [online]. [cit. 2018-04-14]. Dostupné z: https://wavesurfer-js.org/.

[25] *Courseware KIV/WEB* [online]. [cit. 2018-04-14]. Dostupné z: https://courseware.zcu.cz/portal/studium/courseware/kiv/web.

[26] *Jmenný prostor System.Windows.Forms* [online]. [cit. 2018-04-14]. Dostupné z: https://msdn.microsoft.com/en-us/library/system.windows.forms(v=vs.110).aspx.

[27] *Courseware KIV/ZOS* [online]. [cit. 2018-04-14]. Dostupné z: https://courseware.zcu.cz/portal/studium/courseware/kiv/zos.

[28] Fousek, P. *Diplomová práce. Předzpracování řeči s šumovým pozadím pro účely komunikace a rozpoznávání.* České vysoké učení technické v Praze, Fakulta elektrotechnická, 2002. Dostupné z: http://noel.feld.cvut.cz/speechlab/publications/026_diplomka02.pdf.

[29] Jarolín, M. *Bakalářská práce.* Západočeská univerzita v Plzni, Fakulta aplikovaných věd, Katedra kybernetiky, 2012. Dostupné z: https://dspace5.zcu.cz/bitstream/11025/5845/1/M.Jarolin%20-%20Bakalarska%20Prace%202012.pdf.

[30] Min, G. et al. Speech Reconstruction from Mel-frequency Cepstral Coefficients via L1-norm Minimization. 2015. Dostupné z: https://www.researchgate.net/publication/308201459_Speech_reconstruction_from_mel-frequency_cepstral_coefficients_via_l1-norm_minimization.

[31] PECINOVSKÝ, R. *Návrhové vzory: [33 vzorových postupů pro objektové programování]*. Computer Press, 2007. ISBN 978-80-251-1582-4.

[32] TŮMA, J. *Zpracování signálů získaných z mechanických systémů užitím FFT*. Sdělovací technika, 1997. ISBN 80-901936-1-7.

# A Zkratky

**MFCC** – Mel-frequency cepstral coefficients (metoda reprezentace odhadu výkonové spektrální hustoty krátkodobého segmentu akustického signálu)

**DFT** – Discrete Fourier Transform (diskrétní Fourierova transformace)

**FFT** – Fast Fourier Transform (rychlá Fourierova transformace)

**DCT** – Discrete Cosine Transform (diskrétní kosinová transformace)

**STL** – Standard Template Library (softwarová knihovna jazyka *C++*)

**QML** – Qt Meta-object Language (deklarativní jazyk pro tvorbu grafických rozhraní založený na *Javascriptu*)

**LAME** – „LAME Ain't an MP3 Encoder“ (knihovna pro kódování *MP3 souborů*)

**HTTP** – Hypertext Transfer Protocol (aplikační protokol pro výměnu hypertextových dokumentů)

**REST** – REpresentational State Transfer )

**PHP** – „PHP: Hypertext Preprocessor“ (skriptovací jazyk pro tvorbu dynamických webových stránek)

**NDK** – Native Development Kit (sada nástrojů umožňující využití nativních jazyků při vývoji pro platformu *Android*®)

**SDK** – Software Development Kit (sada nástrojů pro vytváření aplikací)

**AJAX** – Asynchronous JavaScript and XML (asynchronní načítání dat ze serveru pomocí *Javascriptu*)

# B Uživatelská příručka

Následující sekce této kapitoly obsahují instrukce pro užívání aplikace ukázky rekonstrukce, klientské aplikace a operátorské konzole systému PocketEAR.

## B.1 Ukázka rekonstrukce

Celá ukázka probíhá následovně:

1. Při spuštění ukázky má uživatel dle vyznačených popisků možnost zaznamenat akustický signál pomocí svého vstupního zvukového zařízení.
2. Zaznamenaný zvuk si může následně přehrát.
3. Následujícím krokem je spuštění parametrizace s bezprostředně následující rekonstrukcí. Při tomto kroku je možné si v seznamu událostí, který obsahuje zmínku o dokončení rekonstrukce každého segmentu, všimnout zmíněné vysoké časové náročnosti.
4. Po dokončení rekonstrukce má uživatel možnost přehrát jak originální, tak rekonstruovaný zvuk, na základě čehož pak může rekonstrukci hodnotit.

**Varování!** Při spuštění ukázky na různých zařízeních se v prvním záznamu velmi často vyskytla akustická chyba vstupního zvukového zařízení, která se projevila jako „zapraskání“. Je tedy doporučeno, aby uživatel pro parametrizaci a následnou rekonstrukci užil až druhou nahrávku.

## B.2 Klientská aplikace

Při prvotním spuštění mobilní aplikace jsou přednastaveny původní IP adresy operátorské konzole (manažerského serveru) a rozpoznávače. Tyto hodnoty je možné změnit na obrazovce nastavení, kam se uživatel dostane stisknutím tlačítka v záhlaví hlavního pohledu (obrázek klíče a šroubováku). Konkrétní podoba tlačítka je zobrazena na obrázku E.1.

Z hlediska hlavního pohledu je ovládání následující:

1. V záhlaví aplikace (hned pod tlačítkem nastavení) je popisek, který uživatele informuje o stavu aplikace. Možnými stavy jsou například:

(a) *Připraven*,

(b) *Probíhá záznam zvuku*,

(c) *Připojování...* – aplikace se pokouší připojit k serverům na nastavených adresách,

(d) *Chyba rozpoznávače!* – rozpoznávač na nastavené adrese není dostupný nebo neodpovídá,

(e) *Chyba manažeru!* – modul pro nahrávání souborů operátorské konzole není na nastavené adrese dostupný nebo hlásí chybu.

2. Stiskem tlačítka nahrávání (tlačítko s mikrofonem) se spustí záznam první série segmentů společně s její parametrizací. Tlačítko změní barvu na růžovou. To znamená, že záznam není možné během záznamu první série zastavit.

3. Jakmile tlačítko nahrávání změní barvu na červenou, záznam lze pozastavit. Případná nedokončená parametrizace probíhá dále.

4. Při příchodu validních odpovědí ze serveru rozpoznávače, tj. byl rozpoznán zvuk, se objeví jejich seznam, který je dle procentuální pravděpodobnosti řazen sestupně.

## B.3 Operátorská konzole

Navigační prvky webové části operátorské konzole jsou umístěny na horní části obrazovky a jednotlivé položky nabídek jsou generovány podle konkrétní role uživatele.

### B.3.1 Návštěvník

Výchozí sekcí návštěvníka je přihlášení. Pokud návštěvník nezná přihlašovací údaje, jeho jedinou pravomocí je změna jazyka prostředí.

Pokud návštěvník má zájem o podílení se na projektu, musí kontaktovat administrátora a požádat jej o registraci.

### B.3.2 Operátor

V jednotlivých sekcích má operátor následující možnosti:

1. ***Nerozpoznané nahrávky*** (`sounds`)

   V této sekci operátor nahlíží na všechny záznamy, které operátorská

konzole přijala od mobilních aplikací a nebyly v nich zatím určeny obsažené zvuky. Již v tomto pohledu má operátor možnost prohlédnout si průběh akustického signálu, díky čemuž okamžitě vidí, že nahrávka obsahuje například pouhé ticho. Nahrávky může přehrávat, vstupovat do sekce, která dovoluje jejich další analýzu, nebo je rovnou odstraňovat. Tato sekce je k vidění na obrázku E.2.

2. ***Rozpoznání zvuků*** (`recognize`)

   Po přechodu k analýze konkrétního záznamu operátor vidí podrobnější průběh akustického signálu, který může, stejně jako v předchozím bodě, přehrát nebo pozastavit. Poslední tlačítko slouží pro přehrání výběru, který je v náhledu signálu možné vytvořit pomocí kursoru myši. Vymezení tohoto náhledu slouží právě k označení zvuků. Následujícími panely pak operátor k takto vymezenému zvuku přidruží kategorii a typ. V případě, že požadovaná kategorie nebo typ zatím neexistuje, může jej vytvořit. Poslední panel slouží pro kontrolu a následné odeslání nového určeného zvuku. Náhled do sekce poskytuje obrázek E.3.

3. ***Nastavení*** (`settings`)

   Zde může operátor aktualizovat své přístupové heslo nebo změnit jazyk prostředí.

Po přihlášení je operátor automaticky přesměrován do sekce *Nerozpoznané zvuky*.

### B.3.3 Administrátor

Úlohou administrátora je registrace a správa operátorů konzole. Jeho další možností je sledování stavu systému. Pro tyto účely slouží následující dvě sekce:

1. ***Přehled operátorů*** (`operators`)

   V této sekci má administrátor možnost nahlédnout do seznamu všech registrovaných operátorů konzole. U nich má počet rozpoznaných zvuků, podle kterého může zběžně určit jejich výkonnost. V případě, že si administrátor nepřeje další aktivitu daného operátora, může jej jednoduše zakázat. Operátor se tak již opětovně nepřihlásí.

2. ***Registrace*** (`register`)

   Účelem této sekce je registrace nového uživatele operátorské konzole,

a to jak operátora, tak administrátora. V registračním formuláři jsou všechna pole povinná.

3. ***Stav systému*** (`stats`)

   V záhlaví této sekce může administrátor zhlédnout stav operátorské konzole, vyjádřený v následujících hodnotách:

   (a) *Operátorů* – vyjadřuje počet operátorů, kteří jsou aktivní,

   (b) *Nahrávek* – znamená počet přijatých nahrávek od klientských aplikací,

   (c) *Zvuků* – počet všech určených zvuků ve všech nahrávkách, všemi operátory,

   (d) *Tříd zvuku* – počet vytvořených tříd zvuku,

   (e) *Typů zvuku* – počet existujících typů zvuku ve všech kategoriích.

   Poslední funkcionalitou této sekce je odstranění zdrojových souborů nahrávek, které jsou uzavřeny operátorem, ale nejsou v nich žádné určitelné zvuky. Takové nahrávky jsou de facto k ničemu, a je tedy vhodné je odstranit.

# C Instalační příručka

## C.1 Překlad ukázky rekonstrukce

Přeložená ukázka rekonstrukce pro operační systém *Windows*® je dostupná na přiloženém nosiči. V případě potřeby přeložení aplikace pro jiný operační systém postupujte následovně:

1. Pokud nemáte nainstalovaný framework *Qt*, stáhněte jej z následujícího odkazu:

   ```
   https://www1.qt.io/download/
   ```

2. Po úspěšné instalaci je třeba vytvořit adresář pro sestavené binární soubory.

3. V tomto adresáři je třeba spustit příkaz *qmake*, který podle zadaného projektového souboru (`ReconstuctExample.pro`) vytvoří příslušný *makefile*.

   ```
   qmake PROJECT_DIR/ReconstuctExample.pro -spec linux-g++
   ```

4. Dále pomocí nástroje *make* spusťte samotný překlad a sestavení programu.

   ```
   make
   ```

5. Po úspěšném překladu je nutné, aby soubor `whitenoise_1024.raw` byl ve stejném adresáři, jako je vytvořený spustitelný soubor `ReconstuctExample`.

6. Program `ReconstuctExample` je pak možné spustit.

   ```
   ./ReconstuctExample
   ```

## C.2 Instalace klientské aplikace

Na přiloženém nosiči jsou binární soubory aplikace přeložené pro platformu *Android*, konkrétně pro architekturu *armeabi-v7a*[1], ve formě *APK*[2] souboru.

[1] Aplikace je přeložena přímo pro instrukční sadu procesorů typu ARM.
[2] Balíčkový soubor, využívaný pro distribuci a instalaci mobilních aplikací.

Instalace pak probíhá následovně:

1. V nastavení mobilního zařízení (položka zabezpečení) je třeba povolit možnost instalace nových aplikací z neznámých zdrojů.

2. Libovolným způsobem přeneste *APK* soubor do mobilního zařízení.

3. Pomocí správce souborů nalezněte nový soubor a kliknutím jej nainstalujte.

V případě, že potřebujete aplikaci pro jinou platformu, je třeba ji přeložit pomocí nástroje *Qt Creator*, který je součástí frameworku *Qt* (popřípadě samostatnými nástroji *qmake*, *make*, *NDK* a *SDK*). Dalším nezbytným krokem je překlad knihovny *LAME* pro konkrétní platformu užitím nástroje *NDK*.

## C.3 Instalace a zavádění operátorské konzole

Jak již bylo uvedeno v sekci 6.1, operátorská konzole systému PocketEAR byla vyvíjena v prostředí rodiny svobodného softwaru LAMP. Je tedy nutné, aby pro správnou funkci všech součástí konzole bylo toto programové vybavení nainstalováno a správně nakonfigurováno.

### C.3.1 Instalace webové aplikace

Všechny adresáře složky *Operátorská konzole* je nutné zkopírovat na hostující zařízení. Při následné konfiguraci webového serveru *Apache* je nutné se držet následujících bodů:

1. Kořenové adresáře virtuálních hostů **je třeba** nastavit až na složky `public` v obou částech operátorské konzole. Zamezí se tím přímý přístup ke zdrojovým souborům aplikace a nahrávkám z klientských aplikací (*Secure by design*[3]).

2. Systémový uživatel, pod kterým je virtuální hostitel spuštěn, **musí mít** právo jak čtení, tak zápisu do všech adresářů operátorské konzole.

3. Po zavedení celé aplikace je nutné v konfiguračním souboru (přesněji v abstraktní třídě `Config`) **nastavit URL adresu**, na níž je aplikace dostupná. Systém bude tak moci odkazovat sám na sebe při chybových hlášeních apod.

[3] *Secure by design* – označení pro software, u kterého platí, že znalost jeho principů neohrožuje bezpečnost.

4. Ujistěte se, že máte nainstalováno rozšíření `DOMDocument` interpreta jazyka *PHP*. V opačném případě je třeba jej doinstalovat pomocí balíčkovacího systému. Například takto bude vypadat příkaz instalace na distribuci Debian:

```
sudo apt-get install php5-dom
```

### C.3.2 Import a nastavení databáze

#### Import databáze

Nedílnou součástí operátorské konzole je databáze *PocketEar*, kterou lze importovat z přiloženého souboru `PocketEar.sql`. Při použití nástroje *phpMyAdmin* lze využít jeho nabídky *Importovat*. Pokud se vyskytnou chyby při nahrávání souboru na server, lze databázi instalovat prostým zkopírováním obsahu souboru do sekce *SQL* a jeho následné spuštění.

**Varování!** Pro úspěšné zavedení databáze je nutný *MySQL* server verze minimálně 5.6.

Při instalaci databáze užitím tohoto skriptu byl do tabulky `Users` vložen první uživatel – administrátor, jehož úlohou je registrace dalších uživatelů. Výchozí email tohoto administrátora je možné změnit pouze přímo v databázi (například užitím nástroje *phpMyAdmin*). Výchozí heslo tohoto administrátora je „`pocketear`“.

**Varování!** Jednou z částí tvorby otisku hesla prvotního administrátora bylo zabezpečující spojení s řetězci `UserConfig::PassSalt` a `PassPepper`. Při změně těchto konstant se otisk hesla v databázi stává neplatným.

#### Vytvoření uživatele serveru MySQL

Dalším neméně důležitým krokem je vytvoření uživatele, skrze kterého bude operátorská konzole přistupovat k databázi. Z hlediska zabezpečení databáze je vhodné tomuto uživateli přidělit pouze práva `SELECT`, `INSERT` a `UPDATE` nad databází operátorské konzole.

Přihlašovací údaje vytvořeného uživatele je třeba vložit do konfiguračního souboru operátorské konzole – konkrétně abstraktní třídy `DatabaseConfig`. Oddělený modul pro nahrávání zvukových souborů má možnost zadání odlišného nastavení připojení k databázi. Pro správnou funkci celé operátorské konzole je dostačující užití již vytvořených přihlašovacích údajů i pro tento modul.

### C.3.3 Vytvoření prvního operátora

Před prvním použitím operátorské konzole je nutné vytvořit prvního operátora, kterému budou následně přidělovány přijaté zvukové soubory. Pokud nebude žádný operátor vytvořen, modul nahrávání nových souborů bude požadavky odmítat.

# D Testování aplikace

## D.1 Kontrolní výpočty s GNU Octave

Všechny výpočty prováděné knihovnou `libpe` byly kontrolovány odpovídajícím výpočtem v prostředí GNU Octave [2] podle programů vytvořených na přiloženém nosiči ve složce *Octave*.

Výsledky těchto kontrolních programů můžete vidět jako ukázkové grafy v kapitole 2.

## D.2 Rychlost klientské aplikace

Následující test kontroluje, zda je mobilní aplikace schopna běhu na daných modelech mobilních zařízeních. Jinými slovy, zda čas, který je třeba pro výpočet mel-frekvenčních kepstrálních koeficientů a jejich odeslání na server rozpoznávače, je menší než čas samotného záznamu. Pokud by tato nerovnost neplatila, fronta čekajících segmentů by se zaplnila a aplikaci by nebylo možné dále použít.

Měření probíhalo užitím třídy `QElapsedTimer`. Při každém testu byl spuštěn záznam deseti sérií segmentů a výsledný čas poté zprůměrován. Výsledky tohoto testu jsou k nahlédnutí v tabulce D.1.

| **Typ mobilního zařízení** | $t_1$ $[ns]$ | $t_2$ $[ns]$ | $t_1 \geq t_2$ |
|---|---|---|---|
| Lenovo K5 | 2983513061 | 956437390 | ✓ |
| Lenovo P70-M | 2998184216 | 652157569 | ✓ |
| Lenovo Vibe P1M | 2995452346 | 1326645785 | ✓ |
| Lenovo Tab 2 | 2995711146 | 970693831 | ✓ |
| **Průměrný čas** | 2993215192 | 976483643 | ✓ |

Tabulka D.1: Tabulka konzumace času při záznamu a parametrizaci akustického signálu.

Význam jednotlivých sloupců je následující:

- $t_1$ – čas (v nanosekundách) potřebný pro záznam a následné váhování 256 segmentů akustického signálu (přibližně 3 s záznamu),

- $t_2$ – čas (v nanosekundách) zkonzumovaný na výpočet mel-frekvenčních kepstrálních koeficientů z 256 segmentů akustického signálu.

Z výsledku testu vyplývá, že rychlost výpočtu mel-frekvenčních kepstrálních koeficientů je na použitých mobilních zařízeních přibližně $\mathbf{3,3}\times$ rychlejší než samotný záznam akustického záznamu.

Tzn., že implementovaný algoritmus je z hlediska výpočetní náročnosti využitelný na testovaných mobilních zařízeních a pravděpodobně i na jiných s podobnými parametry.

## D.3 Funkční testy operátorské konzole

Operátorská konzole byla testována třemi naprosto nezasvěcenými uživateli.

Tito uživatelé získali přihlašovací údaje jak pro operátorský, tak administrátorský účet. Následně byli požádáni o hodnocení:

1. jazykové správnosti,
2. celkové estetičnosti prostředí,
3. intuitivnosti rozhraní.

Výsledkem tohoto hodnocení byl stručný soupis dojmů a postřehů.

Dále byli uživatelé požádáni o test následujících funkcionalit:

1. přihlášení s danými přihlašovacími údaji,
2. přihlášení s neexistujícími údaji,
3. pod právy administrátora otestovat
   (a) zablokování libovolného operátora, zkontrolovat odpovídající počet dostupných operátorů a pokusit se přihlásit pod jeho emailem a heslem,
   (b) použít funkcionalitu odstranění nahrávek, které jsou uzavřeny a neobsahují žádné určené zvuky,
4. pod právy operátora otestovat
   (a) přehrání nahrávek již ve výčtu nerozpoznaných záznamů (sekce *Nerozpoznané záznamy*),
   (b) vytvoření nové kategorie a typu zvuku,

(c) označení nového zvuku v libovolné nahrávce,

(d) pokusit se vytvořit duplicitní kategorii a typ zvuku,

5. změnit jazyk rozhraní pod všemi rolemi uživatelů,

6. odhlásit se.

### D.3.1 Výsledky hodnocení prvního uživatele

Na základě testování uživatel uvedl následující slovní hodnocení:

„*Prostředí PocketEAR je opravdu zdařilé. Všechny prvky jsou plně funkční a jejich ovládání je pro uživatele velice intuitivní. Grafické pojetí působí příjemně a drží se rčení ‚V jednoduchosti je krása.‘ Jediné, co v aplikaci postrádám, je možnost administrátora poslechnout si uzavřené nahrávky.*

*Operátorská konzole je plně funkční a splňuje všechny body určené k testování.*“

### D.3.2 Výsledky hodnocení druhého uživatele

Na základě testování uživatel uvedl následující slovní hodnocení:

„*V rozhraní operátorského účtu bych ocenil možnost pozastavit zvuk, ideálně stejným tlačítkem, kterým jsem zvuk přehrál. V rozpoznávání zvuků chybí možnost odznačit vybranou část nahrávky (nyní lze posunout, zvětšit a zmenšit). Vybraná část zvuku by měla mít minimální možnou velikost, vybraný úsek lze zmenšit tak, že prakticky není, a nelze ho tedy znovu zvětšit. Operátor by měl mít možnost upravit již zvolenou kategorii u vybrané části při Rozpoznávání zvuků. U administrátorského účtu mi chybí možnost zobrazení Tříd zvuku, nahrávek a případná možnost jejich smazání, také chybí možnost poslechnout si uzavřené nahrávky a jejich eventuální znovuotevření. I přesto je rozhraní funkční, intuitivní a esteticky přívětivé. Rozhraní je uživatelsky velmi přívětivé i na mobilním telefonu.*“

### D.3.3 Výsledky hodnocení třetího uživatele

Na základě testování uživatel uvedl následující slovní hodnocení:

„*Prostředí webového rozhraní operátorské konzole systému PocketEAR je po vizuální stránce již od prvního pohledu velice příjemné. Až na drobné diskutabilní nepřesnosti (viz odstavec ‚Problémy‘) vyskytující se v sekci ‚Rozpoznání zvuků‘ nebyly nalezeny žádné problémy, které by byly v rozporu s funkčními požadavky, a tak i v tomto ohledu aplikace prošla testem na výbornou. Proto*

*nezbývá nic jiného, než konstatovat, že se test setkal s programem fungujícím dle očekávání, jehož nepřehlédnutelnou a nespornou výhodou je jeho estetičnost.*

***Problémy***

*Při vytváření nové kategorie (resp. typu) zvuku v průběhu označování zatím nerozpoznaných záznamů byly objeveny drobné chyby.*

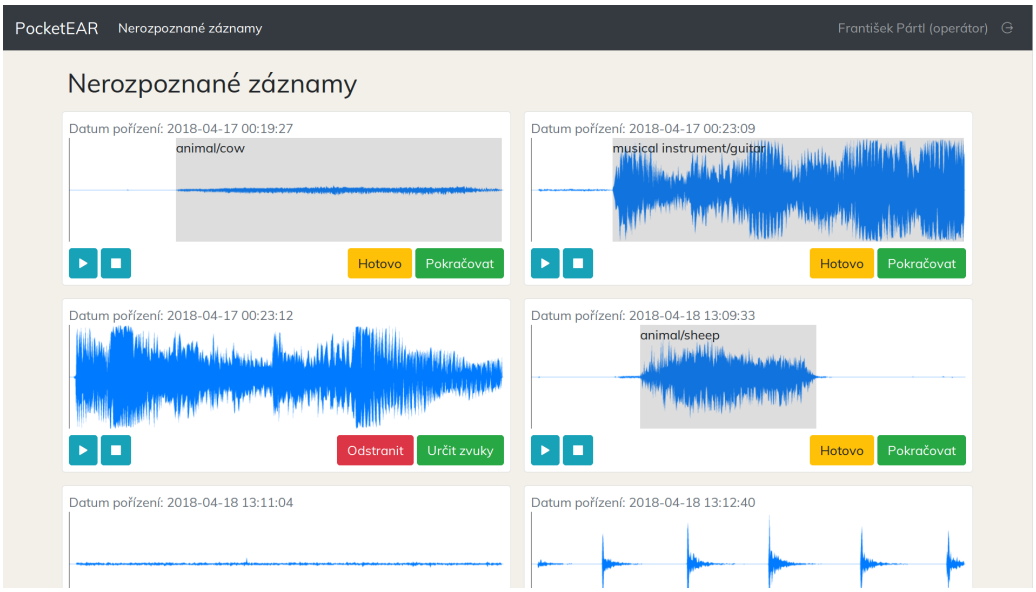
*Je totiž možné vytvořit dvě kategorie s názvem lišícím se pouze ve velikosti (některých) písmen. V tuto chvíli tedy může dojít k situaci, že kromě typu ‚sheep' budou vytvořeny i typy další, jako například ‚Sheep' či ‚shEEP'.*

*Poslední nalezenou situací k zamyšlení je nemožnost vytvoření dvou kategorií (resp. typů), které se budou lišit ‚pouze' tím, že bude součástí jejich názvu znak mezery – například ‚She ep'. Takový případ může být v určité situaci nežádoucí."*

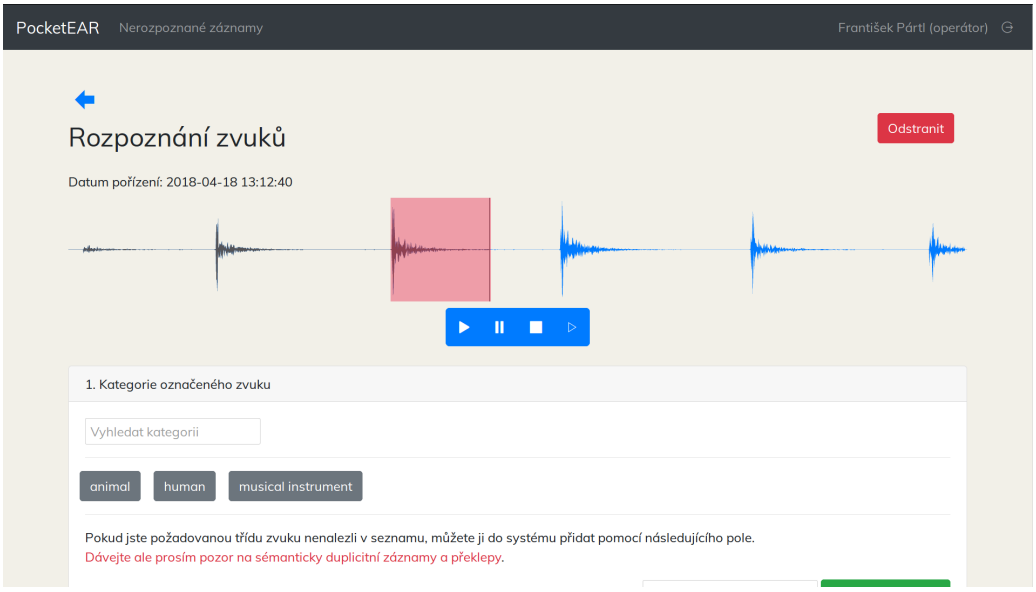
Chyby objevené a popsané tímto uživatelem byly vyhledány a následně úspěšně opraveny.

# E Obrazová příloha

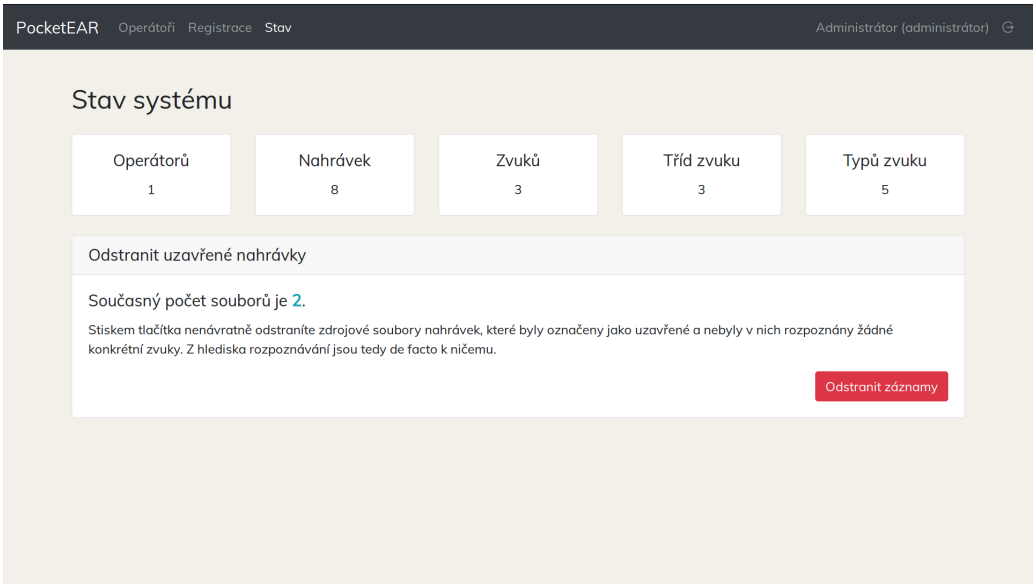

Obrázek E.1: Výpis odpovědí rozpoznávače mobilní aplikací.




Obrázek E.2: Souhrn neklasifikovaných nahrávek operátora.




Obrázek E.3: Označení konkrétního zvuku v nahrávce.




Obrázek E.4: Stav operátorské konzole pro potřeby administrátora.


Obrázek E.5: Nastavení připojení v mobilní aplikaci.