VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

ANDROID APLIKACE PRO ŘÍZENÍ A MĚŘENÍ SPORTOVNÍCH DRILŮ HLASEM/ZVUKY

AN ANDROID APPLICATION FOR CONTROLLING AND MEASURING SPORT DRILLS BY VOICE/SOUND

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. JIŘÍ ODEHNAL**
AUTHOR

VEDOUCÍ PRÁCE — **RNDr. MAREK RYCHLÝ, Ph.D.**
SUPERVISOR

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2018/2019

Zadání diplomové práce

Student: **Odehnal Jiří, Bc.**
Program: Informační technologie Obor: Informační systémy
Název: **Řízení a měření sportovních drilů hlasem/zvuky**
Controlling and Measuring Sport Drills by Voice/Sound
Kategorie: Uživatelská rozhraní

Zadání:

1. Seznamte se s vývojem Android aplikací, použitím rozhraní pro mluvený výstup (TextToSpeech) o možnosti ovládání aplikací hlasem (Google Voice Actions). Prozkoumejte možnosti rozpoznávání zvuků (písknutí, tlesknutí, výstřel, atp.).
2. Navrhněte aplikaci pro OS Android, která umožní zadat textem několik popisů drilů/cvičení s možností jejich parametrizace (náhodný výběr z množiny slov, permutace, číselné řady, pauzy, čekání na hlas/zvuk uživatele, atp.). Aplikace potom umožní přehrání popisů drilů/provedení cvičení s danými parametry a hlasovým výstupem, vč. interakce s uživatelem pomocí jeho hlasových pokynů a zvuků a měření času provedení.
3. Po konzultaci s vedoucím aplikaci implementujte. Zaměřte se na interakci hlasem/zvuky bez nutnosti dotyku/sledování zařízení. Otestujte při cvičení v různě hlučném prostředí.
4. Výsledek popište, vyhodnoťte a zveřejněte jako open-source.

Literatura:

- Phillips, Bill. Android Programming: The Big Nerd Ranch Guide (3rd Edition). ISBN 978-0-13470-605-4.
- Rossi, Mirco, et al. "AmbientSense: A real-time ambient sound recognition system for smartphones." Pervasive Computing and Communications Workshops (PERCOM Workshops), 2013 IEEE International Conference on. IEEE, 2013.

Při obhajobě semestrální části projektu je požadováno:

- 1 a 2.

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Rychlý Marek, RNDr., Ph.D.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 22. května 2019
Datum schválení: 31. října 2018

Abstrakt

Tato diplomová práce se zabývá návrhem a vývojem mobilní aplikace pro platformu Android. Cílem práce je implementovat jednoduché a přívětivé uživatelské rozhraní, které by podporovalo a napomáhalo uživateli v provádění cvičebních sad a cvičebních úkonů. Součástí práce je i implementace podpory detekce zvuků během provádění cviků a předávání hlasových pokynů uživateli. V praxi má aplikace napomoci v pohodlném provádění cvičebních sestav, aniž by byl uživatel nucen mít mobilní zařízení stále v ruce.

Abstract

This master's thesis deals with the design and development of mobile aplication for Android platform. The aim of the work is to implement a simple and user-friendly user interface that would support and assist the user in trainning and sport exercises. The thesis also include implementation of sound detection to support during exercises and voice instruction by application. In practice the application should help in making training exercises more comfortable without the user being forced to keep mobile device in hand.

Klíčová slova

mobilní aplikace, Android, uživatelské rozhraní, detekce zvuku, rozpoznání zvuku, SVM, MFCC, sportovní aktivita

Keywords

mobile application, Android, user interface, sound detection, sound recognition, SVM, MFCC, sports activity

Citace

ODEHNAL, Jiří. *Android aplikace pro řízení a měření sportovních drilů hlasem/zvuky.* Brno, 2019. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce RNDr. Marek Rychlý, Ph.D.

Android aplikace pro řízení a měření sportovních drilů hlasem/zvuky

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana RNDr. Marka Rychlého, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jiří Odehnal
14. května 2019

Poděkování

Tímto bych chtěl poděkovat svému vedoucímu diplomové práce RNDr. Markovi Rychlému, Ph.D. za odborné vedení, rady a připomínky při vypracování této práce. Dále bych chtěl poděkovat všem, kteří mi pomáhali s testováním aplikace a za jejich zpětnou vazbu.

Obsah

Kapitola 1

Úvod

Moderní technologie jsou všude kolem nás a jen stěží se jim v dnešní době dokážeme vyhnout. Většina technologií míří na urychlování a zdokonalování každodenních procesů. Přitom se nejedná jen o platbu v obchodě, vyhledání nejbližší restaurace nebo vyšší komfort při řízení auta, ale také možnost efektivně využít svůj volný čas. Díky hektické době čím dál tím více lidí hledá jistý způsob odreagování ve sportu. Mobilní aplikace, která by cílila právě na tuto skupinu lidí by mohla přinést jistý užitek v podobě zvýšení efektivity a komfortu během cvičení. Aplikace by uživatelům mohla nabídnout podporu v podobě správy a provádění cvičebních sestav, vytváření vlastních drilů (cviků) a možnost hlasové a zvukové interakce s aplikací. Samozřejmostí by byla podpora i stále více populárního intervalového cvičení. Uživatelem sestavené cviky by byly aplikací přímo diktovány uživateli. Tím pádem by uživatel nemusel držet mobilní zařízení v ruce, a přesto by měl cvičební proces plně pod kontrolou. I když aktuálně existují řešení v podobě mobilních aplikací, ve většině případů se jedná o jednoúčelové aplikace zaměřené pouze na jeden typ cviků. Navíc díky všudy přítomné reklamě je jejich využití značně omezené. Zároveň jen minimální procento z nich podporuje přímou interakci s uživatelem. S tím souvisí i další přínos v podobě variability. Aplikace by byla zaměřená na široké spektrum sportů, jenž jsou založeny na provádění jakékoliv posloupnosti cviků nebo intervalů. Aplikaci by tak mohli využít jak nenároční uživatelé (například pro aerobik, běh či posilování), tak i profesionální sportovci (sportovní střelba, bojové sporty apod.)

Cílem práce je navrhnout a implementovat mobilní aplikaci, která by byla schopna podpory cvičebních sad. Součástí práce je analýza existujících řešení a výběr vhodného postupu a algoritmů pro podporu ovládání pomocí hlasu/zvuku. Patřičný důraz je kladen na návrh uživatelského rozhraní a možnost variability cviků. Primárním cílem je tak realizace aplikace poskytující možnost ovládání pomocí hlasu/zvuku, která se zároveň zaměřuje na přehledné, jednoduché a přívětivé uživatelské rozhraní.

Kapitola 2

Platforma Android

Android je open source platforma na bázi Linuxu určená hlavně pro mobilní zařízení, tedy chytré telefony, tablety a fotoaparáty. V současné době se stále více platforma Android objevuje i v televizích, hodinkách a různých dalších zařízení. Android je licencován jako otevřený systém, tudíž všechny části systému jsou volně dostupné. Licencování operačního systému Android však nezůstává pouze u open source licence, ale systém je taktéž šířen pod licencí business friendly, což umožňuje využití systému pro nejrůznější účely prostřednictvím třetích stran.[14]

Platforma je tím pádem otevřena všem vývojářům, kteří pro ni mohou vytvářet aplikace a následně je nejčastěji přes obchod Google Play distribuovat, a to jak placeně, tak i bezplatně.

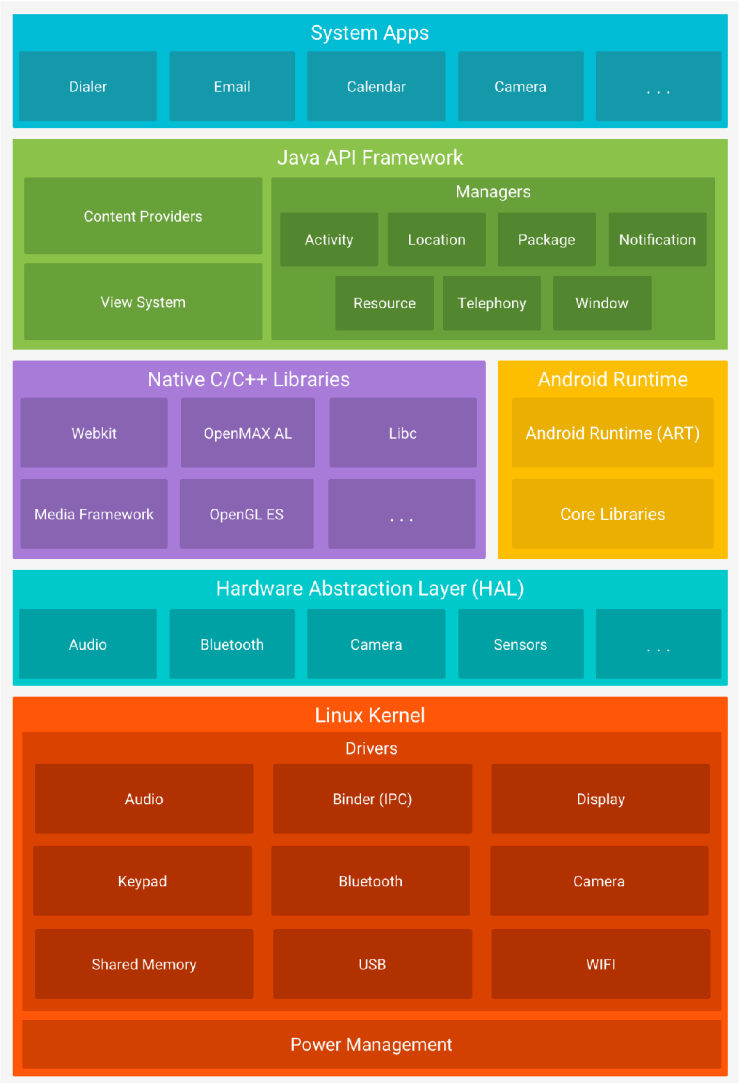
Android byl původně vyvinut v roce 2003 společností Android Inc., kterou v roce 2005 odkoupila společnost Google. V září 2008 přišel v USA na trh první chytrý telefon s Androidem 1.0, který měl na trhu inteligentních telefonů podíl jen 0,5 %. V dubnu 2009 byl vydán Android 1.5, který se záhy stal prvním masově rozšířeným Androidem. Následné verze operačního systému Android pak byly vydávány nejen pod číselným označením, ale i pod jménem, kterým byl název zákusku.[14]

2.1 Architektura operačního systému Android

Pro vývoj aplikací je však potřeba znát alespoň základní informace o principech a architektuře operačního systému, ve kterém budou aplikace spouštěny. Na diagramu 2.1 jsou vyobrazeny základní komponenty architektury Android.[14]

Linux Kernel

Nejnižší vrstvou architektury Android je upravené jádro operačního systému Linux. Úpravy se týkají redukce funkcí a jejich přizpůsobení možnostem mobilních zařízení. Jádro zajišťuje přímou interakci s hardwarem mobilního zařízení, čímž je zabezpečena úplná abstrakce od hardwaru pro vyšší softwarové vrstvy. Zabezpečuje správu paměti, správu procesů, základní síťovou vrstvu a ovladače. Na úrovni jádra je implementováno i zabezpečení systému, správa napájení, vstupně-výstupní operace, či základní grafika. Nicméně vývojáři běžných aplikací s jádrem do přímého kontaktu nepřijdou.[14]



Obrázek 2.1: Schéma architektury operačního systému Android (obrázek převzat z Android Developers[4]).

Hardware Abstraction Layer (HAL)

Hardwarová abstrakční vrstva poskytuje standardní rozhraní, které zpřístupňuje možnosti hardwaru do vyšší úrovně Java API. Modul HAL se skládá z několika knihovních modulů, z nichž každý implementuje rozhraní pro konkrétní typ hardwarové komponenty.[4]

Native C/C++ Libraries

Nad jádrem a HAL vrstvou je situována vrstva knihoven, které poskytují přímý přístup aplikací k různým komponentám systému Android. Jedná se o nativní knihovny napsané v jazyce C/C++. Tvoří tak mezivrstvu mezi komponentami vyšších vrstev a linuxovým jádrem. V případě, že vyvíjíme aplikaci, která vyžaduje kód C nebo C++, můžeme pomocí systému NDK přistupovat k některým z těchto knihoven přímo ze zdrojového kódu. [14]

Android Runtime

U zařízení se systémem Android 5.0 (API 21) a vyšší je každá aplikace spouštěna ve svém vlastním procesu s vlastní instancí Android Runtime (ART). ART je zavedeno pro možnost spouštění více virtuálních strojů na zařízeních s malou pamětí. Používá se k tomu formát souboru DEX, což je formát speciálně navržený pro Android, který je optimalizován pro minimální využití paměti.[4]

Java API Framework

Aplikační Framework obsahuje v aplikacích opakovaně použitelný software, například ovládací prvky, grafické prvky, ikony apod. Framework je napsán v jazyce Java a je nejdůležitější vrstvou pro vývojáře aplikací.[14] Vrstva API je blíže popsána v následující kapitole 2.2.

Systém Apps

Na nejvyšší úrovni architektury operačního systému Android je sada hlavních aplikací např. pro e-maily, SMS zprávy, kalendář, prohlížení webových stránek atd. Aplikace zahrnuté do platformy nemají však žádný zvláštní stav před ostatními aplikacemi, které se uživatel rozhodne nainstalovat. Tudíž aplikace třetích stran se mohou stát např. výchozím webovým prohlížečem, správcem SMS zpráv nebo i výchozí klávesnicí.[4]

2.2 Android API

Jedná se o sadu funkcí operačního systému Android, která je k dispozici skrze rozhraní API (Application Programming Interface), napsané v jazyce Java. Toto rozhraní vytváří stavební bloky, které jsou potřebné pro vývoj aplikací s možností zjednodušit použití jádra a využít modulárních součástí systému a služeb.[1]

Verze API je celočíselná hodnota, která jednoznačně identifikuje verzi rozhraní API, kterou nabízí platforma Android. Každá Android platforma poskytuje API rozhraní, který mohou aplikace využívat pro interakci s jádrem systému. API nabízí následující základní prvky:[1]

- základní sadu balíčků a tříd
- sadu XML prvků a atributů pro deklarování souboru manifest[1]
- sadu XML prvků a atributů pro deklarování a přístup k prostředkům
- sadu Intentů[2]
- sada oprávnění, které aplikace může požadovat

Aktualizace rozhraní API jsou navrženy tak, aby nové API bylo kompatibilní s dřívějšími verzemi. To znamená, že většina změn v API je aditivní a zavádí novou nebo nahrazuje existující funkcionalitu. Pokud jsou části API aktualizovány, starší nahrazené části jsou označeny jako zastaralé, ale nejsou odstraněny, takže stávající aplikace budou

[1]Soubor manifest popisuje základní informace o aplikaci. Soubor je využit operačním systémem Android při překladu aplikace a slouží jako referenční zdroj informací pro obchod Google Play.

[2]Intent je objekt pro zasílání zpráv, který může být použit jako žádost o akci z jiné komponenty aplikace (aktivity, služby,...).

stále funkční. Velmi zřídka může být část API modifikována nebo odstraněna z důvodu udržení robustnosti API a zabezpečení systému. Díky aditivním aktualizacím je zajištěna dopředná kompatibilita s novějšími verzemi platformy Android a verzemi API. Aplikace je tak funkční na všech pozdějších verzích platformy Android. Výjimku tvoří případy, kdy aplikace používá část API, která byla z nějakého důvodu později odstraněna.[1]

Naproti tomu aplikace nejsou zpravidla zpětně kompatibilní se starší verzí API, než ve které byly kompilovány. Každá nová verze platformy Android může obsahovat nové rozhraní API. Příkladem je umožnění přístupu k novým funkcím platformy nebo nahrazení stávajících součásti API. Nové API je přístupné na novějších platformách. Naopak, jelikož starší verze platformy neobsahují nové API, aplikace používající nové API nebudou na těchto platformách fungovat. To lze do jisté míry eliminovat použitím Support Library.[1]

2.3 Android Support Library

Novou aplikaci chceme zpravidla vyvíjet pod nejnovější verzí rozhraní API, neboť očekáváme možnost využití nejnovějších technologií a zároveň podporu aktuálních zařízení. Avšak vývoj pro jednu konkrétní verzi není velmi vhodný, protože na trhu se vyskytuje velké množství zařízení s rozdílnými verzemi API. Zároveň zastoupení jednotlivých verzí na trhu se velmi rychle mění, a tak vývojem aplikace pro jednu konkrétní verzi dokážeme se štěstím pokrýt maximálně 20 % trhu.

Pokud chceme vyvíjet aplikace, které podporují více verzí rozhraní API je třeba zajistit zpětnou kompatibilitu, která nám poskytne nejnovější funkce na starších verzích systému Android. Pro tento účel slouží sada knihoven Android Support Library. Knihovny nám poskytují sadu funkcí a nástrojů, které jsou kompatibilní se všemi knihovnami podporovanými verzemi API. Místo řešení a programování kompatibility mezi jednotlivými verzemi zařízení se tak můžeme zaměřit čistě na vývoj samotné aplikace. Pro pohodlnější vývoj zároveň knihovny nabízí funkce, které nejsou standardně k dispozici ve standardních verzích API a umožňují tak ještě snadnější vývoj.[6]

Android Support Library byla původně jedna knihovna, která se postupně vyvinula v sadu knihoven pro zpětnou kompatibilitu. S každou nově vydanou verzí API je vydána i revize Support Library, která slouží pro podporu vydané verze. Značení Support Library je stejné jako značení verze API. Čili pokud budeme vyvíjet aplikaci např. pod verzí API 28, tak použijeme Spupport Library taktéž s označením 28.

Support Library nabízí několik různých možností použití. Přičemž třídy pro zpětnou kompatibilitu pro starší platformy jsou pouze jedním z mnoha nabízených funkcí. Support Library je možné rozčlenit na následující způsoby, které stručně uvádí, jak podpůrné knihovny a v jakých oblastech používat.[6]

- Zpětná kompatibilita pro nejnovější verze rozhraní API – Velké množství podpůrných knihoven nabízí zpětnou kompatibilitu pro nejnovější třídy a metody.
- Podpůrné knihovny – Support Library poskytují řadu pomocných tříd, zejména pro vývoj uživatelského rozhraní. Příkladem podpůrné knihovny může být třída RecycleView, která poskytuje widget uživatelského rozhraní pro zobrazování a správu velmi dlouhých seznamů.
- Ladění a nástroje – Knihovny nabízí řadu funkcí, které přináší užitek i mimo samotný kód aplikace. Jedná se především o knihovny podpory anotací, podpora překladu aplikace apod.

2.4 Volba verze API

Dříve, než se přistoupí k samotnému návrhu aplikace, je nutné zvolit pod jakou minimální verzí Android API bude aplikace vyvíjena. Android API určuje verze systému Android, a tudíž i rozsah podporovaných zařízení. Od volby API se především odvíjí grafický styl aplikace, množina použitelných nástrojů a množina zařízení, které budou aplikaci podporovat. Ve starších verzích totiž chybí spousta moderních grafických prvků, čímž by bylo nutné větvit zobrazení aplikace pro různé verze systému. Dle tabulky 2.1 je patrné, že vývoj pro verzi Android 4.3 a nižší se již nevyplatí, neboť je používá jen zanedbatelná část uživatelů.

Verze	**Název**	**Verze API**	**Rozšířenost**
2.3.3 - 2.3.7	Gingerbread	10	0.3%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	0.3%
4.1.x	Jelly Bean	16	1.2%
4.2.x	Jelly Bean	17	1.5%
4.3	Jelly Bean	18	0.5%
4.4	KitKat	19	6.9%
5.0	Lollipop	21	3.0%
5.1	Lollipop	22	11.5%
6.0	Marshmallow	23	16.9%
7.0	Nougat	24	11.4%
7.1	Nougat	25	7.8%
8.0	Oreo	26	12.9%
8.1	Oreo	27	15.4%
9.0	Pie	28	10.4%

Tabulka 2.1: Tabulka zobrazující nejrozšířenější verze Android API a jejich zastoupení na trhu. Tabulka nezahrnuje ty verze API, které mají menší zastoupení jak 0.1 %.[2]

Já se rozhodl podporovat co možná nejširší možné rozpětí API verzí s ohledem na možnost použití nejmodernějších nástrojů pro nejnovější verze Android (knihovny, grafické prvky, ...). Proto jako minimální verzi jsem zvolil Android 4.4 KitKat verze API 16. Podpora od této verze tak pokryje zhruba 95 % všech aktuálně používaných zařízení. Verze 4.0.x až 4.3 jsem vyloučil z důvodu již nízkého zastoupení mezi uživateli, které se navíc stále zmenšuje. Jedná se převážně o stará zařízení, která z větší části již výkonem nedostačují moderním aplikacím. Zároveň nejnovější verze Android Support Library podporuje kompatibilitu jen se zařízením s verzí 4.0 a vyšší. Proto z důvodu podpory i zastoupení jsou vyloučeny i verze 2.x a 3.x.

Kapitola 3

Analýza a specifikace požadavků

Pro účely získání základních požadavků, kladených na aplikaci, byly použity tradiční techniky, a to průzkum a rozbor existujících aplikací popsaných v kapitole 3.1. Na základě analýzy existujících systémů byl utvořen základní náhled na funkčnost aplikace a na její grafickou prezentaci. Ze získaných poznatků byla vytvořena textová specifikace požadavků kladených na výslednou aplikaci a byly stanoveny cíle, kterých má aplikace dosáhnout.

3.1 Podobné a existující aplikace

Před vývojem samotné aplikace bylo třeba provést průzkum, zda se na trhu vyskytují aplikace se stejným či podobným zaměřením. Jako zdroj aplikací byl použit obchod Google Play. Během analýzy nebyla zaznamenána žádná aplikace, která by nabízela možnost sledování cvičení a zároveň by nabízela ovládání hlasem. Nicméně v době analýzy existovalo velké množství sportovních aplikací bez možnosti ovládání hlasem. Existovaly i drobné jednoúčelové nesportovních aplikace, které do jisté míry ovládání hlasem podporovaly. Proto v průzkumu existujících aplikací jsou podrobněji rozebráni význačtí zástupci těchto dvou skupin aplikací. Sportovní aplikace:

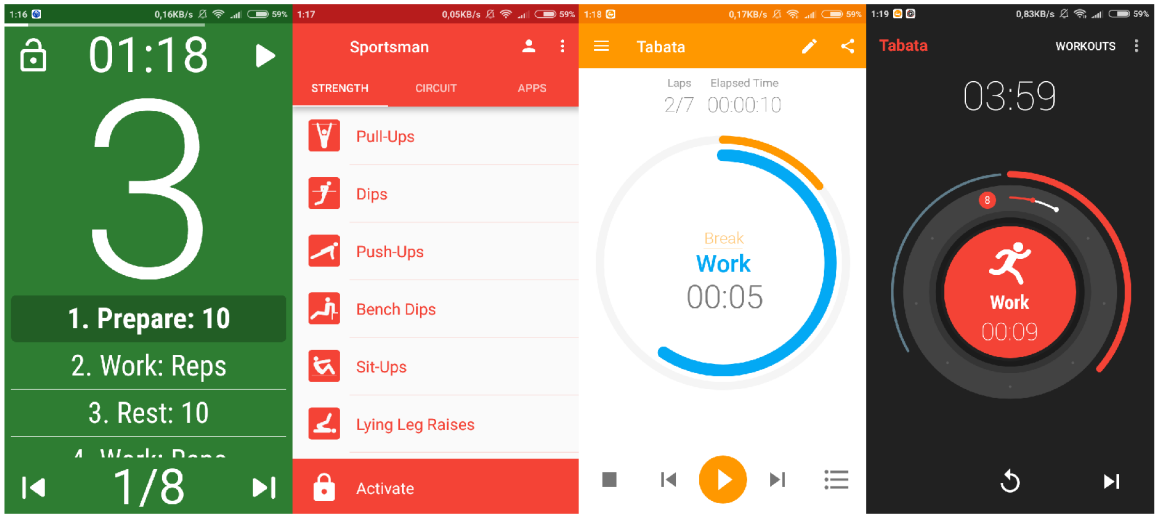
- Tabata Timer: Interval Timer Workout Timer HIIT – je nativní aplikace pro podporu intervalového cvičení. Aplikace uživateli nabízí možnost vytváření vlastních cvičebních sestav (pauzy, přípravy, cvičení, ...) pouze v rámci intervalového cvičení. Tudíž nejsou specifikovány prováděné cviky, ale jde pouze o rozložení zátěže a odpočinku. Aplikace disponuje velmi přehledným a jednoduchým rozhraním, který uživateli nabízí pohodlnou práci nejen při skládání intervalů cvičení, ale i při samotném cvičení. Zároveň jsou v aplikaci přítomny podrobné statistiky, a to vše v české lokalizaci. I když aplikace svou grafickou i funkční prezentací mezi ostatními analyzovanými aplikacemi vyniká, nenabízí podporu hlasového ovládání. Navíc, přítomnost reklam, i když vhodně zakomponovaných do designu, působí rušivě.

- Sportsman – představuje sportovní aplikaci pro podporu různé škály cviků. Aplikace nezobecňuje prováděné cviky, ale naopak, nabízí uživateli možnost volby, jaké cviky (kliky, sedy-lehy, dřepy, ...) bude chtít provádět. Zároveň jsou pro konkrétní cviky či skupiny cviků počítány samostatné statistiky. Aplikace taktéž umožňuje sestavování týdenních plánů přímo dle představ uživatele. Bohužel i přes výhodu pevného členění cviků do kategorií, nemá uživatel možnost sestavit si vlastní cvičební plán. V podstatě má uživatel pouze možnost sestavovat plány jen v rámci pevných kategorií (jen v rámci

jednoho cviku). Aplikace z pohledu uživatelského rozhraní působí přehledně a práce s ním je jednoduchá a intuitivní. Nevýhodou aplikace je nemožnost ovládání hlasem a pro čistě české uživatele je nevýhoda lokalizace pouze v anglickém jazyce.

- Interval Timer – HIIT Tabata Crossfit – je další aplikace pro podporu intervalového cvičení. Hlavní předností je velmi jednoduché uživatelské rozhraní. Na druhou stranu však není možné detailnější nastavení aplikace přímo dle představ a předností uživatele. Stejně jako ostatní aplikace nepodporuje ovládání hlasem a přítomná reklama působí značně rušivě.

- Exercise Timer – je jedna z nejvydařenějších aplikací, které byly v rámci průzkumu analyzovány. Hlavní předností je intuitivní design, díky kterému je i složitější přizpůsobení aplikace jednoduché i pro nového uživatele. Grafické doplňky i rozložení komponent je přehledné a nebýt vložené reklamy, působilo by velmi profesionálně. Aplikace je zaměřena spíše na intervalové cvičení. Tudíž možnost tvorby vlastních sestav je téměř vyloučena. Jednou z předností aplikace je verbální předávání příkazů. Uživatel je informován o aktuálně prováděném intervalu, přípravě na přechod do dalšího intervalu, o poločasech a pauzách. Celá aplikace je lokalizována do anglického jazyka, ovšem díky využití velmi známých slovíček a frází by s jejím používáním neměl mít průměrný Čech problém. I přes mnoho výhod, které tato aplikace nabízí, není možno s ní jakkoliv interagovat pomocí hlasových či zvukových povelů.



Obrázek 3.1: Přehled aplikací. (zleva Tabata Timer: Interval Timer Workout Timer HIIT, Sportsman, Interval Timer – HIIT Tabata Crossfit, Exercise Timer)

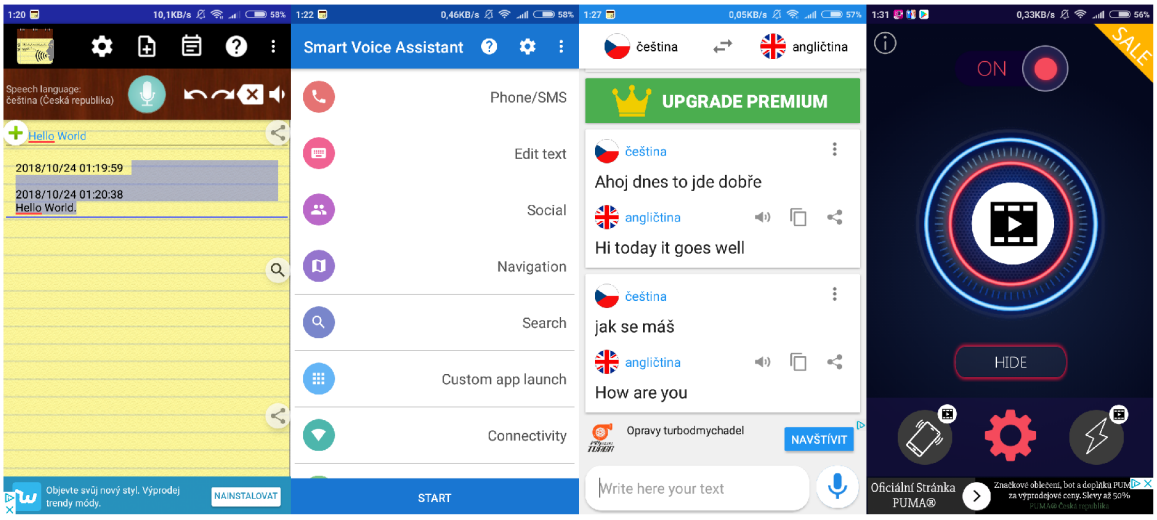
Aplikace detekující zvuky/hlas:

- Voice Note – v podstatě se jedná jen o poznámkový blok, který podporuje nahrávání hlasu a jeho převod do psaného textu, a to vše v reálném čase. Uživatel tudíž nemusí poznámky psát ručně, ale může si je pohodlně nadiktovat. Aplikace reaguje rychle a překlad je přesný i v rušném prostředí. Rozhraní je přehledné, avšak zastaralé a všudypřítomná reklama působí značně rušivě.

- Smart Voice Assistant – je aplikace umožňující nahrát určitý zvuk/slovo/příkaz a spárovat ho s provedením určité akce v telefonu (vytočit hovor, spustit aplikaci, provést vyhledávání, . . .). Aplikace disponuje příjemným a seriózním designem. Aplikace sice umožňuje detailnější nastavení, avšak je značně chaotické. Uživatel tak často neví, zda nastavený příkaz již funguje. To je způsobeno především hlubokým zanořením obrazovek při nastavování aplikace a příkazů.

- Talkao-Talk & Translate – je přehledná aplikace umožňující překlad mezi různými jazyky. Aplikace disponuje velkým množstvím jazyků, do/ze kterých lze překlad provést. Nespornou výhodou oproti jiným aplikacím tohoto druhu je, možnost nahrát překládanou větu, převést ji na text a přeložit do cílového jazyka. Zároveň je překlad přesný i v rušném prostředí. Aplikace uživateli nabízí přehledné a velice jednoduché rozhraní. Díky modernímu designu působí aplikace profesionálním dojmem. Bohužel i v této aplikaci působí vložená reklama jako rušivý element.

- Finding phone by clapping / Finding phone by whistle – je dvojice aplikací od stejného vývojáře, které slouží především pro nalezení mobilního zařízení. Aplikace po spuštění naslouchá okolí a v případě, že detekuje písknutí či tlesknutí, odpoví přehráním zvuku. Jedná se o jednoduchou aplikaci s velmi jednoduchým rozhraním. Bohužel přítomnost reklamy silně degraduje možnost přizpůsobení aplikace (nastavení zvuku, vibrací, . . .). Uživatel je tak vysloveně nucen sledovat reklamní videa v případě, že si chce aplikaci přizpůsobit.



Obrázek 3.2: Přehled aplikací detekující zvuk/hlas. (zleva Voice Note, Smart Voice Assistant, Talkao-Talk & Translate, Finding phone by clapping)

Analýzou aplikací jsme dospěli k závěru, že aplikace, která by nabízela možnost vytvářet sportovní sestavy a zároveň by podporovala ovládání hlasem doposud na trhu není. Ve výběru čistě jen sportovních aplikací se nachází poměrně velké množství propracovaných aplikací, ovšem jen zlomek z nich je k dispozici s českou lokalizací. Zároveň velké množství aplikací je zaměřeno jen na intervalové cvičení, což pro cvičení založené na principu sestav

není vhodné řešení. Taktéž z analýzy vyplynulo, že zhruba polovina analyzovaných aplikací podporuje alespoň základní zvukovou signalizaci, kterou uživatele upozorňují na určité události (začátek cvičení, konec cvičení, ...).

Z oblasti aplikací, které alespoň částečně podporují ovládání hlasem se nachází spíše drobné jednoúčelové aplikace. Až na výjimky však tyto aplikace poskytují jen velmi omezené možnosti nastavení nebo nenabízí nastavení žádné. Převážně se jedná o aplikace pro vyhledávání mobilního zařízení zvukem (většinou se jedná jen o zvuk tlesknutí nebo hvízdání). Samostatnou kapitolu aplikací ovládaných hlasem tvoří aplikace sloužící pro překlad jazyků, které disponují jak propracovanými aplikacemi, tak i dobře odladěnou detekcí hlasu a jeho překladem. Na poli těchto typů aplikací má český uživatel poměrně velký výběr, a dokonce je několik zdařených aplikací i v české lokalizaci.

Z analýzy existujících aplikací tak vyplynulo, že aplikace, která by kombinovala jak sportovní odvětví, tak možnost interakce hlasem, by mohla být značným přínosem nejen pro český trh.

3.2 Neformální specifikace

Primárními uživateli aplikace budou osoby, které se aktivně věnují sportům, jež jsou založeny na sestavách (atletika, kondiční cvičení, bojové sporty apod.). Zejména se tedy bude jednat o osoby pohybující se ve sportovním prostředí. Lze tak očekávat uživatele ze všech věkových skupin a s různou úrovní vzdělání. Proto je třeba předpokládat, že s aplikací budou pracovat nejen technicky znalí uživatelé a je nutno vzít tento aspekt v úvahu především při vývoji uživatelského rozhraní. Sekundárními uživateli budou uživatelé věnující se intervalovému cvičení a HIIT, které není založeno na sestavách, ale intervalech mezi vysokou a nízkou fyzickou aktivitou. Na obě skupiny uživatelů je třeba brát zřetel a je tedy nutno klást důraz na použitelnost a variabilitu aplikace.

Hlavním jádrem aplikace budou sestavy drilů, které si bude uživatel moci sestavit dle libosti. Každý jednotlivý dril musí obsahovat svůj název, který bude zároveň použit jakožto příkaz při provádění drilu. Dril musí v závislosti na svém charakteru obsahovat počet provedení, pauzu mezi aktuálním a následujícím drilem a případně čekání na určitý počet zaznamenaných zvuků, které potvrdí provedení drilu. Zvuk detekující splnění drilu bude možné u jednotlivých drilů navolit. Často se opakující drily bude možno uložit do databáze konceptů. Zároveň každý ukládaný dril bude zařazen do určité skupiny z důvodu oddělení drilů u sportů, které spolu nesouvisí (např. dril pro bojový sport bude uložen pod skupinu bojové sporty). Z obdobného důvodu bude možné pod stejné skupiny zařadit celou cvičební sestavu. Stejně jako sestavy bude možno vytvářet nové a mazat stávající skupiny sportů. Uživatel bude mít k dispozici celkovou hierarchii, na jejímž vrcholu budou skupiny sportů. Při výběru určité skupiny mu budou nabídnuty vytvořené sestavy pro zvolený sport. Výběrem sestavy bude umožněna buď editace sestavy nebo dojde k zahájení provádění sestavy. Během editace bude umožněno uživateli vytvářet nové drily, vkládat do sestavy již dříve uložené drily, editovat již vložené drily a mazat drily v sestavě. Z důvodu zahrnutí intervalového cvičení bude vhodným způsobem umožněno proložit vkládané drily pauzami. Zároveň uživatel musí mít možnost zvolit počet opakování celé sestavy. Během provádění sestavy bude uživatel hlasově navigován pomocí předem připravených drilů. Během provádění sestavy se bude počítat celkový čas za který uživatel sestavu dokončil. Tento čas bude ukládán do databáze pro statistické účely. Zároveň bude uživatel vizuálně (na displeji) upozorněn na aktuálně uplynulý čas a prováděný dril. U drilů čekajících na určitý podnět (zvuk) bude počítán mezičas, který bude třeba taktéž uchovávat.

Předpokládá se, že s aplikací bude vždy pracovat jen jeden uživatel (vlastník mobilního zařízení), tudíž není třeba brát v úvahu registraci nebo přihlašování uživatele.

3.3 Analýza požadavků

S přihlédnutím na obsah textu neformální specifikace (popsané v podkapitole 3.3) a dalších poznatků plynoucích z analýzy exitujících aplikací (popsané v podkapitole 3.1) byly vyvozeny následující požadavky na výslednou aplikaci:

- Aktér – V podkapitole neformální specifikace byli zmíněni primární a sekundární uživatelé. Vzhledem k tomu, že obě skupiny uživatelů si jsou velmi podobní, nebude mezi těmito skupinami dále rozlišováno. Se systémem tedy bude pracovat pouze jediný aktér, kterým je uživatel. Jelikož v aplikaci nebude uvažována registrace ani přihlašování, zůstává uživatel jediným aktérem.
- Požadavky na výkon – Výsledná aplikace by měla být použitelná na obyčejném mobilním telefonu s operačním systémem Android s minimální verzí 4.4. Kromě přístupu k internímu uložišti, mikrofonu a reproduktoru nevyžaduje aplikace žádný specializovaný hardware.
- Funkční požadavky – Podrobněji rozepsány v podkapitole 3.4.
- Požadavky na podporovatelnost – Aplikace bude pracovat samostatně v off-line režimu. Tudíž nebude ke svému provozu vyžadovat připojení k internetu. Co možná nejjednodušší rozšířitelnost o nové vlastnosti je vítaná.
- Požadavky na uživatelské rozhraní – Výstup aplikace bude reprezentován grafickým uživatelským rozhraním, jehož ovládání bude založeno především na intuitivnosti. Za tímto účelem bude v grafickém uživatelském rozhraní využito praktik dashboardů, názorných ikon a jednoduchých akcí. Zároveň pro zajištění dobré čitelnosti veškerého textu bude třeba volit kontrastní pozadí, vhodnou barvu a font písma. Aplikace bude zároveň využívat i zvukové rozhraní, ať už jako vstup (detekce prováděných cviků – zvuky), tak i výstup (verbální předávání příkazů cviků).
- Požadavky na implementaci – využití objektově orientovaného návrhu a jazyka Java.

3.4 Funkční požadavky

Následující funkční požadavky popisují nejdůležitější akce, které musí být v aplikaci pro uživatele k dispozic. Výčet funkcionality neobsahuje veškerou funkcionalitu, ale popisuje pouze stěžejní funkce aplikace. Popis uvádí vstupní předpoklady pro provedení akce, způsob provedení akce a výsledek akce. Popisy budou následně využity ve fázi testování k ověření správné funkcionality aplikace.

Vytvoření/editace skupiny cviků:

- **Vstupy:** Vyplněný název a oblast skupiny. Volba ikony a barvy skupiny. Popis skupiny je nepovinná informace.

- **Zpracování:** Uživatel vyplní údaje o skupině. V případě správného vyplnění a potvrzení vložení/editace dojde k uložení/modifikování informací v interní databázi. V případě nesprávného vyplnění dojde k vyvolání varovné hlášky, která bude vyzývat k opravě údajů.
- **Výstupy:** Vytvoření/editování skupiny cviků. Seznam skupin bude reflektovat vloženou/editovanou skupinu. Data o skupině budou uložena v konzistentním stavu v lokální databázi.

Vytvoření/editace sady drilů:

- **Vstupy:** Existence alespoň jedné skupiny cviků. Vyplnění názvu sady a volba skupiny, pod kterou sada spadá.
- **Zpracování:** Uživatel vyplní údaje o sadě drilů. V případě správného vyplnění a potvrzení vložení/editace dojde k uložení/modifikování informací v interní databázi. V případě nesprávného vyplnění dojde k vyvolání varovné hlášky, která bude vyzývat k opravě údajů.
- **Výstupy:** Vytvoření/editování sady drilů. Seznam sad drilů bude reflektovat vloženou/editovanou sadu. Data o sadě drilů uložena v konzistentním stavu v lokální databázi.

Vložení nového drilu do sady drilů:

- **Vstupy:** Existence sady drilů. Případná existence drilu v databázi předem uložených drilů.
- **Zpracování:** V případě existence drilu v databázi předem uložených drilů bude mít uživatel možnost vybrat ze seznamu požadovaný dril. Pokud neexistuje žádný předem uložený dril nebo se v seznamu předem uložených drilů nenachází uživatelem požadovaný dril, bude uživateli umožněno vytvoření nového drilů.
- **Výstupy:** Dril je vložen do sady drilů. V případě vytvoření nového drilu je možno (volbou uživatele) uložit dril do předem uložených drilů.

Spuštění provádění sady drilů:

- **Vstupy:** Existence alespoň jedné sady drilů.
- **Zpracování:** Na obrazovce editace sady drilů nebo v seznamu sad drilů uživatel volbou tlačítka „play“ spustí provádění sady. V případě prázdné sady drilů (v sadě nejsou umístěny žádné drily) bude uživatel o této skutečnosti upozorněn a bude vyzván k vložení drilů do sady.
- **Výstupy:** V případě splnění všech podmínek pro zahájení provádění sady drilů bude uživatel přesměrován na obrazovku provádění sady drilů. V opačném případě zůstane na aktuální obrazovce.

Provádění sady drilů:

- **Vstupy:** Existence alespoň jedné neprázdné sady drilů. Spuštění provádění sady.
- **Zpracování:** Během provádění sady budou postupně prováděny všechny drily obsažené v sadě. Drily budou prováděny v pořadí, ve které jsou v sadě uloženy nebo v pořadí náhodném (zvolí-li tak uživatel). Při započetí provádění drilu bude uživatel verbálně instruován, jaký dril má aktuálně provést. Následně se bude čekat na zvukový signál, který bude značit dokončení provádění drilu nebo se bude čekat předem určený čas (uživatel dostane prostor k provedení drilu). Po provedení drilu se automaticky přechází na další dril v sadě. O drilech, u kterých se čeká na zvukový signál, je zaznamenáván mezičas provádění. V případě dokončení celé sady drilů a nastavení opakování sady více než jedenkrát, dojde k započetí provádění sady opět od prvního drilu v sadě (volba pro intervalové cvičení). Uživatel má možnost kdykoliv provádění sady pozastavit, či zrušit.
- **Výstupy:** Po úplném dokončení sady drilů (v případě nastavení více iterací, po dokončení všech iterací sady) bude uživatel informován o celkovém čase provádění sady a bude mu nabídnuta možnost opakování sady.

Výběr statistiky pro sadu drilů:

- **Vstupy:** Existence alespoň jedné sady drilů, která byla v minulosti dokončena nebo aktuální dokončení provádění sady drilů.
- **Zpracování:** Dojte k výpisu jednotlivých statistik o provedení sady drilů, nejlepších časech, datech provedení, počtu provedení sady za den/měsíc.
- **Výstupy:** Zobrazení statistik pro konkrétní sadu drilů.

Veškerá data, která uživatel může do aplikace vložit, musí mít možnost i následně editovat a smazat.

Kapitola 4

Metody pro převod textu na řeč

Syntéza řeči je umělé vytvoření lidské řeči. Počítačové programy a knihovny určené k tomuto účelu se nazývají syntezátory řeči, taktéž známé pod výrazem Text-To-Speech (dále jen TTS). TTS je systém, který umožňuje převod psaného textu na mluvený hlas. Obecně lze systémy pro syntézu členit na tři hlavní směry. Systémy založeny na formantech (Formant synthesis), artikulační syntéza (Articulatory synthesis) a syntéza zřetězením (Concatenation synthesis). Systémy, které jednoduše spojují jednotlivá slova nebo části vět, jsou označovány jako Voice Response Systems (dále jen VRS). Tyto systémy se používají převážně tehdy, pokud je požadována jen omezená slovní zásoba (obvykle několik stovek slov). Sestavené věty, které mají být vysloveny, však tvoří jen velmi omezenou strukturu. Typickým příkladem systému VRS je hlášení informací na vlakových nádraží. Z pohledu TTS syntézy jako takové je však nemožné a zbytečné nahrávat a ukládat všechny slova jazyka. Je vhodnější definovat TTS systém jako automatickou reprodukci řeči na základě vztahu grafém $\rightarrow$ foném.[11]

4.1 Základní dělení TTS systémů

Metody založené na syntéze formantů (Formant synthesis)

Syntéza formantů[1] je metoda, která nepoužívá lidské řečové vzory. Místo toho je syntetizovaný hlasový výstup tvořen pomocí aditivní syntézy a akustického modelu. Metoda syntézy formantů je založena na zdroj-filtr modelu. Existují dvě základní struktury, paralelní a kaskádová. Pro lepší výkon se obvykle používá kombinace obou těchto typů. Metoda poskytuje nekonečný počet zvuků, což ji činí flexibilnější než metody založené na zřetězení. Pro vytvoření srozumitelné řeči jsou obecně zapotřebí nejméně tři formanty. Pro kvalitní hlasový výstup postačí pět formantů. [23]

Mnoho systémů založených na syntéze formantů generuje umělý robotický hlas, který nelze zaměnit s přirozeným lidským hlasem. Avšak ne vždy je cílem syntézy maximální přirozenost hlasu. Metody založené na formantech mají oproti jiným metodám obrovskou výhodu ve srozumitelnosti, a to i při vysokých rychlostech řeči. [11]

Zároveň syntezátory založeny na formantech mívají ve většině případů menší velikost než syntezátory založeny na zřetězení. Je to proto, že syntezátory založené na formantech nepracují s databází předem uložených řečových vzorů. [18]

[1] Formanty vznikají rezonancí v dutinách hudebních nástrojů, hlasového ústrojí atd.

Metody založené na řetězení (Concatenation synthesis)

Syntéza založená na zřetězení spojuje dohromady prvky řeči (fonémy, difóny, slabiky, slova nebo věty) extrahované z databáze řečových vzorků. Výsledkem syntézy je umělá řeč, která se nejvíce podobá lidské. Systémy založené na syntéze zřetězením mají velmi omezené znalosti (fonetické) o datech nad kterými pracují. Ve výstupech se mohou objevovat vady, které jsou způsobené rozdílností v amplitudách nebo tónem mezi spojovanými vzorky. Pro vyrovnávání rozdílů mezi amplitudami se využívá technika vyrovnání. Zatímco syntezátor produkuje sekvenci segmentů, které získává z databáze vzorů. Provádí se vhodná zvuková úprava segmentů, aby výsledný tón byl co nejvíce podobný tomu lidskému. [18]

Metody založené na artikulační syntéze (Articulatory synthesis)

Artikulační syntéza je metoda pro syntézu řeči založená na modelu lidského hlasového ústrojí a artikulačním procesu, který v něm probíhá. Artikulační syntéza se snaží co možná nejpřesněji modelovat přirozený proces tvorby řeči. Toho je dosaženo tvorbou syntetického modelu lidské fyziologie a procesu tvorby řeči. Systémy založené na artikulační syntéze zahrnují modul pro generování pohybu hlasového ústrojí (kontrolní model), modul pro konverzi pohybové informace na souvislou posloupnost geometrií hlasového ústrojí (model hlasového ústrojí) a modul pro generování akustických signálů na základě těchto artikulačních informací (akustický model). [23]

4.2 Využití TTS systémů

Syntéza řeči má v dnešním moderním světě stále větší uplatnění. Ať už se jedná o oznamovací nebo varovné systémy, systémy napomáhající nevidomým nebo aplikace pro telekomunikace a multimédia. Implementace takových systému především závisí na způsobu a okruhu použití. V některých případech, jako jsou varovné systémy, není potřeba použití neomezené slovní zásoby, ale bohatě nám postačí jednoduchý slovník nebo databáze předem uložených slov a výrazů. Na druhou stranu systémy pro čtení elektronické pošty nebo systémy napomáhající nevidomým vyžadují sofistikovanější realizaci.

Pravděpodobně nejdůležitější a nejužitečnější oblastí využití syntézy řeči jsou čtecí a komunikační pomůcky pro nevidomé. Přičemž nejdůležitějším faktorem systémů v této oblasti je srozumitelnost. Zároveň slepý člověk nemůže vidět délku vstupního textu, který syntezátor řeči předčítá. Proto systémy pro tento účel by měly poskytnout informace o textu, který má být přečten.

Dalším velmi významným odvětvím použití TTS systémů je u lidí, kteří se narodili se sluchovým či hlasovým postižením. Sluchově postižení mají často problém se správnou výslovností či artikulací. Syntetizovaná řeč dává těmto lidem možnost komunikovat s lidmi, kteří nerozumí znakové řeči.

Syntetizovaná řeč může být použita také v mnoha vzdělávacích situacích. Počítač se syntetizérem může vyučovat 24 hodin denně 365 dní v roce. Může být naprogramován pro speciální úkoly jako je výuka pravopisu a výslovnosti pro různé jazyky.

Nejnovější aplikace syntézy řeči je v oblasti telekomunikace a multimédií. Syntetizovaná řeč byla používaná již značnou dobu ve všech druzích telefonních informačních systémů, avšak kvalita nebyla zdaleka vhodná pro běžné zákazníky. Až s nástupem moderních technologií se kvalita dostala na takovou úroveň, která umožňuje každodenní použití.

Syntéza řeči se stává cennější i pro běžné zákazníky, což činí tyto systémy stále vhodnější pro každodenní použití. Například lepší dostupnost TTS systémů může umožnit zaměstnávat i lidi s komunikačními potížemi nebo napomáhat v každodenních činnostech. [5]

4.3 Volba knihovny TTS

Existuje spousta knihoven, které implementují nebo kombinují některé z metod uvedené v kapitole 4.1. Nás budou však zajímat pouze knihovny vhodné pro aplikaci na platformu Android. Příkladem takové knihovny je AndroidMaryTTS, což je malá open source offline knihovna podporující syntézu řeči na základě zřetězení a HMM (Hidden Markov Model). Bohužel však nepodporuje český jazyk, tudíž je pro náš účel použití nevhodná.

Další rozšířenou knihovnou je eSpeak. Což je open source syntetizátor řeči, který podporuje až 102 různých jazyků (včetně jazyka českého) a akcentů. Knihovna je založena na jádře eSpeak využívající syntézu založenou na formantech v kombinaci s Klatt[2] syntézou. To umožňuje podporu mnoha jazyků při zachování malé velikosti knihovny. Syntezátor umožňuje generování čisté a rychlé řeči, avšak díky využité metodě není výsledná řeč tak přirozená.

Google Text-to-Speech je syntetizační knihovna založena na softwaru WaveNet [3], což je software vyvinutý v DeepMind, dceřiné společnosti společnosti Google. Knihovna podporuje velké množství jazyků. Některé jsou k dispozici již offline, jiné jsou stále ještě překládány vzdáleně. Český jazyk je od druhé poloviny roku 2017 dostupný i v offline režimu.

Hlavním požadavkem na knihovnu je podpora českého jazyka. Dalším důležitým požadavkem je jednoduché zakomponování do celé aplikace (požadavek na jednoduchou integraci). S tím souvisí i možnost pracovat s knihovnou offline (alespoň v základní formě), aby nebylo pro práci s knihovnou vyžadováno připojení k internetu. Tato kritéria splňuje knihovna eSpeak a Google Text-to-Speech. Nakonec jsem se rozhodl pro knihovnu Google Text-to-Speech, neboť je přímo uzpůsobena pro platformu Android, umožňuje jednodušší integraci do aplikace, syntéza poskytuje čistější a přirozenější výstup a má přímou podporu ze strany vývojáře operačního systému Android.

[2]Klatt syntezátor je software pro syntézu řeči navržený Dennisem Klattem v roce 1980. Jedná se o jeden z nejvydařenějších nástrojů pro práci se syntézou řeči.[3]

[3]Technologie WaveNet dokáže vygenerovat řeč, která napodobuje jakýkoliv lidský hlas. V současnosti se jedná o nejdokonalejším systémem TTS. WaveNet funguje na principu strojového učení ve spolupráci s řetězením.

Kapitola 5

Proces rozpoznávání zvuku

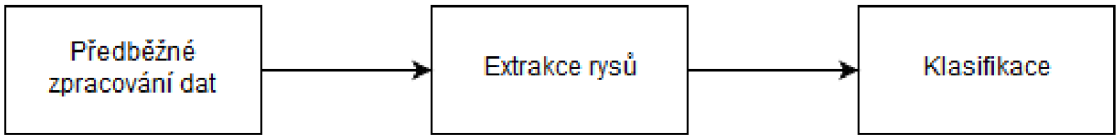
Zvuk je bohatým zdrojem informací, díky kterým je možno zasadit člověka do kontextu každodenního života. Téměř každý tvor produkuje charakteristický zvukový vzor, který je nejčastěji spojen s určitou činností (např. chůze, jezení, mluva, ...). Ovšem nejen tvorové jsou zdrojem zvuku. I prostředí, ve kterém se vyskytujeme, vydává své specifické zvuky (např. zvuk auta, šumění vody, zavírání dveří, ...). Tyto informace mohou být využity pro detekci určitého prostředí, činnosti nebo v případě detekce a rozpoznání mluveného slova i k porozumění lidské řeči. V následující kapitole si uvedeme několik způsobů zpracování zvukových signálů a technik, díky kterým lze detekci, rozpoznání a klasifikaci provést.

Většina problémů spojených s rozpoznáváním a klasifikací je realizována pomocí procesu, který se skládá ze tří fází:[8]

1. **Předběžné zpracování dat** – první fáze, ve které dochází k prvotnímu zpracování zvukových dat a jejich transformaci do vhodné podoby pro extrakci. Více v kapitole 5.1.

2. **Extrakce vlastností** – druhá fáze, kde dochází k získání významných vzorků (rysů) informací ze zvukových dat. Více v kapitole 5.2.

3. **Klasifikace** – poslední fáze, provádí rozčlenění položek do kategorií na základě porovnání rysů. Více v kapitole 5.3.



Obrázek 5.1: Fáze procesu rozpoznávání zvuku.

5.1 Předběžné zpracování dat

Předběžné zpracování dat je první fází procesu zpracování zvuku. Tento krok se liší v závislosti na prováděné klasifikační úloze. Předzpracování dat pro rozpoznávání zvuku (včetně rozpoznávání řeči) vyžaduje nejprve záznam zvuku a jeho následné načtení do počítače.

Tento krok se nejčastěji provádí pomocí mikrofonu nebo mikrofonového pole. Dalším krokem této fáze je převést zvuk z analogového do digitálního formátu pomocí technik vzorkování a kvantizace. Dále následují úpravy jako je např. Fourierova transformace, normalizace a framing, díky kterým se zvukový signál připraví do ideálního stavu pro extrakci rysů.[8]

5.1.1 Digitalizace zvuku

Zvuková data jsou senzorem (mikrofonem) získávány ve spojité (analogové) podobě. Pro reprezentaci zvukových dat v počítači je nutné tato spojitá vstupní data převést do podoby digitální, která je vhodná pro následné zpracování počítačem. Převod spojitého (analogového) signálu na nespojitý (diskrétní) digitální signál se skládá ze dvou fází. Nejprve se provede vzorkování signálu, a potom následuje kvantování.[20]

Vzorkování se provede tak, že rozdělíme vodorovnou osu signálu na rovnoměrné úseky a z každého úseku vezmeme jeden vzorek (na obrázku 5.2 (vlevo) jsou tyto vzorky znázorněny červenými body). Je přitom zřejmé, že tak z původního signálu ztratíme mnoho detailů, protože namísto spojité čáry dostaneme pouze množinu diskrétních bodů s intervalem odpovídajícím použité vzorkovací frekvenci.[20]

Během vzorkování je nutno dodržet Shannonův teorém:

$$f_s > 2f_{max} \tag{5.1}$$

kde f_s je vzorkovací frekvence a f_{max} je nejvyšší frekvence v signálu. Při dodržení Shannonova teorému je reprezentace úplná v tom smyslu, že je možné zrekonstruovat signál před navzorkováním. Nedodržení teorému vede k jevu zvanému aliasing. Aby se tomuto jevu předešlo, provádí se ořezání frekvencí vyšších, než které neodpovídají Shannonovu teorému.

Vzhledem k tomu, že počítače umí vyjádřit čísla pouze s omezenou přesností, je třeba navzorkované hodnoty upravit i na svislé ose. Tato metoda se nazývá kvantování, protože hodnoty vzorku vyjadřujeme v určitých kvantech. Aby bylo možno určit, jakých hodnot má po kvantování nabývat určitý vzorek, je třeba rozdělit prostor kolem jednotlivých hodnot na toleranční pásy (jeden takový pás je vyjádřen na obrázku 5.2 (vpravo) žlutým pásem kolem hodnoty 0). Kterémukoliv vzorku, který padne do daného tolerančního pásu, je při kvantování přiřazena daná hodnota (na obrázku 5.2 (vpravo) jsou tyto kvantované hodnoty znázorněny zelenými body).[20]

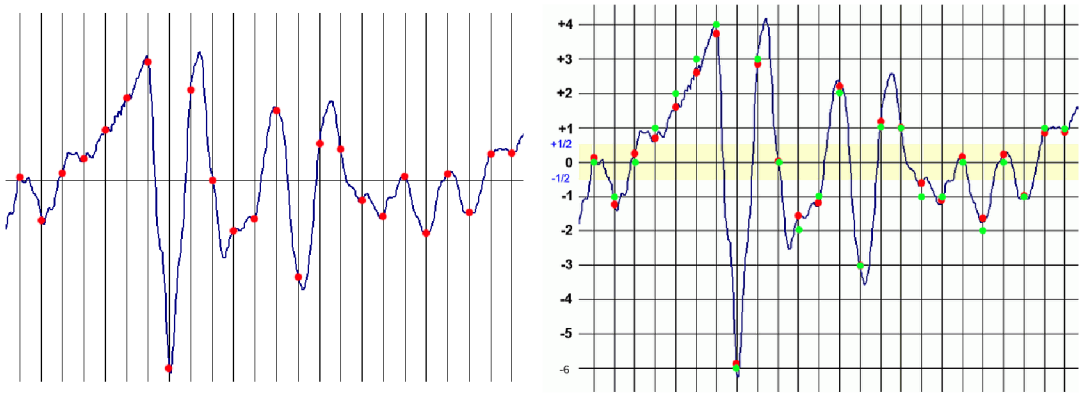
5.1.2 Framing a normalizace

Kvalita zvuku přijatá z mikrofonu je ovlivněna vzdáleností mikrofonu od zdroje zvuku, vzorkovací frekvencí a kvantováním. Proto ještě před rozdělením zvuku na framy (rámce) je důležité provést normalizaci zvukového signálu. Normalizace zajistí vyvážení frevenčního spektra, kdy vysoké frekvence mají obvykle menší hodnoty ve srovnání s nižšími frekvencemi. Zlepší redukci šumu a úpravou původního signálu se také vyhneme výpočetním problémům u Fourierovy transformace. Normalizace původního zvukového signálu se provádí následujícím vzorcem:

$$y(t) = x(t) - \alpha x(t-1) \tag{5.2}$$

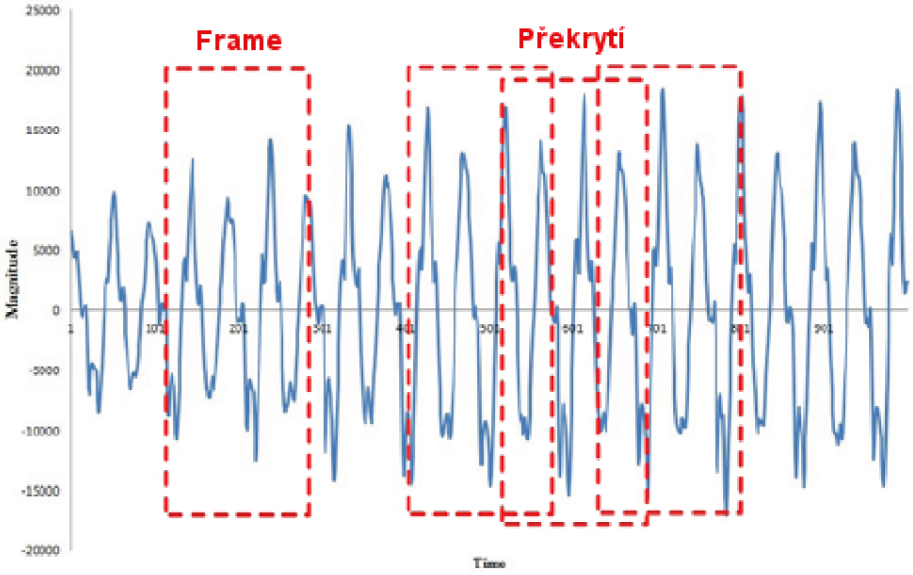
kde $x(t)$ představuje původní vzorek signálu v časové doméně a α značí filtrační koeficient. Hodnota koeficientu bývá zpravidla v intervalu od 0.95 do 0.97.[7]

Normalizovaný signál je dále zpracován metodou tzv. framing. Jedná se o metodu, která vytvoří z dlouhých částí zvukového signálu mnoho krátkých rámců. Parametrem rámce je



Obrázek 5.2: Vzorkovaný spojitý signál (vlevo) a kvantovaný vzorkovaný signál (vpravo). Převzato z (http://www.mp3s.asp2.cz/img/vzorkovani.png) a (http://www.mp3s.asp2.cz/img/kvantovani.png)

velikost rámce (vyjádřeno v milisekundách) a překrytí (vyjádřeno v procentech nebo milisekundách). Velikost rámce udává, jak velké rámce budou tvořeny. Překrytí udává velikost překryvu jednotlivých rámců mezi sebou. Překrytí se používá z důvodu, aby se zabránilo nesrovnalostem při extrakci. Princip překrývání je uveden na obrázku 5.3.[7]



Obrázek 5.3: Demonstrace rámců a překrytí (signály se v rámcích překrývají z jedné třetiny).

5.1.3 Fourierova transformace

Zvukový signál může být upravován různými způsoby, čímž se odhalují různé informace. Jednou z nejdůležitějších úprav je Fourierova transformace. Reprezentací zvukových vzorů

ve formě Fourierovy řady značně zjednodušíme jinak složité numerické výpočty. Proces Fourierovy transformace provádí rozbití komplexní vlny na její charakteristické sinusové složky, což umožňuje snadnou analýzu vlny. Jakýkoliv ve své podstatě složitý signál, jako je např. lidská řeč nebo hudba je možno popsat jako součet různých frekvencí, právě díky Fourierově transformaci. Z Fourierovy transformace tak získáváme spektrum analyzovaného signálu. Fourierova transformace umožňuje signálu (tj. funkci) získat amplitudu pro danou frekvenci. Aplikací různých frekvencí na vlnu, ji tak lze rozdělit na množinu diskrétních frekvenčních komponent. Takto získané hodnoty lze poté snadno analyzovat. Fourierova transformace je tedy zobrazení, které každému signálu (funkci) přiřazuje jinou funkci, z jejichž vlastností lze vyčíst informace o původním signálu.[8]

Existuje mnoho dalších metod, jako je Laplaceova transformace, rychlá Fourierova transformace, diskrétní Fourierova transformace apod., jejichž transformace mohou být použity k analýze zvukového signálu ve frekvenční i časové oblasti. Avšak všechny tyto metody vycházejí ze základu Fourierovy transformace.[8]

5.2 Extrakce vlastností

Extrakce vlastností se provádí proto, aby se zmenšila obrovská množina dat vytvořená ve fázi předzpracování dat. Extrakce zahrnuje výběr prvků vstupních dat, které jednoznačně reprezentují nesenou informaci. Tento krok se opět liší v závislosti na prováděné klasifikační úloze. Pro rozpoznávání zvuku je pro extrakci vlastností používáno nepřeberné množství technik, od jednoduchých (identifikace všech frekvencí a zvuků) až po extravagantní (modelování extrakce vlastností pomocí lidského sluchového systému). Nicméně, bez ohledu na zvolenou metodu, extrakce vlastností je nejobtížnější fází procesu rozpoznávání.[8]

Extrakci lze obecně rozdělit na dva typy: stacionární (frekvenční) extrakci a nestacionární (časově-frekvenční) extrakci. Stacionární extrakce vlastností poskytuje globální výsledek, který se zaměřuje na frekvence obsažené v celém signálu. Při stacionárním extrahování vlastností se nerozlišuje, kde se tyto frekvence vyskytovaly. Naproti tomu extrakce nestacionárních vlastností rozděluje signál na jednotky s diskrétním časem. Lze tak jednotlivé frekvence identifikovat v určitých oblastech signálu, což zároveň napomáhá porozumět analyzovanému signálu.[9]

5.2.1 Stacionární extrakce vlastností

Pro stacionární extrakci vlastností, rozpoznávání řeči, hudebních nástrojů a zvuků se spoléhá pouze na několik technik (každá s několika různými variantami). Významnými zástupci těchto technik jsou například Frequency extraction, Mel frequency cepstral coefficients, Linear prediction cepstral coefficients nebo Bark frequency cepstral coefficients. Všechny tyto techniky si dokáží poradit s rozpoznáváním řeči a některé z nich jsou vhodné i pro rozpoznáváním zvuků.[9]

Je třeba poznamenat, že zatímco Frequency extraction je čistě stacionární technikou, jiné techniky využívající kepstrální koeficienty[1] mohou být považovány na pseudo-stacionární techniky, protože rozdělují signál na časové řezy. V tomto případě se nejedná o techniky přímo časově-frekvenční, protože každý časový řez musí být proveden v kontextu s jiným časovým řezem v pořadí. Udržování kontextu mezi řezy je důležité z důvodu získání informací.[9]

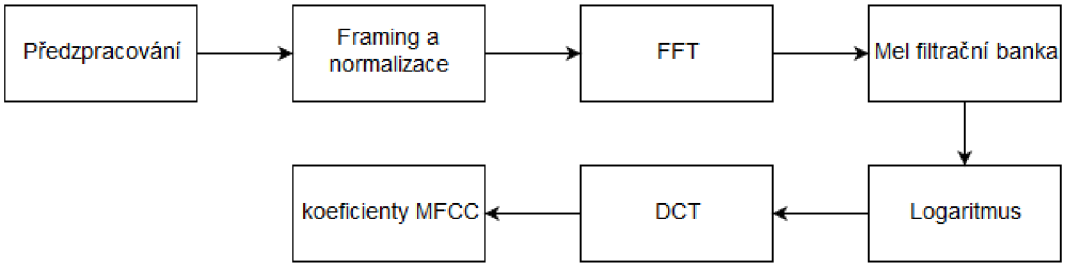
[1] Kepstrum je výsledek inverzní Fourierovy transformace logaritmu spektra signálu.

Mel Frequency Cepstrum (MFC)

Mel-Freqeuncy Cepstral Coefficients (dále jen MFCC) jsou koeficienty, které dohromady tvoří MFC. MFCC popisují kepstrum signálu za použití Mel-frekvenční škály. Tento parametr je velmi často používán v oblastech rozpoznávání řeči a zvuku. Obecný postup extrakce mel-kepstra zahrnuje rozdělení signálu na rámce, získání energie spektra, transformace mel-spektra a nakonec použití Discrete Cosines Transform (dále jen DCT) pro získání koeficientů kepstra.[17]

Postup výpočtu MFCC:[16]

- Rozdělení signálu do rámců.
- Pro každý rámec je vypočítán periodogram odhadu energie spektra.
- Aplikace Mel filtrační banky na energii spektra a součet energií pro každý filtr.
- Výpočet logaritmu všech energií z filtrační banky.
- Výpočet DCT z logaritmu energií.
- Získání koeficientů.



Obrázek 5.4: Obecná metodologie extrakce MFCC.

Jelikož se zvukový signál neustále mění, tak pro zjednodušení jeho zpracování se provádí framing na 20-40 milisekundové rámce. Velikost rámců je třeba pečlivě volit, neboť v případě, že by rámce byly příliš krátké, nebudeme mít dostatek vzorků pro získání spolehlivého spektrálního odhadu. Naopak při dlouhých rámcích se signál v rámci příliš změní. Dalším krokem zpracování je výpočet energie spektra každého snímku. Pomoci spektrálního odhadu periodogramu je provedena identifikace zastoupených frekvencí uvnitř rámce. Avšak spektrální odhad periodogramu stále obsahuje mnoho informací, které nejsou vyžadovány pro zpracování zvuku. Z toho důvodu se provádí shromažďování shluků periodogramů, které sumarizujeme, abychom získali představu o tom, kolik energie existuje v okolí 0 Hertzů. Vzhledem k tomu, že frekvence se zvyšují, budou se zvyšovat i naše filtry. Následně nám škála Mel napoví, jak rozmístit filtrační banku a jak má být široký její záběr.[16]

Škála Mel se vztahuje k vnímané frekvenci nebo intenzitě čistého tónu v měřené frekvenci. Lidé jsou mnohem vnímavější k malým změnám intenzity v nízkých frekvencích než ve vysokých frekvencích. Začlenění této skály tak přibližuje rysy blíže tomu, co lidé opravdu slyší.[16]

Vzorec pro převod frekvence na stupnici Mel:

$$M(f) = 1125ln(1 + f/700) \tag{5.3}$$

Vzorec pro zpětný převod z Mel stupnice na frekvenci:

$$M^{-1}(m) = 700(exp(m/1125) - 1) \tag{5.4}$$

Jakmile získáme energii rámce, provedeme výpočet jejího logaritmu. Tento krok je motivován lidským sluchovým ústrojím, neboť lidský sluch neslyší hlasitost v lineárním měřítku. Obecně, abychom zdvojnásobili hlasitost zvuku, musíme do něj dát 8x více energie. Tato kompresní operace činí algoritmus lépe sladěný s tím, co lidé skutečně slyší.[16]

Následuje výpočet DCT, který je proveden hned z několika důvodů. Vzhledem k tomu, že se filtry ve filtrační bance vzájemně překrývají, jsou energie vzájemně částečně korelované. DCT nám tak zajišťuje dekorelaci. Zároveň z DCT zachováváme pouze 12-13 z 26 koeficientů. Důvodem je, že vyšší DCT koeficienty reprezentují rychlé změny v energiích a zhoršují výsledky následující klasifikace.[16]

Obecnou nevýhodou MFCC je, že hodnoty MFCC nejsou příliš robustní vůči šumu, proto se běžně používá normalizace hodnot, čímž se vliv šumu snižuje.[16]

Delta a Delta-delta

MFCC jsou však statické koeficienty, které popisují pouze energii spektra signálu v rámci. Rozšíření Delta zavádí časovou složku do výpočtu. Delta se počítá z MFCC koeficientů. Mezi jednotlivými rámci jsou vypočteny rozdíly, přičemž lze zvolit, kolik rámců se má takto provázat. Následující rovnice se používá pro výpočet Delta koeficientů:

$$\Delta C_m(t) = C_m(t + d) - C_m(t - d) \tag{3}$$

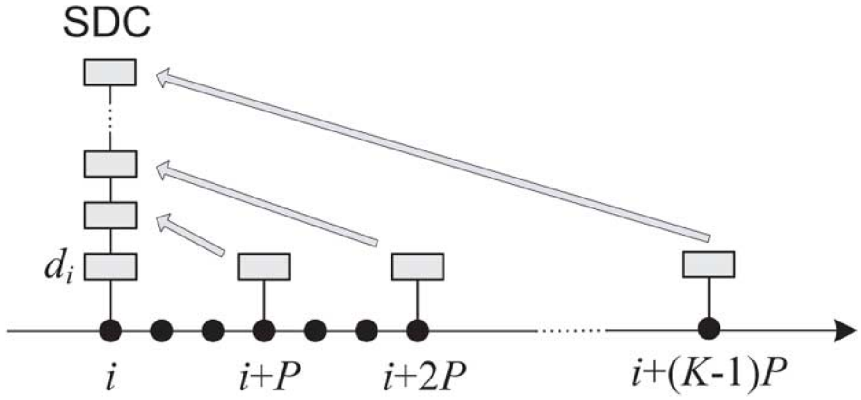
Tato metoda jednoduše odečítá m-tý kepstrální koeficienty rámců bezprostředně před a bezprostředně za aktuálním rámcem. Proměnná d představuje časové zpoždění mezi jednotlivými rámci. Počet Delta koeficientů zpravidla odpovídá počtu koeficientu v MFCC. Obdobnou metodou je Delta-delta, která využívá stejný vzorec. Rozdílem je, že vzorec je namísto MFCC koeficientů aplikován na Delta koeficienty. Delta a Delta-delta koeficienty jsou také někdy nazývány jako diferenciální a akcelerační koeficienty. Obě modifikace jsou využívány v kombinaci s MFCC koeficienty pro lepší a přesnější klasifikaci.[24]

Shifted Delta Cepstrum (SDC)

Shifted Delta Cepstrum (dále jen SDC) je další možné rozšíření MFCC koeficientů. Jedná se poměrně o jednoduché rozšíření vycházející přímo z MFCC koeficientů a využívající informací plynoucí z časové roviny. Jde v podstatě o výpočet Delta koeficientu a jejich provázání. Časové informace se pak získávají konkatenací několika následujících delta rámců. Takto lze získat informaci o změnách aktuálního rámce v čase. Tato modifikace zlepšuje schopnost klasifikace a oproti prostým Delta koeficientům a je vhodnější při klasifikaci zvukových signatur. SDC je specifikováno čtyřmi parametry (N, d, P, k).

$$\Delta C(t, i) = C(t + iP + d) - C(t + iP - d) \tag{5.5}$$

SDC vektor rysů je tvořen zřetězením delta rámců, kde d udává delta zpoždění rámce, N značí počet koeficientů v rámci, P určuje časový posun mezi po sobě následujícími rámci a K představuje, kolik delta rámců je v jednom bloku.[24]



Obrázek 5.5: Diagram zobrazující princip SDC. d_i představuje delta rámec i. Obrázek převzat z publikace [24].

5.2.2 Nestacionární extrakce vlastností

Mezi techniky založené na nestacionární extrakci se řadí Short-time Fourier transform (STFT), Fast (discrete) wavelet transform (FWT) nebo Continuous wavelet transform (CWT). Všechny uvedené metody využívají nejrůznějších algoritmů pro vytvoření časově-frekvenční reprezentace signálu. Zatímco STFT používá standardní Fourierovou transformaci nad několika rámci, metody založené na technice vlnek (walvelet) používají matematickou funkci, která dokáže řešit problémy s rozlišením, které se právě u metod STFT vyskytují.[9]

Continuous Wavelet Transform (CWT)

Pro doplnění uvádím jednu z nejrozšířenějších nestacionárních metod, i když v rámci této práce není tato metoda realizována. Continuous wavelet transform (CWT) je implementace vlnkové transformace pomocí libovolné váhy a téměř libovolných vlnek. Používané vlnky nejsou ortogonální a data získaná touto transformací jsou tak vysoce korelovaná. Pro diskrétní časové řady můžeme využít této transformace s omezením, že nejmenší vlnková transformace musí být rovna vzorkovací frekvenci. Tato modifikace je někdy nazývána Discrete Time Continuous Wavelet Transform (DT-CWT) a jde většinou o používaný způsob výpočtu v reálných aplikacích.[15]

V principu probíhá výpočet CWT přímým použitím definice vlnkové transformace, např. vypočítáme konvoluci signálu s váhou vlnek. Pro každou váhu tak získáme pole o stejné délce N jako má signál. Pomocí M libovolně zvolených vah získáme pole NxM, které přímo představuje časově-frekvenční rovinu. Algoritmus použitý pro tento výpočet může být založen na přímé konvoluci nebo na konvoluci založené na násobení ve Fourierově prostoru (tento přístup je někdy nazýván jako Fast wavelet transform (dále jen FWT)).[15]

Výběr vlnky, která se používá pro časově-frekvenční rozklad je jednou z nejdůležitějších věcí ve výpočtu. Tímto výběrem můžeme ovlivnit rozlišení času a frekvence výsledku. Nemůžeme tím však změnit hlavní vlastnosti transformace (nízké frekvence mají dobré frekvenční a špatné časové rozlišení naopak vysoké frekvence mají dobré časové a špatné frekvenční rozlišení), ale můžeme zvýšit celkovou frekvenci celkového časového rozlišení. To je přímo úměrné šířce vlnek v reálném a Fourierově prostoru. Použijeme-li například vlnku Morlet, můžeme očekávat vysoké frekvenční rozlišení, protože vlnka je dobře zaměřená na frekvence.

Naopak pomocí vlnky DOG (Derivative of Gaussian) získáme dobré rozlišení časové, ale špatné frekvenční.[15]

Navzdory svému běžnému použití pro kódování řeči a zvuků je metoda FTW úspěšně používána i pro rozpoznávací úlohy. Avšak díky své dlouho trvající transformaci pro signály, které svou délkou jsou typické pro enviromentální prostředí, není tato technika vhodná pro aplikaci v reálném čase. Z uvedených vlnkových technik se FTW obvykle používá pro kódování a dekódování signálů, zatímco CWT se používá pro rozpoznávací úlohy.[8]

5.3 Klasifikace

Klasifikace je třetí fází rozpoznávání, která zahrnuje získání relevantních rysů, vytvořených ve fázi extrakce a jejich propojení do jednotlivých klasifikačních tříd (formou rozpoznávání vzorů). Opět tuto fázi lze provádět mnoha způsoby. Pro rozpoznávání zvuku jsou používány mnohé techniky jako je Hidden Markov Models, Support Vector Machines, Gaussian Mixture Models, neuronové sítě nebo Data Reference Model (používané spolu s algoritmem Dynamic Time Warping). Všechny tyto techniky jsou založeny na paradigmatu trénování/testování. Trénování dodává systému konkrétní příklady položek, čímž se systém učí rozpoznávat obecné charakteristiky jednotlivých položek. Následné testování dokáže ověřit, zda se systém úspěšně naučil jednotlivé položky rozpoznávat.[8]

Jelikož existuje velké množství různých klasifikačních algoritmů, uvedeme si dvě nejpoužívanější metody, které jsou navíc efektivní převážně pro rozpoznávání zvuků (nejen hlasu).

5.3.1 Support Vector Machine (SVM)

Support Vector Machine (dále jen SVM) je klasifikátor založen na strojovém učení s učitelem umožňující jak lineární, tak i nelineární klasifikaci. Hlavním cílem klasifikátoru je nalézt nadrovinu, která rozděluje N-dimenzionální prostor rysů tak, že rysy patřící do odlišných tříd leží na opačných stranách poloprostorů. Ve dvourozměrném prostoru je nadrovinou čára dělící rovinu na dvě části, kde každá část reprezentuje jednu třídu.[12]

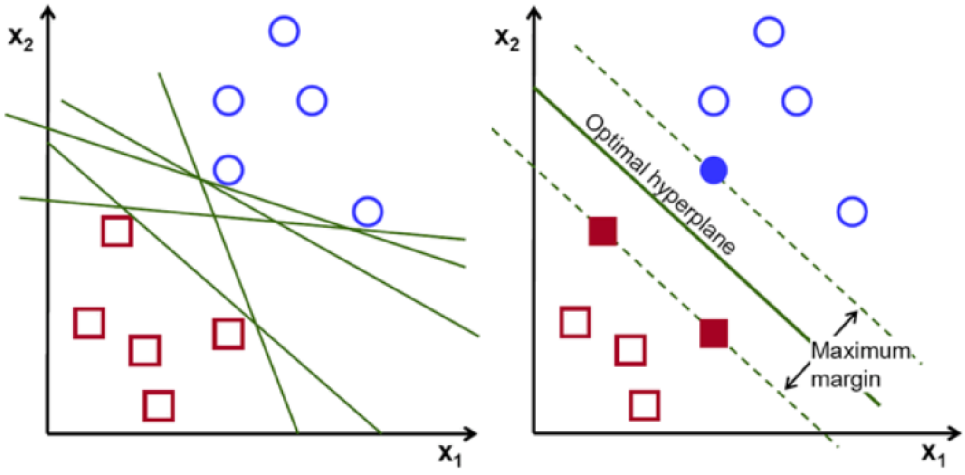
Pokud chceme oddělit dvě třídy datových bodů, existuje obrovské množství nadrovin, pomocí kterých to lze učinit. Cílem je tak nalézt nadrovinu, která dosahuje maximální vzdálenosti mezi datovými body tzv. support vectors obou tříd (viz. obrázek 5.6).

Support vectors jsou datové body, které jsou umístěny blíže k nadrovině a ovlivňují její polohu a orientaci. Důvodem maximalizace vzdálenosti mezi support vectors je dosažení větší jistoty při klasifikaci datových bodů. Pokud bychom měli data jako jsou na obrázku 5.7, tak je zřejmé, že neexistuje žádná linie, která by mohla body v rovině x, y oddělit do dvou tříd.[19]

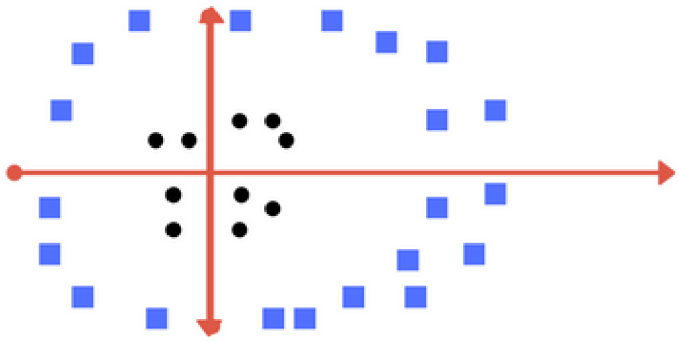
Proto se provede transformace přidáním ještě jedné dimenze, kterou nazveme osou z. Předpokládejme hodnotu bodů v rovině z jako $w = x^2 + y^2$. V tom případě s body můžeme manipulovat jako vzdálenost bodu od počátku z. Nyní, pokud body vykreslíme v ose z, zjistíme, že lze body jednoduše separovat.[19]

Když následně převedeme toto oddělení zpět do původní roviny, provede se mapování linie do kruhové hranice, jak je znázorněno na obrázku 5.8 dole. Tato transformace se nazývá kernel.[19]

Mezi obecné výhody SVM klasifikátoru patří, že je efektivní ve velkých dimenzionálních prostorech. Dále je efektivní v případech, kdy počet dimenzí je větší, než počet vzorků v trénovací množině.[12]



Obrázek 5.6: Datové body lze oddělit mnoha různými nadrovinami (vlevo). Volba optimální nadroviny, která vytváří maximální vzdálenost mezi support vectors body (vpravo).



Obrázek 5.7: Data, která nelze jednoduše lineárně separovat.

Avšak hlavní výhodou SVM je schopnost klasifikovat data v nelineárním prostoru díky tzv. kernel funkcím:

$$f(\mathrm{x}) = \sum_{i=1}^{N_{sv}} \alpha_i y_i K(\mathrm{x}, \mathrm{x_i}) + b \tag{5.6}$$

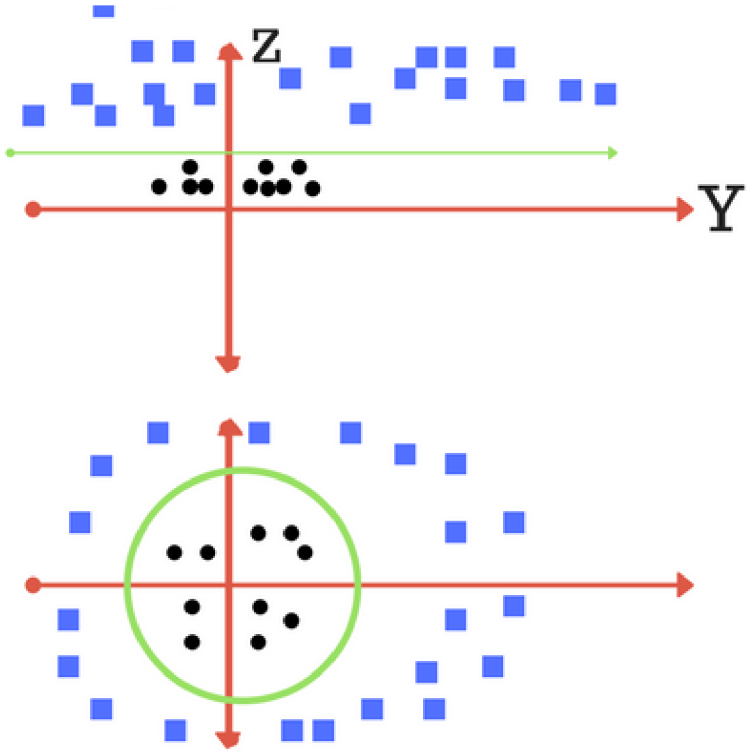
kde x_i jsou podpůrné vektory (support vectors) vybrané z trénovacích dat a $y_i \in \{1, +1\}$ určuje přiřazení do patřičné třídy. $K(x, x_i)$ je kernel funkce, která musí splňovat jisté podmínky:

$$K(\mathrm{x}, \mathrm{y}) = \Phi(\mathrm{x})^t \Phi(\mathrm{y}) \tag{5.7}$$

kde Φ je mapování ze vstupního prostoru do možného N-dimenzionálního prostoru, kde N může být nekonečno. Kernel funkce se používají pro výpočet oddělovací nadroviny ve vyšší dimenzi.[22]

Mezi často používané kernel funkce patří například:[19]

- Polynomiální kernel: $K(x, x_i) = 1 + \sum(x \cdot x_i)^d$



Obrázek 5.8: Přidání dimenze (nahoře). Separování dat a zpětná transformace do původní roviny (dole).

- Exponenciální kernel: $K(x, x_i) = exp(-\gamma \sum(x - x_i^2)$

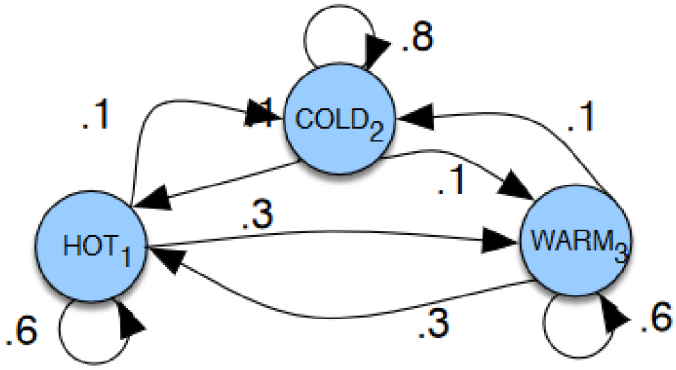
Metoda SVM má však i nevýhodu, kterou je velmi obtížné učení, které je silně komplikováno hledáním vysokého počtu vah a vektorů v mnohadimenzionálním prostoru. [12]

5.3.2 Hidden Markov Models (HMM)

Hidden Markov Models (dále jen HMM) je založen na rozšíření Markovova řetězce. Markovův řetězec neboli model Markovova řetězce nám říká něco o pravděpodobnostech sekvencí náhodných proměnných stavů, z nichž každá může nabývat hodnot z určité množiny. Tyto množiny mohou tvořit slova, značky nebo symboly, které reprezentují libovolné třídy. Markovův řetězec tak vytváří silný předpoklad pro předpovídání budoucnosti pouze na základě aktuálního stavu. Stavy, které předcházely aktuálnímu stavu tak nemají vliv na předpovídání stavů budoucích.[10]

Na obrázku 5.9 je znázorněn Markovův řetezec pravděpodobnosti sledu událostí počasí, které se skládají ze stavů HOT, COLD a WARM. Stavy jsou reprezentovány jako uzly v grafu a přechody s pravděpodobností jako hrany. Hodnoty přechodů opouštějící daný stav se musí rovnat 1.[10]

Markovův řetězec je užitečný, pokud potřebujeme vypočítat pravděpodobnost sledu pozorovatelných událostí. Ve většině případů jsou však události, které nás zajímají, skryté: nepozorujeme je přímo. Například v textu obvykle nepozorujeme zkratky slov. Spíše vidíme slova a slovní spojení, ze kterých musíme jejich zkratky odvodit. Tyto zkratky označujeme



Obrázek 5.9: Markovův model pro počasí zobrazující stavy a přechody mezi nimi. Startovní distribuce π, kde π=[0.1, 0.7, 0.2] představuje pravděpodobnost, že s pravděpodobností 0.1 se bude začínat ve stavu 1 (HOT), 0.7 ve stavu 2 (COLD) a 0.2 ve stavu (WARM).

jako skryté, protože je nelze vypozorovat. HMM nám umožňuje vidět obě události (slova, která vidíme na vstupu) a skryté události (zkratky slov), které považujeme za kauzální faktory v našem pravděpodobnostním modelu.[10]

HMM se skládá z následujících komponent:

- $Q = q_1, q_2, \cdots, q_N$ – množina N stavů.
- $A = q_{11}, \cdots, a_{ij} \cdots a_{NN}$ – A reprezentuje matici pravděpodobnosti přechodu, každá hodnota a_{ij} představuje pravděpodobnost přechodu ze stavu i do stavu j a zároveň platí $\sum_{j=1}^{N} a_{ij} = 1, \forall i$.
- $O = o_1, o_2, \cdots, o_T$ – posloupnost pozorování T, z nichž každý vychází ze slovníku $V = v_1, v_2, \cdots v_V$
- $B = b_i(o_t)$ – posloupnost pravděpodobnosti pozorování, z nichž každá vyjadřuje pravděpodobnost, že pozorování o_t vychází ze stavu i.
- $\pi_1, \pi_2, \cdots, \pi_N$ – počáteční rozdělení pravděpodobnosti stavů. π_i značí pravděpodobnost, že Markovův řetězec bude začínat právě ve stavu i. Stavy, které mají hodnotu π rovnu nule nemohou být počátečními stavy řetězce.

HMM prvního řádu představuje dva předpoklady. Za prvé, stejně jako u Markovova řetězce prvního řádu, závisí pravděpodobnost určitého stavu pouze na bezprostředním předchozím stavu:

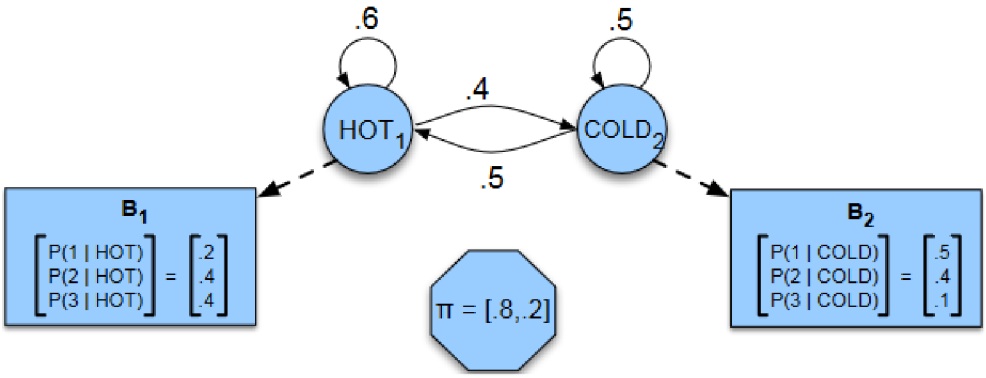
$$P(q_i|q_1 \cdots q_{i-1}) = P(q_i|q_{i-1}) \tag{5.8}$$

Za druhé, pravděpodobnost výstupního pozorování o_i závisí pouze na stavu, který produkoval pozorování q_i a na žádném jiném stavu, ani na žádném jiném pozorování:

$$P(o_i|q_1 \cdots q_i, \cdots, q_T, o_1, \cdots, o_i, \cdots, o_T) = P(o_i|q_i) \tag{5.9}$$

Pro znázornění modelu použijeme příklad se zmrzlinou, který popsal Jason Eisner (2002). Představte si, že jste klimatolog a studujete historii globálního oteplování. Nemůžete však najít žádné záznamy o počasí v létě roku 2020 v Baltimore, Maryland. Avšak najdete

deník Jasona Eisnera, který uvádí, kolik zmrzliny Jason snědl každý letní den. Naším cílem je využít tyto pozorování k odhadu teploty v každý z letních dnů. Úkol zjednodušíme tím, že budeme předpokládat, že existují pouze dva typy dnů: studený (C) a teplý (H). Úkol tedy zní následovně: vzhledem k sekvenci pozorování O (celé číslo reprezentuje počet snědených zmrzlin v daný den) chceme najít skrytou sekvenci Q o stavu počasí (H nebo C), které způsobily, že Jason jedl zmrzlinu.[10]



Obrázek 5.10: Příklad HMM pro úlohu zmrzliny. Dva skryté stavy (H a C) odpovídají teplému (HOT) a chladnému (COLD) počasí a pozorování (z abecedy O=1, 2, 3) odpovídají počtu zmrzlin snědených Jasonem v daný den.

Kapitola 6

Návrh aplikace

V následující kapitole se budu podrobněji zabývat návrhem grafického uživatelského rozhraní mobilní aplikace. Dále popíši způsob ukládání dat v rámci aplikace a zaměřím se na výběr algoritmu pro detekci a rozpoznání zvuků.

6.1 Uživatelské rozhraní

6.1.1 Návrh matice objektů a akcí

Před samotným grafickým návrhem jednotlivých obrazovek aplikace bylo nutné nejprve stanovit, kolik řádově takových obrazovek má být a jaké akce by měly poskytovat. Pro tento

	Operace se skupinami	**Operace se sadami**	**Operace s drily**	**Rychlá editace drilů**	**Spuštění sady**	**Časy a mezičasy provádění sady**	**Statistiky**	**Operace se zvuky**
Přehled skupin	×	×						
Přehled sad drilů		×			×			
Obrazovka drilů ve cvičební sadě		×	×	×	×			
Provádění sady					×	×	×	
Předuložené drily			×	×				
Přehled statistiky						×	×	
Správa zvuků								×

Tabulka 6.1: Matice zobrazuje akce pro základní obrazovky aplikace.

účel jsem využil model matice, která zobrazuje vztah mezi objekty a akcemi. Jinak řečeno, matice zobrazuje hlavní obrazovky aplikace a určuje, jaké akce se mohou v rámci obrazovky provádět. Na základě tohoto modelu lze určit, zda výsledné rozhraní bude pro uživatele lehké či těžké k naučení.

Vytvoření matice objektů/akcí nám umožní vizualizovat, jak jednoduchá či složitá je naše aplikace. Přičemž v našem případě objekt reprezentuje obrazovku aplikace. Velká matice udává, že existuje více konceptů k naučení. Vysoká matice poukazuje na mnoho objektů v aplikaci a široká matice na mnoho akcí k naučení. Matice nám také ukazuje, jak konzistentní či nekonzistentní je náš model. Neboli, jak snadné je pro uživatele přenést to, co se naučili v jedné části aplikace do druhé. Čím má matice větší hustotu, tím snadnější bude aplikace pro uživatele k naučení. [13]

6.1.2 Návrh obrazovek a barevné rozvržení

Při navrhování grafických komponent a jejich umístění je třeba dbát na přívětivost a intuitivnost uživatelského rozhraní. Komponenty jsou proto na obrazovky umístěny tak, aby uživatele naváděly k akci, kterou poskytují. Se samotným rozvržením komponent souvisí i barevné rozložení. Základní schéma je voleno jako tmavý obsah na světlém pozadí. Jako výchozí barva (pro pozadí a velké plochy) je vybrána velmi světlá barva podobná bílé. Nebyla vybrána přímo bílá, neboť bílá barva je značně výrazná a ve spojení s tmavými obsahovými prvky by byl kontrast příliš vysoký. V důsledku toho by došlo k přílišnému záření obrazovky, což by působilo nepříjemně. Barva pro běžný text je zvolena černá nebo velmi tmavý odstín šedé, která je ve spojení se světlým podkladem dobře čitelná. Pro zvýraznění prvků, navigačních tlačítek a panelu nástrojů jsou zvoleny odstíny červené a světle modré barvy. Navíc, pro odlišení cvičebních sad a skupin, je zahrnuto dalších 10 základních barev, jejichž využití bude závislé přímo na uživateli.

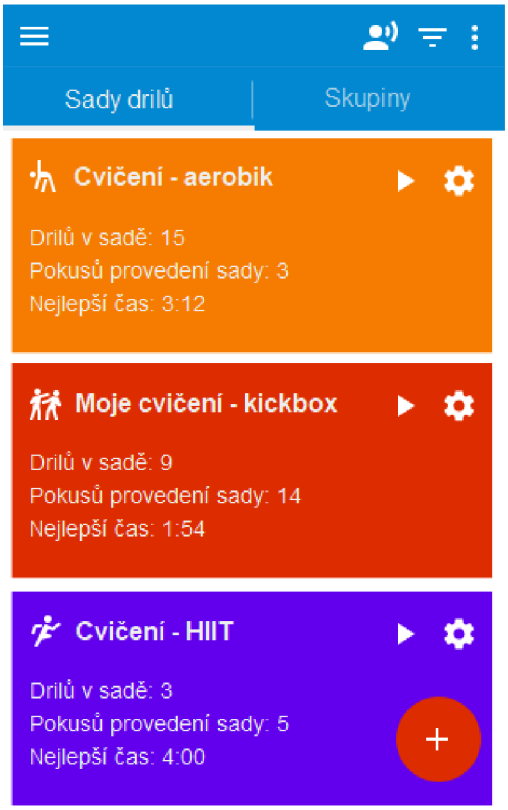
Obrázek 6.1: Vzorky barev použitých v aplikaci.

Pro vzhled a rozvržení prvků je využito postupů tzv. *Material Designu*. Jedná se o postup, jak jednoduše, a přitom efektivně navrhovat grafická uživatelská rozhraní. Napomáhá vhodnému rozvržení komponent tak, aby byly pro uživatele jednoduše čitelné, názorné a logicky rozmístěné. Dalším přínosem těchto postupů je schopnost intuitivně reagovat na dotek a za pomoci zvýraznění a animací dávát uživateli zpětnou vazbu. Tím pádem na každou akci uživatele je automaticky vyvolána reakce, která uživateli potvrzuje provedení akce (animace tlačítka při stisku, zvýraznění hranice seznamu při do scrollování na konec seznamu apod.). Obecně využívání postupů materiál designu tak vede k návrhu mnohem intuitivnějšího a

přehlednějšího uživatelského rozhraní. V souvislosti s Material Designem jsou do aplikace zahrnuty i všeobecně známé a používané ikony, které napomáhají ještě hladšímu a intuitivnějšímu ovládání. Ikony jsou přitom dostatečně názorné, tudíž běžné operace (vkládání, mazání, editace, ...) není třeba popisovat slovně, ale stačí použití ikon.

Zobrazení přehledu cvičebních sad s drily a skupinami

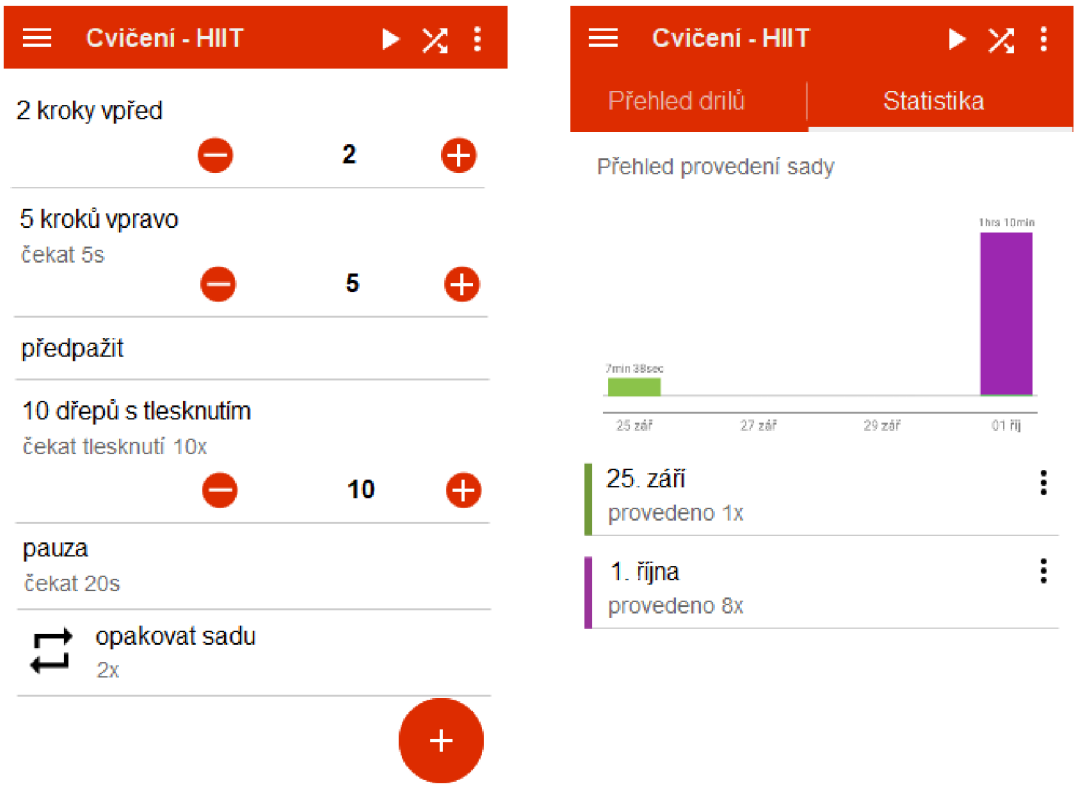
Jelikož aplikace nevyžaduje registraci a ani přihlášení, je tak umožněno libovolnému uživateli aplikaci plně využívat ihned po spuštění. Po spuštění aplikace bude uživateli zobrazena obrazovka s přehledem cvičebních sad s drily a skupiny. Mezi seznamem cvičebních sad a seznamem skupin je možno se přepnout pomocí záložek. Skupiny reprezentují skupiny cviků (typ cvičení) v rámci kterého jsou vytvářeny jednotlivé cvičební sady. Uživatel má možnost vytvářet skupiny nové nebo mazat existující. Každá skupina je kromě svého názvu reprezentována i ikonou, která může napomoci k rozpoznání typu cvičení a případně barvou. Pro nastavení barvy skupiny je k dispozici paleta s několika základními barvami. Barevné odlišení jednotlivých skupin se promítá i v barevném odlišení cvičebních sad, které do dané skupiny patří. Uživatel tak může již na první pohled rozhodnout, které cvičební sady spadají do příslušných skupin. Rozdělení skupin a příslušné barevné oddělení příslušných cvičebních sad je znázorněno na obrázku 6.2. Cvičební sada je v našem případě reprezentována sadou drilů, jenž může uživatel cvičit.



Obrázek 6.2: Návrh rozložení uživatelského rozhraní pro výběr cvičebních sad.

Zobrazení drilů ve cvičební sadě

V případě výběru cvičební sady je uživateli zobrazena obrazovka se seznamem drilů vyskytujících se v sadě. Drily reprezentují jednotlivé cviky, které má uživatel za úkol provádět.



Obrázek 6.3: Návrh rozložení uživatelského rozhraní zobrazení drilů (vlevo) a přehledu statistik cvičební sady (vpravo).

Položky lze rozdělit do základních tří typů:

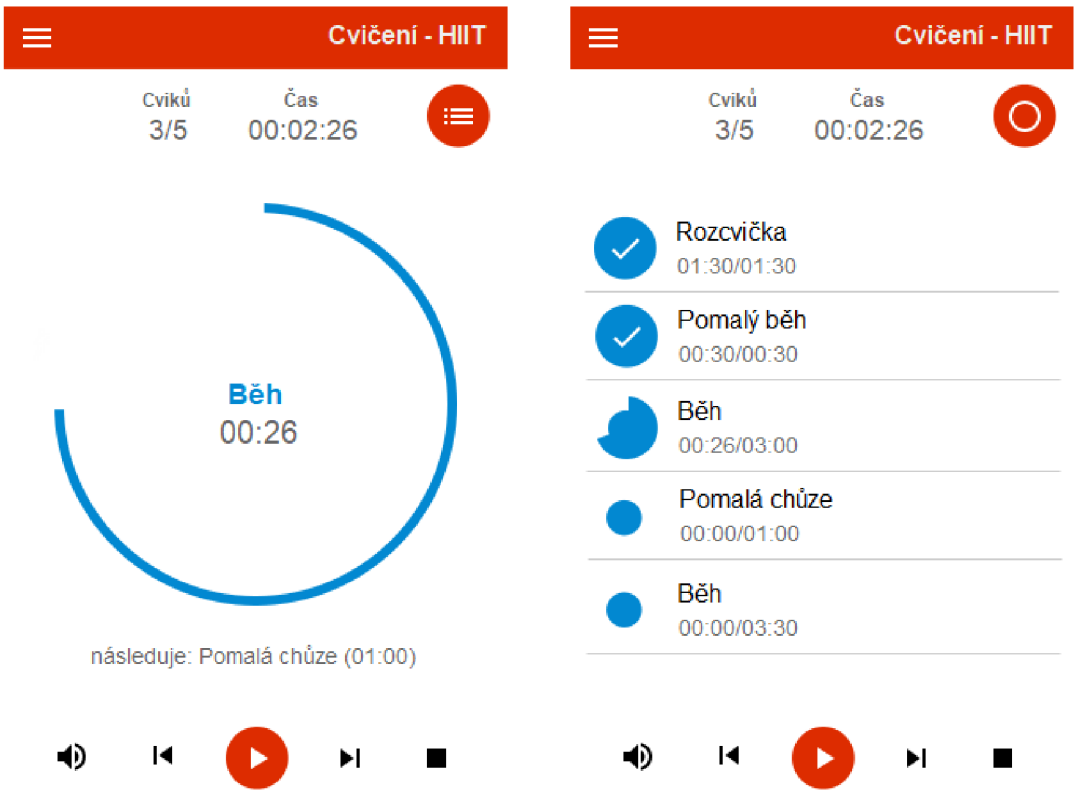
- **Položka s drilem** – představuje jeden prováděný dril. Název drilu zároveň představuje příkaz, který bude aplikace diktovat uživateli. V rámci této položky je možno nastavit mnoho variant, jak má být položka diktována či prováděna. Jednou z možností je nastavení čekání na určitý zvuk, kdy aplikace čeká na zaznění patřičného zvuku a až poté pokračuje v dalším diktování drilů. Dril je také možno nastavit jako variabilní, čímž lze zvolit jedno číslo v rámci drilu, které bude možno rychle (pomocí tlačítek plus a mínus) modifikovat. Jednotlivé drily lze mezi sebou pevně provázat. Tím bude zajištěna návaznost drilů.
- **Pauza** – jednoduchá položka, u které je možno nastavit pouze délku pauzy. Pauza slouží jako mezičas mezi posledním drilem a následujícím drilem. Pauza byla zařazena hned z několika důvodů. V prvé řadě slouží jako čas pro odpočinek či vydechnutí mezi drily. Za druhé ji lze využít pro přípravu cvičebních pomůcek, které budou využívány v následujících drilech. A v neposlední řadě ji lze do cvičební sady zařadit z důvodu přemístění na určitou pozici apod.

- **Položka opakování** – jedná se o právě jednu položku umístěnou na konci každého seznamu. Jde o pevnou položku, která představuje počet opakování provádění sady. Tuto položku nelze odstranit. Lze ji pouze modifikovat nastavením počtu opakování sady. Zařazení této položky je především z důvodu intervalového cvičení, kdy se celá cvičební sada několikrát opakuje.

Všechny drily jsou vloženy do seznamu, kde pořadí drilů v seznamu představuje v základu pořadí jejich provádění.

Statistiky

Obrazovka se statistikami (obrázek 6.3 vpravo) zobrazuje přehledně statistiky jak o jednotlivých cvičebních sadách, tak i o globálním přehledu cvičení. V případě statistik u jednotlivých cvičebních sad jsou zaznamenány dny, kdy byla sada prováděna a celkové časy. Pomocí grafů jsou zobrazeny celkové časy, které uživatel strávit na jednotlivých sadách. Pro vyobrazení statistik je využito technik grafů a diagramů, díky kterým lze názorně zobrazit a porovnat jednotlivé záznamy. Mezi globální statistiky spadá celkový čas strávený nad cvičením, oblíbenost jednotlivých skupin a cvičebních sad atd. Všechny zaznamenané statistiky je možno zpětně uživatelem smazat.



Obrázek 6.4: Návrh rozložení uživatelského rozhraní provádění sady drilů v podobě progress baru (vlevo) a v podobě seznamu (vpravo).

Obrazovka s prováděním cvičební sady

V případě spuštění cvičební sady je uživateli zobrazena obrazovka s prováděním sady. Obrazovka je pomyslně rozdělena do dvou částí. První (horní) částí jsou ukazatelé aktuálního postupu cvičení. Vyskytuje se zde časovač, který udává aktuální uplynulý čas od spuštění sady, počítadlo aktuálně prováděného cviku a progress bar (viz. Obrázek 6.4 vlevo), který zobrazuje aktuálně prováděný cvik, čas přechodu na následující cvik a název následujícího cviku.

Progress bar lze pomocí tlačítka v pravém horním roku přepnout na seznam (viz. Obrázek 6.4 vpravo) cviků. V seznamu jsou názorně zaznačeny provedené cviky, aktuálně prováděný cvik a následující cviky. Volba tohoto přepnutí mezi progress barem a seznamem cviků je především z důvodu přehlednosti pro uživatele. Z průzkumu existujících aplikací totiž vyplynulo, že někteří uživatelé raději preferují přehledný seznam, kde vidí všechny cviky. Naopak druhá skupina uživatelů preferuje spíše progress bar a informace o aktuálně prováděném cviku, přičemž seznam předchozích a následujících cviků není pro ně důležitý. Pro vyhovění oběma těmto skupinám uživatelů jsou do aplikace zahrnuta obě řešení. Záleží tudíž na uživateli, které zobrazení si vybere.

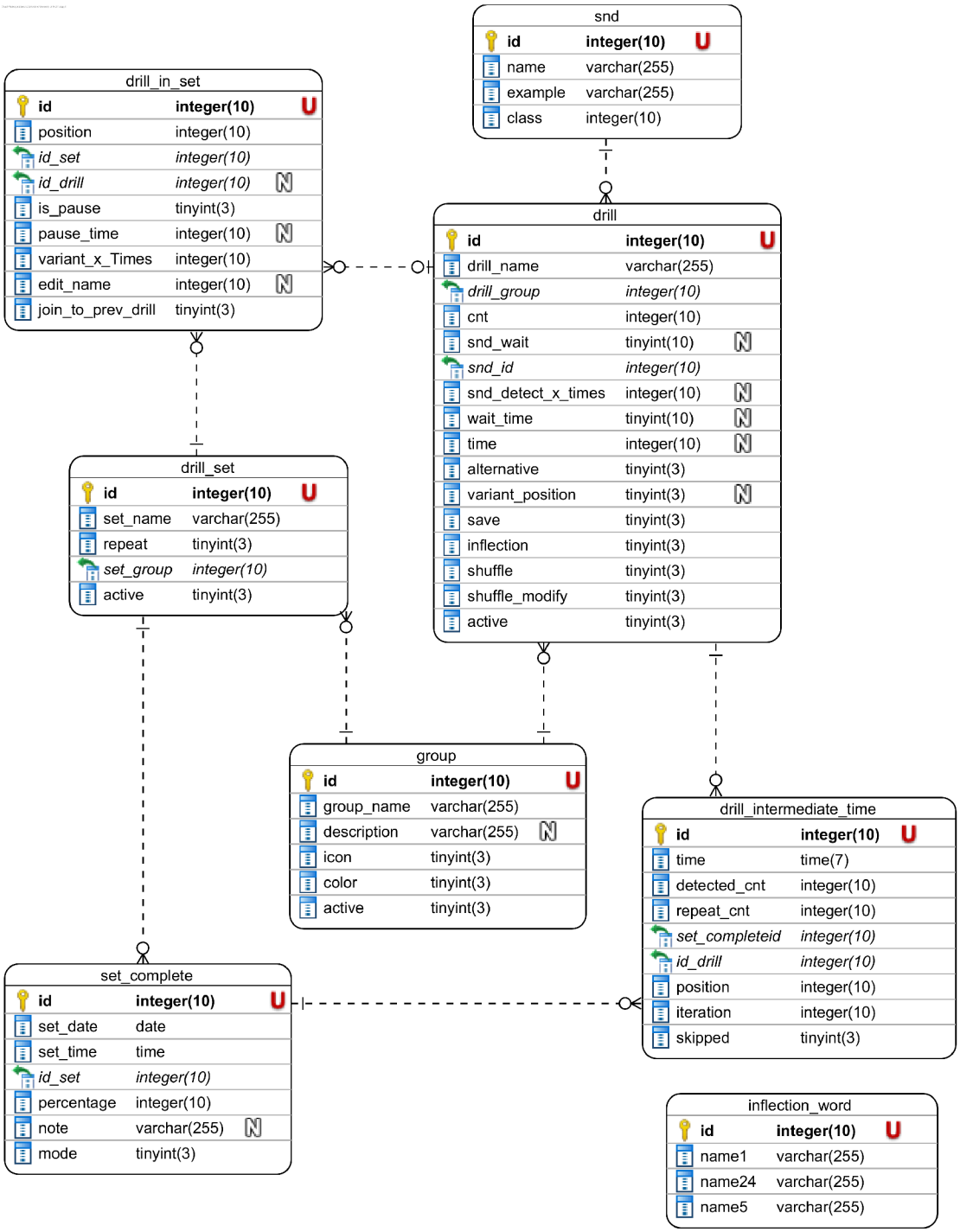
Druhou částí obrazovky je ovládací panel, který obsahuje sadu tlačítek pro ovládání postupu cvičení. Mezi základní tlačítka patří tlačítko pro spuštění/pozastavení cvičení, tlačítko pro přechod na následující cvik a tlačítko pro přechod na předchozí cvik. V případě přechodu na předcházející cvik je cvik zopakován. Dále zde najdeme tlačítko pro zastavení cvičení. Při jeho stisku dojde k předčasnému ukončení cvičení a zobrazí se obrazovka shrnující průběh cvičení. Speciální tlačítko je pro aktivaci/deaktivaci zvuku. Tlačítko lze využít v případě, že chceme ztišit slovní příkazy a zvukové upozornění.

6.2 Ukládání dat

Jelikož je aplikace navržena jako lokální (offline), je třeba zvolit způsob ukládání lokálních dat v rámci mobilního zařízení. V úvahu přichází tři způsoby, jak lze na mobilním zařízení ukládat data interně. Prvním jsou sdílené preference (Shared Preferences), které jsou vhodné pro ukládání menšího množství dat. Tento způsob však není vhodný pro ukládání rozsáhlejších strukturovaných dat. Tím pádem sdílené preference jsou předem vyloučeny. Dalším způsobem je využít poskytovatelů obsahu (Content Providers), díky kterým lze data nejen ukládat, ale i sdílet s jinými aplikacemi v rámci systému Android. Avšak ani tento způsob ukládání není příliš vhodný. Zaprvé není možno v datech efektivně vyhledávat a možnost sdílení dat s ostatními aplikacemi není v rámci naší aplikace ani žádoucí. Třetím způsobem ukládání dat je vestavěná lokální databáze, která funguje na základu SQL. Je tak možno vytvářet vlastní lokální tabulky s daty, v rámci kterých lze využít efektivního dotazování na uložená data. Tento způsob ukládání uživatelských dat je pro navrhovanou aplikaci nejpříhodnější.

6.2.1 Návrh databáze

- Tabulka `drill_set` – představuje cvičební sadu drilů. Sada náleží do určité skupiny (tabulka `group`) a obsahuje množinu cviků, které jsou po spuštění sady prováděny uživatelem. Pro podporu intervalového cvičení je možno sadu libovolně-krát opakovat. Pro uložení počtu opakování slouží číselný atribut `repeat`.



Obrázek 6.5: Schéma databáze

- Tabulka `drill` – jedná se o jednu z klíčových tabulek, která jednak ukládá základní informace o samotném cviku, ale i nastavení, které je možno u cviku provádět. Cvik (dril) je reprezentován názvem, který zároveň slouží jako slovní pokyn, který dá aplikace uživateli během cvičení. Atribut `group_drill` slouží pro přiřazení cviku kon-

krétní skupině (z důvodu vyhledávání a ukládání konceptů). Zároveň atribut `save` značí, zda je dril uložen jako koncept, který se má zobrazovat v nabídce předuložených drilů, či slouží jako jednoúčelový dril. Mezi další význačné atributy patří např. `time_wait`, `wait`, `snd_wait` a `id_snd`, které určují, zda se bude během provádění cviku čekat na detekci zvukového signálu, či se bude jen čekat určitý čas před přechodem na další cvik. Při nastavení drilu pro rychlou úpravu slouží atribut `variant_position`, který určuje pozici čísla v názvu drilu. Tato číselná hodnota může být rychle modifikována pomocí tlačítek přímo v seznamu drilů.

- Tabulka `group` – představuje skupinu, v rámci které jsou ukládány jednotlivé sady drilů. Skupina reprezentuje samostatný okruh cviků (střelba, bojové sporty, aerobik, intervalové cvičení apod.). Krom názvu a detailnějšího popisu skupiny jsou uloženy i informace o barvě skupiny a ikoně, která danou skupinu reprezentuje.

- Tabulka `drill_in_set` – reprezentuje spojovací tabulku mezi cvičební sadou a konkrétními cviky. Tabulka ukládá informace o vložení příslušného drilu do cvičební sady. Krom pozice (atribut `position`) v rámci sady a propojení s předchozím drilem (atribut `join_to_prev_drill`) jsou v tabulce i atributy, které slouží pro reprezentaci pauzy. Pauza je specifický případ, neboť se nejedná o cvik jako takový. V případě pauzy je nastaven atribut `is_pause` na true a zároveň neexistuje odkaz do tabulky `drill`.

- Tabulka `set_complete` – ukládá informace o provedení konkrétní sady. Jedná se o jednoduchou tabulku, která indikuje datum a čas provedení cvičební sady (atribut `date_time`) a dobu provádění celé sady (atribut `time_set`). Tabulka `set_complete` spolu s tabulkou `drill_intermediate_time` slouží především pro statistické účely.

- Tabulka `drill_intermediate_time` – obsahuje data o mezičasech provedení určitých cviků (čekající na zvukový signál) v sadě. Mezičas se váže na konkrétní čas v konkrétním provedení určité sady. Speciálními atributy jsou atributy `iteration` a `position`, které jsou využívány pro mezičasy v rámci intervalového cvičení. Slouží pro určování mezičasů drilů i v rámci konkrétního cyklu.

- Tabulka `snd` – reprezentuje zvuk, který má být detekován. Tabulka obsahuje přehled zvuků, které mohou být propojeny s daným cvikem. Propojení s cvikem indikuje, že cvik bude čekat na zaznění daného zvuku. Množina aktivních záznamů v tabulce zároveň představuje množiny detekovatelných zvuků.

- Tabulka `inflection_word` – představuje samostatnou tabulku, která slouží pro uchování variant slov pro variantní nastavení drilu.

6.3 Detekce a analýza zvuků

Pro návrh a následnou implementaci systému, který by byl schopen detekovat a klasifikovat zvuky do správných tříd, jsem zvolil přístup MFCC koeficientů klasifikovaný pomocí SVM. Hlavním důvodem je, že přístup MFCC se osvědčil právě v oblasti analýzy a klasifikace enviromentálních zvuků. Metoda MFCC si dokáže dobře poradit jak s vokálními, tak i s enviromentálními zvuky, což je pro mou realizaci ideální. Jako klasifikační metoda byl zvolen SVM, a to především z důvodu, že si dokáže poradit s velkým množstvím dimenzí, což je opět vhodné pro náš případ, neboť velikost bloku vektorů rysů je v řádech stovek

rysů. Zároveň bylo klíčové zvolit takovou metodu, která by byla schopna provádět klasifikaci v reálném čase, čemuž SVM opět vyhovuje. Tudíž kombinací MFCC s poměrně přesným klasifikátorem SVM je tak možno dosáhnout poměrně dobrých výsledků, které však budou ověřeny v kapitole 8.

Postup algoritmu

Zvukové signály budou parametrizovat do segmentů, pro které se spočte několik MFCC koeficientů. Ke klasifikaci bude použit klasifikátor SVM, který dosahuje dobrých výsledků v klasifikaci enviromentálních zvuků a oproti používanějšímu GMM (Gausian Mixture Model) je značně rychlejší, což v případě real-time systémů klasifikace je klíčový předpoklad.

Postup výpočtu MFCC:

1. Získáme zvukový signál vzorkovaný na 16kHz.

2. Rozdělíme signál do rámců o velikosti 25ms. Tím získáme délku rámců 0,025 * 1600 = 400 vzorků. Velikost kroku je 10ms (přibližně 160 vzorků), což umožňuje určité překrytí rámců. Prvních 400 vzorků rámce začíná na vzorku 0, další rámec začíná na vzorku 160 atd.

3. Následně vypočteme Fourierovu transformaci pro každý rámec pomocí vzorce:

$$S_i(k) = \sum_{n=1}^{N} s_i(n)h(n)e^{-j2\pi kn/N} \tag{6.1}$$

   kde $h(n)$ je hammingovo okno o délce N vzorků, $s_i(n)$ je i-tý rámec a K je délka Fourierovy transformace. Spektrální odhad energie rámce $s_i(n)$ získáme na základě výpočtu periodogramu:

$$P_i(k) = \frac{1}{N}|S_i(k)|^2 \tag{6.2}$$

   Výsledek $P_i(k)$ nazývaný periodogramový odhad energie spektra získáme umocněním absolutní hodnoty komplexní Fourierovy transformace.

4. Vypočteme Mel filtrační banku, která se skládá 26 trojúhelníkových filtrů. Tyto filtry aplikujeme na odhad energie spektra z bodu 3. Naše filtrační banka obsahuje 26 vektorů o délce 256 prvků. Každý vektor je ve většině hodnot nulový, pouze pro určitou část spektra nabývá nenulových hodnot. Pro výpočet energií filtrů filtrační banky vynásobíme každý filtr s energií spektra a přičteme koeficienty. Jakmile provedeme výpočet pro každý filtr, získáme 26 čísel, které nám ukazují, kolik energie se vyskytuje v každé filtrační bance.

5. Na každý z 26 výpočtů energií z bodu 4 aplikujeme logaritmus.

6. Vypočteme Diskrétní kosinovu transformaci (DCT) pro každý z logaritmů 26 filtračních bank z předchozího kroku. Tím získáme 26 kepstrálních koeficientů, z nichž použijeme pouze koeficienty 2-13.

Postup výpočtu Mel frekvenční banky:

1. Nejprve je třeba zvolit maximální spodní a horní frekvenci. Tím stanovíme rozsah v rámci kterého budeme získávat koeficienty. Já zvolil rozsah 0Hz - 8000Hz.

2. Pomocí rovnice 5.3 převedeme spodní a horní frekvenci na Mel. V mém případě 0Hz je 0 a 8000Hz je 2835.

3. Uvažujeme celkem 26 filtračních bank, proto mezi mezní body 0 a 2835 lineárně rozmístíme dalších 26 bodů. Tím získáme množinu M.

4. Všechny prvky množiny M nyní převedeme zpět na Hertze pomocí vzorce 5.4. Tím vznikne množina Q.

5. Jelikož nemáme rozsah frekvencí potřebných pro umístění filtrů, musíme provést zaokrouhlení frekvencí z množiny Q na nejbližší koeficient Rychlé Fourierovy transformace. Výsledné hodnoty tvoří množinu F.

6. Nyní vytvoříme jednotlivé filtrační banky. První banka začíná v prvním bodě množiny F, roste k vrcholu v bodě druhém a následně klesá směrem k nule, které dosáhne v bodě třetím. Druhá banka začíná v bodě dvě, maxima dosáhne v bodě 3 a následně dosáhne opět nuly v bodě 4 atd. Takto získáme všech 26 filtračních bank. Vzorec pro tento výpočet je následující:

$$H_m(k) = \begin{cases} 0 & k < f(m-1) \\ \frac{k-f(m-1)}{f(m)-f(m-1)} & f(m-1) \leq k \leq f(m) \\ \frac{f(m+1)-k}{f(m+1)-f(m)} & f(m) \leq k \leq f(m+1) \\ 0 & k > f(m+1) \end{cases} \tag{6.3}$$

kde M je počet filtrů a $f()$ je seznam $M + 2$ Mel frekvencí.

Kapitola 7

Implementace

V následující kapitole je popsána implementace zajímavých částí aplikace, použitých externích knihoven a důvod jejich použití. Dále je popsána implementace samotné aplikace a systému pro detekci a rozpoznávání zvuků a hlasů.

7.1 Programovací jazyk a vývojové prostředí

Pro naprogramování aplikační logiky nativní aplikace byl využit programovací jazyk Java. K vývoji aplikace pro operační systém Android jsem zvolil Android Studio verzi 3.3.1. Android Studio je vývojové prostředí vyvíjené ve spolupráci firmy Google a JetBrains. Přestože většina produktů od firmy JetBrains jsou placené komerční nástroje, Android Studio je plně zdarma. Android Studio je postaveno nad Community verzí populárního prostředí IntelliJ IDEA. Díky tomu získává mnoho možností pro práci s kódem, jako je navigace v kódu, inteligentní našeptávání, refaktorizace, analýza kódu apod. Android Studio navíc poskytuje i grafický návrhář, ve kterém je možné navrhovat vzhled buď přímo v XML nebo v tzv. *Design módu*. V případě stylování aplikace přímo v XML, návrhář automaticky zobrazuje náhled výsledného návrhu ve vybraném rozlišení či orientaci obrazovky.

Pro analýzu a nahrávání sady zvuků a hlasů jsem využil nástroj Audacity, což je otevřený (open source), multiplatformní editor digitálního zvuku. Nástroj umožňuje krom nahrávání a editace zvuku i analýzu, spektrografické zobrazení a ukládání zvuku ve všech rozšířených formátech.

7.2 Základní funkcionalita aplikace

V následující podkapitole je popsána implementace těch nejdůležitějších a nejzajímavějších částí mobilní aplikace.

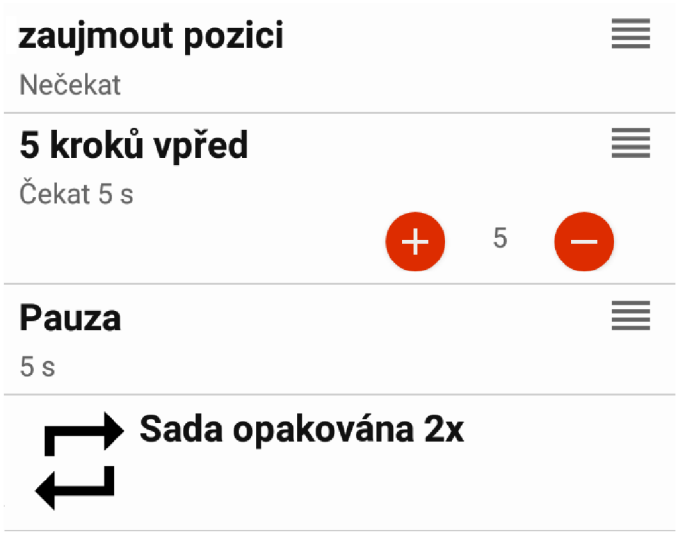
7.2.1 Realizace databáze

Jak už bylo popsáno v kapitole 6.2, pro uložení dat bylo využíváno interního úložiště, které je založeno na SQL. Platforma android již v základní sadě nástrojů nabízí vestavěné úložiště v podobě SQLite databáze. Lze tak vytvořit interní úložiště přístupné pouze z aplikace. I když je SQLite založeno na standardu SQL92, tak z tohoto standardu implementuje v podstatě jen jádro s několika málo funkcemi. Z toho plyne poměrně velké množství omezení, které je třeba při práci s SQLite brát v úvahu. Poměrně velké omezení je v oblasti uložených procedur a trigerů, což není zas takový problém, protože v rámci této práce nejsou

žádné databázové procedury ani trigery používány. Omezení, které je však již třeba brát na vědomí se týká cizích klíčů. Cizí klíče jsou totiž sice vyhodnoceny, ale nejsou vyžadovány restrikce, které by měly být uplatněny na jejich základě. Další větší omezení je v rovině editace tabulek, kdy nelze využít příkazů `ALTER`, tudíž není možno jakkoliv upravovat již jednou vytvořené schéma tabulek. Tento problém bylo třeba obejít zrušením a znovu vytvořením patřičné tabulky. Avšak tento problém se týká spíše změn, které budou prováděny z hlediska udržování aplikace. Zejména bude-li třeba pozměnit schéma tabulek, ale přitom zachovat již uložená uživatelská data. Asi největší překážkou používání SQLite je absence datových typů. Databáze deklaruje pouze dva datové typy `number` (případně `integer`) pro numerická data a `varchar` pro data textová. Avšak reálně je možné do sloupce pro čísla uložit i řetězec, aniž by to vyvolalo jakoukoliv chybu. Jedinou výjimkou je sloupec deklarovaný jako `INTEGER PRIMARY KEY`, kde je požadováno jednoznačné celé číslo. Taková deklarace byla použita pro všechny sloupce jejichž hodnota měla sloužit jako jednoznačný identifikátor záznamů. Mobilní aplikace pracuje celkem se čtyřmi datovými typy: `integer` pro číselné hodnoty, `varchar` pro textové řetězce, `boolean` pro příznaky a `datetime` pro uchování aktuálního data a času. V případě číselných a řetězcových hodnot nebyl problém provést přímé uložení hodnot do databáze. U příznaků bylo nutné provést konverzi z pravdivostní hodnoty na hodnotu číselnou a to následovně (false $\rightarrow$ 0, true $\rightarrow$ 1). Obdobná konverze byla nutná i u data a času, kdy bylo třeba překódovat časový údaj na číslo. Nejvhodnější formát pro uchování časového údaje se jevil linuxový čas, což je celé číslo udávající počet sekund od data 1.1.1970, 00:00:00 UTC.

7.2.2 Seznamy

Jedním z nejdůležitějších prvků aplikace jsou seznamy. Aplikace disponuje mnoha seznamy s různorodými položkami. Základní kontejner seznamu však zůstává stejný. Seznamy tak obsahují pouze rozdílné položky, které pro jednotlivé kategorie a obsahy byly stylovány do samostatných adaptérů. Položky byly navrženy dynamicky pro různé varianty použití v rámci aplikace.



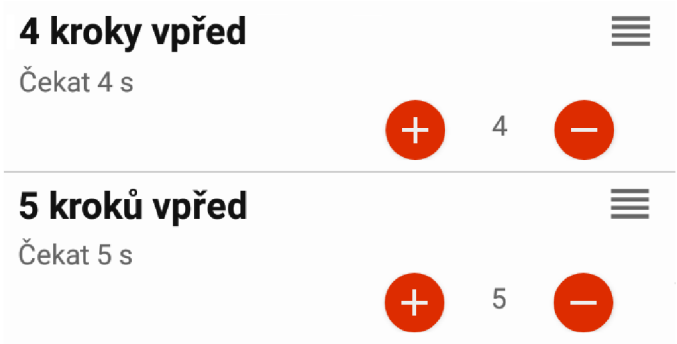
Obrázek 7.1: Ukázka uzpůsobení položky pro různé typy záznamů ve cvičební sadě.

Příkladem může být položka pro zobrazení drilu ve cvičební sadě. V seznamu se vyskytují tři druhy položek, cvičební dril, pauza a počet opakování cvičební sady. Jelikož všechny tři položky využívají stejné rozložení, bylo by zbytečné implementovat pro každý druh položky zvlášť rozložení. Řešením tak bylo využít jeden styl pro všechny tři položky. Prvky, které jsou v rámci konkrétního zobrazení nežádoucí, jsou jednoduše skryty. Stejné položky byly navíc využity i na obrazovce předuložených drilů, kde zobrazení položek je až na drobné rozdíly téměř identické s položkami ve cvičební sadě. Avšak kvůli rozdílnému chování položek v přehledu drilů a ve cvičební sadě bylo nutno implementovat rozdílné adaptéry. Ty zajistily správné mapování dat na konkrétní prvky a postaraly se o rozdílnou reakci na akce vyvolané uživatelem.

7.2.3 Variabilita drilů

Aplikace umožňuje uživateli přizpůsobit vlastní cvičební sadu, která se skládá z množiny drilů. Jednotlivé drily je možné do jisté míry nastavovat a přizpůsobovat, což bylo třeba promítnout i do implementační úrovně. Drily lze rozdělit ve dvou rovinách. V první rovině se dělí na rychle modifikovatelné a nemodifikovatelné a ve druhé rovině na čekající a nečekající.

Každý dril je definován svým názvem, který popisuje jeho úkon. Tento název zároveň slouží jako příkaz, který aplikace verbálně předává uživateli pomocí technologie převodu textu na řeč (TTS). Název drilu může vypadat jako „Vstaň“ nebo „Udělej 5 kliků“. Dril, který obsahuje pouze slova, je automaticky zařazen mezi nemodifikovatelné. Takový dril může být upraven pouze přes editaci drilu. Naopak dril, který obsahuje nějaké číslo, lze označit jako modifikovatelný s určením, jaké číslo v drilu má být modifikováno. Například máme-li dril „Jdi 2 kroky vpřed a 3 kroky vpravo“, uživatel má možnost označit takový dril jako rychle modifikovatelný s tím, že může nastavit, jestli bude chtít modifikovat číslo 2 nebo číslo 3. V případě nastavení drilu jako rychle modifikovatelný a zvolení patřičného čísla je uživateli umožněna rychlá modifikace tohoto čísla přímo ze seznamu drilů ve cvičební sadě (viz. obrázek 7.2). V případě, že se za modifikovaným číslem nachází nějaké slovo, má uživatel možnost nastavit skloňování tohoto slova. Pro tento případ slouží databázová tabulka `inflection_word`, která v sobě obsahuje varianty slov vázané pro různá čísla. Pokud uživatel vybere možnost skloňování slova a slovo není doposud uloženo v databázi, dojde ke zobrazení dialogu, který uživatele vyzve k zadání variant tohoto slova.



Obrázek 7.2: Možnost rychlé modifikace drilu přímo ze seznamu drilů ve cvičební sadě. Krom samotné úpravy drilu je automaticky přizpůsobován i čas čekání, aby odpovídal novým hodnotám.

Každému drilu, ať už rychle modifikovatelnému či nemodifikovatelnému, je možno přiřadit čekání. Čekání je buďto časové nebo zvukové. Časové znamená, že po zaznění příkazu drilu se čeká určitý čas, než se přejde k provádění dalšího drilu. Naopak zvukové čekání značí, že po zaznění příkazu drilu se čeká, než dojde k zaznění určitého zvuku. Oba parametry, tedy čas i zvuk, je možno nastavit přímo uživatelem. Časové čekání je především pro získání dostatečného času pro provedení drilu (zvláště v případě obsahuje-li příkaz drilu složitější cvik). Naopak zvukové čekání slouží pro detekci zvuku, který je během provádění drilu vydán (např. odraz míče).

7.2.4 Převod textu na řeč

Proces cvičení je založen na provádění posloupnosti drilů, které aplikace verbálně diktuje uživateli. Pro realizaci převodu názvu drilu na řeč bylo třeba využít nástroje text-to-speech (TTS). Sada android knihoven již v základu disponuje TTS syntetizátorem, který je schopen jednoduše provést syntézu textového řetězce na zvukový signál. Jelikož má knihovna nastavený jako výchozí jazyk angličtinu, bylo třeba během inicializace TTS nástrojů provést přenastavení na českou lokalizaci. Česká lokalizace je již od roku 2017 dostupná v off-line režimu. V případě, že uživatel českou lokalizaci v mobilním zařízení nemá, je vyvolána akce, která by měla zajistit automatické stažení této knihovny. K tomuto kroku je však nutné provést připojení k internetu. Má-li však uživatel již lokalizaci v zařízení, je následná syntéza TTS využívána off-line bez nutnosti internetového připojování mobilního zařízení. Samotná syntéza je v rámci aplikace řízena pouze dvěma metodami. První metoda slouží pro zahájení syntézy. Jako parametr přijímá řetězec, který má být syntetizován. Druhá metoda slouží pro předčasné ukončení syntézy. Tato metoda se využívá především v případě, kdy je proces cvičení pozastaven nebo ukončen. V případě nezastavení procesu syntézy by došlo k blokování uživatelské akce do doby, než by byl přehrán celý výsledek syntézy. Pomocí několika parametrů lze nastavit výsledný syntetizovaný hlas. Některá nastavení hlasu jsou zahrnuta i do výsledné aplikace a skrze aktivitu nastavení jej mohou uživatelé měnit. Především se jedná o úpravu výšky a rychlosti výsledného hlasu. Mezi významná nastavení obvykle patří i možnost volby vzoru hlasu. Protože je však pro českou lokalizaci dostupný pouze jediný hlasový vzor, nebylo toto nastavení zahrnuto do výsledné aplikace.

7.2.5 Statistiky

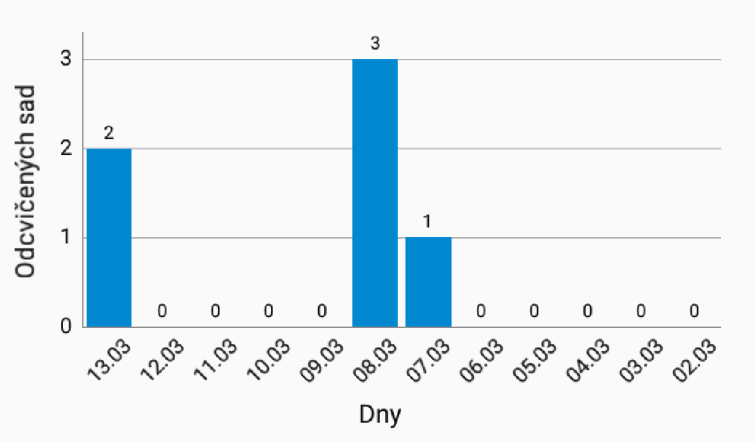
Protože se aplikace zaměřuje na cvičení, které je možno mnohokrát opakovat, vyvstala potřeba jistých statistik, které by dávaly uživateli jistý přehled o prováděných cvicích. Mezi takové údaje patří např. počet provedených cvičení, celkem odcvičených sad, oblíbenost jednotlivých sad nebo detailní záznamy o provedení cvičebních sad. Nejvhodnějším způsobem, jak tyto informace reprezentovat, bylo využití grafů a diagramů. Ty totiž umožňují názornou a grafickou reprezentaci dat oproti prostému textu a tabulkám. Uživatelé se v hodnotách lépe orientují a lépe jim porozumí. Pro implementaci grafů a diagramů byla využita knihovna MPAndroidChart (viz. kapitola 7.3.1). Knihovna umožňuje jak jednoduchou implementaci různých typů grafů, tak i jejich detailní přizpůsobení. Z knihovny jsem využil sloupcový graf pro znázornění počtu odcvičených sad v rámci dnů a měsíců (viz. obrázek 7.3) a vertikální sloupcový graf pro oblíbenost jednotlivých sad.

7.3 Použité externí knihovny

V následující kapitole jsou uvedeny a přiblíženy všechny knihovny třetích stran, které jsou v rámci aplikace využívány.

7.3.1 MPAndroidChart

MPAndroidChart je jednoduchá a velice efektivní knihovna umožňující implementovat přehledné grafy, které lze přizpůsobit přesnému obrazu aplikace. Důvodem zahrnutí této knihovny do projektu je ten, že knihovna nabízí širokou škálu grafů, jako jsou např. lineární grafy, spojnicové grafy, sloupcové grafy, bodové grafy anebo i koláčové a výsečové grafy. Dokonce lze některé z těchto typů grafů i kombinovat. Každý graf lze plně graficky i funkčně přizpůsobit potřebám aplikace, přičemž umožňuje uživateli i jednoduchou práci s grafy v podobě přiblížení, zvýraznění bodů apod. Knihovna je podporovaná pro všechny verze Android API od verze 8. Více o knihovně na oficiální GitHub stránce[1].



Obrázek 7.3: Ukázka použití grafu z knihovny MPAndroidChart v aplikaci.

7.3.2 libSVM

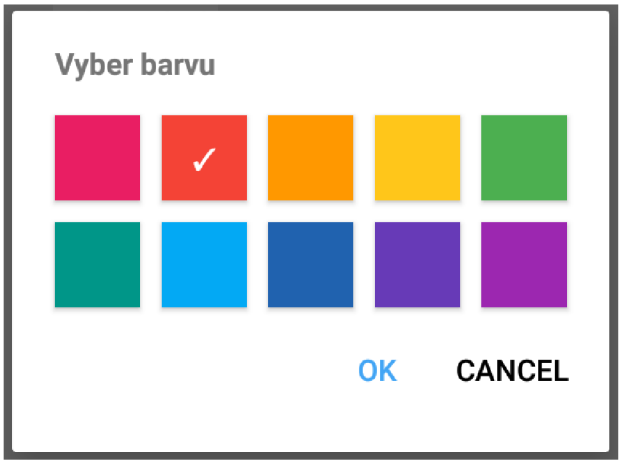
Knihovna libSVM představuje výkonný nástroj v oblasti vektorových klasifikací, vícetřídních klasifikací a regresní analýzy. Oproti jiným knihovnám nabízí poměrně velké množství klasifikačních metod s detailní možností nastavení jednotlivých klasifikačních modelů. Knihovna je primárně vyvíjena v jazyce C++ a Java. Avšak díky řadě rozšíření je dostupná i pro všechny ostatní známé jazyky (např. Python, Ruby, MATLAB, PHP apod.). Knihovna v základu nabízí dvě vícetřídní klasifikace C-SVC a nu-SVC, dvě metody pro regresní analýzu epsilon-SVR, nu-SVR a binární klasifikátor one-class SVM. V rámci této práce nás bude především zajímat metoda C-SVC a one-class SVM. U všech metod je možno zvolit jednu z předpřipravených kernel funkcí (lineární, polynomiální, radiální). Model samozřejmě počítá i s možností dosazení své vlastní kernel funkce. Krom základních parametrů pro každou z metod nabízí knihovna i váhové vybalancování nevyvážených dat, což je přínos především máme-li nevyvážená trénovací data. Značným přínosem je také křížová validace. Knihovna byla pro tuto práci zvolena především díky široké nabídce různých klasifikačních metod

[1] https://github.com/PhilJay/MPAndroidChart

a jejich nastavení, díky kterým se nabízí více možností experimentování s modelem. Zároveň se jedná o neustále vyvíjenou a udržovanou knihovnu, která oproti jiným knihovnám nabízí i spoustu podpůrných a rozšiřujících nástrojů (pravděpodobností odhady, křížovou validaci včetně její analýzy atd.). V rámci aplikace je využívána poslední stabilní verze 3.20 knihovny zaměřené na OS Android. Více informací je k dispozici na oficiálních stránkách knihovny[2].

7.3.3 ColorPicker

Jedná se o knihovnu umožňující jednoduše sestavit dialogové okno pro zobrazení a výběr barevné palety. Knihovna umožňuje přesné nastavení barevné palety, počtu barev a i konkrétní hodnoty barev. Pro lepší zakomponování do grafického návrhu vlastní aplikace je možnost zvolit i tvary jednotlivých selektorů tak, aby byly co nejlépe sjednoceny. V aplikaci je prvek výběru barvy využit při vytváření či editaci skupiny drilů. Více o knihovně ColorPicker na GitHub[3].



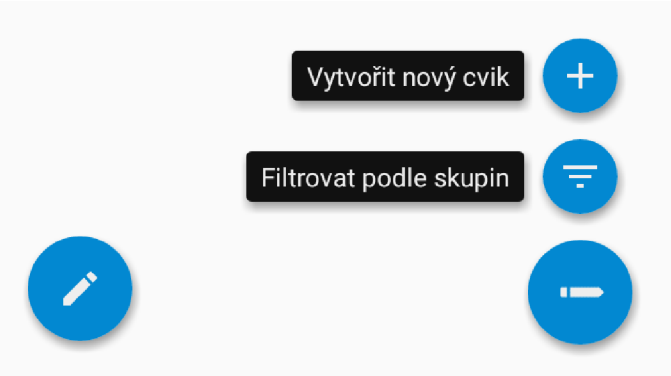
Obrázek 7.4: Ukázka dialogu pro výběr barvy skupiny drilů.

7.3.4 FloatingActionButton

Jde o knihovnu poskytující prvek uživatelského rozhraní pro zobrazení rozevíracího plovoucího tlačítka. Knihovna poskytuje jednoduché rozhraní pro vytvoření jednoduchých i složitých plovoucích tlačítek. Tlačítka navíc poskytují více prvkovou volbu (tzv. plovoucí menu) s možností plného přizpůsobení. Knihovna využívá a rozšiřuje prvky `FloactinActionButton` a `FloatingActionMenu`. Mezi rozšiřující funkce klasických plovoucích oken patří například možnost volby směru rozevírání plovoucího menu, zakomponování progress baru přímo do okraje tlačítka, jednoduchý způsob nastavení velikosti, barvy a stylu celého plovoucího menu nebo jednoduše dynamicky přidávat či odebírat prvky menu. Prvky z této knihovny byly v aplikaci využity především pro zpřehlednění a zjednodušení grafického rozvržení obrazovek. Prvky plovoucích menu byly využívány především pro vkládání/editaci položek

[2]https://www.csie.ntu.edu.tw/ cjlin/libsvm/
[3]https://github.com/kristiyanP/colorpicker

nebo filtrování. Bylo tak umožněno schovat několik funkcí pod jedno jediné tlačítko. Více o knihovně FloatingActionButton na GitHub[4].



Obrázek 7.5: Plovoucí tlačítko s plovoucím menu. Vizuální stránka tlačítka před a po rozevření.

7.3.5 Android-ripple-pulse-animation

Jedná se o malou knihovnu, která nabízí možnost přidat tlačítku pulzující efekt. Knihovna nabízí dva druhy pulzování (plné a prázdné), přičemž velikost, frekvenci i bravu pulzu lze programově přizpůsobit vlastním potřebám aplikace. Efekt popsané knihovny byl v rámci aplikace využit pouze v dialogovém okně, které upozorňuje na aktuální naslouchání aplikace okolním zvukům. Více o knihovně Android-ripple-pulse-animation na GitHub [5].

7.3.6 Android Swipe Layout

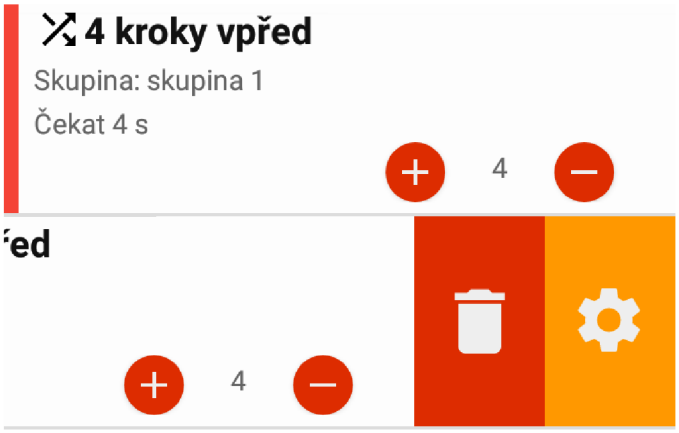
Android Swipe Layout je rozsáhlá a všestranná knihovna pro jednoduchou realizaci tzv. swipe efektu[6] pro libovolný obrazový element. I když pro obdobnou funkcionalitu existuje velké množství různých knihoven, tak knihovna Android Swipe Layout ostatní převyšuje především variabilitou. Knihovna nabízí možnost swipe efektu nad grafickým elementem do všech čtyř stran. Navíc umožňuje libovolně stylovat a přizpůsobit skrytý element. Tím lze vytvářen složité swipe efekty a lépe zužitkovat prostor obrazovky. Zároveň lze knihovnu aplikovat nejen na rozložení ListView (jako většina knihoven), ale pro téměř libovolný element rozvržení (ListView, GridView, RelativeView, ...). V neposlední řadě knihovna disponuje i možností registrovat posluchače pro libovolné skryté prvky, což většina běžných knihoven umožňuje pouze pro tlačítka. Funkcionalita této knihovny byla v aplikaci využita především pro zobrazení skrytých tlačítek pro editaci a mazání existujícího cviku. Více o knihovně na oficiálních stránkách Github[7].

[4]https://github.com/Clans/FloatingActionButton

[5] https://github.com/gaurav414u/android-ripple-pulse-animation

[6]Dotknete se prstem obrazovky a posunujete jím po nějakém obrazovkovém elementu (který provádí jistou akci – nejčastěji vysunutí), a pak prst zvednete.

[7]https://github.com/daimajia/AndroidSwipeLayout



Obrázek 7.6: Využití swipe efektu v seznamu drilů pro skrytí tlačítek pro smazání a editaci.

7.4 Implementace extrakce rysů a klasifikace

Model rozpoznávání zvuků lze rozdělit na dvě hlavní části, část extrakce rysů a část klasifikace. Celý proces extrakce pracuje následovně. Z mikrofonu zařízení je získáván proud bytů. Proud je rozdělen na rámce po 25ms a pro každý rámec je spočtena Rychlá Fourierova transformace. Výpočtem získáme spektrum zvukového signálu. Ze spektra vypočteme spektrální odhad energie rámce, což jsou referenční data pro výpočet MFCC koeficientů. Následně se přechází k výpočtu Mel-filtračních bank, které se aplikují na spektrální odhad energie. Pro výsledné hodnoty se provede výpočet logaritmu a následně výpočet Diskrétní kosinovy transformace. Tím získáme finální MFCC koeficienty. Nad nimi je dále možno provést výpočty Delta případně Delta-delta (tzv. 2 delta), které zpřesňují MFCC koeficienty. Signál je zpracováván po blocích, které se skládají z několika rámců. Výsledný plný vektor rysů bloku je složen z dílčích vektorů rysů rámců.

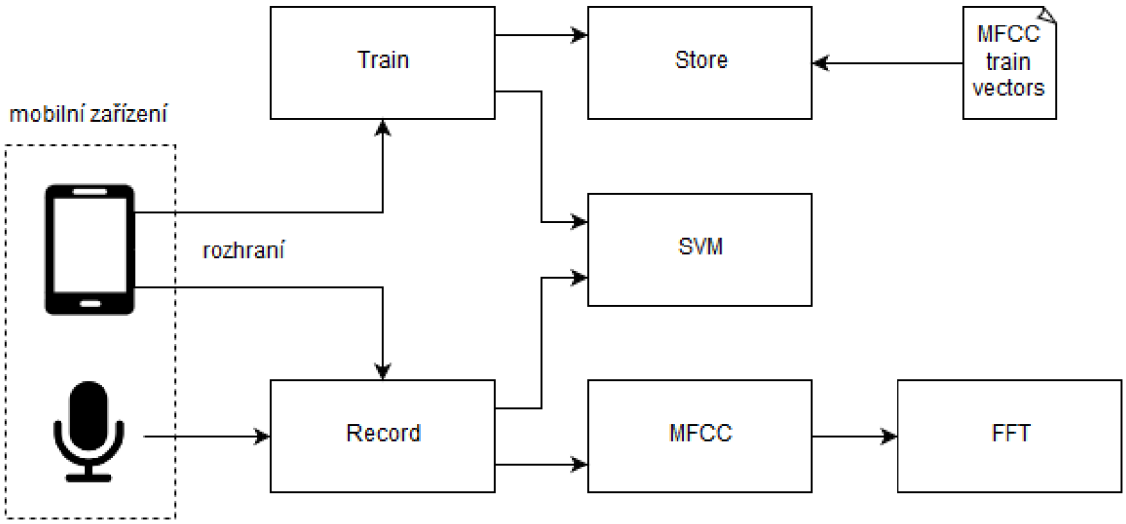
Pro rozpoznávání byla zvolena metoda SVM, které jsou předávány výstupy z části extrakce rysů. V rámci aplikace je využívána libSVM knihovna, která slouží pro klasifikaci vektoru rysů (výsledků extrakce rysů) do několika tříd. Každá z tříd přitom určuje, o jaký zvuk se jedná. Před samotnou klasifikací je třeba provést naučení modelu. Učení je však třeba provést při každém spuštění aplikace. A jelikož extrakce MFCC rysů celé trénovací sady by byla zbytečná výpočetní i časová zátěž. Je předpřipraven trénovací soubor, ve kterém jsou již uloženy všechny vektory rysů trénovací množiny, a to včetně referenčních tříd. Během procesu učení tak není třeba provádět extrakci rysů trénovací množiny, ale stačí tyto rysy pouze nahrát ze souboru.

Po naučení modelu je možné jednotlivé vektory rysů posílat ke klasifikaci. Výsledkem klasifikace je třída, která nejblíže odpovídá zaslanému vektoru. Avšak problémem SVM metody je nutnost zařadit vektor vždy do jedné z naučených tříd. Metoda si neumí poradit s případem, kdy vektor neodpovídá ani jedné třídě. Tento problém byl vyřešen předřazením druhého SVM modelu, který je založen na metodě binární klasifikace. Tento model využívá pouze jednu jedinou třídu, přičemž klasifikace spočívá v určení, zda vektor rysů do dané třídy spadá, či nikoli. U tohoto modelu je naopak nutné pracovat pouze s jednou jedinou třídou. Řešením tedy bylo všechny třídy z více třídního SVM klasifikátoru převést na jednu jedinou třídu. Pro natrénování binárního klasifikátoru bylo možno využít stejnou trénovací množinu, jaká byla použita pro natrénování vícetřídního klasifikátoru. Jen bylo

třeba upravit všechny třídy pouze na jednu jedinou. Následně tak bylo možno využít binární klasifikátor, který rozhoduje, zda se vektor podobá některému z naučených. Vektory, které model zná (které má naučeny), tak označí jako patřící do dané třídy. Naopak vektory, které reprezentují neznámé zvuky, označí jako nepatřící do dané třídy. Vektory patřící do dané třídy jsou následně předány více třídnímu klasifikátoru, který určí konkrétní třídu zvuku.

7.4.1 Model pro práci se zvukovým signálem

Celý model se skládá z několika dílčích bloků. Každý blok je reprezentován nejméně jednou třídou, která zajišťuje implementaci dané problematiky. V následujícím seznamu jsou uvedeny úlohy jednotlivých bloků.



Obrázek 7.7: Schéma názorně zobrazující vztahy mezi bloky modelu extrakce a klasifikace zvukového signálu.

- `Train` – Představuje rozhraní mezi modelem rozpoznávání a zbytkem mobilní aplikace. Rozhraní je reprezentováno třídou, která připravuje a následně přeposílá trénovací množinu vektoru rysů do SVM klasifikátoru. Toto rozhraní je využíváno jen v případě potřeby natrénování modelu. K volání příslušných metod tak dochází pouze během spouštění aplikace, kdy je třeba zajistit natrénování rozpoznávacích modelů SVM.
- `Store` – Představuje statickou třídu, která zajišťuje načtení a následné parsování souboru s trénovací množinou. Třída je schopna načíst soubor jak z interních zdrojů aplikace (výchozí umístění souboru s trénovací množinou), tak z externího uložiště (uživatelem nahrané zvuky). Třída krom samotného načtení souboru zajišťuje i řádné parsování a kontrolu dat. Všechny korektní položky trénovací množiny jsou následně v třídě uchovány pro další manipulaci.
- `Record` – Jedná se o třídu reprezentující rozhraní mezi extrakcí rysů (včetně klasifikace) a zbytkem aplikace. Třída zároveň zajišťuje přístup k mikrofonu zařízení, tudíž řídí celý proces záznamu zvukového signálu. Signál je uchován pouze po dobu bezprostředně nutnou pro zpracování a klasifikaci. Následně se místo uvolňuje pro zpracování dalších částí signálu. Nedochází tedy k dlouhodobému uložení signálu. Po prvotním

předzpracování a rozdělení signálu do bloků jsou jednotlivé bloky předány třídě MFCC k extrakci. Výsledky v podobě vektoru rysů jsou předány klasifikátoru SVM.

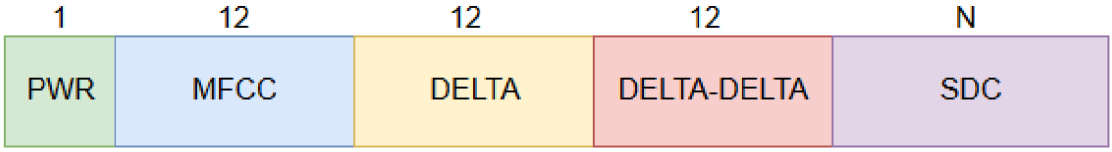
- `MFCC` – MFCC představuje jádro extrakce rysů. Třída přijímá zvukový signál v blocích, které jsou zarovnány na předem zvolený počet rámců. V MFCC nejprve dochází k rozdělení signálu na rámce. Pro každý jednotlivý rámec je spočtena Rychlá Fourierova transformace jejíž výsledkem je spektrum signálu. Následně je provedena samotná extrakce MFCC koeficientů tak, jak byla popsána v kapitole 6.3. V rámci modelu se jedná o nejrizikovější část výpočtu z hlediska doby zpracování. Provádí se tu výpočty násobení rozměrných matic a výpočet DCT pro každý rámec. Pro zajištění co možná nejmenší doby výpočtů bylo třeba používat správné postupy při implementaci a využívat efektivní operace během výpočtu.

- `FFT` – Třída FFT zajišťuje výpočet Rychlé Fourierovy transformace na základě Cooley-Tukey algoritmu. Jedná se o rychlý a efektivní algoritmus, který je založen na dělení transformace v každém kroku na dvě části o velikosti N/2. Velikost výpočtu je tak omezena na mocninu dvou. V rámci aplikace je výpočet prováděn na hodnotu 256. V rámci modelu rozpoznávání se jedná o jeden z klíčových výpočtů, který výrazně ovlivňuje dobu zpracování bloku dat. Proto byl kladen důraz na efektivní implementaci bez zbytečného volání metod a s využitím efektivních operací během výpočtu.

- `SVM` – Jedná se o třídu, která využívá metody a modely z knihovny libSVM pro realizaci klasifikátorů. SVM tak představuje jádro klasifikace. Hlavním úkolem třídy je provést inicializaci a naučení modelu a následně přijímat a zpracovávat požadavky na klasifikaci. Modely klasifikace jsou uchovávány staticky, proto stačí modely naučit pouze při spouštění aplikace. Klasifikaci je pak možno provádět během celé doby práce s aplikací.

7.4.2 Vektor rysů

Pro jednodušší manipulaci se zvukovým signálem je záznam prováděn pouze v mono kanále s přenosovou rychlostí vzorkovanou na 16 kB/s. Zaznamenávaný signál je rozdělen na rámce o velikosti 25ms s překryvem 10ms. Pro každý rámec dojde k extrakci 12 MFCC koeficientu. Těchto 12 koeficientů představuje rysy, které tvoří základ vektoru rysů. Vektor je však možné doplnit ještě dalšími rysy.



Obrázek 7.8: Vzor a rozložení plného vektoru rysů pro jeden rámec s uvedením počtu koeficientů. PWR – energie spektra, MFCC – Mel-kepstrální koeficienty, DELTA – modifikace delta aplikované na MFCC koeficienty, DELTA-DELTA – modifikace delta aplikované na koeficienty DELTA, SDC - Shift Delta koeficienty.

V rámci práce je jako další rys využívána hodnota energie spektra. Dále je využíváno rozšíření MFCC koeficientů v podobě Delta a Delta-delta. Protože MFCC koeficienty se zaměřují pouze na získávání informací z jednoho jediného rámce, je třeba tyto rámce vzájemně

provázat. Rozšíření Delta i Delta-delta zajistí provázání rámců ve zvoleném počtu rámců. Delta generuje opět 12 rysů. Stejná situace je i u Delta-delta. Další rysy je možné generovat pomocí dalšího rozšíření SDC. Velikost SDC rysů je však proměnná na základě nastavení parametrů. Obecně se jedná o násobky poštu rysů MFCC (tedy 12). Ve výsledku úplný vektor rysů (MFCC + Delta + Delta-delta) může mít 37 rysů na každých 25ms záznamu (viz. obrázek 7.8). V případě doplnění o SDC rysy může však dosahovat ještě většího počtu. Zde je však třeba brát v úvahu redundanci a také množství rysů na jeden rámec. Příliš velký počet rysů může mít vliv na počet dimenzí, a tím pádem i na čas potřebný ke klasifikaci. Vliv jednotlivých koeficientů a jejich stupeň významu bude testován v kapitole 8.

7.4.3 Nahrávání uživatelských zvuků

Aplikace krom připraveného modelu poskytuje i možnost nahrání svých vlastních zvuků. Nahrané zvukové vzory jsou pod uživatelem vytvořenou třídou zařazeny mezi vektory rysů a mohou tak být použitý v klasifikaci. Pro zajištění této požadované funkcionality bylo třeba v prvé řadě implementovat správu pro nahrávání zvuků a jejich analýzu. Následně bylo třeba poskytnou správu pro samotné zvuky.

Nejprve se zaměříme na samotné nahrávání zvuků. Po vytvoření nové zvukové třídy je na základě uživatelových akcí vyvoláno dialogové okno pro nahrávání zvuků. Nahrávání zvuku obsluhuje třída `RecordFile`, ve které se nachází asynchronní proces provádějící samotné nahrávání. Jedná se o velmi podobný proces jako je u klasifikace, jen s tím rozdílem, že se zaznamenaný zvuk pouze analyzuje a ukládá. V průběhu nahrávání se zároveň provádí extrakce rysů, které následně reprezentují zaznamenaný zvuk. Jelikož je aplikace uzpůsobena k detekci krátkých zvuků/hlasů, tak je čas nahrávky omezen jen na délku 3 vteřin. Zároveň je prováděna normalizace zvukového signálu s následnou detekcí tichých míst. Na základě experimentálně určeného prahu jsou tichá místa ze signálu odstraněna. Tento krok je zaveden z důvodu, že tichá místa by mohla zkreslovat celkovou klasifikaci. Tichá místa by se mohla ztotožňovat s tichými krátkými úseky během klasifikace a mohlo tak docházet degradaci celkové úspěšnosti klasifikace. Extrahované rysy jsou následně uloženy do souboru s vektory rysů. Samotný signál je uložen ve formátu WAV do externího úložiště. Ukládání původního signálu je především z důvodu následné správy. Jednak si uživatel může kdykoliv poslechnout zaznamenaný vzor a v případě budoucí úpravy či změny způsobu extrakce rysů mohou být všechny rysy znovu extrahovány. Každý zaznamenaný zvukový soubor je pojmenován jako `X-Y`, kde `X` reprezentuje identifikátor třídy zvuku a `Y` značí pořadí zvukového vzoru ve třídě. Ve stejném pořadí jsou ukládány i extrahované rysy v souboru s vektory rysů. Tudíž jakákoliv manipulace s příslušným zvukovým souborem je přímo asociována s konkrétním řádkem v souboru s vektory rysů.

Soubor s vektory rysů je implicitně uložen v interních zdrojích aplikace. Avšak interní zdroje nelze měnit. Proto v případě, že uživatel vytvoří svou vlastní zvukovou třídu, dojde k duplikaci tohoto souboru do externího uložiště. Všechny změny pak probíhají pouze se souborem v externím uložišti, přičemž původní vektor rysů je stále k dispozici v nezměněné podobě v interním uložišti. Toho je možné využít například pro uvedení aplikace do původního nastavení.

Pokud se zaměříme na správu zvukových tříd, tak zde bylo třeba realizovat možnost výběru zvukových tříd, které mají být rozpoznávány. Uživatel má možnost nahrání svých zvuků, přičemž počet nahraných zvukových tříd není nijak omezen. Je však zbytečné, aby všechny zvukové třídy musely být klasifikovány, když je uživatel nebude využívat. Z toho důvodu lze nastavit, které zvukové třídy budou zahrnuty do klasifikace a které nikoliv.

V procesu učení klasifikačního modelu jsou následně do klasifikace zahrnuty pouze ty zvukové třídy, které uživatel zvolil.

Kapitola 8

Experimenty

V následující kapitole jsem se zaměřil především na samotný model extrakce a klasifikace zvuků. S modelem bylo provedeno několik experimentů, které měli určit význam jednotlivých typů rysů a přesnost klasifikace SVM s různými kernel funkcemi.

8.1 Fáze trénování

Před započetím samotných experimentů s modelem bylo nejprve třeba model naučit na určitou množinu zvuků. Modelu bylo také třeba předat informaci o tom, do jaké třídy jednotlivé vzorky trénovací množiny spadají. V prvé řadě bylo tedy třeba sestavit trénovací množinu nahrávek. Nahrávky byly pořízeny na vzorkovací kmitočet 16kHz v mono kanále. Důvodem bylo zajistit dostatek informací v signálu a zároveň umožnit jeho jednoduché zpracování. Nahrávky byly vhodně pojmenovány (do jména nahrávky byla doplněna i informace o třídě) a rozděleny do příslušných tříd. Nad každou nahrávkou byl spuštěn proces extrakce rysů, který zajistil vytvoření vektorů rysů. Vektory byly následně uloženy do souboru (včetně informace o příslušné třídě). Převedení nahrávek na vektory rysů bylo zvoleno takové, jaké bude následně využíváno i v procesu testování. Důvodem uložení rysů do souboru je především ušetření času, který je potřeba k extrakci rysů z nahrávek. Jelikož je třeba při spuštění aplikace vždy znovu provést natrénování klasifikačního modelu, bylo by třeba vždy provádět extrakci rysů z trénovacích dat. Předpřipraveným souborem nám extrakce odpadá a lze rovnou přistoupit k samotnému natrénování klasifikačního modelu. Zároveň se ušetří i místo, neboť velikost souboru s rysy je podstatně menší než velikost všech nahrávek trénovací množiny. Výsledný soubor tedy funguje jako referenční předzpracovaná trénovací množina. Jedinou nevýhodou je nutnost aktualizovat soubor vždy při změně parametrů v extrakci rysů.

V rámci extrakce rysů je možno měnit hned několik parametrů, které ovlivňují jak celkový proces extrakce, tak i následující klasifikaci. Jedná se především o nastavení velikosti rámce a překryvu mezi rámci. Dále je možno vybrat, jaké rysy rámce budou extrahovány a zařazeny do výsledného vektoru. Přičemž jednotlivé typy rysů lze vzájemně kombinovat. Na základě těchto nastavení se mění čas potřebný k extrakci a také se ovlivňuje velikost výsledného vektoru rysů. Nastavení v SVM klasifikaci se odvíjí od zvolené SVM metody a kernel jádra. Obecně se jedná o parametr C, případně nu, který udává penalizaci chyby a určuje hranici mezi korektně klasifikovanými body. Dále parametr $gamma$, což je parametr pro nelineární hyper roviny. Hodnotou se koriguje přesné přizpůsobení modelu trénovací sadě. Parametr, který nás bude dále zajímat je $degree$, což je parametr používaný pouze

v případě polynomiálního kernel jádra. V podstatě určuje stupeň polynomu, který se používá pro nalezení hyper roviny pro rozdělení. Krom samotného nastavování parametrů nás bude zajímat i čas potřebný k natrénování klasifikačního modelu, a především čas potřebný ke klasifikaci bloků rámců.

Trénovací množina v základu obsahuje 244 nahrávek a 10 tříd. 6 tříd reprezentuje jednoduchá slova, která slouží pro základní hlasové ovládání aplikace. Zbylé 4 třídy představují zvuky, které je možní během cvičení zachytit. Délka jednotlivých nahrávek se pohybuje v rozmezí stovek milisekund až po nízké jednotky sekund. Rozložení nahrávek do jednotlivých tříd je uvedeno v tabulce 8.1.

	Trénovací množina	**Testovací množina**
Tlesknutí	24	51
Hvízdnutí	22	54
Odraz míče	25	46
Výstřel	18	45
Start (s)	22	60
Stop (s)	22	59
Pauza (s)	22	60
Další (s)	22	57
Předchozí (s)	23	55
Znovu (s)	22	59

Tabulka 8.1: Rozložení zvukových vzorků pro jednotlivé třídy. (s) - značí, že se jedná o slovo, nikoli o zvuk.

8.2 Fáze testování

Pro otestování modelu bylo třeba sestavit obdobnou množinu jako pro natrénování. Rozložení testovacích nahrávek do jednotlivých tříd je uvedeno v tabulce 8.1. Nahrávky ve svém názvu opět obsahovaly třídu, ke které náležely. Po načtení všech testujících nahrávek se provedla extrakce rysů a vytvoření vektorů. Následně se provedla klasifikace natrénovaným modelem, který každou nahrávku klasifikuje do jedné z deseti tříd. Klasifikované výsledky se následně porovnaly s předpokládanými a spočítala se úspěšnost klasifikace pro danou třídu i pro daný model. Experimentování probíhalo ve dvou směrech. V prvé řadě experimenty probíhaly na testovací množině, která byla po celý čas neměnná. V řadě druhé bylo prováděno kontrolní měření, kdy se úspěšnost klasifikace měřila na počet správně klasifikovaných zvuků a slov v praxi. Testování tedy neprobíhalo na testovací množině, ale měřil se úspěch správně klasifikovaných zvuků zachycených v reálném prostředí. Tento způsob bylo nutno zahrnout z důvodu nasazení klasifikátoru na různých mobilních zařízení. Výsledky těchto testů jsou tedy ovlivněny prostředím, a především samotným mobilním zařízením, na kterém bylo měření prováděno. Těmito experimenty bylo třeba ověřit reálné nasazení modelu do praxe.

Experimentování probíhalo na mobilním zařízení propojeným se softwarem Android Studio, na který byly posílány textové údaje o stavu, časech a výsledcích experimentů. Zařízení, na kterém byly prováděny testy, mělo následující konfiguraci:

Model: Xiaomi Redmi 3S
CPU: Octa-core Max 1.4GHz

RAM: 3,00 GB
OS: Android 6.0.1 (API 23), nádstavba MIUI Global 10.1

8.3 Experiment 1 – porovnání SVM kernel funkcí

Nejdříve jsem se zaměřil na samotný SVM klasifikátor. V rámci této práce jsou pro klasifikaci využívány metody z knihovny libSVM. Knihovna v základu nabízí tři nejrozšířenější a často používané kernel funkce (metody). Jedná se o lineární, radiální a polynomiální funkci, přičemž pro každou z nich je třeba nastavit rozdílné parametry. Funkce se především liší ve způsobu klasifikace a dělení prostoru, ale také v délce učení a v čase potřebném ke klasifikaci. Z toho důvodu byl proveden experiment, který napomůže rozhodnout, která z těchto funkcí je vhodná pro náš model. V rámci tohoto experimentu byla využita celá trénovací i testovací sada tak, jak byla představena v tabulce 8.1. Jako základ extrakce rysů byla použita kombinace energie spektra s dvanácti MFCC koeficienty. Zároveň velikost rozpoznávaného segmentu byla nastavena na velikost 0,51 sekund, tudíž by měla být dostatečně dlouhá, aby obsáhla dostatek informací.

Nastavení modelu	
Vzorkovací frekvence [kHz]	16
Délka rámce [ms]	25
Překrytí rámců [ms]	10
Délka rozpoznávaného segmentu [ms]	510
Celkový počet rysů na rámec [-]	13
Kombinace rysů na rámec [-]	PWR + MFCC
SVM kernel funkce [-]	LINEAR | RADIAL | POLYNOMIAL
Čas extrakce rysů segmentu [ms]	168
Koeficienty SVM	C = 0.89, gamma = 0.001, degree = 3

Tabulka 8.2: Nastavení modelu pro experiment 1.

	LINEAR	**RADIAL**	**POLYNOMIAL**
Tlesknutí	50,9%	56,8%	45,1%
Hvízdnutí	66,6%	68,5%	64,8%
Odraz míče	47,8%	47,8%	41,3%
Výstřel	35,5%	40,0%	26,6%
Start (s)	63,3%	63,3%	58,3%
Stop (s)	62,7%	66,6%	54,2%
Pauza (s)	65,0%	66,6%	46,6%
Další (s)	56,2%	54,4%	42,1%
Předchozí (s)	18.2%	54,5%	32,7%
Znovu (s)	40,7%	47,5%	22,1%
Celkem	**50,7%**	**56,5%**	**43,4%**

Tabulka 8.3: Úspěšnost klasifikace pro jednotlivé kernel funkce.

Z výsledků z tabulky 8.3 vidíme, že každá z metod dosahuje rozdílné úspěšnosti klasifikace a i rozdílných úspěšností v rámci různých tříd. Obecně jsou ale metody poměrně

vyrovnané. Největší úspěšnosti dosáhla metoda radiální, která oproti lineární a polynomiální metodě dosahuje na základním vektoru rysů úspěšnosti 56,5%. Ovšem model musí být schopen provést klasifikaci v reálném čase. Proto nás kromě úspěšnosti klasifikace musí zajímat i doba potřebná pro naučení modelu, a především doba klasifikace segmentu.

	LINEAR	**RADIAL**	**POLYNOMIAL**
Čas naučení modelu [s]	17,52	26,14	24,79
Čas klasifikace segmentu [ms]	72	155	83
Celkový čas zpracování segmentu [ms]	240	323	252
Celková délka rozpoznávaného segmentu [ms]	510	510	510

Tabulka 8.4: Porovnání časů pro jednotlivé funkce.

Z časů vidíme, že ač radiální metoda dosahuje při klasifikaci lepších výsledků, je také zároveň metodou s nejdelšími časy. Klasifikace trvá více než dvojnásobně delší čas než u metody lineární. Zároveň lineární metoda vyžaduje až o třetinu kratší čas k naučení oproti ostatním metodám. Polynomiální metoda sice dosahuje také nízkých časů klasifikace, ale oproti lineární potřebuje delší čas k naučení a i klasifikace je o něco horší. Zároveň si můžeme všimnout, že celkový čas zpracování segmentu je ve všech případech zhruba poloviční oproti délce rozpoznávaného segmentu. Tudíž je zde dostatečný prostor pro zdokonalování modelu. Ve výsledku je pro náš účel nejvhodnější metoda lineární. V úvahu by přicházela ještě metoda radiální, ovšem je třeba brát v úvahu, že se zvýšením počtu rysů stoupne i čas potřebný k naučení modelu.

8.4 Experiment 2 – změna počtu rysů

V následujícím experimentu se zaměříme na vektor rysů, porovnání jednotlivých rysů a vztah mezi velikostí vektoru rysů a časem potřebným pro naučení modelu a klasifikaci. V experimentu bereme jako základ vektoru 12 MFCC koeficientů. Vektor budeme doplňovat o rozšíření Delta, Delta-delta a SDC. Vyzkoušíme také využití SDC koeficientů v různých konfiguracích a jejich vliv na výslednou klasifikaci.

Nastavení modelu	
Vzorkovací frekvence [kHz]	16
Délka rámce [ms]	25
Překrytí rámců [ms]	10
Délka rozpoznávaného segmentu [ms]	510
SVM kernel funkce [-]	LINEAR
Koeficienty SVM	C = 0.89, gamma = 0.001
Parametr zpoždění Delta a Delta-delta [-]	2
SDC parametry [-]	N = 12, d = 2, P = 3 k = 3

Tabulka 8.5: Nastavení modelu pro experiment 2.

Nejprve se zaměříme na porovnání úspěšnosti různých kombinací rysů. V taulce 8.6 si lze všimnout, že už základní varianta (PWR + MFCC) dosahuje celkově 50,7% úspěšnosti. Rozšířením o Delta a Delta-delta dochází k nárůstu úspěšnosti ve všech třídách. Největšího nárůstu si lze všimnout u třídy „Hvízdnutí“, která se dynamikou liší od všech ostatních tříd. Experiment tedy potvrdil, že rozšíření v podobě Delta a Delta-delta má rozhodně přínos a dokáže přispět k lepším výsledkům klasifikace. Pokud však přidáme i SDC rysy, které vektor rysů rozšiřují o dalších 36 rysů, dochází k celkovému poklesu v úspěšnosti. Tento jev způsobuje překrytí rysů Delta s SDC, čímž se zaprvé zanáší významná redundance a v této kombinaci nezpřesňuje výslednou klasifikaci.

	PWR + MFC	**PWR + MFCC + DELTA**	**PWR + MFCC + DELTA + 2DELTA**	**PWR + MFCC + DELTA + 2DELTA + SDC**
Tlesknutí	50,9%	62,7%	66.6%	45,1%
Hvízdnutí	66,6%	85,1%	88,8%	55,5%
Odraz míče	47,8%	54,3%	52,1%	43,4%
Výstřel	35,5%	40,0%	42,2%	31,1%
Start (s)	63,3%	68,3%	65,0%	53,3%
Stop (s)	62,7%	62,7%	66,1%	45,7%
Pauza (s)	65,0%	68,3%	71,6%	53,3%
Další (s)	56,2%	57,8%	63,1%	50,8%
Předchozí (s)	18.2%	30,9%	32,7%	23,6%
Znovu (s)	40,7%	44,0%	45,7%	33,8%
Celkem	**50,7%**	**57,5%**	**59,6%**	**43,6%**

Tabulka 8.6: Úspěšnost klasifikace pro jednotlivé kombinace rysů.

Avšak opět nelze brát pouze samotnou výslednou úspěšnost, ale je třeba ji porovnat i v kontextu časové náročnosti. V tabulce 8.7 vidíme časy potřebné k naučení modelu a časy klasifikace segmentů. Z tabulky lze vyčíst, že s rostoucím počtem rysů na rámec roste nejen čas klasifikace, ale především i čas potřebný k naučení modelu.

Zatímco u základní kombinace (PWR+MFCC) dosahuje model poměrně příznivých hodnot, u kombinace (PWR+MFCC+DELTA+2DELTA), která dosahovala nejlepší úspěšnosti se dostáváme na více než dvojnásobek potřebných časů. U kombinace (PWR + MFCC + DELTA + 2DELTA + SDC) navíc model překračuje celkový čas zpracování segmentu s časem 561 ms. Avšak v tomto experimentu jsou segmenty o velikosti 510 ms. Tím pádem by při použití této kombinace rysů docházelo ke zpoždění a prodlevám, neboť čas zpracování segmentu je delší než čas samotného segmentu.

Z výsledků v tabulce 8.8 vidíme, že samotné SDC koeficienty dokáží nahradit Delta a Delta-delta, i když s poněkud horšími výsledky než v případě Delta a Delta-delta. Zároveň vidíme, že až při konfiguraci (PWR + MFCC + SDC (13, 1, 2, 4)), tedy 61 rysů na rámec dosahuje model stejné úspěšnosti jako konfigurace (PWR + MFCC + DELTA) s pouze 25 rysy na rámec. S vyšším počtem rysů také souvisí i delší časy pro naučení i klasifikaci. Proto pro náš model bude třeba zvolit vhodný kompromis mezi časy (především časem naučení)

	PWR + MFCC	**PWR + MFCC + DELTA**	**PWR + MFCC + DELTA + 2DELTA**	**PWR + MFCC + DELTA + 2DELTA + SDC**
Celkový počet rysů na rámec [-]	13	25	37	73
Čas naučení modelu [s]	17,52	28,03	48,68	104,97
Čas extrakce rysů segmentu [ms]	168	172	178	187
Čas klasifikace segmentu [ms]	72	143	217	374
Celkový čas zpracování segmentu [ms]	240	315	395	561

Tabulka 8.7: Porovnání časové náročnosti zpracování pomocí rozdílných kombinací rysů.

	PWR + MFCC + SDC (12, 1, 2, 2)	**PWR + MFCC + SDC (12, 1, 3, 3)**	**PWR + MFCC + SDC (12, 1, 2, 4)**
Celkový počet rysů na rámec [-]	37	49	61
Čas naučení modelu [s]	50,96	71,21	89,36
Čas extrakce rysů segmentu [ms]	175	176	176
Čas klasifikace segmentu [ms]	209	273	339
Celkový čas zpracování segmentu [ms]	384	449	515
Celková úspěšnost klasifikace [%]	54,3	55,9	57,2

Tabulka 8.8: Porovnání kombinací rozdílných konfigurací SDC v kombinaci s PWR a MFCC rysy. Konfigurace SDC je v zápisu (N, d, P, K).

a mírou úspěšnosti. Jako nejvhodnější se jevila konfigurace (PWR + MFCC + DELTA) s 25 rysy na rámec a časem naučení modelu kolem 28 sekund. Úspěšnost této konfigurace se pohybuje kolem 57%.

8.5 Experiment 3 – změna délky rámce a segmentu

Nyní zjistíme, zda má velikost rámců a segmentu přímý vliv na úspěšnost klasifikace. Použití delších segmentů by mohlo ovlivnit velikost zachyceného vzorku a mohlo by tak vést k lepšímu rozhodnutí, do které třídy patří. Tímto experimentem se tedy pokusíme prokázat, že zvětšením segmentu, či změnou velikosti rámce lze dosáhnou lepších výsledků klasifikace.

Nastavení modelu	
Vzorkovací frekvence [kHz]	16
Délka rámce [ms]	10 | 25 | 35
Překrytí rámců [ms]	2,5 | 10 | 15
Délka rozpoznávaného segmentu [ms]	510 | 680 | 850
Celkový počet rysů na rámec [-]	25
Kombinace rysů na rámec [-]	PWR + MFCC + DELTA
SVM kernel funkce [-]	LINEAR
Koeficienty SVM	C = 0.89, gamma = 0.001
Parametr zpoždění Delta [-]	2

Tabulka 8.9: Nastavení modelu pro experiment 3.

V rámci experimentu došlo ke zvýšení velikosti segmentu ze 510 ms na 680 a 850 ms. Zároveň byly zvoleny tři různé velikosti rámce s rozdílnou velikostí překryvů. Nejprve byl proveden test s referenční velikostí segmentu 510 ms. Z hodnot v tabulce 8.10 je patrné, že změnou velikosti nebo překryvu rámců se úspěšnost nijak výrazně nezměnila. Naopak při menších velikostech rámců se zvýšil počet těchto rámců ve výsledném vektoru, což mělo za následek zvýšení jak času klasifikace, tak i čas potřebný k naučení modelu.

	Délka rámce / Překrytí rámců		
	10 ms / 2,5 ms	**25 ms / 10ms**	**35 ms / 15 ms**
Čas naučení modelu [s]	29,14	27,89	26,32
Čas extrakce rysů segmentu [ms]	187	169	167
Čas klasifikace segmentu [ms]	91	72	76
Celkový čas zpracování segmentu [ms]	278	241	243
Celková úspěšnost klasifikace [%]	57,2	57,5	56,5

Tabulka 8.10: Porovnání časů a úspěšnosti klasifikace pro různé nastavení rámců pro segment o velikosti 510 ms.

Zvětšením velikosti segmentu na 680 ms mělo za následek znatelné zvýšení úspěšnosti pro všechny velikosti rámce. Zvětšením velikosti segmentu došlo i k úměrnému zvýšení časů

potřebných ke zpracování. Největšího zlepšení dosáhla velikost rámce 25 ms s překryvem 10 ms.

	Délka rámce / Překrytí rámců		
	10 ms / 2,5 ms	**25 ms / 10ms**	**35 ms / 15 ms**
Čas naučení modelu [s]	36,78	31,14	29,54
Čas extrakce rysů segmentu [ms]	212	193	190
Čas klasifikace segmentu [ms]	142	94	91
Celkový čas zpracování segmentu [ms]	354	287	281
Celková úspěšnost klasifikace [%]	60,2	63,8	58,8

Tabulka 8.11: Porovnání časů a úspěšnosti klasifikace pro různé nastavení rámců pro segment o velikosti 680 ms.

Zvětšíme-li segment až na velikost 850 ms, uvidíme (viz. tabulka 8.12), že i když stále dochází ke zlepšování úspěšnosti, tak na úkor ceny zpracování se již takto velké segmenty nevyplatí. Navíc zlepšování již není tak znatelné, jako v případě segmentu o velikosti 680 ms.

	Délka rámce / Překrytí rámců		
	10 ms / 2,5 ms	**25 ms / 10ms**	**35 ms / 15 ms**
Čas naučení modelu [s]	48,12	36,14	35,70
Čas extrakce rysů segmentu [ms]	319	249	230
Čas klasifikace segmentu [ms]	212	148	139
Celkový čas zpracování segmentu [ms]	531	397	369
Celková úspěšnost klasifikace [%]	61,4	64,8	59,6

Tabulka 8.12: Porovnání časů a úspěšnosti klasifikace pro různé nastavení rámců pro segment o velikosti 850 ms.

Díky znatelnému zlepšení úspěšnosti v konfiguraci se segmentem o velikosti 680 ms (velikostí rámce 25 ms s překryvem 10 ms), byla tato konfigurace vybrána jako nejvhodnější pro výsledný model. A to i přes to, že došlo k navýšení času potřebného k naučení modelu zhruba o 3 sekundy.

8.6 Experiment 4 – změna počtu tříd a trénovací množiny

V následujícím experimentu vyzkoušíme změnit velikost trénovací množiny a počet tříd, do kterých lze klasifikovat. Zjistíme tak, jaký vliv bude mít změna trénovací množiny vůči

přesnosti klasifikace a také časům zpracování segmentu a naučení modelu. Provedeme dvě úpravy trénovací množiny, kdy nejprve ponecháme počet tříd stejný (tedy 10), ale zvýšíme počet vzorků pro natrénování. V druhém případě ponecháme počet vzorků pro jednotlivé třídy stejný, ovšem zmenšíme počet klasifikovaných tříd.

Nastavení modelu	
Vzorkovací frekvence [kHz]	16
Délka rámce [ms]	25
Překrytí rámců [ms]	10
Délka rozpoznávaného segmentu [ms]	680
Celkový počet rysů na rámec [-]	25
Kombinace rysů na rámec [-]	PWR + MFCC + DELTA
SVM kernel funkce [-]	LINEAR
Koeficienty SVM	C = 0.89, gamma = 0.001
Parametr zpoždění Delta [-]	2

Tabulka 8.13: Nastavení modelu pro experiment 4.

Nejprve provedeme rozšíření trénovací množiny o 50% trénovacích vzorků. Ke každé třídě bylo přidáno 50% jejich aktuálních vzorků, tudíž zůstal zachován poměr mezi třídami. Pokud se podíváme na výsledky měření do tabulky 8.14, tak vidíme, že téměř ve všech třídách se zvedla úspěšnost klasifikace. Bohužel, se zvýšením úspěšnosti se rapidně zvedl i čas potřebný k naučení modelu (viz. tabulka 8.15). Zároveň si můžeme všimnout, že se zvýšil i čas klasifikace segmentu, ovšem v tomto případě se nejedná o tak rapidní nárůst.

	Referenční trénovací množina, 10 tříd	**Trénovací množina zvětšena o 50%, 10 tříd**	**Referenční trénovací množina, 8 tříd**
Tlesknutí	70,5%	68,6%	70,5%
Hvízdnutí	92,5%	94,4%	90,7%
Odraz míče	56,5%	67,3%	-
Výstřel	46,6%	55,5%	55,5%
Start (s)	71,6%	76,6%	70,0%
Stop (s)	67,7%	71,1%	69,4%
Pauza (s)	70,0%	71,6%	70,0%
Další (s)	59,6%	63,1%	-
Předchozí (s)	43,6%	50,9%	43,6%
Znovu (s)	59,3%	67,7%	59,3%
Celkem	**63,8%**	**68,7%**	**65,7%**

Tabulka 8.14: Úspěšnost klasifikace pro jednotlivé změny trénovací množiny.

Druhou úpravou trénovací množiny bylo odstranění náhodných dvou tříd, přičemž u ostatních tříd zůstal počet vzorků původní (viz. tabulka 8.1). V rámci tohoto experimentu chceme ověřit, zda má počet tříd při zachování stejné trénovací množiny vliv na úspěšnost klasifikace. Zároveň chceme zjistit, jak moc se změní časy klasifikace, a především naučení modelu. Z trénovací množiny proto byly odstraněny třídy „Odraz míče“ a „Další (s)“. Z výsledků 8.14

vidíme, že zmenšení počtu tříd nemá zásadní vliv na úspěšnost zachovaných tříd. Naopak v tabulce 8.15 můžeme vidět, že čas naučení modelu se snížil o celých 6 sekund. Paradoxně se však mírně zvýšil čas potřebný ke klasifikaci, což je zapříčiněno jiným rozložením a počtem vektorů v SVM modelu.

	Referenční trénovací množina, 10 tříd	**Trénovací množina zvětšena o 50%, 10 tříd**	**Referenční trénovací množina, 8 tříd**
Počet klasifikovaných tříd [-]	10	10	8
Velikost trénovací množiny [nahrávka]	244	360	197
Čas naučení modelu [s]	31,12	57,06	25,13
Čas klasifikace segmentu [ms]	95	138	107
Celkový čas zpracování segmentu [ms]	288	321	276
Celková úspěšnost klasifikace [%]	63,8	68,7	65,7

Tabulka 8.15: Porovnání časové náročnosti zpracování pomocí rozdílných kombinací rysů.

Z experimentu tedy vyplynulo, že změnou trénovací množiny nebo počtem tříd lze zásadně ovlivnit čas potřebný k naučení modelu. Naopak čas klasifikace je ovlivněn jen minimálně. Zároveň z experimentů vyplívá, že změnou počtu tříd nejsou nijak zásadně ovlivňeny ostatní třídy. Náš model tedy prozatím zůstane v původní konfiguraci, neboť poskytuje optimální kompromis mezi časem potřebným pro přípravu modelu a mírou úspěšnosti.

8.7 Experiment 5 – kontrolní měření v klidném a tichém prostředí

Následující experiment se bude zabývat nasazením modelu do reálného prostředí. SVM klasifikace však dokáže klasifikovat pouze do jedné z definovaných tříd. Tudíž pokud bychom jen čistě integrovali model do aplikace, došlo by k asociaci jakéhokoliv zaznamenaného zvuku s jednou z definovaných zvukových tříd. Z toho důvodu byl definován nový klasifikační model, který klasifikuje pouze do jediné třídy. Jedná se o binární klasifikační model tzv. `SVM one class` klasifikátor, který určuje, zda zaznamenaný zvuk patří do definované třídy či nikoliv. Pro naučení modelu jsou využita všechna testovací data, která v rámci tohoto modelu reprezentují pouze jednu jedinou třídu. Tak lze zajistit, že nahrané zvuky, které jsou podobné těm trénovacím, budou přijaty a zbylé budou odmítnuty. Přijaté zvuky jsou následně předány klasifikačnímu modelu, který již provede klasifikaci do jedné ze zvukových tříd. Po naučení `SVM one class` modelu bylo třeba nejprve ověřit, do jaké míry dokáže přijímat testovací zvuky. Testovalo se tedy pouze kolik procent vzorků z testovacích množiny

model klasifikuje jako přijato. Jako testovací množina byla opět použita testovací množina uvedena v kapitole 8.1, kde všechny třídy byly přeznačeny na jednu jedinou třídu. Vhodné parametry modelu byly zvoleny experimentální cestou.

Nastavení modelu		
	SVM One class	**SVM**
Vzorkovací frekvence [kHz]	16	
Délka rámce [ms]	25	
Překrytí rámců [ms]	10	
Délka rozpoznávaného segmentu [ms]	680	
Celkový počet rysů na rámec [-]	25	
Kombinace rysů na rámec [-]	PWR + MFCC + DELTA	
Parametr zpoždění Delta [-]	2	
SVM kernel funkce [-]	LINEAR	LINEAR
Koeficienty SVM	nu = 0.045 gamma = 0.00001	C = 0.89 gamma = 0.001
Čas naučení modelu [s]	4,39	30,22
Čas klasifikace segmentu [ms]	44	95
Celkový čas zpracování segmentu [ms]	333	

Tabulka 8.16: Nastavení modelů pro experiment 5.

Z výsledků vidíme, že model poměrně dobře určuje, zda se jedná o přijatý zvuk. V tomto bodě bylo provedeno kontrolní měření, kdy bylo simulováno provádění zvuků během cvičební sady. Měření spočívalo v postupném vydávání zvuků/hlasů a měření, zda aplikace zvuk/hlas zaznamenala a klasifikovala do správné třídy. Měření se zúčastnili celkem 4 účastnící (2 mužské a 2 ženské hlasy).

	Úspěšnost přijetí klasifikace modelu SVM one class na testovacích datech
Tlesknutí	79,5%
Hvízdnutí	95,1%
Odraz míče	73,4%
Výstřel	85,9%
Start (s)	77,7%
Stop (s)	73,4%
Pauza (s)	82,7%
Další (s)	70,4%
Předchozí (s)	69,1%
Znovu (s)	71,3%
Celkem	**77,9%**

Tabulka 8.17: Úspěšnost přijetí modelem SVM one class.

Celkem bylo provedeno 100 měření pro každou třídu. Měření bylo prováděno v uzavřené tiché místnosti bez žádných rušivých elementů. Zároveň byly vydávány pouze zvuky, které aplikace dokáže detekovat. Tímto kontrolním měřením bylo třeba ověřit spřažení obou modelů a jejich míru úspěšnosti v reálném prostředí.

	Detekováno	Nedetekováno	Detekováno jako jiná třída
Tlesknutí	62	21	17
Hvízdnutí	81	15	4
Odraz míče	57	21	22
Výstřel	59	13	28
Start (s)	63	22	15
Stop (s)	63	26	11
Pauza (s)	67	14	19
Další (s)	63	25	12
Předchozí (s)	50	28	22
Znovu (s)	59	24	17
Celkem	**62,5%**	**20,9%**	**16,7%**

Tabulka 8.18: Úspěšnost klasifikace pro 100 kontrolních měření každé třídy v prostředí bez okolního rušení.

Z výsledků v tabulce 8.18 je vidět, že úspěšnost reálně klasifikovaných tříd je kolem 60%. 20% tvoří zvuky, které byly odmítnuty již prvním klasifikačním modelem `SVM one class`. Necelých 20% měření první klasifikační model rozpoznal správně, avšak byly chybně zařazeny do výsledné třídy. Ve výsledcích si lze všimnout poměrně dobré klasifikace třídy „Hvízdnutí“. To je zapříčiněno specifickou dynmaikou zvuku, který se značně liší od ostatních zvuků. Jde vidět, že zvuky byly spíše nedetekovány, než detekovány chybně. Kontrolní měření potvrdilo, že experimentálně naměřené úspěšnosti klasifikace jsou dosažitelné i reálném nasazení. jen minimálně.

8.8 Experiment 6 – kontrolní měření v prostředí s rušivými elementy

Předchozí kontrolní měření ukázalo nasazení klasifikačních modelů v reálném prostředí. V následujícím kontrolním měření se opět pokračovalo v ověřením použití modelu v reálném prostředí. Tentokrát byly do prostředí zahrnuty rušivé elementy v podobě hudby či mluvené řeči. Nastavení modelu zůstalo stejné jako v experimentu 5 (viz. 8.16). Měření opět spočívalo ve vydávání zvuků/hlasů čtyřmi účastníky měření. Pro každou třídu bylo provedeno 100 kontrolních měření. Úroveň hluku byla rozdělena do dvou kategorií. První kategorii tvořilo okolní rušení v rozmezí do 40 dB, což odpovídá slabě puštěné hudbě nebo normální řeči. Druhou kategorii tvořilo rušení v rozsahu 40-65 dB, což odpovídá hlasité hudbě nebo hlasité řeči. Cílem bylo zjistit, do jaké míry ovlivňují rušivé elementy výslednou klasifikaci.

Na základě měření bylo zjištěno, že rušení do 40 dB má sice vliv na kvalitu rozpoznávání, ovšem ve většině případů docházelo spíše k neúspěšné detekci, než že by byl zvuk špatně klasifikován. Rušení má tedy vliv spíše na `SVM one class` model.

Oproti tomu u rušení do 65 dB došlo k výraznému nárůstu chybně detekovaných zvuků. V tomto případě však velice záleželo na typu rušení. Pokud se jednalo o detekci zvuku (např. odraz míče), reagovala detekce na rytmickou hudbu, naopak hudba neměla významný vliv na chybnou detekci hlasových vzorků. Oproti tomu konverzace byla chybně detekována jak ve třídách s hlasy, tak i se zvuky. Zároveň došlo k nárůstu samovolné detekce, tedy

	Detekováno	Nedetekováno	Detekováno jako jiná třída
Tlesknutí	59	23	18
Hvízdnutí	76	18	6
Odraz míče	58	23	19
Výstřel	54	16	30
Start (s)	62	25	13
Stop (s)	60	24	16
Pauza (s)	63	18	19
Další (s)	62	27	11
Předchozí (s)	49	30	21
Znovu (s)	56	26	18
Celkem	**59,9%**	**23,0%**	**17,1%**

Tabulka 8.19: Úspěšnost klasifikace pro 100 kontrolních měření každé třídy v prostředí s rušením do 40dB.

	Detekováno	Nedetekováno	Detekováno jako jiná třída
Tlesknutí	49	22	29
Hvízdnutí	71	22	7
Odraz míče	58	24	27
Výstřel	41	18	41
Start (s)	57	22	21
Stop (s)	52	24	24
Pauza (s)	56	17	27
Další (s)	57	24	19
Předchozí (s)	43	29	28
Znovu (s)	48	24	28
Celkem	**52,3%**	**22,6%**	**25,1%**

Tabulka 8.20: Úspěšnost klasifikace pro 100 kontrolních měření každé třídy v prostředí s rušením do 65dB.

detekce některého z rušivých zvuků. Aplikace tak reagovala i v případě, že žádný zvuk nebyl uživatelem vydán. V tabulkách 8.18, 8.19 a 8.20 si lze však všimnout, že zvuk hvízdnutí dosahuje ve všech měřeních poměrně vysoké úspěšnosti klasifikace a zároveň není výrazně ovlivněn rušením. Důvodem je, že zvuk hvízdnutí je tak specifický oproti jiným zvukům, že na něj okolní zvuky, ani rušení nemá významný dopad. Naopak zvuky jako je odraz míče a výstřel jsou velice podobné zvuky, a tudíž byly častěji klasifikovány chybně.

Z výsledků měření tedy vyplynulo, že aplikace je schopna operovat i v prostředí s malým rušením. Ovšem v případě, že se rušení zvýší, dochází k značnému nárůstu chybně klasifikovaných zvuků a ke snížení úspěšnosti klasifikace. Zároveň také záleží na charakteru rušení, kdy mluvená řeč v blízkosti zařízení může ovlivňovat detekci hlasů a naopak puštěná hudba v pozadí může ovlivňovat detekci zvuků.

Všechny experimenty se prováděly s co možná největším důrazem na objektivitu měření a výsledků. Avšak u experimentů 5 a 6 je třeba brát v úvahu, že výsledky jsou ovlivněny okolními vlivy, jako je typ zařízení, typ mikrofonu, kvalita odhlučnění prostředí nebo i výslovnost a hlasitost vydávaných zvuků. Značnou roli také hraje způsob umístění mobilního zařízení do prostoru, respektive natočení mikrofonu ke zdroji zvuku.

8.9 Shrnutí a výsledný model

Pomocí experimentů bylo ověřeno chování modelu pro různé varianty konfigurací. Pomocí experimentů a kontrolních měření bylo možno odhalit vhodné nastavení modelů a dosáhnout tak optimální úspěšnosti klasifikace za optimální čas. Z důvodu požadavku na zpracování zvuků v reálném čase, hrál velmi důležitou roli ve výsledném nastavení i čas potřebný ke klasifikaci, extrakci a naučení modelu. Pomocí experimentů se ověřilo, že jeden SVM klasifikační model dokáže klasifikovat jak zvuky vokálního charakteru, tak i zvuky nevokálního charakteru.

Na základě provedených experimentů bylo možno sestavit výsledný model pro rozpoznávání zvuků v mobilní aplikaci. Z experimentu 8.3 vyplynulo, že ideální kernel funkcí pro výsledný SVM klasifikační model by byla radiání funkce. Avšak pouze za několika procentní zvýšení úspěšnosti bychom zaplatili dvojnásobným prodloužením času klasifikace a téměř dvojnásobným navýšením času potřebným k naučení modelu. Proto je v rámci výsledného modelu zvolena funkce lineární, která má sice horší výslednou klasifikaci, ale dosahuje mnohem přijatelnějších časů. Na základě experimentu 8.4 byl zvolen výsledný vektor rysů pro časový rámec. Jako ideální kombinace rysů se vzhledem k časovému zpracování jevila kombinace PWR + MFCC + DELTA (25 rysů na rámec). Především se u zvolené kombinace dosahovalo poměrně přijatelných časů pro naučení modelu. Experiment {exp3 ukázal, že změnou velikosti rámců, či jejich překryvů dochází pouze k zanedbatelným změnám, oproti referenčnímu nastavení modelu (délka rámce 25ms s překryvem 10ms). Naopak změna velikosti segmentu z 510ms na 680ms přinesla zlepšení výsledné klasifikace (o přibližně 6%). Přitom čas naučení modelu se zvýšil pouze o 3 sekundy a čas zpracování segmentu o necelých 50ms. Pomocí experimentu 8.6 se určil vliv velikosti trénovací množiny na dobu potřebnou pro naučení modelu a úspěšnost klasifikace. Experimentem bylo také určeno, kolik času bude vyžadováno pro naučení modelu rozšířeného o novou třídu. Tento údaj je důležitý především z důvodu, že aplikace umožňuje vložení vlastní množiny vzorku zvuků. Je třeba proto zajistit, že nahraje-li uživatel nové zvuky, bude model stále schopen naučení v přijatelném čase. Experimenty 8.7 a 8.8 ověřili celkovou funkčnost modelu v reálném prostředí. Zároveň byl určen vliv okolních zvuků a rušení na výslednou klasifikaci. Výsledné parametry modelu integrovaného do mobilní aplikace lze nalézt v následující tabulce 8.21.

Nastavení modelu		
	SVM One class	**SVM**
Vzorkovací frekvence [kHz]	16	
Délka rámce [ms]	25	
Překrytí rámců [ms]	10	
Délka rozpoznávaného segmentu [ms]	680	
Celkový počet rysů na rámec [-]	25	
Kombinace rysů na rámec [-]	PWR + MFCC + DELTA	
Parametr zpoždění Delta [-]	2	
SVM kernel funkce [-]	LINEAR	LINEAR
Koeficienty SVM	nu = 0.045 gamma = 0.00001	C = 0.89 gamma = 0.001
Čas naučení modelu [s]	30,24	4,42
Čas klasifikace segmentu [ms]	43	94
Celkový čas zpracování segmentu [ms]	331	

Tabulka 8.21: Výsledná konfigurace modelů integrovaných do mobilní aplikace.

Vidíme, že model dosahuje poměrně dlouhého času naučení. Tento čas bude uživatel nucen čekat, než bude schopen plnohodnotně pracovat s aplikací (provádět sadu cviků). Jelikož přidáváním uživatelských zvuků by se čas učení ještě více prodloužil, byla množina aktivních zvuků v aplikaci omezena na 5. Aplikace tak bude schopna v jeden okamžik detekovat až 5 zvuků a 6 slovních příkazů pro ovládání aplikace. Uživatel bude mít stále možnost nahrávat a přepínat aktivní zvuky, dle jeho preferencí. Vestavěné zvuky budou moci být deaktivovány, čímž se uvolní místo pro nahrání dalších (vlastních zvuků). Toto omezení zajistí, že časy učení modelu nepřekročí takový časový limit, který by byl pro běžné použití neúnosný.

Kapitola 9

Testování

Následující kapitola shrnuje postup testování výsledné aplikace. Jsou zde vyhodnoceny zjištěné výsledky a na jejich základě stanoveny závěry a provedeny případné úpravy aplikace.

9.1 Ověření aplikace na různých verzích API

Jak již bylo zmíněno v kapitole 2.4, aplikace by měla podporovat všechny verze Android API vyšších, než verze Android 4.4 (API 19). Samotný vývoj aplikace byl průběžně ověřován na fyzických zařízení s verzí 6.0 a 9.0. Nicméně i když na obou zařízeních aplikace nevykazovala neobvyklé chování, bylo třeba ověřit, zda se aplikace chová korektně i na ostatních podporovaných verzích API. Testování aplikace probíhalo na emulátoru v rámci vývojového prostředí Android Studio. Testování spočívalo v postupném spouštění aplikace na jednotlivých verzích API. Přičemž hlavní pozornost směřovala na správné vykreslování grafického uživatelského rozhraní a správnou odezvu prováděných akcí. Během testování bylo odhaleno několik odchylek ve vykreslování grafických prvků. Jednalo se především o rozložení prvků a zarovnání. Problémové prvky byly opraveny nebo byly využity jiné způsoby řešení, u kterých se problémy již nevyskytovaly. Během ověřování tak, až na drobnosti, nebyly zaznamenány žádné větší problémy.

9.2 Testování grafického uživatelského rozhraní

Již během návrhu a vývoje aplikace bylo třeba do jisté míry otestovat, zda rozhodnutí a především rozvržení grafických prvků je vhodně zvoleno. Ověřování v těchto fázích je důležité, neboť dokáže odhalit velké nedostatky již v počátcích vývoje a případné úpravy jsou méně náročné. Proto již během vývoje byl testujícím uživatelům předložen návrh jednotlivých obrazovek aplikace. Nad každou z nich proběhla krátká diskuze týkající se rozložení prvků, volby barev či velikosti písma.

Diskuze se zúčastnilo celkem 7 uživatelů. Na základě poznatků plynoucích z diskuzí došlo k drobným úpravám v grafickém návrhu aplikace. Mezi jednu z nejznatelnějších úprav patřilo barevné oddělení cvičebních sad na základě skupiny, do které sada patří. Tato změna tak umožnila rychle graficky odlišit jednotlivé cvičební sady, což vedlo k lepší orientaci uživatele ve cvičebních sadách. Dále se jednalo především o drobné úpravy v podobě velikosti písma, zarovnání a umístění jednotlivých grafických prvků. Díky testování v raných fázích vývoje bylo možno uvedené změny promítnou již v době návrhu aplikace.

9.2.1 Testování použitelnosti aplikace

Testování použitelnosti bylo rozděleno do následujících tří etap:

1. Výběr vhodného vzorku uživatelů a sestavení sady úloh – na základě screeneru byl vybrán optimální vzorek uživatelů. Zároveň došlo k vytvoření sady scénářů, na kterých se ověří fungování aplikace.

2. Testování úloh na skupině uživatelů – vhodné skupině uživatelů byla předložena sestavená sada úloh. Úkolem uživatelů bylo vyplnit jednotlivé úlohy. Během provádění úloh byli uživatelé pozorováni a byly zaznamenávány jejich reakce.

3. Vyhodnocení testů – během testování byly zaznamenány jak pozitivní, tak i negativní ohlasy a reakce. Nad problémy, se kterými se uživatelé potýkali, byly vedeny krátké diskuze. Ze získaných poznatků byly vyvozeny závěry a navrženy opravy nedostatků.

Screener

Než bylo přistoupeno k samotnému testování aplikace, bylo třeba zvolit vhodnou skupinu uživatelů. Cílem bylo získat takové uživatele, kteří by reprezentovali cílovou skupinu, pro kterou je aplikace primárně určena. K tomuto účelu byl sestaven screener.

Jedná se o typ dotazníku, na základě kterého lze rozhodnout, zda potenciální účastník testu spadá mezi zástupce uživatelů. Úlohou screeneru je vyřadit uživatele, kteří nespadají mezi cílovou skupinu uživatelů nebo mají znalosti, které by mohly ovlivnit výsledky testů. Screener má podobu jednoduchého dotazníku, který obsahuje několik otázek s několika možnostmi. Předem určené možnosti jednotlivých otázek přitom rozdělují uživatele do dvou (vhodný, nevhodný) nebo i více skupin.[21]

Screener, který byl poskytnut potenciálním účastníkům testování je zahrnut v příloze A. Každá otázka má předem definované správné volby, které prokazují způsobilost uživatele k testování. Mezi takové odpovědi patřilo hlavně aktivní využívání chytrého mobilního telefonu s platformou Android a aktivní provozování nějakého sportu. Uživatelé, kteří na všechny otázky odpověděli tak, že žádná z možností nespadala do kategorie „nevhodný“, byli vhodnými kandidáty pro následné testování samotné aplikace.

Testování založené na sadě úloh

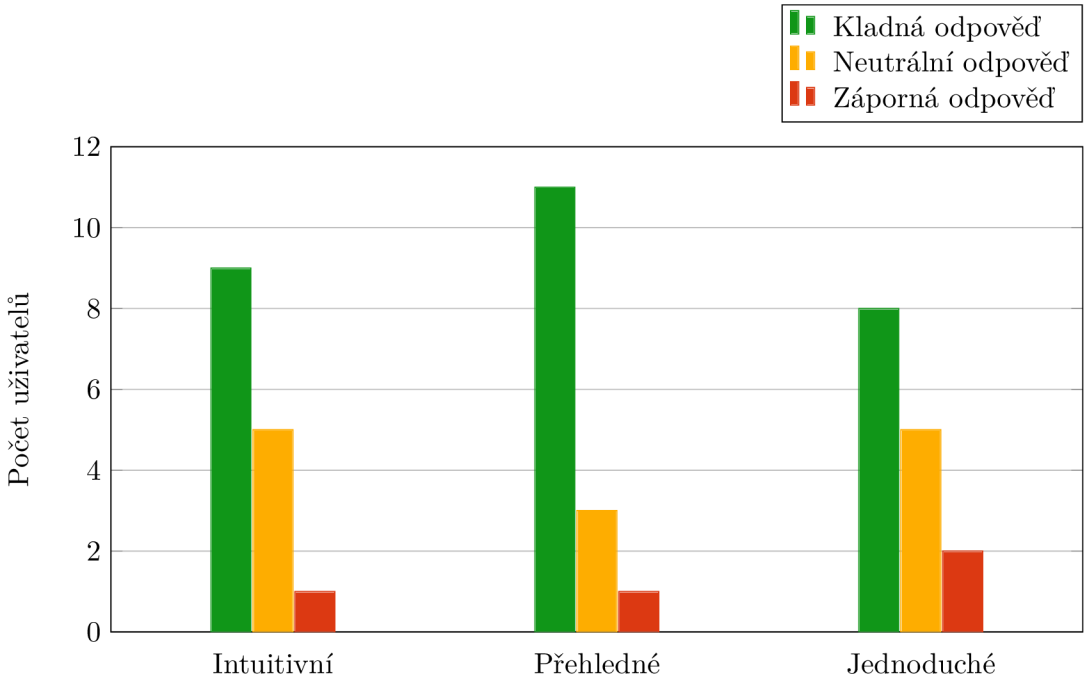
Po výběru vhodné skupiny uživatelů pro testování se přistoupilo k samotnému procesu testování aplikace. Testování probíhalo pomocí dotazníků. Uživatel nejprve vyplnil obecné informace, které sloužily k lepšímu vyhodnocování. Následně se přistoupilo k sadě úloh, které měl uživatel vykonat. Jednalo se o projití celé aplikace a vykonání všech klíčových akcí, které by prováděl běžný uživatel. Během provádění úloh byly pozorovány a zaznamenávány reakce uživatele. Pro lepší pochopení uživatelových pocitů a smýšlení, měl uživatel za úkol verbálně popisovat své pocity a aktuální akce, nad kterými uvažuje. Ve spojení s vlastními poznámkami jsem tak dostal ucelený přehled, jak uživatel s aplikací pracuje a jak je pro něj aplikace pochopitelná. Po provedení úloh byla vedena krátká diskuze o úkolech, aplikaci a celkovém názoru uživatele. Na závěr uživatel vyplnil několik otázek týkajících se subjektivního názoru a ohodnotil aplikaci z různých aspektů. Vzor sestaveného dotazníku se seznamem úloh je k nahlédnutí v příloze B.

Během všech testů jsem byl jakožto moderátor testování přítomen a do samotného procesu testování jsem nijak nezasahoval, až na výjimky, kdy si uživatel opravdu nevěděl

rady. Testování probíhalo v klidném prostředí bez zbytečných rušivých elementů. Zároveň žádný ze započatých testů nebyl nijak neočekávaně přerušen okolními vlivy či situacemi.

Vyhodnocení testování

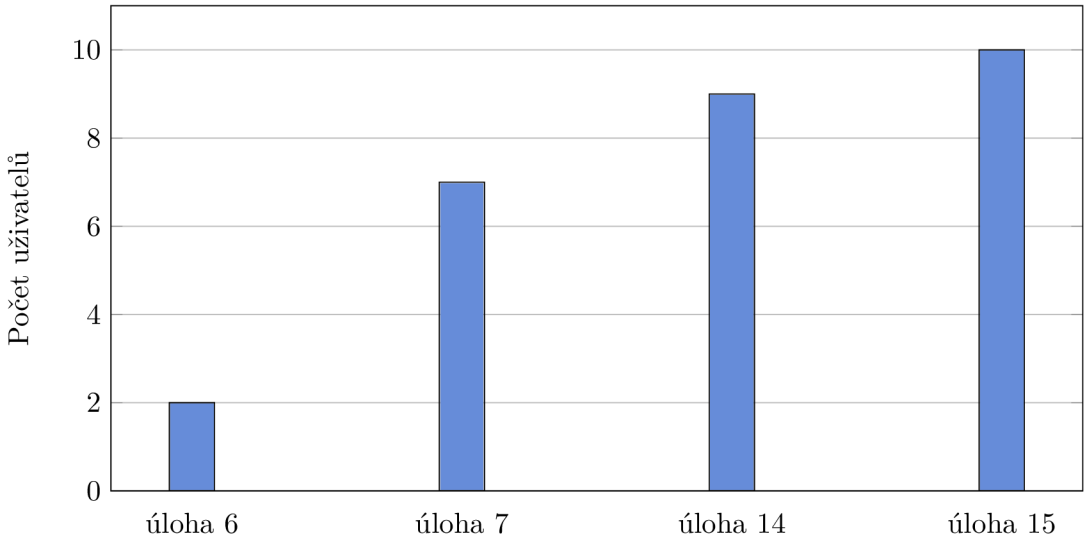
Testování se zúčastnilo celkem 15 uživatelů. Po provedení sady úloh byly uživatelům předloženy otázky týkající se intuitivnosti a celkového pocitu z aplikace. Uživatelé měli subjektivně zhodnotit přívětivost, přehlednost a intuitivnost testované aplikace. Odpovědi uživatelů jsou zaneseny v grafu 9.1. Z odpovědí plyne, že uživatelské rozhraní aplikace lze označit za srozumitelné a přehledné. Aplikace disponuje poněkud složitějším modelem vkládání drilů, což na jednu stranu přináší dostatečnou míru variability, ale na druhou stranu zavádí nové pojmy a způsoby nastavení. Uživatelé, kteří již měli zkušenosti z jiných aplikací s podobným zaměřením, byli zvyklí na jednoduchý (nevariabilní) způsob vkládání drilů. Možnost variability drilu jim tak zpočátku činila drobné potíže, než se zorientovali ve výrazech nebo pochopili účel nastavení. K tomu jim však dopomohla i rychlá nápověda, která vysvětluje princip a využití prvků na konkrétních obrazovkách. Avšak i v nápovědě nebyla některá nastavení dostatečně vysvětlena. Proto byly v nápovědě provedeny drobné úpravy a lepší formulace pojmů. Zároveň byla nápověda rozšířena o názorné obrázky.



Obrázek 9.1: Volba odpovědí uživatelů na otázku jednoduchosti, přehlednosti a intuitivnosti aplikace.

Celkově na otázku, zda by uživatelé uvažovali i nad dalším využíváním aplikace, zhruba 70% dotazovaných odpovědělo kladně. Tím tak byl splněn jeden z požadavků, který si dávala aplikace za cíl. Nabídnout srozumitelnou mobilní aplikaci, která by podporovala cvičební procesy.

Díky moderovanému testování bylo možno důkladněji otestovat použití a funkčnost aplikace. Uživatelé měli možnost vyzkoušet si na vlastní kůži práci s aplikací a posoudit její využití v praxi. Zároveň tím napomohli k odhalení nedostatků a problémů, které by



Obrázek 9.2: Úlohy, se kterými měli uživatelé největší problémy.

mohli mít vliv na plynulost práce s aplikací a její praktické využití. Jak už bylo uvedeno, testování bylo založeno na procházení jednotlivých úloh. Graf 9.2 uvádí úlohy, které dělaly uživatelům největší potíže nebo u kterých byly odhaleny nedostatky. V následujících bodech jsou uvedeny ty největší nedostatky, které bylo třeba řešit.

Úloha 14 – Vytváření drilu – volba skupiny

Během vytváření drilu má uživatel možnost zvolit skupinu, do které má nově vytvářený dril patřit. Během testování bylo však poukázáno na chování selektoru. Selektor seznamu skupin totiž nebere v úvahu skupinu, ve které se aktuálně nachází cvičební sada, v rámci které se dril vytváří. Toto chování není přímo chybou, ale vede k matení uživatelů. Zároveň se tento problém týká pouze drilů, které jsou explicitně uloženy jako koncepty. Během vytváření drilu se uživatel příliš nesoustředí na výběr skupiny. Jednoduše předpokládá, že ukládaný dril je automaticky přiřazen do patřičné skupiny, pod kterou spadá aktuální cvičební sada. Při následném hledání vytvořeného drilu v seznamu uložených drilů tak dochází ke zmatení, neboť uživatel vytvořený dril hledá pod jinou skupinou.

Řešení Řešení bylo poměrně jednoduché. Seznam již obsahuje všechny skupiny vyskytující se v aplikaci. Bylo tedy třeba zajistit nastavení selektoru na skupinu v rámci které se nový dril vytváří. Informace o aktuální skupině je již předávána aktivitě zajišťující vytvoření nového drilu. Stačilo tedy informaci o aktuální skupině promítnout do selektoru seznamu skupin.

Úloha 14 – Editace uložených skloňovaných slov

V případě rychlé modifikace drilu je uživateli nabídnuta možnost skloňování slov. Pokud se v drilu za rychle modifikovaným číslem vyskytuje slovo, je toto slovo možno skloňovat právě ve vztahu k modifikované hodnotě. Příkladem může být dril „1 krok vpřed“. Dril je nastaven jako rychle modifikovatelný a je zaznačena možnost skloňování slov. Pokud uživatel v rámci seznamu drilů modifikuje číslo 1 např. na 2, výsledný příkaz drilu bude

vypadat následovně „2 kroky vpřed“. Aplikace v sobě obsahuje databázi uložených slov a jejich tvarů. Ovšem aplikaci lze využívat pro různé sporty a v rozdílných kontextech. Z toho důvodu, pokud aplikace slovo nezná, požádá uživatele o vyplnění tvarů tohoto slova. Slovo a jeho tvary jsou uloženy do databáze a následně využívány. Během testování došlo ve dvou případech ke stavu, kdy uživatel vyplnil některý z tvarů slov špatně. Bohužel aplikace nenabízí jakoukoliv modifikaci těchto slov. Proto v případě, že uživatel jednou vloží špatný tvar slova, není možnost tento tvar opravit.

Řešení Do obrazovky nastavení aplikace byla vložena položka odkazující na samostatný seznam slov, které uživatel vložil. V seznamu jsou zobrazeny pouze slova, která byla přidána uživatelem. Výběrem položky je uživateli zobrazen dialog, ve kterém může slovo a jeho tvary modifikovat.

Obrázek 9.3: Řešení editace skloňování slov v nastavení aplikace.

Úloha 7 a 15 – Signalizace zaznamenaného zvuku

Se samotným procesem provádění cvičební sady nebyl ze strany uživatelů žádný větší problém. Na čem se však shodla převážná většina testujících uživatelů, je signalizace zaznamenaného zvuku/příkazu. Aplikace totiž provede detekci zvuku (např. slovo pauza) a provede patřičnou akci (např. pozastavení aplikace). Uživatel však neví, jestli aplikace zvuk opravdu zaznamenala nebo pokračuje ve vykonávání drilů. Vezměme v úvahu případ, kdy uživatel vydá příkaz v době, kdy aplikace čeká před přechodem na další dril. Aplikace příkaz zaznamená, ovšem uživatel neví, zda byl příkaz skutečně přijat nebo pokračuje čekání (odpočítávání) před přechodem na další dril. V tomto případě dochází ke stavům, kdy uživatel buď počká, jestli se ozve příkaz následujícího drilu nebo se podívá na obrazovku zařízení. V obou případech tak dochází k narušení procesu cvičení. Vzhledem k faktu, že model rozpoznávání hlasu není dokonalý, může tento stav nastat během cvičení poměrně často. Z pohledu uživatele se tak jedná o neintuitivní chování, které narušuje celkovou práci s aplikací. V krajních případech pak může toto nešťastné chování rozhodovat i o dalším setrvání uživatele u této aplikace.

Řešení Uživateli bylo třeba poskytnout signalizaci o úspěšném zaznamenání zvuku/hlasu, a to ve vhodné formě. V úvahu nepřipadala signalizace vizuální, neboť by uživatele nutila sledovat obrazovku zařízení. Jako nejvhodnější řešení se jevilo využít signalizaci zvukovou. Do aplikace byl proto zakomponován krátký signalizační zvuk. Ten je následně přehrán při každém správně zaznamenaném zvuku/hlasu. Uživatel je tak tímto způsobem informován o reakci aplikace. Navíc slovní příkazy „pauza“ a „start“ jsou doplněny slovní informací

o pozastavení či spuštění aplikace. Uživatel je tedy informován nejen o zaznamenání hlasu, ale také o stavu procesu cvičení.

9.2.2 Shrnutí

Testování lze označit za úspěšné, neboť napomohlo k odhalení drobných nedostatků, které byly následně vyřešeny nebo odstraněny. Zároveň byla ověřena funkčnost a použitelnost aplikace. Uživatelům se úkoly jevily jako jednoduché a až na několik málo případů nebyl problém s jejich vykonáním. Některé úlohy byly pro uživatele trochu časově náročnější, jelikož se jednalo o první seznámení s aplikací. Při několika kontrolních opakování testů se potvrdilo, že uživatel již ví, co má provádět a při opakování úloh se již nevyskytl žádný problém. Obecně byla práce s aplikací ohodnocena jako intuitivní a až na některé drobnosti byla označena za jednoduchou. Tím bylo potvrzeno splnění cíle aplikace. Testování se zúčastnili jak uživatelé pohybující se v prostředí moderních technologií a sportu, tak i průměrní uživatelé chytrých mobilních telefonů.

Kapitola 10

Závěr

Cílem této práce bylo navrhnout a implementovat mobilní aplikaci na platformě Android, která by poskytla podporu v oblasti cvičení. Přínosem aplikace má být usnadnění provádění cvičebních úkonů pomocí hlasových příkazů a detekování a rozpoznávání hlasů či zvuků. Aplikace, krom jiného, nabízí uživateli možnost vytvářet vlastní sady zvuků, a tím tak ještě více přizpůsobit proces cvičení přesným potřebám uživatele. Dle výsledků návrhu, experimentů a uživatelského testování byl tento cíl splněn.

Pro dosažení výsledků bylo třeba provést průzkum aktuálně existujících aplikací a analyzovat jejich funkcionalitu a využitelnost. Na základě získaných poznatků byla vytvořena specifikace aplikace, která do funkcionality zahrnovala i detekci a rozpoznávání zvuků. Bylo zapotřebí nastudovat jednotlivé přístupy a algoritmy pro detekci a rozpoznávání zvuků a následně vybrat vhodný pro implementaci. Zároveň pro následnou realizaci mobilní aplikace bylo zapotřebí nastudovat i pravidla správného návrhu uživatelského rozhraní pro platformu Android. Během finálních experimentů a testování se ověřila jak funkčnost modelu rozpoznávání, tak i funkčnost a přehlednost uživatelského rozhraní. Výsledná aplikace je k dispozici v Google Play Store pod označením Voice sport.

V závislosti na oblíbenosti aplikace a zpětné vazby od uživatelů by mohla být aplikace v budoucnu rozšířena o vzdálený server, který by umožňoval pohodlně vytvářet uživatelské profily, sdílet cvičební sady i statistiky přímo z mobilní aplikace. Taktéž by bylo možné umísti na server klasifikační model, a tak částečně eliminovat dlouhé časy potřebné pro jeho naučení. Budoucí plány budou však úzce souviset s aktuálními trendy, a především s hodnocením a zpětnou vazbou uživatelů z obchodu Google Play Store.

Literatura

[1] *Android API Levels*. Vanderbilt Android Developers.
URL <http://www.dre.vanderbilt.edu/~schmidt/android/android-4.0/out/target/common/docs/doc-comment-check/guide/appendix/api-levels.html>

[2] *Dashboards*. Android Developers.
URL <https://developer.android.com/about/dashboards/index.html>

[3] *Klatt Synthesizer*. Berkeley Linguistics.
URL <http://linguistics.berkeley.edu/plab/guestwiki/index.php?title=Klatt_Synthesizer>

[4] *Platform Architecture*. Android Developers.
URL <https://developer.android.com/guide/platform/>

[5] *Review of Speech Synthesis Technology*. Helsinki University of Technology.
URL <http://research.spa.aalto.fi/publications/theses/lemmetty_mst/chap6.html>

[6] *Support Library*. Android Developers.
URL <https://developer.android.com/topic/libraries/support-library/>

[7] Chang, C.-Y.; Chang, Y.-P.: Application of abnormal sound recognition system for indoor environment. In *2013 9th International Conference on Information, Communications Signal Processing*, Dec 2013, s. 1–5, doi:10.1109/ICICS.2013.6782772.

[8] Cowling, M.: Non-Speech Environmental Sound Classification System for Autonomous Surveillance. 03 2004.

[9] Cowling, M.; Sitte, R.: Comparison of techniques for environmental sound recognition. *Pattern Recognition Letters*, ročník 24, č. 15, 2003: s. 2895 – 2907, ISSN 0167-8655, doi:https://doi.org/10.1016/S0167-8655(03)00147-8.
URL <http://www.sciencedirect.com/science/article/pii/S0167865503001478>

[10] Daniel Jurafsky, J. H. M.: Hidden Markov Models. 11 2018.

[11] Dutoit, T.: *An Introduction to Text-to-Speech Synthesis*. Springer, 1997, ISBN 0-7923-4498-7.

[12] Gandhi, R.: *Support Vector Machine — Introduction to Machine Learning Algorithms*. Towards Data Science.

URL <https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47>

[13] Johnson, J.: *Designing with the Mind in Mind*. Morgan Kaufmann Publishers, 2010, ISBN 978-0-12-375030-3.

[14] Luboslav Lacko: *Vývoj aplikací pro Android*. Computer Press, 2015, ISBN 978-80-251-4347-6.

[15] Lyons, J.: *Continuous wavelet transform* . [Online; navštíveno 09.12.2018].
URL <http://klapetek.cz/wcwt.html>

[16] Lyons, J.: *Mel Frequency Cepstral Coefficient (MFCC)*. [Online; navštíveno 08.12.2018].
URL <http://practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/>

[17] On, C. K.; Pandiyan, P. M.; Yaacob, S.; aj.: Mel-frequency cepstral coefficient analysis in speech recognition. In *2006 International Conference on Computing Informatics*, June 2006, ISSN 2166-5710, s. 1–5, doi:10.1109/ICOCI.2006.5276486.

[18] Onaolapo, J.; Idachaba, F.; Badejo, J.; aj.: A Simplified Overview of Text-To-Speech Synthesis. *Lecture Notes in Engineering and Computer Science*, ročník 1, 07 2014: s. 582–584.

[19] Patel, S.: *SVM (Support Vector Machine) — Theory*. [Online; navštíveno 04.12.2018].
URL <https://medium.com/machine-learning-101/chapter-2-svm-support-vector-machine-theory-f0812effc72>

[20] Pavlíková, P.: *Digitalizace signálu (obraz, zvuk)*. [Online; navštíveno 02.12.2018].
URL
<https://docplayer.cz/15872270-Digitalizace-signalu-obraz-zvuk.html>

[21] Russ Unger, Carolyn Chandler: *A Project Guide to UX Design: For User Experience Designers in the Field or in the Making*. New Riders, 2009, ISBN 978-0321607379.

[22] Sehili, M. A.; Istrate, D.; Dorizzi, B.; aj.: Daily sound recognition using a combination of GMM and SVM for home automation. In *2012 Proceedings of the 20th European Signal Processing Conference (EUSIPCO)*, Aug 2012, ISSN 2076-1465, s. 1673–1677.

[23] Siddhi, D.; Sarigam; Verghese, J. M.: Survey on Various Methods of Text to Speech Synthesis. 2017.

[24] Zhang, W.; he, L.; Deng, Y.; aj.: Time-Frequency Cepstral Features and Heteroscedastic Linear Discriminant Analysis for Language Recognition. *IEEE Transactions on Audio, Speech Language Processing*, ročník 19, 01 2011: s. 266–276.

Přílohy

Seznam příloh

Příloha A

Screener pro mobilní aplikaci

1. Používáte chytrý mobilní telefon?
   - Ano
   - Ne
2. Zvolte operační systém Vašeho telefonu.
   - Android
   - iOS
   - Windows Phone
   - Jiný
3. Používáte mobilní telefon i na jiné úkony, než jen volání a zasílání textových zpráv?
   - Ano
   - Ne
4. Praktikujete aktivně nějaké cvičení?
   - Ano, pravidelně
   - Ano, občas celoročně
   - Ano, pouze sezónně
   - Ne

Příloha B

Formulář pro ověření uživatelského rozhraní aplikace

1. Používáte nebo používal/a jste nějakou mobilní aplikaci pro podporu cvičení?

   ☐ Ano

   Jakou:_____

   ☐ Ne

2. Máte předchozí zkušenosti s testováním?

   ☐ Ano

   ☐ Ne

   Poznámka:

B.1 Sada úloh

1. Spusťte aplikaci Voice sport
2. Seznamte se s uživatelským rozhraním
3. Prohlédněte si cvičební skupiny a cvičební sady
4. Přejděte do cvičební sady s názvem "Demo cvičení"
5. Prohlédněte si cvičební úkony
6. Upravte poslední cvik (dril) tak, aby byl rychle modifikovatelný a před přechodem na další cvik se čekalo 5 sekund
7. Spusť cvičební sadu, během cvičení proveď 1x pozastavení cvičení a 1x přechod na následující cvik
8. Po dokončení sady vlož ke cvičení poznámku a ulož výsledek cvičení
9. Rozevři menu a přejdi na obrazovku statistik
10. V historii cvičebních sad si najdi výsledky o aktuálně dokončeném cvičení a otevři podrobné statistiky

11. Přejdi zpět na přehled cvičebních skupin
12. Vytvoř novou cvičební skupinu pod názvem "Moje cvičení"
13. Pod nově vytvořenou skupinou vytvoř cvičební sadu s názvem "Cvičení 1"
14. Přejdi do právě vytvořené cvičební sady a vytvoř tři drily (cviky):

    1. dril bude mít příkaz "2 úkroky vlevo"a bude nastaven jako rychle modifikovatelný cvik s čekáním 4 sekundy. Cvik bude zároveň uložen pro pozdější užití (uložen mezi koncepty).

    2. dril bude pauza 3 sekundy.

    3. dril bude mít příkaz "2 kroky vpravo s písknutím", bude nastaven jako rychle modifikovatelný a s čekáním na zvuk hvízdnutí.

15. Spusť cvičební sadu a ulož její výsledek
16. Po dokončení cvičební sady si zobraz přehled předuložených cviků (konceptů) → menu → sada uložených cviků
17. Najdi svůj vytvořený a uložený cvik a pomocí rychlé modifikace navyš počet kroků na 3
18. Přejdi do cvičební sady "Cvičení 1"a vlož nový cvik ze seznamu předuložených cviků
19. Přidej do cvičební sady tebou modifikovaný cvik (měl by mít název "3 kroky vlevo")
20. Zvyš počet opakování sady na 3x
21. Přejdi na přehled cvičebních sad a přejmenuj cvičební sadu "Cvičení 1"na "Cvičení 2"
22. Přejdi na přehled cvičebních skupin a smaž skupinu "Moje cvičení"
23. Zkontroluj, jestli se smazala i cvičební sada, která patřila do smazané skupiny
24. Ukončete aplikaci

3. Jak byste ohodnotil/a obtížnost vykonání úloh?

    ☐ Jednoduché
    ☐ Občas jsem si nebyl/a jistý/á, jak daný úkol provést
    ☐ Úlohy byly obtížné
    Poznámka:

4. Připadalo Vám uživatelské rozhraní intuitivní?

    ☐ Ano
    ☐ Informace či ovládací prvky jsem musel/a hledat
    ☐ Rozhraní není intuitivní
    Poznámka:

5. Připadalo Vám uživatelské rozhraní přehledné?

   ☐ Ano
   ☐ Ztrácel/a jsem se
   ☐ Rozhraní není přehledné
   Poznámka:

6. Jak se vám s aplikací pracovalo?

7. Přemýšlíte, že byste aplikaci využíval/a i nadále?

   ☐ Ano
   ☐ Ne
   Poznámka: