FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Služba pro analýzu telefonních hovorů
Student:	Bc. Jan Zípek
Vedoucí:	Ing. Miroslav Balík, Ph.D.
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2018/19

Pokyny pro vypracování

Navrhněte a implementujte službu, která bude převádět nahrávky telefonních hovorů na text a zobrazovat na jeho základě statistiky.
Služba bude umožňovat:

- zobrazení textu obou stran hovoru,
- přiřazení slov k pozici v nahrávce,
- vyhledávání v hovorech pomocí klíčových slov,
- vyhledávání dle kvalitativních dat hovoru,
- vytvoření grafu zobrazujícího nejpoužívanější slova a jejich vzájemné vztahy,
- filtrování všechny jmenované statistiky podle času,
- automatické a ruční tagování hovorů.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 1. února 2018

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

Služba pro převod nahrávek do textu

Bc. Jan Zípek

Katedra softwarového inženýrství
Vedoucí práce: Ing. Miroslav Balík, Ph.D.

9. ledna 2019

Poděkování

Rád bych poděkoval svým rodičům za morální podporu a také děkuji panu vedoucímu Ing. Miroslavovi Balíkovi, Ph.D. za odbornou pomoc při psaní a návrhu práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 9. ledna 2019 .....

Odkaz na tuto práci

Zípek, Jan. *Služba pro převod nahrávek do textu.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2019.

Abstrakt

Tato diplomová práce se zabývá návrhem a implementací služby pro automatický přepis telefonních nahrávek na základě integrace s telefonní ústřednou. Služba umožňuje mezi přepisy vyhledávat na základě metadat i textových dotazů a automatizovaně označuje nahrávky na základě obsahu jejich přepisu.

V práci je probrána analýza existujících řešení, návrh architektury služby, návrh uživatelského rozhraní a jejich realizace.

Klíčová slova Rozpoznávání řeči, architektura služeb, HMM, uživatelské rozhraní

Abstract

This diploma thesis consists of design and implementation of service for automatic transcription of telephone recordings based on integration with call center. Service allows to search between transcriptions based on metadata and text based queries and automatically marks recordings based on their transcript contents.

The didploma thesis deals with analysis of existing solutions, design of service architecture, design of user interface and their realisation.

Keywords Speech recognition, service architecture, HMM, user interface

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Telefonní ústředny jsou v současnosti běžnou součástí velkých i malých firem, které je často používají k poskytování podpory svým zákazníkům.

Ve větší ústředně mohou pracovat až desítky agentů a každý přijímá stovky hovorů denně. Řídit takové množství agentů a hovorů je značně náročný úkol a často až nemožný. Pro kontrolu obsahu hovorů, jestli se agent chová podle stanov, se většinou dají použít pouze pořízené nahrávky hovorů a ruční přepis.

Hodnotit každou nahrávku zvlášť je velmi náročné, taková kontrola lze provádět pouze namátkově. Pokud bude k dispozici textový přepis nahrávek, bude možné je na základě obsahu označovat a nahrávky ručně kontrolovat pouze na základě označení.

Navíc ve chvíli, kdy bude k dispozici textový přepis, bude možné v nahrávkách vyhledávat na základě daných témat a takto je dále kategorizovat a procházet.

Tento problém řeší automatizovaný přepis nahrávek na text, to je dnes z pohledu existujících řešení problém, který lze vyřešit. Ovšem žádné z existujících řešení neumožňuje přímé napojení na telefonní ústřednu, tedy automatické hromadné přepisování všech nahrávek procházejících ústřednou, automatizované vyhodnocení závadnosti obsahu, zobrazení přepisů v závislosti na metadatech hovoru, filtrování na základě obsahu a další funkce, které jsou pro použití pro kontrolu telefonní ústředny důležité.

Tato diplomová práce dokumentuje návrh a implementaci služby a jejího uživatelského rozhraní, která bude umožňovat na základě integrace se softwarem telefonní ústředny přepisování telefonních nahrávek procházejících ústřednou a tyto přepisy dále zpracovávat a zobrazovat v ucelené podobě uživatelům.

Při návrhu bude kladen důraz na možnou rozšířitelnost a spolehlivost služby a efektivnost uživatelského rozhraní, které by mělo být jednoduše použitelné i pro méně zkušené uživatele.

Kapitola 1

Analýza

Aby bylo možné správně navrhnout architekturu celé služby, je nejdříve potřeba podrobně analyzovat zadání, rozvést a kategorizovat její specifikaci a následně tuto specifikaci porovnat s existujícími službami podobného nebo stejného typu, čímž se ujistím, že je skutečně potřeba službu nově navrhnout a implementovat.

Na závěr analýzy se ještě zaměřím na analýzu telefonních nahrávek, se kterými bude systém pracovat. Jejich vlastnosti budou důležité při návrhu postupu přepisu.

1.1 Specifikace

V této sekci rozvinu zadání a kategorizuji jej do přehlednějšího celku, který mi pak umožní lépe a přehledněji navrhnout architekturu celé služby a hlavně tím určím, co je přesně od výsledné služby očekáváno.

Ze zadání vyplývá, že služba bude přijímat nahrávky telefonních hovorů a přepisovat je do textové podoby. Takový přepis je pak potřeba zobrazit uživateli v použitelné formě, takže bude potřeba počítat s funkčním vyhledáváním a filtrováním.

Momentálně se bude služba soustředit na český trh, proto je potřeba aby podporovala český jazyk.

Lze předpokládat, že systém bude používám více než jedním uživatelem, takže bude podporovat uživatelské účty, které se budou dělit na dvě uživatelské role, správce a uživatele. Správce bude moci upravovat a vytvářet uživatelské účty, zatímco uživatel bude moci pouze procházet nahrávky a statistiky.

Pro případ, že bude služba v budoucnu rozšířena o další jazyky, je potřeba, aby bylo uživatelské rozhraní navrženo tak, aby podporovalo integraci překladů.

Výsledky proběhlých přepisů musí být neustále k dispozici, nehledě na to, jestli právě probíhá přepis dalších nahrávek.

1.1.1 Funkční požadavky

Ve funkčních požadavcích specifikuji požadavky, které přímo ovlivňují chování služby a její interakci s uživatelem. Funkční i nefunkční požadavky specifikuji přímo pro celou službu, protože v této fázi ještě není jisté, do jakých modulů a částí bude rozdělena.

F1 Podpora přepisu českých nahrávek Služba bude umožňovat přepis telefonních nahrávek v češtině.

F2 Automatizovaný přepis nahrávek Služba bude umožňovat automatický přepis většího počtu nahrávek bez nutnosti manuálního zásahu.

F3 Integrace s telefonní ústřednou Služba bude podporovat napojení na telefonní ústřednu, na základě kterého bude zajišťovat automatický přepis nahrávek hovorů na ústředně.

F4 Výpis nahrávek Služba bude uživateli umožňovat výpis přepsaných nahrávek. Výpis by měl obsahovat nejméně volajícího, volaného, datum a čas hovoru a textový přepis.

F5 Výpis textového přepisu Služba bude uživateli umožňovat ke každému hovoru zobrazit textový přepis.

F6 Pozice slov v nahrávce Jednotlivá slova v textovém přepisu nahrávky budou mít časovou známku, určující kdy se přesně v nahrávce vyskytla.

F7 Přehrávání nahrávek Služba bude uživateli umožňovat každou nahrávku přehrát.

F8 Označování nahrávek Služba bude umožňovat ručně označovat nahrávky.

F9 Vyhledávání pomocí klíčových slov Služba bude umožňovat vyhledávání v nahrávkách pomocí klíčových slov.

F10 Vyhledávání pomocí filtrů Služba bude umožňovat vyhledávání v nahrávek na základě času, kdy byla nahrávka pořízena, čísla volajícího nebo čísla volaného.

F11 Uživatelské účty Uživatelé služby se budou autentifikovat pomocí uživatelských účtů.

F12 Správa uživatelských účtu Služba bude obsahovat roli správce, která bude uživateli umožňovat vytvářet nové a upravovat existující uživatelské účty.

F13 Zobrazení grafu souvisejících slov Služba bude umožňovat zobrazovat graf slov, který bude znázorňovat, v jaké vzdálenosti a jak často jsou slova používaná ve vztahu vůči vybranému slovu.

F14 Automatizované označování narávek Systém by měl umět automatizovaně označovat nahrávky na základě určitých kvalitativních parametrů.

1.1.2 Nefunkční požadavky

Nefunkční požadavky doplňují funkční požadavky, specifikuji v nich další vlastnosti, které přímo neovlivňují funkcionalitu aplikace, ale specifikují dodatečné vlastnosti služby.

NF1 Minimální závislost na konkrétním systému telefonní ústředny Architektura služby musí být koncipována tak, aby nebyla závislá na jednom konkrétním systému ústředny.

NF2 Možnost horizontální i vertikální škálovatelnosti Služba musí být koncipována tak, aby bylo ji bylo možné na základě množství obsluhovaných hovorů řádně škálovat.

NF3 Multijazyčnost rozhraní systému Systém by měl podporovat možnost změny jazyka rozhraní.

NF4 Dostupnost Množství přepisovaných nahrávek by nemělo ovlivňovat výkon a dostupnost uživatelského rozhraní.

NF5 Podpora software telefonní ústředny Asterisk Služba by měla podporovat integraci s opensource telefonní ústřednou Asterisk.

1.1.3 Případy užití

Případy užití pomocí ručně definovaných scénářů definují, jak bude potenciální uživatel službu používat. Umožňují tak správně utřídit, jak má vlastně služba fungovat z pohledu uživatele. Na jejich základě lze také později vytvářet testovací scénáře, pomocí kterých se bude zjišťovat, jestli služba podporuje veškerou požadovanou funkcionalitu.

Proto by měli pokrývat všechny funkční požadavky, což znázorňuje tabulka pokrytí 1.1.

1.1.3.1 UC1 Zobrazení přepisu a přehrání nahrávky

1. Uživatel se přihlásí
2. Uživatel zobrazí seznam nahrávek
3. Uživatel otevře detail nahrávky s přepisem

Výstup: Detail nahrávky s možností jejího přehrání a textovým přepisem obsahu.

1.1.3.2 UC2 Vyhledání nahrávek obsahující slovo nebo sérii slov

1. Uživatel se přihlásí
2. Uživatel zobrazí seznam nahrávek
3. Uživatel vyfiltruje seznam nahrávek pomocí slov

Výstup: Seznam nahrávek, které obsahují dané slov nebo slova s možností jejich přehrání.

1.1.3.3 UC3 Zobrazení návazných slov určitého slova

1. Uživatel se přihlásí
2. Uživatel vyhledá slovo, které chce zobrazit

Výstup: Graf slov souvisejících s vybraným slovem, znázorňující jaké a jak často jsou slova používaná v návaznosti na vybrané slovo.

1.1.3.4 UC4 Vytvoření uživatelského účtu

1. Uživatel se přihlásí
2. Uživatel si zobrazí seznam účtů
3. Uživatel otevře formulář nového účtu
4. Uživatel uloží formulář nového účtu

Výstup: Nový uživatelský účet je vytvořen a je možné se pomocí jeho přihlašovacích údajů přihlásit.

1.1.3.5 UC5 Změna hesla přihlášeného uživatele

1. Uživatel se přihlásí
2. Uživatel si zobrazí informace o účtu
3. Uživatel zadá nové heslo
4. Uživatel odešle nové heslo

Výstup: Přihlášený uživatel má změněné heslo.

Tabulka 1.1: Pokrytí funkčních požadavků případy užití

	UC1	UC2	UC3	UC4	UC5
F1	X	X	X		
F2	X	X	X		
F3	X	X	X		
F4	X	X			
F5	X	X			
F6	X	X			
F7	X	X			
F8	X	X			
F9		X			
F10		X			
F11	X	X	X	X	X
F12				X	
F13			X		
F14	X	X			

1.2 Analýza existujících řešení

Před počátkem vývoje služby byla provedena základní analýza trhu, jejíž cílem bylo zjistit jestli je vývoj služby potřeba, nebo jestli je možné použít už existující řešení. Cílem této analýzy byly jak služby, které přímo umožňují hromadný převod nahrávek na text a jeho zobrazení, tak i služby které se orientují pouze na převod nahrávek na text.

Analýza byla provedena na základě veřejně dostupných materiálů, jako jsou webové prezentace a dokumentace.

1.2.1 SpeechTech Analytics

Jedná se o službu se zaměřením na přepis nahrávek do textu a jejich následnou analýzu a tvorbu statistik. Toto řešení je komerční a není veřejně dostupné. Pokus kontaktovat vývojáře za účelem porovnání funkcionality byl neúspěšný, takže se při porovnání lze řídit pouze podle webové prezentace.

Služba tedy údajně, podle dostupné webové prezentace nabízí přepis nahrávek do textu, následně pak vyhledávání pomocí klíčových slov a základní kategorizaci hovorů na základě obsahu[4]. Nelze přesně určit jak a v jaké kvalitě jsou tyto vlastnosti dostupné, z přiloženého snímku aplikace nelze vyčíst žádné vlastnosti související s přepisem, zřejmě se jedná o pohled na normální statistiky ústředny, které s automatizovaným přepisem nahrávek nesouvisí.

Tabulka 1.2: Tabulka pokrytí funkčnosti existujícími řešeními

	SpeechTech	Google	Azure
Čeština	X	X	
Hromadný převod nahrávek	X		X
Ucelené zobrazení výsledků převodu	X		
Dostupnost		X	X

1.2.2 Google Cloud Speech-to-Text

Google Cloud Speech-to-Text je služba nabízená společností Google pro přepis obecných nahrávek řeči do textu[5]. Přepis probíhá na vyžádání na vzdálených serverech provozovaných společností, která službu poskytuje. Služba poskytuje pouze a jen přepis do textu, neposkytuje žádné dodatečné statistiky, nebo informace.

Při základním testování pomocí dostupného dema bylo zjištěno, že u telefonních nahrávek, na které cílí má služba, dosahuje použitelných výsledků, průměrně 20 % Word Error Rate (WER). Služba na vyžádání poskytuje k jednotlivým slovům i přesné pozice v nahrávce, které jsou jedním z důležitých požadavků. Včetně češtiny podporuje celkem 120 jazyků[6].

1.2.3 Azure Speech-to-Text

Služba založená na stejném principu jako Google Cloud Speech-to-Text. Oproti této službě navíc umožňuje dodat vlastní data pro doučení nebo i naučení vlastního modelu pro přepis. V současné době nepodporuje češtinu a nepředává časové známky ke slovům, pouze textový přepis [7, 8].

1.2.4 Vyhodnocení existujících řešení

Na základě provedené analýzy se podařilo zjistit, že s určitou pravděpodobností neexistuje služba, která by splňovala všechny naše požadavky, viz tabulka 1.2.

U služeb Google a Azure je nutné navíc zmínit, že přepis je zpoplatněn na základě délky nahrávky, což v některých případech může náklady zvýšit neúnosně, protože, jak bylo určeno v cílech služby, je možné, že se bude jedna o stovky nahrávek o délce v řádech jednotek minut denně.

I přesto by naše služba tyto řešení mohla interně používat pro přepis nahrávek, v případě, že bude objem nahrávek přijatelný a nebude problém, že se nahrávky posílají na externí servery.

Obecně ale žádně ucelené řešení neexistuje, proto jsem vyhodnotil, že návrh a implementace služby dává smysl.

1.3 Základní vlastnosti analyzovaných nahrávek

Tato služba cílí primárně na nahrávky telefonních hovorů, které mají své specifické vlastnosti. Z pohledu služby je důležitá kvalita pořízené nahrávky, která je v rámci České Republiky nižší, což je dáno malou vzorkovací frekvencí a šumem. To má negativní vliv na úspěšnost přepisu, většina modelů i trénovacích dat pracuje s vzorkovací frekvencí 16 kHz, kdežto nahrávky bývají v 8 kHz. Je možné natrénovat model na obě kvality, ale většinou je příhodnější nahrávky upravit, aby odpovídaly potřebné frekvenci. To ale může mít negativní vliv na kvalitu přepisu.

Dále kvalitu hovoru a následného přepisu negativně ovlivňuje množství zařízení, které mohou záznam řeči pořizovat, tedy telefonů. Drobné odlišnosti každého zařízení můžou vést k velkým odlišnostem ve zvukovém záznamu.

Proto je potřeba před samotným přepisem provést základní předběžné zpracování a pokud možno nahrávky alespoň trochu normalizovat. Nízká kvalita způsobuje velké množství šumu, který je potřeba odstranit. Dále je potřeba normalizovat hlasitost, protože i v rámci jednoho účastníka může dojít k náhodným výkyvům hlasitosti, například pokud se v průběhu hovoru trochu vzdálí od telefonu, nebo přepne na reproduktor.

Teoreticky je možné tyto úpravy ponechat na službě na přepis, která se bude využívat a většina možných řešení také provádí alespoň základní předzpracování, ale bylo by lepší alespoň otestovat, jestli by využití vlastního předzpracování nezvýšilo kvalitu přepisu.

Budeme předpokládat, že nahrávky, které bude tato služba přijímat budou mít účastníky pevně oddělené, tedy každý z nich bude umístěn v jednom kanálu nahrávky a tím odpadne nutnost účastníky odlišovat algoritmickými metodami, které by nemusely být přesné[9]. Například ústředna Asterisk takový postup podporuje.

Kapitola 2

Přepis hlasu na text

V této sekci se obecně zaměřím na to, jaká je běžná architektura systémů pro přepis nahrané řeči na text[10]. Tato architektura je totiž používaná naprostou většinou knihoven a pro správné použití těchto řešení je potřeba alespoň základní pochopení, jak tato architektura funguje.

2.1 Základní architektura přepisu hlasu na text

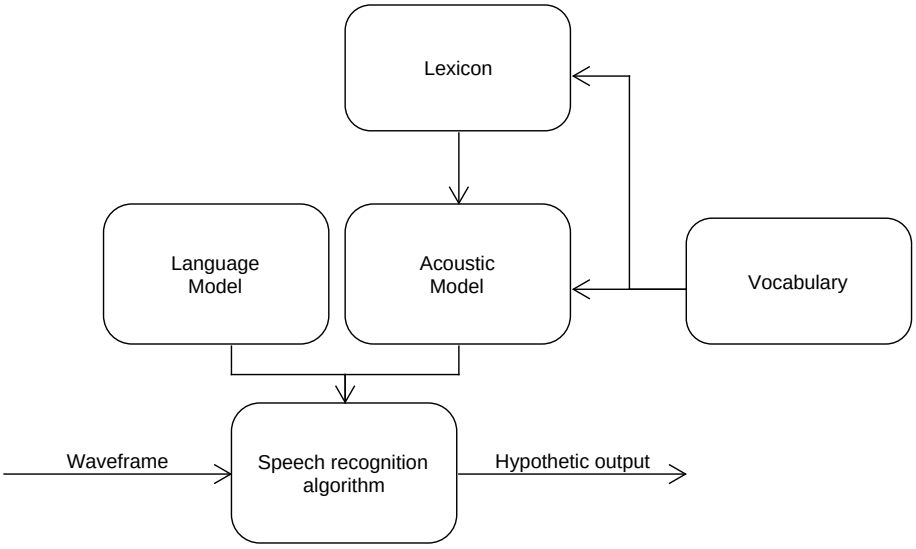
Řešení pro automatický přepis hlasu na text v dnešní době existuje větší množství, ale většina má stejnou, nebo podobnou obecnou architekturu řešení[10].

Taková obecná architektura systému pro převod řeči je znázorněná na diagramu 2.1. Obsahuje 4 důležité části, Vocabulary je jednoduše slovník slov, se kterými systém umí pracovat a Lexicon obsahuje konverzi slovníku na takzvané fonémy, jejichž roli budu probírat dále. Tyto dvě součásti jsou pak propojené s Acoustic Model, tedy akustický model, který převádí zvuk na možná slova, které pak v kombinaci s Language Modelem, tedy jazykovým modelem, tvoří hypotetický přepis zvuku.

2.1.1 Akustický model

Akustický model má tedy na vstupu zvuk a snaží se jej převést na slova. Jeho výstupem nemusí být jenom jedno slovo, ale většinou to je více slov spojených pravděpodobností přítomnosti, proto pak přichází na řadu jazykový model, který vybírá nejpravděpodobnější posloupnost těchto odhadnutých slov.

Zvuk je nejdříve potřeba rozdělit na takzvané utterance, což jsou úseky zvuku oddělené mezerami ticha. V těchto úsecích pak akustický model hledá takzvané fonémy.

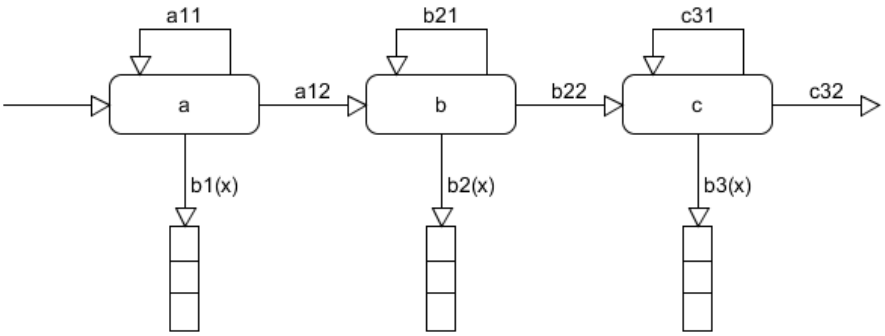


Obrázek 2.1: Obecná architektura systému pro přepis řeči. Převzato z [1]

Fonémy jsou z pohledu jazykových věd základní zvuková jednotka slova. Některé akustické modely pracují i přímo se slovy, ale takový postup se vyplatí pouze pokud systém cílí na omezené množství slov, protože pro naučení takového modelu je potřeba dodat hned několik nahrávek pro každé slovo, které chceme rozpoznávat. Každé slovo má pak vlastní parametry a musí se zvlášť při přepisu vyhodnocovat, což zvyšuje požadavky na prostor a výpočetní výkon, proto je tento postup pro textový přepis, který by měl podporovat tisíce slov, nevýhodný[11].

Orientací na fonémy se model i postup zjednoduší. Model se na základě trénovacích dat a lexikonu naučí rozpoznávat fonémy, kterých je oproti slovům řádově méně a ty pak převádět pomocí lexikonu na slova. Ve chvíli, kdy akustický model umí rozpoznávat fonémy, teoreticky je schopný rozpoznat jakékoli slovo, které se z naučených fonémů skládá, tedy i slova, která nebyla v trénovacích datech.

Fonémy mohou mít různou výslovnost v závislosti na kontextu. Proto pokročilejší systémy pracují s takzvanými triphone, kdy se v potaz nebere pouze daný foném, ale i jeho pravý a levý soused[12]. Běžný postup je, že se model nejdříve naučí rozpoznávat jednotlivé fonémy a poté se pomocí těchto poznatků učí rozpoznávat posloupnosti fonémů, například hlavní foném a jeho levého a pravého souseda[13]. Takové posloupnosti se pak říká triphone a model je pak trifóní. Některé systémy pracují i s quinphone, tedy vzájemnou závislostí 4 fonémů.



Obrázek 2.2: Ukázka HMM o třech stavech

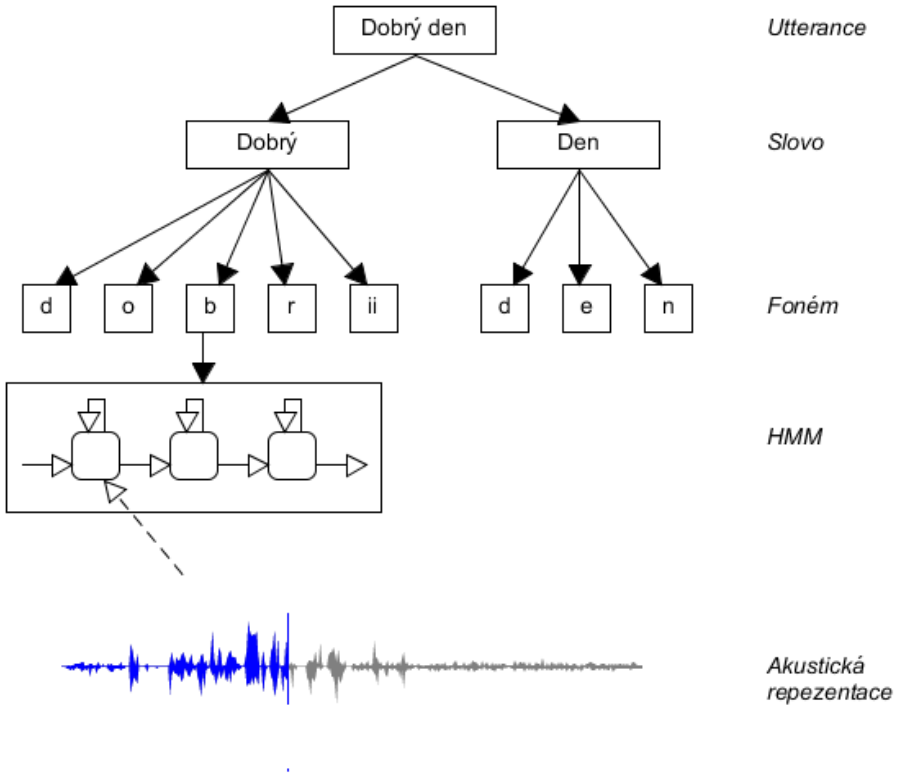
Pro rozpoznání fonému se nejdříve utterance rozdělí na miniaturní úseky, většinou o délce 10 milisekund. Tyto úseky jsou následně převedeny na vektorovou reprezentaci.

Pro vektorovou reprezentaci se většinou používá Mel-frequency cepstrum (MFCC) sekvence, která je pro toto použití momentálně nejvhodnější, protože reprezentuje ideální poměr nízkého počtu výstupních parametrů a nízké ztráty informace[14]. Velikost vektorové reprezentace těchto úseků je důležitá a má přímý vliv na rychlost přepisu a objem potřebných trénovacích dat, v některých případech může každý nový parametr zvýšit výpočetní náročnost až exponenciálně. Po převodu na MFCC pak takový úsek reprezentuje vektor o přibližně 30 číslech.

V dalším kroku akustický model na základě těchto vektorů odhadne posloupnost fonémů. K tomu se používá statistický model HMM[15]. HMM je zvláštním případem stochastického konečného automatu, kde je generována diskrétní sekvence přechodů z jednotlivých stavů za účasti přechodové pravděpodobnosti[3]. Ukázku běžně používaného HMM můžete vidět na obrázku 2.2.

Pravděpodobnost výstupu je většinou spjata s normálním (Gaussovým) rozložením a úlohou je odhadnout posloupnost stavů, které vedli k výstupu. Výstup je v tomto případě vektorová reprezentace úseků a na základě posloupnosti stavů, které k výstupu došli se odhaduje foném. Úlohou při učení modelu je vypočítat parametry normálního rozložení, vedoucí k nejvíce správnému výstupu[1].

Na diagramu 2.3 je možné vidět, jak vypadají jednotlivé vrstvy zpracování vstupu u akustického modelu.



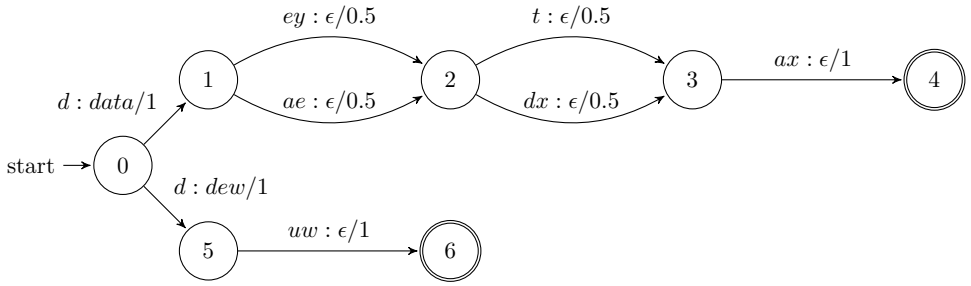
Obrázek 2.3: Základní struktura funkčnosti akustického modelu

Jakmile je k dispozici pravděpodobná posloupnost fonémů, odhadne systém pomocí lexikonu slovo. Součástí lexikonu je mimo jiné i vážený konečný automat, který bere v potaz možné odlišné výslovnosti slov, příklad je možné vidět na diagramu 2.4. Jedná se tedy o vážený končený automat, u přechodů je značen vstupní foném, výstupní slovo a váha přechodu. Vstupem tohoto automatu je tedy posloupnost fonémů a výstupem pravděpodobnost výsledného slova.

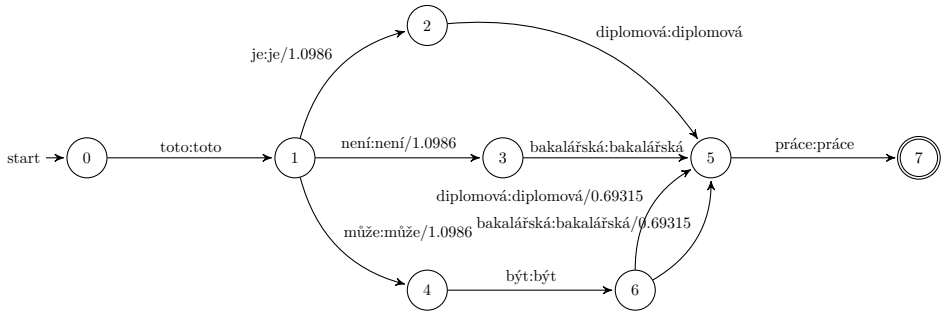
Ve chvíli, kdy jsou k dispozici pravděpodobnosti výskytu slov v daném úseku, data jsou předána zpátky a pomocí jazykového modelu dochází k jejich složení v ucelený textový přepis.

2.1.2 Jazykový model

Jazykový model je běžně reprezentován jako vážený končený automat, což je konečný automat, kde jsou přechody kromě vstupního a výstupního symbolu označeny i váhou[1].



Obrázek 2.4: Ukázka váženého konečného automatu pro slova date a dew, převzato z [2]



Obrázek 2.5: Ukázka jazykového modelu, převzato z [3]

Model se většinou učí na korpusu, tedy přepisu trénovacích dat akustického modelu, ale je možné jej natrénovat i zvlášť, na základě jakéhokoli textu v cílovém jazyce.

Základní příklad takového modelu můžeme vidět na diagramu 2.5. Jedná se o vážený konečný automat, jednotlivé přechody jsou označeny vstupním a výstupním slovem (v tomto případě jsou stejné) a váhou přechodu.

2.1.3 Sjednocení modelů

Skutečný systém sice v rámci architektury respektuje předchozí kroky, ale většinou jsou všechny modely: jazykový, akustický i lexikon úzce propojeny v celém průběhu přepisu a učení.

Je možné si všimnout, že většina interních reprezentací v modelech jsou automaty. Proto se ve většině systémů tyto automaty slučují do jednoho velkého váženého automatu[2], takové sloučení má pozitivní vliv na výkonnost algoritmu a může ve svůj prospěch využít některé vlastnosti determinizace a minimalizace.

Například v systému Kaldi se k přepisu používá HCLG[16], vybudovaný na konci učení, kdy:

H Je reprezentace HMM.

C Reprezentuje kontextuální závislost, na vstupu má N kontextově závislých fonémů a na výstupu fonémy.

L Je lexikon, který na převádí slova na fonémy.

G Je akceptor, který se stará o kódování jazykového modelu.

2.2 Vyhodnocení kvality přepisu

Aby bylo možné vyhodnotit, jestli model přepisu splňuje cílovou funkci, je potřeba mít možnost charakterizovat jeho vlastnosti.

Systém přepisu se většinou testuje na oddělené databázi testovacích nahrávek a jejich přesných přepisů.

Nejběžnější ukazatel, který se používá pro vyhodnocení přesnosti modelu je WER. Ten se počítá se na základě porovnání přepisu vygenerovaného modelem přepisu a ručním, správným, přepisem.

Když definujeme `I` jako počet slov, které bylo potřeba vložit aby z automatického přepisu vznikl správný přepis, `D` jako počet smazaných slov, `S` počet nahrazených slov a `N` celkový počet slov přepisu, pak WER definujeme jako:

$$WER = \frac{I + D + S}{N} \tag{2.1}$$

Výsledek je pak poměr chybovosti přepisu, ten se pak vyjadřuje v procentech a čím menšího WER systém dosahuje, tím úspěšnější model je.

Dalším ukazatelem je přesnost, anglicky značená `Accuracy`, je skoro stejný jako WER, pouze nebere v úvahu počet vložených slov a jeho výstup je otočený, tedy větší procento je lepší:

$$Accuracy = \frac{N - D - S}{N} \tag{2.2}$$

Důležitým parametrem modelu je taky jeho rychlost, ta se většinou značí jako poměr doby přepisu vůči délce nahrávky, značené `RT`. Například pokud přepis nahrávky o délce (`RT`) 1 hodiny trvá 2 hodiny, pak se rychlost přepisu vyjádří jako 2RT.

Kapitola 3

Návrh služby

Na základě specifikace služby a dalších skutečností zjištěních během analýzy, jsem rozhodl, že je potřeba službu a její celkový návrh rozdělí na tři nezávislé moduly. Takový modulární návrh umožňuje větší pružnost řešení a usnadňuje možnou budoucí expanzi služby.

3.1 Moduly služby

Rozdělit modul přepisovače a backendu jsem se rozhodl na základě požadavku na dostupnost přepisů. Přepis nahrávek může být náročný a v případě, že by byl součástí stejného modulu, který dodává data uživateli, mohl by negativně ovlivnit dostupnost.

Frontend je pak oddělen od backendu, protože plánuji, že se bude jednat o tlustého klienta, nejspíše implementovaného v jiném jazyce než backend.

Přepisovač Tento modul se bude zabývat pouze přepisem nahrávky na text.

Backend Tento modul se bude starat o přidělování práce modulu přepisovače a zároveň bude poskytovat klientům možnost textové přepisy procházet, vyhledávat v nich a zobrazovat statistiky.

Frontend Tento modul bude konečnému uživateli umožňovat procházet textové přepisy, vyhledávat a zobrazovat statistiky.

3.2 Návrh obecné architektury

Když máme rozhodnuté jaké moduly bude služba obsahovat a jaká bude jejich zodpovědnost, je nutné navrhnout správnou strukturu celé služby, vyjasnit si role jednotlivých modulů a alespoň obecně si připravit, co bude předmětem jejich komunikace.

Vzhledem k požadavkům na službu jsem rozhodl, že služba by měla být navržena tak, aby mohl každý modul běžet na oddělených serverech. Nejdůležitější bude rozdělení přepisovače a backendu, protože samotný úkon přepisu bude značně náročný na zdroje a jeho spojení s backendem, který je zodpovědný za komunikaci s klienty by mohlo způsobit jeho zpomalení či nedostupnost, což je nežádoucí.

Oddělením přepisovače a backendu je navíc umožněno mít v provozu více instancí přepisovače a rovnoměrně mezi ně rozdělovat zátěž a značně tím zvýšit rychlost přepisu.

Výsledná obecná architekturu je graficky znázorněná na obrázku 3.1. Kromě již zmíněných vlastností je v ní navíc znázorněná možnost, že i samotné tři hlavní moduly budou s nejvyšší pravděpodobností rozdělené na menší moduly, které budou mít vlastní zodpovědnost.

3.3 Návrh komunikace mezi moduly

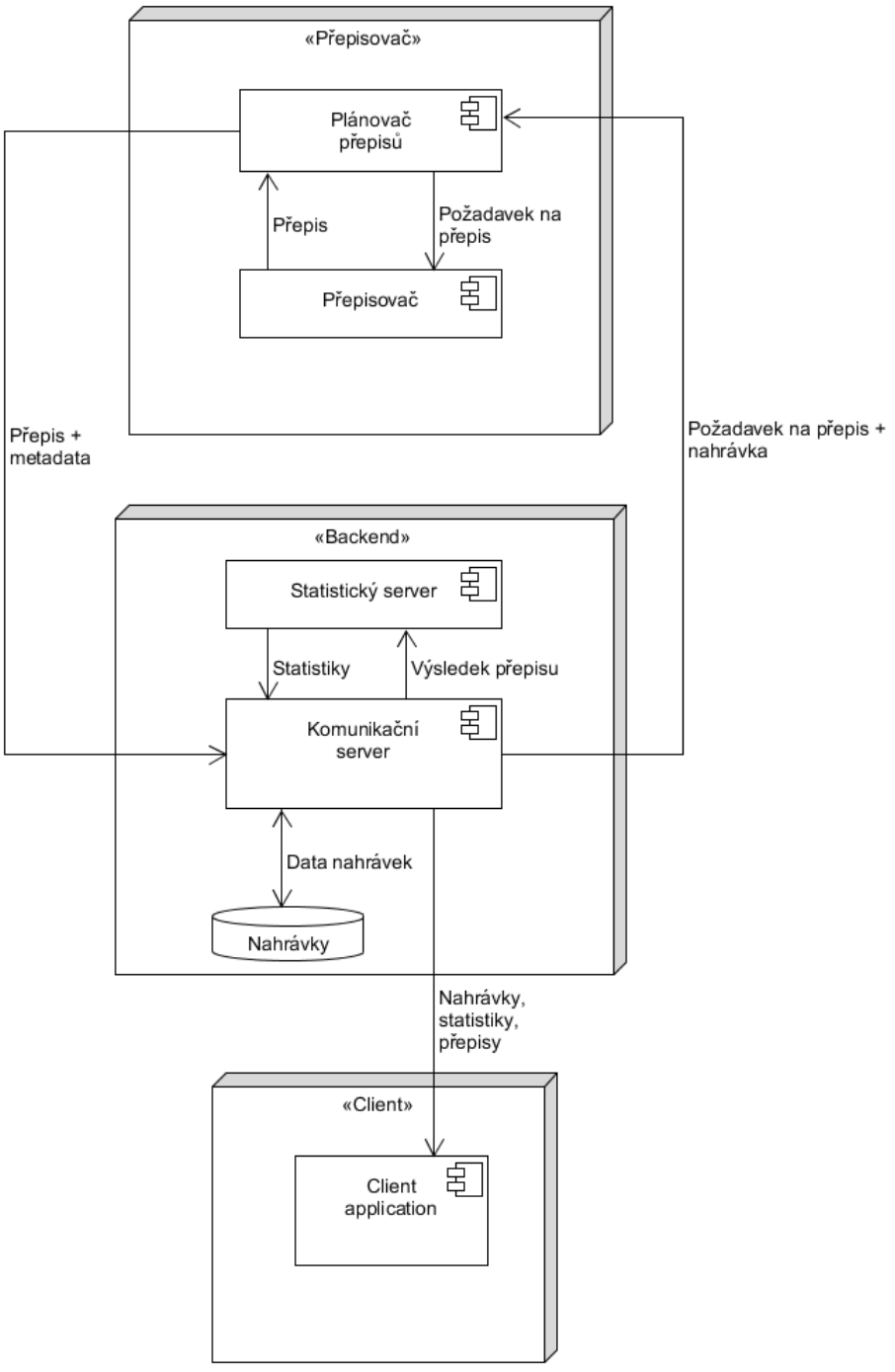
Jednotlivé moduly budou oddělené a pro komunikaci budou používat síťové rozhraní. Protokol a struktura komunikace by měla být stejná mezi všemi moduly, rozdílné protokoly by nepřinesly žádnou výhodu a naopak by se tím zvýšila složitost vývoje.

Jak lze vidět na obecném návrhu architektury, komunikační kanály budou celkem dva, jeden mezi přepisovačem a backendem a druhý mezi backendem a klientem. První se bude používat pro přenos nahrávek a jejich přepisů a druhý pro přenos dat získaných z přepisů.

Z možných existujících standardů pro komunikace jsou pro použití ve službě nejvhodnější následující možnosti:

Representational State Transfer (REST) Standard postavený na protokolu HTTP[17]. V rámci veřejných síťových služeb se jedná o nejpoužívanější standard. Výhodou je jeho jednoduchost, podpora v naprosté většině programovacích jazyků a jednoduchost ladění. Nevýhodou pak je, že v určitých částech je jeho definice vágní a umožňuje více možných řešení, čímž se poněkud ničí funkce standardu. Běžně se při v kombinací s tímto standardem používá pro obsah zpráv JavaScript Object Notation (JSON).

Simple Object Access Protocol (SOAP) Další standard postavený na protokolu HTTP. Jedná se o pevně definovaný standard, ovšem značně složitý pro ladění a vzhledem k použití zastaralého Extensible Markup Language (XML), který je s ohledem na obsah zpráv neefektivní[18]. V našem případě je jeho striktní definice a složitost spíše na škodu.



Obrázek 3.1: Obecná architektura služby

Vlastní řešení Je také možné vytvořit nový vlastní protokol a kompletní komunikační standard. Takové řešení se vyplatí pouze v případě, že od komunikace lze očekávat nějaké velice nestandardní chování, v tomto případě se ovšem bude jednat pouze o velice standardní chování, takže určitě bude stačit najít vyhovující existující řešení.

Porovnáme-li si obsah probíraných protokolů při jednoduchém požadavku, můžeme vidět rozdíly ve složitosti.

Zatímco požadavek a odpověď SOAP obsahuje množství duplicitních informací:

```
GET / HTTP/1.0

<?xml version = "1.0"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV = "http://www.w3.org/2001/12/so..."
  SOAP-ENV:encodingStyle = "http://www.w3.org/2001/...">
  <SOAP-ENV:Body xmlns:m = "http://www.xyz.org/state">
    <m:GetState></m:GetState>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

```
HTTP/1.0 200 OK

<?xml version = "1.0"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV = "http://www.w3.org/2001/12/so..."
  SOAP-ENV:encodingStyle = "http://www.w3.org/2001/...">
  <SOAP-ENV:Body xmlns:m = "http://www.xyz.org/state">
    <m:GetStateResponse>
      <m:QueueSize>15</m:QueueSize>
      <m:QueueDuration>1561</m:QueueDuration>
    </m:GetStateResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

stejný požadavek za použití REST a JSON vypadá následovně:

```
GET /state HTTP/1.0
```

```
HTTP/1.0 200 OK
```

```
{
  "queueSize": 15,
  "queueDuration": 1561
}
```

Vzhledem k tomu, že se bude jednat o komunikaci mezi více moduly, nejspíše implementovanými i ve více programovacích jazycích, díky lepší osobní zkušenosti a také vzhledem k tomu, že služba obsahuje frontend jsem zvolil použití REST s obsahem ve formátu JSON. REST má vzhledem k jednoduché implementaci největší podporu a lze s ním jednoduše pracovat.

Nevýhodu vágní definice obsahu zpráv budu kompenzovat přesnou dokumentací protokolu.

3.4 Návrh přepisovače

Tento modul bude vykonávat ze všech tří potencionálně nejnáročnější činnost, samotný přepis nahrávek na text. Pokud se bude jednat o přepis pomocí vlastního modelu, bude potřeba počítat s tím, že se bude jednat o modul značně náročný na prostředky. Je proto nutné, aby byl připraven na případné škálování do šířky, tedy aby bylo možné ho provozovat na jiném fyzickém či virtuálním stroji, než ostatní moduly, jejichž dostupnost by mohla úloha přepisu vytěžováním prostředků narušovat. Navíc by mělo být možné mít těchto modulů více a rozkládat mezi ně zátěž.

Přepisovač jako takový bude rozdělen do dvou podmodulů. Jeden se bude starat o komunikaci s ostatními moduly a frontu požadavků a druhý o samotný přepis.

Vzhledem k tomu, že přístupů pro přepis je více a každý má své využití, bude lepší, když bude možnost měnit implementaci podmodulu přepisu na základě konfigurace modulu. Tím se zajistí, že modul bude podporovat současné možnosti pro přepis a zároveň bude připraven do budoucna adaptovat novější technologie.

3.4.1 Příprava nahrávek k přepisu

Každou nahrávku bude potřeba před každým přepisem upravit tak, aby byla kvalita výstupu co nejvyšší. O to se bude starat právě přepisovač a to těsně před tím, než předá nahrávku k samotnému přepisu. Jako první bude potřeba normalizovat formát nahrávky, protože jak Kaldi, tak i Google Cloud Speech-to-Text vyžaduje specifické parametry i formát.

Následně je nahrávku potřeba rozdělit na volajícího a volaného. Jak bylo určeno v analýze, nahrávky se kterými bude přepisovač pracovat, budou mít volajícího a volaného předem rozdělené tak, že v jednom kanálu stereo zvuku bude volající a v druhém volaný, takže rozdělení bude bezchybné. V případě, že by takto nahrávka rozdělená nebyla, musel by se přepisovač uchylovat k rozlišování jednotlivých účastníků pomocí fuzzy logiky a takový způsob by nemusel být spolehlivý, obzvlášť v případě, kdy se oba účastníci v nahrávce překrývají.

Dále je potřeba odstranit šum, tak aby byl hlas maximálně čistý a rozpoznatelný. K odstranění šumu bude použit low-pass filtr, který je pro toto použití relativně vhodný a jednoduchý na implementaci[19].

Po těchto úpravách bude nahrávka předána k přepisu.

3.4.2 Řešení přepisu hlasu na text

Pro přepis hlasových nahrávek na text v současné době existuje vícero existujících řešení. Jedná se buď o komerční řešení, kdy je k dispozici pouze určitá forma komunikačního kanálu, pomocí kterého lze získávat výsledky přepisu, nebo kompletně otevřená řešení, které umožňují natrénovat vlastní model pro svoji cílovou doménu a ten poté použít pro přepis.

3.4.2.1 Google Cloud Speech-to-Text

Jak již bylo vymezeno v analýze existujících řešení, jedná se o komerční službu, která pomocí síťového API umožňuje přepis nahrávek na text a to včetně češtiny. V současné době dosahuje tato služba celkem slušných výsledků, ovšem její nevýhodou je nutnost platit za každý jednotlivý přepis. Navíc je potřeba posílat přepisované nahrávky na cizí server, což by se v některých případech dalo považovat za bezpečnostní riziko.

U této služby je možné, že se bude kvalita přepisu v budoucnosti zlepšovat bez nutnosti dodatečných investicí ze strany uživatele této služby.

3.4.2.2 Azure Speech-to-text

Další komerční služba, která umožňuje přepis pomocí síťového API. V současné době bohužel nepodporuje češtinu ani časové značky pro jednotlivá slova, proto je pro naše použití momentálně nevhodná.

3.4.2.3 CMU Sphinx

Kompletní sada nástrojů pro naučení modelu a jeho použití. Tato knihovna je jednoduchá na použití, ovšem už není udržována a proto může v budoucnosti začít značně zaostávat.

3.4.2.4 Kaldi

Podobně jako Sphinx se jedná o sadu nástrojů napsaných v C++ spojených pomocí bash scriptů. Oproti Sphinx je v současnosti projekt stále udržován a podporuje modernější algoritmy. Nevýhodou je poměrně složitá konfigurace a její použití vyžaduje hlubší znalost problematiky.

3.4.2.5 Shrnutí

Na základě analýzy těchto knihoven jsem se rozhodl, že nejlepší, když bude architektura navrhnuta tak, aby bylo možné implementovat do služby jakoukoli ze zmiňovaných knihoven a uživateli pak dát vybrat, kterou knihovnu chce využívat.

V současné době by bylo vhodné implementovat podporu Google Cloud Speech-to-Text a Kaldi, tím se pokryjí oba přístupy, vzdálený i interní přepis.

Kaldi jsem vybral na základě toho, že oproti CMU Sphinx je v současnosti stále v aktivním vývoji, což může v budoucnu přinést možnost použití modernějších a účinějších postupů a tím zlepšit kvalit přepisu.

Během analýzy nahrávek jsem si určil, že bude nejspíše na nahrávkách provádět základní přípravy jako je normalizace hlasitosti, ovšem po konzultaci dokumentace Kaldi a základních zkouškách jsem vyhodnotil, že takové úkony by byly spíše na škodu než k užitku.

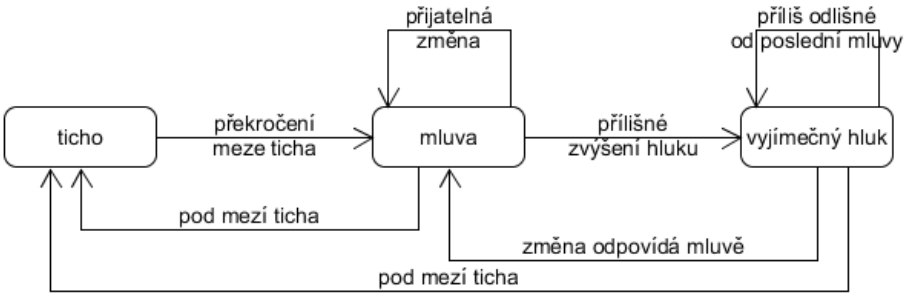
3.4.3 Kvalitativní parametry nahrávky

Součástí analýzy nahrávky je i detekce několika kvalitativních parametrů, jejichž hodnota může alespoň částečně vypovídat o obsahu.

Prvním parametrem je počet překrytí řeči, kdy obě strany hovoru mluví najednou. Takový obsah signalizuje, že dost možná došlo k hádce nebo podobnému nevhodnému chování. Zpracování tohoto parametru usnadní fakt, že jsou k dispozici přepisy obou stran hovoru. Přepisovač navzájem porovná oba textové přepisy a hledá překryvy. Výstupem tohoto parametru je pak poměr počtu slov, která se nějak účastnila překrytí vůči celkovému obsahu přepisu.

Dalším parametrem je počet výkyvů v hlasitosti hovoru. Takový výkyv hlasitosti může v určitých případem znamenat hádku, či neočekávané zvýšení hlasu. K výpočtu tohoto parametru je více přístupů, v tomto řešení používám detekci počtu výkyvů v hlasitosti.

Nejdříve rozdělím nahrávku na menší okna, přibližně o velikosti 10ms, následně v těchto oknech spočítám maximální sílu hlasu. Poté tyto okna procházím a za použití jednoduchého třístavového automatu, který sleduje jestli je hovor ve stavu ticha, mluvení, nebo nečekaného zvýšení, viz diagram 3.2. Výstupem parametru je pak počet úseků, kde došlo k takovému výkyvu.



Obrázek 3.2: Automat pro detekci výkyvů hlasitosti

Tyto parametry přepisovač předává dále ke zpracování současně s přepisem nahrávky, vyhodnocení probíhá pak na straně backendu, který na základě zkalibrovaných hodnot parametrů rozhodne, jestli nahrávky na základě parametrů označit nebo ne.

3.4.4 Komunikace s ostatními moduly

Z pohledu komunikace bude mít modul tři povinnosti, přijímat požadavky na přepis, posílat žadateli výsledek přepisu a informovat žadatele o svém vytížení. Informace o vytížení by měla, v případě, že má žadatel k dispozici více modulů pro přepis umožnit rovnoměrně rozkládat zátěž. S nejvyšší pravděpodobností bude potřeba, aby součástí požadavků na přepis byla i data nahrávky, protože je možné že přepisovač nebude mít přístup k úložišti žadatele.

Aby bylo možné přijímat požadavky i v průběhu přepisu a nemusel backend pokaždé čekat, než je přepis dokončen, bude přepisovač požadavky zařazovat do fronty a backendu bude pouze potvrzovat zařazení do fronty. Backend při požadavku dodá i URL, kterou má přepisovač zavolat ve chvíli, kdy přepis dokončí.

Správné rozložení požadavků mezi více modulů přepisovače bude zodpovědnost backendu, ale bude záležet na informacích dodaných přepisovačem, proto je potřeba rozhodnout, jak bude rozkládání fungovat a jaké informace bude potřeba přenášet.

Pro tuto službu se nabízí následující možnosti:

Náhodné rozdělení Nejjednodušší řešení pro rozdělení práce, každá další nahrávka bude přidělena náhodnému přepisovači. Tato možnost nevyžaduje žádnou dodatečnou komunikaci ani logiku, ovšem nezajišťuje rovnoměrné rozdělení práce. Některé přepisovače mohou být, vzhledem k variabilitě náročnosti přepisu nahrávek více vytížené než ostatní.

Rozdělení podle velikosti front V této metodě je práce rozdělována na základě velikosti front přepisovačů. V případě stejné velikosti je využito náhodné rozdělení. Tato metoda vyžaduje dodatečnou komunikaci pro zjištění velikosti front, ale stále neřeší možnost, že jeden přepisovač má sice stejnou velikost fronty, ale přepisuje mnohonásobně delší nahrávky, čímž může dojít k nevyváženému využití prostředků.

Rozdělení podle délky nahrávek ve frontě Rozšíření předchozí metody s tím rozdílem, že místo velikosti fronty se porovnává celková délka nahrávek ve frontě. Vzhledem k tomu, že se dá očekávat, že doba nutná pro přepis je přímo úměrná délce nahrávky, tato metoda by měla nejefektivněji využívat dostupné zdroje.

Nejvhodnější způsob je tedy rozdělení práce na základě délky nahrávek ve frontě, protože teoreticky nejlépe reprezentuje zatížení modulů přepisovače. Nevýhodou může být o něco větší vytížení komunikačního kanálu, protože potřebné informace se budou muset periodicky obnovovat, jelikož je budou moci ovlivňovat ostatní napojené moduly backendu. Takže povinností modulu přepisovače bude sledovat kolik má ve frontě požadavků a také jakou celkovou délku mají. Tuto informaci bude pak předávat na základě požadavku backendu.

3.4.5 Návrh protokolu

Vzhledem k předem definované funkcionalitě bude přepisovač v rámci REST podporovat následující zdroje:

POST /request Vytvoření nového požadavku na přepis, požadavek by měl obsahovat informace o nahrávce, obsah nahrávky a URL která se musí zavolat ve chvíli kdy je přepis dokončen.

GET /status Vrací informace o vytížení přepisovače. Jak bylo určeno, bude se jednat o velikost fronty a celkovou délku nahrávek ve frontě.

Dá se předpokládat že komunikace v tomto případě bude interní a vždy strojová a tudíž nebude nutné řešit při autorizaci uživatelské účty a udržovat informace o sezení. Autorizace bude probíhat pomocí autorizačního klíče, který se vygeneruje při nasazení modulu přepisovače a bude součástí jeho konfigurace. Klíč bude přítomen v každém požadavku v hlavičce požadavku.

3.4.6 Implementační jazyk

Pro modul přepisovače se nabízí hned několik možností implementace. Pokud bych chtěl implementovat knihovnu Kaldi přímo do kódu, bylo by potřeba použít jazyk C++, protože v něm je celý soubor knihoven implementován. Jazyk C++ je ovšem z možných jazyků nejsložitější na vývoj i údržbu a knihovnu Kaldi lze integrovat i později, buď pomocí pomocných bash scriptů, nebo vytvořením samostatné aplikace, která by se případě integrovala.

Další možností se nabízí jazyk Java, ve kterém by byla implementace jednodušší. Ze zkušeností předpokládám, že při použití Spring frameworku by byla implementace velmi jednoduchá.

Lze předpokládat, že stejně se bude implementovat i backend a použití odlišných jazyků a frameworků u jednotlivých modulů by sice mělo drobné výhody, ale ve výsledku by to vedlo pouze ke zbytečným komplikacím. Proto bude lepší počkat s výběrem jazyka, až se rozhodnu, jaký jazyk použiji pro backend a následně ho použít i pro komunikační logiku přepisovače.

3.4.7 Obecná architektura modulu

Z pohledu architektury se bude jednat o jednoduchý modul, nejsložitější část se bude vykonávat pomocí externích knihoven, které budou pouze integrovány do modulu. Schéma tříd je možné vidět na obrázku 3.3.

Třída `RequestsController` bude předávat požadavky do fronty a poskytovat informace o vytížení. `RestController` prototyp symbolizuje, že se bude jednat o třídu spravující REST komunikaci s ostatními moduly.

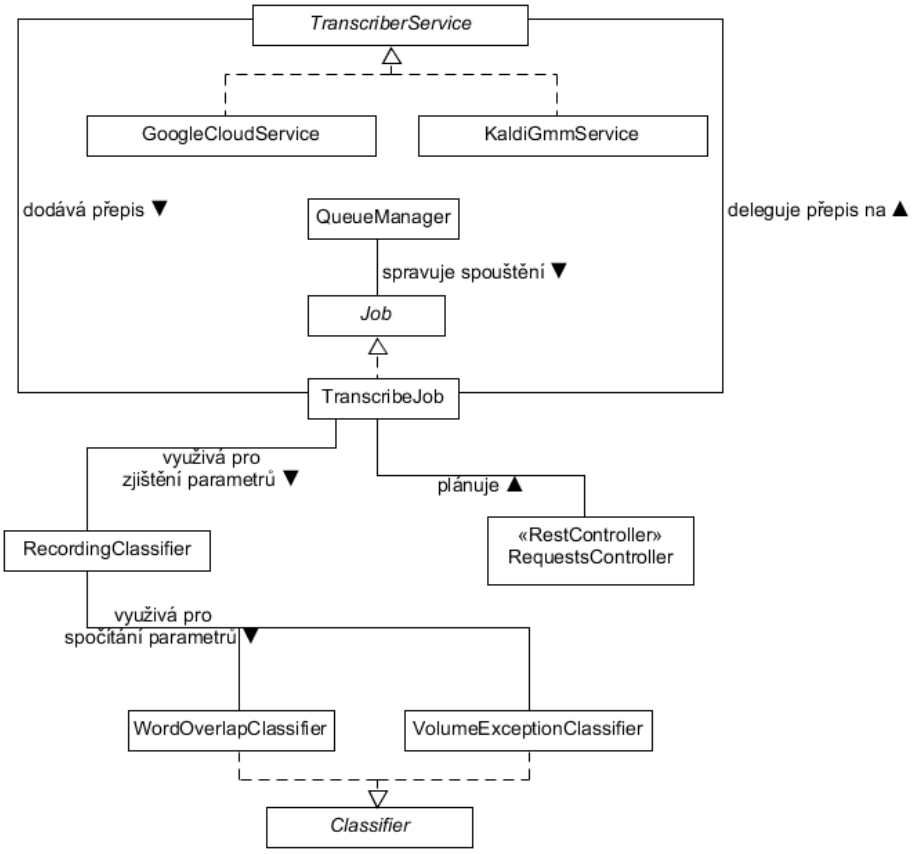
Třída `TranscribeJob` reprezentuje jednotlivý úkol přepisu. V rámci přepisu nejdříve zavolá `TranscriberService`, která vytvoří přepis, který poté současně s nahrávkou předá ke zpracování třídě `RecordingClassifier`, která na základě těchto dat spočítá kvalitativní parametry nahrávky. Výsledná data pak třída `TranscriberJob` předá pomocí síťového komunikačního kanálu backendu, který vytvořil požadavek k přepisu dané nahrávky.

3.5 Návrh backendu

Backend modul bude obstarávat integraci se software telefonních ústředen, předávání nahrávek přepisovači, zpracování přepisů nahrávek a podrobné vyhledávání v seznamu zpracovaných nahrávek.

3.5.1 Vyhledávání v přepisech

Kromě zobrazení přepisu bude služba umožňovat i další funkcionality, pro které bude potřeba přepis dále zpracovat.



Obrázek 3.3: Třídní model přepisovače

Nejdůležitější funkcionalitou bude vyhledávání, které bude kromě klasického vyhledávání klíčových slov podporovat i vyhledávání na základě lingvistické vzdálenosti a vyhledávání různých tvarů zadaných slov.

Pro zajištění této funkcionality už existují hotová a funkční řešení:

Sphinx Vyhledávací server, podporující napojení na SQL, noSQL a souborové databáze. Implementace je v jazyce C++ a funguje jako oddělený modul, se kterým probíhá komunikace pomocí síťového API, což umožňuje škálování do šířky[20].

Apache Lucene Knihovna určená pro textové vyhledávání napsaná v Javě, takže je možné ho přímo integrovat do aplikací používajících JVM[21]. Jedná se o opensource knihovnu, která je stále aktivně udržována a rozšiřovaná a pro toto použití značně rozšířená. V tomto projektu by měla být schopná zajistit vyžadovanou funkcionalitu.

Solr Vyhledávací server postavený na Apache Lucene. Výhodou oproti využití čistého Lucene je jednodušší nastavení, podpora škálování a podrobných statistik. Nevýhodou pak je, že v tomto případě nemá oproti samotnému Lucene žádnou přidanou hodnotu, naprostou většinu přidaných funkcionalit tato služba nepotřebuje[21].

Elasticsearch Další vyhledávací server postavený na Apache Lucene[22]. Z pohledu této služby je podobný řešení Solr. Jedno z nejpoužívanějších řešení tohoto problému.

Vlastní řešení Vytvoření vlastního řešení vyhledávání by vyžadovalo značné vývojové úsilí a ve výsledku by oproti existujícím řešením nepřineslo žádné výhody.

Pro naší službu by bylo možné použít vyhledávací server, protože se jedná o řešení na klíč a navíc obsahují spoustu užitečných funkcí. Ovšem takové řešení je zbytečně předimenzované. V našem případě je potřeba pouze full-text vyhledávání v menším objemu dat, protože i ve větší ústředně velikost prohledávaných dat dosahuje maximálně řádů stovek megabytů.

Proto by použití serveru nebylo efektivní a nejlepší řešení je použít přímou integraci knihovny Lucene, která by měla poskytovat požadovanou funkcionalitu s minimální nutnou konfigurací, takže by toto řešení mělo být přímočaré.

3.5.2 Vytvoření grafu souvisejících slov

Aby bylo možné na frontendu zobrazovat graf slov a frekvence jejich užití, musí si backend vést statistiky využití slov a jejich návaznosti.

Tuto informaci bude backend obnovovat vždy, když přijme nový přepis. Rozdělí jej na slova, porovná se současným seznamem slov a uloží změněné statistiky. Tato úloha může být při větším počtu slov náročnější, ale ne tak, aby ovlivňovala funkčnost backendu, proto by neměl být problém, aby byla vykonávaná při každém příjmu přepisu.

3.5.3 Komunikace s ostatními moduly

Tento modul bude komunikovat jak s modulem přepisovače, tak s modulem frontendu. S modulem přepisovače si bude vyměňovat nahrávky a jejich přepisy, zatímco frontendu bude umožňovat výpis přepisů, přehrávání nahrávek a vyhledávání v přepisech.

Tento modul bude využíván koncovými uživateli, proto bude při autorizaci potřeba počítat s uživatelskými účty a sezeními. Většina zdrojů bude tedy přístupná pouze po vytvoření sezení na základě uživatelského jména a hesla. Autorizace pak bude probíhat pomocí unikátního identifikátoru sezení. Výjimku bude tvořit endpoint pro přijímání přepisů, kde bude autorizace prováděna na základě unikátního klíče, který byl předán přepisovači při vytváření požadavku na přepis.

3.5.4 Návrh protokolu

Vzhledem k předem definované funkcionalitě bude backend v rámci REST podporovat následující zdroje:

POST /transcript Přijetí vytvořeného přepisu od přepisovače. Bude obsahovat identifikátor požadavku, informace o nahrávce a její přepis.

POST/GET/PUT /user Práce s uživatelskými účty.

GET /recordings Vrací seznam nahrávek, tento výpis by měl podporovat i základní stránkování. U každé nahrávky by navíc mělo být možné stáhnout její zvukový obsah.

GET /words Vrací seznam slov, ke každému slovu pak přikládá informaci o jeho užití a jeho nejbližší sousedy, tato informace lze pak využít k vygenerování grafu znázorňující vztahy mezi slovy.

3.5.5 Výběr frameworku

Frameworků pro implementaci REST serveru existuje velké množství, v tomto případě jsem vybíral hlavně mezi těmi nejpopulárnějšími, protože se u nich dá očekávat, že bude v průběhu vývoje dostupná dostatečná podpora a dokumentace, takže bude vývoj dostatečně efektivní.

Hlavním cílem je vybrat framework, který umožní backend implementovat co nejrychleji a s co nejmenším množstvím kódu a tudíž i co nejmenším prostorem pro chyby. Vzhledem k tomu, že backend bude pouze předávat data, neměl by to být problém.

3.5.5.1 Spring boot

Spring boot je nadstavba pro Spring framework. Spring framework je populární a léty prověřený framework pro tvorbu aplikačních serverů. Spring boot je kolekce knihoven, která značně urychluje vývoj základního chování aplikačního serveru[23].

Tento framework je určený pro JVM, proto se v naprosté většině pro vývoj používá Java, ale je možné používat i například modernější Kotlin.

Velkou výhodou je jeho léty prověřená funkčnost, jedná se o nejpopulárnější aplikační framework pro JVM a je neustále ve vývoji. Velká komunita pak znamená, že pro naprostou většinu požadavků už existují řešení, která je potřeba akorát správně nakonfigurovat, čímž lze značně urychlit vývoj.

Tabulka 3.1: Porovnání webových frameworků

	Spring Boot	ASP.NET Core	Laravel
Osobní preference	6	4	7
Rychlost vývoje	10	8	9
Náročnost údržby	8	8	7
Komunita	9	4	5

3.5.5.2 ASP.NET Core

ASP.NET Core je framework pro tvorbu webových aplikací, postavený na .NET Core. Jedná se o relativně nového hráče na trhu, má ovšem podporu od společnosti Microsoft, která je zodpovědná za ASP.NET, což je další léty prověřený framework pro tvorbu webů[24].

Jak název napovídá, běží na .NET Core VM a proto je pro vývoj možné používat rodinu jazyků .NET, z nichž je nejpoužívanější C#. .NET Core VM je také oproti klasickému .NET Frameworku možné používat i na operačním systému Linux, který je nejběžnější volbou pro serverový operační systém.

Přestože se užití tohoto framework začíná rozmáhat, stále je to jen malý hráč na trhu a framework jako takový je stále celkem nový a bez předchozího průzkumu se nedá přesně odhadnout, jestli bude veškerá vyžadovaná funkcionalita dostupná, nebo jestli bude potřeba věci jako autentizaci, nebo CRUD endpointy implementovat ručně.

3.5.5.3 Laravel

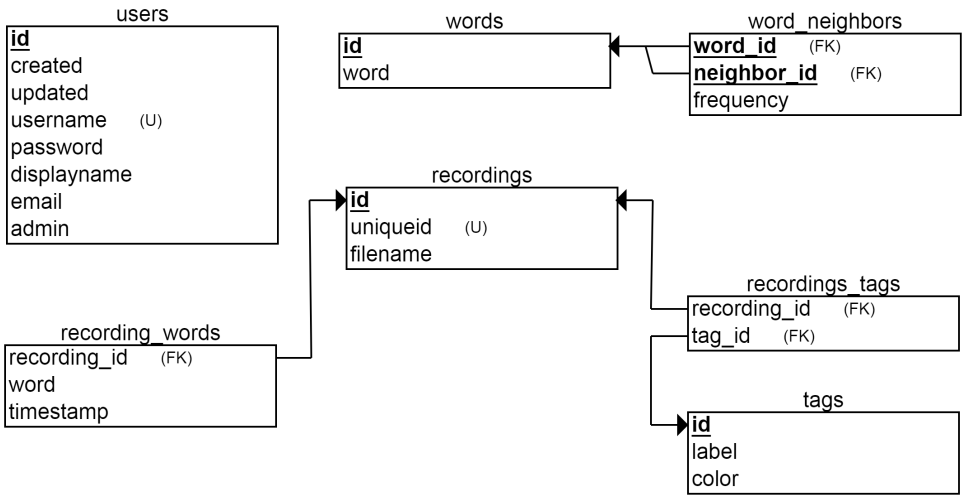
Laravel je jeden z populárnějších frameworků pro jazyk PHP, který je primárně určen pro tvorbu webových služeb[25]. Dá se předpokládat že vývoj v tomto frameworku, za použití PHP 7, by byl rychlý a mohl by dosahovat dobrého výkonu.

Nevýhodou je že se jedná o scriptovací jazyk s dynamickými typy, což může v některých případech vést k nepředvídatelným chybám a negativně tak ovlivnit bezpečnost služby. Navíc by se použitím PHP zkomplikovala integrace se službou Lucene, která se momentálně jeví jako nejvhodnější kandidát pro implementaci fulltextového vyhledávání.

3.5.5.4 Shrnutí

Kromě zmiňovaných vlastností jsem při výběru vhodného frameworku bral v potaz ještě další parametry, které je možné vidět na tabulce 3.1. Tabulka používá desetibodovou stupnici, kdy 10 je nejlepší a 0 nejhorší.

Z porovnání pak nejlépe vychází Spring Boot, který jsme pro implementaci vybral na základě osobní zkušenosti, předpokládané rychlosti vývoje i faktu, že Apache Lucene je také napsán v jazyce Java a usnadní se tím jeho integrace.



Obrázek 3.4: Entity-relationship model databáze

3.5.6 Návrh databáze

Pro správnou funkčnost backendu bude potřeba ukládat některá data do perzistentní databáze. Do databáze se bude ukládat seznam uživatelů, seznam nahrávek a jejich přepisů a seznam slov a jejich vzájemných vazeb, který je vyžadován pro udržování grafu souvisejících slov.

Pro naše požadavky se hodí prakticky jakákoli implementace relační databáze, ke které je dodáván JDBC ovladač. Spring Data, modul Spring frameworku, pak umožňuje pro takovou databázi jednoduchou implementaci REST endpointů, kdy je potřeba jenom nadefinovat strukturu databáze a zbytek vyřeší Spring, bez nutnosti psaní dodatečného kódu.

Navrženou strukturu databáze je možné vidět na diagramu 3.4.

V případě jednotlivých slov u nahrávek může docházet k duplikaci dat, ta je v tomto případě ale zanedbatelná.

3.5.7 Architektura implementace

Na základě požadavků a na základě dokumentace frameworku, jsem navrhl, jak bude vypadat třídní struktura implementace backendu, viz diagram 3.5, který znázorňuje ty nejdůležitější třídy a jejich vzájemné vazby. Jedná se o předběžný návrh a s nejvyšší pravděpodobností se při implementaci alespoň částečně změní.

Stereotypy v návrhu reprezentují anotace, které jsou důležitou součástí Spring frameworku, v tomto případě integrují třídy do frameworku, který na základě anotací implementuje základní očekávané chování tříd.

Interface `PagingAndSortingRepository` označuje repozitáře dat, které kromě základních metod získání dat implementují i podporu pro stránkování, filtrování a řazení. Toho pak využívá funkcionalita `RepositoryRestResource`, která pak vystaví repozitář jako REST zdroj. Toto chování je opět plně automatické, takže není potřeba žádný další kód.

Dále lze v architektuře vidět použití anotace `RepositoryRestController`. Přestože repozitáře ve chvíli, kdy jsou označeny jako `RepositoryRestResource` přístupné automaticky, pro implementaci dodatečného chování je možné vytvořit i kontrolní třídu, která se bude chovat jako REST zdroj repozitáře a zároveň bude umožňovat doimplementaci dalšího chování. Proto počítám s tím, že pro každý repozitář taková třída bude existovat, i když možná nebude potřeba.

Poslední důležitý prvek je `RestController` `TranscriptController`, který se bude starat o přijímání a zpracování přepisů.

Všechny tabulky definované v návrhu databáze pak můžeme vidět jako `Entity`. Tyto třídy budou popisovat strukturu tabulek a jejich základní vlastnosti.

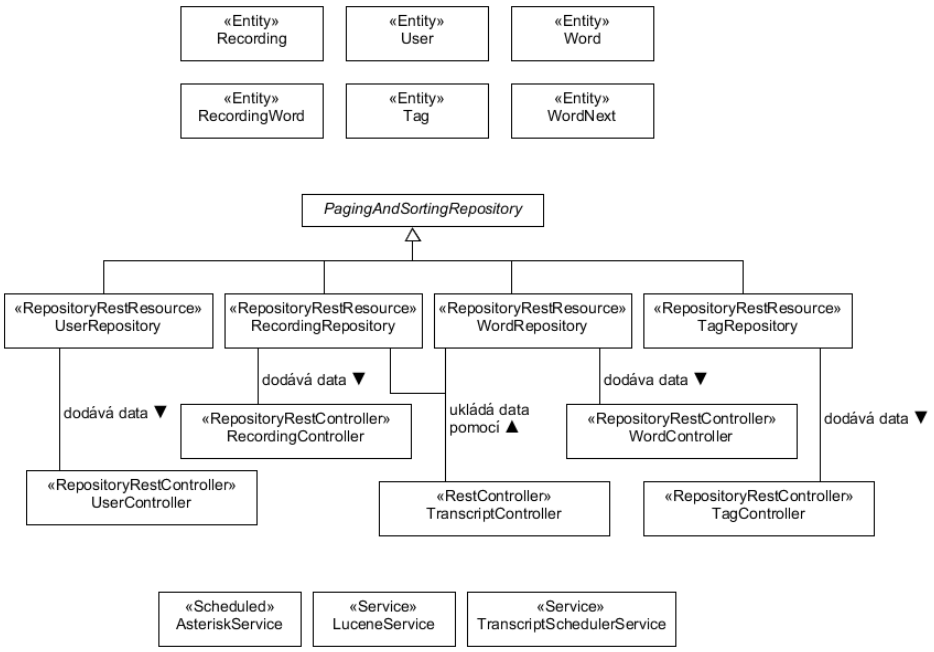
Implementace integrace s dalšími službami, jako je Asterisk a Lucene jsou znázorněné jako služby, anotované jako `Service`, nebo `Scheduled`, kdy je třída volána v pravidelném intervalu. `TranscriptSchedulerService` zajišťuje komunikaci s moduly přepisovače a zároveň bude udržovat seznam a velikosti front jednotlivých přepisovačů, aby byl schopen správně rozkládat zátěž.

3.6 Návrh frontendu

Frontend bude mít na starost přehrávání nahrávek, zobrazování přepisů textu, statistik a základní správu uživatelů systému.

Frontend je možné implementovat více způsoby, pro potřeby této služby jsou nejvhodnější dva možné postupy. Prvním možným postupem je vytvoření webové aplikace. Toto řešení, díky existenci mnohých frameworků, umožňuje rychlé vytvoření funkčního řešení a pro jeho používání je vyžadován pouze webový prohlížeč, který je dostupný na většině operačních systémů a tudíž by se tím zajistila i kompatibilita s více operačními systémy. Nevýhodou je závislost výsledného řešení na aplikacích třetích stran, nad kterými nemáme žádnou kontrolu.

Druhou možností je implementace vlastní samostatné aplikace. V tomto případě je výhodou větší kontrola chování aplikace na koncových zařízeních a menší závislost na aplikacích třetích stran. Nevýhodou ovšem je složitější implementace a s tím související náročnější údržba. Navíc by bylo nutné takovou samostatnou aplikaci instalovat, což může být v případě firem s přísnějšími bezpečnostními požadavky problém.



Obrázek 3.5: Třídní model backendu

3.6.1 Implementace jako webová aplikace

U webové aplikace máme na výběr dva možné přístupy. Generovat obsah na straně serveru a uživateli dodávat už hotový obsah, nebo vytvořit takzvanou Single Page Application (SPA), kdy server uživateli dodá pouze logiku a zobrazovaný obsah je generován na straně uživatele na základě čistých dat dodávaných serverem pomocí předdefinovaného síťového HTTP API.

U obsahu generovaného na straně serveru je výhodou menší náročnost na klienta a v určitých případech i menší objem přenesených dat. Nevýhodou pak větší náročnost na serverovou část, nutnost kombinovat při vývoji více prostředí a jazyků a s nejvyšší pravděpodobností menší plynulost ovládání.

V případě implementace SPA bude zodpovědností serverové části frontendu pouze předání už připravené aplikace klientovi a zbytek logiky se bude provádět na straně klienta. Výhodou SPA je, že logika aplikace není rozdělená mezi klienta a server a tím i daný jednotný jazyk vývoje, což snižuje náročnost vývoje a usnadňuje údržbu. Vzhledem k tomu, jak je navržena komunikace mezi moduly je navíc možné napojit klienta rovnou na backend bez mezičlánku, případně jenom s drobnými úpravami.

Nevýhodami jsou pak větší náročnost aplikace na klienta, závislost funkčnosti aplikace na použitém prohlížeči a nutnost přenést celou aplikaci nehledě na to, kterou část chce uživatel použít, což se může negativně projevit na objemu přenesených dat

Tabulka 3.2: Funkcionalita prohlížečů

Funkcionalita	Podpora*
Komunikace pomocí síťového api	Ano[26]
Přehrávání zvukových nahrávek	Ano[27]**
Ovládání přehrávání zvukových nahrávek	Ano[27]

* Podpora v rámci prohlížečů IE verze 11, Chrome verze 69, Firefox verze 20

** Podporované formáty se v rámci různých prohlížečů liší

Je nutné poznamenat, že v případě nutnosti je možné webovou aplikaci transformovat na samostatnou aplikaci tak, že se bude distribuovat současně s prohlížečem, takže výsledek bude možné používat i bez statického serveru dodávajícího logiku klienta.

3.6.2 Implementace jako samostatné aplikace

Implementace jako samostatné aplikace by měla výhody v možnosti více podrobné optimalizace a menší závislosti na aplikacích třetích stran (prohlížeči). Tato závislost totiž znamená, že pokud daná aplikace obsahuje chybu, nebo neobsahuje vyžadovanou funkcionalitu, nebude možné během vývoje chybu opravit či funkcionalitu dopracovat.

Nevýhodou ovšem je, že oproti webové aplikaci je dosažení požadované funkcionality náročnější, což je dané i nutností soustředění na možná cílová uživatelská zařízení. Je pravděpodobné, že frontend bude používán pouze na osobních počítačích, ovšem lepší je mít otevřenou cestu pro budoucí rozšíření, které by v případě samostatné aplikace mohlo být komplikované.

3.6.3 Vyhodnocení možností implementace

Webová aplikace má zásadní výhody v rychlosti vývoje a absence nutnosti instalace na straně uživatele. Ovšem oproti samostatné aplikaci je tu potřeba počítat se závislostí na prohlížeči, ale jak je možné vidět na tabulce 3.2 potřebná funkcionalita je v současných verzích prohlížečů dostupná, takže to není překážka.

Na základě zmíněných vlastností a dalších parametrů viděných v tabulce 3.3 bylo pro frontend vybráno prostředí webové aplikace. Tabulka používá desetibodovou stupnici, kdy 10 je nejlepší a 0 nejhorší.

U webové aplikace je potřeba určit jestli vybrat SPA, nebo klasickou kombinaci serveru a klienta. Vzhledem k uvedeným kladům a záporům a vzhledem k vyhodnocení 3.4 bylo rozhodnuto, že frontend bude SPA. Tabulka používá stejnou stupinici jako předchozí porovnání.

Tabulka 3.3: Porovnání možných implementací frontendu

	Webová aplikace	Samostatná aplikace
Znalost prostředí	9	7
Rychlost vývoje	10	6
Náročnost údržby	8	5
Komunita	9	7

Tabulka 3.4: Porovnání možných implementací webové aplikace

	Server a klient	SPA
Znalost prostředí	8	9
Rychlost vývoje	8	10
Náročnost údržby	7	9
Komunita	8	9

3.6.4 Architektura implementace

Pro SPA existuje velké množství frameworků, pomocí kterých lze frontend implementovat. Nejpoužívanější frameworky, které mají zároveň i největší komunitu jsou následující:

React Jeden z prvních SPA frameworků, který je zároveň momentálně nejpoužívanější a má tedy největší komunitu[28]. Oproti ostatním frameworkům ovšem pokulhává ve všech ohledech[29, 30]. Architektonický návrh frameworku je značně nešťastný (kód šablon je spojený s kódem zbytku aplikace) a vede k špatně udržitelnému kódu, ze všech porovnávaných frameworků je nejpomalejší a pro doplnění funkcionality nutné pro vytvoření plně funkční SPA je potřeba využít doplňující moduly, někdy i neoficiální, což také zhoršuje udržitelnost kódu.

Angular Společně s React byl jeden z prvních frameworků, ovšem už se mu nepodařilo vybudovat tak velkou komunitu. Tento projekt je oproti ostatním službám už od základu zamýšlen pro tvorbu komplexních projektů a tomu odpovídá i jeho architektura, která vede k modulárnosti a s tím související dobré udržitelnosti kódu. To se ovšem negativně projevuje na náročnosti na vývoj a proto je výhodné tento framework používat spíše u velkých projektů, kde tato architektura vývoj zjednodušuje a zpřehledňuje.

Vue.js Projekt značně podobný React frameworku. Jeho výhodou je celkově menší velikost, jednoduše realizovatelné oddělení kódu šablon a zbytku aplikace a oproti ostatním zmíněným frameworkům dosahuje největšího výkonu. Nevýhodou je pak menší komunita, což je dáno menší popularitou. V současné době ovšem komunita neustále roste a existuje množství užitečných rozšíření, které vývoj ještě více usnadňují.

Vanilla.js Fiktivní framework, kterým se označují SPA vytvořené bez použití frameworku, pouze pomocí čistého Javascriptu. Tento způsob vývoje vede k nejlepšímu výkonu, ovšem u jakéhokoli trochu většího projektu značně zvyšuje náročnost vývoje. Tento přístup se vyplatí pouze u projektů, kde je výkon kritický a i u těch je dobré nejdříve zvážit, jestli není možné využít správným způsobem existující framework.

Z vybraných frameworků jsem vybíral podle zmíněných výhod i nevýhod a také na základě osobní zkušenosti a preference. Nejlépe pak vychází Vue.js, hlavně proto, že s ním mám největší zkušenosti a jsem schopen v něm efektivně a rychle modul implementovat.

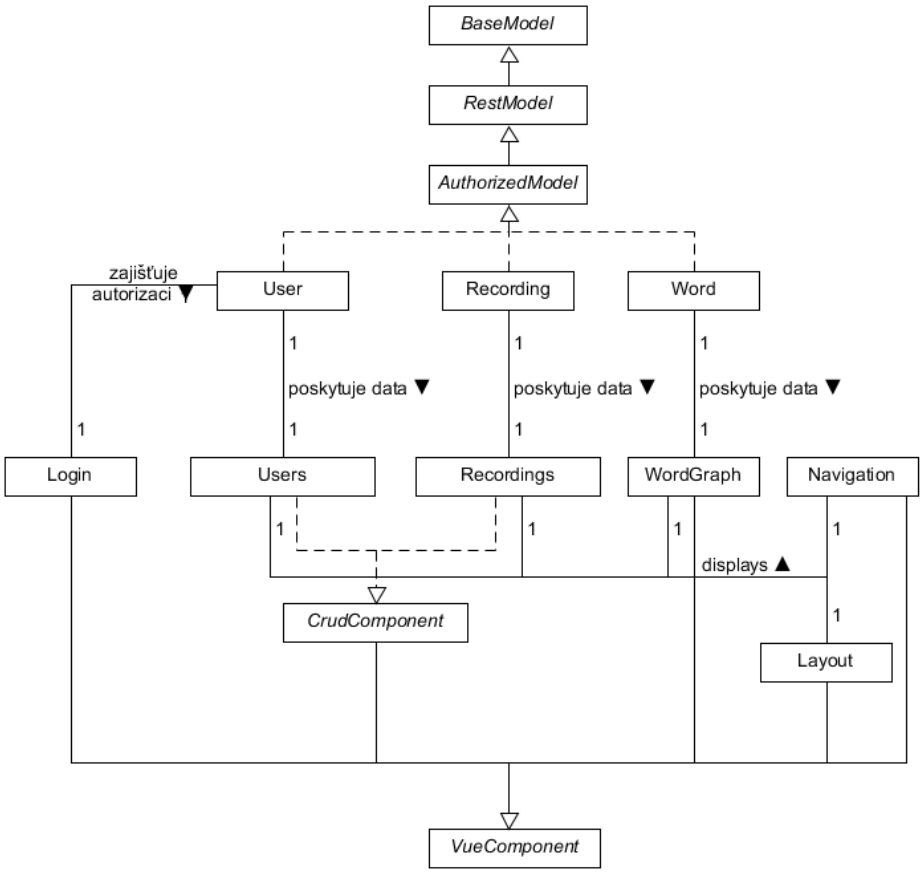
Nutno poznamenat, že tento framework, jako většina zmíněných, se soustředí výhradně na zobrazování dat. Jejich načítání a případné úpravy se musí implementovat zvlášť, buď použitím existujícího řešení, nebo vytvořením vlastní implementace.

3.6.5 Architektura frontendu ve Vue.js

Na základě znalosti frameworku a znalosti požadavků jsem navrhl architekturu frontendu, kterou je možné vidět na diagramu 3.6. Diagram znázorňuje nejdůležitější třídy a jejich vzájemné závislosti. V horní části je možné vidět třídy, které se starají o komunikaci s backendem a v dolní části třídy které interagují s uživatelem. Komunikace je navržená tak, aby bylo možné v případě potřeby změnit její fungování pouze modifikací základního modelu.

VueComponent je základní třída frameworku, která umožňuje zobrazovat interaktivní prvky uživateli. Třídy které z ní dědí se starají jak o zobrazení dat tak i o jeho zpracování, ovšem díky způsobu kterým framework funguje jsou v kódu tyto dvě funkcionality oddělené, protože pro view se používá speciální jazyk založený na HTML.

`CrudComponent` je komponenta která se stará o zobrazování a volitelně i úpravu dat poskytovaných modelem. Podporuje extenzivní konfiguraci, pomocí které lze dosáhnout i komplikovanějšího chování zobrazení a úprav. Díky shrnutí všech výpisů do jedné komponenty dosáhnu konzistentního chování napříč aplikací a navíc si tím usnadním možné dodatečné úpravy.

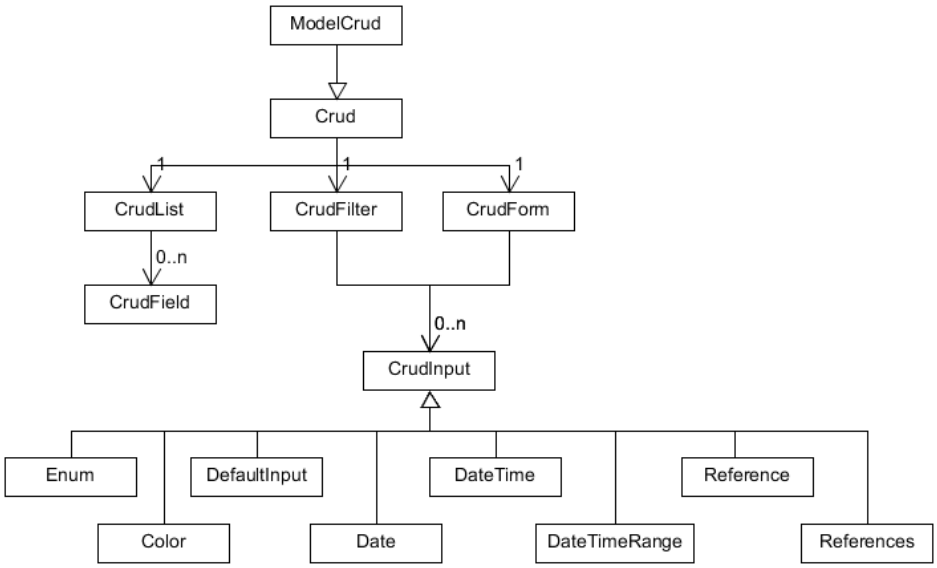


Obrázek 3.6: Výsledná architektura frontendu ve Vue.js frameworku

Tato komponenta bude složitější, proto jsem si předem navrhl její interní architekturu, kterou lze vidět na diagramu 3.7. Základ tvoří `Crud`, komponenta která zpracuje dodaný seznam položek a pomocí dalších komponent je zobrazí uživateli. `ModelCrud` je její rozšíření, které integruje chování zadaného modelu. Komponenta `CrudList` se stará o zobrazení seznamu položek, stránkování, filtrování a, pokud to bude povolené umožní uživateli přejít na úpravu položek. `CrudForm` pak umožňuje samotnou úpravu položek, k tomu má k dispozici komponenty, které jsou navržené na úpravu různých datových typů. Ty pak sdružuje komponenta `CrudInput`.

Komponenty `Users` a `Recordings` pak budou pouze dodávat `CurdComponent` model pro použití a dodatečnou konfiguraci. Tyto modely nemají žádnou návaznost na samotný Vue framework a případně je lze použít i v jiných knihovnách.

`Layout` bude přihlášenému uživateli zobrazovat navigaci a na základě současného stavu aplikace zobrazovat uživatelem vyžádaný obsah.



Obrázek 3.7: Architektura CRUD componenty

Zvlášť je stránka `Login`, která neobsahuje navigaci ani podobné prvky, pouze umožňuje uživateli přihlásit se pomocí uživatelských údajů.

3.6.6 Výběr UI kitu

Z pohledu základní functionality a stylu zobrazení není potřeba začínat od nuly. Existují takzvané UI kity, tedy kolekce frontendových nastylovaných prvků, které už poté stačí poskládat do funkčního rozhraní. Takových kitů existuje větší množství a každý má své pro a proti, proto je potřeba pečlivě zvážit jaký je nejlepší použít.

Na základě základního průzkumu jsem došel k následujícím kandidátům. Tyto kandidáty jsem prozkoumal podrobněji a případně i vyzkoušel jejich funkcionalitu.

Bootstrap Vue Jedná se o obal čistě stylového kitu Bootstrap, který kromě stylů poskytuje i základní funkcionalitu pro většinu obsažených prvků.

Vuetify Mobile first UI kit, který se drží takzvaného material designu, což je soubor design principů navržený pro tvorbu UI určeného pro mobily i počítače zároveň[31].

Buefy Tento kit je podobně jako Bootstrap Vue obal existujícího stylového kitu Bluma. Kromě základních stylů obsahuje i množství různých funkčních komponent jako je podpora většiny formulářových vstupů a podobně.

Element UI Další z řady UI kitů, obsahuje poměrně velkou kolekci komponent a základních rozložení stránky.

Podle vlastních zkušeností a také na základě dokumentace jsem došel k závěru že pro naše použití bude nejvhodnější použít Element UI, obsahuje všechny potřebné komponenty a zdá se že cílí hlavně na počítače, což je i naše hlavní cílová skupina.

3.6.7 Návrh nástrojové sady

Ekosystém javascript aplikací je momentálně stále v urputném vývoji, proto pro každý krok vývoje aplikace existuje hned několik možných nástrojů, proto je nutné si před začátkem vývoje určit, jaké nástroje budu při vývoji používat.

Hned prvním rozhodnutím je výběr správném package manageru, tedy nástroje který obstarává instalaci knihoven. V současné době jsou nejpopulárnější dva nástroje:

Node Package Manager (NPM) Jedná se o package manager dodávaný současně s nodejs, základem jakéhokoli vývoje frontend aplikací v javascriptu.

Yarn Alternativa k základnímu package manageru, v době vzniku obsahoval množství užitečných dodatečných funkcionalit, nejdůležitější byl deterministický přístup k instalaci knihoven, kdy si narozdíl od NPM udržoval vlastní lockfile, kam se zapisovala přesná verze instalace pro každou instalaci, tím se zajistí, že se bude při každé nové instalaci aplikace chovat stejně. Ovšem většina funkcionality se postupem času dostala i do NPM, proto dnes oproti NPM nemá příliš mnoho výhod.

Vzhledem k tom, že Yarn už neposkytuje mnoho výhod, budu jako package manager používat NPM.

Další volbou je způsob kompilace aplikace. K tomu se nám nabízí hned několik řešení, jako nejpopulárnější jsou tyto tři:

Webpack Nejpoužívanější nástroj, určený pro kompilaci komplexních javascript aplikací do balíku spustitelného v prohlížeči. Díky popularitě obsahuje množství dodatečných pluginů a knihoven.

Browserify Stejně jako webpack, je účelem Browserify vytvoření balíčku spustitelného v prohlížeči, ovšem oproti Webpack značně ztrácí na komplexitě a přehlednosti konfigurace. Tento nástroj je vhodný hlavně pro menší projekty.

Rollup Nový nástroj, určený k tvoření balíčků jak pro prohlížeč, tak i pro nodejs. Hlavní výhodou nástroje je minimální nutnost a složitost konfigurace, která ovšem může v případě větších projektů být spíše na škodu.

Všechny zmíněné nástroje je možné použít pro vývoj pro Vue.js, na základě parametrů jsem pak rozhodl, že použiji Webpack. Při výběru jsem se řídil hlavně tím, že webpack je díky množství dodatečných pluginů nejvíce flexibilní a také tím, že s ním mám největší osobní zkušenost.

3.7 Návrh uživatelského rozhraní

Při návrhu uživatelské rozhraní jsem uplatnil postup, kdy jsem nejdříve na základě funkčních požadavků navrhl několik základních obrazovek aplikace. Tento návrh jsem provedl pouze graficky a na jeho základu jsem vytvořil takzvaný Hi-fi prototyp, tedy prototyp aplikace, který už umožní uživateli alespoň nasimulovat většinu funkcionality aplikace.

Tento model jsem poté prezentoval skupině testerů, kteří měli za úkol vykonat několik základních úkonů založených na případech užití. Tyto úkony vykonávali pod mým dohledem, ale bez mé pomoci. Účelem bylo zjistit jestli je aplikace z pohledu uživatele přehledná a intuitivní.

Na základě takto získané zpětné vazby jsem poté provedl několik úprav rozhraní a předělal hi-fi prototyp na samotnou aplikaci frontendu, což byl díky tomu, že pro hi-fi prototyp už byl předem použit stejný framework, jako pro cílovou aplikaci, jednoduchý úkon.

3.7.1 Specifikace požadavků uživatelského rozhraní

Z podstaty služby vyplývá, že toto uživatelské rozhraní bude využívané úzkou skupinou lidí. S nejvyšší pravděpodobností bude využívané v interní síti a na osobních počítačích. Jeho cílová skupina budou administrativní pracovníci a middle management.

Na rozdíl od rozhraní, u kterého se počítá, že bude využíváno veřejností, tak máme předem vymezeno, na jaké skupiny a prostředky je dobré se soustředit.

Nabízí se nám možnost navrhnout rozhraní mobile first, aby bylo hlavně intuitivní a správně rozložené na mobilních zařízeních a pro osobní počítače by se pouze drobně upravilo, aby bylo použitelné. Ovšem v našem případě se nedá počítat s velkým počtem uživatelů mobilních zařízení, naopak s nejvyšší pravděpodobností bude většina uživatelů používat osobní počítače.

Proto bude uživatelské rozhraní navrženo primárně pro osobní počítače a pro menší obrazovky se bude, pokud bude čas, jen drobně optimalizovat, aby jej bylo možné alespoň omezeně používat.

Při implementaci rozhraní se budu soustředit, aby bylo rozhraní alespoň částečně responzivní, to nemusí být o tolik náročnější, než běžný návrh, ale značně se tím usnadní konečné úpravy pro zobrazení na mobilních zařízeních.

Obrázek 3.8: Wireframe přihlašovací obrazovky

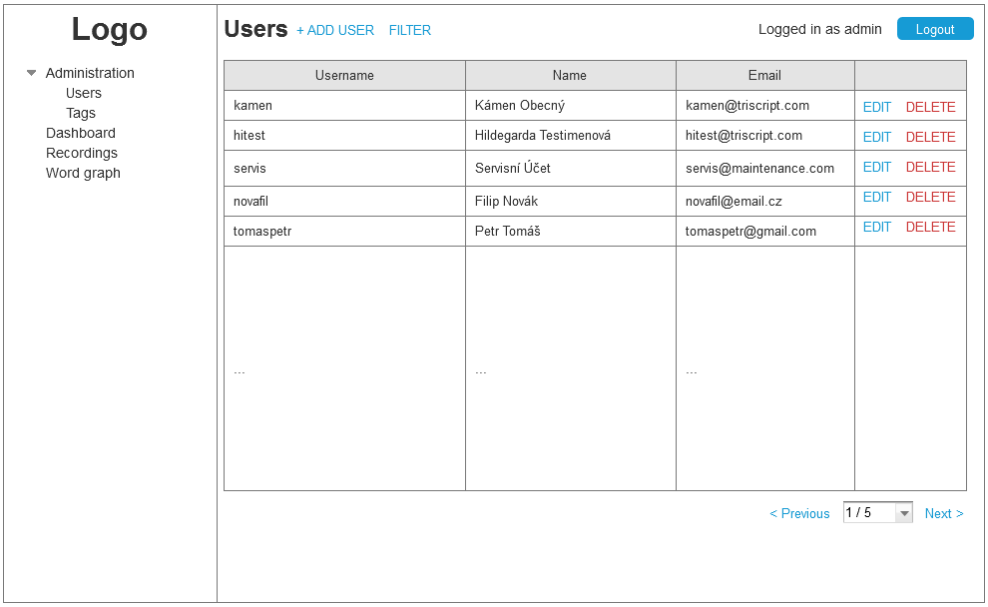
3.7.2 Počáteční návrh

Vytvoření základního počátečního návrhu umožňuje vyjasnit si, jak má vlastně aplikace vypadat a pořádně si utřídit myšlenky. Navíc se tím zjednoduší následující implementace Hi-fi prototypu a umožňuje nám zjistit a vyřešit základní problémy před započetím implementace.

Pro počáteční návrh jsem použil takzvané wireframes, tedy naprosto základní grafické vyjádření rozložení prvků na obrazovce. Tyto návrhy jsem vytvářel hlavně na základě funkčních požadavků a případů užití a to tak, aby soubor návrhů pokryl všechny očekávané prvky aplikace.

Prvním wireframe byl wireframe přihlašovací obrazovky, viděný na obrázku 3.8. Tato obrazovka má jiné rozložení, než zbytek aplikace. To je dáno tím, že uživatel v tuto chvíli nemá k dispozici jinou akci než vyplnění údajů a autorizaci. Tento návrh se tedy snaží uživatele navést právě na tuto akci.

V dalším wireframe; seznam uživatelů (obrázek 3.9) už vidíme rozložení obrazovky pro přihlášeného uživatele. Vlevo je seznam funkcí aplikace, umožňující uživateli mezi nimi přepínat. Vpravo pak vidíme současnou funkci, v tomto případě se jedná o seznam uživatelů a dostupných akcí pro každého uživatele.



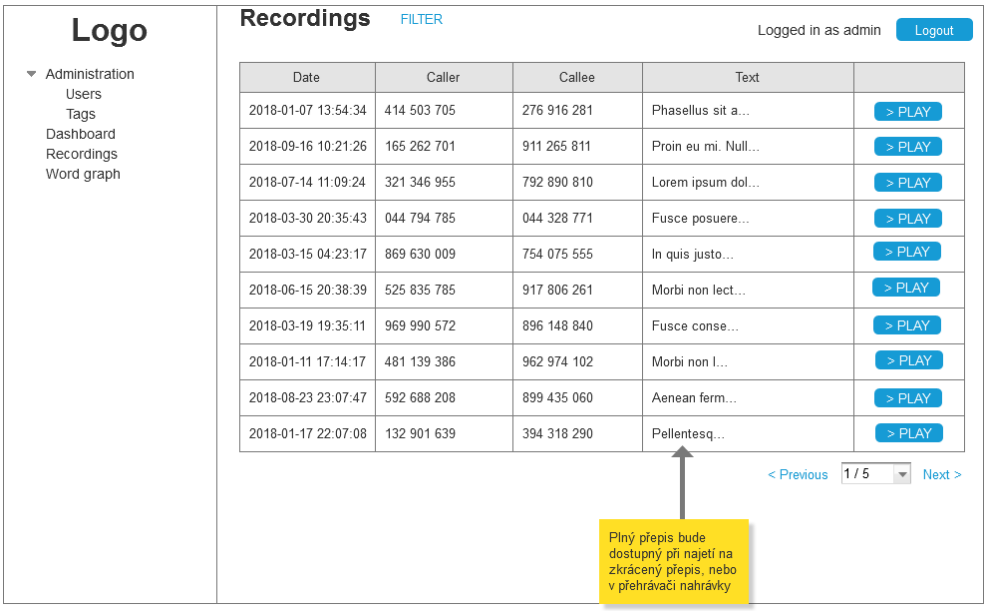
Obrázek 3.9: Wireframe seznamu uživatelů

Ve wireframe seznamu nahrávek (obrázek 3.10) lze vidět, jak bude systém zobrazovat pro uživatele nejdůležitější funkci, seznam nahrávek a jejich přepisů. Kromě základních informací o hovoru zobrazuje i výtažek přepisu, nebo v případě, že je přepis dostatečně krátký, celý přepis. Seznam umožňuje uživateli seznam nahrávek prohledávat pomocí filtrů, které jsou rozvedeny v dalších wireframech. Dále je uživateli nabízena akce přehrání nahrávky, která také nabízí možnost zobrazení celého přepisu.

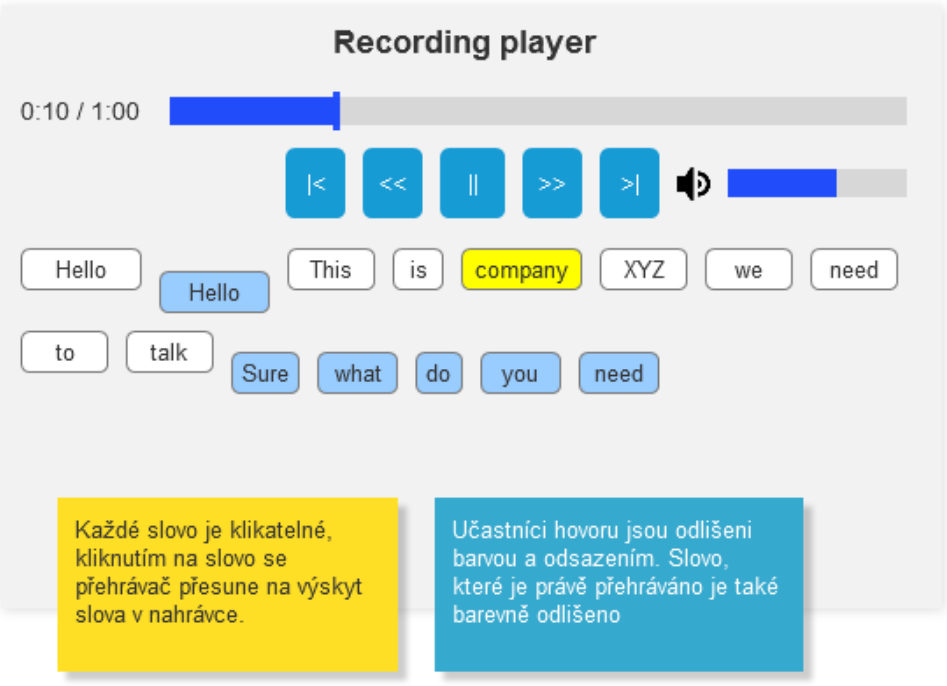
V případě, že se uživatel rozhodne nahrávku přehrát, nebo zobrazit celý přepis, zobrazí se mu nová obrazovka obsahující základní ovládání přehrávání a seznam slov použitých v hovoru. Tento návrh je možné vidět na obrázku 3.11. U seznamu slov je možné pomocí barev rozlišovat volajícího a volaného a zároveň dynamicky reaguje na přehrávač zvýrazňováním právě mluveného slova.

Pro filtrování je použit jednoduchý formulář, viděný na obrázku 3.12, který se zobrazí po kliku na akci filtrování. Formulář umožňuje uživateli specifikovat textový filtr pro některé zobrazené sloupce. Uživatel má také možnost filtry resetovat, aby bylo možné se okamžitě vrátit do výchozího stavu.

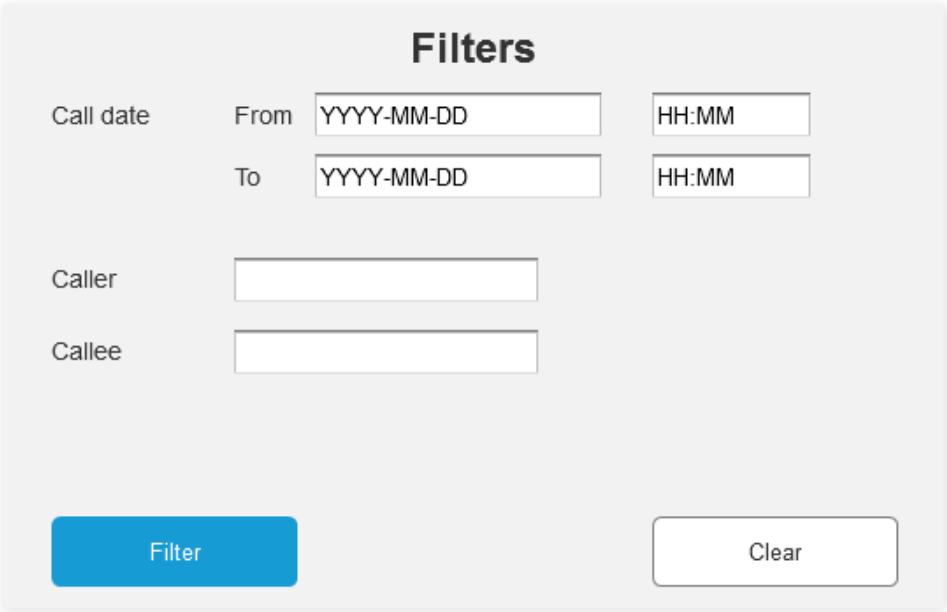
Pro úpravu a vytváření uživatelů se bude používat standardní formulář, obrázek 3.13, který bude dynamicky reagovat na změny a bude zobrazovat chybové hlášky na základě uživatelského vstupu. Formulář bude stejný při úpravě i vytváření, pouze se bude lišit povinnost hesla. V případě úpravy se bude heslo měnit pouze v případě, že uživatel vyplní příslušná pole.



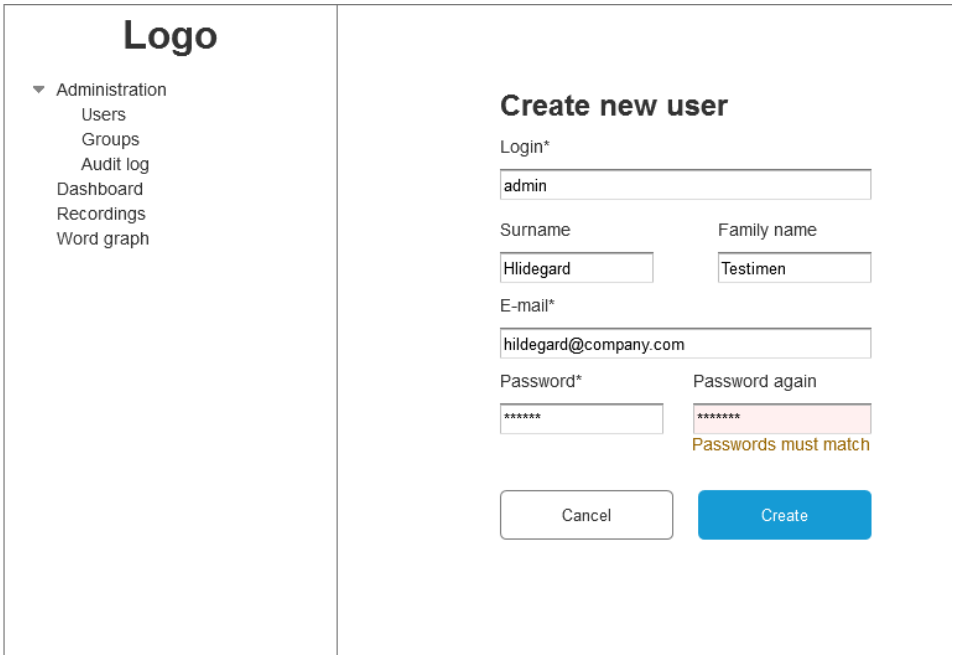
Obrázek 3.10: Wireframe seznamu nahrávek



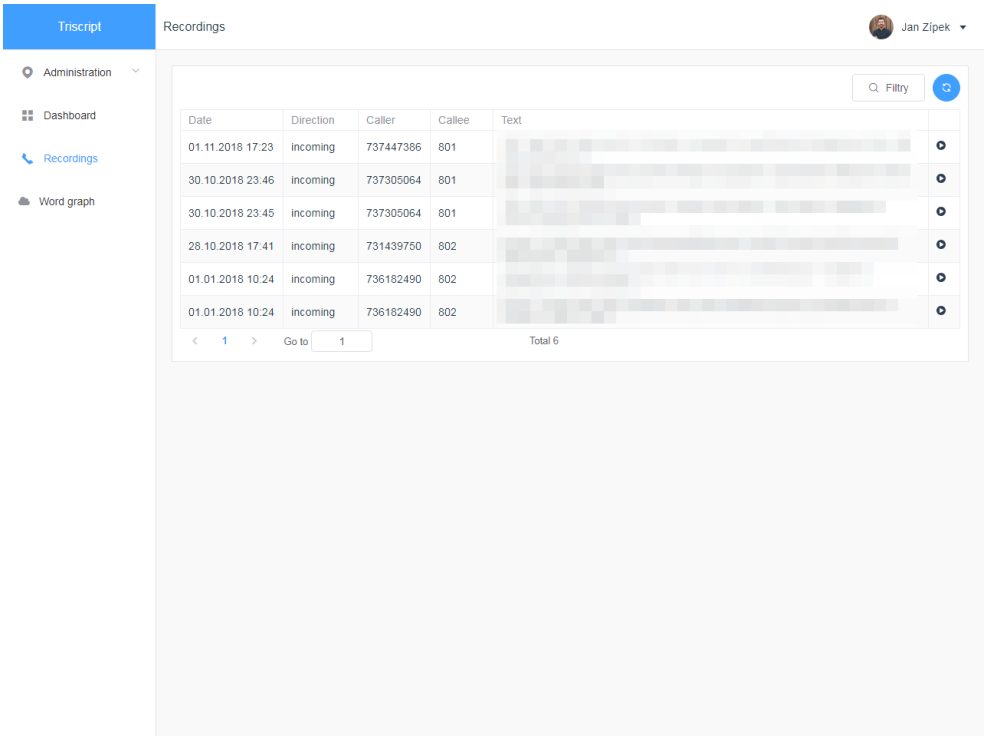
Obrázek 3.11: Wireframe přehrávače nahrávek



Obrázek 3.12: Wireframe filtrů seznamů



Obrázek 3.13: Wireframe formuláře uživatele



Obrázek 3.14: Hi-fi prototyp seznamu nahrávek

3.7.3 Hi-fi prototyp uživatelského rozhraní

Na základě návrhů jsem poté vytvořil hi-fi prototyp. Tento prototyp se vytváří, aby bylo možné získat od uživatelů spolehlivou zpětnou vazbu. Zároveň je vytvářen s minimální a většinou pouze nasimulovanou funkcionalitou, aby bylo možné efektivně a jednoduše provádět na základě zpětné vazby úpravy.

Tento prototyp jsem se rozhodl rovnou implementovat ve stejném prostředí, jako které bylo navrženo pro implementaci frontendu. To má tu výhodu, že ve chvíli, kdy je hi-fi prototyp hotový a otestovaný, je možné rychle přejít k implementaci frontendu. Zvolený framework navíc umožňuje rychlý a jednoduchý návrh a dodatečné úpravy funkčního prototypu.

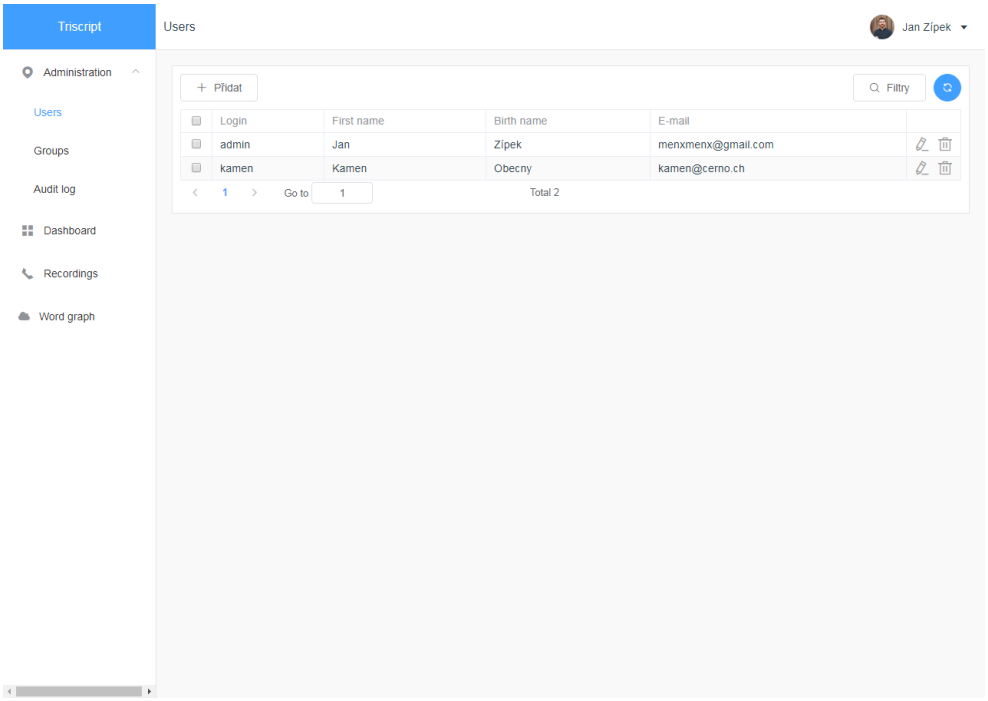
Prototyp tedy už obsahoval alespoň základní funkcionalitu, ovšem nebyl napojen na backend, takže zobrazoval pouze ukázková data.

Tento prototyp byl poté předán testerům, kteří na jeho základě podávali zpětnou vazbu.

Podobu hi-fi prototypu lze vidět na obrázcích 3.14, 3.15, 3.16, 3.17 a 3.18.

3.7.4 Testování hi-fi prototypu

Hi-fi prototyp byl vytvořen na základě wireframů a pro testování bylo vytipováno několik potencionálních uživatelů.



Obrázek 3.15: Hi-fi prototyp seznamu uživatelů

Testování hi-fi prototypu je prováděno s potencionálními uživateli a pod dohledem návrháře rozhraní. Návrhář v tomto případě plní pouze funkci pozorovatele. Někdy je průběh testování nahráván do záznamu a to proto, aby bylo možné si jeho akce zpětně projít při vyhodnocování průběhu.

Tester dostane seznam scénářů, které má bez pomoci provést. Scénáře by měly být koncipovány tak, aby co nejvíce odpovídaly reálnému využití aplikace a zároveň by jejich kolekce měla pokrývat veškerou funkcionalitu aplikace, proto tyto scénáře alespoň částečně vycházejí z případů užití.

3.7.4.1 Testovací scénáře

Hlavními personami scénářů jsou tu manažer a správce, tyto osoby budou nejběžnějšími uživateli rozhraní a někdy se může jednat o stejnou osobu.

Cílem scénářů je zjistit, jestli je uživatel schopen v rozhraní rychle zorientovat a dosáhnout v efektivním čase splnění cílů scénáře.

Testování probíhalo na hi-fi prototypu za pomocí vygenerovaných testovacích dat, upravených tak, aby testovací scénáře měly smysl.

Scénáře byly následující:

Scénář 1 Manažer telefonní ústředny si chce prohlédnout přepis hovoru ze včerejšího večera, o které se mu zmínil jeden z agentů s tím, že se odehrál mezi 15 a 17 hodinou.

Obrázek 3.16: Hi-fi prototyp přehrávače nahrávek

Scénář 2 Manažer si chce vyhledat všechny nahrávky, které se nějak zmiňují o ukončení smlouvy.

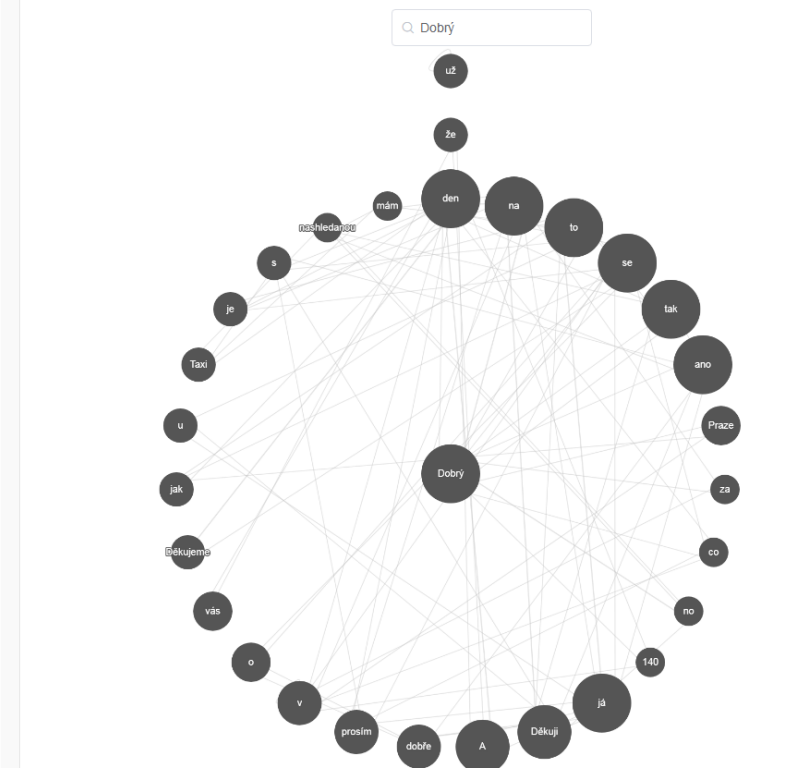
Scénář 3 Manažer si chce zobrazit, jak často je používáno slovo dobrý a jaká slova po něm nejčastěji následují.

Scénář 4 Správce telefonní ústředny chce vytvořit uživatelský účet novému manažerovi.

Scénář 5 Manažer si chce změnit přihlašovací heslo.

3.7.4.2 První testovací subjekt

Informace



Obrázek 3.17: Hi-fi prototyp grafu slov

Věk: 50-60

Povolání: Marketingový manažer

Zkušenost s počítači: Základní

Vyhodnocení

Scénář 1 Trochu problém najít výpis nahrávek a tlačítko filtrovat, jinak bez problému zvládl nastavení filtru.

Scénář 2 Na základě předchozí zkušenosti bez problému, ovšem nulové využití pokročilých možností hledání.

Scénář 3 Trochu problém s vyhledávacím dialogem, uživatel nečekal, že musí na slovo v nabídce kliknout a nestačí pouze zadat celé slovo a zmáčknout enter.

Scénář 4 Bez problému splněno.

Obrázek 3.18: Hi-fi prototyp přihlašovacího formuláře

Scénář 5 Uživatel trochu bloudil, ale nakonec našel správnou sekci vpravo nahoře.

3.7.4.3 Druhý testovací subjekt

Informace

Věk: 18-20

Povolání: Student

Zkušenost s počítači: Střední

Vyhodnocení

Scénář 1 Bez problému, trochu problém s funkčností filtru datového rozmezí, ale nakonec zvládnuto bez pomoci.

Scénář 2 Uživatel trochu bloumal a hledal jak má vyhledávat na základě textu, nakonec ale našel správné pole v seznamu obecných filtrů.

Scénář 3 Nalezeno bez problému, ale graf slov a souvislostí se zdál být nepřehledný.

Scénář 4 Uživatel vytvořen bez problému, ovšem se špatnou rolí, což bude spíše zmatení zadání scénáře.

Scénář 5 Uživatel trochu bloumal, kde najít správné tlačítko, ale nakonec sekci našel.

3.7.4.4 Třetí testovací subjekt

Informace

Věk: 30-40

Povolání: CEO IT Společnosti

Zkušenost s počítači: Vysoká

Vyhodnocení

Scénář 1 Uživatel nemohl najít tlačítko pro filtrování, navíc nebylo úplně jasné, jak správně vyfiltrovat nahrávky v určitém rozsahu, uživatel raději zadal jenom horní limit a hledal ve zbytku ručně.

Scénář 2 Uživatel nejdříve hledal vyhledávací funkci, ale poté mu došlo, že je opět dostupná přes filtry.

Scénář 3 Uživateli nebylo moc jasné, za jakou funkcionalitou se dá tato informace zjistit. Ovšem vhledem k omezenému počtu možností nakonec informaci našel.

Scénář 4 Bez problému.

Scénář 5 Uživatel nenalezl funkci změny profilových dat, protože byla skryta za vysouvacím menu, což ho vůbec nenapadlo.

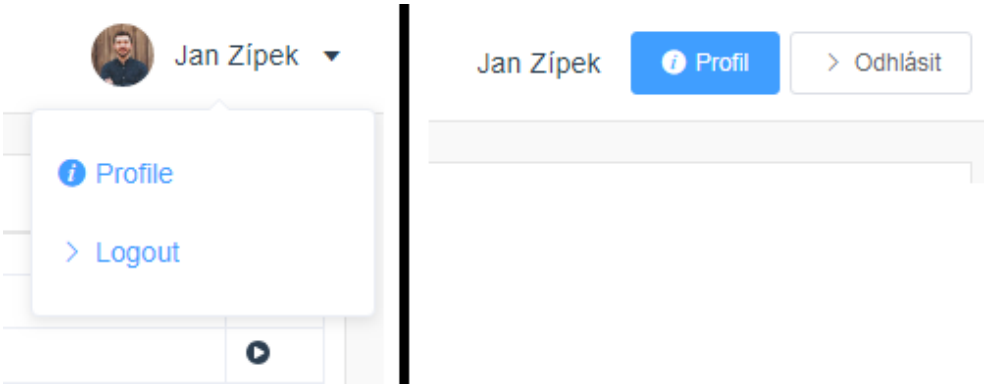
3.7.4.5 Čtvrtý testovací subjekt

Informace

Věk: 20-30

Povolání: Administrativní pracovník

Zkušenost s počítači: Střední



Obrázek 3.19: Úprava akcí přihlášeného uživatele, vlevo původní podoba, vpravo nová

Vyhodnocení

Scénář 1 Nahrávka byla bez problému nalezena, během hledání se zdál uživatel trochu zmatený vzhledem filtrovacího pole pro datum a čas, ale po chvíli se zorientoval.

Scénář 2 Uživatel bez problému našel, jak filtrovat podle textu, což ale může být také proto, že už v předchozím kroku viděl filtrovací dialog a všiml si ho.

Scénář 3 Nejdříve tápal, ale poté vybral správnou položku v menu a slovo bez problému vyhledal a zobrazil.

Scénář 4 Bez problému našel příslušnou sekci a formulář.

Scénář 5 Opět bez problému, neměl problém se skrytými možnosti.

3.7.4.6 Vyhodnocení testování

Na základě testování byl sestaven seznam nejzávažnějších problémů, které uživatelé objevili. Tyto problémy jsem pak kategorizoval podle jejich významnosti a pokusil se je zásahy do návrhu opravit. Seznam problémů a jejich řešení lze vidět v tabulce 3.5.

Jako příklad provedených úprav můžeme vidět obrázek 3.19, kde byla tlačítka pro změnu profilových údajů a odhlášení přesunuta ze skrytého menu.

Tabulka 3.5: Problémy nalezené při testování

ID	Popis	Závažnost	Řešení
1	Nevýraznost tlačítka pro filtrování.	Střední	Tlačítko zvýrazněno
2	Textové vyhledávání v nahrávkách není jednoduché najít.	Vysoká	Vyhledávání v textu přesunuto z dialogu filtrování do speciálního políčka nad seznamem nahrávek.
3	Nepříliš jasná funkcionalita souvislosti slov.	Střední	Seznam nejvyužívanějších slov na rozcestníku předělán, jasnější návaznost na graf souvislosti.
4	Tlačítko pro změnu profilových údajů je těžko k nalezení.	Vysoká	Tlačítko přesunuto z vysouvacího menu přímo do základního rozložení.
5	Tlačítko odhlášení je těžko k nalezení.	Vysoká	Tlačítko přesunuto z vysouvacího menu přímo do základního rozložení.
6	Nízká přehlednost seznamu slov v nahrávce.	Vysoká	Drobné úpravy seznamu slov, hlavní rozlišení volajícího a volaného pouze barvou a zobrazení mezer v případě, že i v hovoru je mezi slovy delší mezera.
7	Uživateli není jasné, jak má použít pokročilé možnosti vyhledávání.	Vysoká	K políčku vyhledávání přidána textová nápověda, která popisuje pokročilé možnosti.

Kapitola 4

Realizace

4.1 Vytvoření modelů pro Kaldi

Jak bylo určeno v analýze, Kaldi je framework, který umožňuje vybudování vlastního systému pro přepis řeči na text. Protože se tedy jedná pouze o kolekci nástrojů, je potřeba nejdříve vytvořit a natrénovat statistické modely popsané v části o běžně používaných algoritmů pro přepis.

Ovšem pro získaní těchto modelů je potřeba mít k dispozici takzvaný korpus, což je kolekce zvukových nahrávek a jejich přepisů. Pro účely testování funkčnosti jsem použil veřejně přístupný korpus Vystadial CZ[32], který obsahuje přibližně 74 hodin nahrávek s 26 374 unikátními slovy ve 110 252 větách. Takový korpus je vhodný pro testování různých algoritmů a jejich efektivity, pro reálné použití už použitelný není, protože i na testovacích datech, které jsou dodávané s korpusem dosahuje pouze 60% WER.

Kaldi dodává několik ukázkových užití, takzvaných receptů na trénování modelů. V našem případě tedy stačilo převzít již existující recept a pouze ho drobně upravit pro mé použití.

Účelem těchto receptů je orchestrace nástrojů dostupných v Kaldi, postup konečného receptu, který přímo vychází z receptu vytvořeného přímo pro mnou využívaný korpus. Postup je následující:

1. Extrakce dat z korpusu, předpokládá se, že korpus je dodán jako seznam nahrávek a jejich přepisů. Aby bylo Kaldi schopné tato data zpracovat, je potřeba vytvořit několik pomocných souborů, seznam nahrávek, seznam nahrávek s přiřazeným přepisem a podobně. V případě, že nemáme korpus předem rozdělený na trénovací a testovací data, je ještě potřeba data rozdělit, aby bylo možné testovat výsledky.

2. Převod nahrávek na MFCCs.

3. První krok tvorby modelu, na základě korpusu se vytvoří model pro monophone. Tento model se musí vytvořit i v případě, že plánujeme používat triphone, protože z tohoto modelu bude vycházet.

4. Dalším krokem je tvorba triphone modelu. Ten používá mimo jiné i předchozí model, proto bylo potřeba ho vytvořit.

5. Nyní natrénujeme výsledný model pro feature-space transformations. Tyto transformace značně zvyšují efektivnost modelu a navíc i zrychlují jejich použití, protože se optimalizují interní struktury modelů.

V průběhu receptu se také automaticky vytváří HCLG grafy, které jsou používané pro přepis a jedná se důležitý výstup celého receptu.

Součástí receptu je skript, který výsledný model zabalí do balíku, který je možné použít při instalaci přepisovače na server.

V praxi se mi pomocí tohoto receptu podařilo vytvořit akustický model, který v průměru dosahuje 56 % WER, na grafu 4.1 je pak vidět, jak se WER měnila v průběhu učení. Na konci není vidět velká změna, to je dáno tím, že se model spíše optimalizuje, aby se zrychlil samotný úkon dekódování.

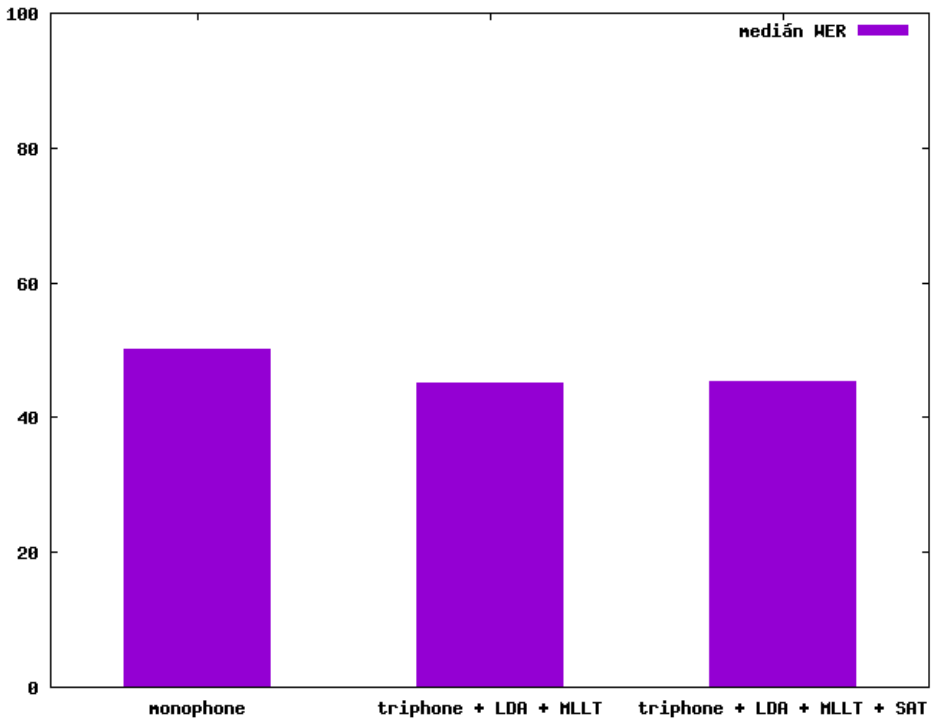
Takový model není pro naše užití ideální, může ale sloužit alespoň částečně. Pokud by se do receptu podařilo dodat větší korpus, výsledky by mohli být lepší. Ostatně, obecně se dá předpokládat, že pro dosažení aspoň 30 % WER běžné řeči, je potřeba mít aspoň 120 hodin nahrávek[33].

Ovšem i s menším objemem data lze docílit hlavní funkcionality, vyhledávání klíčových slov, protože pokud se omezíme na soubor klíčových slov, stačí k trénování menší objem dat a můžeme dosáhnout, pro slova která potřebujeme, lepších výsledků. Na zkoušku jsem do trénovacích dat přidal několik důležitých frází a následně otestoval, jestli je schopen takhle úzce vymezené slova najít. V tom případě už byla klíčová slova správně rozpoznána v 70 % nahrávek.

Nejnovější součástí Kaldi je i integrace neuronových sítí, které můžou teoreticky dosahovat lepších výsledků, ovšem ty mají nevýhodu v tom, že pro dosažení použitelné efektivity je potřeba mnohem větší objem trénovacích dat, které se v tomto případě nedostávají. Přesto je použití neuronových sítí součástí receptu a je možné je použít, pokud má uživatel dostatečně velký korpus.

4.2 Implementace přepisovače

Jak bylo rozhodnuto v analýze, přepisovač jsem implementoval v jazyce Java za použití Spring frameworku. Pro správnou funkčnost modulu bylo potřeba použít plánovací systém, který se stará spouštění a údržbu fronty požadavků na přepis. Pro tento účel jsem použil existující knihovnu Quartz, která kromě plánování úloh umožňuje i jejich persistenci po restartování aplikace a iteraci čekajících úloh, která lze jednoduše použít pro informaci o vytížení přepisovače.



Obrázek 4.1: WER jednotlivých kroků učení

Podmodul přepisovače jsem implementoval jako součást této aplikace s tím, že příprava nahrávky před přepisem je vždy stejná, tedy je rozdělena na volajícího a volaného a drobně normalizována. Po této přípravě je každá nahrávka předána službě, která je definována v konfiguraci, čímž se zajišťuje možná podpora vícero implementací přepisovače.

Před předáním nahrávky přepisovači se provedou základní úpravy nahrávky, nehledě na cílovou službu přepisu. Nejdřív se nahrávka převede na 16 kHz, protože se jedná o vzorkovací frekvenci používanou modelem Kaldi. Poté se nahrávka rozdělí na dvě, volajícího a volaného. Po tomto rozdělení jsou obě části předány do filtrů. Momentálně přepisovač podporuje pouze low-pass filtr a jednoduchý noise reduction filtr. Podporované filtry je možné nastavit nebo úplně vypnout v konfiguraci. Poté je každá část nahrávky předána přepisovací službě.

V současné podobě přepisovač podporuje přepis pomocí Google Cloud Speech-to-Text, nebo pomocí Kaldi GMM modelu, natrénovaného pomocí dodaného receptu.

Integrace Google Cloud Speech-to-Text byla provedena pomocí dostupného SDK pro jazyk Java. Nahrávka musí být nejdřív nahrána na Google cloud storage. Odkaz na tuto nahrávku se pak předá jako požadavek k přepisu. Po dokončení přepisu je pak nahrávka z cloud storage smazána. Nahrávání na Google cloud storage je jediná možnost jak používat Google cloud Speech-to-Text pro soubory delší než 15 sekund.

Integrace s Kaldi funguje na základě receptu pro orchestraci kaldi nástrojů. Tento recept je, stejně jako učící recept, založený na jazyce bash. Na vstupu má soubor s nahrávkou a na výstupu přepis s časovými značkami u jednotlivých slov. Aplikace vyvolá tento recept a zpracuje jeho výstup. V případě úprav v postupu přepisu se bude měnit pouze tento recept a aplikace jako taková bude nedotčena.

Po přepisu je ještě nahrávka předána se všemi podklady, včetně přepisu třídě, která se stará o výpočet kvalitativních parametrů.

4.3 Implementace backendu

Stejně jako implementace přepisovače, backend byl implementován v jazyce Java za pomoci Spring frameworku. Pro napojení na databázi jsem použil Spring Data JPA a Spring REST, který umožňuje jednoduše zpřístupnit data přes REST protokol s minimální konfigurací a nutností implementace vlastní logiky. Pro databázi jsem zvolil MySQL, ovšem změna druhu databáze je pouze otázkou drobné úpravy konfigurace, změně JDBC ovladače a JDBC URI.

Autorizace využívá Spring Security v kombinaci s OAuth2 pluginem, který pomocí Spring JPA repository načítá informace o uživatelích a jejich sezení ukládá do paměti, což znamená že při restartu serveru se musí každý uživatel znovu přihlásit. To by neměl být problém, protože k restartu by nemělo docházet příliš často.

Integrace s ústřednou je implementována jako přímé napojení na databázi opensource ústředny Asterisk, který je momentálně jediný podporovaný systém ústředny. S přidáním podpory pro ostatní řešení ústředny se do budoucna počítá, takže je systém navržen tak, aby pro podporu další ústředny stačilo vyměnit pouze jednu třídu služby.

Pro implementaci pokročilého filtrování jsem přidal knihovnu `QueryDSL`, která v kombinaci s pomocným kódem poskytuje REST rozhraní možnosti filtrování výsledků i pomocí pokročilejších výrazů, jako je rozmezí času.

Apache Lucene je přímo integrováno jako služba. Pro ukládání dat používá svojí vlastní souborovou databázi, jejíž umístění je definované v konfiguraci modulu. Pro každou nahrávku se do této databáze ukládá text a ID nahrávky, při vyhledávání se tedy najde pouze ID nahrávky a všechny detaily se načtou z hlavní databáze. Vyhledávání navíc podporuje i stránkování a základní jazyk pro vyhledávání, který umožňuje specifikovat i složitější požadavky, jako je vzdálenost mezi jednotlivými slovy, nebo podobnost slov.

Pokročilé vyhledávání na základě obsahu nahrávek tedy zajišťuje Lucene, protože takové vyhledávání nepodporuje stejné parametry jako filtrování pomocí metadat, protože používá jinou databázi, je implementováno jako zvláštní REST endpoint `RecordingsController`, který se stará pouze o tento způsob vyhledávání.

Výsledná architektura implementace se liší od původního návrhu, ovšem pouze v detailech řešení, které jsou popsané výše. Výsledná implementace obsahuje desítky tříd, na diagramu 4.2 je možné vidět ty nejdůležitější. Vynechal jsem vlastní implementaci komunikačních tříd některých repozitářů, protože automatická implementace byla dostačující. Navíc jsem u služeb Asterisk a Lucene extrakcí funkcionality do `inteface` a přesunutím implementace do zvláštní třídy umožnil možné budoucí rozšíření o podporu dalších ústředen a vyhledávacích knihoven.

Automatizované označování hovorů se provádí po přijetí přepisu a používají se dva podklady. Prvním je textový přepis, každá uživatelsky definovaná značka umožňuje nastavit přiřazení na základě seznamu slov. Pokud se bude v přepisu alespoň jedno slovo ze seznamu vyskytovat, bude nahrávka označena.

Druhý podklad jsou kvalitativní parametry, k těm jsou předem v databázi definovány speciální značky, které nejsou dostupné uživatelům. Tyto značky jsou přiřazeny v případě, že některý z parametrů překročí předem definovaný interval.

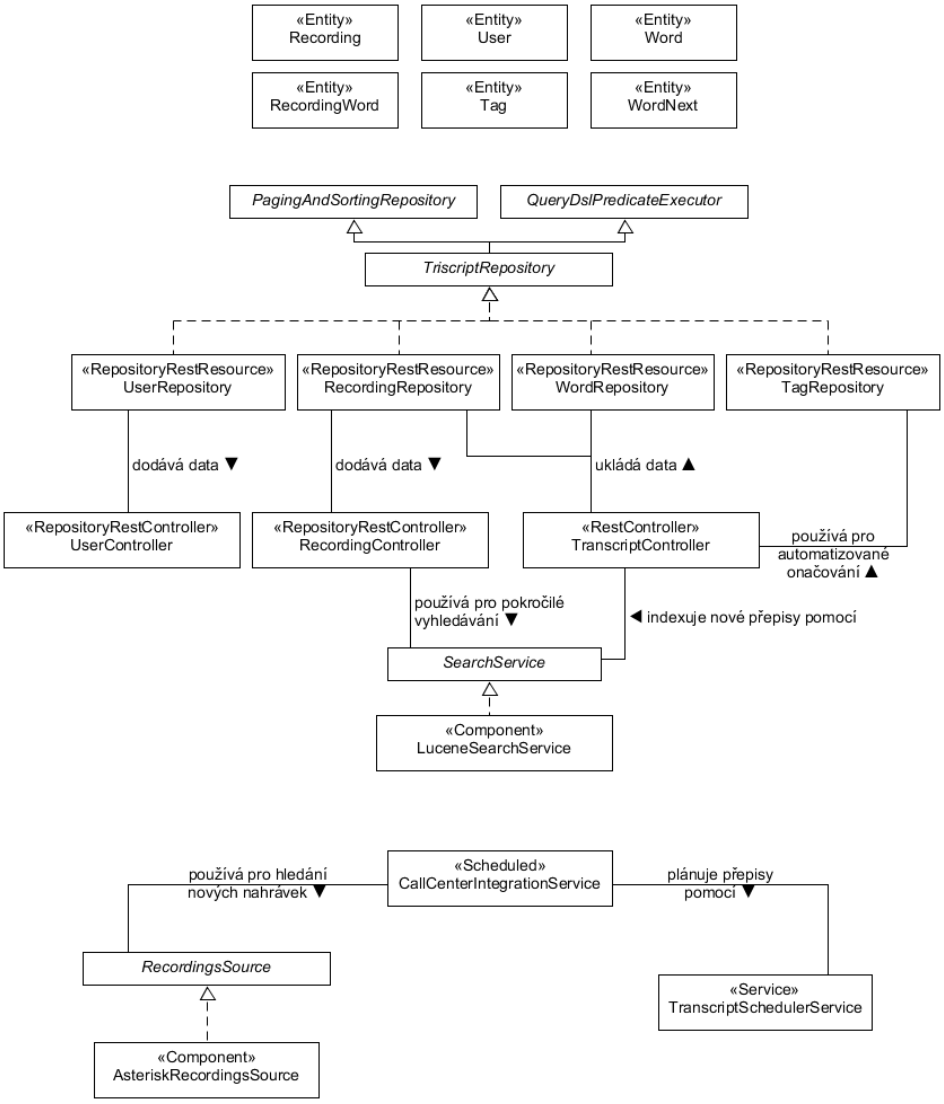
4.4 Implementace frontendu

Frontend jsem implementoval ve dvou fázích, začal jsem prvotním hi-fi návrhem, který sloužil pro uživatelské testování návrhu rozhraní.

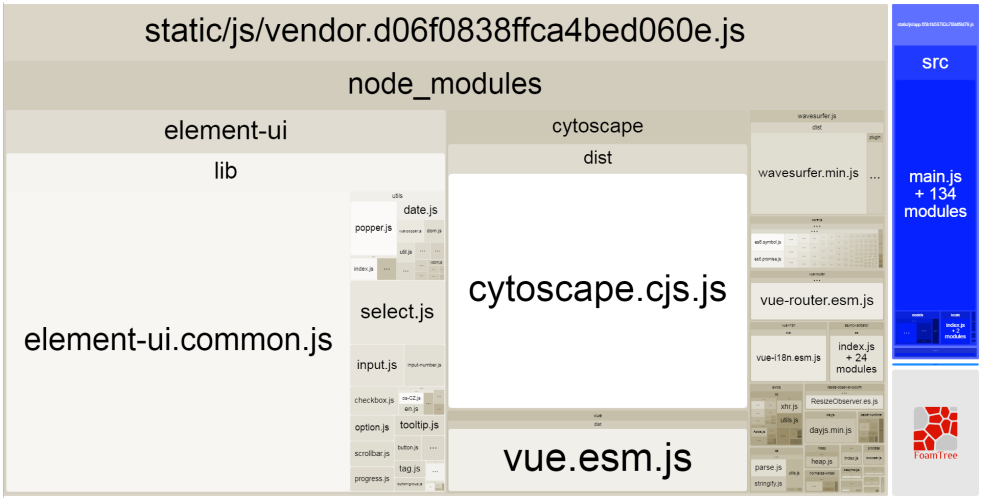
Pro tento prototyp jsem již vytvořil všechnu funkcionalitu potřebnou pro funkci aplikace a pouze u částí, které souvisejí s rozložením rozhraní, jsem použil dočasnou implementaci koncipovanou tak, aby zabrala co nejméně času a bylo jí možné případně zahodit nebo změnit.

V druhé fázi jsem pak už implementoval změny, které vycházeli z testování a veškerý dočasný kód jsem nahradil skutečnou implementací a komunikací s backendem.

Hlavním problémem při implementaci se ukázal být mnou zvolený UI kit. Ten příliš neodpovídal požadavkům služby a v některých případech dokonce bránil správné implementaci. Další problémy způsoboval, když jsem chtěl rozhraní alespoň částečně optimalizovat pro mobilní zařízení, většina prvků s mobilními zařízeními totiž nepočítá a pro správnou funkčnost bych musel provést větší množství úprav, proto jsem se rozhodl na mobily zatím nesoustředit. Také obsahoval velice omezenou selekci ikon, proto jsem jej musel doplnit ještě dalším balíčkem ikon, který negativně ovlivnil velikost výsledné aplikace.



Obrázek 4.2: Architektura implementace modulu backendu



Obrázek 4.3: Výstup nástroje `webpack-bundle-analyzer` po optimalizaci velikosti výstupu

Při implementaci modelů a jejich interakce jsem respektoval původní návrh, nejdříve jsem implementoval model, který uměl pracovat s REST endpointy, pak jsem ho rozšířil o podporu OAuth2 autentifikace a na závěr jsem udělal vlastní obal těchto tříd, který doplnil komunikaci o prvky specifické pro backend modul této služby. Pro komunikaci jsem použil balík `axios`, i když stejnou funkčnost už dnes nahrazuje nativní funkce prohlížeče `window.fetch`, tato služba má ovšem podporovat i starší prohlížeče, takže by stejně bylo potřeba doplnit jí polyfillem, proto jsem radši zvolil univerzálnější knihovnu.

Na konci implementace jsem ještě věnoval práci zmenšení velikosti výsledné aplikace. Pomocí nástroje `webpack-bundle-analyzer`, který umožňuje zjistit, kolik místa ve výsledné aplikace zabírají jednotlivé knihovny, jsem si vytipoval jaké by bylo lepší nahradit. Například knihovna `moment` zajišťovala pouze formátování dat, ovšem z celkové velikosti 1.7 MiB zabírala 226 kiB, tedy přibližně 13 %. Nahrazením této knihovny dalšími optimalizacemi se mi podařilo snížit velikost zkompilované aplikace z 1.7 MiB na 1.2 MiB. Výsledné velikosti pak dominoval hlavně UI kit s 570 kiB (47 %) a knihovna zajišťující vykreslení grafu slov 305 kiB (25 %), tedy důležité nenahraditelné součásti aplikace. Výstupu nástroje `webpack-bundle-analyzer` pro optimalizovanou aplikaci je možné vidět na obrázku 4.3.

Závěr

Cílem této práce bylo navrhnout a implementovat službu pro přepis nahrávek na text v rámci systému telefonní ústředny a tyto přepisy pak uživateli v ucelené formě zobrazit. Věřím, že tento základní cíl se mi podařilo splnit.

Výsledná služba je skutečně schopná splnit všechny úkony, které jsem si na začátku stanovil, navíc jsem věnoval práci tomu, aby byla služba uživatelsky přívětivá. Výsledkem je instalační balík služby, který je možné v kombinaci s přiloženým manuálem správce služby použít k instalaci služby a jejímu použití v praxi.

Hlavním problémem byla očekávaně implementace převodu nahrávek na text, která se i za použití existujících nástrojů Kaldi ukázala být značně časově náročná a na závěr se mi podařilo vytvořit pouze částečně úspěšný výsledek. Hlavním přínosem této části práce je recept na naučení a použití modelů pro přepis v rámci služby Kaldi, který je funkční podle požadavků a pokud by se dodalo dostatečné množství trénovacích dat, věřím že by se výstup dal použít i v reálném provozu.

Tento nedostatek se mi ale podařilo vyvážit možností využití služby Google Cloud Speech-to-Text, která dosahuje velice použitelných výsledků přepisu za cenu zpoplatnění celého procesu a nutnosti předávat nahrávky třetí straně.

Dalším kladným přínosem práce vidím návrh samotné architektury služby, která spojuje několik oddělených modulů a v případě frontendu i jazyků implementace. Ve výsledku nebylo nutné při implementaci příliš architekturu upravovat a služba bez problémů funguje, i když jsou některé moduly na různých serverech.

Služba byla na konci vývoje nasazena v reálném prostředí telefonní ústředny, kde byla využita pro analyzování 288 nahrávek, z nichž bylo celkem 25 označených jako potencionálně závadných. U 15 nahrávek se jednalo o označení na základě potencionálního zvýšení hlasu, ale po přezkoumání se jednalo o výsledek vnějšího rušení hovoru, například projíždějících aut. U 2 nahrávek se jednalo o zvýšení hlasu na straně volajícího. Zbylých 8 nahrávek bylo správně označeno, že obsahují překrývající se slova, ale u těch se jednalo pouze o drobné problémy v komunikaci, nebo překrytí loučení na konci hovoru.

Z výsledků nasazení tedy vyplývá, že služba skutečně plní svůj hlavní účel, označování potencionálně závadných nahrávek. Kromě toho po označení umožňuje uživateli vyhodnotit, jestli je nahrávka skutečně závadná.

Možný budoucí vývoj této služby vidím v rozšíření podpory o další jazyky a tím pokrytí většího trhu, implementaci integrace s více druhy telefonních ústředen a vylepšení algoritmu pro přepis, například přidáním podpory neuronových sítí.

Literatura

[1] Stuttle, M. N.: A Gaussian Mixture Model Spectral Representation for Speech Recognition. 2003.

[2] Mohri, M.; Pereira, F.; Riley, M.: *Speech Recognition with Weighted Finite-State Transducers*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, ISBN 978-3-540-49127-9, s. 559–584, doi:10.1007/978-3-540-49127-9_28. Dostupné z: https://doi.org/10.1007/978-3-540-49127-9_28

[3] Forman, M.: Rozpoznávání řeči s malým slovníkem s nástroji KALDI. 2016.

[4] SpeechTech: SpeechTech Analytics. [cit. 2018-06-20]. Dostupné z: http://www.speechtech.cz/speechtech-analytics/

[5] Google: *Cloud Speech-to-Text*. [cit. 2018-06-20]. Dostupné z: https://cloud.google.com/speech-to-text/docs/

[6] Google: *Language Support*. [cit. 2018-06-20]. Dostupné z: https://cloud.google.com/speech-to-text/docs/languages

[7] Microsoft: *Speech to Text API*. [cit. 2018-06-20]. Dostupné z: https://azure.microsoft.com/en-us/services/cognitive-services/speech-to-text/

[8] Microsoft: *Language and region support for Speech Service API*. [cit. 2018-06-20]. Dostupné z: https://docs.microsoft.com/en-us/azure/cognitive-services/speech-service/language-support

[9] Anguera, X.; Bozonnet, S.; Evans, N.; aj.: Speaker diarization : A review of recent research. *"IEEE Transactions On Audio, Speech, and Language Processing"(TASLP), special issue on "New Frontiers in Rich Transcription", February 2012, Volume 20, N2, ISSN: 1558-7916*, 05 2011, doi:http://dx.doi.org/10.1109/TASL.2011.2125954. Dostupné z: http://www.eurecom.fr/publication/3152

[10] CMU Shpinx: *Basic concepts of speech recognition*. [cit. 2018-11-12]. Dostupné z: https://cmusphinx.github.io/wiki/tutorialconcepts

[11] Yusnita, M. A.; Paulraj, M. P.; Yaacob, S.; aj.: Phoneme-based or isolated-word modeling speech recognition system? An overview. In *2011 IEEE 7th International Colloquium on Signal Processing and its Applications*, March 2011, s. 304–309, doi:10.1109/CSPA.2011.5759892.

[12] Thangarajan, R.; Natarajan, A. M.; Selvam, M.: Word and Triphone Based Approaches in Continuous Speech Recognition for Tamil Language. *WSEAS Trans. Sig. Proc.*, ročník 4, č. 3, Březen 2008: s. 76–85, ISSN 1790-5052. Dostupné z: http://dl.acm.org/citation.cfm?id=1466846.1466850

[13] Chodroff, E.: Kaldi tutorial. [cit. 2018-11-05]. Dostupné z: https://www.eleanorchodroff.com/tutorial/kaldi/training-overview.html

[14] Mohan, B. J.; N., R. B.: Speech recognition using MFCC and DTW. In *2014 International Conference on Advances in Electrical Engineering (ICAEE)*, Jan 2014, s. 1–4, doi:10.1109/ICAEE.2014.6838564.

[15] Gales, M.; Young, S.: The Application of Hidden Markov Models in Speech Recognition. *Foundations and Trends in Signal Processing*, ročník 1, 01 2007: s. 195–304, doi:10.1561/2000000004.

[16] Kaldi ASR: *Decoding graph construction in Kaldi*. [cit. 2018-06-25]. Dostupné z: http://kaldi-asr.org/doc/graph.html

[17] REST. [cit. 2018-10-22]. Dostupné z: https://www.w3.org/2001/sw/wiki/REST

[18] W3C: *SOAP Specifications*. [cit. 2018-10-22]. Dostupné z: https://www.w3.org/TR/soap/

[19] Osman Mohammed Salih, A.: Audio Noise Reduction Using Low Pass Filters. *OALib*, ročník 04, 01 2017: s. 1–7, doi:10.4236/oalib.1103709.

[20] Sphinx team: *Sphinx documentation*. [cit. 2018-10-22]. Dostupné z: https://www.sphinx-doc.org/en/master/contents.html

[21] The Apache Software Foundation: *Apache Lucene Documentation*. [cit. 2018-10-22]. Dostupné z: http://lucene.apache.org/core/7_6_0/index.html

[22] Elasticsearch B.V.: *Elasticsearch Reference*. [cit. 2018-10-22]. Dostupné z: https://www.elastic.co/guide/en/elasticsearch/reference/current/index.html

[23] *Spring Boot Reference Guide*. [cit. 2018-10-22]. Dostupné z: https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/

[24] Microsoft: *Introduction to ASP.NET Core*. [cit. 2018-10-22]. Dostupné z: https://docs.microsoft.com/cs-cz/aspnet/core/?view=aspnetcore-2.2

[25] Otwell, Aylor: *Laravel Documentation*. [cit. 2018-10-22]. Dostupné z: https://laravel.com/docs/5.7/

[26] WWW Consorcium: *XMLHttpRequest*. [cit. 2018-10-22]. Dostupné z: https://dvcs.w3.org/hg/xhr/raw-file/tip/Overview.html

[27] WWW Consorcium: *HTML5*. [cit. 2018-10-22]. Dostupné z: https://www.w3.org/TR/2014/REC-html5-20141028/embedded-content-0.html#the-audio-element

[28] Facebook Inc.: *React Documentation*. [cit. 2018-10-22]. Dostupné z: https://reactjs.org/docs

[29] Schae, J.: A Real-World Comparison of Front-End Frameworks with Benchmarks. 2018, [cit. 2018-10-12]. Dostupné z: https://medium.freecodecamp.org/a-real-world-comparison-of-front-end-frameworks-with-benchmarks-2018-update-e5760fb4a962L

[30] Krause, S.: JS web frameworks benchmark. [cit. 2018-10-12]. Dostupné z: https://stefankrause.net/js-frameworks-benchmark8/table.html

[31] Google: *Material design guidelines*. [cit. 2018-09-10]. Dostupné z: https://material.io/design/

[32] Plátek, O.; Dušek, O.; Jurčíček, F.: Vystadial 2016 – Czech data. 2016, LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University. Dostupné z: http://hdl.handle.net/11234/1-1740

[33] Moore, R.: A comparison of the data requirements of automatic speech recognition systems and human listeners. 2003.

Příloha A

Slovník

backend je aplikace, která většinou funguje jako nepřímá podpora frontendu.

endpoint v kontextu síťové komunikace označuje jeden z možných komunikačních kanálů.

foném nejmenší jednotka řeči používaná k odlišení výrazů jednotlivých slov.

frontend je aplikace, se kterou uživatel přímo interaguje.

hi-fi prototyp je funkční prototyp uživatelského rozhraní, který se funcionalitou přibližuje finálnímu návrhu.

lexicon v kontextu automatického převodu řeči se jedná o model, převádějící slova na fonémy.

package manager je obecně program, který se při vývoji stará o správu externích knihoven.

polyfill je knihovna, která ručně implementuje určitou funkci prohlížeče v případě, že není funkce v dané verzi prohlížeče dostupná.

triphone je vyjádření fonému v kontextu jeho sousedních fonémů.

utterance v kontextu analýzy mluveného jazyka se jedná o nejmenší jednotku mluvy.

Příloha B

Seznam použitých zkratek

HMM Skryté markovovy modely.

JSON JavaScript Object Notation.

MFCC Mel-frequency cepstrum.

REST Representational State Transfer.

SOAP Simple Object Access Protocol.

SPA Single Page Application.

WER Word Error Rate.

XML Extensible Markup Language.

Příloha C

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `exe` ..... adresář se spustitelnou formou implementace
- `src`
  - `impl` ..... zdrojové kódy implementace
    - `server-transcriber` ..... implementace přepisovače
    - `server-backend` ..... implementace backendu
    - `client-frontend` ..... implementace frontendu
    - `kaldi-egg` ..... recept pro vytvoření kaldi modelů
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF
  - `thesis.ps` ..... text práce ve formátu PS

Příloha D

Správcovská příručka

D.1 Instalace

Instalační balíček služby obsahuje dvě složky, `backend` a `transcriber`. Každá složka obsahuje instalační data jednoho ze dvou modulů

D.1.1 Instalace přepisovače

Přepisovač se stará o přepisování nahrávek na text, pokud plánujete použití vlastního přepisovacího modelu, doporučuje se umístění modulu přepisovače na oddělený server od backendu, protože jeho činnost by mohla narušovat dostupnost dat.

Před instalací je potřeba splnit následující požadavky:

Java 8 Jakákoli implementace JVM podporující Javu 8.

MySQL server Databáze MySQL, používá se k persistenci front požadavků na přepis.

Pro instalaci vybalte na libovolnou lokaci obsah složky `transcriber`. V ní naleznete soubor `install_kaldi.sh` a `install.sh`.

V případě, že chcete používat vlastní Kaldi model, vyberte script `install_kaldi.sh`, který doinstaluje kaldi a další potřebné nástroje. Tento krok může zabrat více času. Na závěr pak bude vytvořena základní konfigurace aplikace do soubor `application.properties`. Ten je potřeba po instalaci upravit tak, aby obsahoval validní uživatelské jméno a heslo pro přístup k MySQL databázi.

Pokud nemáte již připravené modely Kaldi, je potřeba je na základě dat natrénovat, viz kapitola D.2.

Pokud chcete používat Google Cloud Speech-to-Text, vyberte script `install_google.sh`, který pouze vytvoří potřebný soubor `application.properties`, ten je také potřeba po instalaci upravit tak, aby obsahoval validní uživatelské jméno a heslo pro přístup k MySQL databázi. Poté je potřeba nakonfigurovat Google Cloud nástroje, viz sekce D.3.

Po správné instalaci by přepisovač měl být nainstalován jako služba triscript-transcriber, pro spuštění tedy stačí zavolat `service triscript-transcriber start`.

D.1.1.1 Hodnoty konfiguračního souboru

server.port Port, pod kterým server běží.

spring.datasource.username Uživatelské jméno pro úložiště úloh.

spring.datasource.password Heslo pro úložiště úloh.

spring.datasource.url JDBC URI pro úložiště úloh, zde lze změnit název databáze.

spring.quartz.job-store-type Typ úložiště perzistentních úloh, nedpoporučuje se měnit.

app.recordings-storage-path Cesta, kam se dočasně ukládají přijaté nahrávky.

app.transcriber.class-name Název třídy, včetně package, která bude obstarávat přepis. V současnosti je k dispozici `services.GoogleCloudService` a `services.KaldiGmmService`.

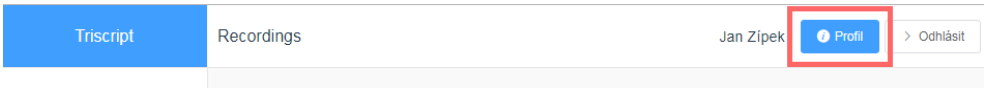
app.transcriber.binary Používá se pouze v případě použití Kaldi, udává cestu kde je umístěn script pro přepis, defaultní cesta by měla odpovídat pokud byl dodržen standardní instalační postup.

app.transcriber.google-bucket Pouze v případě použití Google Cloud, udává název bucketu, do kterého budou nahrávky ukládány před přepisem.

app.transcriber.google-language Poze v případě použití Google Cloud, udává jazyk používaný pro přepis. Možné hodnoty je možné najít v dokumentaci Google Cloud Speech-to-Text.

D.1.2 Instalace backendu

Backend je webový server, který se stará o kolekci dat a zobrazování frontendu uživateli. Pro jeho instalaci stačí vybalit složku `backend`, spustit `install.sh` a upravit application.properties aby odpovídal skutečnosti.



Obrázek D.1: Tlačítko navigace na nastavení profilu

spring.datasource.url Zde je potřeba upravit správně údaje pro přístup k databázi, případně i název databáze.

triscript.asterisk-connection Zde je potřeba upravit správně údaje pro přístup k databázi hovorů asterisku.

triscript.recordings-path Tato cesta by měla odpovídat lokaci nahrávek asterisku.

triscript.transcript-server Seznam dostupných přepisovacích serverů oddělený středníkem.

triscript.callback-host Host používaný jako callback url, tato adresa musí být dostupná z modulů přepisovače.

Po správné instalaci by backend měl být nainstalován jako služba triscript-backend, takže ke spuštění stačí zavolat `service triscript-backend start`.

D.1.3 Konfigurace správcovského účtu

Po nainstalování potřebných služeb je potřeba správně nastavit správcovský účet. Pro to je potřeba se přihlásit do klientského rozhraní, které by mělo být dostupné na serveru backendu. Přihlašovací údaje po instalaci jsou `admin:admin` a výrazně doporučuji si po instalaci je změnit.

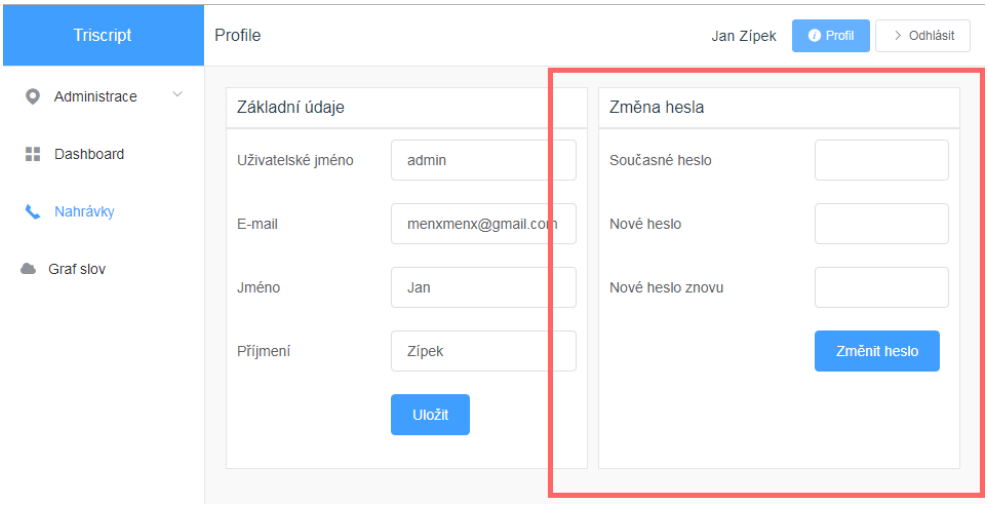
Toho lze docílit přes profilovou stránku, odkaz na ní je v pravém horním rohu, vedle jména uživatele, viz obrázek D.1. V nastavení profilu je poté možné změnit si přes formulář heslo, viz obrázek D.2.

D.2 Vytvoření vlastního modelu Kaldi

Pro natrénování modelu stačí přejí do složky `kaldi/egs/vystadial_cz/s5b` a spustit script `triscript-run.sh`.

Ten používá jako základní data korpus vystadial_cz, pokud chcete používat vlastní dataset, je potřeba nejdříve připravit data.

Ve datech musí být tři složky: `train`, `dev` a `test`. V každé složce by měly být pro každou nahrávku dva soubory: `nahrávka.wav`, obsahující zvukovou nahrávku ve formátu WAV PCM s vzorkovací frekvencí 16kHz a `nahrávka.trn` s textovým přepisem nahrávky.



Obrázek D.2: Formulář změny hesla

Nahrávky by měly být ve složkách rozdělené tak aby naprostá většina byla v `train`, to je soubor nahrávek používaný pro samotné učení. Zbytek ve složkách `dev` a `test` se používá pro testování v průběhu učení, takže by měl obsahovat tak hodinu nahrávek.

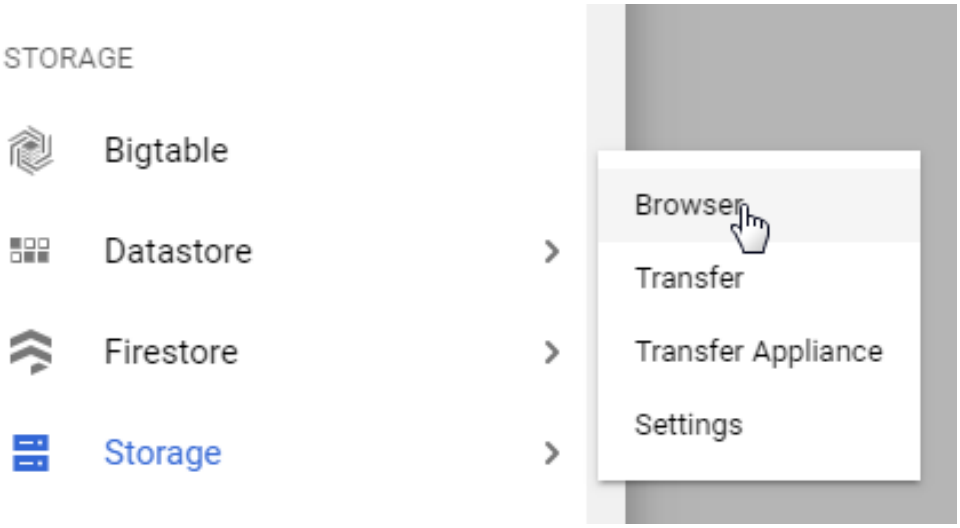
Poté co budete mít připravená data, je potřeba upravit soubor `triscript-run.sh` a změnit dva úvodní parametry, `download_data` na `0`, aby se zbytečně nestahoval úvodní dataset a `data` na cestu k vlastním datům.

D.3 Konfigurace Google Cloud Speech-to-Text

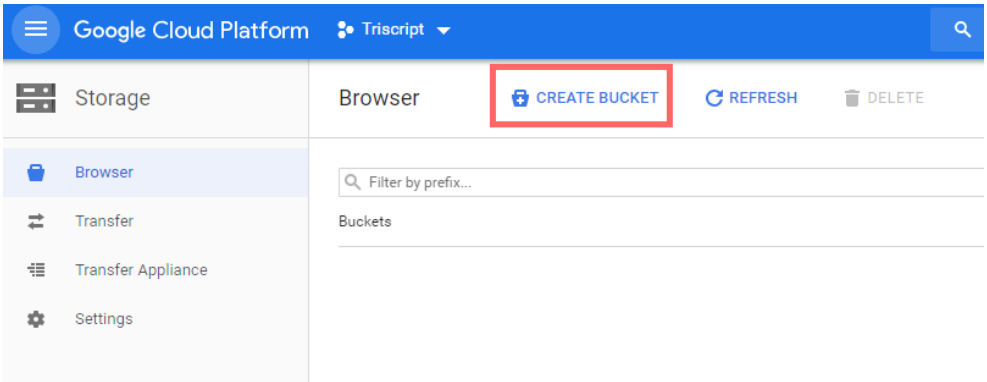
Pro správné používání Google Cloud je potřeba nejdříve nainstaloval google cloud SDK a nakonfigurovat přístupové údaje. K tomu je potřeba návod nalezený v oficiální dokumentaci ke Google Cloud Speech-to-Text.

Pak je potřeba vytvořit Cloud Storage bucket, který se bude používat k ukládání nahrávek. K tomu, podobně jako u předchozího kroku můžete použít Google Cloud Console. V konzoli vyberte v sekci `Storage` položku `Storage`, viz obrázek D.3.

V této sekci je potřeba vytvořit nový bucket, viz obrázek D.4. Název bucketu musí být unikátní a je potřeba si ho zapsat. Po vytvoření pak je potřeba upravit `application.properties` a zadat název vašeho bucketu. Pak bude služba připravena používat Google Cloud Speech-to-Text.



Obrázek D.3: Položka Storage v Google Cloud Console



Obrázek D.4: Vytvoření bucketu v Google Cloud Console