TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta mechatroniky, informatiky

a mezioborových studií

Studijní program: N 2612 – Elektrotechnika a informatika

Studijní obor: 1802T007 – Informační technologie

INDEXACE A PROHLEDÁVÁNÍ MULTIMÉDIÍ

MULTIMEDIA INDEXING AND SEARCH SYSTEM

DIPLOMOVÁ PRÁCE

Autor: **Bc. Karel Blavka**

Vedoucí práce: Ing. Jindřich Žďánský, PhD.

Konzultant: Ing. Petr Červa, Ph.D.

V Liberci 21. 5. 2010

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky a mezioborových studií
Ústav informačních technologií a elektroniky
Akademický rok: 2009/2010

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Bc. Karel BLAVKA**

Studijní program: **N2612 Elektrotechnika a informatika**

Studijní obor: **Informační technologie**

Název tématu: **Indexace a prohledávání multimédií**

Zásady pro vypracování:

1. Seznamte se s možnostmi automatického zpracování multimédií pro účely indexace jejich obsahu.
2. Seznamte se s principy fulltextového vyhledávání v textových dokumentech a tvorbou odpovídajících databází.
3. Navrhněte architekturu systému pro skladování, indexování a prohledávání multimediálních souborů.
4. Navrhněte uživatelské rozhraní.
5. Implementujte a otestujte navržený systém.

Rozsah grafických prací: Dle potřeby dokumentace
Rozsah pracovní zprávy: cca 40 - 50 stran
Forma zpracování diplomové práce: tištěná/elektronická

Seznam odborné literatury:

[1] Acero A., Huang X., Hon H-W.: Spoken Language Processing

[2] Han J., Kamber M.: Data Mining

[3] Loffler J., Biatov K., Eckes Ch., Koulet J.: IFINDER: An MPEG-7-Based Retrieval System For Distributed Multimedia Content

Vedoucí diplomové práce: **Ing. Jindřich Žďánský, Ph.D.**
Ústav informačních technologií a elektroniky
Konzultant diplomové práce: **Ing. Petr Červa, Ph.D.**
Ústav informačních technologií a elektroniky

Datum zadání diplomové práce: **16. října 2009**
Termín odevzdání diplomové práce: **21. května 2010**

prof. Ing. Václav Kopecký, CSc.
děkan

L.S.

prof. Ing. Ondřej Novák, CSc.
vedoucí ústavu

V Liberci dne 16. října 2009

Prohlášení

Byl jsem seznámen s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé diplomové práce pro vnitřní potřebu TUL.

Užiji-li diplomovou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Diplomovou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím diplomové práce a konzultantem.

V Liberci dne 20. května 2010

Karel Blavka

Poděkování

Na tomto místě bych rád poděkoval mým blízkým za jejich podporu, nejen při psaní této práce, ale po celou dobu mého studia.

Také bych chtěl poděkovat mému vedoucímu Ing. Jindřichu Žďánskému, PhD., který mi poskytl četné užitečné poznatky a připomínky při vypracovávání diplomové práce.

Abstrakt

V úvodu této práce se věnuji způsobům zpracování audio složky multimediálních souborů za účelem indexace jejich obsahu. Jsou zde popsány způsoby získávání dat, včetně mého. Dále popis formátu, ve kterém lze získaná data uchovávat.

V následující kapitole se věnuji fulltextovému vyhledávání, které je základem mnou navrženého systému. Je zde porovnání několika vyhledávacích strojů, zvláštní pozornost je věnována fulltextovému vyhledávání pomocí *MySQL* a *Sphinx*.

Následuje kapitola zabývající se návrhem systému pro indexaci a vyhledávání v přepisech multimediálních souborů včetně návrhu uživatelského rozhraní jako webové aplikace. V jednotlivých podkapitolách se věnuji popisu navrženého systému, databázi pro ukládání dat a uživatelskému rozhraní, dále popisuji použité technologie a programovací jazyky.

Na předchozí kapitolu navazuji kapitolou zabývající se implementací celého systému. V první podkapitole popisuji program pro vkládání dat do systému, následuje implementace webové aplikace založené na technologii *AJAX* a podkapitola věnující se konfiguraci *Sphinxu*.

V poslední kapitole se v první části věnuji odzkoušení reálného serveru, na kterém jsem vytvořený systém spustil. V druhé části popisu a seznámení s možnostmi webové aplikace.

Součástí příloh této práce je popis instalace a zprovoznění serveru, a to jak pro *Linux*, na kterém server v současnosti běží, tak na *MS Windows*.

Klíčová slova

multimédia, vyhledávání, fulltext, audio

Abstrakt

In an introduction of this work I describe possibilities of processing of multimedia audio content forms for indexing of their content. Moreover, there are described methods of data acquisition, including mine, next format description where captured data can be saved.

In the next part, full text search is discussed. This is the base of my suggested system. In addition, there is a comparison of several search engines. The main accent is put on full text search using *MySQL* and *Sphinx*.

Then I write about suggestion of system for indexing and searching in transcription of multimedia files, including web application user interface. More specifically, storing database, user interface, then use technologies and programming languages are described.

Then I write about the implementation of the whole system. Firstly, programme for data acquisition, then user interface, which is based on Ajax technology and *Sphinx* configuration.

In the last chapter, firstly, I write about testing of real server that starts up the system. Secondly, I describe an identification of posibilities of web application

Attachment of this work contain a description of installation and starting up of server for Linux and for MS Windows.

Keywords

multimedia, search, fulltext, audio

Obsah

Seznam obrázků

Seznam tabulek

Seznam symbolů a zkratek

GPL ..... všeobecná veřejná licence (General Public License)
HTML ..... HyperText Markup Language
ITE ..... Ústav informačních technologií a elektroniky
XML ..... rozšiřitelný značkovací jazyk
V2T ..... Voice to Text
UTF-8 ..... UCS Transformation Format
UCS ..... Universal Character Set
PZR ..... Počítačové zpracování řeči
PMR ..... Pokročilé metody rozpoznávání řeči
HTK ..... The Hidden Markov Model Toolkit
G2P ..... grammar to phonem
WAV ..... Waveform audio format
MFCC ..... Mel-Frequency Cepstral Coefficients
FFT ..... Fast Fourier transform
IFFT ..... Inverse Fast Fourier transform
DCT ..... Diskrétní kosinovou transformací
HMM ..... Hidden Markov model
ms ..... milisekundy
API ..... Application Programming Interface
GB ..... Giga Byte
PHP ..... Rekurzivní zkratka PHP: Hypertext Preprocessor
SQL ..... Structured Query Language
CSS ..... Cascading Style Sheets
RTMP ..... Real Time Messaging Protocol
AJAX ..... Asynchronous JavaScript and XML
ERD ..... Entity Relationship Diagram

LAMP ..... Linux Apache MySQL PHP | Perl | Python

JSON ..... JavaScript Object Notation

DOM ..... Document Object Model

XHTML ..... EXtensible Hypertext Markup Language

WML ..... Wireless Markup Language

WWW ..... World Wide Web

SGML ..... Standard Generalized Markup Language

W3C ..... World Wide Web Consortium

I/O ..... vstupně/výstupní

URL ..... Uniform Resource Locator

1 Úvod

Důvodem vzniku této práce byla potřeba vyhledávat v přepisech audio stopy multimediálních souborů, a to jak v přepisech vytvořených člověkem tak počítačem.

Tato práce nepřímo navazuje na podobný vyhledávač vytvořený mým vedoucím Ing. Jindřichem Žďánským, PhD. s názvem *Ahmed*. Tento vyhledávač měl však jisté nedostatky, jejich omezení nebo úplné odstranění se staly určujícími požadavky na tuto práci.

Hlavním z požadavků se stala otázka rychlosti vyhledávání, *Ahmed* při zhruba 2000 hodinách nahrávek vyhledával poměrně dlouhou dobu. Dalším z požadavků byla možnost vyhledávat podle mluvčího a přidání další řady možností jak upřesnit výsledky vyhledávání. Posledním z požadavků bylo vytvoření uživatelského rozhraní umožňující vyhledávání a následnou práci s nalezenými výsledky, důraz byl kladen na možnost přehrávání.

Zvláště s ohledem na požadavek rychlého vyhledávání jsem se rozhodl celý systém navrhnout a implementovat od začátku. Výsledkem je vyhledávací systém s názvem *Ahmed II*.

V této práci se dále zabývám i možnostmi jak data pro takovýto systém získávat. Výsledkem je program, který přepisům z archivu rádia *ČR1-Radiožurnál* určuje časové značky jednotlivým slovům nebo celým odstavcům podle zvukových nahrávek.

Cílem práce bylo navrhnout architekturu systému pro skladování, indexování a následné prohledávání obsahu multimediálních souborů, navrhnout uživatelské rozhraní pro snadnou práci při vyhledávání s následnou interpretací nalezených výsledků. Celý systém následně implementovat a otestovat.

2 Zpracování multimédií pro účely indexace jejich obsahu

Multimediální obsah se prozatím skládá maximálně ze dvou složek z videa a audia. V této práci se zabývám zpracováním druhé složky, jmenovitě zvukové neboli audio stopy.

Proč audio?

Lidská řeč, jinak také zvukový projev člověka sloužící ke komunikaci s ostatními, se skládá z promluvy, která v sobě nese konkrétní informaci. Promluva se skládá ze slov, slovo je základní významová jednotka. Dále se dělí na slabiky, což je nejkratší vyslovitelná jednotka řeči. Slabiky lze dále rozdělit na hlásky, nebo také fonémy. Foném je nejmenší rozlišitelná jednotka řeči.

Ovšem předávání informací pouze pomocí řeči bylo nedostačující, a tak se zhruba před 4000 lety př.n.l. objevila písemná podoba řeči. Písma se dělí na logografická, sylabická a alfabetická. Logografická písma jsou ta, kde napsaný znak reprezentuje jedno slovo např. čínština. Písma sylabická (slabičná) zapsaný symbol reprezentuje celou slabiku, Japonština. A nakonec alfabetická písma, s tímto písmem se setkáváme každý den a je jím psána i tato práce. Teoreticky by každá hláska(fonem) měla mít svůj znak. V praxi nemá a vznikají tím jisté potíže. Ani čeština není výjimkou, ale o tom až dále.

Jak jsem se již zmínil, psaná podoba řeči vychází z té mluvené. Slova se skládají z písmen a ty jsou v alfabetickém písmě obdobou fonému. V sylabickém písmě se slova skládají ze slabik. I v alfabetickém písmu lze dělit slova na slabiky, ty jsou pak spojení několika fonémů. Slova tvoří větu, což je obdoba promluvy.

A proč tedy audio, protože pokud v audio stopě někdo mluví, pak toto audio obsahuje informace, o které máme zájem. Chtěli bychom tyto informace dále zpracovávat a uchovat jejich obsah a umožnit jeho zpřístupnění, nejen nám, kteří si jen chceme danou promluvu znova poslechnout, ale třeba i sluchově postiženým, kteří takovou možností nedisponují. Nyní se dostáváme k velkému problému, a to jak mluvenou podobu řeči převést do po-

doby textové. Nejjednodušší je využití lidského intelektu, který je schopný takový překlad provádět doslova "realtime". Ovšem s množstvím, ve kterém se dnes informace nachází, a to převážně v audio podobě, je toto řešení nedostačující. Proto se léta pracuje na možnosti automatického přepisu pomocí počítače, např. *NEWTONDictate* software vyvíjený na katedře *ITE* ve spolupráci s firmou Newton technologies[7].

Ovšem mít pouze přepisy řeči nestačí. Je nutné, aby takto získané informace byl člověk schopný rozumným způsobem využít. Pro lepší orientaci a práci s takovým velkým množstvím dat, je potřeba nabídnout uživateli možnost vyhledávat a vyhledaná data srozumitelným způsobem interpretovat. To je důvodem vzniku této diplomové práce.

2.1 XML

Aby bylo možné data získaná ze zvukové stopy dále počítačově zpracovávat, je nutné uložit je ve tvaru, kterému by počítač rozuměl. Jednou z možností je využití *XML*.

XML schéma, ve kterém je přepis ukládán včetně časových značek jednotlivých slov nebo větších bloků textu (nejčastěji jeden odstavec, v konverzaci promluva jednotlivce). Základní strukturu tohoto *XML* schématu jsem převzal od katedry *ITE*. Z důvodu zachování jisté kompatibility mezi programy vyvíjenými na této katedře a mým systémem.

Do původního převzatého *XML* jsem doplnil atributy *kanal* a *porad* ,jedná se o metadata k dokumentu, které budou sloužit při přesnějším vyhledávání. A atribut *href*, kde by se měla nacházet originální *url* adresa stránky, ze které byla data získána, atribut je nepovinný a používá se výhradně u mnou vytvořeného získávání dat. Protože doplňující informace jsou zaneseny jako atributy, nové schéma zůstává kompatibilní i s již používanými programy, které využívají starší verzi tohoto *XML*. Struktura používaného *XML* schématu je součástí přílohy A.1.

Ukázka části *XML* používaného pro ukládání přepisů:

```
<?xml version="1.0" encoding="utf-8"?>
<Transcription xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <Chapters>
    <Chapter kanal="ČRo␣1␣-␣Radiozurnal" porad="NanoTrans24"
          name="BLOCK␣10.02.2010␣19:45:00">
      <Sections>
        <Section>
          <Paragraphs>
            <Paragraph begin="0" end="183750">
              <Phrases>
                <Phrase begin="70">
                  <Text>Olympiadu</Text>
                </Phrase>
                <Phrase begin="1810">
                  <Text>take</Text>
                </Phrase>
                ...
                <Phrase begin="328610">
                  <Text></Text>
                </Phrase>
              </Phrases>
              <speakerID>1</speakerID>
            </Paragraph>
            <Paragraph begin="329150" end="842420">
              ...
            </Paragraph>
          </Paragraphs>
        </Section>
      </Sections>
    </Chapter>
  </Chapters>
  <SpeakersDatabase>
        <Speakers>
            <Speaker>
                <ID>1</ID>
                <FirstName>Unknown</FirstName>
                <Surname>Speaker</Surname>
                <Sex>unknown</Sex>
                <Comment></Comment>
            </Speaker>
        </Speakers>
    <speakersIndexCounter>1</speakersIndexCounter>
    </SpeakersDatabase>
</Transcription>
```

2.2 Automatický přepis - Newton Dictate

Newton Dictate je systém pro automatický převod řeči na text, tak zvaně *Voice to Text* (*V2T*), je určen pro český jazyk, a je výsledkem dlouholeté spolupráce vědeckovýzkumného týmu SpeechLab pod vedením Prof. Ing. Jana Nouzy, CSc. na Technické univerzitě v Liberci [8] a společností Newton Technologies, a.s. [7].

V současnosti je *Newton Dictate* vybaven slovníkem o více než 500 000 slovech (pokrývá 98% českých slov a slovních tvarů). Dále je k dispozici několik mutací slovníku zaměřených na určitou oblast, např. pro medicínu a právní prostředí (rozsudky, exekuční příkazy, soudní líčení, apod.).

2.3 Určování časových značek pro archiv Radiožurnálu

I když získávání dat pro mnou vytvářený vyhledávač nebylo prioritním cílem, přesto jsem měl pocit, že bych si měl zkusit i tuto část realizovat i já sám. Vzhledem k tomu, že vytvoření vlastního automatického přepisovače je úloha neuvěřitelně komplikovaná, o kterou se pokouší desítky vědeckých týmů, byl jsem nucen poohlédnout se po jiné variantě. Již z dřívějška mi bylo známo, že archiv *ČRo 1 - Radiožurnál* na adrese *http://www.rozhlas.cz/radiozurnal/archiv/* obsahuje u většiny příspěvků přepis, včetně možnosti poslechu či stažení audio souboru s tímto příspěvkem. V debatách jsou dokonce uvedeni i mluvčí.

Pomocí mnou napsaného programu s názvem *downloader-Archiv-Radiozurnal.py* jsem z toho archivu vždy stáhl *html* kód příspěvku a k němu odpovídající audio soubor. Popis viz. kapitola 2.3.1.

Následně jsem z těchto stažených dokumentů a k nim příslušných audio nahrávek pomocí programu *Archiv-Radiozurnal.py* vygeneroval výše popisované *XML* a to včetně časových značek. Verze 2. je schopna určovat časové značky konce a začátku celých odstavců. O něco novější verze 3. již určuje časové značky jednotlivým slovům. Podrobný

popis určování časových značek viz kapitola 2.3.2.

2.3.1 Download archivu

Program *downloader-Archiv-Radiozurnal.py*, má ve své podstatě jednoduchý úkol, který lze rozdělit na tři části:

- Projít audio archiv Českého rozlasu 1 - Radiožurnál.
- Načíst obsah každého dokumentu v tomto archivu, což je *html* dokument, zjistit zda se v tomto *html* kódu nachází odkaz na nějaký stažitelný audio soubor, jehož obsah je relevantní s obsahem dokumentu, tuto skutečnost odhaduje z umístění odkazu v *html* kódu a podle získané adresy audio souboru.
- V případě, že daný dokument takovýto audio soubor obsahuje, program daný dokument uloží na disk a stáhne k němu audio soubor, oba soubory spáruje shodným názvem. Při ukládání dokumentu je změněna kódová sada z *ISO-8859-2*, ve které se archiv nachází na *UTF-8* pro pozdější snadnější zpracování.

2.3.2 Vytvoření XML - určení časových značek

Při vytváření programu *Archiv-Radiozurnal.py* jsem využil znalosti získané ve volitelných předmětech *Počítačové zpracování řeči* (*PZR*) a navazující *Pokročilé metody rozpoznávání řeči* (*PMR*), vedené prof. Ing. Janem Nouzou, CSc., kde jsem se seznámil se způsoby zpracování řeči, a to jak spojité, tak nespojité. Dále jsem se v předmětu PMR seznámil s *HTK*, což je sada nástrojů pro zpracování skrytých Markovských modelů, v originálu *The Hidden Markov Model Toolkit* (*HTK*) [11]. Dále jsme v tomto předmětu s mými kolegy vytvořili několik minut nahrávek spolu s přesnými fonetickými přepisy. Z těchto dat jsme pomocí *HTK* vytvořili jednoduchý fonetický model českého jazyka. Tento model jsem použil při určování časových značek v této práci.

Zpracování dokumentu programem:

Nejdříve program z *html* kódu dokumentu vytáhne informace týkající se obsahu dokumentu, a těmito informacemi jsou: nadpis, datum a čas vysílání, pořad, autor. K získání těchto informací jsou použity regulární výrazy.

Druhý krok: program se pokusí z *html* kódu získat text s přepisem a ten rozdělit na jednotlivé odstavce. Opět jsou využity regulární výrazy a to hned několik. Je to důsledkem toho, že přepisy v tomto archivu jsou i několik let staré a forma, ve které jsou vystaveny na webových stránkách se několikrát mění. Vždy je nutné nejprve určit jaký regulární výraz bude nejlepší použít, a pak jej teprve aplikovat. K určení o jakou verzi překladu jde a jak z ní tedy dostat data, o která máme zájem, využívám několika odchylek v názvu některých tagů, jejich posloupnosti či hodnot atributů.

Třetí krok: pokud se programu podařilo z dokumentu vytáhnout všechna požadovaná data. Je nutné přiřadit časové značky a to buď celým odstavcům (platí pro verzi 2.), nebo jednotlivým slovům (u verze 3). Základní postup získání časových značek je u obou verzí obdobný.

- Vytvoření gramatiky a k ní odpovídající slovník.
- Zpracování gramatiky pomocí *HVite* (součást *HTK*).
- Zpracování výsledků z *HVite*.

Určení časových značek - verze 2.

Z předchozího kroku je text jednotlivých odstavců rozdělen do pole textových řetězců. Ty jsou postupně procházeny a zpracovávány. Zpracování každého odstavce znamená vytvoření kusu gramatiky charakterizující daný odstavec. A přidání v gramatice použitých slov do slovníku.

Protože nevíme, kde odstavec v nahrávce začíná nebo končí a zda je přepis správný, proto jsem k problému označkování začátku a konce odstavce přikročil jako k vyhledá-

vání klíčového slova v nahrávce. Chyby přepisu: nespisovné výrazy v nahrávce jsou při přepisu nahrazeny spisovnými, kus přepisu chybí, v nahrávce se mluví jinou řečí pak začíná simultánní překlad atd. S těmito problémy se musíme vypořádat.

Ovšem určovat začátek a konec z jednoho slova by nebylo příliš přesné. Proto se vezme ze začátku odstavce minimálně 40 znaků, protože se berou celá slova. To samé platí pro konec odstavce. Z toho vyplývá, že pokud je odstavec kratší než součet těchto dvou kusů, musí se vzít celý. Takto získané kusy textu se pomocí knihovny *g2p*, převedou transkripcí psaného textu do fonetické podoby. Více o *g2p* v kapitole 2.3.3.

Neboť je zpracovávána spojitá řeč, můžu si dovolit považovat takto získané části textu za jakási slova a tedy počáteční slovo dostane označení *ws%d*, kde %d je pořadové číslo odstavce. Koncová část *we%d*. V případě, že je použit celý text odstavce, je výsledné slovo pojmenováno *wa%d*. Číslo odstavce je důležité pro pozdější generování *XML*, aby se nalezené časové údaje přiřadily ke správným odstavcům. Stejně tak písmenka uvedená za dvojitým v (w), **s** znamená start, **e** end, **a** all. Slova jsou nyní vložena do slovníku společně s jejich fonetickou transkripcí. Ke gramatice je připojen text *'ws%d any wi%d any we%d any '*. Slovo *wi%d* je pomocné, jedná se o prostředek u dlouhých odstavců. Opět o délce kolem 40 znaků. Slovo *any* je ve slovníku definováno jako maximální možné opakování univerzálního fonému, o který jsem doplnil model získaný v předmětu *PMR*. Jde o model, který by měl charakterizovat jakýkoliv zvuk.

Výsledná **gramatika** a **slovník** pro dokument na adrese:
http://www.rozhlas.cz/radiozurnal/publicistika/_zprava/687986, náhled stránky v příloze na obrázku 7.

Gramatika pro daný dokument vypadá následovně:

```
(SENT-START {any} ws0 {any} wi0 {any} we0 {any} ws1 {any}
wi1 {any} we1 {any} ws2 {any} wi2 {any} we2 {any} ws3 {any}
wi3 {any} we3 {any} ws4 {any} wi4 {any} we4 {any}  SENT-END)
```

Ukázka části **Slovníku** vygenerovaného společně s gramatikou:

```
any [] any
ws0 e k~o~n o~m i ts k~aa v~y e d a v~i x aa z~e y ii ts ii
wi0 ii s~p o~t rz e b u~t a g z~v~a n ii x z~b i t n ii x
s~we0 r o~ts e t u~t o~p o~u~ch k~u~p o~d v~r z~u~y ii y e n
...
ws4 a f i l m z~m i z~e l n a r o~z~dj ii l o~d f i l m
o~v~wi4 d o~v~y e p o~d v~r dj i l o~n e k~r i tj i ts k~ee sh
we4 d z~aa b a v~i k~t e r aa y e r e a l i s~tj i ch tj e y
SENT-START []  si
SENT-END []    si
```

Z takto připravené gramatiky pomocí programu *HParse*, ze sady nástrojů *HTK*, vygenerujeme soubor popisující danou gramatiku grafem (sítí). Tedy seznam uzlů a spojnic.

Dále převedeme audio soubor do formátu *wav*, k této konverzi jsem využil program *ffmpeg*. A ten následně zpracujeme programem *HCopy* jehož výstupem bude soubor s kepstrálními příznaky, přesněji *Mel-Frequency Cepstral Coefficients* (*MFCC*). Výpočet *mfcc* je složen z rychlé Fourierovy transformace (*FFT*), rozdělení spektrálního výkonu do pásem pomocí Melovy stupnice, logaritmus výkonů v každém pásmu a inverzní Fourierovy transformace *IFFT* v praxi často nahrazena *Diskrétní kosinovou transformací*(DCT). Tím, že dochází k zlogaritmování frekvenčního spektra získaného pomocí *FFT*, se výsledek přiblíží lidskému vnímání frekvencí.

Nyní za pomoci programu *HVite* z *HTK*, kterému se předá soubor s grafem gramatiky, slovník, model řeči, soubor s *mfcc* a seznam fonémů dojde k určení časových značek jednotlivých částí gramatiky, slov. Jinak řečeno *HVite* vrátí soubor s časy, které reprezentují doby kdy došlo k přechodu mezi jednotlivými uzly sítě. Více o rozpoznávání pomocí *Hidden Markov model* (*HMM*) a jeho realizaci v *HTK* se lze dočíst v *htkbook*, ke stažení na stránkách *HTK* [11].

Ukázka výstupního souboru z **VHVite** po provedení rozpoznávání:

```
\#!MLF!\#\\
"rozpoznavani/nahravka.rec"\\
272600000 346400000 ws0 -57405.679688\\
2289900000 2318600000 wi0 -23494.822266\\
2676900000 2722700000 we0 -36974.468750\\
...
9849200000 9919300000 ws4 -56723.855469\\
11355800000 11410200000 wi4 -45769.503906\\
11782600000 11859500000 we4 -64869.503906\\
```

Zpracování výsledků rozpoznání, kdy se načte výstupní soubor z *HVite* a analyzuje se jeho obsah. Pomocí značení jednotlivých slov se jednotlivé časové údaje připojí k jednotlivým odstavcům. Získaný čas je nutné vydělit číslem 10000, abychom dostali hodnotu času v ms.

Kromě určení časových značek je nutné pro pozdější přesnější vyhledávání určit mluvčího. Pokud je obsahem dokumentu rozhovor, je vždy jméno a krátký popis mluvčího součástí každého odstavce. Pro odlišení od textu je v prohlížeči psán kurzívou. Tedy je obalen tagem a pomocí regulárního výrazu se dá z textu snadno vytáhnout. Ve výsledném *XML* je ale na konci každého odstavce uváděno ID mluvčího. Proto je v programu funkce *vratIDmluvciho(autor)*, která si vnitřně udržuje seznam mluvčích v daném dokumentu s jejich ID, v každém dokumentu se začínají ID navyšovat od jedné. Jejím vstupním atributem je celý text získaný regulárním výrazem, a to včetně popisu. V případě, že obsahem dokumentu je monolog, pak je za mluvčího označen autor článku, i tomu je přiděleno ID již zmíněnou funkcí.

Čtvrtý krok: uložení výsledků zpracování do validního *XML* dokumentu, popsaného výše. Element *SpeakersDatabase* je generován za pomoci funkce *getSpeakers()*.

Zpracovaní verzí 3 která určuje čas každému slovu, je obdobné. Jen se vytváří fonetický přepis za všechna slova a tím se zvětšuje gramatika, protože musí obsahovat každé slovo. Slovo kratší než tři znaky se sloučí se slovem následujícím. Důvody jsou dva: delší slova se snadněji rozeznávají a sloučením krátkých slov s dalšími se sníží celkový počet slov gramatiky a velikost slovníku. Jednotlivá slova mají označení tvaru *w%d*, kde %d je pořadí slova v textu. Protože knihovna *g2p* neumí přepisovat číslovky, je jejich fonetický přepis nahrazen slovem *any*, ale zůstává jim slovní označení jako normálně přepsaného slova. Pro pozdější jednodušší spojení slov s časy získanými pomocí *HVite*.

2.3.3 Knihovna g2p

Při psaní knihovny *g2p* neboli „grammar to phonem" jsem vycházel z pravidel pro transkripci popsané v dokumentu *Transkripce psaného českého textu do fonetické podoby* od Dany Nejedlové a Marka Volejníka [16]. Poslední verze této knihovny bere v úvahu i spodobu. Avšak nikterak jsem neřešil výjimky, kdy se jinak píše a jinak čte. Vzhledem k jisté robustnosti samotného rozpoznávání a vzhledem k faktu, že takové slovo bude obaleno s největší pravděpodobností slovy netvořícími výjimku, jsem tento problém zanedbal. Druhý důvod proč jsem tento problém nikterak neřešil, je to, že nebylo cílem mé diplomové práce napsat dokonalý *g2p* transkriptor.

Transkripce je složena z několika kroků.

- Upravení textu, (odstranění vícenásobných mezer, vložení znaků začátku a konce textu, atd.).
- Aplikace první sady pravidel (pravidla týkající se dvojic písmen a jejich okolí).
- Aplikace druhé sady pravidel (pravidla pro jednotlivá písmena a jejich okolí).
- Aplikace pravidel spodoby.
- Převod textu do fonetického znakové sady kompatibilní s *HVite*.

2.4 Třetí strana

Přepisy od třetí strany. V mém případě bylo pro testování použito několik přepisů od firmy Newton Media. Tyto přepisy jsou pořizovány automaticky s využitím již zmíněného softwaru NEWTONDictate kapitola 2.2, avšak je provedena jejich korektura člověkem. Ty samozřejmě neměly požadované *XML* schéma, ale jejich struktura je doznané míry podobná. Proto nebyl problém vytvořit konverzi z jejich *XML* do mnou používaného *XML*.

Samozřejmě je možné využít přepisy z jakéhokoliv jiného zdroje. Ovšem je nutná konverze do již zmiňovaného *XML*.

3 Fulltextové vyhledávání

Stalo se celkem běžným požadavkem, pokud vytváříme aplikaci pracující nad nějakým množstvím dat, aby se mezi nimi dalo nějakým způsobem vyhledávat, vybírat z nich jen pro nás v danou dobu relevantní data. V mnoha aplikacích nestačí vyhledávat podle předem definovaných atributů(parametrů). Stále větší a větší množství dat se nachází v textové formě.

Naive Searching. Some people, when confronted with a problem, think „I know, I'll use regular expressions." Now they have two problems.[1] Jamie Zawinsky[2] Proto vzniklo a začalo se používat fulltextové vyhledávání.

Co to vlastně je fulltextové vyhledávání ? Ve fulltextovém vyhledávání vyhledávač zkoumá všechna slova v každém dostupném dokumentu a pokouší se je porovnat se slovy zadanými uživatelem. Lepší vyhledávač řadí výsledky dle relevancí, třeba počtu výskytu slov. Ještě lepší nám dovolí způsob řazení ovlivnit v dotazu. A ty úplně nejlepší nám dovolí výsledky hledání ovlivnit ještě dalšími parametry, zkráceně přefiltrovat. Například datum vytvoření dokumentu, autor dokumentu atd.

Pro usnadnění a hlavně urychlení práce vyhledávače se pomocí indexeru vytváří ze všech dokumentů, nad kterými chceme vyhledávat index, ve kterém se bude vyhledávat namísto dokumentů. Aby se ještě více optimalizovala rychlost vyhledávání, přesněji následného třídění výsledků, které po vyhledávání obvykle následuje, je možné údaje v indexu přerovnat třeba podle relevantnosti slov.

Index však nepřináší pouze užitek, ale i starosti. Při jakékoliv změně dat je nutné přeindexovat index, což téměř vždy znamená kompletně celý. Pokud by však při změně

[1] Překlad: Jednoduché vyhledávání. Někteří lidé, když se střetnou s tímto problémem, řeknou si: „V pohodě, použiji regulární výraz." A ejhle, najednou mají dva problémy.

[2] Jamie Zawinsky - je počítačový programátor, který významně přispěl k vývoji svobodného softwaru, konkrétně projektů Mozilla a XEmacs. Byl také důležitou součástí vývoje prvních verzí komerčního prohlížeče Netscape Navigator.

dat k reindexaci nedošlo, ztratí se provázanost indexu s daty a tedy vyhledávač nám bude vracet chybné výsledky.

Protože znova indexovat kompletně všechny dokumenty, vždy když se některý dokument vloží, smaže či změní, by bylo v mnoha případech výpočetně náročné a zdlouhavé, začaly se používat tak zvané delta indexy, kdy se vytvoří hlavní index. Pokud se něco změní v datech (dokumentech), tak se změny indexují právě do tohoto delta indexu. Vyhledávač je tedy nucen vyhledávat ne v jednom indexu, ale ve dvou. A výsledky z obou z nich zkomparovat a seřadit. Což ale přináší zpomalení vyhledávání, čím více informací delta index obsahuje, tím doba čekání na výsledek narůstá. Jediné řešení je jednou za čas spojit oba dva indexy, právě přeindexováním veškerých dat a tím vynulovat delta index. Tím celý koloběh začíná nanovo.

3.1 Databáze MySQL

Databázový systém *MySQL* vytvořený švédskou firmou MySQL AB, nyní vlastněný a dále vyvíjený Oracle Corporation. Je jedním z nejpoužívanějších databázových open source. Disponuje poměrně dobrým fulltextovým vyhledáváním. Mezi jeho hlavní výhody patří jednoduchá použitelnost, a tím velká obliba uživatelů, tedy velké množství návodů, rad a diskuzí. Pakliže si člověk nevystačí s dokumentací.[5]

Fulltext, který má v sobě *MySQL* implementovaný, bohatě postačuje pro většinu menších projektů. Ovšem v případě velkého počtu dat je tento fulltext nedostačující viz tabulka 1 a tabulka 2. *MySQL* je především databázový systém navržený primárně pro práci s daty pomocí relací nikoli pro práci s textem.

3.2 Vyhledávač Sphinx

Sphinx je sada nástrojů pro fulltextové vyhledávání. Je distribuován pod licencí *GPL 2*. Byl navržen jako standardní vyhledávací nástroj, poskytující rychlé vyhledávání, s co nejmenším indexem a s co největší relevantností fulltextového vyhledávání, s možností využití jinými aplikacemi. Dále byl navržen tak, a by byl dobře integrovatelný s *SQL* databázemi a skriptovacími jazyky.

V současné době *Sphinx* má v sobě integrované ovladače pro načítání dat z *MySQL* nebo *PostgreSQL* pomocí přímého napojení. Dále je možné indexovat data uložená v XML formátu.

Vyhledávací *API* je nativně portováno pro *PHP*, *Python*, *Perl*, *Ruby*, *Java*. Je k dispozici i zásuvný mód pro *MySQL*.

Pokud jde o jméno, *Sphinx* je zkratka, která vznikla z SQL Phrase Index. [4]

3.3 Porovnání fulltextových vyhledávačů

Vzorek pro tabulku 1 : 1,5 milionu záznamů kolem 1 GB textu. Tabulka 1 vytvořena z hodnot uvedených v prezentaci *Practical Full Text Search in MySQL* [6]

Tabulka 1: Porovnání několika vyhledávačů, vzorek 1 GB textu

	Indexace	**Velikost indexu**	**Rychlost vyhledání**
LIKE expression	—	—	22 sec
MySQL FULLTEXT	15 min	466 MB	50-80 ms
Apache Lucene	6 min 50 sec	1323 MB	120 ms
Sphinx Search	2 min 27 sec	933 MB	12 ms
Invrted index	28 sec	48 MB	250 ms

Vzorek pro tabulku 2 : databáze wikipedie 2.5 M záznamů, což dává 15 GB textu. Tabulka 2 vytvořena z hodnot uvedených v prezentaci *Sphinx full-text search engine* [10]

Tabulka 2: Porovnání MySQL a Sphinx na 15 GB textu

	Vytváření indexu pouze nadpisy	**Vytváření indexu i data**
MySQL	273,7 s	+Inf
Sphinx	75,1 s	1159,3 s
	Srovnání 1000 záznamů podle jednoho atributu	**2000 dotazů**
MySQL	1.87 s	89.56 s
Sphinx	0.87 s	25.66 s

Poznámka: U vytváření kompletního indexu byl *MySQL* po 6 hodinách násilně ukončen. Proto i poslední údaj je zkreslen, ve skutečnosti *MySQL* vyhledával nad neúplným indexem cirka 100 000 záznamů, zatímco *Sphinx* nad kompletním 2 500 000 záznamů. Tedy *MySQL* hledalo v 25 krát méně datech.

4 Návrh architektury systému

Systém jsem navrhoval s ohledem na multiplatformnost. Dále jsem se snažil, aby užitý software byl, buď open-source, nebo tak zvaný svobodný software.

Návrh celého systému lze rozdělit na dvě části.

- Návrh systému pro indexaci a vyhledávání
- Návrh uživatelského rozhraní

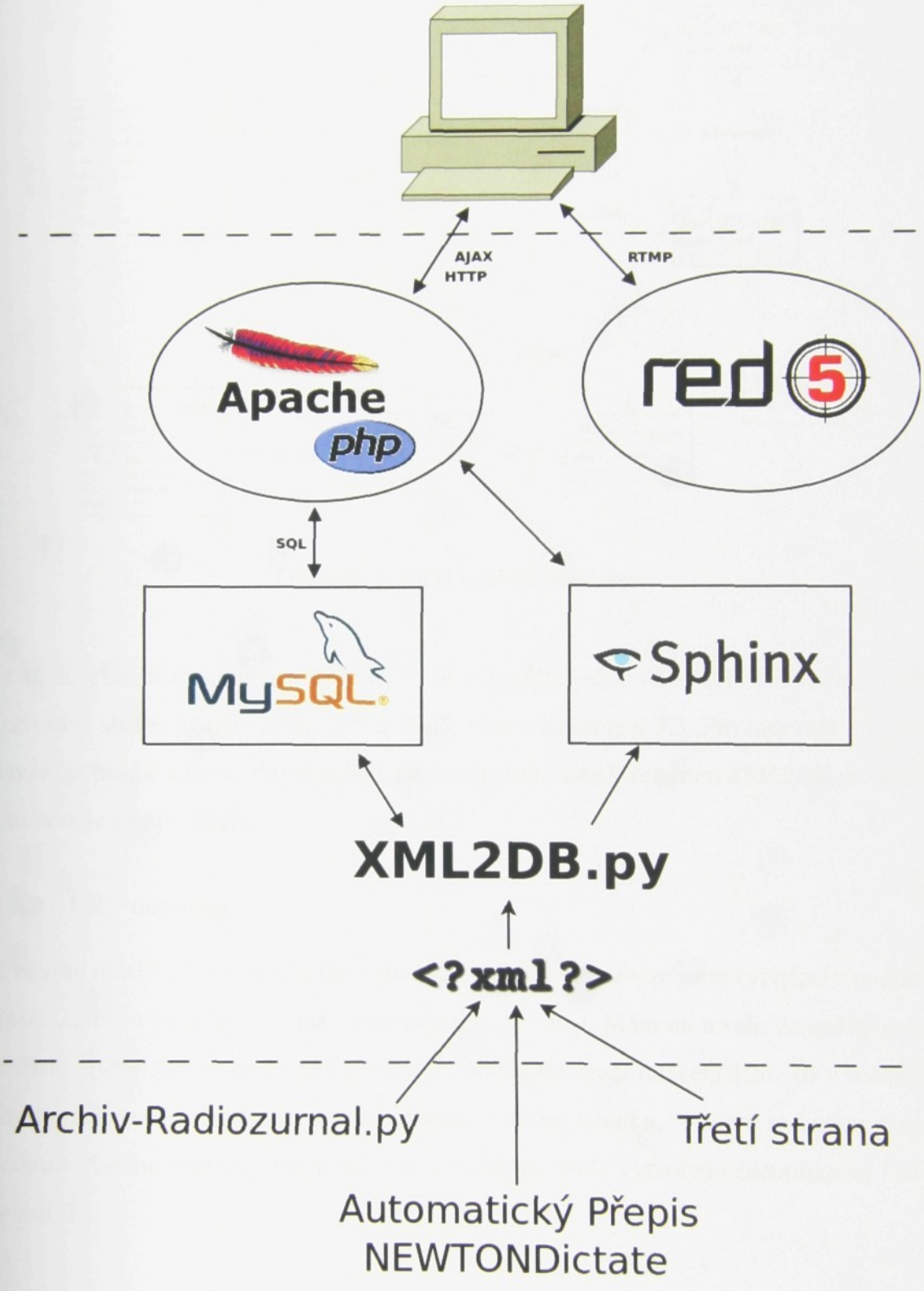
Již zmíněná multiplatformnost ovlivnila i volbu programovacích jazyků, pro pomocné skripty jsem použil jazyk *Python*. Pro webové rozhraní jsem využil skriptovacího jazyka *PHP* jenž generuje obsah stránky v *HTML*, vzhled stránky *CSS* styly a interakci s uživatelem zajišťuje *JavaScript*. Pro komunikaci s databází užívám jazyk *SQL*.

Volba použitých technologií: aby se ušetřil přenos dat mezi uživatelem a serverem, využívám technologii známou jako *AJAX*. Pro streaming audia i videa jsem použil streamingový server *Red5* podporující protokol *RTMP*, který umožňuje *seek* (přehrávat multimediální soubor od libovolného času). Jako úložiště dat slouží databázový systém *MySQL* popsaný v kapitole 3.1. Pro práci s fulltextem jsem použil *Sphinx* popsaný v kapitole 3.2.

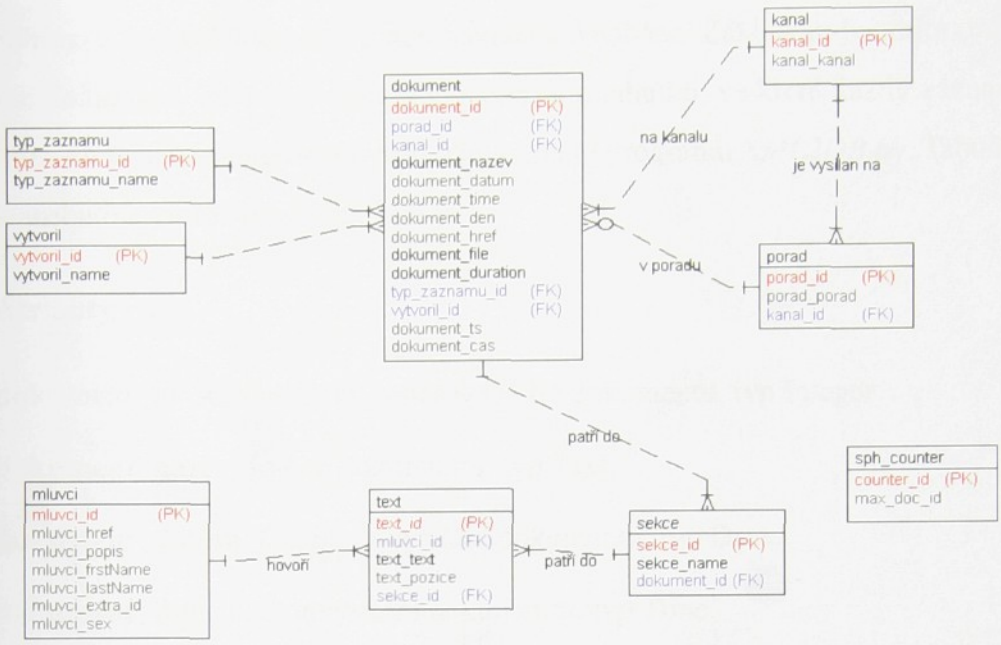
4.1 Návrh systému pro indexaci a vyhledávání

4.1.1 Popis systému

Na obrázku 1 je názorné schéma mnou navrženého systému. Znázorněné jsou zde jednotlivé části systému a jejich vzájemná provázanost a komunikace. Schéma je rozděleno na tři části pomocí čárkované čáry. Vrchní částí je klient využívající systém, popis této části více spadá do návrhu uživatelského rozhraní viz kapitola 4.2. Druhou částí je zde server poskytující službu vyhledávání a zobrazení výsledků. Dále do této části patří i přidávání nových dokumentů do systému. Třetí částí je získávání dat, touto částí jsem se zabýval v kapitole 2. Prostřední část, která se dá označit jako server, se skládá z několika dalších



Obrázek 1: Schéma systému.



Obrázek 2: ERD schéma databáze.

částí. Je užita trojkombinace *Apache*, *PHP* a *MySQL*, více v kapitole 4.3.1. Jako další serverová služba je zde stream server *Red5*, více v kapitole 4.3.3. Pro indexaci a vyhledávání je použit *Sphinx*. Pro vkládání dat do databáze slouží program *XML2DB.py*, jehož vstupem je validní *XML*.

4.1.2 ERD databáze

K návrhu databáze jsem použil *CaseStudio2*. Při návrhu databáze jsem vycházel z podoby *XML*. Zároveň jsem bral v potaz jistá omezení *Sphinxu*. Mám na mysli, že každý další atributy zpřesňující vyhledávání, musí mít podobu boolean, integer, float. To v reálném návrhu znamená vytvořit pro většinu atributů zvláštní tabulku, ve které se budou různé hodnoty daného atributu shromažďovat a každému bude vytvořeno identifikační číslo neboli ID.

Na obrázku 2 je ERD schéma mnou navržené databáze. Základem je entita (tabulka) s názvem ***dokument***, jak název napovídá jedná se o tabulku, ve které každý záznam obsahuje údaje o jednom dokumentu vloženém pomocí programu *XML2DB.py*. Tabulka má několik atributů a vložených klíčů, jsou jimi:

- **Atributy**

  dokument_id Automaticky generované ID dokumentu, typ Integer.

  dokument_nazev Jméno dokumentu, typ Text.

  dokument_datum Datum odvysílání dokumentu, typ Date.

  dokument_time Čas odvysílání dokumentu, typ Time.

  dokument_ts Datum ve formátu Unix timestamp, s časem 00:00:00, typ Indeger.

  dokument_cas Čas přepočítaný na Integer hod*3600+min*60+sec, typ Integer.

  dokument_den Číslo dne v týdnu, kdy byl pořad vysílán Pondělí = 1, .. Neděle = 7, typ Integer.

  dokument_href Určeno pro případný odkaz související s dokumentem, třeba adresy do archivu Radiožurnálu, typ Text.

  dokument_file Atribut s odkazem na multimediání soubor jehož obsah je zároveň obsahem dokumentu, typ Text.

  dokument_duration Délka multimediálního souboru v sekundách, typ Indeger.

- **Vložené klíče**

  kanal_id ID kanálu, propojení s tabulkou kanálů, zdrojů, typ Indeger.

  porad_id ID pořadu, propojení s tabulkou pořadů, typ Indeger.

  typ_zaznamu_id ID typu záznamu, propojení na tabulku typ_zaznamu, typ Indeger.

vytvoril_id Propojení na tabulku ***vytvoril***. ID způsobu vytvoření dokumentu, typ Indeger.

Tabulka ***kanal*** se skládá pouze ze dvou atributů. Slouží pro ukládání názvů kanálů. Pro upřesnění v této tabulce se budou vyskytovat hodnoty jako Radiožurnál, Nova, Prima atd.

- **Atributy**

  kanal_id Automaticky generované ID kanálu, typ Integer.

  kanal_kanal Atribut s jménem kanálu, typ Varchar(250).

Tabulka ***porad*** se skládá pouze ze dvou atributů a jednoho vloženého klíče, který jednotlivé pořady spojuje s kanály. Budou zde uloženy názvy pořadů, pod které daný dokument spadá.

- **Atributy**

  porad_id Automaticky generované ID pořadu, typ Integer.

  porad_porad Atribut s jménem pořadu, typ Varchar(250).

- **Vložené klíče**

  kanal_id Id kanálu, propojení s tabulkou kanálů, zdrojů, typ Integer.

Tabulka ***typ_zaznamu*** obsahuje dva záznamy s ID 1 = Audio a s ID 2 = Video.

- **Atributy**

  typ_zaznamu_id Automaticky vkládané ID pro typ záznamu, typ Integer.

  typ_zaznamu_name Atribut s jménem typu záznamu, typ Varchar(100).

Tabulka ***vytvoril*** obsahuje záznamy kdo vytvořil daný dokument. Jestli šlo o automatický přepis, či o určování časových značek u ručních přepisů, nebo jestli šlo o zdroje z třetích stran.

- **Atributy**

  vytvoril_id Automaticky vkládané ID , typ Integer.

  vytvoril_name Atribut popisující typ vytvoření nebo jeho název, typ Varchar(100).

Tabulka ***sekce*** vyjadřuje kapitolu(sekci) v dokumentu. S dokumentem jsou sekce provázány přes vložený klíč. Každý dokument musí mít alespoň jednu sekci. Protože atribut *sekce_name* může nabývat hodnoty *null*, sekce nemusí mít nadpis.

- **Atributy**

  sekce_id Automaticky generované ID sekce, typ Integer.

  sekce_name Atribut s jménem sekce, typ Varchar(250).

- **Vložené klíče**

  dokument_id ID dokumentu, pod který daná sekce patří, typ Integer.

Tabulka ***text*** zde je zaznamenáván obsah dokumentu, jednotlivé fráze, odstavce, promluvy. Každý záznam má vložený klíč odkazující na mluvčího, tedy kdo ono moudro pronesl.

- **Atributy**

  text_id Automaticky generované ID textu, typ Integer.

  text_text Obsah textu,fráze, typ text.

  text_pozice Určuje pořadí textu v dokumentu, typ Integer.

- **Vložené klíče**

sekce_id ID kanálu, propojení s tabulkou kanálů, zdrojů, typ Integer.

mluvci_id ID mluvčího, který danou frázi vyslovil, typ Integer.

Tabulka ***mluvci*** obsahuje jednotlivé mluvčí, včetně krátkého popisu.

- **Atributy**

  mluvci_id Automaticky generované ID mluvčího, typ Integer.

  mluvci_frstName Jméno, popřípadě i křestní jméno oddělené mezerou, typ Varchar(100).

  mluvci_lastName Příjmení, typ Varchar(100).

  mluvci_popis Popis mluvčího, typ Text.

  mluvci_href odkazy spojené s mluvčím, domovská stránka a jiné, typ Text.

  mluvci_sex Pohlaví, 1 = muž, 2 = žena, 3 = neurčené, typ Smalint.

  mluvci_extra_id Pro případ spojení mluvčího i s jinou databází, typ Varchar(250).

Tabulka ***sph_counter*** je pouze pomocná tabulka pro *Sphinx*, kde si *Sphinx* uchovává hodnotu posledního záznamu v *main* indexu. Je důležitá pro pozdější tvorbu delta indexu.

4.2 Návrh uživatelského rozhraní

Základními požadavky na uživatelské rozhraní jsou jednoduchost, přehlednost, funkčnost, spolehlivost, multiplatformnost. Všechny tyto požadavky se dají zajistit vhodně napsanou webovou aplikací. Webová aplikace lze považovat za jakýsi mezičlánek přes který spolu komunikují uživatel a server. Nadneseně překládá serveru uživatelovy požadavky a obráceně interpretuje uživateli výsledky vrácené serverem na daný požadavek.

Ačkoli webová aplikace přináší mnohé výhody, jsou zde i stinné stránky, některé prohlížeče nedodržují standardy zobrazování nebo neinterpretují *JavaScript* správně. S těmito nuancemi se musí při tvorbě aplikace počítat a případný *JavaScriptový* kód větvit

pro různé prohlížeče. Stejně tak jako u *CSS* kde se využívá takzvaných „Hacků". S těmito problémy je nutné při návrhu počítat a dopředu si určit, pro které prohlížeče budeme podporovat a od jaké jejich verze.

Zpravidla se vybere prohlížeč, prohlížeče řídící se standardy, obvykle Mozilla Firefox, Opera a pro ně se vytvoří aplikace. Poté se testuje v jiných prohlížečích, které části nefungují jak mají a upravuje se kód. Nejproblematičtějším prohlížečem je Microsoft Internet Explorer, poslední verze 8. již většinu standardů dodržuje.

Samotný návrh vzhledu a funkčnosti uživatelského rozhraní prošel několika stádii. První verze vypisovala údaje o dokumentech, ve kterých byl hledaný výraz nalezen. Další verze byla rozšířena o možnost vyhledávat i podle zadaného mluvčího, což byl i jeden ze základních požadavků této práce. Následující verze se soustředily na funkčnost rozhraní a ulehčení práce pro uživatele. Například po kliknutí na text začít přehrávat multimediální soubor spjatý s textem v přehrávači, který je součástí rozhraní. Neboť se v databázi ukládají časové značky pro jednotlivá slova nebo bloky textu lze při přehrávání tyto značky využít. Například přepnutím na správnou pozici při přehrávání, tak k označování právě přehrávaného textu. Dalším podstatně zlepšujícím krokem bylo zvýraznění nalezeného textu ve výpisu a zároveň schování okolního textu, frází. Samotné označování byl značný problém, více v implementaci v kapitole 5.2.3.

Dá se říci, že čtvrtá generace uživatelského rozhraní bylo přidání dalších možností upřesňujících vyhledávání. Jmenovitě:

- Dny v týdnu.
- Datum, od, do ,mezi dvěma datumy.
- Čas, od ,do, mezi dvěma časy, časem mám na mysli 0-24 hod.
- Kanál a k němu související pořad.
- Typ záznamu.

- Podle způsobu vytvoření.
- Pro pokročilé:
  - Mód hledání, jakým způsobem se budou zadaná slova vyhledávat.
  - Rank, způsob napočítávání skóre.

Ovšem možnosti jednotlivých atributů se vybíraly přes rozbalovací nabídku, tag *select*, tedy bylo možno vybrat pouze jednu volbu od každého atributu. Finální pátá generace umožnila kombinovat téměř libovolně dotaz.

Lze tedy sestavit dotaz třeba takového typu: *najdi, zda někdo v sobotu nebo neděli po dvanácté v roce 2010 řekl "svítí sluníčko".*

Zadané slovo, slova se vyhledávají nikoliv v obsahu celého dokumentu, ale v jednotlivých promluvách, neboli v záznamech v tabulce ***text***. Výsledkem hledání je, ale pouze ID dokumentu ve kterém se tato promluva nachází.

Atributem "mód hledání"si lze upravit jakým způsobem bude vyhledávač se zadanými slovy pracovat. Defaultně se vstup bere jako fráze. Při zadání samotného slova na módu hledání nezáleží. K dispozici jsou čtyři módy hledání:

PHRASE vstupní text se vyhledává jako celistvá fráze.

ALL hledaný text musí obsahovat veškerá zadaná slova.

ANY hledaný text obsahuje alespoň jedno ze zadaných slov.

BOOLEAN dovoluje používat operátory zpřesňující výsledky: AND (a), OR (nebo), NOT (negace).

4.3 Použité technologie

4.3.1 LAMP

Kombinaci Linuxu s webovým serverem *Apache*, databázovým systémem *MySQL* a podporou jazyka *PHP*, *Perl* nebo *Python* se nazývá *LAMP* server. Jde o spojení prvních písmen jednotlivých částí serveru.

Apach je softwarový webový server. Je vyvíjen jako open-source. Jeho distribuce je dostupná pro většinu platforem Linux, MS Windows, Mac, Solaris, apod. Je nejpoužívanějším webovým serverem na světě.

MySQL je podrobněji popsán v kapitole 3.1, *PHP* je popsáno v kapitole 4.4.2.

4.3.2 AJAX

Zkratka *AJAX* znamená Asynchronous JavaScript and *XML*, obecně se tak označuje technologie interaktivních webových aplikací. Vyznačují se změnou obsahu stránky bez nutnosti jejího znovu načtení. Vytváří se tak oproti klasickým webovým aplikacím daleko příjemnější prostředí. Tuto technologii ovšem podporují pouze moderní webové prohlížeče.

Podobně jako *LAMP* se *AJAX* skládá z několika technologií.[3]

- XMLHttpRequest technologie pro asynchronní výměnu dat mezi klientem a serverem. Samotná data mohou mít několik podob *XML*, *HTML*, text, *JSON*, *JavaScriptový* kód.
- *JavaScript* a *DOM* pro dynamické změny obsahu stránky.
- *HTML* nebo *XHTML* společně s *CSS* slouží k prezentaci obsahu.

4.3.3 Stream server

Jako streamingový server jsem použil *Red5*. Jedná se o open-source vyvíjený v jazyce Java. Podporuje streaming těchto formátů:

- Video: *FLV, F4V, MP4*
- Audio: *MP3, F4A, M4A*

V mé práci jej používám pro streaming videa ve formátu *flv* a audia ve formátu *mp3*. Pro streaming je používán protokol *RTMP* na defaultním portu 1935.

Real Time Messaging Protocol (RTMP) je proprietální protokol vyvinutý společností Adobe Sysems pro streaming audia, videa a jiných dat přes internet. Mezi flash přehrávačem a serverem.[13]

4.4 Použité jazyky

4.4.1 Python

Python je interpretovaný objektově orientovaný jazyk, který byl vytvořen kolem roku 1990, Guido van Rossum. Od začátku je vyvíjen a jako open source. Instalační balíčky jsou nabízeny pro většinu operačních systémů (Unix, Windows, Mac OS). V Linuxu bývá dokonce *Python* součástí základní instalace. Snad největší předností *Pythonu* je hybridnost. Program lze psát objektově, procedurálně nebo funkcionálně. Popřípadě jednotlivé styly mezi sebou můžeme libovolně míchat. *Python* je vhodný na tvorbu obslužných skriptů, stejně tak jako na vývoj rozsáhlých a náročných programů. Pro *Python* existuje i řada grafických prostředí, která jsou stejně jako *Python* v převážné většině multiplatformní např. *Tkinter*, *Qt4*, *wxPython*. Dalším významným plusem je jednoduché připojování knihoven psaných v jazyce *C* nebo *C++*. *Pythonu* dnes není ani cizí spolupráce s programy psanými v *Jave* nebo *.NET*. [12][1]

4.4.2 PHP

PHP: Hypertextový preprocesor je skriptovací programovací jazyk. Určený pro tvorbu dynamických internetových stránek, lze jej však použít i k tvorbě konzolových a deskopových aplikací. V případě internetových stránek je často kombinován s *XHTML*, *HTML* a *WML*. *PHP* je interpretovaný jazyk, je nezávislý na platformě a obvykle prováděný na straně serveru jehož výstupem je námi požadovaná stránka. Syntaxe jazyka je kombinací *C++*, *Perl*, *Pascal* a *Javy*. Za zmínku patří podpora knihoven, které rozšiřují jeho možnosti. Zvláště pak knihovny potřebné pro spojení s databázovými systémy *MySQL*, *Oracle*, *ODBC*, *PostgreSQL*, *MSSQL*. Nebo knihovny zpracovávající grafiku, jak obrázky, tak jednoduchý flash. *PHP* se stalo velice populárním jazykem, k čemuž přispívá i částečně volná syntaxe.

4.4.3 HTML, XHTML, CSS, JavaScript

Hyper**T**ext **M**arkup **L**anguage je značkovací jazyk pro hypertext, je určen pro tvorbu webových dokumentů a publikací na internetu pomocí systému **W**orld **W**ide **W**eb (WWW). Jazyk se vyvíjel společně s prohlížeči. Jako základ mu posloužil Standard Generalized Markup Language (SGML).

E**X**tensible **H**ypertext **M**arkup **L**anguage, česky rozšiřitelný hypertextový značkovací jazyk je značkovací jazyk určený pro tvorbu hypertextových dokumentů pro prostředí *WWW* navržených *W3C*. Původně mělo jít o pokračování *HTML*, které ukončilo vývoj finální verzí 4.01. Avšak od roku 2007 se pracuje na *HTML* v.5. *XHTML*, které přináší k značkovacímu jazyku *HTML* standardy z *XML*. Tj. povinné ukončování tagů. Dále veškeré atributy malými písmeny. Hodnoty atributů musí být uzávorkovány. Jinými slovy doplňuje klasické *HTML* o větší striktnost, a tím vyšší kompatibilitu a shodnou interpretovatelnost na různých strojích.

Cascading **S**tyle **S**heets, jazyk sloužící k popisu zobrazení dokumentů napsaných pomocí *HTML*, *XHTML*, *XML*. Hlavním smyslem jazyka je oddělit vzhled dokumentu

od jeho struktury, ale hlavně od obsahu.

JavaScript je objektově orientovaný interpretovaný jazyk určený pro *WWW* stránky. Je vykonáván prohlížeči. Jeho základní úlohou je rozhýbat webovou aplikaci dát uživateli pocit interakce.

4.4.4 SQL

Structured **Q**uery **L**anguage je standardizovaný strukturovaný dotazovací jazyk pro práci s daty v relačních databázích. Umožňuje manipulovat s daty, definovat data, řídit přístupová práva, řídit transakce.

5 Implementace systému

Implementace systému se skládala z několika logických kroků. Nejprve bylo nutné vytvořit program, který bude vkládat data do databáze v takové formě jakou potřebuji. Proto jsem napsal program *XML2DB.py*. Dalším krokem bylo vytváření webového rozhraní schopného rozumně a efektivně zobrazovat nalezené výsledky. V době, kdy jsem začínal pracovat na webovém rozhraní, jsem nebyl ještě zcela seznámen s *php api* pro *Sphinx*, proto jsem se rozhodl toto rozhraní vytvořit modulárně. První verzi jsem založil na vyhledávání pomocí fulltextu *MySQL* a ten jsem časem vyměnil za modul podporující *Sphinx*. Sphinx je velice mocný nástroj, co se fulltextového vyhledávání týče. *Sphinx* je sada několika nástrojů. Hlavní jsou:

indexer slouží k indexaci.

search pro vyhledávání pomocí konzoly, slouží k testování, ladění nastavení *Sphinxu*.

searchd služba pro vyhledávání.

Pokud využíváme defaultní systém adresářů pak *searchd* lze spustit pouze jeho zavoláním, bez jakýkoli atributů.

Sphinx je skvělý fulltextový vyhledávač, ovšem všechny jeho schopnosti stojí a padají na jeho správné konfiguraci. Konfigurace je popsána v kapitole 5.3.

5.1 XML2DB.py

XML2DB.py je, jak napovídá koncovka, psán v jazyce *Python*. Program je psán jako konzolová aplikace.

Program tvoří několik funkcí.

SpracovaniXML(file, vytvoril_id, dirMedFile="", priponaMedFile="flv", echo=1)

Toto je hlavní funkce, která zpracovává *XML* soubor. Má dva povinné atributy

a tři nepovinné. V atributu *file* očekává cestu k *XML* souboru. Atribut *vytvoril_id* očekává identifikační číslo odpovídající jednomu ze záznamů v tabulce ***vytvoril***. Jde o ID zdroje dat. Nepovinný atribut *dirMedFile* se použije v případě, že se multimediální soubory nachází v jiné složce než *XML*. Atribut *priponaMedFile* lze využít pokud daný *XML* dokument nemá ve svém názvu příponu multimediálního souboru. Atribut *echo* určuje kolik informací bude daná funkce vypisovat na výstup. Funkce ke svému chodu vyžaduje nainstalovaný **ffmpeg** pomocí něhož zjišťuje délku multimediálního souboru.

getMluvciID(FirstName,LastName,Sex,popis,datum) Funkce komunikuje s databází a snaží se vrátit ID mluvčího z tabulky ***mluvci***. Pokud daného mluvčího nenalezne, vloží jej jako nový záznam. Funkce se snaží, pokud je to možné nebo nutné změnit či doplnit text popisu mluvčího.

projedAdresar(dirData, vytvoril=3, dirMedFile="", priponaMedFile="flv") jak napovídá název, tato funkce má za úkol projít zadaný adresář a nahrát z něj data do databáze pomocí funkce *SpracovaniXML*. Funkce u každého XML dokumentu kontroluje zda jej již nemá v databázi.

smaz(dokument_id) smaže z databáze dokument jemuž odpovídá předané ID, včetně pod něj spadajících sekcí a k nim přiřazených textů.

smazALL(vytvoril) smaže z databáze veškeré dokumenty jenž mají zadané *vytvoril_id*. Funkce ke své činnosti využívá služeb funkce *smaz*.

Způsoby volání programu:

```
python XML2DB.py dataSkola/cro1-201002/ 1
python XML2DB.py dataSkola/cro1-201002/ 1 DPDATA-Auto/
python XML2DB.py dataSkola/cro1-201002/ 1 DPDATA-Auto/ m4a
```

Při volání *XML2DB.py* jsou povinné dva atributy. První určuje adresář obsahující *XML* dokumenty ke zpracování. Druhý je identifikační číslo odpovídající jednomu ze záznamů v tabulce ***vytvoril***.

Zbylé dva atributy jsou nepovinné. Třetí atribut je adresář s multimediálními daty. A čtvrtý atribut přípona jakou má multimediální soubor. Pokud je rozdílná od názvu XML dokumentu, nebo v názvu XML dokumentu není uvedena.

Pokud je programu předán jen jediný atribut a je jím číslo, dojde k rekurzivnímu smazání veškerých záznamů majících dané číslo jako *vytvoril_id*. Tato funkce vznikla při ladění programu.

Příkaz k přeindexování není součástí tohoto skriptu, proto je nutné spustit po nahrání dat do databáze reindexaci *Sphinxu*. Příkaz:

```
sudo indexer --rotate delta
sudo indexer --rotate --all
```

První příkaz slouží k reindexaci delta indexu. Druhý příkaz slouží k reindexaci obou indexů, kdy se vše na-indexuje do *main* indexu a *delta* index bude prázdný. Atribut `-rotate` se použije jen pokud zrovna běží *searchd* jako služba. Jde o reindexaci za běhu.

5.2 Webové rozhraní

Jak jsem se již v úvodu zmínil, webové rozhraní je do jisté míry napsáno modulárně. Ve velké míře je zde využívána technologie zvaná *AJAX*. Úvodní stránka, *index.php*, se načte pouze jednou, a poté se se serverem komunikuje pouze *AJAXem*. Komunikaci lze rozdělit na tři hlavní části.

- Načítání mluvčích pro našeptávač.
- Odeslání požadavku na vyhledání.

- Zobrazení několika prvních nalezených výsledků a donahrání dalších nalezených výsledků.

O veškerý provoz na straně klienta se starají funkce z *function.js*. Jsou to *amluvci*, *ahledat*, *atexty*. Funkce zpracovávající výsledek ze serveru jsou *amluvic_obsluha*, *ahledat_obsluha*, *atexty_obsluha* a *atextyADD_obsluha*. Funkce *atexty_obsluha* slouží k zobrazení několika prvních výsledků, zatímco *atextyADD_obsluha* slouží pro donačítávání dalších výsledků. Obě tyto funkce využívají služeb dalších funkcí. Jsou jimi *addOnClickOnP* tato funkce přidá každému slovu popřípadě odstavci na událost *onclick* funkci *vap* o ní více níže, ta zajišťuje, aby po kliknutí bylo spuštěno přehrávání podle časové značky daného slova nebo odstavce.

Dále pak zneviditelní veškerý přijatý text. O zviditelnění textu s nalezenými výrazy se stará funkce *zobrazSickama*, která je pojmenována podle toho, že vyhledává tag `<i>` a pokud jej najde zobrazí text, ve kterém se nachází. Poslední funkcí je *zobrazKanalPorad*, tato funkce dosadí v textu místo ID kanálu a pořadu jejich názvy.

Na straně serveru je *AJAX* podporován těmito soubory:

- amluvci.php
- sphinx.php
- atexty.php

5.2.1 Skript amluvci.php

PHP script *amluvci.php* slouží k načítání dat pro našeptávač mluvčích. Po přijetí požadavku, profiltruje požadavek na zakázané znaky, které odstraní. Poté se požadavek přefiltruje pomocí funkce *mysql_real_escape_string*, to je ochrana proti tak zvanému *mysql injection*. Následně se vytvoří dotaz pro databázi. Protože není jasné zda text v dotazu je část jména nebo příjmení, hledá se v tabulce ***mluvci*** v obou atributech *mluvci_lastName* (příjmení) a *mluvci_frstName* (jméno). Přičemž se vypisují jako první jména, kde hledaný

text odpovídá části příjmení, a pak teprve pokud odpovídá části jména. Další seřazení proběhne podle abecedy. Výsledkem je seznam jmen jednotlivých mluvčích a jejich popis. Ten je následně obsloužen *JavasScriptovou* funkcí *amluvic_obsluha*. Ta přijatá data vloží do elementu *select* a zobrazí uživateli.

5.2.2 Skript sphinx.php

Dalším *PHP* scriptem je *sphinx.php*. Ten realizuje vyhledávání. Jak název napovídá, využívá k tomu *Sphinx*, který běží na definovaném portu jako služba. Ke *Sphinxu* se připojuje prostřednictvím *sphinxapi.php*, toto *API* je ke stažení společně se zdrojovými kódy *Sphinxu*. Pomocí již zmíněného *API* se vytvoří objekt, tomu se předají parametry pro vyhledávání, omezující podmínky vyhledávání, způsob seskupení výsledků a jeho seřazení. A nakonec hledaný řetězec, společně s výpisem indexů, ve kterých se má hledat. V mém případě jimi jsou *main* index a *delta* index.

Stejně jako u *amluvci.php* je i zde nutné přefiltrovat text hledaného řetězce, zda neobsahuje nepovolené znaky. Provádím to tak, že kontroluji regulárním výrazem, zda obsahuje pouze znaky povolené. V případě, že se narazí na nepovolený znak, je tento znak z dotazu vymazán. Jak jsem již popisoval dříve, veškerá další kritéria při vyhledávání pomocí *Sphinx* musí být buď typu integer, float nebo boolean. Proto je nutné tyto vstupy přefiltrovat a vytvořit z nich validní dotazy. Je nutné tyto doplňující volby ošetřit defaultním nastavením. Většina ze zmíněných voleb již předává hodnoty v integeru odpovídající identifikačním číslům v databázi, u těchto voleb se při filtrovaní spíše zajišťuje validnost.

Ovšem, co se mluvčích týče, tak *sphinx.php* dostane jejich textový seznam a musí si jednotlivá ID odpovídající daným mluvčím zjistit sám. To znamená sestavit *SQL* dotaz pro *MySQL*, připojit se a získat data. Z dat sestavit dotaz pro *Sphinx*. Nakonec je *Sphinx* požádán o samotné hledání, jeho výstupem je seznam ID dokumentů. Navíc ne zcela v opakovaně použitelné podobě, proto si tento seznam vložím do vlastního pole,

které se uloží do *session*[3], v podstatě dojde k jakémusi uložení nalezeného výsledku. Do *session* se ukládá i hledaný výraz. Jako návratová hodnota slouží výpis nalezených shod. Plus výpis shod pro jednotlivá slova. O zpracování výstupu se stará *JavaScriptová* funkce *ahledat_obsluha*, která následně zavolá funkci *ahledat* pro vypsání nalezených výsledků.

5.2.3 Skript atexty.php

Třetí částí je *atexty.php* volaný funkcí *JavaScriptovou* funkcí *ahledat*, ten se stará o zobrazování části nalezených dokumentů. Jako vstup slouží jakési číslo stránky, nalezené dokumenty se totiž zobrazují po skupinách po třech dokumentech. Z pole nalezených dokumentů uloženého v *session* vybere maximálně tři ID, které jsou v zobrazení na řadě. Tyto ID vloží do *SQL* dotazu, kterým z *MySQL* získá texty patřící pod dané dokumenty. *SQL* dotaz vypadá následovně:

```
SELECT sekce_id, dokument_id, dokument_nazev, dokument_datum,
dokument_den, dokument_time, sekce_name, text_text, dokument_file,
text_pozice, mluvci_frstName, mluvci_lastName, mluvci_popis, kanal_id,
porad_id FROM `dokument` LEFT JOIN sekce USING(`dokument_id`) LEFT JOIN
text USING(`sekce_id`) LEFT JOIN mluvci USING(`mluvci_id`) WHERE
`dokument_id` in (sem_seznam_id) ORDER BY FIND_IN_SET(`dokument_id`,
'sem_seznam_id'),`sekce_id`,`text_pozice`
```

Výsledek z tohoto dotazu je nutné zpracovat do uživateli zobrazitelné podoby. Je zapotřebí rozdělit je na jednotlivé dokumenty. Ty dále rozdělit na jednotlivé sekce a pod ně vložit odpovídající texty, fráze společně s jejich mluvčími. Protože z databáze již přichází data patřičně seřazena, což zajistil příkaz *ORDER BY*, stačí sledovat změny *dokument_id* a *sekce_id*. Složitější problém nastává s označením hledaného textu. Jedinou možností je využití regulárních výrazů, jak již zaznělo v kapitole 3 v citátu od Jamie Zawinského. Nahradit fulltextové vyhledávání regulárními výrazy je problém. Zvláště vezmeme-li v po-

[3]session - Proměnná na serveru udržované v paměti serveru po určitou dobu. Přístup k její hodnotě má vždy jen ten samý klient.

taz, že *Sphinx* při vyhledávání ignoruje diakritiku. Musel jsem tedy vytvořit regulární výraz, který bude doslova simulovat vyhledávání *Sphinx* a to tak, aby zobrazené výsledky odpovídaly výsledkům *Sphinxu*.

Jak se ukázalo, vytvořit takový regulární výraz je nemožné. Tento problém jsem nakonec vyřešil několika regulárními výrazy, které se za běhu vytvoří z hledaného textu. Tvorba samotných regulárních výrazů je navíc ovlivněna módem, ve kterém *Sphinx* vyhledává. Je rozdílné pokud se má označit celistvá fráze složená z několika slov, nebo se vyhledává fráze, ve které se jen daná slova vyskytují bez vztahu mezi sebou *SPH_MATCH_ALL*, nebo *SPH_MATCH_ANY*. Pro každý z případů se postaví jiné regulární výrazy, které jsou posléze aplikovány na jednotlivé texty, pokud je daný text nalezen, je obalen do tagu *Nalezene slovo nebo fraze*. Toto řešení u frází, pokud je časovou značkou obaleno každé slovo, vnáší nevalidnost výsledného *HTML* kódu. Jako tag pro časové značky používám tag *<p s="cas"e="cas">slovo(a)</p>*, kde *s* je čas začátku(start) a *e* čas konce slova(end). Po prostém obalení fráze může dojít k *<p s="cas"e="cas">slovoA </p><p s="cas"e="cas">slovoB </p><p s="cas"e="cas">slovoC </p>*, přičemž regulární výraz musí s danými značkami pracovat a brát je jako oddělovače. Co je, ale z příkladu vidět, je ona nevalidnost, kdy tag začíná uprostřed jednoho tagu a končí uprostřed jiného tagu. K ošetření tohoto jevu je použito opět regulárního výrazu spolu s *callbackem*, kdy pokud je daný regulární výraz nalezen, je nalezená část předána funkci, která jej opracuje. V mém případě ono opracování znamená do-přidání příslušných značkovacích tagů.

A jak je v regulárním výrazu řešeno ono ignorování diakritiky? Nejprve je dotaz zbaven diakritiky a následně se pomocí tabulek nahradí vždy dané písmeno všemi jeho mutacemi, ukázka pro písmeno a *a=>[aäá]*. K textu je přidáno jméno mluvčího a výsledný text je obalen tegem *div*. Texty patřící do jednoho dokumentu jsou rovněž obaleny tagem *div* kterému je přidán atribut *file*, jehož obsahem je adresa multimediálního souboru pro pozdější možnost přehrání. V případě, že nebyly z pole výsledků uloženého v *session*

vyčerpány veškeré dokumenty, je na konec kódu přidáno tlačítko pro možnost nahrání dalších tří dokumentů. Takto připravený kód se vrátí klientovy a dále jej zpracuje funkce *atexty_obsluha* pokud, se jedná o výpis prvních třech, nebo *atextyADD_obsluha* pro donačtení dalších dokumentů.

5.2.4 JavaScriptové funkce

Soubor *function.js* má v sobě ještě několik další funkcí, kromě těch které se starají o *AJAX*. Jde o funkce spuštěné při načtení základní stránky *index.php*. Tyto funkce se postarají o naplnění hodnot pro datum a čas, u měsíců se kontroluje vybraný rok, zda není přestupný a podle toho se mění počet dní v únoru. Dále se starají o dynamickou nabídku pořadů pro daný kanál. Data k této nabídce se načítají při nahrávání stránky pomocí souboru *aporatData.php*, který je načte z databáze a vytvoří z nich pole v jazyce *JavaScript*. Na stránce je možnost rozbalit si celý článek, a poté je opět sbalit, funkce se aktivuje tlačítkem s popisem *Zobrazit vše*. Nebo je možné zobrazit článek tak, jak vypadal po vyhledání, tj. před spuštěním přehrávání, funkce je navázána na tlačítko s popiskem *Původní*. Tyto funkce jsou opět součástí *function.js*.

JavaSrciptový soubor *prehravaniflowplayer.js* obsahuje funkce starající se o věci kolem přehrávání. Nejdůležitější funkcí je funkce *vap*, ta spouští přehrávání. Zjistí jaký multimediální soubor je právě přehráván a podle toho se rozhoduje, zda pouze zažádat o přehrávání nového souboru nebo jen stačí změnit pozici v přehrávaném souboru, přesněji streamingu. Dále se tato funkce stará o zapnutí *seekMonitoru*, což je v jistých časových intervalech volaná *JavaSkriptová* funkce *seekMonitor()*, která se stará o označování právě přehrávaných slov a odznačování slov již nepřehrávaných. Dále se stará o rozbalování textu. Tak aby byl vidět vždy následující text. S tím souvisí již zmíněné tlačítko s popiskem *Původní*, která toto rozbalení zruší. Pro komunikaci s flash přehravačem slouží *API flowplayer-3.1.4.min.js*.

O vzhledovou část se stará již zmíněný *CSS* soubor *styl.css*. Bohužel nejsem příliš graficky nadaný a proto jsem se při grafickém návrhu stránky spíše snažil zdůraznit jednotlivé části a funkce.

5.2.5 Stránka info.php

Veškeré *PHP* soubory využívají k navázání spojení s databází soubor *db.php*, a to pomocí *include*. Aby se nám snáze určovalo množství dat v systému, vytvořil jsem pomocnou stránku *info.php*, jejímž obsahem je pouze jednoduchá tabulka s výpisem počtu dokumentů a součet času pro jednotlivé položky v tabulce ***vytvoril***. Poslední řádek tabulky je součet. Viz Obrázek 3.

ID	popis	dokumentu count	duration sum
1	automatický přepis	4847	1213:32:21
2	Newton	57	23:23:40
3	text archiv radia, časy automat v2 (odstavce)	1756	252:53:54
4	text archiv radia, časy automat v3 (jednotliva slova)	992	134:03:41
		7652	1623:53:36

Obrázek 3: Náhled info.php z 14.5.2010

5.3 Konfigurace Sphinx

Konfigurační soubor má defaultní název *sphinx.conf*. Jeho defaultní umístění v Linuxu je v adresáři */usr/local/etc*. Pokud jej máme jinde, pak při volání jakéhokoli nástroje *Sphinx*, přidáváme parametr *-c* nebo *–config* a cestu ke konfiguračnímu souboru.

Konfigurační soubor se skládá ze čtyř částí.

- Definice zdrojů dat.
- Definice indexů pro tyto zdroje dat.
- Definice nastavení indexace.

- Definice nastavení vyhledávače.

5.3.1 Definice zdrojů dat

Zde se definuje připojení k databázi. Způsob kódování dat v mé práci je *utf8*. Dále po kolika dokumentech se budou data z databáze načítat. Dále dotaz do databáze na úpravu tabulky ***sph_counter***. Následuje dotaz pro databázi, kterým se získají data, nad kterými se postaví index. Všechny atributy, přes které chceme vyhledávat, je nutné vyjmenovat a určit jim typ. Ukázka části definice dat pro *main* index.

```
sql_range_step    = 5000 #pocet kolik se nacte dat
sql_query_pre     = REPLACE INTO sph_counter SELECT 1, MAX(text_id) FROM
text
sql_query         = SELECT text_id, mluvci_id, text_text, dokument_id,
kanal_id, porad_id, dokument_den, dokument_duration, typ_zaznamu_id,
vytvoril_id, dokument_ts, dokument_cas FROM `text` LEFT JOIN sekce
USING(`sekce_id`) LEFT JOIN dokument USING(`dokument_id`) WHERE
text_id<=( SELECT max_doc_id FROM sph_counter WHERE counter_id=1 )
sql_attr_uint     = dokument_id#
sql_attr_uint     = mluvci_id#
sql_attr_uint     = kanal_id#
...
```

V této práci využívám dvou indexů *main* a *delta*. Delta index se používá při vložení malého počtu dat do systému, aby se nemusely indexovat znova veškerá data, naindexují se nová data pouze do delta indexu. Vyhledávání pak, ale musí probíhat v obou indexech a výsledky je nutné spojit.

5.3.2 Definice indexů pro tyto zdroje dat

Každý index musí mít svou fyzickou podobu, ta se definuje v této části. Ukázka definice zdroje dat pro *main* index.

```
source       = main
path         = ../data/main
docinfo      = extern
charset_type = utf-8

min_word_len = 1
min_infix_len = 3
enable_star = 1

html_strip = 1
html_remove_elements = style, script

charset_table = 0..9, a..z, _, A..Z->a..z, U+00C0->a, U+00C1->a ...
```

První atributem *source* určujeme jakému zdroji dat index patří, *path* určuje cestu a název fyzického uložení, *docinfo* určuje mód uložení, *charset_type* je znaková sada vstupních dat, *min_word_len* délka nejkratšího indexovaného slova, hodnota 1 znamená indexuje se vše, *min_infix_len* nejkratší prefix při hvězdičkové notaci, *enable_star* povolené nebo zakázání hvězdičkové notace, *html_strip* 1 pokud je vstupem *html* kód, *html_remove_elements* elementy, jejichž vnitřek nesmí být zpracován. Atribut *charset_table* je tabulka povolených znaků a výčet jednotlivých nepovolených znaků společně se znakem, na který se má přepsat. Použití, vstupní znaky zbavit diakritiky. Dále je nutné všechna písmena převést na malá.

5.3.3 Definice nastavení indexace

U indexeru lze nastavit velikost *RAM*, kterou může využít ke své činnosti, dále maximální počet vstupně/výstupních operací za sekundu (I/O). A maximální velikost výstupního indexu. V této práci jsem nastavoval pouze velikost RAM, počet I/O operací jsem nechal defaultně na neomezeném počtu, stejně tak velikost indexu neomezená.

5.3.4 Definice nastavení vyhledávače

Nejdůležitějšími atributy při nastavování vyhledávače, nástroj *searchd* je nastavení adres a portu. Z hlediska bezpečnosti běží li celý systém na jednom počítači, je vhodné jako adresu uvést pouze lokální adresu 127.0.0.1. Defaultní port je 3312, ten bývá na Linuxu obsazen, a proto používám port 3333. Dále lze nastavit adresy logů. A to log samotného vyhledávače, i speciální log pro logování dotazů. Maximální čas na vykonání dotazu. Počet vláken. Maximální počet shod. To je výčet těch nejdůležitějších. V dokumentaci lze dohledat ještě několik speciálních jejichž defaultní hodnoty mi přišly nastavené rozumně a tedy je v konfiguračním souboru neovlivňuji.

6 Vlastnosti a podoba systému

6.1 Rychlost vyhledávání

Co se rychlosti vyhledávání týče, lze tento systém testovat pouze vůči jeho předchůdci. **Ahmed** však již delší dobu nefunguje, a proto nelze uvést žádné přesné hodnoty. Vzhledem k tomu, že vyhledávání na původním **Ahmedovi** s naindexovanými přepisy čítajícími něco kolem 2000 hodin trvalo v řádu sekund, tak na **Ahmedovi II** vyhledání v přepisech čítajících 5264 hodin 29 minut 45 sekund trvá podle složitosti výrazu mezi 1 ms až 200 ms, průměrná hodnota se pohybuje kolem 17 ms. Následné načítání textu pomocí **AJAXu** zpravidla nepřekročí 250 ms. S přihlédnutím k tomu, že **Ahmed II** je rovněž nainstalován na stejném počítači, na kterém kdysi běžel dnes již vysloužilý **Ahmed** (Intel(R) Core(TM)2 CPU 6300 @ 1.86GHz, RAM: DDR II 2 GB 800 MHz), lze **Ahmeda II** považovat za rychlejšího, čímž splňuje jeden ze základních požadavků zadání.

K horní hranici 250 ms se **Ahmed II** přiblíží zpravidla pouze při prvním dotazu za předpokladu, že jej delší dobu nikdo nepoužíval. Z toho usuzuji, že daná prodleva je způsobena načítáním indexu z disku do operační paměti, než samotným vyhledávačem.

Další otázkou bylo jak se projeví přibývající množství dat na rychlosti. Neboť jsem měl k dispozici jen určitý počet dat, rozhodl jsem se provést zatížení smyšlenými daty. Pro tento účel jsem modifikoval program *XML2DB.py* tak, aby článkům které vkládá do databáze náhodně zpřeházel pořadí slov, tím vzniknou sice nesmyslná data, ale pro test zatížení dostačující, zároveň se ale mohou použít příslušné délky nahrávek.

Pro samotný test jsem vytvořil sadu skriptů, který z předpřipraveného korpusu vět vytvoří dotazy o náhodném počtu slov, maximální množství slov lze ovlivnit. Následně takto vytvořené dotazy skript předloží **Ahmedovi II**, a to prostřednictvím webové rozhraní. Požadavky byly zaslány ve stejném tvaru jako by je zaslal prohlížeč pomocí *AJAXu*, proto jsem mohl dotazy upravit i o podmínky vyhledávání. Počet takto zaslaných dotazů je nastavitelný, během testování jsem používal 1000 a 3000 dotazů za sebou. Testovací

program vytváří logy, ve kterých zaznamenává informace o celkovém počtu dokumentů na serveru včetně počtu hodin, který dávají. Maximální počet slov, ze kterých se vytváří dotazy. Jednotlivé dotazy včetně počtu nalezených dokumentů a čas délky hledání. Po skončení dotazování vypočte a uloží průměrnou hodnotu, medián a maximální hodnotu. Testovací skripty jsem spouštěl na svém pc zatímco server s *Ahmedem II* je umístěn v prostorách univerzity.

Typy dotazů:

a Vyhledat slova jako frázi

b Vyhledat slova jako frázi, ale jen ve středu, čtvrtek, sobotu a neděli od 13:00

c Vyhledat slova jako frázi, ale jen v pondělí, úterý, sobota a neděle

d Vyhledat slova jako frázi, ale jen v pátek, sobota a neděle od 11:30 do 22:00

Tabulka 3 obsahuje výsledky testu rychlosti vyhledávání na vzorku 22215 dokumentů čítajících 5264 hodin 29 minut a 45 sekund záznamů. Každý test se skládal z 1000 náhodných dotazů.

Tabulka 4 obsahuje výsledky vyhledávání na vzorku 22215 dokumentů čítajících 5264 hodin 29 minut a 45 sekund záznamů. Nová testovací data byla naindexována do *delta* indexu. Původní platná data v *main* indexu. Každý log 10,11,12 se skládá z 1000 náhodných dotazů, logy 13 a 14 se skládají z 3000 dotazů.

V tabulce 6 umístěné v přílohách jsou zaneseny výsledky paralelního testu. Najednou bylo spuštěno 49 testovacích skriptů, každý s náhodně přidělenými 1000 dotazy, typem dotazu a. Cílem bylo simulovat současné vyhledávání většího počtu uživatelů. Je zde pozorovatelný nárůst maximálního času, a to až za hranici 500 ms, avšak průměrná délka vyhledávání osciluje kolem hodnoty 17 ms. Zvýšený nárůst maximálního času si lze vysvětlit nynějším nastavením *Sphinxu* na pouhých 30 vláken obsluhujících vyhledávání.

Tabulka 3: Test rychlosti vyhledávání v 5264,5 hodinách přepisů, pouze *main* indexu

	Max. počet slov	Typ dotazu	Průměrný čas	Medián	Maximální čas
log0	2	a	0.0210 s	0.018 s	0.074 s
log1	2	b	0.0153 s	0.014 s	0.065 s
log2	2	b	0.0139 s	0.012 s	0.061 s
log3	2	a	0.0144 s	0.012 s	0.061 s
log4	2	a	0.0136 s	0.012 s	0.059 s
log5	4	a	0.0156 s	0.013 s	0.079 s
log6	4	a	0.0144 s	0.013 s	0.082 s
log7	4	c	0.0137 s	0.012 s	0.066 s
log8	3	c	0.0114 s	0.010 s	0.064 s
log9	3	c	0.0114 s	0.009 s	0.078 s

Tabulka 4: Test rychlosti vyhledávání v 5264,5 hodinách přepisů

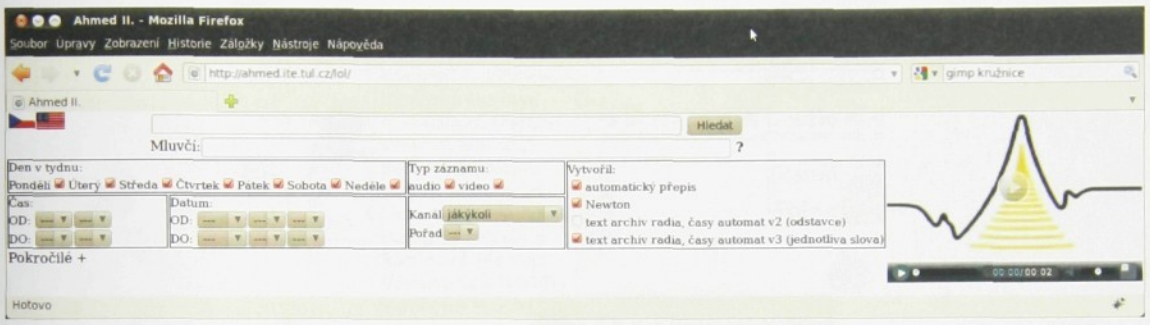
původní data *main* index, „falešná“ data *delta* index

	Max. počet slov	Typ dotazu	Průměrný čas	Medián	Maximální čas
log10	4	c	0.0337 s	0.032 s	0.138 s
log11	2	c	0.0222 s	0.019 s	0.082 s
log12	2	a	0.0180 s	0.015 s	0.068 s
log13	3	a	0.0140 s	0.011 s	0.081 s
log14	3	d	0.0111 s	0.008 s	0.100 s

6.2 Uživatelské rozhraní

Uživatelské rozhraní tvoří webová aplikace. Skrze níž je realizována komunikace uživatele s vyhledávačem a obráceně interpretace nalezených údajů uživateli.

Na obrázku 4 je mnou navržený vzhled této aplikace, jak vypadá po načtení stránky. Kromě samotného vyhledávání podle zadané fráze je zde možnost omezit vyhledávání dle mluvčího. Po zadání prvních tří písmen, buď jména nebo příjmení, se zobrazí seznam mluvčích jejichž promluvu má **Ahmed II** naindexovanou, náhled je na obrázku 5, pokud kurzorem najedeme na některou z osob, zobrazí se nám její popis, pokud je v databázi.

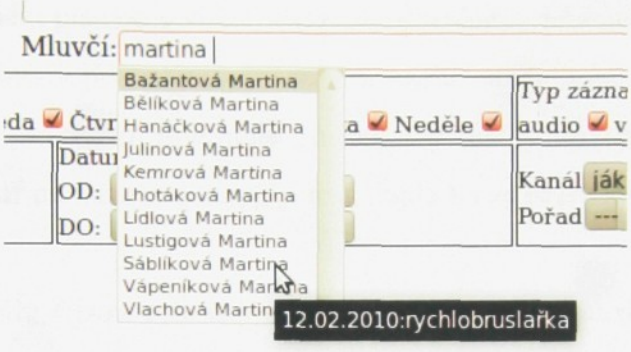


Obrázek 4: Náhled webové aplikace

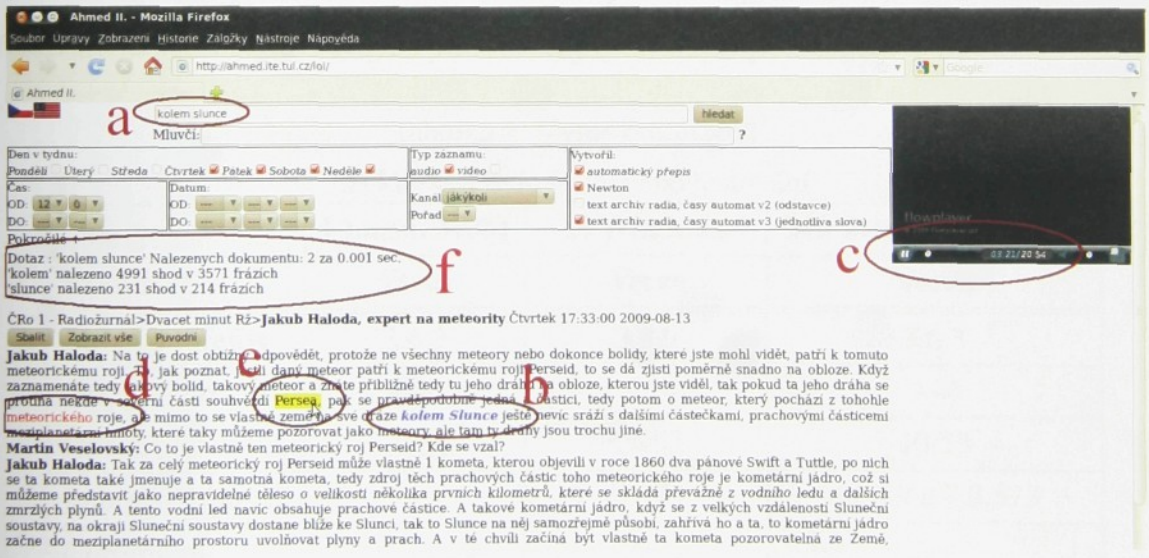
Dalšími volbami ovlivňující vyhledávání jsou:

- Dny v týdnu ty lze určovat pomocí zaškrtávacích políček (*checkboxů*), použití *checkboxů* přináší možnost libovolné kombinace dnů.
- Výběr typu záznamu (multimédia), ze kterého byly přepisy pořízeny, jsou možnosti tři *Audio*, *Video* nebo obojí.
- Datum vysílání nebo vzniku dokumentu. A to hned třemi způsoby: vyplněním pouze datumu *OD* dojde k vyhledání dokumentů vzniklých v zadaný den nebo později, vyplnění pouze datumu *DO* vyhledá pouze dokumenty vzniklé před tímto datem včetně data zadaného, vyplněním obou datumů se vyhledají dokumenty v zadaném rozmezí.
- Podle času denní doby, kdy byl daný dokument vysílán. I zde jsou možné tři kombinace vyhledávání jako u datumu.

- Omezení kanálu, zdroje vysílání př. Prima, ČR1, atd., při výběru kanálu se otvírá možnost upřesnit dotaz i na název pořadu.
- Položka vytvořil, zde si uživatel může vybrat kým byla data pro vyhledávač vytvořena.



Obrázek 5: Ukázka výběru mluvčího



Obrázek 6: Náhled webové aplikace při přehrávání

Na obrázku 6 je zobrazena webová aplikace poté, co skrze ní byly nalezeny dokumenty, které byly publikovány buď ve čtvrtek, pátek, sobotu nebo neděli po dvanácté hodině, a musí se jednat pouze o zvukovou neboli audio nahrávku a hledanou frází je fráze „kolem slunce". Zadání fráze je v elipse označené písmenem **a**, zatím co v ovále označeném písmen **b** je vidět modrým písmem zvýrazněná hledaná fráze.

V ovále označeném písmen **c** si lze všimnou spuštěného přehrávání nalezeného dokumentu, a s ním související zvýraznění přehrávaného textu červenou barvou, ovál **d**. V ovále označeném písmenem **e** je žlutě podbarvený text po té, co na něj bylo najeto kurzorem, po kliknutí na takto zvýrazněný text dojde ke spuštění přehrávání od daného místa.

V posledním oválu **f** jsou zobrazeny informace ohledně vyhledávání, je zde počet nalezených dokumentů společně s časem jak dlouho se hledalo. Pod těmito údaji si lze všimnout doplňujících informací o jednotlivých slovech tvořících dotaz.

Funkčnost uživatelského rozhraní jsem odzkoušel ve třech operačních systémech na několika různých prohlížečích viz tabulka 5 Aplikace se jeví ve všech prohlížečích jako funkční. Screenshoty uvedených prohlížečů jsou součástí přiloženého *DVD*.

Tabulka 5: Výpis prohlížečů
na kterých bylo provedeno testování

	Ubuntu 10.04	**MS Windows 7**	**MS Windows XP SP3**
Prohlížeč	**Verze**	**Verze**	**Verze**
Mozilla Firefox	3.6.3	3.5.9	3.6.3
Google chrome	—	4.1.249	—
Opera	10.10	10.53	10.53
Internet Explorer	—	8.076	6.0.29 a 7.0.573
Chromium	5.0.342.9	—	—
Arora	0.10.2	—	—
Dooble	0.07	—	—

7 Závěr

Cílem DP byly: navržení architektury systému pro skladování, indexování a následné prohledávání obsahu multimediálních souborů, návrh uživatelského rozhraní umožnující práci s tímto systémem, navržený systém a uživatelské rozhraní implementovat a otestovat.

Za obsah multimediálního souboru je v této práci považována lidská řeč ve zvukové stopě. Tu lze převést (přepsat) na text obohacený o časové značky jednotlivých slov nebo bloků textu, v některých případech i o mluvčí. Mým úkolem tedy bylo vytvořit systém, který v těchto textech bude vyhledávat, ale zároveň bude ukládat a využívat ony časové značky a informace o mluvčích. Návrh systému byl ovlivněn i dalším požadavkem, a to rychlým vyhledáváním ve velkém množství přepisů, tisíce hodin záznamů. Dále možností vyhledávání ovlivňovat dalšími parametry např. mluvčím, dnem v týdnu atd.

Celý systém pojmenovaný **Ahmed II**, se mi povedlo sestavit z open source softwaru a mnou vytvořených skriptů. Pro ukládání dat využívám databázový systém *MySQL*, indexaci textových dat a následné vyhledávání realizuji pomocí *Sphinxu*. Data do systému vkládám skrze mnou napsaný skript *XML2DB.py*.

Jako uživatelské rozhraní jsem vytvořil webovou aplikaci založenou na *AJAXu*. Funguje zde našeptávání mluvčích. Vyhledávání se skládá ze dvou částí, nejprve dojde k vyhledání a vypsání výsledku vyhledání, na to dojde k zobrazení prvních výsledků vyhledávání. Výsledek vyhledávání si lze po kliknutí na vypsaný text nechat přehrát. Při přehrávání jsou označována přehrávaná slova nebo části textu podle toho, jak jsou dostupné časové značky. Pomocí časových značek, při kliku na text, dochází k přehrávání od daného místa. Pro upřesnění vyhledávání je k dispozici sada voleb pomocí zaškrtávacích políček (př. u dnů v týdnu) nebo rozbalovací nabídky (př. názvy pořadů).

Hlavní požadavek rychlého vyhledávání jsem myslím splnil, průměrný čas vyhledávání v něco přes 5200 hodinách přepisů se pohybuje kolem kolem 17 ms. Maximální čas vyhledávání zatím nepřekročil 250 ms., více v kapitole 6.1. Předchůdce mého systému

Ahmed při cirka v 2000 hodinách přepisů vyhledával v řádu sekund. Rád bych zdůraznil, že **Ahmed II** běží na stejném počítači jako jeho předchůdce.

Také jsem si vytvořil vlastní způsob získávání dat. Využívám již existujících přepisů z archivu *ČR1-Radiožurnál*, ty jsou ale bez časových značek, proto jsem vytvořil program pro jejich určování.

Práce na DP byla pro mě velkým přínosem, zvláště pak instalace a zprovoznění serveru běžícího na Linuxu a s veřejnou ip. Počáteční obtíže byly časem nahrazeny obdivem tohoto systému. A nakonec se Linux stal operačním systémem i mého pc.

A Přílohy

A.1 XML schéma

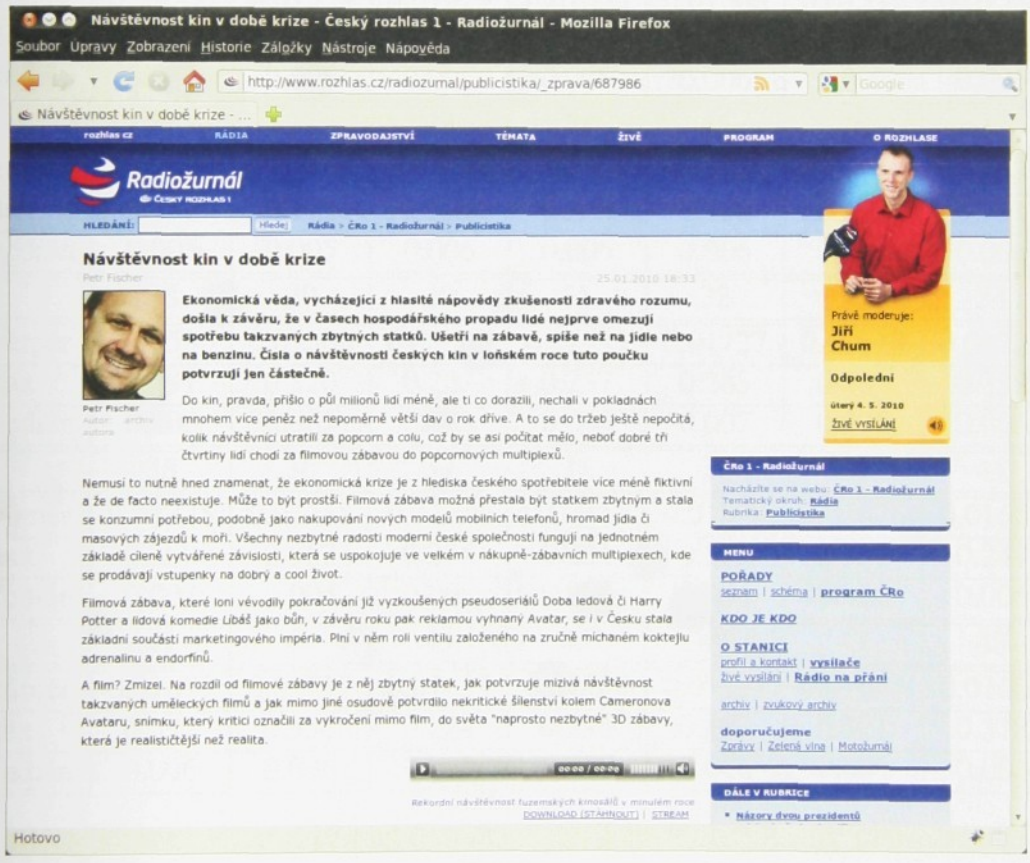
Podle tohoto *XML schématu*[9] jsou ukládány přepisy.

```
<xs:schema xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xs:element name="Transcription">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Chapters" type="TypeChapters" />
        <xs:element name="SpeakersDatabase"
                    type="TypeSpeakersDatabase" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:complexType name="TypeChapters">
    <xs:sequence>
          <xs:element name="Chapter" type="TypeChapter"
                      maxOccurs="unbounded"/>
      </xs:sequence>
  </xs:complexType>

    <xs:complexType name="TypeChapter">
      <xs:sequence>
          <xs:element name="Sections" type="TypeSections" />
      </xs:sequence>
      <xs:attribute name="name" type="xs:string" />
      <xs:attribute name="kanal" type="xs:string" minOccurs="0" />
      <xs:attribute name="porad" type="xs:string" minOccurs="0" />
      <xs:attribute name="href" type="xs:string" minOccurs="0" />
      <xs:attribute name="begin" type="xs:integer" />
      <xs:attribute name="end" type="xs:integer" />
    </xs:complexType>

    <xs:complexType name="TypeSections">
    <xs:sequence>
          <xs:element name="Section" type="TypeSection"
                      maxOccurs="unbounded"/>
      </xs:sequence>
  </xs:complexType>

    <xs:complexType name="TypeSection">
    <xs:sequence>
          <xs:element name="Paragraphs" type="TypeParagraphs" />
```

```
    </xs:sequence>
    <xs:attribute name="name" type="xs:string" />
    <xs:attribute name="begin" type="xs:integer" />
    <xs:attribute name="end" type="xs:integer" />
</xs:complexType>

<xs:complexType name="TypeParagraphs">
  <xs:sequence>
        <xs:element name="Paragraph" type="TypeParagraph"
                    maxOccurs="unbounded"/>
    </xs:sequence>
</xs:complexType>

<xs:complexType name="TypeParagraph">
  <xs:sequence>
        <xs:element name="Phrases" type="TypePhrases" />
        <xs:element name="speakerID" type="xs:integer" />
    </xs:sequence>
    <xs:attribute name="begin" type="xs:integer" />
    <xs:attribute name="end" type="xs:integer" />
</xs:complexType>

<xs:complexType name="TypePhrases">
  <xs:sequence>
        <xs:element name="Phrases" type="TypePhrase"
                    maxOccurs="unbounded"/>
    </xs:sequence>
</xs:complexType>

<xs:complexType name="TypePhrase">
    <xs:sequence>
      <xs:element name="Text" type="xs:string" />
    </xs:sequence>
  <xs:attribute name="begin" type="xs:integer" />
    <xs:attribute name="end" type="xs:integer" />
</xs:complexType>

<xs:complexType name="TypeSpeakersDatabase">
  <xs:sequence>
        <xs:element name="Speakers" type="TypeSpeakers" />
        <xs:element name="speakersIndexCounter"
                    type="xs:integer" />
    </xs:sequence>
</xs:complexType>

<xs:complexType name="TypeSpeakers">
  <xs:sequence>
        <xs:element name="Speaker" type="TypeSpeaker"
                    maxOccurs="unbounded"/>
```

```
        </xs:sequence>
    </xs:complexType>

    <xs:complexType name="TypeSpeaker">
        <xs:sequence>
            <xs:element name="ID" type="xs:integer" />
            <xs:element name="FirstName" type="xs:string" />
            <xs:element name="Surname" type="xs:string" />
            <xs:element name="Sex" type="xs:string" />
            <xs:element name="Comment" type="xs:string" />
        </xs:sequence
    </xs:complexType>
</xs:schema>
```

A.2 Obrázek



Obrázek 7: Náhled dokumentu v archivu Radiožurnálu.

A.3 Tabulka

Tabulka 6: Test rychlosti: 49 paralelních vyhledávání
hodnoty v ms

	0	1	2	3	4	5	6
Průměr	0.01607	0.01735	0.01590	0.01818	0.01578	0.01787	0.01654
Max	0.204	0.222	0.189	0.231	0.271	0.208	0.214
Medián	0.007	0.006	0.006	0.006	0.006	0.006	0.006
	7	8	9	10	11	12	13
Průměr	0.01635	0.01783	0.01564	0.01608	0.01671	0.01866	0.01606
Max	0.237	0.374	0.442	0.419	0.352	0.341	0.502
Medián	0.005	0.006	0.006	0.006	0.006	0.006	0.005
	14	15	16	17	18	19	20
Průměr	0.01563	0.01554	0.01529	0.01655	0.01566	0.01512	0.01584
Max	0.259	0.241	0.256	0.289	0.268	0.205	0.204
Medián	0.006	0.005	0.005	0.006	0.006	0.005	0.005
	21	22	23	24	25	26	27
Průměr	0.01620	0.01448	0.01646	0.01809	0.01810	0.01743	0.01685
Max	0.357	0.173	0.309	0.343	0.220	0.233	0.191
Medián	0.005	0.005	0.006	0.006	0.006	0.006	0.006
	28	29	30	31	32	33	34
Průměr	0.01556	0.01500	0.01660	0.01733	0.01727	0.01618	0.01727
Max	0.191	0.220	0.208	0.237	0.263	0.403	0.358
Medián	0.005	0.005	0.005	0.006	0.007	0.006	0.006
	35	36	37	38	39	40	41
Průměr	0.01775	0.01718	0.01580	0.01715	0.01570	0.01676	0.01655
Max	0.300	0.267	0.286	0.251	0.373	0.359	0.247
Medián	0.007	0.005	0.005	0.006	0.006	0.006	0.006
	42	43	44	45	46	47	48
Průměr	0.01679	0.01752	0.01617	0.01566	0.01551	0.01662	0.01601
Max	0.332	0.350	0.224	0.296	0.193	0.321	0.318
Medián	0.006	0.006	0.005	0.006	0.006	0.006	0.006

B Zprovoznění Serveru

Jak již bylo zmíněno, všechny součásti potřebné pro chod serveru byly vybírány s ohledem na nezávislost na operačním systému. V následující kapitole bude popsána instalace komponent potřebných pro chod serveru. Postup instalace pro Linux je popisován pro distribuci Ubuntu.

B.1 Shinx

B.1.1 Na Linuxu

Sphinx ke svému běhu potřebuje knihovnu pro komunikaci s mysql databází. A tou je libmysqlclient.

- Příkaz pro instalaci mysql clienta:

```
sudo apt-get install libmysqlclient15-dev
```

Protože je sphinx pro unix distribuován ve zdrojových kódech, bude je nutné zkompilovat. Vše potřebné obsahuje tento balíček.

- Příkazy pro instalaci balíčku potřebného ke kompilování.

```
sudo apt-get update
sudo apt-get dist-upgrade
sudo apt-get install build-essential
```

- Stažení a rozbalení zdrojových kódů Sphinx.

```
wget http://www.sphinxsearch.com/downloads/sphinx-0.9.9.tar.gz
tar xvzf sphinx-0.9.9.tar.gz
cd sphinx-0.9.9/
```

Nyní ke slovu přichází tak zvaná "Svatá trojice"(configure, make, make install).

- configure - dojde k vytvoření Makefile, v parametru jsou předány umístění potřebných knihoven z důvodu komunikace Sphinx a MySQL.

```
./configure --with-mysql-includes=/usr/include/mysql
--with-mysql-libs=/usr/lib/mysql
```

- make - provede se kompilace, jeji postup je v Makefile
  `make`

- instalace - zkopírování s kompilovaných souborů na příslušná místa
  `make install`

  /usr/local/bin - složka se spustitelnými soubory Sphinxu

  /usr/local/etc - složka s konfiguračním souborem Sphinxu

  /usr/local/var/data - složka defaultně určená pro data Sphinxu

Nyní je nutné nakonfigurovat Sphinx úpravou souboru sphinx.conf, konfigurace je podrobně popsána v kapitole 5.3.

- Spuštění vyhledávače jako služby provedeme příkazem.
  `sudo searchd`

B.1.2 Na Windows

- Ze stránky *http://sphinxsearch.com/downloads.html* si stáhneme zazipované binární soubory zkompilované již pro windows *Win32 binaries w/MySQL support*.

- Ty rozbalíme do složky *c://Sphinx*.

- Provedeme konfiguraci Sphinx přepsáním souboru sphinx.conf, konfigurace je podrobně popsána v kapitole 5.3.

- Protože Sphinx ke spolupráci s MySQL vyžaduje knihovnu *libmySQL.dll*[4] , která však není součástí MS Windows je nutné tuto knihovnu donahrát do `c:\\windows\system32`.

[4]Ke stažení zde URL:<http://sphinxsearch.com/downloads/libmySQL-5.0.37.zip>

- Provedeme instalaci vyhledávače jako služby, příkazem

```
searchd --install --config sphinx.conf
```

B.2 Red5

B.2.1 Na Linuxu

Red5 ke svému chodu potřebuje Javu, pro její instalaci je nutné přidat repozitář(úložiště obsahujících balíčky) do seznamu repozitářů. Ten se v Ubuntu nachází v */etc/apt/sources.list*. K úpravě lze použít editor *nano*, popřípadě *vim* je–li nainstalován. Na konec sources.list tedy vložíme "deb http://za.archive.ubuntu.com/ubuntu/ jaunty multiverse".

- Příkaz pro editaci sources.list

```
sudo nano /etc/apt/sources.list
```

Nyní je potřeba uplatnit změnu seznamu repozitářů v systému.

- Provede update balíčků, poté je proveden upgrade.

```
sudo apt-get update
sudo apt-get upgrade
```

Dále doinstalujeme několik pomocných programů, *locate* - rychle vyhledává soubory podle jejich jména, využívá k tomu předvygenerovanou databázi, *htop* - grafický *top*, seznam spuštěných procesů, *subversion* - nástroj pro kontrolu verze klient (svn)

- Příkazy pro doinstalování locale,htop,telnet,subversion.

```
sudo apt-get install locate
sudo updatedb &
sudoapt-get install htop telnet subversion
```

Nyní je nutné nainstalovat Javu a Ant. Ant je to samé jako make, ale je na platformě nezávislý a využívá XML namísto Makefile.

- Příkazy pro doinstalování Javy a Ant.

```
sudo apt-get install java-package
sudo apt-get install sun-java6-jdk
sudo apt-get install sun-java6-jre
sudo apt-get install ant
```

- Kontrola zda se Java nainstalovala.

```
java -version
```

- Pokud je nainstalovaná, vypíše podobnou odpověď.

```
java version "1.6.0_0"
OpenJDK Runtime Environment (IcedTea6 1.6.1) (6b16-1.6.1-3ubuntu3)
OpenJDK 64-Bit Server VM (build 14.0-b16, mixed mode)
```

Nyní si vytvoříme pomocnou složku, kterou lze po dokončení instalace smazat. Do ní stáhneme poslední verzi Red5 a pomocí Ant provedeme kompilaci.

- Vytvoření pomocné složky a stažení red5.

```
cd ~
mkdir temp
cd temp
svn co http://red5.googlecode.com/svn/java/server/trunk red5
cd red5
```

- Definice zástupných jmen.

```
export JAVA_HOME=/usr/lib/jvm/java-6-sun-1.6.0.15/
export ANT_HOME=/usr/share/ant/
```

- Spuštění kompilace.

```
/usr/share/ant/bin/ant
```

Po úspěšné kompilaci přesuneme zkompilovaný program do složky /usr/shar/red5. Nakonec nastavíme práva skriptu přes, který budeme server Red5 spouštět.

- Vytoření složky, překopírování programu.

```
sudo mkdir /usr/share/red5
sudo cp -R ./dist/* /usr/share/red5/
```

- Nastavení práv

```
sudo cd /usr/share/red5
sudo chmod 755 red5.sh
```

Nyní máme nainstalovaný server *red5*, složku *temp* můžeme smazat.

Server se spouští příkazem.

```
/usr/share/red5/red5.sh &
```

B.2.2 Na Windows

Ze stránky *http://code.google.com/p/red5/* si stáhneme instalační balíček pro windows. Poté již stačí klikat na tlačítko *Další*.

B.2.3 Společná část

Do prohlížeče zadáme adresu 127.0.0.1:5080, pokud se nám instalace a spuštění povedla, naběhne nám stránka, kde klikneme na odkaz *install*. Zde nainstalujeme oflaDemo. Soubory, které chceme streamovat nahrajeme do složky *webapps/oflaDemo/streams/*.

B.3 Apache, MySQL, PHP, phpmyadmin

B.3.1 Na Linuxu

Instalace na Linuxu je poměrně snadná. Tady lze nainstalovat všechny tři součásti najednou včetně jejich vzájemné konfigurace. Využije se již zmiňovaný LAMP server. Příkaz sloužící k vyvolání nabídky předpřipravených balíků:

```
sudo tasksel Ot
```

Zde zaklikneme položku *LAMP server* a dáme *Ok*. Během instalace budeme požádáni o *root* heslo do databáze.

/var/www/ - defaultní složka pro html a php soubory.

Dále by jsme si měli nainstalovat phpmyadmin nebo jiného správce pro administraci databáze, to lze provést příkazem:

```
sudo apt-get install phpmyadmin
```

Konfigurační soubor pro phpmyadmin nalezneme v */usr/share/doc/phpmyadmin/dbconfig-common*.

B.3.2 Na Windows

Zde máme na výběr více možností, osobně dávám přednost té nejjednoduší, a to použití XAMPP, což je obdoba LAMP u Linuxu. Instalační balíčky XAMPP jsou distribuovány pro většinu platforem. Lze jej tedy použit i místo LAMP serveru na Linuxu.

Na domovské stránce [14] tohoto projektu si stačí stáhnout distribuci pro windows a nainstalovat jako jakoukoliv jinou aplikaci.

xampp/htdocs - defaultní složka pro html a php soubory.

B.4 ffmpeg

Program ffmpeg je nutný pro správný chod programu *XML2DB.py*.

B.4.1 Na Linuxu

Instlaci ffmpegu provedeme následujícím příkazem:

```
sudo apt-get install ffmpeg
```

B.4.2 Na Windows

Z adresy *http://ffmpeg.arrozcru.org/* si lze stáhnout zkomprimovaný exe soubor pro MS Windows [15]. Ten následně rozbalit do složky s prograOtmem *XML2DB.py*.

B.5 Dokončení

V MySQL vytvoříme databázi podle schématu v kapitole 4.1.2.

Do složky */var/www* (Linux) nebo *xampp/htdocs* (Win) nahrajeme soubory webové aplikace a upravíme přístup k databázi v souboru *db.php*.

Literatura a Zdroje

[1] ASCHER, D., LUTZ, M.: *Naučte se Python*
Praha, Vydavatelství Grada Publishing., 2003. ISBN 80-247-0367-X

[2] ŠKULTÉTY, Rastislav: *JavaScript: Kapesní přehled*
Brno, Vydavatelství Computer Press, a.s., 2005 ISBN 80-251-0884-8

[3] ASLESON, R. – SCHUTTA, N. T.:
Ajax: vytváříme vysoce interaktivní webové aplikace.
Computer Press, a.s., 2006. ISBN 80-251-1285-3

[4] Sphinx manual:
[online] URL:<www.sphinxsearch.com/docs/manual-0.9.9.html>

[5] MySQL home page:
[online] URL:<www.mysql.com/>

[6] Practical Full Text Search in MySQL:
[online] URL:<www.slideshare.net/billkarwin/
practical-full-text-search-with-my-sql>

[7] Newton MediaL home page:
[online] URL:<www.newtonmedia.cz>

[8] ITE FM TUL speechlab:
[online] URL:<www.ite.tul.cz/speechlab/index.php>

[9] XML schémata
[online] URL:<www.kosek.cz/xml/schema/wxs.html>

[10] percona.com Technical Presentations Sphinx full-text search engine:
[online] URL:<www.percona.com/files/presentations/opensql2008_sphinx.pdf>

[11] HTK home page:
[online] URL:<htk.eng.cam.ac.uk/>

[12] Python home page:
[online] URL:<www.python.org/>

[13] Red5 home page:
[online] URL:<osflash.org/red5/>

[14] XAMPP home page:
[online] URL:<www.apachefriends.org/en/xampp.html>

[15] ffmpeg home page:
[online] URL:<www.ffmpeg.org/>

[16] Dana Nejedlová, Marek Volejník: *Transkripce psaného českého textu do fonetické podoby*
[online] URL:<itakura.ite.tul.cz/jan/PMR/transkripce.pdf>

Obsah přiloženého DVD

screenshot/ složka s náhledy jednotlivých prohlížečů

www/ složka se zdrojovými kódy webové aplikace

python/ složka se zdrojovými kódy pomocných skriptů

test/ složka s logy vzniklými při testování

ostatni/ složka obsahující jinam nezařaditelné soubory