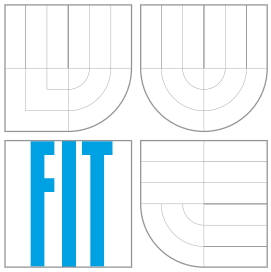
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

PREDNASKY.COM - VYTVOŘENÍ FRAMEWORKU PRO AUTOMATICKÉ ZPRACOVÁNÍ DAT

PREDNASKY.COM - A FRAMEWORK FOR AUTOMATIC DATA PROCESSING

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

JAN KALFUS

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. IGOR SZŐKE, Ph.D.

BRNO 2015

Abstrakt

Tato bakalářská práce řeší návrh a realizaci frameworku v jazyce Python pro automatické zpracování multimediálních dat. Vytvořený framework umožňuje dekomponovat úpravy na jednotlivé atomické úlohy, které mohou být spuštěny v clusteru, a zvládá jejich propojení dohromady včetně automatického řešení závislostí. Funkčnost frameworku byla úspěšně ověřena za pomoci úloh určených ke zpracování videozáznamů přednášek. Nad frameworkem bylo dále vytvořeno webové rozhraní umožňující jeho správu a spuštění zpracování záznamů přednášejícími.

Abstract

This bachelor's thesis presents the design and implementation of a framework for automated multimedia data processing in Python. The created framework enables the user to decompose the editing process into atomic tasks, which can be run in a computer cluster, and handles their interconnection including automatic dependency resolution. The framework's functionality has been successfully tested using tasks to process recordings of lectures. Also, a web interface has been created, which enables the administrator to manage the system and lecturers to submit lecture recordings for processing.

Klíčová slova

multimédia, přednášky, záznamy, framework, automatické zpracování dat, Python, Nette

Keywords

multimedia, lectures, recordings, framework, automatic data processing, Python, Nette

Citace

Jan Kalfus: Prednasky.com - vytvoření frameworku
pro automatické zpracování dat, bakalářská práce, Brno, FIT VUT v Brně, 2015

Prednasky.com - vytvoření frameworku pro automatické zpracování dat

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Igora Szőkeho, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jan Kalfus
16. května 2015

Poděkování

Rád bych poděkoval panu Ing. Igoru Szőkemu, Ph.D., vedoucímu této práce, za jeho rady a připomínky, které mi pomohly při její tvorbě.

Obsah

Kapitola 1

Úvod

Možnost zhlédnout záznamy přednášek představuje jeden ze způsobů, který studentům významně ulehčuje snahu o porozumění probírané látce a umožňuje jim dosáhnout lepších výsledků [1]. Vyučující na Fakultě informačních technologií Vysokého učení technického v Brně mají možnost pořizovat záznamy svých přednášek a dávat je studentům ke stažení prostřednictvím školních video serverů. Proces stahování záznamů a shánění souvisejících materiálů je však pro studenty poměrně nepohodlný. Nabízí se tak alternativa streamování videí ve webovém prohlížeči, která přináší několik výhod. Jsou jimi zejména možnost snadného přehrávání videí na mobilních zařízeních, celkově vyšší pohodlí a potenciál ke zvýšení úrovně porozumění dané látce. Video lze např. rozšířit o slajdy synchronizované s jeho obsahem, o přepis řeči na text, v němž mohou uživatelé vyhledávat, odkazy na web, elektronické materiály, diskuzi k tématu atd. Navíc v případě, že studentovi není jasná pouze určitá pasáž, má možnost ihned zhlédnout pouze potřebný úsek záznamu, aniž by jej musel celý stahovat, což šetří drahocenný čas.

Za tímto účelem vznikl na půdě fakulty projekt webového přehrávače přednášek prednasky.com[1]. Ten zvládá zobrazení slajdů synchronizovaných s obsahem či zobrazení titulků, v současné době však není propojen se školními video servery a obsahuje pouze ukázkové videozáznamy pro demonstraci technologií.

Tato práce si klade za cíl vytvořit systém pro automatizované zpracování pořízených videozáznamů přednášek, jenž umožní tvorbu výstupů potřebných pro použití ve zmíněném webovém přehrávači. Toto téma jsem si zvolil, neboť chci, aby byly výsledky mé práce prakticky využívány. Navíc věřím, že možnost zhlédnout záznamy přednášek online, díky výše uvedeným výhodám, pomůže studentům v jejich studiu.

[1] http://www.prednasky.com

Kapitola 2

Zpracování videozáznamů přednášek

V této kapitole je čtenář seznámen s problematikou ručního a automatizovaného zpracování videozáznamů přednášek.

2.1 Současný stav

V současné době jsou na půdě fakulty pořizovány záznamy přednášek dvojím způsobem – prostřednictvím kamer umístěných v aulách a přenosem videosignálu vysílaným počítačem přednášejícího či vizualizérem. Výstupy putují na video servery, kde dochází k případných ručním úpravám a převodu do značně komprimovaného formátu tak, aby byla zajištěna rozumná velikost záznamu ke stažení.

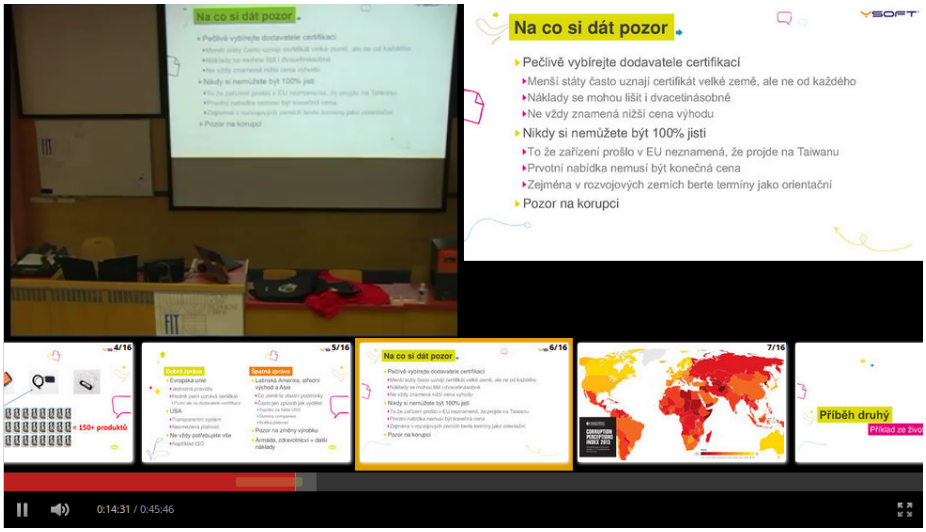
Velikost stažitelných záznamů dosahuje typicky 600 – 800 MB, což i při kvalitním internetovém připojení znamená čekání několik minut na jejich stažení. Záznamy postrádají jakýkoliv popis jejich obsahu a další užitečné informace, které by studentům usnadnily vyhledávání pasáží, jimž v minulosti nerozuměli. Navíc slajdy a jiné studijní materiály, vzhledem k neexistující koncepci zveřejňování materiálů, musejí být složitě hledány na několika místech. Ať už jde o webové stránky jednotlivých předmětů, jenž jsou v drtivé většině případů nepřehledné, nebo v souborech k předmětům v informačním systému, ve kterých se taktéž často není snadné zorientovat.

Řešením těchto problémů by bylo vytvořit systém, jenž by umožňoval zhlédnutí záznamů streamovaně ve webovém přehrávači s potřebnými materiály a informacemi užitečnými k pochopení probírané látky na jednom místě.

Za účelem zkvalitnění záznamů přednášek vznikl projekt Super Lectures[1], jehož autory jsou pan Ing. Igor Szőke, Ph.D. a pan Ing. Josef Žižka. Super Lectures obsahují přehrávač přednášek, který dokáže zobrazit slajdy synchronizované s obsahem videa (viz obr. 2.1), přepis řeči, v němž lze vyhledávat, a další. A právě tento přehrávač (k vidění také na prednasky.com) bude použit v rámci vytvářeného systému k přehrávání záznamů pořízených na fakultě.

Aby však vůbec mohl být použit, je potřeba nejdříve záznamy z fakultních video serverů zpracovat způsobem, jenž umožní získání potřebných metainformací, jako např. časování slajdů či přepis řeči, a propojení výstupů ze zpracování s webovou aplikací. Vzhledem k objemu zdrojových dat a složitosti některých postupů k získání těchto metainformací

[1] http://www.superlectures.com



Obrázek 2.1: Ukázka přehrávače obsahujícího slajdy synchronizované s obsahem videa ze Super Lectures

navíc není možné nadále záznamy zpracovávat ručně. Namísto toho je nutné vytvořit a používat automatizovaný systém.

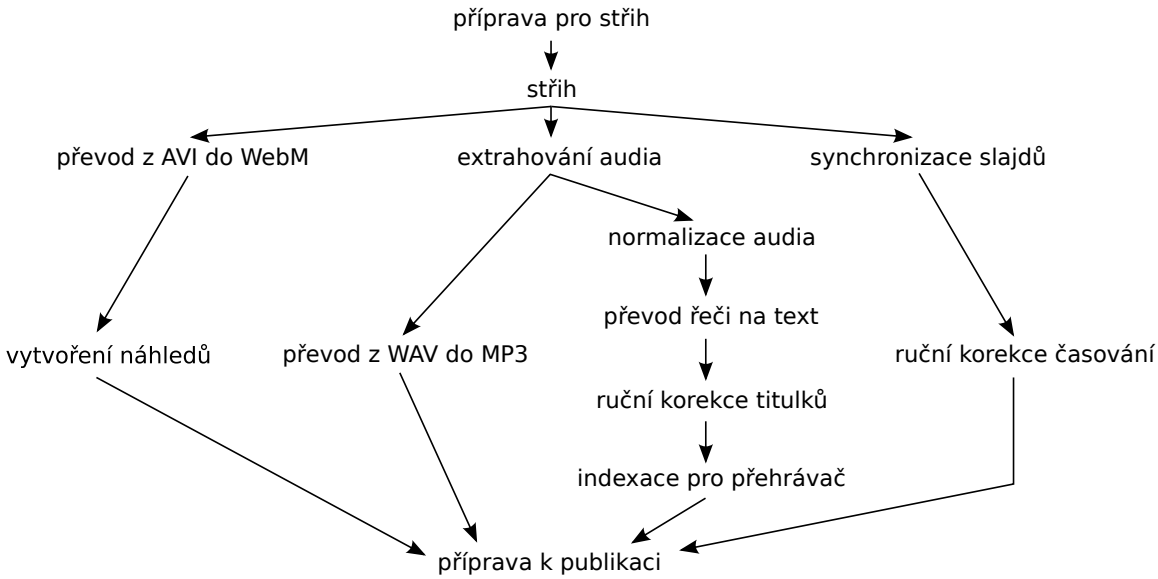
2.2 Automatizované zpracování záznamů

Automatizovaný systém by měl na počátku obdržet zdrojové video a slajdy, které postupnými úpravami přetvoří do podoby použitelné pro zmíněný přehrávač. Jednotlivými úpravami jsou např. převod do jiného formátu, extrahování zvuku a následný přepis řeči na text, nebo vytvoření synchronizace pro slajdy. Některé tyto úpravy na sobě mohou být nezávislé, zatímco některé z nich naopak musejí být aplikovány v přesně stanoveném pořadí. Jde tak vlastně o jakési sekvence atomických úloh transformujících vstupní data na výstupní. Pro lepší představu toho, jak typicky vypadá automatické zpracování záznamů poslouží obrázek 2.2.

Jak také vyplývá z obrázku, ne všechny úpravy mohou probíhat plně automaticky. Střih a ruční úpravy titulků a časování vyžadují interakci s uživatelem. Na druhou stranu, například korekce titulků nemusejí být nutně prováděny vybranými osobami během samotného zpracování. Po zveřejnění videa na webu lze s výhodou dovolit uživatelům, aby mohli u daného záznamu opravit případné nedokonalosti v přepisu řeči. Dojde tím nejen k urychlení zpracování, ale zároveň ke zkvalitnění dat. V úvahu musí být samozřejmě vzata případná možnost zneužití a vytvořena patřičná ochrana proti ní.

U velké části záznamů probíhá zpracování stejným způsobem. Některé záznamy však mohou být upraveny odlišně. Například v případě zveřejnění kompletního záznamu není potřeba střih, zatímco pokud by chtěl přednášející zveřejnit pouze určitý úsek přednášky, mělo by mu to být umožněno. Obecně tak mohou být v některých případech určité kroky vynechány, ve výsledku však dochází ke vzniku relativně malého množství odlišných posloupností atomických úloh.

Vzhledem k množství a délce zpracovávaných záznamů, která obvykle činí 2 – 3 hodiny, je nutno uvažovat také výpočetní náročnost. Konverze videa do jiného formátu nebo přepis řeči na text patří mezi výpočetně velmi náročné operace. Pokud by došlo ke zpracovávání



Obrázek 2.2: Typicky používané úpravy při zpracování záznamů pro Super Lectures přehrávač a jejich návaznost

příliš velkého množství dat najednou, nastalo by nevyhnutelně zahlcení počítače velkým objemem režie při přepínání mezi jednotlivými procesy, což by mělo za následek výrazně delší čas potřebný ke zpracování. Videa tedy musejí být zpracovávány postupně. V ideálním případě by navíc zátěž mohla být distribuována mezi několik počítačů.

Jak vyplývá z výše uvedeného textu, automatizovaný systém musí umožňovat zpracování záznamů pomocí sekvence atomických úloh. Ta se navíc může pro různé záznamy do jisté míry lišit, systém by tedy měl být zároveň široce konfigurovatelný. Dále nesmí chybět vhodné rozložení zátěže. A nelze opomenout ani možnost problémů vyžadujících zásah administrátora, neboť k nim při zřetězeném zpracování multimediálních dat, obzvláště ve větším objemu, může s velkou pravděpodobností dojít. Systém by tak měl dokázat upozornit na případné chyby včetně místa vzniku a usnadnit jejich nápravu.

Kapitola 3

Existující řešení pro zřetězené zpracování dat

Na základě získaných poznatků o automatizovaném zpracování záznamů, uvedených v kapitole 2.2, byla zmapována existující řešení pro zřetězené zpracování dat. Vedoucím práce byl přitom vznesen požadavek na to, aby byl framework dostupný pro programovací jazyky Bash nebo Python a pracoval pod operačními systémy linuxového typu.

Dále došlo k analýze rozpracovaného frameworku pro automatizované zpracování záznamů, dodaného vedoucím práce.

3.1 Existující řešení pro Bash

Hledání existujících řešení pro Bash probíhalo pomocí vyhledávače Google s využitím klíčových slov *bash*, *data processing* a *pipeline*, jejichž použití produkovalo nejvíce relevantní výsledky. Dále byly prohledány webové katalogy jako např. komunitní katalog Awesome[1]. Ten obsahuje komplexní seznamy knihoven, frameworků a nástrojů pro různé programovací jazyky, které jsou spravovány vývojáři s bohatými zkušenostmi v daných oblastech. Mezi jeden ze seznamů patří také Awesome Shell[2], jenž obsahuje řadu hotových řešení pro Bash.

Jak při použití vyhledávače, tak při použití katalogů však bylo hledání neúspěšné. To naznačuje, že systém pro automatizované zpracování multimediálních dat představuje velmi specifický problém. Právě Bash totiž bývá běžně využíván k řetězení různých operací dohromady.

3.2 Existující řešení pro Python

V případě programovacího jazyka Python je situace výrazně lepší. Existuje pro něj několik frameworků umožňujících zřetězení operací při zpracování dat. Následuje popis těch nejzajímavějších.

PaPy

Tento framework nabízí prostředky ke zpracovávání dat v několika krocích. Jednotlivé kroky/úlohy jsou uspořádány mezi sebou na základě požadovaných vstupů a výstupů, jichž

[1] https://github.com/sindresorhus/awesome
[2] https://github.com/alebcay/awesome-shell

může mít každá úloha několik. Díky tomu lze zpracování provádět paralelně a to jak na vícejádrovém procesoru, tak distribuovaně. Framework zajišťuje vytváření procesů, dovoluje omezit jejich počet, zvládá detekci chybových stavů či zaznamenávání aktuálního stavu.

PaPy[3] však bohužel není stavěn pro zpracování binárních dat, což videozáznamy představují, ale pouze textových, přičemž počítá s postupným zpracováním souborů iterováním nad jejich obsahem. Využití tak najde zejména při provádění komplexních matematických a fyzikálních výpočtů.

Ruffus

Ruffus[4] představuje další řešení pro zřetězené zpracování dat. Tato knihovna umožňuje rozdělit proces na jednotlivé úlohy a nastavit, jakým způsobem mají navazovat. Interakce s okolím je řešena prostřednictvím vstupního a výstupního souboru dané úlohy. Způsob práce se soubory má v režii kompletně programátor, umožňuje tedy pracovat i s binárními daty.

Nevýhodou je, že knihovna nedovoluje použití více vstupních nebo výstupních souborů pro jednu úlohu. A dále, neumožňuje paralelizovat operace nad jedněmi vstupními daty, neboť úlohy musejí být propojeny sekvenčně.

Luigi

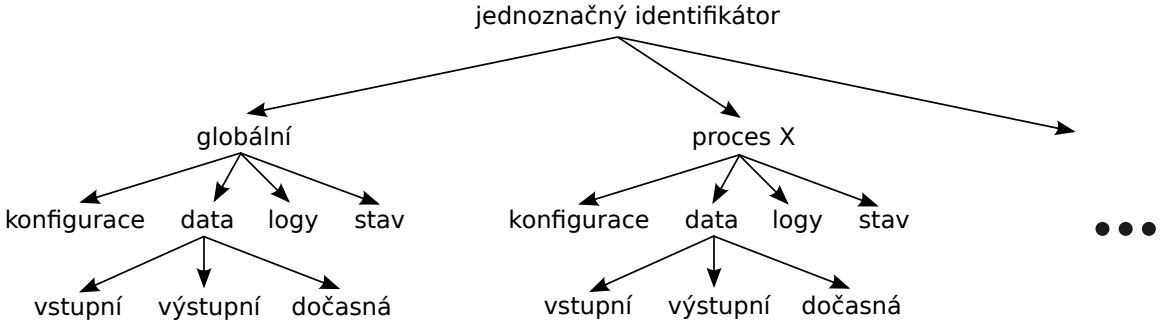
Jako nejzajímavější framework se ukázal být Luigi[5]. Umožňuje rozdělení procesu zpracování na jednotlivé úlohy a zajišťuje jejich vzájemné propojení. Každé úloze lze nastavit závislosti, které musejí být splněny, aby mohla být spuštěna, a jaké jsou její očekávané výstupy (např. jména souborů). Počet závislostí ani výstupů není omezen. Úlohy mohou běžet paralelně, přičemž uživatel může omezit počet současně běžících operací. Framework zvládá automatické generování grafů závislostí, podávání informací o stavu zpracování formou HTML stránek či odesílání emailů s popisem chyby v případě neúspěchu.

Vzhledem k tomu, že framework na první pohled splňoval většinu požadavků, byl v něm vytvořen jednoduchý prototyp a vyzkoušeny jeho vlastnosti. Při tom se ukázaly dvě negativa, která jeho použití ke zpracování záznamů přednášek prakticky znemožňují.

Za prvé, framework neobsahuje nativní podporu pro konfigurační soubory. Snaha o implementaci podpory konfiguračních souborů vedla k významnému nárůstu počtu řádků zdrojového kódu a jeho značné nepřehlednosti. Na vině je nutnost rekurzivní definice závislostí – poslední úloha vyžaduje ke svému spuštění předchozí, ta zase předchozí atd., což s rostoucím počtem variant vede k neúměrné režii.

Závislosti navíc nelze měnit za běhu, s čímž souvisí druhý problém. Zpracování probíhá dávkově. Na začátku se stanoví požadavky a ty jsou obsluhovány. Data z poslucháren však přicházejí postupně a v případě, kdy se zpracovává netriviální úloha nad posledními daty z dávky – např. převod videa, jenž běžně trvá desítky minut – zůstává velká část výpočetního výkonu nevyužita. Framework totiž zabraňuje spuštění zpracování další dávky a v prvé řadě nelze tento stav ani nijak jednoduše detekovat. V případě velkého počtu dávek s relativně malým množstvím dat ke zpracování (případ záznamů přednášek) by tak docházelo k výraznému prodloužení doby potřebné ke zpracování a v extrémních případech by paralelní zpracování bylo zcela nahrazeno sekvenčním.

[3] https://pypi.python.org/pypi/papy/
[4] http://www.ruffus.org.uk/
[5] https://github.com/spotify/luigi



Obrázek 3.1: Struktura tokenu. Za povšimnutí stojí, že token obsahuje jak informace sdílené mezi všemi procesy, tak informace potřebné pouze pro konkrétní procesy.

Framework je tak vhodný spíše ke zpracování neměnných úloh nad velkým množstvím vstupních dat, kdy obsluha jedné dávky trvá dny až týdny.

3.3 Dodaný framework

Vedoucím práce byl dodán framework používaný pro automatické zpracování přednášek na Super Lectures. Jedná se o sadu skriptů v jazyce Bash, která usnadňuje vytváření jednotlivých úloh transformujících zdrojová data do podoby použitelné pro web. Staví na poznatcích z praxe, není však příliš robustní, jak bude popsáno dále.

Základní myšlenkou je oddělení všech částí zpracování od sebe. Pro každou atomickou úlohu se vytvoří skript, tzv. proces, který provede požadované úpravy. Framework pak zajistí propojení jednotlivých procesů dohromady dle nastavení uživatele. Díky tomu lze uspořádání procesů libovolně měnit a případně některé z nich vynechat.

Jednoznačná identifikace vstupních dat a jejich postupné zpracování je řešeno pomocí systému tzv. tokenů. Prakticky jde o adresáře s jednoznačně určujícím názvem obsahující veškeré potřebné informace k provedení úprav (vstupní data, konfiguraci pro procesy) a po provedení jednotlivých procesů také jejich výstupní data (viz obr. 3.1), dále logy a informace o stavu. Název může tvořit např. kombinace data a času vytvoření, id uživatele a náhodně generovaná posloupnost znaků tak, aby nedocházelo ke konfliktům.

Každý z procesů má k dispozici čtyři fronty[6] tokenů – *čekající*, *běžící*, *dokončené* a *s chybou*. Na počátku zpracování je token umístěn do fronty čekajících u procesů, které nemají žádné závislosti. Procesy ve stanoveném intervalu kontrolují obsah fronty, vyberou z ní první token a pokusí se jej zpracovat. V případě úspěchu jej přesunou do fronty dokončených a dále jej umístí také do front *čekající* u všech následujících procesů. V opačném případě se přesune do fronty *s chybou* a zpracování dále nepokračuje. Tokeny ve vstupní frontě přitom mohou být řazeny podle jejich celých identifikátorů nebo pouze částí, takže proces může upřednostnit např. zpracování tokenu pro uživatele majícího vyšší prioritu. Smyčku zpracování tokenu procesy přehledně zobrazuje pseudokód 3.1.

Z hlediska implementace tvoří fronty u procesů složky pojmenované dle jejich účelu a tokeny jsou v nich reprezentovány prázdnými soubory nesoucími názvy dle jejich jednoznačných identifikátorů. Každý z procesů kromě toho dále obsahuje složky pro logy, konfiguraci a pomocné skripty.

[6] z hlediska terminologie není zde ani nikde dále v textu výrazem fronta myšlena stejnojmenná datová struktura, ale řaditelné seskupení tokenů

Algorithm 3.1 Smyčka zpracovávání tokenů procesem

```
if fronta čekajících je prázdná then
    uspi se na nastavený interval;
else
    seřaď frontu podle nastaveného pravidla;
    while fronta obsahuje nezkontrolované tokeny do
        if další token ve frontě obsahuje potřebná vstupní data then
            umísti token do fronty běžících;
            proveď zpracování vstupních dat tokenu;
            if zpracování skončilo úspěšně then
                přesuň token do fronty dokončených;
                umísti token do front čekajících u všech následujících procesů;
            else
                přesuň token do fronty s chybou;
            end if
            ukonči kontrolu fronty;
        end if
    end while
    if neproběhlo žádné zpracování then
        uspi se na nastavený interval;
    end if
end if
```

Jak je vidno, framework staví na poměrně obecných konceptech, které jsou použitelné nejen při zpracování záznamů přednášek, ale i jiných multimediálních dat. Snaha o uplatnění jednoduchosti také při implementaci však s sebou nese jistá negativa, která většinou souvisí i se zvoleným programovacím jazykem Bash.

Za největší problém osobně považuji to, že Bash prakticky neumožňuje tvořit strukturovaný zdrojový kód. S přibývajícími řádky se tak program postupně stává čím dál více neudržovatelný. Vzhledem k absenci pokročilejších konstrukcí a jakékoliv základní knihovny funkcí navíc nutí k používání programátorsky nečistých technik a k závislosti na řadě linuxových nástrojů.

V samotném frameworku se nedostatky Bashe projevují několika způsoby. Jeho autor se snažil o jistou strukturovanost rozdělováním úseků kódu či funkcí do zvláštních souborů. Řada z nich je však na sobě závislá a ve výsledku stejně dochází pouze ke spojování skriptů dohromady pomocí příkazu `source`[7]. Při snaze o pochopení činnosti skriptů tak lze narazit např. na použití funkcí, které jsou definovány neznámo kde a není jasné, co provádějí. Ve výsledku se tak kód stává ještě více nečitelným [2].

Vzhledem k absenci mechanismu výjimek jsou kontrolovány na chyby návratové kódy téměř všech prováděných příkazů. To vede k dalšímu zatemňování zdrojového kódu, ale především stačí jediná opomenutá kontrola a zpracovávaná data se dostanou do nekonzistentního stavu, přičemž provádění skriptu bude dále pokračovat – vše bez jakéhokoliv upozornění uživatele.

Tento problém může být částečně řešen vložením příkazu `set -e` na začátek skriptů, při jehož použití interpret Bashe zajistí, že vrácení nenulového (typicky chybového) kódu u jakéhokoliv z nezřetězených příkazů způsobí ukončení celého skriptu. Tato volba nicméně

[7] příkaz source způsobí provedení těla zvoleného skriptu tak, jako by jej obsahoval právě prováděný skript

nedokáže řešit všechny problémové situace[8] a může ukončit činnost i v legitimních případech, kdy není očekáván nulový návratový kód.

U frameworku dále chybí podpora konfiguračních souborů formátu určeného pro výměnu dat (XML, YAML apod.). Namísto toho je konfigurace umístěna ve skriptech, kde se nastaví potřebné proměnné. Skripty pak framework načítá příkazem `source`, což představuje bezpečnostní riziko. Stačí do konfiguračního souboru umístit škodlivý příkaz, např. `rm -rf /` a ten se při jeho načtení provede.

Z hlediska konfigurace považuji za nešťastný také způsob nastavení propojení procesů. Namísto jediného, přehledného konfiguračního souboru obsahujícího tuto informaci existuje soubor pro každý proces. To pravděpodobně souvisí s další slabinou frameworku, absencí automatického řešení závislostí (angl. *dependency resolution*). Procesy jsou namísto toho samy zodpovědné za to, aby jejich následníci obdrželi potřebné vstupní soubory.

[8] viz manuálové stránky `bash(1)`, sekce SHELL BUILTIN COMMANDS; příklad neošetřené situace na http://www.mail-archive.com/debian-bugs-dist@lists.debian.org/msg473314.html

Kapitola 4

Vytvoření frameworku pro automatické zpracování multimediálních dat

Tato kapitola popisuje návrh a implementaci frameworku pro automatické zpracování záznamů přednášek na základě ústní specifikace požadavků vedoucího práce a získaných poznatků, uvedených v kapitolách 2 a 3.

4.1 Specifikace požadavků a jejich analýza

Požadavky lze shrnout do několika hlavních bodů. Framework by měl umožňovat zejména následující:

- zpracování záznamů pomocí sekvence atomických úloh,
- konfiguraci propojení úloh,
- interakci s uživatelem (např. při střihu),
- zjištění stavu zpracování,
- ukončení zpracování pro vybraný záznam administrátorem,
- spuštění samostatně nebo s napojením na webovou aplikaci,
- spuštění více instancí procesů zpracovávajících data,
- spuštění zpracování v clusteru.

Volba implementačního jazyka byla omezena na Bash a Python, přičemž framework musí pracovat pod operačním systémem linuxového typu.

Z požadavků a zkušeností získaných analýzou stávajících řešení pro zřetězené zpracování dat vyplývá, že bude dobré navrhnout framework tak, aby co možná nejvíce izoloval jednotlivé atomické úlohy od sebe a zároveň usnadnil jejich vytváření. Jeho administrátor se díky tomu bude moci soustředit na samotné zpracování videozáznamů, aniž by musel sám řešit propojení úloh dohromady. Způsob propojení úloh a jejich parametry přitom musejí být nastavitelné a to pro každý zpracovávaný záznam.

Framework dále musí umožňovat interakci systému pro automatické zpracování s uživatelem, aby mohl provést např. ruční korekci titulků či střih prostřednictvím rozhraní ve webovém prohlížeči. Obecně jde o krok zpracování, který provádí jiný systém a je tedy potřeba dovolit jeho propojení s ostatními úlohami, jež nesmí svou činností omezit.

Od vytvořeného systému je také požadováno, aby byl schopen pracovat samostatně a šlo jej využít ke zpracování libovolných dat, zároveň ale aby poskytoval webové rozhraní pro jeho snadnou správu. Bude tedy potřeba navrhnout propojení s databází. To v kombinaci s webovou aplikací zároveň usnadní administrátorovi získat rychle přehled o stavu zpracování a v případě potřeby zasáhnout.

Zpracování multimediálních dat je obvykle činnost náročná na výkon počítače. Lze ji však poměrně dobře paralelizovat. Pro plné využití výpočetního výkonu proto framework musí zvládat paralelní běh vzájemně nezávislých úloh. Vhod dále přijde také paralelní běh více instancí procesů provádějících úpravy. To dovolí využít všechna jádra procesoru i v případě, kdy program provádějící daný krok zpracování není vytvořen pro běh na více jádrech.

Vzhledem k velkému objemu zdrojových dat v případě záznamů přednášek se nicméně očekává spíše běh na více počítačích zároveň, v clusteru. Framework tedy bude nutné připravit na spolupráci se systémem pro dávkové zpracování úloh. Na fakultě se konkrétně používá Sun Grid Engine[1], nicméně framework by měl být vytvořen dostatečně flexibilně na to, aby bylo snadné případné přidání podpory i pro jiný systém.

4.2 Návrh

Návrh byl částečně inspirován dodaným frameworkem, popsaným v kapitole 3.3. Ten totiž vychází z praktických poznatků o automatickém zpracování videozáznamů, přičemž použité koncepty jsou velmi obecné a jejich zdokonalením lze vytvořit robustní řešení pro zřetězené zpracování (nejen) multimediálních dat.

Identifikace dat a jejich postupné zpracování

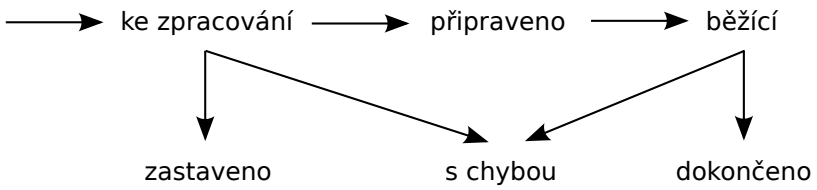
Podobnými koncepty jako u dodaného frameworku jsou zejména systém tokenů a front. Tokeny poskytují datům jednoznačnou identifikaci a usnadňují orientaci v nich, neboť zajišťují umístění logicky souvisejících dat vedle sebe. Na rozdíl od dodaného frameworku však mnou navržené tokeny neobsahují složky pro sdílená data, logy a stav. Předávání dat by měl, z hlediska čistoty řešení, mít v režii kompletně framework. Společné logy a stav pro procesy zpracovávající token pak postrádají smysl, jsou totiž vždy spjaty s některým z procesů.

Koncept front, umožňujících postupné zpracování dat, nezůstal taktéž nezměněn. Fronta *čekající* byla nahrazena dvěma – *ke zpracování* a *připraveno* – a přibyla fronta *zastaveno*. *Ke zpracování* představuje vstupní frontu tokenů, jež má daný proces zpracovat a *připraveno* obsahuje tokeny, které již obsahují i potřebná vstupní data[2] a jejichž zpracování tedy může být okamžitě zahájeno. Rozdělení jsem se rozhodl provést, protože při zpracování v clusteru systém pro dávkové zpracování úloh nejprve zařadí požadavek do vlastní fronty a obslouží jej teprve ve chvíli, kdy má k dispozici dostatek systémových prostředků. Fronta *připraveno* tak lépe vystihuje tento stav a k přesunutí do *běžící* dojde až při skutečném

[1] později Oracle Grid Engine, dnes již není výrobcem podporován

[2] token se vždy může nacházet v jednom čase maximálně v jedné ze všech typů front u daného procesu

zahájení zpracování dat. Fronta *zastaveno* pak obsahuje tokeny, jejichž zpracování ukončil administrátor. Pohyb tokenů mezi všemi typy front přehledně zobrazuje obr. 4.1.



Obrázek 4.1: Možnost pohybu tokenu mezi frontami.

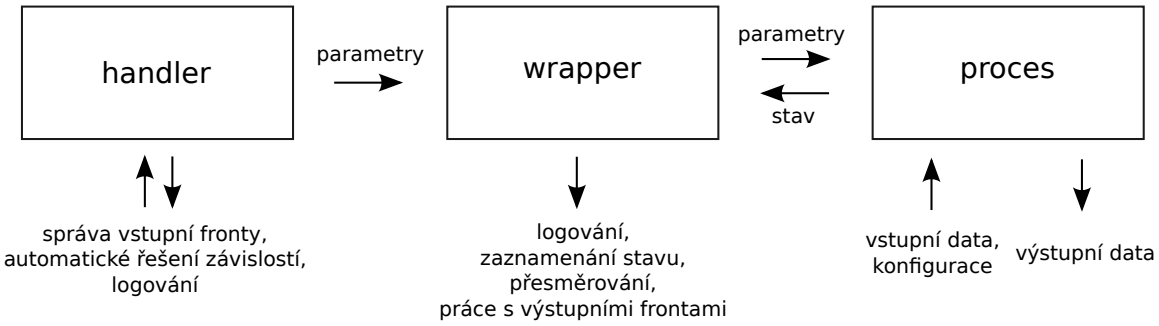
Izolace procesů a propojení úloh

Jak již bylo zmíněno dříve, jedním z hlavních cílů je izolovat procesy od sebe a jejich propojení přenechat plně frameworku. Na základě tohoto požadavku bylo navrhnuto rozdělení každé z úloh na tři části. Jde o handler, wrapper a samotný proces, viz obr. 4.2. Handler představuje skript, který ve smyčce kontroluje vstupní frontu dané úlohy a automaticky řeší závislosti. Wrapper poskytuje rozhraní pro proces a předává mu potřebné informace. Proces vytváří výstupní data na základě vstupů a konfigurace.

Činnost handleru výstižně popisuje pseudokód 4.1. Před pokračováním v textu je silně doporučeno jej nejprve prostudovat.

To, že všechny předchozí procesy úspěšně zpracovaly token pozná handler jeho přítomností v příslušných frontách *dokončeno*. Automatické řešení závislostí probíhá následovně. Na základě konfiguračního souboru pro propojení procesů (popsán dále) se stanoví, které vstupní soubory daný proces ke své činnosti požaduje. Odpovídající soubory jsou hledány mezi výstupy předcházejících procesů a v případě shody dojde k přesměrování souboru z výstupu předchozího procesu na vstup aktuálního.

Wrapper obaluje proces zpracovávající data. Nejdříve přesune zpracovávaný token z fronty *připraveno* do fronty *běžící*. Poté spustí samotný proces a na vstupu mu předá umístění konfiguračního souboru a informace o tom, kde se nacházejí vstupní data a kam má zapisovat ta výstupní. Dále kontroluje stav procesu a zaznamenává jej. Po ukončení zpracování dat procesem přesune token buď do fronty *dokončeno* nebo *s chybou*, podle toho jak operace dopadla. V případě přesunutí do dokončených zároveň přidá token do front *ke zpracování* u všech následujících úloh.



Obrázek 4.2: Znázornění datových toků mezi frameworkem a procesem provádějícím zpracování dat.

Algorithm 4.1 Smyčka handleru

```
zamkni frontu čekajících;
if fronta čekajících je prázdná then
    odemkni frontu čekajících;
    uspi se na nastavený interval;
else
    seřaď frontu čekajících podle nastaveného pravidla;
    while fronta obsahuje nezkontrolované tokeny do
        vyber další token;
        if zpracování tokenu bylo ukončeno administrátorem then
            přesuň token do fronty ukončených;
        else
            if všechny předchozí procesy úspěšně zpracovaly token then
                pokus se vyřešit závislosti;
                if nelze vyřešit závislosti then
                    přesuň token do fronty s chybou;
                else
                    přesuň token do fronty připravených;
                    if má se použít systém pro dávkové zpracování then
                        vytvoř úlohu pro wrapper v systému dávkového zpracování;
                    else
                        odemkni frontu čekajících;
                        zpracuj token pomocí wrapperu;
                        ukonči kontrolu fronty čekajících;
                    end if
                end if
            else if kterýkoliv z předchozích procesů skončil s chybou then
                přesuň token do fronty s chybou;
            end if
        end if
    end while
    odemkni frontu čekajících;
    if neproběhlo žádné zpracování then
        uspi se na nastavený interval;
    end if
end if
```

Rozdělením režie frameworku mezi handler a wrapper je umožněno jak blokující, tak neblokující zpracování. V případě, že se používá systém pro dávkové zpracování (SDZ) úloh v clusteru, přidá handler úlohu ke spuštění wrapperu do SDZ. Ke spuštění wrapperu dojde asynchronně při uvolnění dostatečného množství systémových prostředků. Handler proto po vytvoření úlohy v SDZ ihned pokračuje ve zpracovávání vstupní fronty. Jde tedy o neblokující zpracování. Pokud se SDZ nevyužívá, spouští handler wrapper okamžitě a blokujícím způsobem, což znemožňuje pokračování v jeho činnosti až do dokončení zpracování. Díky tomu může každý handler přímo spustit pouze jednu instanci wrapperu a nedojde k zahlcení počítače při velkém množství tokenů určených ke zpracování.

Aby bylo ale zároveň možné administrátorem spustit více instancí handleru (a tím pádem wrapperu) pro jednu úlohu, musí být zajištěno vzájemné vyloučení přístupu ke vstupní frontě. Handler si proto vstupní frontu na počátku smyčky zamkne a případné další instance čekají na její odemknutí. V případě blokujícího spuštění wrapperu nesmí být těsně před jeho spuštěním opomenuto odemknutí této fronty, aby mohly případné další instance taktéž frontu zpracovávat. Při použití neblokujícího zpracování postrádá využití více instancí handleru smysl a stačí proto, aby odemknul frontu až ve chvíli, kdy se má uspat.

Od frameworku se dále mj. požaduje, aby dokázal zastavit zpracování konkrétního záznamu na základě příkazu od administrátora. Jako jednoduché a účinné řešení se nabízí vytvoření příznaku u tokenu, jehož existenci bude handler při procházení vstupní fronty kontrolovat a případně nezahájí zpracování.

Konfigurace

Framework musí dovolovat konfiguraci propojení procesů pro každý záznam. Pro přehlednost tak bude dobré umožnit vytvoření konfiguračního souboru specifického pro každý token, obsahující popis propojení. U každé úlohy přitom budou definovány požadované vstupní soubory tak, aby mohl handler automaticky vyřešit závislosti.

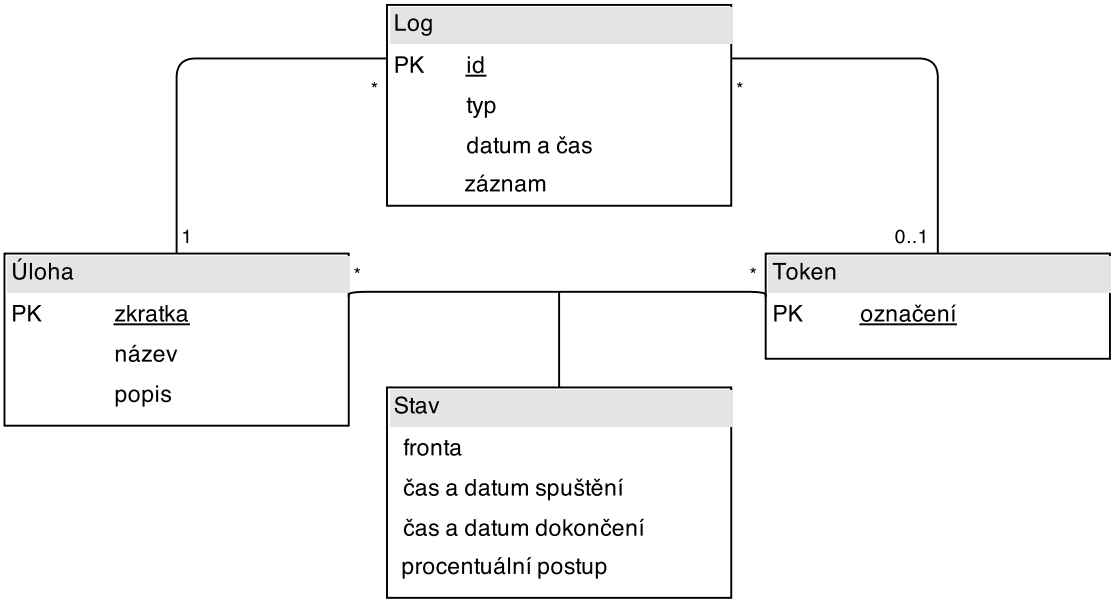
Každý z procesů má obsahovat nastavení, s jakým zpracuje data. Musí ale také akceptovat změnu některých parametrů specifikovanou pro konkrétní záznam. Vedle výchozí konfigurace bude tedy potřeba načítat další, specifickou pro kombinaci tokenu a procesu, a obě z nich poté spojit dohromady.

Vedoucím práce byl také vznesen požadavek na to, aby bylo možné nastavit spuštění v clusteru a odstranění výstupních dat společně pro všechny úlohy[3], s možností změny u konkrétních úloh, tokenů či kombinace tokenu a úlohy. Handler tak musí být připraven na to, aby uměl načíst tyto čtyři konfigurační soubory, spojit je dohromady a případně pozměnit své výchozí chování.

Napojení na databázi

Od frameworku se očekává, že zvládne pracovat samostatně, ale zároveň nad ním bude možné vytvořit vrstvu webového administračního rozhraní. Navržené koncepty umožňují poměrně snadnou manipulaci zvenčí, nicméně pro ukládání informací o stavu je vhodnější namísto textových souborů používat databázi, aby jejich získání bylo co nejsnazší. Z toho vychází datový model reprezentovaný ER diagramem na obr. 4.3, při jehož tvorbě bylo bráno v potaz jeho možné budoucí rozšíření o další entity a vazby v souvislosti se vznikem webového rozhraní pro konkrétní projekt.

[3] odstranění výstupních dat má smysl ve chvíli, kdy je v procesu zpracovávajícím data nebo datech samotných chyba, tu administrátor odstraní a chce spustit danou část zpracování znovu



Obrázek 4.3: Návrh informací ukládaných do databáze.

4.3 Implementace frameworku

Prvním krokem při implementaci bylo zvolit programovací jazyk – Bash nebo Python. Python dostal jednoznačně přednost. Na rozdíl od Bashe disponuje bohatou standardní knihovnou, má velkou vyjadřovací sílu, umožňuje tvorbu strukturovaného kódu, podporuje více programovacích paradigmat včetně objektově orientovaného přístupu a nechybí moderní programovací techniky zlepšující přehlednost kódu jako mechanismus výjimek.

Podstatně obtížnější volbu představoval výběr verze Pythonu. V současné době jsou podporovány dvě, vzájemně nekompatibilní, vývojové větve – Python 2.x a Python 3.x. Volba mezi nimi bývá předmětem řady internetových diskuzí a věnuje se jí i oficiální Wiki[4]. Zatímco Python 3 představuje současnost a budoucnost jazyka, vývoji Pythonu 2 již není věnována prakticky žádná pozornost, maximálně dochází k přenášení některých novinek z Pythonu 3 do starší verze. Nicméně Python 2 využívá dodnes řada projektů a existuje pro něj velké množství knihoven třetích stran, které zatím nebyly učiněny kompatibilními s novou verzí.

Po uvážení všech výhod a nevýhod padla nakonec volba na Python 3, jelikož představuje budoucnost jazyka a zároveň není potřeba využít žádných knihoven třetích stran nekompatibilních s touto verzí. Jedinou použitou knihovnou třetí strany je MySQL Connector[5], která poskytuje prostředky pro snadnou manipulaci s databázovým systémem MySQL.

Při implementaci jsem se rozhodl použít objektově orientovaný přístup všude, kde to mělo smysl. Nejprve byly implementovány základní moduly, u nichž byl kladen důraz na znovupoužitelnost a následně došlo na implementaci samotného handleru a wrapperu využívající tyto moduly.

[4] https://wiki.python.org/moin/Python2orPython3
[5] http://dev.mysql.com/downloads/connector/python/

Fronty a tokeny

U front a tokenů bylo nutné zohlednit, že úprava záznamů přednášek může v některých případech (např. střih) vyžadovat vstupy od uživatele za běhu. Pro co největší pohodlí uživatele se přitom očekává použití webové aplikace, která bude upravovaná data vizualizovat a získá vstupy od uživatele, jež předá dále. V rámci systému pro automatické zpracování záznamů může být taková aplikace teoreticky reprezentována procesem, nicméně to by znemožnilo provádět úpravy v libovolném pořadí, asynchronně. Při uvážení toho, jak byl systém navržen, stačí umožnit, aby aplikace měla k dispozici své vlastní fronty a vytvořila data potřebná pro následující úlohy.

Fronty i tokeny tak musejí být nezávislé na použitém programovacím jazyce a dostupné z okolí systému. To umožňuje reprezentace pomocí složek, podobně jako u dodaného frameworku. Tokeny ve frontách pak zastupují prázdné soubory s názvem odpovídajícím identifikátoru tokenu. Webové aplikaci pro střih díky tomu stačí monitorovat obsah příslušné složky *ke zpracování*, nabídnout uživateli zadání vstupů, vytvořit v tokenu příslušná výstupní data, přesunout token ze složky *ke zpracování* do *dokončeno* a přidat token do vstupních front následujících úloh.

Pro zajištění vzájemného vyloučení při přístupu k frontě *ke zpracování* byla použita metoda `flock` z modulu `fcntl`[6], poskytující rozhraní ke stejnojmennému unixovému systémovému volání. Implementace zamykání složky se nachází ve třídě `DirectoryLocker` v modulu `locker` a umožňuje použít jak blokující, tak neblokující zámek.

Abstrakci práce s frontou poskytuje třída `Queue` v modulu `queue_manager`. Nabízí zejména vybírání položek z fronty, vkládání nových do ní a iterování nad jejím obsahem, přičemž výchozí metodu řazení tokenů dle názvu lze za běhu nahradit libovolnou vlastní. Přesouvání položek mezi frontami pak umožňuje třída `QueueMove` v modulu `process_helper`, která je již šitá na míru tomuto frameworku. Zvládá totiž nejen přesouvání mezi frontami na disku, ale také v databázi.

Izolace procesů a konfigurace

Vytvořený framework počítá s možností použít jako proces skript vytvořený jak v Pythonu, tak v Bashi. V obou případech zajišťuje předání potřebných parametrů (cesty ke vstupním, výstupním složkám, konfiguraci a další), logování a zaznamenávání stavu zpracování. Standardní chybový výstup spuštěného skriptu je přesměrován a očekává se na něm procentuální vyjádření postupu, kdy jednotlivé hodnoty odděluje znak nového řádku. Zprávy zapsané na standardní chybový výstup zase putují do příslušného logu. Proces má tak jasně definované rozhraní a přístup pouze k poskytnutým datům. Implementaci lze nalézt ve třídě `ExecutionWrapper` v modulu `wrapper`.

Framework podporuje konfiguraci pomocí XML souborů. XML formát dnes představuje spolu s formátem JSON standard pro výměnu dat. Oproti JSONu sice vyžaduje více prostoru k uložení stejné informace, pro člověka je však mnohem čitelnější.

Procesům lze předat hodnoty všech základních datových typů obsažených v jazyce Python, včetně seznamu a slovníku. Strukturu tohoto konfiguračního souboru popisuje v jazyce DTD příloha A. Procesy si tento konfigurační soubor na základě předané cesty a třídy `XMLLoader`, poskytované frameworkem, načítají samy. Metoda `load` kontroluje specifikované datové typy a pokouší se hodnoty dle toho naparsovat.

[6] https://docs.python.org/3.4/library/fcntl.html

Aby mohlo být podporováno spouštění skriptů vytvořených v Bashi, bylo nutné navrhnout formát konfiguračního souboru také pro něj. Jelikož Bash nerozlišuje datové typy[7], postrádá XML formát s uváděním datových typů smysl. Pro jednoduchost jsem se uchýlil k použití prostého CSV, kdy řádky obsahují dvojice proměnná – hodnota. Před spuštěním procesu vytvořeném v Bashi mu framework nastaví prostředí dle této konfigurace.

Další konfigurační soubor sdílejí všechny úlohy v rámci tokenu. Obsahuje definici způsobu propojení úloh mezi sebou a jimi vyžadované vstupy, což se mj. využívá při automatickém řešení závislostí. Jeho struktura je popsána v příloze B.

Framework zvládá nejen načítání konfiguračních souborů, ale také jejich spojování. U procesů se počítá s dvojicí konfiguračních souborů, kde první obsahuje definici výchozího nastavení a druhý, obsažený v tokenu, obsahuje nastavení pro daný token. Chování handleru pak může být ovlivněno až čtveřicí konfiguračních souborů, jak bylo specifikováno v návrhu. K jejich spojování slouží metoda `merge` ve třídě `ConfigDictionary`.

Logování a zaznamenání postupu

Při zápisu postupu nebo záznamu do logu existuje, vzhledem k požadavkům, možnost použít jako úložiště textový soubor nebo databázi. Variabilitu umožňuje v obou případech společné rozhraní `IProgressWriter`, respektive `ILogger`. Vytvoření rozhraní sice vzhledem k tzv. duck-typingu[8] v Pythonu není potřeba, avšak snižuje riziko chyb. Aplikováním techniky dependency inversion[9] – vložením instance třídy zapisující do souboru nebo databáze, dle nastavení – pak může být snadno změněna metoda zápisu.

Jak již bylo naznačeno výše, jako databázový systém byl zvolen MySQL. K jeho použití jsem se rozhodl, neboť jde v současné době o nejrozšířenější volně dostupnou[10] technologii tohoto typu. Komunikaci s databází zastřešuje modul `mysql` a v případě potřeby použití jiného databázové systému jej pouze stačí nahradit za jiný, aniž by muselo být zasahováno do dalšího kódu.

Handler a wrapper

S pomocí výše zmíněných základních modulů a dalších pomocných byly vytvořeny skripty `process_handler` a `process_wrapper`. Jejich chování bylo implementováno dle návrhu v kapitole 4.2.

Za zmínku stojí podpora pro systém dávkového zpracování dat Sun Grid Engine. Vyžaduje totiž, aby byl jako úloha předán shell skript. Framework je však vytvořen v Pythonu, `process_handler` proto vytvoří dočasný shell skript obsahující jediný příkaz, jenž spustí `process_wrapper`, čímž elegantně obejde toto omezení. Zpracování pak již probíhá obvyklým způsobem.

4.4 Implementace úloh

Vedoucím práce byl dodán skript používaný k úpravě záznamů přednášek v jazyce Bash. Cílem bylo identifikovat v něm jednotlivé atomické úlohy, oddělit je a implementovat s pomocí vytvořeného frameworku v jazyce Python.

[7] http://tldp.org/LDP/abs/html/untyped.html
[8] https://docs.python.org/3/glossary.html#term-duck-typing
[9] http://en.wikipedia.org/wiki/Dependency_inversion_principle
[10] http://db-engines.com/en/ranking

Po rozdělení vznikly následující úlohy, jejichž implementace je součástí přiloženého CD:

- převod videa z formátu AVI do formátu WebM,
- extrahování audia do formátu WAV,
- normalizace audia,
- převod audia ze sterea na mono,
- převod audia z formátu WAV do MP3,
- vytvoření náhledů videa.

Při provádění úprav jsou využívány volně šiřitelné nástroje FFmpeg[11] a Sound eXchange[12] (SoX). Framework byl zároveň rozšířen o pomocné třídy, dostupné k použití procesy, které usnadňují zjištění stavu zpracování při použití těchto nástrojů. Procesy tak mohou snadno oznamovat svůj procentuální postup, jenž je v reálném čase k dispozici uživateli.

4.5 Testování funkcionality

Pro snadno automaticky testovatelné součásti frameworku – práci s frontami a načítání konfiguračních souborů – byly vytvořeny unit testy, které ověřují jejich správnou funkčnost. Během vývoje byl dále kladen důraz na snadné odhalení případných chyb. Veškeré neodchycené výjimky jsou proto zaznamenány do logů včetně zásobníku volání a řádku kódu, který způsobil problém. Logovány jsou kromě toho také průběžné informační zprávy, aby administrátor v případě selhání zpracování záznamu procesem mohl rychle identifikovat zdroj problému.

Funkčnost řešení byla ověřována s pomocí vytvořených úloh nad vzorkem záznamů z video serverů fakulty pod operačním systémem Xubuntu 14.04 a verzí 3.4.0 interpretu programovacího jazyka Python. Během testování bylo odhaleno několik chyb, jejichž nalezení se ukázalo býti díky použitým technikám bezproblémové. Chyby byly opraveny a řešení je tak plně funkční.

Kompatibilita se systémem pro dávkové zpracování úloh Sun Grid Engine nemohla být plně ověřena. Tento systém již totiž výrobce několik let nepodporuje a není ke stažení. Jeho místo zaujaly dva projekty, které využívají původní otevřený kód. Jde o Open Grid Scheduler[13] a Son of Grid Engine[14]. Druhý z nich je stále aktivně vyvíjen a nabízí rozhraní plně zpětně kompatibilní s původním systémem. Rozhodl jsem se tedy framework otestovat na něm. Při jeho použití nebyly zjištěny žádné problémy a dá se tedy předpokládat, že framework bude bezproblémově spolupracovat i s technologií Sun Grid Engine.

Testována byla jak správná funkčnost systému, tak reakce na chybný formát vstupních dat či selhání procesů. V případě konfiguračních souborů došlo také na kontrolu jejich bezpečnosti. Administrátor by totiž teoreticky mohl některým uživatelům systému umožnit jejich úpravy. U XML souborů nebyla nalezena žádná zranitelnost, v případě nastavení pro procesy vytvořené v Bashi však bylo zjištěno, že je do nich možné umístit škodlivý kód.

[11] https://www.ffmpeg.org/
[12] http://sox.sourceforge.net/
[13] http://gridscheduler.sourceforge.net/
[14] https://arc.liv.ac.uk/trac/SGE

Na vině byla absence escapování hodnot proměnných obsahujících uvozovky, která proto byla následně přidána. Vzhledem k jiným potenciálním zranitelnostem, které není dost dobře možné, vzhledem k beztypovosti jazyka, ošetřit, se však použití skriptů vytvořených v Bashi jako procesů spíše nedoporučuje. Případně by alespoň neměl mít nikdo kromě důvěryhodných osob možnost provádět jejich úpravy.

Kapitola 5

Vytvoření administračního rozhraní

V této kapitole je popsána tvorba webového administračního rozhraní pro prednasky.com k vytvořenému frameworku.

5.1 Specifikace požadavků a jejich analýza

Od webového rozhraní pro prednasky.com jsou očekávány následující funkce, definovány podle uživatelských rolí (následující role vždy zahrnuje předchozí):

- návštěvník – má možnost zobrazení úvodní stránky webu,
- student – může zhlédnout záznamy, psát komentáře,
- učitel – může vytvářet záznamy na základě přijatých dat z video serverů, nastavuje práva pro své záznamy (všichni studenti, studenti předmětu, pouze učitel),
- administrátor – možnost zobrazení front, šablon pro propojení úloh, tokenů včetně postupu, a odhadovaného zbývajícího času, vytváření šablon, přiřazení studentů a učitelů k předmětům,
- superadministrátor – může zastavit zpracování, nastavovat konfiguraci úloh a šablon propojení.

Každý záznam přednášky má, kromě přiřazení k předmětu a přístupových práv, obsahovat následující: video, slajdy, časování slajdů, titulky, abstrakt, název, klíčová slova, odkazy na externí zdroje, komentáře, audio stopu, seznam předchozích záznamů a seznam navazujících záznamů.

Jak je vidno, v požadavcích se nachází nejen specifikace pro administrační rozhraní k vytvořenému frameworku, ale částečně také požadavky na administrační rozhraní pro budoucí web prednasky.com i samotný web, který studentům fakulty umožní zhlédnutí streamovaných záznamů přednášek. Jde o velkou výhodu, neboť bude možné již nyní připravit databázi na to, aby byla s budoucím webem co možná nejvíce kompatibilní.

Z požadavků vyplývá, že bude nutné automatizovaně přijímat záznamy z video serverů, nabídnout je k zařazení administrátorům a následně ke zpracování učitelům. Při obdržení

nového videa tak pro něj systém bude muset vytvořit záznam v databázi, ideálně rozšířený o doplňující informace jako délka záznamu nebo náhledy.

Záznamy budou zpracovávány podle šablon předem připravených administrátory. Přednášející si při tvorbě přednášky zvolí jednu z nich a předá záznam ke zpracování. Každá šablona má obsahovat způsob propojení úloh včetně požadovaných vstupních souborů a nastavení pro všechny úlohy.

5.2 Návrh

Požadavky na webové rozhraní nijak nespecifikují strukturu webových stránek. Rozhodl jsem se proto vytvořit návrh struktury, který reflektuje logickou návaznost jednotlivých částí a zohledňuje přitom přístup osobami s různou úrovní oprávnění, viz obr. 5.1.

Obrázek 5.1: Návrh struktury webového administračního rozhraní

Autorizace

Namísto omezení přístupu ke stránkám čistě dle rolí jsem se rozhodl použít seznam oprávnění[1] (angl. access control list). Oproti použití pouhých rolí je mnohem flexibilnější, umožňuje totiž kdykoliv přidat nebo odebrat oprávnění jednotlivým rolím typicky prostřednictvím konfiguračního souboru, aniž by muselo být zasahováno do zdrojového kódu aplikace. Vytvořený seznam oprávnění a přiřazení oprávnění k rolím je obsaženo v příloze C.

Vytváření přednášek

Před tím, než jsou záznamy nabídnuty přednášejícímu ke zpracování, musejí být přijaty z video serverů a přiřazeny k předmětu administrátorem. Záznamy z fakultních video serverů budou přitom nahrávány do jediného, předem zvoleného, předávacího adresáře. Jejich příjem do adresáře bude nutné automaticky detekovat a do databáze zanést informace o zdrojovém videu. Jako užitečné informace zanášené do databáze kromě názvu souboru jsem stanovil velikost přijatého souboru, délku videa, datum a čas přijetí a náhledy.

Při tvorbě přednášky si přednášející zvolí zdrojové video, doplní k němu název, popis, klíčová slova, nahraje slajdy, vybere jednu ze šablon a předá video ke zpracování. Všechny

[1] http://cs.wikipedia.org/wiki/Access_control_list

tyto zadané informace se uloží do databáze, kromě slajdů. Ty jsou potřebné během zpracování a použití databáze by celou situaci zbytečně komplikovalo, nehledě na fakt, že ukládání souborů do databáze se obecně nedoporučuje [3]. Je tak nutné vymyslet způsob, jakým slajdy uložit na disk a zároveň mít o nich záznam v databázi, aby s nimi později bylo možné pracovat (nabídnout studentům jejich stažení apod.).

Naivním přístupem by bylo ukládat do databáze relativní cestu k souboru od složky, předem stanovené k ukládání slajdů. Problémem při použití tohoto způsobu je, že nelze zajistit jednoznačnou identifikaci souboru dle jeho názvu. Velice reálně totiž hrozí situace, kdy dva přednášející nahrají slajdy se stejným názvem. Aby proto byla zajištěna jejich jednoznačná identifikace, rozhodl jsem se navíc pro každé nahrané slajdy vypočítat MD5 hash na základě jejich obsahu, ten uložit do databáze a nahraný soubor přejmenovat dle hashe. Do databáze pak už stačí kromě hashe uložit jen původní název souboru. Takto uložené slajdy není problém kdykoliv zpracovat a na základě informací v databázi lze umožnit uživatelům jejich stažení.

Kromě již zmíněných informací je nutné do databáze ukládat také šablony zpracování a umožnit administrátorům jejich vytváření a editaci. Obecně vzato je šablona orientovaný graf, ve kterém vrcholy značí úlohy a hrany jejich propojení. Každá úloha může využívat buďto konfiguraci výchozí nebo specifickou pro danou šablonu. To samé platí o požadovaných vstupních souborech. Graf by pro přehlednost bylo dobré uživateli vizualizovat.

Odhad zbývajícího času

Jedním z požadavků je také zobrazení procentuálního postupu zpracování a odhadovaného zbývajícího času. Jelikož framework do databáze přímo ukládá procentuální postup, není jeho zobrazení nijak složité. Obtížnější situace nastává v případě zobrazení odhadu zbývajícího času, jelikož ten framework přímo neposkytuje.

Při uvážení informací ukládaných frameworkem do databáze a za předpokladu, že délka zpracování závisí na velikosti zdrojového videa však lze vytvořit lineární model odhadu. K výpočtu jsem se rozhodl využít datum a čas zahájení T_s a ukončení T_e zpracování, který framework ukládá, a velikost příslušného zdrojového videa S, jež se uloží při přijmutí záznamu z video serveru. Na základě těchto informací lze z posledních n záznamů v databázi vypočítat množství vstupních dat zpracovaných procesem za jednotku času P:

$$P = \sum_{i=1}^{n} \left(\frac{S_i}{T_{ei} - T_{si}} \right)$$

Z čehož je jednoduché určit celkový čas potřebný ke zpracování T_p aktuálního záznamu daným procesem za předpokladu, že je známa jeho velikost S:

$$T_p = \frac{S}{P}$$

A následně dle procentuálního postupu Pr zbývající čas T_r:

$$T_r = \left(1 - \frac{Pr}{100} \right) * T_p$$

Ve všech výše uvedených rovnicích je samozřejmě nutné pracovat se stejnými jednotkami, pro dosažení vysoké přesnosti nejlépe sekundami pro rozdíl časů a bajty v případě velikosti, čímž se získá výsledek v sekundách. Dále, výpočet první rovnice musí probíhat

pouze nad záznamy z databáze, které mají validně zadány oba časy. A nakonec, aby nebylo nutné počítat první rovnici, což může být při použití vysoké hodnoty n náročné jak na databázi, tak na výpočet, stačí ji spočítat jednou po nasbírání statisticky relevantního množství dat a uložit k danému procesu do databáze. Pro zpřesnění odhadu stačí případně toto číslo čas od času aktualizovat.

Databáze

Na základě výše uvedených informací a znalostí o vytvořeném frameworku pro automatické zpracování záznamů byla vytvořena databáze. Její kompletní relační schéma lze nalézt v příloze D.

5.3 Použité webové technologie

Jelikož dnes existuje celá řada řešení usnadňujících tvorbu webu, ať už jde o aplikační logiku nebo vzhled stránek, rozhodl jsem se jich využít. Vedoucím práce bylo doporučeno použití PHP frameworku Nette[2] ve verzi 2.2.7 a CSS frameworku Bootstrap[3] verze 3.3.2.

Bootstrap

V dnešní době přistupuje k webovým stránkám velká část návštěvníků přes své chytré telefony a tablety. Vytvořit responzivní web, který přizpůsobí vzhled a chování těmto zařízením je tak prakticky nutností. S cílem usnadnit tvůrcům stránek optimalizaci pro všechny typy zařízení vzniklo několik různých CSS frameworků. Mezi nejpopulárnější patří Foundation[4] a právě doporučený Bootstrap.

Bootstrap poskytuje řadu znovupoužitelných HTML komponent jako jsou tlačítka, nástrojové lišty, tabulky nebo formuláře. Zásadní přínos při tvorbě responzivního designu má však mřížka. Tvoří ji 12 sloupců, které mění svou šířku dle rozlišení zobrazovacího zařízení. Části webové stránky se umístí do této mřížky, přičemž pomocí CSS stylů lze zvolit, kolik sloupců budou zabírat dle čtyř typů zařízení – mobilní telefony, tablety, desktopy s nízkým rozlišením a desktopy s vysokým rozlišením. Vzhled se poté zařízení automaticky přizpůsobí.

Nette

Z pohledu aplikační logiky je v případě webových stránek téměř vždy nutné řešit opakující se problémy. Jsou jimi zejména autentizace a autorizace uživatelů, práce s databází a dynamická změna obsahu stránek. Nejoblíbenějšími volně dostupnými frameworky řešícími nejen tyto problémy jsou Zend[5], Symfony[6], Laravel[7] a již zmíněné Nette.

Nette se v České republice těší velké oblibě, zejména díky tomu, že jeho autorem je český vývojář David Grudl a na webu lze nalézt řadu tutoriálů v češtině. Hlavními jeho přednostmi jsou technologie Context-Aware Escaping[8] zabraňující vzniku bezpečnostních

[2] http://nette.org/
[3] http://getbootstrap.com/
[4] http://foundation.zurb.com/
[5] http://framework.zend.com/
[6] https://symfony.com/
[7] http://laravel.com/
[8] http://doc.nette.org/cs/2.3/templating#toc-context-aware-escaping

děr, vysoký výkon, pokročilý ladicí nástroj Tracy, návrh využívající nových objektových vlastností PHP 5 a použití architektury typu model-view-controller.

Díky architektuře model-view-controller lze u aplikací s grafickým uživatelským rozhraním oddělit od sebe kód aplikační logiky (model) od kódu zobrazujícího data (view). Propojení obou částí pak zajišťuje controller (v Nette presenter). Tato architektura zpřehledňuje aplikaci a usnadňuje její budoucí vývoj.

Nette dále obsahuje třídu Database usnadňující práci s databází, takže není nutné používat speciální databázový framework. Database poskytuje abstrakci při tvorbě SQL dotazů, usnadňuje práci s více tabulkami a snaží se vytvářet efektivní SQL dotazy, aby nebylo nutné přenášet zbytečná data.

Paradoxně největší slabinou frameworku je, i přes velkou českou komunitu, absence kvalitní dokumentace. Na oficiálních stránkách lze nalézt několik tutoriálů, které vysvětlují základy práce s frameworkem, při používání pokročilejších funkcí však člověk snadno narazí na absenci dokumentace a nezbývá než dlouho a někdy marně „googlit“ nebo procházet zdrojové kódy.

Cytoscape.js

Pro vizualizaci propojení úloh poslouží javascriptová knihovna Cytoscape.js[9]. Knihovna je určena pro vizualizaci libovolných grafových struktur. Disponuje širokými možnostmi nastavení, umožňuje nejen vizualizaci, ale i manipulaci se zobrazeným grafem, a především má kvalitní dokumentaci a na oficiální stránkách lze nalézt velké množství příkladů. Navíc je optimalizována i pro použití na mobilních zařízeních.

5.4 Implementace

Aplikační logika webového rozhraní byla implementována v jazyce PHP s využitím již zmíněného frameworku Nette. Při implementaci bylo nutné vytvořit znovupoužitelnou vizuální komponentu pro stránkování. Nette v základu žádné vizuální komponenty kromě formulářů nenabízí a na webu nebyla žádná volně stažitelná komponenta pro stránkování v době implementace nalezena. Jinak se při implementaci neobjevil prakticky žádný větší problém, nicméně místy jsem pocítil absenci kvalitnější dokumentace, což zbytečně prodloužilo dobu potřebnou pro vývoj.

Součásti aplikace byly odděleny dle návrhového vzoru model-view-controller a strukturovány standardním způsobem[10]. Všechny modelové třídy v aplikaci obsahují pouze logiku nutnou pro získání nebo uložení dat do databáze a práci s vytvořeným frameworkem. To umožňuje mj. kdykoliv změnit způsob, jakým se s databází pracuje, aniž by bylo potřeba zasahovat do jiného kódu. View tvoří šablony pro systém Latte, jenž je součástí frameworku Nette. Ten se ukázal býti skvělým pomocníkem, neboť napomáhá eliminaci duplicitního kódu možností dědit šablony či znovupoužití komponent, automatickým escapováním obsahu pak zase zabraňuje vzniku bezpečnostních děr. Presentéry (controllery) reagují na akce prováděné uživatelem a předávají data z modelu do view. Presentéry zároveň logicky oddělují jednotlivé části webu a jejich struktura odpovídá struktuře webu. Níže uvádím ty nejdůležitější spolu s popisem stěžejních funkcí:

[9] http://js.cytoscape.org/

[10] http://doc.nette.org/cs/2.3/quickstart/getting-started#toc-obsah-sandboxu

- BasePresenter – obsahuje tovární metodu pro vytvořenou komponentu stránkovače, všechny presentéry využívající stránkování od něj dědí,
- CoursesPresenter – vytváření a editace předmětů,
- LecturesPresenter – vytváření přednášek,
- ProcessPresenter – přidávání procesů, zobrazení front, logů a editace konfigurace,
- StatsPresenter – aktualizace statistik využívaných při odhadu zbývajícího času zpracování,
- TemplatesPresenter – vytváření, úprava a odstraňování šablon zpracování,
- TokenPresenter – prohlížení stavu zpracování tokenů, možnost zastavit zpracování,
- UsersPresenter – změna oprávnění uživatelů, přiřazení k předmětům,
- VideoPresenter – zobrazení záznamů přijatých z video serverů, přiřazení k předmětu a přednášejícímu.

Příjem dat z video serverů

Pro zajištění automatického přijmu dat z video serverů bylo nutné vytvořit program, který monitoruje obsah předávacího adresáře a zaznamenává nově přijaté soubory do databáze. Pro tento účel jsem vyvinul skript `folder_daemon` v jazyce Python, který detekuje nově vytvořená videa ve zvoleném adresáři a zaznamená informace o nich do databáze.

Skript využívá knihovnu třetí strany Pyinotify[11], jež usnadňuje monitorování událostí nastalých při práci se souborovým systémem pod operačními systémy založenými na Linuxovém jádře. V tomto případě se konkrétně využívá detekce události konce zápisu souboru. V tu chvíli však ještě nemusejí být všechna data zapsána na disku a proto před tím, než se začnou získávat informace o délce videa a vytvoří náhledy, je operační systém prostřednictvím systémového volání požádán o zapsání všech změn daného souboru z vyrovnávací paměti na disk.

Získané informace jsou zobrazeny ve webovém administračním rozhraní, jak demonstruje obrázek 5.2. Náhledy lze zvětšit a procházet mezi nimi pomocí směrových kláves nebo myši.

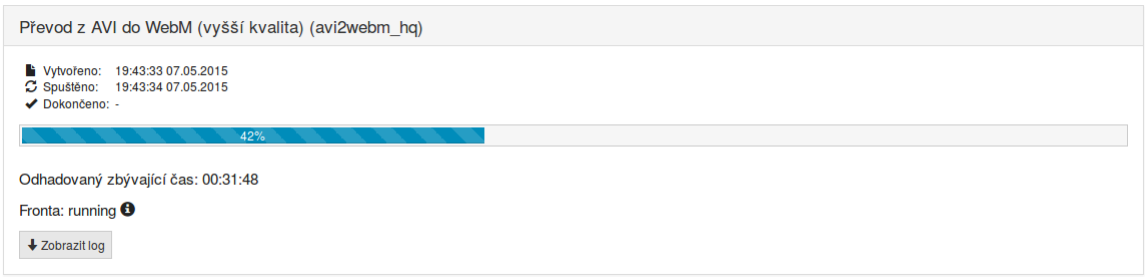
Zobrazení odhadovaného zbývajícího času

Na základě návrhu byl implementován lineární model predikce odhadu zbývajícího času. Využívá posledních 1000 platných záznamů z databáze o zpracování pro každý z procesů a dále informace o velikosti zdrojových videí. Při vykreslení stránky obsahující odhad se využívá hodnoty počet bajtů zpracovaných za sekundu (bps) daným procesem, jež je uložena v databázi. Nedochází tak ke zbytečným výpočtům. Na druhou stranu je potřeba hodnoty bps pravidelně aktualizovat, aby byl odhad co nejpřesnější. K tomu slouží v administračním rozhraní volba *aktualizovat statistiky*. Pokud není žádoucí provádět aktualizaci ručně, je možné použít kód ve skriptu a spouštět jej pravidelně např. pomocí softwarového démona cron.

Při prohlížení podrobností o tokenu je odhadovaný zbývající čas zobrazen pro každý proces použitý k jeho zpracování pod stavovou lištou, která ukazuje procentuální postup

[11] https://github.com/seb-m/pyinotify

Obrázek 5.2: Zobrazení informací o videu ve webové aplikaci, vytvořených skriptem přijímajícím data z video serverů



Obrázek 5.3: Ukázka zobrazení postupu zpracování a odhadu zbývajícího času

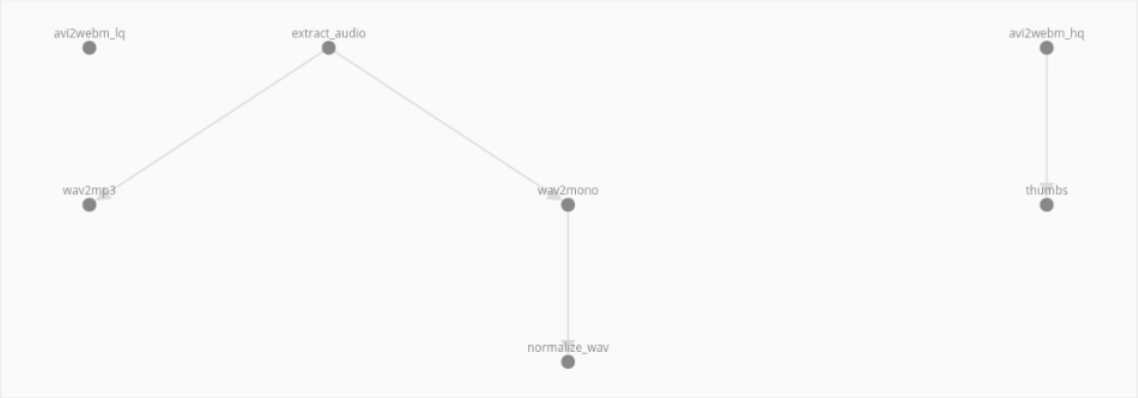
zpracování, viz obr. 5.3. Protože vypočítaný čas je v sekundách, převádí se na zobrazení ve formátu hodin, minut a sekund pomocí vlastního Latte filtru aplikovaného v šabloně stránky.

Vizualizace propojení procesů

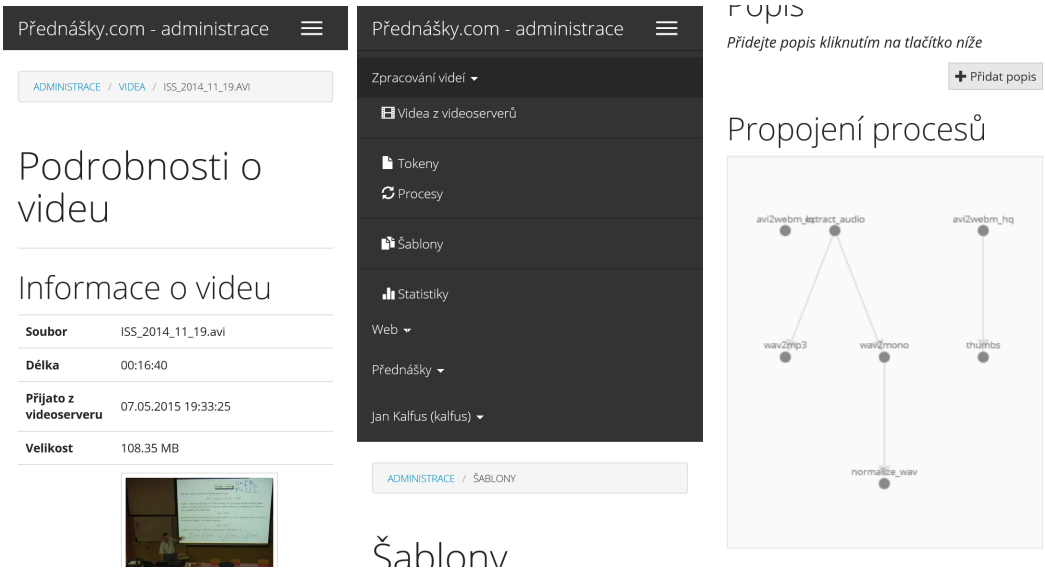
S cílem usnadnit uživatelům orientaci při tvorbě šablon zpracování jsem se rozhodl vizualizovat propojení procesů pomocí grafu. Využita k tomu byla dříve zmíněná knihovna Cytoscape.js. Zobrazení je nakonfigurováno tak, aby se jednotlivé uzly zobrazovaly v jejich logickém pořadí odshora dolů. Definice hran a uzlů jsou zadávány ve formátu JSON, který je generován na základě informací o propojení procesů v šabloně, uložených v databázi. Ukázku vizualizace lze nalézt na obrázku 5.4.

5.5 Testování funkcionality

Správné zobrazení administračního rozhraní bylo testováno v několika webových prohlížečích a na různých zařízeních: Mozilla Firefox 36 (desktop), Google Chrome 41 (desktop), Internet Explorer Mobile 11 (telefon) a Safari pod iOS 7 (iPad). Z hlediska funkčnosti nebyl objeven žádný problém, který by znemožňoval jeho použití. Všiml jsem si však, že ne vždy dokáže javascriptová knihovna Cytoscape.js na zařízeních s úzkou zobrazovací plochou správně vykreslit graf – dochází k překrývání popisů uzlů (viz obr. 5.5 vpravo). Tento spíše kosmetický problém nicméně nemá žádný vliv na funkčnost a při otočení mobilního telefonu na šířku



Obrázek 5.4: Ukázka vizualizace propojení procesů v šabloně



Obrázek 5.5: Zobrazení webového rozhraní na mobilních telefonech

díky rozšíření zobrazovací plochy mizí. Na mobilních telefonech byla dále provedena optimalizace zobrazení některých součástí jako např. menu, které se u těchto zařízení rozbalí až po ťuknutí na symbol tří linek tak, aby jinak zabíralo co nejméně místa (na obr. 5.5 vlevo ukázka sbaleného menu, uprostřed rozbaleného).

Napojení webového rozhraní na vytvořený framework se ukázalo býti takřka bezproblémové. Pouze v případě zobrazení záznamů z logů byl zjištěn drobný nedostatek. Šablonovací systém Latte totiž nedokáže automaticky převést unixový znak konce řádku z databáze na html tag `<br>` a namísto toho jej escapuje. To bylo nutné napravit převedením znaků nového řádku na tag `<br>` a teprve následným escapováním textu.

Aplikace byla testována také z hlediska zabezpečení. Díky použití Nette Database pro práci s databází a ochrany formulářů v Nette není možné provést útoky typu SQL Injection[12] ani Cross-Site Request Forgery[13]. Automatickým escapováním v šablonách je

[12] http://cs.wikipedia.org/wiki/SQL_injection
[13] http://cs.wikipedia.org/wiki/Cross-site_request_forgery

pak zajištěna ochrana také proti útokům typu cross-site scripting[14]. Z hlediska autorizace nemají obyčejní administrátoři právo měnit role jiných administrátorů ani superadministrátora a nemohou povýšit učitele či studenty na administrátory. To může pouze superadministrátor. Konzistence dat v databázi je zajištěna nastavenými integritními omezeními a žádný pokus o jejich porušení zadáním neplatných hodnot neuspěl.

[14] http://cs.wikipedia.org/wiki/Cross-site_scripting

Kapitola 6

Závěr

Cílem této práce bylo navrhnout a implementovat framework pro automatické zpracování záznamů přednášek včetně webového rozhraní. Nejprve bylo zapotřebí nastudovat, jakým způsobem typicky probíhá zpracování záznamů, zmapovat současná řešení pro automatické zpracování dat a analyzovat framework pro zpracování úloh dodaný vedoucím práce. Na základě těchto informací byl následně proveden návrh a implementace frameworku v jazyce Python 3.4.

Vytvořený framework umožňuje rozdělit zpracování multimediálních dat na atomické úlohy, které automaticky propojuje, přičemž dokáže úlohám předávat požadované vstupní soubory. Některé z vlastností frameworku lze konfigurovat a změnit tak jejich chování, ať už pro všechny úlohy, vybrané z nich nebo pro konkrétní záznam. Pro každý záznam lze dále nastavit způsob propojení úloh. Framework zvládá spolupráci se systémem pro správu dávkového zpracování úloh Sun Grid Engine a data je tak možné zpracovávat v počítačovém clusteru. Vedle toho může být napojen také na databázi a poskytnout v reálném čase informace o aktuálním stavu zpracování či logovací záznamy.

K frameworku bylo navrženo a implementováno také webové administrační rozhraní, jež umožňuje jeho správu a využívá možnost napojení frameworku na databázi. Struktura databáze a webového rozhraní byla navržena tak, aby web v budoucnu bylo možné snadno rozšířit o část, která umožní studentům prohlížení zpracovaných videozáznamů spolu s dalšími rozšiřujícími informacemi. Webové rozhraní bylo implementováno s využitím PHP frameworku Nette, což napomohlo zlepšit udržovatelnost kódu a zabezpečení aplikace, a dále CSS frameworku Bootstrap, díky kterému lze administrační rozhraní bez problémů používat i na mobilních zařízeních.

Funkčnost řešení byla úspěšně otestována pomocí úloh využívaných ke zpracování záznamů nad vzorkem dat z fakultních video serverů a je tedy připraveno k použití. Pro maximální využití potenciálu bude však nutné přidat vedle existujících úloh také úlohy pro přepis řeči na text a synchronizaci slajdů s obsahem. V budoucnu, po rozšíření webového rozhraní o část, která umožní přednášejícím nejen přednášky vytvářet, ale i kompletně spravovat, by bylo dobré také rozhraní ve spolupráci s profesory otestovat z pohledu uživatelského rozhraní a případně upravit pro zajištění maximální uživatelské přívětivosti.

Literatura

[1] Brecht, H. D.; Ogilby, S. M.: Enabling a Comprehensive Teaching Strategy: Video Lectures. *Journal of Information Technology Education: Innovations in Practice*, ročník 7, 2008.
URL http://jite.org/documents/Vol7/JITEV7IIP071-086Brecht371.pdf

[2] Martin, R. C.; Feathers, M. C.; Ottinger, T. R.; aj.: *Clean Code*. Pearson Education, Inc, 2008, ISBN 978-0-13-235088-4.

[3] McCraw, S. M.; Dolan, Q.; Robert, P.: Development-Database vs Filesystem. online, únor 2011.
URL https://wiki.wocommunity.org/display/documentation/Development-Database+vs+Filesystem

Příloha A

Formát XML konfiguračního souboru pro procesy a framework

Níže je v jazyce DTD popsána struktura XML konfiguračního souboru, který se využívá pro procesy, wrapper a handler. Umožňuje definici hodnot základních datových typů boolean, integer, float, string a také seznamu a slovníku.

```
<!ELEMENT configuration (variable*, list*, dictionary*)>

<!ELEMENT variable EMPTY>
<!ATTLIST variable name CDATA #REQUIRED>
<!ATTLIST variable type (bool|int|string|float) #REQUIRED>
<!ATTLIST variable value CDATA #REQUIRED>

<!ELEMENT list (item*)>
<!ATTLIST list name CDATA #REQUIRED>
<!ELEMENT item EMPTY>
<!ATTLIST item type (bool|int|string|float) #REQUIRED>
<!ATTLIST item value CDATA #REQUIRED>

<!ELEMENT dictionary (entry*)>
<!ATTLIST dictionary name CDATA #REQUIRED>
<!ELEMENT entry (key, value)>
<!ELEMENT key (CDATA)>
<!ELEEMTN value (CDATA)>
```

Příloha B

Formát XML konfiguračního souboru pro propojení úloh

Níže je v jazyce DTD popsána struktura XML konfiguračního souboru, který se využívá pro propojení úloh a automatické řešení závislostí.

```
<!ELEMENT configuration (process+)>

<!ELEMENT process (input*, next*, previous*)>
<!ATTLIST process id CDATA #REQUIRED>

<!ELEMENT input (CDATA)>
<!ELEMENT next (CDATA)>
<!ELEMENT previous (CDATA)>
```

Příloha C

Oprávnění a role uživatelů

Seznam uživatelských oprávnění pro web:

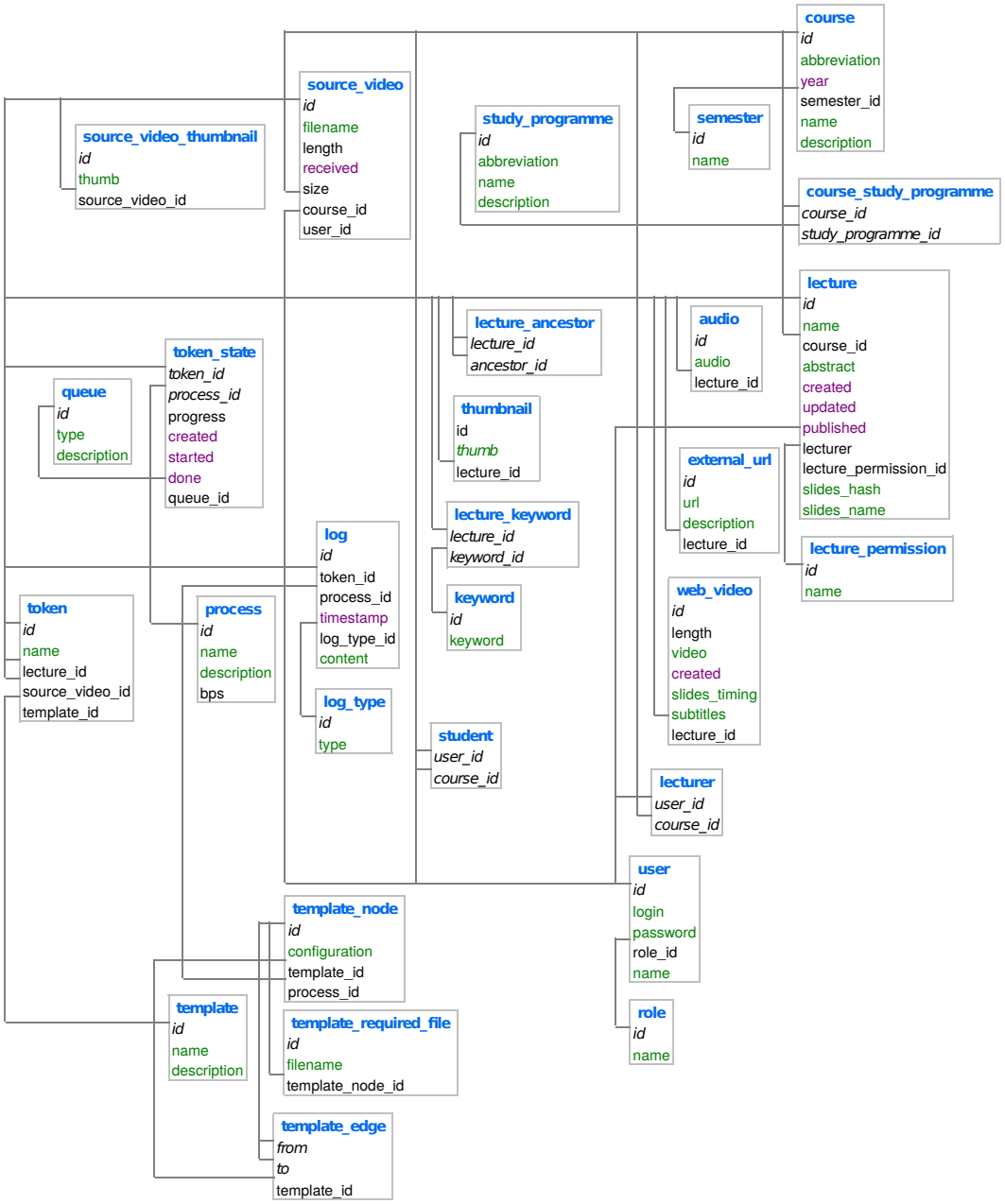
- public - veřejně přístupné části webu
- viewVideos - prohlížení videí
- administration-lectures - možnost vytvořit přednášku na základě přiřazených videí
- administration-token - prohlížení tokenů
- administration-token-kill - zastavení zpracování tokenů
- administration-courses - správa předmětů
- administration-process - prohlížení procesů, logů a front
- administration-users - správa uživatelů, bez možnosti povýšit někoho na administrátora či superadministrátora nebo upravit práva existujících administrátorů a superadministrátorů
- administration-users-super - možnost povýšit uživatele na administrátora, upravovat práva existujících administrátorů
- administration-video - správa videí přijatých z video serverů
- administration-stats - správa statistik zpracování
- administration-configuration - konfigurace procesů
- administration-templates - správa šablon

Přiřazení oprávnění rolím, přičemž následující role obsahuje vždy navíc oprávnění té předchozí:

- návštěvník - public
- student - viewVideos
- učitel - administration-lectures
- administrátor - administration-token, administration-courses, administration-process, administration-users, administration-video, administration-advanced, administration-templates
- superadministrátor - vše

Příloha D

Relační schéma databáze



Příloha E

Obsah CD

Přiložené CD má následující adresářovou strukturu a obsahuje:

- `/text` – text písemné zprávy ve formátu PDF a zdrojové kódy pro LaTeX
- `/sources/framework` – zdrojové kódy frameworku, manuál a příklady
- `/sources/tasks` – zdrojové kódy vytvořených úloh
- `/sources/web` – zdrojové kódy webového rozhraní
- `/poster` – plakát
- `/video` – video