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

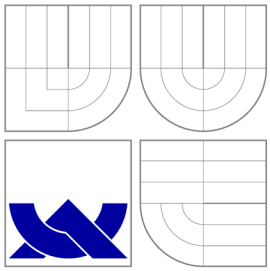
PREDNASKY.COM – SYSTÉM PRO AUTOMATICKÉ ZPRACOVÁNÍ PŘEDNÁŠEK

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

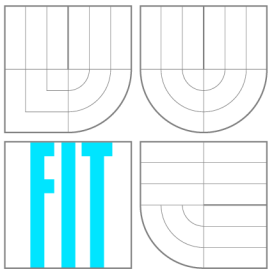
AUTOR PRÁCE
AUTHOR

Bc. PAVEL ČERNÝ

BRNO 2015



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 – SYSTÉM PRO AUTOMATICKÉ ZPRACOVÁNÍ PŘEDNÁŠEK

PREDNASKY.COM – SYSTEM FOR AUTOMATIC LECTURE PROCESSING

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. PAVEL ČERNÝ

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. IGOR SZÖKE, Ph.D.

BRNO 2015

Abstrakt

Cílem této práce je vytvoření systému pro automatizované zpracování videí z přednášek. Práce je rozdělena na dvě hlavní části. Bash Framework, který je tvořen z několika oddělených atomických úloh. Na jeho vstupu je video, a jakmile je toto video zpracované, je k němu přidána úvodní část, titulky a synchronizovaná prezentace. Druhou částí je webová aplikace, která slouží pro přehrávání, editaci a vytváření videí. Dále nám poskytuje správu jednotlivých procesů vytvořeného frameworku a celkovou administraci systému.

Abstract

The objective of this thesis is to create system for automated processing of the video lectures. There are two main parts in this thesis. Bash Framework, which is created from several separated atomic tasks. At the beginning we have a simple video prepared for processing. When it is processed, there are added intro part, subtitles and synchronized presentation. Second part of this thesis is a web application. It is designated for playing, editing and creating videos. Further it provides process management of created framework and whole system administration.

Klíčová slova

Bash, framework, atomické úlohy, přednášky, video server, zpracování videa, tokenové zpracování, prednasky.com, responzivní design.

Keywords

Bash, framework, atomic tasks, lectures, video server, video processing, token processing, prednasky.com, responsive design.

Citace

Pavel Černý: Prednasky.com – systém pro automatické zpracování přednášek, diplomová práce, Brno, FIT VUT v Brně, 2015

Prednasky.com – systém pro automatické zpracování přednášek

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Igora Szökeho, Ph.D.

.....
Pavel Černý
27. května 2015

Poděkování

Tímto bych rád poděkoval vedoucímu své práce, panu doktoru Igoru Szökemu, za ochotu a trpělivost na konzultacích a při tvorbě práce.

Obsah

Kapitola 1

Úvod

Cílem této práce je vytvoření celistvého systému, který bude zpracovávat videa z přednášek. Vstupem bude samotná nahrávka přednášejícího a výstupem by mělo být video ve webové aplikaci se všemi dostupnými materiály.

Hlavní částí tohoto systému bude bash framework, který se bude skládat z několika atomických, na sobě nezávislých, úloh. Celkově se bude jednat o stažení videa, střih, konverzi audia a videa, převod zvuku na řeč, synchronizaci s případnou prezentací, indexace, manuální editace a finálním krokem bude publikace.

Další důležitou částí bude webové rozhraní, které bude sloužit zejména pro samotné přehrávání videa, dále pak pro editaci, kontrolu stavu rozpracovaných videí, zakládání nových úloh, správu procesů a podobně.

V druhé kapitole obecně popisuji Bash Framework – proč jsme si jej zvolili, dále pak princip delegování úloh, základní popis konfigurace, nastavení a struktury programu. V další části jsou podrobně popsány funkce frameworku a způsob obsluhy procesů. V závěru se zmiňuji o napojení Frameworku na SGE, což je systém pro dávkové zpracování úloh.

V následující kapitole tohoto dokumentu se věnuji popisu a teoretickému základu atomických úloh. Na začátku kapitoly je popsáno jak jsou procesy spouštěny, kontrolovány a přeposílány. Nechybí také souhrnný diagram všech procesů a jejich závislostí. Dále jsem vybral tři zajímavé procesy, které se dělí vždy na čtyři části – v první je popsáno k čemu daný proces slouží a jeho standardní vstup a výstup, v druhé části je specifikováno, co se děje před spuštěním procesu, v třetí je rozebráno samotné spuštění a v poslední části je popsáno přeposláni procesu. Ostatní procesy jsou popsány v příloze.

Ve čtvrté kapitole se věnuji druhé důležité části práce – webovému rozhraní. Na začátku popisuji role, které se v systému nacházejí. Dále se věnuji výběru frameworku a šablonovacího systému, kde rozebírám i ostatní volby. Následující část je věnována technologiím, které jsem v projektu použil. Poslední částí této kapitoly je návrh a implementace, kde je popsáno, jak fungují a jak jsou implementovány nejdůležitější části systému.

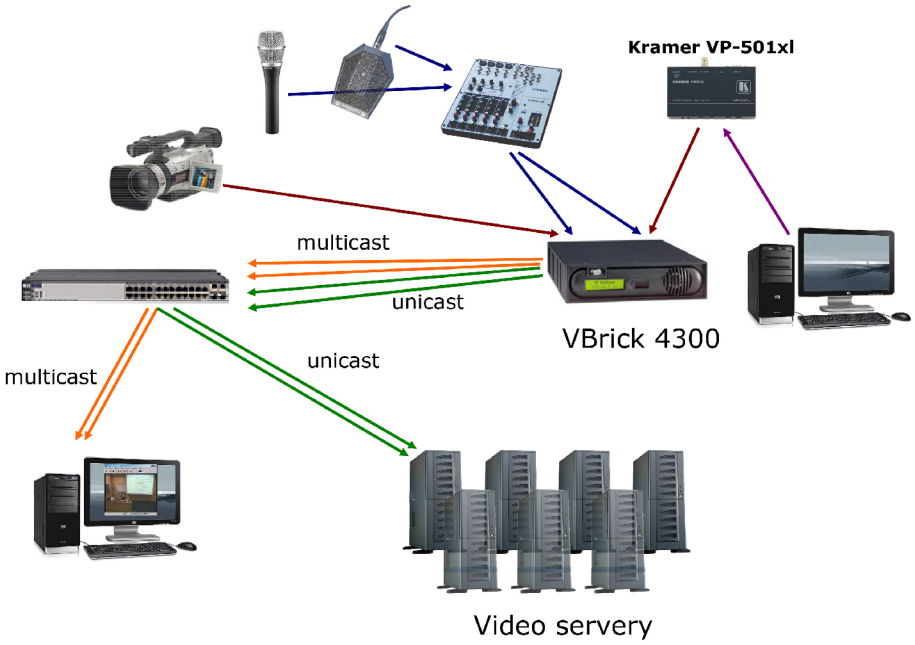
Na začátku páté kapitoly je zobrazen diagram případů užití, kde můžeme vidět aktéry systému a dané případy užití. Následně popisuji tří zajímavé specifikace případů užití.

V poslední, šesté kapitole, se zaměřuji na bezpečnost systému jako celku. V první části se zaobírám bezpečnosti Bash Frameworku a v druhé části bezpečnosti webového rozhraní.

1.1 Motivace

Jednou z motivací bylo změnit stávající systém sledování přednášek. Nyní popíši systém nahrávání videí na fakultě.

Na FIT VUT jsou v posluchárnách instalovány HD kamery (např. Sony HDR-FX1), které snímají obraz analogově a posílají jej do HW enkodérů. Na fakultě jsou čtyři druhy těchto enkodérů – VBrick 4200/4300 a Teracue ENC-100/200. Většina jsou MPEG-2 enkodéry kromě Teracue ENC-200, což je MPEG-4 enkodér [14].



Obrázek 1.1: Schéma zapojení na FIT VUT. Převzato z [14].

Na obrázku 1.1 můžeme vidět zapojení s enkodérem VBrick 4300, který poskytuje čtyři UDP streamy. V tomto sestavení je možné zároveň ukládat nahrávaný obsah (unicast – zasílaní paketů pouze na video servery) a streamovat jej do sítě (multicast – data ke koncovým uživatelům „tečou“ sítí pouze jednou).

Pokud je obsah streamovaný přímo do sítě jedná se o „realtime“ streaming – „živý“ přenos. Nás bude více zajímat „on demand“ streaming tedy na vyžádání [5]. Jako studenti se můžeme přihlásit na video servery a stáhnout si nahranou přednášku. Zde nacházíme problém – „User Experience“ tedy jakési porozumění uživateli, prožitek uživatele je zde velmi špatný, což je také jeden z cílů, které bych chtěl zlepšit.

Dále existuje webová aplikace *prednasky.com*, která je částečně napojena na video servery. Jsou v ní ale bohužel pouze dvě skupiny zastaralých videí (z roku 2011 a 2013), které jsou veřejné. V aplikaci například chybí napojení na IS FIT, protože neobsahuje přihlášení studentů a tudíž nelze přehrát videa pouze pro zapsané studenty, které jsou na video serverech přítomné. Tuto webovou aplikaci budu tedy přetvářet.

Dalším cílem je také maximálně zjednodušit předávání videí mezi těmito systémy. K dosažení těchto cílů je zapotřebí vytvořit framework, který bude videa zpracovávat. Dále je nutné tento výstup z frameworku převzít a zobrazit ve webové aplikaci. Těmto prvkům se věnuji v následujících kapitolách.

Kapitola 2

Bash framework

Záznamy z přednášek se aktuálně zpracovávají tím způsobem, že jsou publikovány na video serveru FIT, kde si je může student pouze stáhnout – bez jakýchkoli dalších dat. K záznamu je pouze na začátek přidán obrázek, kde je upozornění o kopírování a dále je provedena konverze do správného formátu. Vždy je tedy nutné si záznam před zhlédnutím stáhnout.

Zpracování záznamu přednášek je velmi variabilní – můžeme mít jednoduché video, které se pouze překonvertuje do požadovaného formátu. V opačném případě je však, s kontrolou správce nebo bez, nutné přidat intro snímek, detekovat plátno a synchronizovat prezentaci.

Základní myšlenka byla taková, že na servru budou spuštěny programy ve smyčce, které zpracovávají jednotlivé kroky při kompletování videa. Velký důraz byl kladen na počáteční požadavky:

- atomicita jednotlivých procesů,
- persistence,
- přenositelnost,
- rozšiřitelnost,
- bezpečnost,
- konfigurace.

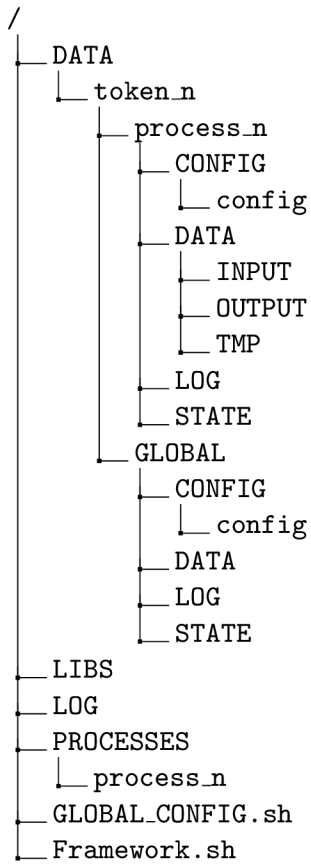
Jako vhodné řešení jsme zvolili vytvoření bash frameworku a to z toho důvodu, že většina použitých programů jsou unixového typu pracující se standardním vstupem a výstupem. Mohl bych zvolit řešení naprogramování aplikace v některém z kompilovaných jazyků, která bude spouštět jednotlivé úlohy, ale toto řešení by bylo obtížně rozšiřitelné už jen z toho důvodu, že při změně je zapotřebí program přeložit. Dále jsem přemýšlel o řešení ve skriptovacích jazycích, ale od tohoto jsem také opustil ze dvou důvodů – psát program v bashi mi přišlo více přímočaré a za druhé, nenalezl jsem vhodný framework v daných skriptovacích jazycích, který by byl vhodný k řešení daného problému.

Pokud shrnu úkoly frameworku, tak bude provádět základní operace nad složkami a soubory (vytvoření, mazání, změna), spouštění programů a zpracování jejich vstupů a výstupů. Nejvíce procesorového času zabere samotný běh jednotlivých externích programů – z tohoto důvodu není relevantní rychlost samotného Bashe. Pokročilé skriptovací jazyky (Perl, Python, PHP, atp.) navíc podporují mnoho funkcí, které bych nevyužil. Posledním důvodem bylo doporučení od vedoucího práce – zpracovat Framework v Bashi z důvodů snadné přenositelnosti.

Při hledání již existujících řešení ohledně tokenového zpracování jsem nenalezl nic vhodného. Pro Bash existuje mnoho pomocných knihoven s užitečnými funkcemi, ale nic komplexního, co by se blížilo k mému problému. Využil jsem některé části, z těchto knihoven:

- Bash Shell Function Library (bsfl)[1] – inspirace pro logování.
- Bashinator[2] – inpirace logování, chybové zpracování, zpracování chyb.
- Shell Script Framework tool (shesfw)[3] – zpracování parametrů.

Nicméně ani mimo jazyk bash jsem nenalezl nic uspokojivého, co se zaobírá tokenovým zpracováním. Při implementaci jsem čeptal hlavně z [10, 15].



Obrázek 2.1: Adresářová struktura.

2.1 Tokenové zpracování úloh

Pro zachování persistence čili aktuálního stavu systému v případě poruchy jsme měli na výběr ze dvou možností:

- databáze,

[1] Viz `https://code.google.com/p/bsfl/`.
[2] Viz `http://www.bashinator.org/`.
[3] Viz `https://code.google.com/p/shesfw/`.

- ukládání stavu na disk.

Náš problém jsme se rozhodli řešit druhou možností, protože připojovat se z bash skriptu na databázi není přímočaré a elegantní jako ukládání stavu na disk. Samozřejmě vše také závisí na frekvenci zápisu/čtení dat, kterého u našeho projektu bude velmi málo, takže také z toho důvodu můžeme použít zápis na disk. Navíc by bylo potřeba běžící databázový server.

Každá úloha ke zpracování bude reprezentována *tokenem* – souborem, fyzicky uloženým na disku. Tento token bude „probublávat“ přes jednotlivé atomické procesy. Pokud bude mít proces na vstupu token, zpracuje svou část a přepošle tento token jinému procesu. Podrobněji k atomickým procesům v kapitole č. 3.

2.2 Konfigurace, struktura

Bash framework má pevnou konfigurační strukturu. Základem je globální konfigurace celého frameworku, následně ji rozšiřuje globální konfigurace tokenu. Tyto dvě konfigurace by měly mít společný kontext. Dále pak bude načítaná konfigurace jednotlivých procesů a jako nejvyšší prioritu má konfigurace úloh – tokenů.

Framework má pevně danou strukturu, viz obrázek 2.1.

Složka `DATA`

Tento adresář bude obsahovat složky s názvy zpracovaných tokenů, kde nalezneme tyto podsložky:

- **Složka `GLOBAL`** – Obsahuje konfigurační soubor `CONFIG.sh`, který, jak je popsáno na začátku podkapitoly č. 2.2, se načítá jako druhý v pořadí. Měl by ovlivňovat chod frameworku, jako celku a platit tedy pro všechny následně spuštěné procesy.

  Dále obsahuje podsložku `DATA`. Jedná se opět o globální složku, jejichž obsah, složky `INPUT`, `OUTPUT` a `TMP`, bude dostupný všem procesům.

  Podsložka `LOG` slouží pro logování globálních událostí nad tokenem.

  Podsložka `STATE` slouží pro zaznamenání globálního stavu tokenu.

- **Složky s názvy procesů** – Obsahuje soubor `CONFIG.sh`, který je načítán jako poslední konfigurační soubor a tedy má největší prioritu.

  Dále obsahuje podsložku `DATA`, která obsahuje složky `INPUT`, `OUTPUT` a `TMP`. Před spuštěním každého procesu by měla být složka `INPUT` již naplněna vstupními daty. V průběhu procesu se ukládají dočasné data do složky `TMP` a na konci se výstupní data uloží do složky `OUTPUT`.

  Podsložka `LOG` slouží pro logování všech kroků a událostí v daném procesu.

  Podsložka `STATE` slouží pro zaznamenání aktuálního stavu, ve kterém se nachází daný proces. Následné využití například pro webovou aplikaci.

Složka `LIBS`

Tento adresář obsahuje tyto pomocné knihovny a programy:

- `can_token_be_run.sh` – Program zkontroluje zda token prošel kontrolou u daného procesu (v takovémto případě obsahuje slovo `YES`) a vrátí `YES` nebo `NO`.
- `error_handler.sh` – Knihovna pro obsluhu chyb a zápisu do logu. Při každé takové skutečnosti se nejprve zapíše časová známka, poté proces a nakonec zpráva.
- `list_waiting_tokens.sh` – Program zkontroluje složku `A_WAIT` a vrátí pole čekajících tokenů.
- `progress_lib.sh` – Knihovna pro tisk aktuálního stavu daného procesu pokud zpracovává token. Zapisuje se do souboru `state.progress` složky `DATA/token_n/process_n/ /STATE`.
- `required_vars_lib.sh` – Program, který zkontroluje zda jsou zadané proměnné, které jsou obsaženy v prvním parametru. Pokud nejsou, program skončí s chybou.
- `hash_lib.sh` – Knihovna, která je používána pro práci s asociativními poli v textových řetězcích. Využívá program `awk`[4].
- Dále obsahuje skripty pro obsluhu procesů, které jsou popsány v kapitole 2.4.

Složka `LOG`

Tento adresář obsahuje logovací soubory hlavního programu:

- `logfile.log` – Standardní výstup. Zaznamenávány jsou spouštěné volby programu.
- `error.log` – Standardní chybový výstup.

Složka `PROCESSES`

Adresář obsahuje složky s názvy procesů ve formátu `ID-PROCESU_NÁZEV-PROCESU`. Více informací v kapitole 3, kde je struktura podrobně popsána.

Ostatní

Kořenová složka dále obsahuje soubory `Framework.sh`, ve kterém jsou implementovány funkce kapitoly 2.3 a soubor `GLOBAL_CONFIG.sh`, který obsahuje základní konstanty a „makra“ pro běh programu:

- Konstanty umístění složek `DATA`, `LIBS`, `PROCESSES` a `LOG`. Dále pak konstanty doby čekání na další tokeny.
- Funkci pro nastavení logovacích souborů.
- Obalové funkce pro základní operace (`mkdir`, `rm`, `mv`, `cp`, `eval`) ošetřené chybovým hlášením.

[4] AWK je interpretovaný počítačový jazyk, navržený pro zpracovávání textových dat, ať už v podobě textových souborů nebo proudů. Název AWK je odvozen z příjmení svých tvůrců, kterými jsou Alfred V. Aho, Peter J. Weinberger a Brian W. Kernighan. Viz `http://en.wikipedia.org/wiki/AWK`.

2.3 Funkce

Funkce frameworku by měly poskytnout uživateli maximální komfort při obsluze. K dispozici jsou tyto funkce:

- **Spuštění všech procesů** (`-S`) – Program s tímto parametrem spustí funkci `start_all_processes`, která projde složku `PROCESSES`, vybere čísla procesů a u každého zavolá framework s parametrem `-s` a číslem procesu. Framework je spuštěn na pozadí s příkazem `nohup`, což znamená, že bude ignorovat HUP[5] signál. Dále je nastaven standardní a chybový výstup do souborů pro log u aktuálních procesů.
- **Zastavení všech procesů** (`-X`) – Tato volba zastavuje procesy v předem stanovených úsecích – pokud proces čeká na token nebo pokud je aktuálně zpracovávaný token právě ukončen. Program spustí funkci `stop_all_processes`, která vloží `STOP` token do složky `PROCESSES/process_n/A_STATE`.
- **Ukončení všech procesů** (`-K`) – Oproti předchozí možnosti tato volba ukončuje všechny procesy v jakékoli fázi zpracování. Program volá funkci `kill_processes`, která pomocí příkazu `ps`[6] zjistí všechny id procesů začínající řetězcem `"Framework.sh -s"`. Následně jsou tyto procesy ukončeny příkazem `kill`.
- **Restart všech procesů** (`-R`) – Všechny procesy jsou ukončeny a opět spuštěny. Tato volba volá dvě dříve popsané funkce `kill_processes` a `start_all_processes`.

```
TOKEN_NAME
├── process_n
│   ├── CONFIG/config
│   ├── DATA
│   │   ├── INPUT
│   │   ├── OUTPUT
│   │   └── TMP
│   ├── LOG
│   └── STATE
└── GLOBAL
    ├── CONFIG/config
    ├── DATA
    ├── LOG
    └── STATE
```

Obrázek 2.2: Adresářová struktura dat tokenu.

- **Spuštění procesu** (`-s PROCESS_NUMBER`) – Tato možnost spouští proces zavoláním funkce `start_proces`, která pouze nastaví proměnné `PROCESS_ID` a `PROCESS_NAME` a načte skript `process_handler.sh`, viz kapitola 2.4.
- **Spuštění procesu na pozadí** (`-d PROCESS_NUMBER`) – Tato volba volá funkci `start_process_bg`, která spouští framework s parametrem `-s` na pozadí opět pomocí příkazu `nohup` s přesměrováním standardních výstupů do logovacích souborů.

[5] SIGHUB je signál zaslaný procesu pokud je terminál, který tento proces spustil, právě uzavřen.
[6] Process status – poskytuje informace o procesech běžících v paměti.

- **Zastavení procesu** (`-x PROCESS_NUMBER`) – Program spustí funkci `stop_process`, která vloží `STOP` token do složky `PROCESSES/process_n/A_STATE`. Proces se buď ihned zastaví nebo až když dokončí zpracování aktuální úlohy.
- **Ukončení procesu** (`-k PROCESS_NUMBER`) – Tato volba volá funkci `kill_process`, kde si zjistí id procesu a následně proces(y) ukončí příkazem `kill`.
- **Restart procesu** (`-r PROCESS_NUMBER`) – Zvolený proces je ukončen a znovu spuštěn. Tato volba volá dříve popsané funkce `kill_process` a `start_process_bg`.
- **Vytvoření adresářové struktury pro data tokenu** (`-d TOKEN_NAME`) – Program vytvoří ve složce `DATA` adresářovou strukturu pro daný token. Viz obrázek 2.2. Konfigurační soubory ponechává prázdné. Adresáře se vytváří pro všechny dostupné procesy. Tato volba není povinná pro nový token.
- **Vytvoření kostry nového procesu** (`-c PROCESS_NAME [PROCESS_FOREWARD]`) – Tato volba velmi usnadní vytváření nových procesů. Nejprve vytvoří adresářovou strukturu se soubory. Viz obr 2.3. Tyto soubory jsou předvyplněny tak, že uživatel pouze zadá jaký soubor se má na vstupu kontrolovat (předvyplněn je soubor `file_to_process.txt`), poté zadá část exekutivní do souboru `main_exec.sh` a nakonec specifikuje jaký soubor se má přeposlat ostatním procesům v souboru `main_foreward.sh`.

```
PROCESS_NAME
├── A_DONE
├── A_ERR
├── A_RUN
├── A_STATE
├── A_WAIT
├── CONFIG/default
├── LOG
├── SCRIPTS
├── main_exec.sh
├── main_foreward.sh
├── main_check.sh
└── main_makedirs.sh
```

Obrázek 2.3: Adresářová struktura procesu.

- **Spuštění více instancí procesu** (`-i PROCESS_COUNT`) – Použitelné pouze s `-b` parametrem, který spouští procesy na pozadí. Proměnná `PROCESS_COUNT` udává počet spuštěných procesů.
- **Vytvoření šablony nastavení z tokenu** (`-e "TOKEN TEMPLATE"`) – Vytvoří šablonu nastavení z předaného tokenu a uloží ji pod zadaným názvem.
- **Kopie konfiguračních souborů z šablony** (`-f "TEMPLATE TOKEN"`) – Tato volba zkopíruje nastavení procesů v šabloně do předaného tokenu.
- **Zobrazení stromu přeposílání dat** (`-T`) – Program prochází všechny procesy a načítá konfigurační soubory. Z těchto souborů extrahuje informace o procesech, kterým

se přeposílá výstup (proměnná `PROCESS_FORWARD_TOKEN_TO` a dále informaci o procesech, na který musí aktuální proces čekat (proměnná `PROCESS_WAIT_FOR`). Výstup je ve formátu `JSON`[7], který je snadno zpracovatelný zejména ve webové aplikaci. Ukázka na obrázku 2.4.

```
{
  "P0402_slidesvideosync": [
    "forward": ["P0403_slidescorrection"],
    "waitfor": ["P0202","P0401","P0502"]
  ]
}
```

Obrázek 2.4: Ukázka výstupu.

- **Zobrazení stromu přeposílání dat konkrétního tokenu** (`-t TOKEN_NAME`) – Oproti předchozí možnosti tato volba navíc načítá konfigurační soubory daného tokenu, které mohou výstup upravit. Výstup je opět ve formátu `JSON`.
- **Zobrazení běžících procesů** (`-p`) – Framework zavolá proces `ps`, kde vyhledáme procesy začínající na `Framework.sh -s` a vypíšeme jejich parametr, což je id procesu.
- **Zobrazení nápovědy** (`-h`) – Nápověda.

2.4 Obsluha procesů

Hlavním prvkem celého bash frameworku je procedura `process_handler.sh`. Jedná se o „obalovou“ funkci pro spouštění procesů. Všechny kroky, které zde popisuji jsou logovány.

Prvním krokem je načtení hlavního konfiguračního souboru `GLOBAL_CONFIG.sh`. Dále jsou nastaveny soubory pro logování – v této chvíli do globální složky `/LOG`. Následně je kontrolována proměnná `PROCESS_RUNINLOOP`, která nám určuje, zda bude proces spuštěn v nekonečné smyčce. Pokud není nastavena, automaticky se načte hodnota `YES` z konfiguračního souboru.

Následně vstupuje proces do nekonečné smyčky. Dané kroky algoritmu:

1. Nejprve se načítají tokeny do fronty skriptem `list_waiting_tokens.sh`, který zobrazí soubory uložené ve složce `A_WAIT`.
2. Následně jsou tyto tokeny zpracovávány po jednom ve smyčce následovně:
   (a) Kontrola, zda token má svou podložku v adresáři `/DATA`.
   (b) Načtení konfiguračních souborů (procedura `process_config_load.sh`):
      i. Globální konfigurace tokenu ve složce `DATA/token_n/GLOBAL/CONFIG` – nepovinná.

[7] JavaScript Object Notation (JavaScriptový objektový zápis) je způsob zápisu dat (datový formát) nezávislý na počítačové platformě, určený pro přenos dat, která mohou být organizována v polích nebo agregována v objektech. Vstupem je libovolná datová struktura (číslo, řetězec, boolean, objekt nebo z nich složené pole), výstupem je vždy řetězec. Viz `http://www.json.org/`.

ii. Globální konfigurace spuštěného procesu ve složce `PROCESSES/process_n/ /CONFIG` – povinná.

iii. Lokální konfigurace procesu od zpracovávaného tokenu ve složce `DATA/ /token_n/process_n/CONFIG` – nepovinné.

(c) Kontrola, zda existuje `STOP` token ve složce `A_STATE`. Pokud ano, je proces ukončen a `STOP` token odstraněn.

(d) Proměnná `PROCESS_FORCED_CLEAR` – pokud je nastavena, provede se smazání dat aktuálního tokenu, nehledě na to, zda již byl ukončen nebo ne.

(e) Proměnná `PROCESS_FORCED` – pokud není nastavena a proces byl již dříve ukončen, tak by byl token přesunut do složky `A_DONE`, zapsán čas ukončení a byla by zavolána funkce pro přeposlání tokenu (`process_forward`). Následoval by návrat na bod 2. V opačném případě algoritmus pokračuje dále.

(f) Vytvoření složek tokenu zavoláním funkce `process_makedirs`, popsána níže.

(g) Spuštění kontroly tokenu pomocí funkce `process_check`, taktéž popsána níže.

(h) Vyhodnocení předchozí kontroly kontroly skriptem `can_token_be_run.sh`, který pouze zkontroluje obsah tokenu a vrátí jeho hodnotu. Pokud je tato hodnota `YES` proces pokračuje, jinak jdi na krok 2.

(i) Token je přesunut do složky pro právě zpracovávané tokeny – `A_RUN`.

(j) Spuštění procesu daného tokenu pomocí funkce `process_run`, která je popsána níže.

(k) Pokud byl proces ukončen řádně, pokračuj. V opačném případě je token přesunut do složky `A_ERR` a zapsán čas chyby. Poté pokračuj na krok 2.

(l) Proměnná `TOKEN_RUN_FOREVER` – pokud není nastavena, tak je proces přesunut do složky `A_DONE`, zapsán čas ukončení procesu a přesměrování tokenu pomocí funkce `process_forward`. Pokud tato proměnná je nastavena, přesuneme token zpět do fronty `A_WAIT`.

(m) Jdi zpět na krok 2. Pokud je fronta tokenů prazdná, pokračuj krokem 3.

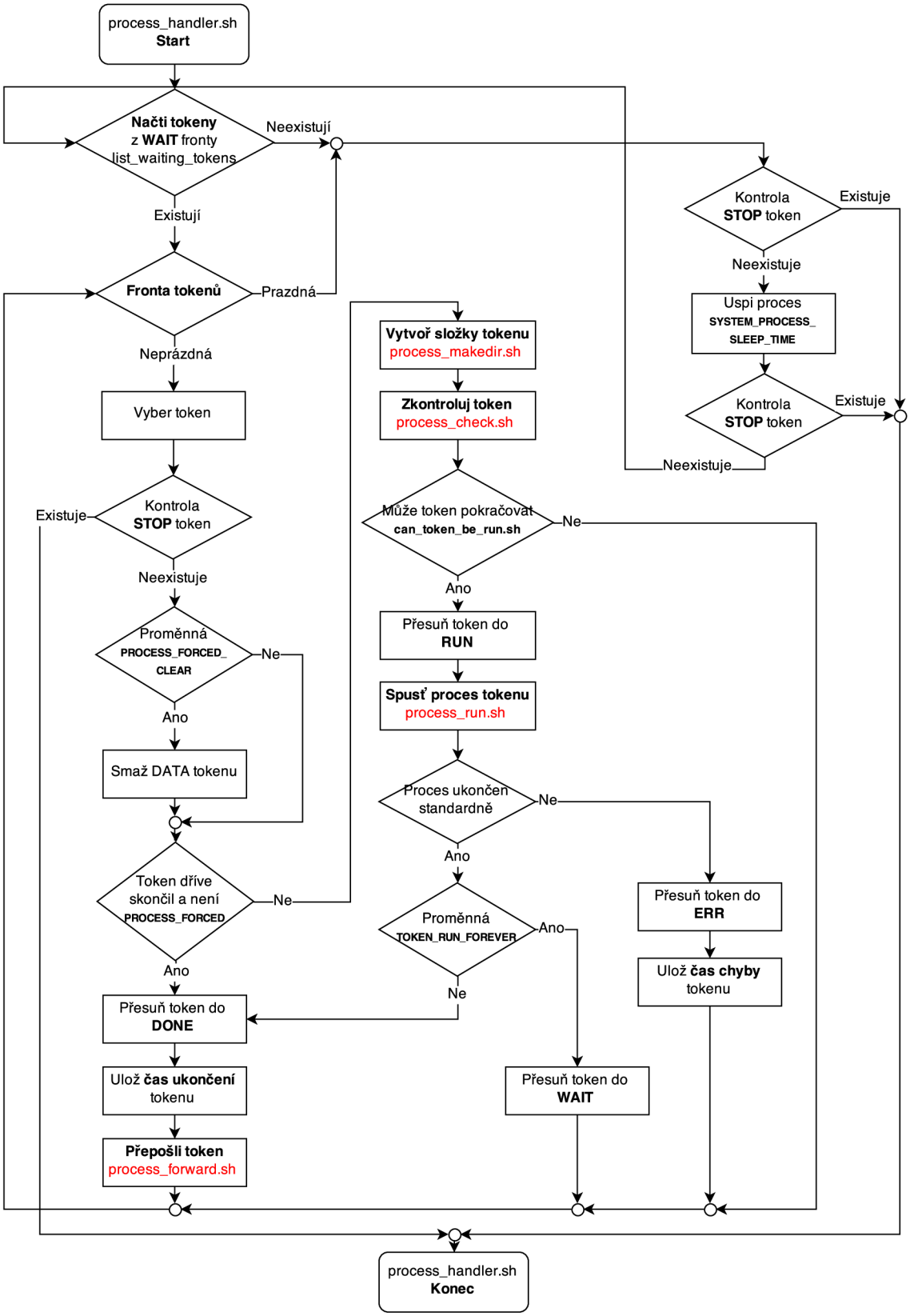
3. Kontrola, zda existuje `STOP` token ve složce `A_STATE`.

4. Pokud není nastavena proměnná `PROCESS_RUNINLOOP`, tak uspi proces na `SYSTEM_PROCESS_SLEEP_TIME` (Základní nastavení je 1m). V opačném případě je proces ukončen.

5. Opětovná kontrola `STOP` tokenu.

6. Jdi zpět na krok 1.

Pro znázornění je celý tento algoritmus zobrazen na obrázku 2.5.

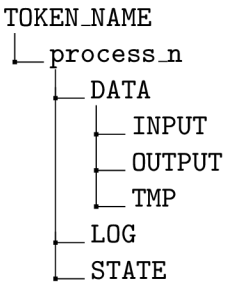


Obrázek 2.5: Vývojový diagram procedury `process_handler.sh`

Funkce `process_makedirs`

Tato funkce spouští skript `process_makedirs.sh` se zadanými proměnnými. Nejprve jsou načteny konfigurační soubory stejně jako v hlavním procesu – globální konfigurace, globální konfigurace tokenu, globální konfigurace procesu a lokální konfigurace procesu daného tokenu. Dále jsou nastaveny logovací soubory.

V hlavní části tento skript vytvoří základní adresářovou strukturu ve složce `/DATA` pro aktuálně zpracovávaný token – viz obrázek č. 2.6. Dále je načtena procedura `main_makedirs.sh` aktuálního procesu, ve které se vytváří složky dle libosti.



Obrázek 2.6: Adresářová struktura dat tokenu.

Kdyby tento skript skončil chybou neovlivní to zpracování dalších tokenů. Právě proto není skript načítán, ale spouštěn.

Funkce `process_check`

Obdobně jako v předchozí funkci je spuštěn skript `process_check.sh`. Dále je zde implementována metoda `process_check_wait_for`, která zkontroluje podle nastavení, zda daný proces nečeká na stejný token z více procesů. Jedná se o proměnnou `PROCESS_WAIT_FOR`. Samotná kontrola je pak volána z načtené procedury `main_check.sh`.

Základní kostra každé procedury by měla být následující:

1. Proměnná `PROCESS_FORCED_CHECK` – pokud je nastavena na `YES`, tak je kontrola přeskočena a proces pokračuje ve zpracování.
2. Metoda `process_check_wait_for` – kontrola čekání na procesy.
3. Libovolná kontrola vstupních souborů.

Pokud vše proběhne v pořádku, je do tokenu zapsána hodnota `YES`, která se dále využívá ve skriptu `can_token_be_run.sh`.

Funkce `process_run`

Tato funkce spouští skript `process_run.sh`. Na rozdíl od předchozích funkcí se zde navíc načítá knihovna `progress_lib.sh`, která má na starost zaznamenávání postupu zpracování. Hlavní procedura každého procesu `main_exec.sh` se načítá podle proměnné `PROCESS_SKIP`, kterou můžeme nastavit v případě, pokud chceme pouze zopakovat konkrétní část z procesů od tokenu.

Funkce `process_foreward`

Poslední část zpracování procesu tokenu obstarává skript `process_forward`. K dispozici máme implementované metody `forward_token_or_error` a `forward_output_or_error`, které využíváme v načtené proceduře `main_forward.sh`.

Procedura `main_forward.sh` v základním nastavení prochází proměnnou `PROCESS_FORWARD_TOKEN_TO` a přeposílá token dalším procesům pro zpracování. Metoda `forward_output_or_error` data nekopíruje, pouze přeposílá umístění dat v souboru.

2.5 SGE

V průběhu vývoje vznikl požadavek na podporu SGE[8]. Pokud je použití SGE povoleno (proměnná `SGE_ENABLED`), framework zaobaluje veškeré příkazy a odesílá je do fronty pro zpracování. Ve většině případů se jedná o neblokující operaci. S každou úlohou se odesílá příkaz pro případ chyby ve skriptu, který by token přenesl do chybové fronty.

Pokud v rámci jednoho procesu odesíláme více příkazů na zpracování, které jsou na sobě závislé, jsou nastaveny parametry tak, aby druhý příkaz na první počkal. S posledním příkazem se odesílá skript, který v případě úspěchu spustí přeposlání tokenu dalším procesům. Pokud SGE vypneme, procesy spouští příkazy klasicky – synchronně.

K dispozici u každého příkazu můžeme nastavit tyto parametry:

- `SGE_JOB_NAME` – název úlohy.
- `SGE_DEPENDENCY` – určuje, zda má úloha čekat na některou jinou úlohu.
- `SGE_PRIORITY` – nastaví prioritu úlohy.

[8] Sun Grid Engine, nově Oracle Grid Engine (OGE) – systém pro správu dávkového zpracování úloh.

Kapitola 3

Atomické procesy

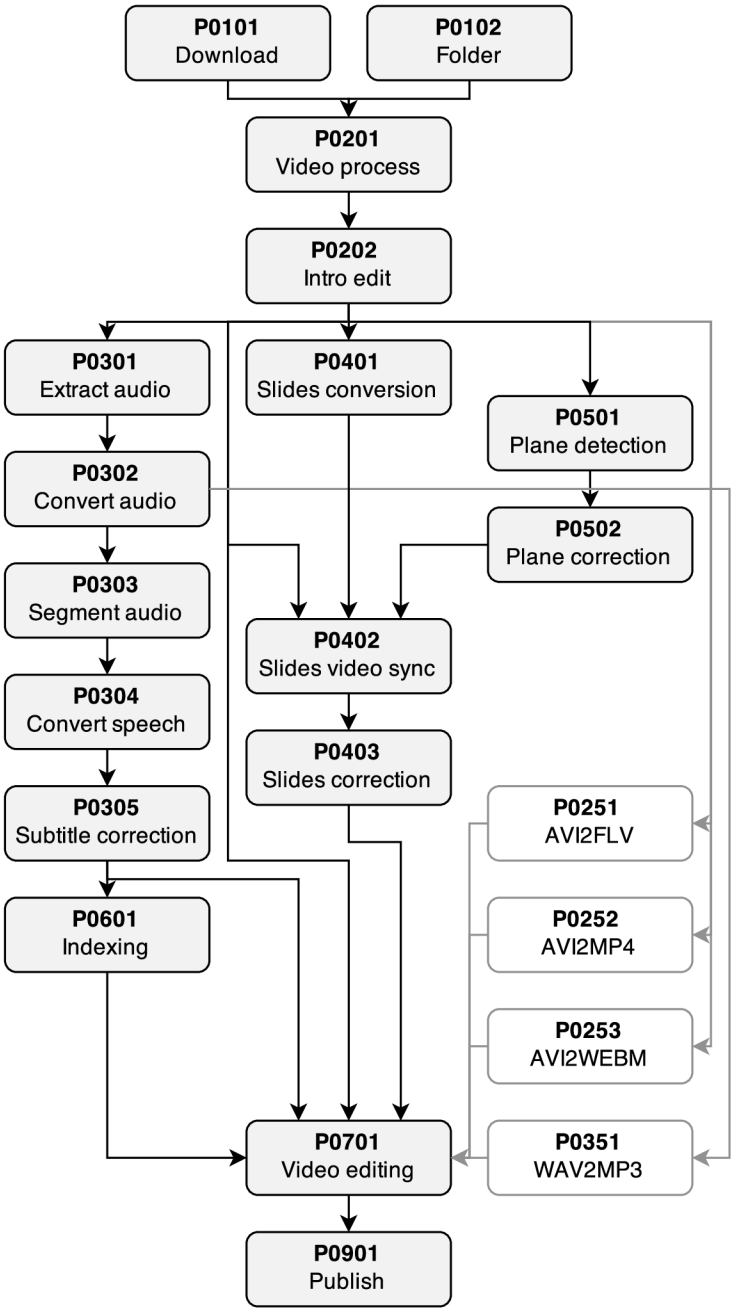
Procesy, uložené ve složce `/PROCESSES`, mají pevně daný formát „proces-ID_proces-název“. Struktura procesu je zobrazena na obrázku č. 2.3.

Adresář procesu obsahuje:

- `A_DONE` – Ukončené tokeny.
- `A_ERR` – Chybné tokeny.
- `A_RUN` – Aktuálně zpracovávaný token.
- `A_STATE` – Složka pro `STOP` token.
- `A_WAIT` – Čekající tokeny.
- `CONFIG` – Složka s konfiguračními soubory.
- `LOG` – Složka s logy.
- `SCRIPTS` – Složka s pomocnými skripty.
- `main_exec.sh` – Část spuštění procesu.
- `main_foreward.sh` – Přeposlání procesu.
- `main_check.sh` – Kontrola před spuštěním procesu.
- `main_makedirs.sh` – Vytvoření adresářů.

Každý token „probublává“ strukturou procesů až k poslednímu procesu `publish`, který video zveřejní. Struktura je zobrazena na obrázku č. 3.1. Jedná se o základní nastavení, které lze měnit proměnnou `PROCESS_FORWARD_TOKEN_TO` každého procesu.

V následujících kapitolách popíši tři zajímavé procesy. Jendá se o proces `P0102`, který automaticky zpracovává soubory v dané složce. Poté proces `P0201`, který převádí video do požadovaného formátu a v poslední řadě proces `P0501`, který provádí detekci plátna ve videu. Detailní popis ostatních procesů nalezneme v příloze C.



Obrázek 3.1: Diagram procesů – konfigurace možného propojení.

3.1 P0102 Folder

Tento proces je mírně odlišný od ostatních procesů, protože jako jediný vytváří – generuje tokeny a navíc, je tento proces opakován v rámci jednoho tokenu – viz sekce spuštění. Úlohou tohoto procesu je vytvoření tokenů ze souborů ve vstupní složce a přeposlání těchto tokenů k dalšímu zpracování. Základní rozhraní tohoto procesu je následující:

- `VSTUP` — Adresa požadované složky, uložená v souboru `file_to_process.txt` ve

složce `DATA/token_n/GLOBAL/DATA/INPUT` (globální data pro aktuální token).

- `VÝSTUP` — Standardní výstup tento proces nemá, ale generované tokeny za výstup můžeme považovat.

Sekce kontroly

Program bude pokračovat jedině za předpokladu, že bude soubor `file_to_process.txt` ve složce `DATA/token_n/GLOBAL/DATA/INPUT` nenulový.

Sekce spuštění

Tento proces prochází soubory ve vstupní složce a generuje z nich tokeny pro další zpracování. Jak již bylo avizováno, toto se děje opakovaně, což je umožněno nastavením proměnné `TOKEN_RUN_FOREVER` na hodnotu `YES`. Nastavení přikáže frameworku, aby při dokončení zpracování opět token zařadil do čekací fronty a je zrušená sekce přesměrování. Pokud ale kontrolujeme opětovně stejnou složku, tak si musíme zaznamenat, kde jsme v předchozím zpracování skončili. Pro tento případ slouží soubor `timestamp.txt`, který se nachází ve složce `DATA/token_n/GLOBAL/DATA/TMP` (globální dočasná data pro aktuální token).

Prvním krokem je kontrola, zda soubor `timestamp.txt` existuje. Následně je buď vytvořen nebo je vybrán obsah tohoto souboru do proměnné, jako časová známka posledního souboru, který byl zpracován.

Následně vyberu všechny soubory ze složky na vstupu a seřadím si je podle časové známky editace. V průchodu je porovnávám s mou uloženou časovou známkou a pokud se objeví novější soubor, zpracuji jej.

S takovýmto souborem je zacházeno jako s novým tokenem. Název tokenu je složeninou z názvu souboru a časové známky. Nejprve je vytvořena datová struktura tokenu. Následně je do této struktury zapsán proces `P0102_folder` a označen jako dokončený (vložením `state.done` do složky `STATE` daného procesu).

Dalším krokem je kontrola, zda u tokenu není konfigurační soubor (musí mít stejný název s příponou `.conf`). Pokud tento soubor existuje, je načten a jeho data jsou zapsány do globální složky nového tokenu. Aktuálně načítám tyto položky – třídu, rok, verzi a „rodičovský token“ (popsaný v podkapitole č. 4.4.2). Tyto informace se dále používají při synchronizaci s webovým rozhraním.

Posledním krokem je suplování přesměrování tohoto vytvořeného procesu – umístění vstupního souboru je uloženo do souboru a zasláno na vstup do složky `DATA` procesu `P0201 video process`.

Sekce přesměrování

Jak již bylo napsáno, proces je volán opakovaně s aktuálním tokenem, tudíž je sekce přesměrování přeskočena. Nicméně nastavení přesměrování je použito v sekci spuštění, kde jsou podle toho přesměrovány nově vytvořené tokeny.

3.2 P0201 Video process

Úkolem tohoto krátkého procesu je pouze konverze vstupního videa do jednotného kontejneru – v našem případě `mkv`[1]. Tento proces má ještě další úlohu a to zastavení tokenu pro webovou aplikaci. V takovém případě proces získá z videa potřebné informace a čeká na uživatelův pokyn pro pokračování. Základní rozhraní tohoto procesu je následující:

- `VSTUP` — Adresa požadovaného videa, uložená v souboru `file_to_process.txt` ve složce `DATA/token_n/process_n/DATA/INPUT`.
- `VÝSTUP` — Soubor `video.mkv` ve výstupní složce `DATA/token_n/process_n/DATA/OUTPUT`

Sekce kontroly

Program bude pokračovat jedině za předpokladu, že bude soubor `file_to_process.txt` ve složce `DATA/token_n/process_n/DATA/INPUT` nenulový.

Sekce spuštění

Standardně tento proces načte vstupní video libovolného formátu a pomocí programu `avconv` jej převede do jednotného kontejneru souboru `video.mkv`. Program má parametr `-codec:v copy` a `-codec:v copy`, což znamená, že je kodek video a audio stopy je kopírován ze vstupu na výstup. Dále má parametr `-f mkv`, který určuje typ multimediálního kontejneru.

Pokud je ale u tokenu nastavena proměnná `TOKEN_RUN_FOREVER`, tak se předchozí zpracování přeskočí. Dále je volán program `avprobe`, který zjistí následující informace z videa, potřebné pro další zpracování – počet snímků za sekundu a délku videa. Dále se exportují programem `avconv` tři náhledy z videa. Tyto data se ukládají do globální složky tokenu. Jakmile je token synchronizován objeví se učiteli ve videích ke zpracování. Po té, co učitel toto video nastaví, aplikuje se některá z šablon nastavení na tento token a proměnnou `TOKEN_RUN_FOREVER` odstraní, čímž může proces pokračovat.

Sekce přesměrování

Proces je přednastaven k zaslání výstupu (respektive souboru, který obsahuje cestu výstupu) do složky `DATA` procesu `P0202 intro edit`.

3.3 P0501 Plane detection

Úlohou tohoto procesu je detekce plátna ve videu. Základní rozhraní tohoto procesu je následující:

- `VSTUP` — Umístění konvertovaného videa, uloženého v souboru `intro_video.txt` ve složce `DATA/token_n/process_n/DATA/INPUT`.
- `VÝSTUP` — Souřadnice detekovaného plátna a ostatní informace (viz sekce spuštění) v xml souboru `coordinates.xml` ve složce `/token_n/process_n/DATA/OUTPUT`.

[1] Matroska pod zkratkou `mkv` je moderní otevřený svobodný multimediální kontejner (podobný AVI), který umožňuje pojmout většinu moderních video a audio formátů. Dokáže též pojmout několik různých audio stop včetně prostorového zvuku. Viz `http://en.wikipedia.org/wiki/Matroska`.

Sekce kontroly

Program bude pokračovat jedině za předpokladu, že bude soubor `intro_video.txt` ve složce `DATA/token_n/process_n/DATA/INPUT` nenulový.

Sekce spuštění

Veškeré zpracování obstarává program `det_plane` s následujícími parametry – vstupní videosoubor, výstupní xml soubor s koordináty plátna a parametr `-s`, který značí počet přeskočených snímků ze začátku videa.

Program `det_plane` používá knihovnu OpenCV. Nejprve načte vstupní video soubor. Jednotlivé n-té snímky videa jsou v cyklu zpracovány následovně:

1. Snímek je rozmazán Gaussovým filtrem a podvzorkován.
2. Převod snímku do odstínů šedi.
3. V dalším cyklu je provedeno postupné prahování v rozsahu $< 25, 255 >$ s krokem 25. Tyto hodnoty se dají nastavit ve vstupním konfiguračním souboru.
4. V takto prahovaných snímcích jsou vyhledány obrysy.
5. U každého obrysu je vypočtena plocha a pokud je v rozsahu 5-ti až 100% velikosti snímku, tak pokračujeme. Tyto hodnoty můžeme taktéž upravit v konfiguračním souboru.
6. Následně jsou body obrysu aproximovány Ramer-Douglas-Peucker algoritmem[2]. Nejprve s nízkou maximální vzdáleností (4) od eliminovaných bodů. Dále proběhne kontrola, zda je počet nově aproximovaných bodů v intervalu $< 4, 12 >$. Pokud ano, body jsou opět aproximovány, ale tentokrát s vysokou maximální vzdáleností (50) od eliminovaných bodů.
7. Výsledkem těchto aproximací by měl být obrys, který je definován 4-mi body – hledané plátno. Pokud tedy počet souhlasí, obrys je uložen.

Jakmile máme nalezené obrysy pro každý n-tý snímek jsou tyto obrysy rozděleny do skupin podobných obrysů (v ideálním případě pouze 1 skupina – 1 plátno) a nad těmito skupinami je proveden Gaussův filtr pro výsledný obrys. Dále je provedeno spojování přilehlých obrysů a odfiltrování překrytých regionů, které se překrývají více jak 70% (lze nastavit v konfiguračním souboru).

Posledními kroky jsou zpětné převzorkování nalezených regionů a uložení výsledků do souboru podle parametru, v našem případě `coordinates.xml`.

Sekce přesměrování

Proces je přednastaven k zaslání výstupu (respektive souboru, který obsahuje cestu výstupu) do složky `DATA` procesu `P0502 plane correction`.

[2]Ramer-Douglas-Peucker je algoritmus pro redukci počtu bodů v křivce, která je aproximována množinou bodů. Algoritmus je také znám pod názvy Douglas-Peucker, iterative end-point fit algoritmus nebo split-and-merge algoritmus. Základní forma algoritmu byla navržena v roce 1972 Urs Ramrem a v roce 1973 Davidem Douglasem a Thomasem Peuckerem [12, 4].

Kapitola 4

Webové rozhraní

Tato část projektu zastřešuje Bash framework. Pro jednotlivé skupiny uživatelů poskytuje maximální komfort pro používání. Systém bude obsluhovat tyto skupiny uživatelů:

- **Běžný návštěvník** – poskytuje přehrávání veřejných videí a vyhledávání.
- **Přihlášený uživatel** – studentovi navíc poskytuje uložení videí k pozdějšímu přehrání, „sledování“ předmětu pro nové videa, přidávání a hodnocení komentářů a historii přehrávání.
- **Učitel** – systém nabízí zejména správu nově nahraných přednášek, kde je potřeba doplnit informace k videu a zkontrolovat nastavení před zpracováním. Proces, který čeká na vstup od učitele je P0201. Systém nabídne i řadu „předpřipravených“ šablon nastavení. Jakmile učitel schválí video, token (popřípadě více tokenů, pokud má záznam dva zdroje nahrávání) pokračuje dále ve *stromu procesů*. Popřípadě může učitel nahrát své vlastní video do systému nebo pouze zadat URL s videem o kterou se postará proces P0101. Zpracování se zastaví dále u procesu P0701, kde bude mít učitel možnost *vystřihnout* část videa a vytvořit z ní samostatný úsek nebo zkontrolovat jednotlivé části celkového procesu a popřípadě je upravit (časování prezentace nebo časování titulků).
- **Administrátor** – má dozorovou kontrolu nad všemi procesy, může kontrolovat a nastavovat tokeny a k nim přiřazené videa. Ukázku vygenerovaného stromu procesů dle aktuální konfigurace pro daný token můžeme vidět pod titulkem práce – vygenerováno za pomocí js frameworku Raphaël[1]. Dále vidí veškeré logy jednotlivých atomických úloh, stavy procesů a tokenů. Administrátor dále přiřazuje učitelům předměty a nastavuje výčtové tabulky v databázi. Spravuje také nastavení práv v systému, které popisuji v kapitole č. 4.4.5.
- **Super administrátor** – může navíc spouštět, restartovat a vypínat procesy (viz obrázek č. 4.8) a rozšiřovat / kompilovat překlady.

4.1 Výběr frameworku

Systém je naprogramován v jazyce PHP. Tento skriptovací jazyk jsem si vybral z důvodů dřívějších zkušeností, široké škále podpory, doplňků a snadné dostupnosti. Aplikace je na-

[1] Raphaël framework usnadňuje práci s vektorovou grafikou. Využívá k tomu SVG, každý objekt je vytvořen také jako DOM objekt. Viz `http://raphaeljs.com/`.

psána s důrazem na objektový přístup. Při implementaci jsem čerpal z [6, 3].

Jako první krok při vytváření aplikace byl výběr frameworku. Na základě vytváření předchozích projektů jsem vyžadoval alespoň tyto tři funkce, které by framework měl umět:

- **Routování** – Snadné přidávání stránek, podpora různých HTTP requestů, snadná administrativa, podpora sémantických URL nebo také „Cool“ URL.
- **Šablonovací systém** – Dědičnost šablon, cachování šablon, předkompilace.
- **Lokalizace** – správa překladů.

V následujících kapitolách se věnuji jednotlivým frameworkům, které by byly vhodné pro implementaci. Popisuji jejich funkce, výhody a nevýhody.

Yii framework

Jedná se o open-source PHP framework určený pro vývoj rozsáhlých webových aplikací. Zkratka Yii je akronym pro „Yes, It Is!“. Klade důraz na pragmatičnost, znovupoužitelnost a jednoduchost použití.

Yii je striktně objektově orientovaný systém, každá komponenta frameworku je nezávislá, konfigurovatelná a rozšiřitelná. Podpora DRY[2] designu a RAD[3] přístupu [13]. V současné době je ve verzi 2.0.3, která byla publikována 1. března 2015. Vlastnosti frameworku:

- Rozdělení aplikační a prezentační logiky na základě architektonického principu MVC, který umožňuje relativní nezávislost prezentační, aplikační a databázové vrstvy aplikace a tedy jejich lepší správu.
- Přístup ke zdrojům dat pomocí Database Access Objects (DAO) a Active Record.
- Integruje jQuery – oblíbenou javascriptovou knihovnu.
- Jednoduché a bezpečné zpracování formulářů a validace vstupních dat.
- Autentizace a autorizace – kontrola přístupu na základě hierarchických rolí (RBAC).
- Podpora motivů a rychlé visuální změny aplikace.
- Webové služby – automatické generování specifikací a zpracování požadavků WSDL služeb.
- Lokalizace (L10N) a internacionalizace (I18N) – překlad zpráv, formátování čísel a údajů o datu a čase.
- Vrstvená cache pro data, stránky, fragmenty stránek a dynamický obsah, která významně snižuje reakční čas aplikace.
- Zpracování, archivování a filtrace chyb.
- Zabezpečení a odolnost aplikace vůči různým druhům útoku.
- Generování validního HTML kódu.

[2] Don’t repeate yourself – snaha o neopakování částí kódu.

[3] Rapid application development – metodologie vývoje, spadá pod agilní techniky, vhodný pro vývoj na základě uživatelského rozhraní.

Jedná se o rozsáhlý a robustní systém pro velké projekty velkého rozsahu. Pro naše potřeby je tato skutečnost spíše nevýhodou. Jako nevýhodu považuji to, že je programátor odkázán na veliké množství modulů a je příliš odstíněn logiky zpracování požadavků.

CakePHP

Open-source webový aplikační framework. Založen na MVC[4] architektuře a konceptech frameworku Ruby on Rails (napsán v jazyce Ruby). Distribuován pod licencí MIT. V současné době je ve verzi 3.0.3 vydána 3. května 2015. Nevýhodou je chybějící šablonovací systém.

Symfony

Symfony je webový aplikační framework pro vývoj webových aplikací pro PHP 5 vycházející z architektury MVC. Celý framework je z velké části inspirován jinými webovými aplikačními frameworky jako Ruby on Rails, Django a Spring. Symfony je open-source, je vydáván pod MIT licencí.

V současné době je ve verzi 2.6 vydána 29. listopadu 2014. Taktéž jako CakePHP, framework neobsahuje šablonovací systém.

Zend framework

Open source, objektově orientovaný, webový aplikační framework implementovaný v PHP 5 a licencovaný pod New BSD license. Zend Framework je vyvíjen s ohledem na jednoduchý vývoj webových aplikací.

Užívá modulární architektury která umožňuje vývojářům použít jen ty komponenty, které potřebují. Částečné závislosti mezi komponentami však existují. Zend Framework v sobě zahrnuje komponenty pro MVC aplikace, autorizaci a autentizaci, implementuje různé druhy cache, filtrů a validátorů pro uživatelská data, jazykové komponenty a mnoho dalších. V současné době je ve verzi 2.4.1 vydána 7. května 2015. Vlastnosti frameworku:

- Všechny komponenty jsou plně objektově orientované a vyhovují direktivě `E_STRICT`.
- Modulární architektura typu „užij-co-potřebuješ“ minimalizuje křížové závislosti mezi komponentami.
- Rozšiřitelná implementace MVC s podporou layoutů a šablonovacím systémem.
- Podpora pro multi-databázové systémy zahrnuje MySQL, Oracle, IBM DB2 MSSQL Server, PostgreSQL, SQLite a Informix Dynamix Server.
- Kompozice e-mailu a schopnost jej odeslat / přijmout skrze mbox, Maildir, POP3 nebo IMAP4.
- Flexibilní cache sub-systémy s podporou mnoha typů back-endů jako paměť nebo soubor.

Jedná je opět o robustní aplikační framework pro projekty většího rozsahu – obdobně jako Yii framework. Od verze 2.0 je kladen důraz na objektový přístup.

[4]Model-View-Controller – softwarová architektura, která rozděluje datový model aplikace, uživatelské rozhraní a řídicí logiku do tří nezávislých komponent tak, že modifikace některé z nich má jen minimální vliv na ostatní.

Nette framework

Open source framework pro tvorbu webových aplikací v PHP 5. Zaměřuje se na eliminaci bezpečnostních rizik, podporuje AJAX[5], DRY, KISS[6], MVC a znovupoužitelnost kódu. Využívá událostmi řízené programování a z velké části je založen na použití komponent.

Původním autorem Nette Frameworku je David Grudl, o jeho další rozvoj se stará organizace Nette Foundation. V současné době je ve verzi 2.3.2 vydána 6. března 2015. Vlastnosti frameworku:

- Zabezpečení a eliminace výskytu bezpečnostních děr jako je XSS[7], CSRF[8], session hijacking, session fixation atd.
- Mnoho ladících nástrojů, které pomohou zavčasu odhalit všechny chyby aplikace.
- Využití moderních technologií AJAX / AJAJ, Dependency Injection, SEO, DRY, KISS, MVC, Web 2.0, cool URL.
- Plně objektový návrh.
- Velká podpora pluginů a rozšíření.

Lavarel

Open source aplikační framework, postaven na MVC architektuře. Za roky 2014 a 2015 je označován za nejoblíbenější framework[9].

První vydání bylo v roce 2012, vychází z frameworků jako jsou Symfony nebo Composer. Mimo jiné také disponuje těmi nejověřenějšími postupy inspirovanými technologiemi jako je Ruby on Rails, ASP.NET MVC a Sinatra. Základní vlastnosti frameworku:

- autentifikace – kontrola přístupu uživatelů
- routování – správa, směrování a zpracování dotazů na jednom místě
- databáze – veškeré nástroje pro komunikaci s databází
- mail – posílání emailu s přílohami a vloženými soubory
- sessions – zastává veškeré agendy okolo sessions
- caching – kešování používaných dat

[5] Asynchronous JavaScript and XML – asynchronní zpracování dotazů bez nutnosti obnovení webové stránky.

[6] Keep it short and simple – obecný návrhový vzor, kde je kladen důraz na jednoduchost nežli komplexnost.

[7] Cross-site scripting – metoda narušení WWW stránek využitím bezpečnostních chyb ve skriptech (především neošetřené vstupy). Útočník díky těmto chybám v zabezpečení webové aplikace dokáže do stránek podstrčit svůj vlastní javascriptový kód.

[8] Cross-site Request Forgery – je jedna z metod útoku do internetových aplikací (typicky implementovaných skriptovacími jazyky nebo CGI) pracující na bázi nezamýšleného požadavku pro vykonání určité akce v této aplikaci, který ovšem pochází z nelegitimního zdroje.

[9] Viz `http://www.sitepoint.com/best-php-framework-2015-sitepoint-survey-results/` a `http://www.sitepoint.com/best-php-frameworks-2014/`.

Slim

První z kategorie mikro-frameworků, také se jedná o open-source projekt, publikovaný pod MIT licencí. Mezi jeho základní funkce patří:

- Routování – Poskytuje mapování na jakékoli HTTP požadavky, podpora segmentace adres.
- Middleware – Volitelné spouštění funkcí před nebo po požadavku.
- HTTP – Jednoduchá abstraktní vrstva nad požadavkem a odpovědí.
- Session – Podpora kryptování session dat.
- Cache – Obalové funkce pro snadnější práci s HTTP cache.
- Kryptace – Zabezpečení session a cookie.

Výhodami tohoto frameworku je malá velikost a to, že je velmi rychle pochopitelný kvůli jeho jednoduchosti. Nevýhodou je nutnost dodání modulů pro práci s databázi, s formuláři nebo například s šablonami.

Flight

Poslední framework, který se dostal do mého výběru je framework Flight. Taktéž spadá do segmentu mikro-frameworků. Obsahuje pouze router, jednoduchý šablonovací systém a abstraktní vrstvu nad požadavky.

Závěr

Prednasky.com jsem se rozhodl implementovat ve frameworku Slim. Vybral jsem si jej z několika důvodů:

1. Jedná se o mikro-framework – malá velikost kódu, neobsahuje části, které by nebyly využité.
2. Velmi snadný k pochopení, jednoduché použití – manuál obsahuje přibližně deset stran. Toto je také důležité při předávání projektu dalším správcům.
3. Objektový přístup, užitečné moduly.

Pokud bychom ale zvolili jiný framework, neudělali bychom chybu, všechny mají své určité kvality. Zvolení mikro-frameworku má také své nevýhody. Například musíme zajistit komunikaci s databází, dále zvolit šablonovací systém nebo knihovnu pro práci s formuláři.

4.2 Výběr šablonovacího systému

Právě šablonovací systém byla druhá věc, kterou bylo nutné si zvolit. Existuje mnoho těchto systémů. Jednu z pokročilých funkcí, kterou jsem vyžadoval byla dědičnost šablon. Do mého užšího výběru se dostaly následující položky.

Smarty

Jeden z prvních šablonovacích systému. Nyní ve verzi 3.1.21. Verze 3 byla přepsána do objektového návrhu. Na systému probíhá aktivní vývoj. Obsahuje největší množství různých funkcí a tagů. Z tohoto důvodu je také považován za jeden z pomalejších šablonovacích systémů.

Dwoo

Open-source systém. Aktuálně ve verzi 2.0. Nabízí více jak 60 bloků a funkcí. Je kompatibilní se Smarty šablonami. Byl vytvořen jako odpověď na zastaralý Smarty šablonovací systém v roce 2008. Nabízí plně objektový přístup. Nabízí podporu funkcí, bloků, vkládání šablon, podmínky, smyčky, přidělování proměnných, dědičnost šablon a obsluhu výjimek.

```
1  .<head>
2    <title>{block "title"}My site name{/block}</title>
3    {* comment css includes, etc. *}
4  </head>
5  <body>
6    <h1>{block "page-title"}Default page title{/block}</h1>
7    <div id="content">
8      {block "content"}
9        {date_format "now" "Y-m-j"}
10       <ul id="navigation">
11       {foreach $navigation item}
12         <li><a href="{$item.href}">{$item.caption}</a></li>
13       {/foreach}
14       </ul>
15     {/block}
16   </div>
17 </body>
```

Obrázek 4.1: Ukázka syntaxe šablonovacího systému Dwoo. Funkce frameworku `block` se používá při dědičnosti pro přepsání rodiče. Ostatní funkce voláme stejně jako v PHP viz řádek č. 9 – `date_format`.

Open Power Template

Taktéž open-source systém. Nyní ve verzi 2.0.6, která byla vydána 9. září 2010, což naznačuje ukončující vývoj. Byl inspirován Smarty projektem. Oproti Dwoo podporuje vykreslování formulářů a lokalizaci. Je možné využít funkce pro práci s XML, protože tento šablonovací systém z XML syntaxe vychází, viz obrázek č. 4.2.

Rain TPL

Jeden z nejmenších a také nejrychlejších šablonovacích systémů. Obsahuje pouze 8 tagů a 4 metody. Bohužel nepodporuje dědičnost šablon nebo uživatelské blokové funkce.

```
<?xml version="1.0" ?>
<opt:root escaping="yes">
  <opt:prolog version="1.0" />
  <opt:dtd template="xhtml10transitional" />
    <body>
      <p>{$introduction}</p> <!-- display a list -->
      <opt:show name="list">
      <ol>  <li opt:section="list">{$list.item}</li>  </ol>
      </opt:show>
    </body>
</opt:root>
```

Obrázek 4.2: Ukázka syntaxe šablonovacího systému Open Power Template.

Twig

Twig šablonovací systém vychází syntaxí z Jinja a Django systémů, viz 4.3. Opět se jedná o open-source systém, publikován pod BSD licencí. Je to přednastavený šablonovací systém ve frameworku Symfony. Obsahuje mnoho předdefinovaných tagů, filtrů a funkcí.

```
{% extends "base.html" %}
{% block navigation %}
    <ul id="navigation">
    {% for item in navigation %}
        <li>
            <a href="{{ item.href }}">
                {% if 2 == item.level %}  {% endif %}
                {{ item.caption|upper }}
            </a>
        </li>
    {% endfor %}
    </ul>
{% endblock navigation %}
```

Obrázek 4.3: Ukázka syntaxe systému Twig vycházející z Jinja syntaxe šablonovacího systému pro python.

Závěr

Šablonovací systém Dwoo jsem vybral pro tento projekt. Prvním důvodem byla úzká návaznost na PHP funkce – není zapotřebí vytvářet filtry jako v ostatních systémech (např. {{`var|upper`}}), namísto toho, je můžeme volat přímo – {`strtoupper $var`}. Systém Dwoo umožňuje definovat i vlastní funkce, pokud bychom to vyžadovali. Dále má přehlednější syntax oproti systému Open Power Template a vyhýbá se použití různých tagů (např. u systému Twig – `{ ... }`, `{{ ... }}` a `{% ... %}`).

Dále jsem se zaměřil na výkonnostní srovnání. V tabulce č. 4.1 můžeme vidět srovnání tří systémů. Testování probíhalo na jednoduché šabloně (jeden výstup a for smyčka nad třemi položkami), která byla děděna. Čas je průměrován deseti běhy. Jeden běh je definován jako kompilace a 10 000 zobrazení. Všechny proměnné byly escapovány. [11].

Šablonovací systém	Čas [s]	Paměť [ko]	Šablon renderovaných za sekundu
Twig	3	1,190	3,333
Dwoo	6.9	1,870	1,449
Smarty	14.9	3,230	671

Tabulka 4.1: Srovnání šablonovacích systémů. Převzato z [11].

Smarty vychází v testu nejhůře, může to být dáno z více důvodů, např. zastaralý design nebo také to, že se jedná o nejkomplexnější systém. Dwoo šablonovací systém vychází v porovnání dobře, i když není tak rychlý jako systém Twig, oproti tomu můžeme vidět nižší paměťovou náročnost.

Další srovnání je přímo od tvůrců systému Dwoo[10] – má spíše informativní charakter. V článku není popsaná metodika testování, pouze to, že se jedná o použití smyčky.

Šablonovací systém	Čas spuštění [ms]	Paměť [kB]
PHP	650	36
Rain TPL	4234	280
Twig	8134	666
Dwoo	10507	317
Smarty	10675	1024

Tabulka 4.2: Srovnání šablonovacích systémů podle. Převzato z[10].

V tabulce č. 4.2 můžeme vidět podle očekávání, že Rain TPL je nejrychlejší. Je to dáno tím, že se řadí k těm nejjednodušším. Dále výsledky odpovídají předchozímu srovnání v tabulce č. 4.1.

4.3 Ostatní komponenty

Jelikož jsme si zvolili mikro-framework Slim bylo by dobré si vybrat nějaký nástroj pro práci s databázi. Pro tuto úlohu jsem si zvolil dvě knihovny, které popisuji v následujících kapitolách.

4.3.1 RedBeanPHP 4

RedBeanPHP[11] je objektově relační manažer. Open-source projekt publikovaný pod BSD licencí vytvořen Gabor de Mooijim v roce 2009. Existují moduly pro frameworky jako je Lavarel nebo Zend.

Tato knihovna nepotřebuje jakékoli předchozí nastavení. Přesně se adaptuje na předanou databázi – podporuje MySQL, MariaDB, SQLite, PostgreSQL a CUBRID. Tímto se také odlišuje od ostatních ORM, které potřebují k chodu nastavit alespoň modely. Pokud celá tabulka nebo jen jeden sloupec v tabulce není vytvořen, knihovna se o jeho vytvoření postará za běhu v některých případech i o změnu typu. Pokud je vývoj dokončen, je možné schéma „zmrazit“ a ukončit tyto změny za běhu.

Ukázka funkcí a vlastností knihovny:

[10] Viz `http://dwoo.org/benchmark/loop/`.
[11] Viz `http://redbeanphp.com/`.

- Vytvoření bean objektu. Je možné vytvořit i více objektů, viz druhý řádek:

```
$book     = R::dispense( 'book' );
$twoBooks = R::dispense( 'book', 2 );
```

- Přidání vlastností:

```
$book->title  = 'Learn to Program';
$book->rating = 10;
```

- Uložení bean objektu:

```
$id = R::store( $book );
```

- Načtení bean objektu z databáze s daným `$id`:

```
$book = R::load( 'book', $id );
```

- Smazání jedné nebo více bean objektů:

```
R::trash( $book );
R::trashAll( $books );
```

- Obnovení bean objektu z databáze:

```
$bean = $bean->fresh();
```

- Hledání bean objektu (parametry jsou automaticky escapovány):

```
$book  = R::find( 'book', ' rating > ? ', [ 4 ]);         // více knih
$book  = R::findOne( 'book', ' title = ? ', [ 'SQL' ] ); // jedna kniha
$books = R::findAll( 'book' );                           // všechny knihy
```

- Podpora přímého `SQL`:

```
R::exec( ... )       // spuštění dotazu
R::getAll( ... )     // vrací multidimenzionální pole
R::getRow( ... )     // vrací jeden řádek
R::getCol( ... )     // vrací jeden sloupec
R::getCell( ... )    // vrací jednu buňku
```

- Funkce pro konverzi z pole na bean objekt.
- Pokročilé debug funkce – logování na obrazový výstup nebo do souboru.
- Podpora relací mezi tabulkami `1:n`, `n:1` a `n:m`.
- Implementovaná agregační funkce `COUNT` – `R::count( ... )`.
- Modely – můžeme si definovat pro každou tabulku model objekt, se kterým se bude pracovat, pokud budeme dotazovat data z této tabulky. Modely v tomto projektu nepoužívám. Jedná se o zajímavou funkci, která by mohla být implementována jako rozšíření.

Výše uvádím základní funkce. Knihovna jich obsahuje mnohem více, viz[11]. Velká výhoda spočívá v tom, že nemusíme psát dlouhé `INSERT` a `UPDATE` dotazy, ale stačí nám místo toho dvě funkce, které se o tyto operace postarají.

4.3.2 XCRUD

XCRUD knihovna[12] poskytuje generátor pro CRUD[13] operace nad tabulkami pod PHP a MySQL. Přednosti knihovny:

- Velmi jednoduchá syntaxe:

```
1 echo Xcrud::get_instance()->table('produkty');
```

  Vygeneruje editační tabulku pro produkty v databázi.

- Možnost vygenerovat více tabulek na jedné stránce.
- Rozhraní plně podporuje AJAX. Všechny operace jsou asynchronně zpracovány.
- Validační funkce.
- Možnost navázání uživatelských callback funkcí – vhodné například při ukládání hesla do databáze v hashované podobě.
- Mnoho možností úpravy vzhledu a zobrazení jednotlivých dat.
- Podpora uploadu souborů.
- Podpora spojování tabulek, subselectů z jiných tabulek a relací mezi tabulkami.
- Kompatibilní s frameworkem Bootstrap.
- Každý výstup je možné exportovat do CSV nebo vytisknout.

Knihovnu používáme hlavně v administrační části, což nám velmi ulehčí práci s tabulkami. Její implementace využívá návrhový vzor jedináček, tím pádem může být kdekoli ve skriptu inicializovaná a neprojeví se to na výkonu. Můžeme pracovat s tabulkami z různých databází na jedné stránce.

Podporuje všechny datové typy MySQL databáze (např. typy ENUM nebo SET automaticky převádí na seznam položek). Nicméně pokud nevyhovují, podle potřeby je můžeme měnit. Díky integrovaným komponentám podporuje například výběr data, času nebo dokonce souřadnic jako bodu z Google map.

Pro bezpečnost používá session, kde při každém požadavku vygeneruje validační klíč, který je ihned po použití zakázán.

4.3.3 Raphaël knihovna

Jednou z klíčových částí této práce jsou atomické úlohy. Ve webové aplikaci byl kladen důraz na to, aby tyto procesy byly patřičně prezentovány. Je vhodné zobrazit jejich návaznosti na sebe a aktuální stav, ve kterém se právě nachází.

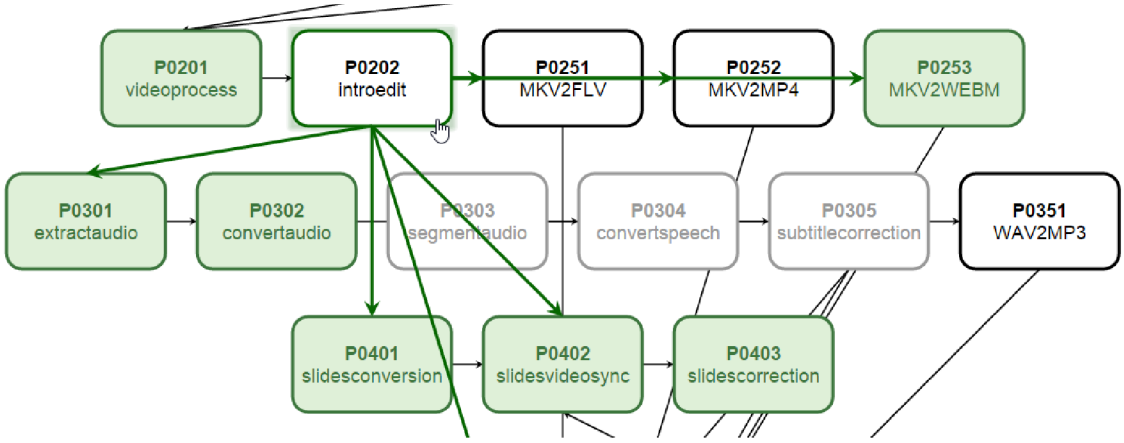
Zobrazení úloh a vazeb mezi nimi se dá řešit několika způsoby. Jako první řešení mě napadlo úlohy zobrazovat jako `<div>` tagy, které budou podle umístění pozicované. Jelikož se jedná o `DOM` objekty, není problém navázat uživatelskou interakci (hover evekt, klik, ...) na tyto bloky. Problém nastává při zobrazení vazeb – čar, v lepším případě šipek, mezi procesy. Zde jsem nenašel adekvátní řešení.

[12] Viz `http://xcrud.com/`.

[13] Create-Read-Update-Delete – Vytvoření, čtení, úprava a mazání dat nad vybranou tabulkou.

Druhou možností bylo použít prvek `<canvas>` z HTML5 značkovacího jazyka. Na plátno bych vykreslil všechny procesy a vztahy mezi nimi. Zde problém nastal při zpracování uživatelské interakce. Jelikož prvky na plátně nejsou samostatné `DOM` objekty, bylo by zapotřebí kontrolovat pozici myši pokud se nachází / nenachází nad objekty. Toto řešení nebylo ideální.

Poté jsem nalezl knihovnu Raphaël[14]. Jedná se o JavaScriptovou knihovnu, která usnadňuje práci s vektorovou grafikou na webu. K tomu používá SVG W3C doporučení a VML[15] To znamená, že každý grafický objekt je také `DOM` objektem, takže je jednoduché na něj navázat JavaScriptové funkce. Má širokou podporu prohlížečů – Firefox 3.0+, Safari 3.0+, Chrome 5.0+, Opera 9.5+ a Internet Explorer 6.0+.



Obrázek 4.4: Ukázka zobrazení procesů jako SVG objektů pomocí knihovny Raphaël.

Framework vychází ze specifikace SVG a podporuje většinu geometrických objektů – kruh, elipsa, křivka, čára, obdelník a jiné. Dále podporuje transformace, načítání obrázků, gradienty nebo například stínování.

Na obrázku č. 4.4 můžeme vidět procesy a vazby mezi nimi včetně aktivního stavu u procesu `P0202 Intro edit`. Více k stromu procesů v podkapitole 4.4.

4.3.4 Bootstrap

Webové rozhraní splňuje určité standardy responzivního designu – k tomu nám napomáhá framework Bootstrap[16]. Nejpopulárnější HTML, CSS a JS framework pro vývoj responzivních projektů. Vyvinuto firmou Twitter – sada nástrojů pro práci s HTML, CSS a JavaScriptem. Dále jsem čerpal informace z [9].

Na obrázcích č. 4.5 a 4.6 můžeme vidět, jak vypadá systém na mobilních zařízeních nebo na tabletech.

Obrázek 4.5: Ukázka zobrazení na mobilních zařízeních.

[14] Viz `http://raphaeljs.com/`.
[15] Vector Markup Language – formát založený na XML pro dvou-dimenzionální data.
[16] Viz `http://getbootstrap.com/`.

Obrázek 4.6: Ukázka zobrazení na středně velkých zařízeních.

4.3.5 PFBC

Framework Slim neobsahuje žádnou podporu pro generování nebo validaci formulářů. Vytváření formulářů můžeme provádět bez jakékoli knihovny, ale toto řešení je velmi nepraktické a není dobré pro změny a udržování systému v aktuálním stavu.

Formulářových knihoven není mnoho, pro tento projekt jsem vybral knihovnu PFBC[17]. Vyvinuto jako open-source v roce 2009. Od verze 3 je plně kompatibilní s knihovnou Bootstrap. Podporuje 32 formulářových prvků[18]. Formulář můžeme vygenerovat podle čtyř šablon – vedle sebe, pod sebou, v řádku nebo jako vyhledávač. Knihovna obsahuje široké nastavení validací – regulární výraz, email, číslo, url, datum, znaky a číslo nebo uživatelskou validaci. Obsahuje také podporu pro AJAX.

4.3.6 Upload

Upload[19] je jednoduchá knihovna pro upload včetně nahrávaných souborů. Soubory můžeme omezit velikostí, mime typem nebo koncovkou souboru.

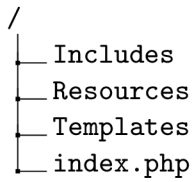
[17] PHP Form Builder Class – viz `http://code.google.com/p/php-form-builder-class/`.

[18] Button, Captcha, Checkbox, Checksort, CKEditor, Color, Country, Date, DateTimeLocal, DateTime, Email, File, Hidden, HTML, jQueryUIDate, Month, Number, Password, Phone, Radio, Range, Search, Select, Sort, State, Textarea, Textbox, Time, TinyMCE, Url, Week, YesNo.

[19] Viz `https://github.com/codeguy/Upload`.

4.4 Návrh a implementace

V úvodu této kapitoly popisuji základní implementaci všech komponent a způsob jejich volání. Dále popisuji jednotlivé části systému a ke konci kapitoly se věnuji databázovému modelu.



Obrázek 4.7: Adresářová struktura projektu.

Struktura projektu je zobrazena na obrázku č. 4.7. Popis složek:

- `Includes` – Obsahuje včechny knihovny.
- `Resources` – Obsahuje `CSS` styly, `JS` soubory a obrázky.
- `Templates` – Obsahuje šablony pro stránky.

Při spuštění aplikace se jako první volá `loader.php`, který registruje autoloadery (viz podkapitola Autoloader níže), načte všechny knihovny (XCRUD, RedBeanPHP, Slim, Dwoo a PFBC) a nastaví databázové spojení. Základem aplikace je třída `IQApp`, implementovaná podle vzoru jedináček. Tato třída je reflexí aplikace. Hlavními atributy této třídy jsou:

- **user** – Instance třídy `User`.
- **app** – Instance frameworku Slim.
- **dwoo** – Instance frameworku Dwoo.
- **i18n** – Instance překladové třídy `i18n`.

Dále se tato třída stará o ukládání posledně navštívených stránek do `SESSION`. Což je použito například při přesměrování na předchozí stránky.

Při vytvoření instance z této třídy se postupně instancují objekty v atributech. Dalším krokem je registrace zpracování cest – „routing“. Pro tento účel je volána třída `Router`, která zpracovává požadavky od klienta a volá adekvátní odpovědi. Router je blíže popsán v podkapitole níže. Poté, co se registruje `Router`, je pouze nastaven typ aplikace (produkce nebo vývoj) a dále proběhne samotné spuštění.

Router

Framework Slim poskytuje dobrou podporu pro směrování stránek. V aplikaci toto zajišťuje obalová třída `Router`, která rozšiřuje abstraktní třídu `ARouter`. V této třídě se definuje, jak budou požadavky zpracovávány. Framework Slim nám poskytuje tyto funkce (proměnná `$app` je instance třídy `Slim`):

- Zpracování `GET`, `POST`, `PUT`, `DELETE`, `OPTIONS` a `HEAD` požadavků.

```
1 $app->get('/Video', function () { ... });
```

V aplikaci standardně využívám požadavky typu `GET` pro zobrazení informací, požadavky `POST` pro ukládání a požadavky `DELETE` pro smazání dat. Ostatní typy jsou prozatím nevyužity.

- V aplikaci můžeme definovat více typů požadavků pro jedno zpracování.

```
1 $app->map('/Video', function() { ... })->via('GET', 'POST');
```

- Široká podpora parametrů.

```
1 $app->get('/Video/:id', function ($id) { ... });
```

Volitelné parametry.

```
1 $app->get('/Video(/:id(/:name))', function($id = 0, $name = '') { ... });
```

Libovolné parametry. Například můžeme zadat adresu `/Video/1/2/ahoj`, kde `1, 2, 'ahoj'` budou v poli `$data`.

```
1 $app->get('/Video/:data+', function ($data) { ... });
```

- Podmínky pro parametry. Jakýkoli parametr můžeme omezit regulárním výrazem, což zvyšuje bezpečnost celé aplikace.

```
1 $app->get('/Slide/:fileId/:size', function () { ... })
2     ->conditions(array('size' => 'SMALL|MEDIUM|LARGE'));
```

Můžeme také definovat podmínky v rámci celé aplikace.

```
1 \Slim\Route::setDefaultConditions(array(
2     'firstName' => '[a-zA-Z]{3,}'
3 ));
```

- Sdružování cest do skupin – zpřehledňuje kód. Podporuje vnořování skupin. Cesta položky ve skupině se skládá z cesty skupiny a ze své části.

```
1 $app->group('/Video/:id', function () use ($app) {
2   $app->get('', function ($id) { ... });
3   $app->post('', function ($id) { ... });
4   $app->get('/File/:fid', function ($id, $fid) { ... });
5 });
```

Další užitečnou komponentou frameworku Slim je „middleware“. Jedná se o uživatelsky definované funkce, které můžeme volat před zpracováním požadavku. V našem případě to jsou funkce, které se starají o kontrolu práv při přistupování na stránky. Tyto funkce jsou definovány v třídě `ARouter`, která je rozšířena třídou `Router`. Ukázka volání:

```
1 $app->get('/Administration', \IQ\ARouter::$checkPermissions, function () {
2     (new \IQ\Classes\Administration($this->a))->show();
3 });
```

Pro jeden požadavek můžeme volat více různých „Middlewarů“.

Autoloader

Při objektově orientovaném přístupu se vytváří jeden PHP soubor pro každou třídu. Pokud vyvíjíme aplikaci, tak musíme všechny tyto třídy načíst pomocí příkazů `require` nebo `include`, což neusnadňuje práci nehledě na to, že můžeme snadno zapomenout na načtení nějakého souboru. V PHP od verze 5 toto není nutné, pokud se snažíme použít nějakou třídu, která nebyla prozatím definovaná, zavolají se autoloadery, které jsou registrované a požadovaný soubor se načte.

4.4.1 Instance stránek

Systém obsahuje dva druhy odpovědi na požadavek:

1. **„Jednoduchá odpověď"** – V tomto případě se volá některá ze statických funkcí. Zde se nevolají žádné další funkce, což je potřeba například při zpracování AJAX požadavků. Nejčastěji se jedná o zpracování formuláře, kde se na konci zpracování provede přesměrování:

```
1 $app->post('/Login', function () {
2     \IQ\Classes\Login::processForm();
3 });
```

2. **„Odpověď stránkou"** – Pro kažkou stránku je nutné vytvořit třídu, která bude rozšiřovat třídu `APage`. A to z toho důvodu, že při instanciaci takové třídy se volá konstruktor rodiče, kde se provádí tyto úlohy:

   - Kontrola práv – můžeme přepsat.
   - Instanciace Dwoo objektu a načtení šablony. Pokud šablona není zadaná a například vytváříme objekt z třídy `Lectures`, tak se systém pokusí načíst šablonu `Lectures.tpl`.
   - Předání globálních dat šabloně – informace o uživateli a práva.
   - Instanciace navigačního tabu – třída `Breadcrumb`.
   - Generování menu.
   - Volání abstraktní funkce `content`, která doplní data do šablony.

   Příklad vytvoření login stránky:

```
1 $app->get('/Login', function () {
2     (new \IQ\Classes\Login($this->app))->show();
3 });
```

Po instanciaci třídy voláme, pro zobrazení, funkci `show`, kde šablonovací systém Dwoo zkompiluje šablonu (pokud tak již nebylo učiněno). Posledním krokem je spuštění kompilované šablony s daty.

4.4.2 Administrace – Framework

Tuto sekci bude obsluhovat pouze administrátor nebo super administrátor. Jednou z hlavních částí této práce je propojení Frameworku s webovou aplikací. Je důležité mít na paměti,

že systém může fungovat zcela bez zásahu webového rozhraní. Data, která byla nutná přenést do systému a uložit je do databáze jsou dvojího typu – tokeny a procesy. Proto je v systému zavedena jednostranná synchronizace.

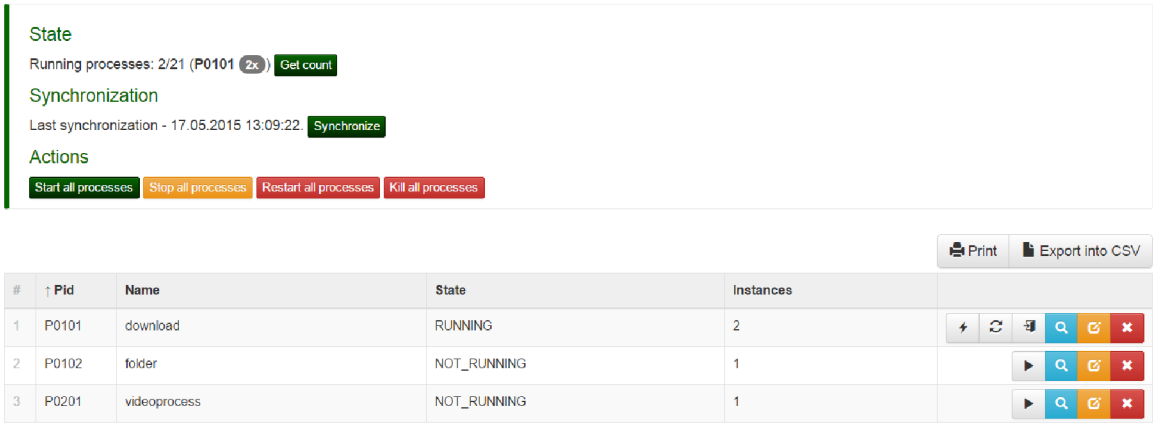
Synchronizace tokenů bude zavedena do `CRON` plánovače úloh jako volání určité adresy s bezpečnostním klíčem. Tato úloha spadá i do následující kapitoly, protože spolu s tokeny se provádí vkládání vázaných videí do databáze.

Synchronizace tokenů je rozdělena na tyto podúlohy:

1. Načtení tokenů z disku – podsložek složky `/DATA`.
2. Načtení tokenů z databáze.
3. Seřazení tokenů z disku podle toho, zda mají „rodičovskou“ referenci na jiný token, viz kapitola níže. Pokud by totiž seřazené nebyly, mohlo by se stát, že první vložíme do databáze video, které má referenci na dosud neuložené video.
4. Synchronizace jednotlivých tokenů z disku (viz kapitola níže).
5. Vkládání videí za základě výsledků synchronizace tokenů (viz kapitola níže).

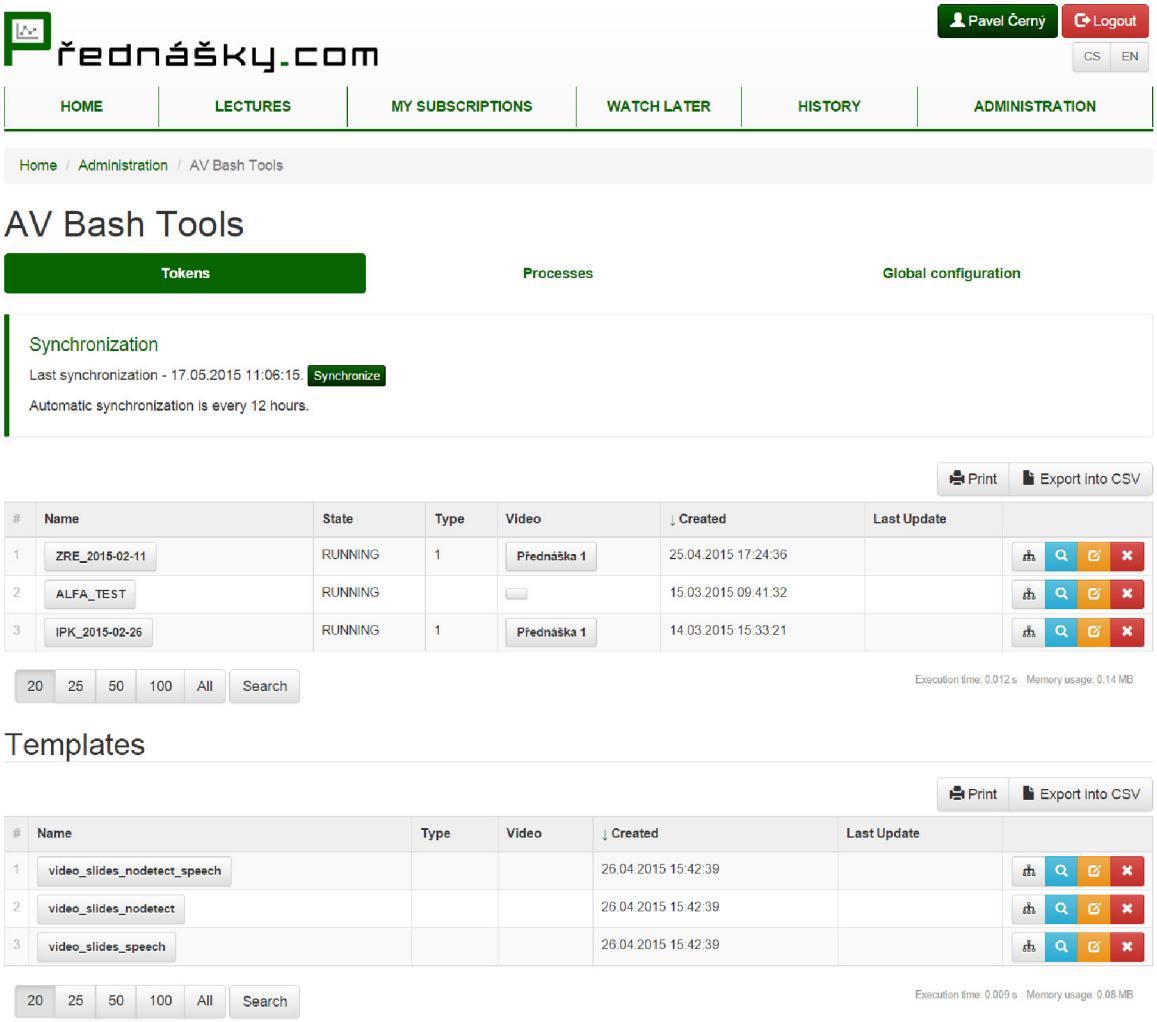
Po synchronizaci máme v systému potřebná data, se kterými se bude dále pracovat. Administrace je dělena na dvě záložky – tokeny a procesy.

V první záložce nalezneme vynucenou synchronizaci, seznam tokenů a seznam šablon nastavení pro tokeny, viz obrázek č. 4.9. Tyto šablony se vytváří pomocí Bash Frameworku parametrem `-f` viz kapitola č. 2.3.



Obrázek 4.8: Administrace procesů – synchronizace, akce a seznam. Aktuálně spuštěný proces P0101 ve dvou instancích.

V druhé záložce se nachází správa procesů. Jako první vidíme počet aktuálně spuštěných procesů (parametr `-p` Bash Frameworku), který můžeme kdykoli obnovit. Dále tato stránka obsahuje synchronizaci, která je popsána níže. Administrátor může také spustit „globální“ akce nad procesy, jako je spuštění, zastavení, kill nebo restart všech procesů analogicky odpovídajícím parametrům Bash Frameworku. Jako poslední, můžeme vidět tabulku s procesy, viz obrázek č. 4.8. S jednotlivými procesy můžeme dělat analogické úkony jako s akcemi nad všemi procesy – ve všech případech se volá Bash Framework s danými parametry. Při editaci procesu můžeme také nastavit počet spouštěných instancí.



Obrázek 4.9: Administrace – synchronizace, seznam tokenů a seznam šablon.

„Rodičovská“ reference

V průběhu vývoje vznikl požadavek, že jedna výsledná přednáška může mít dva a více zdrojů záběru – například na přednášejícího a na plátno. Pakliže se nahrávané přednášky ukládají do dané složky automaticky, je nutné tuto vazbu nějakým způsobem zaznamenat. Videa ve složce zpracovává proces `P0102 Folder` (viz kapitola č. 3.1) a ten přebírá doplňující informace z přiložených `conf` souborů a posílá je do globální `DATA` složky od daného tokenu. V těchto doplňujících informacích může být právě položka `parent`, která znázorňuje tuto „rodičovskou“ referenci.

Ve výsledku se první při synchronizaci vloží „rodičovský“ token. Tomu se automaticky vytvoří video záznam v databázi. Následně se vkládá token „potomek“, u kterého je reference na rodiče, díky čemuž je vložen do rodičovského videa.

Synchronizace tokenu

Při synchronizaci probíhá kontrola, zda se jedná o šablonu. Pokud ano, nespouští se synchronizace videa k tomuto tokenu. Dále se porovnají tokeny v databázi s tokeny na disku

a podle potřeby se provede vložení. Druhým krokem je synchronizace videa.

Synchronizace videa

Zde se nejedná o synchronizaci v pravém slova smyslu, ale spíše o napojení tokenu na video. Jako první se provádí načtení informací z globální `DATA` složky tokenu, kde jsou prozatím informace o předmětu, délce trvání, školním roce, verzi, rodičovském tokenu a náhledech – některé tyto informace generuje proces `P0201 video process` (viz kapitola č. 3.2). Podle toho, zda jsou tyto informace načteny se ukládají k videu do databáze.

Synchronizace procesů

Synchronizace procesů probíhá obdobně jako synchronizace tokenů – procesy se načtou z adresáře `/PROCESSES`, porovnají se s procesy v databázi a v případě potřeby se provede vložení.

4.4.3 Administrace – videa

Tuto sekci vidí uživatelé s právy učitele. Zde budou mít učitelé jediný úkol a to dohlížet a kontrolovat své přednášky. Mohou nicméně tuto práci delegovat na jiné uživatele formou nastavení práv, viz kapitola 4.4.5. Stránka se dělí na pět sekcí:

1. **Videa v přípravě** – Zde budou videa za situace, kdy je objeví nahrávané video v kontrolované složce a dojde na řadu ke zpracování k procesu `P0201 video process`, který video pozastaví. Dále musí být učitel přiřazený k danému předmětu, aby video mohl zpracovat.

   Pokud je vše v pořádku, učitel vidí základní informace o videu a může pokračovat kliknutím na „Konfigurovat pro zpracování“. Tato funkce je popsána v kapitole níže.

2. **Videa v procesu** – V této sekci může učitel vidět videa, která se právě zpracovávají. Jeden z požadavků na systém byl, aby mohl učitel vidět, kdy zhruba budou videa ve zpracování hotova. K tomu slouží odkaz „časový odhad“, viz kapitola níže.

3. **Videa k publikování** – Zde jsou videa, která byla již zpracovaná a jsou připravena k publikování. V této sekci můžeme každé video zkontrolovat a pokud bude vše v pořádku, video schválíme.

4. **Publikovaná videa** – Tabulka s publikovanými videi. Video můžeme „odpublikovat“ přesunutím do čekajících videí.

5. **Čekající videa** – Videa, která čekají na uživatelskou interakci.

Konfigurace pro zpracování

Na této stránce vyplňuje učitel veškeré údaje k výsledné přednášce. Formulář lze vyplnit pouze z části a dokončit později. Nachází se zde tyto sekce:

- **Základní informace** – Název přednášky, předmět, školní rok, fakulta, obor a škola.
- **Autoři a mluvčí** – Mluvčí mohou být stejní jako autoři. Při vyplňování je možnost automaticky doplnit z databáze uživatelů systému.

- **Video tokeny** – Seznam video tokenu, které jsou přiřazené k danému videu. Zde je dobré vyplnit čas nahrání videa a typ video tokenu. Může se jednat o záběr na plátno (v takovém případě systém vybere toto video k synchronizaci prezentace s videem, viz proces `P0402 slide video sync`) nebo záběr na mluvčího.

- **Přidání prezentace** – Je vhodné, aby učitel také přidal k videu prezentaci. Může se stát, že přednáška překrývá více prezentací. I s tímto problémem systém počítá, jednoduše stačí nahrát více souborů a systém provede jejich spojení.

  S přidáním prezentace souvisí také výběr oblasti plátna, kde uživatel interaktivně vyznačí tuto oblast. Pokud tak neučiní systém sám odhadne, kde se plátno nachází, viz proces `P0501 plane detection`. Výběr plátna je postaven na prvku `canvas` značkovacího jazyka HTML5 spolu s JavaScriptem.

- **Popis videa** – Abstrakt, klíčová slova, prerekvizity a postrekvizity z již publikovaných videí.

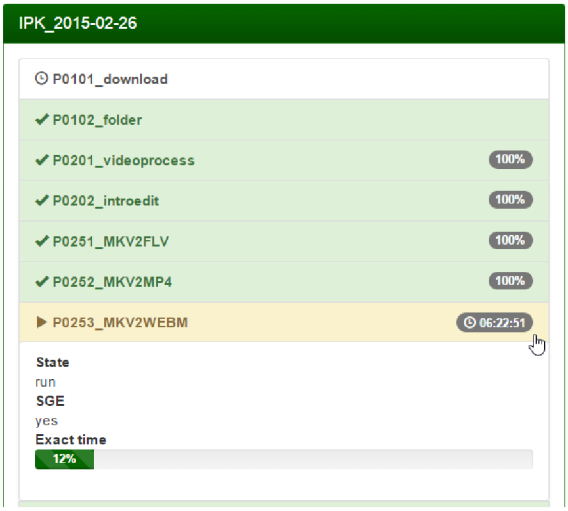
- **Pokročilé nastavení** – Uživatel má možnost nahrát časování prezentace, titulky, audio stopu, intro snímek nebo náhledy videa.

Během zpracování formuláře se vybírají šablony konfigurací pro video tokeny podle toho, jaké soubory uživatel nahrál a jak nastavil konfiguraci. Například se nikdy nestane to, že by se pro dva tokeny pro stejné video synchronizovala jedna prezentace. Kopírování šablon do tokenů se provádí pomocí Bash Frameworku (parametr `-f`). Nahrávané soubory se kopírují nebo přesouvají do příslušných procesu pro následné zpracování.

Časový odhad

Na této stránce má učitel možnost vidět jednotlivé video tokeny a jejich časový odhad doby zpracování. Systém provádí dva druhy časového odhadu.

Před spuštěním aktuálního procesu můžeme vidět odhad doby trvání, který je určen jako násobek koeficientu a délky videa specifikovaného v konfiguračním souboru u daného procesu. Po nasazení se musí tyto koeficienty přenastavit, aby přibližně odpovídaly realitě.



Obrázek 4.10: Odpočet u procesu P0253.

Jakmile je proces spuštěn a využívá ke zpracování program `avconv`, tak se začne automaticky generovat průběh tohoto zpracování do logu. Právě tento logovací soubor je klíčový k určení přesného odhadu dokončení úlohy. Při obnovení stránky se načte poslední řádek z tohoto logu, tento řetězec se rozparsuje a určí se přesná zbývající doba zpracování, jedno procentní časový interval a hotová část v procentech. Z těchto dat je vytvářen výsledný odpočet.

Na stránce se nedozvíme informaci o celkovém zpracování přes všechny tokeny a procesy, což může být námět pro možné rozšíření. Výpočet musí zohlednit paralelní běh procesů a počet tokenů k videu. Jako další rozšíření může být kontrolování SGE fronty úloh a informování uživatele o její délce a eventuálně i ovlivnění výpočtu doby trvání zpracování.

4.4.4 Administrace – ostatní

V této kapitole popisuji ostatní možnosti v administraci. Položka „Seznamy“ je důležitá pro administrátory, protože obsahuje přiřazování předmětu k učiteli. Takovému učiteli se pak zobrazují videa pro nastavení. Je možné také přiřadit předmět studentovi, ale je nutné myslet na to, že mu musíme také přiřadit práva k přístupu na stránky.

Další tabulky, které může administrátor v seznamech spravovat jsou předměty, studijní obory, fakulty a školy. Nad těmito tabulkami lze provádět všechny standardní operace – vkládání, úprava, smazání, prohlížení a vyhledávání. U předmětů se také nastavuje viditelnost videí, které pod předmět spadají.

V administraci dále nalezneme položku „Uživatelé“. Do této sekce může přistupovat pouze super administrátor, který může měnit veškeré údaje o uživatelích, včetně nastavování hesel.

4.4.5 Administrace – práva

V administraci můžeme nastavovat práva pro stránky nebo pro jednotlivé uživatele. Práva jsou v systému řešena na několika úrovních. Základní kontrolu zabezpečuje *middleware* u třídy `ARouter` – `checkPermissions`, který kontroluje přístup následujícím způsobem:

1. Kontrola, zda stránka odpovídá skupině práv, ve které se nachází uživatel. Pokud selže pokračujeme dalším krokem.

2. Kontrola, zda se stránka nachází v právech u konkrétního uživatele. Super administrátor může specifikovat buďto obecný tvar stánky (např. `/User/:id`), který bude odpovídat všem stránkám s libovolným `id` nebo konkrétní stránku s daným `id` – `/User/10`. V databázi se jedná o tabulky `Rights` a `Rights_group`.

Další *middleware* – `checkVideoPermissions` se stará o kontrolu práv, zda může uživatel vidět dané video. To je závislé na tom, jaká je nastavena viditelnost práv u předmětu, ke kterému je video zapsané. Tyto práva se budou synchronizovat s IS FIT. Implementace synchronizace je plánována k nasazení systému. Aktuálně se musí viditelnost nastavovat ručně.

Dále můžeme práva řešit na úrovni jednotlivých stránek přepsáním metody `permissions` třídy `Page`. Toto je využito například při editaci informací o uživateli, kdy stránku mohou vidět všichni uživatele, ale pouze se svým `id`. Kdežto super administrátor může vidět a editovat všechny stránky.

Pro šablony bylo nutné doimplementovat tag, který bude, podle práv, zobrazovat nebo skrývat svůj obsah. Framework Slim má pro tento účel dobrou podporu. Tag `permission` je použit například v šabloně administrace.

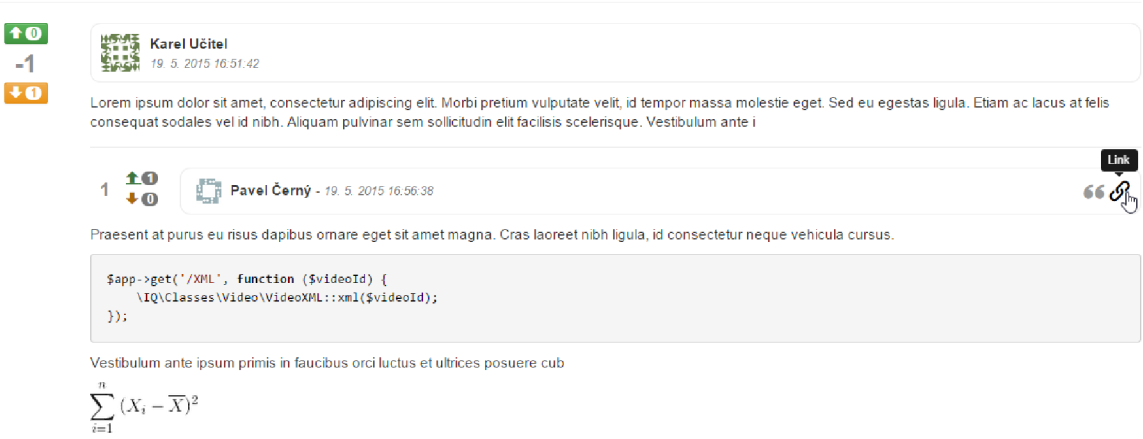
4.4.6 Přehrávání

Zpočátku byl projekt koncipován tak, že bude vytvořen Bash Framework a ten se napojí na stávající webovou prezentaci. V průběhu času, kdy se vytvářelo rozhraní administrace jsem si uvědomil, že nemá smysl se vázat na hotové řešení přehrávání přednášek od stávajícího systému. Tyto dva systémy by bylo značně komplikované napojovat na sebe.

Jako první jsem přemýšlel, jakou funkcionalitu by měla aplikace uživatelům poskytovat. Zde jsem se inspiroval od největšího internetového serveru pro sdílení videí – YouTube[20]. Systém tedy poskytuje přehrávání videí, `sledování` předmětů pro nové videa, shlédnutí videa později a historii.

Samotné přehrávání je postaveno na HTML5 přehrávači *SLplayer*. Tento přehrávač vytvořil pan Jan Peša [7], který je založený na starším webovém prohlížeči přednášek vytvořený panem Josefem Žižkou [8]. SLplayer byl striktně vázán na server SuperLectures.com, z kterého stahoval veškeré informace. Pan Žižka prováděl úpravy HTML5 přehrávače, aby jsem tento přehrávač mohl použít i bez napojení na server SuperLectures.com.

Přehrávač SLplayer má na vstupu XML soubor, ve kterém jsou všechny důležité informace – popis, autory, mluvčí, datum záznamu, datum přidání, adresu miniatury, seznam video souborů v různých formátech (procesy `P025x`), audio soubory (proces `P0351`), seznam obrázků z prezentace (proces `P0402`) a statistiky. Veškerá tato data jsou načítána buďto z procesů nebo z databáze. Statistiky jsou vytvářeny přehrávačem, který odesílá každých 5 sekund požadavek na server o tom, kde se právě uživatel ve videu nachází, o jaké video se jedná a popřípadě na jaké tlačítko uživatel klikl. Tyto informace rozparsuji a ukládám do databáze.



Obrázek 4.11: Ukázka komentáře – vložený kód a rovnice.

Pod přehrávačem se nachází diskuse, které jsem přikláněl nemalý důraz a to z toho důvodu, že se zde může probírat problematika studia a použití by mělo být maximálně komfortní. Základem je textový editor CKEditor[21], který je rozšířen o dva pluginy. Prním

[20] Viz `https://www.youtube.com/`.
[21] Viz `http://ckeditor.com/`

je CodeSnipped, který se stará o vkládání a formátování programovacích jazyků. Druhým pluginem je Equation Editor, který podporuje vkládání a formátování matematických rovnic v LaTeX formátu.

Diskuse je dvouúrovňová s hodnocením příspěvků. Pouze přihlášený uživatel se může připojit do diskuse. Učitelé navíc mohou mazat příspěvky. Na komentáře můžeme odkazovat nebo je citovat v dalším příspěvku, viz obrázek č. 4.11. Při implementaci jsem se snažil využívat co nejvíce AJAX.

4.4.7 Databáze

Databáze reflektuje persistentní data v systému. Hlavní tabulky jsou `video`, `token` a `user`. Na tabulku `video` jsou navázány klíčová slova, následující a předchozí video, tokeny, autoři a mluvčí, předmět, studijní obor a ostatní tabulky. Podrobné schéma databáze je zobrazeno v příloze B.

Jako databázový systém jsem zvolil MySQL[22] s úložištěm dat InnoDB, které má podporu cizích klíčů nebo transakcí. Právě cizí klíče hojně v databázi používám. Je nutné dobře ošetřit relace na rodičovské klíče pro kaskádové mazání či editaci těchto klíčů.

V databázi je na pohled jedna zvláštnost a to, že video má vazbu jak na předmět, tak na studijní obor a dále předmět má také vazbu na studijní obor. Toto zapojení se může zdát matoucí. Avšak vazba video – studijní obor je zde pouze pro naplnění spojovací tabulky předmět – studijní obor. Situace by se dala vyřešit jiným způsobem. U videa při editaci by se zobrazily všechny studijní obory, které se v dané chvíli vztahují k předmětu a učitel by tento seznam zkontroloval. Tyto tabulky jsou nutné pro vyhledávání videí podle oboru, tato funkce v systému není a je to námět na možné rozšíření. Jakmile bude tuto problematiku někdo zkoumat, může si vybrat řešení, které se mu bude více vyhovovat – systém je připraven na obě varianty.

[22] MySQL je databázový systém, vytvořený švédskou firmou MySQL AB, nyní vlastněný společností Sun Microsystems, dceřinou společností Oracle Corporation. Jeho hlavními autory jsou Michael „Monty“ Widenius a David Axmark. Viz `http://www.mysql.com/`.

Kapitola 5

Případy užití

Diagram případů užití (Use Case Diagram) se používá k popisu chování systému z hlediska uživatele a zachycuje, které typy uživatelů se systémem pracují a jaké činnosti v rámci systému vykonávají [2].

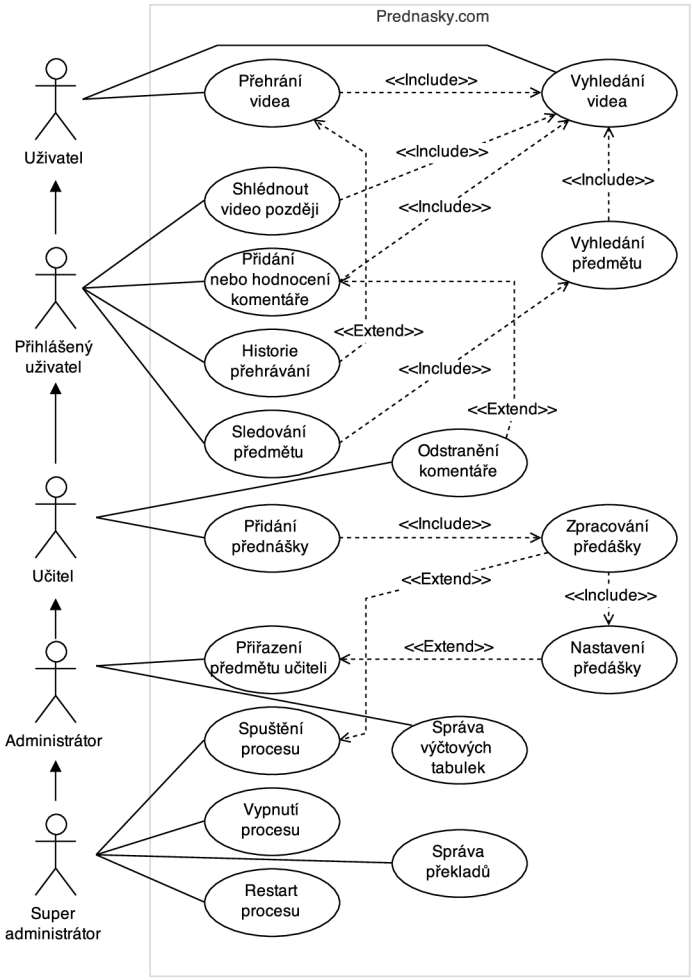
Dle [1] spočívá modelování případů užití v následujících krocích:

- Nalezení hranic systému.
- Vyhledání aktérů.
- Nalezení případů užití
- Specifikace případů užití.
- Určení alternativních scénářů.
- Opakování předchozích bodů, dokud nedojde k ustálení případů užití, aktérů a hranic systému.

Náš systém reprezentuje diagram případů užití přiložen na obrázku č. 5.1.

Můžeme vidět čtyři aktéry systému a jeho pomyslné hranice. Mezi všemi aktory je vztah generalizace, tudíž nadřazený aktor může provádět veškeré úkony jako předek. Diagram zobrazuje nejdůležitější případy užití.

V následujících podkapitolách si popíšeme tři specifikace případů užití, jejichž formát je převzatý z [2].



Obrázek 5.1: Diagram případů užití.

5.1 Přehrání videa

Jedná se o jeden ze základních úkonů, tudíž musí být co nejvíce přímočarý a jednoduchý.

Název případu užití	**Přehrání videa**		
Identifikace případu užití	UC1		
Cíl případu užití	Nalezení a přehrání videa.		
Primární aktér(ři)	Uživatel, přihlášený uživatel.		
Pomocný aktér(ři)	—		
Vstupní podmínky	—		
Výstupní podmínky	Video je přehráno uživatelem.		
Základní scénář	**Krok**	**Role**	**Akce**
	1	Aktér	Vyhledá video k přehrání.
	2	Systém	Kontroluje, zda je uživatel přihlášen
	3	Systém	Kontroluje, zda má uživatel právo pro přehrávání videa.
	4	Aktér	Spustí video.
	5	Systém	Začne se zasíláním videa po částech.
	6	Systém	Zaznamenává statistiky shlédnutí.
	7	Aktér	Shlédne videa.
Alternativní scénář 2a: Uživatel nemá právo pro přehrání videa	**Krok**	**Role**	**Akce**
	2a1	Systém	Pokud uživatel nemá právo pro shlédnutí, bude mu zobrazena hláška o nedostatečných právech.
	2a2	Systém	Nabídne uživateli formulář pro přihlášení.

Tabulka 5.1: Specifikace případu užití – přehrání videa.

5.2 Přidání přednášky

Tento případ užití popisuje přidání přednášky do systému a její zpracování.

<table>
<tr><td>Název případu užití</td><td colspan="3">Přidání přednášky</td></tr>
<tr><td>Identifikace případu užití</td><td colspan="3">UC2</td></tr>
<tr><td>Cíl případu užití</td><td colspan="3">Přidání přednášky do systému.</td></tr>
<tr><td>Primární aktér(ři)</td><td colspan="3">Učitel.</td></tr>
<tr><td>Pomocný aktér(ři)</td><td colspan="3">Administrátor.</td></tr>
<tr><td>Vstupní podmínky</td><td colspan="3">Učitel musí být přihlášen do systému. Administrátor zapsal předmět od videa k učiteli. Procesy pro zpracování videí jsou spuštěné.</td></tr>
<tr><td>Výstupní podmínky</td><td colspan="3">Video je přidáno do systému.</td></tr>
<tr><td rowspan="8">Základní scénář</td><td>Krok</td><td>Role</td><td>Akce</td></tr>
<tr><td>1</td><td>Systém</td><td>Synchronizuje přednášky a zobrazí video v administraci od učitele.</td></tr>
<tr><td>2</td><td>Aktér</td><td>Nastaví video – vyplní všechny potřebné údaje</td></tr>
<tr><td>3</td><td>Aktér</td><td>Uloží data a pošle video ke zpracování.</td></tr>
<tr><td>4</td><td>Systém</td><td>Podle nastavení zvolí správnou šablonu pro dané video a spustí zpracování.</td></tr>
<tr><td>5</td><td>Systém</td><td>Oznámí aktérovi, že je zpracování dokončeno zobrazením v administraci.</td></tr>
<tr><td>6</td><td>Aktér</td><td>Zkontroluje a schválí video.</td></tr>
<tr><td>7</td><td>Systém</td><td>Publikuje video.</td></tr>
<tr><td rowspan="3">Alternativní scénář 2a: Učitel nezašle video ke zpracování</td><td>Krok</td><td>Role</td><td>Akce</td></tr>
<tr><td>2a1</td><td>Systém</td><td>Pokud učitel nepotvrdí zpracování videa, systém uloží všechna zadaná data.</td></tr>
<tr><td>2a2</td><td>Systém</td><td>Nabídne učiteli seznam rozpracovaných videí k pozdějšímu zpracování.</td></tr>
</table>

Tabulka 5.2: Specifikace případu užití – přidání přednášky.

5.3 Sledování předmětu

Tato specifikace případu užití popisuje sledování předmětu pro nové přednášky.

<table>
<tr><td>Název případu užití</td><td colspan="3">Sledování předmětu.</td></tr>
<tr><td>Identifikace případu užití</td><td colspan="3">UC3</td></tr>
<tr><td>Cíl případu užití</td><td colspan="3">Přidání předmětu do sledování aktora</td></tr>
<tr><td>Primární aktér(ři)</td><td colspan="3">Přihlášený uživatel.</td></tr>
<tr><td>Pomocný aktér(ři)</td><td colspan="3">—</td></tr>
<tr><td>Vstupní podmínky</td><td colspan="3">Uživatel musí být přihlášen do systému.</td></tr>
<tr><td>Výstupní podmínky</td><td colspan="3">Uživatel má zapsán předmět ve sledování.</td></tr>
<tr><td rowspan="7">Základní scénář</td><td>Krok</td><td>Role</td><td>Akce</td></tr>
<tr><td>1</td><td>Aktér</td><td>Vyhledá video v systému.</td></tr>
<tr><td>2</td><td>Systém</td><td>Kontroluje, zda má uživatel právo pro přehrávání videa.</td></tr>
<tr><td>3</td><td>Aktér</td><td>Klikne na předmět u videa.</td></tr>
<tr><td>4</td><td>Aktér</td><td>Klikne na sledování předmětu</td></tr>
<tr><td>5</td><td>Systém</td><td>Uloží uživatele k předmětu od videa.</td></tr>
<tr><td>6</td><td>Systém</td><td>Pokud bude přidáno video do systému, uživatel jej uvidí v seznamu sledovaných videí.</td></tr>
<tr><td rowspan="3">Alternativní scénář 2a: Uživatel nemá právo pro sledování předmětu.</td><td>Krok</td><td>Role</td><td>Akce</td></tr>
<tr><td>2a1</td><td>Systém</td><td>Pokud uživatel nemá právo pro sledování bude mu zobrazena hláška o nedostatečných právech.</td></tr>
<tr><td>2a2</td><td>Systém</td><td>Přesměruje uživatele na předchozí stránku.</td></tr>
</table>

Tabulka 5.3: Specifikace případu užití – sledování předmětu.

Kapitola 6

Bezpečnost

Každá část mé práce je elementárně zabezpečena.

1. **Bash Framework** – Každý proces kontroluje vstup (cesta k souboru), zda neobsahuje nepovolené znaky – uvozovky, vnořené uvozovky, escapované zpětné lomítka. Nicméně uživatel přímo Framework neobsluhuje a činí tak skrze webové rozhraní. V tomto případě uživatel může změnit pouze adresu stahovaného videa, které je ovšem validováno před uložením do souboru na platnou webovou adresu.

2. **Webové rozhraní**:

   - **SQL Injection** – Ve všech mých dotazech používám ORM[1] framework RedBeanPHP 4[2], který u všech svých dotazů escapuje automaticky parametry. Nemůže tak k SQL Injection dojít.
   - **Přístup ke stránkám** – Zabezpečení je řešeno na dvou úrovních. Zaprvé na úrovni definice stránek v souboru `router.php`, kde je toto řešeno jako middleware frameworku Slim při spouštění stránek – práva jsou uloženy v databázi jak pro skupiny uživatelů, tak pro jednotlivce. Druhou možností je kontrola jednotlivých stránek zvlášť – pomocí přepsání metody `permissions`, která vrací `true` nebo `false` podle uvážení. Například je toto využito při úpravě informací o uživateli – metoda je volána automaticky ještě před zpracováním stránky. Viz kapitola č. 4.4.5.
   - **Cross site scripting (XSS)** – Všechny víceřádkové vstupní pole kontroluji na výskyt HTML tagů, které odstraňuji funkcí `strip_tags`, kde mám povoleny pouze některé tagy. Na straně klienta je vstup ošetřen knihovnou pro editaci – CKEditor[3], která nedovoluje zadání nespecifikovaných HTML tagů.
   - **Soubory** – Pro upload souborů používám knihovnu Upload[4]. Každý soubor má definovanou validaci na velikost a mime typ.

[1] Objektově relační mapování.
[2] Viz `http://redbeanphp.com/`.
[3] CKEditor je WYSIWYG textový editor distribuovaný pod licencemi GPL, LGPL a MPL. Viz `http://ckeditor.com/`.
[4] Viz `https://github.com/codeguy/Upload`.

Kapitola 7

Závěr

V rámci této práce se podařilo vytvořit celistvý systém pro usnadnění zpracování videí nejen z přednášek, ale i různých konferencí. Využívaný bude hlavně studenty, proto byl kladen také důraz na předání informací jak už videem tak i samotnou diskusí, která je jeho součástí.

Důležitá část práce je Bash Framework, který tvoří základ pro zpracování přednášek. Při jeho popisu se nachází princip tokenového zpracování, jak systém nakonfigurovat, podrobný popis funkcí, obsluhy procesů a celková struktura Frameworku. Tato část je zcela dokončena, Bash Framework je lehce přenositelný a lze jej snadno obsluhovat. Poskytuje mnoho funkcí, které pokrývají nejběžnější úkony se samotnými procesy. Určuje rozhraní mezi jednotlivými tokeny a webovou aplikací. Následně popisuji základní sestavení procesů a jejich propojení. Vybrány jsou tři zajímavé procesy, které jsou rozebrány detailně. Ostatní procesy, které zpracovávají jednotlivé části videa se nachází v příloze.

Nejen k přehrávání videí je vytvořené webové rozhraní. Slouží také jako administrační nástroj pro správu Frameworku, jednotlivých tokenů a videí. V úvodu příslušné kapitoly popisuji výběr frameworku a šablonovacího systému a dále komponenty, které jsem v aplikaci použil. V závěru popisuji podrobně implementační detaily, synchronizaci webové aplikace s Bash Frameworkem a diskutuji možná rozšíření aplikace. Několik jednoduchých ukázek, jak s vytvořeným systémem pracovat a informace o možných uživatelích systému, demonstrují přiložené případy užití.

Jako každá internetová aplikace, i vytvářený systém je elementárně zabezpečen. Jak na úrovni Bash Frameworku, tak na úrovni webové aplikace. V závěru práce analyzuji různé bezpečnostní hrozby.

Za největší přínosy považuji zpracování projektu jako celku, budoucí nasazení a skutečné využití tohoto projektu. Vytvořený systém je připraven ke spuštění a k reálnému využití, nasazení na servery FIT a zpřístupnění studentům je naplánováno na léto 2015.

Projekt nabízí mnoho prostoru k dalšímu rozvoji. V části Bash Frameworku jde například o funkce pro vrácení tokenu do různých front v případě chyby nebo lepší správa globálních dat tokenu. Na úrovni webového rozhraní jsou možnosti rozsáhlejší také díky tomu, že se ze začátku uvažovalo pouze o propojení se stávajícím systémem přednášek. Například se může jednat o přesnější výpočet doby zpracování, přidání webové korekce k časování prezentace nebo editor titulků. Některé z těchto funkcí se budou implementovat v nejbližší době, na projektu je plánováno nadále pokračovat.

Literatura

[1] Arlow, J.; Neustadt, I.: *UML 2 a unifikovaný proces vývoje aplikací: Objektově orientovaná analýza a návrh prakticky.* Brno: Computer Press, 2008, ISBN 9788025115039.

[2] Buchalcevova, A.; Pavlíčková, J.; Pavlíček, L.: *Základy softwarového inženýrství – materiály ke cvičení.* Praha: Vysoká škola ekonomická, 2007, ISBN 9878024512709.

[3] Chaffer, J.; Swedberg, K.: *Learning jQuery.* Packt Publishing, Čtvrté vydání, 2013, ISBN 178216314X, 9781782163145.

[4] Douglas, D.; Peucker, T.: Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *The Canadian Cartographer*, ročník 10, č. 2, 1973: s. 112 – 122, ISSN 57-6770-U.

[5] Follansbee, J.: *Hands-on guide to streaming media: An introduction to delivering on-demand media.* Amsterdam: Focal press, 2006, ISBN 9780240808635.

[6] Hayder, H.: *Object-Oriented Programming with PHP5: Learn to Leverage PHP5's OOP Features to Write Manageable Applications with Ease.* Packt Publishing, 2007, ISBN 1847192564, 9781847192561.

[7] Jan, P.: *Prednasky.com – Systém jako modul.* Diplomová práce, FIT VUT v Brně, 2012.

[8] Žižka Josef: *Webový prohlížeč přednášek.* Diplomová práce, FIT VUT v Brně, 2009.

[9] Marcotte, E.: *Responsive web design.* New York, New York, USA: A Book Apart, 2011, ISBN 9780984442577.

[10] Newham, C.; Vossen, J.; Albing, C.; aj.: *Bash Cookbook: Solutions and Examples for Bash Users.* O'Reilly Media, Inc., 2007, ISBN 0596526784.

[11] Potencier, F.: Templating Engines in PHP. `http://fabien.potencier.org/article/34/templating-engines-in-php`, 2009-10-07 [cit. 2015-05-11].

[12] Ramer, U.: An iterative procedure for the polygonal approximation of plane curves. *Computer Graphics and Image Processing*, ročník 1, č. 3, 1972: s. 244 – 256, ISSN 0146-664X.

[13] Safronov, M.; Winesett, J.: *Web Application Development with Yii 2 and PHP.* Birmingham, UK: Packt Publishing Ltd, 2014, ISBN 1783981881, 9781783981885.

[14] Skokanová, J.: Videokonference a streaming multimédií [online]. `https://wis.fit.vutbr.cz/FIT/st/course-files-st.php/course/ /ISA-IT/lectures/archiv-2011/isa8-multimedia.pdf`, 2011-11-10 [cit. 2015-05-10].

[15] Taylor, D.: *Wicked Cool Shell Scripts: 101 Scripts for Linux, Mac OS X, and Unix Systems*. San Francisco, CA, USA: No Starch Press, 2004, ISBN 1593270127, 9781593270124.

Příloha A

Obsah CD

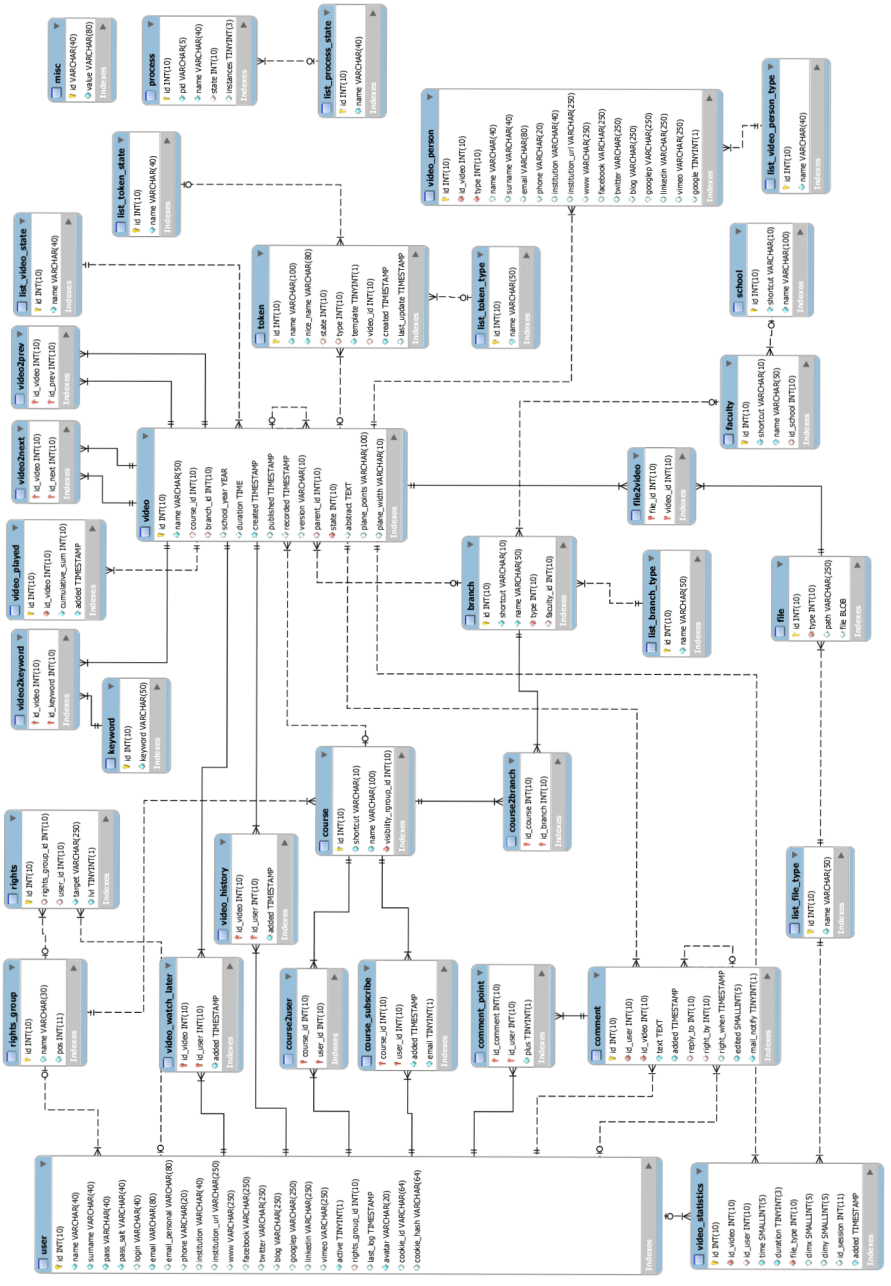
CD obsahuje tyto položky:

- **Framework** – Bash Framework s vzorovými šablonami ve složce `DATA`.
- **WebApp** – Webová aplikace.
- **DIP-xcerny49.pdf** – Text práce.
- **DIP-source** – Zdrojové soubory textu práce v jazyce LaTeX.

Ve webové aplikaci je nutné nastavit konfigurační soubor `config.php` ve složce `Includes` před spuštěním.

Příloha B

Databázové schéma



Obrázek B.1: Databázové schéma.

Příloha C

Procesy

C.1 P0101 Download

Úlohou tohoto procesu je stáhnout požadované video ze zadané adresy a následně překonvertovat do požadovaného formátu. Základní rozhraní tohoto procesu je následující:

- `VSTUP` — URL adresa požadovaného videa, uložená v souboru `download.url` ve složce `DATA/token_n/GLOBAL/DATA/INPUT` (globální data pro aktuální token).
- `VÝSTUP` — Konvertované video v souboru `video.avi` ve složce pro standardní výstup `DATA/token_n/process_n/DATA/OUTPUT`

Sekce kontroly

Program bude pokračovat jedině za předpokladu, že bude soubor `download.url` ve složce `DATA/token_n/GLOBAL/DATA/INPUT` nenulový.

Sekce spuštění

Prvním krokem procesu je rozhodnutí, zda je soubor umístěn na internetovém úložišti nebo na serveru `youtube.com`.

Pokud se jedná o první možnost, program zkontroluje, zda URL adresa obsahuje název souboru a v opačném případě skončí proces s chybou. Dále probíhá kontrola, zda soubor v adrese obsahuje koncovku. Pokud ne, je přidělena koncovka `.avi`. Následně bude soubor stažen programem `wget`[1] do složky pro dočasné soubory – `DATA/token_n/process_n/DATA/ /OUTPUT` pod jménem `video.avi`.

Pokud chceme stáhnout video ze serveru `youtube.com`, tak použijeme skript `youtube-dl`[2], který načte zdroj videa ze stránky a uloží do složky pro dočasné soubory.

[1] GNU Wget je počítačový program, který slouží jako jednoduchý a výkonný stahovač souborů. Implementuje přenos souborů přes protokoly HTTP, HTTPS a FTP. Jeho jméno vzniklo ze složeniny World Wide Web a get. Viz `http://www.gnu.org/software/wget/`.

[2] Youtube-dl je program příkazové řádky ke stahování videí z YouTube.com a z pár dalších stránek. K běhu je zapotřebí Python verze 2.6, 2.7 nebo 3.3+ a je platformě nezávislý. Měl by být funkční v Unixových systémech, ve Windows nebo v Mac OS X. Je vydán jako volné dílo – můžete jej modifikovat, distribuovat nebo jakkoli použít. Projekt je vyvíjen na github.com. Viz `http://rg3.github.io/youtube-dl/`.

Sekce přesměrování

Proces je přednastaven k zaslání výstupu (respektive souboru, který obsahuje cestu výstupu) do složky `DATA` procesu `P0201 video process`.

C.2 P0102 Folder

Tento proces je mírně odlišný od ostatních procesů, protože jako jediný vytváří – generuje tokeny a navíc, je tento proces opakován v rámci jednoho tokenu – viz sekce spuštění. Úlohou tohoto procesu je vytvoření tokenů ze souborů ve vstupní složce a přeposlání těchto tokenů k dalšímu zpracování. Základní rozhraní tohoto procesu je následující:

- `VSTUP` — Adresa požadované složky, uložená v souboru `file_to_process.txt` ve složce `DATA/token_n/GLOBAL/DATA/INPUT` (globální data pro aktuální token).
- `VÝSTUP` — Standardní výstup tento proces nemá, ale generované tokeny za výstup můžeme považovat.

Sekce kontroly

Program bude pokračovat jedině za předpokladu, že bude soubor `file_to_process.txt` ve složce `DATA/token_n/GLOBAL/DATA/INPUT` nenulový.

Sekce spuštění

Tento proces prochází soubory ve vstupní složce a generuje z nich tokeny pro další zpracování. Jak již bylo avizováno, toto se děje opakovaně, což je umožněno nastavením proměnné `TOKEN_RUN_FOREVER` na hodnotu `YES`. Nastavení přikáže frameworku, aby při dokončení zpracování opět token zařadil do čekačí fronty a je zrušená sekce přesměrování. Pokud ale kontrolujeme opětovně stejnou složku, tak si musíme zaznamenat, kde jsme v předchozím zpracování skončili. Pro tento případ slouží soubor `timestamp.txt`, který se nachází ve složce `DATA/token_n/GLOBAL/DATA/TMP` (globální dočasná data pro aktuální token).

Prvním krokem je kontrola, zda soubor `timestamp.txt` existuje. Následně je buď vytvořen nebo je vybrán obsah tohoto souboru do proměnné, jako časová známka posledního souboru, který byl zpracován.

Následně vyberu všechny soubory ze složky na vstupu a seřadím si je podle časové známky editace. V průchodu je porovnávám s mou uloženou časovou známkou a pokud se objeví novější soubor, zpracuji jej.

S takovýmto souborem je zacházeno jako s novým tokenem. Název tokenu je složeninou z názvu souboru a časové známky. Nejprve je vytvořena datová struktura tokenu. Následně je do této struktury zapsán proces `P0102_folder` a označen jako dokončený (vložením `state.done` do složky `STATE` daného procesu).

Dalším krokem je kontrola, zda u tokenu není konfigurační soubor (musí mít stejný název s příponou `.conf`). Pokud tento soubor existuje, je načten a jeho data jsou zapsány do globální složky nového tokenu. Aktuálně načítám tyto položky – třídu, rok, verzi a „rodičovský token“ (popsaný v podkapitole č. 4.4.2). Tyto informace se dále používají při synchronizaci s webovým rozhraním.

Posledním krokem je suplování přesměrování tohoto vytvořeného procesu – umístění vstupního souboru je uloženo do souboru a zasláno na vstup do složky `DATA` procesu `P0201 video process`.

Sekce přesměrování

Jak již bylo napsáno, proces je volán opakovaně s aktuálním tokenem, tudíž je sekce přesměrování přeskočena. Nicméně nastavení přesměrování je použito v sekci spuštění, kde jsou podle toho přesměrovány nově vytvořené tokeny.

C.3 P0201 Video process

Úkolem tohoto krátkého procesu je pouze konverze vstupního videa do jednotného kontejneru – v našem případě `mkv`[3]. Tento proces má ještě další úlohu a to zastavení tokenu pro webovou aplikaci. V takovém případě proces získá z videa potřebné informace a čeká na uživatelův pokyn pro pokračování. Základní rozhraní tohoto procesu je následující:

- `VSTUP` — Adresa požadovaného videa, uložená v souboru `file_to_process.txt` ve složce `DATA/token_n/process_n/DATA/INPUT`.
- `VÝSTUP` — Soubor `video.mkv` ve výstupní složce `DATA/token_n/process_n/DATA/OUTPUT`

Sekce kontroly

Program bude pokračovat jedině za předpokladu, že bude soubor `file_to_process.txt` ve složce `DATA/token_n/process_n/DATA/INPUT` nenulový.

Sekce spuštění

Standardně tento proces načte vstupní video libovolného formátu a pomocí programu `avconv` jej převede do jednotného kontejneru souboru `video.mkv`. Program má parametr `-codec:v copy` a `-codec:v copy`, což znamená, že je kodek video a audio stopy je kopírován ze vstupu na výstup. Dále má parametr `-f mkv`, který určuje typ multimediálního kontejneru.

Pokud je ale u tokenu nastavena proměnná `TOKEN_RUN_FOREVER`, tak se předchozí zpracování přeskočí. Dále je volán program `avprobe`, který zjistí následující informace z videa, potřebné pro další zpracování – počet snímků za sekundu a délku videa. Dále se exportují programem `avconv` tři náhledy z videa. Tyto data se ukládají do globální složky tokenu. Jakmile je token synchronizován objeví se učiteli ve videích ke zpracování. Po té, co učitel toto video nastaví, aplikuje se některá z šablon nastavení na tento token a proměnnou `TOKEN_RUN_FOREVER` odstraní, čímž může proces pokračovat.

Sekce přesměrování

Proces je přednastaven k zaslání výstupu (respektive souboru, který obsahuje cestu výstupu) do složky `DATA` procesu `P0202 intro edit`.

[3]Matroska pod zkratkou `mkv` je moderní otevřený svobodný multimediální kontejner (podobný AVI), který umožňuje pojmout většinu moderních video a audio formátů. Dokáže též pojmout několik různých audio stop včetně prostorového zvuku. Viz `http://en.wikipedia.org/wiki/Matroska`.

C.4 P0202 Intro edit

Tento proces bude přidávat intro před video. V základním nastavení se bude jednat o logo fakulty následované varováním proti vytváření kopií. Základní rozhraní tohoto procesu je následující:

- `VSTUP` — Adresa požadovaného videa, uložená v souboru `converted_video.txt` ve složce `DATA/token_n/process_n/DATA/INPUT`.
- `VÝSTUP` — Soubor `video.avi` ve výstupní složce `DATA/token_n/process_n/DATA/OUTPUT`

Sekce kontroly

Program bude pokračovat jedině za předpokladu, že bude soubor `converted_video.txt` ve složce `DATA/token_n/process_n/DATA/INPUT` nenulový.

Sekce spuštění

Základem tohoto procesu je program `melt`, který podle předaného XML konfiguračního souboru přidá do videa libovolné intro snímky nebo varování. Tento proces je otestován, ale v aktuálním stavu se přeskakuje. Při nasazení bude finálně nastaven s nově dodanými konfiguračními soubory.

Sekce přesměrování

Proces je přednastaven k zaslání výstupu (respektive souboru, který obsahuje cestu výstupu) do složky `DATA` procesům `P0301 extract audio`, `P0401 slide conversion` (pouze pro inicializaci procesu), `P0402 slide video sync`, `P0501 plane detection` a `P0701 video editing`.

C.5 P0301 Extract audio

Tento proces slouží pro extrahování audia a audio informací z videa. Základní rozhraní tohoto procesu je následující:

- `VSTUP` — Umístění konvertovaného videa, uloženého v souboru `intro_video.txt` ve složce `DATA/token_n/process_n/DATA/INPUT`.
- `VÝSTUP` — Zvuková stopa videa v souboru `audio.wav`, délka zvukové stopy videa v souboru `audio.length` a počet kanálů v souboru `audio.channelnum`. Vše se standardní složce pro výstup – `DATA/token_n/process_n/DATA/OUTPUT`.

Sekce kontroly

Program bude pokračovat jedině za předpokladu, že bude soubor `intro_video.txt` ve složce `DATA/token_n/process_n/DATA/INPUT` nenulový.

Sekce spuštění

Proces extrahuje audio pomocí programu `avconv` z balíku softwaru `libav`[4]. Jako bezztrátový kodek je zvolen `PCM signed 16-bit little-endian`. `PCM` značí pulzně kódovou modulaci a převádí zvukový signál v časové doméně na signál digitální (posloupnost amplitud) s danou vzorkovací frekvencí. `Signed 16-bit` značí znaménkový 16-ti bitový rozsah $< -32768, 32767 >$ jednoho vzorku. `Little-endian` značí architekturu daného procesoru. Formát souboru je nastaven na `wav`.

Dále potřebujeme zjistit délku audio stopy a počet kanálů. K tomu nám poslouží program `soxi`[5] (přepínač `-d` pro délku audia a přepínač `-c` pro počet kanálů).

Sekce přesměrování

Proces je přednastaven k zaslání výstupu (respektive souboru, který obsahuje cestu výstupu) do složky `DATA` procesu `P0302 convert audio`.

C.6 P0302 Convert audio

Proces má za úkol převést audio vstup do vhodného formátu pro převod řeči na text a pro video, pokud je potřeba. Základní rozhraní tohoto procesu je následující:

- `VSTUP` — Umístění extrahované audio složky uloženéhé v souboru `wav_audio.txt`, dále délky audio stopy v souboru `length_audio.txt` a počet kanálů v souboru `channelnum_audio.txt`. Vše ve složce `DATA/token_n/process_n/DATA/INPUT`.
- `VÝSTUP` — Soubor `converted_audio_asr.wav` obsahující zvukovou stopu pro automatický převod řeči a soubor `converted_audio_video.wav`, který obsahuje audio stopu převedenou pro video. Dále je pouze přeposlán soubor `length_audio.txt`, který je využit v následujících procesech. Vše se standardní složce pro výstup – `DATA/token_n/process_n/DATA/OUTPUT`.

Sekce kontroly

Program bude pokračovat jedině za předpokladu, že budou soubory `wav_audio.txt`, `length_audio.txt` a `channelnum_audio.txt` ve složce `DATA/token_n/process_n/DATA/ /INPUT` nenulové.

Sekce spuštění

Konfigurační soubor je automaticky načten při startu procesu. Z tohoto souboru jsou pak načteny informace, kolik výstupních souborů bude vytvořeno a s jakými parametry. Základní nastavení je takové, že se vytvářejí dva výstupní soubory, kde první je určen pro převod řeči a má následující parametry:

- Výstupní kanál – pouze levý.

[4]Libav je svobodný software, který poskytuje knihovny a programy pro práci s multimédií. Nejvíce používané části jsou libavcodec – audio/video knihovna, libavformat – knihovna pro práci s audio/video kontejnery a avconv – program pro konverzi multimediálních souborů. Zdrojové kódy jsou publikovány pod licencí LGPL. Viz `https://libav.org/`.

[5]Sound eXchange Information – program pro zobrazení audio metadat.

- Vzorkovací frekvence – 8000.
- Normalizace – ano.

Druhý soubor je určen pro video následné video zpracování a má tyto parametry:

- Výstupní kanál – stereo.
- Vzorkovací frekvence – 16000.
- Normalizace – ne.

Nejprve se porovnává počet vstupních a výstupních kanálů. Pokud počet nesouhlasí je vstup zpracován programem `sox`[6], kde je buď vybrán pravý nebo levý kanál, popřípadě mix kanálů do jednoho, vše dle parametrů.

Dále probíhá normalizace, taktéž pomocí programu `sox`, kde nejprve zjistíme informaci o úrovni hlasitosti a poté, je stejným programem parametrem `-v` hlasitost normalizována.

Posledním krokem je převzorkování, které taktéž provádí program `sox` parametrem `rate -h` (`h` – vysoká kvalita).

Sekce přesměrování

Proces je přednastaven k zaslání výstupu (respektive souboru, který obsahuje cestu výstupu) do složky `DATA` procesu `P0303 segment audio`.

C.7 P0303 Segment audio

Proces `segment audio` má za úkol analyzovat vstupní audio stopu a určit takové segmenty, kde se mluví a kde nikoliv. Základní rozhraní tohoto procesu je následující:

- `VSTUP` — Umístění konvertované audio stopy uloženéhé v souboru `converted_audio_asr.wav` a délky audio stopy v souboru `length_audio.txt`. Vše ve složce `DATA/token_n/process_n/DATA/INPUT`.
- `VÝSTUP` — Soubor `audio.segm` obsahující úseky řečových a neřečových segmentů. Uloženo ve standardní složce pro výstup – `DATA/token_n/process_n/DATA/OUTPUT`.

Sekce kontroly

Program bude pokračovat jedině za předpokladu, že budou soubory `converted_audio_asr.txt` a `length_audio.txt` ve složce `DATA/token_n/process_n/DATA/ /INPUT` nenulové.

Sekce spuštění

Hlavní částí procesu je program `vtlnestim`, který má za vstup zvukovou stopu a výstupem je soubor s časovými úseky `SPEECH_SEGMENT` a `NONSPEECH_SEGMENT`. Viz obrázek C.1. K tomuto programu nemám zdrojové kódy a proto ho nemůžu popsat do detailů.

[6] SoX – Sound eXchange je platformě nezávislý program příkazové řádky, který umožňuje konverzi mnoha audio formátů. Také umí přidat do audio stopy různé efekty a umí tyto soubory přehrávat. Viz `http://sox.sourceforge.net/`.

```
103400000 156600000 <SPEECH_SEGMENT> -10.722419
156600000 160300000 <NONSPEECH_SEGMENT> -75.105324
160300000 210700000 <SPEECH_SEGMENT> -13.050686
210700000 211000000 <NONSPEECH_SEGMENT> -45.629509
```

Obrázek C.1: Ukázka výstupu.

Posledním krokem je filtrace segmentů programem `filter_segments.sh`, který pouze přidá na začátek a konec souboru neřečový segment, a to z toho důvodu, ať výstup pokrývá celou časovou osu.

Sekce přesměrování

Proces je přednastaven k zaslání výstupu (respektive souboru, který obsahuje cestu výstupu) do složky `DATA` procesu `P0303 convert speech`.

C.8 P0304 Convert speech

Sekce kontroly

Program bude pokračovat jedině za předpokladu, že bude soubor `segmented_audio.txt` ve složce `DATA/token_n/process_n/DATA/INPUT` nenulový.

Sekce spuštění

Tento proces je plně připraven na doplnění programu pro převod řeči na text.

Sekce přesměrování

Proces je přednastaven k zaslání výstupu (respektive souboru, který obsahuje cestu výstupu) do složky `DATA` procesu `P0305 subtitle correction`.

C.9 P0401 Slides conversion

Úlohou tohoto procesu je zkonvertování prezentace ve formátu PDF na skupinu obrázků ve formátu JPG. Tato konverze je potřebná pro další zpracování, kde jsou snímky porovnávány s video záznamem. Základní rozhraní tohoto procesu je následující:

- `VSTUP` — Umístění prezentace, uložené v souboru `file_to_process.txt` ve složce `DATA/token_n/process_n/DATA/INPUT`.
- `VÝSTUP` — Jednotlivé strany prezentace zkonvertované v rozlišeních 1024, 640 a 240 delší strany v samostatných složkách. Dále pak umístění jednotlivých souborů ve třech XML souborech pro každé rozlišení. Vše se standardní složce pro výstup – `DATA/ /token_n/process_n/DATA/OUTPUT`.

Sekce kontroly

Program bude pokračovat jedině za předpokladu, že bude soubor `file_to_process.txt` ve složce `DATA/token_n/process_n/DATA/INPUT` nenulový.

Sekce spuštění

Podle konfiguračního souboru vybereme způsob převodu prezentace. Na výběr máme ze dvou možností:

- Programem `acroread`, kde jsou nastaveny přepínače `-toPostScript` – provede převod vstupu na postscript soubor, `-optimizeForSpeed` – zapne optimalizace pro rychlejší převod, `-choosePaperByPDFPageSize` – zvolí velikost výstupu podle velikosti PDF stránky. Výstupem tohoto programu je soubor `slides.ps`, který dále zpracovává program `gs` – Ghostscript a „tiskne“ jej do JPG souborů. Kvalita a rozlišení je nastaveno v konfiguračním souboru. Bohužel program `acroread` – Adobe Reader® ukončil podporu pro Linuxové systémy a poslední dostupná verze je 9.5.5.
- Programem `convert`[7], který přímo převede prezentaci na skupinu JPG souborů. Kvalita a rozlišení je opět nastaveno v konfiguračním souboru.

Dalším krokem je změna velikosti. V průchodu všemi soubory vytvoříme, programem `convert`, tři varianty rozlišení – 1024x768, 640x480 a 240x180 v samostatných složkách.

Posledním krokem je vytvoření XML souborů, které reflektují umístění jednotlivých obrázků. Tyto soubory jsou dále využívány v ostatních procesech.

Sekce přesměrování

Proces je přednastaven k zaslání výstupu (respektive souboru, který obsahuje cestu výstupu) do složky `DATA` procesu `P0402 slides video synchronisation`.

C.10 P0402 Slides video synchronisation

Proces má za úkol synchronizovat prezentaci s videem. Základní rozhraní tohoto procesu je následující:

- `VSTUP` — Umístění konvertovaného videa, uloženého v souboru `intro_video.txt`, konvertované snímky z prezentace v souboru `xml_1024.txt` a pozici plátna ve videu v souboru `xml_coordinates.txt`. Vše ve složce `DATA/token_n/process_n/DATA/INPUT`.
- `VÝSTUP` — Synchronizovaná prezentace s videem v souboru `xml_sync_slides.xml` ve složce `/token_n/process_n/DATA/OUTPUT`.

Sekce kontroly

Program bude pokračovat jedině za předpokladu, že budou soubory `intro_video.txt`, `xml_1024.txt` a `xml_coordinates.txt` ve složce `DATA/token_n/process_n/DATA/INPUT` nenulové.

[7] Convert je program příkazové řádky ze softwarového balíku ImageMagick®, který poskytuje převod mezi formáty, změnu velikosti, rozmazání, ořez, vyhlazení, převrácení, převzorkování a mnoho dalšího. Publikováno pod licencí Apache 2.0. Viz `http://www.imagemagick.org/script/command-line-tools.php`.

Sekce spuštění

Základem tohoto procesu je program `det_slides` s následujícími parametry – vstupní videosoubor, konfigurační soubor (`xml_coordinates.txt`) a xml soubor s lokací obrázků `xml_1024.txt`.

Program zkoumá oblast plátna (předaná v konfiguračním souboru) a pokud se ve videu tato oblast změní, tak jí uloží jako obrázek zvlášť do souboru. Dalším krokem je nalezení „specifických bodů“ – feature vectoru v těchto extrahovaných obrázcích a porovnání s feature vectory z obrázků z prezentace. Pokud se z části zhodují tak je tato oblast zaznamenaná do XML souborů a vyhodnocena jako synchronizovaná oblast. Výstupem je tedy XML soubor s cestami k obrázkům prezentace a dobou trvání ve videu.

Další krok je úprava tohoto výstupního XML programem `xml_convert.sh`, který pouze přeskládá xml entity v souboru pro webovou aplikaci, odstraní nesynchronizované snímky videa a nepotřebné data.

Sekce přesměrování

Proces je přednastaven k zaslání výstupu (respektive souboru, který obsahuje cestu výstupu) do složky `DATA` procesu `P0701 video editing`.

C.11 P0501 Plane detection

Úlohou tohoto procesu je detekce plátna ve videu. Základní rozhraní tohoto procesu je následující:

- `VSTUP` — Umístění konvertovaného videa, uloženého v souboru `intro_video.txt` ve složce `DATA/token_n/process_n/DATA/INPUT`.
- `VÝSTUP` — Souřadnice detekovaného plátna a ostatní informace (viz sekce spuštění) v xml souboru `coordinates.xml` ve složce `/token_n/process_n/DATA/OUTPUT`.

Sekce kontroly

Program bude pokračovat jedině za předpokladu, že bude soubor `intro_video.txt` ve složce `DATA/token_n/process_n/DATA/INPUT` nenulový.

Sekce spuštění

Veškeré zpracování obstarává program `det_plane` s následujícími parametry – vstupní videosoubor, výstupní xml soubor s koordináty plátna a parametr `-s`, který značí počet přeskočených snímků ze začátku videa.

Program `det_plane` používá knihovnu OpenCV. Nejprve načte vstupní video soubor. Jednotlivé n-té snímky videa jsou v cyklu zpracovány následovně:

1. Snímek je rozmazán Gaussovým filtrem a podvzorkován.
2. Převod snímku do odstínů šedi.
3. V dalším cyklu je provedeno postupné prahování v rozsahu $< 25, 255 >$ s krokem 25. Tyto hodnoty se dají nastavit ve vstupním konfiguračním souboru.

4. V takto prahovaných snímcích jsou vyhledány obrysy.

5. U každého obrysu je vypočtena plocha a pokud je v rozsahu 5-ti až 100% velikosti snímku, tak pokračujeme. Tyto hodnoty můžeme taktéž upravit v konfiguračním souboru.

6. Následně jsou body obrysu aproximovány Ramer-Douglas-Peucker algoritmem[8]. Nejprve s nízkou maximální vzdáleností (4) od eliminovaných bodů. Dále proběhne kontrola, zda je počet nově aproximovaných bodů v intervalu $< 4, 12 >$. Pokud ano, body jsou opět aproximovány, ale tentokrát s vysokou maximální vzdáleností (50) od eliminovaných bodů.

7. Výsledkem těchto aproximací by měl být obrys, který je definován 4-mi body – hledané plátno. Pokud tedy počet souhlasí, obrys je uložen.

Jakmile máme nalezené obrysy pro každý n-tý snímek jsou tyto obrysy rozděleny do skupin podobných obrysů (v ideálním případě pouze 1 skupina – 1 plátno) a nad těmito skupinami je proveden Gaussův filtr pro výsledný obrys. Dále je provedeno spojování přilehlých obrysů a odfiltrování překrytých regionů, které se překrývají více jak 70% (lze nastavit v konfiguračním souboru).

Posledními kroky jsou zpětné převzorkování nalezených regionů a uložení výsledků do souboru podle parametru, v našem případě `coordinates.xml`.

Sekce přesměrování

Proces je přednastaven k zaslání výstupu (respektive souboru, který obsahuje cestu výstupu) do složky `DATA` procesu `P0502 plane correction`.

C.12 P0601 Indexing

Úlohou tohoto procesu je indexace titulků videa. Základní rozhraní tohoto procesu je následující:

- `VSTUP` — Umístění titulků, uložených v souboru `subtitles_file.txt` ve složce `DATA/ /token_n/process_n/DATA/INPUT`.
- `VÝSTUP` — Titulky a soubor s indexací v `subtitles_file.txt` a `index.txt` ve složce `/token_n/process_n/DATA/OUTPUT`.

Sekce kontroly

Program bude pokračovat jedině za předpokladu, že bude soubor `subtitles_file.txt` ve složce `DATA/token_n/process_n/DATA/INPUT` nenulový.

Sekce spuštění

Tento proces je plně připraven na doplnění programu pro převod indexaci řeči.

[8] Ramer-Douglas-Peucker je algoritmus pro redukci počtu bodů v křivce, která je aproximována množinou bodů. Algoritmus je také znám pod názvy Douglas-Peucker, iterative end-point fit algoritmus nebo split-and-merge algoritmus. Základní forma algoritmu byla navržena v roce 1972 Urs Ramrem a v roce 1973 Davidem Douglasem a Thomasem Peuckerem [12, 4].

Sekce přesměrování

Proces je přednastaven k zaslání výstupu (respektive souboru, který obsahuje cestu výstupu) do složky `DATA` procesu `P0701 video editing`.

C.13 P0701 Video editing

Úlohou tohoto procesu je kontrola výsledného videa. Základní rozhraní tohoto procesu je následující:

- `VSTUP` — Umístění konvertovaného videa, uloženého v souboru `intro_video.txt` ve složce `DATA/token_n/process_n/DATA/INPUT`.
- `VÝSTUP` — Přeposlané video v souboru `intro_video.txt` ve složce `/token_n/process_n/ /DATA/OUTPUT`.

Sekce kontroly

Program bude pokračovat jedině za předpokladu, že bude soubor `intro_video.txt` ve složce `DATA/token_n/process_n/DATA/INPUT` nenulový.

Sekce spuštění

Úlohou tohoto procesu je zastavit zpracování a počkat na synchronizaci s webovou aplikací. Pokud se token dostane až zde, tak je zpracování hotovo. Uživatel bude o tomto stavu obeznámen v seznamu videí – video přejde do stavu „před publikací“. Úlohou uživatele je zkontrolovat výsledné video a potvrdit publikaci. V plánu před nasazením je také editace jednotlivých částí zpracování – titulky, prezentace a střih videa na menší zajímavé části.

Pokud uživatel schválí toto vide smaže se z nastavení tokenu `PROCESS_RUN_FOREVER` a token bude pokračovat k finálnímu procesu.

Sekce přesměrování

Proces je přednastaven k zaslání výstupu (respektive souboru, který obsahuje cestu výstupu) do složky `DATA` procesu `P0901 publish`.

C.14 P0901 Publish

Tento proces bude „uklízet“ veškerá data v celém stromu procesu. Základní rozhraní tohoto procesu je následující:

- `VSTUP` — prázdný.
- `VÝSTUP` — prázdný.

Sekce spuštění

Tento proces je připraven na doprogramování. Jakmile se bude systém nasazovat, je nutné specifikovat co je potřeba a ostatní soubory se buď smažou nebo zabalí a archivují.

C.15 P0305, P0403, P0502 – Subtitle, Slides a Plane correction

Tato sada procesů má pouze kontrolní účel. V základním nastavení přeposílá vstup na výstup. Pokud uživatel zadá, že chce zkontrolovat výsledek předchozího procesu, tak se do konfiguračních souborů jednoho z těchto procesů nastaví proměnná `TOKEN_RUN_FOREVER`, která zajistí, aby proces počkal na uživatelskou interakci.

C.16 P0251, P0252, P0253, P0351 – MKV2FLV, MKV2MP4, MKV2WEBM a WAV2MP3

Úlohou těchto procesu je konverze videa a audia. Základní rozhraní těchto procesu je následující:

- `VSTUP` — Umístění videa nebo audia, uloženého v souboru `intro_video.txt` nebo `converted_audio_video.txt` ve složce `DATA/token_n/process_n/DATA/INPUT`.
- `VÝSTUP` — Překonvertované audio nebo video ve složce `/token_n/process_n/DATA/ /OUTPUT`.

Sekce kontroly

Program bude pokračovat jedině za předpokladu, že bude soubor `intro_video.txt` nebo `converted_audio_video.txt` ve složce `DATA/token_n/process_n/DATA/INPUT` nenulové.

Sekce spuštění

Tyto procesy pouze volají program `avconv`, který provádí konverze. Finální parametry se odladí při nasazení projektu.

Sekce přesměrování

Proces je přednastaven k zaslání výstupu (respektive souboru, který obsahuje cestu výstupu) do složky `DATA` procesu `P0901 publish`.