VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA PODNIKATELSKÁ

FACULTY OF BUSINESS AND MANAGEMENT

ÚSTAV INFORMATIKY

INSTITUTE OF INFORMATICS

NÁVRH A TVORBA SOFTWAROVÉHO ŘEŠENÍ PRO TELEVIZNÍ VYSÍLÁNÍ

DESIGN AND CREATION OF SOFTWARE SOLUTION FOR TV BROADCASTING

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE — Bc. Jiří Psota

AUTHOR

VEDOUCÍ PRÁCE — Ing. Petr Dydowicz, Ph.D.

SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA PODNIKATELSKÁ

Zadání diplomové práce

Ústav:	Ústav informatiky
Student:	**Bc. Jiří Psota**
Vedoucí práce:	**Ing. Petr Dydowicz, Ph.D.**
Akademický rok:	2022/23
Studijní program:	Informační management

Garant studijního programu Vám v souladu se zákonem č. 111/1998 Sb., o vysokých školách ve znění pozdějších předpisů a se Studijním a zkušebním řádem VUT v Brně zadává diplomovou práci s názvem:

Návrh a tvorba softwarového řešení pro televizní vysílání

Charakteristika problematiky úkolu:

Úvod
Vymezení problému a cíle práce
Teoretická východiska práce
Analýza problému a současné situace
Vlastní návrh řešení, přínos práce
Závěr
Seznam použité literatury

Cíle, kterých má být dosaženo:

Diplomová práce se zabývá návrhem a vývojem softwarového řešení pro správu a řízení odbavování televizního vysílání. Cílem práce je zanalyzovat současný stav a stanovit požadavky na nové řešení. Na základě těchto požadavků bude realizováno odpovídající řešení, které se bude opírat o příslušná teoretická východiska. Výsledkem bude plnohodnotný funkční komplexní celek.

Základní literární prameny:

BASL, J. a R. BLAŽÍČEK. Podnikové informační systémy. Podnik v informační společnosti. Praha: Grada, 2008. 283 s. ISBN 978-80-247-2279-5.

MOLNÁR, Z. Automatizované informační systémy. Praha: Strojní fakulta ČVUT, 2000. 126 s. ISBN 80-01-02269-2.

MOLNÁR, Z. Efektivnost informačních systémů. Praha: Grada Publishing, 2000. 142 s. ISBN 80-7169-410-X.

ŘEPA, V. Analýza a návrh informačních systémů. Praha: Ekopress, 1999. 403 s. ISBN 80-86119-13-0.

SODOMKA, P. a H. KLČOVÁ. Informační systémy v podnikové praxi. Brno: Computer Press, 2010. 501 s. ISBN 978-80-251-2878-7.

Termín odevzdání diplomové práce je stanoven časovým plánem akademického roku 2022/23

V Brně dne 5.2.2023

L. S.

doc. Ing. Miloš Koch, CSc. garant	doc. Ing. Vojtěch Bartoš, Ph.D. děkan

Abstrakt

Diplomová práce se zaměřuje na návrh a tvorbu softwarového řešení pro televizní vysílání. Výsledkem je komplexní funkční celek, který umožňuje řízení odbavování televizního vysílání včetně provádění hromadných aktualizací v jednotlivých odbavovacích stanicích.

Klíčová slova

software, ffmpeg, IPTV, datové modelování, databáze, PHP, Python

Abstract

The thesis focuses on the design and development of a software solution for television broadcasting. The result is a complex functional unit that enables the management of TV broadcast clearance, including the implementation of bulk updates in individual clearance stations.

Key words

software, ffmpeg, IPTV, data modelling, databases, PHP, Python

Bibliografická citace

PSOTA, Jiří. *Návrh a tvorba softwarového řešení pro televizní vysílání* [online]. Brno, 2023 [cit. 2023-04-30]. Dostupné z: https://www.vut.cz/studenti/zav-prace/detail/152098. Diplomová práce. Vysoké učení technické v Brně, Fakulta podnikatelská, Ústav informatiky. Vedoucí práce Ing. Petr Dydowicz, Ph.D.

Čestné prohlášení

Prohlašuji, že předložená diplomová práce je původní a zpracoval jsem ji samostatně. Prohlašuji, že citace použitých pramenů je úplná, že jsem ve své práci neporušil autorská práva (ve smyslu Zákona č. 121/2000 Sb., o právu autorském a o právech souvisejících s právem autorským).

V Brně dne 30. dubna 2023

……

podpis autora

Poděkování

Rád bych poděkoval panu Ing. Petru Dydowiczovi, Ph.D. za vedení této diplomové práce. Dále děkuji své rodině i přátelům za jejich podporu.

OBSAH

ÚVOD

Tato diplomová práce se zaměřuje na problematiku řízení odbavování televizního vysílání, které disponuje prostředky pro správu aktualizací a také umožňuje přístup pro více dodavatelů videoobsahu.

V kapitole věnované teoretickým možnostem řešení jsou uvedeny softwarové prostředky, s jejichž podporou jsem realizoval návrh řešení zadání. Následující kapitola uvádí přehled o současném stavu možností, jakými lze řešit odbavování televizního vysílání. Tato kapitola je rovněž doplněna o existující alternativy vztahující se k této problematice. Jednotlivé alternativy jsou vzájemně porovnány, vyhodnoceny a na tomto základě jsou určeny nové požadavky, jejichž splnění se očekává od tohoto řešení.

V kapitole vlastního návrhu je rozveden návrh řešení s podporou vývojových diagramů. Na tento návrh navazuje tvorba softwarového řešení, která sestává z praktických fragmentů kódu vývoje, jenž je doplněn o odpovídající popis.

Závěr práce věnuji ekonomickému zhodnocení a přínosům, které návrh nabízí.

VYMEZENÍ PROBLÉMU A CÍLE PRÁCE

Cílem této diplomové práce je návrh a vývoj softwarového řešení pro správu a řízení odbavování televizního vysílání ve firemním prostředí. Komplexnost tohoto řešení sestává od navržení architektury jednotlivých odbavovacích stanic, přes identifikaci a znázornění dílčích procesů, až po logovací systém, prostřednictvím kterého je možné retrospektivně zjistit podstatné informace o stavu odbavování. Pro dodavatele videoobsahu je nezbytné najít takové řešení, aby bylo uživatelsky přívětivé s jednoduchým ovládáním.

V této práci představím prostředky, s jejichž pomocí chci dosáhnout vytyčených cílů. Následně provedu analýzu prostředků v kontextu současného stavu za účelem zjištění, jaké možnosti odbavování televizního vysílání jsou v současné době k dispozici. Na základě zmíněné analýzy provedu návrh a vývoj vlastního softwarového řešení, které bude vyhovovat zjištěným požadavkům.

První kapitola obsahuje odborný popis používaných prostředků užitých ve třetí kapitole. Analytická část práce byla sestavena na základě následující metodologie: nejprve je identifikováno a popsáno firemní prostředí, pro které je tato práce určena. Na základě současných existujících řešení byly nalezeny jejich nedostatky a následně určeny požadavky na nové softwarové řešení. Tyto požadavky byly doplněny o zjištěné poznatky po konzultaci se zadavatelem. Stejně jako analytická část, obsahuje také kapitola pojednávající o vlastním návrhu řešení, konkrétní metodologii. Celý návrh řešení se opírá o předpoklady, jež musí být splněny. Na tomto základě byla navržena architektura a procesy, které jsou klíčové pro samotnou tvorbu řešení, jež se převážně skládá z datového a funkčního modelování.

1 TEORETICKÁ VÝCHODISKA PRÁCE

1.1 Databáze

Tato kapitola pojednává obecně o databázích a různých datových modelech. Součástí je také vysvětlení jazyka SQL a dalších rozšířených možností databází.

Definice 1: Databáze je organizovaný soubor strukturovaných informací neboli dat, které se obvykle ukládají v elektronické podobě v počítačovém systému. Databáze je obvykle řízena systémem pro správu databáze (DBMS – database management system). Data a systém DBMS společně s přidruženými aplikacemi se označují jako databázový systém, často zkráceně jako databáze (22).

Definici je nezbytné doplnit o vysvětlení rozdílu mezi pojmy data a informace. Učiní-li člověk rozhodnutí, která se opírá o data, pak lze tato data nazvat informacemi, protože datům přiřazuje význam a smysl (3, str. 5).

1.1.1 Datové modely

Existuje více možných datových modelů, které se postupem času vyvíjely a v současné době jsou tyto nejznámější:

- lineární,
- hierarchický,
- síťový,
- relační,
- objektový (3, str. 20).

Ad lineární) Tento datový model využívá samostatné tabulky, které nejsou vzájemně propojeny. Pro jednoduché systémy, které nevyžadují vzájemné propojování, je tento model plně dostačující. Jako příklad z praxe tohoto typu modelu lze uvést kartotéku pacientů, kdy každá zásuvka odpovídá jedné tabulce (3, str. 21).

Ad hierarchický) Hierarchický datový model má uspořádané segmenty (segment lze přirovnat k tabulce), což umožňuje lépe pracovat s daty. Příkladem může být situace, kdy učitelé vyučují předměty. Seznam učitelů je uveden v prvním (rodičovském) segmentu a předměty, které učitelé vyučují, jsou umístěny v hierarchicky níže položené úrovni.

Ad síťový) Síťový model je analogií hierarchického modelu, avšak poskytuje možnost libovolného vzájemného propojování požadovaných segmentů (3).

Ad relační) Tento model vznikl zkombinováním několika lineárních modelů, který využívá prostředku nazvaného relační klíč. V současné době se tento model nejvíce používá a je mu věnována samostatná podkapitola.

Ad objektový) Objektový model je nejnovější a je vystavěn na základním prvku, jež se nazývá objekt. Objekt má definované jednak atributy, tak i metody, které určují chování objektu (3, str.23).

1.1.2 Relační datový model

Existují-li množiny, v terminologii teorie množin pak domény, například čísel studentů – D_1, jmen studentů – D_2 a příjmení studentů D_3, pak relace na doménách D_1, D_2, …, D_n je dvojice $R = (R, R^*)$, kde $R = R(A_1: D_1, A_2: D_2, \ldots A_n: D_n)$ je schéma relace, A_i jsou jména atributů a $R^* \subseteq D_1 \times D_2 \times \ldots \times D_n$ je tělo relace (3, str. 26).

Schéma relace je tvořeno atributy <*A, D, T*>, kde *A* je konečná množina hodnot jmen atributů, *D* je zobrazení přiřazující všem jménům atributů a $\in A$ doménu $D(a)$. Počet atributů relace se označuje jako stupeň neboli řád relace a je konstantní. Tělo relace představuje realitě odpovídající podmnožinu kartézského součinu (3, str. 27). Výsledkem násobení libovolného počtu množin je množina *n*-tic. Tato množina již je variabilní. Kardinalita těla relace $m = |R^*|$ se označuje jako kardinalita relace.

Mezi základní pojmy se definují pojmy jako nad-klíč, klíč a primární klíč:

- Nad-klíč, je množina atributů, pomocí kterých můžeme identifikovat řádek tabulky,
- Klíč je nejmenším možným nad-klíčem,
- Primární klíč je vybraným klíčem, který jednoznačně identifikuje řádek tabulky.

Relaci je možné zobrazit pomocí tabulky za následujících podmínek:

1) každý řádek tabulky odpovídá jedné *n*-tici relace,
2) pořadí řádků je nevýznamné,
3) žádné dva řádky nejsou identické,
4) pořadí sloupců je nevýznamné,
5) význam každého sloupce je určen názvem atributu,

6) žádné dva názvy sloupců tabulky (atributů relace) nejsou stejné,
7) hodnoty ve sloupcích jsou atomické (tzn. dále na menší části nedělitelné) (3, str. 27).

V současné době nejpoužívanějšími databázovými systémy jsou Oracle, Microsoft SQL Server, Sybase, MySQL/MariaDB, PostgreSQL (Linux), Informix (IBM), DB-2 (IBM), Interbase, Firebird, SQLLite, Caché (OOP), Terradata (25).

1.1.3 Databázový systém SQL

SQL je zkratka z anglického Structured Query Language, což v doslovném překladu znamená strukturovaný dotazovací jazyk. SQL je standardizovaný databázový nástroj pro práci s relačními databázemi, který byl navržen tak, aby jej mohl používat jak expert, tak i neodborník (4). Další vlastností tohoto nástroje je to, že je neprocedurální, takže konstrukce dotazu je intuitivní.

Nejjednodušší struktura SQL dotazu sestává z klíčových slov SELECT a FROM. Za SELECT se vypisuje množina atributů, které je potřeba zobrazit a za klauzuli FROM se uvádí název relace, ze které se vybrané atributy mají získat. Výčet atributů lze nahradit zástupným znakem hvězdičky, čímž lze vybrat všechny existující atributy dané relace. Součástí tohoto dotazovacího jazyka jsou i rezervovaná slova, která by datoví analytici neměli při datovém modelování používat. Tato slova je důležité psát syntakticky správně, ale velikost písmen se nerozlišuje. Většina významnějších dodavatelů databázových systémů nabízí produkty založené na SQL (4).

Navzdory standardizaci SQL existují u různých dodavatelů databázových systémů určité nuance (24). Zmíněné nuance demonstruje příklad níže.

Pro účely příkladu musí platit předpoklad, že jsou k dispozici dva identické datové modely, přičemž jeden je zapsán v systému MySQL a druhý v systému Microsoft SQL.

Tabulka č. 1: Porovnání rozdílu konstrukce SQL dotazu dvou různých společností
Zdroj: vlastní zpracování

Oracle – MySQL	Microsoft – MS SQL (Lite)
SELECT * FROM tabulka LIMIT 10	SELECT TOP 10 * FROM tabulka

Oba dotazy výše poskytnou shodný výsledek, i když je na první pohled nápadná odlišná konstrukce dotazu. Microsoft nemá definovanou klauzuli LIMIT, takže v tomto případě

se realizuje omezení maximálního počtu zobrazených *n*-tic pomocí doplňujícího výrazu TOP, který se nachází bezprostředně za klauzulí SELECT.

„MySQL je velmi rychlý a robustní systém pro správu relačních databází (RDBMS).“ (6)

MySQL server řídí přístup k datům a využívá architekturu klient-server. Do serveru mohou přistupovat pouze autorizovaní uživatelé. MySQL server Community je veřejně a zdarma dostupný. Existují i další placené edice, například Standard, Enterprise, Cluster CGE (5). Podle (5) je MySQL celosvětově nejpopulárnější open-source databáze.

1.1.4 Sestavení pohledů (views)

Pohled představuje tabulku sestavenou z výsledků prohledávání dotazem SELECT členěné do jedné nebo více tabulek. Jedná se o libovolně strukturovaný dotaz, který je uložený v databázi, ačkoliv sám pohled žádná data neobsahuje. Při zavolání pohledu se virtuální tabulka dynamicky sestavuje (7). Datoví analytici tak mohou vytvořit nad celým datovým modelem několik vybraných pohledů, ke kterým budou mít přístup např. běžní uživatelé, kteří nemusí být experty. Výhodou může být i skutečnost, že SQL server nabízí možnost řízení přístupu pro pohledy (26). Lze nastavit, aby uživatel databáze měl přístup pouze ke čtení vybraných pohledů.

```
CREATE VIEW pohled_alfa AS
SELECT atribut1, atribut2 FROM relace1 INNER JOIN relace2 ON
relace1.key = relace2.key WHERE relace1.atribut2 = 5
```

Příklad výše vytváří pohled s názvem *pohled_alfa* a je uveden v SQL. Následně lze tento pohled zavolat zadáním dotazu níže.

```
SELECT * FROM pohled_alfa
```

Pro zjednodušení lze uvést, že vytvoření pohledu nastavuje substituci pro SQL dotaz a umožňuje snazší používání databáze.

1.1.5 Uložená procedura

Uložená procedura je podprogram podobný podprogramu v běžném výpočetním jazyce, uložený v databázi. Procedury SQL lze použít k vytvoření skriptů pro rychlé dotazování transformací, aktualizaci dat, generování základních sestav, zlepšení výkonu aplikací, modularizaci aplikací a zlepšení celkového návrhu databáze (8). Procedura může

obsahovat vstupní parametry a lze ji spustit příkazem *CALL nazev_procedury(parametry)*.

Již v kapitole pojednávající o SQL vyplynulo, že každý výrobce svých databázových systémů má navzdory standardizaci SQL vlastní proprietární odlišnosti. Není tomu jinak ani v případě uložených procedur.

Společnost Oracle využívá procedurální jazyk PL/SQL, který je výkonný, ale přitom jednoduchý databázový programovací jazyk a také je navržen tak, aby ve své syntaxi zahrnoval příkazy jazyka SQL (9).

Společnost Microsoft disponuje rovněž procedurálním jazykem, který se nazývá TSQL. Funkčnost TSQL je analogická k PL/SQL.

1.1.6 Kurzor

Prostřednictvím kurzoru je možné zpracovávat data záznam po záznamu. Není nutné pracovat s celou relací. V kurzoru je umístěna sada výsledků, která se v několika vlastnostech liší od výsledku příkazu SELECT:

- kurzor se deklaruje odděleně od jeho vlastního vykonání,
- kurzor nese v deklaraci název, na který je možné se odkazovat,
- sada výsledků je v kurzoru tak dlouho otevřená, dokud ji uživatel neuzavře,
- kurzor disponuje speciální množinou příkazů, které slouží pro procházení záznamů (10).

K údajům v kurzorové tabulce lze přistupovat z několika základních kroků, resp. částí kurzoru:

- deklarace kurzoru,
- otevření kurzoru,
- práce s kurzorem a jeho procházení (výběr údajů),
- uzavření kurzoru,
- dealokace kurzoru (10).

1.2 Počítačová síť

Prostřednictvím počítačové sítě je možné propojit několik zařízení, která disponují možností vzájemné komunikace mezi sebou. Tato síť sestává z aktivních a pasivních prvků. Nejpoužívanějším komunikačním protokolem je Ethernet.

1.3 Dělení počítačových sítí

Počítačové sítě lze dělit na základě různých hledisek, kterými mohou být například: rozlehlost, topologie, typ přepojování, postavení uzlů, atd. Pro tuto práci je směrodatným hlediskem rozlehlost sítí.

1.3.1 Rozlehlost sítí

Rozlehlost sítí může být různá, proto nelze vymezit jednoznačné hranice mezi pojmy uvedenými níže.

LAN (Local Area Network)

V překladu do češtiny se jedná o místní počítačovou síť. Z hlediska své velikosti se nejčastěji jedná o podnikovou či domácí síť. Počet připojených síťových zařízení v případě domácí sítě obvykle nepřevyšuje vyšší desítky. Podniková síť bývá značně rozsáhlejší, kdy počet současně připojených zařízení může dosahovat až do vyšších stovek či tisíců. Rozsáhlost podnikové sítě znamená, že je tato síť k dispozici v několika budovách podniku v rámci celého areálu.

WAN (Wide Area Network)

Volný překlad pojmu WAN vyjadřuje rozlehlou síť. Tuto síť tvoří více vzájemně propojených LAN. WAN lze považovat za celosvětovou síť Internet (11).

MAN (Metropolitan Area Network)

Síť MAN obsahuje několik menších LAN, přičemž z hlediska rozlehlosti se jedná o oblast obce, města či regionu. MAN je větší nežli LAN a současně je MAN menší než WAN (11).

1.4 Síťové modely a architektury

1.4.1 Referenční ISO/OSI model

Nutnost disponovat počítačovými sítěmi si několik společností již v minulosti uvědomovalo, a proto si začaly vyvíjet vlastní uzavřené systémy. Postupem času se zjistilo, že by bylo vhodné mít možnost tyto systémy propojit. To však většinou z hlediska kompatibility jednotlivých proprietárních systémů nebylo možné. Na základě těchto poznatků vznikla potřeba stanovit konkrétní pravidla, jež budou jednoznačně definovat, jakým způsobem se budou přenášet data v sítích a mezi nimi.

Řešení zmíněného problému měl poskytnout referenční model OSI, který vypracovala Mezinárodní organizace pro normalizaci ISO (11). „*Referenční model ISO/OSI je obecným modelem definující síťovou komunikaci*“ (12). Dále tento model popisuje sedm vzájemně navazujících vrstev. Schématické uspořádání tohoto modelu se nachází dále.

VRSTVY ISO/OSI
aplikační
presentační
relační
transportní
síťová
linková
fyzická

Obrázek č. 1: Vyobrazení vrstev modelu ISO/OSI
Zdroj: vlastní zpracování

Aplikační orientaci mají první tři vrstvy, přičemž jsou řazené zdola na základě důležitosti. Ostatní vrstvy modelu se zabývají přenosem. Obecný způsob komunikace mezi vrstvami (vertikální spolupráce) je řešena tak, že podřízená vrstva zpracuje úkol a dále jej předá nadřízené vrstvě. Tento model nespecifikuje rozsah a způsob zpracování úlohy. Veškeré související instrukce, jak se budou úkoly zpracovávat, stanovuje výrobce síťového zařízení. Model ISO/OSI rovněž poskytuje doporučení pro zajištění komunikace dvou stejných vrstev modelu mezi různými sítěmi z hlediska horizontální spolupráce.

1.4.2 Architektura TCP/IP

Architektura TCP/IP je odvozena od referenčního modelu ISO/OSI a má odlišný počet vrstev. ISO/OSI disponuje sedmi vrstvami, zatímco TCP/IP pouze čtyřmi vrstvami. Tato architektura je v současné době nejrozšířenější a stala se standardem.

V ISO/OSI se nachází fyzická a linková vrstva, ale v TCP/IP jsou tyto dvě vrstvy reprezentovány jako vrstva síťového rozhraní. Architektura TCP/IP svou první vrstvu nedefinuje, protože může využívat různé přenosové síťové technologie, např. Ethernet. Síťovou vrstvu tvoří přenosový protokol IP. Rozdílné uspořádání modelu ISO/OSI a architektury TCP/IP je patrné na obrázku níže.

Obrázek č. 2: Znázornění vrstev referenčního modelu ISO/OSI a architektury TCP/IP
Zdroj: vlastní zpracování dle 1, str. 84

1.4.3 Architektura Ethernet

Název Ethernet se vztahuje k množině implementací lokálních sítí (LAN), která zahrnuje čtyři základní kategorie:

- Ethernet a specifikaci IEEE 802.3 pro místní sítě LAN, které pracují s rychlostí 10Mbit/s na koaxiálním nebo krouceném kabelu,
- Fast Ethernet, specifikaci lokální sítě LAN pracující s rychlostí 100Mbit/s na krouceném kabelu,
- Giga Bit Ethernet 1000BASE-T, specifikaci sítě, která pracuje s rychlostí 1000 Mbit/s s využitím optických vláken nebo kroucených kabelů,
- Nově projektovanou optickou páteřní architekturu 10 GB Ethernet podle standardu 802.3ae (5, str. 64).

Ethernet se vyznačuje vysokou flexibilitou, relativní jednoduchostí a snadnou implementací. Jedná se o stále nejpoužívanější architekturu, navzdory existenci mnoha lepších možností (17).

V modelu ISO/OSI Ethernet reprezentuje fyzickou a linkovou vrstvu. Ethernet využívá přístupovou metodu CSMA/CD. Předpokladem této metody je nekonečně rychlé šíření signálu, což z fyzikálního hlediska není reálné. Vzhledem ke zmíněnému předpokladu byly stanoveny maximální vzdálenosti, při kterých ještě může tato metoda fungovat. Termín kolizní doména se používá pro maximální rozměr sítě (11).

1.5 Protokoly TCP, UDP, HTTP, SAP

1.5.1 Protokol TCP (Transmission Control Protocol)

Přenosový komunikační protokol TCP poskytuje spolehlivý přenos dat v prostředí IP a také zajišťuje řízení přenosu a plně duplexní provoz včetně multiplexace dat. Z hlediska referenčního modelu ISO/OSI protokol TCP odpovídá transportní vrstvě. TCP doručuje nestrukturovaný tok bytů, který je identifikován pořadovými čísly. Proto mohou aplikace předávat souvislé bloky dat. TCP řeší členění dat do segmentů a předává je vrstvě IP k doručení (17, str. 338).

Spolehlivost přenosu dat je realizována na základě metody doručování paketů mezi koncovými uzly. Metoda funguje tak, že v propojených sítích se označují sekvence bytů, přičemž se očekává potvrzení, zda byly protistranou přijaty příslušné přenosové segmenty. Není-li segment přijat v předem vymezeném čase, pak se takový segment znovu vyšle (17). Díky tomuto mechanismu jsou minimalizovány problémy, které mohou během přenosu vzniknout.

1.5.2 Protokol UDP (User Datagram Protocol)

Stejně jako protokol TCP pracuje i UDP, z hlediska referenčního modelu ISO/OSI, na transportní vrstvě. Protokol UDP je značně jednodušší nežli TCP. UDP nijak nezajišťuje spolehlivost, řízení toků ani nedisponuje funkcí nápravy chyb (17, str. 337).

Vzhledem k jednoduchosti tohoto protokolu, se v praxi využívá např. pro přenos multimédií, kde není potřeba ověřovat spolehlivost doručování dat.

1.5.3 Protokol HTTP (Hypertext Transfer Protocol)

Tento protokol pracuje na aplikační vrstvě modelu ISO/OSI a je definován v dokumentu RFC 2616. Protokol HTTP se používá v rámci celosvětové informační iniciativy World-Wide Web od roku 1990. První verze protokolu HTTP, označovaná jako HTTP/0.9, byla jednoduchým protokolem pro přenos surových dat po internetu. Protokol HTTP/1.0, definovaný v RFC 1945, vylepšil protokol tím, že umožnil, aby zprávy využívaly nový formát a obsahovaly více podrobností, např. metainformace o přenášených datech (1).

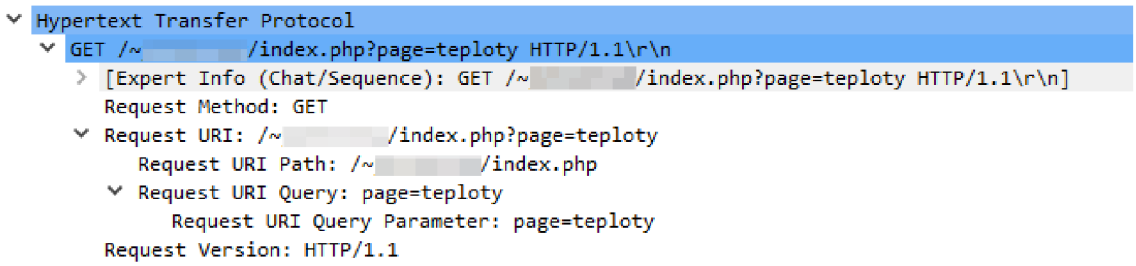
Později vznikaly další požadavky na protokol HTTP, a proto se začaly zavádět nové verze tohoto protokolu – HTTP/1.1, HTTP/2. Tento protokol se především využívá pro webové stránky a pracuje v architektuře klient-server.

Iniciální krok musí provést klient odesláním konkrétního požadavku serveru. Tento požadavek lze serveru zaslat až 9 různými metodami, přičemž povolené metody přenosu jsou stanoveny příslušným webovým serverem v hlavičce HTTP paketu (1).

V této práci postačí následující dvě přenosové metody:

- GET,
- POST.

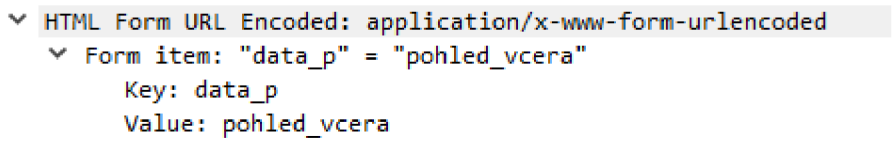
Požadavek GET přenáší data prostřednictvím hlavičky protokolu HTTP. Omezení maximální délky požadavku není dokumentem RFC stanoveno, avšak reálné omezení může existovat, protože záleží na konkrétním použitém webovém serveru, jeho kódování, příp. na použitém webovém prohlížeči (1).



Obrázek č. 3: Snímek z aplikace Wireshark – metoda GET
Zdroj: vlastní zpracování

Na čtvrtém řádku na obrázku výše lze vidět, že byla pro přenos dat použita metoda GET. Řádek níže obsahuje celý požadavek. Je tedy možné číst všechna přenášená data, včetně názvů proměnných a jejich hodnot z hlavičky HTTP paketu.

Oproti tomu se metoda POST liší ve způsobu přenosu dat, protože tato metoda využívá k přenosu tělo paketu HTTP.



Obrázek č. 4: Snímek z aplikace Wireshark – metoda POST
Zdroj: vlastní zpracování

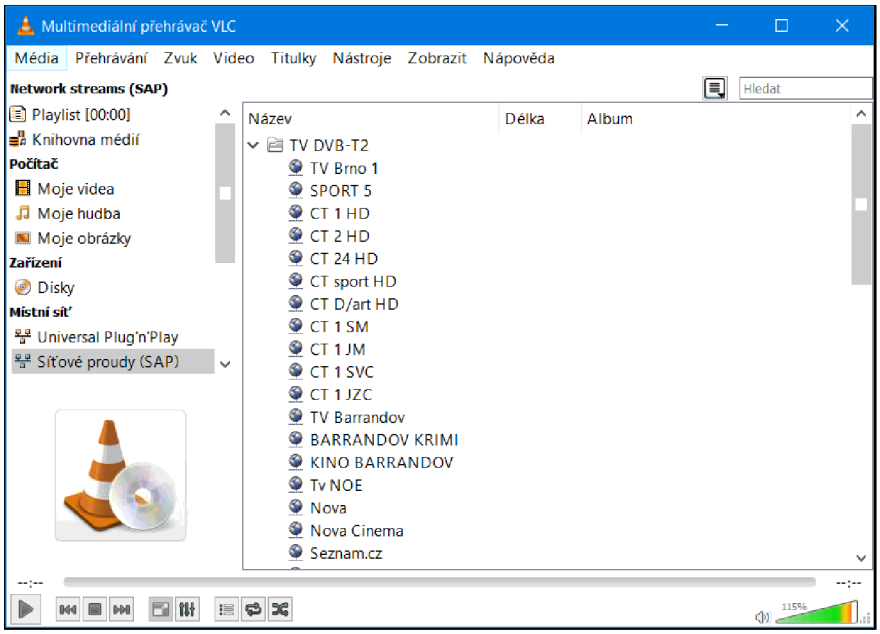
Z obrázku výše je patrné, že klient zaslal serveru požadavek metodou POST s proměnnou *data_p* a její hodnotou: pohled_vcera. Data jsou, stejně jako u metody GET, čitelná jako prostý text, ale běžným způsobem nejsou přenášená data metodou POST viditelná pro uživatele.

Aby bylo zabráněno čtení těchto dat, existuje protokol HTTPS, který vychází z protokolu HTTP, pouze s rozdílem, že přenášená data šifruje SSL nebo TLS certifikátem. Pokud by se příklad výše zopakoval se stejnými parametry, ale s použitím protokolu HTTPS, byla by data viditelná pouze jako dlouhý textový řetězec, tj. jako výsledek šifrovací funkce.

1.5.4 Protokol SAP (Session Announcement Protocol)

Protokol SAP využívá UDP. SAP je definován standardem RFC 2974. Oznamovatel nabízí potenciálním příjemcům v síti informace o dostupných multicastových adresách (a portech), na kterých příjemci mohou přijímat multimediální obsah. Zmíněný oznamovatel pravidelně vysílá paket do sítě na konkrétní multicastovou adresu a port. Neexistuje žádný mechanismus randez-vous. Z tohoto plyne, že oznamovatel si nevede žádnou evidenci o stavu posluchačů a pouze odešle tento paket a jeho doručení se nekontroluje, což odpovídá skutečnosti, protože se k tomuto účelu používá protokol UDP. Není-li síť z hlediska IP adresních rozsahů modifikována, pak se pro adresu vysílání SAP serveru využívá, v případě IPv4, výchozí multicastová adresa 224.2.127.254. Standardem je definován také výchozí port 9875, který musí být využíván SAP oznamovatelem. Pokud je vytvořen adresní prostor multicastových adres, pak dle zmíněného standardu je SAP server k dispozici na nejvyšší adrese v rámci příslušného adresního prostoru (2). Výše použitý pojem „oznamovatel“ pochází přímo ze standardu RFC 2974 a v praxi je možné oznamovatele interpretovat jako server. Ekvivalentem ke druhému pojmu „příjemce“ může být libovolné zařízení v síti, které podporuje protokol SAP. Příklad fungování protokolu SAP v praxi je uveden v následujícím odstavci.

Pro účely tohoto příkladu se předpokládá pouze jedna kolizní doména. Je-li spuštěn SAP server, dochází k pravidelnému vysílání paketů do sítě. Projeví-li zájem o tento paket klient, resp. posluchač, začne naslouchat na výchozí adrese 224.2.127.254:9875, a bude naslouchat tak dlouho, dokud paket neobdrží. Aplikace, které protokol SAP podporují, mají přednastavený časový limit, jak dlouho je přípustné na SAP paket čekat. Jakmile paket přijde, klientovi se zpřístupní jeho obsah a ten jej následně může využít.



Obrázek č. 5: Snímek z aplikace VLC – ukázka využití protokolu SAP
Zdroj: vlastní zpracování

Jak je z obrázku výše patrné, aplikace VLC si získala, pomocí protokolu SAP, seznam všech dostupných multimediálních proudů. V tomto konkrétním případě má každý zobrazený televizní program vlastní multicastovou IP adresu.

1.6 Přenosové metody

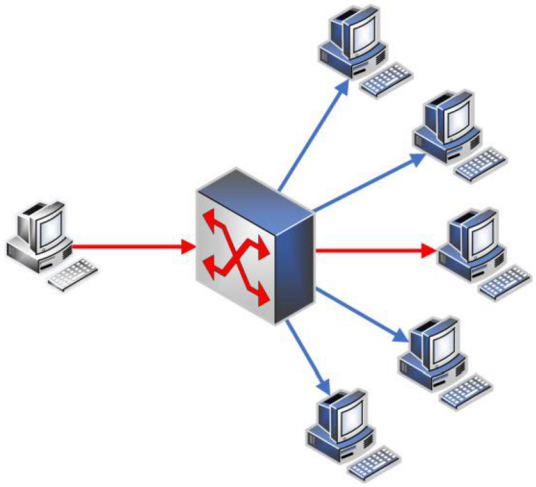
Přenosové metody v rámci LAN lze rozdělit do tří kategorií: *unicast*, *multicast*, *broadcast*. Jednotlivé kategorie se liší počtem cílových uzlů, ke kterým jsou pakety směrovány (17, str. 61).

1.6.1 Unicast

V tomto případě jsou pakety směrovány z jednoho konkrétního místa v síti do druhého určeného místa v síti. Vysílající strana musí jednoznačně definovat příjemce

prostřednictvím adresy. Dále je paket odeslán do sítě a o další okolnosti doručení paketu se stará síť (17).

Schematické znázornění této přenosové metody se nachází níže. Modré čáry reprezentují pouze připojení ke směrovači. Pomocí červených čar jsou označeny cíle, do kterých směruje příslušná přenosová metoda.



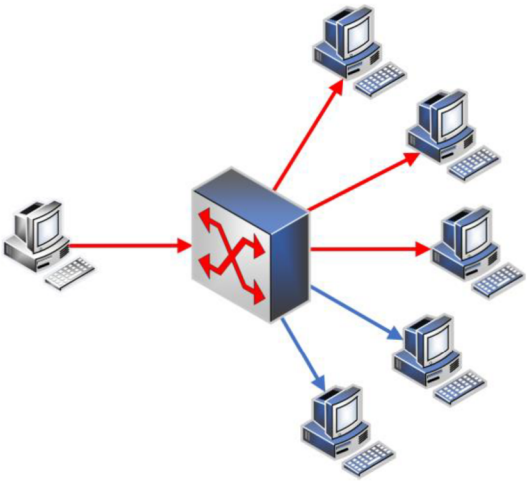
Obrázek č. 6: Schématické znázornění přenosové metody – unicast
Zdroj: vlastní zpracování

1.6.2 Multicast

Tato přenosová metoda má jedno zdrojové místo v síti a určenou skupinu příjemců. Počet příjemců ve skupině není konstantní. Každý uzel sítě se může o členství ve skupině přihlásit, ale i odhlásit. Členství ve skupině je určováno pomocí protokolu IGMP v rámci adresace IPv4. Stejně jako u IPv4 adresace, je i v IPv6 adresaci protokol MLD, prostřednictvím kterého je možné určovat členství v multicastových skupinách.

Pro správnou funkčnost multicastu v síti je nezbytné, aby aktivní prvky v síti podporovaly protokol IGMP. Bude-li v síti použit aktivní prvek, který touto funkcí nedisponuje, vysílání se bude šířit v celé síti jako broadcast.

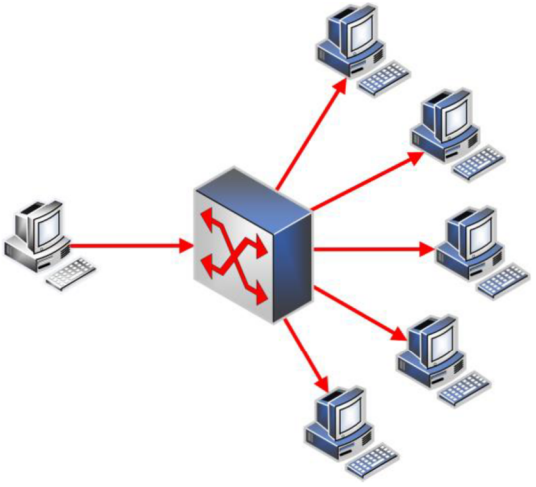
Multicast využívá specifické obory IP adres, které jsou vyhrazené.



Obrázek č. 7: Schématické znázornění přenosové metody – multicast
Zdroj: vlastní zpracování

1.6.3 Broadcast

V případě Broadcastu se paket jednoho uzlu sítě pošle všem ostatním uzlům sítě. Rozposlání duplikovaných paketů všem uzlům je úlohou samotné sítě.



Obrázek č. 8: Schématické znázornění přenosové metody – broadcast
Zdroj: vlastní zpracování

1.7 Programovací jazyky

1.7.1 PHP

„PHP bylo původně zkratkou (z anglického Personal Home Page), ale její význam se změnil po vzoru rekurzivní konvence pojmenování GNU (z anglického Gnu's Not Unix) – takže nyní znamená Hypertext Preprocessor.“ (6, str. 30)

PHP je skriptovací jazyk, který se spouští na straně serveru. Návrh tohoto jazyka byl koncipován přímo pro web (6). Kód jazyka PHP je interpretován webovým serverem. Výsledkem spuštění skriptu PHP bývá nejčastěji kód HTML, který je odeslán uživateli. Prostřednictvím PHP je možné spustit i jiný proces. PHP má otevřený zdrojový kód.

```php
<?php
  $cislo1 = 5;
  $cislo2 = 5;

  echo "Součet čísel " . $cislo1 . " a " . $cislo2 . " je " . ($cislo1 + $cislo2) . ".";
?>
```

Obrázek č. 9: Ukázka kódu PHP
Zdroj: vlastní zpracování

Kód jazyka PHP se píše do souboru s příponou *.php* a v tomto souboru se PHP uvádí do párového tagu *<?*, přičemž první tag musí navíc obsahovat písmena *php* – viz první řádek ukázky kódu výše. V ukázce výše je také demonstrována deklarace dvou proměnných *$cislo1* a *$cislo2*. Oběma proměnným je přiřazena tatáž hodnota 5. Pátý řádek ukázky obsahuje příkaz *echo*, který zprostředkovává výstup pro uživatele.

Výsledkem je následující textový řetězec: *Součet čísel 5 a 5 je 10.*

Je-li v příslušném zařízení instalováno pouze PHP, je možné tento skript spustit pomocí konzole. Pokud je zařízení vybaveno navíc webovým serverem a je-li skript uložen v odpovídajícím adresáři s příslušnými oprávněními, pak si může výše zmíněný výsledek nechat zobrazit libovolný uživatel prostřednictvím webového rozhraní.

1.7.1.1 Superglobální proměnné

PHP disponuje tzv. superglobálními proměnnými. Tyto proměnné je možné volat odkudkoli, např. z funkce.

- **$_GET[“]** – Tato proměnná obsahuje datový typ Asociativní pole. Jedná se o běžné pole hodnot s rozdílem, že hodnoty v tomto poli nejsou indexovány čísly, nýbrž slovy. Vysvětlení praktického užití této proměnné je uvedeno níže.

Existuje-li na serveru s nainstalovaným webovým serverem a skriptovacím jazykem PHP skript s názvem *test.php*, pak je možné tomuto skriptu deklarovat proměnné s hodnotami.

```
http://libovolny.server.d/test.php?promenna1=hodnota1&promenna2=hod2
```

Bude-li odkaz výše v jeho plné délce zadán např. do libovolného webového prohlížeče na straně klienta, pak na straně serveru dojde ke spuštění skriptu s názvem test.php. Zadáním a potvrzením tohoto odkazu ve webovém prohlížeči se serveru odešle požadavek GET. Na základě tohoto požadavku server jednoznačně určí, zda se jedná o relevantní dotaz a poskytne klientovi příslušnou odpověď. Aby bylo možné skript vyvinout tak, aby mohl adekvátně reagovat na měnící se proměnné a hodnoty, využívá ke čtení obsahu požadavku GET superglobální proměnnu. Tato proměnná nese identický název, ovšem doplněný o dolar a podtržítko *$_GET[]*. Plnohodnotný výpis proměnné *$_GET[]*, zprostředkovaný funkcí pro analýzu obsahu proměnných *var_dump()*, je uveden na obrázku níže.

```
array(2) {
  ["promenna1"]=>
  string(8) "hodnota1"
  ["promenna2"]=>
  string(4) "hod2"
}
```

Obrázek č. 10: Výsledek analýzy obsahu proměnné $_GET v PHP
Zdroj: vlastní zpracování

- **$_POST[‘‘]** – Tato proměnná je rovněž superglobální a z hlediska datového typu je identická jako proměnná *$_GET[]*. Tato proměnná lze použít tehdy, když je pro přenos dat zvolena metoda POST.
- **$_REQUEST[‘‘]** - Stejně jako obě předchozí proměnné, je i tahle superglobální s týmž datovým typem. Rozdíl spočívá v tom, že *$_REQUEST[]* získává hodnoty, které server obdržel buď metodou GET nebo metodou POST.

1.7.2 Python

Python je výkonný programovací jazyk, který zahrnuje objektově orientované programování. Dále Python disponuje efektivními vysokoúrovními datovými strukturami. Python je možné používat na většině platformách. Interpreter tohoto jazyka

je k dispozici zdarma ke stažení včetně široké palety knihoven, modulů, nástrojů, a dalších programů (18).

1.8 FFmpeg

FFmpeg je velmi rychlý převodník videa a zvuku, který mimo jiné disponuje funkcí zachytávání živého zdroje zvuku či videa. Dokáže také převádět mezi různými druhy kodeků a kontejnerů. FFmpeg lze také používat pro vytváření živých vysílání v rámci Ethernetu a pro kombinování klidně i více vstupů (19). FFmpeg má otevřený zdrojový kód a je multiplatformní.

FFmpeg, stejně jako všechny další aplikace uvedené v následujících podkapitolách, jejichž název začíná písmeny FF, disponují výhradně konzolovým rozhraním. Existují svobodné projekty, které grafickou nadstavbu pro ffmpeg nabízí, ale pro účely této práce je rozhraní příkazového řádku plně dostačující.

1.8.1 FFplay

Jedná se o velmi jednoduchý a přenosný multimediální přehrávač, který používá knihovny FFmpegu (20).

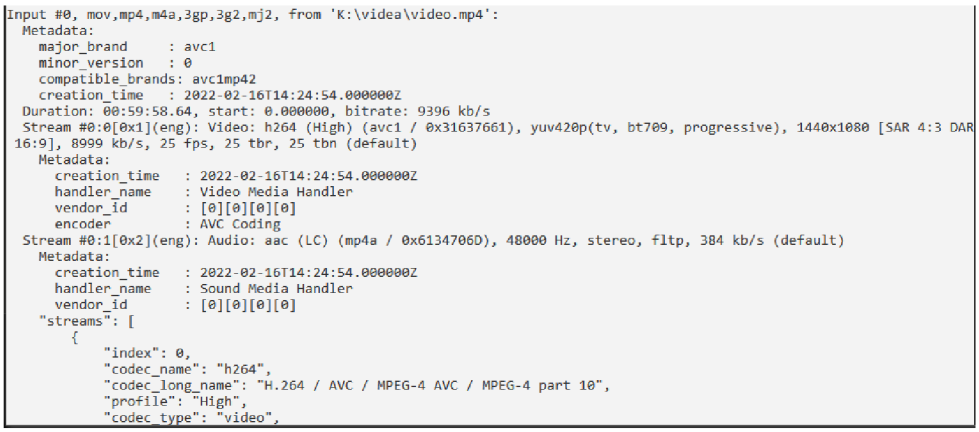
Veškeré parametry zobrazení je nezbytné definovat při spouštění FFplay jako argumenty v příkazovém řádku. Obsahuje-li příkaz spuštění ffplay pouze zdroj přehrávání, výstupem je okno přizpůsobené velikosti videa.

1.8.2 FFprobe

FFprobe slouží k analýze multimediálních souborů, přičemž výstupem analýzy může být několik druhů strukturovaných textových podob. Například při analýze videosouboru lze zjistit, jaký kodek, obrazové rozlišení, datový tok, snímkovou frekvenci a dalšími parametry daný soubor disponuje. FFprobe nabízí možnost analyzování vzdálených zdrojů pomocí URL. Součástí prováděné analýzy je i detekce chyb, takže lze určit, zda je videosoubor přehratelný či nikoli.

```
ffprobe -show_streams -print_format json "K:\videa\video.mp4"
```

Příkaz výše zanalyzuje soubor *video.mp4* a výstup zobrazí v strukturovaném textovém formátu *json* – viz obrázek níže.



Obrázek č. 11: Vyobrazení části výsledku ffprobe v příkazovém řádku
Zdroj: vlastní zpracování

Pro srovnání je níže uveden tentýž příklad, ale místo správného a funkčního videosouboru bude použito poškozené video, které nelze přehrát.

```
C:\>ffprobe -show_streams -print_format json "K:\videa tests\video2-chyba.mp4"
{
[mov,mp4,m4a,3gp,3g2,mj2 @ 0000024b951d0740] moov atom not found
K:\videa tests\video2-chyba.mp4: Invalid data found when processing input

}
```

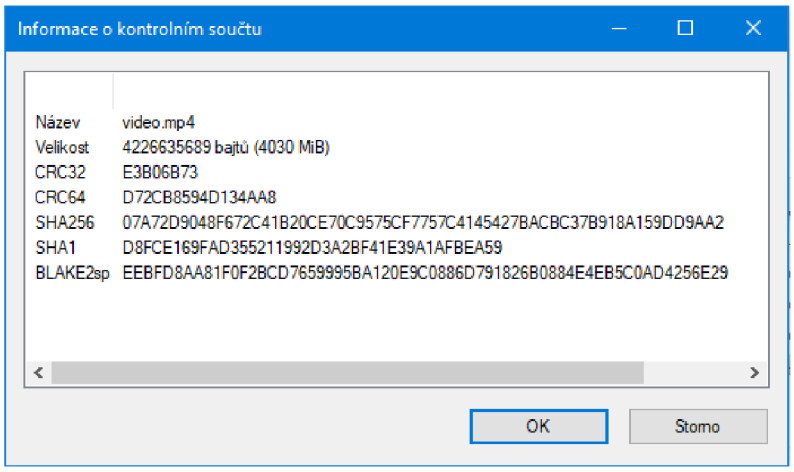
Obrázek č. 12: Vyobrazení výsledku ffprobe v příkazovém řádku
Zdroj: vlastní zpracování

Jak je z obrázku výše patrné, ffprobe tuto skutečnost detekoval a zobrazil chybovou zprávu.

1.9 Kontrolní součty

Kontrolní součty jsou čísla nebo řetězce, které slouží k ověření, zda data byla přenesena nebo uložena bez chyb. Tyto součty se vypočítávají z datových souborů nebo zpráv a jsou použity k porovnání s kontrolním součtem, který byl vypočten po přenosu nebo ukládání dat.

Existuje několik typů kontrolních součtů, včetně paritních bitů, kontrolních součtů CRC, MD5 hashů a SHA1 hashů. Paritní bity jsou jednoduché a používají se k ověření, zda jsou v datech liché nebo sudé bity. MD5 hash je složitější kontrolní součet, který vytváří unikátní řetězec pro určitý soubor nebo zprávu. Hashe MD5 a SHA1 se používají k ověření integrity dat.

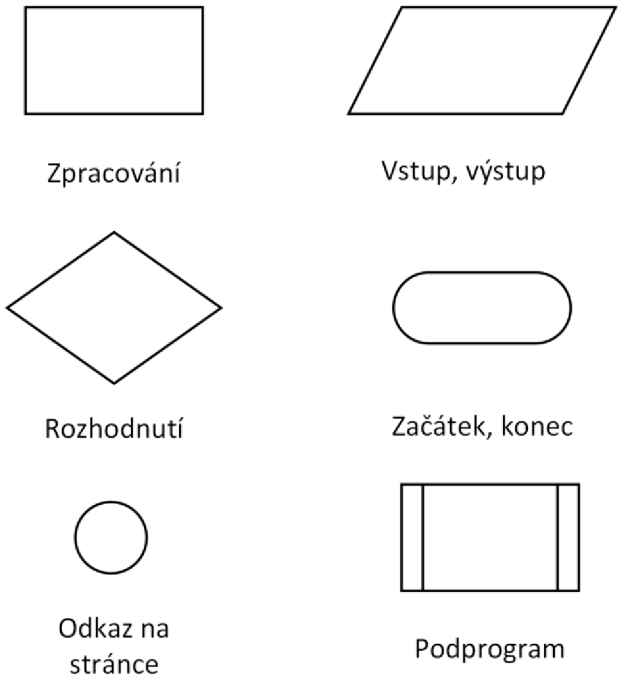


Obrázek č. 13: Ukázka kontrolních součtů vypočítaných aplikací 7zip
Zdroj: vlastní zpracování

1.10 Vývojový diagram

„Slovní zápis algoritmu má jen velmi omezené možnosti použití.“ (21, str. 6)

Pomocí vývojového diagramu lze širšímu okolí uživatelů graficky znázornit algoritmus. Obrazce, uvedené níže, se používají pro grafické znázornění.



Obrázek č. 14: Některé grafické obrazce vývojového diagramu
Zdroj: vlastní zpracování dle: 21, str. 6

Jednotlivé obrazce se spojují tzv. spojnicemi. Spojnice je pravoúhlá a orientovaná čára, což znamená, že má směr a zakresluje se jako šipka. Při konstrukci vývojového diagramu se dodržuje pravidlo shora dolů, a proto není nutné spojnice doplňovat o šipku. Je-li

porušeno zmíněné pravidlo, což může nastat, kdy spojnice mění směr, pak je nezbytné šipku zakreslit. Odkaz na stránce se využívá jako spojka mezi částmi diagramu. Toho se například využívá v situacích, kdy je zapotřebí, aby diagram pokračoval na další straně. Kosočtverec se využívá při rozhodování. Obvykle symbolizuje podmínku jako v programovacím jazyce. Do tohoto obrazce vstupuje jedna spojnice, ale vystupují alespoň dvě; v tomto místě dochází k vyhodnocení podmínky, zda platí či nikoli a dále dochází k větvení. Diagram musí být ucelený a všechny spojnice z jednotlivých obrazců musí mít nějaký cíl.

2 ANALÝZA SOUČASNÉHO STAVU

2.1 Popis firmy

Hlavní náplní firmy jsou elektromontážní práce, výroba rozváděčů nízkého napětí a dodávky elektrických a elektronických zařízení. Dále firma vyvíjí činnost v oblasti strojírenské výroby, kde se specializuje na výrobu prvků pro technologická zařízení mísíren krmných směsí. Podstatnou pracovní náplní firmy je i správa a odbavování 14 informačních kanálů v několika obcích v České republice.

Díky využívání moderních technologií a speciální techniky, jakož i znalosti řešení daného problému a schopnosti rychle reagovat, se úspěšně prosazuje nejen na domácí půdě, ale i v zahraničí. Dokladem toho jsou již realizované dodávky zahraničním zákazníkům. Firma dále poskytuje veřejnou službu internetového připojení v daných lokalitách. Společně s internetovým připojením nabízí tato firma také IPTV a VoIP telefonii (13).

2.2 Historie firmy

Firma byla založena v roce 1991. Na jejím založení se podílely tři fyzické osoby. V roce 1993 se firma transformovala na společnost s ručením omezeným a v dnešní době již zaměstnává 24 zaměstnanců (13).

2.3 Organizační struktura

Z hlediska organizační struktury se jedná o hierarchické uspořádání. Manažer společnosti deleguje činnosti na všechny ostatní zaměstnance. Také pravidelně podává zprávy řediteli společnosti o stavu a průběhu řešení aktuálních projektů. Ředitel tyto zprávy vyhodnocuje a v případě zjištěných nedostatků operativně jedná s manažerem o těchto situacích kvůli zvolení optimálního řešení. Manažer dále vyhodnocuje docházku zaměstnanců z docházkového systému a následně tyto analyzované údaje předává účetnímu oddělení. Na základě toho je účetním oddělením stanovena mzda jednotlivým zaměstnancům. Další povinností manažera je, že v případě zjištěných chyb v běhu některého z jejich serverů, kontaktuje externího specialistu, který má smluvní povinnost vzniklý problém neprodleně vyřešit.

Náplní práce operátorů IT oddělení je především monitoring aktivních síťových prvků a dalších síťových zařízení. Dále se IT zaměstnanci věnují aktivnímu vylepšování monitorovacích prostředků. Součástí náplní práce IT operátorů je pomoc zákazníkům

s řešením technických problémů. Tuto pomoc realizují prostřednictvím telefonních hovorů či osobní návštěvou přímo u klienta.

Firma zaměstnává jednoho externího zaměstnance, který má za úkol dohlížet na provoz odbavovacích stanic v několika obcích a také má povinnost provádět aktualizace vysílacího obsahu. Nadřazeným orgánem dohledu nad televizní činností je Rada pro rozhlasové a televizní vysílání (RRTV). Ta dohlíží na publikovaný obsah v jednotlivých místních informačních kanálech.

2.4 Současné řešení odbavování televizního vysílání

V současné době tato společnost realizuje odbavování informačních kanálů v několika obcích. Obsah těchto kanálů se skládá z úřední desky příslušné obce a z koupeného audiovizuálního obsahu regionální televize. Tato televize dodává nový obsah jednou týdně. Úřední deska se obměňuje v nepravidelných intervalech.

Odbavování se provádí prostřednictvím počítače s obyčejným videopřehrávačem, který obsahuje seznam přehrávaných videosouborů a je nastaven na cyklické přehrávání. Je-li nutné provést aktualizaci videosouborů, ať už ze strany úřadu obce či ze strany regionální televize, postupuje se plně manuálně. V odbavovacích počítačích není řešen vzdálený přístup, a proto je nezbytné při každé aktualizaci videosouborů se fyzicky dostavit k počítači a manuálně nakopírovat nové soubory a zařadit je do seznamu přehrávaných položek v rámci zmíněného videopřehrávače. Celý proces je poněkud komplikovaný, protože v době přibližně před 20 lety, kdy se tyto informační kanály zřizovaly, internet nebyl dostupný ve všech obcích. Od doby zřízení odbavovacích počítačů nedošlo k výrazným aktualizacím ani k modernizaci žádného zařízení. Z tohoto důvodu je současné řešení odbavování zastaralé a již nevyhovující. S tím souvisí vzniklé problémy, které jsou popsány v následující podkapitole.

2.4.1 Problémy současného řešení odbavování televizního vysílání

V dnešní době už mají všechny obce k dispozici připojení k internetu, ale jak již bylo řečeno, odbavovací zařízení jsou zastaralá, neboť je ještě v mnoha stanicích nainstalován dnes již dávno nepodporovaný operační systém Windows XP. Za těchto okolností nelze zpřístupnit daná zařízení on-line. V současném stavu lze sledovat proces, který se uskutečňuje stále stejným způsobem s jistou pravidelností. Jedná se o proces aktualizace videosouborů, který by se měl rozšířit o automatizaci hromadné distribuce. Dále není

v současném stavu řešen problém, který spočívá v přístupnosti odbavování více stranám. To znamená, aby aktualizaci obsahu mohla individuálně provádět nasmlouvaná televize a také obecní úřad.

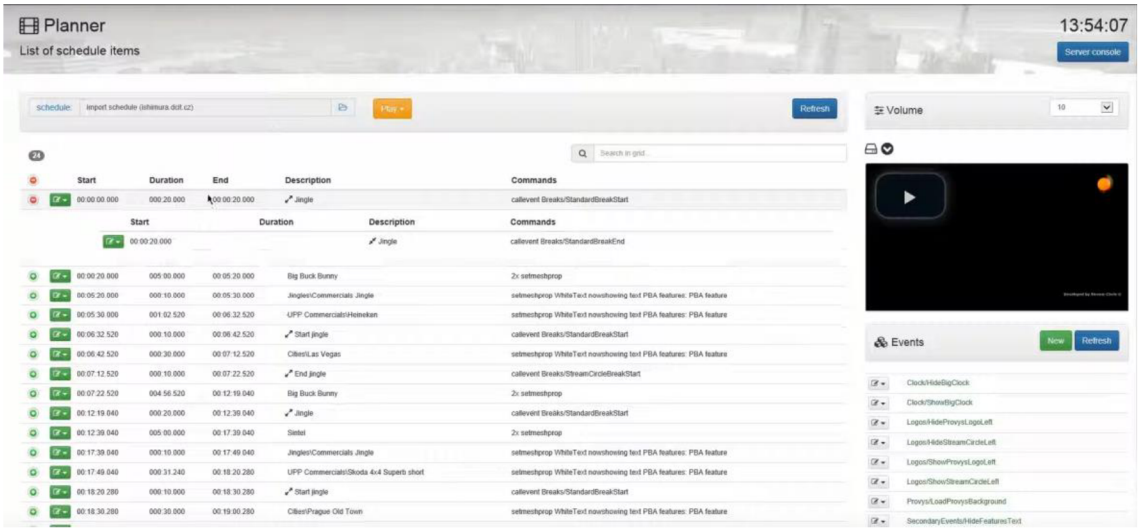
2.5 Dostupná existující řešení

V této kapitole jsou popsána nalezená existující řešení odbavování televizního vysílání.

2.5.1 Stream Circle

Stream Circle je softwarová platforma pro automatizaci a přehrávání televizních kanálů. Tento software podporuje všechny nezbytné části kanálových operací, tj. od správy obsahu, plánování kanálu a vkládání dynamické grafiky až po spouštění seznamů videosouborů a provoz serveru pro přehrávání (14).

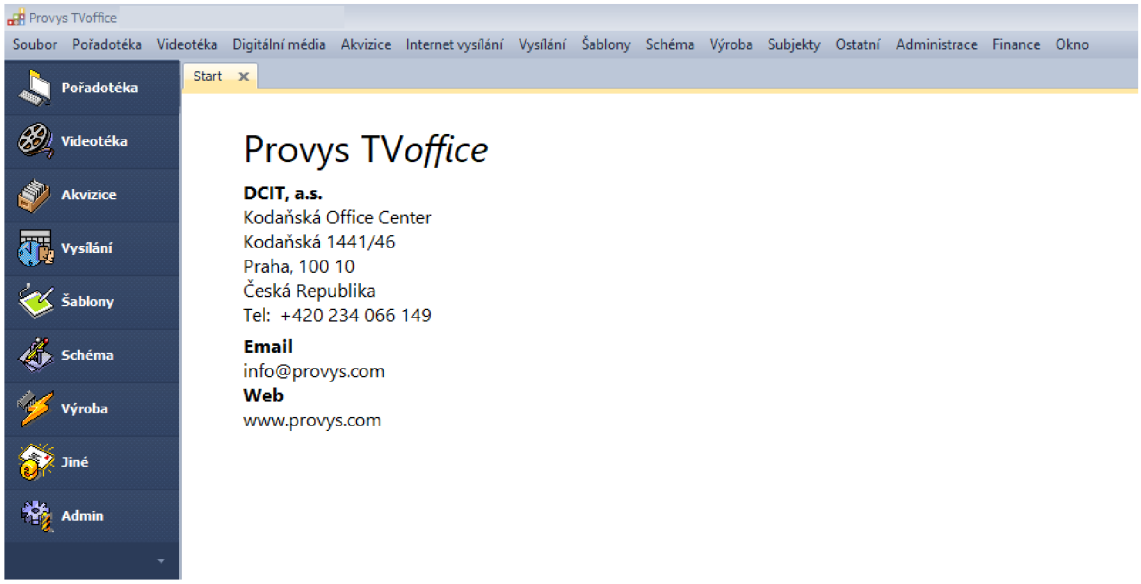
Tento software se využívá pro plnohodnotné odbavování větších televizí. Je vybaven čtením metadat videosouborů, poskytováním informací o programové nabídce divákům, možností odesílání výstupu prostřednictvím multicastu či do libovolného podporovaného fyzického videozařízení, atp… Současně je také vyvinut tak, aby v případě výpadku jednoho serveru byl okamžitě zastoupen náhradním serverem. Jde o licencovaný a komerční software.



Obrázek č. 15: Snímek z aplikace Stream Circle
Zdroj: vlastní zpracování dle 23

2.5.2 Provys

V současné době existuje na trhu česká firma DCIT, a.s., která vyvíjí software PROVYS TV Office. Tento software je modulární informační systém pro řízení televizních stanic (15). Hlavní funkcí tohoto softwaru je odbavování, podružná funkce je archivace. Tento software využívají celoplošné televize v celém světě (16). V ČR software PROVYS využívá např. Česká televize nebo TV Nova. Ve světě tento software využívá např. slovinská veřejnoprávní televize nebo americká televizní stanice AMC. Stejně jako Stream Circle je i tento software licencovaný.

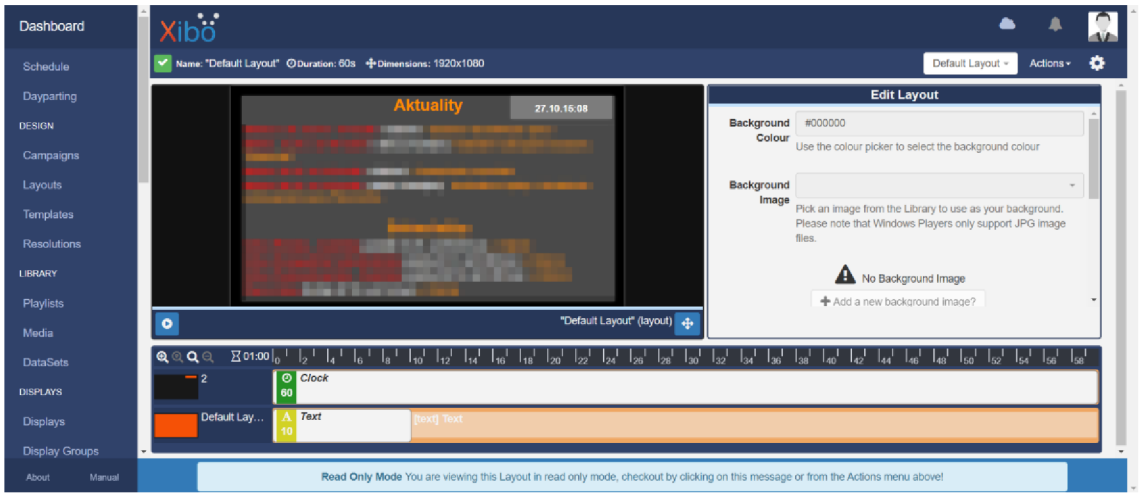


Obrázek č. 16: Snímek z aplikace Provys TV Office
Zdroj: vlastní zpracování

2.5.3 Xibo

Tento software je primárně určen k tvorbě informačních tabulí, nicméně má plný potenciál využití i pro méně náročný způsob odbavování, kterým může být např. informační kanál obce. Je založen na architektuře klient-server, přičemž server může být hostován přímo u Xibo. Strana klienta je reprezentována jako tzv. display, což v praxi znamená, že se tento display připojí k serveru a v reálném čase zobrazuje to, co uživatel připravil na straně serveru. Xibo je multiplatformní. Klienty lze instalovat např. na operační systémy Windows, WebOS, Ubuntu, Android. Jeden připravený display, resp. rozhraní informační tabule, lze zobrazit na několika displejích. Za každý tento display se platí samostatný měsíční poplatek.

Jedná se o všestranný software, který v rámci měsíčního předplatného nabízí mnoho podpůrných služeb včetně zákaznické podpory a přehledné webové správy zobrazovaného rozhraní. Xibo se prezentuje užíváním Open Source kódu pod licencí AGPLv3, avšak jejich softwarové řešení je k dispozici buď ve čtrnáctidenní verzi zdarma anebo v jednom ze tří možných předplatných, případně lze požádat o specifickou objednávku.



Obrázek č. 17: Administrátorské rozhraní softwaru Xibo
Zdroj: vlastní zpracování

2.5.4 Souhrn poznatků

Z předchozích podkapitol bylo zjištěno, že žádný software plnohodnotně nevyřeší zmíněné problémy. Jak již bylo uvedeno, Stream Circle je vhodný pro větší televize, protože obsahuje mnoho funkcionalit, které nejsou firmou KME požadovány. Kromě toho by se celé řešení značně prodražilo, protože by každá odbavovací stanice potřebovala vlastní licenci tohoto softwaru. Tuto potíž může vyřešit Xibo, protože pracuje živě v architektuře klient-server. To znamená, že v případě výpadku internetového připojení by došlo k zastavení odbavování. Dalším negativem jsou jednotlivě placené displeje. Xibo je spíše vhodné do libovolného areálu, kde existuje fyzická místní síť, která zajistí potřebnou stabilitu připojení. Eventuálně lze použít i tam, kde by mohly být tolerovány občasné výpadky.

Provys je až příliš komplexní a robustní. Sice nabízí ještě širší řadu služeb nežli Stream Circle, ale pro uvažované použití je málo vhodný a drahý.

2.6 Koncept nového řešení

2.6.1 Předpoklady

Tato práce pro níže uvedené požadavky předpokládá existenci aktualizovaných a moderních odbavovacích zařízení, jež jsou připojeny, dle dostupných možností stability, ke světové síti internet.

2.6.2 Požadavky

Požadavky na nové řešení musí být sestaveny se zohledněním všech zjištěných aspektů. Je zapotřebí zajistit, aby každé odbavovací zařízení mělo lokálně k dispozici všechny potřebné videosoubory. Dále je nezbytné automatizovat hromadnou distribuci jednotlivých aktualizací. Toho by bylo možné docílit za použití architektury klient-server. Uživatel provede jednu aktualizaci na serveru a ten následně tuto změnu zavede v platnost v jednotlivých odbavovacích stanicích. Posledním požadavkem je zajištění diferencovaného přístupu do odbavovacího řešení. Přínos spočívá v tom, že by aktualizaci svých videosouborů mohl provádět externí dodavatel audiovizuálního obsahu anebo zaměstnanec obecního úřadu, bude-li zapotřebí např. aktualizovat informace na úřední desce. Aktualizace videosouborů musí proběhnout tak, aby nebyl narušen divácký komfort.

Nové řešení musí disponovat uživatelskou jednoduchostí při aktualizaci obsahu. Dále je požadováno přehledné webové rozhraní, aby bylo možné zjistit současný stav odbavování. Toto rozhraní musí nabízet možnost zjištění historie přehrávání.

3 VLASTNÍ NÁVRH ŘEŠENÍ

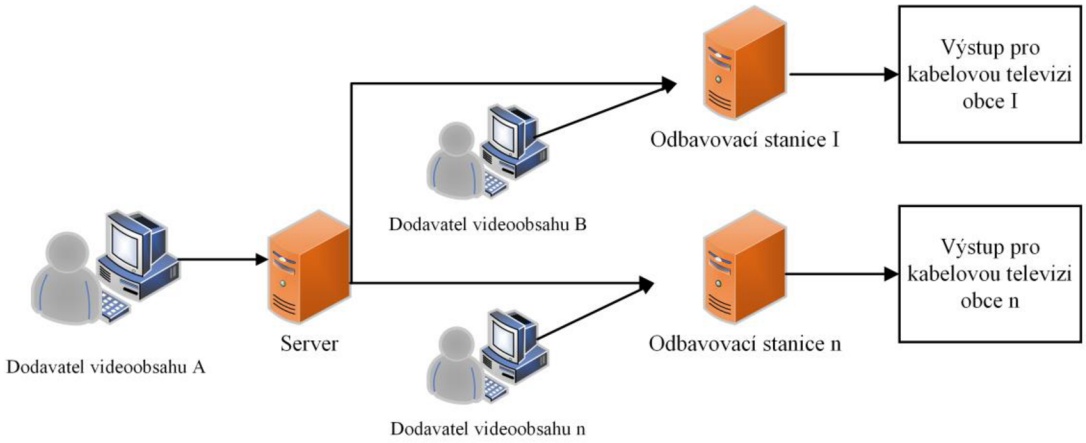
Návrh řešení musí vyhovovat všem požadavkům, které byly zjištěny v závěru předchozí kapitoly. Toto řešení je realizováno za pomocí skriptovacího jazyka PHP, Pythonu, ffmpegu a jeho dalších součástí, webového rozhraní a programovacího jazyka C#.NET.

3.1 Funkční předpoklady

Pro realizaci tohoto návrhu musí být validní následující předpoklady:

- Funkční server s nainstalovaným operačním systémem Ubuntu Server.
- Dodavatelé videoobsahu používají alespoň Microsoft Windows 7 SP1 a novější.
- Odbavovací stanice jsou nainstalované, taktéž s OS Ubuntu Server. Odbavovací stanice, kterým náleží infrastruktura, ve které se šíří televizní vysílání prostřednictvím DVB-C, mají doinstalováno grafické rozhraní (např. LXDE) a také mají dostupný převodník z digitálního na analogový signál.
- Server, všechny odbavovací stanice a všichni dodavatelé videoobsahu mají k dispozici připojení k internetu.

3.2 Architektura navrhovaného řešení



Obrázek č. 18: Ilustrační schéma konceptu architektury řešení
Zdroj: vlastní zpracování

Z obrázku je patrné, že byla navržena architektura klient-server, jak vyplynulo z požadavků na nové řešení. Nyní dodavatel videoobsahu A, což je v tomto případě regionální televize, odesílá aktualizace pouze serveru. Ostatní dodavatelé, což jsou zejména obecní úřady, mohou přímo disponovat se svým vyhrazeným vysílacím časem.

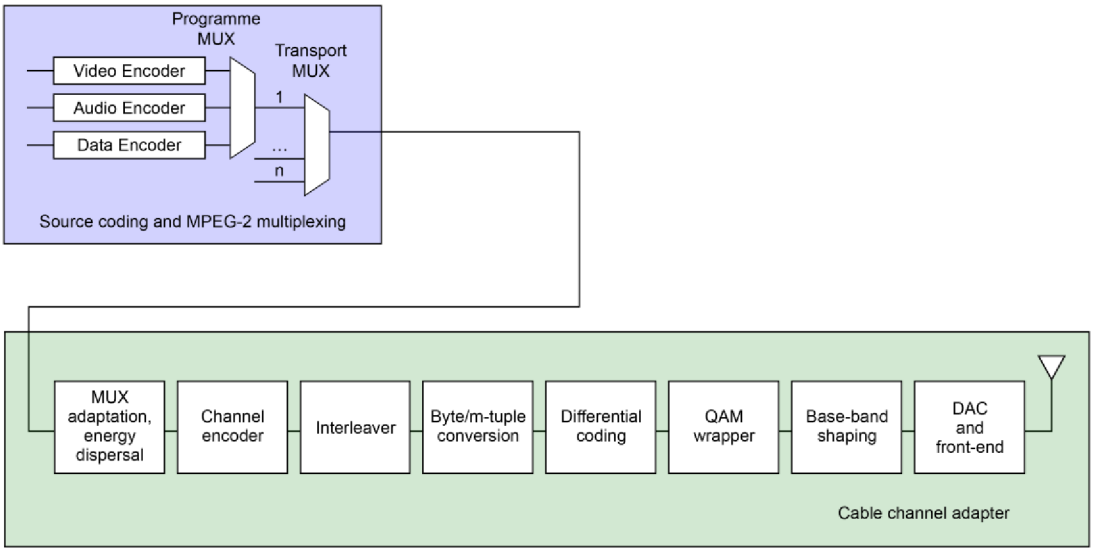
3.2.1 Server

Nejdůležitější součástí tohoto řešení je server, který bude udržovat aktivní síťové spojení se všemi uzly sítě prostřednictvím VPN a dále bude poskytovat službu hromadné distribuce nových videosouborů, kterou se zabývá tato práce. Dále bude na serveru provozována služba managera pro monitorovací systém. K tomuto managerovi budou, rovněž v rámci VPN spojení, připojeni jednotliví agenti v podobě odbavovacích stanic. Konkrétní instalací ani konfigurací monitorovacího řešení se tato práce nezabývá.

3.2.2 Odbavovací stanice

Počet odbavovacích stanic je dle analýzy současného stavu 14. Aktualizace v těchto stanicích jsou přijímány jednak vzdáleně, tj. od serveru a jednak lokálně, tj. od obecních úřadů. I v tomto případě se postupuje podle procesu aktualizace videosouborů.

Druh výstupu z těchto stanic závisí na používané technologií distribuce televizního vysílání. Část těchto stanic odbavuje televizní vysílání prostřednictvím standardu DVB-C a zbylá část infrastruktury je již modernizovaná a televizní vysílání se zde šíří v rámci Ethernetu, tj. IPTV.



Obrázek č. 19: Schéma vysílacího systému DVB-C
Zdroj: (27)

Pro odbavování standardem DVB-C firma využívá analogový signál. Tímto signálem se zásobují enkodéry programového multiplexoru. Programové multiplexory se sdružují do Transportních multiplexorů a dále následuje posloupnost činností, které v tomto

případě není zapotřebí podrobněji rozvíjet. Vše je vyobrazeno ve schématickém znázornění výše.

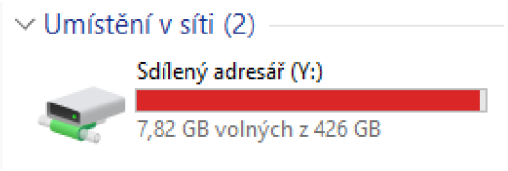
Vzhledem k tomu, že se infrastruktura kabelových televizí postupem času stále modernizuje, všechny odbavovací stanice budou realizovány tak, aby mohly fungovat jak v analogovém, tak i v digitálním režimu.

3.3 Návrh softwarového řešení

3.3.1 Řešení pro dodavatele videoobsahu

Jak již bylo řečeno, dodavatelům videoobsahu musí být poskytován uživatelsky velmi jednoduchý způsob, kterým budou moci provádět aktualizace svých videosouborů a tím přímo zasahovat do svého vyhrazeného vysílacího času.

Pro toto řešení navrhuji využít přenosový protokol SMB, protože lze v operačním systému Windows připojit vzdálený adresář tak, že se uživateli zobrazuje jako „místní diskový oddíl s přiřazeným písmenem jednotky“.



Obrázek č. 20: Příklad použití protokolu SMB v klientském počítači s OS Windows 10
Zdroj: vlastní zpracování

Protokol SMB je optimalizovaný pro datové přenosy. Díky této vlastnosti není zapotřebí vyvíjet alternativní přenos souborů, např. prostřednictvím protokolu HTTPS.

Každému dodavateli videoobsahu lze přiřadit vlastní přihlašovací údaje včetně nastavení přístupových práv. Vzhledem k tomu, že v menších obcích občas dochází k výpadkům internetového připojení, řešení bude doplněno o výpočty kontrolních součtů příslušných videosouborů. Není smysluplné požadovat po uživateli provádění samostatné kontroly, neboť se člověk může snadno zmýlit anebo kontrolu úplně zanedbat. Navíc je značně obtěžující pravidelně tuto kontrolu manuálně provádět. Využije se faktu, že dodavatelé videoobsahů jsou uživatelé OS MS Windows. V programovacím jazyce C#.NET je pro zmíněné dodavatele vytvořen uživatelsky přívětivý program, který provede výpočty kontrolních součtů lokálně, a vypočítané výsledky odešle na server a následně spustí kopírování prostřednictvím zmíněného protokolu SMB.

Server zaregistruje, že je očekávána aktualizace od příslušného dodavatele a od této doby bude pravidelně kontrolovat, zda již bylo kopírování dokončeno. Dále se bude postupovat přesně podle procesu, který v následující kapitole znázorňuje diagram č. 2.

Alternativním řešením by mohla být dodatečná funkce do uživatelské aplikace, která by zajistila informování serveru po dokončení kopírování. Spoléhat se na toto řešení není vhodné, neboť existuje riziko nestabilního internetového připojení. Dále existuje riziko uživatelské. Program by musel být navržen tak, aby ho nebylo možné ukončit v době průběhu kopírování souborů. I kdyby se toto podařilo splnit, může nastat např. chyba v operačním systému uživatele („blue screen of death“) a program bude ukončen i bez přímého vlivu uživatele. Faktorů, které mohou vstoupit do tohoto alternativního řešení je více než v případě výše navrženého řešení. Proto je spolehlivější tuto roli přenechat serveru, který je optimalizován a jištěn i proti ztrátě napájení.

3.3.2 Řešení pro odbavovací stanice

Toto řešení se řídí podle procesu zjišťování nových aktualizací. Jednotlivé odbavovací stanice se budou pravidelně serveru dotazovat, zda existuje nová aktualizace. Stejně jako v předchozí podkapitole i zde lze uvážit alternativní řešení, avšak to má také analogická úskalí. Potíž by mohla nastat, pokud v době dokončení kopírování od dodavatele na server nebude aktivní připojení s nějakou stanicí. Serveru se nepodaří doručit informaci o dostupnosti nové aktualizace a v tento okamžik musí být vytvořeno opatření, které zajistí pozdější informování. Z toho plyne, že by server musel být vybaven systémem řízení front, kdy se bude pravidelně pokoušet kontaktovat vzdálenou stanici. Původní úvaha je v tomto ohledu jednodušší. Není potřeba zřizovat systém řízení front a jakmile bude stanice (klient) znovu on-line, pošle dotaz za účelem zjištění existence nové aktualizace. Tímto je navrženo řešení, aby každá odbavovací stanice disponovala lokálně uloženými a vždy aktuálními videosoubory. Následující odstavec popisuje problematiku odbavování těchto souborů.

V analýze současného stavu bylo uvedeno, že v některých obcích se nachází infrastruktura kabelové televize, která pro přenos využívá DVB-C a v některých dalších obcích se využívá pro přenos IPTV. Pokud je zapotřebí šířit televizní vysílání v rámci Ethernetu, navrhuji k tomuto účelu využít protokol UDP. Tento protokol neověřuje, zda byl paket doručen, a proto je k tomuto účelu vhodný. Každá odbavovací stanice musí být

vybavena ffmpegem. Řízení tohoto softwaru musí být navrženo tak, aby byl dodržen časový harmonogram a současně, aby ffmpeg měl lokální přístup ke všem videosouborům, jež se mají odbavit. Díky tomuto přístupu může ffmpeg tyto soubory otevřít a zahájit živý síťový přenos na předem určenou multicastovou IP adresu a port. Tímto způsobem lze v síti vysílat i více proudů. Aby se potenciální příjemci mohli dozvědět o všech dostupných živých přenosech, aplikuje se protokol SAP. Název, konfigurace a další podrobnosti praktického využití nejsou součástí této práce.

Pro vysílání ve standardu DVB-C je vyžadován analogový vstupní signál. Tak, jak je celé řešení popsáno výše, lze použít i pro tento případ. Ovšem s rozdílem, že se ffmpegu změní cílová IP adresa z multicastové na zpětnou unicastovou adresu (tj. 127.0.0.1). Následně se k ffmpegu spustí ffplay, který bude přijímat proud z lokální smyčky a výstup vykreslí prostřednictvím své grafické karty. Přímo do odbavovací stanice se připojí převodník signálu a díky tomu bude možné odbavovat dle standardu DVB-C.

3.4 Návrh procesů v rámci softwarového řešení

Nedílnou součástí tohoto komplexního softwarového řešení jsou procesy. Tato kapitola se zabývá jejich návrhem, popisem včetně schematického znázornění prostřednictvím vývojových diagramů. Na základě těchto procesů budou navrhovány a realizovány dílčí části řešení.

3.4.1 Proces aktualizace videosouborů

Tento proces je zcela stěžejní a příliš obsáhlý. Z tohoto důvodu je tento proces rozdělen na dva podprocesy.

Proces odeslání videosouborů od dodavatelů na server či odbavovací stanici

Tento podproces je zahajovací. Iniciuje jej uživatel, resp. dodavatel videoobsahu prostřednictvím aplikace s uživatelským rozhraním.

Nejprve se aplikace připojí k databázi k příslušnému serveru. Pokud stav poslední aktualizace indikoval potíže se soubory, je uživatel informován o této skutečnosti a následně je uživatel dotázán, zda si přeje opravit, respektive doplnit poslední aktualizaci. Pokud má zájem uživatel provést opravu, označí ty soubory, které se nepodařily správně překopírovat. Pokud byl stav poslední aktualizace v pořádku, pak uživatel může provést vybrání nových souborů určených pro odeslání na server.

Dále je uživatel dotázán, zda je výběr souborů správný. Pokud není, opakuje se pouze předchozí krok, což je opětovné označení nových, případně poškozených souborů. Následuje kontrola samotných souborů, určených k odeslání na server. Tato kontrola zahrnuje výpočet kontrolních součtů jednotlivých souborů. Dále je potřeba serveru zaslat požadavek GET, který bude obsahovat všechny vypočítané součty, včetně názvů příslušných souborů. Pokud se nepodaří tyto údaje předat serveru, je informován uživatel a dojde k ukončení aplikace. V opačném případě aplikace zjistí, který adresář na serveru je prázdný. Aplikace spustí externí kopírování všech souborů do tohoto adresáře a aplikace se ukončuje.

Jakmile server obdržel od klienta požadavek GET, byl vytvořen záznam v databázi se všemi vstupními údaji a od tohoto stavu se spouští následující proces, který čeká, jakmile bude dokončeno kopírování souborů do příslušného adresáře.

Celý proces je pomocí vývojového diagramu vyobrazen níže.

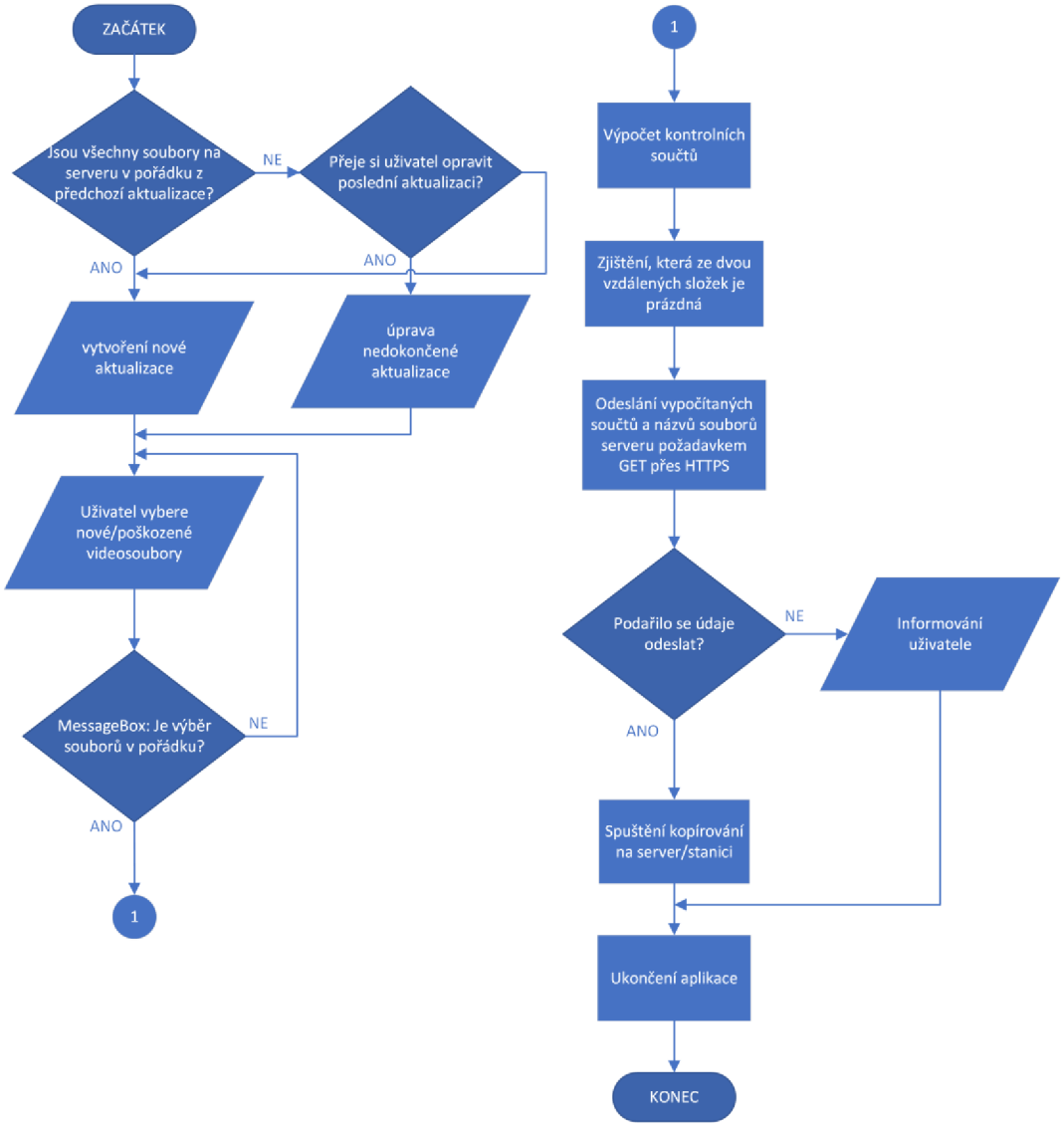


Diagram č. 1: Schematické znázornění procesu odeslání videosouborů od dodavatelů na server či odbavovací stanici
Zdroj: vlastní zpracování

Procesy kontroly dokončení kopírování videosouborů od dodavatelů

Tento podproces přímo navazuje na předchozí podproces a probíhá jak na straně serveru, tak i na straně odbavovacích stanic. Tento podproces je v případě serveru nepřímo iniciován uživatelem a v případě stanic je nepřímo iniciován procesem Zjišťování nových aktualizací. Nepřímá iniciace je z důvodu vyčkávacího bloku v podprocesu. Podrobněji je tento způsob řešení vysvětlen v závěru kapitoly č. 3.9. Podproces začíná čekáním v délce 15 minut, protože jsou kopírované soubory velmi objemné a se zohledněním rychlosti připojení internetu některých odbavovacích stanic, je nepravděpodobné, že by bylo kopírování dokončeno v tak krátkém časovém intervalu. Tato zvolená doba je přiměřená. Následuje získání údajů příslušného dodavatele z databáze. Na základě získaných údajů se vyhodnocuje, zda již existují všechny soubory, které uživatel odeslal. Pokud některý ze souborů chybí, v databázi dojde k přepnutí stavu aktualizace na č. 1 a následuje patnáctiminutová pauza. Dále se vyhodnocuje, zda dané soubory existují alespoň dvě hodiny. Pokud ne, dojde k přepnutí stavu v databázi na č. 2 a opět program čeká 15 minut. Jsou-li obě podmínky vyhodnoceny jako platné, následuje porovnání kontrolních součtů. Nesouhlasí-li jeden z těchto součtů, v databázi se přepne stav na č. 3, poškozené soubory se odstraní a dojde k informování uživatele o všech zmíněných skutečnostech prostřednictvím e-mailu. Poslední kontrolou v tomto procesu je kontrola přehratelnosti nových videosouborů. Bude-li zjištěno, že některý z videosouborů není možné přehrát, navzdory souhlasnému kontrolnímu součtu, je v databázi nastaven stav na č. 4 a opět dojde k informování uživatele e-mailem. Situace, kdy videosoubor není přehratelný a současně jsou shodné kontrolní součty, může nastat tehdy, když dodavatel videoobsahu dodá chybně vytvořený tento soubor. To znamená, že se soubor nepoškodil během přenosu, nýbrž během jeho vytváření. Pokud jsou všechny podmínky platné, je tato aktualizace označena v databázi stavem číslo 5, což vyjadřuje, že se přenos zdařil a že je možné tuto aktualizaci uvést v platnost. Každý výše popsaný stav se vztahuje jednotlivě na každou aktualizaci. Konkrétní význam stavů aktualizací je blíže vysvětlen v tabulce č. 2 u procesu Řízení aktualizací.

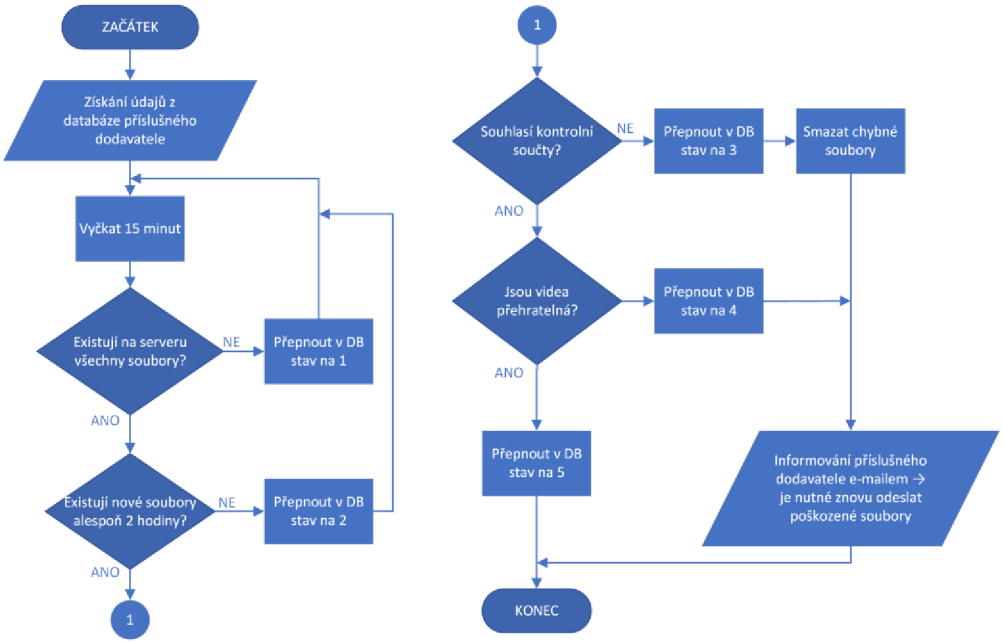


Diagram č. 2: Schematické znázornění procesu kontroly dokončení kopírování videosouborů od dodavatelů
Zdroj: vlastní zpracování

Proces zjišťování nových aktualizací (na odbavovacích stanicích)

Tento proces začíná získáním informací o poslední aktualizaci, která je dostupná na serveru od vybraného dodavatele. Informace se získává přímo z databáze serveru. Pokud již byla nejnovější aktualizace přijata, proces končí. V opačném případě se v databázi příslušné odbavovací stanice vytvoří nový záznam doplněný identifikátorem aktualizace serveru, aby bylo možné rozpoznat, zda již daná stanice přijala aktualizaci či nikoli. Dále dojde ke spuštění stahování všech náležitých videosouborů ze serveru. Posledním krokem tohoto procesu je spuštění procesu s názvem Proces kontroly dokončení kopírování videosouborů od dodavatelů. Celý proces je zobrazen pomocí vývojového diagramu níže.

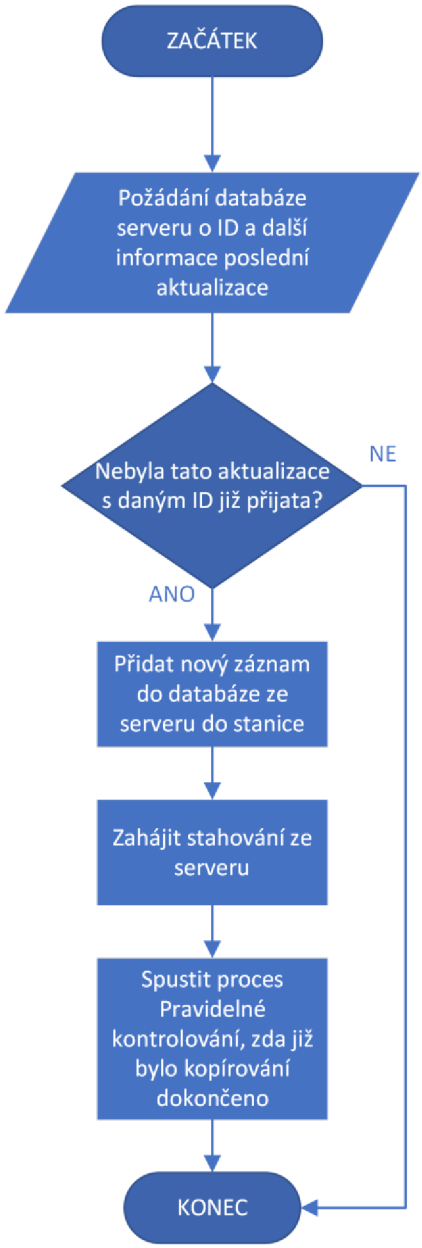


Diagram č. 3: Schematické znázornění procesu zjišťování nových aktualizací
Zdroj: vlastní zpracování

Proces řízení aktualizací

Tento proces je využíván k řízení aktualizací jak na serveru, tak i na jednotlivých stanicích. Existuje však jedno omezení, které spočívá v tom, že server využívá pouze prvních pět ID stavů, zatímco na stanicích se využívá všech osmi stavů. Toto omezení musí být zohledněno a je nezbytné přizpůsobit datový model tak, aby odpovídal tomuto požadavku. Každá aktualizace má definovaný svůj stav, který se v jejím průběhu mění. Jednotlivé stavy jsou uvedeny v tabulce níže.

Tabulka č. 2: Stavy aktualizací včetně popisu
Zdroj: vlastní zpracování

ID stavu	Popis stavu
1	Na serveru nebyly nalezeny všechny očekávané soubory
2	Jeden z očekávaných souborů existuje méně než 2 hodiny
3	Jeden z kontrolních součtů nesouhlasí
4	Jedno z videí není přehratelné. Soubor obsahuje chybu, která nebyla způsobena přenosem.
5	Přenos všech souborů proběhl bez problémů. Všechny soubory jsou připraveny k odbavování.
6	Aktualizace byla uvedena v platnost.
7	Aktualizace již byla odbavena
8	Aktualizace byla přeskočena, protože existovala novější

Tyto uvedené stavy jsou následně použity v datovém modelu. Stavy 2, 3 a 4 obsahují popis obecným způsobem o stavu aktualizace, ze kterých jednoznačně nevyplývá, kde přesně vznikla potíž. Tato formulace je správná, protože, je-li poškozen jeden soubor v rámci celé aktualizace, je neplatná celá aktualizace. Datový model musí zohlednit kromě těchto stavů i možnost zjištění, kde nastal konkrétní problém.

Následující diagram vyjadřuje přijetí nejnovější aktualizace. Nejprve se zjišťuje, zda existuje aktualizace se stavem 5. Pokud ano, dojde k přepnutí stavu 6 na 7 a následně se přepne stav 5 na 6. Pokud existuje ještě starší aktualizace se stavem 5, pak jsou stavy těchto aktualizací nastaveny na 8, což indikuje jejich přeskočení.

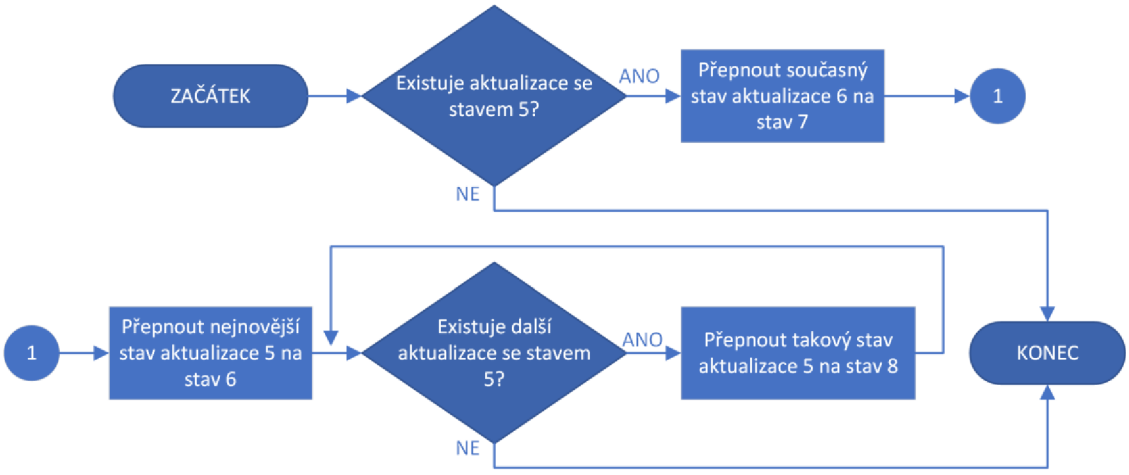


Diagram č. 4: Schematické znázornění procesu řízení aktualizací
Zdroj: vlastní zpracování

3.4.2 Proces přehrávání

Proces přehrávání začíná pokusem o připojení k databázi. Není-li připojení úspěšné, je o této skutečnosti informována odpovědná osoba. V opačném případě se získá aktuální čas z operačního systému za účelem zvolení odpovídajícího videosouboru k odbavování dle připraveného rozvrhu. V dalším rozhodovacím bodě se zjišťuje, zda je dostupná nová aktualizace souborů. Pokud ano, dojde k zavolání uložené procedury v databázi, která provede potřebné změny dle procesu Řízení aktualizací. Nebude-li aktualizace zjištěna, bude se opakovat předchozí odbavování. Dále dojde ke zjištění, zda existují obě složky v přednastavených cestách a pokud ne, opět se upozorní administrátor na tuto chybu prostřednictvím e-mailu a proces se ukončí. Existují-li obě složky, pak se vyprázdní starý obsah z jedné složky a ponechá se nový obsah ve druhé složce. Následně se sestaví kompletní absolutní cesta k chystanému budoucímu odbavování včetně přípony souboru.

V této fázi se vychází ze skutečnosti, že jeden dodavatel dodává čtyři plnohodnotná videa o délce jedné hodiny a druhý dodavatel poskytuje pouze jeden videosoubor s libovolným názvem, ale s požadavkem na cyklické přehrávání. Takže se dle druhu dodavatele určí, zda se odbavování spustí se smyčkou či nikoli. V obou případech se odbavování spouští tak, aby byl dodržen předepsaný rozvrh. Nyní už proces čeká do dokončení odbavování daného videosouboru, přičemž ihned poté se proces vrací na začátek, znovu se pokusí vytvořit nové spojení s databází a pokračuje dále stejným způsobem. Celý proces je znázorněn pomocí vývojového diagramu níže.

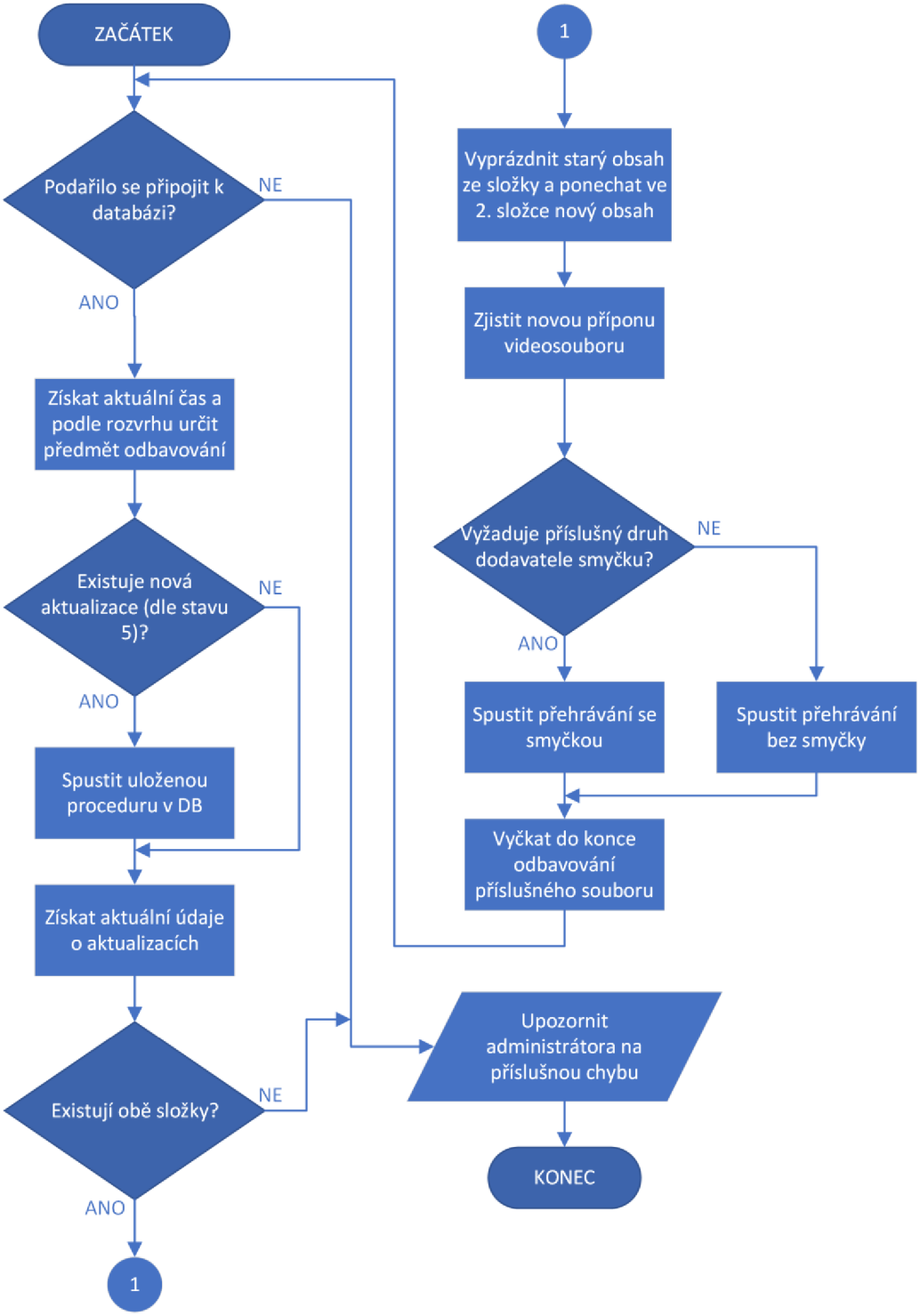
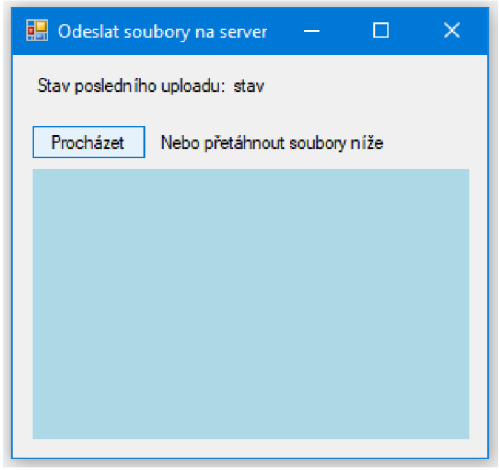


Diagram č. 5: Schematické znázornění procesu přehrávání
Zdroj: vlastní zpracování

3.5 Tvorba aplikace pro dodavatele videoobsahu

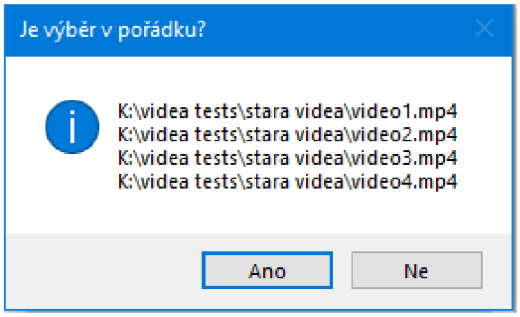
Tato aplikace se řídí dle navrženého procesu v podkapitole 3.4.1. Vzhledem k tomu, že primárním účelem je vytvořit jednoduchou aplikaci z hlediska uživatelského rozhraní, uživatel si může vybrat, jakým způsobem vybere vstupní soubory.



Obrázek č. 21: Ukázka aplikace pro dodavatele videoobsahu
Zdroj: vlastní zpracování

Soubory lze buď přetáhnout, pomocí funkce Drag&Drop, do modrého rámečku nebo lze využít tlačítko Procházet, které otevře dialogové okno operačního systému Windows pro výběr souborů.

Ať už uživatel zvolí jakoukoli možnost vybrání souborů, vždy se zobrazí dotaz pro potvrzení, zda byly vybrány správné soubory – viz dialogové okno níže.



Obrázek č. 22: Ukázka aplikace pro dodavatele videoobsahu v C# – dotaz na správný výběr
Zdroj: vlastní zpracování

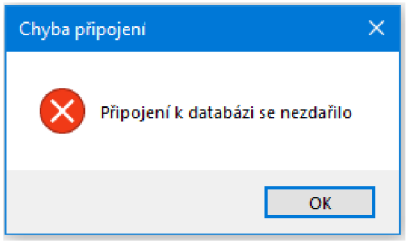
Nejprve se tato aplikace musí pokusit se připojit k databázi k příslušnému serveru. K tomuto účelu je využívána funkce níže.

```
private static DataTable ProvedDotaz(string dotaz)
{
    string connectionString = "server=192.168.35.5;user=uzivatel;password=lZnmJ82s7J8daNG;database=nazev_databaze;";
    using (var connection = new MySqlConnection(connectionString))
    {
        var dataTable = new DataTable();
        var command = new MySqlCommand(dotaz, connection);
        try
        {
            connection.Open();
        } catch
        {
            MessageBox.Show("Připojení k databázi se nezdařilo", "Chyba připojení", MessageBoxButtons.OK, MessageBoxIcon.Error);
            Application.Exit();
        }

        var dataAdapter = new MySqlDataAdapter(command);
        dataAdapter.Fill(dataTable);
        return dataTable;
    }
}
```

Obrázek č. 23: Programový zápis funkce *ProvedDotaz*
Zdroj: vlastní zpracování

Pokud se připojení nezdaří, je zobrazena chybová hláška, která je vyobrazena níže a celá aplikace se ukončí.



Obrázek č. 24: Chybová hláška aplikace pro dodavatele – nelze se připojit k databázi
Zdroj: vlastní zpracování

V rámci této tvorby je nedílnou součástí funkce *VypocitejSHA1*, která slouží pro výpočty kontrolních součtů SHA1 u souboru, jehož cesta je vstupním argumentem této funkce.

```
private static string VypocitejSHA1(string cesta)
{
    using (var sha1 = SHA1.Create())
    {
        using (var stream = File.OpenRead(cesta))
        {
            byte[] hash = sha1.ComputeHash(stream);
            return BitConverter.ToString(hash).Replace("-", "");
        }
    }
}
```

Obrázek č. 25: Programový zápis funkce *VypocitejSHA1*
Zdroj: vlastní zpracování

Výstupem funkce *VypocitejSHA1* je řetězec znaků, jež reprezentuje hash příslušného souboru. Zbývající funkce zajišťuje odeslání požadavku GET serveru.

```
public static string PosliGET(string url)
{
    using (var httpClient = new HttpClient())
    {
        string responseData = "";
        try
        {
            var response = httpClient.GetAsync(url).Result;
            responseData = response.Content.ReadAsStringAsync().Result;
        } catch (Exception e)
        {
            MessageBox.Show("Nepodařilo se navázat spojení se serverem\nDůvod: " +
              e.Message.ToString() + "\nAplikace bude ukončena", "Chyba spojení",
              MessageBoxButtons.OK, MessageBoxIcon.Error);
            Application.Exit();
        }
        return responseData;
    }
}
```

Obrázek č. 26: Programový zápis funkce *PosliGET*
Zdroj: vlastní zpracování

Několik nezbytných kroků (jako jsou např.: funkce pro ověření názvů souborů nebo funkce pro zahájení kopírování), které netvoří dominantní části konstrukce, nejsou uvedeny.

3.6 Řešení na straně serveru

Toto řešení je založeno na relační databází SQL, protože musí poskytovat informace o aktualizacích jednak jednotlivým odbavovacím stanicím a jednak dodavatelům videoobsahu. Nejprve bude navržen datový model.

3.6.1 Návrh datového modelu

Celý datový model je nejprve navržen obecně, kdy jsou jednotlivé relace zobrazeny prostřednictvím tabulek. Praktická tvorba je realizována v nástroji, pro správu MySQL či MariaDB databází, phpMyAdmin. Datový model existuje ve dvou blízkých podobách. Níže je popsán první model, jež se využije na serveru. Za touto podkapitolou následuje další podkapitola, která obsahuje rozšíření tohoto modelu, jež se využije na odbavovacích stanicích.

Tabulka č. 3: Vyobrazení relace *aktualizace_dodavatelu*
Zdroj: vlastní zpracování

Název atributu	Datový typ	Popis
id_aktualizace_PK	INT(8)	
dat_cas	TIMESTAMP	
id_dodavatele_CK	TINYINT(2)	
id_stav_CK	TINYINT(2)	
nazev_slozky	VARCHAR(250)	

První relace má název *aktualizace_dodavatelu* a obsahuje celkem 5 atributů. První atribut je ID aktualizace a má datový typ INTEGER s vlastností unsigned. Je využíván jako primární klíč. Druhým atributem je datum a čas, který využívá datový typ TIMESTAMP. Vždy při vytvoření nové *n*-tice se do hodnoty tohoto atributu automaticky vloží výchozí hodnota, kterou je datum a čas doby vložení. Třetí atribut je využíván jako cizí klíč pro relaci *seznam_dodavatelu*. Následující atribut se nazývá *id_stav_CK* a označuje cizí klíč z relace stavy. Tyto poslední zmíněné cizí klíče jsou připojeny k číselníkům, které jsou definovány v závěru této kapitoly. Posledním atributem je *nazev_slozky*, který sestává z datového typu VARCHAR o délce 250 znaků. Hodnota bude obsahovat název složky, do které se vkládá nová aktualizace. I když je velikost tohoto atributu relativně vysoká, s přihlédnutím k faktu, že se bude ukládat pouze název složky, nehraje tato délka z hlediska databáze zásadní roli. Jedná se o to, že v případě počtu znaků do 255, je princip ukládání na disk řešen podle vzorce: příslušný vložený počet znaků (v bajtech) plus jeden bajt. Změna nastává od dvou set padesátého šestého znaku, kdy se konstantě přičítá místo jednoho bajtu, dva bajty. Tím je dokončen návrh první relace.

Tabulka č. 4: Vyobrazení relace *aktualizace_video*
Zdroj: vlastní zpracování

Název atributu	Datový typ	Popis
id_videa_PK	INT(8)	
id_aktualizace_dodavatelu_CK	INT(8)	
nazev_videa	VARCHAR(260)	
kontrolni_soucet	VARCHAR(250)	
kontrolni_soucet_je_spravne	TINYINT(1)	Používá se jako dat. typ boolean Výchozí hodnota 0

je_prehratelne	TINYINT(1)	Používá se jako dat. typ BOOLEAN Výchozí hodnota 0

Tato relace uchovává všechny potřebné informace o odesílaných souborech. První atribut je *id_videa_PK*. Druhý atribut obsahuje propojení na předchozí relaci prostřednictvím cizího klíče. Jedná se o vztah 1:N, což znamená, že jedna aktualizace má jeden a více souborů. Současně má každý soubor svůj název, včetně přípony, a kontrolní součet. Atribut *kontrolni_soucet_je_spravne* nabývá hodnot True, False, analogicky také 1, 0, což je z hlediska uvedeného datového typu, přesnější interpretace. Dále má tento atribut definovanou výchozí hodnotu 0, protože vzhledem k navrženému procesu se ve výchozím stavu předpokládá, že žádný kontrolní součet nesouhlasí.

Následující relace je používána relací *aktualizace_dodavatelu* pouze jako číselník.

Tabulka č. 5: Vyobrazení relace *seznam_dodavatelu*
Zdroj: vlastní zpracování

Název atributu	Datový typ	Popis
id_dodavatele_PK	TINYINT(2)	
nazev_dodavatele	VARCHAR(250)	

Tato relace je nezbytná pro identifikaci dodavatele videoobsahu. Pomocí ní bude možné v budoucnu škálovat celé toto řešení a poskytnout možnost přístupu i případným dalším dodavatelům.

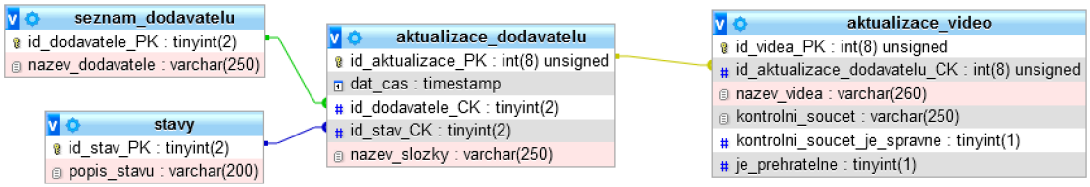
Stejně jako předchozí relace, je i tato používána jako číselník pro relaci *aktualizace_dodavatelu.*

Tabulka č. 6: Vyobrazení relace *stavy*
Zdroj: vlastní zpracování

Název atributu	Datový typ	Popis
id_stav_PK	TINYINT(2)	
popis_stavu	VARCHAR(200)	

Na základě navržených procesů tato relace sdružuje jednotlivé stavy, které mohou v průběhu procesu nastat.

Výsledný ER diagram je zobrazen níže.



Obrázek č. 27: Vyobrazení ER diagramu pro server v prostředí phpMyAdmin
Zdroj: vlastní zpracování

3.6.2 Rozšíření pro odbavovací stanice

Toto rozšíření zahrnuje doplnění atributu *id_aktualizace_serveru* do relace *aktualizace_dodavatelu*, aby bylo možné sledovat identifikátory aktualizací, které ještě nebyly ze serveru získány. Výchozí hodnotou tohoto atributu je NULL, protože porovnávání ID aktualizací není vyžadováno u všech dodavatelů.

Jak bylo předesláno v úvodu předchozí podkapitoly, tato část je zaměřena na rozšíření datového modelu výše tak, aby řešení mohlo v rámci procesu Přehrávání provádět během odbavování logování.

Tabulka č. 7: Vyobrazení relace *videa_log*
Zdroj: vlastní zpracování

Název atributu	Datový typ	Popis
id_log_PK	BIGINT(16)	
id_videa_CK	INT(8)	Cizí klíč
typ_akce	TINYINT(2)	Číselník obsahující popisy akcí
dat_cas_log	DATETIME	
popis	VARCHAR(230)	

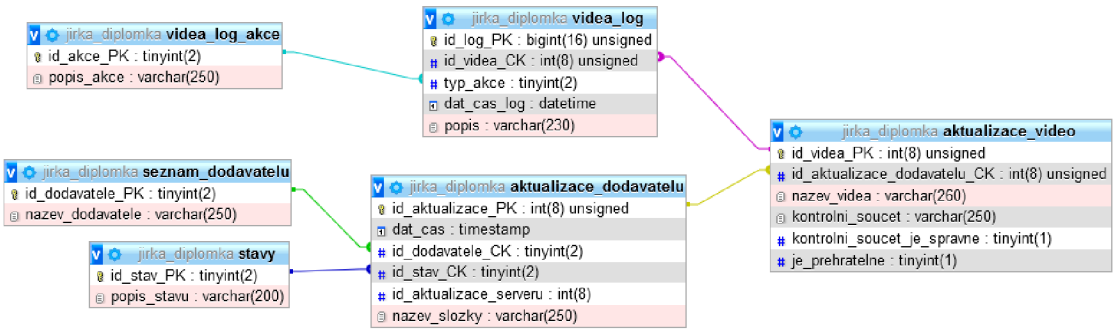
Nová relace *videa_log* je vazbou N:1 připojena k existující relaci *aktualizace_video*. Tato vazba existuje pro to, aby bylo možné zpětně zjistit datum a čas videosouboru, který byl odbavován.

Tabulka č. 8: Vyobrazení relace (číselníku) *videa_log_akce*
Zdroj: vlastní zpracování

Název atributu	Datový typ	Popis
id_akce_PK	TINYINT (2)	
popis_akce	VARCHAR (250)	Cizí klíč

Relace výše obsahuje názvy akcí, jež jsou evidovány v rámci logu v předchozí relaci.

Rozšířený ER diagram je vyobrazen níže.



Obrázek č. 28: Vyobrazení ER diagramu pro odbavovací stanice v prostředí phpMyAdmin
Zdroj: vlastní zpracování

3.6.3 Omezení datového modelu

V rámci datového modelování je tato podkapitola nezbytnou součástí, neboť doplňuje úplnost celého modelu.

Obecnou omezující vlastností jsou datové typy atributů relací, které mají svou konkrétní délku a další vlastnosti. Tato délka omezuje maximální možné řetězce znaků, které lze uložit. V některých případech byla použita dodatečná vlastnost datového typu INTEGER, která se nazývá unsigned. Tato specifická vlastnost omezuje ukládání záporných čísel. Povoluje pouze kladná čísla od nuly.

3.6.4 Pohled aktualizace

Pohled s názvem *pohled_aktualizace* využívá proces Přehrávání ke zjištění všech potřebných údajů. Tento pohled sdružuje všechny relace, jež jsou spjaté s aktualizacemi. Pro vytvoření tohoto pohledu byl využit SQL dotaz, který je zobrazen níže.

```sql
CREATE VIEW pohled_aktualizace AS
SELECT ad.id_aktualizace_PK, ad.dat_cas, ad.nazev_slozky, ad.id_stav_CK, s.popis_stavu,
ad.id_dodavatele_CK, sd.nazev_dodavatele, av.id_videa_PK, av.nazev_videa, av.
kontrolni_soucet, av.kontrolni_soucet_je_spravne, av.je_prehratelne
FROM aktualizace_dodavatelu ad
LEFT JOIN aktualizace_video av ON ad.id_aktualizace_PK = av.id_aktualizace_dodavatelu_CK
INNER JOIN seznam_dodavatelu sd ON ad.id_dodavatele_CK = sd.id_dodavatele_PK
INNER JOIN stavy s ON ad.id_stav_CK = s.id_stav_PK
```

Obrázek č. 29: Programový zápis SQL dotazu pro vytvoření pohledu *pohled_aktualizace*
Zdroj: vlastní zpracování

3.6.5 Uložená procedura

Nedílnou součástí tohoto řešení je také uložená procedura, jejímž úkolem je přijmout novou aktualizaci prostřednictvím řízení stavů. Tato procedura je realizována podle navrženého procesu Řízení aktualizací.

```sql
CREATE PROCEDURE prijmout_aktualizaci (IN vstup_id_dodavatele TINYINT(2))
BEGIN
    DECLARE iterace INTEGER;
    DECLARE id_aktualizace INTEGER(8);
    DECLARE exit_loop BOOLEAN DEFAULT FALSE;
    DECLARE kurozr_aktualizace CURSOR FOR SELECT id_aktualizace_PK FROM
    aktualizace_dodavatelu WHERE (id_stav_CK = 5) AND (id_dodavatele_CK =
    vstup_id_dodavatele) ORDER BY dat_cas DESC;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET exit_loop = TRUE;


    OPEN kurozr_aktualizace;

    SET iterace = 1;
    cyklus: LOOP
        FETCH FROM kurozr_aktualizace INTO id_aktualizace;
    IF exit_loop THEN
            LEAVE cyklus;
        END IF;

        IF iterace = 1 THEN
            UPDATE aktualizace_dodavatelu SET id_stav_CK = 7 WHERE (id_stav_CK = 6)
            AND (id_dodavatele_CK = vstup_id_dodavatele);
            UPDATE aktualizace_dodavatelu SET id_stav_CK = 6 WHERE (id_stav_CK = 5)
            AND (id_aktualizace_PK = id_aktualizace) AND (id_dodavatele_CK =
            vstup_id_dodavatele);
        ELSE
            UPDATE aktualizace_dodavatelu SET id_stav_CK = 8 WHERE (id_stav_CK = 5)
            AND (id_aktualizace_PK = id_aktualizace) AND (id_dodavatele_CK =
            vstup_id_dodavatele);
        END IF;

        SET iterace = iterace + 1;
    END LOOP cyklus;

    CLOSE kurozr_aktualizace;

END
```

Obrázek č. 30: Programový zápis uložené procedury *prijmout_aktualizaci*
Zdroj: vlastní zpracování

Vytvoření procedury stanovuje klauzule *CREATE PROCEDURE*. Za touto klauzulí se nachází název procedury a v kulaté závorce je definována vstupní proměnná s názvem *vstup_id_dodavatele*, jejíž datový typ je TINYINT(2). Tento datový typ je shodný s tím, který je definován v rámci relace *seznam_dodavatelu*. Celý programový kód, z něhož se procedura sestává, je obsažen mezi klauzulemi BEGIN a END.

Nejprve se deklarují 4 proměnné s určitými datovými typy, jež jsou uvedeny níže.

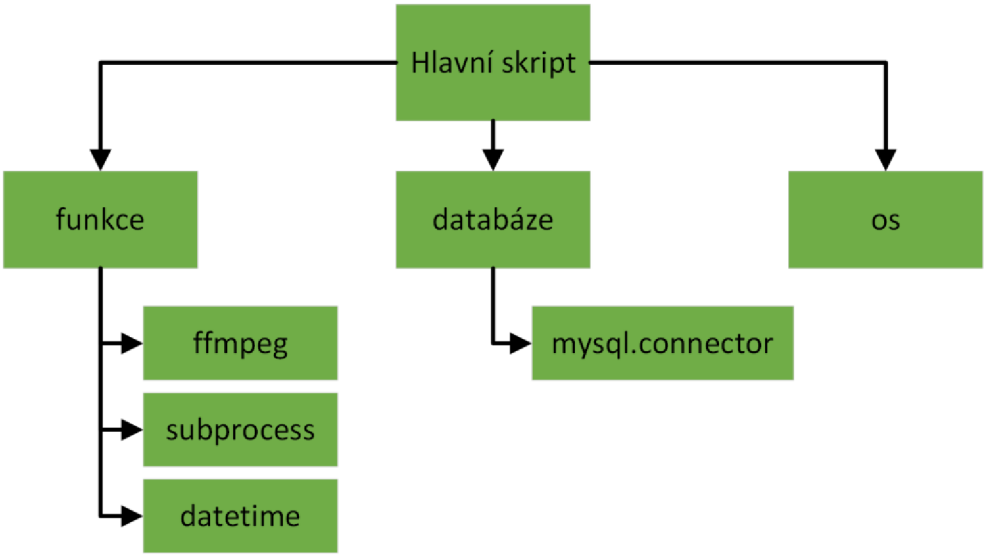
1) *iterace* s datovým typem INTEGER, která bude sloužit pro cyklus.
2) *id_aktualizace* INTEGER(8). Tento datový typ je shodný s typem v relaci *aktualizace_dodavatelu*. Tato proměnná je využívána pro aktuální hodnotu v rámci kurzoru.
3) *exit_loop* BOOLEAN Tato proměnná je inicializována na výchozí hodnotu False. Nastavuje ji kurzor k zastavení cyklu v případech, kdy neobsahuje data a také když kurzor vyhledá poslední prvek definované množiny.
4) *kurozr_aktualizace* CURSOR. Tato proměnná je využívána jako kurzor, přičemž její hodnotou je výsledek následujícího SQL dotazu:

```
SELECT id_aktualizace_PK FROM aktualizace_dodavatelu WHERE
(id_stav_CK = 5) AND (id_dodavatele_CK = vstup_id_dodavatele)
ORDER BY dat_cas DESC;
```

V následující části se nachází příkaz OPEN, který otevírá definovaný kurzor. Dále se nastavuje počáteční hodnota proměnné *iterace* na číslo 1. Poté se spouští cyklus, který na základě kurzoru prochází všechny příslušné *n*-tice a do proměnné *id_aktualizace* ukládá hodnoty z relace *aktualizace_dodavatelu* z atributu id_aktualizace_PK. Kód pokračuje první podmínkou, která dle hodnoty proměnné *exit_loop* buď ukončí nebo ponechá cyklus v chodu. Další podmínka se uskuteční tehdy, a jen tehdy, je-li cyklus v první iteraci. V takovém případě se změní stavy aktualizací ze stavu s identifikátorem 6 na stav 7 a dále se změní stav 5 na 6. SQL dotaz, jímž je definován kurzor, je sestaven tak, aby byly aktualizace procházeny od nejnovější. Provede-li uživatel více platných aktualizací, uvede se v platnost pouze jedna nejnovější. Ostatní aktualizace se stavem 5 budou přepnuty na stav 8, což znamená, že taková aktualizace byla přeskočena. Tato situace je zřejmá z konstrukce druhé podmínky. Ke konci cyklu je uvedeno inkrementování proměnné *iterace* a samotný závěr obsahuje pouze uzavírací klauzule cyklu, kurzoru a celé procedury. Tato procedura bude spouštěna v rámci procesu Přehrávání.

3.7 Řešení na straně odbavovacích stanic

Toto řešení vychází z procesu Přehrávání a lze klasifikovat do několika souvisejících modulů. Vývojový diagram je obecnější a lépe popisuje celé souvislosti. Při praktické tvorbě nebude veškerý kód uveden pouze v jednom skriptu. Využije se možnosti klasifikace, což znamená, že bude vytvořeno více uspořádaných modulů.



Obrázek č. 31: Vyobrazení hierarchie uspořádání modulů
Zdroj: vlastní zpracování

Prvním modulem je sada připravených funkcí. Druhý modul je velmi krátký a obsahuje pouze údaje, jež jsou nezbytné pro připojení k databázi. Třetí modul první úrovně připojuje funkce operačního systému, protože je zapotřebí pracovat s adresáři a soubory na disku.

V rámci třetí úrovně modulů je k funkcím připojen *ffmpeg*, který je již existujícím hotovým modulem. Dále je v téže úrovni připojen *subprocess* a *datetime*. *Subprocess* je používán ke spouštění samotného *ffmpegu* a *datetime* je klíčový pro funkci, která určuje, jaký dodavatel bude v současnou hodinu vysílat dle rozvrhu. Databáze používá adaptér pro možnost připojení k databázi MySQL.

Pomocí příkazu import je možné připojit různé moduly – viz obrázek níže.

```
1   import ffmpeg
2   import subprocess
3   import datetime
```

Obrázek č. 32: Ukázka importu modulů v Pythonu
Zdroj: vlastní zpracování

Modul není nutné načítat celý. Před příkaz import lze doplnit klauzuli *from*, která definuje importování konkrétní části či funkce z celého modulu.

3.7.1 Modul funkce – část I

```
5  def spustit(vstup, smycka = 0):
6      video_format = "mpegts"
7      server_url = "udp://239.0.0.10:1234?pkt_size=1316"
8
9      process = (
10         ffmpeg
11         .input(vstup, stream_loop = smycka)
12         .output(
13             server_url,
14             t = str(datetime.timedelta(seconds = sec_do_hod())),
15             vcodec = "copy",
16             acodec = "copy",
17             f = video_format)
18         .run_async(pipe_stdout=False)
19     )
20     return process
21
22 def sec_do_hod():
23     print("")
```

Obrázek č. 33: Programový zápis funkce *spustit* (Python)
Zdroj: vlastní zpracování

Stěžejní funkcí celého řešení je funkce s názvem *Spustit*, která obsahuje dvě vstupní proměnné. První proměnnou *vstup* je cesta k souboru, který se má odbavovat. Druhou proměnnou je *smycka*, která stanovuje, zda se bude daný videosoubor přehrávat opakovaně či nikoli. Hodnotami může být buď 0 nebo -1. Nula vyjadřuje neopakovat, zatímco -1 znamená opakovat do nekonečna. V této práci jsou tyhle dvě možnosti dostačující.

Celá funkce je zaměřena na nastavení vstupních a výstupních parametrů *ffmpegu*. Nejprve se definuje kontejner. Ten by měl být nejvhodnější *mpegts* především díky své flexibilitě. Následující proměnná *server_url* obsahuje protokol, cílovou multicastovou adresu, na kterou se bude vysílat, port a také velikost odchozích paketů. Použitá adresa i port je pouze ilustrační. Firma sice má připravený IP adresní plán, nicméně tento plán není natolik zásadní, aby bylo nezbytné se jím při realizaci striktně řídit.

Na řádku 9 funkce *Spustit* je deklarována univerzální, uspořádaná a neměnná proměnná *process*. Jedná se o tzv. Tuple, do čehož je možné vložit více prvků různých datových typů. V tomto případě je Tuple použito pro konfiguraci vstupních a výstupních parametrů vysílání ffmpegu. Oba vstupní parametry již byly popsány výše. Níže jsou podrobně vypsány výstupní parametry.

- Cílová adresa vysílání – *server_url*
- Celková doba vysílání – *t*

Hodnota parametru *t* je složena z více funkcí, které jsou lépe patrné z obrázku níže.

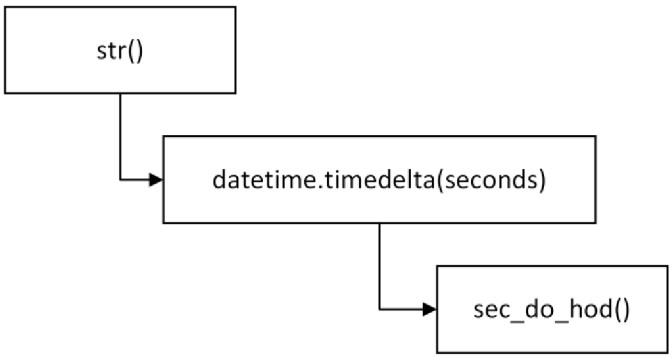


Diagram č. 6: Hierarchie složených funkcí parametru
Zdroj: vlastní zpracování

Dle nejnižší úrovně hierarchie, v diagramu výše, je vnitřní funkcí *sec_do_hod()*. Tato funkce je součástí modulu *funkce* a je definována na konci této podkapitoly. Výsledkem vnitřní funkce jsou sekundy, které jsou vstupní hodnotou pro funkci *timedelta.* Tato funkce převádí sekundy do následujícího časového formátu „HH:MM:SS“. Vstupním argumentem *ffmpegu* je standardní textový řetězec. Z tohoto důvodu je použita ještě vnější funkce *str()*, díky které lze převést čísla či zmíněný časový formát do textového řetězce. Díky této složené sadě funkcí je možné dodržovat odbavovací rozvrh.

- Kodek videa

V rámci ukázky je použita možnost *copy*, která vyjadřuje zachování původního kodeku videa. Jakmile bude řešení implementováno, možnost *copy* se nahradí kodekem H264.

- Kodek audia

Stejně jako v případě video-kodeku je kodek audia řešen identickým způsobem. Ovšem s rozdílem, že při implementaci se využije kodeku AAC.

- Výstupní formát: *mpegts*
- Spuštění podprocesu

Po dokončení všech deklarací, funkce vrátila připravený proces, který se ihned v hlavním skriptu spouští.

```
35 def sec_do_hod():
36     now = datetime.datetime.now()
37     return 3600 - now.second - now.minute*60
```

Obrázek č. 34: Programový zápis funkce *sec_do_hod()*
Zdroj: vlastní zpracování

Funkce *sec_do_hod()* je velmi jednoduchá. Nejprve si získá aktuální čas z operačního systému a ihned na dalším řádku je příkaz *return*, který má za úkol provést aritmetické operace a vrátit vypočítanou hodnotu. Konkrétně se zjišťuje, kolik zbývá sekund do nejbližší celé hodiny.

3.7.2 Úprava zdrojového modulu FFmpeg

Samotný FFmpeg neposkytuje modul pro Python. V této práci je používán modul pocházející z Githubu s názvem *ffmpeg-python*, přičemž tento dlouhý název je zkrácen a pracuje se pouze s označením *ffmpeg*. Lze říci, že tento modul poskytuje pouze rozhraní pro manipulaci s FFmpegem jako takovým.

Tato podkapitola je věnována úpravě zdrojového kódu modulu *ffmpeg-python* vývojáře z Githubu, který se dopustil malé chyby se zásadním dopadem. FFmpeg je v první kapitole definován jako převodník a současně jako nástroj pro vytváření živých přenosů. Na obě skutečnosti sice bral dotyčný vývojář zřetel, ale nedostatečně. Má-li FFmpeg produkovat živý přenos v reálném čase, je zapotřebí přidat vstupní argument *-re* na první místo, ještě před definici vstupu, tj. argumentem *-i*. Vývojář v dokumentaci zmíněného modulu přímo uvádí, jakým způsobem se má argument *-re* zapsat do funkce – viz obrázek níže.

```
5  def spustit(vstup, smycka = 0):
6      video_format = "mpegts"
7      server_url = "udp://239.0.0.10:1234?pkt_size=1316"
8
9      process = (
10         ffmpeg
11         .input(vstup, stream_loop = smycka)
12         .output(
13             server_url,
14             t = str(datetime.timedelta(seconds = sec_do_hod())),
15             vcodec = "copy",
16             acodec = "copy",
17             f = video_format)
18         .global_args("-re")
19         .run_async(pipe_stdout=False)
20     )
21     return process
```

Obrázek č. 35: Programový zápis funkce *Spustit* s globálními argumenty
Zdroj: vlastní zpracování

Jedná se o zcela korektní, ale neúplně funkční zápis. Problém nastane v okamžik spuštění funkce *Spustit*. FFmpeg se spustí se všemi deklarovanými parametry, ale globální argument *-re* je tímto modulem uveden až na konci. Vzhledem k tomu, že FFmpeg vyžaduje zápis tohoto argumentu na úplný začátek, je nezbytné upravit zdrojový kód.

V tomto konkrétním řešení se bude FFmpeg využívat pouze pro vytváření živého přenosu. Z tohoto důvodu lze zdrojový kód upravit tak, aby zmíněný argument byl při každém spuštění uveden na správném místě.

Soubor modulu, jež je zodpovědný za spuštění FFmpegu, se nazývá *_run.py*. Pomocí libovolného textového editoru je nezbytné ve funkci *_get_input_args* přidat prvek do pole s názvem *args*. Prvek v poli bude obsahovat argument *-re*. Celá funkce po úpravě je vyobrazena níže.

```
def _get_input_args(input_node):
    if input_node.name == input.__name__:
        kwargs = copy.copy(input_node.kwargs)
        filename = kwargs.pop('filename')
        fmt = kwargs.pop('format', None)
        video_size = kwargs.pop('video_size', None)
        args = ['-re']
        if fmt:
            args += ['-f', fmt]
        if video_size:
            args += ['-video_size', '{}x{}'.format(video_size[0], video_size[1])]
        args += convert_kwargs_to_cmd_line_args(kwargs)
        args += ['-i', filename]
    else:
        raise ValueError('Unsupported input node: {}'.format(input_node))
    return args
```

Obrázek č. 36: Programový zápis funkce *_get_input_args* po úpravě
Zdroj: vlastní zpracování

3.7.3 Modul funkce – část II

Modul *funkce* je doplněn ještě o jednu funkci s název *urci_video()*, která se používá k určování rozvrhu odbavování.

Její majoritní součástí je časový rozvrh, podle kterého se realizuje odbavování. Vzhledem k smluvním podmínkám, které ukládá provozovateli kabelové televize RRTV, je určen tento rozvrh dle jednotlivých hodin v rámci celého dne. Dále tento rozvrh určuje, jaký dodavatel je na řadě v příslušné hodině.

```
39  def urci_video():
40      now = datetime.datetime.now()
41      rozvrh = {
42          0: [1, "video1"],
43          1: [1, "video2"],
44          2: [2, ""],
45          3: [1, "video3"],
46          4: [1, "video4"],
47          5: [2, ""],
48          6: [1, "video1"],
49          7: [1, "video2"],
50          8: [2, ""],
51          9: [1, "video3"],
52          10: [1, "video4"],
53          11: [2, ""],
54          12: [1, "video1"],
55          13: [1, "video2"],
56          14: [2, ""],
57          15: [1, "video3"],
58          16: [1, "video4"],
59          17: [2, ""],
60          18: [1, "video1"],
61          19: [1, "video2"],
62          20: [2, ""],
63          21: [1, "video3"],
64          22: [1, "video4"],
65          23: [2, ""],
66      }
67      hodina = now.hour
68      if (now.minute == 59 and now.second >= 58):
69          hodina = hodina + 1
70      return rozvrh.get(hodina, "nothing")
```

Obrázek č. 37: Programový zápis funkce *urci_video()*
Zdroj: vlastní zpracování

Funkce, která je patrná na obrázku výše, obsahuje proměnnou *rozvrh*. Datový typ této proměnné je slovník, resp. *dict*. Proměnná *rozvrh* obsahuje 24 klíčů, přičemž každý má přiřazenou hodnotu v podobě dvouprvkového pole. Každý klíč reprezentuje jednu hodinu dne. Prvním prvkem pole je identifikátor dodavatele (shodný s identifikátorem v databázi) a druhým prvkem je název videa, je-li definován.

Analýza současného stavu uvádí požadavek umožnění odbavování více dodavatelů současně. Dále z analýzy plyne, že jeden dodavatel je místní televize, která dodává 4 videosoubory a druhým dodavatelem je příslušná obec, která do vysílání přispívá svojí úřední deskou a informuje své občany o aktualitách v obci. Informativní video obce je vždy jedno a přehrává se ve smyčce.

Na základě výše uvedených požadavků byla odpovídajícím způsobem navržena struktura časového rozvrhu. Aby bylo možné dodržet časový rozvrh RRTV, bylo při návrhu potřebné nějakým kritériem zohlednit pořadí přehrávaných videosouborů. Z tohoto

důvodu je po místní televizi, jakožto dodavateli, vynucováno jednoznačné názvosloví jednotlivých videosouborů. Tento požadavek ovšem není vyžadován u druhého dodavatele, resp. obce. Má-li dodávat tento dodavatel pouze jeden videosoubor, nezáleží na jeho názvu, protože je tento název uložen v databázi a skript, jež je popsán v následující kapitole, tuto skutečnost zohledňuje.

Dále definice funkce pokračuje získáním aktuálního času z operačního systému a získanou hodnotu ukládá do proměnné *hodina*. Následuje podmínka, která je spíše preventivní. Jedná se o pojistku, aby v případě dřívějšího volání této funkce došlo ke spuštění odbavování videosouboru, který se má spustit až od následující hodiny.

3.7.4 Hlavní skript

```
import funkce
import db

while True:
    video = funkce.urci_video()
    vymena_slozek = False

    db.kurzor.execute("SELECT id_aktualizace_PK, dat_cas, id_dodavatele_CK, id_stav_CK FROM aktualizace_dodav
    odpoved = db.kurzor.fetchall()
    if (len(odpoved) > 0):
        db.kurzor.execute("CALL prijmout_aktualizaci(" + str(video[0]) + ")")
        odpoved = db.kurzor.fetchall()
        vymena_slozek = True

    db.kurzor.execute("SELECT id_aktualizace_PK, dat_cas, nazev_slozky, nazev_videa, id_stav_CK, id_videa_PK
    odpoved = db.kurzor.fetchall()

    if (vymena_slozek == True):
        cesta = funkce.cesta + str(video[0]) + "/"
        if (odpoved[0][2] == "vid_A"):
            print("smazat obsah slozky .vid_B")
            funkce.vymazat_slozku(cesta + ".vid_B")
            funkce.os.rename(cesta + "vid_A", cesta + ".vid_A")
            funkce.os.rename(cesta + ".vid_B", cesta + "vid_B")
        elif (odpoved[0][2] == "vid_B"):
            print("smazat obsah slozky .vid_A")
            funkce.vymazat_slozku(cesta + ".vid_A")
            funkce.os.rename(cesta + ".vid_A", cesta + "vid_A")
            funkce.os.rename(cesta + "vid_B", cesta + ".vid_B")
```

Obrázek č. 38: Programový zápis Hlavního skriptu – část 1
Zdroj: vlastní zpracování

Hlavní skript začíná importováním dvou předem připravených modulů. Konkrétně se jedná o moduly *funkce* a *db*. Tento skript je realizován podle procesu Přehrávání. Jak je z obrázku patrné, celý kód je vložen do cyklu *while*, který probíhá do nekonečna, díky výrazu True. Cyklus začíná získáním hodnoty funkce *urci_video()* z modulu *funkce*. Tato hodnota je uložena do proměnné *video*. Tímto krokem je určeno, co by se v danou chvíli mělo začít odbavovat. Další proměnnou je *vymena_slozek*, která dále rozhoduje, kdy se přijme nová aktualizace. Součástí tohoto rozhodnutí je SQL dotaz, kdy se zjišťuje, zda se v databázi nachází nová aktualizace se stavem 5. Pokud ne, pošle se další SQL dotaz,

tentokrát na stav 6, což znamená že je žádána poslední přijatá aktualizace. V případě, kdy bude nalezena nová aktualizace, dojde ke spuštění uložené procedury *prijmout_aktualizaci* v databázi. Ta provede sérii změn stavů. Následně se nastaví hodnota proměnné *vymena_slozek* na True a skript pokračuje stejným způsobem, jako v případě nedetekované nové aktualizace, tj. pošle se SQL dotaz na aktualizaci se stavem 6. K oběma zmíněným SQL dotazům je nutné poznamenat, že jsou tyto dotazy doplněny o klauzuli *WHERE*, která podmiňuje získané výsledky. Jedná se o omezení podle identifikátoru dodavatele. Toto omezení je velmi důležité, protože každý dodavatel má svoje sady aktualizací. Vzhledem k tomu, že z obrázku výše nejsou patrné SQL dotazy, jsou podrobněji uvedeny níže.

```
db.kurzor.execute("SELECT id_aktualizace_PK, dat_cas, id_dodavatele_CK,
id_stav_CK FROM aktualizace_dodavatelu WHERE (id_stav_CK = 5) AND
id_dodavatele_CK = " + str(video[0]) + " ORDER BY dat_cas DESC")
```

Obrázek č. 39: Programový zápis Hlavního skriptu – část 1 – SQL dotaz 1
Zdroj: vlastní zpracování

SQL dotaz, uvedený výše, využívá relaci *aktualizace_dodavatelu*, zatímco dotaz uvedený níže využívá *pohled_aktualizace*. První dotaz má za úkol pouze detekovat aktualizaci připravenou k přijetí. Z druhého dotazu se již využívají údaje, jež jsou nezbytné pro pokračování skriptu.

```
19      db.kurzor.execute("SELECT id_aktualizace_PK, dat_cas, nazev_slozky, nazev_videa,
        id_stav_CK, id_videa_PK FROM pohled_aktualizace WHERE (id_stav_CK = 6) AND
        id_dodavatele_CK = " + str(video[0]) + " ORDER BY dat_cas DESC")
```

Obrázek č. 40: Programový zápis Hlavního skriptu – část 1 – SQL dotaz 2
Zdroj: vlastní zpracování

Pokud bylo rozhodnuto, že má dojít k přijetí nové aktualizace, následuje vymazání starého obsahu příslušné složky a k přejmenování obou složek. Název příslušné složky je získán z databáze.

Přejmenovávání názvů složek je bezpečností opatření pro minimální působení rizika uživatele. Prakticky toto opatření existuje proto, aby byl uživatelům omezen přístup do složky, ze které se právě realizuje odbavování. Návrh tohoto řešení předpokládá, že každý uživatel, resp. dodavatel, bude mít ve svém počítači připojený síťový disk, do kterého se budou kopírovat vybrané soubory prostřednictvím Aplikace pro dodavatele. Kdyby byly obě složky uživateli dostupné, mohlo by, ze strany uživatele, dojít ke smazání obsahu jedné nebo obou z nich. Této situaci lze předejít následujícím opatřením. Složka, z jejíhož

obsahu se právě čtou soubory v rámci odbavování, bude už ve fázi přijetí aktualizace přejmenována. Vlastností linuxových operačních systému je to, že názvy souborů či složek začínají tečkou a nejsou pro běžné uživatele viditelné. Této vlastnosti je v tomto řešení využito. Tímto krokem odpadají případné problémy spojené s konzistencí dat.

Toto opatření by mohlo být také realizováno úpravou přístupových práv k dané složce. V tomto případě musí být správně nastaven vlastník složky, aby mohl spravovat přístupová oprávnění pro ostatní uživatele. Tato varianta je sice mnohem více spolehlivá vůči faktoru uživatele, ale pro potřeby této práce postačí předchozí jednodušší způsob.

```
31      index = 0
32      i = 0
33      for radek in odpoved:
34          if (radek[3].find(video[1]) != -1):
35              index = i
36              break
37          i = i + 1
38
39      if (len(video[1]) == 0):
40          nazev_videa = odpoved[index][3]
41          smycka = -1
42      else:
43          nazev_videa = video[1] + str(odpoved[index][3][odpoved[index][3].find("."):len(odpoved[index][3])])
44          smycka = 0
45
46      cela_cesta = funkce.cesta + str(video[0]) + '/.' + str(odpoved[index][2]) + '/' + nazev_videa
47
48      sql = "INSERT INTO videa_log (id_videa_CK, typ_akce, dat_cas_log) VALUES (%s, %s, %s)"
49      hodnota = (odpoved[index][5], 1, funkce.datetime.datetime.now())
50      db.kurzor.execute(sql, hodnota)
51      db.db_conn.commit()
52
53      proces = funkce.spustit(cela_cesta, smycka)
54      proces.wait()
```

Obrázek č. 41: Programový zápis Hlavního skriptu – část 2
Zdroj: vlastní zpracování

3.8 Příjem dat od dodavatelů

Tento proces je využíván jak na serveru, tak i na jednotlivých stanicích. Jeho úkolem je přijmout textová data od dodavatele, ověřit jejich správnost a následně všechna data odeslat do příslušné databáze.

Všechna data přijímá skript PHP v proměnných, jež jsou uvedené v tabulce níže.

Tabulka č. 9: Struktura vstupních proměnných v rámci procesu Příjem dat od dodavatelů
Zdroj: vlastní zpracování

Názvy proměnné	Datový typ	Popis
id_dodavatele	INTEGER	
Slozka	STRING	Název cílové složky, do které je kopírováno
*ns**	STRING	Názvy souborů
*ks**	STRING	Kontrolní součty daných souborů

Proměnná *ns** je pouze zkráceným zápisem několika proměnných. Za hvězdičku se dosazuje kladné celé číslo od jedničky (včetně) až do konečného počtu všech souborů. Tímto způsobem lze jednoduše s těmito proměnnými pracovat, což je níže demonstrováno přímo na konkrétním kódu.

Skript začíná deklarací funkce s názvem *konec*, která má vstupní proměnnou *$chyba*, jejíž hodnoty jsou desetinná čísla. Každé číslo reprezentuje některý z chybových stavů, které se mohou v průběhu kódu vyskytnout. Ke konci této funkce se uživateli zobrazí, o jakou chybu se jedná a následuje ukončení celého skriptu pomocí funkce *exit()*.

```
7   function konec($chyba = 0) {
8       $popis_chyby = "";
9       switch ($chyba) {
10          case 1:
11              $popis_chyby = "Proměnná složka nebo id_dodavatele není definována";
12              break;
13          case 1.1:
14              $popis_chyby = "Chybný datový typ vstupní hodnoty";
15              break;
16          case 2:
17              $popis_chyby = "Některý z názvů souborů nemá kontrolní součet";
18              break;
19          case 3:
20              $popis_chyby = "Nebyla definována ani jedna vstupní proměnná ns nebo ks";
21              break;
22          case 4:
23              $popis_chyby = "Proměnná ns nebo ks je definovaná, ale neobsahuje žádnou hodnotu";
24              break;
25          case 5:
26              $popis_chyby = "Byl použit nepovolený znak.";
27              break;
28          case 6:
29              $popis_chyby = "Chyba v dotazu SQL";
30              break;
31          default:
32              $popis_chyby = "Nespecifikovaná chyba";
33      }
34      echo  "ID chyby: " . $chyba . " - " . $popis_chyby . "<br>";
35      exit();
36  }
```

Obrázek č. 42: Programový zápis funkce *Konec*
Zdroj: vlastní zpracování

Skript pokračuje kontrolou, zda byl zadán název složky, identifikátor dodavatele a zda je tento identifikátor číslo – viz níže. Pokud je některá z podmínek porušena, dochází k volání funkce *Konec*.

```
38      if (!isset($_GET["slozka"])) {
39          konec(1);
40      }
41
42      if (!isset($_GET["id_dodavatele"])) {
43          konec(1);
44      }
45
46      if (!is_numeric($_GET["id_dodavatele"])) {
47          konec(1.1);
48      }
```

Obrázek č. 43: Programový zápis skriptu v rámci procesu Příjem dat od dodavatelů – část I
Zdroj: vlastní zpracování

Dále se zjišťuje, kolik je definovaných proměnných *ns** a *ks**. Děje se tak zejména proto, aby bylo možné porovnat, zda má každý soubor svůj kontrolní součet.

```
50      $i = 1;
51      $pocty["ns"] = 0; //název souboru
52      $pocty["ks"] = 0; //konstrolní součty
53      $konec = true;
54      while($konec == true) {
55          if (!isset($_GET["ns" . $i])) {
56              $konec = false;
57          } else {
58              $pocty["ns"]++;
59          }
60          if (!isset($_GET["ks" . $i])) {
61              $konec = false;
62          } else {
63              $pocty["ks"]++;
64          }
65          $i++;
66      }
67
68      if ($pocty["ns"] != $pocty["ks"]) {
69          konec(2);
70      }
```

Obrázek č. 44: Programový zápis skriptu v rámci procesu Příjem dat od dodavatelů – část II
Zdroj: vlastní zpracování

Pokud by nebyla definována žádná proměnná *ns* či *ks*, počítadla *$pocty[„ks“]* a *$pocty[„ns“]* by zůstaly na nule, což by nezjistila ani podmínka na řádku 68 na obrázku výše. Z tohoto důvodu stačí doplnit ještě jednu podmínku, která ověří, zda počet iterací cyklu *While* (v proměnné *$i*) nezůstal roven jedné.

```
75      if ($i == 1) {
76          konec(3); //nebyla definovana ns nebo ks
77      }
78
79      for($i = 1; $i <= $pocty["ns"]; $i++) {
80          if (empty($_GET["ns" . $i]) || empty($_GET["ks" . $i])) {
81              konec(4);
82          }
83
84          if ((preg_match('/[^A-ZÀ-ža-z0-9 _-.]/', $_GET["ns" . $i]))) {
85              konec(5);
86          }
87
88          if ((preg_match('/[^A-Za-z0-9]/', $_GET["ks" . $i]))) {
89              konec(5);
90          }
91      }
```

Obrázek č. 45: Programový zápis skriptu v rámci procesu Příjem dat od dodavatelů – část III
Zdroj: vlastní zpracování

Cyklus *for* se zabývá kontrolou hodnot jednotlivých proměnných *ns** a *ks**. První podmínka ověřuje, zda hodnoty nejsou prázdné. Pokud by některá z hodnot byla prázdná, následuje ukončení celého skriptu prostřednictvím funkce *konec*. Druhá podmínka ověřuje pomoci regulárních výrazů, zda se v hodnotách proměnných u názvů souborů nenachází nepřípustné znaky. Povolenými znaky jsou malá a velká písmena včetně diakritiky od a do Z. Dále do množiny povolených znaků náleží čísla v intervalu <0;9>, mezery, podtržítka a spojovníky. U proměnné *ks** je množina povolených znaků podstatně omezenější, protože při použití hashovací funkce SHA1 není zapotřebí, aby tato množina byla stejně velká nebo dokonce větší nežli množina u předchozí proměnné.

Tímto cyklem je dokončena kontrola vstupních dat. Nyní je nezbytné všechna zkontrolovaná vstupní data odeslat do databáze.

Prvním krokem je připojení do databáze SQL. Při realizaci je v této práci používána databáze MySQL od společnosti Oracle.

```
97      $db["ServerName"] = "192.168.35.5";
98      $db["UserName"] = "uzivatel";
99      $db["Password"] = "LZnmJ82s7J8daNG";
100     $db["DBname"] = "nazev_databaze";
101
102     $conn = mysqli_connect($db["ServerName"], $db["UserName"], $db["Password"], $db["DBname"]);
103
104     if (!$conn) {
105         die("Připojení k databázi selhalo: ");
106     }
```

Obrázek č. 46: Programový zápis skriptu v rámci procesu Příjem dat od dodavatelů – část IV (připojení DB)
Zdroj: vlastní zpracování

Nejprve jsou prostřednictvím asociativního pole přiřazeny hodnoty čtyřem prvkům pole s názvem *$db*. Tyto hodnoty obsahují údaje, jež jsou nezbytné pro připojení k databázi. Za touto deklarací se nachází nová proměnná *$conn*, jejíž obsahem je výstup z funkce *mysqli_connect*. Ověření, zda se zdařilo připojit se k databázi provádí podmínka na konci obrázku výše. Pokud se připojení nezdaří, následuje ukončení celého skriptu pomocí funkce *die*, jejíž vstupní hodnotou je zpráva pro uživatele.

```
108     $slozka = $_GET["slozka"];
109     $id_dodavatele = $_GET["id_dodavatele"];
110     $slozka = chr(34) . $slozka . chr(34);
111
112     $sql = 'INSERT INTO aktualizace_dodavatelu (id_dodavatele_CK, nazev_slozky) VALUES (' . $id_dodavatele . ', ' . $slozka . ')';
113
114     if (mysqli_query($conn, $sql)) {
115         echo "<br>New record created successfully 1<br>";
116     } else {
117         mysqli_close($conn);
118         konec(6);
119     }
```

Obrázek č. 47: Programový zápis skriptu v rámci procesu Příjem dat od dodavatelů – část V (1. dotaz)
Zdroj: vlastní zpracování

Na řádku 108 a 109 je použita proměnná *$_GET*. Tato proměnná je implicitně vytvářena samotným jazykem PHP a to tehdy, když odkaz na tento skript obsahuje proměnné a hodnoty přímo za odkazem např. v adresním řádku prohlížeče. Jednotlivé metody získávání hodnot a proměnných jsou uvedeny v teoretických východiscích. Na následujícím řádku je hodnota proměnné *$slozka* doplněna o uvozovky, jak na začátku textového řetězce, tak i na jeho konci. Znak uvozovek je vložen pomocí funkce *chr(34)*, jejíž vstupním argumentem je celé číslo, které pochází z ASCII tabulky. Ordinální hodnota čísla 34 vyjadřuje právě uvozovky. Skript pokračuje deklarací proměnné *$sql*, která obsahuje následující SQL dotaz.

```
$sql = 'INSERT INTO aktualizace_dodavatelu (id_dodavatele_CK,
nazev_slozky) VALUES (' . $id_dodavatele . ', ' . $slozka . ')';
```

Na základě vytvořeného spojení s databází a SQL dotazu lze přistoupit k jeho odeslání do databáze pomocí funkce *mysqli_query*. Výsledek, který vrátila tato funkce, je rovnou vyhodnocen v podmínce a je určeno, zda se podařilo novou *n*-tici do databáze vložit či nikoli. V případě, že vykonání tohoto SQL dotazu skončilo chybou, dochází k ukončení celého skriptu voláním funkce *konec*.

Vzhledem k tomu, že při návrhu databáze byl zvolen umělý automaticky vyplňovaný primární klíč v podobě identifikátoru každé aktualizace, je zapotřebí se databáze znovu dotázat, jaký identifikátor byl přiřazen této nově vzniklé *n*-tici. Toto zpětné zjištění ID je

nezbytné, protože bude následovat přidávání nových *n*-tic do relace *aktualizace_video*, která má vazbu na relaci *aktualizace_dodavatelu*.

```
122     $sql = 'SELECT id_aktualizace_PK FROM aktualizace_dodavatelu WHERE
        id_dodavatele_CK = ' . $id_dodavatele . ' AND nazev_slozky = ' . $slozka . '
        AND id_stav_CK=1 ORDER BY dat_cas DESC LIMIT 1';
123     $result = mysqli_query($conn, $sql);
124
125     if (mysqli_num_rows($result) > 0) {
126         $radek = mysqli_fetch_assoc($result);
127     } else {
128         mysqli_close($conn);
129         konec(6);
130     }
```

Obrázek č. 48: Programový zápis skriptu v rámci procesu Příjem dat od dodavatelů – část VI (2. dotaz)
Zdroj: vlastní zpracování

Na obrázku výše lze vidět další SQL dotaz, který je doplněný o složenou podmínku, která omezuje získávané výsledky. Navíc tento dotaz obsahuje řazení podle atributu *dat_cas*, a to od nejnovějších *n*-tic. Všechny tři podmínky mají zajistit, aby se podařilo získat správné ID. Pro vykonání tohoto dotazu se opět spustí funkce *mysqli_query*, která svými výsledky naplní proměnnou *$result*. Podmínka vyhodnotí, zda je přijatý počet *n*-tic větší než nula. Pokud ano, deklaruje se nová proměnná *$radek*, jejíž datovým typem bude asociované pole, které bude obsahovat všechny dotazem žádané názvy atributů a jejich hodnoty. V tomto konkrétním případě bude proměnná *$radek[„id_aktualizace_PK“]* obsahovat jedno číslo reprezentující identifikátor aktualizace pocházející z databáze.

```
133 for($i = 1; $i <= $pocty["ns"]; $i++) {
134     $ns = chr(34) . $_GET["ns" . $i] . chr(34);
135     $ks = chr(34) . $_GET["ks" . $i] . chr(34);
136
137     $sql = 'INSERT INTO aktualizace_video (id_aktualizace_dodavatelu_CK, nazev_videa,
        kontrolni_soucet) VALUES (' . $radek["id_aktualizace_PK"] . ', ' . $ns . ', ' . $ks . ')';
138
139     if (mysqli_query($conn, $sql)) {
140         echo "<br>New record created successfully<br>";
141     } else {
142         mysqli_close($conn);
143         konec(6);
144     }
145     sleep(0.01);
146 }
147
148 mysqli_close($conn);
149
150 exec("screen -S ID_AKTUALIZACE_kontrolovani -dm php -e proces_pravidelneho_kontrolovani.php
    " . $radek["id_aktualizace_PK"]);
```

Obrázek č. 49: Programový zápis skriptu v rámci procesu Příjem dat od dodavatelů – část VII (3. dotaz)
Zdroj: vlastní zpracování

Vzhledem k tomu, že součástí jedné aktualizace je jeden nebo více souborů, je nutné pro každý tento soubor vytvořit jednu *n*-tici v relaci *aktualizace_video*. Aby bylo možné k jedné aktualizaci přiřazovat *n* videosouborů, použije se získané ID aktualizace, kvůli kterému se databázi zasílal dotaz SELECT výše. Pro vyřešení tohoto problému je použit cyklus *for*. Tento cyklus počítá jednotlivé průchody od čísla 1 až do celkového počtu souborů. V úvodu cyklus přiřazuje hodnoty proměnným *$ns* a *$ks* pocházející přímo ze superglobální proměnné *$_GET* doplněné o uvozovky. Na základě tří proměnných *$ns*, *$ks* a ID poslední aktualizace, je sestaven SQL dotaz. Stejně jako v případech výše, i zde se stejným způsobem vyhodnotí, zda se úspěšně podařilo tento dotaz vykonat. Na konci cyklu se nachází funkce *sleep*. Ta slouží ke zpomalení dotazování se na databázi, aby se předešlo případnému riziku nárazového zahlcení databáze. Je povoleno odesílat jeden dotaz po 10 ms. V závěru tohoto skriptu následuje ukončení spojení s databází MySQL a spuštění Procesu kontroly dokončení kopírování videosouborů od dodavatelů.

Spuštění zmíněného procesu se realizuje prostřednictvím utility zvané *Screen*. Tento *Screen* lze použít pro vytvoření více virtuálních konzolí, kde každá konzole může spouštět své vlastní programy a příkazy. Kdyby hodnota funkce *exec()* v obrázku výše neobsahovala *screen* a rovnou by mělo dojít ke spuštění skriptu *proces_pravidelneho_kontrolovani.php*, nastala by situace, že by se v okamžik volání funkce *exec* čekalo, než dojde k dokončení volaného procesu. Nyní by vznikl problém, protože PHP má ve své konfiguraci nastavenou maximální délku spuštění skriptu. Vzhledem k tomu, že internetové připojení v některých obcích má různorodé charakteristiky, nelze se spoléhat na tento přednastavený čas. Z tohoto důvodu by takové řešení bylo velmi málo flexibilní a je vhodné využít funkci *exec* k vytvoření nové virtuální konzole utilitou *screen*, kterou lze využívat neomezeně dlouhou dobu. Každá tato konzole disponuje svým názvem, který lze specifikovat pomocí argumentu *-S* a dále konzole obsahuje příkaz, určený argumentem *-dm*, který má být proveden. Celá řešená problematika v tomto odstavci je prakticky znázorněna na obrázku výše, na řádku č. 150.

3.8.1 Tvorba řešení pro Proces kontroly dokončení kopírování videosouborů

Tato podkapitola je zaměřena na tvorbu řešení, jež se řídí podle Procesu kontroly dokončení kopírování videosouborů od dodavatelů.

Je-li PHP skript spuštěn přes konzoli s argumentem *-e*, lze takovému skriptu předat vstupní argument.

```
php -e nazev_skriptu.php vstupni_argument_skriptu
```

Pokud je PHP skript spuštěn s argumentem -e, pak je v tomto skriptu definována proměnná *$argv*, které obsahuje, z hlediska datového typu, pole se standardně indexovanými, tj. číselně počínaje nulou, položkami. Níže jsou uvedeny některé hodnoty, kterými toto pole disponuje.

- *$argv[0]* = cesta a název skriptu
- *$argv[1]* = první vstupní argument

```
4     if (!isset($argv[1])) {
5         echo "Konec, protoze nebylo zadano vstupni id aktualizace\n";
6         exit();
7     } else {
8         if (!is_numeric($argv[1])) {
9             echo "Vstupni argument neni cislo\n";
10            exit();
11        }
12        $id_aktualizace = $argv[1];
13    }
14
15    require_once "funkce.php";
```

Obrázek č. 50: Programový zápis skriptu kontrolujícího doručení souborů – část I
Zdroj: vlastní zpracování

Poznatky uvedené výše byly zohledněny při realizaci tohoto řešení. Skript začíná kontrolou, zda je či není definováno pole *$argv* s indexem 1. Není-li definice splněna, pak skript končí chybovou zprávou. V opačném případě je zkontrolováno, zda je vstupní argument číslo. Opět pokud není, skript je ukončen. V opačném případě je vytvořena nová proměnná *$id_aktualizace*, které je přiřazen tento jeden vstupní argument. Pomocí příkazu *require_once* je do jednoho skriptu připojen skript druhý s názvem *funkce.php*, který obsahuje sadu funkcí, jež jsou zapotřebí proto, aby řešení mohlo plně korespondovat s navrženým procesem.

3.8.1.1 Funkce

První funkce *vytvor_pole_cest* má za úkol na základě vstupních proměnných vrátit jedno pole sestávající se z absolutních cest každého souboru. Hodnoty všech vstupních proměnných pocházejí z databáze, což je zobrazeno až za touto podkapitolou.

```php
function vytvor_pole_cest($id_dodav, $slozka, $soubor) {
    $zakladni_cesta = '/mnt/odbavovani/dodavatel-';
    $cesty = [];
    for ($i = 0; $i < count($slozka); $i++) {
        $cesty[$i] = $zakladni_cesta . $id_dodav[$i] . '/' . $slozka[$i] . '/' . $soubor[$i];
    }
    return $cesty;
}
```

Obrázek č. 51: Programový zápis funkce *vytvor_pole_cest*
Zdroj: vlastní zpracování

Druhá funkce *ExistenceSouboru* začíná ověřením, zda existují zadané soubory. Z tohoto důvodu tato funkce má pouze jednu vstupní proměnnou *$cesty*, jejíž hodnotou je pole sestavené z absolutních cest k souborům. Pokud nebude existovat jediná ze zadaných cest, funkce vrátí číslo 1, v opačném případě se postupuje ke kontrole, zda příslušný soubor existuje alespoň dvě hodiny. Z jednotlivých souborů se získává datum a čas poslední změny prostřednictvím funkce *filectime*. V průběhu kopírování se soubor průběžně upravuje, resp. zvětšuje, s čímž souvisí, že se postupně aktualizuje datum poslední změny. Najde-li se jediný soubor, jež není uložen déle než zmíněnou dobu, funkce vrací číslo 2. Návratové hodnoty funkce nejsou voleny náhodně, nýbrž pochází z návrhu procesu Řízení aktualizací. Pokud všechny kontroly dopadnou úspěšně, funkce *ExistenceSouboru* vrací nulu.

```php
function ExistenceSouboru($cesty) {
    $datum_nyni = date("Y-m-d H:i:s");

    for ($i = 0; $i < count($cesty); $i++) {
        if (!(file_exists($cesty[$i]))) {
            return 1;
        }
        $datum = filectime($cesty[$i]);
        $datum = date("Y-m-d H:i:s", $datum);
        $datum = date("Y-m-d H:i:s", strtotime('+2 hours', strtotime($datum)));
        if ($datum > $datum_nyni) {
            return 2;
        }
    }
    return 0;
}
```

Obrázek č. 52: Programový zápis funkce *ExistenceSouboru*
Zdroj: vlastní zpracování

Čtvrtá funkce *KonSoucty* se používá k porovnání kontrolních součtů. Tato funkce má tří vstupní proměnné:

- *$cesty* = absolutní cesty souborů [pole],
- *$ks* = kontrolní součty souborů pocházející z databáze, resp. od dodavatelů [pole],
- *$id_vid* = primární klíč souboru (neboli videa) z databáze [pole].

Vzhledem k tomu, že existuje požadavek, jež stanoví, že toto řešení musí poskytnout možnost rozpoznat poškozené soubory, je deklarováno nové pole *$souhlasi_id_vid*, které se následně naplní identifikátory těch souborů, jejichž SHA1 kontrolní součet bude souhlasit. Dále se spustí cyklus *for*, který postupně prochází a porovná kontrolní součty každého přijatého souboru se součty, jež byly vypočítány u dodavatele. Tato funkce primárně vrací pole, přičemž na nultém indexu se vždy nachází booleovská hodnota. False je využito v případě, že nesouhlasí ani jeden ze součtů. Poškozený soubor je rovnou odstraněn PHP funkcí *unlink*. Souhlasí-li alespoň jeden, je vráceno True společně s polem, jež obsahuje jednotlivé identifikátory souborů, jejichž součet je v pořádku.

```
53  function KonSoucty($cesty, $ks, $id_vid) {
54      $souhlasi_id_vid = [];
55
56      for ($i = 0; $i < count($cesty); $i++) {
57          if (sha1_file($cesty[$i]) == $ks[$i]) {
58              array_push($souhlasi_id_vid, $id_vid[$i]);
59          } else {
60              unlink($cesty[$i]);
61          }
62      }
63
64      if (count($souhlasi_id_vid) != 0) {
65          return array(true, $souhlasi_id_vid);
66      } else {
67          return array(false);
68      }
69  }
```

Obrázek č. 53: Programový zápis funkce *KonSoucty*
Zdroj: vlastní zpracování

Součástí této sady funkcí je rovněž funkce *JePrehratelny*. Úlohou této funkce je detekovat, zda je příslušný videosoubor přehratelný. Tato funkce je vytvořena analogicky po vzoru předchozí funkce. Opět je přítomno pole *$prehratelne_id_vid*, které sdružuje identifikátory těch souborů, které přehrát lze. Pro vyhodnocení, zda je či není daný videosoubor přehratelný, se využívá aplikace *ffprobe*, která je volána prostřednictvím funkce PHP *shell_exec*. Jak vyplynulo z teoretických východisek, z kapitoly 1.8.2, tato aplikace v případě poškozeného videosouboru vrací explicitně označenou chybu: *[error]*.

Argument „*-loglevel level*“ zajišťuje, že se každá informace v rámci odpovědi *ffprobe* zapíše na nový řádek. Této skutečnosti je ve funkci *JePrehratelny* využito, a pomocí PHP funkce *explode* a regulárního výrazu *\n* dojde k rozdělení odpovědi z *ffprobe* do jednotlivých prvků standardně indexovaného pole s názvem *$t*. Dále následuje čtení všech řádků odpovědi *ffprobe* a vyhledávání textového řetězce *[error]*. Struktura návratových hodnot a datových typů je opět analogická s předchozí funkcí. Pokud je zjištěno, že některý soubor přehratelný není, je bezprostředně odstraněn.

```php
function JePrehratelny($cesty, $id_vid) {
    $prehratelne_id_vid = [];
    for($j = 0; $j < count($cesty); $j++) {
        $t = shell_exec("ffprobe -hide_banner -loglevel level " . $cesty[$j] . " 2>&1");
        $t = explode("\n", $t);
        for($i = 0; $i < count($t); $i++) {
            if (strpos($t[$i], "[error]") > 1) {
                continue;
            } else {
                unlink($cesty[$i]);
            }
            array_push($prehratelne_id_vid, $id_vid[$i]);
        }
    }

    if (count($prehratelne_id_vid) != 0) {
        return array(true, $prehratelne_id_vid);
    } else {
        return array(false);
    }
}
```

Obrázek č. 54: Programový zápis funkce *JePrehratelny*
Zdroj: vlastní zpracování

Poslední dvě procedury zjednodušují konstrukci SQL dotazů a rovnou odesílají data do databáze.

```php
function UpravStavDB($conn, $id_aktualizace, $stav) {
    $sql = 'UPDATE aktualizace_dodavatelu SET id_stav_CK = ' . $stav . ' WHERE id_aktualizace_PK = ' . $id_aktualizace;

    if (!mysqli_query($conn, $sql)) {
        die("UpravStavDB - chyba při vykonání dotazu SQL: " . mysqli_error($conn));
    }
}

function UpravUdajDB($conn, $id_videa, $atribut, $hodnota) {
    $sql = 'UPDATE aktualizace_video SET ' . $atribut . ' = ' . $hodnota . ' WHERE id_videa_PK = ' . $id_videa;

    if (!mysqli_query($conn, $sql)) {
        die("UpravUdajDB - chyba při vykonání dotazu SQL: " . mysqli_error($conn));
    }
}
```

Obrázek č. 55: Programový zápis procedur *UpravStavDB* a *UpravUdajDB*
Zdroj: vlastní zpracování

3.8.1.2 Pokračování v tvorbě řešení

Přesně dle navrženého procesu se i nadále postupuje v realizaci tohoto řešení. Dalším krokem je získání dat z databáze, za účelem sestavení absolutních cest ke všem

videosouborům. Nejprve je vytvořeno připojení do databáze a poté se definuje SQL dotaz. Ten se následně odešle a jsou-li přijata nějaká data, pak se tato získaná data uloží do asociovaného pole *$data*. V opačném případě dojde k zobrazení chybové zprávy, odpojení od databáze a následuje ukončení skriptu funkcí *exit()*.

```
18      require_once "db.php";
19
20      $sql = 'SELECT id_dodavatele_CK, nazev_slozky, nazev_videa, kontrolni_soucet, id_videa_PK FROM
        pohled_aktualizace WHERE id_aktualizace_PK = ' . $id_aktualizace;
21      $result = mysqli_query($conn, $sql);
22      $data = [];
23      $data["id_dodavatele_CK"] = [];
24      $data["nazev_slozky"] = [];
25      $data["nazev_videa"] = [];
26      $data["kontrolni_soucet"] = [];
27      $data["id_videa_PK"] = [];
28
29      if (mysqli_num_rows($result) > 0) {
30          while ($row = mysqli_fetch_assoc($result)) {
31              array_push($data["id_dodavatele_CK"], $row["id_dodavatele_CK"]);
32              array_push($data["nazev_slozky"], $row["nazev_slozky"]);
33              array_push($data["nazev_videa"], $row["nazev_videa"]);
34              array_push($data["kontrolni_soucet"], $row["kontrolni_soucet"]);
35              array_push($data["id_videa_PK"], $row["id_videa_PK"]);
36          }
37      } else {
38          echo "\nŽádná data nenalezena.\n";
39          mysqli_close($conn);
40          exit();
41      }
```

Obrázek č. 56: Programový zápis skriptu kontrolujícího doručení souborů – část II
Zdroj: vlastní zpracování

Za pomocí cyklu *while* se bude každých 15 minut monitorovat, zda již existují v lokálním adresáři všechny příslušné videosoubory a zda existují alespoň dvě hodiny, což znamená, že čas poslední změny je o dvě hodiny menší nežli současný čas. V tomto cyklu se opakovaně volá funkce *ExistenceSouboru*. Pokud tato funkce vrátí nulu, dojde k ukončení cyklu *while* a pokračuje se dále. V opačném případě je návratová hodnota funkce odeslána do databáze.

```
46      $SoubExist = false;
47      $cesty = vytvor_pole_cest($data["id_dodavatele_CK"], $data["nazev_slozky"], $data["nazev_videa"]);
48
49      while ($SoubExist == false) {
50          sleep(15*60);
51
52          $ExistenceSouboru = ExistenceSouboru($cesty);
53          if ($ExistenceSouboru != 0) {
54              UpravStavDB($conn, $id_aktualizace, $ExistenceSouboru);
55          } else {
56              $SoubExist = true;
57          }
58      }
```

Obrázek č. 57: Programový zápis skriptu kontrolujícího doručení souborů – část III
Zdroj: vlastní zpracování

Předposledním krokem tvorby této částí řešení je ověření kontrolních součtů. Nejprve se volá funkce *KonSoucty*, jejíž výsledek se ukládá do stejnojmenné proměnné *$KonSoucty*. Pokud funkce vrátila False, znamená to, že ani jeden kontrolní součet není v pořádku. V opačném případě se postupně projdou všechny soubory cyklem *foreach* a v databázi v relaci *aktualizace_video* se upraví údaje těch souborů, které mají kontrolní součet v pořádku, protože se ve výchozím nastavení datového modelu předpokládá, že ani jeden součet nesouhlasí. Podmínka za cyklem upravuje stav celé aktualizace s ID 3, což vyjadřuje právě onen zmíněný chybný součet některého ze souborů.

```
60      $KonSoucty = KonSoucty($cesty, $data["kontrolni_soucet"], $data["id_videa_PK"]);
61
62      if ($KonSoucty[0] == true) {
63          foreach ($KonSoucty[1] as &$id) {
64              UpravUdajDB($conn, $id, 'kontrolni_soucet_je_spravne', 1);
65          }
66          if (count($data["id_videa_PK"]) != count($KonSoucty[1])) {
67              UpravStavDB($conn, $id_aktualizace, 3);
68              echo "Některý z kontrolních součtů nesouhlasí";
69              exit();
70          }
71
72      } else {
73          UpravStavDB($conn, $id_aktualizace, 3);
74          echo "ani jeden kontrolní součet není v pořádku";
75          exit();
76      }
```

Obrázek č. 58: Programový zápis skriptu kontrolujícího doručení souborů – část IV
Zdroj: vlastní zpracování

Nakonec se realizuje kontrola, zda jsou či nejsou videosoubory přehratelné. Tato kontrola je principiálně identická s předchozí kontrolou.

```
80      $prehratelnost = JePrehratelny($cesty, $data["id_videa_PK"]);
81      if ($prehratelnost[0] == true) {
82          foreach ($prehratelnost[1] as &$id) {
83              UpravUdajDB($conn, $id, 'je_prehratelne', 1);
84          }
85
86          if (count($data["id_videa_PK"]) != count($prehratelnost[1])) {
87              UpravStavDB($conn, $id_aktualizace, 4);
88              echo "některý z videosouborů není přehratelný";
89              exit();
90          }
91
92      } else {
93          UpravStavDB($conn, $id_aktualizace, 4);
94          echo "každé video je poškozené";
95          exit();
96      }
97
98  UpravStavDB($conn, $id_aktualizace, 5);
99
100 mysqli_close($conn);
```

Obrázek č. 59: Programový zápis skriptu kontrolujícího doručení souborů – část V
Zdroj: vlastní zpracování

3.9 Zjištění aktuálního stavu odbavování

Pro zjištění, zda je odbavování v provozu, se využije PHP funkce *shell_exec*, přičemž spouštěný příkaz je uveden na obrázku níže.

```php
function bezi_ffmpeg() {
    $bezi = shell_exec("pgrep -l ffmpeg");

    if (strlen($bezi) == 0) {
        return false;
    } else {
        return true;
    }
}
```

Obrázek č. 60: Zjištění aktuálního stavu odbavování
Zdroj: vlastní zpracování

Je-li *ffmpeg* spuštěn, pak je vrácen následující textový řetězec.

```
ka@server:~$ pgrep -l ffmpeg
4180 ffmpeg
```

Obrázek č. 61: Příkaz ke zjištění spuštění ffmpeg
Zdroj: vlastní zpracování

Číslo vyjadřuje PID procesu a za číslem se nachází název běžícího procesu. Pokud *ffmpeg* neběží, není vrácen žádný textový řetězec. Z tohoto důvodu je validní zápis výše uvedené funkce v PHP.

Obrázek č. 62: Vyobrazení stavu odbavování ve webovém rozhraní
Zdroj: vlastní zpracování

Výše je navržena webová stránka, která interpretuje výsledky získané z funkce *bezi_ffmpeg*. Jsou tedy detekovány pouze 2 stavy, a to odbavuje se (*ffmpeg* je spuštěn)

a neodbavuje se (*ffmpeg* není spuštěn). Všechny ostatní možné stavy jsou vyloučeny díky důkladnému řešení při přenosu souborů.

3.10 Ekonomické zhodnocení

Z hlediska nákladů na softwarové řešení se jedná zejména o náklady za čas potřebný k návrhu a také na vývoj zahrnující testovací provoz včetně oprav případně zjištěných chyb. Tento návrh již zahrnuje veškeré okolnosti, jež byly pro vývoj potřebné. Rovněž se musí počítat i s rizikem, že se během testování vyskytne nepředpokládaná chyba.

Tabulka č. 10: Odhad nákladů softwarového řešení
Zdroj: vlastní zpracování

Náklad	Pracnost (MD)	Cena (Kč)
Návrh řešení	28	182 000,-
Tvorba řešení	19	123 000,-
Testování	4	26 000,-
Oprava chyb	8	52 000,-
Vytvoření dokumentace	5	32 000,-

Určí-li se částka na 1 MD 6 500 Kč, pak celkové náklady na zhotovení tohoto kompletního softwarového řešení činí 415 000 Kč. Tato částka nezahrnuje náklady na implementaci. Firma využívá svobodného softwaru, a proto firma ušetří na nákladech za licence. Celkové náklady budou hrazeny firmou formou faktury.

3.11 Přínosy navrhovaného řešení

V diplomní práci předložené softwarové řešení pomůže společnosti KME, s.r.o. výrazně snížit režie, které souvisí se zajištěním provozu odbavování televizního vysílání. Firma získá možnost provádět hromadné distribuce aktualizací automatizovaně, takže po implementaci tohoto řešení již nebude nutné, aby zaměstnanec firmy musel autem pravidelně objíždět jednotlivá místa odbavovacích stanic a manuálně, třeba z flash disku, kopírovat nové videosoubory. Díky zmíněné hromadné distribuci stačí, aby dodavatel videosouborů provedl jediné odeslání těchto dat na server a další šíření aktualizací se realizuje s podporou vyvinutého software automaticky. Současně toto softwarové řešení poskytuje zpětnou vazbu o stavu přenosu videosouborů a je možné zjistit aktuální stav

odbavování na jednotlivých stanicích. Díky využití systému logování lze snadno nahlížet do historie odbavování, a tak získat přehled, kdy a kým byla provedena hledaná změna.

ZÁVĚR

Cílem této diplomové práce bylo navrhnout a vytvořit softwarové řešení pro správu a řízení odbavování televizního vysílání ve firemním prostředí. Toto nové řešení spojuje prvky existujících řešení s požadavky plynoucími z analýzy současného stavu.

První část práce pojednává o teoretických východiscích, na jejichž základě byl realizován návrh a tvorba řešení. Nejprve jsem v této kapitole rozvedl pojem databáze a uvedl některé druhy datových modelů. Podrobněji jsem se zabýval relačním databázovým modelem, jehož princip je použitý v rámci softwarové realizace. Dále jsou uvedeny pojmy týkající se databázových systémů, zejména SQL, vytváření pohledů, uložené procedury a kurzory. V první kapitole je také pojednána problematika počítačových sítí a s ní spjaté modely, síťové architektury a některé přenosové protokoly. Oblast počítačových sítí je doplněna o přenosové metody typu unicast, multicast a broadcast. Teoretická východiska uzavírá oblast programovacích jazyků a aplikace ffmpeg.

V následující kapitole jsem se věnoval současným možnostem odbavování televizního vysílání. Tato kapitola je doplněna o nynější existující alternativy, které jsou vzájemně porovnány a vyhodnoceny. Výstupem toho bylo formulování nových požadavků, kterými disponuje toto nové softwarové řešení.

Poslední kapitola obsahuje algoritmy procesu aktualizace videosouborů a procesu přehrávání podle požadavků společnosti KME, s.r.o. Majoritní část návrhu je podložena procesy, které jsou náležitě identifikovány, popsány a také graficky znázorněny pomocí vývojových diagramů. Praktická realizace softwaru podle návrhu obsahuje programové fragmenty kódu s popisem jejich funkčnosti.

V závěru kapitoly uvádím odhad ekonomického zhodnocení a přínosy tohoto nového řešení.

Popisované programové prostředky byly vyvíjeny podle zadání diplomní práce věnované návrhu a tvorbě softwarového řešení pro televizní vysílání s využitím technických i programových prostředků odpovídající současné úrovni poznání. V konečné verzi je uváděné programové řešení ve shodě se zadáním diplomní práce a také vyhovuje požadavkům popisovaného firemního prostředí. Z uvedeného vyplývá, že bylo dosaženo vytyčených cílů.

SEZNAM POUŽITÝCH ZDROJŮ

(1) *RFC 2616* [online]. RFC [cit. 2023-01-11]. Dostupné z: https://datatracker.ietf.org/doc/html/rfc2616

(2) *RFC 2974*. [online]. RFC [cit. 2023-01-11]. Dostupné z: https://datatracker.ietf.org/doc/html/rfc2974

(3) KOCH, Miloš a Vysoké učení technické v Brně. *Datové a funkční modelování*. Brno: Akademické nakladatelství CERM, 2006, s. [1a]. ISBN 80-214-3252-7.

(4) CONOLLY, Thomas, Carolyn E. BEGG a Richard HOLOWCZAK. *Mistrovství - databáze: profesionální průvodce tvorbou efektivních databází*. Brno: Computer Press, 2009, s. [1a]. ISBN 978-80-251-2328-7.

(5) *MySQL :: About MySQL* [online]. Oracle, 2021 [cit. 2021-5-10]. Dostupné z: https://www.mysql.com/about/

(6) WELLING, Luke a Laura THOMSON. *Mistrovství PHP a MySQL*. Brno: Computer Press, 2017. ISBN 978-80-251-4892-1.

(7) *SQL Databases* [online]. Microsoft [cit. 2023-01-20]. Dostupné z: https://learn.microsoft.com/en-us/sql/relational-databases/views/views

(8) *SQL Procedure* [online]. W3Resource [cit. 2023-01-22]. Dostupné z: https://www.w3resource.com/sql/sql-procedure.php

(9) *Oracle Database PL/SQL* [online]. [cit. 2023-02-13]. Dostupné z: https://www.oracle.com/cz/database/technologies/appdev/plsql.html

(10) LUHAN, J. *Databázové systémy* [přednáška]. Brno: VUT, 21. 11. 2019.

(11) HORÁK, Jaroslav a Milan KERŠLÁGER. *Počítačové sítě pro začínající správce*. 5., aktualiz. a rozš. vyd. Brno: Computer Press, 2013, 303 s. ISBN 978-80-251-3176-3.

(12) JORDÁN, Vilém a Viktor ONDRÁK. *Infrastruktura komunikačních systémů I: univerzální kabelážní systémy*. Druhé, rozšířené vydání. Brno: CERM, akademické nakladatelství, 2015. ISBN 978-80-214-5115-5.

(13) *KME s.r.o.* [online]. [cit. 2023-03-03]. Dostupné z: https://www.kme.cz/kme/kontakty/

(14) *Stream Circle - TV automation & playout platform* [online]. [cit. 2023-03-12]. Dostupné z: https://streamcircle.com/

(15) *PROVYS TV Office byl úspěšně otestován na architektuře Sun - Lupa.cz* [online]. Praha: Internet Info, 2007 [cit. 2022-3-10]. Dostupné z: https://www.lupa.cz/tiskove-zpravy/provys-tv-office-byl-uspesne-otestovan/

(16) *Provys – Integrated software solution for broadcasting organisations* [online]. DCIT, 2020 [cit. 2022-2-10]. Dostupné z: https://provys.com/

(17) JIROVSKÝ, Václav. *Vademecum správce sítě*. Praha: Grada, 2001. ISBN 80-716-9745-1.

(18) *Python documentation* [online]. [cit. 2023-03-15]. Dostupné z: https://docs.python.org/3/tutorial/index.html

(19) f*fmpeg Documentation* [online]. [cit. 2023-03-23]. Dostupné z: https://ffmpeg.org/ffmpeg.html

(20) f*fplay Documentation* [online]. [cit. 2023-03-23]. Dostupné z: https://www.ffmpeg.org/ffplay.html

(21) TRETEROVÁ, Eliška. *Návrh a vývoj algoritmů: modul-vývojové diagramy a příkazy jazyka Borland Pascal.* Ostrava: Ostravská univerzita, 2003. Systém celoživotního vzdělávání Moravskoslezská. ISBN 80-7042-854-6.

(22) *Co je to databáze* [online]. Oracle, 2014 [cit. 2021-3-1]. Dostupné z: https://www.oracle.com/cz/database/what-is-database/

(23) *Scheduling for Stream Circle in PROVYS* [online]. [cit. 2023-03-26]. Dostupné z: https://www.youtube.com/watch?v=qiCnsblkfbE

(24) BIN UZAYR, Sufyan, ed. *Mastering MySQL for the web: a beginner's guide*. Boca Raton: CRC Press, 2022. Mastering computer science. ISBN 978-1-032-13512-0.

(25) KEMPER, Alfons, EICKLER, André. *Datenbanksysteme: Eine Einführung*. München: Oldenbourg Verlag, 2011. ISBN 978-3-486-59082-3.

(26) DELOBEL, Claude; JOUANOT, Fabrice; MANDRAN, Nadine; PETIT, Jean-Marc; ROUSSET, Marie-Christine. *Bases de données : concepts, utilisation et développement*. Paris: Dunod, 2015. ISBN 978-2100-716175.

(27) CATV service in FTTH - Huawei Enterprise Support Community DVB-C. In: *Huawei Technologies Co., Ltd.* [online]. [cit. 2023-03-12]. Dostupné z: https://forum.huawei.com/enterprise/en/data/attachment/forum/202210/14/193506sz9vvul65z5hhfk2.jpg

SEZNAM POUŽITÝCH ZKRATEK A SYMBOLŮ

AAC	Advanced Audio Coding
CK	cizí klíč (Foreign key)
DBMS	systémem pro správu databáze (Database Management System)
DVB-C	Digital Video Broadcasting – Cable (Digitální televizní kabelové vysílání. Standard pro přenos digitálního televizního signálu)
ER diagram	entito-relační diagram (Entity-relationship Diagram)
HTML	hypertextový značkovací jazyk (Hyper Text Markup Language)
HTTP	Hypertext Transfer Protocol (komunikační protokol používaný pro přenos hypertextových dokumentů a dat)
ID	identifikátor (unikátní hodnota, která slouží k jednoznačné identifikaci záznamu v databázi)
IPTV	Internet Protocol Television
LXDE	Lightweight X11 Desktop Environment (desktopové prostředí pro systémy Unix)
MD	pracnost (člověkoden Man-Day nebo jen Manday)
MS	Microsoft (společnost)
n-tice	uspořádaný seznam konečného počtu n objektů (konkrétní varianty nazývané uspořádané dvojice, uspořádané trojice atd.)
OS	operační systém
PHP	Hypertext Preprocessor (skriptovací programovací jazyk)
PK	primární klíč (Primary Key)
RRTV	Rada pro rozhlasové a televizní vysílání
SAP	Session Announcement Protocol
SHA1	kryptografická hashovací funkce (Secure Hash Algorithm 1)
SMB	Server Message Block (síťový komunikační protokol)
SQL	jazyk pro manipulaci s daty (Structured Query Language)
TCP	Transmission Control Protocol (transportní protokol používaný)
UDP	User Datagram Protocol (transportní protokol používaný v počítačových sítích pro přenos dat)
VPN	Virtual Private Network (Virtuální privátní síť)

SEZNAM POUŽITÝCH OBRÁZKŮ

SEZNAM POUŽITÝCH TABULEK

SEZNAM POUŽITÝCH DIAGRAMŮ