VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

ZPŘÍSTUPNĚNÍ OBRAZU Z IP KAMER V PROHLÍŽEČI

WEB BROWSER FOR IP CAMERAS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **PAVEL ČERNÝ**
AUTHOR

VEDOUCÍ PRÁCE **doc. Ing. FRANTIŠEK ZBOŘIL, Ph.D.**
SUPERVISOR

BRNO 2020

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav inteligentních systémů (UITS) Akademický rok 2019/2020

Zadání bakalářské práce

Student: **Černý Pavel**
Program: Informační technologie
Název: **Zpřístupnění obrazu z IP kamer v prohlížeči**
Web Browser for IP Cameras
Kategorie: Umělá inteligence

Zadání:

1. Prozkoumejte způsob přenosu obrazu z IP kamer a možnosti přenosu a zobrazení obrazu (videa) do webových prohlížečů. Věnujte pozornost možnostem HTML5 (např. použití tagu <video>). Seznamte se s problematikou používaných kodeků.
2. Na základě průzkumu navrhněte systém, který bude umožňovat především zobrazit živý obraz z kamer, včetně jednoduchého administračního rozhraní (správa kamer a přístupových práv uživatelů).
3. Systém implementujte pomocí vhodných programovacích jazyků. Použijte volně dostupné knihovny a programy (licencované pod GNU GPL, Apache a podobně). Webové rozhraní nechť je funkční v aktuálních verzích nejrozšířenějších prohlížečů (Firefox, Chrome, MS Edge).
4. Zhodnoťte dosažený výsledek a navrhněte možnosti dalšího vývoje.

Literatura:

- Russel, S., Norvig, P.: Artificial Intelligence, A Modern Approach, Pearson, 2009

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Zbořil František, doc. Ing., Ph.D.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1. listopadu 2019
Datum odevzdání: 14. května 2020
Datum schválení: 31. října 2019

Abstrakt

Cílem práce je zpřístupnit živý obraz z IP kamer vybraným uživatelům bez nutnosti instalace speciálního klientského software – pouze za použití webového prohlížeče. Řešení bylo dosaženo použitím programů FFmpeg (zajišťuje komunikaci s kamerou a konverzi videa do formátu WebM), Stream-m (přijímá data od FFmpeg a odesílá klientům) a vytvořením webové aplikace postavené na frameworku Spring, která zmíněné programy ovládá a zároveň umožňuje správu údajů o kamerách a autorizaci uživatelů. Výsledný systém přináší snížení úsilí nutného pro zprovoznění a údržbu klientských stanic. Díky zmenšení velikosti obrazu při konverzi videa systém značně redukuje datový tok v síti způsobený sledováním kamer a minimalizuje zátěž na stroje klientů.

Abstract

The aim of the work described in this thesis was to enable live video streaming from IP cameras to selected users, using simply a modern web browser with no need to install special client-side software. The solution was found by utilizing programs FFmpeg (for communication with camera and video conversion to WebM format), Stream-m (for distribution of the data received from FFmpeg to clients), and a newly-created web application built on the Spring framework, which was used to control the aforementioned programs, and also allows camera management and user authorization. The resulting system reduces the effort required for client nodes deployment and maintenance. Thanks to the image size downscaling, the system brings about a considerable reduction of data flow in the network caused by camera output, and minimises the load on client computers.

Klíčová slova

IP kamera, CCTV, kamerové systémy, framework Spring, HTML5 video, WebM, WebP, FFmpeg, Stream-m, Facility Management, JPA

Keywords

IP camera, CCTV, Spring framework, HTML5 video, WebM, WebP, FFmpeg, Stream-m, Facility Management, JPA

Citace

ČERNÝ, Pavel. *Zpřístupnění obrazu z IP kamer v prohlížeči.* Brno, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce doc. Ing. František Zbořil, Ph.D.

Zpřístupnění obrazu z IP kamer v prohlížeči

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana doc. Ing. Zbořila, PhD. z Ústavu inteligentních systémů FIT VUT. Další informace mi poskytl pan RNDr. Adam Kučera, PhD. z OFM SUKB Masarykovy Univerzity. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Pavel Černý
11. června 2020

Poděkování

Chtěl bych poděkovat panu docentu Františku Zbořilovi mladšímu za ochotu ujmout se vedení mé práce. Velké díky patří především panu Adamu Kučerovi z Masarykovy Univerzity za poskytnuté informace a cenné připomínky zvláště v době dokončování práce.

Obsah

Kapitola 1

Úvod

Zvláště v prostředí větších firem dnes najdeme systémy bezpečnostních kamer, známé také jako systémy uzavřeného televizního okruhu (CCTV[1]). Používají se především na místech se zvýšeným nebezpečím, ať už jde o hrozbu kriminálního (například krádež) nebo technického (například pracoviště s výskytem nebezpečných látek) charakteru.

Dříve se používaly analogové kamerové systémy. Obraz ze skupiny kamer se přenášel ve formě analogového signálu do záznamového zařízení a z něj do zobrazovacího zařízení. Dnes se již ve větší míře používají digitální IP kamerové systémy, které přinesly řadu výhod. Protože jsou IP kamery připojeny prostřednictvím počítačové sítě, zaniká technické omezení na jedno dohledové stanoviště (i když se stále jedná o výchozí a důležitou součást systému CCTV) a je teoreticky možné obraz sledovat na libovolných stanicích. To je výhodné zvláště v prostředí institucí, které využívají kamerový systém nejen k monitorování bezpečnostních incidentů, ale i specializovaných pracovišť (laboratoří). V takovém případě může být vhodné umožnit sledování konkrétní kamery nebo skupiny kamer vedoucímu nebo jiným pracovníkům laboratoře.

V praxi je ale nutné řešit několik problémů. Sledování IP kamer obvykle není možné bez použití speciálního programového vybavení. Ale i v případě, kdy kamery používají běžné komunikační protokoly a lze je sledovat v běžném přehrávači videa (např. VLC), není toto řešení příliš vhodné například z důvodu přenášení zbytečně velkého datového proudu nebo expozice přístupových údajů ke kameře uživatelům a jejich neefektivní aktualizace při změně konfigurace. Sledování kamer také může komplikovat samotná síťová topologie. Kamery (a jiné technologie instalované v budovách) se totiž z bezpečnostních či jiných důvodů často nacházejí ve zvláštní síti chráněné síťovými bezpečnostními prvky, případně fyzicky zcela oddělené od běžné datové počítačové sítě.

S vyjmenovanými problémy se potýkal také můj zaměstnavatel. Pracovníci laboratoří měli zájem o přístup k obrazu kamer, ale jeho zprostředkování bylo mnohdy problematické. Protože se mi problém zdál zajímavý, vybral jsem si ho jako téma bakalářské práce.

Cílem práce je zmíněné problémy eliminovat a zpřístupnit živý obraz z IP kamer autorizovaným uživatelům prostřednictvím webového prohlížeče, bez nutnosti instalace speciálního software.

Kapitola 2 představuje historický kontext bezpečnostních kamer a videa na webu. V kapitole 3 jsou uvedeny požadavky na systém, rešerše aktuálního stavu a shrnutí problémů existujících řešení. Kapitola 4, rozebírá jednotlivé problémy, které je třeba vyřešit na cestě od kamery k uživateli. Následuje kapitola 5 odhalující zajímavé detaily technické realizace.

[1] CCTV – Closed Circuit Television (uzavřený televizní okruh)

Kapitola 2

Historie a současná situace

V této kapitole se budu zabývat historií bezpečnostních kamer, především z hlediska jejich technologického vývoje. S uvedením do historického kontextu nastíním výhody současné situace i potenciální výzvy. Jelikož předmětem práce je zobrazení obrazu ve webovém prohlížeči (podrobněji popsáno v kapitole 3), stručně zde popíši i vývoj zobrazování videí na webu.

2.1 Historie a současnost bezpečnostních kamer

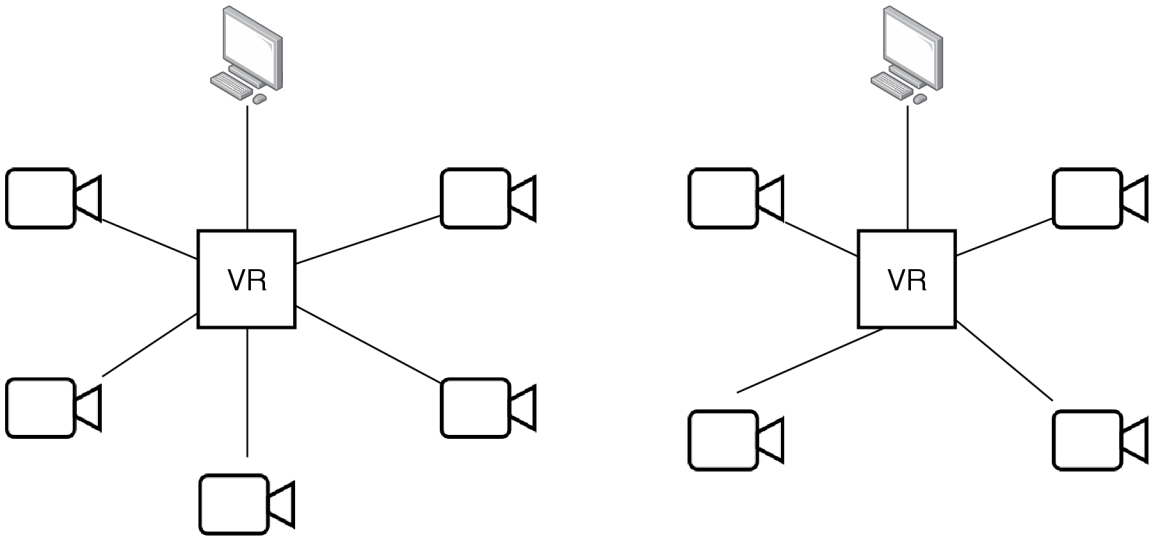
Fyzické rozhraní pro připojení a přenos dat do systému u analogových kamer tvořil zpravidla konektor BNC[1], v menší míře (a spíše u zařízení nižších tříd) konektor RCA[2], také známý pod názvem CINCH. Jeho prostřednictvím byla každá kamera napojena na koaxiální kabel vedoucí do záznamového zařízení. Systémy uspořádané tímto způsobem tvořily tzv. hvězdicovitou topologii. Ukázka takové topologie je znázorněna na obrázku 2.1.

Z počátku byl obraz zaznamenáván na magnetickou pásku a video signál ze záznamového zařízení do zobrazovacího zařízení veden rovněž analogovou cestou. Později se na trh dostaly digitální záznamová zařízení (DVR[3]). Signál z kamery byl stále veden analogově, ale v DVR byl kompozitní analogový signál prostřednictvím konverzní karty převeden na digitální informaci, již bylo možné dále zpracovávat. Záznamová zařízení se tak stala video servery, schopnými provádět analýzu obrazu a doručovat obraz přes počítačovou síť. Jak ilustruje obrázek 2.2, ačkoli je základ topologie shodný s čistě analogovými systémy, jsou již DVR připojeny k počítačové síti.

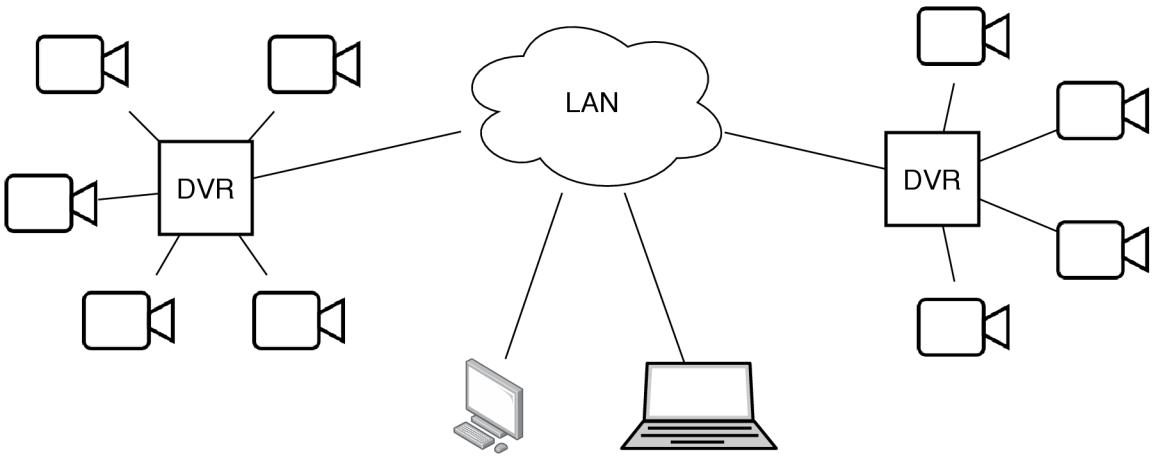
Jednou z nevýhod analogových systémů byla například nutnost vedení celé kabelové trasy od každé kamery do záznamového zařízení. Z důvodu fyzikálních limitů byla omezená maximální délka koaxiálního kabelu na cca 100 metrů, potom docházelo k nadměrnému tlumení a zhoršení kvality obrazu. Ze stejného důvodu byla omezena také maximální velikost obrazu, kterou bylo možné v uspokojivé kvalitě přenést. Kvůli snaze o zkracování kabelového vedení a omezenému počtu analogových vstupů docházelo k nadměrnému navyšování počtu DVR v systému. Přenášený analogový obraz nebyl zabezpečen – pokud útočník získal fyzický přístup ke kameře nebo k vedení obrazového signálu, mohl obraz z kamery sledovat.

Přes vyjmenované nevýhody se analogové systémy i dnes stále používají, zejména v prostředí menších lokalit, kde jejich výhody převažují nad nevýhodami. Příkladem takových

[1] BNC – typ konektoru standardně používaný na připojení koaxiálními kabely
[2] RCA – typ konektoru pro přenos audio či video signálu, používaný zejména v domácnostech
[3] DVR – Digital Video Recorder (digitální záznamník videa)



Obrázek 2.1: Příklad topologie analogového kamerového systému. Prvek VR představuje analogové záznamové zařízení s omezeným počtem vstupů. Při větším počtu kamer je třeba přidat celý samostatný okruh.



Obrázek 2.2: Ukázka topologie analogového systému s DVR. Obraz z kamer je stále přenášen analogovým signálem (se všemi důsledky), digitální záznamové zařízení převádí tento signál na digitální informaci. Zařízení DVR můžou být připojena do počítačové sítě (LAN) – potom lze obraz sledovat na různých počítačích v síti.

výhod je například snadnější zprovoznění a údržba systému (veškerá nastavení se provádí na jednom zařízení, v DVR), nezatížení počítačové sítě nebo menší nároky na kapacitu úložiště záznamu, způsobené menší velikostí a kvalitou obrazu.

Oproti analogovým systémům přináší IP kamery řadu výhod. IP kamery je možné připojit k běžné počítačové lokální síti (LAN) pomocí běžných síťových konektorů (RJ-45) a kabelů (UTP). Individuální kabelové trasy jsou výrazně kratší, protože kabel stačí vést k nejbližšímu síťovému prvku. Většinou není nutné, aby IP kamera byla v místě instalace přímo napojena na elektrickou síť – díky technologii PoE[4] mohou být napájeny týmž kabelem, kterým se přenášejí i data. Pokud daný síťový přepínač (switch) tuto technologii nepodporuje, lze zapojení realizovat přes PoE injektor (zařízení, které je zapojeno mezi přepínač a IP kameru za účelem doplnění funkcionality PoE). Ten může být umístěn kdekoli na cestě mezi přepínačem a kamerou, například i ve stejné skříni spolu s přepínačem. Již v kameře je zabudován video server schopný obsloužit několik klientů. Záznamová zařízení typu DVR byly nahrazeny síťovými rekordéry (NVR[5]). Ty již nejsou centrálními body systému, spíše se podobají roli klienta odebírajícího stream (přenos). Ukázka takové topologie je znázorněna na obrázku 2.3.

Dnešní IP kamery disponují plynulejším obrazem a také vyšším rozlišením, díky čemuž je v některých případech možné použití menšího počtu kamer, než bylo nutné u analogových systémů. Pokročilé kamery poskytující analýzu obrazu uleví výkonu kamerových serverů, kde musely být analytické funkce implementovány před nástupem IP kamer. Bezpečnost pak zvyšují možnosti autentizace a šifrování dat. Útočníkovi již k úspěšnému odchytu obrazu nestačí jen získat připojení k místní síti, musí znát také IP adresu kamery, kterou chce sledovat, a autentizační údaje. Mezi další výhody patří jednoduchost, flexibilita a škálovatelnost nebo možnost záznam zároveň sledovat i nahrávat. [24]

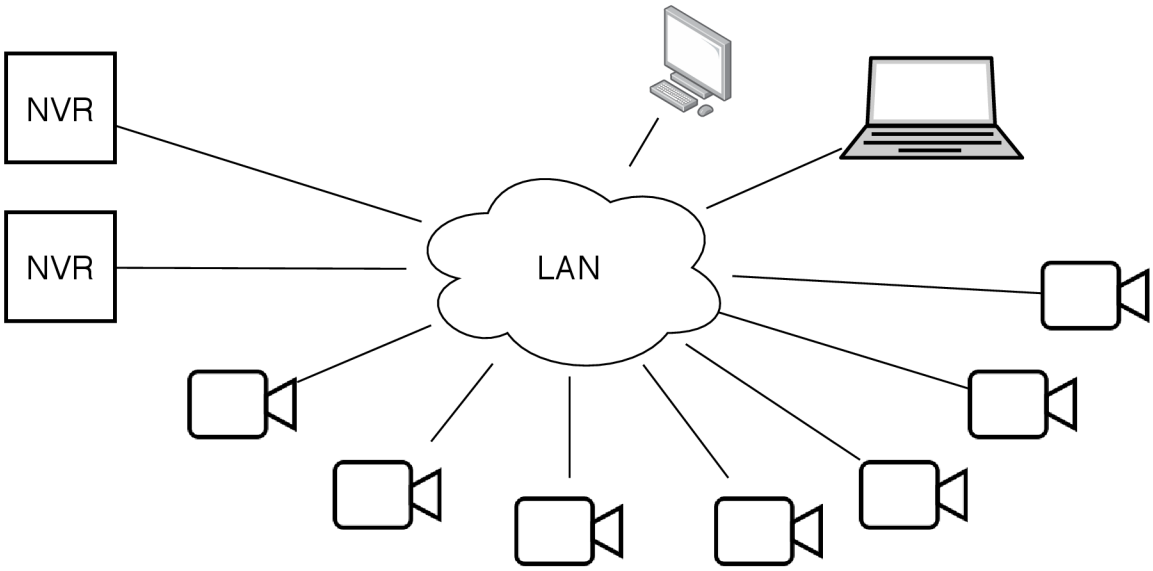
Ač přinesly digitální (IP) kamerové systémy nové možnosti a odstranily některá technická omezení, objevily se nové výzvy, které je třeba řešit. Jednou z nich jsou problémy spojené se správou klientských stanic.

2.2 Vývoj videa na webu

Situace před vznikem standardu HTML5[6] pro média na webu nebyla příznivá. Jejich přehrávání nebylo implementováno v prohlížečích; autoři webových stránek museli využít nějaký externí přehrávač v podobě zásuvného modulu pro webový prohlížeč, komponent ActiveX, Java appletu či programu typu Flash. Autoři webových stránek k umístění videa používali HTML značek `<APPLET>`, `<EMBED>`, `<OBJECT>`[17] či jejich kombinace za účelem zajištění větší míry kompatibility napříč internetovými prohlížeči. [13] Někteří tvůrci dokonce do HTML dokumentu umístili pouze odkaz, který vedl ke stažení souboru nebo spuštění přehrávače instalovaného v systému. Je zřejmé, že spuštění takového obsahu nebylo vždy jednoduché a mnohdy vedlo k frustraci uživatelů. Uživatel byl často nucen instalovat různý dodatečný software pro zhlédnutí videa na různých webových stránkách, což zejména u méně zkušených uživatelů nezřídka končilo neúspěchem.

V posledních letech došlo k vymizení technologií Java a ActiveX z prostředí webových stránek. Důvodem je ukončení jejich podpory v prohlížečích kvůli obavám o bezpečnost uživatelů. [9] Některé prohlížeče zvolily jako dočasné řešení upozornění uživatele na možné nebezpečí vyskakovacím oknem s možností ruční aktivace doplňku. [23] Kromě toho, že se

[4] PoE – Power over Ethernet (technologie napájení přes Ethernetový kabel)
[5] NVR – Network Video Recorder (síťový záznamník videa)
[6] HTML – Hypertext Markup Language (značkovací jazyk pro tvorbu WWW dokumentů)



Obrázek 2.3: Příklad topologie digitálního (IP) kamerového systému. Kamery jsou součástí počítačové sítě (LAN). Je možné se z libovolného počítače připojit k jednotlivým kamerám a sledovat jejich obraz. Záznamová zařízení (NVR) již nejsou centrálním prvkem systému, vystupují v roli klienta s funkcí záznamu.

jednalo o technologie umožňující stáhnout a spustit programový kód s neomezeným přístupem ke zdrojům počítače (zejména k souborovému systému), využívaly zastaralé komponenty, které trpěly řadou zranitelností, například rozhraní NPAPI. [7][19][20]

Spolu s vývojem internetu, zvyšujícím se výkonem počítačů a stoupající rychlostí internetu stoupala i popularita videa na webu. Postupem času si oblíbenost získaly zejména přehrávače implementované jako objekty Flash. Toto řešení však bylo stále poměrně těžkopádné, jelikož na straně uživatele vyžadovalo instalaci přehrávače formátu Flash a na straně tvůrce webu představovalo konverzi obsahu do formátu SWF. Bylo žádoucí zavést standard, který by předepisoval nativní podporu videa v prohlížečích.

Třebaže implementace není zcela jednotná, je dnes již situace značně příznivější. Standard HTML5 [5] definuje objekt `HTMLMediaElement` určený pro přehrávání médií. Pro vkládání videa do dokumentu definuje novou značku `<video>`.

2.3 Přehled technických aspektů a problémů přímého přenosu na web

Při snaze o zajištění přenosu videa na web se setkáváme zejména s dvěma otázkami:

- V jakém formátu musí být obrazová data
- Jakým způsobem data přenést

První otázka vede na průzkum používaných multimediálních kontejnerů[7] a formátů kódování videa a ověření, které (zda) mají nativní podporu v internetových prohlížečích. Zod-

[7] Multimediální kontejner – typ metasouboru pro ukládání metadat společně s proudy kódovaných mediálních dat

povězení druhé otázky vyžaduje prověření komunikačních protokolů používaných kamerami a internetovými prohlížeči.

Na webu se obvykle můžeme setkat s formáty MP4 obsahující data kódovaná obvykle ve formátu H.264, FLV (video kódované v Sorenson Spark, VP6, případně H.264)[8]. Problémem rozšířeného formátu H.264 je zatížení patentem, který vyžaduje odvádění poplatků za jeho použití společnosti MPEG LA. V roce 2010 firma Google vydala formát WebM přímo dedikovaný pro účely přehrávání videa na webu. Jedná se o formát mediálního kontejneru vytvořený zjednodušením open source kontejneru Matroska. Obrazová data obsažená v kontejneru WebM mohou být kódovaná ve formátu VP8 nebo modernějším (a efektivnějším) VP9. Motivací k vytvoření tohoto formátu byla především snaha poskytnout svobodný formát, který lze použít bez nutnosti placení poplatků za jeho použití. [25] Formáty kódování VP8 a VP9 jsou vysoce efektivní. Kvalita obrazu kódovaného ve formátu H.264 a ve formátu VP8 při použití srovnatelných parametrů se přinejmenším při subjektivním srovnání zdá být velmi podobná. Technicky je komprese formátu H.264 jen mírně efektivnější. [22]

Kamery zpravidla komunikují kontrolním protokolem RTSP[9] a dvojicí protokolů RTP[10] (pro přenos dat) a RTCP[11] [21] (pro monitorování kvality služeb). Tyto ale nejsou běžně implementovány v prohlížečích.

Přenos videa v prostředí webu je řešen různými způsoby. Pro komunikaci se standardně využívají protokoly HTTP[12] a RTMP[13]. Protokol RTMP pro tuto práci není vhodný, protože není implementován přímo v prohlížeči; jeho použití je spojeno s přehrávačem Flash, který přestává být podporován [11]. Zejména pro statický obsah (VoD[14] je atraktivní využití technologií HLS[15] či DASH[16], které umožňují dynamické přizpůsobení kvality videa podle aktuálního vytížení sítě. To vyžaduje, aby bylo video předem kódované po krátkých úsecích a v různých kvalitách a klient si jen vybere soubor v kvalitě podle aktuální potřeby. Ačkoli je možné tyto techniky využít i pro živé přenosy, v prostředí této práce nepřinese žádný užitek; nasazení takových technik naopak je v tomto případě kontraproduktivní, protože by vyžadovalo kódování každého spuštěného streamu v několika různých kvalitách a zbytečně zatěžovalo server. Zřídka využívanou alternativou pro přenos pohyblivého obrazu je načítání a zobrazování jednotlivých obrázků v rychlém sledu (např. 12 požadavků za sekundu).

[8] Zdroj – https://en.wikipedia.org/wiki/Flash_Video

[9] RTSP – Real-Time Streaming Protocol (textový protokol pro řízení streamování multimédií)

[10] RTP – Real-time Transport Protocol (protokol pro přenos multimediálních dat v počítačové síti)

[11] RTCP – RTP Control Protocol (protokol obvykle používaný splečně s RTP určený k monitorování kvality přenosu)

[12] HTTP – Hypertext Transport Protocol (textový protokol určený pro komunikaci s webovými servery)

[13] RTMP – Real-Time Messaging Protocol (protokol vyvinutý firmou Macromedia za účelem streamování mediálních dat pro přehrávač formátu Flash)

[14] VoD – Video On Demand (video na vyžádání)

[15] HLS – HTTP Live Streaming (technika pro adaptivní streamování vytvořená firmou Apple)

[16] DASH – Dynamic Adaptive Streaming over HTTP (technika adaptivního streamování vytvořená pracovní skupinou MPEG)

Kapitola 3

Analýza situace

Aplikace bude vyvíjena pro prostředí Masarykovy univerzity. V této kapitole budou nastíněna specifika tohoto prostředí a na jejich základě budou definovány požadavky na tento systém. Dále budou popsány možnosti řešení a bude zhodnoceno, do jaké míry odpovídají požadavkům.

3.1 Kontext prostředí

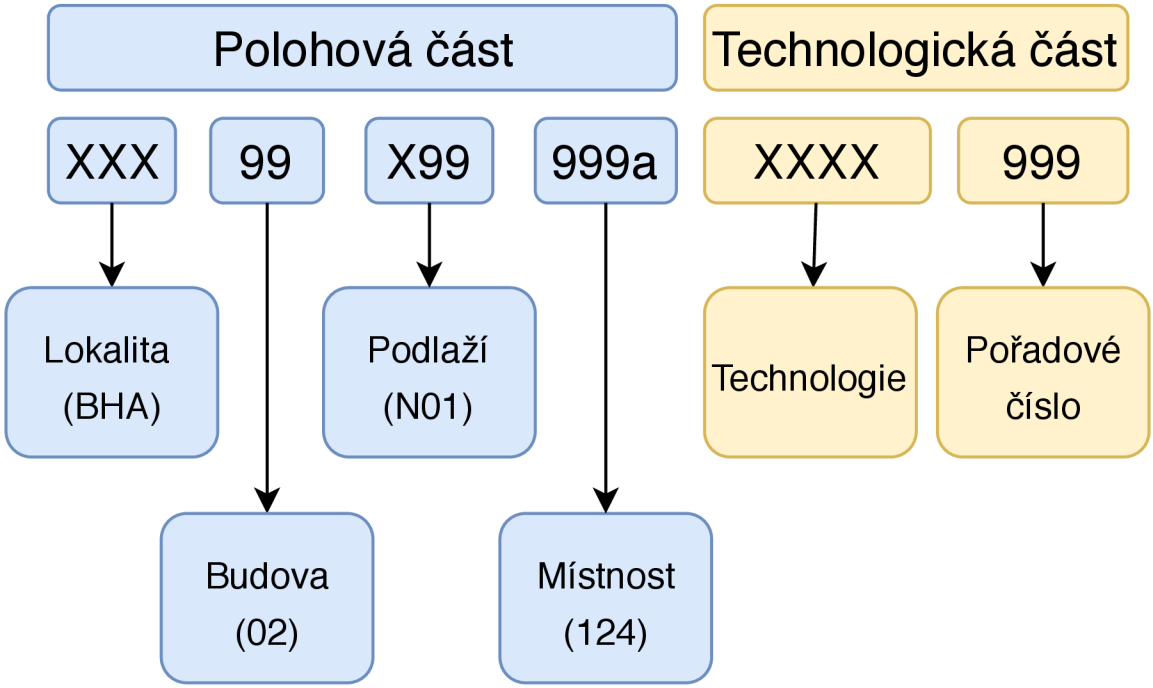
Běžným využitím kamer je shromáždění jejich výstupů do jednoho místa – tzv. dohledového stanoviště (pult centralizované ochrany). Operátoři takového stanoviště mohou mít kromě obrazu z kamer přístupná i jiná data z bezpečnostních systémů – elektronickou požární signalizaci nebo zabezpečovací systémy. Realizace takového stanoviště je poměrně přímočará – výstupy všech kamer vedou na jedno místo, které může být patřičně vybaveno adekvátním hardwarem i softwarem.

V prostředí Masarykovy univerzity ale existuje požadavek přístupu k obrazu z kamer i mimo tato centralizovaná dohledová stanoviště, např. vedoucí laboratoří, vedoucí ostrahy. Vzniká potřeba umožnit přístup z kamer značně více osobám než pouze operátorům stanovišť. Tato oprávnění vznikají, zanikají a mění se v průběhu času a bylo by z pohledu správy neefektivní každému novému uživateli muset poskytovat speciální vybavení. Vznikla tedy potřeba jednoduše přenositelného, na zdroje (klientů, i samotné infrastruktury) nenáročného systému, který bude elegantně umožňovat přístup různým uživatelům (ověřených univerzitní autentizací) k živému obrazu z různých kamer, specifikovaných sofistikovaně pomocí hierarchie lokalit.

UČO

Na Masarykově univerzitě se pro jednoznačnou identifikaci osob používá UČO[1]. Většinou se jedná o šestimístné číslo, ale vzhledem k historickým změnám existují čísla i v jiném formátu, ve speciálních případech mohou dokonce obsahovat i písmena. Z toho důvodu je nejlepší možností uložit jej jako řetězec. V aplikaci UČO slouží jako identifikátor přihlášeného uživatele, při autentizaci se předává do aplikace ze systému jednotného přihlášení a v rámci uživatelského rozhraní jej lze použít k vyhledání osoby.

[1] UČO – univerzitní číslo osoby



Obrázek 3.1: Struktura technologického kódu s příkladem konkrétního polohového kódu (BHA02N01124). Písmeno v identifikaci místnosti je nepovinné a slouží např. při rozdělení jedné místnosti na dvě.

Technologický kód

Technologický kód je řetězec používaný pro identifikaci zařízení a lokality, ve které jsou umístěny. Obdobné řetězce se používají napříč v téměř všech větších institucích, řídí se podle Evropské normy (ČSN EN 15221 či novější ČSN EN ISO 41011). Na Masarykově univerzitě se tento kód skládá ze dvou částí:

- polohová část – specifikuje lokalitu, konkrétní budovu, podlaží a místnost
- technologická část – označuje technologii a její pořadové číslo v rámci dané místnosti

Tyto dvě části dohromady tvoří jednoznačnou identifikaci jednoho konkrétního prvku technologie (např. kamery, svítidla, hasicího přístroje). Obrázek 3.1 znázorňuje strukturu technologického kódu. Formát a pravidla pro tvorbu technologického kódu jsou podrobně popsány v Metodice technologického pasportu [14]. Podrobnější struktura technologické části kódu není pro účely této práce relevantní.

Díky rozdělení do úrovní podle lokalit, budov, podlaží a místností vzniká hierarchická struktura. Porovnáváme-li vztah dvou lokalit, potom podřazená lokalita bude ta, jejíž polo-

hový kód bude delší a zároveň její prefix bude tvořit polohový kód nadřazené lokality. Toho je využito pro účel správy oprávnění ke kamerám – podrobněji popsáno v kapitole 4.3.

3.2 Požadavky na systém

Cílem práce je vyvinout řešení, které bude splňovat následující:

- Bude přehrávat živý obraz z kamer
- Bude umožňovat snadnou správu kamer
- Bude implementovat správu práv pro každého uživatele
- Bude využívat HTML5 video a formát WebM
- Bude podporovat (univerzitní) SSO[2] autentizaci
- Bude fungovat v internetových prohlížečích (aktuální verze Chrome, Firefox, Edge)

3.3 Průzkum existujících řešení

Kamerové systémy se vyskytují po celém světě v mnoha organizacích. Není vyloučeno, že již někdo řešil podobný problém a vyvinul řešení, na které by bylo možné navázat. Je pravděpodobné, že je volně k dispozici software, ať už v podobě hotového řešení, nebo částí (například programových knihoven), na kterých lze stavět systém přizpůsobený na míru prostředí Masarykovy univerzity.

Hotová řešení

Je možné, že již existují systémy, které splňují požadavky kladené na systém a které by bylo možné použít tak, jak jsou nebo s menšími úpravami. Jedním z požadavků je bezplatné nasazení, nebo alespoň možnost jednorázového zakoupení licence s trvalou platností.

Nimble Streamer Prvním zkoumaným řešením je freeware streamovací server Nimble Streamer[3]. S produktem je nabízena možnost rozšíření programu o přídavné moduly, placené formou předplatného, doplňující funkcionalitu překódování formátu, vkládání reklamy či správy digitálních práv. Na webu projektu je prazentována možnost využití programu jako DVR. Tento software by měl umožnit komunikaci s kamerou protokolem RTSP. Ačkoli je podporováno několik protokolů, včetně technologií HLS a DASH, formát VP8 a VP9 umí přijímat i odesílat pouze protokolem RTSP, ani placený překódovací modul nepodporuje kódování do těchto formátů. Z dostupných informací není jasné, zda program umožňuje dynamické vytváření streamů.

Wowza Streaming Engine Druhým zkoumaným řešením je Wowza Streaming Engine[4]. Jedná se o program vytvořený v jazyce Java, s možností nasazení na operačních systémech Windows, Linux a Mac, přímo nabízený jako řešení pro použití k bezpečnostním kamerám.

[2] SSO – Single Sign On (Systém jednotného přihlášení)
[3] Nimble Streamer – https://wmspanel.com/nimble
[4] Wowza Streaming Engine – https://www.wowza.com/products/streaming-engine

Tento software se zdá být kvalitním a robustním řešením s mnoha možnostmi (např. možnost integrace s IoT[5] systémy). Tento systém splňuje všechny technické požadavky. Veškeré kladné vlastnosti vyvažuje nutnost platit drahé předplatné za každou nasazenou instanci.

Využitelné komponenty

Protože snaha o nalezení mezi hotovými řešeními vhodného kandidáta skončila neúspěchem, je třeba přistoupit k průzkumu komponent, které by bylo možné využít k dosažení řešení.

Icecast Streamovací server Icecast[6] je distribuovaný pod licencí GNU GPL, primárně určený pro streaming podcastů. Již podporuje streamování videa ve formátu WebM, ale nepodporuje dynamickou tvorbu streamů. Pro vytvoření streamu je nutné uložit potřebné informace do konfiguračního souboru a restartovat aplikaci.

Stream-m Stream-m[7] je minimalistický streamovací server v jazyce Java zveřejněný pod licencí MIT. Tento program podporuje přenášení streamů ve formátu H.264 přes protokol RTMP a streamů ve formátu WebM přes protokol HTTP. Program sice neumí dynamické vytváření streamů, ale s menším úsilím je možné tuto funkcionalitu do programu implementovat. Výhodou je, že nabízí zachycování snímků videa a statistiky aktivních přenosů.

Red5 Streamovací server Red5[8] je implementovaný v jazyce Java pod licencí Apache. Vznikal jako open source, dnes je nabízena i placená nadstavba s rozšířenými funkcemi. Ve výchozím stavu podporuje protokol RTMP, včetně rozšíření RTMPT, RTMPS, a RTMPE. Formou zásuvného modulu lze doplnit podporu protokolů RTSP, WebSocket a HLS. Tento program nepodporuje formát WebM. Možnosti dynamického vytváření streamů z informací na stránce projektu nejsou jasné.

FFmpeg Nástroj FFmpeg[9] pro konverzi formátů multimediálních souborů licencovaný pod GNU LGPL určený pro použití v rozhraní příkazové řádky. Pro ty, kdo rozumí významu uváděných parametrů, je jeho použití jednoduché – není třeba žádná konfigurace, stačí sestavit příkaz pro spuštění programu. Pomocí programu FFmpeg je možné video kódovat do formátu WebM, ale je zapotřebí program přeložit s dodatečnými parametry pro zahrnutí kodeku `libvpx`. Výhodou je možnost použití protokolu RTSP jako vstup.

3.4 Bezpečnostní úvahy

Obraz z bezpečnostních kamer lze jednoznačně považovat za citlivá data, v případě záznamu osob i za osobní údaje. Z toho důvodu je nutné maximálně omezit náhodný či dokonce organizovaný přístup nepovolané osoby k němu.

V případě systému pro sledování kamer možná bezpečnostní rizika jsou:

[5] IoT – Internet of Things (internet věcí)
[6] Icecast – https://icecast.org/
[7] Stream-m – https://github.com/vbence/stream-m
[8] Red5 – https://github.com/Red5/red5-server
[9] FFmpeg – https://ffmpeg.org/

Zneužití přístupových údajů ke kameře Řešením je omezení přístupu k těmto údajům. Tyto nebudou ukládány na klientské stanici a v systému k nim budou mít přístup jen určení správci.

Odposlechnutí obrazových dat Pro případ zachycení dat videa na cestě mezi serverem a klientem je žádoucí, aby komunikace probíhala šifrovaně a útočník neměl možnost data dešifrovat. Pro zabezpečenou komunikaci klienta s webovým serverem je standardem protokol HTTPS[10], který zajišťuje důvěrnost.

Podvržení obrazových dat Může dojít k situaci, kdy útočník bude chtít klientovi podvrhnout nepravé video s úmyslem zabránit sledování obrazu z kamery. Protože protokol HTTPS zajišťuje také autentizaci a integritu, podvržení po cestě mezi serverem a klientem by mělo být zabráněno. Pro případ, že by se útočník rozhodl video publikovat na server, může být jako jednoduchá ochrana implementováno ověření heslem při pokusu o publikaci. Aby bylo minimalizováno riziko získání tohoto hesla, je vhodné, aby bylo automaticky generováno ad-hoc, pro každý vytvořený stream a v délce a sadě znaků zvolené tak, aby nebylo možné na běžném stroji v rozumném čase toto heslo prolomit.

[10]HTTPS – Hypertext Transfer Protocol Secure (protokol umožňující zabezpečenou komunikaci především mezi webovým serverem a klientem)

Kapitola 4

Návrh streamovacího systému

Protože nebylo nalezeno řešení, které by zcela vyhovovalo stanoveným požadavkům, je třeba přistoupit k návrhu vlastního řešení.

4.1 Prvky systému

Navrhovaný systém musí zvládnout:

1. komunikovat s kamerou
2. zajistit konverzi videa do formátu WebM
3. distribuovat obraz připojeným uživatelům

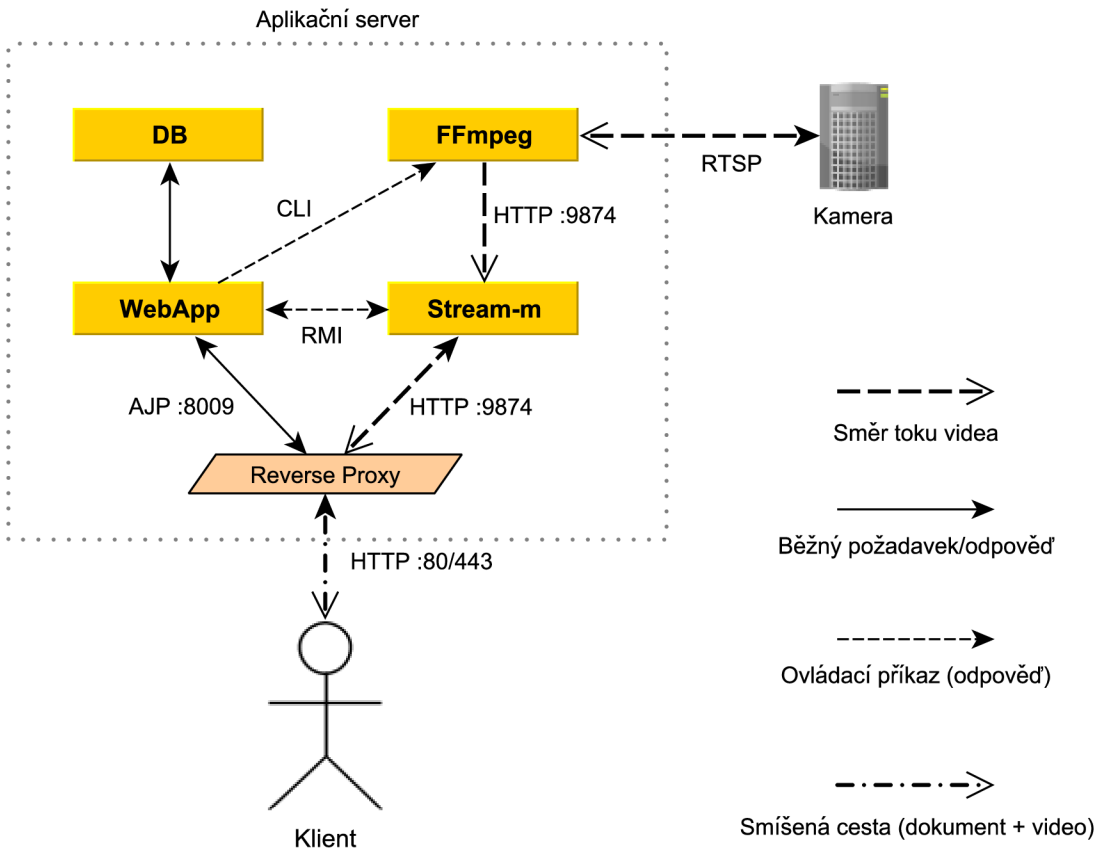
První dva body splňuje program FFmpeg, který byl jako součást řešení vybrán zejména kvůli konverzi videa. Protože umožňuje jako zdroj dat použít protokol RTSP (společně s protokoly RTP a RTCP), není třeba dalšího software, pro spojení s kamerou. Pro distribuci videa uživatelům jsem vybral program Stream-m, protože jeho vlastnosti přesně odpovídají požadovaným funkcionalitám. Nad těmito programy je třeba vytvořit aplikaci, která bude představovat uživatelské rozhraní, uchovávat údaje o kamerách a zajišťovat vytváření a rušení streamu podle potřeby. Pro ukládání dat, která mají přečkat ukončení aplikace (perzistentních dat) bude využit databázový systém. Nakonec mezi vyjmenované aplikace a uživatele bude umístěn reverzní proxy server. Jeho význam je diskutován dále v textu.

Celkové schéma systému včetně bližšího popisu principu funkce systému je znázorněno na obrázku 4.1.

4.2 Architektura webové aplikace

Zejména proto, že stream server Stream-m je implementován v jazyce Java, jsem se rozhodl webovou aplikaci implementovat ve stejném jazyce. Díky tomu je možné pro vzájemnou komunikaci těchto programů využít rozhraní RMI[1]. Webové aplikace v jazyce Java se standardně vyvíjí ve formě tzv. modulů. Takové moduly se obvykle řídí danou adresářovou strukturou, do které jsou umístěny soubory typu objekt servlet (programový kód v jazyce Java dědící z třídy `HttpSerrvlet`, určený k tvorbě webových aplikací), JSP (soubor, ve kterém je možno prolínat programový kód s definicí uživatelského rozhraní, principem se

[1] RMI – Remote Method Invocation (technologie pro vzdálenou invokaci metod v jazyce Java)



Obrázek 4.1: Blokový návrh systému. Při požadavku na spuštění přenosu je třeba stream registrovat na straně streamovacího serveru (Stream-m), tj. zprostředkovat mu informace, na základě kterých programu FFmpeg umožní publikování videa. Webová aplikace (WebApp) za tímto účelem vygeneruje identifikátor (název) a heslo pro vytvářený stream. Dále je na základě údajů spojených s kamerou a vygenerovaných registračních údajů sestaven příkaz pro spuštění programu FFmpeg. FFmpeg naváže spojení s kamerou ovládacím protokolem RTSP, provede autentizaci a spustí přenos videa. Jakmile obdrží a překóduje určité množství dat, začne video přes protokol HTTP publikovat na server Stream-m. Z vygenerovaného identifikátoru je sestavena adresa (URI) pro daný stream. Po jejím navštívení se uživateli zobrazí živý přenos.

podobá jazyku PHP) a zpravidla XML určený k mapování obslužných metod na přípojné body (tzv. endpointy) protokolu HTTP. Ačkoli to pro nasazení a běh není nutné, je běžné takto vytvořenou webovou aplikaci (webový modul) zabalit jako archiv WAR[2]. Ten je poté nasazen na webový kontejner – speciální webový server, který dokáže takové webové aplikace spouštět. Obvykle jsou používány kontejnery Apache Tomcat nebo Jetty.

Já jsem pro vývoj zvolil moderní framework Spring Boot, který přináší značně efektivnější přístup k tvorbě webových aplikací v jazyce Java. Různé úkony lze elegantně zařídit použitím k tomu určených anotací. Některé běžné úkony Spring provádí zcela automaticky. I tyto však je možné přizpůsobit aktuálním potřebám, ať už použitím anotací nebo změnou konfigurace. V neposlední řadě Spring Boot umožňuje při překladu přímo do aplikace vložit webový server, takže si programátor může vybrat, zda aplikaci přeloží do archivu WAR pro použití v kontejneru, nebo jako samostatnou aplikaci.

Webové aplikace založené na frameworku Spring často využívají architektury MVC, která odděluje logické části aplikace – datový model, definici webového uživatelského rozhraní a aplikační logiku, kontroler. Uživatelské rozhraní se ve Spring definuje pomocí šablon. Při vytváření aplikace si lze vybrat z několika šablonovacích systémů. Já jsem zvolil systém Apache Freemarker[3], zejména pro jeho jednoduchost, přehlednost a čistotu výsledného kódu. Kontroler definuje identifikátory a chování pro jednotlivé koncové body HTTP požadavků (tzv. endpointy). Data, která mají být přístupná v šabloně, předáváme z kontroleru pomocí modelu.

Já jsem zodpovědnost v aplikaci rozdělil do tří kontrolerů.

- *MainController* slouží jako vstupní bod pro uživatele. Narozdíl od ostatních kontrolerů naplňuje princip architektury MVC – pracuje s modelem a výsledkem v něm implementovaných endpointů je pohled (view). Při přístupu na některý z endpointů tohoto kontroleru jsou zpravidla nastaveny potřebné atributy v modelu, poté dojde ke zpracování kódu příslušné šablony a výsledný HTML dokument je odeslán uživateli.

- *ApiController* je určen pro zpracovávání asynchronních požadavků z aplikace odeslaných pomocí technologie AJAX[4]. Při požadavku na tento kontroler není nutné zpracovávat šablony a přenášet znovu celý dokument. Výsledkem požadavku je krátká odpověď ve formátu JSON[5], o prezentaci získaných dat se stará klientská část aplikace.

- *FileResourceController* je určen pro práci se statickým obsahem. V současnosti jsou v něm implementovány endpointy pro získávání a aktualizaci náhledových obrázků kamer. Tento kontroler byl vytvořen proto, že jeho funkce sémanticky nespadá do kompetencí výše zmíněných kontrolerů.

Data uchovávaná v systému

Pro aplikaci je zásadní schopnost uchování dat. Já jsem zvolil databázový systém MySQL, protože s ním mám zkušenosti.

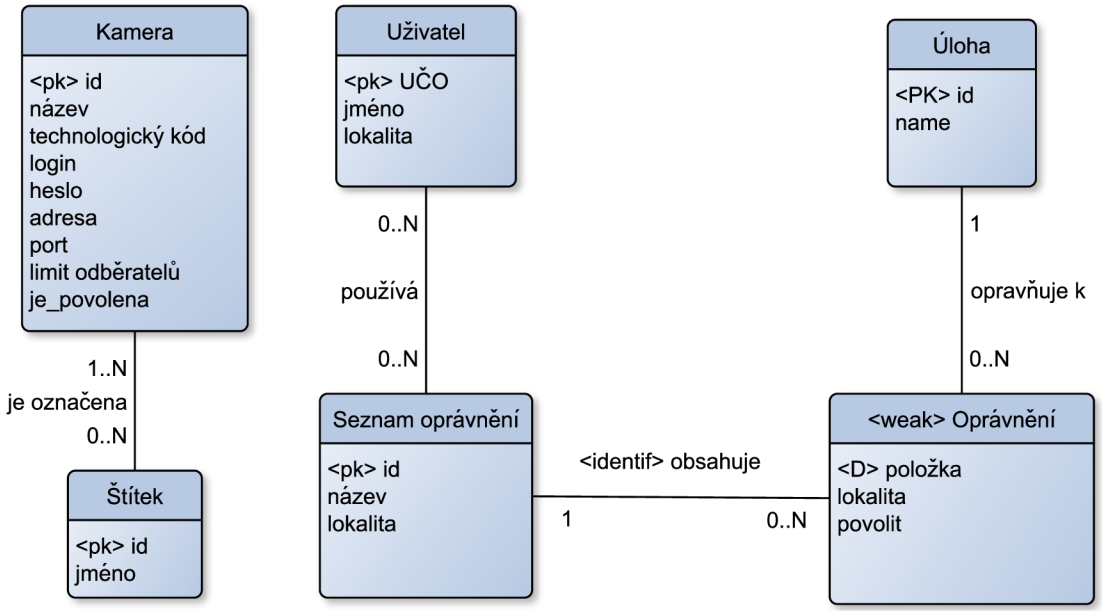
V systému je potřeba uchovávat několik entitních množin. Zcela jistě jsou to informace o kamerách, zejména jejich adresa v síti a číslo portu, na kterém očekávají spojení a údaje

[2]WAR – Web Archive (formát archivu pro webové aplikace v Javě, obdoba archivu JAR)

[3]Šablonovací systém Apache Freemarker https://freemarker.apache.org/

[4]AJAX – Asynchronous Javascript And XML (technologie pro asynchronní načítání dat)

[5]JSON – JavaScript Object Notation (formát pro serializaci strukturovaných dat vycházející ze syntaxe jazyka Javascript)



Obrázek 4.2: ER diagram vyvíjené aplikace. Zobrazuje entitní množiny používané v aplikaci a vztahy mezi nimi.

pro autentizaci. Zvláště pro účely řízení přístupu budeme uchovávat technologický kód kamery, který obsahuje informaci o jejím fyzickém umístění. V některých případech může být užitečné omezit počet uživatelů, kteří mohou zároveň sledovat stream z dané kamery nebo zcela zakázat spuštění streamu.

Každou kameru bude možné označit různými štítky. Ty můžou sloužit k označení jakýchkoli vlastností kamer (např. „otočná“). Štítky nejsou předem definované, mohou je vytvářet sami uživatelé při editaci kamery. Stejným štítkem může být označeno jakékoli množství kamer. Následně bude možné podle těchto štítků vyhledávat a filtrovat tak kamery se společnými vlastnostmi.

Za účelem autorizace je nutné znát množinu uživatelů, kteří mohou aplikaci používat. Protože bude aplikace napojena na systém jednotného přihlášení (SSO), není třeba uchovávat přihlašovací údaje uživatelů, stačí znát řetězec, kterým jsou v systému jednotného přihlášení identifikováni. Pro uživatelskou přívětivost bude uchováváno také jejich jméno. Lokalita bude sloužit pro řízení přístupu ke správě uživatelů.

Nakonec je třeba v systému uchovávat entity potřebné pro řízení přístupu. Tyto jsou podrobněji popisovány dále v sekci 4.3.

Vizualizaci všech entitních množin v systému zobrazuje diagram 4.2.

Perzistentní data ve frameworku Spring

Spring framework podporuje rozmanité technologie pro manipulaci s perzistentními (trvalými) daty. Pro práci s relační databází kromě standardního rozhraní JDBC[6] poskytuje objektově-relační mapování, tedy automatický převod entity z relačního modelu databáze

[6]JDBC – Java DB Connection (API pro přístup k relačním databázím pro jazyk Java)

na objekt v objektově orientovaném programu a naopak. K tomuto účelu využívá JPA[7], standardní implementaci objektově-relačního mapování v jazyce Java.

Výhodou objektově-relačního mapování je, že programátor nemusí sestavovat každý dotaz v jazyce SQL[8] a volat funkce databázového rozhraní. Programátor pracuje s entitami ve formě běžných objektů, což vytváří dojem, jako by byly implementovány ve stejném programovacím jazyku jako zbytek programu. Významnou výhodou je také fakt, že program neobsahuje konstrukce specifické pro konkrétní databázový systém; pro jeho změnu stačí upravit konfiguraci projektu.

4.3 Autorizace

Způsob autorizace (řízení oprávnění k objektům) lze implementovat různými způsoby. Často se užívá jednoho z následujících dvou přístupů:

- *Role* – Definuje úkony, které je daná osoba (či skupina osob) oprávněna vykonávat, zpravidla ale nedefinuje množinu objektů, ke které se oprávnění vztahuje (tedy se vztahuje na všechny). Může představovat např. reálnou pracovní pozici.
- *Seznam oprávnění (ACL[9])* – Definuje, kdo může s daným objektem provádět jakou akci [2]

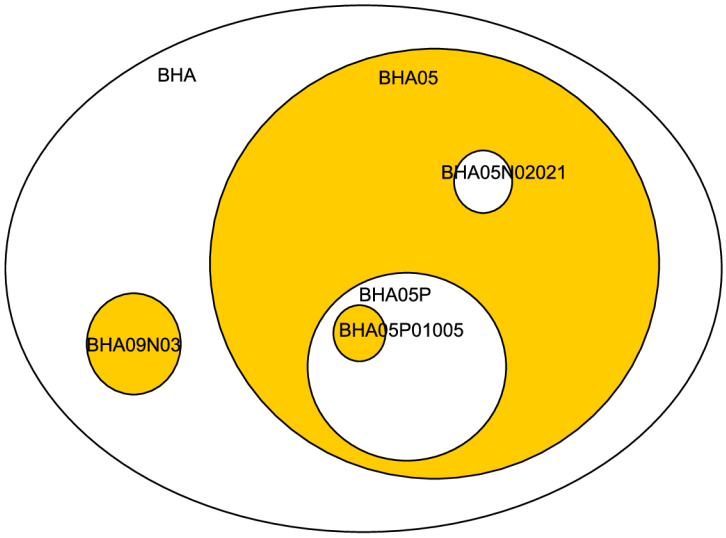
Já jsem se rozhodl pro řešení v podobě modifikovaného seznamu oprávnění. Důvodem je, že role neodpovídají požadavkům na systém. Určují sice, které úkony jsou povoleny, ale je potřeba také specifikovat, pro které objekty (resp. množiny objektů) jsou tato oprávnění platná.

Obvykle se pravidla oprávnění přiřazují objektu, ke kterému udělují oprávnění. V tomto systému by příklad takového systému oprávnění vypadal takto: každá kamera má svůj seznam oprávnění. V tomto seznamu oprávnění je definováno, kterými uživateli *může být sledována* nebo spravována, podobně u každého uživatele je definováno, kým může být editován. Tento způsob implementace oprávnění je výhodný u hierarchických objektů. Přiřazením pravidla oprávnění nadřazenému objektu je pravidlo aplikováno i na podřízené objekty. Protože kamery jsou rovnocenné objekty (nemají hierarchii), je pro naši aplikaci zvolen opačný, v tomto případě efektivnější přístup – seznamy oprávnění jsou přiřazeny uživateli; v tomto seznamu je definováno, které objekty tento uživatel *může spravovat* nebo, v případě kamery, sledovat.

Ve větších subjektech může být užitečné řídit oprávnění podle lokality. Jelikož lokality jsou z principu hierarchické (Budova –> podlaží –> místnost), lze na jejich základě efektivně vybírat podmnožiny i takových objektů (např. kamer), které nemají vlastní hierarchii a k takto vybraným podmnožinám objektů uživatelům definovat oprávnění. Uživatel tedy dostane oprávnění pouze k určitým lokalitám (podmnožinám objektů) a nemůže přistoupit k objektům (kamera, uživatel, seznam oprávnění) mimo tyto lokality. V aplikaci bude lokalitu reprezentovat polohový kód facility managementu, Pravidla polohových kódů používaných v této práci jsou uvedena podrobněji v kapitole 3.

V systému se nachází tři typy objektů (entit), ke kterým bude potřeba udělovat oprávnění: kamery, uživatelé a seznamy oprávnění. Je přirozené, že kamery mají stanoveno trvalé

[7] JPA – Java Persitence Api (specifikace rozhraní pro jazyk Java umožňující objektově relační mapování)
[8] SQL – Structured Query Language (strukturovaný dotazovací jazyk)
[9] ACL – Access-Control List (seznam pro řízení přístupu)



Obrázek 4.3: Příklad vizualizace hierarchie lokalit ve formě množin. Žlutě jsou vyznačeny lokality, ke kterým ACL povoluje přístup, bíle lokality, ke kterým je přístup zamítnut

umístění ve známé lokalitě. Poměrně intuitivně lze umístění (sídlo) přiřadit také uživateli – např. jeho kancelář nebo pracoviště. Na seznamy oprávnění lze tento přístup rovněž aplikovat, třebaže je o něco složitější na pochopení, jelikož se jedná o abstraktní, hůře představitelnou entitu. Lokalitu u seznamu oprávnění lze chápat jako lokalitu, které se týká resp. ke které má udělovat oprávnění. Tato by měla být co nejkonkrétnější a zároveň musí být nadřazena lokalitám jednotlivých pravidel v tomto seznamu.

Ačkoliv by ve většině situací mohly stačit pouze povolovací pravidla, může být užitečné mít možnost některé podmnožiny objektů z oprávnění vyjmout. Například může dojít k situaci, kdy má být povolen přístup k většímu množství souřadných lokalit a je jednodušší zadat jedno pravidlo povolující přístup k nadřazené lokalitě (např. podlaží) a pravidlo zakazující konkrétní lokalitu (např. přísněji střeženou místnost), než mnoho pravidel povolující přístup k jednotlivým souřadným lokalitám (místnostem). Tím však vzniká situace, kdy je možné uživateli definovat různě zanořená pravidla povolující nebo zakazující přístup k různým podmnožinám objektů. Třebaže nemusí jít o záměr, může taková situace vzniknout jako vedlejší efekt například při přiřazení více seznamů oprávnění stejnému uživateli. Při aplikaci pravidel se tyto seznamy spojí a vytvoří jeden seznam, který může obsahovat neomezeně zanořená pravidla. Na obrázku 4.3 je zobrazen příklad takového validního oprávnění.

Z výše uvedených indicií lze odhadnout, že pravidlo pro definici oprávnění se skládá z částí:

- **Typ oprávnění:** úkon, ke kterému uživatele zmocňujeme (např. "view_stream", "edit_camera", ...)
- **Lokalita:** Řetězec označující lokalitu podle zásad facility managementu
- **Informace o povolení / zamítnutí přístupu**

Třebaže se u některých typů oprávnění vyskytuje závislost na jiném typu a dává smysl, aby tato oprávnění byla nějak hierarchicky sdružena (např. aby editor kamer mohl kamery

automaticky také sledovat), rozhodl jsem se je implementovat jako nezávislá a správné přiřazení závislých oprávnění ponechat zodpovědnosti správce oprávnění. Pokud by se toto ukázalo jako problematické, lze přistoupit k řešení v podobě hlídání těchto závislostí na úrovni uživatelského rozhraní.

4.4 Proxy

Ačkoli se ze strany uživatele navržený systém jeví jako jedna aplikace, reálně klient přistupuje ke dvěma různým programům (webová aplikace a stream server) na aplikačním serveru, přičemž každý z těchto programů běží na odlišném portu. Tato skutečnost je hlavním důvodem pro nasazení proxy, které zajistí společné rozhraní pro přístup k oběma aplikacím.

Připojení k různým portům může být problém například kvůli politice „Same-Origin“, která z bezpečnostních důvodů omezuje načítání prostředků („resource“) pocházejících z různých zdrojů („origin“). Pro účely politiky Same-Origin je jako zdroj považována trojice „scheme“ (označení protokolu – http, https,...), „host“ (doména) a port. [26] Jelikož existují případy, kdy má smysl do dokumentu vkládat obsah z různých zdrojů (typicky se jedná například o webové fonty), vznikl mechanizmus CORS[10], který definuje výjimky pro politiku Same-Origin. [10] Zmíněný problém by tedy pomocí CORS by bylo možné vyřešit. Implementace CORS vyžaduje na straně serveru minimálně zahrnutí hlavičky `Access-Control-Allow-Origin` do odpovědi na HTTP požadavek, což lze snadno zařídit.

Dalším důvodem pro nasazení proxy je použití zabezpečeného protokolu HTTPS, který standardně přenáší data šifrovaná, čímž zajišťuje důvěrnost. Framework Spring sice umožňuje aktivaci HTTPS nastavením určených properties v konfiguračním souboru, Stream-m ale komunikuje pouze pomocí HTTP a implementace zabezpečeného protokolu by byla příliš náročná a náchylná k chybám. V serverovém programu Apache (také známý pod názvem httpd), který bude sloužit jako proxy, slouží pro HTTPS komunikaci modul `mod_ssl`[11].

Pokud uživatel nezadá protokol (schéma), zejména starší prohlížeče standardně posílají požadavek nejdříve na port 80 nezabezpečeným protokolem HTTP. Protože ale bude systém přenášet citlivá data, budeme požadovat komunikaci výhradně protokolem HTTPS. To lze zařídit přesměrováním z HTTP na HTTPS třeba pomocí modulu Rewrite[12].

Posledním významným důvodem pro použití proxy serveru je připojení k systému jednotného přihlášení (SSO[13]), které se dnes hojně využívá ve větších firmách. Výhodou SSO systémů je především možnost přihlášení do různých důvěryhodných aplikací, připojených do systému SSO, stejným uživatelským účtem. Protože na straně aplikace nejsou ukládány žádné přihlašovací údaje, uživatelé ani vývojáři či správci aplikací se nemusí obávat o bezpečnost těchto údajů.

V případě této práce je použit moderní autentizační protokol OpenID Connect [15]. Jako implementace tohoto protokolu poslouží modul `mod_auth_oidc` pro webový server Apache. V našem případě postačí základní konfigurace – proxy server se postará o autentizaci uživatele a poté aplikaci předá jeho identifikátor v systémové proměnné `REMOTE_USER`. Webová aplikace tento identifikátor získá jako řetězec voláním metody `getRemoteUser()` na objektu `HttpServletRequest`, pokusí se uživatele vyhledat v databázi a ověří jeho oprávnění.

[10] CORS – Cross-Origin Resource Sharing
[11] Modul SSL – https://httpd.apache.org/docs/current/mod/mod_ssl.html
[12] Modul Rewrite – https://httpd.apache.org/docs/current/mod/mod_rewrite.html
[13] SSO – Single Sign On (Systém jednotného přihlášení)

Ačkoliv by všechny tyto úkony bylo možné pomocí frameworku Spring implementovat do jedné aplikace, rozhodl jsem se pro použití více programů, protože toto řešení je více modulární a tedy nabízí větší prostor pro změnu technologií. Je možné například změnit autentizační protokol, aniž by bylo nutné znovu překládat webovou aplikaci.

Kapitola 5

Implementace

V této kapitole budou popsány některé zajímavé problémy, na které jsem narazil během implementace.

5.1 Řízení streamování

Spuštění streamu

Aby uživatel mohl sledovat přenos z kamery, musí být spuštěno streamování. Spuštění streamu zahrnuje:

1. Získání údajů o kameře z databáze
2. Registraci streamu – vytvoření potřebných atributů na straně serveru Stream-m
3. Spuštění streamu – spuštění programu FFmpeg

Před spuštěním streamování je třeba na straně stream serveru provést registraci – to znamená informovat server o tom, že má povolit publikaci streamu s uvedeným jménem a heslem. Streamovací server Stream-m si tyto informace uchovává ve formě objektů typu Properties. Jde o potomka třídy Hashtable, hashovací tabulky, která umožňuje mapování objektu na jiný objekt – při vyhledávání v tabulce jeden z objektů slouží jako klíč, druhý představuje hledanou hodnotu. Třída Properties toto mapování specifikuje pouze na řetězce na straně klíče i hodnoty. Kromě metod pro práci s dvojicemi řetězců obsahuje také metody pro jejich načítání ze souboru i ukládání; podporuje soubory ve formátu XML[1] i ve formátu properties (vytvořeného k tomuto účelu). Třída a soubory typu Properties byly vytvořeny za účelem perzistentního uchování dvojic řetězců a jsou běžně využívány pro uchování konfiguračních parametrů. Lze s nimi ale pracovat (přidávat, měnit) i za běhu programu.

[1] XML – eXtensible Markup Language (rozšiřitelný značkovací jazyk)

Pro každý stream jsou uchovávány následující informace

- `stream.[název streamu]` – uchovává informaci o tom, zda je stream s daným jménem povolen nebo zakázán (to může být užitečné například v případě, že uživatel předem připravuje konfiguraci pro stream, který ještě nemá být možné spustit)
- `stream.[název streamu].password` – heslo, kterým se autor streamu (v tomto případě program FFmpeg) autentizuje. V našem případě je, stejně jako název streamu, generováno aplikací.
- `stream.[název streamu].limit` – maximální počet klientů, kteří mohou zároveň sledovat tento stream

Pro účely vyhledávání streamu podle kamery jsem přidal property s názvem `cam.id` obsahující identifikátor kamery používaný v databázi.

Stream-m tyto informace standardně načítá při spuštění z konfiguračního souboru. Toto není vyhovující řešení, protože systém musí být schopný streamy vytvářet dynamicky za běhu aplikace.

Potom již je sestaven příkaz pro program FFmpeg, kterým je zahájeno odebírání obrazu z kamery, překódování a odesílání videa. Po čase jsem objevil, že dochází k zajímavému problému – po určité době (vždy se jednalo o cca dvě hodiny) dojde k zamrznutí programu FFmpeg, a tedy i streamu. Problém tkvěl v tom, že program FFmpeg na standardní chybový výstup generoval nemalé množství informací a po určité době došlo k přeplnění bufferu chybového výstupu, protože obsah bufferu nebyl odebírán. Problém byl vyřešen přesměrováním chybového výstupu do `/dev/null`.

Komunikační rozhraní mezi webovou aplikací a stream serverem

Pro zajištění komunikace mezi ovládající aplikací a Stream-m bylo nutné vytvořit nějaké komunikační rozhraní. Kandidátními technologiemi byly REST, CLI, Socket s vlastním protokolem a RMI. Všechny vyžadují úpravy aplikace Stream-m. Nakonec byla zvolena technologie RMI[2], protože je implementačně výhodná a jednoduchá, je nativní, používá standardní rozhraní jazyku Java a vzdálené volání vypadá stejně, jako volání běžné metody...

Spuštěním programu FFmpeg zahájíme přenos obrazu z kamery, jeho konverzi do formátu WebM a publikování na server Stream-m. Při příchodu publikačního požadavku od FFmpeg jsou na serveru vytvořeny potřebné datové struktury pro uchování obrazových dat, poté je stream připraven k odběru.

Metody pro spuštění a zastavení programu FFmpeg implementuje třída FFControl. Program FFmpeg neposkytuje žádné programové rozhraní (API[3]), proto je ovládán prostřednictvím příkazového rozhraní operačního systému (CLI[4]).

Zastavení streamu

Každý běžící stream zabírá nezanedbatelné množství technických prostředků. V tomto směru jsou nejvíce znatelné překódování videa a kapacita sítě. Je třeba si uvědomit, že kromě obsazení výpočetní a přenosové kapacity (a jejich potenciálního zahlcení) v tom

[2] RMI – Remote Method Invocation (technologie pro vzdálenou invokaci metod v jazyce Java)
[3] API – Application Programming Interface (rozhraní pro programování aplikací
[4] CLI – Command Line Interface (příkazové, textové rozhraní operačního systému)

smyslu, že je nelze využít jinak, představuje každá aktivní výpočetní úloha zvýšení spotřeby elektřiny. Každý aktivní stream generuje elektrickou spotřebu na straně kamery, na straně serveru, a také na cestě mezi nimi. Z těchto důvodů je žádoucí nechávat spuštěné pouze streamy, které uživatelé sledují.

Nabízí se několik možností, jak zastavení streamu uskutečnit:

- Zastavit stream okamžitě po odchodu posledního klienta
- Zastavit stream po uplynutí individuálního časového limitu (timeoutu) nečinnosti streamu
- Periodicky vyhledávat a zastavovat neaktivní streamy

Okamžité zastavení streamu je řešení implementačně nejjednodušší, zato ale ne zcela praktické. Za specifických okolností může být výhodné nezastavovat stream okamžitě po ukončení jeho sledování. Například v případě, kdy uživatel omylem zavře okno prohlížeče a chtěl by se ke sledování vrátit. Nebo v případě, kdy se jedná o intenzivně sledovanou kameru. Může se stát, že jeden uživatel se odpojí, ale krátce nato stejnou kameru začne sledovat jiný uživatel. Jiným příkladem takové situace může být střídání směn na dohledovém stanovišti.

Individuální řízení odloženého zastavení streamu je atraktivní řešení, neboť nejlépe reflektuje specifické situace zmíněné v předchozím odstavci. Nadto je zde možnost nastavení různých časových limitů pro různé kamery. Tyto individuální limity by mohly být nastaveny ručně u každé kamery nebo být generované podle nějaké heuristiky založené třeba na statistikách sledovanosti jednotlivých kamer. Takové řešení se však v současnosti zdá zbytečně komplexní.

Z vyjmenovaných možností nejlépe vychází periodické vyhledávání streamů, které nejsou sledovány. Toto řešení se zdá být nejelegantnější, protože není zapotřebí implementovat obousměrnou komunikaci mezi webovou aplikací a stream serverem. Řízení streamů tak bude kompletně přenecháno pouze webové aplikaci.

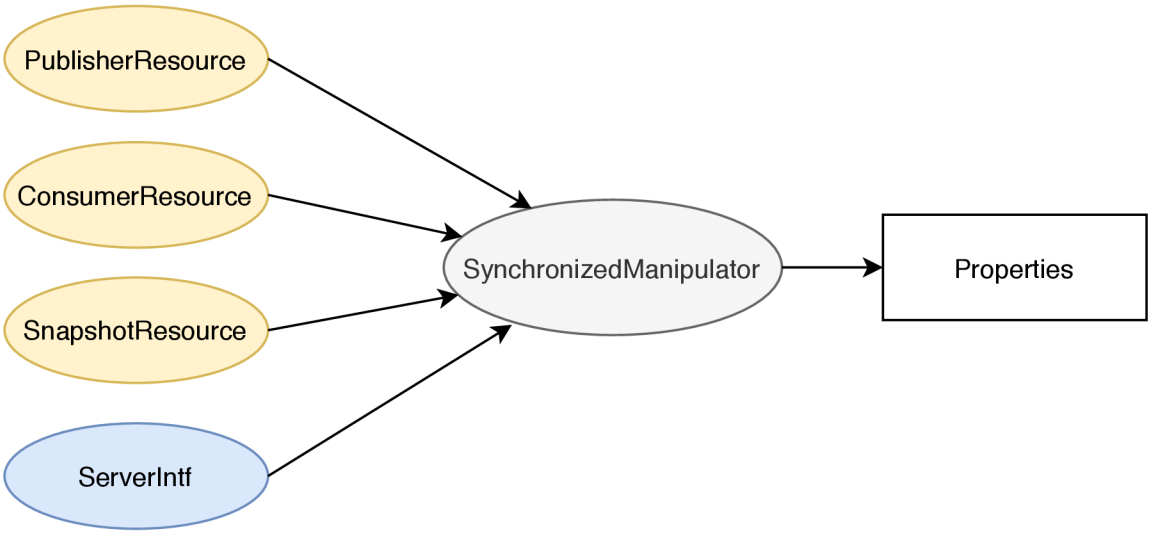
Každý stream v programu Stream-m si v proměnné `numClients` uchovává počet uživatelů, kteří jej odebírají. Původním účelem této proměnné bylo zamezit odběru daného streamu v případě, že bylo dosaženo maximálního počtu klientů definovaného při registraci streamu (property `stream.[název streamu].limit`). Této proměnné ale můžeme využít i k detekci neaktivního streamu. Implmentace na straně Stream-m je přímočará. Do RMI rozhraní stačí přidat metodu, která projde všechny existující streamy a zastaví ty, jež mají nulový počet klientů.

Ve webové aplikaci můžeme k plánování úlohy využít Spring anotace `@Scheduled`[5]. Intervaly mezi invokacemi úlohy lze nastavit pomocí konfigurační direktivy `scheduler.stop-stream-delay`

Synchronizace

Přestože je v dokumentaci třídy `Properties` deklarováno, že není zapotřebí externí synchronizace při sdílení objektu tohoto typu více programovými vlákny, docházelo k situacím, kdy se úprava atributů v něm obsažených neprojevila v ostatních vláknech. Pro vyřešení této situace jsem navrhnul vytvoření třídy `SynchronizedManipulator`, která jediná bude přistupovat přímo k objektu `Properties` a všechny ostatní objekty budou využívat jejích metod jako rozhraní pro práci s atributy `Properties`. `SynchronizedManipulator` obsahuje metody pro vytváření registračních atributů (blíže viz kapitolu 4). Těch, bude tato nová

[5]Spring dokumentace https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/scheduling/annotation/Scheduled.html



Obrázek 5.1: Ukázka topologie vložené třídy `SynchronizedManipulator`. Místo, aby objekty `PublisherResource` a další uchovávaly referenci na objekt `Properties`, mají k dispozici pouze referenci na objekt `SynchronizedManipulator`, která obsahuje synchronizované operace.

třída používat synchronizačních prostředků jazyka Java pro definice atomických operací nad atributy.

5.2 Pomocné metody kontrolerů webové aplikace

Vytvořená webová aplikace je postavena na architektuře MVC[6]. Jejím jádrem jsou tři kontrolery – `MainController`, `ApiController` a `FileResourceController`, ve kterých je implementována převážná většina aplikační logiky. `MainController` je dedikován požadavkům, jejichž výsledkem má být HTML[7] dokument, `ApiController` představuje rozhraní pro požadavky realizované prostřednictvím AJAX[8], `FileResourceControler` poskytuje náhledové obrázky a umožňuje jejich aktualizaci.

Předávání autentizovaného uživatele

Za povšimnutí stojí metoda `preHandle()`, která se vyskytuje ve všech kontrolerech této aplikace. Je označena Spring dekorátorem `@ModelAttribute`, který způsobuje její invokaci při požadavku na kterýkoli endpoint definovaný v daném kontroleru. Tělo metody `preHandle()` je provedeno dříve než metoda obsluhující konkrétní endpoint. Díky tomu je možné provádět například inicializace modelu, které mají být společné pro všechny endpointy – lze sem tedy vyčlenit část kódu, který by se jinak musel opakovat v každé metodě. V naší aplikaci je zde implementováno především vyhledání autentizovaného uživatele. Pokud je nalezen (existuje v databázi), je uložen do modelu pro použití v jednotlivých obslužných metodách.

[6] MVC – Model, View, Controller (doporučená architektura pro aplikace s grafickým uživatelským rozhraním)

[7] HTML – Hypertext Markup Language (značkovací jazyk pro vytváření webových dokumentů)

[8] AJAX – Asynchronous Javascript And XML (technologie pro asynchronní načítání dat)

Díky sofistikovanému systému vkládání závislostí, kterým framework Spring disponuje, je možné data z modelu získat jako argument v metodě obsluhující daný endpoint. Tento způsob získávání modelových atributů (nebo jiných dat) je pro programátora komfortnější a také pomáhá přehlednosti programového kódu. Pro jeho využití je třeba umístit anotaci `@ModelAttribute` do hlavičky metody před datový typ formálního parametru.

Oba zmíněné způsoby využití anotace `@ModelAttribute` jsou obsaženy v ukázce ve výpisu 5.1.

```
@Controller
public class ExampleController{
    UserRepository users;

    @ModelAttribute
    public void preHandle(Model model, HttpServletRequest request){
        // ziskani identifikatoru autentizovaneho uzivatele
        String userName = request.getRemoteUser();
        // vyhledani uzivatele v databazi
        User u = users.findByIdstring( userName );
        if( u == null ) // uzivatel nenalezen
            throw new Forbidden();

        model.addAttribute("user", u); // predani uzivatele do modelu
    }

    @GetMapping("/example-endpoint")
    public String exampleEndpoint( @ModelAttribute User u ){
        // Uzivatel z modelu byl do metody predan jako argument
        if( ! u.isAuthorized("manage_permissions") )
            throw new Forbidden();

        ...
    }
}
```

Výpis 5.1: Ukázka kódu s využitím anotace `@ModelAttribute`. Použití anotace `@ModelAttribute` v kontroleru v místě předcházejícím předpisu metody způsobí její invokaci při požadavku na kterýkoli endpoint definovaný v tomto kontroleru. Díky tomu lze do zvláštní metody vyčlenit část kódu, který by se prováděl v každé obslužné metodě. Při použití v místě deklarace formálního parametru způsobí vložení (injektování) hodnoty atributu do tohoto parametru.

Zpracování výjimek

Podobný koncept jako `@ModelAttribute` představuje mechanizmus zpracování výjimek. Pro jeho využití Spring framework nabízí tři možné způsoby definice – na úrovni výjimek, na úrovni kontrolerů a na globální úrovni. V této aplikaci jsou výjimky zpracovávány na úrovni jednotlivých kontrolerů. Stejná výjimka je pak v každém kontroleru zpracovávána odlišně. Ve třídě `MainController` je výsledkem zpracované výjimky pohled (view) pro zprostředkování informace o chybě ve vizuální formě, v kontroleru `ApiController` je výsledkem objekt

JSON[9] s chybovým kódem, a případně chybovou zprávou (vizuální prezentace je zajištěna kódem na straně klienta).

Zpracování výjimek, není-li definováno na úrovni výjimky (zpravidla definováním chybového kódu HTTP odpovědi), je vyčleněno do určené metody. Metodu určenou pro zpracování výjimek je třeba dekorovat anotací `@ExceptionHandler`[10]. Takto označená metoda potom zachycuje výjimky, které propustily (příp. samy vygenerovaly) obslužné metody v kontroleru. Metod označených anotací `@ExceptionHandler` může být více – podobně jako je tomu u bloků `catch`, je vybrána ta metoda, která nejlépe vyhovuje typu zpracovávané výjimky. Na rozdíl od `catch` bloků, u metod zpracovávajících výjimky nezáleží na pořadí v jakém jsou definovány. Pokud by však standardní způsob výběru vhodného zpracovatele nebyl vyhovující, je možné výběr ovlivnit použitím anotace `@Order` nebo implementací `HandlerExceptionResolver`.

Nevýhodou je, že do metod označených `@ExceptionHandler` se standardně nepředává Model[11]. Pokud s ním potřebujeme pracovat (za účelem předání dat do šablony), musíme vytvořit instanci třídy `ModelAndView`.

Výpis 5.2 demonstruje použití anotace `@ExceptionHandler` na úrovni kontrolerů. [3]

```
class ExampleController{
    ...

    @ExceptionHandler
    public String errPage(Exception e){
        ModelAndView m = new ModelAndView("error-page-template");
        ...
        m.addObject("user", u);
        return m;
    }
}

class ExampleApiController{
    ...

    @ExceptionHandler
    public @ResponseBody ReturnObject handleCustomException( CustomException e){
        ...
    }

    @ExceptionHandler
    public @ResponseBody ReturnObject handleOtherException( Exception e){
        ...
    }
}
```

Výpis 5.2: Zpracování výjimek použitím anotace `@ExceptionHandler`. Ukázka obsahuje třídy dvou kontrolerů. Každý z nich definuje vlastní způsob zpracování (i stejných) výjimek.

[9] JSON – JavaScript Object Notation (formát pro serializaci strukturovaných dat vycházející ze syntaxe jazyka Javascript)

[10] Spring dokumentace https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/bind/annotation/ExceptionHandler.html

[11] Model – Objekt implementující datový model architektury MVC

Třída `ExampleController` demonstruje nutnost manuální inicializace modelu v případě, že jej chce programátor použít pro předání atributů do uživatelského rozhraní.

5.3 Implementace perzistentní vrstvy

Definice entity v JPA[12] se provádí vložením k tomu určených anotací do třídy, která má danou entitu (resp. tabulku v databázi) představovat. Aby Spring mohl vytvářet instance těchto tříd je třeba kromě anotace `@Entity`, uvést bezparametrický konstruktor. Použitím dalších anotací na členské proměnné či metody třídy lze objektově-relační mapování dále specifikovat. Takto je možné definovat například názvy databázových tabulek a sloupců (anotace `@Table`, `@Column`), integritní omezení (např. `@NotNull`, `@Max`) nebo i vztahy mezi entitami (`@OneToMany`, `@ManyToMany`, `@JoinColumn`). Výpis 5.3 znázorňuje definici entitní třídy.

```
@Entity
@Table(name = "cams")
public class Camera {
    @Id
    private int id;

    @Column(name = "limit")
    private short lim;

    @ManyToMany(targetEntity = Tag.class, mappedBy = "cams")
    private Set<Tag> tags;


        ... (standardni setter a getter metody)
}
```

Výpis 5.3: Příklad implementace entity ve Spring Data. Anotace `@Table` a @Column uvádí název odpovídající tabulky, resp. sloupce v databázi. Pokud nejsou uvedeny, automaticky se hledají názvy shodné s názvy třídy (resp. proměnné). Anotace `@ManyToMany` definuje mapování vztahu M:N mezi entitami Camera a Tag.

Databázové dotazy se v JPA definují prostřednictvím repozitářů. Repozitář je standardní rozhraní v jazyce Java, který rozšiřuje některé rozhraní objektově-relační vrstvy, typicky `JpaRepository` nebo `CrudRepository`). Každá deklarace metody v tomto rozhraní představuje konkrétní databázový dotaz. Jednoduché dotazy dokáže JPA odvodit ze signatury metody – tedy z názvu metody, návratového typu a datových typů parametrů. Například deklarace `String findNameById(int id)` představuje ekvivalent SQL dotazu `SELECT ‘name‘ FROM [table] t WHERE t.id = ’id’`. Složitější dotazy pak lze specifikovat například pomocí anotace `@Query` a dotazu v jazyce HQL[13] (viz výpis 5.4). Takový dotaz je zpravidla kratší a přehlednější než dotaz v SQL. Dotazování totiž není zaměřeno na strukturu databáze, ale na objekt představující danou entitu. To je pak příjemné zejména v případě, kdy se v dotazu vyskytují entity ve vztahu M:N, protože není potřeba vypisovat spojení přes spojovací tabulky.

[12] JPA – Java Persistence API (specifikace rozhraní pro jazyk Java umožňující objektově relační mapování)
[13] HQL – Hibernate Query Language (objektově orientovaný dotazovací jazyk, syntaxí podobný SQL)

```
@Query("SELECT k FROM Camera k JOIN k.tags t WHERE t.id = :id")
public List<Camera> findByTagsId(@Param("id") int tagid);
```

Výpis 5.4: Příklad dotazu na entity ve vztahu M:N definovaný pomocí jazyka HQL (na prvním řádku). Ač je jazyk HQL podobný jazyku SQL, umožňuje pohodlnější definici dotazu. Dotaz je zaměřen na strukturu objektu, ne databázové tabulky. V příkladu je vybírán objekt třídy (entity) `Camera` po spojení vztahem M:N s entitní množinou `Tag`.

Používání repozitáře (tedy práce s databází) je velmi intuitivní, spočívá na invokacích metod deklarovaných v repozitáři. Objekty reprezentující repozitáře Spring vytvoří automaticky a díky systému vkládání závislostí reference na tyto objekty vloží do třídních proměnných označených anotací `@Autowired`. Ukázka práce s repozitářem je ve výpisu 5.5.

```
@Service
class ExampleServiceClass{
    @Autowired
    private UserRepository users;

    public Set<User> getAdultUsers(){
        return users.findByAgeGreaterThanOrEqual(18);
    }

    ...
}
```

Výpis 5.5: Příklad použití repozitáře. Po spuštění programu Spring automaticky vytvoří instanci třídy `ExampleServiceClass` a také automaticky implementovaný objekt odvozený z rozhraní `UserRepository`. Reference na objekt repozitáře je poté injektována do pole users označeného anotací `@Autowired`. V metodách objektu třídy `ExampleServiceClass` je poté možné používat „dotazy“ definované v rozhraní `UserRepository`. Výsledek databázového dotazu je automaticky konvertován do objektu třídy `User`.

Implementace slabé entity v MySQL a Spring Data

Slabá entitní množina je existenčně závislá na silné entitní množině. Slabá entitní množina se vyznačuje tím, že její primární klíč je složený z primárního klíče silné entitní množiny (který je pro slabou entitu zároveň cizím klíčem), na které je závislá, a diskriminátoru (parciálního klíče), kterým se od sebe slabé entity odlišují. [28] V návrhu datového modelu aplikace se objevila slabá entitní množina „Oprávnění“, protože oprávnění sémanticky nemůže existovat samostatně, bez seznamu oprávnění.

Na straně databázového systému je záležitost poměrně přímočará. Ačkoli databázový systém MySQL nemá podporu pro generování identifikátorů (diskriminátorů) slabé entitní množiny, lze v něm definovat složený primární klíč. Generování diskriminátoru lze zařídit jednoduchým databázovým triggerem (viz výpis 5.6). Díky tomu při vkládání nového záznamu (v tomto případě pravidla oprávnění) není třeba uvádět hodnotu diskriminátoru, databáze ji automaticky doplní stejně jako běžný primární klíč. [1]

```
CREATE TRIGGER create_permission_key
BEFORE INSERT
ON permissions FOR EACH ROW
BEGIN
    DECLARE max_item INT DEFAULT 0;
/* Pri ukladani prvni polozky je vysledkem funkce MAX null.
  COALESCE vraci prvni hodnotu ze seznamu, ktera neni null. */
    SELECT COALESCE(MAX(item),0) INTO max_item
    FROM permissions WHERE ACL = NEW.ACL;
    SET NEW.item = max_item + 1;
END
```

Výpis 5.6: Kód MySQL triggeru pro generování identifikátoru

Na straně aplikace je situace složitější. Složený primární klíč musíme definovat jako zvláštní třídu obsahující atributy, které mají být do klíče zahrnuty. Potom máme dvě možnosti:

- Klíčové atributy stejným způsobem uvést i v třídě entity
- V třídě entity uvést pouze referenci na třídu primárního klíče

Já jsem zvolil první možnost protože lépe odpovídá způsobu práce s entitou v aplikaci. Ve třídě entity (v našem případě jde o třídu `Permission`) uvedeme identifikační údaje jako kombinaci vztahu N:1 a běžného celočíselného identifikátoru a oba tyto atributy označíme anotací `@Id` (ukázka je ve výpisu 5.7). Definice klíčových atributů musí být absolutně shodná v entitní třídě a ve třídě primárního klíče (v našem případě `PermissionId`). Jako poslední krok je potřeba entitní třídu propojit s klíčovou třídou pomocí anotace `@IdClass`. Obě třídy také musí implementovat rozhraní `Serializable`.

Poznámka: Vztah N:1 mezi slabou a silnou entitou je nutné definovat také na straně silné entity.

```
public class PermissionId implements Serializable{
        @Id
        @ManyToOne
        private ACL acl;

        @Id
        private int item;

        public PermissionId(){}
}

@Entity
@IdClass(PermissionId.class)
public class Permission implements Serializable{
        @Id
        @ManyToOne
        private ACL acl;

        @Id
```

```
        private int item;

    ...
}
```

Výpis 5.7: Implementace složeného primárního klíče entity v JPA
Klíčová třída (`PermissionId`) musí obsahovat stejný předpis klíčových atributů jako samotná entitní třída. Entitní třída nese anotaci `@IdClass`. Obě třídy musí implementovat rozhraní `Serializable`

Jelikož je entita identifikována klíčovou třídou `PermissionId` a ne pouze číselnou hodnotou, musí to být reflektováno i v repozitáři. Místo typu `Integer` bude parametr generického rozhraní `CrudRepository` (resp. `JpaRepository`) obsahovat klíčovou třídu `PermissionId`, jak je ilustrováno ve výpisu 5.8.

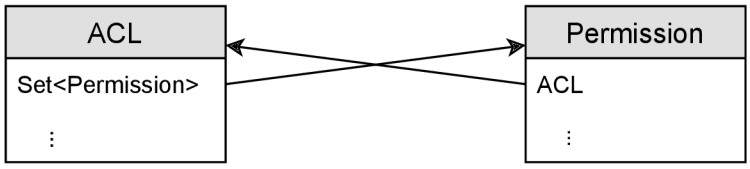
```
public interface PermissionRepository
  extends CrudRepository<Permission, PermissionId> {
    @Query("FROM Permission WHERE acl.id = :listid AND item = :itemd")
    public Permission findByKey(int listid, int itemd);
}
```

Výpis 5.8: Ukázka předpisu repozitáře pro slabou entitní množinu a možná varianta definice dotazu pro vyhledávání pomocí složeného klíče. Na místě typu identifikátoru v parametru generického rozhraní `CrudRepository` je specifikována třída `PermissionId`, která představuje složený primární klíč slabé entitní množiny `Permission`.

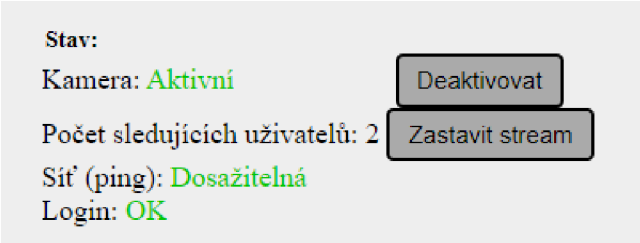
Ačkoli se zdá, že již jsou všechny problémy vyřešeny, celé řešení ještě není funkční. Problém nastává při ukládání nových instancí slabé entity – obousměrný vztah mezi silnou a slabou entitou musí namapovat programátor. Jelikož pracujeme s objekty, jsou vztahy mezi entitami vyjádřeny vzájemnými referencemi na objekty reprezentující danou entitu. Pro vysvětlující příklad je použita slabá a silná entita z aplikace. Slabou entitou bude oprávnění (`Permission`), silnou entitou seznam oprávnění (`ACL`). Vzájemná závislost entitních objektů je ilustrována na obrázku 5.2.

Při vytváření `Permission` musí být zadána reference na existující objekt `ACL` a reference na tento nový objekt `Permission` musí být zase vložena do seznamu oprávnění v objektu `ACL`. Z pohledu objektu `Permission` je situace jednoduchá. V případě editace existujícího seznamu oprávnění (`ACL`) referenci známe (objekt se již nachází v databázi), v případě vytváření nového použijeme referenci získanou voláním konstruktoru. Zajímavěji se situace jeví z pohledu seznamu oprávnění. Ve výchozím stavu silná entita (`ACL`) nepředpokládá, že má zodpovědnost za jiné entity, tedy předpokládá, že slabé entity (`Permission`), na něž má referenci již v databázi existují. Při mapování objektů `ACL` na relace (tedy při ukládání entity do DB) se ověřuje platnost vztahů. Perzistentní vrstva se pokouší vyhledat referencované entity, což skončí chybou a je konstruována výjimka `javax.persistence.EntityNotFoundException`. O zodpovědnosti za ukládání slabých entit můžeme silnou entitu (`ACL`) informovat nepovinným parametrem anotace pro mapování vztahu. Přidáním parametru `cascade = {CascadeType.ALL}` do anotace `@OneToMany` jsou zajištěny téměř všechny úkony správy slabé entity.

Stále však nefunguje odstraňování oprávnění (`Permission`) ze seznamu oprávnění (`ACL`), jelikož se nejedná o přímou úpravu silné entity. Anotace pro definici vztahu však poskytuje také parametr `orphanRemoval`, který zajistí odstranění entit, které ze silné entitní množiny



Obrázek 5.2: Vzájemná závislost objektů slabé (Permission) a silné (ACL) entity. Pro úspěšné uložení objektů v databázi je nutné, aby vzájemný vztah mezi entitami byl oboustraně namapován – tedy aby členská proměnná na straně `Permission` obsahovala platnou referenci na `ACL` a množina na straně `ACL` obsahovala referenci na tento objekt `Permission`.



Obrázek 5.3: Indikátor stavu kamery.

nejsou referencovány, z databáze. Kompletní definice vztahu 1:N ze strany silné entity (ACL) je uvedena ve výpisu 5.9.

```
@OneToMany(targetEntity=Permission.class, mappedBy="acl",
    cascade = {CascadeType.ALL}, orphanRemoval=true)
private List<Permission> items;
```

Výpis 5.9: Definice vztahu na straně silné entity. Pro zajištění správné funkcionality je zapotřebí nastavit nepovinné parametry `cascade` a `orphanRemoval` anotace definující vztah mezi slabou a silnou entitou

5.4 Kontrola spojení s kamerou

Pro spuštění přenosu je potřebné znát síťovou adresu, port na kterém kamera očekává RTSP spojení a přihlašovací údaje ke kameře. V případě, že by správce kamery některý z těchto údajů zadal chybně, stream by se nepodařilo spustit. Pokud by chybu sám nezaznamenal, na problém by narazil až uživatel. Za účelem předcházení těmto chybám byla do aplikace integrována minimalistická kontrola připojení a autentizace zadanými údaji. Výsledek je hned po uložení kamery indikován a zadavatel údajů může patřičně reagovat. Výstřižek obrazovky s indikátorem stavu kamery je zachycen na obrázku 5.3.

RTSP[14] [18] je textový protokol vycházející z HTTP. Mimo jiné používá stejné metody autentizace [6].

Zatím je implementována pouze autentizační metoda `Basic`. U této metody je jméno a heslo ve formátu `jmeno:heslo` pouze kódováno pomocí base64 a odesláno v hlavičce Authorization. Ve výpisu 5.10 je ukázka komunikace protokolem RTSP s autentizací Basic.

[14]RTSP – Real-Time Streaming Protocol

```
K: OPTIONS rtsp://10.0.1.25:554 RTSP/1.0
    CSeq: 1

S: RTSP/1.0 401 Unauthorized
    Server: Rtsp Server/2.0
    CSeq: 1
    WWW-Authenticate: Basic realm="RtspSever2.0"

K: OPTIONS rtsp://10.0.1.25:554 RTSP/1.0
    CSeq: 2
    Authorization: Basic YWRtaW46YWRtaW4=

S: RTSP/1.0 200 OK
    Server: Rtsp Server/2.0
    CSeq: 2
    Public: OPTIONS,DESCRIBE,SETUP,PLAY,PAUSE,TEARDOWN,SET_PARAMETER,
            GET_PARAMETER,ANNOUNCE
```

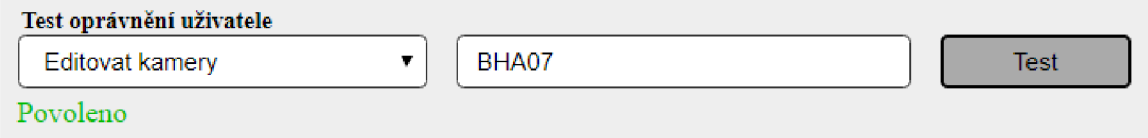
Výpis 5.10: Ukázka části komunikace protokolem RTSP zahrnující autentizaci. Označení „K:“ představuje zprávy od klienta (webové aplikace), označení „S:“ nesou zprávy od serveru (kamery). `CSeq` je sekvenční číslo identifikující výměnu klient - server. První zpráva (požadavek) je odeslána bez autentizace. V odpovědi kamery je uvedeno, že vyžaduje autentizaci metodou Basic. Dále všechny zprávy od klienta musí obsahovat hlavičku `Authorization` obsahující údaje pro autentizaci, jinak server opět odpoví zprávou s kódem 401 (Unauthorized).

5.5 Implementace oprávnění založených na lokalitě

Třída `User` reprezentující entitu uživatele v systému implementuje metodu `boolean isAuthorized(String typ_opravneni, String lokalita)` sloužící ke zjištění, zda má být danému uživateli umožněn, nebo zakázán požadovaný úkon.

Při zjišťování oprávnění uživatele dojde ke sloučení všech jemu přiřazených seznamů oprávnění a seřazení všech pravidel podle lokality od menších (které mají delší polohové kódy) po větší. Následně se tato seřazená pravidla prochází od prvního po poslední. První pravidlo, které má vyhovující typ a lokalitu, určí, zda bude přístup povolen, nebo zamítnut. Pokud se v seznamu žádné vyhovující pravidlo nenachází, je uživateli přístup k prostředku implicitně zakázán.

Pro transparentnost vyhodnocování přístupových oprávnění jsou na stránce editace uživatele vypsána pravidla výsledného seznamu oprávnění, který vznikne sloučením všech seznamů oprávnění uživatele a seřazením pravidel – ve stejné podobě, jaká se používá pro vyhodnocování oprávnění. Na jejich základě by mělo být možné ověřit, zda uživatel má požadovaná oprávnění. Protože ale takové „ruční“ vyhodnocení správcem může být v případě většího množství pravidel náročné a může být vyhodnoceno chybně, je na téže stránce k dispozici jednoduchý formulář, kterým lze oprávnění snadno ověřit. Na obrázku 5.4 je výstřižek snímku rozhraní zachycující tento formulář.



Obrázek 5.4: Ukázka formuláře pro rychlé ověření uživatelských oprávnění. Může být použit uživatelem nebo správcem přidělujícím oprávnění uživateli v případě podezření, že některé oprávnění nebylo nastaveno správně.

Konflikty oprávnění

Uživateli může být přiřazeno více seznamů oprávnění (např. vedoucí oddělení může mít na starosti více pracovišť, dočasné rozšíření pravomocí, přizpůsobeni oprávnění na míru konkrétnímu uživateli). To ilustruje obrázek 5.5. Pokud je uživateli přiřazeno více ACL (seznamů oprávnění), které stanovují pravidla oprávnění k překrývajícím se lokalitám, může dojít ke konfliktu těchto pravidel. To znamená, že ve výsledném ACL po sjednocení dílčích ACL se vyskytují alespoň dvě pravidla vztahující se ke stejné lokalitě a stejnému úkonu. Jsou-li všechna shodná pravidla povolující nebo zakazující, je vše v pořádku (přesto je však zobrazeno upozornění na hrozící konflikt). Problém nastává, pokud některé z těchto pravidel přístup povoluje a jiné ho zakazuje. V tom případě není možné deterministicky rozhodnout, které pravidlo bude aplikováno. Zjišťování chyb v oprávnění způsobených konflikty může být velmi problematické, zvláště potom v případě, že každý z těchto seznamů oprávnění danému uživateli přiřadil jiný správce. Proto je aplikace vybavena detekcí konfliktních pravidel – tyto jsou na stránce „Editace uživatele“ vizuálně zvýrazněny. Je v zájmu uživatele i správce, aby veškeré konflikty byly vyřešeny.

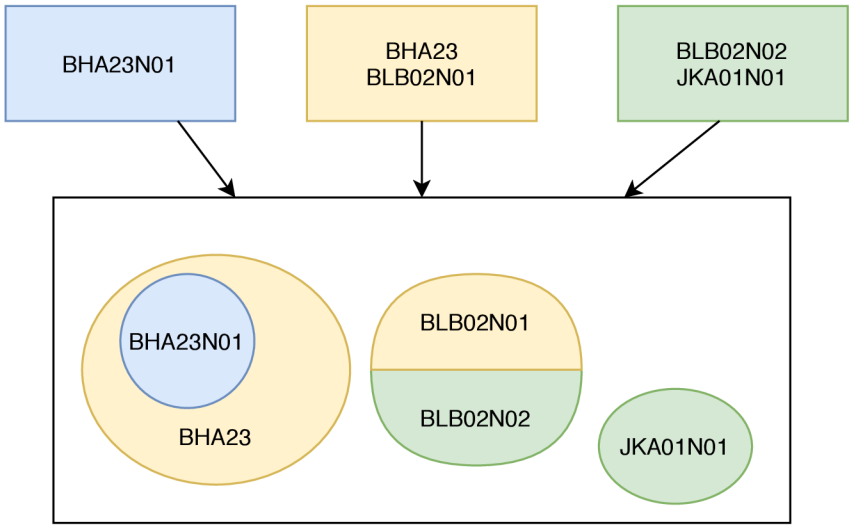
Této situace není třeba se obávat, pokud uživatel nepoužívá více seznamů oprávnění. V rámci jednoho seznamu oprávnění (ACL) totiž není možné konfliktní oprávnění zadat – to je ošetřeno už na úrovni databáze pomocí unikátního klíče.

Princip hledání všech objektů, ke kterým má uživatel přístupová práva

Ač se může zdát, že nalezení všech objektů, ke kterým byl uživateli přidělen přístup, je jednoduché, není tomu tak. Situace je komplikována tím, že oprávnění pracují s lokalitou, která pro aplikaci obyčejným řetězcem bez jakékoli sémantiky. Lokalita je v systému vyjádřena pouze řetězci ve formátu polohového kódu (o polohových kódech pojednává kapitola 3). Není v databázi reprezentována žádnou entitní množinou, z pohledu aplikace nemá známou hierarchii, žádné vazby na ostatní entity. Výběr množiny objektů v takovém systému zcela závisí na porovnávání řetězců lokalit, ve kterých se objekty nacházejí s řetězci lokalit v pravidlech oprávnění. Vybírány jsou objekty, jejichž polohový kód začíná polohovým kódem nadřazené lokality.

V případě „zanořených“ oprávnění, tedy oprávnění, která povolují nebo zakazují danou akci na různých úrovních stejné lokality (budova, podlaží, ...) je použito množinových operací – rozdíl za účelem vyloučení objektů, které vyhovují zakazujícím pravidlům a sjednocení pro zahrnutí povolených objektů. Pro ilustraci může posloužit příklad z vizualizace hierarchie lokalit pomocí množin na obrázku 4.3. V tabulce 5.1 jsou vypsána pravidla oprávnění odpovídající této vizualizaci.

Tato pravidla jsou nejdřív seřazena vzestupně podle délky řetězce označujícího lokalitu. Potom lze s jistotou prohlásit, že jsou zpracována nejdříve pravidla obecnější (s lokalitou



Obrázek 5.5: Ilustrační příklad uživatelských oprávnění zahrnující více zdrojových seznamů oprávnění (ACL). Horní řada představuje zdrojové ACL, spodní obdélník představuje výsledný seznam oprávnění po sloučení pravidel zdrojových ACL. Ve výsledném ACL jsou pravidla vizualizována jako množiny objektů, ke kterým (na základě lokality) definují oprávnění. Barevný kód slouží pro demonstrativní rozlišení, které (pod)množiny byly definovány kterým zdrojovým ACL.

Povoleno	Lokalita
0	BHA
1	BHA05
0	BHA05P
1	BHA05P01005
0	BHA05N02021
1	BHA09N03

Tabulka 5.1: Pravidla oprávnění ukázkového příkladu zachycující situaci ilustrovanou v obrázku 4.3. Předpokládejme, že tato pravidla jsou již vyfiltrována podle typu (tedy všechny definují oprávnění ke stejnému úkonu). Ve sloupci „Povoleno“ budou povolující pravidla označena kódem „1“, zakazující pravidla kódem „0“.

hierarchicky nadřazenou) a následně pravidla konkrétnější. Dále je vytvořen dvourozměrný seznam, ve kterém indexy vnějšího seznamu představují úrovně zanoření pravidel oprávnění, vnitřní seznamy budou obsahovat jednotlivá pravidla. Poté je seřazený zdrojový seznam procházen a jednotlivá pravidla jsou zařazována do výsledného dvourozměrného seznamu. Na sudých úrovních (indexech vnějšího seznamu) jsou pravidla povolující, na lichých pravidla zakazující. Cílem tohoto úkonu je:

1. Eliminovat nadbytečná pravidla – pravidla která nepřináší informaci o změně množiny (pravidlo na podřadné úrovni, které nese stejnou informaci o povolení/zákazu přístupu jako pravidlo jemu nadřazené), například zakazující pravidlo na nulté úrovni

2. Prozkoumat strukturu množiny objektů – identifikovat pravidla která z výsledné množiny odebírají podmnožiny objektů a která do ní objekty přidávají

3. Pravidla oprávnění seskupit tak, aby bylo možné je efektivně procházet a pohodlně tvořit výslednou množinu objektů sjednocováním a rozdílem dílčích množin

Zjednodušený pseudokód 5.11 demonstruje princip zařazování pravidel oprávnění do úrovní dvourozměrného seznamu. Výsledný stav dvourozměrného seznamu je prezentován v tabulce 5.2.

Nakonec je vytvořena výsledná množina, která bude obsahovat objekty, ke kterým má uživatel povolen přístup. Je iterováno přes indexy vnějšího seznamu dvourozměrného seznamu. V každé iteraci, ve které je hodnota indexu sudá, jsou vybrané objekty přidány do výsledné množiny, v iteraci s lichou hodnotou indexu jsou z výsledné množiny vybrané objekty odebrány. Z důvodu úspory výpočetního času je v této fázi výsledná množina tvořena pouze číselnými identifikátory entit v databázi; převod entit vybraných z databáze na objekty je až posledním krokem před návratem z metody.

0		1		2	
1	BHA05	0	BHA05N02021	1	BHA05P01005
1	BHA09N03	0	BHA05P		

Tabulka 5.2: Výsledná struktura pravidel. Sloupce tabulky vyjadřují úroveň zanoření a zároveň odpovídají indexům vnější úrovně dvourozměrného seznamu. Na sudých pozicích jsou situována povolující pravidla, na lichých pravidla zakazující. První pravidlo (pro lokalitu BHA) bylo vynecháno, protože nepřináší zaznamenání hodnou informaci (na nulté úrovni mají smysl jen povolující pravidla; na všechny objekty mimo tato povolující pravidla se vztahuje implicitní zákaz).

```
Vytvor prazdny dvourozmerny seznam
WHILE nejsou zpracovana vsechna pravidla
    Vyjmi prvni pravidlo ze zdrojoveho seznamu
    IF vybrane pravidlo je povolujici
        uroven_zanoreni_pravidla = 0
    ELSE
        uroven_zanoreni_pravidla = 1

    nalezena_uroven_zanoreni = false
    WHILE uroven_zanoreni_pravidla <= pocet_urovni_vysledneho_seznamu
        IF uroven_zanoreni_pravidla > 0
        AND pravidlo NENI PODRIZENO nekteremu pravidlu v predchozi urovni
            BREAK

        IF uroven_zanoreni_pravidla == pocet_urovni_vysledneho_seznamu
        OR pravidlo NENI PODRIZENO nekteremu pravidlu v teto urovni
            nalezena_uroven_zanoreni = true
            BREAK

        Inkrementuj uroven_zanoreni_pravidla o 2

    IF nalezena_uroven_zanoreni
        IF uroven_zanoreni_pravidla == pocet_urovni_vysledneho_seznamu
            Zvys pocet_urovni_vysledneho_seznamu
        Zarad pravidlo na uroven_zanoreni_pravidla do vysledneho seznamu
```

Výpis 5.11: Pseudokód algoritmu pro nalezení úrovně zanoření pravidla oprávnění v dvourozměrném seznamu. Nevyhovující pravidla jsou zahozena.

5.6 Náhledové obrázky z kamer v aplikaci

Lidé většinu informací přijímají zrakem. Vizuální prvky hrají důležitou roli v reálném i virtuálním prostoru, protože uživatelům pomáhají v orientaci; za přítomnosti vizuálních prvků je možné se v rozhraní zorientovat mnohem rychleji, než za přítomnosti pouze textových popisků. [27] Proto budou v aplikaci zobrazovány náhledové obrázky.

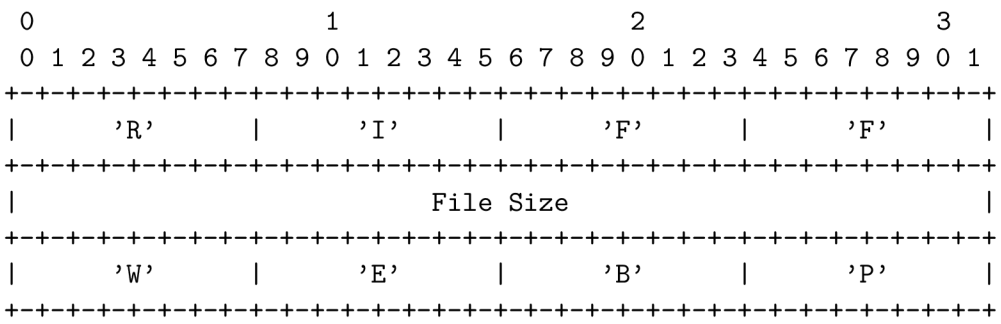
V roce 2010 firma Google vydala formát WebP, moderní formát pro obrázky. WebP je formát založený na multimediálním kontejneru (typ metasouboru pro ukládání metadat

společně s proudy kódovaných mediálních dat) RIFF[15]. Může obsahovat statické obrázky o velikosti až 16383 pixelů v obou rozměrech, ve ztrátové i bezeztrátové kompresi, ale i animace, včetně podpory průhlednosti. Hlavní výhodou oproti jiným obrazovým formátům je menší velikost souboru (podle webu projektu o 26 % oproti PNG [8] [16]), což z něj dělá atraktivní formát pro použití na webu. Tyto údaje, deklarované na webu projektu, odpovídají i reálným zkušenostem vývojářů. V řadě případů se daří dosáhnout snížení velikosti obrázků až na polovinu [12].

Aktualizace náhledového obrázku kamery

Stream-m již má zabudovanou podporu pro poskytování WebP obrázků. Stará se o něj třída `SnapshotResource` standardně připojený ke koncovému bodu `/snapshot/[nazev streamu]`. Obrázek je tvořen samonosným snímkem (nezávislým na znalosti předchozích či následujících snímků) kódovaném ve formátu VP8, stejně jako klíčový snímek („key frame“) videa formátu WebM. Protože je pro účely streamování ukládán vždy aktuální klíčový snímek, jistě bylo pro autora vytvoření třídy `SnapshotResource`, která dokáže data klíčového snímku získat a vložená do mediálního kontejneru odeslat klientovi, přímočaré.

Struktura kontejneru WebM je následující (převzato z [8]):



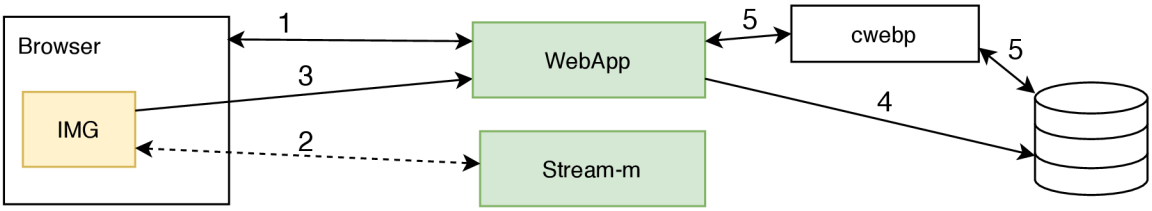
Za touto hlavičkou již následují části „chunk“, obsahující označení typu a samotná data. V případě použití varianty „jednoduchého formátu“ (Simple file format) následuje čtveřice bajtů obsahující ASCII znaky „`VP8 `“ a obrazová data. Pro sestavení obrázkového souboru stačilo zjistit rozměry obrázku, tyto doplnit do hlavičky a celou hlavičku předřadit kódovaným obrazovým datům.

Při zkoušení načtení obrázku z tohoto zdroje jsem ovšem narazil na problém – obrázek se sice načetl, ale nezobrazoval. Po analýze jsem zjistil, že v třídě `SnapshotResource`, nebyla ošetřena endianita dat vkládaných do hlavičky. Bajty kódující velikost obrázku nebyly uloženy ve formátu little-endian, jak definuje formát kontejneru WebP, ale v opačném pořadí. Po provedení opravy už zobrazování fungovalo, i když ne zcela spolehlivě – někdy se obrázek stále nezobrazoval. I proto jsem způsob ukládání náhledu navrhl způsobem popsaným dále.

Aktualizace náhledu probíhá následovně:

- Klientská část aplikace získá cestu (URI) ke zdroji stream serveru poskytujícímu obrázky. V případě potřeby je spuštěn stream z dané kamery.
- Uživatel opakovaně načítá obrázek, dokud není spokojen s výsledkem – obrázek se stahuje do prohlížeče klienta

[15] RIFF – Resource Interchange File Format (souborový formát pro výměnu mediálních prostředků vytvořený firmou Microsoft)



Obrázek 5.6: Princip uložení náhledového obrázku. 1) Nejdřív je z klientské části vyslán požadavek na server. Webová aplikace, pokud již neběží, zařídí spuštění streamu z požadované kamery a v odpovědi odešle URI tohoto streamu. 2) Uživatel opakovaně načítá aktuální náhled kamery z URI získané v bodě 1. 3) Jakmile je uživatel spokojen s výsledkem, odešle výsledný obrázek na server. 4) Obrázek je webovou aplikací uložen do souboru. 5) Pokud je uložený obrázek ve formátu PNG, vyvolá aplikace konverzi programem `cwebp` do formátu WebP.

- Obrázek je nahrán na server a serverovou částí webové aplikace uložen do souboru.

Princip aktualizace náhledových obrázků je ilustrován na obrázku 5.6.

V klientské části je k účelům uchování načteného obrázku použit element `canvas`. Jakmile má být obrázek nahrán na server, pomocí metody toDataURL() je získán obrázek ve formě řetězce, zakódovaný pomocí kódování base64. V této formě je odeslaný na server. Problém je v tom, že do WebP obrázky kóduje momentálně pouze prohlížeč Chrome; Firefox a Edge vrátí data ve formátu PNG. Původně jsem zamýšlel toto řešit ukládáním souboru bez přípony. Pro prohlížeč to není problém – ten by poznal že jde o obrázek ve formátu WebP, resp. PNG podle informací v hlavičce souboru. Nakonec jsem se rozhodl implementovat překódování do WebP, protože poměr velikostí souborů je skutečně dramatický. Porovnání velikostí na vlastní sadě zkušebních obrázků ukázalo, že soubory ve formátu PNG jsou několikanásobně větší, než soubory WebP. U obrázků z kamer získaných v reálném prostředí se velikosti PNG souborů pohybovaly kolem 230 kB, oproti tomu stejný obrázek ve formátu WebP měl necelých 20 kB. K překódování obrázků je použit program `cwebp`.

5.7 Konfigurace reverse proxy serveru

V kapitole 4 byl popsán význam proxy serveru pro navrhovaný systém včetně uvedení modulů, které je třeba nakonfigurovat. Veškeré nastavení je možné provést v jednom standardním souboru pro konfiguraci webu. Tento soubor bude umístěn v obvyklém adresáři[16] pro tento typ souborů – `/etc/apache2/sites-available`. Pro aplikaci nastavených vlastností je třeba tuto konfiguraci označit jako aktivní příkazem `a2ensite [nazev konfiguracniho souboru]`, který především vytvoří symbolický odkaz na tento soubor v adresáři `/etc/apache2/sites-enabled`, a následně vynutit načtení konfigurace příkazem `systemctl reload apache2`.

Základním úkolem je sjednocení rozhraní pro webovou aplikaci a stream server. To vyžaduje nastavení cest, které budou sloužit jako endpointy (koncové body) pro tyto programy. Veškeré HTTP požadavky, které budou vyhovovat těmto cestám, budou přeposlány odpovídající aplikaci, odpovědi na tyto požadavky budou stejnou cestou poslány zpět. Při požadavku na Stream-m bude k přeposílání požadavků použito protokolu HTTP, v případě

[16] Uvedené cesty platí pro OS Ubuntu, v jiných systémech mohou být odlišné

požadavku na webovou aplikaci se použije binární protokol AJP[17]. Při konfiguraci endpointů je třeba si uvědomit, že u těchto direktiv záleží na pořadí. Pro dosažení správného efektu je potřeba zadat nejdříve konkrétnější endpointy a pokračovat k obecnějším. Pravidla se prochází postupně a první vyhovující pravidlo procházení zastaví. Pro Stream-m server stačí vyhradit dva endpointy – pro video (URI začínající `/streams`) a náhledové obrázky (`/snapshot`). Veškeré ostatní požadavky budeme směrovat do webové aplikace, proto je pro ni použito obecné pravidlo `/`. Použití direktiv pro reverzní proxy ilustruje výpis 5.12.

```
# Endpoints for stream server
ProxyPass /streams/ http://localhost:9874/streams/
ProxyPass /snapshot/ http://localhost:9874/snapshot/

# Endpoint for web application
ProxyPass / ajp://localhost:8009/
```

Výpis 5.12: Ukázka použití direktiv pro reverzní proxy. Aby byla zvolena správná cesta je třeba uvést nejdříve konkrétnější pravidla, potom obecnější.

SSO Autentizace

Webová aplikace je prostřednictvím proxy serveru připojena k systému jednotného přihlášení (SSO[18]). V případě programu Apache je toto realizováno pomocí modulu `mod_auth_oidc`[19]. Pro zprovoznění SSO autentizace stačí nastavení několika konfiguračních direktiv (například) v souboru s konfigurací webové aplikace.

Jednou z dostupných možností konfigurace je direktiva `OIDCRemoteUserClaim`. Ta určuje, který tzv. claim[20] bude aplikaci předán pomocí systémové proměnné `REMOTE_USER` (tato proměnná obvykle obsahuje přihlašovací jméno uživatele z autentizace protokolem HTTP, pokud byla použita). Takto lze nastavit předávání libovolného identifikátoru (nebo jiného údaje) uživatele nabízeného poskytovatelem identit. V prostředí Masarykovy univerzity (MU) si aplikace vystačí se standardním identifikátorem s klíčem „sub“. Tento claim obsahuje hodnotu ve tvaru `UCO@muni.cz`, kde `UCO` je univerzitní číslo osoby, jednoznačný identifikátor uživatele používaný aplikacemi napříč MU. V implementované aplikaci jsou uživatelé identifikováni pouze částí `UCO`, zbývající část identifikátoru „sub“ není ukládána.

Webová aplikace s proxy serverem komunikuje pomocí binárního protokolu AJP. K tomuto účelu je na straně webové aplikace potřeba provést nastavení komunikačního rozhraní. Protože je při překladu jako součást aplikace integrován webový server Tomcat, je toto rozhraní ve formě tzv. konektoru, reprezentovaného objektem typu Connector[21]. Ve výchozím nastavení je v konektoru nastavena vlastnost `tomcatAuthentication`, která způsobuje, že Tomcat se pokouší zajistit provádění autentizace vlastními prostředky. Kvůli tomu v aplikaci není dostupná hodnota `REMOTE_USER`, nastavená na straně proxy serveru. Pro použití externí autentizace je nutné kromě obvyklých nastavení (např. číslo portu, verze protokolu AJP) zakázat zajišťování autentizace serverem Tomcat. Potom je možné hodnotu `REMOTE_USER` v kontroleru získat voláním metody `getRemoteUser()` nad objektem `HttpServletRequest`. Výpis 5.13 obsahuje ukázku zakázání autentizace vestavěným serverem Tomcat.

[17] AJP – Apache JServ Protocol
[18] SSO – Single Sign On (Systém jednotného přihlášení)
[19] Stránka OpenId Connect modulu pro Apache httpd – https://github.com/zmartzone/mod_auth_openidc/
[20] Claim – dvojice klíč-hodnota obsahující informaci o uživateli [4]
[21] Connector – http://tomcat.apache.org/tomcat-7.0-doc/config/ajp.html

```
Connector connector = new Connector("AJP/1.3");
    ...
connector.setAttribute("tomcatAuthentication", false);
```

Výpis 5.13: Ukázka zakázání autentizace integrovaným serverem Tomcat

Kapitola 6

Závěr

Tato bakalářská práce měla za cíl zpřístupnit aktuální obraz z IP kamerového systému vybraným uživatelům prostřednictvím internetového prohlížeče a tak snížit úsilí nutné pro zprovoznění a údržbu klientských stanic. Tohoto cíle bylo dosaženo s využitím volně dostupných programů FFmpeg a Stream-m a vytvořením uživatelské nadstavby v podobě webové aplikace.

Vytvořená aplikace využívá značky `<video>` mladého standardu HTML5. Pomocí programu FFmpeg je video z kamery překódováno do otevřeného formátu WebM, program Stream-m tato obrazová data pomocí protokolu HTTP(S) distribuuje do prohlížečů uživatelů. Z důvodu usnadnění implementace komunikačního rozhraní mezi serverovou částí webové aplikace a programem Stream-m byl použit jazyk Java rozšířený o framework Spring. Tato aplikace kromě sledování kamer umožňuje uživatelům správu údajů o kamerách a přístupových práv uživatelů. Autentizace je zajištěna napojením na systém jednotného přihlášení (SSO) pomocí technologie OpenId Connect.

Výsledný systém nabízí otevřené řešení pro sledování kamer, které lze snadno nasadit ve firemním prostředí. Nabízí se propojení se systémy pro správu budov (BMS). Protože se zároveň s konverzí formátu provádí zmenšení obrazu (konkrétní velikost je možné nastavit v konfiguraci), přináší systém znatelnou redukci datového toku v síti způsobeného sledováním kamer – zvláště, nachází-li se v síťové topologii blízko zdroji (kamerám). Komunikaci s kamerou a dekódování obrazu v plné velikosti provádí pouze aplikační server, takže na stroje klientů je vyvíjena minimální zátěž a je možné přenos sledovat i na stroji disponujícím malým výpočetním výkonem (například mobilní zařízení) a také je sníženo riziko zahlcení kamer. Tyto vlastnosti jsou tím markantnější, čím více uživatelů sleduje danou kameru. Jelikož na straně uživatele nejsou ukládány přístupové údaje ke kamerám, je minimalizováno riziko bezpečnostních incidentů získání těchto údajů neoprávněnou osobou.

Možným směrem dalšího vývoje je rozšíření uživatelského rozhraní o konfigurovatelnou obrazovku, na které bude možné v jednom okně (resp. panelu) prohlížeče současně sledovat více kamer a o vyhledávací jazyk, aby bylo možné výsledky filtrovat podle více kritérií. Dále by bylo možné práci doplnit zejména o distribuci konverze videa na více strojů a aplikovat mechanizmy rozložení zátěže (load blancing) a zálohy při výpadku hlavního stroje (failover).

Literatura

[1] Balasubramaniam, V. *Composite Primary Keys in JPA* [online]. Květen 2019 [cit. 2020-04-03]. Dostupné z: https://www.baeldung.com/jpa-composite-primary-keys.

[2] Cankaya, H. C. Access Control Lists. In: Tilborg, H. C. A. van a Jajodia, S., ed. *Encyclopedia of Cryptography and Security* [online]. Springer US, 2011, s. 9–12 [cit. 2020-06-01]. ISBN 978-1-4419-5906-5. Dostupné z: https://doi.org/10.1007/978-1-4419-5906-5_770.

[3] Chapman, P. *Exception Handling in Spring MVC* [online]. Listopad 2013 [cit. 2020-04-15]. Dostupné z: https://spring.io/blog/2013/11/01/exception-handling-in-spring-mvc.

[4] David. Understanding Claims. *Stack Overflow* [online]. Odpověděl Jordan Stewart. Srpen 2018 [cit. 2020-04-03]. Dostupné z: https://stackoverflow.com/questions/37067938/understanding-claims.

[5] Faulkner, S., Eicholz, A., Leithead, T., Danilo, A. a Moon, S. *HTML 5.2* [online]. Prosinec 2017 [cit. 2020-05-06]. Dostupné z: https://www.w3.org/TR/html52.

[6] Fielding, R. a Reschke, J. *Hypertext Transfer Protocol (HTTP/1.1): Authentication* [online]. Červen 2014 [cit. 2020-06-08]. DOI: 10.17487/RFC7235. Dostupné z: https://tools.ietf.org/html/rfc7235.

[7] Google. *NPAPI Deprecation: Developer Guide - The Chromium Projects* [online]. [cit. 2020-05-05]. Dostupné z: https://www.chromium.org/developers/npapi-deprecation.

[8] Google. *A New Image Format for the Web* [online]. 2019 [cit. 2020-04-09]. Dostupné z: https://developers.google.com/speed/webp.

[9] Hoffman, C. What ActiveX Controls Are and Why They're Dangerous. *How-To Geek* [online]. Květen 2013, [cit. 2020-04-15]. Dostupné z: https://www.howtogeek.com/162282/what-activex-controls-are-and-why-theyre-dangerous/.

[10] Kesteren, A. van. *Cross-Origin Resource Sharing* [online]. Leden 2014 [cit. 2020-05-16]. Dostupné z: https://www.w3.org/TR/2014/REC-cors-20140116/.

[11] Laforge, A. *Saying Goodbye to Flash in Chrome* [online]. Červenec 2017 [cit. 2020-05-12]. Dostupné z: https://www.blog.google/products/chrome/saying-goodbye-flash-chrome/.

[12] Michálek, M. *WebP obrázky: datově úsporná alternativa k JPEG, PNG i GIF* [online]. Červen 2018 [cit. 2020-04-09]. Dostupné z: https://www.vzhurudolu.cz/prirucka/webp.

[13] Minnick, J. *Web Design with HTML5 and CSS3, Comprehensive*. 8. vyd. Cengage Learning, 2016. ISBN 9781305578166.

[14] OFM SUKB. *Metodika technologické pasportizace MU*. Oddělení facility managementu Správy Univerzitního kampusu Bohunice, Masarykova Univerzita, červen 2019.

[15] OpenID. *OpenID Connect* [online]. The OpenID Foundation [cit. 2020-05-01]. Dostupné z: https://openid.net/connect/.

[16] Osmani, A. *Essential Image Optimization* [online]. 2018 [cit. 2020-04-09]. Dostupné z: https://images.guide/.

[17] Raggett, D., Le Hors, A. a Jacobs, I. *HTML 4.01 Specification* [online]. Prosinec 1999 [cit. 2020-06-10]. Dostupné z: https://www.w3.org/TR/html401/.

[18] Rao, A., Lanphier, R., Stiemerling, M., Schulzrinne, H. a Westerlund, M. *Real-Time Streaming Protocol Version 2.0* [online]. Prosinec 2016 [cit. 2020-04-03]. DOI: 10.17487/RFC7826. Dostupné z: https://tools.ietf.org/html/rfc7826.

[19] Schuh, J. *Saying Goodbye to Our Old Friend NPAPI* [online]. Září 2013 [cit. 2020-05-05]. Dostupné z: https://blog.chromium.org/2013/09/saying-goodbye-to-our-old-friend-npapi.html.

[20] Schuh, J. *The Final Countdown for NPAPI* [online]. Listopad 2014 [cit. 2020-05-05]. Dostupné z: https://blog.chromium.org/2014/11/the-final-countdown-for-npapi.html.

[21] Schulzrinne, H. *RTP: A Transport Protocol for Real-Time Applications* [online]. Červenec 2003 [cit. 2020-05-13]. DOI: 10.17487/RFC3550. Dostupné z: https://tools.ietf.org/html/rfc3550.

[22] Sharrab, Y. a Sarhan, N. Detailed Comparative Analysis of VP8 and H.264. In: *Multimedia (ISM), 2012 IEEE International Symposium*. Prosinec 2012. DOI: 10.1109/ISM.2012.33. ISBN 978-1-4673-4370-1. Dostupné z: https://www.researchgate.net/publication/261074990_Detailed_Comparative_Analysis_of_VP8_and_H264.

[23] Smedberg, B. *Plugin Activation in Firefox* [online]. Září 2013 [cit. 2020-05-19]. Dostupné z: https://blog.mozilla.org/futurereleases/2013/09/24/plugin-activation-in-firefox/.

[24] SPAWAR. *CCTV Technology Handbook* [online]. Space and Naval Warfare Systems Center Atlantic, červenec 2013 [cit. 2020-06-01]. Dostupné z: https://www.dhs.gov/sites/default/files/publications/CCTV-Tech-HBK_0713-508.pdf.

[25] WebM. *The WebM Project* [online]. [cit. 2020-05-26]. Dostupné z: https://www.webmproject.org.

[26] Wiki W3. *Same Origin Policy - Web Security* [online]. 2009 [cit. 2020-04-07]. Dostupné z: https://www.w3.org/Security/wiki/Same_Origin_Policy.

[27] Zemčík, P. *Tvorba uživatelských rozhraní – Studijní opora.* Fakulta informačních technologií, Vysoké učení technické v Brně, listopad 2006.

[28] Zendulka, J. a Rudolfová, I. *Databázové systémy – Studijní opora.* Fakulta informačních technologií, Vysoké učení technické v Brně, únor 2016.

Příloha A

Obsah média

- **./pomocne_programy** — zdrojové kódy programů FFmpeg a cwebp používaných aplikací
- **./prelozene_casti** — přeložená webová aplikace (webCC) a stream server (Stream-m)
- **./snimky_obrazovky** — ukázkové snímky obrazovky aplikace
- **./text_prace** — zdrojové kódy LaTeX, vygenerované PDF práce, obrázky
- **./zdrojove_kody** — zdrojové kódy aplikace (webCC) a stream serveru (Stream-m), návod k překladu a instalaci

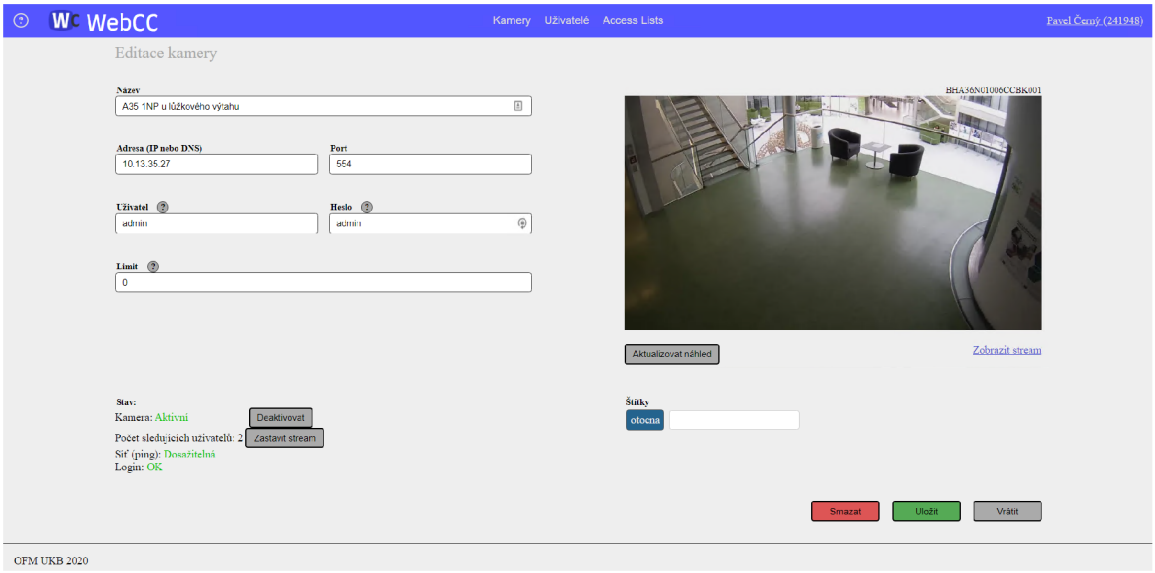
Příloha B

Diagram případů užití

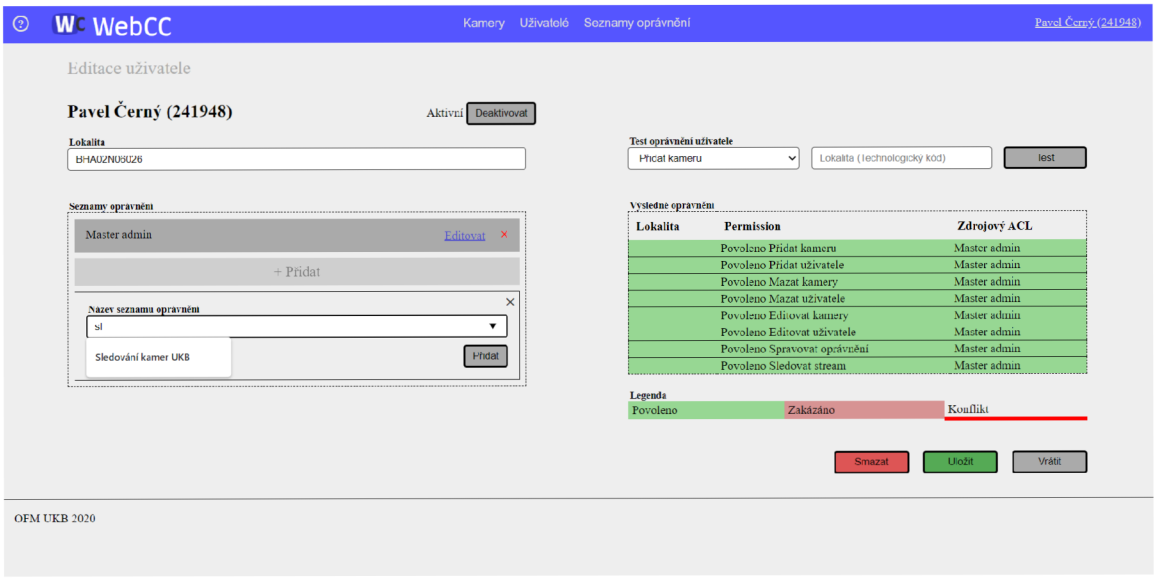
Obrázek B.1:

Příloha C

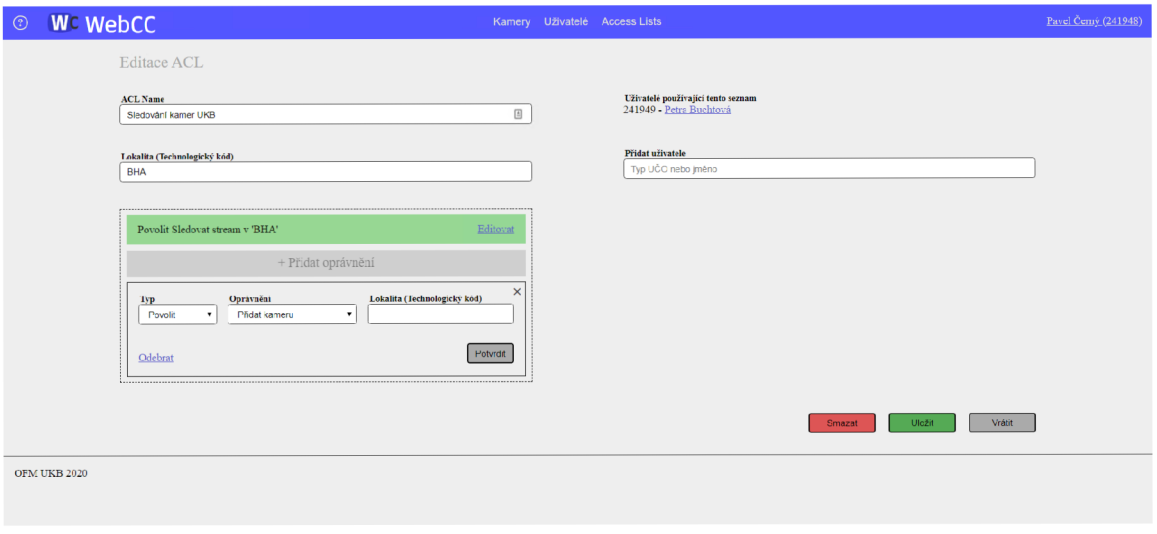
Snímky obrazovek webové aplikace



Obrázek C.1: Náhled uživatelského rozhraní pro editaci kamery.



Obrázek C.2: Náhled uživatelského rozhraní pro editaci uživatele.



Obrázek C.3: Náhled uživatelského rozhraní pro editaci seznamu oprávnění.