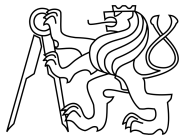
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Webová aplikace pro videokonference
Student:	Bc. Jan Havlíček
Vedoucí:	Ing. Martin Balík, Ph.D.
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2016/17

Pokyny pro vypracování

Analyzujte existující řešení pro realizaci videokonference a jejich vlastnosti. Zaměřte se na desktopová i webová řešení a porovnejte je.
Navrhněte a implementujte webovou aplikaci na platformě JavaEE pro videokonferenci, která bude splňovat následující požadavky:
1) Sdílení prezentace ve formátu pptx, pdf a synchronizace průchodu slajdů s prezentujícím.
2) Duplexní přenos zvuku, možnost zapnutí/vypnutí u posluchačů.
3) Přenos obrazu videokamery, prozkoumejte možnosti zobrazení obrazu v části prezentace a zobrazení na celou obrazovku.
4) Možnost zapojení více posluchačů a související podpora přenosu více obrazů a zvuku.
Pro implementaci zvolte vhodné technické řešení a frameworky.
Aplikaci otestujte z hlediska výkonu, závislosti na propustnosti sítě i z hlediska kvality uživatelského rozhraní a použitelnosti.

Seznam odborné literatury

Dodá vedoucí práce.

L.S.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 15. ledna 2016

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra softwarového inženýrství

Diplomová práce

Webová aplikace pro videokonference

Bc. Jan Havlíček

Vedoucí práce: Ing. Martin Balík, Ph.D.

17. února 2017

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 17. února 2017 .....

Odkaz na tuto práci

Havlíček, Jan. *Webová aplikace pro videokonference.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2017.
Dostupný také z WWW: ⟨`https://glassqa.felk.cvut.cz:10081/CTUConference/`⟩.

Abstrakt

V rámci této práce byla navržena a implementována webová aplikace pro videokonference, která umožňuje vytvářet skupinové hovory nebo přednášky s přednášejícím a posluchači. Využívá k tomu novou webovou technologii WebRTC a Kurento media server, který tuto technologii podporuje. Aplikace poskytuje možnost přidání slajdů a synchronizovaný průchod slajdy v průběhu přednášky. Implementuje také různé způsoby zobrazení přenášených audiovizuálních streamů a slajdů včetně možnosti zobrazení na celou obrazovku nebo adaptace rozvržení příchozích audiovizuálních streamů během skupinového hovoru na základě toho, který účastník právě hovoří. Přínosem této práce by mělo být také zhodnocení technologických možností v oblasti přenosu audiovizuálních dat. Popisuje jednotlivé protokoly a nástroje, které umožňují přistupovat ke kameře a mikrofonu uživatele a které umožňují komunikaci v reálném čase včetně audiovizuální komunikace. Součástí je i zhodnocení, jakým způsobem lze testovat měřitelné parmetry audiovizuálního přenosu dat, tak aby bylo možné zhodnotit například adaptibilitu na propustnost sítě.

Klíčová slova videokonference, synchronizace slajdů, media server, WebRTC, Kurento, Java, Spring, TypeScript, Angular 2, diplomová závěrečná práce

Abstract

The main goal of the thesis is to design and implement the videoconference application, that enables to create group calls or lectures with the lector and multiple listeners. It uses new web technology called WebRTC with conjunction of Kurento media server. The application supports slideshow during the call and synchronization of listener's slides with the lector's one. It also implements various layouts of video streams and slides that the user can choose from, including full-screen mode and the layout for group call which can adapt to the currently speaking user. This work also aimed to discussion new web technologies and tools that enable to access to web camera and microphone devices and that offer support of real-time communication, especially audiovisual communication. One part of this paper focuses to discussion and testing, how it is possible to measure the quality of media transmission and the adaptability to network throughput.

Keywords videoconference, slides synchronization, media server, WebRTC, Kurento, Java, Spring, TypeScript, Angular 2, master's thesis

Obsah

Seznam obrázků

Úvod

Cílem této práce je zhodnocení technologických možnosti pro tvorbu interaktivní webové aplikace, jejímž primárním cílem by bylo zprostředkování audiovizuálních hovorů s podporou zobrazení a synchronizace slajdů v průběhu přenosu, a dále návrh a implementace takové aplikace.

V úvodu je třeba si říci, co vlastně od takové aplikace očekáváme, to znamená jaké případy použití by měla tato aplikace podporovat. Dále je třeba zhodnotit výhody a nevýhody současných řešení.

Případy užití

Aplikace budeme chtít zaměřit především na vzdělávací účely, tak aby bylo možné přenášet hovory včetně synchronizovaných slajdů. Zároveň by aplikace mohla umožňovat skupinové hovory pro podporu různých výzkumných skupin. V závislosti na použití by měl mít uživatel možnost nastavit si zobrazení tak, aby co nejlépe vyhovovalo přednášejícímu i posluchači. Požadavky jsou podrobněji popsány v následující kapitole.

Přednášky a webináře

Jedním z možných použití je online přenos přednášek na univerzitě nebo přenos různých webinářů. Zde je běžně jeden nebo dva přednášející, popřípadě někdo, kdo ovládá kameru. Ostatní účastníci jsou pouze posluchači, to znamená, že se od nich neočekává žádný audiovizuální přenos.

Můžeme uvažovat dvě situace. Jednodušší situace nastává v případě, kdy má přednášející k počítači, na kterém prezentuje, zároveň připojenou kameru. V tomto případě předpokládáme, že má přednášející možnost přistpoupit k této aplikaci a sdílet nejen slajdy, ale také audiovizuální přenos.

Složitější situace může nastat v případě přednášek a videokonferencí, které se konají fyzicky v nějaké přednáškové místnosti, pak záleží na infrastuktuře dané přednáškové místnosti, ale běžně se můžeme také setkat se situací, kdy

kameraman natáčí video nezávisle na přednášejícím a musí otáčet kameru buď na přednášejícího nebo na slajdy, podle toho, co je v dané chvíli důležitější. Budeme uvažovat, že kameraman má možnost vždy připojit kameru k internetu a vysílat tak audiovizuální data v reálném čase.

Technologické srazy

Pomineme-li placené kurzy, které pořádají různé firmy, lze v poslední době zaznamenat také rozmach srazů nadšenců různých technologií, minimálně v oblasti informačních technologií. Pořádání srazů je většinou komunitní záležitostí, přičemž prostory a technické zázemí poskytují zavedené i menší firmy, které tak mají možnost se zviditelnit. Infrastruktura v těchto případech se může skládat jednoduše z notebooku přednášejícího a kamery, kterou může mít připojenou přímo k notebooku nebo jinak zapojenou tak, aby mohla vysílat stream do internetu.

Firemní nebo skupinové meetingy

Další možným případem užití mohou být skupinové hovory (audio nebo video hovory) v rámci firemní skupiny, pokud firma působí na více místech, nebo výzkumné skupiny spolupracující napříč fakultami nebo univerzitami. V tomto případě všichni účastníci hovoru vysílají i přijímají audiovizuální signál.

Kapitola 1

Existující konferenční aplikace z hlediska použitelnosti

V této kapitole jsou shrnuty základní rysy některých aplikací, které lze využít k vytvoření skupinových hovorů nebo webinářů v prostředí webových prohlížečů.

1.1 Popis jednotlivých aplikací

1.1.1 Skype

Když se řekne videokonferenční aplikace, Skype napadne asi každého. Původně šlo pouze o desktopovou aplikaci. Pro přenos audiovizuálních dat včetně adaptace na propustnost sítě implementuje Skype vlastní řešení. Donedávna to byla jediná možnost, jak těchto výsledků dosáhnout a pokud jde o kvalitu nabízeného audiovizuálního přenosu, pak je Skype opravdu funkčním řešením, především z důvodu, že se zpracováním audiovizuálních dat zabývají již velmi dlouho.

V poslední době je dostupná také služba „Skype for web“ s webovým rozhraním. Pokud ale chceme vyvolat audio nebo video hovor, jsme požádáni o instalaci plug-inu do prohlížeče a to jak v Internet Exploreru, který neposkytuje novou technologii WebRTC umožňující implementaci bez doplňků do prohlížeče, tak v prohlížeči Firefox a Chrome, ve kterém je již WebRTC ve verzi 1.0 implementováno.

Pokud jde o funkce Skypu, zaměřuje se především na skupinové hovory. Neumožňuje však vytvářet semináře s posluchači ani možnost přidávání slajdů. Umožňuje však sdílení obrazovky s ostatními uživateli.

1.1.2 GoToWebinar

GoToWebinar je služba, která umožňuje pořádat webové semináře nebo se k nějakému webovému semináři přihlásit. Z pohledu posluchače je příjemné,

že není potřeba registrace, pokud to uživatel, který webinář pořádá, umožní. Zájemci o webinář přijde na e-mail přihláška s odkazem, pomocí kterého je následně autorizován k přístupu webináře. Po autorizaci je však nutné stáhnout si aplikaci, ve které se následně seminář přehrává. Nejde tedy o webové řešení, ale je třeba desktopová aplikace pro samotné přehrávání. Uživatel, který vytváří videokonferenci, má možnost nastavit limit pro počet přihlášených uživatelů. Aplikace umožňuje během přednášky připojit více přednášejících, popřípadě moderátora, který přednášku uvede. Umožňuje zobrazovat malé náhledy přednášejících. Dále umožňuje sdílení obrazovek. To lze použít především, pokud jde o seminář, kde přednášející programuje nebo seznamuje posluchače s různými nástroji. Posluchači je zároveň během hovoru zobrazena navigační lišta (aplikace tak nezabírá celou plochu obrazovky, ale lze ji minimalizovat do úsporné lišty). Na této liště lze také najít tlačítko „Zvednout ruku“, které umožňuje se dočasně připojit do video hovoru pomocí hlasového přenosu. To slouží pro případné diskuse v průběhu nebo v závěru aplikace a odpadá tak potřeba přepínat se s dotazy do chatu.

1.1.3 BigBlueButton

Jde o řešení pro přenos webinářů založené na Adobe Flash Player. Pokud se uživatel připojí k webináři, zobrazí se mu dlaždicové zobrazení s připojenými posluchači, s videi přednášejících, chat a hlavní dlaždice s prezentací. U prezentace je dostupná nástrojová lišta, která umožňuje kreslit do aktuálního slajdu, ohraničovat bloky nebo vpisovat text. Tyto úpravy se sdílí spolu se slajdy mezi posluchače. Jak tento dlaždicový systém vypadá, je znázorněno na snímku A.4 v příloze A. V rámci této aplikace je možné pořídit záznam celé videokonference, kdy záznam videa obsahuje celý dlaždicový layout včetně chatu, prezentace a video streamů přednášejících.

1.1.4 Facebook

Přestože Facebook primárně neslouží k vytváření skupinových hovorů a webinářů, přesto umožňuje video konverzace, popřípadě vytváření živých streamů. O zajímavé řešení jde především v tom ohledu, že pro přenos audiovizuálních dat již implementuje WebRTC technologii.

1.2 Shnutí

Pokud bychom měli shrnout běžné požadavky, které jsou kladeny na tyto aplikace, pak jde především o možnost zobrazení více videí s uživateli, možnost sdílet slajdy s prezentací nebo možnost sdílet obrazovku. V případě zobrazení více videí je většinou volena varianta s malými náhledy, které je možno zvětšit.

V rámci skupinového hovoru umožňuje Skype i zvýraznění právě hovořícího uživatele. Tato vlastnost by bylo možné použít i pro zvětšení okna tohoto

uživatele. GoToWebinar a BigBlueButton se specializují především na webináře. Běžnou praxí je zobrazování jednoho nebo více přednášejících, zobrazení slajdů a možnost psaní komentářů během webináře do chatu.

Užitečnou funkcí se jeví jak funkce zvednutí ruky u aplikace GoToWebinar, čímž zjednodušuje komunikaci, která by jinak musela být prováděna pomocí chatu. GoToWebinar má také pohodlnou funkci minimalizace okna s hovorem.

Další užitečnou funkcí je i vpisování poznámek do slajdů. Tuto funkci by bylo možné implementovat v rámci webových technologií pomocí canvas elementu a bylo by možné tuto funkci rozšířit i tak, že by si mohl i posluchač vpisovat do prezentace vlastní poznámky, které by byly dostupné pouze pro něj.

Některé požadavky byly implementovány v rámci této práce, jiné jsou již nad rámec (například nahrávání záznamu, možnost více přednášejících nebo sdílení obrazovek). Požadavky na aplikaci jsou dále shrnuty a podrobněji popsány v následující kapitole. Další kapitoly se pak zabývají popisem technologií, které je možné pro implementaci těchto nebo podobných požadavků použít.

Kapitola 2

Požadavky na aplikaci

V této kapitole jsou definovány požadavky, které jsou požadovány pro implementaci této aplikace. Jde především o základní požadavky, které vyžadují technologickou analýzu, kterou se dále zabývá následující kapitola, a na základě kterých bude navržena celková architektura aplikace.

Tato kapitola zavažuje požadavky, které tato aplikace přímo neimplementuje, ale které s danou problematikou souvisí a které by bylo vhodné uvažovat v rámci architektonického návrhu, aby mohla být aplikace dále rozšiřitelná. Tyto požadavky by mohly také vyžadovat i jiné aplikace zabývající se komunikací v reálném čase a především aplikace zabývající se audio a video komunikací v prostředí webových prohlížečů.

Vzhledem k tomu, jaký rozmach technologií v této oblasti v prostředí webových aplikací v nedávné době nastal, mohou být snadněji implementovány i takové požadavky, které dříve vyžadovaly implementaci pomocí Flash playeru, Java appletů nebo JWS a podobných plug-inů. Technologická analýza se bude tedy zabývat i tím, jaké možnosti tyto technologie nabízí při implementaci těchto požadavků.

2.1 Požadavky na aplikaci

2.1.1 Požadavky pro přenos audiovizuálních dat

1. Aplikace by měla umožňovat přenos audiovizuálních dat mezi více uživateli. V rámci návrhu architektury by tedy měla být zvážena topologie, jakým budou data distribuována mezi uživateli, a technologie, které umožní její implementaci.

2. Uživatel by měl mít možnost připojit se jako posluchač, pokud by měl hovor charakter přednášky nebo webináře. Zároveň by měl mít možnost přijímat i vysílat audiovizuální data v rámci skupinového hovoru.

2.1.2 Požadavky pro podporu slajdů

1. Během hovoru by měl mít prezentující uživatel možnost vložit slajdy. Tyto slajdy by měly být synchronizovatelné. To znamená, že pokud přednášející uživatel přepne na další slajd, měly by se slajdy posunout automaticky i všem posluchačům.

2. Uživatel, který je posluchačem a jehož slajdy jsou synchronizovány s prezentujícím, by měl mít možnost během prezentace přejít i na jinou stránku slajdů, pokud si chce něco ujasnit nebo dočíst. V tom případě by měl přehrávač slajdů umožňovat synchronizovat slajdy zpět s přednášejícím a dále notifikovat uživatele, pokud mezitím přednášející přešel na další slajd.

2.1.3 Požadavky na zobrazení

1. Aplikace by měla umožňovat volbu rozvržení na stránce. V případě přednášky by měl mít možnost uživatel zvolit, zda má být upřednostněno zobrazení slajdů, zobrazeí videa, nebo zda mají být video a slajdy zobrazeny vedle sebe.

2. Aplikace by měla umožňovat zobrazení ve fullscreen módu.

2.2 Další možné požadavky související s audio a video komunikací a s komunikací v reálném čase

1. Pro podporu hovoru s více účastníky může být užitečná funkce zvýraznění právě hovořícího uživatele. V rámci analýzy pro tento požadavek by mělo být zjištěno, jakým způsobem lze provádět v prostředí webové aplikace analýzu získaných audiovizuálních dat, zda ji lze provádět na straně klienta, či na straně serveru, jaké jsou dopady na výkon a jaká je míra synchronizace tohoto zvýraznění se zpožděním přenosu audiovizuálních dat pro koncového uživatele.

2. Pro podporu přednášek by mohlo být užitečné také automatické rozpoznávání řeči pro zobrazení titulků, pokud by byly přednášky vedené v cizím jazyce a posluchač nebyl dostatečně zdatný pro poslech v tomto jazyce. Opět se nabízí otázka, zda lze v prostředí webové aplikace provádět rozpoznávání řeči, ať už na úrovni serveru nebo na straně klienta. Pokud by existovala možnost zobrazení titulků, by dalším rozšířením mohl být překlad těchto titulků do vybraného jazyka.

   Některé služby pro prezentaci edukativních videí (např. Lynda.com), umožňují zobrazení titulků během přehrávání videí. Dalším možným požadavkem by tedy mohl být i překlad v reálném čase.

3. Často bývají přednášky i nahrávány a poskytovány studentům pro další studium. Aplikace by tedy mohla být rozšířena i o ukládání audiovizuálního záznamu (společně s titulky a časy přepnutí slajdů, aby mohla být přednáška později opět sledována včetně slajdů).

4. Jak již bylo zmíněno v předchozí kapitole popisující existující aplikace, běžně používanou funkčností konferenčních aplikací je také sdílení plochy přednášejícího uživatele. V rámci analýzy bude tedy část věnována i této problematice.

5. Webinářové aplikace umožňují také funkci „Zvednout ruku“, která umožňuje přednášejícímu vyvolat posluchače a ten může na omezenou dobu vstoupit do diskuse pomocí hlasového vstupu.

2.2.1 Požadavky na škálovatelnost a adaptibilitu na propustnost sítě

1. Aplikace by měla být škálovatelná, to znamená, že by měla být rozšiřitelná v případě většího zatížení. Pokud bude zvolena centralizovaná topologie přenášení audiovizuálních dat, pak by měla být možnost implementovat rozložení zátěže na několik media serverů.

2. Dále by mělo být možné přizpůsobit kvalitu přenosu propustnosti sítě, ve které se uživatel nachází. Je tedy třeba zhodnotit použitou technologii s ohledem na možnost adaptace, zda ji umožňuje technologie jako taková, nebo zda je třeba pro adaptibilitu implementovat vlastní řešení.

Aplikace v současné podobě neřeší požadavky, které by byly třeba definovat, pokud by měla být aplikace opravdu nasazena v komerčním prostředí. Takovými požadavky by bylo především, jaké organizace a jakým způsobem se budou moci přihlašovat (zda bude možné implementovat různá LDAP řešení), zda bude možné posílat různé pozvánky na webináře bez nutnosti registrace (s možností přidání pozvánky do svého kalendáře), jakým způsobem se bude moci uživatel přihlásit k webináři nebo jaké limity pro počet posluchačů bude moci administrátor skupiny nastavovat. Různých požadavků může být mnoho, implementace se proto zaměřuje především na požadavky přímo spojené s vytvořením hovoru a s jeho průběhem.

Kapitola 3

Existující technologie komunikace v reálném čase pro použití v prostředí webu

Hlavním požadavkem na tuto aplikaci je komunikace v reálném čase. Tato kapitola tedy popisuje a zhodnocuje možnosti, které lze v prostředí webových prohlížečů použít. V této kapitole jsou popsány jednotlivé protokoly a jejich použití pro implementaci komunikace v reálném čase, v další kapitole jsou popsány frameworky, které by mohly umožňovat zjednodušení práce s těmito protokoly.

V rámci přehledu dostupných technologií je třeba také zhodnotit, jak jsou jednotlivé protokoly a frameworky vhodné pro práci s různým typem dat. Data, která je třeba v reálném čase přenášet v rámci této aplikace, ale i v obecném případě, mohou být:

- textová data, tzn. data, která mohou být zobrazována v různých komponentách (v našem případě např. aktualizace zpráv v chatu nebo aktualizace stavu kontaktů v kontaktním seznamu) nebo notifikace, na které může aplikace okamžitě reagovat (např. nofitikace o příchozím hovoru). Pro tento typ dat je důležité zachovávat pořadí dat a zajistit bezeztrátovost. Nejvhodnějšími kandidáty tedy budou protokoly, které jsou postaveny nad transportním protokolem TCP.

- audiovizuální data, která jsou streamována ze serveru nebo peer-to-peer mezi dvěma klienty. V obou případech je třeba zajistit, aby byla data přenášena v reálném čase s co nejmenším zpožděním. Je třeba tedy implementovat řešení, které poskytne adaptibilitu na aktuální propustnost sítě.

- binární data, tzn. především větší binární soubory. které by měly být přenášeny bezeztrátově (v této aplikaci jde například o přenos slajdů,

které je třeba nahrát na server a dále rozdistribuovat mezi účastníky hovoru).

3.1 Protokoly pro komunikaci v reálném čase

Nejběžnějším webovým protokolem je HTTP. Tento protokol lze také použít pro potřeby komunikace v reálném čase. Jelikož je ale tento protokol bezestavový a umožňuje pouze posílat požadavky, které server zpracovává, a poté vrací odpovědi, nelze data ze serveru obdržet přímo v okamžiku, kdy k nim má přístup. Toto chování lze ale simulovat různými technikami. Tyto techniky se obecně nazývají Comet. Vedle technik využívajících protokolu HTTP vznikly v posledních letech nové protokoly a metody, které neprovádí pouze simulaci tohoto chování, ale opravdu umožňují posílání dat ze serveru ke klientu nebo dokonce oběma směry.

3.1.1 Polling

Jde o základní techniku, při které se klient pravidelně dotazuje serveru, zda nedošlo k aktualizaci dat. Pokud server před posledním dotazem obdržel novou informaci, pak ji vrátí jako odpověď na tento dotaz. V opačném případě vrátí informaci, že od posledního dotazu nedošlo k žádné aktualizaci.

Nevýhodou tohoto přístupu je pravidelné dotazování na server a představuje tedy zbytečnou zátěž pro server v okamžiku, kdy nemá žádné nové údaje, které by mohl klientovi poslat.

Z hlediska použitelnosti pro různý typ dat, většina moderních prohlížečů podporují v rámci XMLHttpRequest API posílání dat nejen v textové podobě, ale umožňují také posílat binární data (ta jsou v javascriptu reprezentována datovými typy Blob nebo ArrayBuffer) [1].

3.1.2 Long polling

Long polling je další technikou, kterou lze simulovat notifikace ze serveru v reálném čase. Iniciativu opět prování klient odesláním požadavku. Pokud ale server nemá žádná nová data, která by mohl klientovi odeslat, místo okamžité odpovědi podrží spojení až do okamžiku, kdy jsou nová data dostupná a teprve tehdy odešle klientovi odpověď s těmito daty.

Výhodou oproti klasickému pollingu je to, že klient není nucen posílat stejný požadavek stále dokola. Nevýhodou je, že server musí udržovat mnoho spojení, pokud je třeba na odpověď čekat dlouho.

Dříve byl problém také detekce přerušení spojení. Pokud klient náhle ztratil spojení, musel sám zkontrolovat, zda došlo ke ztrátě spojení či nikoliv. Nyní ale všechny moderní prohlížeče implementovaly v rámci XMLHttpRequest API také vlastnost „timeout“. Po uplynutí této doby je dotaz ukončen.

3.1.3 WebSocket

WebSocket je nový plně duplexní protokol postavený nad TCP. Jeho vlastnosti vychází z vlastností TCP protokolu. Z toho vyplývají výhody i nevýhody. Výhodou je jistě spolehlivost (bezeztrátový přenos dat) a zachování pořadí zpráv. Pokud chceme posílat notifikace, komunikovat v chatu, pak je potřeba zachování pořadí zpráv nezbytná. Zachování pořadí může být ale nevýhodou, pokud budeme chtít posílat větší objem dat. Například pokud bychom chtěli implementovat požadavek na synchronizaci slajdů, pak je třeba zvážit, jakým způsobem se bude soubor s prezentací distribuovat. Pokud bychom se chtěli využít WebSocket jako primární protokol pro komunikaci, pak by bylo třeba navazovat více spojení, aby přenos velkých dat neblokoval přenos krátkých notifikací. Toto řešení je ale komplikované a navíc je to přesně to, co moderní prohlížeče mají vyřešené v rámci přenosu pomocí HTTP protokolu. V tomto ohledu tedy není WebSocket příliš vhodný na přenos větších dat.

Detekce výpadku spojení Detekce výpadku spojení, stejně jako vyvolání obnovení stránky lze okamžitě detekovat jak na klientské, tak na serverové straně a lze tedy v závislosti na události vyvolat opatření, například odpojení od videohovoru apod.

3.1.3.1 STOMP

WebSocket jako takový je oproti protokolu HTTP více nízkoúrovňový. Specifikace HTTP obsahuje definice stavových kódů, formát zpráv, podporu metadat v hlavičkách apod. WebSocket žádný takový subprotokol nespecifikuje a je tedy na každém, jakým způsobem komunikaci nad tímto protokolem implementuje.

Pokud bychom nechtěli použít žádný existující subprotokol, pak je třeba definovat formát posílaných zpráv. Další možností je využít nějaký z existujících protokolů, který by byl použitelný v rámci WebSocket protokolu. Jedním takovým protokolem je STOMP (Streaming Text-Oriented Messaging Protocol). Jde o obecný protokol, který byl vyvinut jako rozhraní pro komunikaci se systémy pro řízení zpráv, které bývají implementovány v podnikových řešeních.

Zpráva v rámci tohoto protokolu je textová zpráva, která obsahuje hlavičku s metadaty, tělo a dále specifikuje téma (topic), k jehož odběru se mohou přihlásit další konzumenti. Umožňuje tedy komunikaci nejen 1:1, ale také 1:N, pokud se více konzumentů zaregistrovalo pro odběr zpráv daného tématu. Téma může být definováno libovolně, zpravidla se používá zápis podobný adresářové struktuře (např. „/call/incoming-handouts“).

Podpora WebSocket a STOMP protokolu ve Spring frameworku Spring framework od verze 4 umožňuje komunikaci pomocí WebSocketu. Pod-

poruje jak komunikaci pomocí podprotokolu STOMP, tak i čistou variantu bez podprotokolů. Umožňuje poslat zprávu všem uživatelům, kteří naslouchají danému tématu, a dále také umožňuje poslat zprávu jednomu konkrétnímu uživateli. To framework zajišťuje tím, že téma zprávy prefixuje pomocí identifikátoru uživatele (např.: „/user/1/call/incoming-handouts“). Při implementaci čistého řešení bez podprotokolů má uživatel dostupný jeden přístupový bod. Tou je jedna metoda, ve které uživatel implementuje zpracování příchozí zprávy. Pokud naopak implementujeme podporu WebSocketu s podprotokolem STOMP, pak lze vytvářet controllery, které mají odbodbnou strukturu, jako controllery ze Spring MVC. Pouze se na vybranou metodu nemapuje volaná URL, ale téma zprávy. „RequestMapping“ je nahrazen na „MessageMapping“ anotaci.

Podpora protokolu STOMP na straně klienta Na straně klienta existuje několik knihoven, které implementují podporu pro STOMP. Nejznámější je knihovna Stomp.js [2]. Ta je doporučována i v dokumentaci Spring frameworku. Bohužel ale v roce 2015 autor této knihovny ukončil její vývoj. Nicméně jde o propracovanou knihovnu, která i v současné podobě oskytuje kvalitní rozhraní. Další knihovnou je např. MSGS.js.

3.1.4 Možnosti použití frameworku nad protokoly pro komunikaci v reálném čase

V závislosti na tom, co od komunikace požadujeme, lze využít jednu z předchozích technik pro komunikaci mezi klientem a serverem. V případě, že chceme implementovat obousměrnou komunikaci mezi klientem a serverem, pak je ideálním kandidátem použití protokolu WebSocket. V dnešní době je takřka plně podporován všemi moderními desktopovými prohlížeči i prohlížeči v mobilních telefonech. Nicméně aplikace, které využívají obousměrné komunikace, vznikaly i v době, kdy WebSocket protokol neexistoval nebo byla jeho podpora nedostatečná. V této době vzniklo několik nástrojů, které obousměrnou komunikaci implementovalo pomocí Comet technik a následně přidaly i podporu pro WebSocket. Nejzajímavější z těchto frameworků v oblasti jazyka Java jsou Atmosphere nebo Lightstreamer.

Atmosphere framework [3] je ideální nástroj, pokud chceme jednotné API nezávisle na tom, jaká technika bude pro obousměrnou komunikaci použita. Tento framework umožňuje komunikaci pomocí WebSocketu (podporuje i komunikaci pomocí podprotokolu STOMP [4]) a v případě, že není možné WebSocket protokol použít, použije se některá z výše zmíněných Comet technik. Tento framework zajišťuje serverovou i klientskou část komunikace.

Lightstreamer [5] je komplexní framework pro implementaci komunikace v reálném čase. Podporuje také komunikaci pomocí protokolu WebSocket i pomocí ostatních streamovacích technik nad HTTP protokolem.

Spring framework ve své implementaci WebSocket protokolu umožňuje také nastavit fallback na HTTP technologie pomocí SockJS protokolu. Na straně klienta je pak třeba implementovat SockJS klienta, který zprávy obsluhuje.

Nabízí se tedy otázka, zda je pro současnou aplikaci vhodné použít nějaký z těchto nástrojů nebo zda lze použít protokol WebSocket samostatně. Jak bude zmíněno dále v této kapitole, budeme chtít využít webové technologie pro potřeby přenosu audiovizuálních dat. Všechny prohlížeče, které tyto technologie umožňují, zároveň podporují i WebSocket protokol. Co se týče podpory prohlížečů, mohli bychom využít samotný WebSocket protokol. Problémem však může být, zda některé firewally nebudou tuto komunikaci blokovat a zda nevznikne problém, pokud bude klient (popřípadě i server) za nějakým proxy serverem. Pokud jde o firewally, při běžném nastavení jimi projde komunikace bez povšimnutí. WebSocket protokol totiž běží na portu 80 (pro zabezpečenou komunikaci na portu 443) stejně jako HTTP. Některé proxy však mohou způsobit potíže [6]. Z tohoto důvodu by bylo vhodné fallback na techniky simulující obosuměrnou komunikaci nad HTTP použít.

3.2 Protokoly pro přenos médií

3.2.1 Dynamické adaptivní streamování (HLS, HDS, HSS, MPEG-DASH)

Před nástupem technologií pro dynamické adaptivní streamování se běžně používalo takzvané progresivní streamování. Při tomto způsobu stahování dat nebylo možné měnit kvalitu přenášených dat v průběhu přenosu, kvalita byla dána kvalitou videa. Dále nebylo možné v přehrávání přeskočit na jinou část videa, dokud nebylo video celé staženo. Postupně různé společnosti implementovaly adaptivní streamovací protokoly, které tento problém řešily. V té době vzniklo několik proprietárních specifikací a protokolů, které se zabývají adaptibilitou přenosu těchto dat v závislosti na propustnosti sítě. Apple definoval HLS (HTTP Live streaming), Microsoft HSS (HTTP Smooth streaming) a Adobe HDS (HTTP Dynamic streaming). Nakonec vznikla ale také nezávislá specifikace pro přenos těchto dat MPEG-DASH. Tento standard již využívá Google a Netflix.[7]

3.2.1.1 MPEG-DASH (ISO/IEC 23009-1)

Jde o standard pro streamování audiovizuálních dat s možností přizpůsobení datového toku šířce pásma. Je založený na HTTP, tak aby byl snadno schopný

překonat firewall a NAT. Video je potřeba na serveru zpracovat. Nejprve se provede fragmentace na jednotlivé rámce tak, aby bylo možné posílat posílat jen různé části videa. Pro použití DASH (Dynamic Adaptive Streaming over HTTP) je dále potřeba rámce zpracovat tak, aby byly dostupné v různé kvalitě. Pro zpracování videa lze použít například knihovnu ffmpeg. Dále klientská strana očekává manifest soubor, který popisuje, v jakých formátech server data poskytuje a v jakých různých rozlišeních. Tento manifest musí server také připravit. Existují i různé návody, jak lze tento proces provést. [8] Na straně klienta vznikla specifikace Media Source Extensions[9], která umožňuje adaptivní streamování na základě propustnosti. Pomocí tohoto rozhraní klient odhadne propustnost sítě a na základě manifestu, který získá ze serveru, se rozhodne, v jakém rozlišení bude chtít data streamovat.[10] Toto rozhraní je implementováno v prohlížečích nativně bez nutnosti použití dalších doplňků.

Tyto technologie řeší především přenos audiovizuálních dat ze serveru ke klientovi. Jsou tedy ideální například pro implementaci přehrávače. To by mohlo být užitečné v případě, pokud by aplikace v budoucnu umožňovala nahrávání záznamu, které by poté bylo možné znovu přehrávat. Popřípadě by bylo možné tuto technologii použít pro webináře, kdy by kameraman streamoval data na nějaký server a ten by poté prezentoval data posluchačům. Nelze ale jednoduše zapojit více účastníků do hovoru. Technologie pro adaptivní streamování dat však neřeší otázku, jak pracovat s kamerou a mikrofonem na straně klienta, aby bylo možné vysílat data od klienta na server nebo dále dalším klientům. V dřívější době bylo potřeba pro přístup ke kameře nebo mikrofonu externích zásuvných modulů, které tuto funkci poskytovaly. Nejběžnějšími jsou Adobe Flash nebo Java Applet. V dnešní době se však začíná prosazovat sada technologií, protokolů a JavaScriptových API, které jsou zastřešeny pod názvem WebRTC a které právě tuto možnost poskytují bez nutnosti instalovat externí zásuvné moduly.

Pokud bychom chtěli přenos médií implementovat v prostředí jazyka Java, pak by slibnou technologií pro implementaci podobné aplikace mohl být Java Media Framework [11], tedy komplexní framework pro práci s médii. Protokol pro přenos médií stejně jako u technologie Adobe Flash, byl postaven na protokolu RTP (Real-time protocol), který byl založen na UDP. Na stranu klienta lze tuto technologii aplikovat pomocí technologie Java applet nebo JWS (Java Web Start). Pomocí nich je možné implementovat běžnou Java SE aplikaci a poté ji publikovat v okně prohlížeče. Jak již bylo ale řečeno, bylo by pro implementaci v prohlížeči potřeba pluginů, které již některé prohlížeče blokují. Následující podkapitola se bude již zabývat nově vznikajícím standardem, který řeší přenos médií a přístup k audiovizuálním zařízením.

3.2.2 WebRTC

Specifikace WebRTC [12] definuje aplikační rozhraní pro ECMA Script, které umožňuje komunikaci mezi dvěma uživateli. Obecně je tímto pojmem označován soubor technologií a javascriptových API, které poskytují práci s datovými kanály nebo které umožňují přistupovat ke kameře a mikrofonu, nebo které umožňují vytvářet záznam nebo obrazové snímky. [13] Některé části jsou již standardizované, některé body jsou stále předmětem probíhaící standardizace. Jde o technologii, která podporuje peer-to-peer komunikaci mezi dvěma účastníky. Přestože jde o komunikaci mezi dvěma klienty, pro samotnou inicializaci komunikace je potřeba takzvaný signalizační server. Ten zprostředkovává výměnu informací mezi oběma klienty, které slouží k samotnému navázání spojení. K distribuci informací o prohlížeči, podporovaných kodecích pro přenos multimediálních dat lze využít Session Description Protocol (SDP) a k distribuci informací pro traverzování NAT je použit protokol Interactive Connectivity Establishment (ICE). Tomuto procesu inicializace spojení se říká signalizace. [14]

Tato technologie je v současné době podporována pouze v prohlížeči Chrome a Firefox. Prohlížeč Edge přišel s vlastním řešením ORTC, který oproti současnému standardu WebRTC 1.0 neprovádí výměnu SDP zpráv a jeho rozhraní s nemusí být kompatibilní. V současné době jsou snahy o spojení obou standardů, Pokud bude nyní ale třeba využít nějaký framework nebo media server, pak je třeba zjistit, zda podporuje i ORTC, pokud bychom chtěli zajistit podporu i pro prohlížeč Edge.

3.2.2.1 Možnost použití pro víceuživatelské hovory

Přestože je tato technologie určena pro komunikaci peer-to-peer, je možné vytvořit různé topologie pro komunikaci mezi více účastníky. O možných topologiích pojednává kapitola . Obecně lze říci, že tuto technologii lze použít pro implementaci víceuživatelských hovorů, přičemž je na nás, jakou topologii si zvolíme.

3.2.2.2 Možnosti použití pro výměnu textu a příloh

Součástí WebRTC je i možnost posílání dat pomocí speciálního datového kanálu. Oproti WebSocketu, AJAX technologiím apod. jde opět o čistě peer-to-peer řešení. Toto řešení může být tedy výhodné v okamžiku, kdy nechceme použít centralizované řešení a využijeme peer-to-peer vlastnosti, kdy si mohou uživatelé vyměňovat data přímo bez nutnosti posílání přes server.

Datový kanál je implementován pomocí SCTP protokolu. [15] Tento protokol je konfigurovatelný a datový přenos tak může být prováděn ve spolehlivém nebo nespolehlivém módu (obdobně jako u komunikace přes TCP nebo UDP). Stejně tak lze konfigurovat, zda mají být zprávy doručovány v pořadí, v jakém byly vyslány, nebo nikoliv. [16]

V případě, že bychom chtěli implementovat datovou komunikaci centralizovaně pomocí serveru, pak by to znamenalo zbytečné komplikování architektury. Media server, který by sloužil jako prostředník, by tuto funkci navíc musel také umožňovat, což nemusí být zaručeno. Místo toho by tedy v takovém případě bylo vhodnější použít standardní protokol HTTP.

3.2.3 Možnosti nastavení parametrů pro přenosu dat

Dále nás zajímá, jak lze nastavit parametry pro přenos dat a jak lze tyto parametry měnit v průběhu hovoru. Možnost úpravy parametrů přenosu dat nás zajímá z různých důvodů. Zaprvé bychom mohli chtít během hovoru změnit typ přenášených audiovizuálních dat. Pokud by byl započat jako hlasový, pak by některý z uživatelů mohl chtít v průběhu hovoru začít vysílat i vizuální data. Druhým důvodem je možnost upravovat parametry jako je snímková frekvence, přenosová rychlost nebo velikost obrazu tak, abychom docílili co nejlepšího výsledného obrazu v rámci dané propustnosti sítě, po které jsou audiovizuální data přenášena.

3.2.3.1 Nastavení parametrů

V rámci Media Capture and Streams API [17], které umožňuje v prohlížeči přistupovat k mikrofonu a kameře[18], lze nastavit parametry, které definují, zda má být z těchto zařízení získán audio nebo video signál. Pokud chceme získat audio nebo video signál, pak lze nastavit specifické parametry pro audio nebo video signál. [19] Pokud má být získán video signál, pak lze definovat minimální a maximální velikosti videa nebo snímkovou frekvenci. V prostředí mobilních zařízení lze definovat, zda má být pro získání videa použita zadní nebo přední kamera. Pro audio signál lze nastavit vzorkovací frekvence, hlasitost, latence, nastavení, zda má být odstraňováno echo apod.

Pokud bychom chtěli změnit vstupní parametry v průběhu hovoru, je možné buď ukončit spojení a vytvořit nové, nebo lze implementovat proces zvaný „renegociace“. Pokud bychom ukončili a znovu navazovali spojení, pak bychom museli opět provádět celý proces signalizace včetně výměny SDP zpráv i ICE kandidátů. Pokud bychom prováděli pouze renegociaci, pak není potřeba výměna ICE kandidátů, ale dojde k výměně SDP zprávy s upravenými vstupními parametry a následnému vytvoření nového audiovizuálního streamu dle nových parametrů.

3.2.3.2 Přizpůsobení propustnosti sítě

WebRTC umožňuje adaptaci na aktuální propustnost sítě. Pokud inicializujeme komunikaci s definovanými vstupními parametry, pak zbývá dopočítat přenosovou rychlost. WebRTC se snaží samo nalézt optimální přenosovou rychlost (tedy počet bitů přenesených za sekundu) tak, aby docházelo k minimální latenci a byly dodrženy vstupní parametry, jako je např. zmíněná

snímková frekvence. Pokud dojde ke změně propustnosti sítě, bude automaticky upravena i přenosová rychlost.

Kapitola 4

Existující frameworky a media servery

Z předchozí rešerše je patrné, že ideální technologie pro implementaci přenosu audiovizuálních dat v prostředí webu je WebRTC. Jde o technologii velmi mladou, nicméně již nyní vznikají ucelená řešení, která usnadňují proces navázání komunikace a která také usnadňují implementaci víceuživatelské komunikace a nabízí další možnosti zpracování audiovizuálních dat.

4.1 Kurento

Nejzajímavějším projektem v této oblasti je open-source projekt Kurento. Jde o media server, který jako prostředník mezi účastníky poskytuje několik funkcí, kterými jsou například skupinová komunikace, nahrávání záznamu, konverze kódování mezi kodeky, podpora rozšířené reality, mixování nebo analýza řeči.

Projekt Kurento nezahrnuje pouze media server, ale obsahuje také Kurento API (též označované jako „klientské API“, které zpřístupňuje funkce media serveru vývojářům. [20] Toto API je implementováno pro různé platformy, mezi nimiž je například Java nebo Node JS. Mezi klientským API a media serverem samotným probíhá komunikace pomocí interního protokolu, který je založen na protokolu WebSocket a používá JSON-RPC.

Kurento také dále poskytuje doplňující funkcionalitu v podobě javascriptového balíčku „kurento-utils“ určeného pro webové klienty. Tento balíček usnadňuje signalizační proces a práci s jednotlivými WebRTC spojeními.

4.1.1 Spojení více uživatelů

Pokud se uživatel chce spojit s jiným uživatelem nebo více uživateli, bude Kurento Media server sloužit jako prostředník. To znamená, že všichni uživatelé budou spojeni pouze s tímto media serverem a ten bude distribuovat audiovizuální signál mezi ostatní uživatele. To provádí pomocí „Media Pipeline“. Jde

o průběh zpracování audiovizuálních dat uvnitř media serveru. Součástí této pipeline jsou jednak koncové body pro vysílají a příjmající uživatele, dále to může být koncový bod pro nahrávání, různé filtry a jiné funkce.

Pokud chceme vytvořit hovor s více uživateli, je potřeba jeden vstupní koncový bod vysílajícího uživatele propojit s výstupními koncovými body všech přijímajících uživatelů. Toto provedeme právě pomocí klientského API.

4.1.2 Požadavky na implementaci

Pro použití tohoto řešení je tedy třeba vyhradit vlastní server pro instalaci Kurento media serveru. Tento server vyžaduje exaktní verzi operačního systému, konkrétně verzi Ubuntu 14.04 LTS, aby měl zajištěnu veškerou potřebnou funkcionalitu. To je jediné velké omezení tohoto řešení. V rámci této práce jsem se pokoušel o zprovoznění media serveru na poskytnutém systému Debian Jessie, nicméně některé potřebné balíčky nebylo možné doinstalovat a neexistuje zatím žádný návod, který by naznačoval, že lze jiný operační systém použít.

Klientské API lze následně implementovat v naší Java EE aplikaci. Tato aplikace zároveň představuje i signalizační server, přičemž Kurento poskytuje i funkce pro sestavení SDP zpráv. Pro komunikaci mezi aplikací a Kurento media serverem, popřípadě pro komunikaci v rámci signalizace, je třeba zajistit, abychom měli podporu protokolu WebSocket.

4.2 Red5

Red5 je open-source media server, který slouží pro streamování dat. V placené verzi nabízí možnosti škálování a podporu SDK pro mobilní zařízení. Podporuje protokoly HLS, WebSocket, RTSP. Nově se také snaží implementovat WebRTC technologie.

V průběhu psaní této práce bylo uveřejněno demo, prezentující WebRTC komunikaci mezi dvěma uživateli. [21] Další ukázky použití s WebRTC jsou dodávány jako součást Red5 Pro serveru. Žádná další dokumentace pro použití s WebRTC neexistuje a proto varianta použití Red5 serveru nebyla dále uvažována.

Kapitola 5

Rešerše dalších technologií pro implementaci základních požadavků

V této kapitole jsou popsány další technologie a knihovny, které lze použít pro implementaci požadavků definované v kapitole 2.

5.1 Technologie související s přenosem audiovizuálních dat

Tato sekce se zaměří na detekci řeči, tedy možnost určit, zda daný účastník právě hovoří nebo ne, dále také rozpoznání řeči a případné zobrazení titulků.

5.1.1 Detekce řeči

Detekci řeči lze použít pro implementaci interaktivního layoutu při skupinovém hovoru, kdy by byl zvýrazněn nebo předán do popředí uživatel, který právě hovoří. Tuto funkcionalitu lze implementovat pomocí sady nových API, konkrétně Web Audio API. [22] Nad audio a video elementy lze získat takzvaný AudioContext. AudioContext API [23] dále umožňuje vytvořit analyzér [24], který je dále navázán na audiovizuální stream [25] získaný z těchto elementů a pomocí kterého lze následně analyzovat zvukový signál. Můžeme tak zjistit intenzitu zvuku v dB a podle tohoto detekovat, zda uživatel právě hovoří nebo vysílá jiný zvuk. [26]

Naštěstí tuto detekci nemusíme implementovat sami, existuje jednoduchá javascriptová knihovna **hark** [27], která tato API využívá a která vrací události pro začátek a konec vysílání zvukového signálu.

5.1.2 Rozpoznání řeči

Rozpoznávání řeči je další funkce, která by mohla být vyžadována jako další možné rozšíření aplikace.

Pokud bychom rozpoznávání řeči chtěli implementovat na straně serveru, pak bychom tuto funkci potřebovali implementovat v rámci zpracování audiovizuálního signálu uvnitř media serveru. Kurento media server bohužel tuto funkci momentálně neumožňuje, nicméně díky své modularitě lze implementovat vlastní modul pro rozpoznávání řeči.

O rozpoznávání řeči lze uvažovat i na straně prohlížeče. Nové **Web Speech API** [28] umožňuje jak syntézu hlasu z textu, tak také rozpoznávání řeči. Toto API sice není standardizované, nicméně prohlížeče Firefox a Chrome již tyto funkce implementují a lze je tedy použít alespoň v těchto prohlížečích.

5.1.3 Nahrávání video záznamu

Nahrávání audiovizuálního záznamu lze opět na straně serveru i na straně klienta. Na straně serveru lze tuto funkci implementovat přímo pomocí Kurento media serveru. [29] Ten obsahuje modul pro nahrávání, který lze nastavit pomocí Kurento klientského API jako jeden z koncových bodů v pipeline, která definuje způsob zpracování audiovizuálních dat uvnitř media serveru. Pomocí tohoto API můžeme přidat koncový bod do vytvořeného hovoru a poté dále specifikovat cestu k souboru, který bude vytvořen.

Druhou variantou je implementace v prostředí prohlížeče. Zde vzniká specifikace MediaStream Recording [30], která umožňuje navázání na Media and Streams API, přičemž nabízí jednoduché API pro zapnutí a vypnutí nahrávání daného streamu a dále poskytuje událost, která je vyvolána v okamžiku, kdy jsou data dostupná. Výstupní data jsou reprezentována datovým typem Blob, který lze přímo uložit do souboru na straně klienta, nebo lze tato data poslat na server pomocí AJAXu nebo WebSocket protokolu. Nad touto API vznikla knihovna RecordRTC [31], která nahrávání usnadňuje a dále poskytuje možnosti pro nahrávání dalších dat, která lze získat například z canvas elementu nebo ze sdílení obrazovky.

5.2 Technologie pro implementaci slajdů

Pro implementaci slajdů bude třeba zvážit, jak se budou slajdy distribuovat, v jakém formátu, zda bude možné provádět na serveru konverze mezi formáty a jaké formáty lze v prohlížeči vůbec zobrazit - zda existují knihovny pro zobrazení slajdů, které budou umožňovat synchronizaci slajdů, tzn. zda umožní implementovat automatické změny stránky pomocí javascriptu.

5.2.1 Zobrazení slajdů

Pro implementaci slajdů lze použít knihovnu PDF.js [32] pro zobrazování PDF dokumentů. Tu implementuje společnost Mozilla a zároveň ji využívá jako primární prohlížeč PDF dokumentů i internetový prohlížeč Firefox. Tato knihovna poskytuje základní API pro načtení souboru (umožňuje zadat zdrojový dokument pomocí URL, nebo jako binární typované pole TypedArray ze specifikace ES6, které lze získat po vložení souboru uživatelem), dále umožňuje mimo jiné zobrazit dokument buď celý, nebo pouze vybranou stránku s možností zobrazenou stránku pomocí JavaScriptu měnit. To je přesně to, co od prohlížeče slajdů vyžadujeme. Zároveň je to jediná komplexní JavaScriptová knihovna pro zobrazení PDF souborů v prohlížeči.

Pokud bychom chtěli podporovat také zobrazovaní slajdů v jiném formátu, lze použít například open-source knihovnu WebODF [33]. Ta podporuje zobrazení Open document formátů.

5.2.2 Konverze formátů

Pokud bychom chtěli implementovat zobrazení slajdů ve formátu PPTX, pak bude potřeba implementovat konverzi tohoto formátu na PDF nebo Open document formát. Konverzi lze provádět pouze na straně serveru. Na straně klienta žádná knihovna neexistuje.

V prostředí jazyka Java existuje několik nástrojů, přičemž každý nástroj spoléhá na jiný způsob konverze do PDF. Některé využívají nástroje OpenOffice nebo LibreOffice, jiné provádí konverzi pomocí jiných XML typů, některé využívají webové služby. [34]

Jednou z knihoven, která využívá OpenOffice nebo LibreOffice, je JODConverter [35]. Jde o komplexní knihovnu, bohužel byl její vývoj pozastaven. Jde o nadstavbu nad OpenOffice nebo LibreOffice, které je třeba mít instalované na serveru. Zároveň je třeba před samotným zavoláním konverze spustit OpenOffice či LibreOffice jako službu.

Dalším nástrojem je docx4j [36]. Ten umožňuje manipulovat s formáty PPTX, DOCX a XLSX a poskytuje fasádu pro implementaci konverze do PDF. Pro konverzi do PDF však nově používá primárně svoji komerční webovou službu Plutext PDF Creator [37]. docx4j lze ale použít i v kombinaci s Apache FOP, přičemž se konverze provádí dvojí konverzí přes XLS-FO formát. Další možností je použití docx4j s iText PDF converterem [38]. Licence iText PDF convertoru je AGPL, která definuje, za jakých podmínek může být použita zdarma, v ostatních případech je její použití zpoplatněno.

XDocReport [39] je dalším nástrojem, který umožňuje pracovat s docx dokumenty. Načtení provádí pomocí Apache POI XWPF a pro generování do PDF využívá opět iText (umožňuje také generování do XHTML pomocí SAX).

Konverze pomocí Apache FOP je údajně mnohem pomalejší než pomocí iText konverteru a zároveň spotřebovává na serveru značné množství paměti. [34] Z tohoto důvodu se nejeví jako příliš vhodné řešení. Ostatní řešení vyžadují dodatečnou instalaci OpenOffice nebo LibreOffice na server nebo využívají služby a knihovny, jejichž použití na základě jejich licence podléhá některým restrikcím.

5.3 Sdílení obrazovky

Pokud bychom chtěli implementovat funkci sdílení obrazovky, pak je situace poněkud složitější. Moderní trend prohlížečů implementovat nové funkce bez použití doplňků byl v tomto případě porušen ve prospěch zajištění bezpečnosti. Prohlížeče Chrome a Firefox původně implementovali řešení bez potřeby doplňků, nicméně z prohlížečů byla tato funkce opět v dalších verzích odebrána právě z důvodu bezpečnosti. Vzhledem k tomu, že tato funkce umožňuje sdílet data, která jsou mimo hranice prohlížeče samotného, je od uživatele vyžadována větší účast než pouhé potvrzení se zapnutím této funkce. Touto větší účastní se rozumí instalace rozšíření do svého prohlížeče. Pro Chrome existuje např. appear.in. Podobné rozšíření existuje v současné době i pro prohlížeč Firefox.

Jak již bylo zmíněno dříve, lze pomocí knihovny RecordRTC ze sdílení obrazovky nahrávat záznam na straně klienta. Obdobně jako jiný typ audiovizuálních dat lze záznam obrazovky nahrávat i na straně media serveru.

Kapitola 6

Návrh architektury řešení

6.1 Návrh architektury komunikace

6.1.1 Topologie

Při návrhu přenosu médií mezi účastníky musíme zvážit topologii, v jaké budou účastníci spojeni. To znamená způsob, jakým se bude distribuovat video od jednoho účastníka k ostatním účastníkům hovoru. Měli bychom zvážit, jaké jsou jejich výhody a nevýhody - jak jsou tyto topologie stabilní při výpadcích, jaká je náchylnost na propustnost atd., a dále jejich proveditelnost a náročnost implementace.

6.1.1.1 Centralizovaná (Hvězdicová)

Jednou z možností je centralizovaná topologie. Komunikace mezi každými dvěma uživateli probíhá přes nějaký media server, tzn. přes prostředníka, který distribuuje audiovizuální signál všem ostatním účastníkům hovoru. Pro každého vysílajícího účastníka je tedy potřeba navázat jedno příchozí spojení do media serveru a $n-1$ odchozích spojení z media serveru osatním uživatelům, pro n účastníků hovoru. Pro hovor s více uživateli je tedy potřeba, aby server obsloužil n^2 streamů audiovizuálních dat. V případě přednášky nebo webináře, kde je vysílající pouze lektor nebo vybraný kameraman, je potřeba pouze n streamů.

Škálování Pokud bychom chtěli snížit zatížení media serveru, můžeme využít horizontálního škálování a nasadit více media serverů a implementovat vyvažování zátěže. Pokud bychom měli n vysílajících účastníků a k media serverů, pak $nmodk$-tý účastník bude vysílat přes k-tý media server a každý posluchač bude z každého media serveru přijímat n/k audiovizuálních streamů.

Tento typ škálování je vhodný, pokud by převažovaly hovory se všemi vysílajícími účastníky. Pokud by naopak převažovaly přednášky, kde by byl

jeden prezentující a mnoho posluchačů, pak bychom museli řešit škálování zrdcadlením, tzn. prezentující by komunikoval s jedním video serverem a ten by přeposílal data ostatním media serverům a ostatní posluchači by byli postupně připojováni k jednotlivým media serverům a každý media server by tak musel zpracovat n/k streamů plus jeden stream pro komunikaci mezi media servery.

Toto je však nad rámec této práce a možnosti škálování zde byly uvedeny především pro znázornění možnosti rozšiřitelnosti směrem ke komerčnímu produktu.

Nevýhodou je tedy vysoké zatížení media serveru, který by měl být připojen do sítě s vysokou propustností, aby dokázal obsloužit až kvadratický počet audiovizuálních streamů oproti počtu připojených uživatelů.

Výhodou je však možnost použití existujícího řešení media serveru jako je například Kurento media server. Ten mimo přeposílání dat umožňuje také další funkce, jak bylo již zmíněno v kapitole 4.

6.1.1.2 Decentralizovaná

Další možností je implementace decentralizované topologie. V případě, že by se uživatel chtěl spojit s n účastníky skupinového hovoru, pak by při centralizované topologii po síti, která jej spojuje s media serverem, bylo přenášeno n příchozích a 1 odchozí stream. Pokud bychom implementovali decentralizovanou topologii ve které by každý byl spojen s každým, pak by uživatel musel ale také n různých streamů odesílat. Nyní by již neexistoval žádný prostředník, který by data distribuoval. Pokud by byl uživatel na místě s nízkým připojením, pak by zbytečně zatěžoval svou okolní síťovou infrastrukturu.

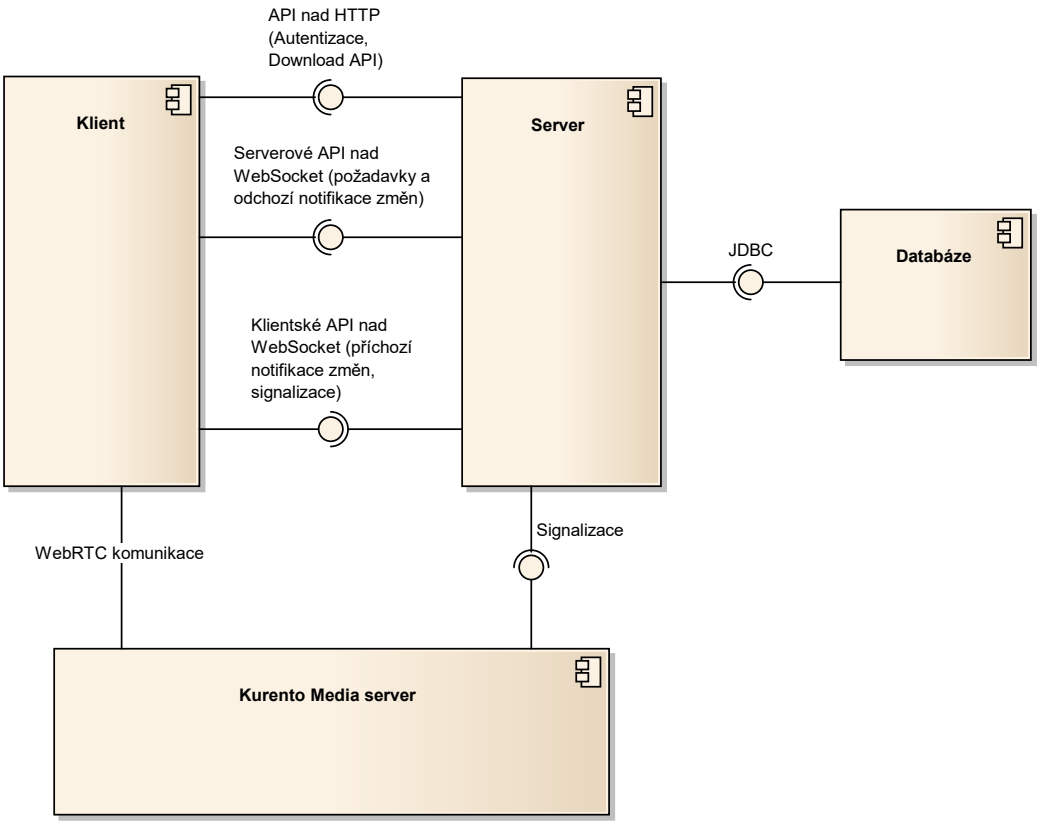
Zároveň bychom se v takovém případě ochudili o možnosti, které nám media server může nabídnout (například již zmíněné nahrávání).

Další možností je implementovat nějakou hybridní topologii. V tom případě by někteří účastníci distribuovali audiovizuální stream od jiných účastníků a bylo by třeba počítat, kdo má být s kým propojen a dále by musela být vyřešena adaptabilita na příchod a odchod účastníka z hovoru. Obecně je tento způsob náročný na návrh a implementaci.

6.2 Návrh základní architektury aplikace

6.2.1 Základní architektura

Základní architektura (viz obrázek 6.1) se skládá z klientské a serverové části aplikace, dále z databáze, která uchovává perzistentní informace o kontaktech a konverzacích, a media serveru, který slouží jako prostředník pro zajištění komunikace mezi uživateli. Aplikace tedy implementuje řešení s centralizovanou topologií komunikace.



Obrázek 6.1: Komponentový diagram znázorňující základní architekturu aplikace

Klientská část aplikace by neměla umožňovat zobrazovat pouze statické stránky, ale také by měla umožňovat dynamicky měnit obsah na základě různých notifikacích o změně stavu. Tyto změny jsou typicky vyvolány akcemi jiných uživatelů.

Serverová část aplikace umožňuje práci s persistentními daty i s daty, které jsou platné pouze v okamžiku probíhajícího hovoru. Umožňuje také notifikovat uživatele o aktuálních změnách dat a dále je zodpovědná za signalizační proces, tzn. za zprostředkování WebRTC komunikace mezi uživateli.

Komunikace mezi klientem a serverem probíhá pomocí HTTPS a pomocí WebSocketu. HTTPS slouží pro přihlašování uživatele a pro API které slouží k nahrání a stažení dokumentů, např. slajdů. Ostatní výměna dat mezi

klientem a servrem je implementována nad protokolem WebSocket. Tím lze dosáhnout notifikace klienta o změnách v reálném čase. V následující sekci je vysvětleno, proč byly zvoleny tyto protokoly a proč byla jejich úloha takto rozdělena.

Proces navázání audiovizuální komunikace mezi uživatelem a Kurento media serverem (tzv. signalizační proces) zprostředkovává serverová část aplikace. Komunikace je navázána vždy mezi jedním uživatelem a KMS, neboť je pro komunikaci zvolena centralizovaná topologie. Pro navázání audiovizuální komunikace klient pošle na server požadavek. Server poté zprostředkuje výměnu SDP zpráv mezi klientem a KMS a iniciuje navázání komunikace. Signalizace směrem ke klientovi může využít stejného WebSocketového spojení jako zbytek aplikace. Výměnu SDP zpráv směrem k media serveru zajišťuje Kurento klient pomocí Kurento protokolu.

Jak je implementována klientská a serverová část aplikace, je podrobněji popsáno v kapitole 7. Ještě je však třeba zvážit, zda bude aplikace postavena klasickým způsobem, tzn. renderování šablon bude probíhat na straně serveru, nebo zda bude logika vykreslování grafických komponent přenesena na stranu klienta.

6.2.2 Rozdělení zodpovědnosti mezi klientskou a serverovou část aplikace

Architektura aplikace by měla odrážet fakt, že většina klíčových vlastností aplikace, ať už jde o samotné streamování videa, inicializaci a ukončování hovorů, diskusi či synchronizaci slajdů prezentace, vyžaduje real-time komunikaci mezi uživateli.

6.2.2.1 Serverově orientovaný přístup

Na běžné aplikace, u kterých není potřeba tak pružně reagovat na různé události, bývá dostačující variantou použití nějakého řešení pro serverové generování stránek s podporou překreslování komponent stránky pomocí AJAXu, přičemž real-time komunikace by byla implementována mimo pomocí javascriptu a vhodného komunikačního protokolu jako je například WebSocket protokol.

Tento klasický přístup však nekoresponduje s charakterem této aplikace. Zde není komunikace v reálném čase pouze doplňkem k hlavní aplikaci, ale hlavním požadavkem a je třeba tomu přizpůsobit celou architekturu aplikace.

Charakter aplikace se také spíše blíží single page aplikaci. To znamená, že hlavní logikou není prezentování jednotlivých stránek a přecházení mezi nimi, ale provádění dynamických operací a aktualizace obsahu v rámci jedné stránky, jejíž komponenty musí dynamicky reagovat na podněty, které přicházejí ze serveru od jiných uživatelů aplikace.

Jelikož většina uživatelských akcí vyvolává tyto dynamické události, jeví se serverové generování, byť s podporou AJAXových volání pro překreslování komponent, značně nepohodlné a je třeba zvolit nástroje, které se soustředí primárně na toto dynamické chování.

V rámci implementace byla implementována ukázka řešení za použití serverového renderování, nicméně i v praxi se potvrdilo, že je tento přístup značně svazující.

6.2.2.2 Front-endově orientovaný přístup

Výše zmíněné důvody byly důvodem pro zvážení přenesení vykreslovací logiky více na stranu klienta. V takovém případě server poskytuje definované aplikační rozhraní a zajišťuje poskytování aktualizovaných dat. Přenesení části logiky na stranu klienta také usnadňuje implementaci protokolu pro komunikaci v reálném čase.

6.3 Návrh použití komunikačních protokolů

Komunikace mezi klientskou a serverovou částí probíhá oběma směry. Klientská část posílá na server běžné požadavky, zároveň ale také posílá zprávy, na jejichž základě je třeba, aby server notifikoval další uživatele. Pro tento typ komunikace je vhodné použití WebSocket protokolu, který umožňuje obousměrnou komunikaci.

Dále signalizační proces a proces inicializace hovoru jsou založeny na výměně zpráv a WebSocket je také přirozenou volbou. Nabízí se tedy varianta, že bude veškerá komunikace mezi klientem a serverem pro přehlednost implementována nad jedním protokolem. Následující dvě podsekce ukazují, že tato strategie bohužel není příliš vhodná. Ukazují, že alespoň autentizace a API pro nahrávání a stahování souborů je třeba implementovat nad protokolem HTTP.

Pro přenos ostatních dat mezi klientem a serverem je WebSocket protokol přesto zvolen. Pokud bychom se snažili běžné dotazy implementovat pomocí HTTP a posílání zpráv implementovat pomocí WebSocketu, pak by došlo k rozdělení API do dvou různých míst a to i v případě, že by implementovaly jeden logický celek. Nevýhodou totoho řešení je však nutnost alespoň nějaké implementace komunikace typu „požadavek - odpověď“. Přesto byl zvolen přístup snažit se co největší část aplikačního rozhraní implementovat nad jedním protokolem, kterým je WebSocket.

6.3.1 API pro nahrávání a stahování souborů

Problém s implementací přes WebSocket ale nastává v okamžiku, kdy chceme implementovat API pro nahrávání a stahování souborů. To je potřeba minimálně pro sdílení slajdů během hovoru. Pokud bychom chtěli implementovat

nahrávání a stahování pomocí WebSocket protokolu, pak bychom museli pro každý soubor navázat nové spojení. WebSocket totiž pracuje nad protokolem TCP a tudíž zachovává pořadí poslaných zpráv. Pokud bychom pomocí jednoho spojení poslali velký soubor, pak by byla pozastavena ostatní komunikace v rámci tohoto spojení, dokud nebude přenesen celý soubor. Tím by mohlo docházet k nechtěným prodlevám a ztratili bychom tak výhodu téměř okamžité reakce na vyvolané události. Navazování nového WebSocketového spojení a jeho autorizace pro každý stahovaný soubor není elegantní řešení. Navíc automatické otvírání spojení pro paralelní načítání dat umožňují moderní prohlížeče v rámci HTTP protokolu automaticky a bylo by zbytečné tuto funkcionalitu implementovat v rámci WebSocketu znovu. HTTP protokol je tedy lepší variantou alespoň pro implementaci API pro nahrávání a stahování souborů.

6.3.2 Autentizace a autorizace socketu

Důležité je si uvědomit, že HTTP a WebSocket fungují odděleně a tedy neznamená, že pokud uživatele autentizuji pomocí HTTP, pak máme automaticky autorizované i WebSocketové spojení.

Po úspěšné autentizaci server vytvoří sezení a vrátí v HTTP odpovědi cookie parametr SESSIONID. Pomocí tohoto SESSIONID se dále uživatel autorizuje při další HTTP komunikaci.

Koncový bod pro WebSocket však obsluhuje všechny příchozí zprávy, které jsou pomocí WebSocketu posílány a server neví, které WebSocketové spojení patří k uživateli, který se právě přihlásil.

6.3.2.1 Autentizace pomocí WebSocketu

Nyní máme ke zvážení několik možností. Pokud bychom chtěli veškerou komunikaci provádět přímo pomocí WebSocketu, pak bychom pomocí ní mohli provádět i autentizaci a na straně serveru bychom dokázali autorizovat websocketové spojení, přes které by autentizace proběhla. Toto řešení má ale několik nedostatků.

Nejprve, pokud by uživatel obnovil stránku, WebSocketové spojení by se muselo navázat znovu a znovu by bylo potřeba autorizovat socketové spojení. Kromě varianty, kdy by bylo pokaždé potřeba autentizovat uživatele, což není uživatelsky přívětivé, bylo by možné implementovat token, který by si klient někam uschoval a pomocí kterého by se znovu autorizoval. Velikou nevýhodou tohoto přístupu je ono uschování tokenu na straně klienta, neboť k našemu tokenu by se mohl dostat nějaký útočník a autorizovat se pomocí našeho tokenu. Další zabezpečování pomocí IP adresy stále nebudí příliš důvěry.

Další nedostatek vyplývá z hlavního požadavku, kterým je synchronizace slajdů, tedy obecně nahrávání a stahování souborů. Tento problém byl popsán v podsekci výše. Pokud bychom prováděli autentizaci pouze přes WebSocket,

pak bychom narazili při implementaci API pro nahrávání a stahování souborů na další problém související s autorizací. Pokud bychom toto API implementovali nad WebSocket protokolem, pak bychom museli vyřešit, jak bychom autorizovali nově vytvářená spojení určená pro přenos datově náročných dat. (Nejspíše by bylo potřeba implementovat výměnu nějakého tokenu). V případě, že bychom chtěli toto API postavit nad HTTP, pak by nastal problém v okamžiku, kdy bychom chtěli autorizovat požadavky posílané na server přes HTTP protokol.

6.3.2.2 Autentizace pomocí HTTP

Druhou možností je autentizace klasicky pomocí HTTP s následnou autorizací WebSocketového spojení. Pokud použijeme tento přístup, pak máme napůl vyhráno, alespoň pokud jde o nahrávání a stahování souborů. Implementace takového API přes HTTP je běžnou záležitostí, kterou můžeme provést pomocí běžných frameworků a autorizace je zajištěna klasicky pomocí SESSIONID.

Nyní je potřeba ještě vyřešit autorizaci WebSocketového spojení. Toto spojení je navázáno automaticky při každém načtení stránky. Autorizaci tohoto spojení můžeme provést pomocí tokenu při obnovení stránky nebo po úspěšné autentizaci. V tomto případě si jej ale nemusíme nikam ukládat, neboť i po obnovení stránky můžeme poslat HTTP požadavek pro vygenerovnání nového tokenu a ten rovnou použít pro autorizaci socketového spojení. Využijeme tedy toho, že máme alespoň HTTP kanál, pomocí kterého dokáže server klienta identifikovat.

V aplikaci je tedy implementována autentizace pomocí HTTP a autorizace WebSocketového spojení pomocí jednorázového tokenu ukládaného do dočasné cache. Po obnovení stránky je tedy odeslán požadavek na vrácení tokenu a v případě, že je uživatel již přihlášen, může token využít k autentizaci socketu. Tím se token automaticky smaže z cache.

6.4 Návrh základních funkcí aplikace

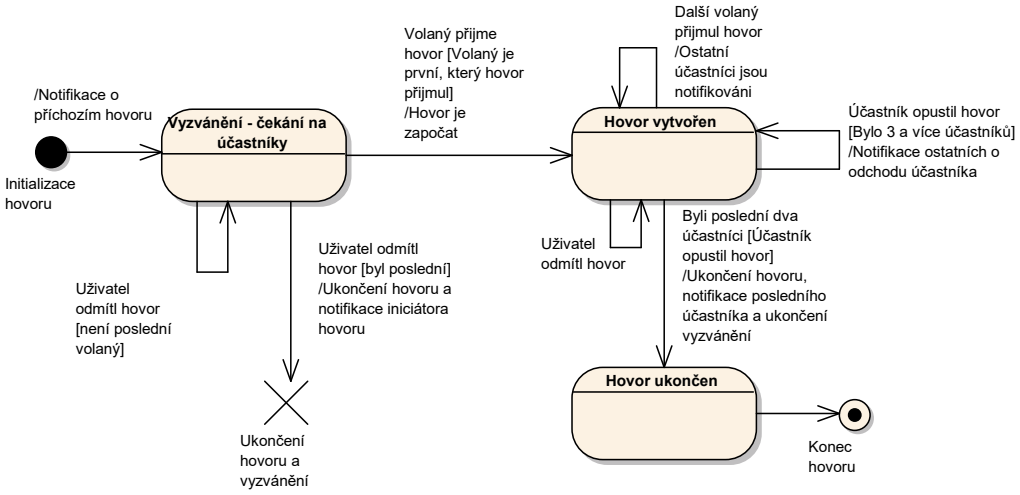
6.4.1 Typ hovorů

Aplikace umožňuje dva typy hovorů, skupinový hovor a přednášku (nebo také webinář). Tyto typy hovorů budou dále probrány z hlediska jejich možností a uživatelských rolí a dále z hlediska jejich navázání a ukončení.

6.4.1.1 Skupinové hlasové nebo video hovory

Skupinové hlasové nebo video hovory jsou prvním typem hovorů. Tyto hovory se dělí na případ, kdy je vysílán pouze hlasový signál a na případ, kdy je vysílán i vizuální signál. V obou případech v tomto typu hovoru všichni uživatelé přijímají i vysílají hlasový nebo audiovizuální signál a mají tedy stejné

možnosti. Tento typ hovoru je tedy užitečný pro různé pracovní skupiny, kde všichni komunikují se všemi a sdílí své myšlenky a informace. V rámci hovoru lze použít také slajdy. Slajdy může v tomto hovoru přidat kterýkoliv z účastníků. Slajdy, které jsou zobrazeny ostatním jsou vždy synchronizovány s tím, kdo slajdy přidal.



Obrázek 6.2: Stavový diagram pro skupinové hovory

Iniciace hovoru začíná tak, že uživatel vybere hlasový nebo video hovor. V tu chvíli se iniciátorovi zobrazí dialogové okno a uživatel čeká, až se připojí ostatní účastníci. Toto dialogové okno umožňuje iniciátorovi hovor ukončit. Zároveň jsou všichni ostatní dostupní členové skupiny notifikováni a mohou zprávu přijmout nebo odmítnout. V tuto chvíli je hovor ve stavu „Vyzvánění“, jak je znázorněno na obrázku 6.2.

V případě, že všichni účastníci odmítnou, hovor je ukončen v okamžiku odmítnutí posledního uživatel a iniciátor je o tomto notifikován. Hovor může být dále také ukončen, pokud iniciaci zruší sám iniciátor. V tom případě je třeba notifikovat všechny uživatele, kteří hovor doposud neodmítli.

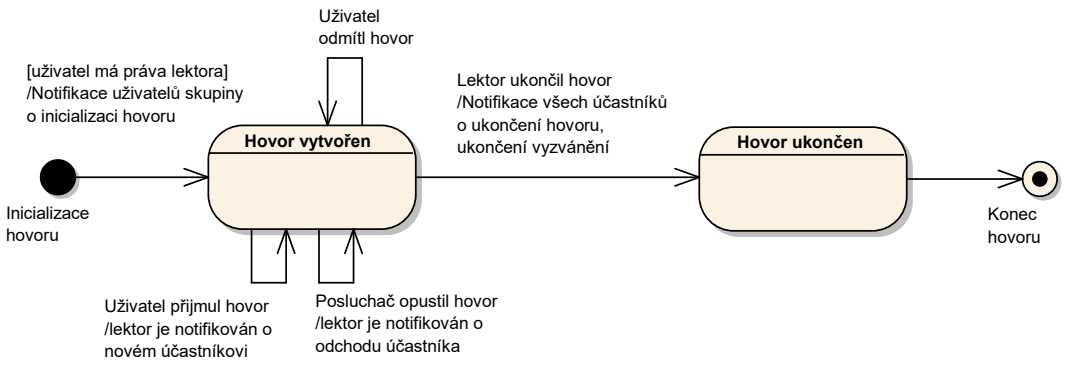
Naopak hovor je započat v okamžiku, kdy se k hovoru připojí první uživatel. V tom okamžiku je připojen také iniciátor. Další uživatelé se již připojí k existujícímu hovoru a proběhne předání informací o nově příchozím účastníkovi hovoru, aby mohlo dojít k navázání komunikace mezi všemi stávajícími účastníky a novým účastníkem.

Ukončení hovoru nastává v okamžiku, kdy již zbyli poslední dva účastníci a jeden z nich hovor ukončí. Pokud je tedy hovor započat, pak již iniciátor nemůže ukončit sám celý hovor, ale hovor je vždy ukončen odpojením předposledního uživatele. Poslední uživatel je o tomto notifikován a celý hovor zaniká.

6.4.1.2 Přednáška nebo webinář

Přednáška nebo webinář je dalším typem hovoru. Tento typ hovoru rozděluje účastníky do rolí, mezi které patří lektor, posluchač a detašovaný kameraman. Lektor má možnost vysílat audiovizuální signál a má možnost iniciovat a ukončit hovor. Dále má také jako jediný možnost přidat do hovoru slajdy. Slajdy zobrazené ostatním účastníkům jsou synchronizovány se slajdy lektora. V této implementaci není aktuálně možné přidat dalšího přednášejícího.

Může se však stát, že vysílání audiovizuálního signálu má na starost jiná osoba než přednášející. Typická situace je v přednáškových místnostech, ve kterých není taková infrastruktura, která by umožňovala přednášejícímu z přednášejícího pultu ovládat vysílání audiovizuálních dat. V tom případě bývá přenos těchto dat prováděn někým v sále, kdo ovládá kameru a kdo může být připojen do internetu. V takovém případě může přednášející vybrat uživatele, který bude pro aktuální hovor detašovaným kameramanem. Audiovizuální data bude tedy namísto lektora přenášet tento uživatel.



Obrázek 6.3: Stavový diagram pro semináře a webináře

Iniciace hovoru může být vyvolána pouze uživatelem v roli lektora. Oproti skupinovému hovoru odpadá proces vyzvánění. Ihned po iniciaci je hovor započat, iniciátor je připojen do hovoru a ostatní dostupní členové skupiny jsou notifikováni o právě vzniklém hovoru. Do tohoto hovoru se mohou připojit

jako posluchači nebo jej mohou odmítnout. Oproti předchozímu typu hovoru zde také není hovor ukončen, pokud všichni dostupní členové skupiny odmítnou tuto přednášku. Někdo nový se totiž může teprve přihlásit a k přednášce se připojit později.

Ukončit přednášku může pouze přednášející uživatel. V tom případě jsou všichni ostatní účastníci notifikováni o ukončení hovoru.

Celý tento proces je naznačen na obrázku 6.3.

6.4.2 Kontakty

Aplikace umožňuje privátní hovory a dva druhy skupin. Každý typ skupin plní jiný případ užití. Privátní hovory jsou určeny pro podporu komunikace mezi dvěma uživateli. Například pokud se dva účastníci chtějí domluvit na detailech webináře nebo přednášky a nechtějí tyto informace sdílet se zbytkem skupiny.

Pracovní skupina Aplikace mimo implementuje dva typy skupin. Pracovní skupina je skupina spolupracovníků, jejichž předpokládané audio a videohovory budou mít charakter spíše skupinového hovoru, kde budou moci všichni komunikovat dohromady. Půjde především o schůzky vzdálených pracovišť apod. V rámci skupiny tohoto typu je tedy možné iniciovat skupinové audio nebo video hovory, možnost iniciovat přednášky nebude dostupné.

Skupina má svého administrátora. Tím je nejprve uživatel, který skupinu vytvořil, zbytek uživatelů jsou v roli členů týmu. Administrátor má oproti ostatním členům týmu možnost spravovat uživatelská oprávnění v této skupině.

Skupina pro semináře a školení V rámci skupiny určené pro semináře a školení jsou definovány role administrátora, lektora a posluchače. Běžný člen skupiny je v roli posluchače. Ten nemůže iniciovat žádný hovor, může se pouze připojit k vytvořenému hovoru. Lektor má pravomoc iniciovat přednášku nebo webinář. Jiný typ hovoru v této skupině není povolený, nelze vyvolat skupinový hovor. Lektor je také jediný, který je oprávněn probíhající hovor ukončit. V rámci hovoru má lektor možnost vysílat audiovizuální data. Administrátor má role lektora a zároveň může spravovat uživatelská oprávnění v této skupině.

Kapitola 7

Implementace celkového řešení

V kapitole 6 zabývající se návrhem architektury byly rozděleny role klientské a serverové části aplikace a bylo zde popsáno, jakým způsobem budou mezi sebou jednotlivé části komunikovat. V této kapitole je dále popsáno, jak jsou jednotlivé části aplikace implementovány, jaké byly použity nástroje a frameworky. Dále je v této kapitole také popsáno, jak byly implementovány jednotlivé klíčové funkce aplikace.

7.1 Implementace klientské části aplikace

Z návrhu vyplynulo, že je vhodné použít nějaké robustní řešení na straně klienta, přičemž serverová strana bude poskytovat aplikační rozhraní a nebude se starat o vykreslování grafických komponent aplikace. Pro implementaci komplexní klientské aplikace se v dnešní době nabízí několik možností.

7.1.1 Výběr frameworku

Jednou z možností je použití komplexního frameworku Angular 2. [40] Ten poskytuje podporu pro tvorbu grafických komponent, podporu routování, injektování závislostí pomocí Dependency injection apod. Jednou z velkých výhod je jeho komplexnost, kterou umocňuje použití s další komplexní knihovnou RxJS (Reactive extensions for JS). [41] Tato knihovna poskytuje rozhraní pro implementaci návrhového vzoru Observer, který se ideálně hodí právě pro obsluhu příchozích zpráv ze serveru. Díky tomu je možné okamžitě reagovat na změnu a automaticky aktualizovat obsah již zobrazených komponent.

Další možností je použití knihovny React [42] a balíku souvisejících knihoven, se kterými se tato knihovna běžně používá. React je spojován s architektonickým návrhem Flux [43], který definuje, jakým způsobem data prochází aplikací. Pomocí těchto knihoven lze také reagovat na příchozí notifikace ze serveru a automaticky aktualizovat obsah grafických komponent.

Nevýhodou použití Angular 2 a RxJS je velikost těchto knihoven. Přestože pro použití knihovny React je potřeba poskládat více knihoven, výsledná velikost všech použitých knihoven je menší než v případě Angularu. Výhodou Angularu je naopak jeho komplexnost a také komplexnost knihovny RxJS, která byla zvolena pro obsluhu modelu. Zároveň je implementace v Angularu přímočařejší, neboť Angular poskytuje jednotnou dokumentaci, doporučuje i primární jazyk, ve kterém je také psaná dokumentace. Je tak zajištěna konzistence používaných technik a postupů. V prostředí Reactu vzniklo například schizma mezi objektovým a funkcionálním přístupem a různé ukázky jsou implementovány různým způsobem.

Komplexnost a konzistence programátorského stylu nakonec převážila a pro implementaci klientské části aplikace byl zvolen Angular 2 a RxJS.

Důsledkem volby tohoto frameworku je i volba jazyku pro implementaci klientské části, kterou je TypeScript. [44] TypeScript je jazyk postavený nad aktuální verzí ES6, přičemž ji obohacuje o možnost typování a přidává další syntaktický cukr jako jsou rozhraní, výčtové typy nebo privátní atributy tříd. Tato volba s sebou také přinesla obtíže během implementace. Pro všechny knihovny, které nejsou implementovány v TypeScriptu, ale v čistém JavaSciptu, je totiž třeba doplnit definiční soubory, které transpileru TypeScriptu říkají, jaké má daná knihovna rozhraní. Pro většinu nejpoužívanějších knihoven existuje definiční soubor v projektu DefinitelyTyped. [45] Pro knihovny, které chceme použít a pro které neexistuje definiční soubor v tomto projektu, je třeba vytvořit vlastní definiční typ [46], který popisuje jejich rozhraní. Během implementace této práce vzniklo několik definičních typů.

Příloha B se dále podrobněji zabývá problematikou související s implementací frontendové části aplikace v jazyce TypeScript. Popisuje, jak lze do této aplikace přidat další definiční typy a jak lze vytvářet vlastní, tak aby bylo možné využít další JavaScriptové knihovny, které tyto definiční typy nemají.

7.1.2 Struktura klientské části aplikace

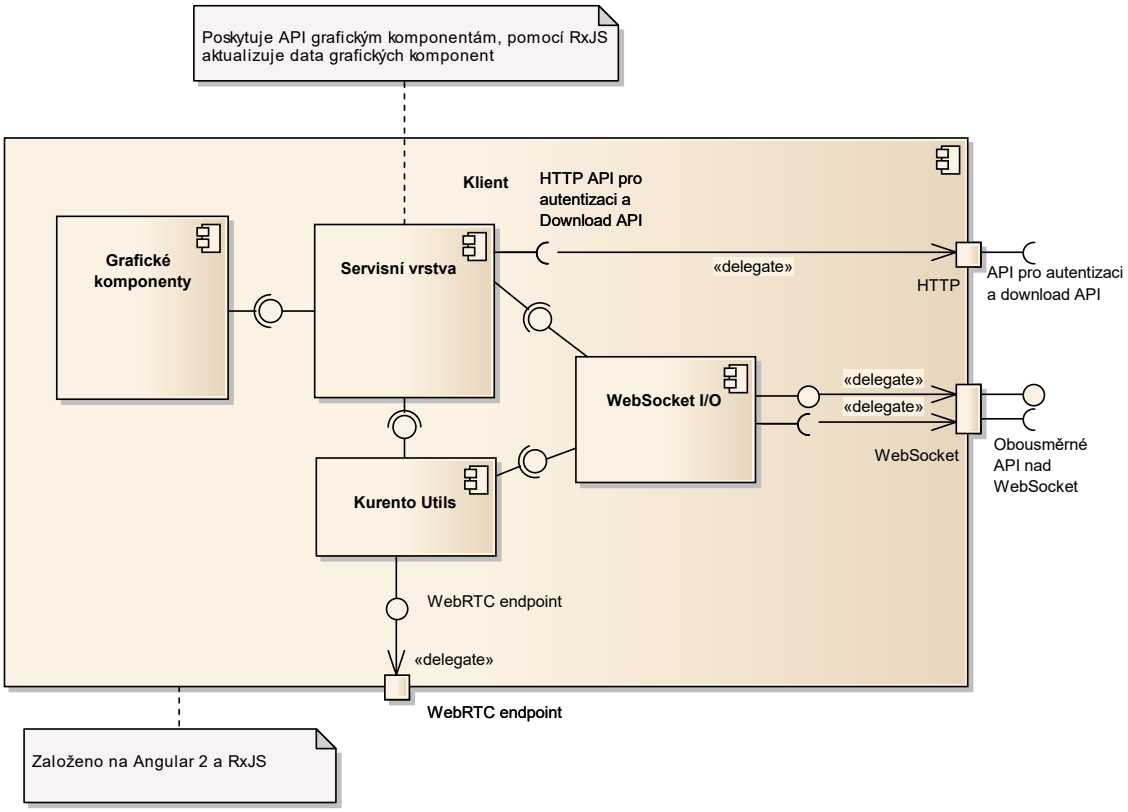
Jak je vidět na obrázku 7.1, klientská aplikace je rozdělena do několika vrstev.

7.1.2.1 Grafické komponenty

Vrstva grafických komponent implementuje jednotlivé samostatné celky aplikace. Ty se vykreslují pomocí šablonovacího systému Angularu. Komponenty si udržují základní vnitřní stav. Data aplikace jsou ale do komponent předávána buď na požádání komponenty nebo automaticky na základě událostí příchozích ze serveru.

7.1.3 Servisní vrstva

Data ze serveru předávají komponentám služby ze servisní vrstvy. Servisní vrstva poskytuje grafickým komponentám základní rozhraní pro komunikaci



Obrázek 7.1: Komponentový model popisující architekturu klientské části aplikace

a práci s daty. Služby v servisní vrstvě implementují vzor Observable, takže je možné komponenty aktualizovat automaticky na základě změny dat.

7.1.4 Komunikační vrstva

Komunikační vrstva zajišťuje komunikaci nad HTTP a WebSocket protokoly. Komunikační vrstva nad protokolem HTTP je zajištěna službou implementovanou v rámci Angular frameworku a dále tenkou službou, která zjednodušuje přenos souborů v rámci AJAXové komunikace.

Pro implementaci komunikační vrstvy nad protokolem WebSocket se nabízí dvě možnosti, které jsou popsány v následujících oddílech.

7.1.4.1 Použití podprotokolu STOMP

Jak bylo popsáno v rešeršní části, v rámci komunikace přes WebSocket je možné použít nějaký podprotokol. Byl zde zmíněn protokol STOMP. Ten se primárně používá v systémech pro řízení zpráv. Lze také použít jako podprotokol při komunikaci přes WebSocket. K tomu, aby bylo možné použít tento podprotokol, je třeba jej implementovat na serverové i klientské straně. Na straně klienta zde existuje několik knihoven. Nejpřímočařejší je knihovna STOMP.js. [2] [47] Ta bohužel již není dále spravována (pokud nepočítáme různé odnože implementující nějaké doplňující vlastnosti), nicméně jde o komplexní a zatím asi nejpoužívanější knihovnu pro práci s tímto protokolem.

Tato knihovna nepodporuje TypeScript, takže je třeba naimplementovat vlastní definiční soubor. V rámci testování možnosti použití protokolu STOMP byl vytvořen tento definiční soubor a dále se podařilo zprovoznit protokol STOMP pomocí jednoduché servisní vrstvy napsané v TypeScriptu, která knihovnu STOMP.js používá. Tato služba byla převzata z projektu ng2-stomp-demo. [48] Službu se nepodařilo vzít přesně tak jak je, nicméně na přiloženém CD v adresáři **STOMP** (viz přílohu D) je přiložena modifikace této služby včetně ukázky klienta, který tuto službu využívá, tak jak byla tato testovací implementace zprovozněna. Příloha obsahuje v adresáři **TypeScript definition files** i definiční soubor pro STOMP.js a dále také serverovou část této testovací implementace.

Ukázka implementace pomocí protokolu STOMP byla uskutečněna až v průběhu implementace jako demonstrace použitelnosti tohoto řešení. Pro implementaci tohoto protokolu by ale byla potřeba změna přístupu, takže bylo od této varianty upuštěno.

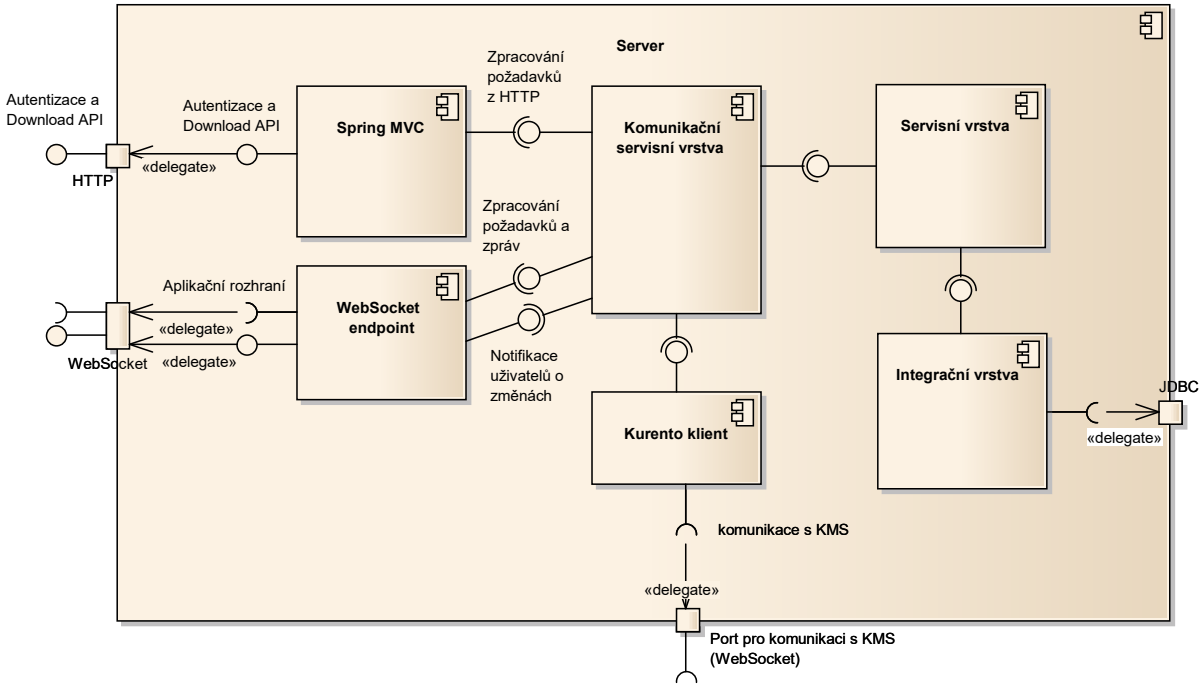
7.1.4.2 Implementace komunikace bez podprotokolu

Další možností je postavit si nad WebSocket protokolem vlastní rozhraní. V aktuální verzi aplikace je komunikační vrstva implementována jedinou třídou. Ta zprostředkovává WebSocketové spojení se serverem a umožňuje základní práci nad tímto protokolem. Obdobně jako jsou u protokolu STOMP definována témata, k jejichž odběru se lze přihlásit, implementuje tato vrstva v rámci posílané zprávy její typ a obsah. Zpráva je jednoduchý JSON se dvěma atributy.

Tato vrstva umožňuje posílat zprávu dvěma způsoby. Jednak lze poslat jednoduchou zprávu, na kterou odesílatel neočekává odpověď. To je právě způsob, jakým WebSocket funguje. Dále však také implementuje možnost poslat požadavek, na který očekává odpověď. Toho je docíleno tak, že pokud je zpráva odeslána jako požadavek, pak očekává, že do definovaného časového intervalu obdrží ze serveru zprávu se stejným typem zprávy. Pokud taková zpráva ze serveru přijde, pak je považována za odpověď na poslaný dotaz. Pokud je posláno více požadavků najednou se stejným typem zprávy, pouze

s jinými parametry v těle zprávy, pak lze definovat takzvaný deskriptor, tzn. cestu k atributu v těle zprávy, který bude porovnáván spolu s typem zprávy. Poté je zpráva ze serveru označena za odpověď na požadavek, pokud odpovídá typ zprávy a hodnota v těle zprávy, která je definovaná deskriptorem. V případě, že klient neobdrží výsledek do definovaného časového intervalu, vyvolá komunikační vrstva chybu, kterou může servisní vrstva obsloužit.

7.2 Implementace serverové části aplikace



Obrázek 7.2: Komponentový model popisující architekturu serverové části aplikace

Serverová část aplikace se rozděluje na základní serverovou aplikaci a media server. Serverová aplikace pracuje s persistentními daty, implementuje aplikační rozhraní a umožňuje komunikaci s Kurento media serverem. Media server slouží jako prostředník během audiovizuální komunikace.

Pro implementaci aplikační části byl zvolen Spring framework. Architektura této části je vícevrstvá, jednotlivé vrstvy jsou popsány v oddílech níže.

Dále je také popsáno, jakým způsobem je implementováno samotné aplikační rozhraní a jakým způsobem probíhá komunikace s media serverem.

7.2.1 Implementace aplikačního rozhraní

Jak bylo řečeno v sekci věnované implementaci klientské části aplikace, byl pro implementaci zvolen přístup s vykreslováním grafických komponent na straně klienta. Serverová aplikace tak poskytuje pouze aplikační rozhraní, které klientská část volá pomocí HTTP a WebSocket protokolu. Pro serverovou část aplikace je třeba tedy vybrat takové nástroje, pomocí kterých bude možné takové aplikační rozhraní nelépe naimplementovat. Nástroje pro serverově generování jako JSF, Struts apod. nepřichází v úvahu. Vhodným kandidátem se zdá být využití frameworku Spring s použitím Spring MVC. Spring MVC umožňuje jednoduchou implementaci REST API, podporuje snadné routování URL na metody controllerů pomocí anotací těchto metod.

Pro implementaci aplikačního rozhraní nad WebSocket protokolem lze použít také několik způsobů. Dle specifikace Java EE 7 lze využít Java API for WebSocket (JSR 356) [49]. To obsahuje ServerEndpoint, který definuje jeden přístupový bod, do kterého přichází všechny zprávy. Obsluhu zprávy a konverze dat je třeba řešit vlastní implementací. Pokud chceme odesílat zprávy různým uživatelům nebo definovaným skupinám, pak je třeba naimplementovat i obsluhu odesílání.

Výhodou jednoduchého endpointu je to, že máme přímou kontrolu nad sokety a dokážeme provázat konkrétní soket na uživatele a pokud chceme poslat zprávu nějakému uživateli, můžeme dohledat správný socket a poslat do něj požadovanou zprávu. Nevýhodou je, že musíme implementovat obsluhu a konverzi příchozích a odchozích dat. Naštěstí pro převod objektů na obyčejný JSON lze použít knihovnu GSON [50], která provádí převod automaticky. Pouze konverzi výčtových typů je třeba přesně specifikovat.

Další možností je použití Spring frameworku, který od verze 4 podporuje práci s WebSockety. [51] Ten umožňuje implementovat přístup podobný tomu předchozímu, ve kterém máme jeden přístupový bod, v němž implementujeme vlastní obsluhu příchozích a odchozích zpráv. Druhý přístup implementovaný frameworkem Spring využívá podprotokol STOMP. [52] Tímto protokolem definujeme pro každou zprávu její téma a pro obsluhu příchozích zpráv implementujeme controllery podobné těm ze Spring MVC, pouze namísto mapování metod controlleru na URL se provádí mapování na téma příchozí zprávy. Téma odchozí zprávy lze pro danou metodu controlleru popsat opět pomocí anotace nebo lze zvolit téma odchozí zprávy dynamicky. Pokud chceme poslat zprávu pouze jednomu uživateli, pak je k tématu zprávy přidán prefix definovaný jedinečným klíčem uživatele, přičemž uživatel obdrží pouze zprávy s příslušným prefixem v tématu. Na straně klienta se uživatel registruje odběru k tématu již bez specifikace tohoto prefixu. [53] [54]

Výhodou tohoto řešení je podobnost s implementací aplikačního rozhraní pomocí Spring MVC, framework za nás také řeší konverzi příchozích a odchozích dat. Matoucí však může být, jak lze autorizovat přihlášení odběru uživatelů k vybraným tématům. Ukázky znázorňují, jak se může uživatel na straně klienta připojit k poslouchání vybraného tématu. Nicméně rozhodnutí, kterému uživateli bude zpráva poslána, by mělo být učiněno již na straně serveru. Na straně serveru lze poslat zprávu konkrétnímu uživateli tak, že k tématu odchozí zprávy připojíme prefix s jedinečným klíčem pro vybraného uživatele. Pokud bychom ale chtěli posílat zprávy více uživatelům náležícím nějaké skupině, pak bychom museli najít způsob, jak přidat prefix pro celou skupinu a jak určit (a dynamicky měnit) uživatele, kterým má být zpráva pro daný prefix posílána. Dále autorizaci příchozích zpráv na základě příslušnosti k dané skupině je i v tomto případě třeba implementovat vlastními silami.

Přestože implementace pomocí Spring frameworku s podporou STOMP podprotokolu nabízí čistší a přehlednější zápis a poskytuje možnost konverze dat příchozích zpráv, pro posílání zpráv více uživatelům na základě příslušnosti k vybrané skupině, což je jedna z klíčových vlastností potřebných při implementaci komunikačního rozhraní, nebylo během zkoumání možností v tomto ohledu nalezeno žádné uspokojivé řešení.

Z tohoto důvodu byl pro implementaci této aplikace zvolen přístup s obsluhou v jednom přístupovém bodě. Z důvodu kompatibility se Spring MVC bylo toto řešení implementováno s pomocí Spring frameworku.

Jak bylo zmíněno v dřívějších kapitolách, v rámci implementace této práce byla otestována i práce s protokolem STOMP. Na přiloženém CD v adresáři **STOMP** je ukázka serverové strany této implementace.

7.2.1.1 Vrstva implementující obsluhu WebSocket protokolu

V rámci vrstvy implementující obsluhu WebSocket protokolu byla implementována služba pro odesílání zpráv a služba pro příjem zpráv.

Služba pro odesílání zpráv implementuje možnost odesílání zpráv více účastníkům zároveň na základě jejich příslušnosti k nějaké skupině. Lze tak například jednoduše poslat zprávu všem uživatelům, kteří se účastní hovoru, všem uživatelům, kteří čekají na příjem nebo odmítnutí hovoru nebo všem uživatelům vybrané skupiny. Služba využívá knihovny GSON pro transformaci DTO objektů na prostý formát JSON, který je následně odeslaný pomocí socketu.

Služba pro příjem zpráv implementuje základní možnosti autorizace (zda je uživatel účastníkem hovoru, členem skupiny nebo členem skupiny s konkrétními právy), usnadňuje načítání dat a dále umožňuje obsluhu rozdělit mezi několik příjemců na základě typu příchozí zprávy. Typ zprávy je vždy ve formátu „typ.podtyp“, takže každý příjemce má definovaný typ zpráv, kterým naslouchá a vlastní obsluhu provádí jednotlivé metody příjemce na základě podtypu zprávy.

7.2.2 Servisní vrstva

Servisní vrstva definuje jednotné rozhraní nezávisle na tom, jaký protokol nebo framework je použitý pro implementaci samotného aplikačního rozhraní. Servisní vrstva se rozděluje ještě na dvě vrstvy, které jsou označovány jako komunikační servisní vrstva a servisní vrstva pro perzistentní data. Služby druhé zmíněné vrstvy v rámci daného požadavku provádí operace nad daty, které jsou spravovány persistentním úložištěm, včetně různých transformací dat. Tuto službu dále využívá komunikační servisní vrstva. Ta potom představuje zmíněné aplikační rozhraní. Zajišťuje delegování práce s persistentními daty předchozí servisní vrstvě a dále obsluhuje komunikaci s uživateli, která slouží k notifikaci různých událostí jako jsou změny stavu nebo dat.

7.2.3 Integrační vrstva

Integrační vrstva slouží pro manipulaci s persistentními daty. V této aplikaci je implementována pomocí Java Persistence API. Tato vrstva komunikuje s MySQL databází.

7.2.4 Komunikace s Kurento media serverem

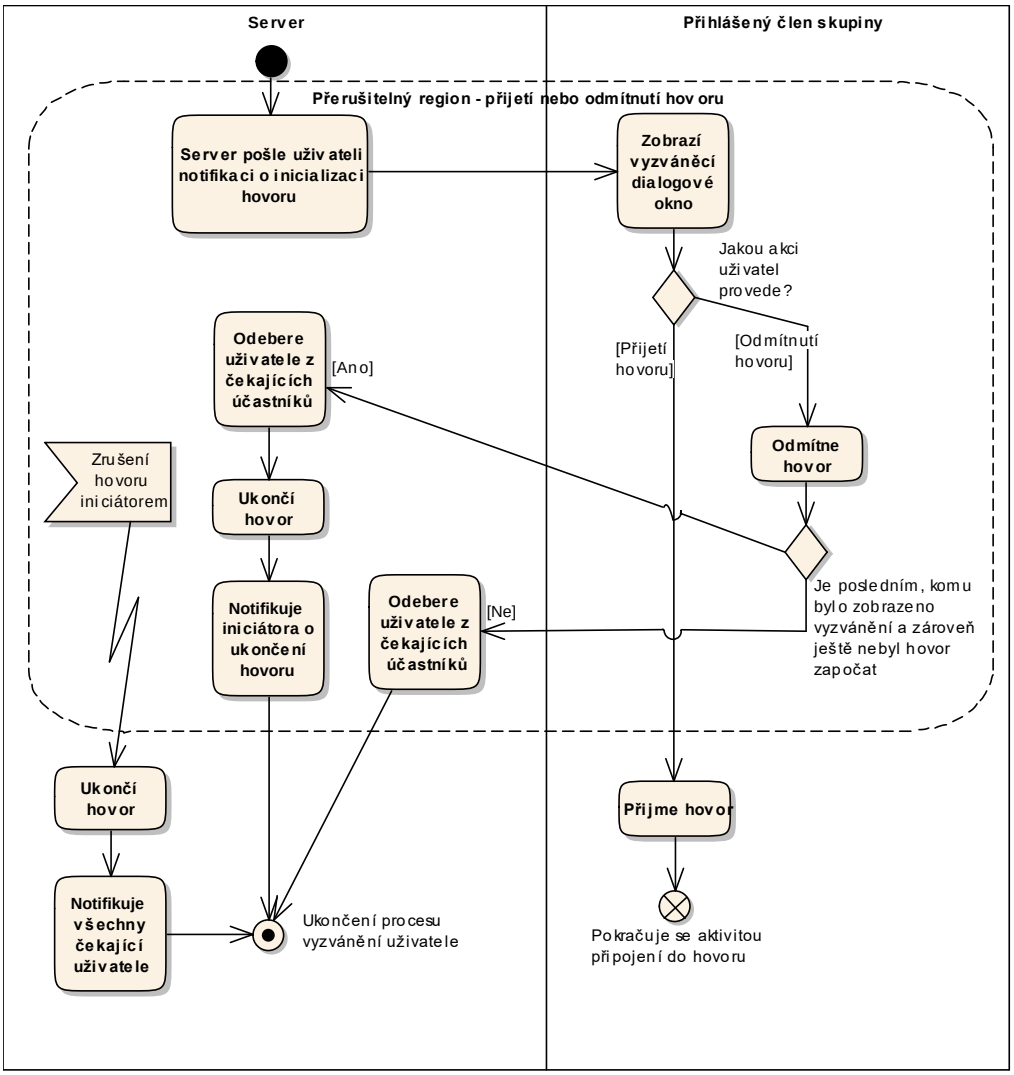
Pro komunikaci s Kurento media serverem poskytuje Kurento framework takzvané klientské API. To zajišťuje WebSocketové spojení s Kurento media serverem a zprostředkovává veškerou komunikaci potřebnou pro signalizační proces, nastavení toku dat mezi jednotlivými účastníky hovoru a další nastavení vytvořeného hovoru.

7.3 Implementace jednotlivých funkcí aplikace

7.3.1 Navázání audiovizuálního hovoru mezi mezi uživateli

7.3.1.1 Iniciace hovoru a vyzvánění

Iniciátor hovoru zvolí typ hovoru. V tu chvíli je třeba notifikovat ostatní aktivní členy skupiny o příchozím hovoru. Diagram na obrázku 7.3 znázorňuje proces vyzvánění z pohledu jednoho pozvaného uživatele. Každému pozvanému uživateli se zobrazí vyzváněcí dialogové okno. Zde má uživatel možnost hovor přijmout nebo odmítnout. Pokud uživatel odmítne hovor, pak je odebrán ze seznamu čekajících uživatelů. V případě, že je tento uživatel poslední, který má zobrazené vyzváněcí dialogové okno a hovor nebyl započat (tato situace platí pouze pro skupinový hovor, u webináře tato situace nemůže nastat, neboť hovor je započat automaticky přidáním iniciátora do hovoru), pak již není nikdo, kdo by se mohl do hovoru připojit a je třeba ukončit hovor a notifikovat iniciátora. Tomu je na základě této notifikace také zrušeno vyzváněcí



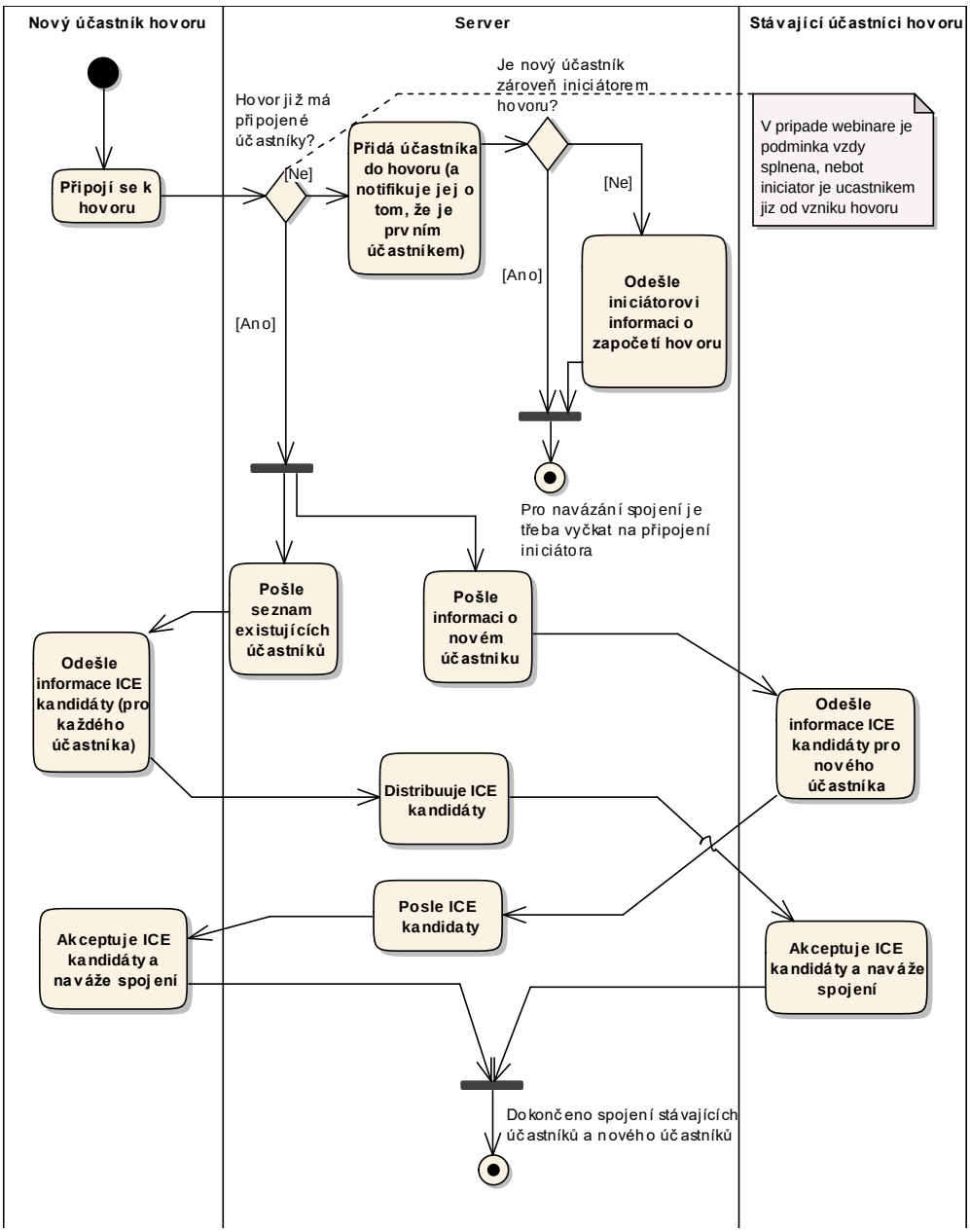
Obrázek 7.3: Diagram aktivit popisující proces vyzvánění z pohledu jednoho uživatele

dialogové okno. V případě, že uživatel přijme hovor, pak dále proce pokračuje aktivitou připojení do hovoru. Tato aktivita je znázorněna v diagramu na obrázku 7.4.

Pokud byl iniciován skupinový hovor, pak iniciátor vyčkává na potvrzení prvního uživatele. Zároveň ale může ještě před připojením prvního účastníka hovor zrušit. To je v diagramu znázorněno přerušitelným regionem a signálem „Zrušení hovoru iniciátorem“. V tom případě je hovor také ukončen a všichni uživatelé jsou notifikováni o ukončení a vyzváněcí dialogové okno je všem

uživatelům zavřeno.

7.3.1.2 Připojení účastníka do hovoru



Obrázek 7.4: Diagram aktivit popisující proces přidání uživatele do hovoru

V okamžiku, kdy se uživatel připojí k hovoru, v případě, že ještě žádný

jiný uživatel není účastníkem hovoru (v tomto případě se jedná o skupinový hovor, u webináře již je iniciátor prvním účastníkem), je uživatel do hovoru přidán a zároveň je iniciátor notifikován a na základě notifikace se také připojí k hovoru. Pokud se uživatel připojí a hovor již obsahuje nějaké účastníky, pak všem stávajícím účastníkům je poslaná informace o nově příchozím uživateli a nově příchozí uživatel obdrží zprávu, která obsahuje seznam všech stávajících účastníků.

V tuto chvíli je třeba provést výměnu ICE kandidátů, aby bylo možné navázat hovor. Každý stávající účastník pošle ICE kandidáty pro nového účastníka a nový účastník pošle ICE kandidáty pro každého stávajícího účastníka. Server provede distribuci ICE kandidátů uživatelů a media serveru a dojde k navázání spojení. Tím je proces ukončen.

7.3.2 Zobrazení a synchronizace PDF dokumentů

Jedním z hlavních požadavků je sdílení a synchronizace slajdů během hovoru přednášky. Tento problém můžeme rozdělit na několik podproblémů. Nejprve je třeba navrhnout způsob, jakým se budou slajdy nahrávat na server, jakým způsobem budou na serveru ukládány, jaké formáty bude aplikace podporovat a zda bude umožňovat nějaké konverze mezi formáty tak, aby bylo možné zobrazit co největší škálu různých formátů, které jsou používány pro tvorbu prezentací. Dále je třeba navrhnout, jak bude probíhat notifikace o jejich nahrání a notifikace během prezentace o posunu na další slajd, Nakonec je třeba zvolit, pomocí jakých technologií bude prezentace zobrazena uživateli a jakým způsobem bude moci s prezentací manipulovat.

7.3.2.1 API pro nahrávání a stahování dokumentů

Pro implementaci nahrávání slajdů do aplikace máme více variant. Můžeme využít WebSocket protokol, který je implementovaný v rámci aplikace. Jak bylo ale popsáno v kapitole (výše), bylo by nutné pro přenos každého souboru otevřít nový socket, aby nedošlo k blokování spojení po dobu, než bude soubor přenesen na server nebo ze serveru k uživateli. Pokaždé bychom ale museli socket autorizovat. Pomocí WebSocketu můžeme posílat i binární data, ale bylo by potřeba navrhnout, jak ke zprávě s binárními daty přidat nějaká metadata - např. k jaké konverzaci je dokument určen, jak se jmenuje, apod. Implementace pomocí WebSocketu tedy není ideální.

Další variantou je využít standardně HTTP protokol a nad ním postavit jednoduché API. Jelikož je již Spring MVC použitý pro autentizaci, můžeme jednoduše autorizovat každý požadavek na toto API a zároveň nám odpadne potřeba ručně vytvářet sockety - stačí využít AJAX volání tohoto API. Metadata o nahrávaném nebo stahovaném API můžeme zjsitit z HTTP hlaviček (např. název souboru) nebo z URL, kteoru voláme (např. specifikace, jaké konverzace se dokument týká).

V rámci této práce jsem tedy zvolil implementaci pomocí Spring MVC, které umožnilo implementovat toto API v souladu se standardy REST API, pouze s tím rozdílem, že vyžaduje, aby byl uživatel přihlášený k aplikaci.

Pokud se zaměříme na funkční požadavky na API, mělo by umožnovat nahrávání a stahování slajdů, dále by toto API mohlo být rozšířeno na nahrávání a stahování obecných souborů. Slajdy by mohly být dostupné také po skončení prezentace a tedy by s nimi dále mohlo být zacházeno jako s jiným obyčejným souborem.

7.3.2.2 Konverze PPTX a ODF formátů do PDF

Konverze nakonec nebyla z časových důvodů a z důvodů složitější implementace implementována. Slajdy je možné nahrát ve formátu PDF.

7.3.2.3 Notifikace spojené se synchronizaci slajdů

Notifikace nejprve proběhne v okamžiku, kdy jsou slajdy do hovoru přidány. Další notifikace jsou účastníkům rozesílány ve chvíli, kdy uživatel, který slajdy vložil, posouvá stránky slajdů. Tyto notifikace zajistí vyvolání automatického posunu na další stránku i u ostatních uživatelů. Tím je zaručena synchronizace.

7.3.2.4 Zobrazení PDF

V rámci klientské části aplikace byla implementována komponenta pro synchronizovatelné slajdy pro framework Angular 2. Tato komponenta využívá jinou komponentu pro Angular 2, která implementuje knihovnu PDF.js, která byla zmíněna v rešeršní části.

Uživatel má možnost v průběhu hovoru slajdy různě procházet, takže může být na jiné stránce, než je právě prezentující uživatel. Komponenta, která slajdy implementuje, poskytuje tlačítko, kterým se sesynchronizuje zpět na stránku, na které se nachází prezentující. V okamžiku, kdy byl uživatel na jiném slajdu a mezitím přednášející přešel na další slajd, tlačítko pro synchronizaci začne blikat, aby tak bylo zřejmé, že se prezentace posunula a že by se mohl chtít sesynchronizovat zpět.

Tato komponenta může být použita i jinde, což lze aktuálně i v implementované aplikaci. Při nahrávání slajdů má uživatel na výběr, zda chce slajdy nahrát jen pro daný hovor, nebo zda se mají uložit i do celé konverzace. Pokud zvolí variantu s uložením do konverzace, v chatu se zobrazí zpráva se dvěma tlačítky. Jedno slouží pro stažení PDF a druhé pro opětovné zobrazení slajdů pomocí této komponenty.

7.3.3 Zobrazení videa na stránce a ve full-screen módu

Pokud jsou v rámci hovoru zobrazeny slajdy a zároveň je třeba uživateli zobrazit video stream, pak má uživatel možnost přepínat mezi třema různými módy.

První zobrazuje video a slajdy vedle sebe. Druhý mód upřednostňuje slajdy a video zobrazuje malé v pravém dolním rohu a třetí mód naopak upřednostňuje video a zobrazuje malé slajdy na témže místě.

Pokud je vytvořen skupinový hovor, pak se zobrazují videa všech účastníků stejně veliká. Layout je vždy přepočítán tak, aby mohla být videa zobrazena co největší. V okamžiku, kdy jsou ve skupinovém hovoru účastni alespoň 4 účastníci, zobrazí se další dvě možnosti. Obě možnosti zobrazují jedno hlavní video a dále seznam malých videí ve spodní části, přičemž v první variantě si uživatel vybírá ručně kliknutím, které video chce vidět veliké, zatímco v druhé variantě je vždy hlavní video voleno dynamicky na základě toho, který z účastníků právě hovoří.

Layout se přizpůsobí i v okamžiku, kdy jsou slajdy nahrány v rámi skupinového hovoru. Ukázky, jak daný layout vypadá, lze nalézt v příloze A

Pro všechny tyto layouty platí, že je lze zobrazit ve full-screen módu. Je možné definovat, jaký element má být zobrazen na full-screen. v aplikaci se do fulls-creen módu převede blok s videi se slajdy a navigační lištou.

7.3.4 Vypnutí zvuku na straně posluchače

V rámci implementace bylo implementováno i tlačítko, které umožňuje vypnout zvuk pro všechny příchozí video elementy najednou.

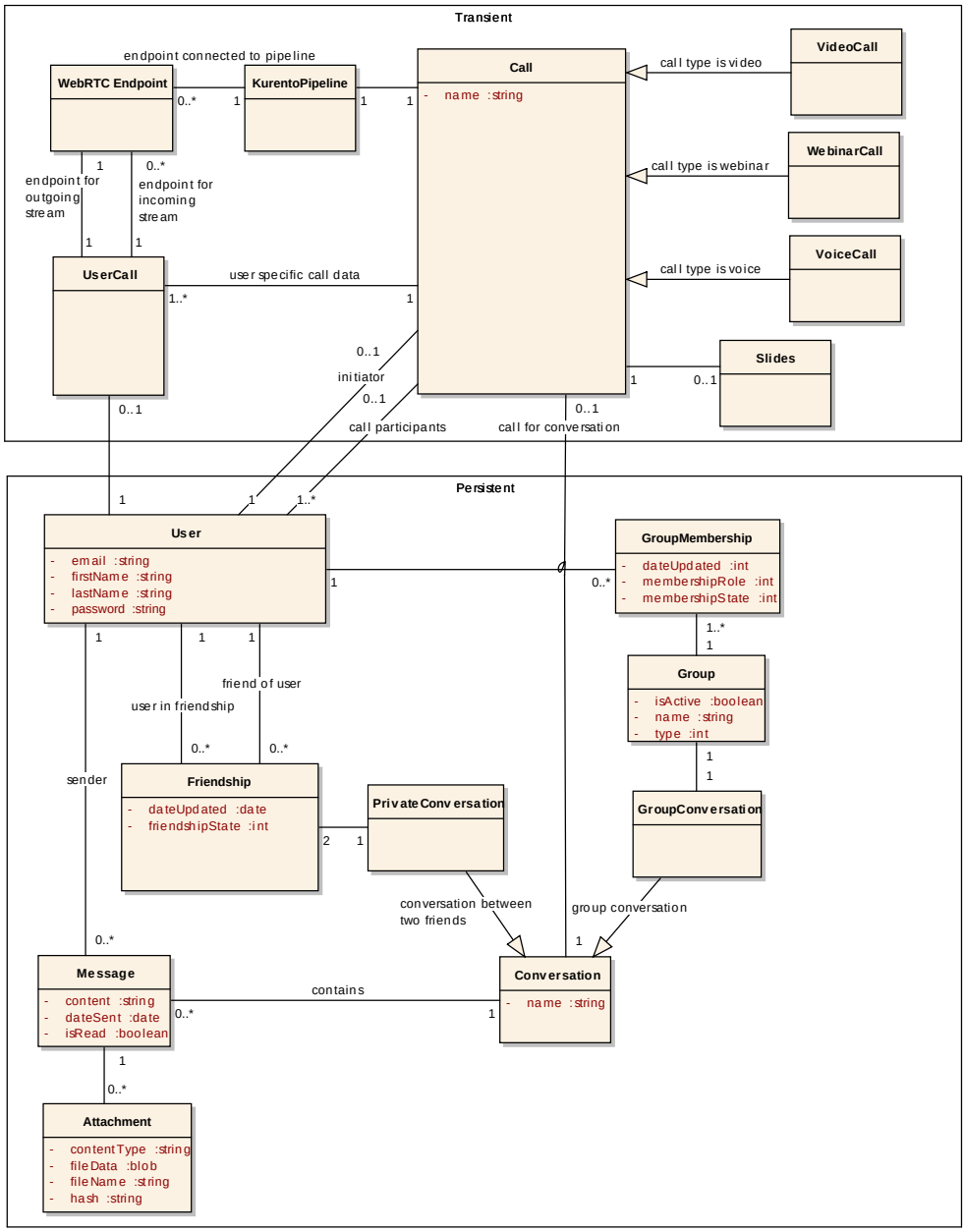
7.4 Detekce řeči

Jak bylo zmíněno v rešeršní části, pro detekce řeči lze použít na straně prohlížeče knihovnu hark. Ta poskytuje jednoduché rozhraní, které umožňuje nastavit požadovanou hlasitost v decibelech a následně zaregistrování obsluhy událostí „onspeak“ a „onstopspeak“.

Pro tuto knihovnu však neexistoval pro TypeScript definiční soubor. Ten byl vytvořen až v rámci této práce. Tento definiční soubor je pro další možné použití uložený na přiloženém CD v adresáři „Attachments/Typescript definition files“.

7.5 Implementace základních domén aplikace

V této sekci jsou znázorněny základní domény aplikace. Jak je vidět z obrázku 7.5, model je rozdělen na dvě části. V první části jsou data, která jsou persistentní. Jsou to data týkající se skupin, členství uživatelů v těchto skupinách a přátelství mezi uživateli a konverzace včetně zpráv. Persistentní část modelu vlastně říká, že živatel má možnost mít členství ve skupině nebo být v přátelství s jiným uživatelem. V rámci každé skupiny je možné vést konverzace. Tato data se ukládají do databáze.



Obrázek 7.5: Doménový model

Naproti tomu v průběhu hovoru vznikají dočasná data, která souvisejí s probíhajícím hovorem. V rámci hovoru si pamatujeme příslušnost uživatele k probíhajícímu hovoru, přičemž pro každého uživatele je třeba mít možnost přistupovat k WebRTC koncovým bodům. Dále je k hovoru možné přidat slajdy.

7.6 Další podpůrné technologie pro vývoj

Pro vývoj především klientské strany, bylo použito několik technologií, které pomáhají zpracování zdrojů. Jde především o kompilování TypeScriptových souborů do JavaScriptu a jejich spojení do jednoho souboru, kompilace SASS souborů do CSS a spojení do minifikovaného souboru, minifikace obrázků a nástroj pro tvorbu spritů. Všechny tyto nástroje jsou implementovány pomocí nástroje Gulp. Adresář se zdrojovým kódem projektu obsahuje také README.md a celý podadresář „readme“, který obsahuje návody, jak tyto nástroje v tomto projektu použít. Podrobnější popis, jak je implementováno zpracování těchto zdrojových souborů, je v příloze B.

Kapitola 8

Testování

Testování aplikace bylo provedeno ze dvou různých hledisek. Jednak bylo provedeno uživatelské testování na základě předem definovaného scénáře. Druhé testování slouží k popisu.

8.1 Testování použitelnosti

V rámci této práce bylo stanoveno několik testovacích scénářů, které měli uživatelé otestovat. Tyto scénáře jsou popsány níže a poté následuje vyhodnocení testování. To popisuje, zda vykonání kroku scénáře činilo uživatelům problémy a dále uvádí způsob a náročnost nápravy.

8.1.1 Testovací scénáře pro uživatelské testování

Registrace

1. Zaregistrujte se do aplikace
2. Požádejte o přístup do skupiny „XYZ“
3. Po schválení přejděte do skupiny a napište, že jste zde

Provedení Webináře

1. Přijměte příchozí hovor
2. Zobrazte si prezentaci tak, abyste měli jako hlavní slajdy a přednášejícího v pozadí
3. Vyzkoušejte různé možnosti zobrazení
4. Přejděte do chatu, napište zprávu
5. Vraťte se do hovoru

6. Minimalizujte okno s přednáškou (aniž byste ji ukončili) a opět zvětšte. Poté zobrazte ve fullscreen módu

7. zkuste si v průběhu přednášky vrátit ve slajdech o několik stránek zpátky a poté se vrátit tam, kde je aktuálně přednášející

8. Ukončete hovor a stáhněte si slajdy z přednášky

Provedení skupinového hovoru

1. Vyvolejte skupinový video hovor

2. Nahrajte slajdy tak, aby bylo možné si je prohlédnout i po skončení hovoru.

3. Pokud je 4 a více uživatelů v hovoru, vyberte si jednoho, který má být zvětšen

4. Pokud je 4 a více uživatelů v hovoru, nastavte zobrazování tak, aby byl vždy zobrazen velký stream uživatele, který právě hovoří

Správa skupiny

1. Založte si vlastní skupinu specializovanou na webové semináře

2. Přijměte jako administrátor skupiny pozvánku od zájemce o členství.

3. Zobrazte si přehled členů skupiny

8.1.2 vyhodnocení

Nyní zde budou popsány největší potíže, které se opakovaly a dále bude popsána možná náprava. Některé potíže také pramenily z toho, že aplikace není dokončená jako celek a testovatelné jsou jen některé zde popsané scénáře.

Registrace

1. **Po odeslání požadavku a schválení není zřetelná notifikace**. V době testování se zobrazil pouze zvoneček s číslem. Někteří testovací uživatelé si toho nevšimli. Proto byla přidána i flash message.

2. **Hláška „Request sent“ zůstává u skupiny i v okamžiku, kdy přijde notifikace o potvrzení**. To vedlo ke zmatení uživatele. Nápravou by mělo být zmizení této skupiny z nabídky po notifikaci o potvrzení.

Provedení webináře a skupinového hovoru

1. **Zde se jako největší problém jeví nepochopení významu ikonek**. Výsledkem bylo postupné projíždění jedné po druhé a zobrazování tooltipů. Možné řešení by mohlo být úvodní tutoriál po registraci, popřípadě rychlejší zobrazování výraznějších tooltipů.

2. **Při zobrazení velkých slajdů video překrývá značnou část obrazovky**. Řešením by mohlo být minimalizace videa při dvojitém poklepání na video. Totéž platí pro případ, kdy je zobrazeno velké video a malé slajdy.

3. **Pokud klikám na slajdech na šipky doleva nebo doprava a nehýbu myší, tak lišta zmizí**. Nápravou by mohlo být, že pokud bude kurzor uživatele nad lištou, pak se lišta nebude schovávat.

4. **Ikonka se synchronizací u slajdů by se neměla schovávat.** Ikonka sice notifikuje, že přednášející změnil slajd, ale uživatel si toho nemá šanci všimnout, pokud na slajdy nepřejede myší.

5. **Při minimalizaci okna si několik lidí nevšimlo, že lišta s aktuálním hovorem zůstala na horní části obrazovky**. Řešením by mohlo být zvýraznění lišty, popřípadě výraznější blikání ikonky symbolizující kameru.

Správa skupiny

1. **lišta s detailem se automaticky po nějakém čse sroluje po pouhém najetí myši**. Původně to bylo myšleno aby si toho uživatel všimnul, ale tato funkce by měla být odebrána.

8.2 Testování v závislosti na propustnosti sítě

Druhý typ testů slouží ke zhodnocení použití a implementace technologie WebRTC pokud jde o adaptibilitu na změny v propustnosti sítě. To znamená, že nás zajímá, zda a případně jakým způsobem je adaptibilita implementována a zároveň bychom si toto tvrzení chtěli potvrdit pomocí kvantifikovaných měření.

8.2.1 Veličiny ovlivňující kvalitu audiovizuálních dat

Pokud chceme popsat kvalitu přenášených dat, musíme se zaměřit především vstupní veličiny jako je rozlišení zdrojového obrazu (tzn. jeho výška a šířka), frame rate (počet snímků za sekundu) nebo bitrate (přenosová rychlost v bitech za vteřinu).

Jak již bylo popsáno předchozích v kapitolách, při navazování peer-to-peer spojení pomocí technologie WebRTC lze specifikovat požadavky na přenášená data. V případě videa jde především o velikost zdrojového obrazu (tzn. výška a šířka zdrojového videa, které je získáváno z kamery) a snímkovou frekvenci. Tyto požadované jsou konstantní v průběhu přenosu dat a WebRTC se snaží těmto parametrům co nejvíce vyhovět. Pokud bychom chtěli tyto parametry upravit, pak je třeba buď ukončit stávající přenos dat a navázat nové spojení, nebo lze použít metodu renegociace, což znamená, že je třeba mezi účastníky znovu poslat SDP zprávy se vstupními parametry a nahradit stávající audiovizuální streamy a nové odpovídající těmto parametrům. Veličina, která určuje, jaký objem dat se bude sítí přenášet, je přenosová rychlost (bitrate). Ta je dopočítávána dynamicky v závislosti na námi zvolených hodnotách vstupních veličin a na aktuální propustnosti sítě. Výsledky tohoto testu ukazují, že po navázání spojení dochází k hledání optimálního bitrate. WebRTC to provádí tak, že nejprve nastaví bitrate vysoký a nízký a poté začne bitrate zvyšovat tak, aby byla maximalizována kvalita přeneseného audiovizuálního streamu a zároveň aby byla minimalizováno zpoždění signálu.

Během tohoto testování byla provedena měření s různými vstupními hodnotami rozlišení a snímkové frekvence a bylo sledováno jakým způsobem se bitrate dynamicky mění.

8.2.2 Získání aktuálních statistik

Technologie WebRTC popisuje v rámci své specifikace také API pro získání statistik [55], které umožňují sledovat aktuální parametry přenášených dat a parametry přenosu nad transportním protokolem RTP. WebRTC vedle transportního protokolu implementuje RTCP protokol (RTP Control protocol), který sám žádná audiovizuální data nepřenáší, ale slouží k poskytování zpětné vazby a umožňuje kontrolovat kvalitu poskytované služby.

Statistiky lze získat nad různými zdroji, přičemž každý zdroj poskytuje data pro jiné veličiny. Nad koncovými body lze sledovat statistiky pro komunikaci nad RTP protokolem (obsahuje různé vlastnosti pro koncový bod pro příchozí nebo odchozí datový tok). Dalším zdrojem může být například datový kanál nebo kanály pro audiovizuální data. Na kanály pro audiovizuální data lze sledovat aktuální snímkovou frekvenci, počet ztracených nebo poškozených snímků apod.

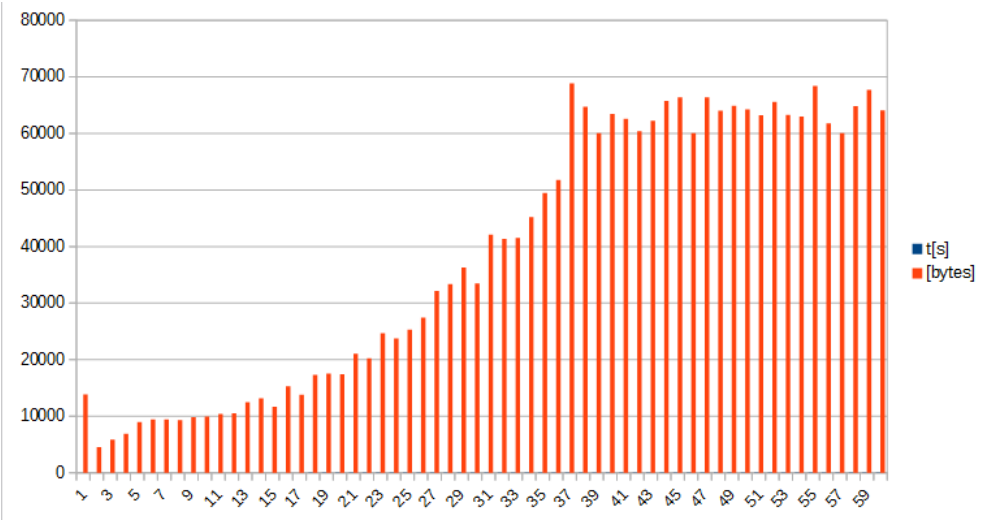
Tyto statistiky lze získat na straně klientské (pomocí javascriptového API v prohlížeči) a také Kurento media server poskytuje možnost pro získání těchto dat nad svými koncovými body, přičemž formát získávaných statistik je v souladu se specifikací definovanou v rámci WebRTC. [56] Data, která lze takto získat, se v případě klientské strany i v případě strany media serveru aktualizují jednou za vteřinu. V rámci této práce bylo implementováno získávání statistik na obou stranách.

V aplikaci na straně serveru byl vytvořen v rámci WebSocketového API přístupový bod, kterému lze předat identifikátor konverzace a odpovědí na toto volání je seznam všech dostupných veličin pro všechny příchozí i odchozí koncové body WebRTC komunikace pro danou konverzaci.

Na straně klienta byla vytvořena služba, kterou lze po zahájení hovoru spustit získávání statistik. Tyto statistiky získává pravidelně v definovaném intervalu. Tyto statistiky obsahují mimo jiné přenesený počet bytů. Jde o kumulativní hodnotu, přesto jsme schopni z této hodnoty v každém čase spočítat přenosovou rychlost.

8.2.2.1 Adaptace přenosové rychlosti

Pomocí statistik na klientské straně byla otestována adaptace na propustnost sítě. Graf 8.1 znázorňuje přenosovou rychlost v průběhu první minuty. Z grafu je vidět, jakým způsobem se WebRTC adaptuje na optimální přenosovou rychlost. Od nízkých přenosových rychlostí postupně zkouší vyšší rychlosti až narazí na optimální mez. Odpovídá to také tomu, že obraz videa je zprvu velmi nekvalitní a postupně se kvalita zlepšuje až narazí na optimální rychlost.



Obrázek 8.1: Znázornění adaptace přenosové rychlosti v průběhu první minuty spojení

Závěr

V rámci této práce byly prozkoumány některé aplikace, které se zabývají videokonferenčními hovory nebo poskytováním různých webinářů. Některé jsou čistě desktopové, některé lze použít v prostředí prohlížeče, avšak k tomu implementují vlastní doplňky do prohlížečů a některé již využívají nově vznikajícího webového standardu pro přenos audiovizuálních médií.

V rámci této práce jsem se rozhodl implementovat nově vznikající standard. Spolu s tímto standardem vzniká celá řada nástrojů, které implementaci této nové technologie usnadňují. Tyto nástroje byly prozkoumány a jeden z nich (Kurento media server) byl také použit jako stežejní prvek celé architektury.

Díky této technologii a dalším podpůrným webovým specifikacím je možné přistupovat ke kameře a posílat tak v reálném čase tok dat získaný touto kamerou ostatním účastníkům. Distribuci tohoto záznamu mezi ostatní účastníky hovoru zajišťuje použitý media server. Uživatel může vysílat vlastní audiovizuální data a zároveň může přijímat několik různých datových toků od ostatních účastníkům. Tím je zaručen duplexní přenos zvuku i videa. Zároveň je možné na straně posluchače zvuk dočasně vypnout. Možným dalším rozšířením by bylo umožnit zvuk dočasně vypnout i na straně vysílajícího. Rozhraní, kterým lze přistupovat ke kameře a mikrofonu, umožňuje zvolit, jaký typ médií chce uživatel přenášet. Toho bylo využito například v případě implementace hlasových hovorů. Při nich dochází pouze k přenosu zvuku bez videa.

Pro vykreslení slajdů byla vytvořena znovupoužitelná komponenta, která je postavena nad projektem společnosti Mozilla. Tato komponenta umožňuje pohyb v prezentaci po jednotlivých stránkách a dále umožňuje synchronizaci mezi prezentujícím a posluchači. Posluchači se tak automaticky zobrazí aktuální slajd, kdykoliv prezentující přejde na další stránku. Posluchač má také možnost během přednášky ve slajdech listovat, přičemž jedním tlačítkem se opět může sesynchronizovat s přednášejícím. Pokud přednášející mezitím přejde na další slajd, komponenta notifikuje uživatele a ten má opět možnost se synchronizovat s přednášejícím. Díky implementaci komunikace pomocí Web-

Socket protokolu, který umožňuje plně duplexní přenos mezi klientem a serverem, je synchronizace prováděna s minimálním zpožděním. Slajdy je možné také nahrát tak, aby byly dostupné i po skončení hovoru. V tom případě je lze nalézt v konverzaci, v rámci které byl hovor prováděn.

Dále bylo implementováno několik možných rozvržení, mezi kterými si účastníci hovoru mohou v průběhu hovoru vybírat. Celá tato rozvržení je možné zobrazit i v módu na celou obrazovku, aby tak byla využita co největší plocha displeje. Možná rzvržení je závislá na tom, jaký typ hovoru je prováděn. Aplikace umožňuje provádět přednášku nebo skupinový hovor. Během přednášky posluchači sledují video s přednášejícím a současně slajdy s prezentací. Aplikace v tomto případě umožňuje zobrazit video i slajdy vedle sebe, dále umožňuje zobrazit slajdy jako hlavní s malým zobrazením videa nebo naopak video s přednášejícím jako hlavní s malým zobrazením slajdů. Naproti tomu během skupinového hovoru je třeba zvolit rozvržení takové, aby bylo možné zobrazit videa pro dynamický počet účastníků. V tomto případě je opět možné zvolit z několika variant. První variantou je dlaždicové zobrazení, při kterém jsou všechny příchozí videa zobrazena stejně veliká a vyplňují co nejvíce volného prostoru. Další zobrazení jsou možná v případě, kdy jsou alespoň 4 účastníci hovoru. V tom případě je možné zobrazit video jednoho účastníka velké a ostatní jsou zobrazeny jako menší náhledy. Uživatel má možnost vybrat, které video si chce zobrazit jako hlavní a konečně má také možnost zvolit variantu, kdy se bude hlavní video měnit dynamicky a bude se volit podle toho, kdo právě hovoří.

Pro poslední zmíněný dynamický způsob zobrazení příchozích video streamů opět využívá nová webová rozhraní za pomoci knihovny, která toto API volá a zjednodušuje. Zároveň je to ukázka toho, jak je tento ekosystém nových webových technologií rozsáhlý a umožňuje implementovat nejrůznější požadavky, které jsou s přenosem audiovizuálních dat spojeny. Těmto technologiím a jejich možnému využití pro rozšíření funkcionality stávající aplikace se věnuje Kapitola 5.

Všechny tyto layouty je možné zobrazit i v módu full-screen, aby byla využita co největší plocha displeje.

V rámci možností hovoru byla také implementována možnost ztlumit zvuk pro všechny příchozí video streamy najednou.

V rámci testování byla zhodnocena implementace přenosu audiovizuálních dat pomocí WebRTC z pohledu propustnosti sítě. Z testů je patrné, jak během přenosu dat pomocí WebRTC dochází k adaptaci na propustnost sítě, přičemž je to patrné zejména v okamžiku, kdy je přenos těchto dat zahájen a dochází k hledání optimální přenosové rychlosti.

Součástí aplikace bylo také uživatelské testování, které odhalilo několik nedostatků. Některé z těchto nedostatků byly opraveny, u ostatních bylo zhodnoceno, jaký dopad měly na výslednou použitelnost a popsána náročnost nápravy.

Celkový přínos této práce by měl být především v široké analýze různých

možností, které nové webové technologie a nová rozhraní v oblasti přenosu audiovizuálních dat umožňují, dále v analýze, která se snažila nalézt vhodnou technologii pro implementaci komunikace v reálném čase (především pro potřeby synchronizace slajdů a interaktivní diskuse doplňující audiovizuální hovor). Přínos implementace by měl být především v tom, že se nesnaží roubovat nové technologie na staré zaběhnuté postupy, ale snaží se implementovat nové nástroje, nové frameworky, které mohou odrážet dynamické vlastnosti této aplikace.

Použití nových technologií však také znamená i některé problémy z počátečního nedostatečného pochopení. Stejně tak tomu bylo i během návrhu a implementace této aplikace. Především pro komunikaci mezi klientem a aplikačním serverem pomocí WebSocket protokolu by v budoucnu bylo třeba implementovat více robustní řešení. V rámci této práce zvažuji možnost použití podprotokolu, který umožňuje takové věci jako je potvrzení přijetí zpráv, autorizaci WebSocketového spojení, které je nyní implementováno vlastními silami apod. V textu je popsán alespoň postup, jakým způsobem lze tento podprotokol implementovat a obsahuje dále jednoduchou ukázkovou implementaci na straně serveru i klienta.

V rámci implementace se z důvodu komplexnosti nedostalo na některé vlastnosti, které byly navrženy v rámci požadavků na aplikaci. Například v rámci webového semináře možnost volby externího kameramana nebo přidání druhého přednášejícího. Aplikace však implementuje typy skupin, v rámci kterých lze provádět skupinové hovory nebo webináře a definuje role, které v rámci hovoru může zastávat. Je tak připravena pro možnost rozšíření. Také některé podpůrné požadavky, které nesouvisejí tolik se samotným tématem práce, avšak tvoří finální produkt použitelným (některé notifikace, úprava uživatelských rolí v rámci skupiny), nebyly z časové náročnosti doimplementovány. Důraz byl kladen především na hlavní požadavky, které souvisí s přenosem médií a jeho zobrazením.

Literatura

[1] Mozilla Developer Network and individual contributors: *XMLHttpRequest - Web APIs*. 2017, [cit. 2017-02-03]. Dostupné z: `https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest`

[2] Mesnil, J.: *jmesnil/stomp-websocket: Stomp client for Web browsers and node.js apps*. 2017, [cit. 2017-02-04]. Dostupné z: `https://github.com/jmesnil/stomp-websocket`

[3] Atmosphere: *Atmosphere/atmosphere: Realtime Client Server Framework for the JVM, supporting WebSockets with Cross-Browser Fallbacks*. 2017, [cit. 2017-02-04]. Dostupné z: `https://github.com/Atmosphere/atmosphere`

[4] Atmosphere: *Atmosphere/atmosphere-stomp: Stomp implementation*. 2017, [cit. 2017-02-04]. Dostupné z: `https://github.com/Atmosphere/atmosphere-stomp`

[5] Alinone, A.: *Push Technology, Comet, and WebSockets: 10 years of history from Lightstreamer's perspective*. 2011, [cit. 2017-02-04]. Dostupné z: `http://blog.lightstreamer.com/2011/07/push-technology-comet-and-websockets-10.html`

[6] Lubbers, P.: *How HTML5 Web Sockets Interact With Proxy Servers*. 2010, [cit. 2017-02-04]. Dostupné z: `https://www.infoq.com/articles/Web-Sockets-Proxy-Servers`

[7] Verizon digital media services: *HDS, HLS, HSS — Adaptive HTTP Streaming Demystified*. 2013, [cit. 2017-02-04]. Dostupné z: `https://www.verizondigitalmedia.com/blog/2013/07/hds-hls-hss-adaptive-http-streaming/`

[8] developer network, M.; individual contributors: *Transcoding assets for Media Source Extensions*. 2016, [cit. 2017-02-04]. Dostupné

z: https://developer.mozilla.org/en-US/docs/Web/API/Media_Source_Extensions_API/Transcoding_assets_for_MSE

[9] Wolenetz, M.; Smith, J.; Watson, M.; aj.: *Media Source Extensions^{TM}*. 2016, [cit. 2017-02-04]. Dostupné z: https://www.w3.org/TR/media-source/

[10] Lederer, S.: *Transcoding assets for Media Source Extensions*. 2016, [cit. 2017-02-04]. Dostupné z: https://www.smashingmagazine.com/2016/04/html5-media-source-extensions-bringing-production-video-web/

[11] Sullivan, S.; Winzeler, L.; Brown, D.; aj.: *Programming with the Java media framework*. John Wiley & Sons, Inc., 1998.

[12] Bergkvist, A.; Burnett, D. C.; Jennings, C.; aj.: *WebRTC 1.0: Real-time Communication Between Browsers*. 2016, [cit. 2017-02-04]. Dostupné z: https://www.w3.org/TR/webrtc/

[13] Alvestrand, H.; Håkansson, S.: *Web Real-Time Communications Working Group*. 2016, [cit. 2017-02-04]. Dostupné z: https://www.w3.org/2011/04/webrtc/

[14] Sergiienko, A.: *WebRTC Cookbook*. Packt Publishing Ltd, 2015.

[15] Stewart, R.: *RFC 4960 - Stream Control Transmission Protocol*. 2007, [cit. 2017-02-04]. Dostupné z: https://tools.ietf.org/html/rfc4960

[16] Ristic, D.: *WebRTC data channels: WebRTC data channels for high performance data exchange - HTML5 Rocks*. 2014, [cit. 2017-02-04]. Dostupné z: https://www.html5rocks.com/en/tutorials/webrtc/datachannels/

[17] Burnett, D. C.; Bergkvist, A.; Jennings, C.; aj.: *Media Capture and Streams*. 2016, [cit. 2017-02-04]. Dostupné z: https://www.w3.org/TR/mediacapture-streams/

[18] Mozilla Developer Network and individual contributors: *MediaDevices.getUserMedia() - Web APIs*. 2017, [cit. 2017-02-04]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/MediaDevices/getUserMedia

[19] Network, M. D.; individual contributors: *MediaTrackConstraints - Web APIs*. 2016, [cit. 2017-02-04]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/MediaTrackConstraints

[20] kurento.org: *Kurento Architecture*. [cit. 2017-02-17]. Dostupné z: http://doc-kurento.readthedocs.io/en/stable/mastering/kurento_architecture.html

[21] Red5: *Simple Red5 signaled WebRTC Demo (CLIENT AND SERVER)*. 2017, [cit. 2017-02-16]. Dostupné z: https://github.com/rajdeeprath/red5-webrtc-demo

[22] Adenot, P.; Wilson, C.: *Web Audio API*. 2015, [cit. 2017-02-04]. Dostupné z: https://www.w3.org/TR/webaudio/

[23] Mozilla Developer Network and individual contributors: *AudioContext - WebAPIs*. 2016, [cit. 2017-02-04]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/AudioContext

[24] Mozilla Developer Network and individual contributors: *AudioContext.createAnalyser()*. 2016, [cit. 2017-02-04]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/AudioContext/createAnalyser

[25] Mozilla Developer Network and individual contributors: *AudioContext.createMediaStreamSource() - WebAPIs*. 2016, [cit. 2017-02-04]. Dostupné z: https://developer.MozillaDeveloperNetworkandindividualcontributors.org/en-US/docs/Web/API/AudioContext/createMediaStreamSource

[26] Mozilla Developer Network and individual contributors Developer Network and individual contributors: *AnalyserNode - WebAPIs*. 2016, [cit. 2017-02-04]. Dostupné z: https://developer.MozillaDeveloperNetworkandindividualcontributors.org/en-US/docs/Web/API/AnalyserNode

[27] Dumaine, X.: *otalk/hark - Converts an audio stream to speech events in the browser*. 2017, [cit. 2017-02-04]. Dostupné z: https://github.com/otalk/hark

[28] Shires, G.; Wennborg, H.: *Web Speech API Specification*. 2012, [cit. 2017-02-04]. Dostupné z: https://dvcs.w3.org/hg/speech-api/raw-file/tip/speechapi.html

[29] kurento.org: *Java - Recorder - Kurento 6.6.0 documentation*. 2016, [cit. 2017-02-04]. Dostupné z: http://doc-kurento.readthedocs.io/en/stable/tutorials/java/tutorial-recorder.html

[30] Casas-Sanchez, M.; Barnett, J.; Leithead, T.: *MediaStream Recording*. 2016, [cit. 2017-02-04]. Dostupné z: https://www.w3.org/TR/mediastream-recording/

[31] Khan, M.: *RecordRTC: WebRTC audio/video recording*. 2017, [cit. 2017-02-04]. Dostupné z: http://recordrtc.org/

[32] Mozilla Developer Network and individual contributors and individual contributors: *PDF.js*. 2017, [cit. 2017-02-04]. Dostupné z: https://mozilla.github.io/pdf.js/

[33] KO GmbH: *WebODF*. 2017, [cit. 2017-02-04]. Dostupné z: http://webodf.org/

[34] Zerr, A.: *How to convert docx/odt to pdf/html with Java?* 2012, [cit. 2017-02-04]. Dostupné z: https://angelozerr.wordpress.com/2012/12/06/how-to-convert-docxodt-to-pdfhtml-with-java/

[35] Nasato, M.: *JODConverter*. 2017, [cit. 2017-02-04]. Dostupné z: http://www.artofsolving.com/opensource/jodconverter

[36] Plutext: *plutext/docx4j: JAXB-based Java library for Word docx, Powerpoint pptx, and Excel xlsx files*. 2017, [cit. 2017-02-04]. Dostupné z: https://github.com/plutext/docx4j

[37] Plutext: *Plutext PDF Converter - Online test*. 2017, [cit. 2017-02-04]. Dostupné z: http://converter-eval.plutext.com/client_java.html

[38] iText Group NV: *iText*. 2017, [cit. 2017-02-04]. Dostupné z: http://itextpdf.com/

[39] OpenSagres: *opensagres/xdocreport*. 2017, [cit. 2017-02-04]. Dostupné z: https://github.com/opensagres/xdocreport

[40] Google: *One framework. - Angular*. 2017, [cit. 2017-02-04]. Dostupné z: https://angular.io/

[41] ReactiveX: *ReactiveX*. 2017, [cit. 2017-02-04]. Dostupné z: http://reactivex.io/

[42] Facebook Inc.: *A JavaScript library for building user interfaces - React*. 2017, [cit. 2017-02-04]. Dostupné z: https://facebook.github.io/react/

[43] Wheeler, K.: *Getting To Know Flux, the React.js Architecture*. 2014, [cit. 2017-02-04]. Dostupné z: https://scotch.io/tutorials/getting-to-know-flux-the-react-js-architecture

[44] Microsoft: *TypeScript - JavaScript that scales.* 2016, [cit. 2017-02-04]. Dostupné z: https://www.typescriptlang.org/

[45] DefinitelyTyped: *DefinitelyTyped/DefinitelyTyped: The repository for high quality TypeScript type definitions.* 2017, [cit. 2017-02-04]. Dostupné z: https://github.com/DefinitelyTyped/DefinitelyTyped

[46] DefinitelyTyped: *Creating a definition file | DefinitelyTyped.* 2017, [cit. 2017-02-04]. Dostupné z: http://definitelytyped.org/guides/creating.html

[47] Mesnil, J.: *STOMP Over WebSocket.* 2012, [cit. 2017-02-04]. Dostupné z: http://jmesnil.net/stomp-websocket/doc/

[48] Finnigan, S.: *sjmf/ng2-stompjs-demo: Angular 2 demo using stomp.js in Typescript.* 2017, [cit. 2017-02-04]. Dostupné z: https://github.com/sjmf/ng2-stompjs-demo/

[49] Oracle: *JSR 356, Java API for WebSocket.* 2013, [cit. 2017-02-04]. Dostupné z: http://www.oracle.com/technetwork/articles/java/jsr356-1937161.html

[50] Google: *google/gson: A Java serialization/deserialization library that can convert Java Objects into JSON and back.* 2017, [cit. 2017-02-04]. Dostupné z: https://github.com/google/gson

[51] Pivotal Software: *26. WebSocket Support.* 2017, [cit. 2017-02-04]. Dostupné z: http://docs.spring.io/spring/docs/current/spring-framework-reference/html/websocket.html

[52] Pivotal Software: *Using WebSocket to build an interactive web application.* 2017, [cit. 2017-02-04]. Dostupné z: https://spring.io/guides/gs/messaging-stomp-websocket/

[53] Katkar, S.: *Spring 4 WebSockets with SockJS, STOMP and Spring Security 3.2.* 2014, [cit. 2017-02-04]. Dostupné z: http://sunitkatkar.blogspot.cz/2014/01/spring-4-websockets-with-sockjs-stomp.html

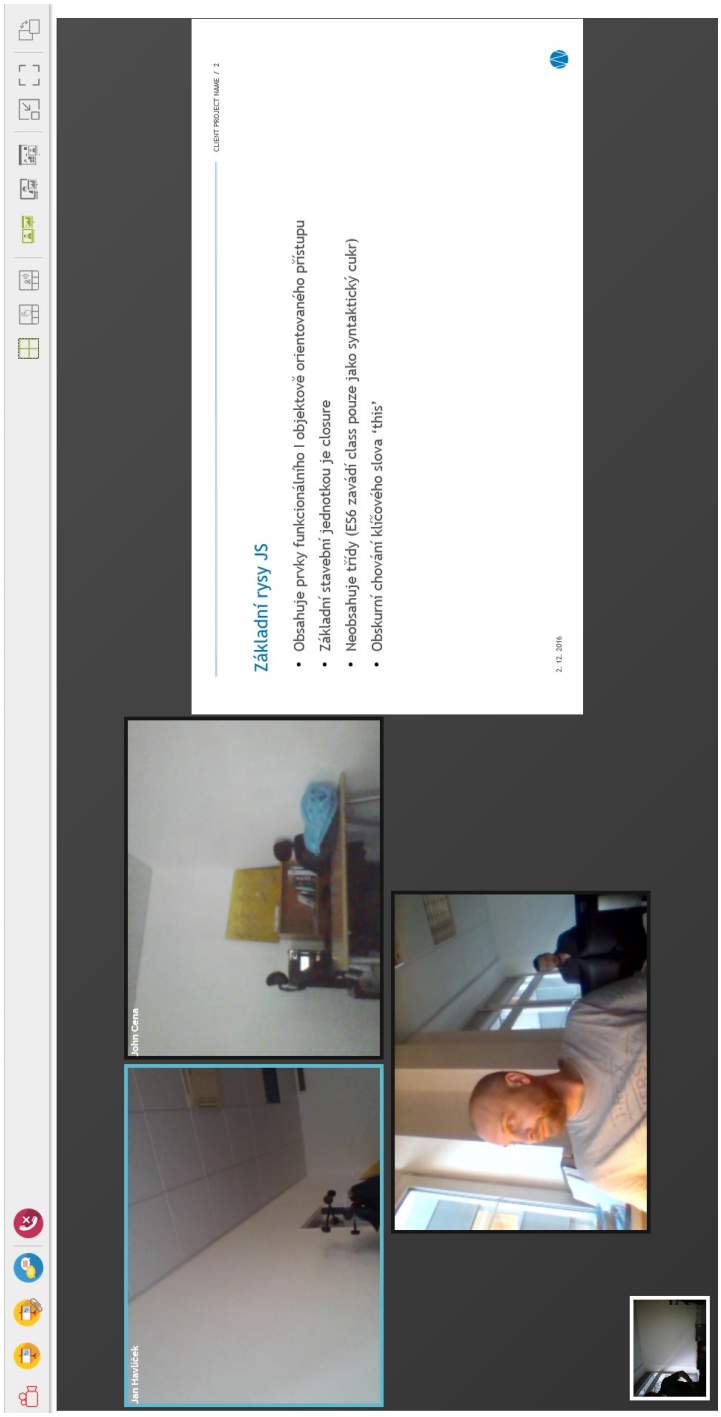
[54] Humphrey, P.: *Spring Framework 4.0 M2: WebSocket Messaging Architectures.* 2013, [cit. 2017-02-04]. Dostupné z: https://dzone.com/articles/spring-framework-40-m2

[55] Alvestrand, H.; Singh, V.: *Identifiers for WebRTC's Statistics API.* 2017, [cit. 2017-02-16]. Dostupné z: https://www.w3.org/TR/webrtc-stats/

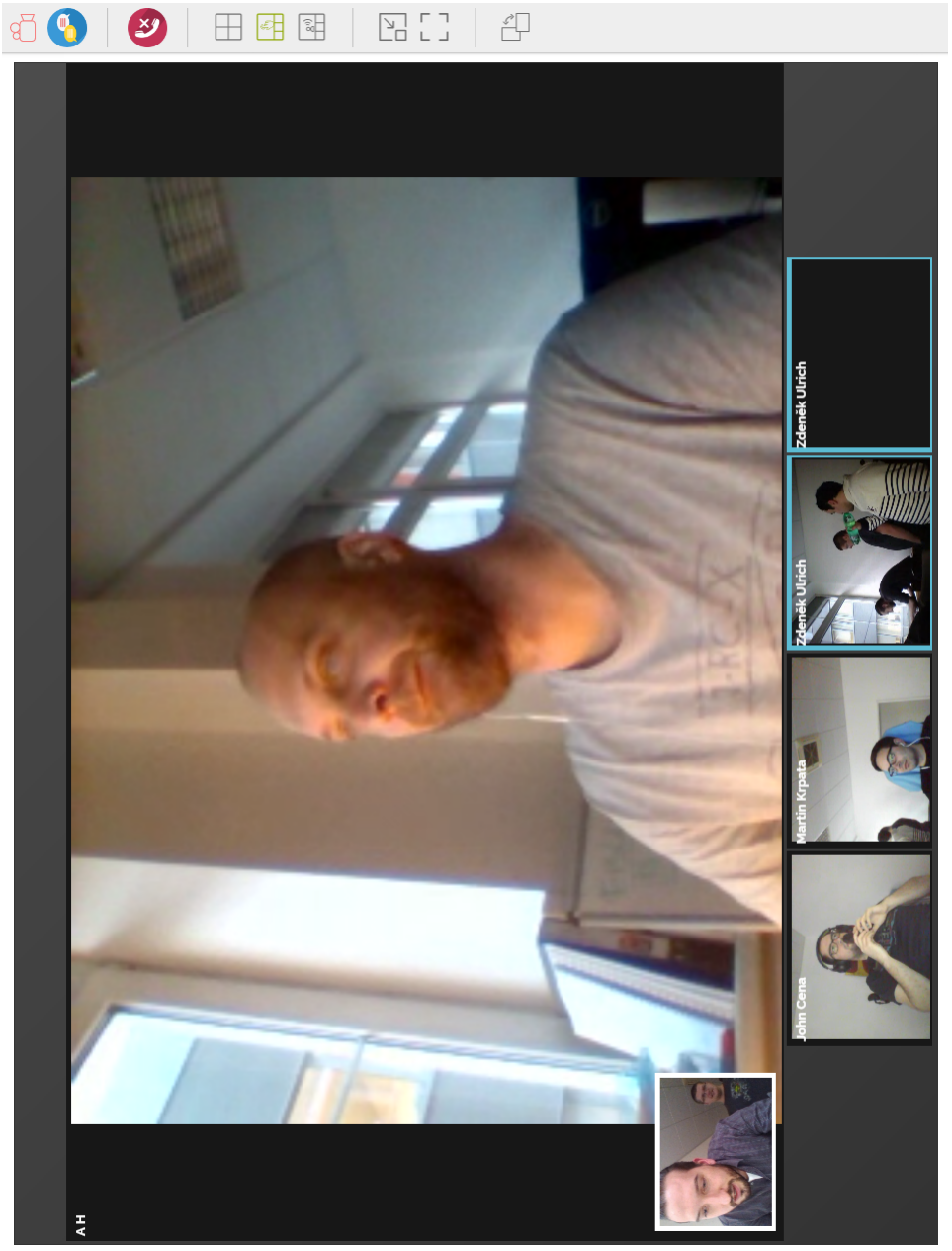
[56] Kurento: *WebRTC Statistics.* 2017, [cit. 2017-02-16]. Dostupné z: http://doc-kurento.readthedocs.io/en/stable/mastering/webrtc_statistics.html

Příloha A

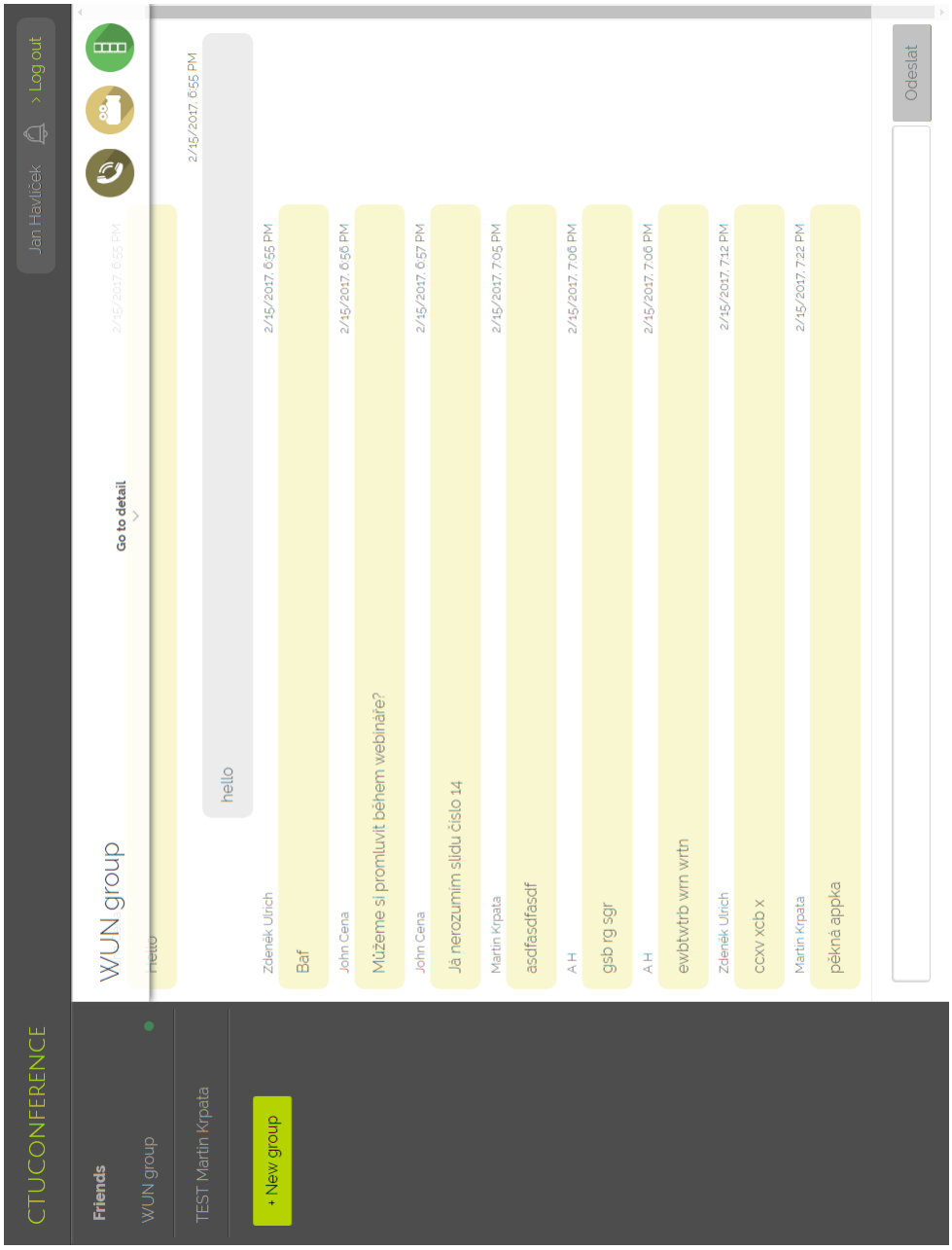
Snímky aplikace



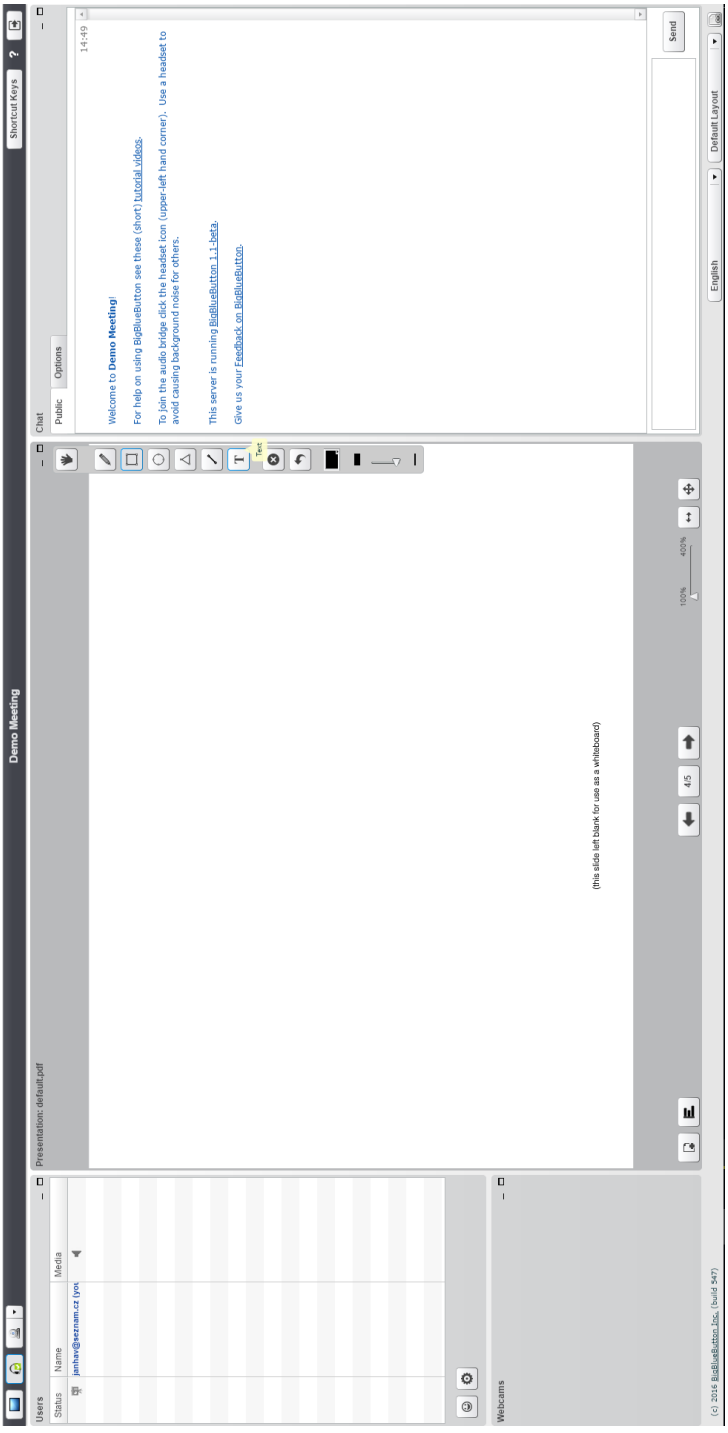
Obrázek A.1: Obrazovka se skupinovým hovorem a slajdy



Obrázek A.2: Obrazovka se skupinovým hovorem



Obrázek A.3: Obrazovka chatu s možnostmi pro vyvolání hovoru



Obrázek A.4: Ukázka layoutu webinářové aplikce BigBlueButton

Příloha B

Technologie pro vývoj front-endové části aplikace

Tato příloha poskytuje podrobnější informace o tom, pomocí jakých technologií jsou zpracovávány zdrojové soubory a obrázky při vývoji klientské části aplikace, a jak tyto technologie při vývoji použít.

Rozsáhlou nápovědu pro použití zde popsaných technologií a knihoven pro kompilaci klientských zdrojů, lze nalézt v kořenovém adresáři v souboru `README.md` a dále také v celém adresáři `/readme`.

B.1 Gulp jako nástroj pro spouštění úloh

V rámci implementace této práce byl pro kompilování zdrojových souborů a obrázků použit nástroj Gulp. Jde o nástroj, pomocí kterého lze nadefinovat základní operace, které jsou vykonávány při vývoji klientské části aplikace. Jde například o kompilaci jazyků jako je SASS do čistého CSS nebo TypeScript pro kompilaci do JavaScriptu. Dále umožňuje například vytváření spritů z obrázků a minifikaci obrázků. Další sekce v této kapitole popisují jednotlivé funkce, které byly implementovány.

Sestavování klientské části pomocí Gulpu lze implementovat jako součást sestavovacího procesu v rámci nástroje Maven. V tom případě se během procesu sestavování aplikace spustí i příkazy pro sestavení zdrojů pro klientskou část. To ale není vždy nezbytné a nabízí se tedy další možnost, kterou je sestavování klientské části nezávisle na sestavení aplikace v rámci Mavenu. Příkaz nástroje Gulp lze spouštět manuálně nebo lze v rámci Gulpu spustit automatického hlídače, který hlídá změny v souborech a na základě toho spouští vybrané příkazy automaticky. Příkazy Gulpu se tak budou pouštět pouze v případě, že k nějaké změně dojde a dále pouze v případě, že vývojář právě aktivně vyvíjí klientskou část aplikace.

Není důvod při každém sestavování aplikace volat skript pro kompilaci

klientských zdrojových souborů, proto byla v rámci této práce tedy zvolena pro vývoj varianta odděleného vývoje klientské části a Gulp tedy nebyl implementován do sestavovacího procesu v rámci nástroje Maven. Vývojář má tedy možnost kompilovat si klientské zdrojové soubory ručně nebo využít automatické hlídání změn pomocí nástroje Gulp. Projekt však také obsahuje inicializační skript, aby i bylo zprovoznění vývojového prostředí pro nového vývojáře co nejpohodlnější. Tím se zabývá následující sekce.

B.2 Inicializační skript pro podporu vývoje klientské části

V rámci vývoje klientské části je třeba před jeho započetím provést několik operací. Základní prerekvizitou je nainstalovaný Node JS. Obecně lze říci, že čím novější verze, tím lepší (odstraňuje spoustu optimalizačních nedostatků). Dále je třeba pomocí balíčkového nástroje npm pro Node JS nainstalovat globálně `gulp`, tak aby bylo možné jej volat z příkazové řádky.

Pro zjednodušení celého procesu byl v rámci tohoto projektu připravený skript, který spustí všechny další operace. Stačí zavolat `npm run build`. Tento skript je implementován jako npm skript a je definovaný v souboru `package.json` První operací je stažení všech potřebných závislostí pro Node JS. Ty jsou definovány v souboru `package.json` a obsahují závislosti, které jsou potřeba pro samotnou aplikace a dále také závislosti, které jsou potřeba pro vývoj a sestavování zdrojů klientské části aplikace. Dále skript obsahuje příkazy implementované pomocí nástroje Gulp. Prvním je minifikace obrázků a příprava spritů z jednotlivých obrázků (včetně vygenerovaných SASS souborů pro právě vytvořené sprity), a dále samotná kompilace typescriptových a SASS souborů.

Původně byl v rámci tohoto skriptu implementováno i stažení závislostí pro typescript pomocí příkazu `typings` a dále stažení závislostí pomocí balíčkovacího nástroje `bower`, nicméně v průběhu implementace byly tyto příkazy ze základního skriptu vyjmuty, neboť se tyto technologie během implementace této staly překonanými.

B.2.1 Základní adresářová struktura

Kořenový adresář obsahuje základní konfigurační soubory. `package.json` je určen pro definici závislostí, které jsou používány jak v cílové aplikaci, tak při samotném vývoji aplikace. Dále `tsconfig.json`, který specifikuje chování pro překladač TypeScriptu

Zdrojové soubory jsou umístěny v adresáři `/src/main/websrc`. Zde jsou všechny typescriptové, SASS soubory fonty a obrázky.

Všechny zkompilované a zpracované zdroje jsou ukládány do adresáře `/src/main/webapp/assets`.

B.3 Zpracování TypeScriptu

Jazyk TypeScript je jazyk vytvořený společností Microsoft a jedná se vedle CoffeeScriptu a Dartu o jeden z jazyků překládaný do JavaScriptu. Výhodou oproti těmto ostatním jazykům překládaným do JavaScriptu může být typovost a dále také syntaktická blízkost TypeScriptu a čistého JavaScriptu. TypeScript se snaží především obohatit jazyk o možnost typové kontroly, která umožňuje odhalení chyb již v době překladu, což také může usnadnit vývoj. TypeScript se snaží podporovat datové konstrukty definované v rámci stávající specifikace JavaScriptu (třídy, definice vlastností tříd, atd.), a dále tyto konstrukty doplňuje o takové možnosti, jako jsou výčtové typy, rozhraní, privátní atributy tříd a další.

Nová verze frameworku Angular2 je implementována již pomocí tohoto jazyka a taktéž veškerá dokumentace je primárně psána pro TypeScript. To je tedy hlavní důvod, proč byl v rámci této práce zvolen právě tento jazyk.

B.3.1 Kompilování TypeScriptu do JavaScriptu

V prostředí prohlížeče není možné importovat javascriptové soubory uvnitř jiných javascriptových souborů. Na straně serveru naproti tomu existuje několik způsobů pro psaní modulů a ES6 přináší také standardizovanou notaci pro implementaci modulů. Nástroj Browserify umožňuje v době překladu importovat různé moduly do sebe právě pomocí notace používané v prostředí Node JS. Vývojář má tak možnost strukturovat svůj kód do různých souborů. Dále existují pluginy pro Browserify, které umožňují přeložit ES6 notaci pro zápis modulů (kterou využívá i TypeScript), do notace pro ES6.

Babelify je plugin využívající nástroj Babel. Ten slouží pro překlad z ES6 do ES5 (který automaticky převede i novou ES6 syntax pro import modulů do notace pro Node JS). Plugin Tsify, který je implementován v tomto projektu, umožňuje kompilovat soubory TypeScriptu (opět automaticky překládá ES6 zápis importů do notace pro Node JS). Browserify dále sloučí všechny soubory do jednoho cílového podle toho, jak jsou navzájem importovány.

Výsledný soubor lze pak načítat v prohlížeči efektivněji než několik desítek menších souborů. Nevýhodou je delší první načtení, neboť výsledný soubor může být značně veliký. V současné době však neexistuje jiné řešení, které by umožňovalo dynamický import javascriptových modulů uvnitř jiných javascriptových souborů. To může změnit v blízké době masivní nasazení nového protokolu HTTP/2 a podpora prohlížečů načítání modulů dle standardu ES6.

Příkaz pro kompilaci Pro spuštění kompilace TypeScriptu je třeba spustit příkaz `gulp ts`. Ten zpracuje soubory tranzitivně importované do souboru `/src/main/websrc/scripts/main.ts` a vytvoří cílový soubor `/src/main/webapp/js/all.min.js`

B.3.2 Typové definice

TypeScript však má také některé nevýhody nebo úskalí. Hlavním úskalím je fakt, že většina existujících knihoven není implementována s ohledem na typy a je tedy třeba definici typů doimplementovat, aby bylo možné tyto knihovny v projektu použít. Naštěstí existuje open-source komunita **DefinitelyTyped**, která sdružuje kvalitní typové definice pro většinu nejpoužívanějších JavaScriptových knihoven a my je tedy můžeme pouze naimportovat.

B.3.2.1 Import existujících definičních souborů

V dřívější verzi TypeScriptu (<2.0) bylo nutné pro stažení typových definicí použít další modul pro Node JS `typings`. Všechny importované definice bylo potřeba definovat v souboru `typings.json`. Ten mohl vypadat následovně:

```
{
  "globalDependencies": {
    "core-js": "registry:dt/core-js#0.0.0+20160725163759",
    "jasmine": "registry:dt/jasmine#2.2.0+20160621224255",
    "node": "registry:dt/node#6.0.0+20160831021119"
  }
}
```

Následně bylo potřeba zavolat příkaz `typings install`, pomocí kterého byly tyto definice staženy.

Od Typescriptu verze 2.0 je již tento způsob označený jako zastaralý a namísto toho stačí přidat další závislost do souboru `package.json` a stáhnout ji stejně jako všechny ostatní závislosti. Soubor `package.json` může obsahovat např. toto:

```
"devDependencies": {
   "@types/core-js": "^0.9.35",
   "@types/jasmine": "^2.5.38",
   "@types/node": "0.0.2",
   ...
}
```

B.3.3 Vlastní implementace definičních souborů

Může se však také stát (a v rámci této práce se také stalo), že pro knihovnu, kterou bychom chtěli použít, neexistuje. Jedinou možností, pokud se nechceme použití dané knihovny vzdát, je vlastní implementace definičního souboru.

Konfigurační soubor `tsconfig.json` mimo jiné definuje adresář, ve kterém bude vyhledávat vlastní definiční soubory. Aktuálně jsou ukládány do adresáře `types`. V tomto adresáři názvy podadresářů odpovídají nazvům adresářů jednotlivých balíčků, které jsou staženy pomocí nástroje npm v adresáři

`/node_modules`. V adresáři `/types` pro každý balíček, pro který chceme vytvořit definiční soubor, vytvoříme soubor index.d.ts a v něm popíšeme definici typů pro daný balíček. Návod, jak vytvářet definiční typy a zároveň jak je vytvářet tak, aby mohly být přijaty jako součást projektu DefinitelyTyped, jsou popsány právě na stránkách tohoto projektu.

V příloze jsou v adresáři `TypeScriptdefinitionfiles` uloženy všechny použité vlastní definiční soubory. Jedná se o knihovny **hark** pro identifikaci zvuku z audiovizuálního streamu, **kurento-utils**, která je základním pilířem aplikace, a dále stompjs pro podporu podprotokolu STOMP při použití protokolu WebSocket.

B.4 Kompilování SASS do CSS

Další základní funkcí je kompilování SASS (SCSS dialektu) do CSS. Součástí je automatické prefixování některých vlastností, pro které existují vendor prefixy, pomocí modulu `autoprefixer`.

Po zkompilování do CSS je opět provedeno sloučení souborů do jednoho cílového souboru a jeho případná minifikace.

Příkaz pro kompilaci Pro spuštění kompilace SASS je třeba spustit příkaz `gulp sass`. Ten zpracuje soubory tranzitivně importované do souboru `/src/main/websrc/stylesheets/main.scss` a vytvoří cílový soubor `/src/main/webapp/css/all.min.css`.

Tento skript již zpracovává i SASS soubory, které jsou vygenerovány při vytváření obrázkových spritů. Proto je nejprve před tímto příkazem potřeba zavolat ještě příkaz pro zkompilování těchto spritů, pokud došlo ke změně obrázků.

B.5 Tvorba obrázkových spritů

Aplikace umožňuje použití běžných obrázkových spritů, ale dále také umožňuje použití spritů složených z více SVG souborů. Tyto SVG sprity lze použít dvěma různými způsoby v závislosti na tom, jaké výhody SVG chceme využít.

Pro vygenerování běžných obrázkových spritů je třeba nahrát soubory do adresáře `/src/main/websrc/images/sprite` a po zkompilování vznikne jeden sloučený obrázkový soubor v adresáři `/src/main/webapp/images` a SASS soubor s definicemi obrázků, který je naimportován do hlavního SASS souboru. V SASS lze poté použít tyto definice následovně:

```
.my-icon {
  @include sprite($my-image);
}
```

Obdobně lze vytvořit SVG sprite, který lze použít na pozadí stejně jako běžný obrázkový sprite. Zdrojové SVG soubory je třeba umístit do adresáře `/src/main/websrc/images/svg-sprite` a výsledný soubor `svg-sprite.png` se opět uloží do adresáře `/src/main/webapp/images` a SASS soubor s definicemi obrázků je opět naimportován do hlavního SASS souboru.

```
.my-svg-icon {
  @include svg-sprite($svg-my-image);
}
```

Příkaz pro kompilaci Pro vytvoření spritu je třeba spustit příkaz `gulp sprite` nebo pro vytvoření spritu složeného ze SVG obrázků lze spustit příkaz `gulp svg-sprite`.

B.6 Minifikace obrázků

Další optimalizaci lze provádět i na obrázcích, které nejsou použity ve spritech. Pro ty je zde možnost minifikace. Minifikaci obrzákových zdrojů lze provést pomocí příkazu `gulp imagemin`.

Příloha C

Seznam použitých zkratek

KMS Kurento Media Server

SDP Session Description Protocol

ICE Interactive Connectivity Establishment

STOMP Simple (or Streaming) Text Oriented Messaging Protocol

RTP Real-time Protocol

SSE Server sent event

HLS HTTP Live Streaming

HDS HTTP Dynamic Streaming

HSS HTTP Smooth Streaming

DASH Dynamic Adaptive Streaming over HTTP

MPD Manifest Presentation Description

RTC Real-time Communication

SCTP Stream Control Transmission Protocol

NAT Network address translation

STUN Session Traversal Utilities for NAT

TURN Traversal Using Relays around NAT

RxJS The Reactive Extensions for JavaScript

ES6 ECMA Script 6

XLS-FO XSL formatting objects

Příloha **D**

Obsah přiloženého CD

- `Attachments` ..... Ukázky některých implementací
  - `STOMP` ..... Ukázková implementace komunikace s protokolem STOMP
  - `STOMP-less websocket solution` Implementace použité komunikační části bez použití STOMP protokolu
  - `TypeScript definition files` ..... Vlastní definiční soubory
    - `hark` ..... Definice pro knihovnu detekce hlasu
    - `kurento-utils` ..... Definice pro JS část frameworku Kurento
    - `stompjs` ..... Definice pro klientskou část STOMP.js knihovny
- `Conference` ..... Celý projekt - klientská i serverová část
  - `db` ..... Importní skript pro inicializaci databáze
  - `README.md` ..... Návod pro vývoj klientské části aplikace
- `Documents` ..... Text práce
  - `text-sources` ..... zdrojová forma práce ve formátu LaTeX
  - `DP_Havlicek_Jan_2017.pdf` ..... text práce ve formátu PDF
- `Conference.war` ..... Webový modul aplikace