FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Webový konferenční systém
Student:	Marek Bajtalon
Vedoucí:	Ing. Viktor Černý
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Webové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2021/2022

Pokyny pro vypracování

Pokyny k vypracování:
- analyzujte dostupné konferenční systémy a vyhodnoťte jejich výhody a nevýhody
- navrhněte řešení, které zohlední výstupy analýzy a nabídne otevřené zdrojové kódy, které lze snadno rozšiřovat o novou funkcionalitu
- implementujte prototyp, který bude obsahovat alespoň základní funkcionalitu jako konferenční hovor, sdílení obrazovky a chat
- poskytněte kompletní dokumentaci pro další vývojáře a vše společně s projektem umístěte veřejně na GITHub, aby okolo projektu mohla vzniknout komunita
- řešení otestujte na základní funkčnost a při zatížení, které odpovídá jedné výukové skupině (asi 20 uživatelů)

Elektronicky schválil/a Ing. Jaroslav Kuchař, Ph.D. dne 12. února 2021 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Webový konferenční systém

Marek Bajtalon

Katedra softwarového inženýrství
Vedoucí práce: Ing. Viktor Černý

13. května 2021

Poděkování

V první řadě bych chtěl poděkovat svému vedoucímu Ing. Viktoru Černému za rady a pomoc během psaní této bakalářské práce a také své rodině za čas a shovívavost během mého studia. Dále bych chtěl poděkovat přátelům a dobrovolným studentům, kteří mi pomohli s otestováním této práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principu při 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 předpisu. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisu, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programu, 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 zpusobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelum). 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 zpusobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 13. května 2021 . . . . . .

Odkaz na tuto práci

Bajtalon, Marek. *Webový konferenční systém*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2021.

Abstrakt

Tato práce se zabývá analýzou, návrhem, implementací a testováním prototypu webového konferenčního systému. Nejprve se podíváme na tři aktuálně nejpopulárnější programy pro vzdálenou komunikaci, dále se zaměříme na technologie, které využijeme při implementaci našeho prototypu. Ten bude umožňovat hlasovou a video komunikaci mezi uživateli a také bude podporovat chat a sdílení obrazovky. Dále si navrhneme wireframy – ty popisují, jak bude aplikace vypadat a dále se již zaměříme na implementaci klientské části (frontendu) a serverové části (backendu). V práci je použit převážně jazyk JavaScript, konkrétně nadstavba TypeScript, aby se práce lépe udržovala a psala.

Klíčová slova WebRTC, konferenční systém, webová aplikace, WebSockets, Angular, Socket.IO, TypeScript

Abstract

The bachelor thesis addresses an analysis, design, implementation and testing of a web conference system prototype. At first, we are going to compare the three most popular communication applications available at the moment. Then we are going to address technologies that are going to use to implement our prototype. The prototype is going support voice and video communication between users and an environment for chat and screen share. Then we are going to design wireframes – they are used for visualization of the user interface of the prototype. After that we are going to focus on the implementation of the client side (frontend) and server side (backend). In the end we are going to test the prototype on group of 20 users. The thesis is written in JavaScript with the TypeScript addition to make sure the thesis is easy to maintain and write.

Keywords WebRTC, conference system, web application, WebSockets, Angular, Socket.IO, TypeScript

Obsah

Úvod **1**

1 Cíl práce **3**

2 Analýza **5**
2.1 Analýza současně dostupných řešení 5
2.1.1 Zoom 5
2.1.2 Teams 6
2.1.3 Google Meet 6
2.2 Analýza 6
2.3 Funkční požadavky 6
2.4 Nefunkční požadavky 7
2.5 Analýza použitých technologií 8
2.5.1 JavaScript 8
2.5.2 TypeScript 8
2.5.3 Angular 8
2.5.4 Node.js 8
2.5.5 npm 8
2.5.6 Socket.io 9
2.5.7 Express.js 9
2.5.8 FontAwesome 9
2.5.9 Bootstrap 9
2.5.10 MDBootstrap 9
2.5.11 WebRTC 9
2.5.12 SimplePeer 14
2.5.13 Let's Encrypt 14

3 Návrh **15**
3.1 Wireframe 15

3.2 Komunikace 19
3.2.1 Backend 20
3.2.2 Frontend 21
3.2.3 Události knihovny SimplePeer 21
3.2.4 Vlastní události 21

4 Implementace **23**
4.1 Implementace Frontendu 23
4.1.1 Získání audio/video streamu 23
4.1.2 Získání streamu z obrazovky 24
4.1.3 Přehrávání získaného audia/videa 25
4.1.4 Chat 26
4.1.5 Renegotiation 28
4.1.6 Směrování 28
4.2 Implementace Backendu 29
4.2.1 STUN a TURN server 29
4.2.2 HTTPS 29
4.2.2.1 Certbot 30
4.2.2.2 Express.js 30
4.2.3 Přesměrování z HTTP na HTTPS 30

5 Testování **33**

Závěr **35**

Literatura **37**

A Seznam použitých zkratek **41**

B Výsledná podoba aplikace **43**

C Obsah přiloženého CD **51**

Seznam obrázků

2.1 Znázornění WebRTC mesh topologie 10
2.2 Znázornění WebRTC MCU topologie 11
2.3 Znázornění WebRTC SFU topologie 12
2.4 Znázornění principu STUN serveru 14

3.1 Obrazovka pro napojení do místnosti 16
3.2 Obrazovka pro vytvoření nové místnosti 16
3.3 Obrazovka místnosti s otevřeným chatem 17
3.4 Obrazovka místnosti s otevřeným seznamem uživatelů 17
3.5 Wireframe pro telefony – Připojení/tvorba místnosti 18
3.6 Wireframe pro telefony – Hovor 18
3.7 Sekvenční diagram pro navázání spojení s novým uživatelem 19

4.1 Ukázka kódu k získání videa od uživatele 24
4.2 Porovnání dialogů pro přístup k mikrofonu 24
4.3 Dialog pro výběr sdílení obrazovky v Google Chrome 25
4.4 Dialog pro výběr sdílení obrazovky v Mozilla Firefox 25
4.5 Ukázka kódu – získání streamu 26
4.6 Ukázka kódu – přehrávání audia 26
4.7 Ukázka kódu – Chat 27
4.8 Ukázka kódu – Šablona chatu 27
4.9 Ukázka kódu – Odesílání zprávy v chatu 28
4.10 Ukázka kódu – směrování 29
4.11 Ukázka rozdílu v adresním řádku u Google Chrome 30
4.12 Ukázka kódu – HTTPS 31
4.13 Ukázka kódu – Přesměrování 31

B.1 Obrazovka pro napojení a tvorbu místnosti 43
B.2 Obrazovka s probíhajícím hovorem 44
B.3 Obrazovky s modálními okny 45

B.4 Vysouvací panel 46
B.5 Seznam místností 47
B.6 Vytvoření nové místnosti 47
B.7 Hovor 48
B.8 Modální okno pro sdílení obrazovky 48
B.9 Postranní panel s chatem 49
B.10 Postranní panel se seznamem uživatelů 49

Seznam tabulek

2.1 Srovnání třech nejpopulárnějších konferenčních programů 5
2.2 Výhody a nevýhody Mesh topologie 10
2.3 Výhody a nevýhody MCU topologie 11
2.4 Výhody a nevýhody SFU topologie 12
2.5 Srovnání využití prostředků dle topologie 13

3.1 Události 20
3.2 Události vyvolané knihovnou SimplePeer 21
3.3 Vlastní události mezi uživateli 22

Úvod

S příchodem koronaviru se život každého z nás zásadně změnil. Studium a práce se přesunula převážně na internet. To znamená, že programy pro vzdálenou komunikaci se staly nesmírně populární a nechávají si za ně jejich společnosti slušně zaplatit.

Pokud bychom chtěli nějakou funkcionalitu pozměnit nebo přidat, tak zde úplně nepochodíme. Zdrojové kódy programů jsou v rukách velkých firem a nic takového nepřichází v úvahu. Nevíme vlastně také, kudy data tečou a co se s nimi děje.

A přesně to byl jeden z impulzů, proč jsem s tímto nápadem přišel za mým vedoucím. Chtěl jsem vytvořit webový konferenční systém, který si budou moci uživatelé spustit u sebe na serveru. A pokud budou mít zájem, tak i doplnit nějakou funkcionalitu, která by jim chyběla nebo nevyhovovala. Jak se mi to povedlo se dozvíte v této bakalářské práci.

Kapitola 1

Cíl práce

Cílem této práce je zanalyzovat dostupná řešení, podívat se na vhodné technologie a provést návrh a implementaci prototypu vlastního konferenčního systému. Jehož instanci si bude možné spustit na vlastním stroji. Samotný klient nebude vyžadovat stažení žádného dalšího softwaru a poběží pouze v prohlížeči. Tím zajistíme co největší kompatibilitu, protože drtivá většina dnes prodávaných zařízení má internetový prohlížeč.

Znamená to tedy vytvořit klientskou část (frontend) i serverovou část (backend) a spojit je dohromady. K tomu využít moderní technologie, které prohlížeče nabízí a ideálně to vše ještě napsat za pomocí jednoho programovací jazyka, aby se kód lépe psal a udržoval.

Výsledný zdrojový bude zveřejněn na platformě GitHub, kde budou moci ostatní uživatelé přispívat do projektu.

Kapitola 2

Analýza

Následující kapitola se zabývá analýzou dostupných řešení. Výsledkem této analýzy by měl nástřel technologií potřebných k implementaci prototypu. Dále se jimi budeme více zaobírat v implementační části.

2.1 Analýza současně dostupných řešení

Konferenčních programů je na trhu k dispozici opravdu hodně. V této části se zaměříme na tři nejpopulárnější z nich. Srovnávacím kritériem bude počet aktivních uživatelů za den.

Tabulka 2.1: Srovnání třech nejpopulárnějších konferenčních programů

	Zoom		Teams		Google Meet	
vlastník	Eric Yuan		Microsoft		Google	
počet uživatelů denně	300 milionů [1]		115 milionů [2]		100 milionů [3]	
typ licence	$14.99/rok	zdarma	$5/měsíc	zdarma	$8/měsíc	zdarma
max. počet účastníků v jedné schůzce	100 lidí	100 lidí	300 lidí	100 lidí	150 lidí	100 lidí
max. délka schůzky	30 hodin	40 minut	24 hodin	1 hodina	24 hodin	1 hodina
nutnost stažení aplikace	NE		NE		NE	

2.1.1 Zoom

Aktuálně nejpopulárnější program pro video konference. Spuštěn byl již v září roku 2012, kde podporoval maximálně 15 uživatelů. Aplikace je nyní dostupná pro každou platformu a nabízí i webovou verzi, která je ale ochuzena o některé funkce.

Díky pandemii a nárustu počtu uživatelů se stala pátou nejstahovanější aplikací v roce 2020. Předběhla tak aplikace jako Messenger nebo Snapchat [4]. Hlavní záminkou pro vývoj aplikace Ericem Yuanem tvořila jeho přítelkyně,

která bydlela v jiném státe. Chtěli se vídat častěji, a tak dostal tento nápad [5].

Zoom měl ale dříve problémy se zabezpečením. Docházelo například k úniku emailových adres [6] nebo třeba k falešné informaci, že spojení je end-to-end šifrované [7]. Dalším pomyslným hřebíčkem do rakve mohl být také případ z minulého roku, kdy data některých uživatelů tekla přes čínské servery [8]. Chyby by už měly být opraveny, ale i tak je k Zoomu spousta lidí skeptická.

2.1.2 Teams

Velmi oblíbený nástroj pro distanční výuku a práci. Je součástí předplatného Office 365, takže má skvělou integraci kancelářských nástrojů. Microsoft chtěl původně za 8 miliard koupit Slack [9], ale nakonec se rozhodl rozšířit funkcionalitu Skype for Business. Později se jej Microsoft rozhodl nahradit za Teams.

Desktopová aplikace je postavena na Electron frameworku. Jedná se o nástroj pro tvorbu multiplatformních aplikací, kde je většina zdrojového kódu sdílena mezi platformami. Možná i zde se Microsoft trochu inspiroval, protože i Slack je vytvořen v Electronu.

Hlavní výhodou, převážně pro školy, může také být možnost psát testy a odevzdávat úkoly.

2.1.3 Google Meet

Jedná se o nástupce za Google Hangouts, který byl součástí již nyní zrušené sociální sítě Google+ a za aplikaci Google Chat. Ten sloužil jako konkurent už zmíněného Slacku.

První verzi vydal tiše Google v únoru 2017 na platformu iOS [10]. Byla pouze na pozvánky a po chvíli byla zase z AppStoru stažena. Poté byla o měsíc spuštěna už oficiálně.

Díky pandemii nabral Google Meet také na popularitě a stal se devátou nejstahovanější aplikací pro mobilní zařízení za rok 2020 [4]. Pro desktop je situace trochu jiná, žádná aplikace neexistuje a jediné co je potřeba – je prohlížeč.

2.2 Analýza

Součástí softwarové vývoje je i analýza požadavků. Ty dělíme na funkční (co má být provedeno) a nefunkční (jakým způsobem).

2.3 Funkční požadavky

- **F01 – Vytvoření místnosti** – Umožňuje uživateli vytvořit si vlastní místnost se zvoleným jménem. Dále má možnost si zvolit, zda bude místnost viditelná v seznamu místností či nikoliv.

- **F02 – Zobrazení existujících místností** – Uživatel má možnost si zobrazit již založené místnosti, které jsou veřejné a to včetně počtu jejich uživatelů.
- **F03 – Připojení do místnosti** – Uživatel se může do místnosti připojit buďto přes seznam místností anebo pomocí URL odkazu vygenerovaném po založení místnosti.
- **F04 – Přenos zvuku** – Aplikace umožňuje uživatelům připojených do stejné místnosti hlasovou komunikaci.
- **F05 – Přenos videa** – Aplikace umožňuje sdílet obraz z připojené videokamery mezi ostatní účastníky.
- **F06 – Sdílení obrazovky** – Aplikace umožňuje sdílet obrazovku (popř. specifické okno aplikace) mezi ostatní účastníky.
- **F07 – Chat** – V aplikaci bude dostupné rozhraní pro přenos zpráv mezi uživatele dané místnosti.
- **F08 – Vypnutí mikrofonu** – Uživatel si bude moci vypnout mikrofon tak, aby ho ostatní uživatelé v místnosti neslyšeli.
- **F09 – Zobrazení ostatních účastníků** – Pokud je uživatel v místnosti, má možnost si zobrazit seznam připojených uživatelů.
- **F10 – Individuální nastavení hlasitosti** – Uživatel si může pro každého uživatele nastavit zvlášť hlasitost.
- **F11 – Volba jména** – Uživatel si může zvolit jméno, pod kterým ho ostatní uvidí.
- **F12 – Sdílení místnosti** – Aplikace nabídne možnost vytvořit odkaz, který bude sloužit pro přímé napojení do místnosti.

2.4 Nefunkční požadavky

- **N01 – Responzivita** – Klientská část aplikace bude dobře škálovat uživatelské rozhraní na různých druzích zařízení a rozlišení.
- **N02 – Bezpečnost** – Aplikace bude přístupná pouze přes protokol HTTPS.
- **N03 – Frontend** – Klientská část aplikace bude napsána v Angularu.
- **N04 – Backend** – Backend bude napsán v Node.js.
- **N05 – Volná dostupnost** – Aplikace bude volně dostupná a bude možné si spustit její instanci na vlastním stroji.

2.5 Analýza použitých technologií

2.5.1 JavaScript

Jedná se o skriptovací jazyk, který byl vytvořen za účelem oživit statické webové stránky. Původně sloužil jako čistě klientský jazyk na straně prohlížeče, ale v poslední době se hlavně díky Node.js rozšířil i na servery [11]. Slovo Java v jeho názvu nemá s programovacím jazykem Java nic společného, autoři ho zvolili pouze z marketingových důvodů. Jelikož budeme dělat webovou aplikaci, je JavaScript celkem jasná volba.

2.5.2 TypeScript

Projekt společnosti Microsoft, jehož cílem bylo vytvořit nadstavbu nad klasický JavaScript, která přidává statické typování. Je tak vhodný pro aplikace větších rozměrů. Kód napsaný v TypeScriptu je před spuštěním kompilován zpět na čistý JavaScript. To nám může pomoci objevit chyby ještě před samotným spuštěním i přesto, že je JavaScript interpretovaný jazyk. Toto je jeden z hlavních důvodů, proč jsem si TypeScript vybral [12].

Je dobré ještě poznamenat, že není nutno všude používat statické typování a jeho jiné funkce, i čistý JavaScript je validním TypeScript kódem.

2.5.3 Angular

Jeden z velmi oblíbených frameworků pro tvorbu jednostránkových webových aplikací. Před ostatními frontendovými frameworky jsem ho upřednostnil kvůli tomu, že je napsán na TypeScriptu a chci v celé práci používat jeden jazyk.

Jeho hlavním stavebním kamenem jsou komponenty. Ke každé komponentě náleží třída a k ní pak HTML šablona a styly. To nám umožňuje oddělit aplikační logiku od uživatelského rozhraní. Dalším důležitým stavebním kamenem jsou služby, starají především o distribuci dat mezi komponentami. Služby si můžeme představit jako singletony – tedy v běhu aplikace existuje pouze jedna jejich instance [13].

2.5.4 Node.js

Node.js je platforma založená na JavaScript enginu z prohlížeče Google Chrome. Její hlavní záměr je psaní rychlých a škálovatelných webových aplikací. Normální JavaScript vyžaduje pro běh webový prohlížeč, ale díky Node.js ho můžeme spustit zcela samostatně [11].

2.5.5 npm

Jako každý správný programovací jazyk má i JavaScript svůj balíčkovací systém. Hlavním důvodem proč takový balíčkovací systém používat je jednoduchá instalace a správa rozšiřujících knihoven. V adresáři kde npm spustíme

je vytvořen package.json soubor. Ten si pamatuje, jaké verze knihoven máme nainstalovány. Stačí tak verzovat pouze ten a ne samotné knihovny.

2.5.6 Socket.io

Javascript knihovna pro komunikace klienta (prohlížeče) a serveru v reálném čase. V jejím jádru využívá knihovna standard WebSockets, který v dnešní době už podporuje většina prohlížečů [14]. Vybral jsem si ji hlavně proto, že je knihovna jednoduchá na použití a umí v případě nekompatibility prohlížeče s WebSockety přepnout do normálních HTTP požadavků/odpovědí. My se tak jako vývojáři už nemusíme o nic starat [15].

2.5.7 Express.js

Jednoduchý backendový framework pro Node.js. V aplikaci ho využijeme jako webový server pro zobrazení naší Angular aplikace [16]. Ačkoliv Express.js umí i směrování, ponecháme ho na Angularu.

2.5.8 FontAwesome

I když to z názvu není úplně zřejmé, jedná se o jednoduchou knihovnu obsahující několik tisíc ikon. Ty mají široké využití a v naší aplikaci jí využijeme především pro snadnější orientaci v uživatelském rozhraní [17].

2.5.9 Bootstrap

Knihovna zaměřující se především na tvorbu uživatelských rozhraní. Knihovna má svůj vlastní grid systém, který se zaměřuje na responzivitu. To znamená že se rozložení stránky přizpůsobuje zařízení, na kterém je zobrazeno.

Dalším klíčovým prvkem Bootstrapu je redesign základních HTML prvků a přidání vlastních komponent jako jsou modální okna, formuláře, navigace atd. [18].

2.5.10 MDBootstrap

Material Design for Bootstrap rozšiřuje knihovnu Bootstrap o Material Design. Material Design je systém pro návrh uživatelských rozhraní od Googlu. Velkou inspiraci bere z reálného světa – papíru a inkoustu a snaží se o sjednocení uživatelských rozhraní pro web, Android a iOS [19].

MDBoostrap jsem si vybral hlavně proto, že podporuje integraci komponent s Angularem, což původní Bootstrap neumí [20].

2.5.11 WebRTC

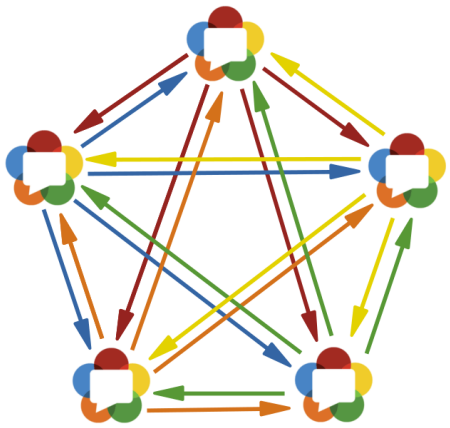
Hlavní základ naší aplikace tvoří právě WebRTC. Jedná se o otevřené API, které nabízí webovým prohlížečům a mobilním aplikacím možnost komunikace

a přenosu audio/videa v reálném čase. Aktuálně ho podporuje přes 93 % prohlížečů [21].

Hlavním problémem je to, že WebRTC byl designován jako peer-to-peer technologie. To se nám může v některých situací hodit, ale pokud chceme spojit větší množství uživatelů najednou, můžeme narazit na překážky. V následující části si popíšeme tři různé WebRTC topologie a jejich výhody/nevýhody.

Mesh

Nejjednodušší topologie na implementaci, každý klient přijímá data od každého klienta a posílá je každému klientovi [22].



Obrázek 2.1: Znázornění WebRTC mesh topologie

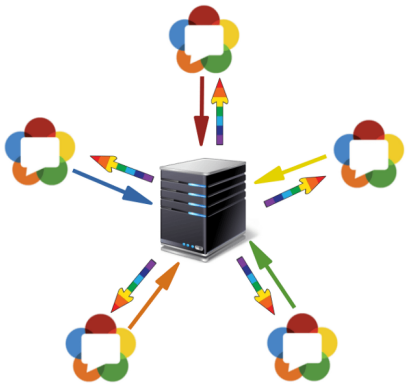
Tabulka 2.2: Výhody a nevýhody Mesh topologie

Mesh topologie	
Výhody	**Nevýhody**
+ není potřeba žádný server pro přenos	– s narůstajícím počtem uživatelů se exponenciálně zvyšuje datový tok
+ snadnější na implementaci	
+ end-to-end šifrování	

MCU

Klienti už zde nekomunikují jednotlivě mezi sebou, ale využívají k tomu server. Ten od každého z nich přijímá audio/video, sníží jeho datový tok (zkomprimuje) a pošle ho ostatním klientům.

Další možností jak ušetřit datový tok, je to, že posíláme pouze jeden (hlavní) video zdroj v plné kvalitě a ostatní zdroje v nízké kvalitě. To už ale záleží na implementaci dané služby a také na rozložení uživatelů v uživatelském rozhraní [22].



Obrázek 2.2: Znázornění WebRTC MCU topologie

Tabulka 2.3: Výhody a nevýhody MCU topologie

MCU topologie	
Výhody	**Nevýhody**
+ nízké nároky na upload/download klienta	– vyžaduje vysoký výkon na straně serveru pro encoding videí
+ jednoduchý klient	

SFU

Klienti zde opět nekomunikují mezi sebou, ale od MCU se liší tím, že s audiem/videem se už na straně serveru nic nedělá a posílá se v takové podobě, jaké dorazilo [22].

Obrázek 2.3: Znázornění WebRTC SFU topologie

Tabulka 2.4: Výhody a nevýhody SFU topologie

SFU topologie	
Výhody	**Nevýhody**
+ narozdíl od MCU není potřeba velmi výkonný server	– vyžaduje vysoký bandwidth na straně serveru
+ oproti Meshi není tak náročný na upload klienta	– oproti MCU vyžaduje vyšší download klienta

Modelová situace a srovnání

Nyní ukážeme srovnání všech topologií v místnosti, kde je 20 lidí. Pro ukázku jsem vybral ten nejhorší scénář – tedy každý sdílí video z webkamery a chce vidět všechny ostatní. Z mého testování odpovídá upload FHD videa zhruba na 2Mbit/s, a proto ho také použijeme ve výpočtech.

Tabulka 2.5: Srovnání využití prostředků dle topologie

	Mesh	MCU	SFU
upload klienta	20 * 2 = 40Mb/s	2Mb/s	2Mb/s
download klienta	20 * 2 = 40Mb/s	19 * 0.3[1] + 1*2 = 7,7Mb/s	20 * 2 = 40Mb/s
upload serveru	0	20 * 2 + 20 * 19 * 0,3[1] = 114Mb/s	20 * 19 * 2 = 760Mb/s
download serveru	0	20 * 2 = 40Mb/s	20 * 2 = 40Mb/s
vytížení serveru	nízké	vysoké	střední

Jak je vidět, každá topologie má svoje výhody a nevýhody. Už i při 20 uživatelích jsou čísla u SFU uploadu dost vysoká, je tedy jasné, že tato bakalářská práce nemá šanci konkurovat ostatním aplikacím s rozsáhlými datovými centry. Proto jsem se rozhodl zabývat se v této bakalářské práci topologií Mesh. Ta není tak náročná na serverové prostředky, ale spíše na uživatelovo připojení. A hlavní výhodou je end-to-end šifrování.

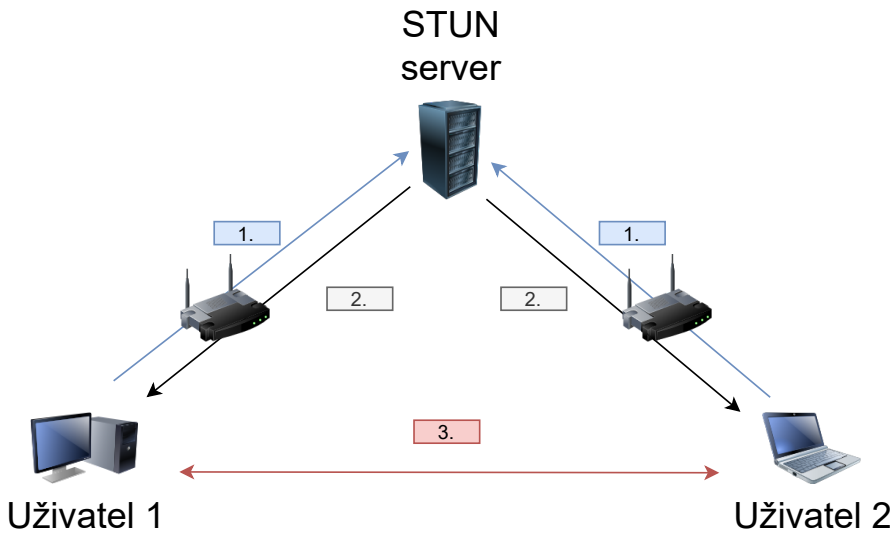
Dále je ještě také dobré zmínit, že k takovéto situaci by nemělo docházet často, jednalo se o nejhorší scénář a zároveň jsme nepočítali s dynamickým bandwidthem, který protokol WebRTC podporuje. Ten upravuje kvalitu a snižuje tak potřebný datový tok k jeho přenosu.

STUN

Další důležitou technologií z kolosu WebRTC je STUN (*Session Traversal Utilities for NAT*). Abychom se mohli dva uživatele propojit napřímo mezi sebou, musí znát svojí veřejnou IP adresu a typ NATu, za kterým jsou „schovaní“. Přesně k těmto účelům slouží STUN server. Ve většině případů je potřeba pouze při zahájení spojení mezi uživateli. Na obrázku 2.4 si v krocích 1–3 ukážeme, jak takový server funguje [23].

1. Uživatel 1 a Uživatel 2 kontaktují STUN Server
2. STUN Server jim vrátí odpověď s informacemi o jejich IP adrese
3. Získanou adresu a informace použijí uživatelé k vytvoření WebRTC nabídky/odpovědi a na konci vytvoří přímé spojení mezi sebou

[1] počítáme s tím, že jedno hlavní video je v plné kvalitě a ostatní v kvalitě horší s bitratem 0,3Mb/s



Obrázek 2.4: Znázornění principu STUN serveru

TURN

I když máme STUN server správně nastaven, může dojít k situaci, kdy je uživatel v síti, která je nějakým způsobem omezena (např. jsou povoleny pouze některé porty, zakázáno UDP apod.). V tom případě přichází na scénu TURN server, který funguje jako prostředník mezi uživateli a data tečou přes něj. Podle testování *Philippa Hanckeho* potřebovalo zhruba 18 % spojení TURN server [24].

2.5.12 SimplePeer

Knihovna obalující WebRTC standard. Ušetří nám spoustu práce při navazování spojení a komunikací s TURN a STUN servery. Má jednoduché rozhraní pomocí událostí, které také využijeme. Dokonce se dá spustit i na serveru, ale tím se v této bakalářské práci zabývat nebudeme [25].

2.5.13 Let's Encrypt

Jedná se o certifikační autoritu, která nabízí TLS/SSL certifikáty zdarma. Dříve nebylo normou, že by webové stránky byly zabezpečené, certifikáty nebyly zdarma a certifikační autority si za to nechávaly slušně zaplatit.

Let's Encrypt založili původně dva pracovníci z Mozilly, jejich cílem bylo to, aby se HTTPS stalo normou a to se jim celkem povedlo. Jedinou menší nevýhodou Let's Encrypt je platnost certifikátu. Ta je pouze tři měsíce a pak je potřeba ji obnovit.

Jinak je ale velmi populární a to i nejen mezi menšími vývojáři, jeho vývoj sponzorují společnosti jako Facebook, Cisco a IBM.

Kapitola 3

Návrh

V následující kapitole se zaměříme na návrh naší webové aplikace s použitím technologií, které jsme si ukázali v minulé kapitole – Analýze. Nejdříve se podíváme na návrh uživatelského rozhraní a poté na návrhu způsobu komunikace mezi klientem/serverem a jednotlivými klienty.

3.1 Wireframe

Wireframe slouží k návrhu uživatelského rozhraní a rozložení komponent na stránce. Naše aplikace bude obsahovat dvě hlavní obrazovky.

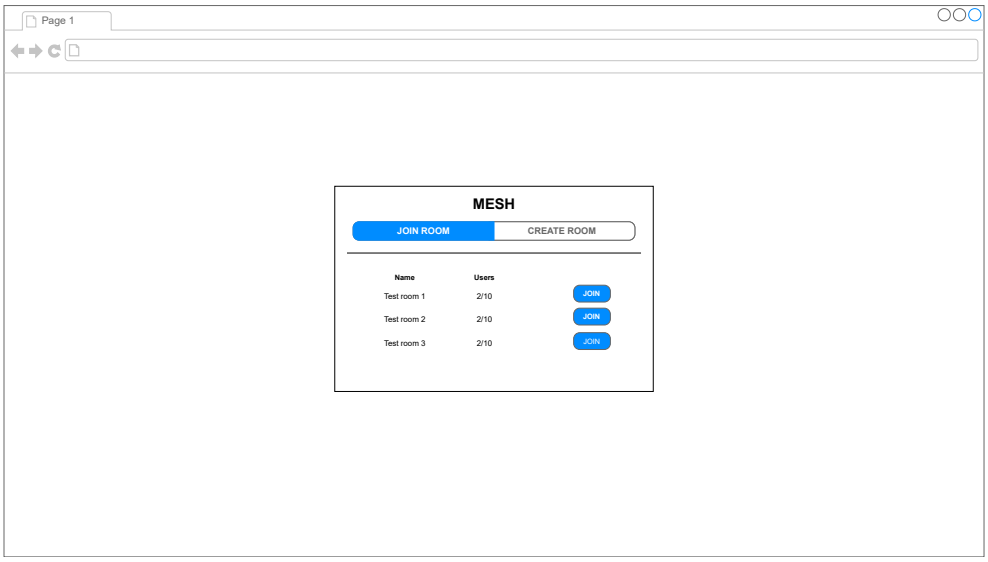
První slouží k napojení do místnosti anebo k vytvoření místnosti. Druhá obrazovka je už pro samotný konferenční hovor, jelikož se jedná o jednostránkovou aplikaci, řešíme spoustu funkcionalit přes modální okna. Okno s hovorem je rozděleno do čtyř částí, které si teď popíšeme.

1. Zvolený uživatel (zde USER1) se zobrazuje uprostřed stránky, pokud má zapnuté video, je celá tato část věnována jemu.

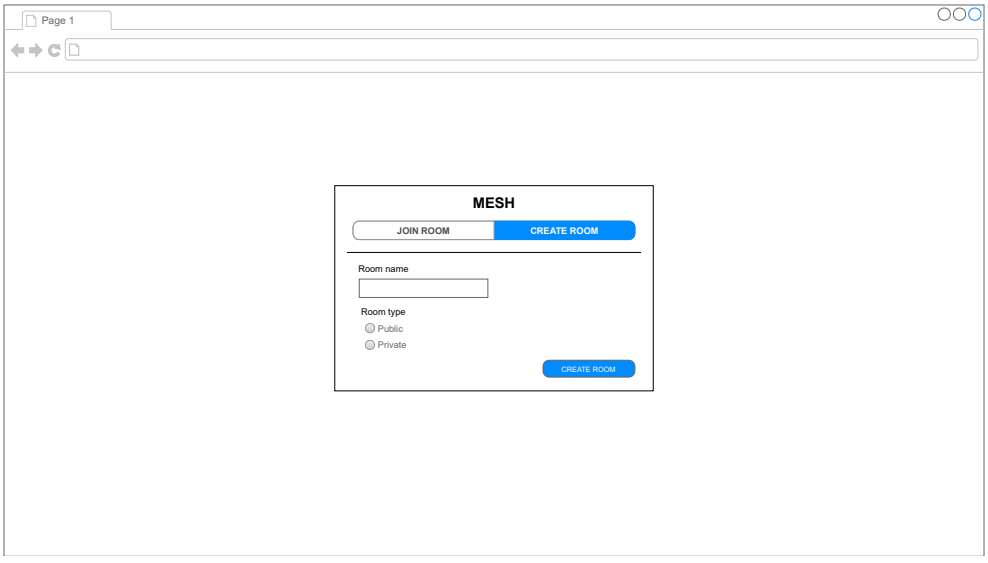
2. Seznam uživatelů společně s miniaturami videí, pokud nějaké sdílejí. Po kliknutí na uživatele se stane uživatel zvoleným a zobrazí se v části 1.

3. Ovládací panel s tlačítky a ukazatelem doby trvání hovoru. V ovládacím panelu se nachází tlačítka pro sdílení kamery, obrazovky, pro zobrazení chatu a pro opuštění hovoru.

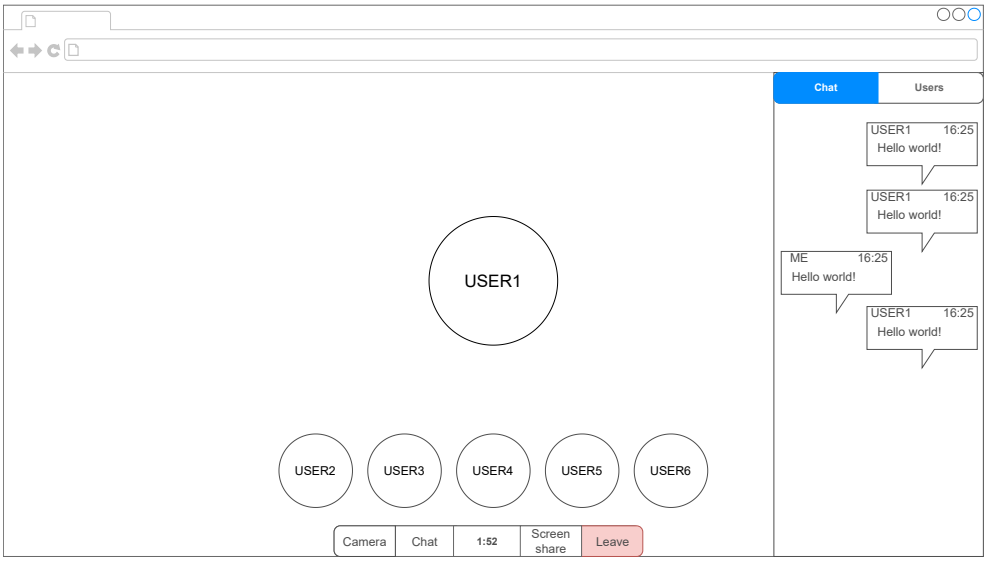
4. Boční vysouvací panel s chatem a seznamem uživatelů. Boční panel je možno skrýt a pomocí tlačítek v horní části přepínat mezi chatem/seznamem uživatelů.



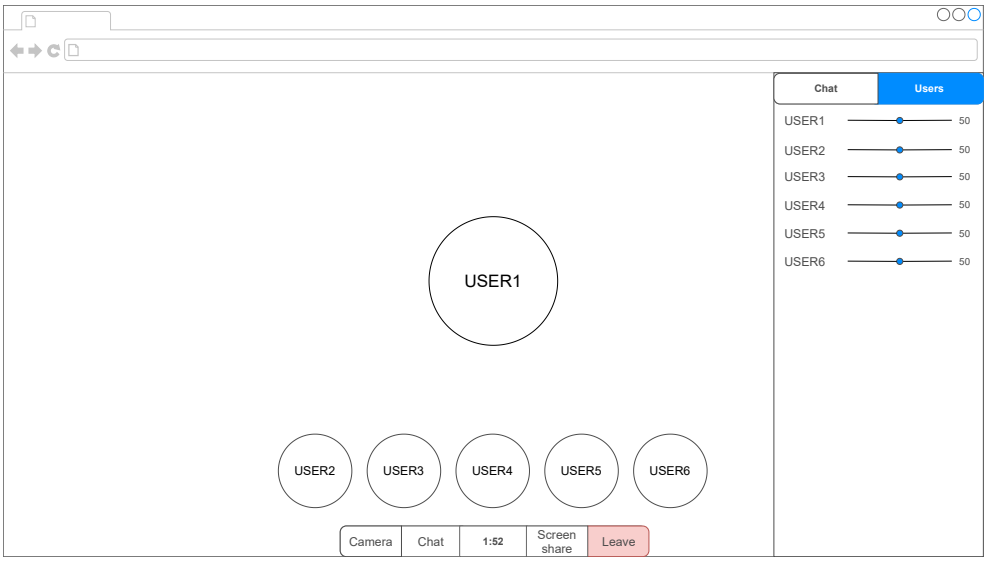
Obrázek 3.1: Obrazovka pro napojení do místnosti



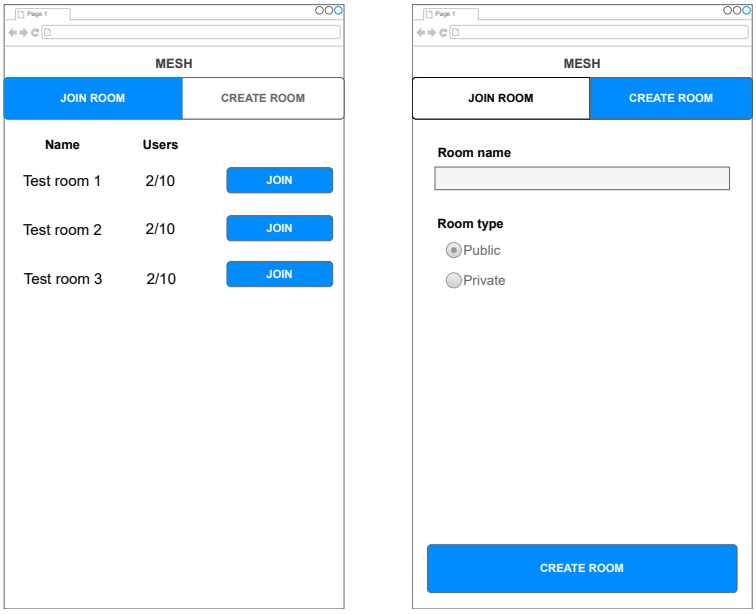
Obrázek 3.2: Obrazovka pro vytvoření nové místnosti



Obrázek 3.3: Obrazovka místnosti s otevřeným chatem

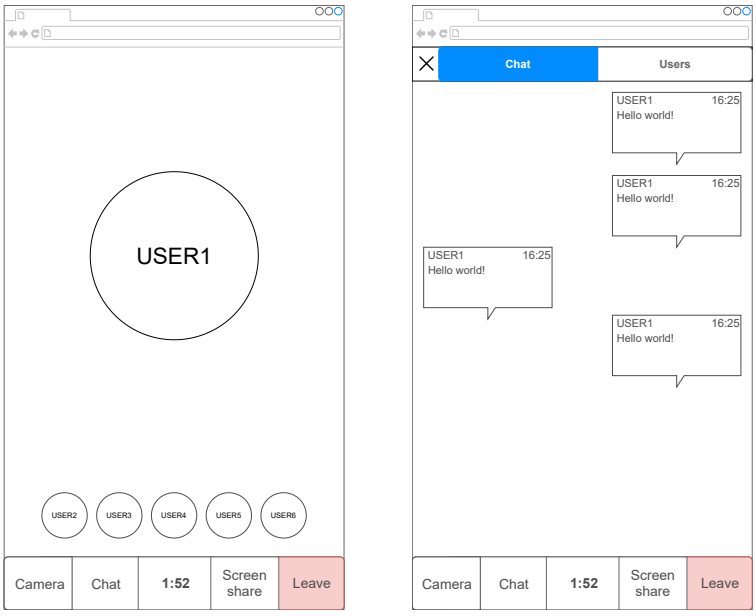


Obrázek 3.4: Obrazovka místnosti s otevřeným seznamem uživatelů



(a) Seznam místností (b) Vytvoření místnosti

Obrázek 3.5: Wireframe pro telefony – Připojení/tvorba místnosti

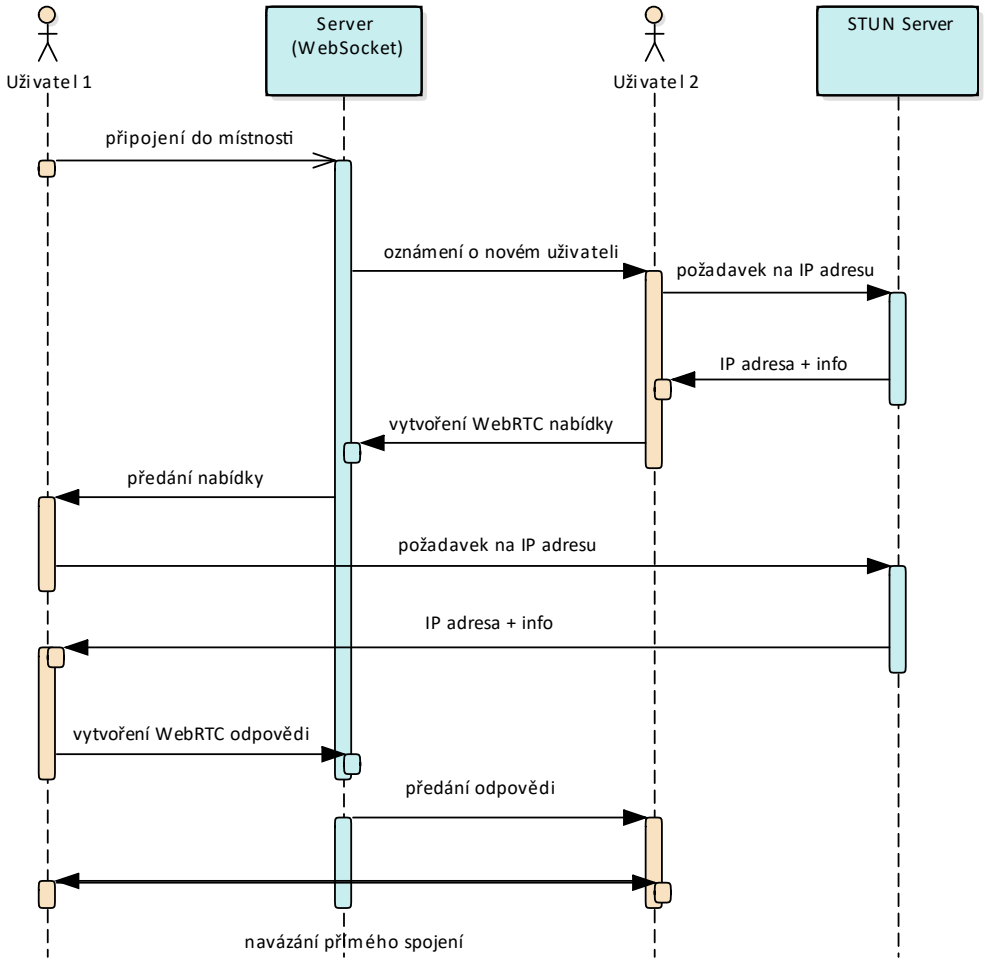


(a) Obrazovka s hovorem (b) Chat

Obrázek 3.6: Wireframe pro telefony – Hovor

3.2 Komunikace

V následující části se podíváme na ukázkový scénář toho, jak bude vypadat spojení nového uživatele s uživatelem, který je již v místnosti.



Obrázek 3.7: Sekvenční diagram pro navázání spojení s novým uživatelem

1. Uživatel 1 se spojí se Socket.IO serverem a zobrazí si výpis místností
2. Připojí se do místnosti
3. Server oznámí připojeným uživatelům nově příchozího uživatele
4. Uživatel 2 zkontaktuje STUN server
5. STUN server mu odpoví s odpovídající veřejnou IP adresou uživatele a dalšími informacemi

6. Uživatel 2 na základě informací od STUN serveru vytvoří WebRTC nabídku a předá jí Socket.IO serveru
7. Server ji předá Uživateli 1
8. Uživatel 1 také zkontaktuje STUN server
9. STUN server mu také odpoví
10. Uživatel 1 vytvoří WebRTC odpověď a předá jí serveru
11. Server předá odpověď Uživateli 2
12. Přímé spojení je navázáno a odteď probíhá přímá komunikace mezi nimi

3.2.1 Backend

Aby zapadlo všechno do sebe, budeme potřebovat navrhnout nějaký způsob komunikace mezi klientem/serverem a mezi jednotlivými klienty. Pro první případ používáme knihovnu Socket.IO, kde je možné si vytvořit vlastní formát zpráv. V následující tabulce 3.1 si ukážeme zprávy, které budeme přijímat na serveru od klientů.

Tabulka 3.1: Události

Typ zprávy	**Parametry**	**Popis**
createRoom	jméno, typ	Vytvoření nové místnosti.
joinRoom	id místnosti	Připojení uživatele do místnosti.
disconnect		Jedná se událost od samotné knihovny Socket.IO, slouží k detekci odpojení socketu.
leaveRoom		Stejná funkce jako u předchozí zprávy, zavolá se, pokud uživatel opustí hovor.
peer	nabídka, id nabídky	Slouží k navázání přímého spojení mezi klienty.
rooms		Vrací seznam místností a jejich počet uživatelů.
setName	jméno uživatele	Slouží k nastavení jména uživatele.

3.2.2 Frontend

V následující části si popíšeme události vytvořené knihovnou SimplePeer, na ní v podstatě celá naše aplikace stojí. Bude proto důležité tyto události zpracovávat a na některé z nich se dále zaměříme i v implementační části.

3.2.3 Události knihovny SimplePeer

Tabulka 3.2: Události vyvolané knihovnou SimplePeer

typ zprávy	**parametry**	**popis**
connect		Událost vyvolaná při úspěšném navázání přímého spojení mezi dvěma uživateli.
stream	stream	Dostali jsme nový stream od druhé strany.
error	chybová hláška	Během komunikace nastala chyba. Pokud už byl uživatel připojen, je nutné ho odstranit ze seznamu uživatelů.
close		Uživatel se odpojil, stejný scénář jako u předchozí události.
signal		Byla vytvořena nabídka/odpověď, kterou je potřeba předat druhému uživateli.
data	data	Událostní zpráva vytvořená námi, viz tabulka. 3.3

Poslední událost využijeme pro naše vlastní zprávy a popíšeme si je v další tabulce.

3.2.4 Vlastní události

Čtenáři se může zdát divné to, že dáváme uživateli oznámení o příchozím streamu a o vypnutí streamu, když už knihovna SimplePeer obsahuje událost pro stream. Je to kvůli tomu, že dostaneme objekt MediaStream. Ten obsahuje

seznam s audio/video stopami a unikátní id, ale už nic o tom, o jaký typ videa se jedná.

To vyřešíme tím, že ještě předtím, než odešleme stream, tak pošleme onstream zprávu, ta bude obsahovat jeho identifikátor a typ. Identifikátor si společně s typem uložíme do mapy a potom, co stream dostaneme už ho dokážeme snadno identifikovat.

Dalším problémem je detekce toho, kdy uživatel stream vypne. Knihovna SimplePeer nic takového nenabízí, a proto se o to musíme postarat ručně. Řešení je celkem jednoduché, jednoduše pošleme zprávu streamoff s identifikátorem streamu a na základě toho ho odstraníme z mapy a vypneme.

Tabulka 3.3: Vlastní události mezi uživateli

Typ zprávy	**Parametry**	**Popis**
streamoff	id	Došlo k vypnutí video streamu.
onstream	id, typ	Informace o příchozím streamu a jeho typu.
muted	ano/ne	Uživatel si vypnul/zapnul mikrofon.
chat	id uživatele, zpráva	Nová zpráva v chatu.
name	jméno	Uživatel si změnil jméno.

Kapitola 4

Implementace

Další fází softwarového vývoje je implementace, v této části si ukážeme implementaci klientské části (frontendu) a serverové části (backendu). Jako vstup pro ní nám poslouží výsledky analytické části a návrhové části.

Výsledný zdrojový kód bude dostupný na adrese portálu github.com (server[1], client[2]) a také na přiloženém CD. Verze se mohou lišit, nejnovější bude vždy na portálu GitHub. Jedná se o platformu, kde je možné zdarma uchovávat a verzovat svůj kód a spolupracovat na něm s ostatními lidmi. Dříve bylo potřeba za službu platit, pokud jsme chtěli mít svůj kód soukromý, ale po odkoupení Microsoftem to již není podmínkou [26]. Snímky obrazovky zachycující výsledky implementace naleznete v příloze A.

4.1 Implementace Frontendu

4.1.1 Získání audio/video streamu

Z nadpisu se může zdát, že se jedná o lehkou záležitost. To by platilo, pokud bychom psali desktopovou aplikaci, ve světě prohlížečů to ale takhle nefunguje. Pro přístup k připojeným multimediálním zařízením slouží `MediaDevices API`. Zajímat nás bude konkrétně funkce `getUserMedia()`. Pokud ji zavoláme se správnými parametry, dostaneme od prohlížeče stream s audio/video stopami.

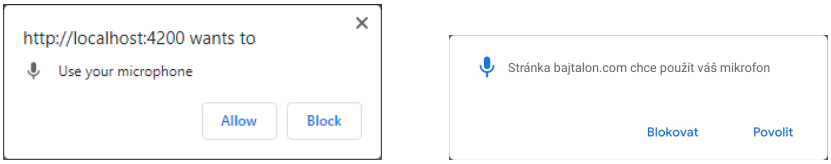
Ukázka kódu pro získání videa z kamery v rozlišení 1920x1080 s 15 snímky za sekundu a audiem.

[1] `https://github.com/Bajt24/meshserver`
[2] `https://github.com/Bajt24/meshclient`

```
navigator.mediaDevices.getUserMedia(
{
    video: {
        width: 1920,
        height: 1080,
        frameRate: 15
    },
    audio: true
})
```

Obrázek 4.1: Ukázka kódu k získání videa od uživatele

Pokud je kamera a mikrofon na zařízení dostupný, zobrazí se nám v prohlížeči dialog, který musíme potvrdit. V každém prohlížeči vypadá dost podobně, na obrázku 4.2 jsem pro srovnání vybral Google Chrome na desktopu a na Androidu. Dost zvláštní je prohození tlačítek Povolit/Blokovat.



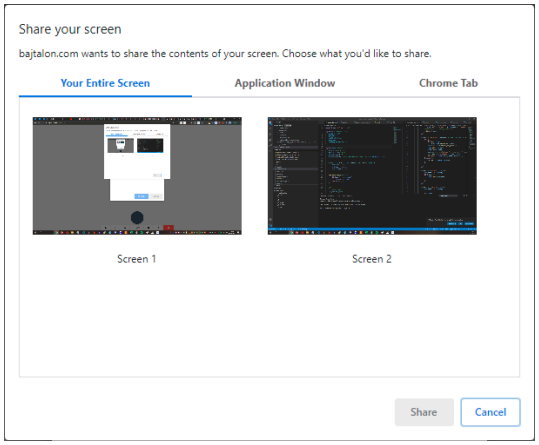
(a) Chrome na desktopu (b) Chrome na Androidu

Obrázek 4.2: Porovnání dialogů pro přístup k mikrofonu

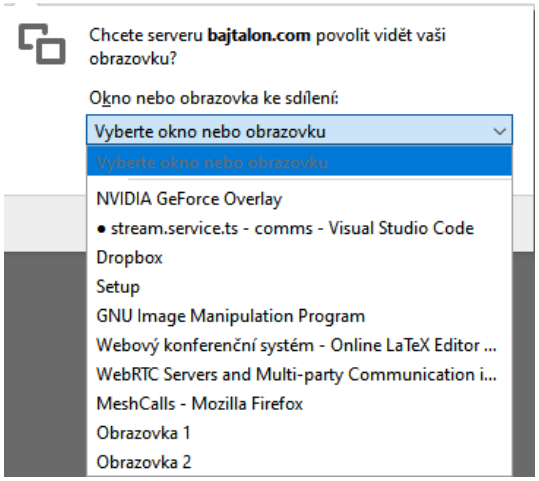
4.1.2 Získání streamu z obrazovky

V předchozí sekci jsme se zabývali MediaDevices API, to zde použijeme také. Pro získání obrazovky použijeme funkci `getDisplayMedia()`. Toto volání už není mezi prohlížeči tak rozšířené, jako to minulé. Podporují ho pouze desktopové prohlížeče a každý z nich si dialog přizpůsobuje k obrazu svému. Na obrázkách 4.3 a 4.4 můžeme vidět srovnání prohlížečů Google Chrome a Mozilla Firefox.

Dobré je ještě také připomenout, že volání funkcí `getUserMedia()` a `getDisplayMedia()` je možné pouze na zabezpečených stránkách s protokolem HTTPS. Toto omezení se samozřejmě netýká vývoje na lokálním serveru (localhostu).



Obrázek 4.3: Dialog pro výběr sdílení obrazovky v Google Chrome



Obrázek 4.4: Dialog pro výběr sdílení obrazovky v Mozilla Firefox

4.1.3 Přehrávání získaného audia/videa

V návrhu jsme si v tabulce 3.2 ukázali způsob komunikace mezi klienty. Pokud obdržíme od jiného uživatele audio/video stream, budeme potřebovat ho nějak přehrát. Na prvním řádku v ukázce kódu 4.5 vidíme, že získaný stream je typu `MediaStream`. Ten obsahuje funkci `getVideoTracks()`, kterou zjistíme, zda stream obsahuje nějaké video stopy nebo ne. Podle toho pak nastavíme uživatelovy proměnné.

```
this.peer.on("stream", (stream: MediaStream) => {
    if (stream.getVideoTracks().length > 0) {
        this.videoStream = stream;
    } else {
        this.stream = stream;
        this.setSoundMeter(this.stream);
        this.isMuted = false;
    }
})
```

Obrázek 4.5: Ukázka kódu – získání streamu

Nyní se podíváme do kódu se šablonou, kde vidíme HTML element `<audio>`. Ten slouží k přehrávání zvukových souborů nebo audio streamů, které specifikujeme nastavením proměnné `srcObject`. Hranaté závorky znamenají ve světě Angularu *property binding*, jedná se o způsob nastavování proměnných objektů ze šablon. Další proměnnou je `volume`, zde nastavujeme hlasitost streamu, tu budeme nastavovat u každého uživatele individuálně.

Atribut `autoplay` pak slouží k automatickému přehrávání streamu, pokud bychom ho nepoužili, stream by se spustil až poté, co bychom na něj zavolali funkci `play()`. A poslední věcí je direktiv `*ngIf`, ten slouží jako podmínka, pod kterou se má tento HTML element vykreslit. V našem případě se vykreslí, pokud vybraný uživatel není prázdný (null).

Pro zobrazení videa bude šablona vypadat dost obdobně, akorát použijeme HTML element `<video>`.

```
<audio *ngIf="selectedUser" [srcObject]="selectedUser.stream"
autoplay [volume]="selectedUser.volume">

</audio>
```

Obrázek 4.6: Ukázka kódu – přehrávání audia

4.1.4 Chat

V této části se zaměříme na odesílání zpráv a přijímání zpráv, jedná se už o čistě frontendovou záležitost, zprávy si posílají klienti mezi sebou, server do toho už není nijak zapojen. V návrhové části jsme si v tabulce 3.3 zavedli vlastní událost pro novou zprávu a tu zde také využijeme.

Pro okno s chatem vytvoříme novou komponentu – `VoiceChatComponent`, jejíž konstruktor je vidět na ukázce 4.7. Zde se přihlásíme k odběru o nové příchozí zprávě ze služby `PeerService`. Po oznámení přidáme zprávu do pole

`msgs` a řekneme Angularu, ať sescrolluje v okně s chatem až dolu, kde jsou nejnovější zprávy.

```
constructor(private ps: PeerService) {
    this.ps.onChatMessage.subscribe((m: Message) => {
        this.msgs.push(m);
        this.scrollToBottom();
    });
}
```

Obrázek 4.7: Ukázka kódu – Chat

O zobrazení zpráv se pak postará šablona, tu lze vidět v ukázce 4.8. Zprávy vykreslujeme pomocí direktivu `*ngFor`, jedná se o for cyklus, který pro každou zprávu v poli `msgs` vytvoří nový `<div>`. Dále pak může vidět na řádcích níže složené závorky s proměnnými, jedná se o způsob vypsání hodnoty proměnných do šablony v Angularu, kterému říkáme *interpolace*. Nemusíme se bát nějakých útoků typu XSS[1], to už je vyřešeno za nás.

```
<div id="chat" #chat [scrollTop]="chat.scrollHeight">
   <div *ngFor="let item of this.msgs" class="chatBubble">
        <div class="chatMsgHeader">
            <p>{{item.senderName}}</p>
            <p>{{item.time | date:'H:mm'}}</p>
        </div>
        <p class="chatText">
            {{item.text}}
        </p>
    </div>
</div>
```

Obrázek 4.8: Ukázka kódu – Šablona chatu

Jelikož se zprávy posílají mezi klienty, tak pokud chceme zprávu zobrazit všem, musíme ji také všem odeslat. K tomu slouží metoda `sendChatMessage()` ze služby `PeerService`. V mapě `users` si uchováváme všechny uživatele. Ty v cyklu projdeme a zprávu jim odešleme.

[1] jedná se o bezpečnostní chybu ve webové aplikaci, kterou útočníci využívají ke spuštění jejich vlastního JavaScript kódu

```
sendChatMessage(msg: string) {
    for (let [key, value] of this.users) {
        if (key == "me")
            continue;
        value.sendData("chat", msg);
    }
}
```

Obrázek 4.9: Ukázka kódu – Odesílání zprávy v chatu

4.1.5 Renegotiation

Pokud chceme do stávajícího WebRTC připojení přidat nebo odebrat multimediální stream, je potřeba provést *renegotiation*. To znamená, že musí dojít k vytvoření nové WebRTC nabídky a nové WebRTC odpovědi. Knihovna SimplePeer se o vytvoření nových nabídek/odpovědí na klientovi stará už za nás, ale na serveru je dobré si na to dát pozor.

V podstatě je potřeba znova provést část s nabídkami/odpověďmi z diagramu 3.7, ale už není potřeba kontaktovat znovu STUN server.

4.1.6 Směrování

Směrování neboli *routing* v Angularu je způsob, jak zobrazovat uživateli různé komponenty na základě URL adresy. V Angularu se o to stará `RoutingModule`, naše aplikace obsahuje dvě hlavní obrazovky, takže nebudeme potřebovat složité směrování.

Jak můžeme na kódu 4.10 vidět, budou nám stačit dvě cesty – `call` a `landing`, všechny ostatní cesty (zde značíme **) budeme přesměrovávat na `landing`.

```
const routes: Routes = [
    {
        path: 'call', component: VoiceRoomComponent
    },
    {
        path: 'landing', component: LandingComponent
    },
    {
        path: '**', redirectTo: '/landing'
    }
];
```

Obrázek 4.10: Ukázka kódu – směrování

Ve funkčním požadavku F03 jsme zmiňovali možnost připojit se do místnosti pomocí URL adresy. To vyřešíme pomocí GET parametru `id` v adrese. Každé místnosti vygenerujeme po jejím vytvořením unikátní čtyřznakový identifikátor. Pokud by tedy měla místnost identifikátor `a1b2`, bude adresa pro připojení vypadat takto `/call?id=a1b2`.

4.2 Implementace Backendu

4.2.1 STUN a TURN server

V analytické části jsme zmiňovali nutnost STUN a TURN serveru, bez nich by nám spojení nefungovalo. Samozřejmě je možné si takový server napsat sám, specifikace jsou volně k dispozici. Pro účely této bakalářské práce jsem se rozhodl využít už hotové řešení, které bude určitě i bezpečnější. Na npm jsem našel balíček node-turn [27], který přesně plní tento účel.

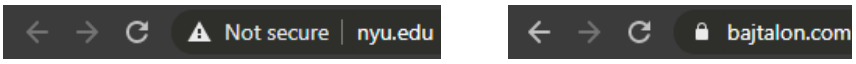
Node-turn nabízí možnost spustit ho jako samostatný proces anebo ho integrovat přímo do naší aplikace. Lepší pro nás bude druhá varianta, protože nabízí možnost dynamicky přidávat a měnit přihlašovací údaje. To se dá využít v případech, kdy nechceme, aby náš server využíval někdo cizí. Implementace bude vypadat tak, že pro každou nově vytvořenou místnost vytvoříme i nové přihlašovací údaje ke STUN/TURN serveru a uživatelům je pošleme.

4.2.2 HTTPS

V jednom z našich nefunkčních požadavků jsme zmiňovali, že chceme, aby byla naše stránka přístupná pouze přes protokol HTTPS. Nejde tedy jenom o to, aby se v adresní řádce neukazovalo *Not secure*, viz obrázek 4.11, ale i o bezpečnost. Pokud by stránka byla nezabezpečená, mohly by se například data

z formulářů dostat do špatných rukou. Také jak již bylo zmíněno v implementaci frontendu, nefungovalo by nám získávání videa/audio v prohlížeči.

V analytické části jsme se seznámili se službou Let's Encrypt. V této sekci se tedy zaměříme na to, jak od ní certifikát získat a jak ho použít v naší aplikaci.



(a) Stránka servírovaná přes HTTP

(b) Stránka servírovaná přes HTTPS

Obrázek 4.11: Ukázka rozdílu v adresním řádku u Google Chrome

4.2.2.1 Certbot

Program Certbot ulehčuje celý proces získávání a obnovování certifikátu. Certifikát můžeme obnovovat i ručně za pomocí Certbota a to příkazem `certbot renew`, ale mnohem pohodlnější je nastavit ho jako plánovanou úlohu (cron). Pak se o certifikát už nemusíme starat.

4.2.2.2 Express.js

Výše zmíněné bylo potřeba udělat proto, abychom získali 3 soubory – `privkey.pem`, `cert.pem` a `chain.pem`. Jedná se soubory potřebné ke spuštění HTTPS serveru. Implementace bude vypadat takto – 4.12.

Pokud se načtení certifikátu podaří, spustí se server na portu 443, což je port používaný právě pro HTTPS. Jinak se spustí na portu 80.

4.2.3 Přesměrování z HTTP na HTTPS

Naše aplikace už teď běží na serveru, ale má to jednu velkou nevýhodu. Běží pouze na HTTPS, to znamená, že pokud uživatel zadá adresu serveru bez `https://` předpony, prohlížeč bude ve výchozím nastavení zkoušet pouze protokol HTTP. A pokud na něm nic nenajde, nezkusí ani HTTPS.

Možností jak toto vyřešit je více, mohl by to za nás vyřešit i nějaký webový server typu Apache nebo Nginx, ale v této práci se podíváme na to, jak to zajistit přímo v Node.js. Nejlehčí metodou bude spustit další webový server, který bude běžet na portu 80 (HTTP) a naše uživatele bude přesměrovávat na zabezpečený port 443 (HTTPS).

Pokud tedy uživatel přistoupí přes HTTP, vrátíme mu status kód 301 Redirect společně s novou adresou, která už obsahuje správně `https://` předponu jak je vidět na 4.13.

```
try {
    let credentials = {
        key: fs.readFileSync('privkey.pem', 'utf8'),
        cert: fs.readFileSync('cert.pem', 'utf8'),
        ca: fs.readFileSync('chain.pem', 'utf8')
    };
    this.httpServer = https.createServer(credentials, this.app);
    this.isHTTPS = true;
} catch (e) {
    this.httpServer = require('http').createServer(this.app);
}
...
let port = 80;
if (this.isHTTPS)
    port = 443;

let httpSrvr = this.httpServer.listen(port, () => {
    console.log('Server started at :' + port);
});
```

Obrázek 4.12: Ukázka kódu – HTTPS

```
http.createServer((req, res) => {
    res.writeHead(301,
    { "Location": "https://" + req.headers['host'] + req.url });
    res.end();
}).listen(80);
```

Obrázek 4.13: Ukázka kódu – Přesměrování

Kapitola 5

Testování

Posledním bodem této bakalářské práce bylo podrobit ji testování. A to konkrétně při zatížení jedné výukové skupiny – přibližně 20 uživatelů.

Toto se nám podařilo zrealizovat společně s mým vedoucím Ing. Viktorem Černým dne 6. 5. 2021. Ten oslovil studenty z předmětu BI-PSI, který vyučuje a našlo se více než dost dobrovolníků ochotných pomoci.

Testovací scénář

Pro testování byl sestaven následující scénář:

1. dobrovolníci si změří rychlost svého připojení
2. v aplikace se vytvoří a dobrovolníci se do ní připojí
3. po napojení si každého dobrovolníka vyvoláme a vyzkoušíme funkčnost mikrofonu
4. to samé uděláme i pro webkamery a sdílení obrazovky
5. vyzkoušíme chat, každý dobrovolník napíše nějakou zprávu do chatu
6. posledním bodem bude nejhorší možný scénář, každý uživatel zapne kameru a obrazovku zároveň

Dále byl vytvořen dotazník, který měli dobrovolníci za úkol vyplnit. Ptali jsme se v něm na to, jak byli spokojeni s kvalitou audia a videa, na rychlost jejich připojení a co by případně zlepšili.

Napojení do místnosti a test audia

Napojení do místnosti proběhlo v pořádku, většina účastníků neměla s připojením problém, některým se ale nepodařilo spojení navázat a během hovoru vypadávalo. Nejčastěji hodnotili kvalitu audia známkou 2.

Testování videa a chat

Dále přišlo na řadu testování videa, to už bylo o dost horší, u některých bylo video vidět, ale u některých zase ne. Byla zde vidět korelace mezi rychlostí připojení a kvalitou/stabilitou videa. Ti co měli pomalé připojení, tak většinou zaznamenávali problémy, ostatní na tom byli dobře. Celkově ho nejčastěji hodnotili známkou 3. Chat fungoval v pořádku, což bylo očekávané. Nevyžaduje žádný vysoký datový tok narozdíl od videa.

Finální test

Finální test dopadl stejně jako test videa, opět měli nevýhodu ti, co mají pomalé připojení. Tady se ukázalo to, co bylo už předem v práci zmiňováno. Peer-to-peer topologie není vhodná na takto velký počet uživatelů, špatně škáluje pokud je sdíleno více video streamů a hodí se tak pro menší počet lidí.

Uživatelské rozhraní

Dále jsme se uživatelů ptali na to, jak se jim líbí uživatelské rozhraní a co by na něm zlepšili. Většina byla spokojená, u některých zlobil ukazatel ztlumeného mikrofonu. Uživatelé mluvili i když měli u jejich jména ukázáno, že mají mikrofon ztlumený. Dalším problémem bylo zobrazování v dolní části s kolečky, občas se stalo to, že část s nimi přetekla a nebyla vidět.

Zhodnocení

Celkově ale dopadlo testování dobře, zjistili jsme a potvrdili jsme si, kde má aplikace slabé stránky a kde naopak silné. Bylo by vhodné přidat nějaký způsob logování v klientu aplikace, abychom zjistili, proč u některých účastníků docházelo během hovoru k výpadkům spojení.

Závěr

Cíle které jsme si stanovili na začátku práce se podařilo splnit více než dobře. Do prototypu se podařilo naimplementovat všechno ze zadání a funkčních požadavků. Aplikace umožňuje vytvořit místnost, ve které je umožněna hlasová a video komunikace. Dále aplikace disponuje rozhraním pro chat a přenos obrazovky. Je možné si spustit server na vlastním stroji a zdrojové kódy si upravit k libosti své.

V testování se potvrdilo, že WebRTC topologie Mesh není moc vhodná pro vyšší počet uživatelů. To ale nic nemění na tom, že aplikace by neměla své uplatnění. Hodí se převážně pro menší týmy, které nechtějí utrácet za drahá řešení a omezený počet uživatelů jim nevadí.

Aplikace podporuje většinu prohlížečů a to i na mobilních zařízeních. Dále je pak upraveno uživatelské rozhraní dle rozlišení a poměru stran zařízení tak, aby byl uživatelský zážitek co nejlepší.

Další možnosti rozšíření práce

Aplikace obsahuje základní funkcionalitu a představuje tak dobrou kostru pro další vývoj. Spousta důležitých funkcí ještě chybí, vybral jsem pár nápadů, které by stály za to do aplikace přidat.

- Možnost tvorby vlastního účtu – nastavení jména, profilové fotky apod.
- Přihlášení/Registrace – to souvisí i s předchozím bodem, aplikace by se rozšířila o databázi
- Více možností rozložení – aktuálně nabízí aplikace v hovoru zobrazení jednoho velkého uživatele a zbytek jako miniatury, hodilo by se vytvořit nějaké další rozložení, třeba dlaždicové, kde by všichni měli stejně místa
- Administrace místnosti – pokud uživatel vytvoří místnost, stal by se z něj administrátor, který by měl možnost vyhodit uživatele nebo ho třeba ztišit

Literatura

[1] Zoom grows to 300 million meeting participants despite security backlash [online]. *The Verge*, duben 2020, [cit. 2021-03-21]. Dostupné z: https://www.theverge.com/2020/4/23/21232401/zoom-300-million-users-growth-coronavirus-pandemic-security-privacy-concerns-response

[2] Microsoft Teams reaches 115 million DAU—plus, a new daily collaboration minutes metric for Microsoft 365 [online]. *Microsoft*, říjen 2020, [cit. 2021-03-21]. Dostupné z: https://www.microsoft.com/en-us/microsoft-365/blog/2020/10/28/microsoft-teams-reaches-115-million-dau-plus-a-new-daily-collaboration-minutes-metric-for-microsoft-365/

[3] Google's Meet teleconferencing service now adding about 3 million users per day [online]. *The Verge*, duben 2020, [cit. 2021-03-21]. Dostupné z: https://www.theverge.com/2020/4/28/21240434/google-meet-three-million-users-per-day-pichai-earnings

[4] Here Are The 10 Most Downloaded Apps Of 2020 [online]. *Forbes*, leden 2021, [cit. 2021-03-21]. Dostupné z: https://www.forbes.com/sites/johnkoetsier/2021/01/07/here-are-the-10-most-downloaded-apps-of-2020/?sh=c11d13e5d1ab

[5] SUCCESS A Bill Gates speech inspired Zoom founder to start an internet business—now he's a billionaire [online]. *The Verge*, duben 2019, [cit. 2021-03-21]. Dostupné z: https://www.cnbc.com/2019/04/18/zoom-ipo-bill-gates-speech-inspired-founder-to-move-to-us.html

[6] Zoom is leaking some user information because of an issue with how the app groups contacts [online]. *The Verge*, březen 2020, [cit. 2021-03-27]. Dostupné z: https://www.theverge.com/2020/3/31/21201956/

zoom-leak-user-information-email-addresses-photos-contacts-directory

[7] Zoom lied to users about end-to-end encryption for years, FTC says [online]. *Ars Technica*, září 2020, [cit. 2021-03-27]. Dostupné z: https://arstechnica.com/tech-policy/2020/11/zoom-lied-to-users-about-end-to-end-encryption-for-years-ftc-says/

[8] Zoom admits some calls were routed through China by mistake [online]. *TechCrunch*, duben 2020, [cit. 2021-03-27]. Dostupné z: https://techcrunch.com/2020/04/03/zoom-calls-routed-china/

[9] Microsoft reportedly wanted to buy Slack for $8 billion [online]. *MS PowerUser*, březen 2016, [cit. 2021-03-21]. Dostupné z: https://mspoweruser.com/microsoft-reportedly-wanted-buy-slack-8-billion/

[10] Google quietly launches Meet, an enterprise-friendly version of Hangouts [online]. *TechCrunch*, únor 2017, [cit. 2021-03-21]. Dostupné z: https://techcrunch.com/2017/02/28/google-quietly-launches-meet-an-enterprise-friendly-version-of-hangouts/

[11] Node.js [online]. březen 2021, [cit. 2021-03-23]. Dostupné z: https://nodejs.org/en/

[12] Typescript : Typed JavaScript at Any Scale. [online]. březen 2021, [cit. 2021-03-23]. Dostupné z: https://www.typescriptlang.org/

[13] Angular [online]. březen 2021, [cit. 2021-03-23]. Dostupné z: https://angular.io/

[14] Can I use WebSockets? [online]. březen 2021, [cit. 2021-03-23]. Dostupné z: https://caniuse.com/?search=websocket

[15] Socket.IO [online]. březen 2021, [cit. 2021-03-23]. Dostupné z: https://socket.io/

[16] Express - Node.js web application framework. květen 2021, [cit. 2021-05-10]. Dostupné z: https://expressjs.com/

[17] Font Awesome. květen 2021, [cit. 2021-05-10]. Dostupné z: https://fontawesome.com/

[18] Bootstrap : The most popular HTML, CSS and JS library in the world [online]. březen 2021, [cit. 2021-03-23]. Dostupné z: https://getbootstrap.com/

[19] Introduction - Material Design. duben 2021, [cit. 2021-04-08]. Dostupné z: https://material.io/design/introduction

[20] Material Design for Bootstrap 5 & 4. duben 2021, [cit. 2021-04-08]. Dostupné z: https://mdbootstrap.com/

[21] Can I use WebRTC? [online]. březen 2021, [cit. 2021-03-22]. Dostupné z: https://caniuse.com/?search=webrtc

[22] WebRTC Servers and Multiparty Communication in WebRTC. *Ant Media*, leden 2019, [cit. 2021-04-01]. Dostupné z: https://antmedia.io/webrtc-servers/

[23] What is a STUN/TURN Server? červenec 2018, [cit. 2021-04-05]. Dostupné z: https://blog.ivrpowers.com/post/technologies/what-is-stun-turn-server/

[24] What kind of TURN server is being used? srpen 2017, [cit. 2021-04-05]. Dostupné z: https://medium.com/the-making-of-whereby/what-kind-of-turn-server-is-being-used-d67dbfc2ff5d

[25] simple-peer [online]. březen 2021, [cit. 2021-03-25]. Dostupné z: https://github.com/feross/simple-peer

[26] Friedman, N.: New year, new GitHub: Announcing unlimited free private repos and unified Enterprise offering [online]. leden 2019, [cit. 2021-05-10]. Dostupné z: https://github.blog/2019-01-07-new-year-new-github/

[27] node-turn. květen 2021, [cit. 2021-05-10]. Dostupné z: https://www.npmjs.com/package/node-turn

Příloha A

Seznam použitých zkratek

URL Uniform Resource Locator

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

HTML Hypertext Markup Language

NPM Node Package Manager

API Application Programming Interface

SFU Selective Forwarding Unit

MCU Multipoint Conferencing Unit

FHD Full High Definition

NAT Network Address Translation

STUN Session Traversal Uti-lities for NAT

TURN Traversal Using Relays around NAT

UDP User Datagram Protocol

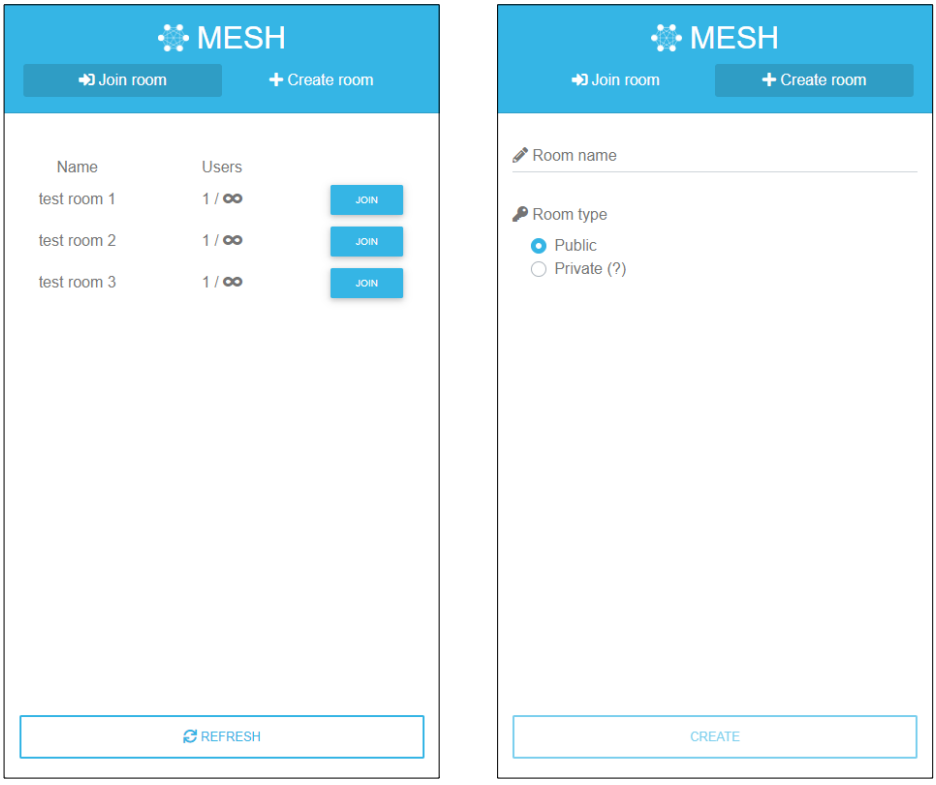
XSS Cross-site scripting

TLS Transport Layer Security

SSL Secure Sockets Layer

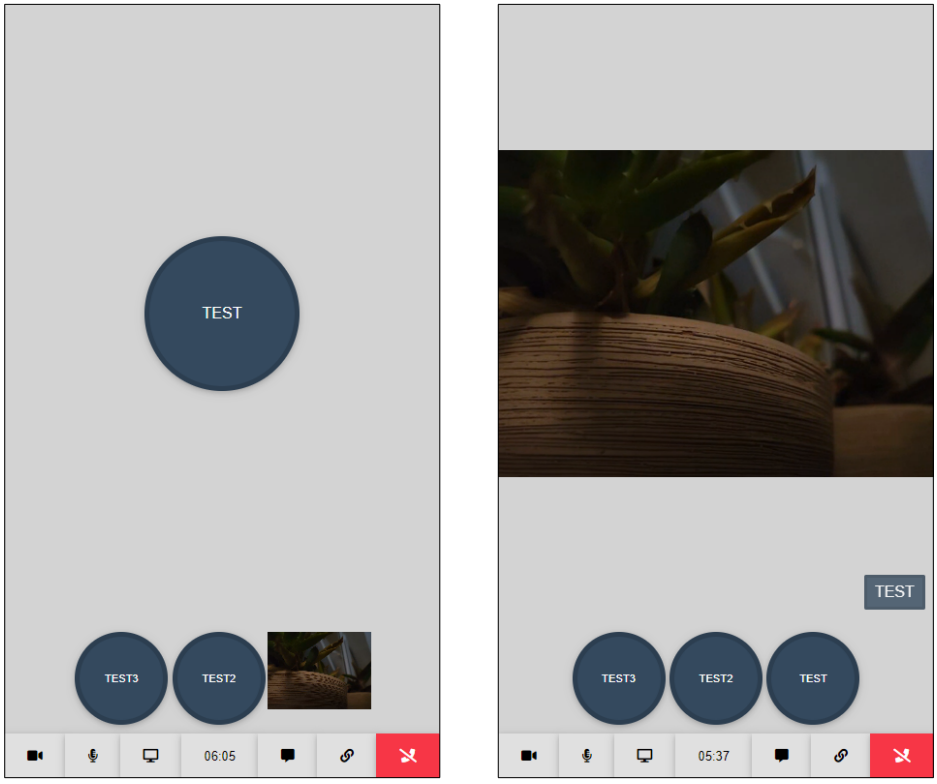
Příloha B

Výsledná podoba aplikace



(a) Seznam dostupných místností (b) Tvorba vlastní místnosti

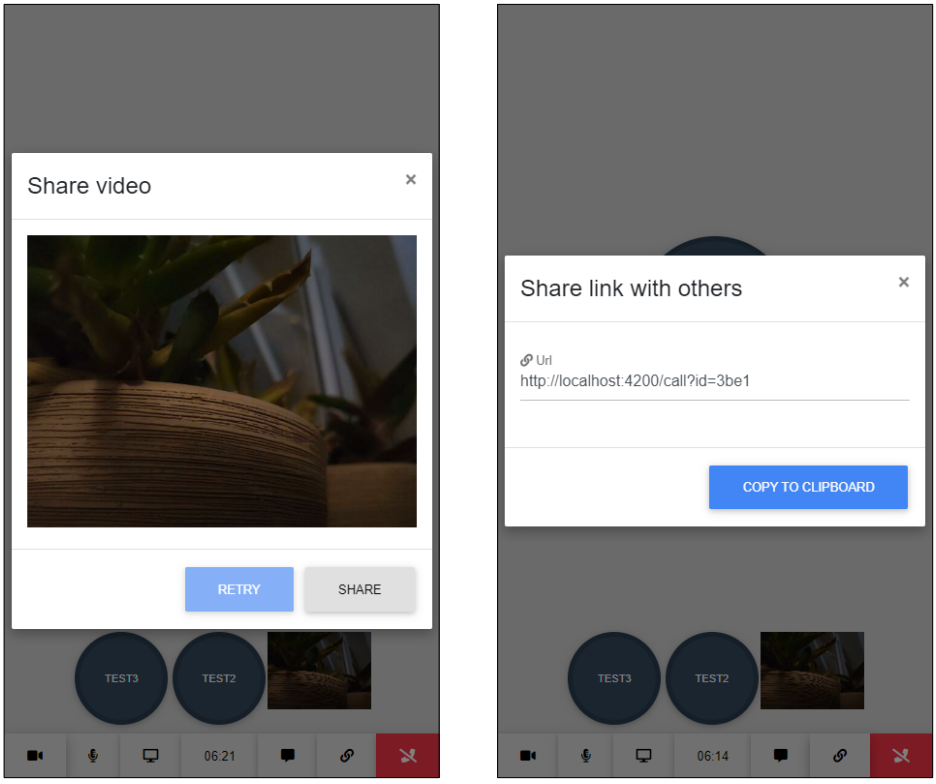
Obrázek B.1: Obrazovka pro napojení a tvorbu místnosti



(a) Hovor

(b) Vybraný uživatel sdílející webkameru

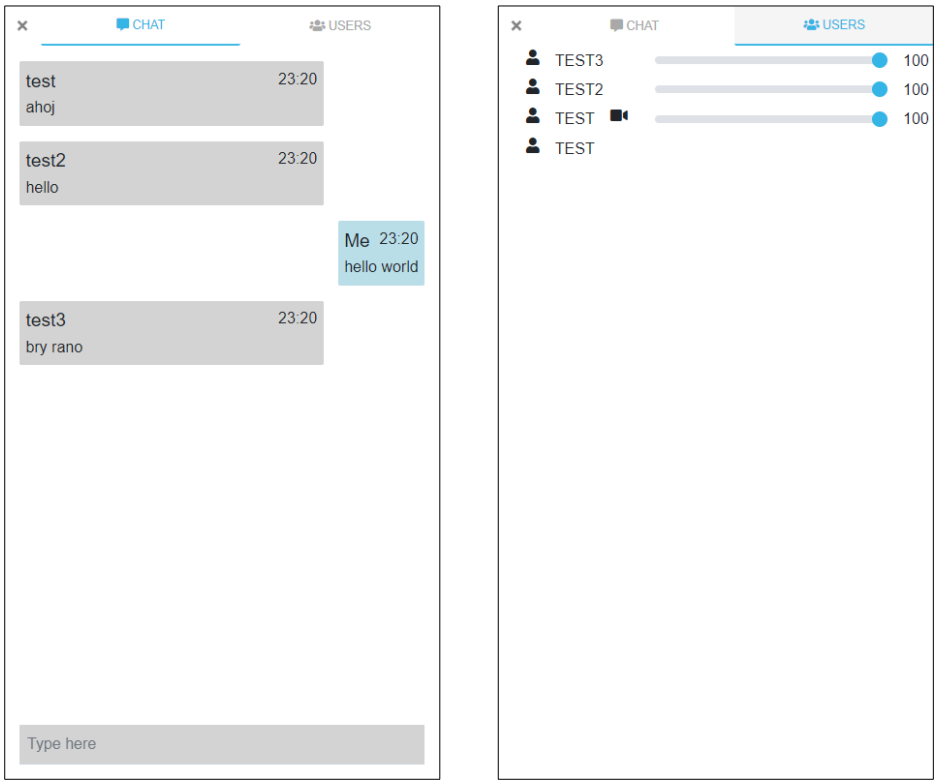
Obrázek B.2: Obrazovka s probíhajícím hovorem



(a) Modální okno pro sdílení webkamery

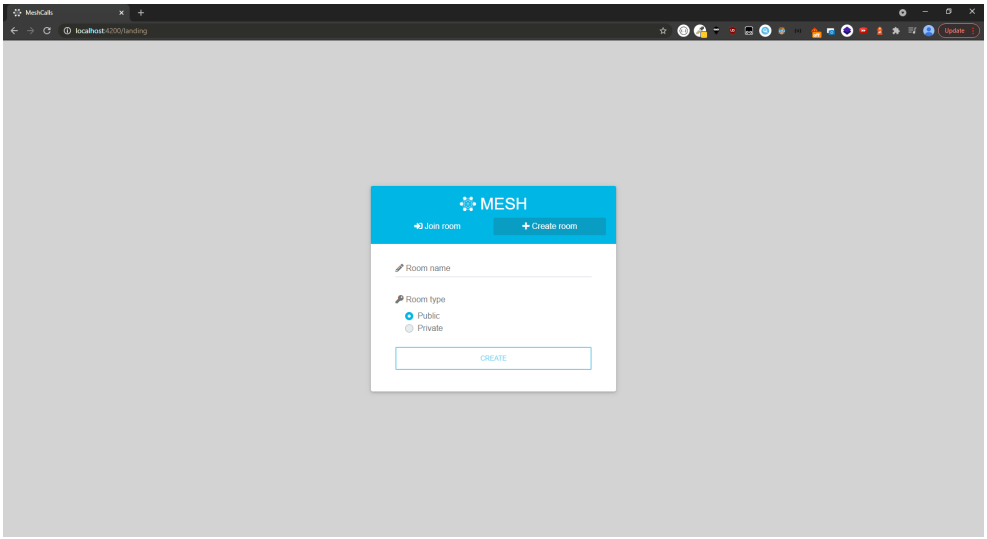
(b) Modální okno pro sdílení odkazu místnosti

Obrázek B.3: Obrazovky s modálními okny

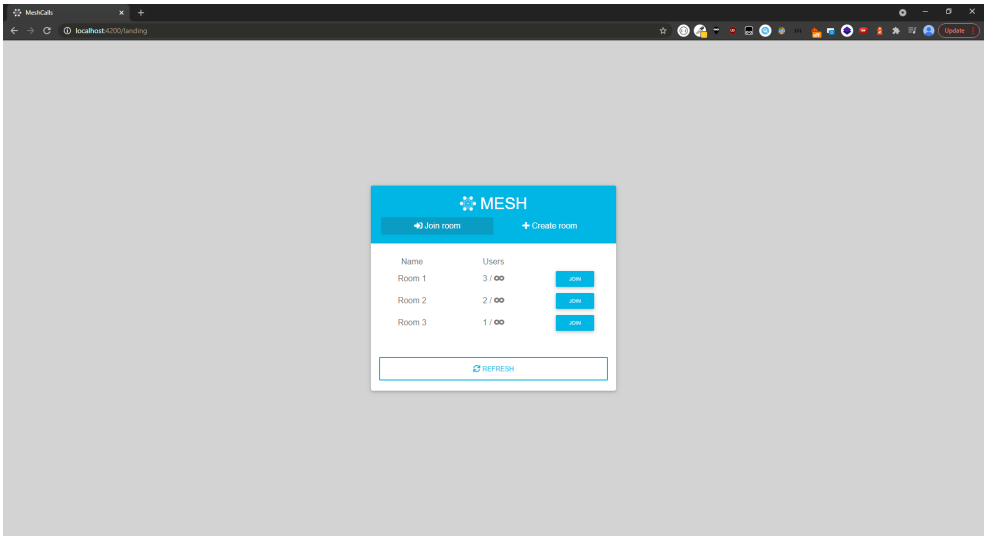


(a) Chat (b) Seznam uživatelů

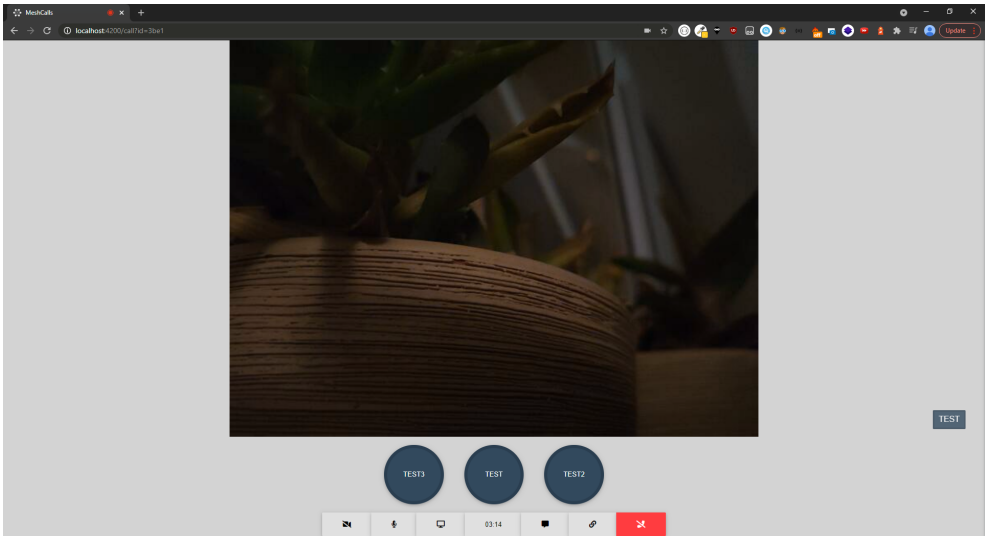
Obrázek B.4: Vysouvací panel



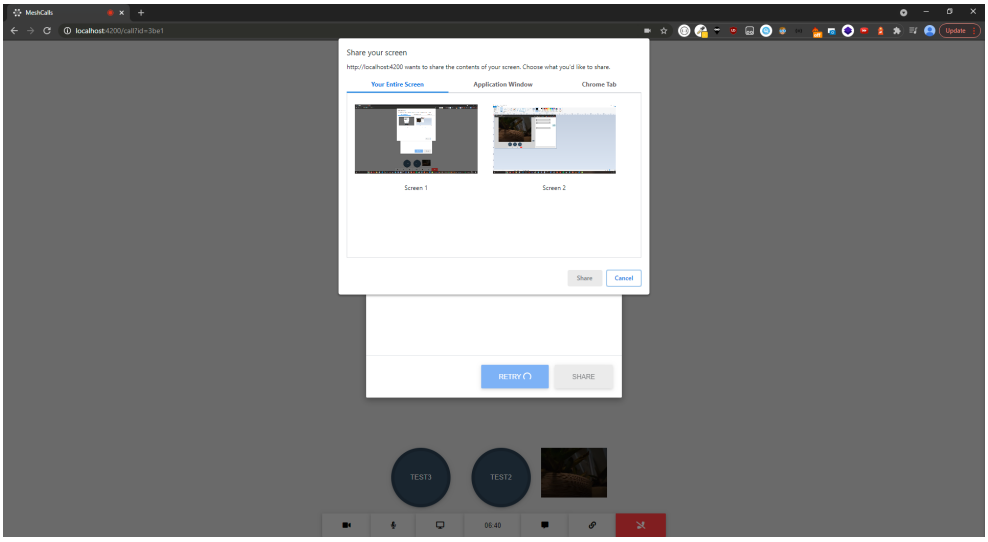
Obrázek B.5: Seznam místností



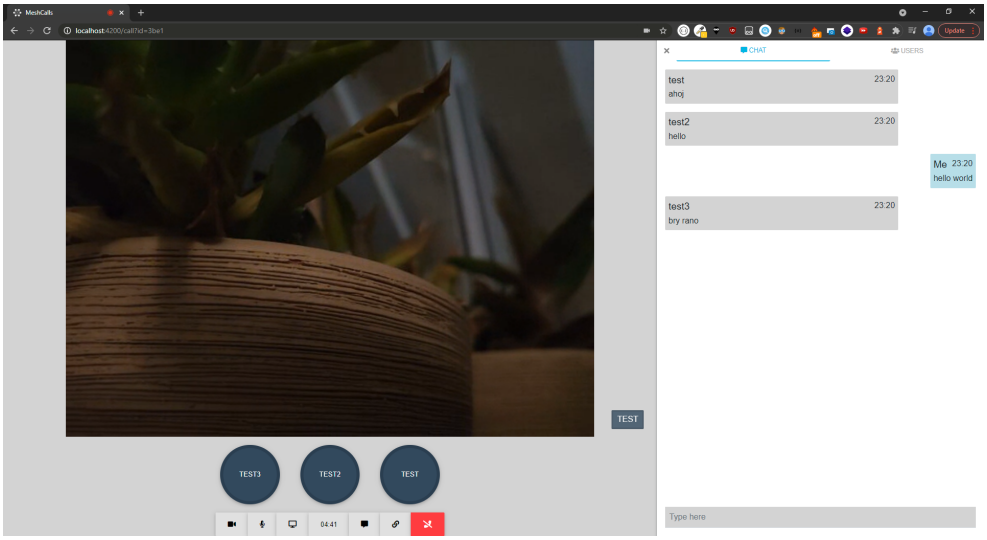
Obrázek B.6: Vytvoření nové místnosti



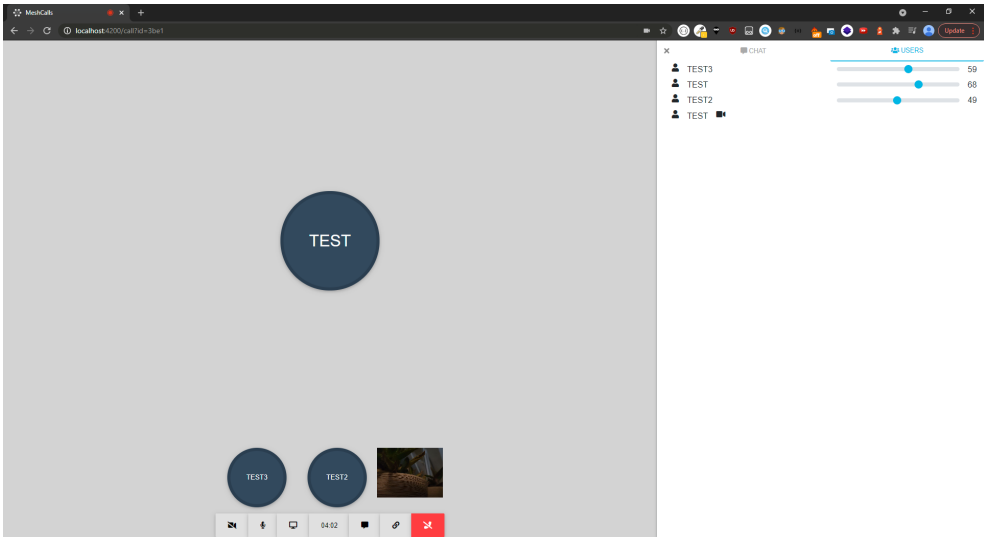
Obrázek B.7: Hovor



Obrázek B.8: Modální okno pro sdílení obrazovky



Obrázek B.9: Postranní panel s chatem



Obrázek B.10: Postranní panel se seznamem uživatelů

Příloha C

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `docs`
  - `client` ..... dokumentace klientské části.
  - `server` ..... dokumentace serverové části.
- `src`
  - `client` ..... zdrojové kódy klientské části
  - `server` ..... zdrojové kódy serverové části
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF