Zpráva o realizaci

Dosažená funkcionalita

- Obousměrná komunikace minimálně 3 klientů současně, a to ve video i audiu.
- Samostatné serverové řešení nezávislé na dalších externích službách v Node.js.
- Webový klient distribuovaný přes server v Node.js nezávislý na dalších externích službách s GUI odpovídajícím celkové funkcionalitě řešení.
- Přístup do komunikace po zvolené formě autorizace.
- Přenos souborů.
- Textová online komunikace.
- Sdílení obrazovky zvoleného klienta.
- Možnost komunikace v samostatných místnostech.

Nedosažená funkcionalita

- Šifrování komunikace.
- Možnost nahrávání komunikace.

konceptuální popis řešení a způsob implementace

Řešení obsahuje klienta, node server a mongodb server. Klient je distribuovaný z node serveru. Oba servery běží ve vlastním docker kontejneru.

Autentizace uživatelů je na bázi přístupového tokenu, který je vrácen serverem po zadání správných přístupových údajů.

Komunikace mezi serverem a klientem je zajištěna pomocí knihovny SocketIO, čímž je zajištěna spolehlivost (automatické znovu připojení), obousměrná komunikace (možnost být WebRTC signal server díky pushování do klienta, realtime app možnosti, možnost request-response modelu), broadcasting... Server má 1 socket rozdělený do 3 prostorů "namespace" (tzn. veškerá komunikace jde reálně přes jeden socket a je multiplexována do jednotlivých prostorů): pro anonymní uživatele, pro přihlášené uživatele a pro členy místnosti. Prostor pro přihlášené uživatele pomocí middlewaru zajišťuje, že jsou všechny jeho eventy dostupné pouze pro autentizované uživatele. Stejný princip platí u socketu pro členy místnosti, kde se navíc řeší členství v místnosti.

Schůzka (videohovor) je zajištěna pomocí full-mesh (každý s každým) peer-to-peer WebRTC spojení s použitím "perfect negotiation" vzoru (https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API/Perfect_negotiation). Ten je potřeba v případě, že se nahrazují, odebírají či přidávají tracky do streamu při probíhajícím spojení (screen share), v takovém případě totiž musí proběhnout znovuvyjednání spojení "renegotiation".

Routování na serveru zajišťuje Express. Všechny routy začínající "/public/" jsou servírovány ze serveru, složky public (např. přílohy chatu). Všechny routy "/socket.io/" jsou předány ke zpracování SocketIO. Všechny ostatní routy se server snaží najít v buildu klienta. Všechno ostatní, i neexistující, je nasměrováno do index.html buildu klienta, neboť server nezná klientské routy.

Mongodb databáze slouží k uchování dat jako je historie chatu, vytvořené místnosti, uživatelé, ...

Hlavní technologie na klientovi je Svelte. Jedná se o jednoduchý, reaktivní, kompilovaný frontend framework.

Použité knihovny

- Express: Nodejs server framework
- SocketIO: Knihovna pro realtime komunikaci mezi serverem a klientem
- Typescript: Typovaný javascript
- Svelte: Reaktivní frontend framework
- Mongodb: MongoDB driver
- Yup: Validační knihovna
- Svelte-navigator: Frontend router
- Sveltik: Knihovna pro snadnější práci s formuláři
- Vite: Buildovací nástroj
- uuid: knihovna pro generování náhodných unikátních identifikátorů

Shrnutí dosažených výsledků včetně kladů a záporů vytvořeného řešení.

Podařilo se dosáhnout minimální požadované funkcionality a většiny variantní funkcionality.

Klady:

- Typescript, stejný jazyk na serveru i klientu: Klient a server využívají typescript. To usnadňuje implementaci klientu. Typescript klient si může server přidat do závislostí a využít typy serveru. Ví tak, jaká pole jsou od něj v jeho požadavcích vyžadována a jakého jsou typu. To samé platí pro odpovědi ze serveru, klient ví co mu přijde. Vznikla tak i jakási api dokumentace. Při případné změně typů na serveru, např. přidání povinného pole do požadavku, bude klient hlásit errory a bude tak snadnější odhalit, kde je problém. Také jsou v některých případech sdílené validace mezi klientem a serverem - to dává jistotu, že pokud projde formulář validací na klientu, bude dobře zvalidován i na serveru. Nebo při případné změně validace, např. požadování komplexnějšího hesla, se změna po aktualizaci závislosti projeví i na klientovi.
- Docker: snadné zapojení dalších lidí do projektu.
- Validace formulářů, stavové hlášky.
- Error handling, omezené množství míst, kde by byla chyba ignorována či nezobrazena.
- Responsivní design.
- Historie chatu.

Zápory:

- Full-mesh spojení je síťově a procesorově náročné a omezuje počet účastníků schůzky na max jednotky účastníků. K vylepšení by bylo potřeba přejít na SFU, nebo MCU architekturu.
- SocketIO knihovna na serveru omezuje klienty na technoligie, pro které je dostupný SocketIO client.
- Neotestováno komplexně (různé sítě, různá zařízení, různé prohlížeče, zátěžové testy, unit testy, e2e testy...), budou přítomny bugy.
- Nekompletní funkcionalita typu mazání místností, odebírání členů místností, autorizace členů místností, identifikace účastníků.
- Architekturu serveru by bylo dobré vylepšit, přidat DAL vrstvu, docílit nezávislost na uložišti dat.
- Bezpečnost tokenů, ukládat hashované tokeny, časová expirace + nebyla dosažena šifrovaná komunikace - možnost odchycení tokenu. Nebo vyřešit implementací hotového řešení jako je JWT.
- Pomalá detekce odpojení účastníka schůzky.