Návrh a tvorba online komunikační skupiny pro peer terapii

Bc. Jiří Šrytr

Diplomová práce
2024

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav informatiky a umělé inteligence

Akademický rok: 2023/2024

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Jiří Šrytr**
Osobní číslo:	**A22315**
Studijní program:	**N0613A140022 Informační technologie**
Specializace:	**Softwarové inženýrství**
Forma studia:	**Prezenční**
Téma práce:	**Návrh a tvorba online komunikační skupiny pro peer terapii**
Téma práce anglicky:	**Design and Creation of an Online Communication Group for Peer Therapy**

Zásady pro vypracování

1. V teoretické části nastudujte různé kryptografické protokoly poskytující end-to-end šifrování zpráv a vyberte vhodný protokol k implementaci aplikace.
2. Dále v rámci teoretické části prostudujte a popište technologie použité k implementaci aplikace.
3. Formulujte funkční a nefunkční požadavky na online komunikační skupinu pro peer terapii.
4. Navrhněte informační architekturu aplikace, popište datový model a uživatelské scénáře.
5. Na základě návrhu aplikaci prakticky implementujte a popište její strukturu a představte důležité části implementace, především pak řešení zabezpečené komunikace.

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam doporučené literatury:

1. OPPLIGER, Rolf. *End-to-end Encrypted Messaging*. Artech House, 2020. ISBN 978-1630817329.
2. LOMBARDI, Andrew. *WebSocket: Lightweight Client-Server Communications*. O'Reilly Media, 2015. ISBN 978--1449369279.
3. SPRINGER, Sebastian. *Node.js: The Comprehensive Guide to Server-Side JavaScript Programming*. Rheinwerk Computing, 2022. ISBN 978-1493222926.
4. ROLDÁN, Carlos Santana. *React 18 Design Patterns and Best Practices: Design, build, and deploy production-ready web applications with React by leveraging industry-best practices*. 4th ed. Birmingham, United Kingdom: Packt Publishing, 2023. ISBN 978-1803233109.
5. React Reference Overview – React. *React* [online]. [cit. 2023-11-08]. Dostupné z: https://react.dev/reference/react
6. Documentation | Node.js. *Node.js* [online]. c2023 [cit. 2023-11-08]. Dostupné z: https://nodejs.org/en/docs/
7. MongoDB Documentation. *MongoDB: The Developer Data Platform | MongoDB* [online]. c2023 [cit. 2023-11-08]. Dostupné z: https://docs.mongodb.com/

Vedoucí diplomové práce: **Ing. Radek Vala, Ph.D.**
Ústav informatiky a umělé inteligence

Datum zadání diplomové práce: **5. listopadu 2023**
Termín odevzdání diplomové práce: **13. května 2024**

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
DĚKAN
L.S.

doc. Ing. Jiří Vojtěšek, Ph.D. v.r.
děkan

prof. Mgr. Roman Jašek, Ph.D., DBA v.r.
ředitel ústavu

Ve Zlíně dne 5. ledna 2024

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně;
- byl/a jsem seznámen/a s tím, že na moji diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování diplomové práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne 10. 5. 2024

Jiří Šrytr, v.r.
podpis studenta

ABSTRAKT

Cílem této diplomové práce je navrhnout a vytvořit webovou aplikaci pro online peer terapii určenou především lidem trpícím syndromem vyhoření. Primárním cílem je propojit určitý počet lidí právě s jedním terapeutem, který celou skupinu povede. V rámci aplikace bude pomocí dotazníků monitorován stav uživatelů, jejích případné zlepšení či zhoršení a také jejich zpětná vazba. V teoretické části budou rozebrány technologie použité k vývoji a zabezpečení aplikace. Praktická část bude zaměřena na samotný návrh a implementaci aplikace. Budou zde popsány funkcionální a nefunkcionální požadavky, návrh informační architektury a drátěné modely, dále bude rozebrán datový model, diagram případů užití a navazující uživatelské scénáře. Následně zde bude popsána struktura samotné aplikace a důležité části implementace. Text se také bude věnovat zabezpečení aplikace a bezpečnosti dat.

Klíčová slova: webová aplikace, webové technologie, instant messaging, End-to-End, Node.js, React, WebSocket

ABSTRACT

The goal of this diploma thesis is to design and create a web application for online peer therapy intended primarily for people suffering from burnout syndrome. The primary goal is to connect a certain number of people with just one therapist who will lead the entire group. Within the application, questionnaires will be used to monitor the status of users, their possible improvement or deterioration, as well as their feedback. In the theoretical part, the technologies used to develop and secure the application will be discussed. The practical part will focus on the actual design and implementation of the application. Functional and non-functional requirements, information architecture design and wireframe models will be described here, and the data model, use case diagram and subsequent user scenarios will be analyzed. Subsequently, the structure of the application itself and important parts of the implementation will be described here. The text will also address application security and data security.

Keywords: web application, web technologies, instant messaging, End-to-End, Node.js, React, WebSocket

Tímto bych chtěl vyjádřit upřímné díky Ing. Radku Valovi, Ph.D. za jeho odborné vedení a cenné rady. Rovněž bych chtěl poděkovat organizaci Nevyhasni, neboť bez ní by realizace tohoto projektu nebyla možná.

Prohlašuji, že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

OBSAH

ÚVOD

Problém syndromu vyhoření je všude na světě, nevyjímaje Českou republiku, stále častější a lidí s tímto problémem neustále přibývá. Aby bylo možné těmto problémům předcházet a efektivně je řešit, je zásadní rozvíjet nové přístupy a zdroje podpory. Proto vznikla organizace Nevyhasni, jejíž cílem je za pomoci odborníků z řad vědců, psychologů či lékařů šířit osvětu o syndromu vyhoření, přinášet do společnosti funkční systémy prevence syndromu vyhoření v podobě přednášek a workshopů a vytvořit online platformu, která bude bezpečným místem pro všechny, kterých se tento problém týká. V rámci Nevyhasni vzniká i online komunikační skupina pro peer terapii, která bude výstupem této diplomové práce.

Cílem této práce je navrhnout a vytvořit webovou aplikaci, která bude sloužit jako účinný pomocník lidem, kteří trpí syndromem vyhoření. Hlavní myšlenkou této aplikace je poskytnout bezpečný a pohodlný prostor, kde budou moci lidé sdílet svoje zkušenosti a navzájem se podporovat pod dohledem profesionálního terapeuta.

V teoretické části se práce z počátku zaměří na problematiku bezpečnosti instant messaging aplikací, rozebere základní kryptografické principy, představí možnosti koncového šifrování, popíše existující kryptografické protokoly, které poskytují koncové šifrování a na základě stanovených kritérií vybere jeden z těchto kryptografických protokolů, který bude využit v implementaci aplikace. Poté budou popsány jednotlivé technologie, na kterých bude výsledná aplikace postavena.

Praktická část se bude věnovat návrhu a následné implementaci online komunikační skupiny pro peer terapii. Nejdříve bude vyhotoveno základní zadání projektu, které poslouží jako základní stavební kámen aplikace a od kterého se bude odvíjet následný sběr a dokumentace funkcionálních i nefunkcionálních požadavků. V další části již bude probíhat návrh aplikace, který začne návrhem informační architektury a drátěného modelu webové aplikace. Dále bude pokračovat návrhem datového modelu a diagramu případů užití od kterého se posléze budou odvíjet zdokumentované uživatelské scénáře.

Praktická část bude dále pokračovat popisem struktury klientské a serverové části implementované aplikace včetně struktury databáze, poté vysvětlením implementace vybraného kryptografického protokolu s ukázky kódu a nakonec předvedením a popisem důležitých částí výsledné aplikace.

I. TEORETICKÁ ČÁST

1 ZABEZPEČENÍ IM APLIKACÍ

V digitálním věku, kdy se naše komunikace stále více přesouvá do online prostoru, se otázky bezpečnosti stávají neoddělitelnou součástí našich životů. Lidé si začínají uvědomovat, jak moc je jejich soukromí drahocenné a jaká jsou rizika v případě jeho narušení třetí stranou. I právě díky tomuto uvědomění se velké technologické společnosti začínají zaměřovat na implementaci bezpečných opatření, která zajistí soukromou a bezpečnou komunikaci.

Jedním z klíčových prvků, který přináší zvýšenou úroveň bezpečnosti do digitální komunikace, je end-to-end šifrování zpráv. Tato technologie se stala standardem v některých z nejvýznamnějších instant messaging (IM) aplikacích, jako jsou Telegram, WhatsApp, Wire a Signal. End-to-end šifrování zpráv vytváří prostor pro důvěrnou konverzaci mezi odesílatelem a příjemcem a eliminující možnost neautorizovaného přístupu k obsahu. Tento způsob zabezpečení komunikace bude dále v rámci této kapitole rozebrán.

1.1 Základy kryptografie

Kryptografie se zabývá technikami utajování obsahu zpráv převodem do formy, která je bez konkrétní znalosti nečitelná. Primárním cílem je zabránit neoprávněnému přístupu k informacím, tedy zajistit, aby zprávy pochopila a zpracovala pouze osoba, pro kterou jsou určeny. Kryptografie se opírá o matematiku a teoretickou informatiku a zahrnuje mnoho technik a metod, včetně šifrování, digitálních podpisů, hashovacích funkcí a protokolů pro zabezpečenou komunikaci. [13]

Šifrováním označujeme algoritmus, který převádí čitelnou zprávu, označovanou jako otevřený text, do podobny nečitelné pro neoprávněné uživatele neboli na šifrovaný text. Algoritmus, který pak převádí šifrovaný text zpět do čitelné podoby je označován jako algoritmus dešifrovací. [13]

Typickým příkladem šifer používaných v minulosti jsou tzv. klasické (konvenční) šifry, mezi které se řadí například substituční a transpoziční šifry. Substituční šifra nahrazuje abecedu otevřeného textu za jinou abecedu šifrovaného textu. Platí pravidlo, že pořadí znaků zůstává stejné, tedy znak, který je na prvním místě v otevřeném textu je na prvním místě i v šifrovaném textu. Příkladem jednoduché substituční šifry může být například dobře známá Caesarova šifra. Oproti substitučním šifrám pak transpoziční šifry mění pořadí znaků v textu a abeceda otevřeného a šifrovaného textu zůstává stejná. [49] Substituční a transpoziční šifry však nejsou odolné vůči relativně jednoduchým typům prolomení, například proti frekvenční

analýze, tedy analýze četnosti znaků. V dnešní době jsou tyto metody nahrazeny moderními symetrickými a asymetrickými šiframi, které nabízejí výrazně vyšší úroveň bezpečnosti a jsou schopny odolávat různým formám kryptoanalýzy. [50]

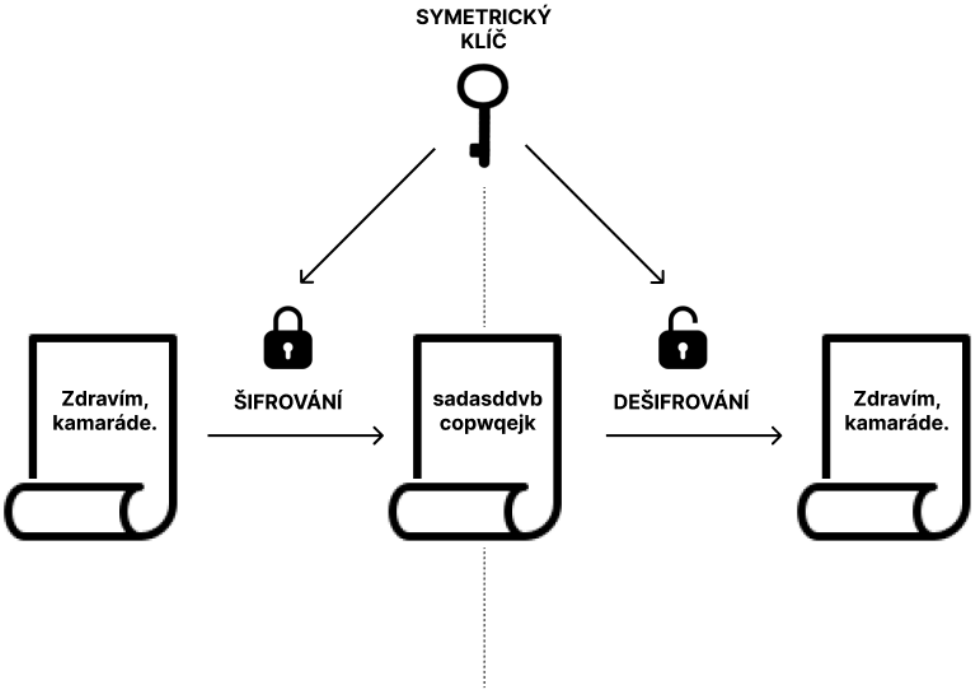
Při šifrovacím a dešifrovacím procesu je využito tzv. kryptografických klíčů, které musí být pečlivě chráněny. Vzhledem k tomu, že není možné dlouhodobě udržet dešifrovací algoritmus v tajnosti a většinou je veřejně známý, klíčovým prvkem, který zajišťuje bezpečnost, je samotný šifrovací klíč. [14]

Kryptografie nese tedy tyto důležité vlastnosti [13]:

- **Důvěrnost:** Informace jsou přístupné pouze tomu, komu jsou určeny, a žádná jiná osoba k nim nemá přístup.
- **Integrita:** Než informace dorazí od odesílatele k určenému příjemci, nelze ji bez povšimnutí měnit žádnou třetí stranou.
- **Autentizace:** Prokázání identity odesílatele a příjemce či původu a místa určení informací.
- **Nepopiratelnost:** Příjemce a ani odesílatel nemohou později popřít dřívější provedení nějaké operace.

1.1.1 Symetrická kryptografie

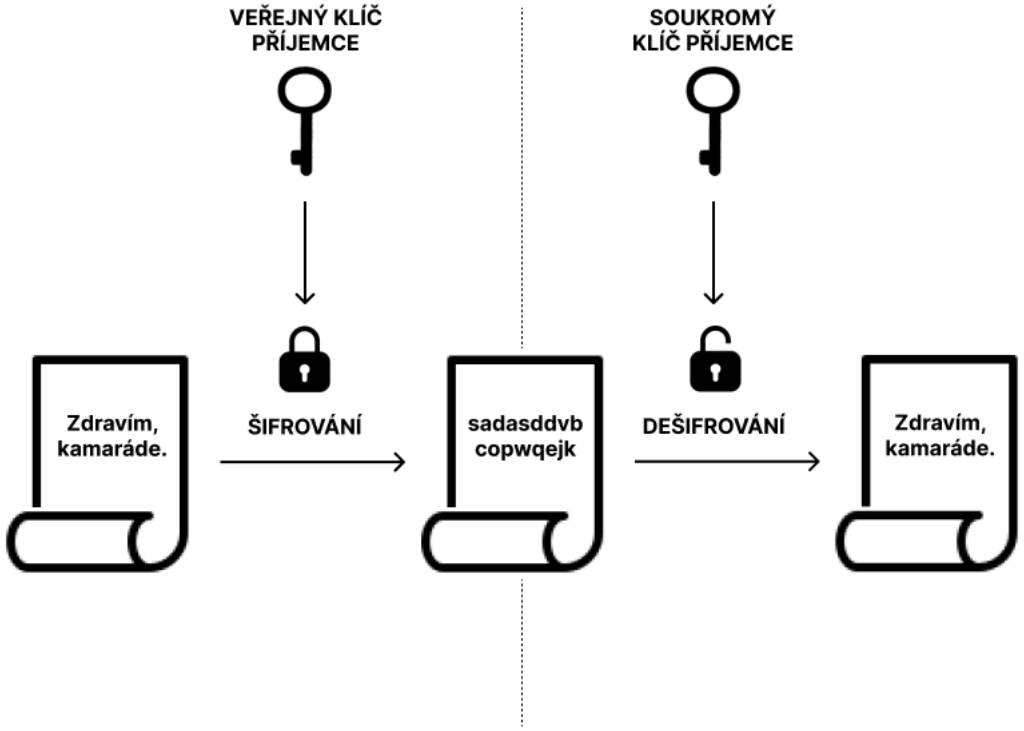
Symetrická kryptografie představuje systém, kde je využit jeden klíč pro obě operace, tedy jak pro šifrování, tak pro dešifrování. Mezi výhody symetrické kryptografie patří jednoduchost a rychlost šifrovacích a dešifrovacích algoritmů. Problémem symetrické kryptografie pak je nutnost sdílení a přenosu klíče mezi příjemcem a odesílatelem. Mezi nejznámější symetrické kryptografické systémy se řadí například Data Encryption System (DES) a Advanced Encryption System (AES). [13]



Obrázek 1 Znázornění symetrické kryptografie (vlastní obrázek)

1.1.2 Asymetrická kryptografie

Asymetrický kryptografický systém k procesu šifrování a dešifrování využívá páru klíčů, klíče veřejného a soukromého. Veřejný klíč se využívá k zašifrování zprávy a soukromý klíč k jejímu dešifrování. Tedy zašifrovat zprávu příjemci je umožněno každému účastníkovi, ale zpětně ji dešifrovat může pouze samotný příjemce. [13] Oproti symetrické kryptografii spočívá hlavní výhoda asymetrické kryptografie v tom, že již není nutné sdílet klíč mezi účastníky komunikace. Naopak nevýhodou jsou vysoké výpočetní nároky a s tím spjatá rychlost. Například RSA je až 1000x pomalejší než AES. U asymetrické kryptografie je také nutné ověřit autenticitu veřejného klíče, tedy jestli skutečně pochází od očekávaného majitele. Pokud by k tomuto ověření nedošlo komunikace by mohla být kompromitována útokem man-in-the-middle, kdy útočník bez povšimnutí vymění účastníkům komunikace jejich veřejné klíče za jeho veřejný klíč. [15]



Obrázek 2 Znázornění asymetrické kryptografie (vlastní obrázek)

1.2 Cíle a výzvy zabezpečení komunikace v IM aplikacích

V této podkapitole budou obecně popsány cíle a výzvy zabezpečení v IM aplikacích, které je nutné mít při vývoji aplikace na paměti. Zabezpečení IM aplikací má několik cílů, kdy základní z nich byly již v této práci zmíněny v rámci popisu vlastností kryptografie – jedná se o **důvěrnost**, **integritu**, **autentizaci** a **nepopiratelnost**. Zde je ale třeba brát v potaz i další cíle [17]:

- **Perfect Forward Secrecy:** Říká, že když je dlouhodobý klíč, který je použit k šifrování zpráv, kompromitován, žádnou z předešlé komunikace nelze dešifrovat.
- **Perfect Backward Secrecy (někdy též Future Secrecy):** Když je dlouhodobý klíč kompromitován, žádná z budoucí komunikace nelze dešifrovat.
- **End-to-End šifrování:** Značí, že zprávy by měly být zašifrovány mezi dvěma (či vícero) klienty, nikoliv však mezi klientem a serverem.

Aby byly možné splnit všechny tyto cíle, nebo alespoň většinu z nich, a zároveň zachovat dobrou uživatelskou zkušenost (UX), musí vývojáři čelit následujícím výzvám [17]:

- **Trust establishment**: Jedná se o problém autentičnosti, tedy vývojáři musí zaručit, že uživatelé komunikují pouze s tím, s kým opravdu chtějí komunikovat. Kvůli zachování dobré uživatelské zkušenosti je také potřeba zajistit, aby uživatelé nemuseli vynaložit téměř žádné úsilí v rámci managementu klíčů.
- **Conversation security:** Týká se problému bezpečnosti a soukromí zpráv. Vývojáři musí řešit, jak jsou data a metadata šifrována. Opět by neměla být požadována žádná interakce od uživatel navíc k zajištění bezpečnosti konverzace.
- **Transport privacy:** Definuje, jak jsou zprávy vyměňovány a řeší primárně problém schovávání metadat. Navíc také řeší stále aktuální problém – jak může aplikace vyhledat kontakty a zároveň zachovat soukromí uživatele a jeho kontaktů.

1.3 E2EE šifrování

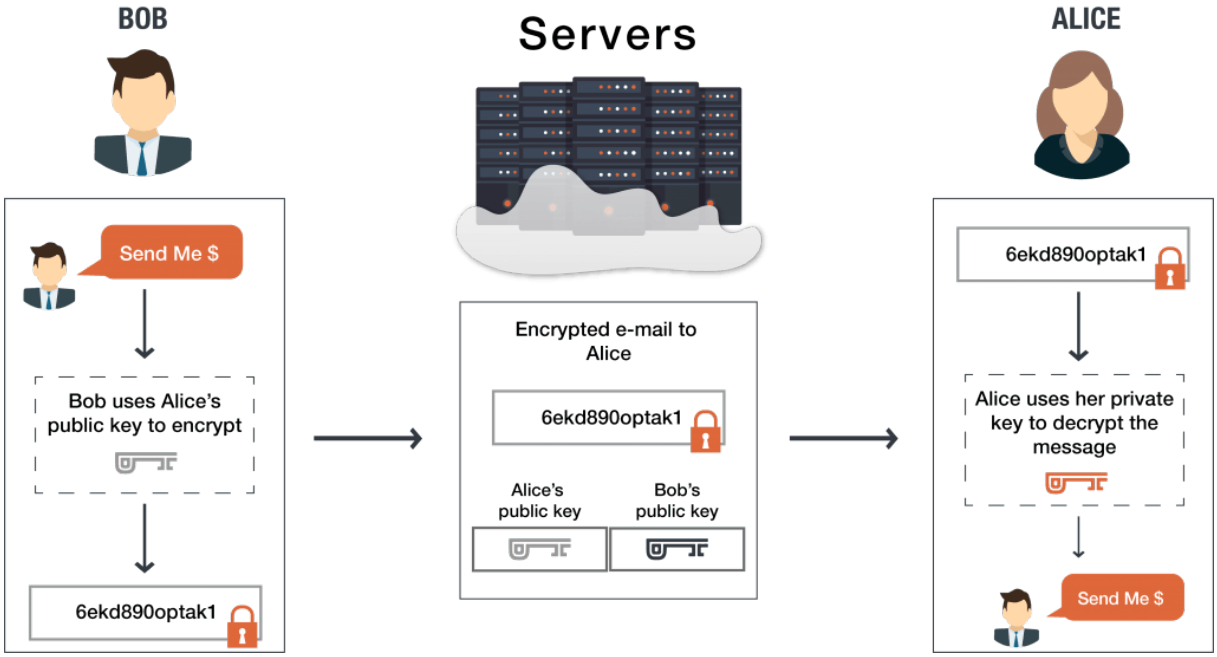
End-to-end šifrování, zkráceně E2EE, je komunikační systém, který chrání data odesílaná mezi koncovými body, v případě chatovací aplikace tedy mezi jednotlivými účastníky konverzace. Nikdo, včetně poskytovatele komunikačního systému, poskytovatelů telekomunikačních služeb, poskytovatelů internetu nebo zlomyslných aktérů, nemá přístup k odeslaným zprávám. [1]

E2EE tedy zajišťuje, aby data nemohla být přečtena a ani modifikována nikým jiným než reálným odesílatelem a příjemcem (případně vícero příjemci). Zprávy jsou před odesláním zašifrovány odesílatelem s tím, že poskytovatel aplikace je nemá možnost žádným způsobem dešifrovat a ukládá je v zašifrované podobě. Příjemci poté získávají zprávu zašifrovanou a dešifrují si ji sami. [1]

Při E2EE dochází k šifrování na úrovni klienta, tedy zařízení uživatele, který odesílá zprávu. Zprávy a soubory jsou zašifrovány pomocí veřejného klíče ještě před tím, než opustí telefon nebo počítač uživatele. Veřejný klíč je dostupný všem, avšak data jsou dešifrovatelná pouze soukromým klíčem, který je uložen v zařízení konkrétního uživatele a je dostupný pouze příjemcům zpráv. Díky tomuto faktu, i v případě, že se útočník dostane na server a získá přístup k datům, stále není schopný tato data číst, protože nemá soukromé klíče potřebné k jejich dešifrování. Tento proces, kdy dochází k vytváření páru soukromého a veřejného klíče se označuje jako asymetrická kryptografie. [2]

Pro lepší pochopení problematiky bude uveden typický příklad „Bob a Alice“. Dva uživatelé, tedy Bob a Alice, si vytvoří uživatelský účet v chatovací aplikaci, která využívá

E2EE pro zabezpečení dat. Systém poté každému z nich přiřadí veřejný a soukromý klíč, přičemž veřejný klíč je uložen na serveru a jejich soukromé klíče jsou, jak již bylo zmíněno, uloženy na jejich zařízeních. V případě, že Alice chce zaslat zprávu Bobovi, k zašifrování použije Bobův veřejný klíč. Jakmile Bob přijme tuto zprávu, použije svůj soukromý klíč k dešifrování. Kdyby poté chtěl Bob Alici odpovědět, opakoval by totožný proces, s tím, že by zašifroval zprávu pomocí veřejného klíče Alice. [2]



Obrázek 3 Princip E2EE na příkladu Bob a Alice [2]

1.3.1 Význam a výhody E2EE

Jak již bylo nastíněno výše v této kapitole, E2EE má v dnešní době digitálních technologií obrovský význam a disponuje řadou výhod. Níže budou popsány nejzásadnější z nich:

- **Zajišťuje, že jsou data v bezpečí před zlomyslnými aktéry** – díky E2EE je uživatel jediný, kdo má soukromý klíč potřebný k dešifrování zpráv. Data jsou v bezpečí i v případě, že byl server prolomen útočníkem. [2] Toto není výhodou jen pro koncové uživatele, ale také pro poskytovatele softwarových služeb. Například v roce 2013 došlo k enormnímu úniku dat služby Yahoo, kdy útočník získal data všech uživatelů, tedy přibližně tří miliard uživatelských účtů. Nejenže tento útok poškodil reputaci společnosti mezi uživateli, ale také uškodil jejich vyjednávací pravomoci

s dalšími velkými společnostmi. V důsledku tohoto útoku muselo Yahoo snížit dohodu se společností Verizon o minimálně 350 miliónů amerických dolarů. [4]

- **Chrání soukromí uživatelů** – velká část poskytovatelů služeb, jako je například Google nebo Microsoft, dešifruje data na jejich serverech, díky čemu jsou schopni tato data číst. V případě E2EE, má k dešifrovaným datům pouze držitel příslušného soukromého klíče. [2]
- **Zajišťuje integritu dat –** přenášená data jsou v ochráněna před neoprávněnou manipulací a modifikací. Tedy je zajištěno, že data zůstanou nezměněna a určený příjemce jim může důvěřovat. [2]

1.3.2 Výzvy a omezení E2EE

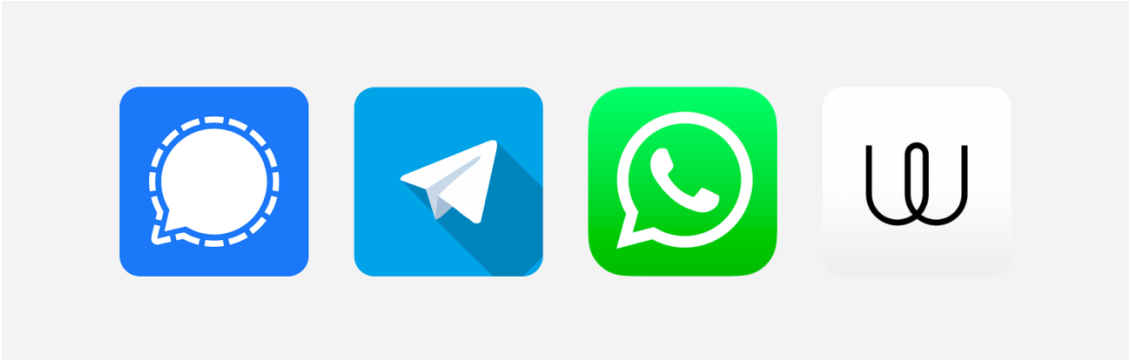
Přestože šifrování typu end-to-end nabízí společnostem a uživatelům mnoho cenných výhod, bezpečnostní praxe stále trpí několika nedostatky, na které je potřeba při implementaci pamatovat.

- **Neskryje skutečnost, že jsou data přenášena** – přestože E2EE zajistí, aby data byla bez soukromého klíče nečitelná, nezakryje fakt, že dochází k jejich přenosu. Takže je stále možné najít záznamy o transakcích, které obsahují datum odeslání, odesílatele, příjemce atp. To může útočníkovi poskytnout užitečné informace. [5]
- **Příliš mnoho soukromí** – ačkoliv se tento fakt může zprvu jevit pouze jako výhoda vlády a orgány činné v trestním řízení mají obavy, že E2EE může chránit také lidi sdílející nezákonný obsah. Poskytovatelé pak nejsou v případě vyšetřování schopni poskytnout orgánům k tomuto obsahu přístup. [5]
- **Nebezpečí u koncového zařízení** – pokud se na koncovém zařízení vyskytne problém se zabezpečením nebo se zařízení dostane do nesprávných rukou, dešifrovaná data ze zařízení mohou být útočníkem odcizena. [4]
- **Nemožnost přepínaní zařízení bez narušení bezpečnosti –** uživatelé často chtějí přepínat mezi různými zařízeními a ke svým účtům se přihlašovat z vícero zařízení. Kromě toho se může stát, že se zařízení ztratí, rozbije se či jinak znehodnotí. Soukromý klíč uživatele je uložen a generován přímo na konkrétním zařízení a jeho přesunutí do zařízení jiného může narušit celkovou bezpečnost. [7]

1.4 E2EE v praxi

Poptávka po soukromí v digitálním prostoru roste. Lidé se uvědomují, že pokud používají nešifrovanou aplikaci pro zasílání zpráv, jejich komunikace by mohla být vystavena neoprávněným očím jako jsou nejen zlomyslní aktéři, ale i inzerenti reklam a samotné společnosti poskytující konkrétní softwarovou službu. Nemluvě o tom, co by se mohlo stát v případě úniku dat. Soukromé informace uživatel mohou být prodány online, použity ke krádeži identity a jiným kyberzločinům.

Velká spousta technologických společností mezi jejichž produkty patří instant messaging aplikace se v posledních letech snažila vyhovět poptávce uživatel po zabezpečeném a soukromém prostoru pro komunikaci. Nejbezpečnější z těchto aplikací pak využívají šifrování typu E2EE, aby zajistili, že korespondenci uvidí opravdu pouze odesílatel a příjemce zprávy, nikdo z výše zmiňovaných aktérů. Mnoho z nejlépe šifrovaných IM aplikací je tak silně šifrováno, že se do nich nedokážou dostat ani vládní složky či policie, což je ve společnosti vytváří značné dilema, protože ochrana soukromí uživatel znamená i ochranu uživatel vykonávajících trestnou činnost.



Obrázek 4 Známé aplikace s E2EE (v pořadí zleva Signal, Telegram, WhatsApp, Wire)

1.4.1 Signal

Signal je považován za jednu z nejbezpečnějších IM aplikací. Veškeré zprávy jsou zabezpečeny pomocí E2EE, navíc Signal je nezisková organizace, takže nemá důvod sbírat uživatelská data pro osobní zisk. Zabezpečení aplikace Signal je postaveno na open-source kryptografické sadě specifikací označované jako Signal Protocol. Důvěra v Signal Protocol je podpořena faktem, že jej využívá nejen Signal, ale také další velké IM aplikace jako je WhatsApp či Facebook Messenger. Aplikace Signal disponuje kromě textových zpráv i

zabezpečenými hlasovými a video hovory. Aplikace je dostupná jak pro mobilní zařízení (Android a iOS), tak i pro desktop (Windows, MacOS a Linux). [9]

Signal shromažďuje jen minimální metadata, která jsou nutná pouze k efektivnímu poskytování jeho služeb. Jedná se o čas vytvoření uživatelského účtu a čas posledního připojení uživatele k serveru. [12]

1.4.2 Telegram

Telegram je velmi populární aplikace, která kromě možnosti klasické chat konverzace, nabízí i mnoho dalších funkcí. S Telegramem mohou uživatelé kromě psaní zpráv také vytvářet různé komunity, psát blogové příspěvky, pořádat živé vysílání atd. Na úkor těchto funkcí má ale menší záruku zabezpečení než jiné aplikace. Pomocí E2EE jsou zabezpečeny pouze speciální konverzace a E2EE tedy není nastaveno defaultně. Pro zabezpečení využívá Telegram vlastního řešení nazvaného jako MTProto, které je stejně jako Signal Protocol open-source. [9]

Co se týče shromažďování metadat, Telegram je v této otázce poměrně kontroverzní. Zásady ochrany osobních údajů společnosti Telegram uvádí, že v rámci předcházení spamu a případného zneužití shromažďují informace, jako jsou IP adresy, podrobnosti o zařízení, historie změn uživatelských jmen a další. Data jsou pak uložena po dobu 12 měsíců. [12]

1.4.3 WhatsApp

WhatsApp patří mezi nejpoužívanější aplikace pro zasílání textových zpráv a volání na světě. Od roku 2016 WhatsApp používá Signal Protocol pro implementaci E2EE a defaultně šifruje veškeré zprávy. Její vývojáři neustále přidávají nové funkce pro lepší zabezpečení a ochranu soukromí. Například testují možnost přenosu historie konverzací mezi různými zařízeními a používání jednoho uživatelského účtu na různých zařízeních najednou. Nyní je již možné, díky tzv. E2EE zálohám, přenést historie konverzací i z jedné platformy na druhou, tedy z Android na iOS a stejně tak i opačně. [10]

Zálohu je možné zabezpečit E2EE buď heslem, které si uživatel vybere, anebo 64místným klíčem, který zná pouze uživatel. Tvorba klíče a šifrování zálohy probíhá lokálně na zařízení uživatele a až poté se tato zašifrovaná záloha nahrává na příslušný cloud v závislosti na používanou platformu (Google Drive či iCloud). WhatsApp tedy zaručuje, že stejně jako ke zprávám, se ani k záloze nikdo kromě uživatele nedostane. [11]

1.4.4 Wire

Aplikace Wire poskytuje zabezpečení pomocí E2EE pro textové zprávy s podporou sdílení GIF, audio a video a dalších typů souborů, dále také hlasové hovory a video hovory. Aplikace také nabízí multiplatformní synchronizaci mezi zařízeními a podporu pro více účtů, což uživatelům umožňuje oddělit například osobní a pracovní komunikaci.

Wire používá svůj vlastní kryptografický protokol nazývaný Proteus, který je založen na protokolu Signal. Stejně jako v případě protokolu Signal, je i protokol Proteus open-source. Oproti ostatním výše zmíněným aplikacím, trpí Wire nedostatkem ve formě absence dvoufaktorové autentizace (2FA). [10]

2 KRYPTOGRAFICKÉ PROTOKOLY PRO IM APLIKACE

S rostoucí poptávkou po soukromí a bezpečí uživatelů na internetu roste i počet nově vzniklých kryptografických protokolů, které popisují nejlepší postupy implementace E2EE.

Síla E2EE se odvíjí od použitých kryptografických algoritmů a metod použitých pro správů klíčů. Mezi běžné algoritmy používané k implementaci E2EE se řadí například Rivest-Shamir-Adleman (RSA), který je aktuálně považován za spolehlivý a bezpečný. Dalším důležitým faktorem pro bezpečnou implementaci E2EE je, správa klíčů. Efektivní správa klíčů zahrnuje generování, distribuci, ukládání a jejich pravidelné obnovování. Pokud by byly metody správy klíčů implementovány špatně, zcela jistě by mohli vzniknout zranitelnosti a celková bezpečnost komunikačního kanálu by byla ohrožena. Právě proto by se vývojáři aplikací, které využívají E2EE pro zabezpečení dat, neměli pokoušet implementovat E2EE pouze podle svých uvážení, ale měli by využít kvalitně sepsaných standardů a protokolů. [6]

Mezi nejznámější kryptografické protokoly vhodné pro IM aplikace se řadí například Signal Protocol, Proteus či MTProto. Tyto protokoly budou dále popsány v rámci této kapitoly.

2.1 Signal Protocol

Signal Protocol je velmi populární kryptografický protokol, který poskytuje end-to-end šifrování nejen pro konvenční real-time chatové, ale i hlasové konverzace. Obsahuje několik nezávislých specifikací, které jsou vhodné pro řešení různých typů problémů. Tento protokol byl vyvinut společností Open Whisper Systems v roce 2013 a poprvé byl veřejnosti představen v rámci open-source aplikace TextSecure, ze které později vznikl dnes známý Signal. Kromě aplikace Signal je tento protokol využíván i dalšími dobře známými aplikacemi jako je například WhatsApp, Facebook Messenger, Google Messages a Skype. [8]

Signal je postaven na dobře známých, vyzkoušených a testovaných bezpečnostních standardech. Protokol také prošel rozsáhlou kontrolou ze strany bezpečnostní komunity a odborníků na kryptografii. V článku „A Formal Security Analysis of the Signal Messaging Protocol“ po provedení hloubkové bezpečnostní analýzy došli vědci k závěru, že je z kryptografického hlediska v pořádku. [12]

Odborníci na kryptografii také chválí jejich implementaci „future secrecy“. Signal ji řeší pomocí nové techniky známé jako „ratcheting“, která používá nový klíč pro každou

novou zprávu. To znamená, že v případě kompromitace klíče relace nebude útočník schopen dešifrovat budoucí komunikaci. [12]

2.1.1 X3DH

Extended Triple Diffie-Hellman, zkráceně X3DH, je jedním ze základních stavebních kamenů Signal protokolu. Jedná se o protokol pro asynchronní výměnu klíčů. Pomocí tohoto protokolu se dvě strany, ověřené na základě veřejných klíčů, dohodují a vytváří sdílený tajný klíč pro další komunikaci. Fakt, že je tento protokol navržen pro asynchronní výměnu klíčů, umožňuje, aby byl jeden z komunikujících uživatel offline. Celý proces by se dal rozdělit do 3 základních kroků, které je nutné provést pro zahájení šifrované komunikace [16]:

1. První uživatel (Bob) publikuje jeho *identity key* a *prekeys*
2. Druhý uživatel (Alice) získá *prekey bundle* a odešle inicializační zprůvu
3. První uživatel (Bob) zprávu přijme a zpracuje ji

Publikace identity key a prekeys

První uživatel, dále označován pouze jako Bob, tedy na server publikuje *identity key*, který je platný dlouhodobě a nemění se. Dále publikuje jeho *signed prekey*, který je podepsán pomocí jeho soukromého *identity key*, a *prekey signature*. Všechny informace, *identity key, signed prekey* a *prekey signature*, jsou pak uloženy na serveru. Ostatní uživatelé, kteří chtějí s Bobem komunikovat, poté mohou podpis zvalidovat pomocí Bobova veřejného *identity key*. Kromě toho Bob publikuje i sadu tzv. *one-time prekeys*, které jsou unikátní, indexovatelné a použitelné vždy pouze jednou. Po každém požadavku je jeden z těchto klíčů vymazán a pokud jich zbývá už pouze minimum, server zažádá Boba o nové. [16]

Získání prekey bundle a odeslání inicializační zprávy

Druhý uživatel, dále označován pouze jako Alice, při zahájení komunikace získává tzv. *prekey bundle* uživatele se kterým chce komunikovat. Tento svazek se tedy skládá z již zmiňovaných klíčů [16]:

- Identity key
- Signed prekey

- Prekey signature
- One-time prekey

Alice si poté ověří *prekey signature* a pokud potvrzení proběhne úspěšně, vygeneruje si svůj tzv. *ephemeral key*. Následně spočítá sdílený soukromý klíč pomocí funkce pro odvození klíče (KDF) jejíž výpočet lze vidět níže [16]:

$$DH_1 = DH(ID\ key_A, signed\ prekey_B)$$

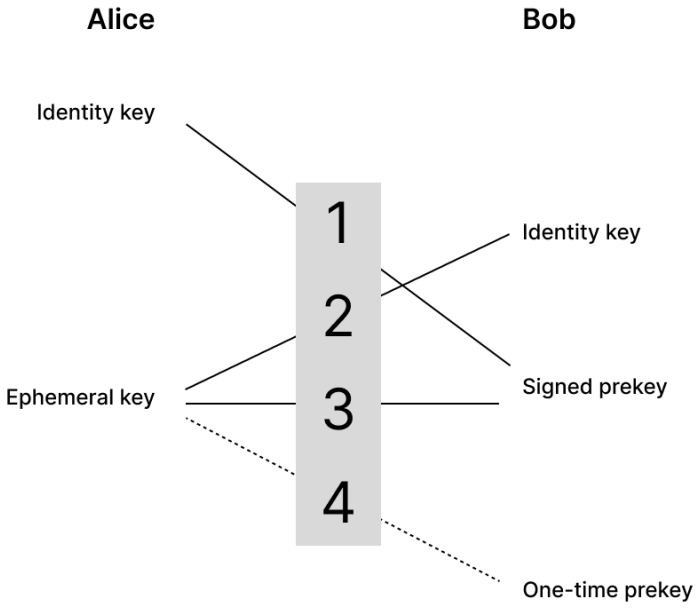
$$DH_2 = DH(ephemeral\ key_A, ID\ key_B)$$

$$DH_3 = DH(ephemeral\ key_A, signed\ prekey_B)$$

$$DH_4 = DH(ephemeral\ key_A, onetime\ prekey_B)$$

$$KEY = KDF(DH_1 \mid DH_2 \mid DH_3 \mid DH_4)$$

Níže uvedený diagram demonstruje prováděné Diffie-Hellman výpočty.



Obrázek 5 Demonstrace výpočtů X3DH [16]

Po výpočtu sdíleného symetrického klíče Alice vymaže svůj ephemeral klíč a Diffie-Hellman výstupy a ponechá si pouze onen vypočítaný klíč. Poté jsou ještě vypočítány tzv.

associated data (AD) z identity klíčů obou účastníků, které obsahují informace o identitě obou komunikujících stran. Poté již může Alice odeslat inicializační zprávu, která obsahuje její identity a ephemeral klíč, identifikátor udávající, který z Bobových one-time prekeys využila a zašifrovanou zprávu společně s associated daty. [16]

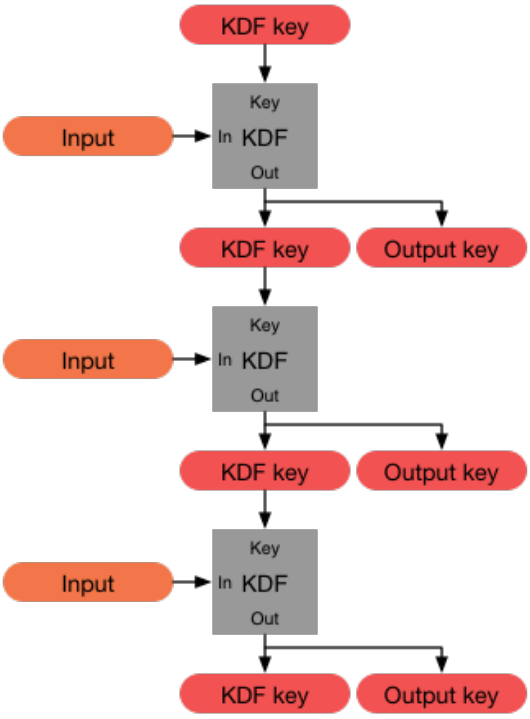
Příjem a zpracování zprávy

V poslední fázi Bob přijímá zprávu od Alice a načte své soukromé klíče, konkrétně identity key, signed prekey a Alicí vybraný one-time prekey. Pomocí těchto klíčů zopakuje Diffie-Hellman a KDF výpočty stejně jako Alice, tím získá sdílený klíč a také smaže ostatní hodnoty. Následně také provede výpočet associated dat a dešifruje inicializační šifrovanou zprávu a pomocí sdíleného klíče a associated dat. Jestliže dešifrování zprávy selže, Bob ukončí protokol a vymaže sdílený klíč. Pokud ale dešifrování proběhne v pořádku, protokol je dokončen, a nakonec Bob ještě smaže použitý one-time prekey.

2.1.2 Double Ratchet

Double Ratchet algoritmus je důležitou součástí Signal Protocolu a zajišťuje tzv. future secrecy pro každou vyměněnou zprávu mezi uživateli. Tento algoritmus pracuje s vytvářením a správou sekvencí dočasných klíčů použitelných pro šifrování a dešifrování zpráv. [18]

Jádrem Double Ratchet algoritmu jsou Key Derivation Function řetězce neboli KDF chains. KDF lze definovat jako kryptografickou funkci, která bere tajný a náhodný klíč a vstupní data a vrací výstupní data. Tato výstupní data jsou k nerozeznání od dat náhodných za předpokladu, že není znám klíč. Jedna část výstupu KDF řetězce se používá k šifrování a dešifrování zpráv a druhá část je pak použita jako vstupní KDF klíč pro další iteraci. Tento princip je demonstrován v níže uvedeném diagramu. [18]



Obrázek 6 Demonstrace principu KDF chains [18]

V relaci mezi dvěma účastníky, Alicí a Bobem, si každá strana uloží KDF klíč pro tři řetězce: root chain, sending řetězec a receiving řetězec, kdy se odesílací řetězec Alice shoduje s přijímacím řetězcem Boba a naopak. [18]

Proces Double Ratchet se skládá ze dvou hlavních částí, a to Diffie-Hellman Ratchet a Symmetric-Key Ratchet. [18]

- **Diffie-Hellman Ratchet:** Když si dva účastníci, Alice a Bob, vyměňují zprávy, vyměňují si také nové veřejné Diffie-Hellman klíče, poté se Diffie-Hellman výstupní tajemství (output secret) stávají vstupem do root řetězce. Výstupní klíče z root řetězce se stanou novými KDF klíči pro sending a receiving řetězce.
- **Symmetric-Key Ratchet:** Slouží vyloženě k tvorbě klíčů pro šifrování a dešifrování a také vstupních klíčů pro další iterace. Každá zpráva je zašifrována pomocí jednorázového klíče, který je odvozen od předchozího klíče a pak je ten původní zahozen. To znamená, že i když je jeden symetrický klíč prolomen, nemá to vliv na bezpečnost předchozích či následujících zpráv.

2.2 Proteus

Proteus je open-source kryptografický protokol použitý v aplikaci Wire, který poskytuje end-to-end šifrování zpráv a médií. Jedná se o nezávislou implementaci protokolu Axolotl (předchůdce protokolu Signal) / Double Ratchet, který je naopak odvozen od protokolu Off-the-Record. [19]

Stejně jako Signal Protocol i protokol Proteus prošel poměrně rozsáhlým a nezávislým auditem společnostmi Kudelski Security a X41 D-Sec, které zveřejnily společnou recenzi implementace kryptografického protokolu pro zasílání zpráv. Zjistili, že má „vysokou míru bezpečnosti díky nejmodernějším kryptografickým protokolům a algoritmům a postupům softwarového inženýrství zmírňujícím riziko softwarových chyb“. [20]

2.3 MTProto

MTProto je open-source kryptografický protokol vyvinutý společností Telegram, který sama společnost ve své aplikaci využívá. Přestože je MTProto open-source a umožňuje komukoli zkontrolovat jeho silné a slabé stránky, je v něj obecně značná nedůvěra. To pramení z faktu, že jej používá pouze Telegram a není tak důkladně testován jako jiné široce dostupné kryptografické protokoly. Několika výzkumníkům se také podařilo objevit bezpečnostní chyby. Například tým vědců z ETH Zurich a University of London uvádí, že MTProto neposkytuje záruku zabezpečení. Podle jejich vyšetřování obsahuje bezpečnostní chyby, jako je schopnost manipulovat se sekvencemi zpráv a detekovat zprávy zašifrované klientem nebo serverem a potenciál pro útoky typu Man-In-The-Middle. Na základě těchto zjištění ale Telegram opravil všechny tyto zranitelnosti. Takových zranitelností však může být ještě mnoho a objevit je mohou až budoucí výzkumy. [21]

2.4 Volba vhodného kryptografického protokolu k implementaci

Vývojáři aplikací, které budou využívat E2EE, by také při výběru vhodného kryptografického protokolu měli myslet na důležité věci jako je obtížnost integrace do jejich vyvíjené aplikace a aktualizace a podpora ze strany poskytovatele protokolu.

2.4.1 Faktory porovnání kryptografických protokolů

Kryptografické protokoly je možné porovnávat na základě několika různých faktorů, mezi ty nejdůležitější se pak může řadit například následující:

- **Bezpečnost** – Jaká úroveň bezpečnosti je protokolem poskytována?
- **Transparentnost** – Je kryptografický protokol otevřený pro audit a analýzu komunitou? Otevřený kód umožňuje nezávislým výzkumníkům provádět revize a případně potvrdit bezpečnost protokolu.
- **Šíření a popularita** – Jak je protokol rozšířen? Velký uživatelský základ může naznačovat jeho důvěryhodnost a spolehlivost.
- **Složitost integrace** – Jak snadno lze protokol integrovat do různých platforem a aplikací?
- **Aktualizace a podpora** – Jak často je protokol aktualizován? Poskytuje protokol podporu nezávislým společnostem a vývojářům?

2.4.2 Výběr kryptografického protokolu

Níže budou ve formátu tabulky porovnány výše zmíněné kryptografické protokoly poskytující end-to-end šifrování na základě definovaných parametrů. Data v níže uvedené tabulce pochází z těchto zdrojů: [12] [19] [20] [21].

Tabulka 1 Porovnání kryptografických protokolů

	Signal	**Proteus**	**MTProto**
Bezpečnost	Shledán bezpečným	Shledán bezpečným	Bezpečnostní nedostatky
Transparentnost	Open-source	Open-source	Open-source
Šíření a popularita	Velmi populární	Menší uživatelská základna, využívá pouze Wire	Menší uživatelská základna, využívá pouze Telegram
Složitost integrace	Dostupné implementace ve formě knihoven pro různé programovací jazyky	Dostupné implementace ve formě knihoven pro Rust a TypeScript	Neexistují oficiální knihovny od společnosti Telegram

Aktualizace a podpora	Aktualizace i podpora	Aktualizace i podpora	Aktualizace, ale slabší podpora

Na základě průzkumu bylo zjištěno, že nejlepším možným řešením je zvolit velmi populární Signal Protocol, který je podpořen kryptografickou odbornou veřejností, je otestován různými společnostmi v praxi a má velkou komunitu uživatelů. Vývojový tým tohoto protokolu, Signal, také na svém oficiálním GitHub účtu poskytuje open-source implementace v podobě knihoven pro různé programovací jazyky.

Jako možná alternativa se nabízí protokol Proteus, který je vlastně odnoží protokolu Signal a je také podpořen kryptografickou odbornou veřejností a v rámci nezávislých studií. Oficiální poskytovatel protokolu, Wire, také na svém GitHub účtu nabízí implementaci v podobě knihoven pro programovací jazyky Rust a TypeScript. Nevýhodou oproti protokolu Signal je pak menší uživatelská základna a absence ověření fungování v praxi větším množstvím společností. Další obrovskou je, že tento protokol nemá žádnou oficiální dokumentaci.

Co se týče posledního protokolu, MTProto, který vyvinula společnost Telegram pro vlastní účely, existují určité neshody a kontroverze ohledně jeho bezpečnosti. MTProto také, stejně jako Proteus, není implementován žádnou jinou společností, než která jej vyvinula. Další obrovskou nevýhodou pro potřeby vývoje vlastní aplikace je absence oficiální implementace tohoto protokolu v podobě knihoven, Telegram poskytuje pouze teoretickou znalost, nikoliv pak oficiální knihovní implementaci.

3 ROZBOR TECHNOLOGIÍ VYUŽITÝCH K IMPLEMENTACI

V této kapitole budou popsány veškeré technologie, jako jsou programovací jazyky, knihovny, frameworky atp., které jsou využity k implementaci aplikace. Hlavním programovacím jazykem, na kterém je postavena klientská i serverová část aplikace, je JavaScript. JavaScriptu a odvozeným technologiím bude věnována celá podkapitola. Důležitou částí této kapitoly bude také knihovna, která je v aplikaci využita pro účely implementace vybraného kryptografického protokolu pro end-to-end šifrování. Také zde budou znázorněny technologie jako dokumentová databáze MongoDB, komunikační protokol WebSocket, kaskádové styly a JWT.

3.1 Javascript technologie

JavaScript je skriptovací, objektově orientovaný, interpretovaný jazyk, který je nejčastěji využíván k tvorbě interaktivních webových stránek a aplikací. Může se jednat například o komplexní animace nebo funkční tlačítka, která vyvolávají různé akce (např. zobrazení různých pop-up oken, menu či odeslání požadavku na server). [22]

V dnešní době má však JavaScript mnohem více využití než tvorbu interaktivních webových stránek. JavaScript se pomocí dalších technologií dá využít například i pro tvorbu mobilních a desktopových aplikacích či k tvorbě komplexních serverových aplikací. Dobrým příkladem v rámci mobilních aplikací je technologie React Native, která v tomto jazyce umožňuje tvořit cross-platform aplikace jak pro Android, tak i pro iOS. V rámci desktopových aplikací je pak velmi populární framework Electron, díky kterému lze naprogramovat cross-platform aplikaci pro různé desktopové operační systémy. Tvorbu serverových aplikací pak umožňuje dobře známý Node.js a na něj navazující frameworky jako je Express.js, který bude v této kapitole dále přiblížen.

Na straně klienta je jádro JavaScriptu rozšířeno o objekty pro ovládání prohlížeče a jeho Document Object Model (DOM), což umožňuje aplikacím například předat hodnoty z JavaScriptu do HTML formuláře, upravit HTML elementy na stránce či reagovat na uživatelské události, jako je kliknutí myší na tlačítko atd. [22]

Na straně serveru je pak jádro JavaScriptu rozšířeno o objekty pro spouštění JavaScriptu na serveru. Toto rozšíření umožňuje aplikaci například komunikovat s databází nebo manipulovat se soubory na serveru. [22]

Jinými slovy, v prohlížeči může JavaScript měnit vzhled webové stránky (DOM) a odesílat požadavky na server. A podobně JavaScript na serveru může na tyto požadavky, zaslané kódem spuštěným v klientské aplikaci, reagovat. [22]

3.1.1 TypeScript

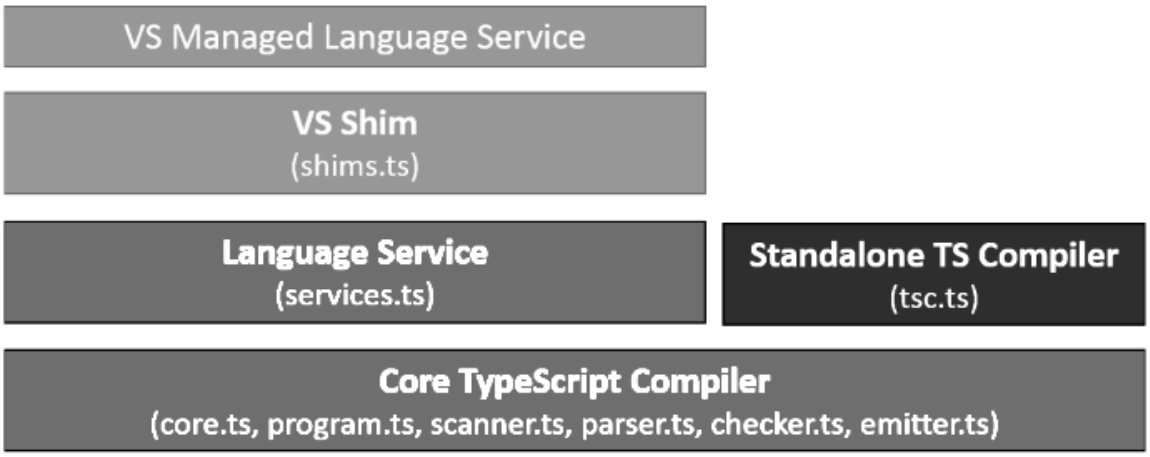
TypeScript je silně typovaný, objektově orientovaný a transpilovaný programovací jazyk, který pracuje jako nadstavba jazyku JavaScript. Byl vyvinut v roce 2012 společností Microsoft primárně z důvodu řešení některých nedostatků klasického JavaScriptu. Hlavní návrhář, který stojí za jazykem TypeScript, je Anders Hejlsberg, který v minulosti ve společnosti Microsoft navrhoval jazyk i C#, což je i důvodem, proč je syntaxe těchto jazyků poměrně podobná. [23]

TypeScript zahrnuje veškerou existující syntaxi JavaScriptu a přidává další jeho specifickou syntaxi pro definovaní a používání typů. To v praxi mimo jiné znamená, že veškerý kód napsaný v JavaScriptu bude funkční i v TypeScriptu. Jak již bylo také zmíněno, TypeScript je tzv. transpilovaný, což znamená, že je překládám do JavaScriptu. Webové prohlížeče, respektive jejich enginy, totiž neumí TypeScript spouštět přímo, ale je nutné jej nejprve převést do klasického JavaScriptu. [23]

Primární výhodou tohoto jazyka je statické typování. Zatímco u JavaScriptu, který je dynamicky typovaný, se typy proměnných nastavují až za běhu programu, TypeScript nabízí možnost specifikovat typy v době kompilace prostřednictvím právě statického typování. Což znamená, že jakmile je proměnná deklarována jako určitý typ, není možné ji poté možné přiřadit typ jiný. Statické typování může zabránit mnoha chybám, které jsou pro dynamicky typované jazyky typické. [24]

TypeScript je vnitřně rozdělen do tří hlavních vrstev. Každá z těchto vrstev je rozdělena na podvrstvy nebo komponenty [25]:

- **Jazyk:** Obsahuje prvky jako syntaxe, klíčová slova atd.
- **The TypeScript Compiler (TSC):** Provádí analýzu a kontrolu typů. Také překládá TypeScript kód do JavaScript kódu, jelikož prohlížeče neumí spouštět přímo TypeScript kód.
- **TypeScript Language Services:** Generuje informace, které jsou nápomocné editorům a dalším nástrojům v poskytování lepších asistenčních funkcí jako je jako je IntelliSense nebo automatizovaný refaktoring.

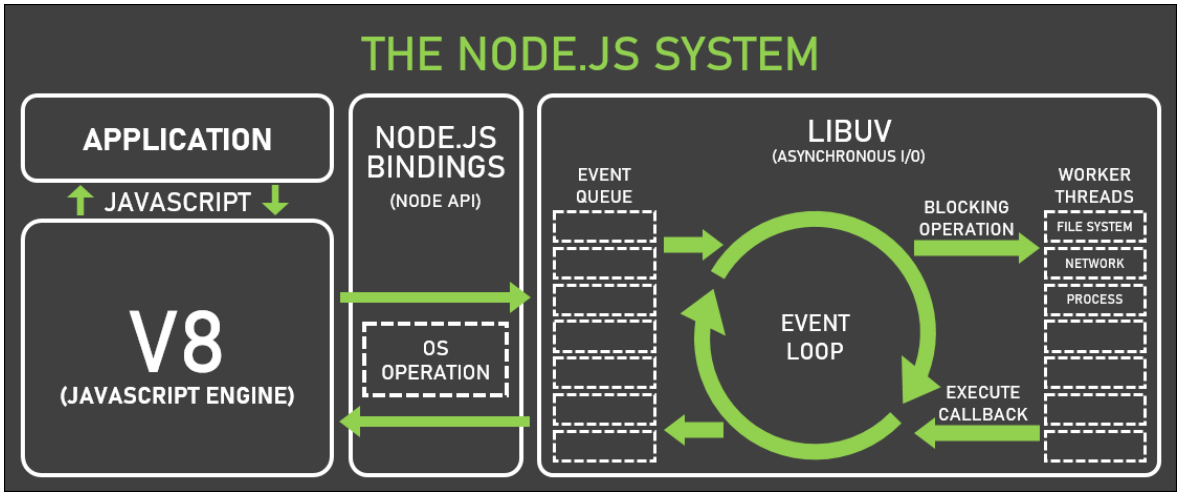


Obrázek 7 Vrstvy jazyka TypeScript [25]

3.1.2 Node.js

Node.js je JavaScriptové běhové prostředí, které umožňuje vytvářet velmi škálovatelné serverové aplikace pomocí programovacího jazyka JavaScript. Node.js umožňuje spouštět JavaScript mimo prohlížeč díky JavaScriptovému enginu V8, který byl vyvinut společností Google v rámci projektu Chromium. Dodnes je V8 jádrem prohlížeče Google Chrome. [26]

Node.js aplikace běží v jednom procesu, tedy nevytváří nové vlákno pro každý požadavek. Jednou z typických vlastností Node.js je jeho architektura řízená událostmi (event-driven). Díky této architektuře Node.js dokáže zpracovat více souběžných operací bez blokování provádění jiných úloh. Toho je dosaženo díky non-blocking I/O, kde jsou vstupní a výstupní operace prováděny asynchronně, tedy programu je umožněno pokračovat ve vykonávání dalších úloh, zatímco čeká na dokončení I/O operací. Což v praxi znamená, že při vykonávání operací, jako je čtení z databáze nebo souborového systému, se hlavní vlákno nezastaví, aby čekalo na dokončení těchto operací. Místo toho se tyto operace provádějí paralelně, a jakmile jsou dokončeny, výsledek se oznámí hlavnímu vláknu prostřednictvím callback funkce. [26]



Obrázek 8 Node.js event-loop architektura [27]

Když uživatel na server odešle požadavek, dochází k vytvoření události. Pokud událost vyvolává asynchronní úkoly, tedy úkoly, které nelze dokončit najednou, Node.js musí znát, která událost nastala první a v jakém pořadí by měla být dokončena. Node.js předá tuto operaci do pozadí a pokračuje ve zpracování dalšího kódu. Jakmile je asynchronní operace dokončena, výsledek se zařadí do fronty událostí a zpracuje se, jakmile to bude možné. [28]

Pod kapotou Node.js využívá knihovnu libuv, která zajišťuje asynchronní I/O operace. Libuv používá pool vláken (thread pool) pro operace, které nelze provést asynchronně na úrovni operačního systému. To zahrnuje operace, jako je práce s soubory, kde libuv deleguje práci do vedlejších vláken, aby neblokovala hlavní událostní smyčku. Díky těmto mechanismům může Node.js efektivně zpracovávat tisíce souběžných spojení s minimální režií, což je činí ideální pro vývoj I/O-intenzivních aplikací, jako jsou webové servery, API, real-time aplikace a další. [29]

3.1.3 Express.js

Express.js je velmi populární a minimalistický webový framework pro tvorbu serverových aplikací v Node.js, který je hojně využívaný jako nástroj, který obohacuje Node.js o routing a middleware. Od svého založení v roce 2010 TJ Holowaychukem se tento framework stál důležitou součástí ekosystému Node.js a postupem času se stal tak populární, že velké množství zdrojů o Node.js odkazuje právě na Express.js. V roce 2015 byl Express.js začleněn do

inkubátoru Nadace Node.js pod správou IBM a tím byla zajištěna jeho dlouhodobá udržitelnost. [30]

Express.js se vyznačuje modulární architekturou založenou na middleware, která umožňuje vývojářům vytvářet a registrovat middleware funkce pro různé účely od autentizace API klíčů po zpracování chyb. Tento přístup, společně s podporou routování, činí Express.js adaptabilním nástrojem pro vývoj Node.js aplikací. Express.js je také dobře známý pro svou pozoruhodnou rychlost. Jeho rychlost se připisuje jeho lehké architektuře a neblokujícímu I/O modelu Node.js, který umožňuje efektivní zpracování tisíců simultánních požadavků bez výrazného snížení výkonu. [30]

Níže bude demonstrován jednoduchý kód, kde lze vidět obslužné funkce pro dva ukázkové endpointy. První pro login, kde lze vidět, že je volána metoda *post*, která označuje HTTP POST požadavek na serverový endpoint */login*. Druhý pro získání dat produktů, kde je volána metoda *get*, která naopak značí HTTP GET požadavek na endpoint */products*. U druhého příkladu lze také vidět, že druhým parametrem je funkce *isAuthenticated*, která představuje middleware zajišťující kontrolu přihlášeného uživatele. Pokud by tedy uživatel nebyl přihlášen, neměl k danému endpointu dostatečné oprávnění nebo by například neměl platný token, endpoint by mu nevrátil data, nýbrž chybovou hlášku.

```
route.post('/login', (req, res) => {
  // Obslužný kód pro přihlášení uživatele ...
});

route.get('/products', isAuthenticated, (req, res) => {
  // Obslužný kód pro získání produktů z DB
  // s přidaným middlewarem pro kontrolu uživatele
});
```

Kód 1 Demonstrace routes a middlewaru v Express.js (vlastní kód)

3.1.4 React

React je open-source JavaScriptová knihovna vytvořená společností Facebook s cílem vytvářet uživatelská rozhraní pro single-page aplikace. React byl poprvé použit v roce 2011 v aplikaci Facebook, v roce 2012 byl implementován v aplikaci Instagram a v roce 2013 zveřejněn pro veřejnost jako open-source. [31] React je ideální pro aplikace, které rychle a dynamicky mění svá data. Umožňuje zobrazovat měnící se části uživatelského rozhraní bez potřeby překreslování celou stránku. React využívá tzv. diffing algoritmus, který zajišťuje

efektivní a rychlou aktualizaci uživatelského rozhraní. Jakmile se změní stav komponenty, React vytvoří nový tzv. Virtual DOM, který následně porovná s aktuálním stavem reálného DOM. Díky tomu v DOM stromu aktualizuje pouze to, co je potřeba, zatímco zbytek ponechává beze změny. [32]

Pro renderování React využívá tzv. JSX (JavaScript XML). JSX je rozšíření JavaScriptu, které umožňuje definovat uživatelské rozhraní ve formátu, který se podobá značkovacímu jazyku HTML.

React je známý pro svou architekturou založenou na komponentách. Komponenty jsou nezávislé, znovupoužitelné části kódu, na které lze nahlížet jako na stavební bloky aplikace. Každá komponenta může spravovat svůj vlastní stav, tedy může uchovávat a manipulovat s daty uvnitř sama sebe. React nabízí dvě možnosti tvorby komponent, a to pomocí tříd a pomocí funkcí. [33]

Komponenty vytvořené použitím funkcí jsou vlastně klasické javascriptové funkce, které vrací JSX kód a mohou přebírat různé parametry. V rámci těchto komponent se používají tzv. hooks, které zde zajišťují manipulaci se stavem a dalšími reaktivními vlastnostmi. Mezi základní hooky patří useState, který umožňuje vytvářet, upravovat a držet stav komponenty. Dále hook useEffect, který pak umožňuje reagovat na prvotní načtení komponenty, na její „odpojení“ (tzv. unmount) a na změnu jejího stavu či parametrů.

Komponenty vytvořené pomocí tříd jsou naproti tomu zapsané jako javascriptová třída, která obsahuje konstruktor, kde je definovaný počáteční stav a metody dané komponenty. Parametry takové komponenty jsou pak dostupné v rámci vlastnosti její třídy nazvané jako *props.* Každá komponenta vytvořená pomocí tříd implementuje metodu *render*, jenž vrací JSX kód určený k vykreslení. Obdobou výše zmíněného hooku useEffect jsou zde tzv. life-cycle metody. Metoda *componentDidMount* reaguje na načtení komponenty, metoda *componentDidUpdate* reaguje na změnu stavu či parametrů a metoda *componentWillUnmount* reaguje na „odpojení“ komponenty. [34] Rozdíl mezi komponenty vytvořenými pomocí tříd a pomocí funkcí je demonstrován v jednoduchém příkladě níže.

```
const BasicComponentFn = () => {
  const [val, setVal] = useState(0);
  const handleClick = () => setVal(val + 1);

  return (
    <div>
```

```
      <h1>Na tlačítko kliknuto {val} krát.</h1>
      <button onClick={handleClick}>Klikni</button>
    </div>
  );
}

class BasicComponentClass extends Component {
  constructor(props) {
    super(props);
    this.state = { val: 0 };
  }
  handleClick = () => this.setState({ val: this.state.val + 1 });

  render() {
    return (
      <div>
        <h1>Na tlačítko kliknuto {this.state.val} krát.</h1>
        <button onClick={this.handleClick}>Klikni</button>
      </div>
    );
  }
}
```

Kód 2 Rozdíl mezi class a function komponentou (vlastní kód)

3.1.5 Vite

Vite je velmi rychlý build nástroj, který vytvořil programátor Evan You, tvůrce populárního front-end frameworku Vue.js. Cíl Vite je poskytnout vývojářům rychlejší a pohodlnější uživatelskou zkušenost při vývoji moderních webových aplikací. [35]

Vite slouží primárně jako velmi sofistikovaný nástroj pro bundle JavaScriptových modulů. Bundling je proces, který prochází a spojuje JavaScriptové moduly do souborů, které poté mohou být spouštěny v prohlížeči. Mezi nejběžněji používané nástroje pro bundling patří Webpack, Rollup či Parcel, které se však v případě zpracovávání velkých projektů s obrovským množstvím modulů často setkávají s problémy s výkonem. V takových případech není neobvyklé, že spuštění vývojového serveru trvá neúměrně dlouho, nebo že se změny provedené v souborech projeví až po několika sekundách. Vite tyto problémy řeší díky podpoře nativních JavaScriptových modulů v prohlížeči. Provozuje server, který na požádání kompiluje a poskytuje jakékoli potřebné závislosti právě prostřednictvím těchto modulů. Tento přístup umožňuje Vite zpracovávat a poskytovat pouze požadovaný kód. [36]

Vite vylepšuje dobu spuštění vývojového serveru tak, že rozděluje moduly do dvou kategorií, a to do závislostí (dependencies) a do zdrojového kódu (source code). Závislosti představují čistý JavaScriptový kód, který pravděpodobně zůstane neměnný během vývojového procesu. Všechny závislosti jsou tzv. *prebundled* pomocí nástroje zvaného esbuild, který je naprogramovaný v jazyce Go. Poté po zbytek vývojového procesu budou tyto moduly servírovány již jaké bundle a nebudou dále zahrnovány v dalších Vite runtime procesech. Naopak zdrojový kód se bude během vývoje měnit velmi často a tyto soubory jsou servírovány přímo jako ES moduly. [36]

Pro produkční sestavení Vite používá bundle nástroje Rollup, kde přebírá kód vytvořený během vývojového procesu a provede jeho sestavení do optimalizovaných souborů. Provede kompilaci a spojení modulů do menšího počtu balíčků a aplikuje minifikaci k redukci velikosti souborů čímž usnadní jejich načítání. [36] Vite také využívá techniku zvanou *tree shaking* k odstranění nepoužívaného kódu a *code splitting* pro rozdělení aplikace do menších částí, které jsou pak načítány postupně na vyžádání. Vite také optimalizuje assety, jako jsou obrázky a styly tak, aby byly co nejefektivněji distribuovány a cachovány v prohlížečích uživatelů. [37]

3.2 Signal Protocol TypeScript knihovna

Knihovna *libsignal-protocol-typescript* je implementací protokolu Signal v jazyce TypeScript a je založena na knihovně *libsignal-protocol-javascript*. Knihovna byla vyvinuta společností Privacy Research LLC, což je výzkumná skupina v oblasti bezpečnosti a ochrany soukromí se sídlem v Palo Alto v Kalifornii. [38]

Jak již v této práci bylo nastíněno, protokol Signal používá koncept takzvaných prekeys. Při prvním spuštění klient generuje jeden podepsaný prekey a velký počet nepodepsaných prekeys, které jsou následně předány serveru.

Klienti vytvářejí relace, tzv. sessions, které se používají pro všechny následující operace šifrování a dešifrování. Session je založena buď prostřednictvím *PreKeyBundle*, kdy klient získává *PreKeyBundle* příjemce od serveru, nebo přes *PreKeySignalMessage*, kterou obdrží od příjemce. Stav každé session je ukládán v trvalých záznamech, které obsahují informace o identitě, prekeys a session samotné. Ukázka vytvoření session je demonstrována v kódu níže. [38]

Knihovna využívá WebCrypto pro symetrické kryptografické operace a generování náhodných čísel, zatímco pro operace s veřejnými klíči je použit tzv. AsyncCurve. [38]

```
const startSessionWithBoris = async () => {
  // get Boris' key bundle. This is a DeviceType<ArrayBuffer>
  const borisBundle = directory.getPreKeyBundle('boris')

  // borisAddress is a SignalProtocolAddress
  const recipientAddress = borisAddress

  // Instantiate a SessionBuilder for a remote recipientId + deviceId tuple.
  const sessionBuilder = new SessionBuilder(adiStore, recipientAddress)

  // Process a prekey fetched from the server. Returns a promise that resolves
  // once a session is created and saved in the store, or rejects if the
  // identityKey differs from a previously seen identity for this address.
  await sessionBuilder.processPreKey(borisBundle!)

  // Now we can encrypt a messageto get a MessageType object
  const senderSessionCipher = new SessionCipher(adiStore, recipientAddress)
  const ciphertext = await senderSessionCipher.encrypt(starterMessageBy-
tes.buffer)

  // The message is encrypted, now send it however you like.
  sendMessage('boris', 'adalheid', ciphertext)
}
```

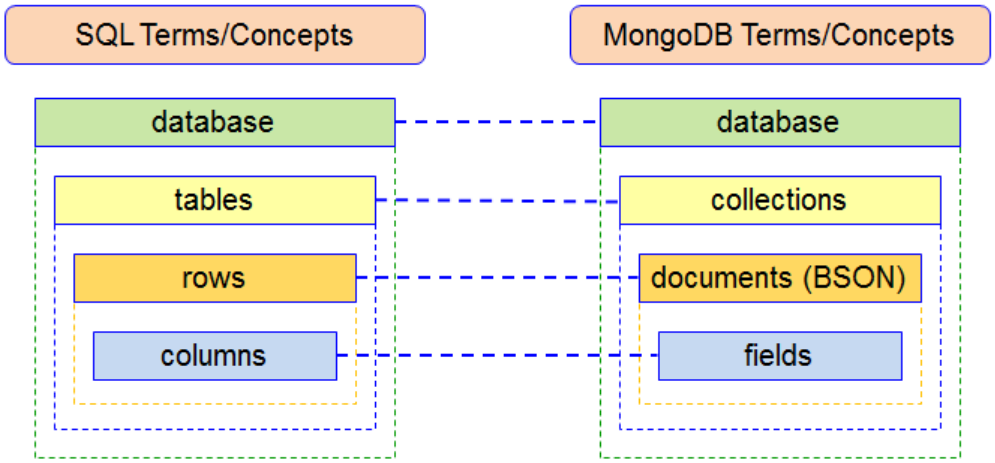
Kód 3 Vytvoření session pomocí libsignal-protocol-typescript [38]

3.3 Databáze MongoDB

MongoDB je jednou z nejznámějších NoSQL databází. Konkrétně jde o tzv. dokumentovou databázi, která je zároveň open-source.

Nerelační dokumentovou databázi lze použít k ukládání a dotazování dat ve formě dokumentů podobných JavaScript Object Notation (JSON). Pro vývojáře je poté zjednodušen proces manipulace s daty a jejich ukládání do databáze, protože v kódu aplikace pracují se stejnou či velmi podobnou reprezentací dat jako je v databázi. Dokumentové databáze také umožňují vytvářet dokumenty s různými poli, které jsou následně uchovávány v rámci jedné kolekce (tato terminologie je vysvětlena na obrázku níže v paralele s konvenčními relačními databázemi). Pokud je nutné ukládat nestrukturovaná data, jako jsou emaily či příspěvky na sociálních sítích, dokumentová databáze může být díky tomuto přístupu

vhodným řešením. Některé dokumentové databáze, ale umožňují ověřovat schéma a přidávat určitá omezení na strukturu. [39]



Obrázek 9 MongoDB terminologie v návaznosti na SQL databáze [40]

MongoDB také poskytuje flexibilní dotazování dat, tzv. *queries*. Dotazy mohou být formulovány na základě různých kritérií, které mohou zahrnovat konkrétní hodnoty, rozsahy hodnot nebo i složitější podmínky využívající logické operátory. Efektivita těchto dotazů je často závislá na použití indexů, které značně zlepšují rychlost vyhledávání. MongoDB umožňuje vytvořit index na jakémkoli poli dokumentu, a to včetně polí uvnitř vnořených objektů. Indexace v MongoDB je realizována pomocí datové struktury označované jako B-tree. Podporovány jsou různé typy indexů, od jednoduchých B-tree indexů, přes textové indexy pro full-textové vyhledávání, až po geoprostorové indexy pro práci s lokalizačními a geografickými daty. Díky indexaci je možné rychle nalézt dokumenty odpovídající dotazovacím kritériím a nemusí se tak zbytečně provádět skenování celé kolekce. [41]

MongoDB ukládá data ve formátu zvaném BSON, což je binární reprezentace JSON dokumentů. Oproti typickému JSON formátu ale obsahuje další datové typy, například [42]:

- **ObjectId:** běžně používaný jako unikátní identifikátor
- **Binary data:** ukládá binární data, typicky multimediální obsah
- **Date:** představuje datum a čas, což JSON ve své základní formě nepodporuje přímo a čas se musí ukládat jak řetězec
- **32-bit integer (int) / 64-bit integer (long):** celočíselné datové typy

- **Regular expression (regex):** ukládá regulární výrazy

3.3.1 Mongoose

Mongoose je Node.js knihovna zajišťující Object Data Modeling (ODM) pro MongoDB. Je to vlastně alternativa pro známe objektově relační mapování (ORM) známe z typických relačních databázích. Cílem Mongoose je umožnit vývojářům vynutit specifické schéma na úrovni aplikace a zajistit tak konzistenci a strukturu dat [43]. Níže v přiloženém kódu bude demonstrován příklad vytvoření schématu pomocí Mongoose.

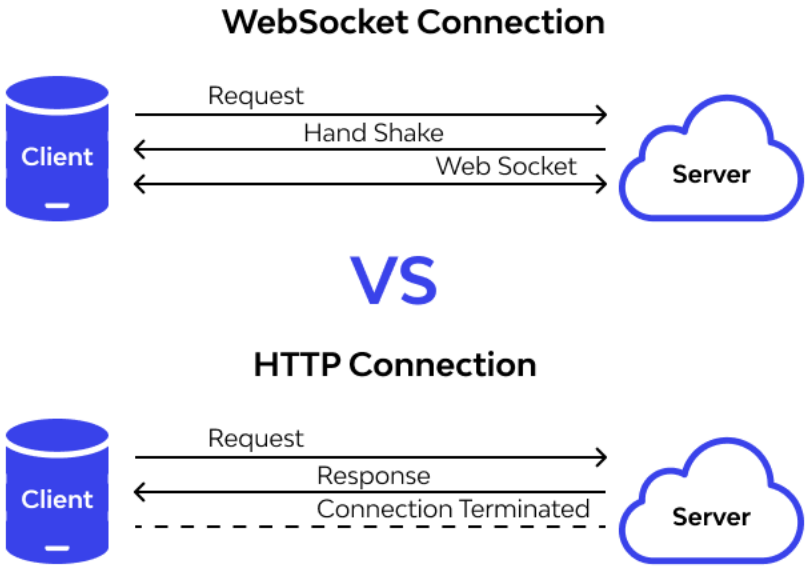
```
import { model, Schema } from 'mongoose'
import { IUser } from '../types/IUser';

const ObjectId = Schema.Types.ObjectId;

const userSchema = new Schema<IUser>({
    username: {
        type: String,
        required: true
    },
    password: {
        type: String,
        required: true
    },
    email: {
        type: String,
        required: true,
        validate: [emailValidation, 'Zadané pole musí být ve formátu e-mailu.'],
    },
    role: {
        type: String,
        required: true
    }
});

module.exports = model<IUser>('User', userSchema);
```

Kód 4 Demonstrace vytvoření schématu pomocí Mongoose (vlastní kód)

3.4 WebSocket

WebSocket je protokol umožňující oboucestnou komunikaci v reálném čase mezi webovým klientem a serverem. Tento protokol je navržen tak, aby bylo možné po inicializaci HTTP

požadavku přejít na trvalé WebSocket spojení. U WebSocket protokolu je po navázání spojení umožněno, aby byla data odesílána jak serverem, tak klientem ve formě zpráv, aniž by bylo nutné pro každou zprávu navazovat nové spojení. Tato vlastnost je vhodná pro aplikace, které vyžadují častou a rychlou výměnu zpráv, jako jsou herní aplikace či real-time komunikační systémy (chat aplikace). Rozdíl protokolu Websocket a HTTP je znázorněn na obrázku níže. [44]



Obrázek 10 Rozdíl protokolu Websocket a HTTP [44]

3.4.1 Javascript WebSocket server

WebSocket server není nic jiného než aplikace, která naslouchá na jakémkoli portu TCP serveru, který se řídí specifickým protokolem. [45] Samotný Node.js však nedisponuje řešením pro WebSocket a tak je nutné využít známe knihovny *WS*. Tato knihovna umožňuje poměrně jednoduché nastavení. Nejprve je tedy nutné knihovnu nainstalovat a pak už jen vytvořit instanci WebSocket serveru na příslušném portu a endpointu. Poté je již možné vytvořit obslužné funkce, které fungují jako callback funkce a ve kterých se definuje, co má server v určitých situacích dělat. V uvedeném příkladu níže lze vidět zjednodušený proces od importování ws modulu, přes vytvoření instance WebSocket serveru až po jeho obsluhu.

```
const WebSocket = require('ws');
```

```
// Vytvoření instance WSS
const wsServer = new WebSocket.Server({
   server: httpServer,
   path: '/wsserver,
   port: 3001
});

wsServer.on('connection', (ws) => {
    ws.on('message', handleWssMessage);
    ws.on('close', handleWssClose);

    ws.send('Welcome!');
});
```

Kód 5 Ukázka WebSocket serveru v Node.js (vlastní kód)

3.4.2 WebSocket na straně klienta

Na klientské části, tedy v prohlížeči uživateli, aplikace ke komunikaci s WebSocket serverem využívají nativní WebSocket API. Toto API umožňuje otevřít obousměrnou interaktivní komunikační relaci mezi prohlížečem a serverem. Nejprve je nutné vytvořit novou instanci třídy WebSocket, kde se v konstruktoru předává adresa WebSocket serveru, která by již místo typického http měla běžet na ws či zabezpečeném wss. Vytvořením této instance se klient automaticky pokusí připojit k zvolenému serveru. Jakmile je spojení navázáno, lze začít přenášet data. Zprávy jsou poté odesílány pomocí metody vytvořeného objektu WebSocket, která je zde pojmenovaná jako *send*. WebSocket API je řízené událostmi, když je zpráva přijata, je do objektu WebSocket zaslána událost. Pro zpracování příchozích zpráv je nutné do kódu přidat posluchač událostí nebo použít handler *onmessage*. Níže je opět uveden příklad implementace.

```
var ws = new WebSocket('ws://localhost:8080');

// Event handler pro přijetí zprávy od serveru
ws.onmessage = (event) => {
  console.log('Zpráva ze serveru ', event.data);
};

// Event handler pro otevření spojení
ws.onopen = (event) => {
  console.log('Spojení otevřeno.');
```

```
  // Odeslání zprávy serveru ihned po otevření spojení
  ws.send('Ahoj!');
};
```

Kód 6 Ukázka WebSocket API v JavaScriptu (vlastní kód)

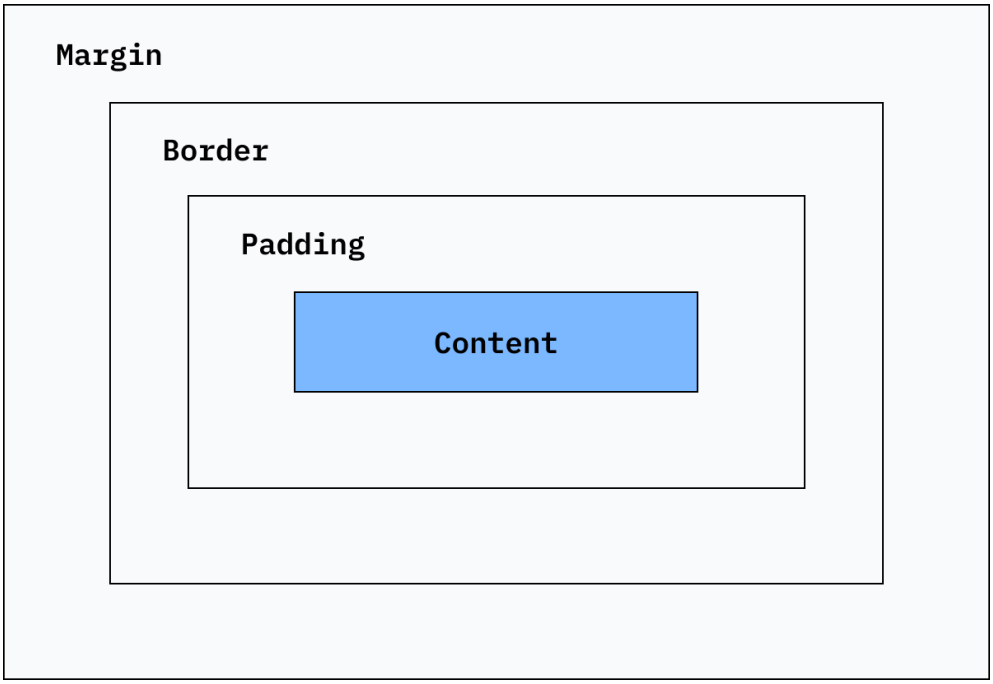
3.5 Kaskádové styly

Kaskádové styly, neboli Cascading Style Sheets (CSS), je jazyk, který slouží k definování způsobu zobrazení HTML elementů na webových stránkách a v aplikacích. CSS umožňuje měnit elementární prvky jako je barva textu či pozadí, velikost textu atd., ale hlavně také umožňuje definovat rozložení (layout) celé webové stránky či aplikace.

K aplikaci určitých stylů na HTML element je využito tzv. selektorů, které umožňují specifikovat na které elementy se mají dané styly aplikovat. Mezi základními selektory se řadí typové selektory, cílené přímo na HTML tag (např. *button*), třídní selektory (*.nazev_tridy*) a ID selektory (*#nazev_id*).

Styly lze do HTML dokumentu vkládat tzv. *inline*, přímo v HTML elementu pomocí atributu *style*, interně, v rámci tagu *style* umístěného v hlavičce dokumentu, nebo externě, pomocí externího CSS souboru propojeného s HTML dokumentem přes *link* tag. Každá z těchto metod má své využití, přičemž externí preferovány jsou externí styly kvůli lepší udržitelnost a čistotě kódu.

V CSS je v podstatě vše zabaleno uvnitř tzv. box modelu. Box model popisuje obdélníkové boxy generované pro veškeré prvky z DOM (Document Object Model) stromu. Tedy každý HTML element v dokumentu je reprezentován jako obdélník, který se skládá ze 4 vrstev, a to z obsahu, paddingu (vnitřní odsazení), rámečku a marginu (vnějšího odsazení). Tyto čtyři oblasti poté společně určují skutečnou velikost boxu v dokumentu. Pro lepší vizualizaci je toto znázorněno na obrázku níže. [46]

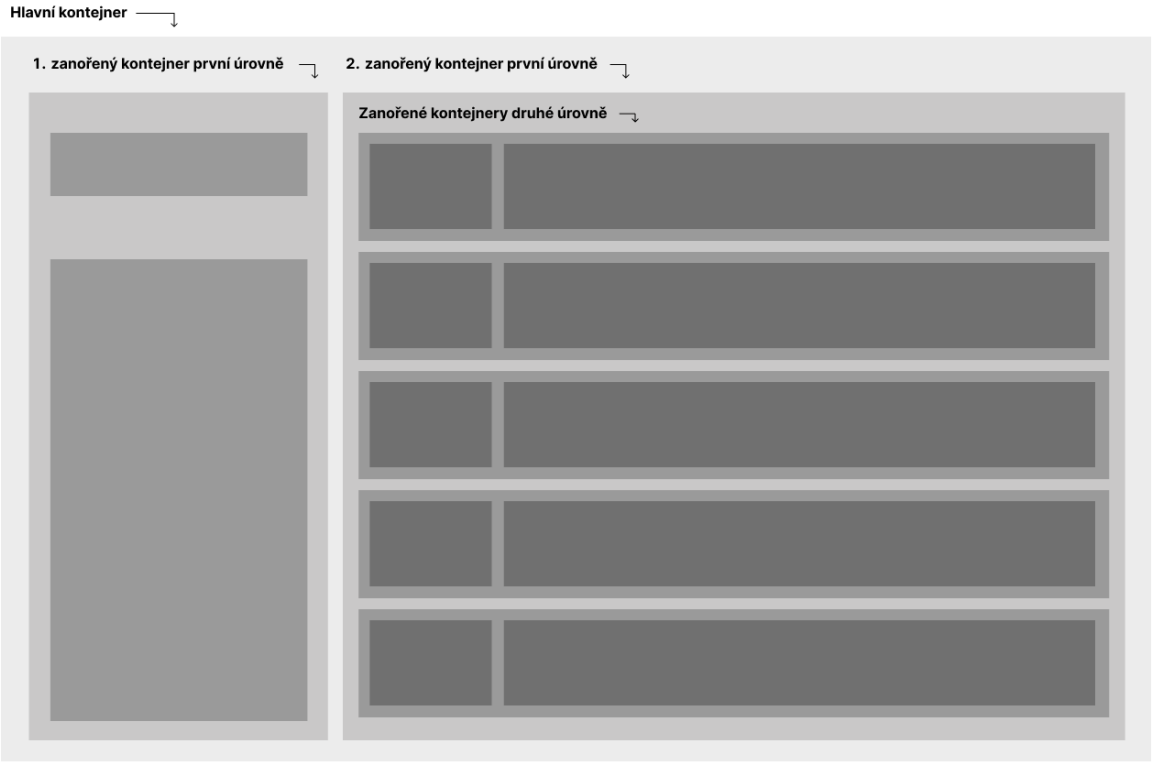


Obrázek 11 CSS Box Model [46]

V CSS existuje několik různých způsobů tvorby layoutů, které byly postupem času do CSS přidávány. Mezi dnes nejvyužívanější patří například:

- **Flexbox** – Jeho hlavní myšlenkou je dát kontejneru možnost měnit šířku a výšku svých položek tak, aby co nejlépe zaplnily dostupný prostor. Prvky mohou být přizpůsobeny tak, aby zabíraly dostupný prostor nebo se zmenšily tak, aby zapadly do prostoru, a to i když jejich přesná velikost není předem známá. Flexbox je určen pro „jednorozměrný layout“, tedy buď prvky skládá vodorovně anebo svisle. Pokud ale je nutné dosáhnout komplexnějšího layout systému, lze ho vytvořit postupným skládáním a zanořováním flexbox kontejnerů. Níže na obrázku je vidět, jak lze zanořování kontejnerů provádět. První box je hlavním flexbox kontejnerem, který obsahuje dva potomky umístěné vedle sebe. Tyto dva potomci jsou samy také flexbox kontejnery. První z nich obsahuje další dva potomky umístěné pod sebou. Druhý je pak obsahuje mnoho potomků, kteří jsou také umístěny pod sebou a každý z těchto potomků je opět zároveň kontejnerem, který drží dva potomky vedle sebe.
- **CSS Grid** – Grid, oproti flexboxu, poskytuje dvourozměrný systém rozložení, který umožňuje vytvářet komplexní layouty založené na řádcích a sloupcích. Tento model je obzvláště užitečný, když je potřeba vytvořit přesné a detailní rozložení

jako je například galerie obrázků či rozhraní aplikací. Jeho použití je však oproti flexbox layotu o poznání složitější.



Obrázek 12 Flexbox layout (vlastní obrázek)

3.6 JSON Web Token

JWT neboli JSON Web Token, je otevřený standard používaný k sdílení bezpečnostních informací mezi dvěma stranami, tedy klientem a serverem. Každý JWT obsahuje zakódované JSON objekty, včetně sady tvrzení (claims). JWT mohou být podepsány pomocí kryptografických algoritmů, aby se zajistilo, že informace nemohou být po vydání tokenu změněna. [47]

Nejčastější scénář použití JWT je autorizace. Jakmile je uživatel přihlášen, každý následující požadavek bude obsahovat JWT, což umožní uživateli přístup k serverovým endpointům, službám a zdrojům, které jsou s tímto tokenem povoleny. Umožněno je také tzv. jednotné přihlašování (Single Sign On), což je funkce, která uživateli umožňuje přihlašovat se k různým aplikacím pod jedním uživatelským účtem. Dalším možným využitím JWT je

bezpečný přenos informací mezi dvěma stranami, nikoliv však soukromý. Vzhledem k tomu, že JWT mohou být podepsány, je zajištěno, že odesílatelé jsou opravdu ti, za koho se vydávají. Navíc, jelikož je podpis vypočítán s využitím hlavičky a payloadu, je možné také ověřit, že obsah nebyl pozměněn. [47]

JSON Web Token je složen ze tří částí, které jsou odděleny tečkami a formát typicky vypadá takto: *xxxxx.yyyyy.zzzz*. Níže budou popsány části JWT [47]:

- **Header:** Header se obvykle skládá ze dvou částí, z typu tokenu a používaného podepisovacího algoritmu, jako je HMAC SHA256 nebo RSA.
- **Payload:** Payload obsahuje tvrzení (claims). Claims jsou prohlášení o entitě (obvykle uživateli) a další údaje.
- **Signature:** Podpis se používá k ověření, že zpráva nebyla po cestě změněna, a v případě tokenů podepsaných soukromým klíčem také k ověření, že odesílatel JWT je skutečně ten, za koho se vydává.

II. PRAKTICKÁ ČÁST

4 SBĚR POŽADAVKŮ

Vývoj softwarové aplikace často začíná velmi důležitou fází, kterou je sběr požadavků. Tato fáze je zásadní pro úspěch celého projektu, protože definuje, co má aplikace dělat, jaké problémy má řešit a jaké potřeby koncových uživatelů má splňovat. Typicky se během této fáze intenzivně komunikuje se zadavateli projektu a potenciálními budoucími uživateli a postupně se tak sbírají požadavky na vyvíjený software.

V této práci, jejíž výstupem je webová aplikace, konkrétně online komunikační skupina pro peer terapii, je zadavatelem samotný tým Nevyhasni. Ke sběru požadavků proběhlo několik konzultací týmu společně s odborníky na syndrom vyhoření, kteří zastupují jednu stranu uživatel. Prostřednictvím sociálních sítí se také komunikovalo s lidmi, kteří projevili o online skupinu zájem a zastupují tak druhou stranu potenciálních uživatelů.

4.1 Základní zadání projektu

V této podkapitole bude sepsána motivace, cíl a také zadání projektu ve velmi obecné podobě. Cílem tohoto obecného zadání je mít zdokumentované hlavní myšlenky od kterých se následně budou odvíjet nadefinované požadavky na vyvíjený software. Obecné zadání je tedy základním stavebním kamenem projektu.

4.1.1 Motivace a cíl projektu

Dle průzkumu Psychiatrické kliniky pražské Všeobecné fakultní nemocnice se zhruba každý čtvrtý Čech setká se syndromem vyhoření. Cílem Nevyhasni je dlouhodobě napomáhat ke snižování výskytu syndromu vyhoření v populaci a také tento problém dlouhodobě mapovat napříč Českou republikou. [48]

Výstup této práce, tedy online komunikační skupina pro peer terapii ve formě webové aplikace, je nedílnou součástí Nevyhasni. Cílem je tedy vytvořit platformu, která lidem potýkajícím se se syndromem vyhoření poskytne možnost setkávat se, sdílet své zkušenosti a vzájemně se podporovat s dalšími lidmi, kteří jsou v podobné situaci. A to bez ohledu na jejich geografickou polohu či časové možnosti. Vše pod záštitou zkušeného terapeuta, který bude skupinu moderovat.

4.1.2 Popis hlavních částí projektu

Online komunikační skupiny pro peer terapii se bude skládat z následujících částí:

1) Komunikační okno (chat)
    a. Hlavní komunikační okno
    b. Komunikační okno dané zprávy (vlákno)
2) Uživatelské účty
    a. Nastavení
    b. Změna hesla
3) Integrovaný dotazník
4) Panel pro moderátora skupiny

Ad 1) Webová aplikace bude obsahovat hlavní komunikační okno (chat) ve kterém se bude odehrávat komunikace mezi všemi účastníky skupiny. Dále bude komunikace rozšířena o možnost odpovídat na zprávy ve vláknech, tedy bude k dispozici okno, ve kterém bude probíhat komunikace v kontextu dané zprávy.

Ad 2) Webová aplikace bude disponovat uživatelskými účty, které budou rozděleny do dvou rolí, a to moderátor skupiny (typicky odborník na terapii) a uživatel, který se potýká se syndromem vyhoření. V rámci uživatelského účtu bude uživatelům umožněno svůj profil editovat a kdykoliv provést změnu hesla.

Ad 3) Webová aplikace bude obsahovat integrovaný dotazník ve formě chatbota. Tento dotazník bude primárně sloužit k prvotní identifikaci psychického stavu uživatelů online skupiny pro jejího moderátora, který tak získá základní přehled o jednotlivých členech skupiny.

Ad 4) Webová aplikace bude obsahovat panel pro moderátora skupiny. V tomto panelu bude moderátor schopen monitorovat aktivitu uživatelů skupiny jako například datum jejich posledního přihlášení či poslední odeslané zprávy. Také zde moderátorovi bude k dispozici okno pro odeslání speciální zprávy tzv. „oživováku“, který bude sloužit k oživení a správnému směrování konverzace. Tato speciální zpráva bude v konverzaci zvýrazněna a uživatelům skupiny bude také zaslána notifikace prostřednictvím e-mailu.

4.2 Funkcionální požadavky

Funkcionální požadavky definují chování a funkcionalitu vyvíjeného softwarového produktu. Zaznamenávají tedy co má aplikace umět, jaké problémy má řešit a jak se má v daných případech chovat. Funkcionální požadavky budou v rámci této kapitoly nadefinovány v tabulkách, na které bude vždy navazovat delší text, který bude tyto požadavky rozvíjet o důležité poznatky. Tyto doplňující texty budou umístěny pod každou tabulkou.

4.2.1 Obecné funkcionální požadavky

V této podkapitole budou specifikovány obecné funkcionální požadavky, které nadefinují, jakými částmi má aplikace disponovat. Funkcionální požadavky na konkrétní části aplikace pak budou blíže specifikovány dále v rámci kapitoly 4.2.

Tabulka 2 Obecné funkcionální požadavky

Identifikátor	**Požadavek**
FP1	Webová aplikace bude poskytovat uživatelské účty
FP2	Webová aplikace bude disponovat komunikačním rozhraním
FP3	Webová aplikace bude nabízet integrovaný dotazník
FP4	Webová aplikace bude poskytovat panel pro moderátora skupiny

Ad FP1) Webová aplikace bude poskytovat uživatelské účty rozdělené do dvou rolí, a to moderátor a uživatel potýkající se s problémem syndromu vyhoření. Bude umožněno vytvořit si uživatelský účet, přihlásit se, odhlásit se a upravovat uživatelský účet.

Ad FP2) Webová aplikace bude disponovat komunikačním rozhraním, které je jejím jádrem. Komunikační rozhraní je určeno pro skupinovou konverzaci, které se účastní moderátor a ostatní členové skupiny.

Ad FP3) Webová aplikace bude nabízet integrovaný dotazník pojat formou chatovacího robota. Po vyplnění bude dotazník odeslán a poskytnut odbornému moderátorovi skupiny.

Ad FP4) Webová aplikace bude poskytovat panel pro moderátora skupiny, který bude sloužit k monitorování aktivity uživatel skupiny a tvorbě speciálních moderátorských zpráv.

4.2.2 Funkcionální požadavky na uživatelské účty

Tabulka 3 Funkcionální požadavky na uživatelské účty

Identifikátor	Požadavek
FP1.1	Webová aplikace bude zahrnovat dvě uživatelské role
FP1.2	Webová aplikace bude uživateli umožňovat vytvořit uživatelský účet
FP1.3	Webová aplikace bude uživateli umožňovat přihlášení
FP1.4	Webová aplikace bude uživateli umožňovat odhlášení
FP1.5	Webová aplikace bude uživateli umožňovat upravit jeho účet
FP1.6	Webová aplikace bude uživateli umožňovat změnit přístupové údaje
FP1.7	Webová aplikace bude uživateli poskytovat možnost resetovat zapomenuté heslo

Ad FP1.1) Webová aplikace bude zahrnovat dvě uživatelské role. První rolí je moderátor skupiny, který je v rámci skupiny vždy pouze jeden. Druhou rolí je pak typický uživatel, tedy člen skupiny.

Ad FP1.2) Webová aplikace bude uživatelům umožňovat vytvářet uživatelské účty pomocí vyplnění a odeslání registračního formuláře. Nově registrovanému uživateli se také bude vytvářet svazek klíčů umožňující šifrovanou komunikaci, soukromé části klíčů pak budou uloženy v lokálním perzistentním úložišti a veřejné klíče budou publikovány na server.

Ad FP1.3) Webová aplikace bude uživatelům umožňovat přihlašovat se ke svým uživatelským účtům prostřednictvím formuláře. Po vyplnění a odeslání formuláře bude uživatel ověřen a v případě správnosti údajů přihlášen do online skupiny.

Ad FP1.4) Webová aplikace bude uživatelům umožňovat odhlásit se ze svých uživatelských účtů.

Ad FP1.5) Webová aplikace bude uživateli umožňovat upravit jeho uživatelský účet. Bude umožněno přidat vlastní celé jméno (či zůstat ve skupině přihlášen pod soukromější variantou, tedy přezdívkou). Dále bude umožněno uložit krátkou biografii, věk a také nahrát vlastní profilovou fotografii.

Ad FP1.6) Webová aplikace bude uživatelům poskytovat možnost změny jejich přístupových údajů, tedy e-mailové adresy a hesla.

Ad FP1.7) Webová aplikace bude uživatelům v případě zapomenutí hesla poskytovat možnost svoje heslo obnovit pomocí jejich e-mailové adresy.

4.2.3 Funkcionální požadavky na komunikační rozhraní

Tabulka 4 Funkcionální požadavky na komunikační rozhraní

Identifikátor	**Požadavek**
FP2.1	Webová aplikace bude uživateli umožňovat komunikaci mezi účastníky v hlavním vlákně
FP2.2	Webová aplikace bude uživateli umožňovat odpovědět na jakoukoliv zprávu
FP2.3	Webová aplikace bude uživatelům umožňovat odlišit speciální zprávy od moderátora v hlavním vlákně
FP2.4	Webová aplikace bude uživatele informovat o nových zprávách
FP2.5	Webová aplikace bude uživateli postupně načítat další zprávy
FP2.6	Webová aplikace bude rolovat oknem po odeslaní nové zprávy
FP2.7	Webová aplikace bude uživateli umožňovat zobrazit náhled profilu komunikujících uživatel

Ad FP2.1) Webová aplikace bude uživateli umožňovat skupinovou konverzaci mezi všemi účastníky v hlavním vlákně. Hlavní vlákno bude reprezentováno typickým „chatovacím“ oknem. Uživatelé odsud mohou odesílat zprávy, které se následně po přijetí v tomto okně zobrazí všem členům. Zprávy budou řazeny od nejstarší po nejnovější směrem ze shora dolů, tedy nové zprávy budou vždy nejníže. Aplikace bude také zajišťovat, že veškeré zprávy budou před odesláním uživatelem zašifrované. Uživatelem přijatá a zpracovaná zpráva bude uchovávána v rámci lokálního perzistentního úložiště.

Ad FP2.2) Webová aplikace bude uživateli umožňovat odesílat odpovědi na veškeré zprávy z hlavního vlákna. Tedy každá zpráva vyskytující se v hlavním vlákně bude zároveň novým

vláknem pro odpovědi na tuto zprávu. Tyto odpovědi budou zobrazovány v sekundárním okně, které bude uživatelům umožňovat zobrazovat a odesílat odpovědi. Dále bude aplikace zajišťovat, že všechny odpovědi budou před samotným odesláním uživatelem zašifrované. Příchozí uživatelem přijaté a zpracované odpovědi budou také ukládány v rámci lokálního úložiště.

Ad FP2.3) Webová aplikace bude uživatelům umožňovat vizuálně odlišit speciální zprávy od moderátora skupiny v hlavním komunikačním vlákně. Tyto speciální zprávy budou označeny jako „oživovák“ a oproti klasickým zprávám budou výraznější, aby byly snadno identifikovatelné.

Ad FP2.4) Webová aplikace bude uživatele informovat o nových zprávách prostřednictvím notifikačního okna, které se zobrazí ve spodní části hlavního komunikačního okna. Toto notifikační okno bude obsahovat informaci o počtu nově příchozích zpráv a bude poskytovat možnost rychlé navigace k novým zprávám.

Ad FP2.5) Webová aplikace bude umožňovat uživatelům plynulé načítání dalších zpráv v konverzaci. Když se bude uživatel posouvat v konverzaci nahoru, budou se mu postupně načítat starší zprávy.

Ad FP2.6) Webová aplikace bude rolovat komunikačním oknem po odeslání nové zprávy. Tedy okno se automaticky posune dolů tak, aby uživatel mohl pokračovat v konverzaci bez nutnosti manuálního posouvání.

Ad FP2.7) Webová aplikace bude uživateli umožňovat zobrazit náhled jakéhokoliv komunikujícího uživatele, a to přímo v hlavním či sekundární komunikačním okně. Náhled uživatelského profilu bude reprezentován malým oknem, které bude obsahovat informace vyplněné daným uživatelem (např. jméno, věk, povolání atp.).

4.2.4 Funkcionální požadavky na integrovaný dotazník

Tabulka 5 Funkcionální požadavky na integrovaný dotazník

Identifikátor	**Požadavek**
FP3.1	Webová aplikace bude poskytovat dotazník ve formě chat bota s dvěma typy otázek
FP3.2	Webová aplikace bude uživateli zobrazovat dané otázky postupně

FP3.3	Webová aplikace bude uživateli umožňovat odpovídat na otázky
FP3.4	Webová aplikace bude umožňovat upravovat odpovědi na otázky
FP3.5	Webová aplikace bude uživateli umožňovat odeslat dotazník

Ad FP3.1) Webová aplikace bude poskytovat dotazník, který bude realizován jako jednoduchý chat bot, který se uživatele bude ptát na dva typy různých otázek. První typ otázek bude reprezentován číselnou škálou a druhý typ otázek pak volnou textovou odpovědí. Dotazník bude mít vlastní komunikační okno podobné hlavnímu komunikačnímu oknu. Robot se nejprve uživateli představí a vysvětlí jakým způsobem na následující otázky odpovídat.

Ad FP3.2) Webová aplikace bude uživateli zobrazovat dané otázky postupně, tedy další otázku uživateli zobrazí až poté, co odpoví na předchozí.

Ad FP3.3) Webová aplikace bude uživateli umožňovat odpovídat na pokládané otázky. U prvního typu otázek výběrem hodnoty z číselné škály 1 až 7, kdy se tato škála bude vykreslovat přímo pod pokládanou otázkou. U druhého typu otázek pak bude uživatel odpovídat volnou textovou odpovědí.

Ad FP3.4) Webová aplikace bude uživateli umožňovat v průběhu času měnit odpovědi na již zodpovězené otázky.

Ad FP3.5) Webová aplikace bude uživateli umožňovat odeslat vyplněný dotazník po jeho dokončení, a to tak, že bude odeslána odpověď na poslední otázku dotazníku. Po odeslání robot bude uživatele informovat o úspěšném odeslání a vybídne ho k navrácení do hlavního komunikačního okna.

4.2.5 Funkcionální požadavky na panel pro moderátora

Tabulka 6 Funkcionální požadavky na panel pro moderátora

Identifikátor	**Požadavek**
FP4.1	Webová aplikace bude moderátorovi umožňovat sledovat aktivitu uživatel
FP4.2	Webová aplikace bude moderátorovi umožňovat odeslat speciální zprávu

FP4.3	Webová aplikace bude moderátorovi umožňovat zobrazit odpovědi z dotazníku

Ad FP4.1) Webová aplikace bude moderátorovi poskytovat možnost sledovat aktivitu všech účastníků. Data budou zobrazována v tabulce, kde každý řádek bude reprezentovat daného uživatele a mezi sledované prvky bude patřit datum posledního přihlášení, datum poslední odeslané zprávy a počet odeslaných zpráv.

Ad FP4.2) Webová aplikace bude moderátorovi umožňovat odesílat speciální zprávy, takzvané „oživováky“. Pokud moderátor odešle tuto zprávu, uživatelé budou notifikování prostřednictvím e-mailu. Úkolem těchto speciálních zpráv je, jak již název napovídá, oživit a směrovat diskusi mezi účastníky.

Ad FP4.3) Webová aplikace bude moderátorovi umožňovat zobrazit uživatelské odpovědi z integrovaného dotazníku. Pro přehlednost bude moderátor moci zobrazovat či skrývat detail těchto odpovědí.

4.3 Nefunkcionální požadavky

Nefunkcionální požadavky jsou očekávané vlastnosti systému které, oproti požadavkům nefunkcionálním, nejsou přímo spojeny s jeho funkcionalitou, ale ovlivňují jeho výkon, spolehlivost, bezpečnost nebo další aspekty.

Tabulka 7 Nefunkcionální požadavky

Identifikátor	**Požadavek**
NFP1	Webová aplikace bude splňovat kritéria responzivního designu
NFP2	Webová aplikace bude fungovat napříč všemi moderními prohlížeči
NFP3	Webový server bude implementován v technologii Node.js
NFP4	Webová aplikace bude implementována v technologii React
NFP5	Webová aplikace bude používat databázi MongoDB
NFP6	Webová aplikace bude používat IndexedDB jako lokální úložiště

NFP7	Webová aplikace bude ke komunikaci se serverem využívat HTTP a WS
NFP8	Webová aplikace bude zabezpečovat komunikaci prostřednictvím E2EE

Ad NFP1) Webová aplikace bude splňovat kritéria responzivního designu, tedy bude optimalizována pro použití na různých zařízeních, jako jsou počítače, tablety a mobilní telefony.

Ad NFP2) Webová aplikace bude fungovat napříč všemi moderními prohlížeči, což zahrnuje populární prohlížeče jako Chrome, Firefox či Safari.

Ad NFP3) Serverová část webové aplikace bude implementována s použitím technologie Node.js, která umožní vyvíjet server v jazyce JavaScript.

Ad NFP4) Klientská část webové aplikace bude implementována pomocí JavaScriptové knihovny React.

Ad NFP5) Webová aplikace bude k ukládání dat využívat nerelační dokumentovou databázi MongoDB.

Ad NFP6) Webová aplikace bude pro ukládání dlouhodobě potřebných lokálních dat využívat perzistentního úložiště IndexedDB.

Ad NFP7) Webová aplikace bude v rámci komunikace se serverem využívat protokol HTTP pro běžné požadavky a protokol WS (WebSocket) pro přenos zpráv v reálném čase.

Ad NFP8) Webová aplikace bude zabezpečovat veškerou komunikaci mezi uživateli (zprávy i odpovědi) prostřednictvím E2E šifrování, které bude implementováno pomocí kryptografického protokolu Signal Protocol.

5 NÁVRH APLIKACE

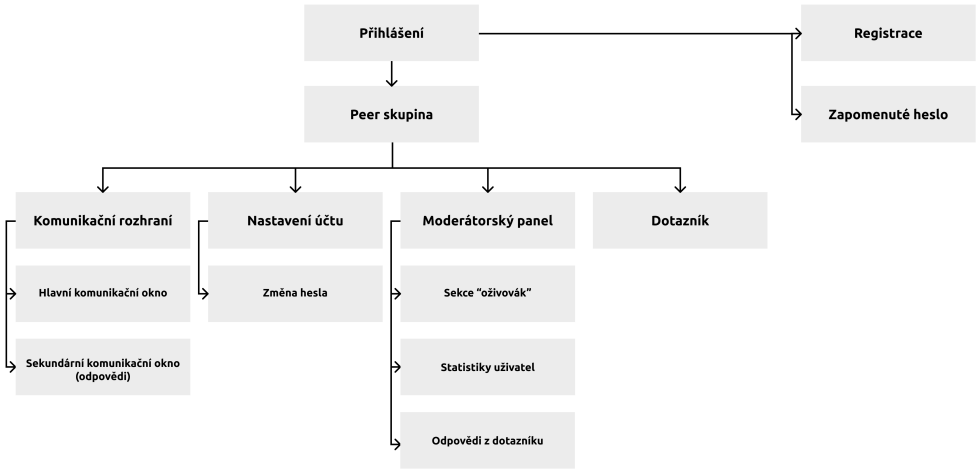
Návrh vyvíjené aplikace je další fází vývoje softwarového produktu a přímo navazuje na předchozí fázi, kde byly definovány funkcionální a nefunkcionální požadavky. Po jejich sběru je třeba tyto požadavky analyzovat, interpretovat a přeložit do konkrétních technických specifikací a designových rozhodnutí, které povedou k vytvoření plánovaného produktu.

5.1 Návrh informační architektury

V této podkapitole bude navržena a popsána informační architektura. Informační architektura představuje základní strukturu a organizaci informací v rámci webové aplikace. Toto hierarchické uspořádání určuje, jak jsou jednotlivé stránky propojeny mezi sebou a jakým způsobem jimi mohou uživatelé procházet. Jejím cílem je zajistit, že se uživatelé v aplikaci budou moci snadno a intuitivně navigovat, rychle nalézt požadované informace a splnit své požadované úkony.

5.1.1 Informační architektura

Navržená informační architektura vyvíjené webové aplikace je vidět na níže uvedeném schématu. V tomto schématu lze vidět jakým způsobem jsou jednotlivé části aplikace propojeny, tedy jak jsou všechny části uspořádány a jak na sebe navazují.



Obrázek 13 Informační architektura aplikace

5.2 Návrh drátěného modelu aplikace

Drátěný model aplikace slouží jako prvotní vizuální reprezentace, jenž přímo navazuje na navrženou informační architekturu. Jedná se o konceptuální náčrt, který zobrazuje rozložení jednotlivých prvků na stránce a způsoby navigace v aplikaci. Drátěný model poskytuje předběžnou představu o tom, jak bude aplikace vypadat a jak s ní budou moci uživatelé interagovat. Tento model poté slouží jako základ pro návrh uživatelského rozhraní a další vývoj aplikace.

V této podkapitole budou představeny veškeré části vyvíjené aplikace ve formě prvotního návrhu, tedy drátěné modely všech stránek. Tento návrh byl zhotoven společně s profesionálním UI/UX designérem z týmu Nevyhasni.

5.2.1 Základní stránky

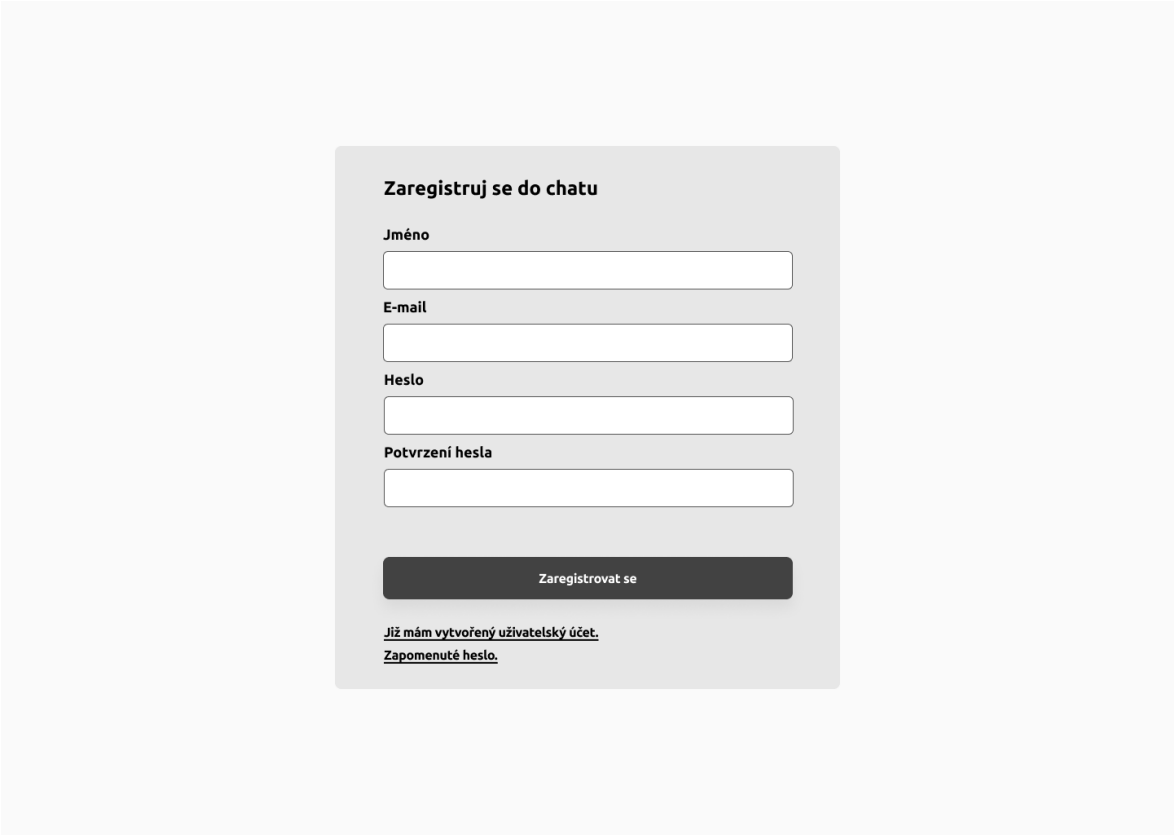
Níže budou představeny základní stránky vyvíjené webové aplikace, tedy:

- Stránka pro přihlášení
- Stránka pro registraci
- Stránka pro obnovení zapomenutého hesla.

Stránka pro přihlášení, vyobrazená na obrázku č. 14, obsahuje pouze přihlašovací formulář se vstupy pro e-mail a heslo, tlačítko a odkazy na prokliknutí na registrační stránku a stránku pro obnovu zapomenutého hesla. Tato stránka je vesměs velmi jednoduchá a neobsahuje žádné další prvky tak, aby uživatel nebyl ničím rušen a v aplikaci se jednoduše orientoval.

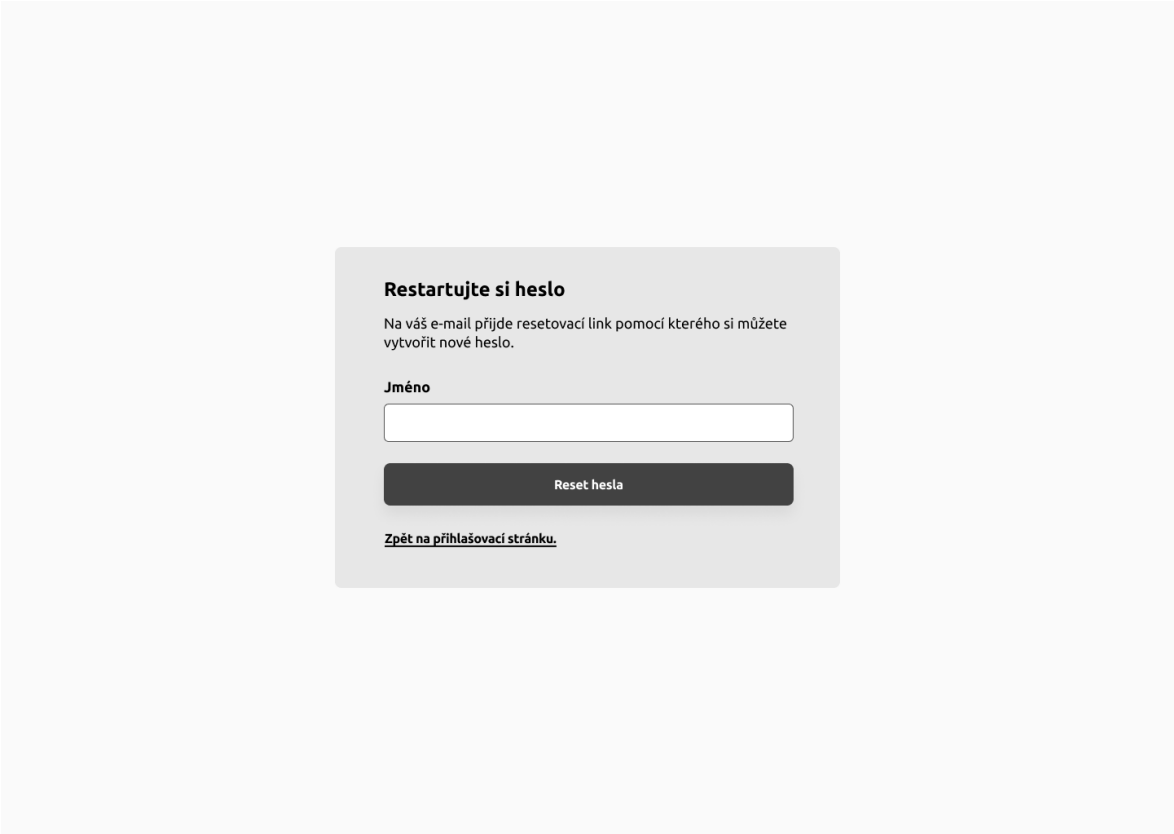
Obrázek 14 Wireframe: přihlašovací stránka

Stránka pro registraci, viditelná na obrázku č. 15, je opět velmi jednoduchá a obsahuje pouze formulář vstupy pro uživatelské jméno, e-mail, heslo, potvrzení hesla, tlačítko a odkazy na prokliknutí na stránku pro přihlášení a stránku pro obnovení zapomenutého hesla.



Obrázek 15 Wireframe: stránka pro registraci

Stránka pro obnovení zapomenutého hesla, na obrázku č. 16, je poslední z těchto jednoduchých formulářových stránek. Opět obsahuje formulář, který tentokrát disponuje pouze jedním vstupem pro e-mail, tlačítkem pro odeslání formuláře a odkaz na prokliknutí na stránku pro přihlášení.



Obrázek 16 Wireframe: stránka pro obnovu hesla

5.2.2 Hlavní rozhraní

Níže bude představena stránka, která je jádrem vyvíjené aplikace, tedy stránka s komunikačním rozhraním. Bude zde popsána postranní navigace, hlavní a sekundární komunikační okno a také náhled uživatelského profilu v komunikačním okně.

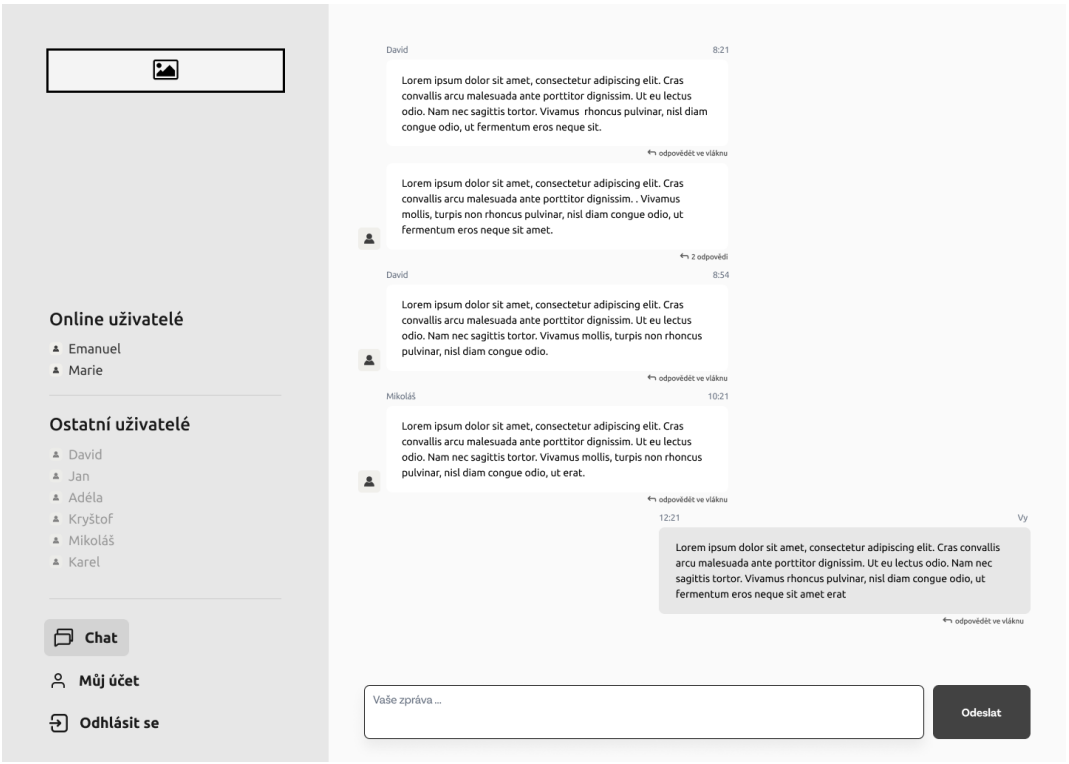
Hlavní stránka skupiny je zobrazena v různých situacích na obrázcích níže (obr. 17-19) a jedná se o hlavní stránku na kterou bude uživatel přesměrován po úspěšném přihlášení do aplikace. Na této stránce lze již vidět navržené uživatelské rozhraní aplikace, tedy navigační panel s různými prvky vlevo a na pravé straně se pak nachází komunikační okno.

Navigační panel ve vrchní části obsahuje prostor pro logo. Pod logem lze vidět dva seznamy uživatelů, první seznam představuje momentálně přihlášené uživatelé a druhý seznam představuje ostatní uživatelé, tedy nepřítomné uživatele. Pod seznamy uživatel lze vidět tři navigační položky. První položka, tedy „Chat“, představuje současnou stránku, proto je také v navigaci zvýrazněna. Druhá položka, označená jako „Můj účet“, poté představuje stránku s uživatelským účtem, kde lze provádět různé úpravy. Jako poslední z položek je „Odhlásit se“, které slouží k odhlášení z aplikace.

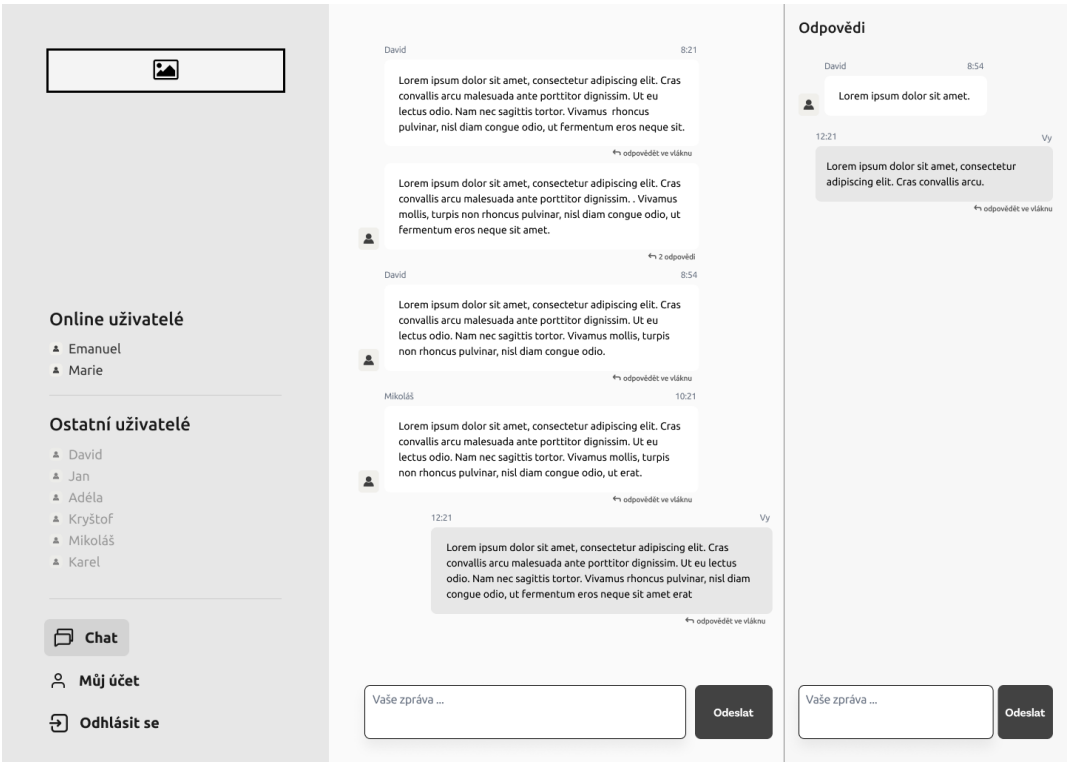
Hlavní komunikační okno je možné rozdělit do dvou částí, a to do části se zprávami a části s vstupem pro odeslání nové zprávy. Příchozí zprávy od členů skupiny se zobrazují na levé straně a zprávy odeslané od současně přihlášeného uživatele se pak tomuto uživateli zobrazují na straně pravé. Nad jednotlivými zprávami lze vidět jméno uživatele a čas odeslání. Pokud se jedná o zprávu příchozí, na levé straně je také fotografie tohoto uživatele. Pod každou zprávou je pak text „Odpovědět ve vláknu“, který představuje možnost na tuto zprávu odpovědět. Pokud však pro tuto zprávu již existují odpovědi, je text nahrazen počtem odpovědí. U svých zpráv uživatel nevidí fotografii a informační text nad zprávou je otočen, tedy na levé straně je čas odeslání a na pravé je text „Vy“. Část se vstupem pro odeslání zprávy je pak velmi jednoduchá a jedná se pouze o vstupní pole a tlačítko pro odeslání.

Sekundární komunikační okno viditelné na obrázku č. 18 na pravé straně obrazovky. Po otevření tohoto okna se zmenší hlavní okno a poskytne sekundárnímu oknu prostor. Je velmi podobné hlavnímu oknu s tím, že je menší a nejsou zde již možnosti odpovídat.

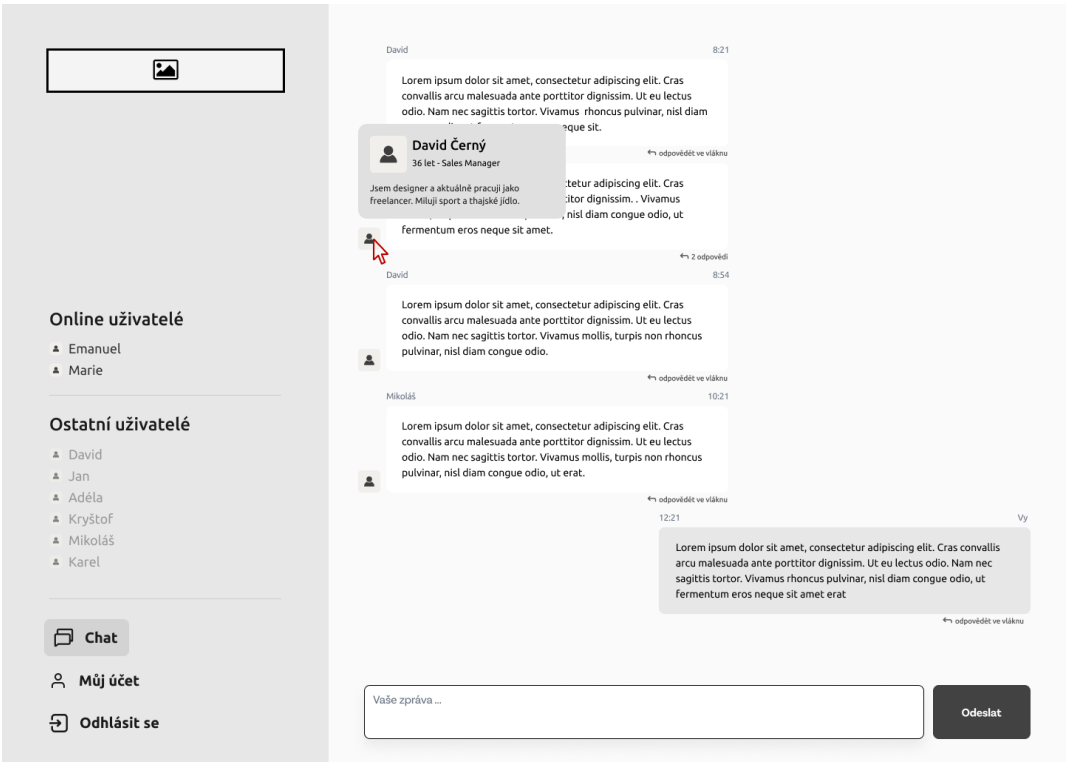
Náhled uživatelského profilu, který lze vidět na obrázku č. 19, je realizován jako malé okénko s informacemi o daném uživateli jako je jméno, povolání, věk, krátká biografie a fotografie. Jak je nastíněno na obrázku, tento náhled bude zobrazen po najetí kurzorem na fotografii vedle zprávy.



Obrázek 17 Wireframe: Hlavní stránka s komunikačním oknem



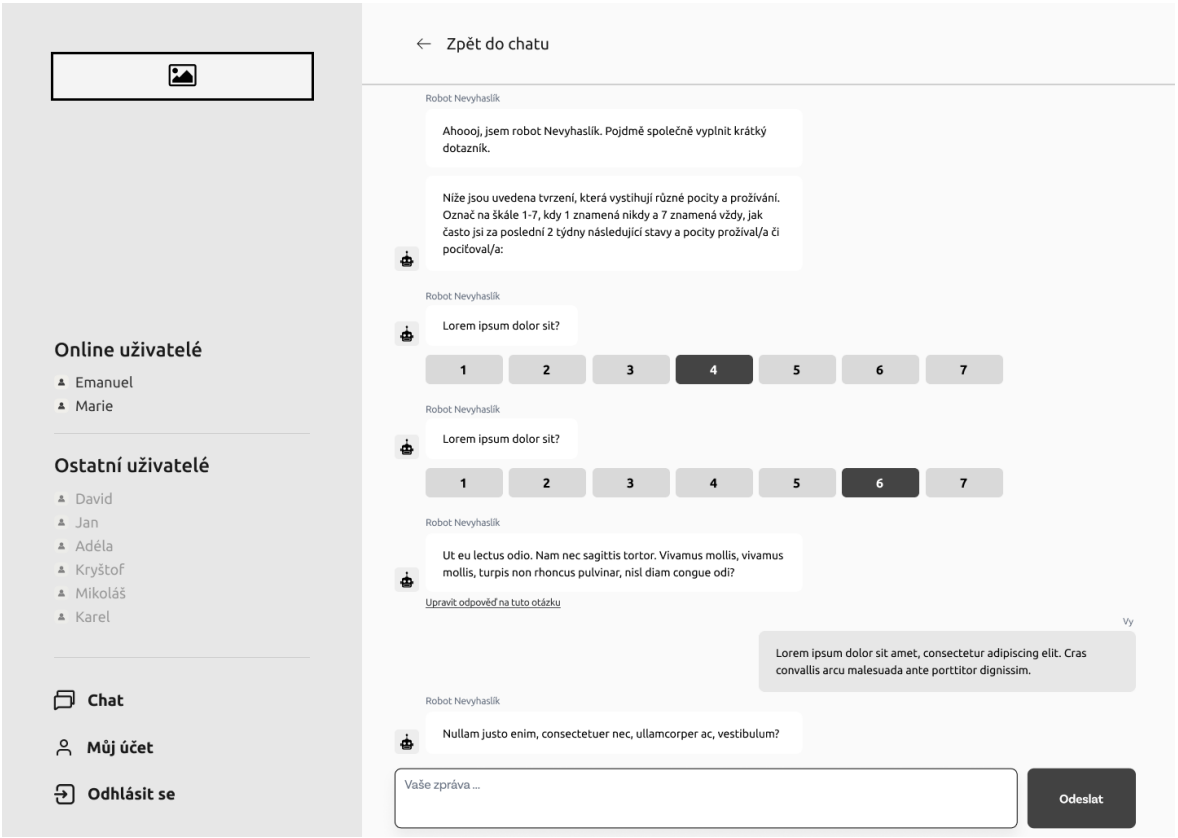
Obrázek 18 Wireframe: Hlavní stránka se sekundárním komunikačním oknem



Obrázek 19 Wireframe: Hlavní stránka s náhledem profilu

5.2.3 Integrovaný dotazník

Níže bude představen drátěný model stránky, která představuje integrovaný dotazník ve formě chatbota (viz. obrázek 20). Stejně jako u hlavní stránky s komunikačním oknem, i zde zůstává stejná navigace na levé straně obrazovky. V postranní navigaci se odkaz na tento dotazník nenachází, protože na dotazník se půjde dostat pouze skrze speciální pop-up okno. Ve vrchní části obrazovky také přibyla lišta pro možnost navigovat se zpět. Dotazník je tedy realizován v podobě komunikačního okna, kde komunikuje robot s aktuálně přihlášeným uživatelem. Robot pokládá dva typy otázek, a to otázky s číselnou odpovědí v intervalu 1 – 7 a otázky s volnou textovou odpovědí. U prvního zmíněného typu otázek odešle robot s otázkou rovnou i možnost odpovědí, tedy sadu tlačítek s hodnoty od 1 do 7. Jakmile uživatel kliknutím na jedno z tlačítek odpoví, robot pokládá další otázku. U druhého typu, tedy u volných odpovědí, uživatel komunikuje s robotem prostřednictvím odesílání zpráv stejně jako u klasické komunikace. Pod zodpovězenou otázkou druhého typu se také nachází text „Upravit odpověď na tuto otázku“ pomocí kterého bude možné odpověď zpětně upravovat.



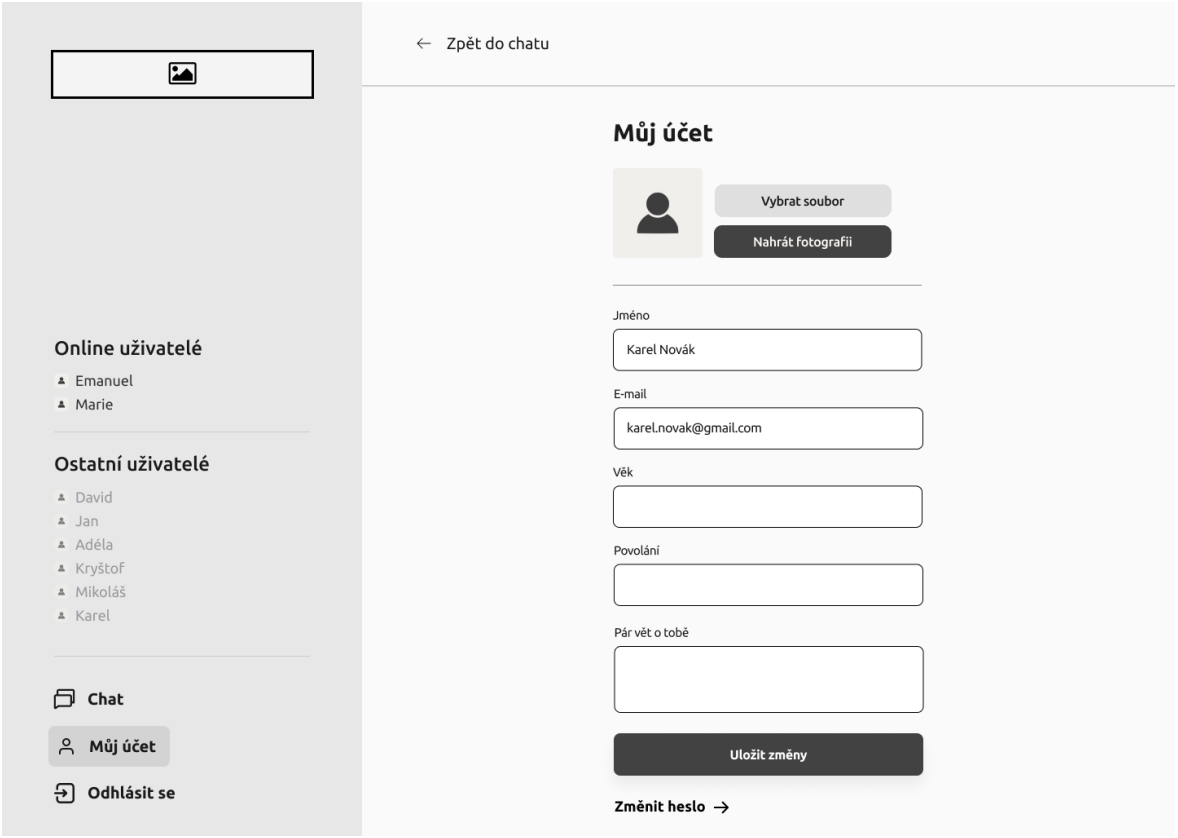
Obrázek 20 Wireframe: Stránka s integrovaným dotazníkem

5.2.4 Uživatelské nastavení

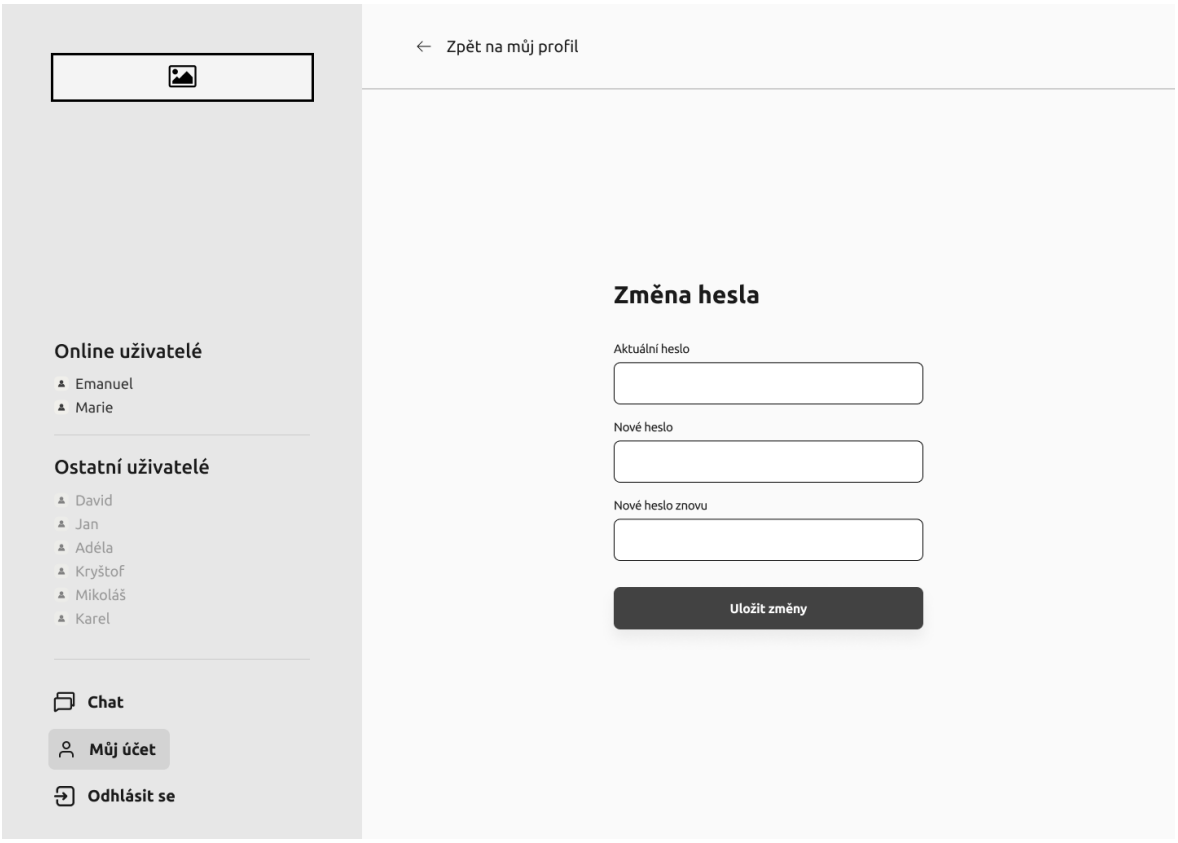
Na níže budou představeny stránky, které umožňují uživateli upravovat některé jeho údaje, nahrávat profilovou fotografii či měnit heslo.

Stránka pro nastavení uživatelského účtu je demonstrována na obrázku č. 21. Postranní a vrchní navigace zůstává opět stejná. Nastavení obsahuje sekci, kde je možné změnit si profilový obrázek, kdy se pomocí tlačítka „Vybrat soubor“ otevře dialogové okno pro výběr obrázku a pomocí tlačítka „Nahrát fotografii“ se změna aplikuje. V druhé části se pak nachází formulář, kde uživatel může vyplnit údaje jako je jméno, email, věk, povolání či krátkou biografii. Pod formulářem je také vidět odkaz, který uživatele po kliknutí přesměruje na stránku pro změnu hesla.

Stránku pro změnu hesla lze vidět na obrázku č. 22. Opět se jedná o podobné rozhraní jako u předchozí popisované stránky. Stránka obsahuje pouze formulář pro změnu hesla se třemi poli pro staré heslo, nové heslo a potvrzení nového hesla.



Obrázek 21 Wireframe: Nastavení uživatelského účtu



Obrázek 22 Wireframe: Stránka pro změnu hesla

5.2.5 Moderátorský panel

Na níže uvedeném obrázku (obr. 23) lze vidět drátěný model stránky pro moderátora. Jak je vidět v postranní navigaci, oproti ostatním stránkám zde přibyla i položka „Moderátor“. Tato položka bude v aplikaci přístupná pouze moderátorům. Moderátorský panel se dělí do tří sekcí: pole pro speciální zprávy (oživováky), statistika uživatel a odpovědi z dotazníku.

Sekce pro tvorbu speciální zprávy, tedy „oživováku“, je první sekcí na této stránce. Jedná se o velký textový vstup, informační text a tlačítko pro odeslání.

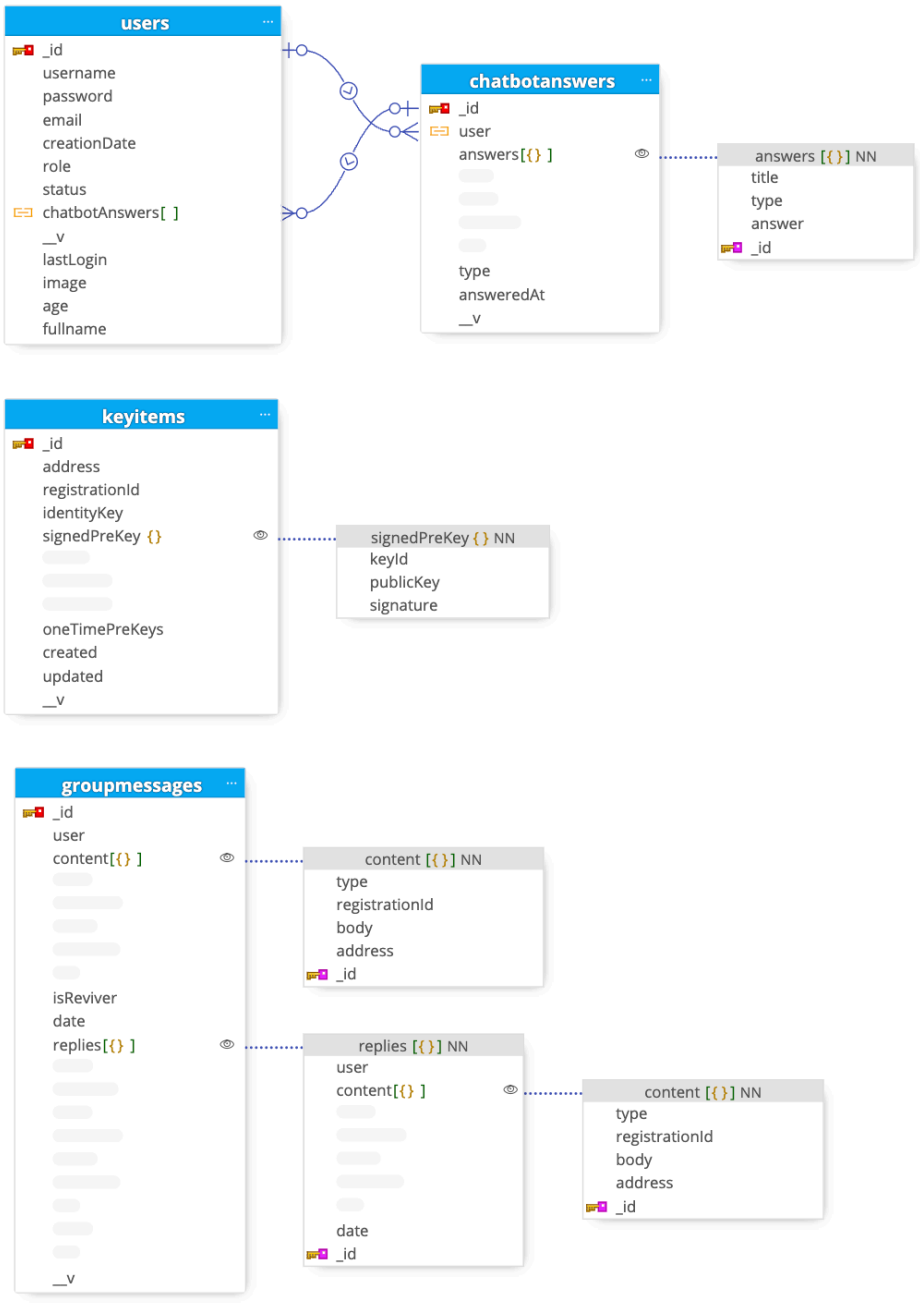
Sekce se statistikami uživatel je pak reprezentována tabulkou uživatelů a obsahuje primárně údaje o aktivitě uživatel. Kromě jména a emailu uživatele je zde informace o datu posledním přihlášení, poslední odeslané zprávě a počet všech odeslaných zpráv.

Sekce s odpověďmi z dotazníku obsahuje box, kde každý box reprezentuje sadu odpovědí uživatele. Na pravé straně boxu se nachází tlačítko, které po kliknutí zobrazí či schová odpovědi daného uživatele. Pokud je box nerozbalený, tlačítko obsahuje text „Zobrazit detail“, pokud je tomu naopak, tlačítko obsahuje text „Schovat detail“.

Obrázek 23 Wireframe: Moderátorský panel

5.3 Návrh datového modelu

V této podkapitole bude popsán datový model vyvíjené webové aplikace. Jedná se o návrh pro aplikaci, jenž pracuje výhradě s databází nerelačního dokumentového typu, konkrétně s MongoDB. Tento typ databází je založen na odlišném principu oproti tradičním relačním databázím a umožňuje v rámci kolekcí zanořovat pole či objekty. Na níže uvedeném obrázku (obr. 24), který prezentuje datový model navržený v aplikaci Moon Modeler, budou kolekce označené modře a jejich zanořené pole a objekty šedivě.



Obrázek 24 Datový model aplikace

Jak je z obrázku patrné, model obsahuje čtyři hlavní kolekce, a to users, chatbotanswers, keyitems a groupmessages, tyto kolekce pak většinou obsahují i nějaký zanořený objekt či pole. Níže následuje popis těchto kolekcí i zanořených objektů a polí:

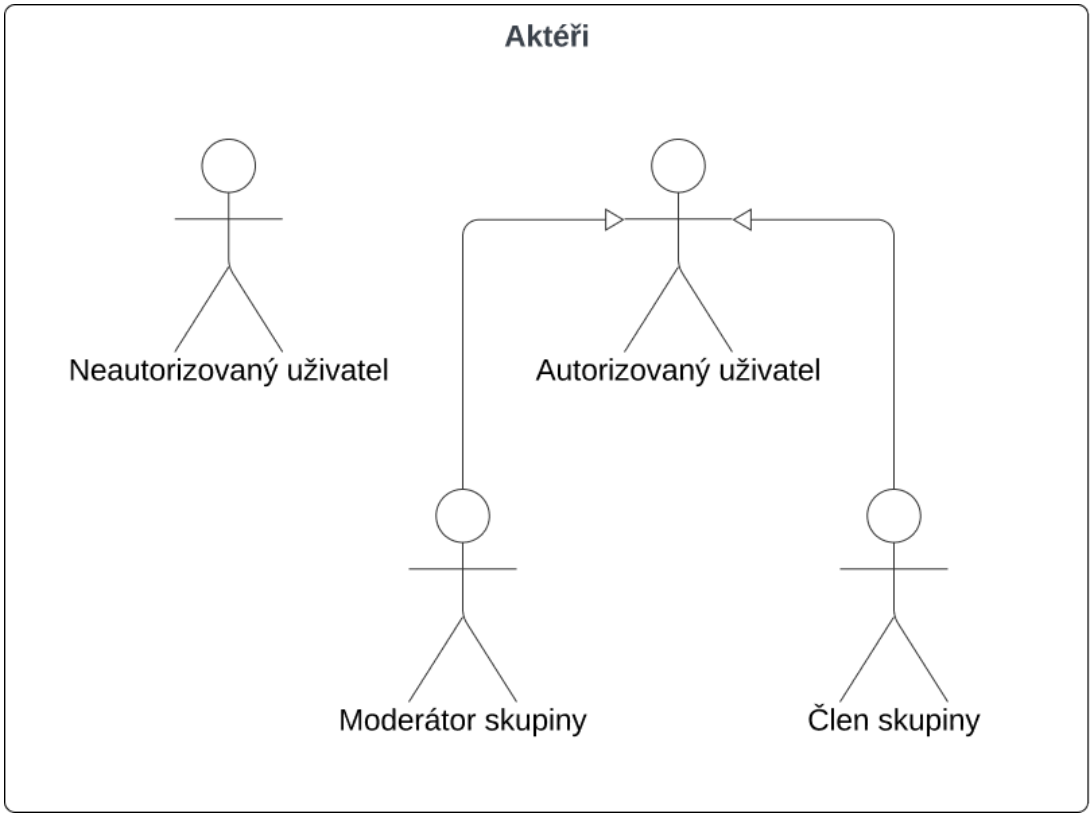
- **users** – Kolekce pro ukládání veškerých uživatelských dat, kromě klíčových svazků, které jsou ukládány v rámci jiné kolekce.
- **chatbotanswers** – Kolekce pro ukládání odpovědí na integrovaný chatbot dotazník. Tato kolekce je napojena na uživatelskou kolekci *users* a obsahuje zanořené pole *answers*, které drží výhradě pouze otázku, její odpověď a typ.
- **keyitems** – Kolekce pro ukládání klíčových svazků uživatel podle definicí kryptografického protokolu Signal Protocol. Tato kolekce obsahuje zanořený objekt *signedPreKey*, která obsahuje veřejný klíč a podpis.
- **groupmessages** – Kolekce pro ukládání veškerých odeslaných zpráv. Jak je vidět, tato kolekce má zanořené pole *content*, které drží obsah zprávy v zašifrované podobě pro různé uživatele. Dalším zanořeným polem je *replies*, což je pole pro ukládání odpovědí na danou zprávu. A stejně jako zprávy, i odpovědi mají zanořené pole *content*, které obsahuje odpovědi v zašifrované podobě.

5.4 Aktéři a případy užití

V rámci této podkapitoly budou popsáni všichni aktéři vyvíjené webové aplikace. Dále bude demonstrováno, jak se aplikace chová z pohledu uživatele pomocí případů užití. Veškeré níže uvedené diagramy byly vytvořeny pomocí online nástroje LucidChart.

5.4.1 Aktéři aplikace

Na níže uvedeném diagramu lze vidět veškeré aktéry aplikace, tedy všechny typy budoucích uživatel systému.



Obrázek 25 Aktéři webové aplikace

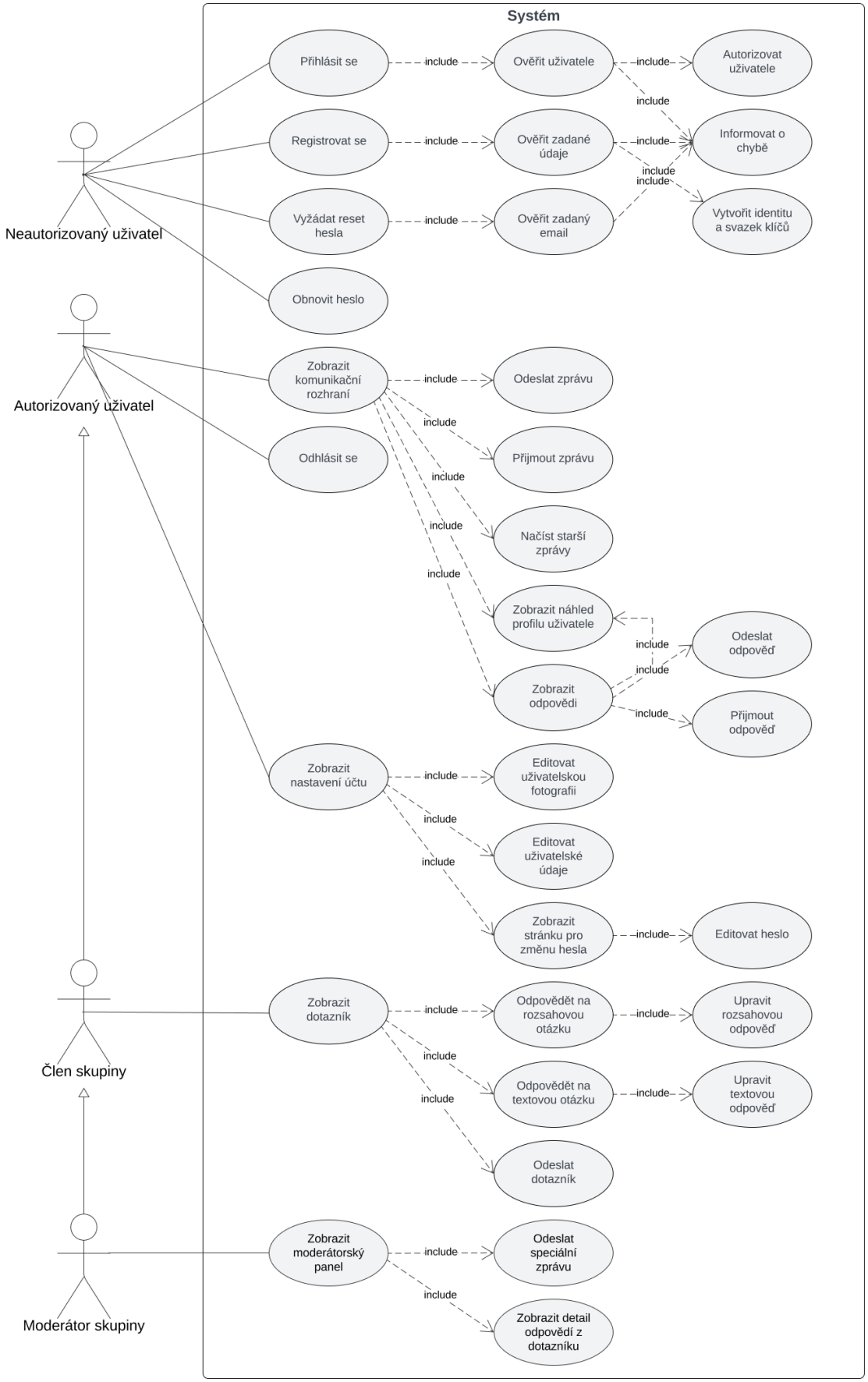
Nyní budou všichni uvedení aktéři podrobněji popsáni:

- **Neautorizovaný uživatel** – Jedná se o návštěvníka aplikace, který v aplikaci není registrován nebo přihlášen. Skupina je uzavřený systém přístupný pouze autorizovaným uživatelům, tudíž jediné, co může dělat je vytvořit si účet, přihlásit se či zažádat o obnovení zapomenutého hesla.

- **Autorizovaný uživatel** – Jedná se o uživatele, které je již v aplikaci registrován a přihlášen. Autorizovaný uživatel může komunikovat prostřednictvím komunikačního rozhraní, odpovídat na zprávy v rámci vláken, editovat svůj uživatelský profil, měnit své heslo a zobrazovat náhled profilů ostatních členů. Tento uživatel pak může v aplikaci provádět další různé úkony na základě jeho přiřazené role, které se od autorizovaného uživatele odvíjejí.
- **Člen skupiny** – Je základním uživatelem skupiny a jedná se o uživatele, který se v životě potýká se syndromem vyhoření. Je odvozen od autorizovaného uživatele, tedy dědí všechny jeho možnosti. Kromě těchto vlastností může vyplňovat integrovaný dotazník.
- **Moderátor skupiny** – Je dalším uživatelem skupiny, který v aplikaci zastává roli moderátora. Moderátorem je člověk, který má zkušenosti s pomocí lidem, kteří prochází syndromem vyhoření a často se jedná o odborného terapeuta. Také je odvozen od autorizovaného uživatele a dědí veškeré jeho vlastnosti. Kromě toho má k dispozici moderátorský panel a může tedy využívat veškeré jeho funkce. Jedná se o možnost odesílat speciální zprávy („oživováky“), zobrazovat statistiky členů skupiny a také číst odpovědi z integrovaného dotazníku.

5.4.2 Diagram případů užití

Níže, na obrázku 26, bude uveden diagram případů užití, což je nástroj používaný v softwarovém inženýrství pro zachycení funkcionálních požadavků systému a interakce mezi různými uživateli a systémem. Slouží k vizualizaci, jak uživatelé, zde aktéři, interagují se systémem za účelem dosažení konkrétních cílů. Tento diagram poskytuje strukturovaný přehled o tom, co systém musí provádět, aniž by specifikoval, jak tyto funkce budou implementovány. Je klíčový pro pochopení hranic systému a definování rozsahu projektu z pohledu uživatelských interakcí.



Obrázek 26 Diagram případů užití

5.5 Uživatelské scénáře

V návaznosti na navržený diagram případů užití budou v rámci této kapitoly popsány uživatelské scénáře. Uživatelské scénáře jsou dokumentovány s cílem popsat konkrétní interakce mezi uživatelem a systémem za účelem dosažení specifického cíle. Každý scénář detailně vysvětluje postup, jakým jsou vykonávány dané úkoly, a tedy jakým způsobem je dosaženo daného cíle krok za krokem.

5.5.1 Autorizace a související scénáře

V této podkapitole budou zdokumentovány veškeré uživatelské scénáře pro neautorizované aktéry, tedy například scénáře týkající se autorizace, registraci či resetu hesla.

Tabulka 8 [U001] Registrace

[UC001] Registrace		
Charakteristika: Dokumentace procesu registrace, autorizace a vytvoření identity nového uživatele		
Aktér: Neautorizovaný uživatel		
Vstupní podmínky: Aktérovi je zobrazena stránka pro registraci		
Výstupní podmínky: Je vytvořen uživatelský účet Uživatel je autorizován Je vytvořena a publikována identita a svazek klíčů uživatele		
Krok	**Aktér/Systém**	**Popis**
1	Aktér	Aktér vyplní zadá údaje do vstupních polí
2	Aktér	Aktér klikne na tlačítko „Zaregistrovat se“
3	Systém	Systém provede validaci zadaných dat
4	Systém	Systém autorizuje uživatele
5	Systém	Systém vytvoří identitu a svazek klíčů nového uživatele
6	Systém	Systém uloží identitu a soukromé klíče do lokálního uložiště
7	Systém	Systém publikuje identitu a veřejné klíče na server

8	Systém	Systém informuje uživatele o úspěšné registraci
Alternativní scénáře:		
UC001a – Alternativní scénář: Neúspěšná validace vstupních dat		

Tabulka 9 [U001a] Neúspěšná validace vstupních dat

Alternativní scénář: [UC001a] Neúspěšná validace vstupních dat		
Charakteristika: Zachycení situace, kdy byla do formuláře zadána neplatná data		
Krok	**Aktér/Systém**	**Popis**
4	Systém	Systém zachytí neplatné údaje
5	Systém	Systém zobrazí hlášku s chybou

Tabulka 10 [UC002] Přihlášení

[UC002] Přihlášení		
Charakteristika: Dokumentace procesu přihlášení a navazující autorizace		
Aktér: Neautorizovaný uživatel		
Vstupní podmínky: Aktérovi je zobrazena stránka pro přihlášení		
Výstupní podmínky: Aktér je autorizován Aktér ověří existující sessions či vytvoří nové		
Krok	**Aktér/Systém**	**Popis**
1	Aktér	Aktér vyplní zadá údaje do vstupních polí
2	Aktér	Aktér klikne na tlačítko „Přihlásit se“
3	Systém	Systém provede validaci zadaných dat
4	Systém	Systém autorizuje uživatele
5	Systém	Systém ověří, zda existují platné sessions pro další uživatele
6	Systém	Systém odvodí symetrický klíč pro šifrování lokálního úložiště
7	Systém	Systém přesměruje uživatele na hlavní stránku
Alternativní scénáře:		

UC002a – Alternativní scénář: Neúspěšná validace zadaných dat
UC002b – Alternativní scénář: Některé sessions nebyly platné či chyběly

Tabulka 11 [UC002a] Neúspěšná validace zadaných dat

Alternativní scénář: [UC002a] Neúspěšná validace zadaných dat		
Charakteristika: Zachycení situace, kdy byla do formuláře zadána neplatná data		
Krok	**Aktér/Systém**	**Popis**
4	Systém	Systém zachytí neplatné údaje
5	Systém	Systém zobrazí hlášku s chybou

Tabulka 12 [UC002b] Některé sessions nebyly platné či chyběly

Alternativní scénář: [UC002b] Některé sessions nebyly platné či chyběly		
Charakteristika: Zachycení situace, kdy některé sessions ještě nebyly vytvořeny či nebyly platné		
Krok	**Aktér/Systém**	**Popis**
6	Systém	Systém vytvoří nové sessions
7	Systém	Systém nové sessions uloží
8	Systém	Systém přesměruje uživatele na hlavní stránku

Tabulka 13 [UC003] Vyžádání obnovení hesla

[UC003] Vyžádání obnovení hesla		
Charakteristika: Dokumentace procesu vyžádání obnovení zapomenutého hesla		
Aktér: Neautorizovaný uživatel		
Vstupní podmínky: Aktérovi je zobrazena stránka pro obnovení zapomenutého hesla		
Výstupní podmínky: Aktérovi je zaslán odkaz pro změnu hesla		
Krok	**Aktér/Systém**	**Popis**
1	Aktér	Aktér zadá email do vstupního pole

2	Aktér	Aktér klikne na tlačítko „Reset hesla“
3	Systém	Systém ověří platnost emailové adresy a existenci uživatele
4	Systém	Systém odešle email s odkazem na zadanou adresu
5	Systém	Systém zobrazí hlášku o úspěšně odeslaném emailu
Alternativní scénáře: UC003a – Alternativní scénář: Neúspěšné ověření emailové adresy		

Tabulka 14 [UC003a] Neúspěšné ověření emailové adresy

Alternativní scénář: [UC003a] Neúspěšné ověření emailové adresy		
Charakteristika: Zachycení situace, kdy emailová adresa nebyla platná či uživatel s touto emailovou adresou neexistuje		
Krok	**Aktér/Systém**	**Popis**
4	Systém	Systém zachytí neplatnou emailovou adresu
5	Systém	Systém zobrazí hlášku s chybou

Tabulka 15 [UC004] Obnovení hesla

[UC004] Obnovení hesla		
Charakteristika: Dokumentace procesu obnovení zapomenutého hesla		
Aktér: Neautorizovaný uživatel		
Vstupní podmínky: Aktérovi si úspěšně vyžádal obnovení zapomenutého hesla (UC003)		
Výstupní podmínky: Aktérovi je obnoveno heslo		
Krok	**Aktér/Systém**	**Popis**
1	Aktér	Aktér načte stránku z emailu pomocí odkazu
2	Aktér	Aktér vyplní vstupní pole pro heslo a jeho potvrzení
3	Aktér	Aktér klikne na tlačítko „Obnovit heslo“
4	Systém	Systém ověří zadaná hesla
5	Systém	Systém uloží nové heslo
6	Systém	Systém informuje aktéra o úspěšně obnoveném hesle

Alternativní scénáře:
UC004a – Alternativní scénář: Neúspěšné ověření zadaných hesel

Tabulka 16 [UC004a] Neúspěšné ověření zadaných hesel

Alternativní scénář: [UC004a] Neúspěšné ověření zadaných hesel		
Charakteristika: Zachycení situace, kdy se zadaná hesla neshodují nebo heslo nesplňuje podmínky		
Krok	**Aktér/Systém**	**Popis**
5	Systém	Systém zachytí neplatné údaje
6	Systém	Systém zobrazí hlášku s chybou

5.5.2 Komunikační rozhraní

V této podkapitole budou popsány veškeré uživatelské scénáře, které přímo souvisejí s činností uživatel v rámci komunikačního rozhraní.

Tabulka 17 [UC005] Zobrazení rozhraní pro komunikaci

[UC005] Zobrazení rozhraní pro komunikaci		
Charakteristika: Dokumentace procesu zobrazení rozhraní pro komunikaci		
Aktér: Autorizovaný uživatel		
Vstupní podmínky: -		
Výstupní podmínky: Aktérovi je zobrazeno rozhraní pro komunikaci a načteny zprávy (nepřečtené z databáze, přečtené z lokálního úložiště) Nově přečtené zprávy jsou uloženy do aktérova lokálního uložiště		
Krok	**Aktér/Systém**	**Popis**
1	Aktér	Aktér klikne na položku „Chat“ v navigaci
2	Systém	Systém přesměruje aktéra na stránku s rozhraním pro komunikaci
3	Systém	Systém načte nepřečtené zprávy z databáze
4	Systém	Systém načte přečtené zprávy z lokálního uložiště
5	Systém	Systém dešifruje nepřečtené zprávy v rámci E2EE
6	Systém	Systém dešifruje zprávy z lokálního úložiště pomocí aktérova symetrického klíče
7	Systém	Systém zobrazí dešifrované nepřečtené i přečtené zprávy v rozhraní
8	Systém	Systém zašifruje nově přečtené zprávy symetrickým klíčem aktéra pro uložení do lokálního úložiště
9	Systém	Systém uloží šifrované zprávy do lokálního úložiště aktéra

Tabulka 18 [UC006] Odeslání zprávy

[UC006] Odeslání zprávy
Charakteristika:

Dokumentace procesu odeslání zprávy v hlavním komunikačním okně		
Aktér: Autorizovaný uživatel		
Vstupní podmínky: Aktér se musí nacházet na stránce s komunikačním rozhraním (UC005)		
Výstupní podmínky: Zpráva je odeslána uživatelům skupiny Zpráva je přijata a zobrazena uživatelům skupiny Přijatá zpráva je uložena do lokálního úložiště		
Krok	**Aktér/Systém**	**Popis**
1	Aktér	Aktér napíše zprávu do vstupního pole
2	Aktér	Aktér klikne na tlačítko „Odeslat“
3	Systém	Systém zašifruje zprávu pro všechny uživatele
4	Systém	Systém odešle zašifrovanou zprávu všem uživatelům
5	Systém	Systém obdrží zprávu v zašifrované podobě
6	Systém	Systém dešifruje a zobrazí aktérovi přijatou zprávu
7	Systém	Systém zašifruje zprávu symetrickým klíčem aktéra pro uložení do lokálního úložiště
8	Systém	Systém uloží šifrovanou zprávu do lokálního úložiště aktéra

Tabulka 19 [UC007] Načtení starších zpráv

[UC007] Načtení starších zpráv		
Charakteristika: Dokumentace procesu postupného načítání starších zpráv		
Aktér: Autorizovaný uživatel		
Vstupní podmínky: Aktér se musí nacházet na stránce s komunikačním rozhraním (UC005)		
Výstupní podmínky: Aktérovi jsou načteny starší zprávy		
Krok	**Aktér/Systém**	**Popis**
1	Aktér	Aktér se posune pomocí scrollu do vrchní části komunikačního okna
2	Systém	Systém načte další starší zprávy

Tabulka 20 [UC008] Zobrazení odpovědí

[UC008] Zobrazení odpovědí		
Charakteristika: Dokumentace procesu zobrazení odpovědí na konkrétní zprávu (sekundární komunikační okno)		
Aktér: Autorizovaný uživatel		
Vstupní podmínky: Aktér se musí nacházet na stránce s komunikačním rozhraním (UC005)		
Výstupní podmínky: Aktérovi je zobrazeno sekundární komunikační okno a načteny odpovědi (nepřečtené z databáze, přečtené z lokálního úložiště) Nově přečtené odpovědi jsou uloženy do aktérova lokálního uložiště		
Krok	**Aktér/Systém**	**Popis**
1	Aktér	Aktér klikne na tlačítko „Opovědět ve vlákně“ pod konkrétní zprávou
2	Systém	Systém zobrazí sekundární komunikační okno
3	Systém	Systém načte nepřečtené odpovědi z databáze
4	Systém	Systém načte přečtené odpovědi z lokálního uložiště
5	Systém	Systém dešifruje nepřečtené odpovědi v rámci E2EE
6	Systém	Systém dešifruje odpovědi z lokálního úložiště pomocí aktérova symetrického klíče
7	Systém	Systém zobrazí nepřečtené i přečtené odpovědi v rozhraní
8	Systém	Systém zašifruje nově přečtené odpovědi symetrickým klíčem aktéra pro uložení do lokálního úložiště
9	Systém	Systém uloží šifrované odpovědi do lokálního úložiště aktéra

Tabulka 21 [UC009] Zobrazení náhledu uživatelského profilu

[UC009] Zobrazení náhledu uživatelského profilu
Charakteristika: Dokumentace procesu zobrazení náhledu uživatelského profilu konkrétního uživatele
Aktér: Autorizovaný uživatel
Vstupní podmínky:

Aktér se musí nacházet na stránce s komunikačním rozhraním (UC005) nebo v sekundárním komunikačním okně (UC008)		
Výstupní podmínky: Aktérovi je zobrazen náhled uživatelského profilu konkrétního uživatele		
Krok	**Aktér/Systém**	**Popis**
1	Aktér	Aktér najede kurzorem myši na profilovou fotografii uživatele umístěnou vedle jeho zprávy
2	Systém	Systém zobrazí náhled uživatelského profilu

5.5.3 Nastavení uživatelského účtu

V této podkapitole budou nadefinovány veškeré uživatelské scénáře, které souvisejí s panelem pro nastavení uživatelského účtu.

Tabulka 22 [UC010] Zobrazení uživatelského nastavení

[UC010] Zobrazení uživatelského nastavení		
Charakteristika: Dokumentace procesu zobrazení stránky pro nastavení uživatelského profilu		
Aktér: Autorizovaný uživatel		
Vstupní podmínky: -		
Výstupní podmínky: Aktér je přesměrován na stránku pro nastavení uživatelského účtu		
Krok	**Aktér/Systém**	**Popis**
1	Aktér	Aktér klikne na položku „Můj účet“ v navigaci
2	Systém	Systém aktéra přesměruje na stránku pro nastavení uživatelského účtu

Tabulka 23 [UC011] Editace uživatelských údajů

[UC011] Editace uživatelských údajů		
Charakteristika: Dokumentace procesu editace uživatelských údajů		
Aktér: Autorizovaný uživatel		
Vstupní podmínky: Aktér se musí nacházet na stránce pro nastavení uživatelského profilu (UC010)		
Výstupní podmínky: Aktérovi jsou upraveny uživatelské údaje		
Krok	**Aktér/Systém**	**Popis**
1	Aktér	Aktér vyplní vstupní pole novými údaji
2	Aktér	Aktér klikne na tlačítko „Uložit změny“
3	Systém	Systém provede validaci zadaných údajů

4	Systém	Systém uloží nové údaje
Alternativní scénáře:		
UC011a – Alternativní scénář: Neúspěšné ověření nových údajů		

Tabulka 24 [UC011a] Neúspěšné ověření nových údajů

[UC011a] Neúspěšné ověření nových údajů		
Charakteristika:		
Dokumentace situace, kdy aktér zadal neplatné údaje		
Krok	**Aktér/Systém**	**Popis**
4	Systém	Systém zachytí chybu
5	Systém	Systém informuje aktéra o konkrétní chybě

Tabulka 25 [UC012] Editace uživatelské fotografie

[UC012] Editace uživatelské fotografie		
Charakteristika:		
Dokumentace procesu editace uživatelské fotografie		
Aktér:		
Autorizovaný uživatel		
Vstupní podmínky:		
Aktér se musí nacházet na stránce pro nastavení uživatelského profilu (UC010)		
Výstupní podmínky:		
Aktérovi je nastavena nová uživatelská fotografie		
Krok	**Aktér/Systém**	**Popis**
1	Aktér	Aktér klikne na tlačítko „Vybrat soubor“
2	Systém	Systém otevře dialogové okno pro výběr souboru
3	Aktér	Aktér vybere požadovaný obrázek
4	Systém	Systém ověří, zdali je vybraný soubor obrázkem
5	Systém	Systém zobrazí nově vybraný obrázek
6	Aktér	Aktér klikne na tlačítko „Nahrát fotografii“
7	Systém	Systém uloží novou uživatelskou fotografii
Alternativní scénáře:		
UC012a – Alternativní scénář: Špatný typ vybraného souboru		

Tabulka 26 [UC012a] Špatný typ vybraného souboru

[UC012a] Špatný typ vybraného souboru		
Charakteristika: Dokumentace situace, kdy aktér vybral soubor, který není obrázek		
Krok	**Aktér/Systém**	**Popis**
5	Systém	Systém zjistí, že byl vybrán špatný typ souboru
6	Systém	Systém informuje aktéra o chybě

Tabulka 27 [UC013] Zobrazení stránky pro změnu hesla

[UC013] Zobrazení stránky pro změnu hesla		
Charakteristika: Dokumentace procesu zobrazení stránky pro změnu hesla		
Aktér: Autorizovaný uživatel		
Vstupní podmínky: Aktér se musí nacházet na stránce pro nastavení uživatelského profilu (UC010)		
Výstupní podmínky: Aktérovi je přesměrován na stránku pro změnu hesla		
Krok	**Aktér/Systém**	**Popis**
1	Aktér	Aktér klikne na odkaz „Změnit heslo“ pod formulářem
2	Systém	Systém aktéra přesměruje na stránku pro změnu hesla

Tabulka 28 [UC014] Změna hesla

[UC014] Změna hesla		
Charakteristika: Dokumentace procesu změny hesla		
Aktér: Autorizovaný uživatel		
Vstupní podmínky: Aktér se musí nacházet na stránce pro změnu hesla (UC013)		
Výstupní podmínky: Aktérovi je změněno heslo		
Krok	**Aktér/Systém**	**Popis**

1	Aktér	Aktér vyplní vstupní pole pro staré, nové heslo a potvrzení
2	Aktér	Aktér klikne na tlačítko „Uložit změny“
3	Systém	Systém provede kontrolu, zdali je zadané staré heslo správné a zdali se nové heslo shoduje s potvrzením hesla
4	Systém	Systém provede změnu hesla a nové heslo uloží
5	Systém	Systém dešifruje lokální zprávy a odpovědi symetrickým klíčem aktéra
6	Systém	Systém vytvoří nový klíč odvozený z nového hesla
7	Systém	Systém zašifruje lokální zprávy a odpovědi novým klíčem
8	Systém	Systém uloží zašifrované zprávy do lokálního úložiště
Alternativní scénáře:		
UC014a – Alternativní scénář: Neúspěšná kontrola hesel		

Tabulka 29 [UC014a] Neúspěšná kontrola hesel

[UC014a] Neúspěšná kontrola hesel		
Charakteristika:		
Dokumentace situace, kdy aktér zadal špatné staré heslo či zadal chybné potvrzení nového hesla		
Krok	**Aktér/Systém**	**Popis**
5	Systém	Systém zjistí, že bylo zadáno špatné heslo
6	Systém	Systém informuje aktéra o konkrétní chybě

5.5.4 Integrovaný dotazník

V této podkapitole budou zdokumentovány všechny uživatelské scénáře, které se odehrávají na stránce s integrovaným chatbot dotazníkem.

Tabulka 30 [UC015] Zobrazení integrovaného dotazníku

[UC015] Zobrazení integrovaného dotazníku		
Charakteristika: Dokumentace procesu zobrazení integrovaného dotazníku ve formě chatbota		
Aktér: Člen skupiny		
Vstupní podmínky: -		
Výstupní podmínky: Aktér je přesměrován na stránku s integrovaným uživatelským dotazníkem		
Krok	**Aktér/Systém**	**Popis**
1	Systém	Systém zobrazí aktérovi pop-up okno
2	Aktér	Aktér klikne na tlačítko „Vyplnit dotazník“ v pop-up okně
3	Systém	Systém přesměruje aktéra na stránku s dotazníkem

Tabulka 31 [UC016] Odeslání odpovědi na rozsahovou otázku

[UC016] Odeslání odpovědi na rozsahovou otázku		
Charakteristika: Dokumentace procesu odeslání odpovědi na otázku, na níž je možné odpovědět pouze v rozsahu čísel 1 až 7.		
Aktér: Člen skupiny		
Vstupní podmínky: Aktér se musí nacházet na stránce s integrovaným dotazníkem (UC015)		
Výstupní podmínky: Nová odpověď na danou otázku je aktérem odeslána a uložena		
Krok	**Aktér/Systém**	**Popis**
1	Systém	Systém zobrazí rozsahovou otázku s možnostmi odpovědí
2	Aktér	Aktér klikne na jednu z možností pod otázkou (čísla 1-7)

3	Systém	Systém uloží novou odpověď na tuto otázku
4	Systém	Systém v komunikačním okně zobrazí odeslanou odpověď a další otázku

Tabulka 32 [UC017] Odeslání odpovědi na textovou otázku

[UC017] Odeslání odpovědi na textovou otázku		
Charakteristika: Dokumentace procesu odeslání odpovědi na otázku, na níž je možné odpovědět pouze textem jako na typickou zprávu.		
Aktér: Člen skupiny		
Vstupní podmínky: Aktér se musí nacházet na stránce s integrovaným dotazníkem (UC015)		
Výstupní podmínky: Nová odpověď na danou otázku je aktérem odeslána a uložena		
Krok	**Aktér/Systém**	**Popis**
1	Systém	Systém zobrazí textovou otázku
2	Aktér	Aktér zapíše odpověď do vstupního pole
3	Aktér	Aktér klikne na tlačítko „Odeslat“
4	Systém	Systém uloží novou odpověď na tuto otázku
5	Systém	Systém v komunikačním okně zobrazí odeslanou odpověď a další otázku

Tabulka 33 [UC018] Odeslání vyplněného dotazníku

[UC018] Odeslání vyplněného dotazníku		
Charakteristika: Dokumentace procesu odeslání vyplněného dotazníku		
Aktér: Člen skupiny		
Vstupní podmínky: Aktér se musí nacházet na stránce s integrovaným dotazníkem (UC015)		
Výstupní podmínky: Vyplněný dotazník je odeslán a uložen		
Krok	**Aktér/Systém**	**Popis**

1	Aktér	Aktér odpoví na poslední otázku
2	Systém	Systém odešle a uloží dotazník
3	Systém	Systém informuje aktéra o úspěšném dokončení dotazníku

Tabulka 34 [UC019] Úprava rozsahové odpovědi

[UC019] Úprava rozsahové odpovědi		
Charakteristika: Dokumentace procesu úpravy odpovědi na rozsahovou otázku (interval 1 až 7)		
Aktér: Člen skupiny		
Vstupní podmínky: Aktér se musí nacházet na stránce s integrovaným dotazníkem (UC015) Aktér se musí mít zodpovězenou alespoň jednu otázku tohoto typu		
Výstupní podmínky: Odpověď na konkrétní rozsahovou otázku je změněna		
Krok	**Aktér/Systém**	**Popis**
1	Aktér	Aktér klikne na jednu z možností (1-7) pod danou otázkou
2	Systém	Systém uloží novou odpověď na tuto otázku

Tabulka 35 [UC020] Úprava textové odpovědi

[UC020] Úprava textové odpovědi		
Charakteristika: Dokumentace procesu úpravy odpovědi na textovou otázku		
Aktér: Člen skupiny		
Vstupní podmínky: Aktér se musí nacházet na stránce s integrovaným dotazníkem (UC015) Aktér se musí mít zodpovězenou alespoň jednu otázku tohoto typu		
Výstupní podmínky: Odpověď na konkrétní textovou otázku je změněna		
Krok	**Aktér/Systém**	**Popis**
1	Aktér	Aktér klikne na tlačítko „Upravit odpověď na tuto otázku“ pod vybranou otázkou

2	Systém	Systém zobrazí jeho původní odpověď ve vstupním poli
3	Aktér	Aktér upraví svou odpověď ve vstupním poli
4	Aktér	Aktér klikne na tlačítko „Odeslat“
5	Systém	Systém uloží novou odpověď na tuto otázku a vymění ji za starou v komunikačním okně

5.5.5 Moderátorský panel

V této podkapitole budou popsány uživatelské scénáře pro aktéra *Moderátor skupiny* a jedná se tak o scénáře spjaté s moderátorským panelem.

Tabulka 36 [UC021] Zobrazení moderátorského panelu

[UC021] Zobrazení moderátorského panelu		
Charakteristika: Dokumentace procesu zobrazení moderátorského panelu		
Aktér: Moderátor skupiny		
Vstupní podmínky: -		
Výstupní podmínky: Aktér je přesměrován na stránku s moderátorským panelem		
Krok	**Aktér/Systém**	**Popis**
1	Aktér	Aktér klikne na položku „Moderátor“ v navigaci
2	Systém	Systém aktéra přesměruje na stránku s moderátorským panelem

Tabulka 37 [UC022] Odeslání speciální zprávy

[UC022] Odeslání speciální zprávy
Charakteristika: Dokumentace procesu odeslání speciální zprávy označované jako „oživovák“
Aktér: Moderátor skupiny

Vstupní podmínky: Aktér se musí nacházet na stránce s moderátorským panelem (UC021)		
Výstupní podmínky: Speciální zpráva je odeslána uživatelům skupiny		
Krok	**Aktér/Systém**	**Popis**
1	Aktér	Aktér vyplní vstupní pole pro speciální zprávu
2	Aktér	Aktér klikne na tlačítko „Přidat a odeslat oživovák“
3	Systém	Systém zašifruje speciální zprávu pro všechny uživatele
4	Systém	Systém odešle zašifrovanou speciální zprávu uživatelům
5	Systém	Systém odešle členům skupiny email s informací o nové speciální zprávě

Tabulka 38 [UC023] Zobrazení detailu odpovědí z dotazníku

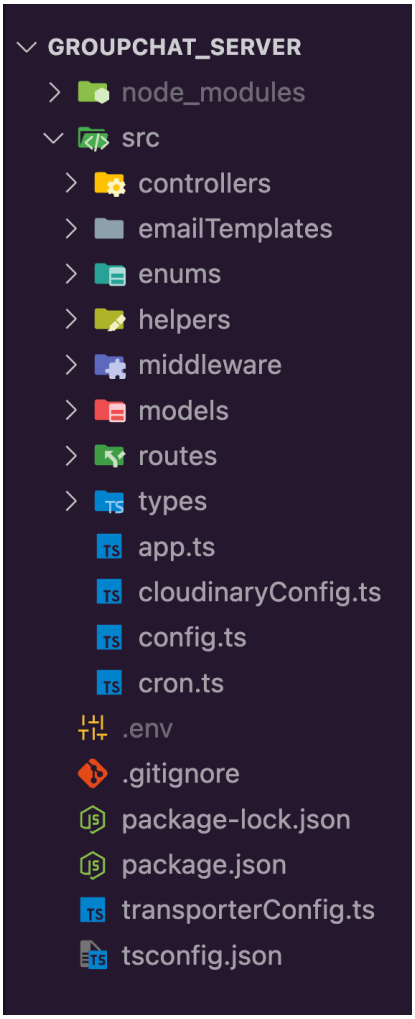
[UC023] Zobrazení detailu odpovědí z dotazníku		
Charakteristika: Dokumentace procesu zobrazení detailu odpovědí z dotazníku od konkrétního uživatele		
Aktér: Moderátor skupiny		
Vstupní podmínky: Aktér se musí nacházet na stránce s moderátorským panelem (UC021)		
Výstupní podmínky: Aktérovi je zobrazen detail odpovědí z dotazníku od daného uživatele		
Krok	**Aktér/Systém**	**Popis**
1	Aktér	Aktér v sekci „Odpovědi z dotazníku“ klikne na tlačítko „Zobrazit detail“ u konkrétního uživatele
2	Systém	Systém zobrazí detail odpovědí vybraného uživatele

6 STRUKTURA IMPLEMENTOVANÉ APLIKACE

V rámci této kapitoly bude popsána struktura adresáře implementované webové aplikace, která je důležitá pro pochopení organizace a správy kódu. Webová aplikace se dělí na dvě části, klientskou část a serverovou část a vlastně se tak jedná o dvě separátní a spolu komunikující aplikace, kdy každá z nich má vlastní adresář. Serverová část poskytuje API endpointy pro různé úkony a WebSocket endpoint pro veškerou komunikaci mezi členy skupiny. Klientská část pak se serverem pomocí odesílání požadavků na tyto endpointy se serverem komunikuje.

6.1 Adresářová struktura serveru

Níže bude detailněji popsána struktura serverové části aplikace, která je implementována v technologii Node.js a ExpressJS, které byly popsány v teoretické části. Přehled adresářové struktury serveru je vidět na obrázku č. 27.



Obrázek 27 Sturktura adresáře serveru

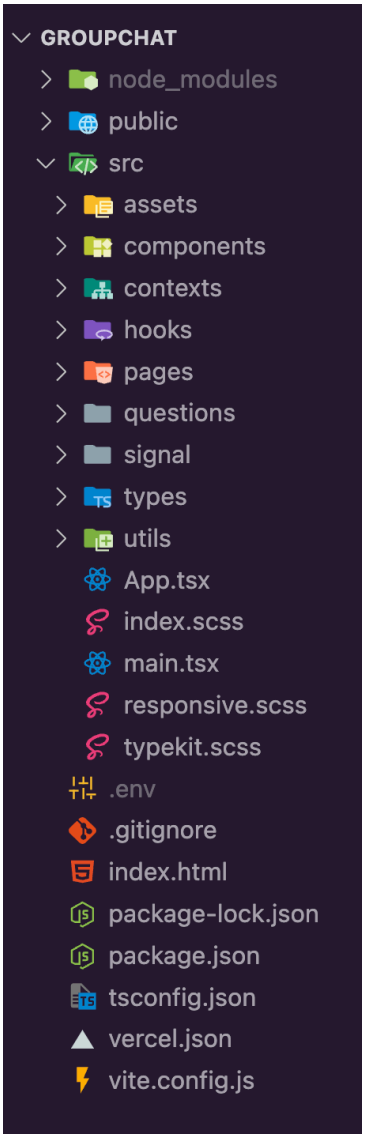
Následuje detailnější popis všech částí serveru:

- **Adresář src** – Tento adresář obsahuje veškeré zdrojové soubory serveru, kromě *node_modules*, *package.json* a některých konfiguračních souborů.
- **Adresář controllers** – Zde se nacházejí soubory, které obsluhují logiku pro zpracování požadavků od klientů. Kontroléry rozhodují, co se stane, když uživatel pošle požadavek na určité endpointy API.
- **Adresář emailTemplates** – Obsahuje šablony pro e-maily, které aplikace používá k odesílání e-mailů uživatelům, například pro odeslání notifikace o nové zprávě od moderátora skupiny.
- **Adresář enums** – Zde jsou definovány enum typy (enumerations), které pomáhají organizovat kód tím, že poskytují soubor pevně definovaných konstant, jež jsou používány v různých částech aplikace.
- **Adresář helpers** – Tento adresář obsahuje pomocné funkce, které jsou používány napříč celou aplikací. Tyto funkce mohou zahrnovat logiku pro formátování dat, validace a další běžné úlohy.
- **Adresář middleware –** Adresář middleware obsahuje middlware funkce, které jsou zde používány pro zpracování požadavků předtím, než dosáhnou koncových bodů (endpointů), například pro ověření uživatele.
- **Adresář models** – Obsahuje definice schémat a modelů v Mongoose pro MongoDB databázi. Tyto modely určují strukturu dat a jsou klíčové pro interakci s databází.
- **Adresář routes** – Definuje cesty (routes), které server rozpoznává. Každá cesta je spojena s kontrolérem, který určuje, jak bude na požadavek reagováno.
- **Adresář types** – Může obsahovat definice TypeScript typů nebo rozhraní (interface), které jsou používány pro zajištění typové bezpečnosti v aplikaci.
- **Soubor app.ts** – Hlavní soubor aplikace, který inicializuje server, nastavuje middleware, routy a další základní konfigurace. Také se zde nachází obslužný kód pro manipulaci s WebSocket endpointem.
- **Soubor cloudinaryConfig.ts** – Konfigurační soubor pro Cloudinary, službu pro správu médií, která umožňuje ukládat a spravovat obrázky a jiný obsah. V aplikaci slouží výhradně pro ukládání uživatelských fotografií.

- **Soubor config.ts** – Obsahuje globální konfigurační nastavení aplikace.
- **Soubor cron.ts** – Soubor pro plánování a spouštění pravidelných úloh (cron jobs). Je zde pouze připraven pro budoucí rozšiřování aplikace a není zatím využíván.
- **Soubor .env** - Skrytý soubor, který obsahuje hodnoty jako jsou například tajné klíče nebo databázové přístupy, které nejsou zahrnuty přímo do kódu.
- **Soubor transporterConfig.ts** – Konfigurační soubor pro nastavení nodemailer transportera, který se používá pro odesílání e-mailů uživatelům.

6.2 Adresářová struktura webového klienta

Níže bude blíže zdokumentována struktura klientské části aplikace, která je implementována pomocí knihovny React popsané v rámci teoretické části. Základní přehled adresářové struktury React webové aplikace je nastíněn na obrázku č. 28.



Obrázek 28 Struktura adresáře klienta

Následuje detailnější popis všech částí klienta:

- **Adresář public** – Typicky zde mohou být uloženy soubory, jako jsou ikony, favicon, a další statické assety. Obsah tohoto adresáře se nezpracovává build systémem a je servírován přímo, což znamená, že je ideální pro umístění statického obsahu, který by měl být rychle a efektivně dostupný bez potřeby dalšího zpracování.
- **Adresář src** – Hlavní adresář obsahující zdrojové soubory pro klientskou část aplikace. Tento adresář obsahuje všechny komponenty, kontexty, hooky, a další zdroje potřebné pro aplikaci. Neobsahuje *node_modules*, *package* a konfigurační soubory.

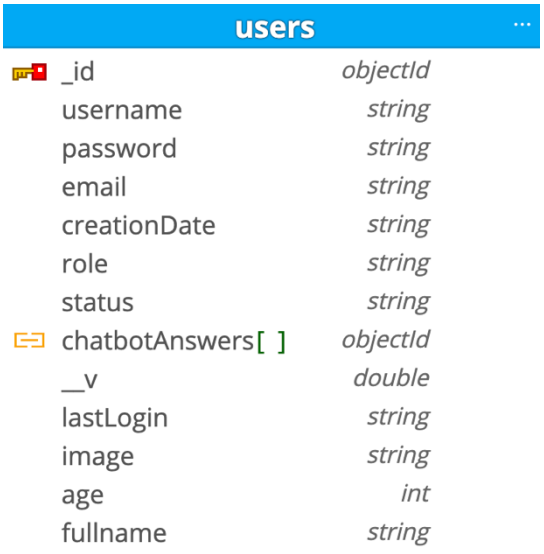
- **Adresář compontents** – Obsahuje znovupoužitelné komponenty, které tvoří stavební bloky uživatelského rozhraní aplikace, jako jsou například karty, dialogová okna atd.
- **Adresář contexts** – Slouží pro definici React kontextu, které umožňují sdílení stavu mezi různými komponentami.
- **Adresář hooks** – Obsahuje vlastní React hooky, které jsou používány pro logiku sdílenou napříč komponentami.
- **Adresář pages** – Zahrnuje soubory, které definují jednotlivé stránky aplikace. Každá "stránka" je vlastně komponenta, která může zahrnovat více menších komponent.
- **Adresář questions** – Obsahuje soubory s otázky pro integrovaný dotazník.
- **Adresář signal** – Obsahuje soubory spjaté s implementací protokolu Signal Protocol. Jedná se o vše od definice Signal Store, přes funkce pro sestavení sessions či vytvoření identity, po funkce pro samotné šifrování a dešifrování.
- **Adresář types** – Slouží pro definice TypeScript typů a rozhraní (interface).
- **Adresář utils** – Obsahuje pomocné funkce a utility, které jsou používány napříč aplikací.
- **Soubory index.scss, responsive.scss a typekit.scss** – Soubory pro globální stylování pomocí SCSS. *Index* obsahuje základní stylování, *responsive* pak responsivní stylování pro zařízení různých velikostí a *typekit* definice fontů.
- **Soubor main.tsx** - Vstupní bod aplikace, který se stará o inicializaci a vykreslování hlavních komponent (stránek) aplikace.
- **Soubor .env** - Soubor pro proměnné, který obsahuje citlivá nastavení mimo zdrojový kód.
- **Soubor index.html** - Základní HTML soubor, který slouží jako šablona pro React aplikaci. Zde se nachází kořenový DOM uzel, do kterého React vykresluje komponenty.
- **Soubor vercel.json, vite.config.js** – Konfigurační soubory pro Vercel a Vite, které se používají pro nasazení a lokalní vývoj aplikace.

6.3 Struktura databáze

V této podkapitole následuje detailnější popis struktury databáze jejíž schéma již bylo nastíněno v rámci návrhu datového modelu. Jak již bylo v této práci zmíněno, jedná se o nerelační databázi dokumentového typu MongoDB, což umožňuje data strukturovat velmi odlišně oproti typickým relačním databázím. Popis struktury databáze bude psán z hlediska implementace, tedy bude popsáno k čemu dané kolekce slouží, jakou hrají roli v aplikaci a jak se budou v daných situacích chovat. Vždy bude předvedena daná kolekce, případně i se zanořeným objektem či polem, a k ní i kód Mongoose modelu.

6.3.1 Users kolekce

Kolekce users je kolekcí pro ukládání uživatelů a jak lze na obrázku č. 29 vidět, ukládá se zde uživatelské jméno, hash hesla, emailová adresa, datum vytvoření profilu, role (moderátor/člen), status, datum posledního přihlášení, URL profilového obrázku, věk a celé jméno. Krom těchto jednoduchých typů je zde i propojení s kolekcí *ChatbotAnswers*, která reprezentuje odpovědi na dotazník a bude popsána v další části této kapitoly. Tato kolekce zde není jako pouhý zanořený objekt, aby bylo možné jednoduše získávat její dokumenty bez návaznosti na konkrétního uživatele. V níže uvedeném kódu č. 7 lze vidět realizace této kolekce pomocí Mongoose.



Obrázek 29 Users kolekce

```
const ObjectId = Schema.Types.ObjectId;
const userSchema = new Schema<IUser>({
    username: {
        type: String,
```

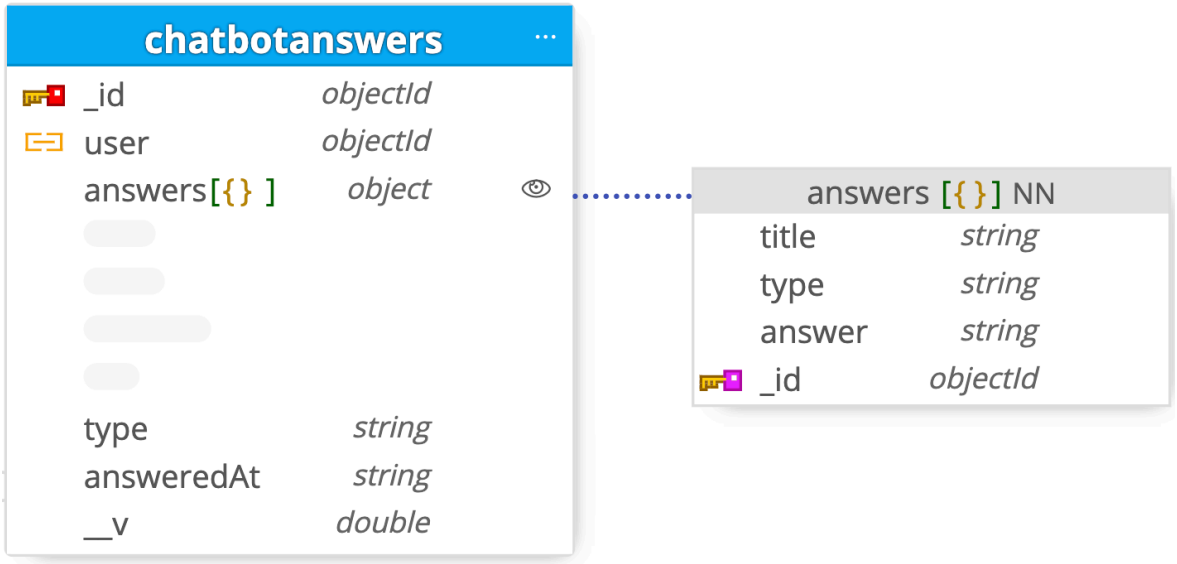
```
        required: true
    },
    password: {
        type: String,
        required: true
    },
    email: {
        type: String,
        required: true,
        validate: [emailValidation, 'Zadané pole musí být ve formátu e-mailu.'],
    },
    creationDate: {
        type: String,
        required: true
    },
    role: {
        type: String,
        required: true
    },
    fullname: String,
    lastLogin: String,
    image: String,
    age: Number,
    profession: String,
    status: String,
    chatbotAnswers: [{ type: ObjectId, ref: 'ChatbotAnswer' }],
    biography: String,
});

module.exports = model<IUser>('User', userSchema);
```

Kód 7 User schéma v Mongoose

6.3.2 ChatbotAnswers kolekce

Kolekce ChatbotAnswers slouží k ukládání uživatelských odpovědí na integrovaný dotazník. Jak lze pozorovat na obrázku č. 30 níže, tato kolekce je napojená na kolekci Users, takže každý jeden dokument z této kolekce patří danému uživateli, ale zároveň se jedná o samostatnou kolekci, na kterou se lze dotazovat bez závislosti na konkrétním uživateli, což je vhodné například pro moderátora, který v aplikaci může zobrazit výpis všech odpovědí. Dále tato kolekce obsahuje typ, což později umožní rozlišit na který z dotazníků bylo uživatelem zodpovězeno a také datum, kdy byla odpověď vytvořena. V neposlední řadě je zde zanořené pole odpovědí. Formát tohoto pole, respektive objektu, lze vidět na pravé straně uvedeného obrázku. Jedná se o jednotlivé odpovědi, které se skládají z titulku otázky,

samotné odpovědi a typu otázky (v současnosti se jedná o již popsané intervalové a textové otázky). Reprezentace této kolekce v kódu lze vidět v kódu č. 8.



Obrázek 30 ChatbotAnswers kolekce

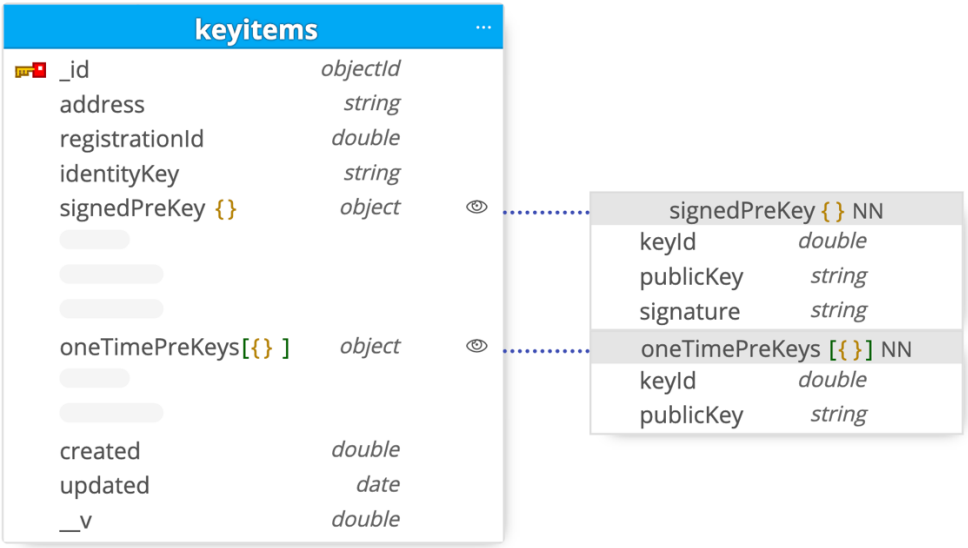
```
const ObjectId = Schema.Types.ObjectId
const answerSchema = new Schema({
    title: String,
    type: String,
    answer: String
});

const chatbotAnswerSchema = new Schema({
    user: {
        type: ObjectId,
        ref: 'User'
    },
    answers: [answerSchema],
    type: String,
    answeredAt: String
})

module.exports = model('ChatbotAnswer', chatbotAnswerSchema)
```

Kód 8 ChatbotAnswer schéma v Mongoose

6.3.3 KeyItems kolekce

Kolekce KeyItems je určená pro publikování identity a klíčového svazku po vytvoření uživatelského účtu v rámci protokolu Signal Protocol. Ukládá se zde adresa uživatele, registrační ID, klíč identity, datum vytvoření a datum případné úpravy a také zanořený objekt

podepsaného prekey a pole jednorázových prekeys. Podepsaný prekey se skládá z ID klíče, veřejného klíče a samotného podpisu. Jednorázové prekeys jsou složeny z ID klíče a veřejného klíče. Implementace v kódu bude opět uvedená níže v kódu č. 9.



Obrázek 31 KeyItems kolekce

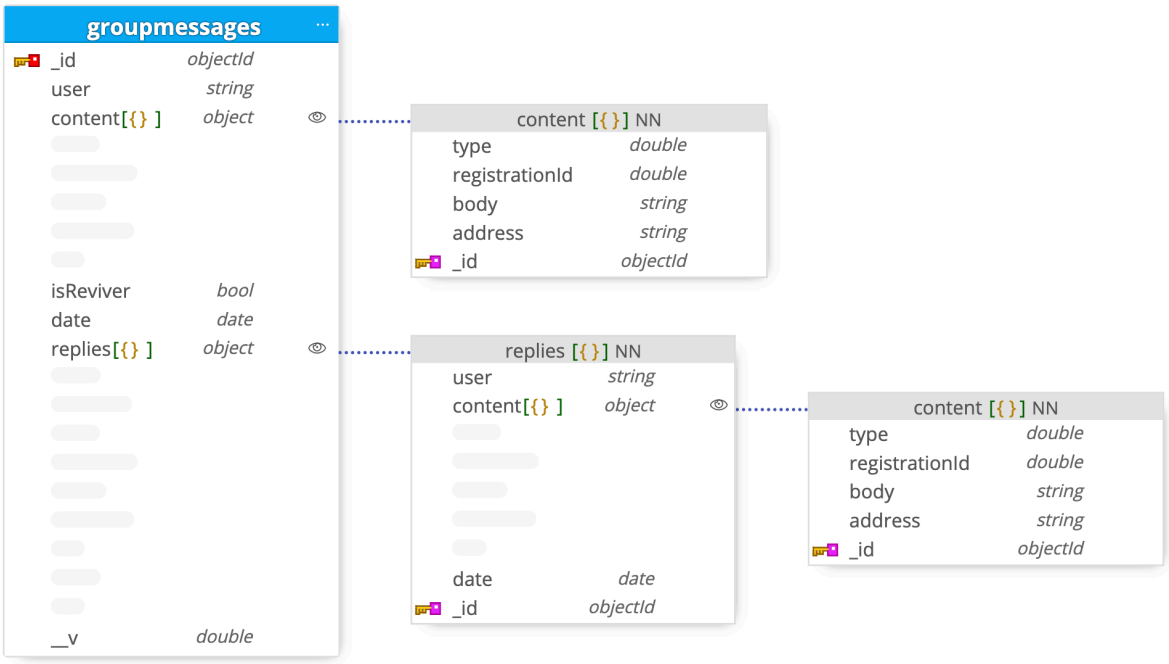
```
const keyItemSchema = new Schema<IKeyItem>({
    address: { type: String, required: true },
    registrationId: { type: Number, required: true },
    identityKey: { type: String, required: true },
    signedPreKey: {
        keyId: Number,
        publicKey: String,
        signature: String,
    },
    oneTimePreKeys: [{
        keyId: Number,
        publicKey: String,
    }],
    created: { type: Number, required: true },
    updated: { type: Number, required: true },
});

module.exports = model<IKeyItem>('KeyItem', keyItemSchema);
```

Kód 9 KeyItem schéma v Mongoose

6.3.4 GroupMessages kolekce

Kolekce GroupMessages je kolekcí pro ukládání jednotlivých skupinových zpráv. Skupinová zpráva se skládá z uživatele, který zprávu odeslal, data odeslání a boolean informace,

jestli se jedná o speciální moderátorskou zprávu „oživovák“. Dále jsou zde zanořené pole pro šifrovaný obsah a odpovědi. Zanořené pole pro šifrovaný obsah *content* je složeno z typu zprávy, pro odlišení, jestli se jedná o tzv. prekey nebo session zprávu. Prekey zpráva je vlastně iniciální zpráva, která slouží k úspěšnému navázání session. A session zpráva je pak zpráva, která byla odeslána v rámci již vytořené session. Dále je zde registrační ID odesílatele a adresa příjemce. V neposlední řadě je zde *body*, které pak obsahuje již zašifrovanou zprávu. Šifrovaný obsah *content* je polem, jelikož každý uživatel musí vytvořit session s každým uživatelem, takže šifrovaný text se poté pro různé uživatele liší. Tato kolekce bude opět znázorněna i v kódu č. 10.



Obrázek 32 Kolekce GroupMessages

```
const contentSchema = {
    type: Number,
    registrationId: Number,
    body: String,
    address: String
}
const baseSchema = {
    user: String,
    content: [new Schema(contentSchema)],
    isReviver: Boolean,
    date: {
        type: Date,
        default: new Date()
```

```
    }
};

const replySchema = new Schema(baseSchema);
const groupMessageSchema = new Schema({
    ...baseSchema,
    replies: [replySchema]
});

module.exports = model('GroupMessage', groupMessageSchema);
```

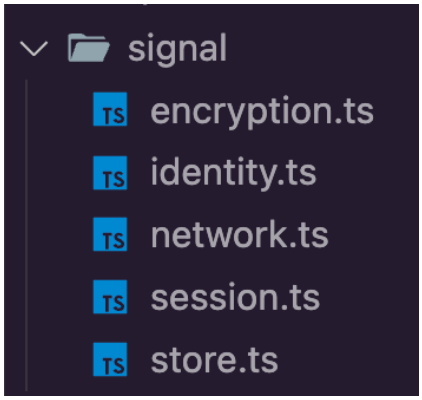
Kód 10 GroupMessage schéma v Mongoose

7 IMPLEMENTACE ZABEZPEČENÍ ZPRÁV

Tato kapitola bude věnována popisu implementace E2EE, tedy implementace kryptografického protokolu Signal Protocol pomocí knihovny *libsignal-protocol-typescript* od společnosti Privacy Research LLC.

7.1 Základní struktura

V rámci popisu struktury klientské části aplikace byl zmíněn adresář *signal*. Zde jsou umístěny veškeré soubory, které v aplikaci zajišťují E2EE. Nyní budou popsány jednotlivé soubory tohoto adresáře, a to jakou roli v celém procesu zastávají.



Obrázek 33 Adresář signal

- **Soubor encryption.ts** – Jak již název napovídá, tento soubor zastává roli samotného šifrování. Obsahuje dvě funkce, a to funkci pro šifrování a funkci pro dešifrování.
- **Soubor identity.ts** – Tento soubor obsahuje funkci pro vytvoření identity a klíčového svazku. Používá se bezprostředně po registraci nového uživatele.
- **Soubor network.ts** – Tento soubor obsahuje čtyři funkce a je ho možné rozdělit do na dvě části. První část se skládá z funkcí pro odeslání požadavku na server, konkrétně požadavek na publikování klíčového svazku na server a požadavek na získání klíčového svazku jiného uživatele. Druhá část zde pak zastává roli dvou pomocných funkcí, které mají za úkol serializovat klíčový svazek do požadovaného tvaru před samotnou publikací na server, a naopak deserializovat klíčový svazek jiného uživatele po jeho získání ze serveru.
- **Soubor session.ts** – Tento soubor drží funkci, která má za úkol zkontrolovat, zda již existuje session a v případě, že neexistuje vytvořit session novou.

- **Soubor store.ts** – Jedná se o soubor, který drží třídu zvanou *SignalStore*, která zaštiťuje lokální úložiště veškerých potřebných dat vygenerovaných knihovnou *libsignal-protocol-typescript*. Tato třída je pak v kódu použita podle návrhového vzoru Singleton. Původní kód v tomto souboru pochází z open source projektu právě zmiňované Privacy Research LLC, nicméně není přímo součástí této knihovny. Kód byl rozšířen o perzistentní úložiště v podobě IndexedDB, tak aby potřebná data nebyla při odpojení ztracena.

7.2 Průchod kryptografickým procesem

V této podkapitole bude chronologicky popsán celý kryptografický proces. Tedy bude zdokumentováno, jakým způsobem aplikace vytváří prvotní identitu uživatele, otevírá sessions, šifruje zprávy, dešifruje zprávy a také jakým způsobem přijaté zprávy lokálně ukládá.

7.2.1 Vytvoření a publikace prekey svazku

Celý proces začíná při registraci nového uživatele. Po registraci uživatele se totiž bezprostředně potom provede tvorba a následná publikace jeho identity a klíčového svazku. Pokud tedy registrační proces proběhne úspěšně a je vytvořen nový uživatel zavolá se funkce ze souboru *signal/identity.ts*. V rámci této funkce je nejprve vygenerováno registrační ID pomocí funkce *generateRegistrationId* z knihovny, poté je vygenerován pár identity key za pomocí funkce *generateIdentityKeyPair*. Obě tyto hodnoty jsou pak uloženy do *SignalStore*. Pokračuje se vygenerováním one-time prekeys, které jsou také uloženy do *SignalStore*. Následně je vytvořen podepsaný (signed) prekey pomocí funkce *generateSignedPreKey*, která jako parametr přebírá identity key a náhodně vygenerované ID. Tato funkce se postará i o samotný podpis (signature) pomocí soukromé části identity key. I tento podepsaný prekey je uložen v rámci *SignalStore*. Nakonec je pomocí vlastní funkce *storeKeyBundle* ze souboru *signal/network.ts* publikován klíčový svazek obsahující veřejné části klíčů. Celý tento proces lze vidět v kódu č. 11.

```
const registrationStorageKey: string = 'registrationID';
const identityKeyPairStorageKey: string = 'identityKey';

export const createIdentity = async (username: string) => {
  // Create registrationId and identity key pair
  const registrationId: number = KeyHelper.generateRegistrationId();
  const identityKeyPair: KeyPairType<ArrayBuffer> = await KeyHelper.generateIden-
tityKeyPair();
```

```
  signalStore.put(registrationStorageKey, registrationId)
  signalStore.put(identityKeyPairStorageKey, identityKeyPair)

  // Create a one-time use prekey and a signed pre-key
  const baseKeyId = Math.floor(10000 * Math.random());
  const preKey: PreKeyPairType<ArrayBuffer> = await KeyHelper.generatePreKey(ba-
seKeyId);
  signalStore.storePreKey(`${baseKeyId}`, preKey.keyPair)

  // Create a signed prekey
  const signedPreKeyId = Math.floor(10000 * Math.random());
  const signedPreKey: SignedPreKeyPairType<ArrayBuffer> = await KeyHelper.genera-
teSignedPreKey(
    identityKeyPair,
    signedPreKeyId
  );
  signalStore.storeSignedPreKey(signedPreKeyId, signedPreKey.keyPair)

  const publicSignedPreKey: SignedPublicPreKeyType = {
    keyId: signedPreKeyId,
    publicKey: signedPreKey.keyPair.pubKey,
    signature: signedPreKey.signature,
  };
  const publicPreKey: PreKeyType = {
    keyId: preKey.keyId,
    publicKey: preKey.keyPair.pubKey,
  };

  storeKeyBundle(username, {
    registrationId,
    identityKey: identityKeyPair.pubKey,
    signedPreKey: publicSignedPreKey,
    oneTimePreKeys: [publicPreKey],
  });
};
```

Kód 11 Funkce pro vytvoření identity (signal/identity.ts)

7.2.2 Sestavení sessions

Jakmile má uživatel vytvořený účet a identitu, může se přihlásit. Po přihlášení uživatel začne kontrolovat sessions pomocí funkce vlastní *openSession* ze souboru *signal/session.ts*. Tato funkce se tedy nejprve podívá do *SignalStore* a zkontroluje, zdali session pro daného uživatele již existuje. Pokud session neexistuje sestaví novou. V procesu vytváření nové session nejprve musí získat prekey bundle daného uživatele ze serveru, tudíž zavolá vlastní funkci *getPreKeyBundle* ze souboru *signal/network.ts*. Poté vytvoří tzv. *SessionBuilder* pomocí kterého následně díky jeho metodě *processPreKey* zpracuje získaný prekey bundle uživatele

a tím tuto session i uloží do *SignalStore*. Tato funkce vrací nový tzv. *SessionCipher*, který může sloužit k šifrování inicializačních zpráv, nicméně není to nezbytně nutné, protože se vytváří před šifrováním každé zprávy. Tento proces je opět vidět v kódu č. 12.

```
export const openSession = async (username: string, token: string) => {
  let address = new SignalProtocolAddress(username, 1);
  const addressStr = `${address.name}.${address.deviceId}`;
  const session = await signalStore.loadSession(addressStr);

  // Check if session has been already established for that user, if not build
new one
  if (!session) {
    const keyBundle = await getPreKeyBundle(username, token);
    const sessionBuilder = new SessionBuilder(signalStore, address);

    const session = await sessionBuilder.processPreKey(keyBundle!);
    console.log({ session, keyBundle });
  }

  const sessionCipher = new SessionCipher(signalStore, address);
  return sessionCipher;
}
```

Kód 12 Funkce pro sestavení session (signal/session.ts)

7.2.3 Šifrování odesílaných zpráv

Před odesláním každé zprávy proběhne proces, který zprávu zašifruje. Šifrování řeší funkce *encryptMessage* ze souboru *signal/encryption.ts*. Tato funkce nejprve převede textovou zprávu na buffer a poté se připraví k šifrování zprávy pro každého uživatele, jelikož je nutné zprávu zašifrovat vždy v rámci session sestavené mezi uživateli. Získá adresu na základě které poté vytvoří *SessionCipher* ze session pro daného uživatele uložené v *SignalStore*. Pomocí metody *encrypt* vytvořené *SessionCipher* zašifruje zprávu a přidá adresu příjemce, aby bylo jasné, pro koho je určená k přijetí a následnému dešifrování. Tato funkce poté vrací zašifrované zprávy pro všechny příjemce, které jsou následně rozeslány prostřednictvím WebSocketu. Důležité je pak i chování serveru při přijetí zprávy skrze WebSocket. Server se nejprve podívá, kteří uživatelé jsou momentálně připojeni a těm zašle pro ně zašifrovanou zprávu přímo. Pokud však daný uživatel není přítomen, uloží šifrovanou zprávu do databáze, ze které ji pak daný uživatel získá po dalším připojení do aplikace. Tento proces je znázorněn v kódu č. 13.

```
export const encryptMessage = async (users, newMessage) => {
  const text = new TextEncoder().encode(newMessage).buffer;
  const messagePromises = users.map(async user => {
    const address = new SignalProtocolAddress(user.username, 1);
    const sessionCipher = new SessionCipher(signalStore, address);

    const ciphertext = await sessionCipher.encrypt(text);
    const remoteAddress = sessionCipher.remoteAddress.name;

    const editedCiphertext = { ...ciphertext, registrationId: await signal-
Store.getLocalRegistrationId()}

    return {
      ...editedCiphertext,
      address: remoteAddress
    };
  });

  // Content includes all encrypted messages for each user in the group
  const messageContent = await Promise.all(messagePromises);
  return messageContent;
}
```

Kód 13 Funkce pro šifrování zpráv (signal/encryption.ts)

7.2.4 Dešifrování přijatých zpráv

Pokud je uživatel ve skupině při odeslání zprávy přítomný, zprávu zachytí WebSocket. Jakmile uživatel zprávu zachytí, je automaticky volána funkce *decryptMessage* ze souboru *signal/encryption.ts*. Stejně jako u šifrování, funkce nejprve vytvoří SessionCipher ze session daného uživatele ze *SignalStore*. Poté je nutné zkontrolovat o jaký typ zprávy se jedná. Při šifrování této zprávy totiž bylo rozhodnuto, jestli jde o inicializační zprávu, tzv. PreKey Message nebo o zprávu šifrovanou na již domluvené session. Inicializační zpráva je označena číslem 3, zatímco klasická zpráva je označena číslem 1. Pokud se jedná o inicializační zprávu, je dešifrování realizováno pomocí funkce *decryptPreKeyWhisperMessage*, v druhém případě pak pomocí *decryptWhisperMessage*. Dešifrovaná zpráva je poté převedena do textu a funkcí vrácena. Implementace této funkce pro dešifrování lze opět pozorovat v kódu č. 14.

Jestliže však uživatel při odeslání zprávy není ve skupině přítomen, zprávu přijme a dešifruje až po dalším připojení, a to pomocí HTTP požadavku. Proces dešifrování však

probíhá stejným způsobem. Ve chvíli, kdy uživatel zprávu z databáze příjme, tato zpráva je z databáze smazána.

Veškeré přijaté zprávy se ukládají do perzistentního lokálního uložiště IndexedDB. Před uložením jsou však ještě pro vyšší bezpečnost zašifrovány symetrickým klíčem, který je odvozen z hesla uživatele. Po dalším přihlášení se pak zprávy z IndexedDB opět dešifrují a načtou do paměti aplikace.

```
export const decryptMessage = async (data) => {
  const msgContent = data.content;
  const senderAddress = new SignalProtocolAddress(data.user, 1);
  const sessionCipher = new SessionCipher(signalStore, senderAddress);

  try {
    let plaintext;
    if (msgContent.type === 3) {
      plaintext = await sessionCipher.decryptPreKeyWhisperMessage(msgCon-
tent.body, 'binary');
    } else if (msgContent.type === 1) {
      plaintext = await sessionCipher.decryptWhisperMessage(msgContent.body, 'bi-
nary');
    } else {
      console.error('Unknown tyope of encoded message.');
    }

    return {
      ...data,
      content: new TextDecoder().decode(plaintext)
    };
  } catch (error) {
    console.error('Decryption error:', error);
    return null;
  }
}
```

Kód 14 Funkce pro dešifrování zpráv (signal/encryption.ts)

8 POPIS IMPLEMENTOVANÉ APLIKACE

V této kapitole budou popsány důležité části implementované webové aplikace. Bude prezentováno, jak daná část aplikace vypadá, jaký je její účel a také bude nastíněno, jak na pozadí funguje.

8.1 Komunikační rozhraní

Komunikační rozhraní je jádrem celé aplikace a slouží jako centrální prostor pro interakci a výměnu informací mezi členy skupiny. Je to místo, kde mohou uživatelé sdílet své názory, zkušenosti a získat podporu od ostatních členů komunity. Vedle uživatelů je zde významná role přidělena moderátorovi, který je zároveň odborníkem na terapii v oblasti syndromu vyhoření. Jeho přítomnost zajišťuje, že diskuse zůstávají konstruktivní a zaměřené na téma. K zaměření diskuse na určité téma a k obecnému oživení konverzace může moderátor odeslat speciální zprávu „oživovák“. Moderátor aktivně přispívá odbornými radami a podporou. Jeho schopnost poskytovat odborné vedení a terapeutické intervence přináší dodatečnou hodnotu diskuzím a pomáhá udržovat komunitu orientovanou na vzájemnou pomoc.

8.1.1 Hlavní komunikační okno

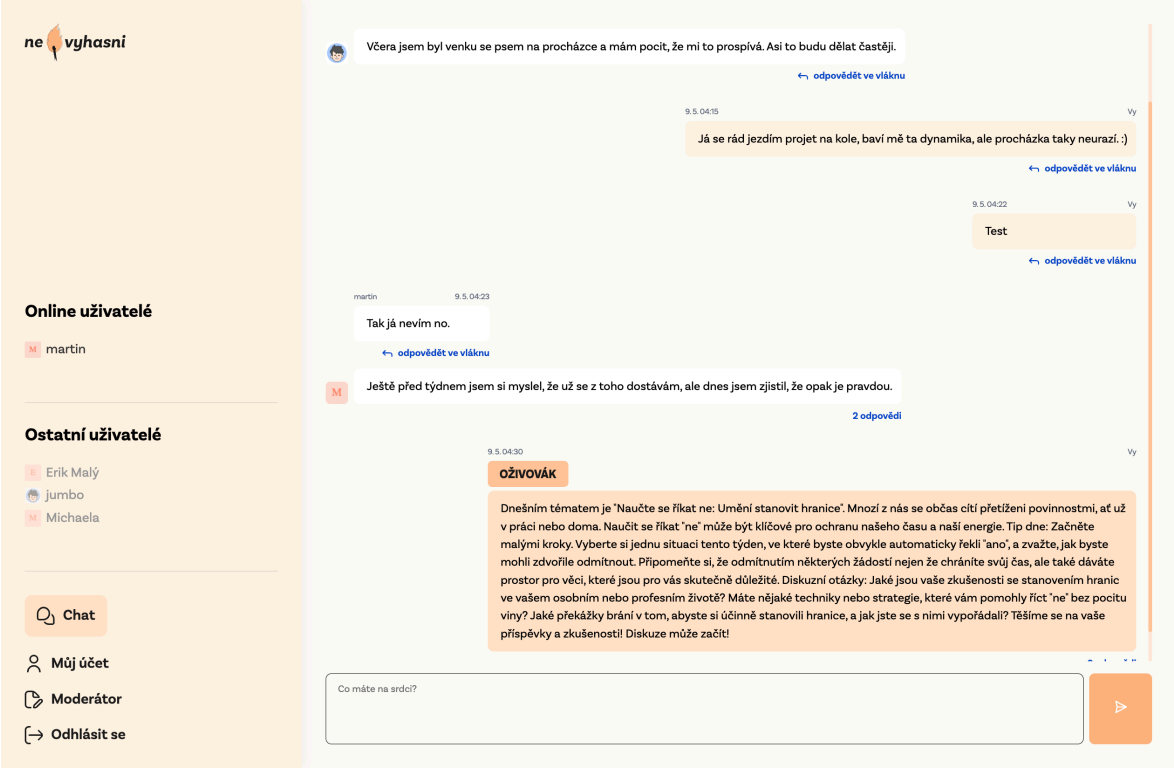
Hlavní komunikační okno je stránkou, kterou uživatel vidí jako první po úspěšném přihlášení. V tomto okně vidí uživatel veškeré zprávy od ostatní členů i moderátora.

Zprávy, které již v minulosti přijal jsou načteny z lokálního úložiště IndexedDB, kde jsou uloženy v zašifrované podobě uživatelovým symetrickým klíčem odvozeným od jeho hesla. Klíč k dešifrování těchto zpráv se vytváří po každém přihlášení. Nicméně neznamená to, že se uživatel musí pokaždé přihlásit, aby bylo možné lokální zprávy dešifrovat. Odvozený klíč se totiž ukládá také do IndexedDB s tím, že jakmile uživatel přijde do aplikace, klíč se načte do stavu aplikace a z IndexedDB se pro vyšší bezpečnost smaže. Před opuštěním aplikace je tento klíč opět nahrán do IndexedDB a při další návštěvě se tento proces opakuje stejným způsobem.

Zprávy, které uživatel ještě nepřijal jsou pak po přihlášení stahovány z databáze MongoDB. Po přijetí E2E zašifrovaných zpráv proběhne proces dešifrování a pomocí sestavených sessions jsou tyto zprávy dešifrovány, zobrazeny v komunikačním okně a následně zašifrovány symetrickým klíčem uživatele a uloženy lokálně do IndexedDB. Nově přijaté zprávy jsou také odstraněny z databáze MongoDB. Pokud uživatel dostane zprávu ve své

přítomnosti v aplikaci, zpráva je doručena v reálném čase prostřednictvím protokolu WebSocket a provádí se stejný proces dešifrování a uložení zpráv. Jestliže se přítomný uživatel nachází v komunikačním okně výše a nevidí tedy na nové zprávy, je mu ve spodní části okna zobrazena notifikace v podobě malého boxu.

Pro odeslání zprávy aplikace poskytuje v rámci komunikačního okna vstupní pole, kam uživatel svou novou zprávu zapíše a následně tlačítkem vedle vstupního pole odešle. Před odesláním je zpráva také zašifrována pomocí sestavených sessions a až poté odeslána skrze protokol WebSocket.

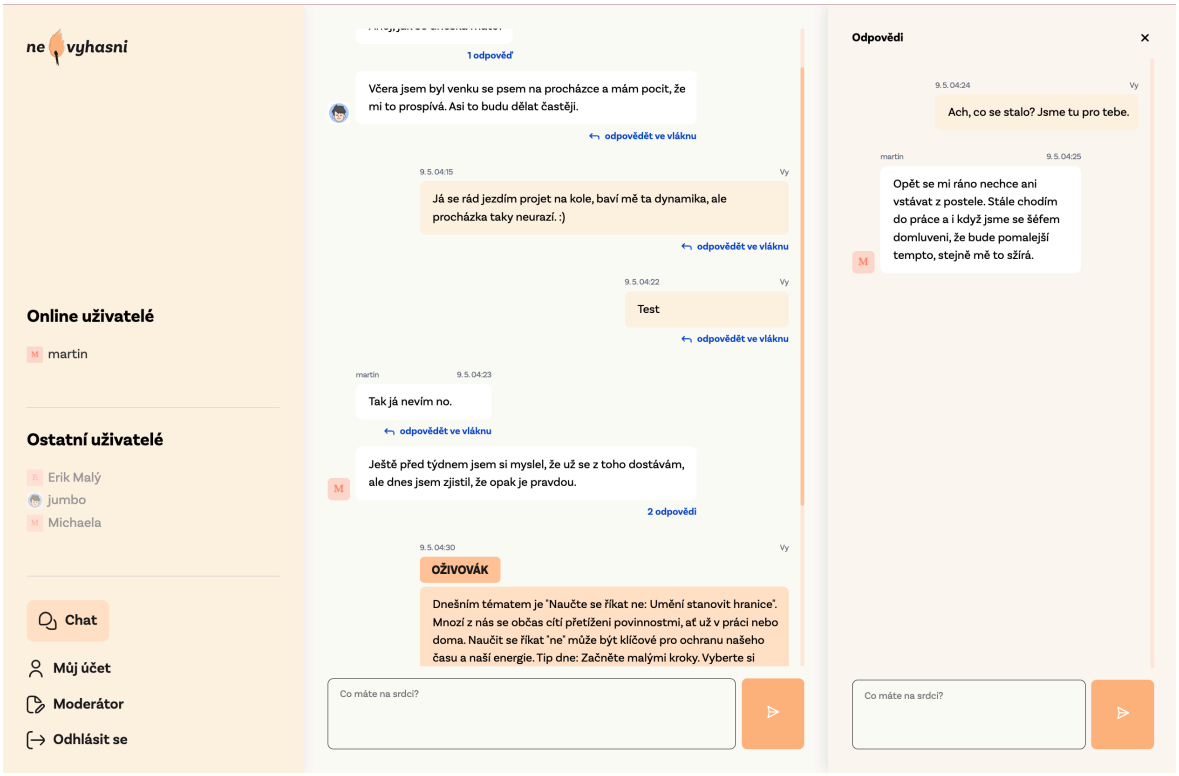


Obrázek 34 Aplikace: Hlavní komunikační rozhraní

8.1.2 Sekundární komunikační okno (vlákno)

K sekundárnímu komunikačnímu oknu, tedy k vláknu, se uživatel dostane po kliknutí na tlačítko pod kteroukoliv zprávou. Pokud zpráva neobsahuje žádné odpovědi, je tlačítko označeno textem „Odpovědět ve vláknu“, pokud odpovědi ale obsahuje, je označeno počtem odpovědí, které jsou také v aplikaci aktualizovány v reálném čase. Jakmile má uživatel otevřené sekundární komunikační okno nějaké zprávy, jsou mu načteny odpovědi na danou zprávu velice podobným způsobem jako tomu je u hlavních zpráv. Veškeré procesy šifrování, dešifrování či ukládání odpovědí v lokálním úložišti fungují na stejných principech

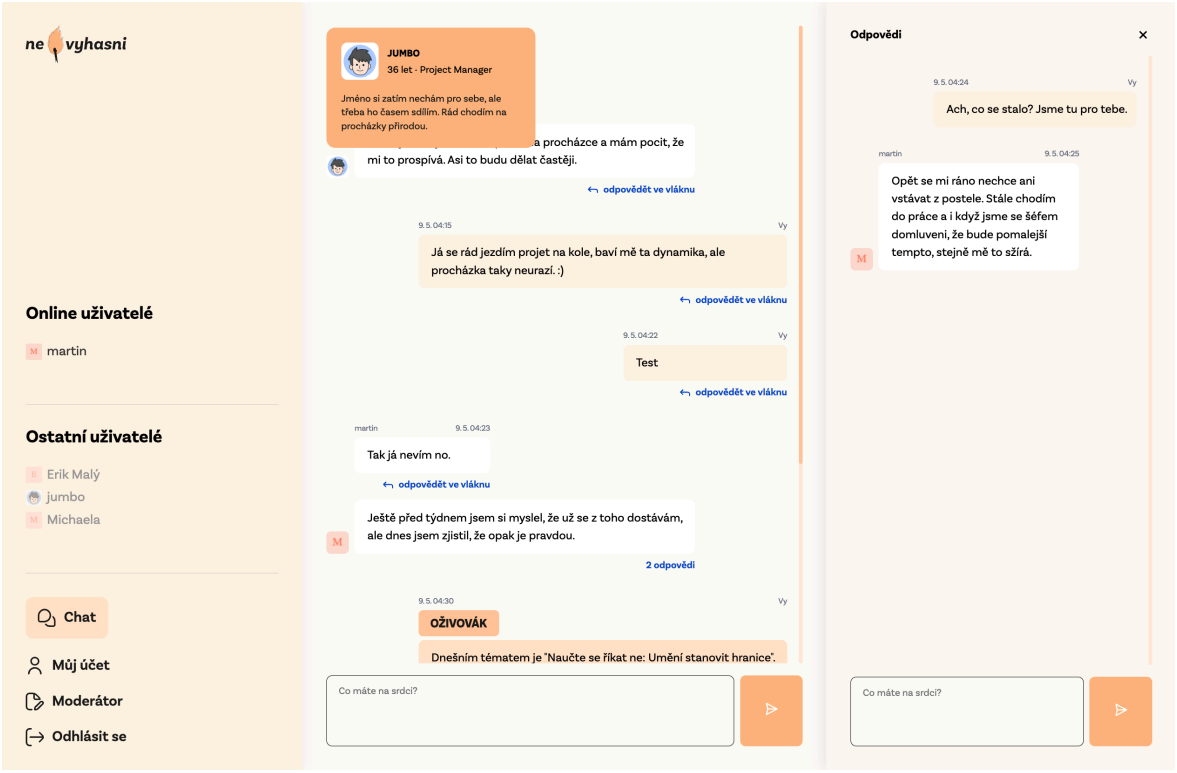
jako klasické zprávy. Uživateli je zde k dispozici opět vstupní pole s tlačítkem pro odeslání zprávy a ve vrchní části se nachází křížek pro zavření tohoto okna.



Obrázek 35 Aplikace: Sekundární komunikační okno (vlákno)

8.1.3 Náhledy uživatelských profilů

Interakce v hlavním komunikačním okně je rozšířena o možnost zobrazení velmi jednoduchého uživatelského profilu jakéhokoliv uživatele, který kdy ve skupině zanechal zprávu. Tento náhled si může uživatel zobrazit najetím kurzoru myši na profilový obrázek vedle dané zprávy. Informace o uživatelích jsou načteny a uloženy do stavu aplikace při každé návštěvě.



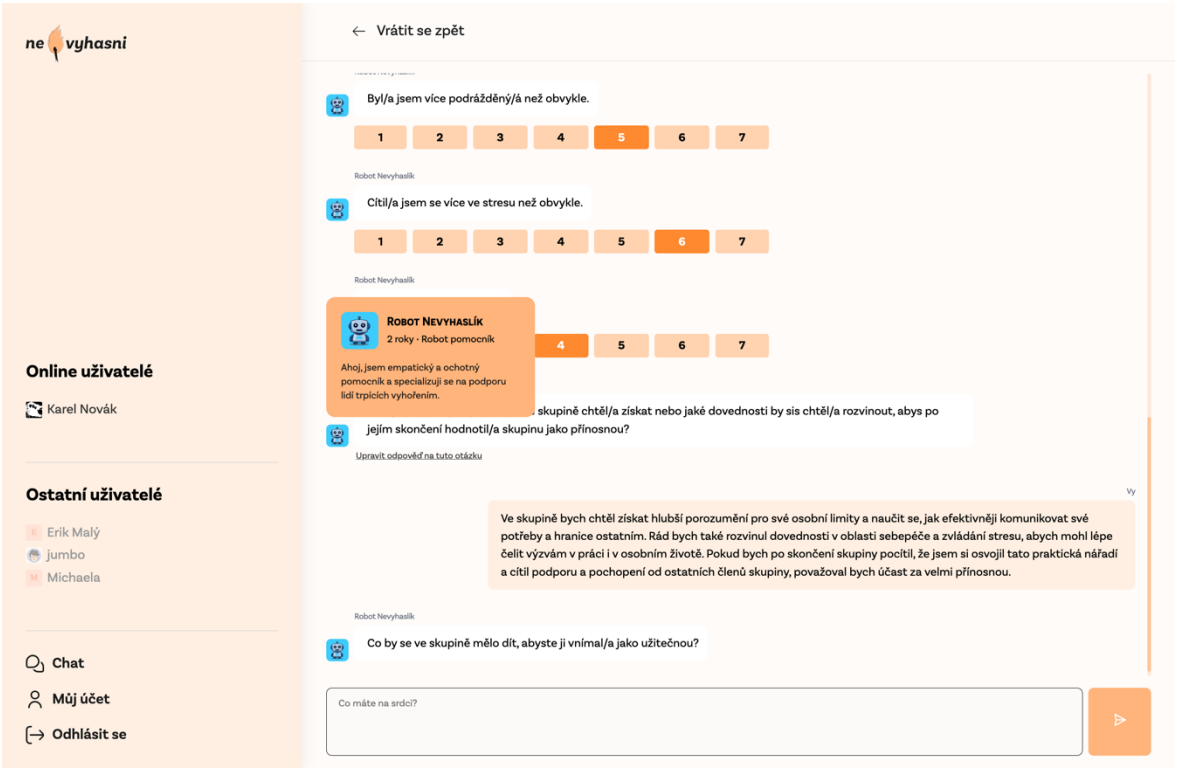
Obrázek 36 Aplikace: Náhled profilu

8.2 Integrovaný chatbot dotazník

Při prvních návštěvách aplikace je uživatelům zobrazováno pop-up okno, které je informuje o možnosti vyplnit vstupní dotazník, který má sloužit moderátorovi skupiny jako stručný přehled o členech skupiny a jejich psychickém stavu.

Pokud se uživatel rozhodne vyplnit dotazník, je přesměrován na stránku s jednoduchým chatbotem, který mu pokládá dotazy. Jak již bylo v této práci popsáno, v současné době existují dva typy dotazů, tedy dotaz s možností odpovědět na škále od 1 do 7 a dotaz s možností odpovědět prostřednictvím psané zprávy. Do dotazníku uživatele uvede robot, který byl pojmenován jako Nevyhaslík. Pokud se uživatel při vyplňování dotazníku rozhodne odejít, zodpovězené otázky zůstanou uloženy v rámci lokálního úložiště LocalStorage a při další návštěvě této stránky jsou opět načteny. Nové otázky jsou uživateli předkládány postupně s každou zodpovězenou otázkou. Odpověď může také uživatel kdykoliv upravit. U otázek prvního typu jednoduchým kliknutím na jinou možnost a u otázek typu druhého pomocí tlačítka pod danou otázkou, která uživateli načte jeho odpověď zpět do vstupního pole a po odeslání starou odpověď touto nahradí. Dotazník je odeslán prostřednictvím požadavku s odpovědí v jeho těle na příslušný endpoint po zodpovězení poslední otázky. Jakmile se tak stane, danému uživateli již není zobrazováno zmiňované pop-up okno.

Momentálně je navržen a implementován pouze jeden druh dotazníku, a to úvodní dotazník, který jak již bylo řečeno, poskytne prvotní přehled o uživatelích a jejich stavu. Nicméně aplikace je připravena na případné rozšíření o další typy dotazníků v budoucnu, které by mohly být v průběhu fungovaní skupiny uživatelům zpřístupňovány a moderátor by tak získal přehled o vývoji jednotlivých členů.



Obrázek 37 Aplikace: Chatbot dotazník

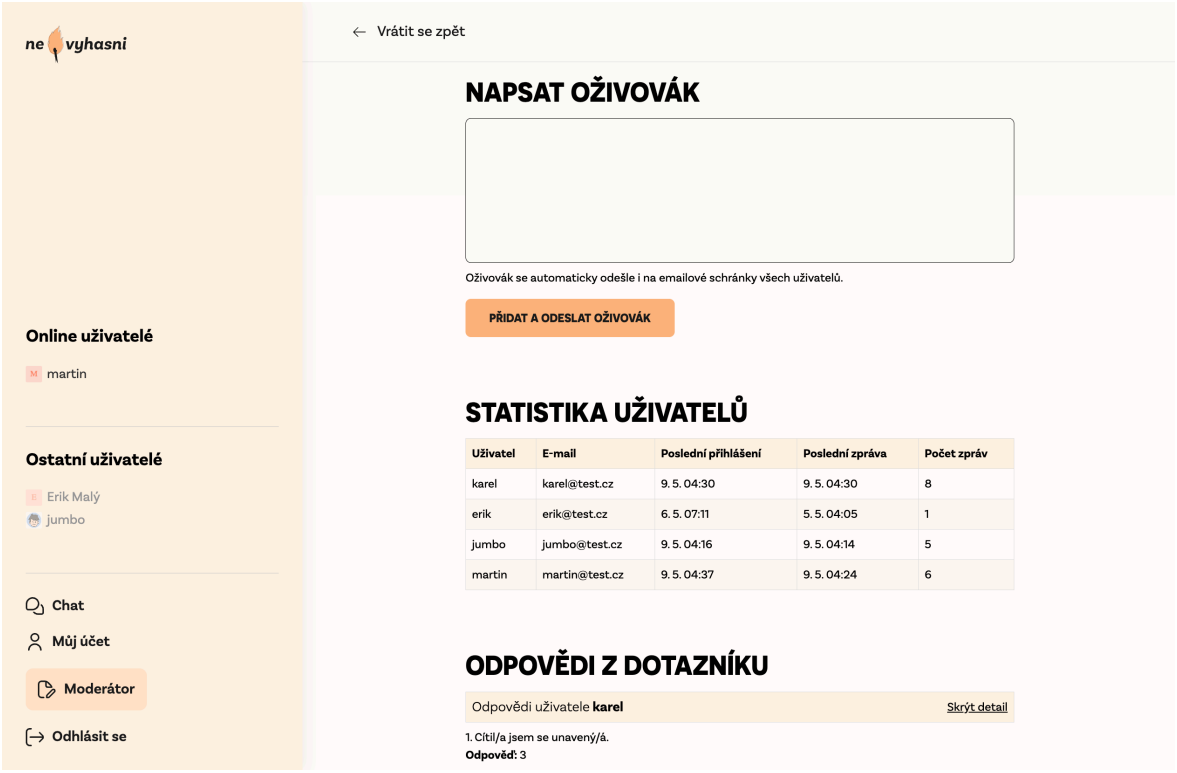
8.3 Panel pro moderátora

Panel pro moderátora v aplikaci zastává roli důležitého nástroje, který je dostupný pouze uživatelům s rolí „moderátor“. Jeho účelem je poskytnout moderátorovi jednoduché rozhraní odkud bude moci zasílat speciální zprávy (oživováky), které rozproudí a nasměrují konverzaci. Krom toho moderátorovi poskytuje prostředí, kde může přehledně sledovat, jak jsou uživatelé skupiny aktivní a také procházet odpovědi z dotazníku.

Odeslání oživováku funguje na stejném principu jako odeslání klasické zprávy. Tedy také je odesílán prostřednictvím protokolu WebSocket a také je E2E šifrován. Na pozadí se liší nastaveným atributem *isReviver* na hodnotu *true*. Pokud tedy moderátor vyplní vstupní pole ve svém panelu a odešle jej pomocí tlačítka pod ním, všem členům bude doručena

zašifrovaná zpráva označená jako oživovák. Kromě toho je automaticky všem uživatelům odeslán notifikační email s informací o této nové zprávě.

Co se týče uživatelských statistik a odpovědí na dotazník, ty jsou do aplikace při načítání stránky získávány prostřednictvím HTTP požadavků, ke kterým má přístup také pouze uživatel role „moderátor“. Odpovědi na dotazník jednotlivých uživatel si může moderátor zobrazovat tlačítkem pro zobrazení detailu.



Obrázek 38 Aplikace: Panel pro moderátora

ZÁVĚR

Hlavní náplní této práce bylo navrhnout a implementovat online komunikační skupinu pro peer terapii v podobě webové aplikace. Primární myšlenkou této aplikace je vytvořit efektivní nástroj, který bude pomáhat lidem, kteří procházejí syndromem vyhoření a poskytne jim spolehlivý a bezpečný prostor zaštítěný odborným terapeutem ke sdílení zkušeností a vzájemné podpoře.

Teoretická část nejprve nastínila základní kryptografické principy včetně problematiky koncového šifrování. Následoval průzkum populárních kryptografických protokolů pro koncové šifrování, jehož výsledkem bylo stanovení výběrových kritérií a následný výběr vhodného kryptografického protokolu, tedy protokolu Signal Protocol. V další kapitole teoretické části byly popsány jednotlivé technologie za pomoci, kterých je výsledná aplikace implementována.

Do praktické části bylo uvedeno obecným zadáním projektu, které poskytlo solidní základ pro následný sběr a dokumentaci požadavků na aplikaci. Posléze byla praktická část orientována na návrh webové aplikace, kdy byla nejprve navržena informační architektura, na kterou se navázalo návrhem drátěného modelu, který byl doplněn o informace o jednotlivých budoucích stránkách. Poté byl navržen datový model aplikace a následně diagram případů užití. Jednotlivé případy užití byly poté detailně popsány v rámci uživatelských scénářů, které znázorňují konkrétní interakce mezi uživatelem a systémem.

Další část již byla zaměřena na samotnou implementaci webové aplikace. Nejprve byla vysvětlena struktura obou částí aplikace a struktura databáze. Následně bylo s ukázky kódu popsáno jakým způsobem bylo do aplikace implementováno koncové šifrování protokolu Signal Protocol. Nakonec byly předvedeny důležité části vyvinuté aplikace.

Důležitým výstupem této diplomové práce tedy je funkční prototyp online komunikační skupiny pro peer terapii ve formě webové aplikace, na který se bude moci v budoucnu navázat dalším uživatelským testováním v praxi a přidáváním nových zajímavých funkcí.

SEZNAM POUŽITÉ LITERATURY

[1] *End-to-end encryption*. Online. Wikipedia, the free encyclopedia. Dostupné z: https://en.wikipedia.org/wiki/End-to-end_encryption. [cit. 2023-11-26].

[2] BERLOVE, Orlee. *What is end-to-end encryption and how does it work?* Online. PreVeil. 2023. Dostupné z: https://www.preveil.com/blog/end-to-end-encryption/. [cit. 2023-11-26]

[3] *What are the best practices for testing and auditing your data encryption in transit and at rest?* Online. LinkedIn. Dostupné z: https://www.linkedin.com/advice/0/what-best-practices-testing-auditing-your#:~:text=The%20most%20common%20protocol%20for,that%20is%20exchanged%20between%20them. [cit. 2023-11-26].

[4] HITER, Shelby. *End-to-End Encryption: Important Pros and Cons*. Online. CIO Insight: Enterprise Technology News & Trends for CIOs. 2021, updated on February 15, 2023. Dostupné z: https://www.cioinsight.com/security/end-to-end-encryption/. [cit. 2023-11-26].

[5] LUTKEVICH, Ben. *What is End-to-End Encryption (E2EE) and How Does it Work?* Online. TechTarget. 2021. Dostupné z: https://www.techtarget.com/searchsecurity/definition/end-to-end-encryption-E2EE#:~:text=In%20E2EE%2C%20the%20data%20is,any%20other%20entity%20or%20service. [cit. 2023-11-26]

[6] DOUGHERTY, Robert. *Exploring E2EE: Real-world Examples of End-to-End Encryption*. Online. Kiteworks Private Content Network: Regulatory Compliance Software Platform. 2023. Dostupné z: https://www.kiteworks.com/secure-file-sharing/real-world-examples-of-end-to-end-encryption/#:~:text=Common%20encryption%20algorithms%20used%20in,ensuring%20the%20security%20of%20E2EE. [cit. 2023-11-26].

[7] KNODEL, M.; CELI, S.; KOLKMAN, O. a GROVER, G. *Definition of End-to-end Encryption*. Online. IETF | Internet Engineering Task Force. 2023. Dostupné z: https://www.ietf.org/archive/id/draft-knodel-e2ee-definition-11.html. [cit. 2023-11-26].

[8] *Signal Protocol*. Online. Wikipedia, the free encyclopedia. Dostupné z: https://en.wikipedia.org/wiki/Signal_Protocol. [cit. 2023-11-26].

[9] EDDY, Max. *The Best Secure Messaging Apps for 2023*. Online. The Latest Technology Product Reviews, News, Tips, and Deals | PCMag. 2023. Dostupné z: https://www.pcmag.com/picks/best-secure-messaging-apps. [cit. 2023-11-26].

[10] SPADAFORA, Anthony. *The best encrypted messaging apps in 2023*. Online. Tom's Guide | Tech Product Reviews, Top Picks and How To. 2023. Dostupné z: https://www.tomsguide.com/reference/best-encrypted-messaging-apps. [cit. 2023-11-26].

[11] WAGENSEIL, Paul. *WhatsApp now offers 'end-to-end' encrypted backups — here's how it works*. Online. Tom's Guide | Tech Product Reviews, Top Picks and How To. 2021. Dostupné z: https://www.tomsguide.com/news/whatsapp-encrypted-backups. [cit. 2023-11-26].

[12] BALOCH, Rafay. *A Detailed Comparison of Security and Privacy on Signal and Telegram*. Online. Cyber Security Services - Protection against Cyber Attacks by Cyber Citadel. Dostupné z: https://www.cybercitadel.com/signal-vs-telegram-a-detailed-comparison-of-security-and-privacy/. [cit. 2023-11-26].

[13] *Cryptography and its Types.* Online. GeeksforGeeks | A computer science portal for geeks. 2024, aktualizováno 26. 4. 2024. Dostupné z: https://www.geeksforgeeks.org/cryptography-and-its-types/. [cit. 2024-05-10].

[14] *Kryptografie*. Online. In: Wikipedia: the free encyclopedia. San Francisco (CA): Wikimedia Foundation, aktualizováno 24. 4. 2024. Dostupné z: https://cs.wikipedia.org/wiki/Kryptografie. [cit. 2024-05-10].

[15] NAKOV, Svetlin. Asymmetric Key Ciphers. Online. *Practical Cryptography for Developers*. 2020. Dostupné z: https://cryptobook.nakov.com/asymmetric-key-ciphers. [cit. 2024-05-10].

[16] MARLINSPIKE, Moxie, PERRIN, Trevor (ed.). *The X3DH Key Agreement Protocol*. Online. Signal. 2016. Dostupné z: https://signal.org/docs/specifications/x3dh/. [cit. 2024-05-10].

[17] SECURITY AND PRIVACY ACADEMY [@SecPrivAca]. *Principles of Secure Messaging*. Online. 2023. Dostupné z: YouTube, https://www.youtube.com/watch?v=_XDbUAM4Wo0. [cit. 2024-05-10].

[18] MARLINSPIKE, Moxie, PERRIN, Trevor (ed.). *The Double Ratchet Algorithm*. Online. Signal. 2016. Dostupné z: https://signal.org/docs/specifications/doubleratchet/#overview. [cit. 2024-05-10].

[19] *Security*. Online. Wire – Support. 2023. Dostupné z: https://support.wire.com/hc/en-us/articles/4405940312081-Security. [cit. 2024-05-10].

[20] *Wire's independent security review*. Online. In: Medium. 2017. Dostupné z: https://wireapp.medium.com/wires-independent-security-review-61f37a1762a8. [cit. 2024-05-10].

[21] *Telegram Security: Is Telegram safe? Why Crypto companies use Telegram*. Online. Communication Compliance Solutions for Businesses | LeapXpert. 2022. Dostupné z: https://www.leapxpert.com/telegram-security-is-telegram-safe-why-crypto-companies-use-telegram/. [cit. 2024-05-10].

[22] *Introduction - JavaScript*. Online. MDN Web Docs. 2023. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Introduction/. [cit. 2024-05-10].

[23] SHUBEL, Meredith. *What Is TypeScript?* Online. The New Stack | DevOps, Open Source, and Cloud Native News. 2022. Dostupné z: https://thenewstack.io/what-is-typescript/. [cit. 2024-05-10].

[24] KARSH, Patrick. *Static Typing in TypeScript: Typescript Basics*. Online. In: Medium. 2023. Dostupné z: https://patrickkarsh.medium.com/static-typing-in-typescript-typescript-basics-9dbf527f1ef9. [cit. 2024-05-10].

[25] JANSEN, Remo H. *The TypeScript architecture*. Online. In: Packt Subscription | Advance your knowledge in tech. 2018. Dostupné z: https://subscription.packtpub.com/book/web-development/9781788391474/1/ch01lvl1sec02/the-typescript-architecture. [cit. 2024-05-10].

[26] *What is Node.js and How it Work?* Online. In: Medium. 2023. Dostupné z: https://medium.com/@asiandigitalhub/what-is-node-js-and-how-it-work-490f5ecba665#6d16. [cit. 2024-05-10].

[27] NAGARAJAN, Vignesh. *Understanding the Node.js Event Loop*. Online. In: Medium. 2023. Dostupné z: https://medium.com/@vickypaiyaa/understanding-the-node-js-event-loop-23858526b2ff. [cit. 2024-05-10].

[28] OLUSOLA, Samuel. *What is Node.js?* Online. In: JavaScript in Plain English. 2020. Dostupné z: https://javascript.plainenglish.io/what-is-node-js-5fe50e4332c8. [cit. 2024-05-10].

[29] ZAKARYAN, Mushegh. *Understanding libuv: The Powerhouse Behind Node.js*. Online. In: Medium. 2023. Dostupné z: https://zmushegh.medium.com/understanding-libuv-the-powerhouse-behind-node-js-b5349c8f0d75. [cit. 2024-05-10].

[30] ZANINI, Antonello. *Express.js adoption guide: Overview, examples, and alternatives*. Online. In: LogRocket Blog - Resources to Help Product Teams Ship Amazing Digital Experiences. 2023. Dostupné z: https://blog.logrocket.com/express-js-adoption-guide/. [cit. 2024-05-10].

[31] CAMUS, Ariel. *Introduction to ReactJS: A Guide for Beginners*. Online. In: Microverse. 2022. Dostupné z: https://www.microverse.org/blog/introduction-to-reactjs-a-guide-for-beginners. [cit. 2024-05-10].

[32] RAWAT, Prateek a MAHAJAN, Archana. *ReactJS: A Modern Web Development Framework*. Online. International Journal of Innovative Science and Research Technology. 2020, roč. 5, č. 11, s. 1-2. ISSN 2456-2165. Dostupné z: https://ijisrt.com/assets/upload/files/IJISRT20NOV485.pdf. [cit. 2024-05-10].

[33] *What is React.js? - Brief Framework Intro / Overview*. Online. Sanity: The Composable Content Cloud. 2024. Dostupné z: https://www.sanity.io/glossary/react-js. [cit. 2024-05-10].

[34] GUPTA, Mohith. *UseEffect in Functional Component Vs Class Component Methods*. Online. In: Stackademic. 2023. Dostupné z: https://blog.stackademic.com/useeffect-in-functional-component-vs-class-component-methods-also-starting-a-learn-react-hooks-626923fac5b5. [cit. 2024-05-10].

[35] SIVAN, Vishnu. *Introduction to Vite: the lightning-fast module bundler*. Online. In: DEV Community. 2022. Dostupné z: https://dev.to/codemaker2015/introduction-to-vite-the-lightning-fast-module-bundler-32mm. [cit. 2024-05-10].

[36] Why Vite. Online. Vite | Next Generation Frontend Tooling. 2019. Dostupné z: https://vitejs.dev/guide/why.html. [cit. 2024-05-10].

[37] JAIN, Royal. *Vite: 90% faster Webpack alternative*. Online. In: Medium. 2024. Dostupné z: https://medium.com/@royaljain0203/vite-90-faster-webpack-alternative-7eeba6e284af. [cit. 2024-05-10].

[38] *Signal Protocol Typescript Library (libsignal-protocol-typescript)*. Online. Npm. 2023. Dostupné z: https://www.npmjs.com/package/@privacyresearch/libsignal-protocol-typescript. [cit. 2024-05-10].

[39] *What Is a Document Database?* Online. Cloud Computing Services - Amazon Web Services (AWS). Dostupné z: https://aws.amazon.com/nosql/document/. [cit. 2024-05-10].

[40] MAMUN, Iftekher. *SQL Database vs MongoDB Part 2*. Online. In: Medium. 2019. Dostupné z: https://medium.com/@imamun/sql-database-vs-mongodb-part-2-ab-bbf1d94bd3. [cit. 2024-05-10].

[41] *Indexes — MongoDB Manual*. Online. MongoDB: The Developer Data Platform. 2023. Dostupné z: https://www.mongodb.com/docs/v5.3/indexes/. [cit. 2024-05-10].

[42] *BSON Types - MongoDB Manual v7.0*. Online. MongoDB: The Developer Data Platform. 2024. Dostupné z: https://www.mongodb.com/docs/manual/reference/bson-types/. [cit. 2024-05-10].

[43] KUKIC, Ado a VLAEVA, Stanimira. *MongoDB & Mongoose: Compatibility and Comparison*. Online. In: MongoDB Developer Center. 2021, Updated Apr 02, 2024. Dostupné z: https://www.mongodb.com/developer/languages/javascript/mongoose-versus-nodejs-driver/. [cit. 2024-05-10].

[44] *WebSocket Vs HTTP.* Online. Wallarm | Integrated App and API Security Platform. 2024. Dostupné z: https://www.wallarm.com/what/websocket-vs-http-how-are-these-2-different. [cit. 2024-05-10].

[45] *Writing WebSocket servers*. Online. MDN Web Docs. 2024. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API/Writing_WebSocket_servers. [cit. 2024-05-10].

[46] KONEČNÝ, Ondřej. *Jak funguje CSS Box Model*. Online. In: Frontend Garden — Český online magazín o webovém frontendu. 2020. Dostupné z: https://frontend.garden/clanky/jak-funguje-css-box-model/. [cit. 2024-05-10].

[47] *Introduction to JSON Web Tokens.* Online. JSON Web Tokens - jwt.io. 2024. Dostupné z: https://jwt.io/introduction. [cit. 2024-05-10].

[48] *Lidí trpících syndromem vyhoření přibývá. Někteří zaměstnavatelé se mu snaží předcházet*. Online. In: ČT24 — Nejdůvěryhodnější zpravodajský web v ČR | Česká

televize. 2023. Dostupné z: https://ct24.ceskatelevize.cz/clanek/domaci/lidi-trpicich-syndromem-vyhoreni-pribyva-nekteri-zamestnavatele-se-mu-snazi-predchazet-4257. [cit. 2024-05-10].

[49] ŠENKEŘÍK, Roman. *Kryptologie: Symetrická, asymetrická a hybridní kryptografie. Klasifikace šifer*. Univerzita Tomáše Bati ve Zlíně, 2020. PDF.

[50] *Frekvenční analýza*. Online. In: Wikipedia: the free encyclopedia. San Francisco (CA): Wikimedia Foundation, aktualizováno 4. 12. 2023. Dostupné z: https://cs.wikipedia.org/wiki/Frekven%C4%8Dn%C3%AD_anal%C3%BDza. [cit. 2024-05-10]

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

IM	Instant Messaging.
DES	Data Encryption System
AES	Advanced Encryption System.
E2EE	End-to-End Encryption
IP	Internet Protocol
2FA	Two-factor Authentication
X3DH	Extended Triple Diffie-Hellman
KDF	Key derivation function
JWT	JSON Web Token
DOM	Document Object Model
HTML	Hyper Text Markup Language
TSC	TypeScript Compiler
API	Application Programming Interface
HTTP	Hypertext Transfer Protocol
JSX	JavaScript Syntax eXtension
JSON	JavaScript Object Notation
ODM	Object Data Modeling
ORM	Object Relational Mapping
WS	WebSocket
CSS	Cascading Style Sheets

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH

PŘÍLOHA P I: P

Obsah CD přiloženého k diplomové práci:

- text diplomové práce
- zdrojové kódy vytvořené aplikace
- návrh aplikace: drátěný model, diagram případů užití a datový model