Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informatiky a kvantitativních metod

Aplikace pro podporu komunikace se zahraničními partnery

Bakalářská práce

Autor: Vojtěch Navrátil
Studijní obor: Aplikovaná informatika

Vedoucí práce: doc. RNDr. Petra Poulová, Ph.D.

Hradec Králové Srpen 2020

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a s použitím uvedené literatury.

vlastnoruční podpis

V Hradci Králové dne 14.8.2020 Vojtěch Navrátil

Poděkování:
Děkuji vedoucí bakalářské práce doc. RNDr. Petře Poulové, Ph.D. za metodické vedení práce a cenné rady.

Anotace

Cílem práce je návrh a implementace aplikace pro evidenci kontaktů a umožnění komunikace mezi nimi. Práce začíná analýzou stěžejních částí nejpoužívanějších komunikačních aplikací. V návrhové části jsou poznatky z analýzy spojeny s vlastní iniciativou za účelem vytvoření požadavků na aplikaci a k návrhu jejích částí. Implementační část popisuje technologie použité pro vývoj aplikace, postupy při implementaci aplikace, její klíčové funkcionality a logiku, se kterou byly realizovány. V implementační části je také snaha nacházet alternativní postupy pro realizaci funkcionalit a porovnat je mezi sebou.

Annotation

Title: Application supporting communication with foreign partners

The aim of this thesis is to design and implement application for keeping records of contacts and allowing communication between them. The thesis begins with analysis of key functionalities of the most used communication applications. Obtained knowledge from analysis and own initiative are used in the design section for creating requirements and designing the application. The implementation part of this thesis describes technologies used for application development, the process of implementation, key functionalities of application and the logic behind them. There is also an effort of finding alternative ways to realize functionalities and comparing them.

Obsah

1 Úvod ... 1

2 Analýza existujících aplikací ... 2

2.1 Facebook Messenger ... 2

2.2 MS Teams ... 3

2.3 Slack ... 4

3 Návrhová část ... 6

3.1 Obecné požadavky ... 6

3.2 Funkční požadavky ... 6

3.3 Návrh grafického rozhraní ... 7

3.4 Návrh databáze ... 8

4 Implementace aplikace ... 12

4.1 Použité technologie ... 12

4.1.1 Javascript ... 12

4.1.2 VueJS ... 12

4.1.3 NodeJS ... 14

4.1.4 Socket.IO ... 15

4.1.5 JSON Web Token ... 15

4.1.6 MongoDB databáze a Mongoose plugin ... 16

4.2 Příprava aplikace k vývoji ... 18

4.2.1 Založení a spuštění projektů ... 18

4.2.2 Příprava serverové části ... 18

4.3 Grafické rozhraní ... 20

4.4 Popis klíčových funkcionalit aplikace ... 22

4.4.1 Přihlášení, udržování údaje o přihlášení a zabezpečení dat ... 22

4.4.2 Evidence kontaktů ... 23

4.4.3 Editace osobních dat 26
4.4.4 Vytváření konverzací 27
4.4.5 Zasílání zpráv 27
4.4.6 Mazání zpráv 29
4.4.7 Informace o přijetí zprávy a notifikace 29
4.4.8 Skupinové konverzace 31
4.4.9 Nahrávání souborů 34
4.4.10 Stahování souborů 34
4.4.11 Stav aktivity uživatele 36
5 Závěr 38
6 Seznam použité literatury 40
7 Přílohy 42

Seznam obrázků

Obr. 1 Náčrt struktury grafického rozhraní 8
Obr. 2 Diagram návrhu databáze MySQL 9
Obr. 3 Ukázka grafického rozhraní aplikace 21
Obr. 4 Ukázka způsobu evidence kontaktů 24
Obr. 5 Ukázka detailních informací kontaktu 26
Obr. 6 Ukázka rozhraní pro editaci osobních údajů 27
Obr. 7 Ukázka rozlišení odesílatelů ve skupinové konverzaci 32
Obr. 8 Ukázka grafického rozhraní pro administraci skupin 33

1 Úvod

Bakalářská práce se zabývá návrhem a tvorbou aplikace pro evidenci kontaktů například ze zahraničních organizací a komunikaci s nimi. V dnešní době je komunikace po internetu každodenní rutinou většiny populace a může být problém najít tu správnou aplikaci pro daný účel. Existuje spousta aplikací, které poskytují možnost komunikace a na první pohled tedy dělají totéž, ale existuje mnoho faktorů, které od sebe aplikace výrazně odlišují a mění uživatelskou zkušenost.

Práce je rozdělena do tří částí. Analytická část je zaměřena na analýzu zejména grafického rozhraní, uživatelské přívětivosti, systému evidence kontaktů a způsobu realizace komunikace soukromé i skupinové vybraných existujících aplikací. V návrhové části jsou definovány požadavky na aplikaci, návrh grafického rozhraní a databáze. Implementační část se pak věnuje popisu implementace aplikace a jejích funkcí.

2 Analýza existujících aplikací

Pro analýzu byly vybrány jedny z nejpopulárnějších aplikací používaných pro komunikaci a evidenci kontaktů. Analyzováno bylo zejména grafické rozhraní a způsob použití klíčových funkcí aplikace z pohledu uživatele. U komplexnějších aplikací byla analýza soustředěna na funkcionality důležité pro tuto práci.

V dnešní době existuje mnoho aplikací, které zprostředkovávají komunikaci, ale i přesto že jejich účel je velmi podobný, uživatelská zkušenost se může kompletně lišit. Za základní rozdělení takovýchto aplikací by se dalo považovat rozdělení podle použití – neformální použití pro běžné uživatele a použití pro business a firmy.

Neformální aplikace mají obecně méně případů užití než aplikace pro business. Soustředí se hlavně na grafický design, uživatelskou přívětivost a zábavné funkce, které uživatelům zpříjemňují používání aplikace.

Aplikace pro podporu komunikace v organizaci jsou většinou mnohem komplexnější a nabízejí mnohem více funkcionalit jako například propojení s externími aplikacemi. Pro využití jejich plného potenciálu je však zpravidla nutné mít zakoupený plán, jehož cena je určena druhem plánu a počtem aktivních uživatelů.

2.1 Facebook Messenger

Facebook Messenger (běžně označován pouze jako Messenger) je jednou z nejpoužívanějších aplikací pro komunikaci. Vyvinut z původního Facebook chatu, pro vzrůstající popularitu jak sociálních médií, tak instantní chatové komunikace. Dnes je Messenger v kompletně separátní formě dostupný i v podobě samostatné aplikace pro mobilní zařízení. [1]

Messenger je určen pro běžné uživatele. Jeho jednoduchý design a uživatelské rozhraní znamenají uživatelskou přívětivost. Ve webovém rozhraní se v levé části nachází seznam konverzací. Důležité je zmínit že soukromé i skupinové konverzace jsou sjednoceny v jednom seznamu a pokud uživatel chce využít funkci vyhledávání, které se soustředí na vyhledávání ostatních uživatelů, musí znát

minimálně dvě písmena ze jména skupinové konverzace, aby ji pomocí filtru vyhledal.

Prostřední část je vyplněná konverzací. Konverzace je řešená klasickým způsobem – nejnovější zprávy se nachází dole, vypisuje se jich pouze 40 nejnovějších a pro načtení starších zpráv je potřeba vyjet posuvníkem nahoru. V pravé části aplikace je v soukromé konverzaci k nalezení odkaz na Facebookový profil kontaktu, funkce jako vyhledávání v konverzaci či změna barvy konverzace, nastavení notifikací a seznam zaslaných souborů a obrázků. Ve skupinovém chatu se navíc nachází změna jména chatu a administrace členů.

Evidence kontaktů jako taková není řešena v samotném messengeru. Messenger sice měl funkci využívat aplikaci bez Facebook účtu, za použití telefonního čísla, ale tato funkce byla koncem roku 2019 odstraněna[2]. Nyní je tedy Facebook účet nutností pro používání Messenger, a tudíž Messenger čerpá kontakty z Facebooku. Pomocí vyhledávače v Messengeru lze najít uživatele Facebooku a zahájit s ním konverzaci. Pokud má uživatel zájem vidět detail kontaktu a zobrazit si informace o něm, bude přesměrován na jeho Facebookový profil.

2.2 MS Teams

MS Teams je pracovní prostředí od firmy Microsoft určené pro organizace. Nabízí širokou škálu funkcí jako třeba formy textové komunikace, videokonference, kolaborace na souborech nebo integrace s kalendářem a jinými aplikacemi. Pro používání MS Teams je nutné mít Office 365 účet s odpovídajícím plánem. [3]

Chatovou komunikaci MS Teams poskytuje jak soukromou, tak skupinovou. Stejně jako u Messengeru nejsou odděleny, ale nachází se v jednom seznamu, ve kterém se dá filtrovat. Chat má standardní podobu, nejnovější zprávy se řadí odspoda a oddělení odeslaných a přijatých zpráv je řešeno jejich umístěním na pravou a levou stranu. Sdílení souborů v chatu funguje jinak než v běžných aplikacích. Soubor se uloží na uživatelův OneDrive pro byznys a je automaticky sdílen se všemi uživateli v konverzaci. Na souborech lze tedy online spolupracovat a měnit je.

Pro komunikaci MS Teams nabízí kromě soukromého a skupinového chatu komplexnější funkcionalitu, kterou nazývá týmy. Tým je pracovní prostor pro skupinu kontaktů. V týmu se dají vytvořit kanály pro rozdělení komunikace například podle témat nebo částí projektu. Kanál ovšem nefunguje přímo jako skupinový chat, přidávají se do něj příspěvky, kterým Microsoft říká konverzace, a diskuze probíhá pod nimi. Tato struktura připomíná spíše Facebookové příspěvky a komentáře než skupinový chat. Sdílení souborů v kanálu je podobné jako v chatu s rozdílem, že se neukládá na OneDrive ale do SharePoint složky týmu.

Evidence kontaktů v MS Teams je komplikovanější, než by uživatel očekával. Samotná aplikace nemá v levém menu položku pro kontakty. Kontakty jsou k nalezení pod položkami Chat a Hovory. Tam se ovšem nenachází seznam s uživateli v aplikaci, které by si uživatel mohl prohlédnout. Kontakty se musí přidávat po jednom a vyhledávají se podle jména. V položce Chat jdou přidávat do kontaktů pouze uživatelé, kteří jsou ve stejné organizaci MS Teams. V položce hovory lze přidat i jiný kontakt, ale lze k němu uložit pouze telefonní číslo. Detail kontaktu se zobrazí po najetí myší, má podobu malé karty a uvádí pouze základní informace spolu s tlačítky pro základní akce jako například zahájení konverzace nebo hovoru. Při analýze nebyla zjištěna možnost zobrazení detailnějších informací o kontaktu. Další pozoruhodnou informací o kontaktech je, že MS Teams nepodporuje integraci s Outlook kontakty, což je nedostatek, na který si uživatelé často stěžují.

Obecně má grafické rozhraní podobnou strukturu jako předchozí analyzovaná aplikace. V levé části se nachází ovládací prvky, které určují, co bude obsahem zbylé větší části obrazovky. Protože je MS Teams komplexní aplikací nabízející mnoho funkcí, navigace k některým službám může být pro mnohé uživatele obtížnější.

2.3 Slack

Slack je platforma vyvíjená společností Slack Technologies. Podobně jako MS Teams byl Slack navržen pro usnadnění komunikace mezi kolegy v organizaci. Hlavním způsobem komunikace Slacku jsou kanály. Kanály mohou být veřejné nebo soukromé. Obsah veřejných kanálu je viditelný pro všechny uživatele a

kdokoliv se do nich může přidat. Veřejné kanály tedy slouží jako spíše rozdělení komunikace, a nikoliv jako rozdělení uživatelů. Soukromé kanály nejsou viditelné v seznamu kanálů a pro zobrazení jejich obsahu musí být uživatel do kanálu přidán. Kanály jsou identifikované názvem, který musí být unikátní.

Slack umožňuje i soukromou a skupinovou komunikaci. Skupinové konverzace se používají ovšem jen ve velmi málo případech, pro veškerou skupinovou komunikaci je doporučeno používat kanály.

Specialitou Slacku je jednoduchost a unifikace celého prostředí. Například funkce vyhledávání dokáže vyhledávat skrze všechny dostupné komunikační kanály a dají se v ní použít modifikátory pro zúžení vyhledávání. Funkce procházení souborů také poskytuje výčet souborů sdílených kdekoliv v aplikaci.

Jednoduchost Slacku platí i pro jeho grafické rozhraní. Navigace skrze aplikaci se oproti MS Teams zdá mnohem snadnější. V levé části aplikace se opět nachází ovládací prvky obsahu zbylého prostoru obrazovky. Patří mezi ně i seznam aktivních kanálů a konverzací. Kompletní seznam kanálů se dá nalézt v prohlížeči kanálů. Podobně se všechny kontakty dají nalézt v seznamu kontaktů, který ale graficky není navržen pro prohlížení, nýbrž spíše pro vyhledávání. Kontakty jsou zobrazovány v mřížce, na stránku se jich nevejde mnoho a jsou k nim uvedeny jenom tři základní informace. Grafické rozhraní komunikace v kanálech působí až prostě. Zprávy nemají výrazné ohraničení a změna odesílatele zprávy je vertikálně oddělena pouze tučným jménem odesílatele. To může učinit kanál při větším množství zpráv nepřehledným.

Stejně jako MS Teams je Slack aplikací určenou pro byznys a pro využití plného potenciálu aplikace je třeba platit nějaký plán, jehož cena se odvíjí od druhu plánu a počtu uživatelů.

3 Návrhová část

Tato kapitola se zabývá návrhem aplikace a definicí požadavků na základě analýzy a vlastní úvahy.

3.1 Obecné požadavky

Jedním z hlavních obecných požadavků aplikace je její snadná dostupnost, a proto bude realizována jako webová aplikace. Webová aplikace umožňuje uživateli přístup k datům a funkcím odkudkoliv a z jakéhokoliv zařízení bez nutnosti instalace.

Mezi další obecné požadavky patří:

- Dynamičnost aplikace (dynamicky přepisovaný obsah za běhu aplikace bez nutnosti aktualizace stránky)
- Jednoduché a přívětivé grafické rozhraní
- Zabezpečení uživatelských dat

3.2 Funkční požadavky

Dvěma základními účely této aplikace jsou evidence kontaktů a realizace komunikace mezi nimi. Tyto dva základní účely však doprovází mnoho menších funkcionalit, které jsou potřebné ke zlepšení uživatelského zážitku z aplikace.

Soubor funkčních požadavků je následující:

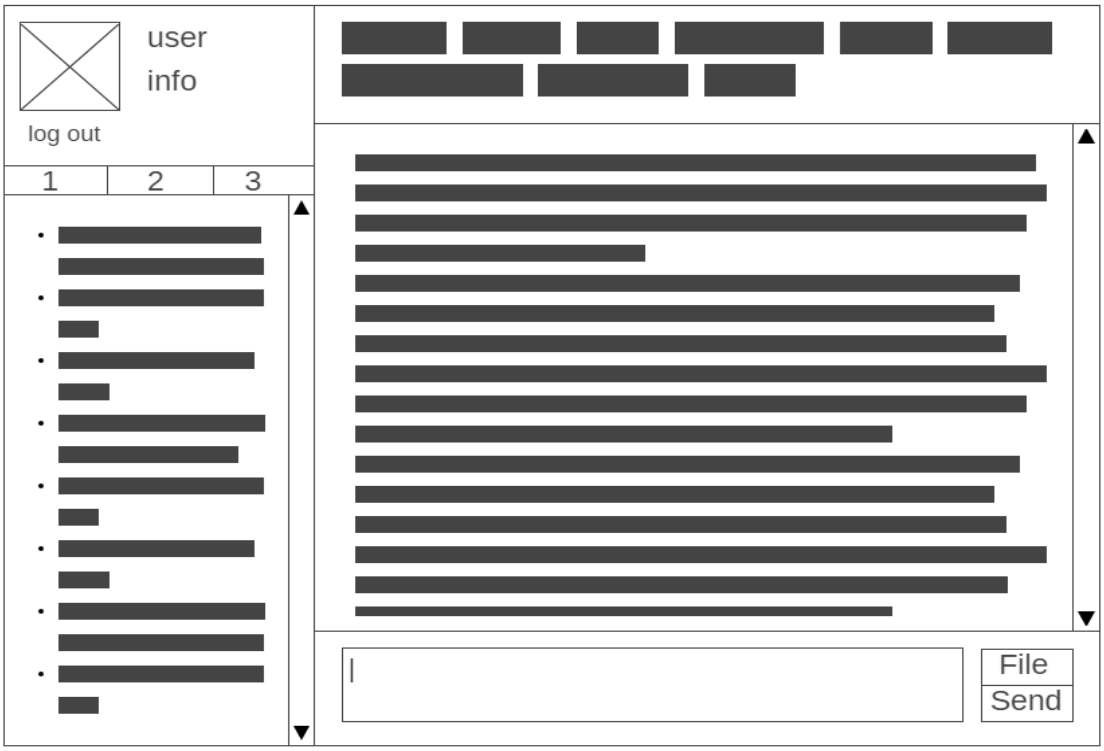
- Uživatel se bude moci registrovat a jeho heslo bude bezpečně uloženo v databázi v podobě hashe
- Aplikace bude mít systém evidence kontaktů takový, aby byl pro uživatele pohodlný a nemusel přidávat každý kontakt po jednom
- V kontaktech bude možné filtrovat podle jednoho nebo více kritérií
- Aplikace bude poskytovat možnost zobrazení detailnějších informací o kontaktu
- Aplikace bude umožňovat komunikaci a sdílení souborů
- Komunikace bude poskytována ve formě soukromé i skupinové

- V konverzacích bude možno vyhledávat zprávy a zobrazit seznam sdílených souborů
- Skupinová konverzace bude mít nástroj pro správu uživatelů a nastavení jejího názvu
- Ve skupinové konverzaci bude existovat identifikátor pro odesílatele zprávy
- Skupinová konverzace bude zaznamenávat a ohlašovat změny jako „Uživatel X opustil skupinu“
- Aplikace bude upozorňovat na příchozí zprávy
- Aplikace bude mít vizuální indikátory pro nepřečtené zprávy
- Uživateli bude poskytnuta možnost ztlumit upozornění na příchozí zprávy jak globálně, tak pro každou konverzaci zvlášť

Aplikace bude navržena jako on-premises, což znamená, že bude běžet na serveru organizace, která ji vlastní. Vlastnická organizace bude umožňovat registraci do aplikace ostatním organizacím, se kterými chce komunikovat, pomocí whitelistu domén. Uživatelé z vlastnické organizace v aplikaci uvidí všechny registrované uživatele. Vlastnická organizace ovšem pravděpodobně nebude chtít, aby ostatní organizace viděly, s kým spolupracuje. Ostatní organizace proto uvidí pouze kolegy ze stejné organizace a uživatele z vlastnické organizace.

3.3 Návrh grafického rozhraní

Z analýzy vyplynulo, že pro tento typ aplikací existuje standard pro rozložení grafického rozhraní. V analyzovaných aplikacích se po levé straně nacházely ovládací prvky, kterými se dal ovládat obsah hlavní části obrazovky. Pomocí nástroje Wireframe.cc byl vytvořen náčrt struktury rozložení aplikace, který si lze prohlédnout na obrázku 1.



Obr. 1 Náčrt struktury grafického rozhraní

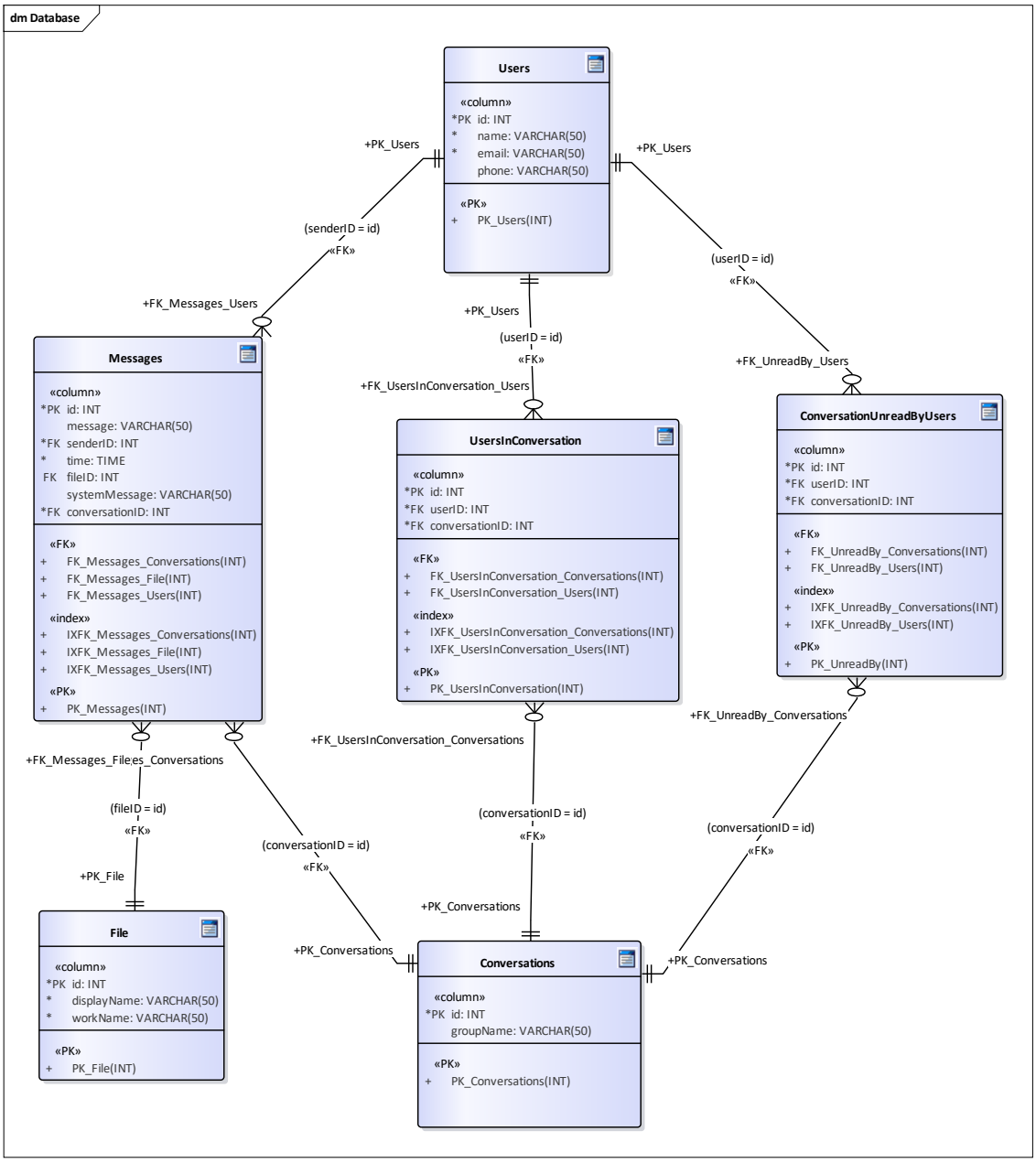
3.4 Návrh databáze

V aplikaci je třeba udržovat dva druhy dat. Data o uživatelích a data o konverzacích. Data o uživatelích jsou přímočará, neobsahují žádné komplexní struktury, pouze jednoduché typy jako jméno, email, telefonní číslo nebo třeba pracoviště. Informace o konverzacích ale nabývají složitějších struktur. O nich je potřeba udržovat:

- Seznam uživatelů v konverzaci
- V případě skupinové konverzace její název
- Seznam uživatelů, kteří mají v konverzaci nepřečtenou zprávu
- Všechny zprávy v konverzaci a u nich:
    - Identifikaci odesílatele zprávy
    - Samotný text zprávy
    - Čas odeslání
    - Pokud byl odeslán soubor tak informaci o něm

- Případnou systémovou zprávu (např. „Uživatel opustil konverzaci“)

Při návrhu byly v úvaze dvě databáze – MySQL a MongoDB. MySQL je nejpopulárnější technologií v kategorii relačních SQL databází a MongoDB je nejpopulárnější NoSQL technologie. [5]



Obr. 2 Diagram návrhu databáze MySQL

Jak lze vidět na obrázku 2, záznam konverzací by se musel skládat z pěti různých tabulek, zejména proto, že je potřeba u záznamů uchovávat datový typ

pole, který se nedá v MySQL jednoduše realizovat. Pro vyjádření tohoto datového typu by bylo třeba vytvořit novou tabulku. Například pro vyjádření, že v konverzaci se nachází x uživatelů, by bylo nutné do nové tabulky přidat x záznamů obsahujících ID uživatele a ID konverzace, do které patří. Získávání dat by pak probíhalo na serverové straně pomocí operátoru JOIN. Druhým řešením by bylo pole serializovat a uložit ho jako řetězec. To by však znamenalo, že při jakékoliv změně by se muselo přepsat celé pole v databázi. Server by si musel řetězec převést zpět na pole, provést změny, opět serializovat a až poté uložit.

V porovnání s možnostmi, jak tento problém vyřešit v MongoDB se však tyto dvě metody prokazují jako velmi nepraktické. V MongoDB lze ukládat i komplexní datové typy, a tak lze celou konverzaci uložit jako jediný záznam obsahující pole, a dokonce i pole objektů. Datové schéma pro kolekci konverzací vytvořené v Mongoose vypadá následovně:

```
const conversationSchema = new Schema({
    groupName: {
        type: String,
        required: false,
        default: ""
    },
    users: {
        type: Array,
        required: true
    },
    messages: {
        type: Array,
        required: true
    },
    unreadBy: {
        type: Array,
        required: false
    }
})
```

Záznam má tak danou základní strukturu, ale stále existuje volnost. Například pole *messages* může mít jinou strukturu, pokud se jedná o zprávu textovou či o zprávu obsahující informace o poslaném souboru. Práce s těmito záznamy je také mnohem snadnější. Dotazy pro získávání dat není třeba modifikovat JOIN operátorem a s polem se dá pracovat na databázové úrovni. MongoDB totiž nabízí operátory jako *push*, *pull*, *each* či *in*, které umožňují provádět

operace s poli přímo v dotazu. Další výhodou MongoDB je, že pracuje s daty v podobě JSON objektů, který je standardem v Javascriptu, ve kterém bude většina aplikace napsaná. Toto sjednocení datových typů je příjemné pro vývojářovu orientaci a práci s daty. Z těchto důvodů byla tedy zvolena databáze MongoDB.

4 Implementace aplikace

Tato kapitola se zaměřuje na popis technologií použitých při vývoji, přípravu a založení projektu a také popisu a rozebrání funkcionalit aplikace. Při implementaci aplikace byla snaha o nacházení alternativních způsobů, jak realizovat dané funkcionality a jejich porovnání.

4.1 Použité technologie

4.1.1 Javascript

Javascript je multiplatformní skriptovací jazyk, používaný k realizaci dynamičnosti webových stránek. JavaScript je client-side, takže běží na straně uživatele v prohlížeči. Umožňuje tedy měnit obsah stránek za běhu, oproti server-side jazykům jako je například PHP, kde je pro změnu stránky nutné ji aktualizovat. Pomocí JavaScriptu tedy lze na stránce jednoduše vytvořit třeba dropdown menu, skrývat a zobrazovat komponenty na základě eventů které může spustit uživatel (např. kliknutí myší nebo stisknutí klávesy) nebo provádět výpočty jako sumu celkové ceny po přidání zboží do košíku na e-shopu ihned, namísto posílání požadavku na server, načítání stránky a nucení uživatele čekat. [6]

JavaScript ale nemusí být využit přímo na změnu dat na stránce, ale je vhodný i k její validaci. JavaScript totiž podporuje regulární výrazy, a tak je snadné zjistit, zda uživatel zadal platnou e-mailovou adresu a nezatěžovat server s dotazem, který by byl neplatný.

Javascript interpretovaný jazyk, tzn. že se za běhu překládá do strojového kódu, kterému rozumí počítač. Je dynamicky typovaný, takže typy proměnných lze za chodu měnit. Je také objektově orientovaný a využívá funkcionální paradigma, takže je možné do proměnné uložit i celou funkci. [6]

4.1.2 VueJS

VueJS (nebo jenom Vue) je frontendový progresivní framework pro tvorbu uživatelských rozhraní uvedený na trh stejnojmennou společností. Používá model založený na komponentách, díky kterému je možné oddělit logiku i layout jedné části stránky a využít ji na jakémkoliv jiném místě. [7]

Struktura Vue souboru je tvořena ze tří částí. První je část šablony, kde se pomocí html tagů a takzvaných Vue direktiv definuje obsah stránky a jeho rozložení. Druhou částí je skript, kde se nachází javascriptový kód a poslední je část stylů s CSS kódem.

Ve Vue je možné pomocí direktiv svázat komponentu v části šablony s javascriptovými daty. Toto vázání má několik podob a využití, například:

- `{{data}}` – pro vypsání javascriptových dat v šabloně
- `v-bind:data` – pro předání dat podřízené komponentě která s nimi může pracovat podle vlastní logiky
- `v-bind:style` – data lze svázat i k atributům html elementů, komponentě lze snadno přidělit například styl, který se může měnit
- `v-model` – obousměrné vázání dat používané u *input* elementů

Všechny tato vazby dat jsou reaktivní. Pokud se změní data, která jsou k něčemu vázaná, Vue automaticky předá komponentě změněná data a překreslí ji. [8]

Mezi další užitečné funkce Vue patří kondicionální renderování. Pomocí direktiva *v-if* nebo *v-show* lze přímo v šabloně definovat podmínku pro zobrazení elementu. V následující ukázce kódu je kondicionální renderování demonstrováno na elementu *div*. Pokud je proměnná *showDiv true*, na stránku se vypíše obsah elementu.

```
<div v-if="showDiv">Hello</div>
```

Tato direktiva jsou samozřejmě také reaktivní. Pokud se za běhu změní hodnota proměnné *showDiv*, Vue bude reagovat a div schová nebo vykreslí na základě její hodnoty. Rozdíl mezi *v-if* a *v-show* je ten, že element s *v-show* je vždy renderován a je pouze schováván pomocí CSS. Hodí se tedy pro často přepínané elementy. Oproti tomu *v-if* neprovede žádný render pokud je podmínka vyhodnocena jako nepravdivá a navíc zajišťuje, že všechny posluchače a vnitřní komponenty jsou řádně zničeny a vytvořeny při změně podmínky.[8]

Vue používá virtuální DOM (Document Object Model). DOM je objektová reprezentace html dokumentu v paměti, vytvořená při načtení stránky. DOM má

stromovou strukturu, obsahuje html elementy v podobě objektů a slouží k interakci a manipulaci s těmito elementy (například změna stylování či obsahu). Některé aplikace mohou mít velké množství elementů a práce s velkým DOM stromem se stane neprakticky pomalou. Proto existuje Virtuální DOM. Virtuální DOM je reprezentace běžného DOM javascriptovými objekty a funguje jako prostředník mezi Vue a DOM. Vue pracuje s virtuální verzí Document Object Modelu a provádí změny na něm. Po provedení změn se virtuální DOM porovná s DOM, Vue zjistí, kde se tyto Document Object Modely liší a namísto aktualizace celého DOM aktualizuje pouze potřebnou část. [9]

4.1.3 NodeJS

NodeJS (dále jenom Node) je open-source prostředí, které umožňuje spustit javascriptový kód mimo prohlížeč. Je používaný zejména pro psaní webových serverů. Node poskytuje bohatou knihovnu modulů, které pomáhají zjednodušit vývoj aplikace. Moduly jsou něco jako javascriptové knihovny. Jsou to části kódu, například kolekce funkcí, jsou odděleně uloženy v javascriptovém souboru a je možné je importovat a použít kdekoliv v kódu.[10]

Node využívá správce balíčků zvaný npm, který nabízí přes 800 000 balíčků. Balíček je sada jednoho nebo více modulů seskupených do jedné složky. Pomocí npm lze jednoduše nainstalovat jakýkoliv balíček, a přidat ho do seznamu závislostí. Tento seznam definuje, jaké balíčky jsou potřeba pro běh aplikace. Správce balíčků dokáže doinstalovat všechny balíčky definované v seznamu závislostí jednoduchým příkazem *npm install*. [11]

Příklady balíčků použitých v této práci jsou:

- ExpressJS – nástroj usnadňující vývoj webové aplikace v prostředí Node
- nodemon – automaticky restartuje server po změně ve zdrojovém kódu, to je užitečné při vývoji aplikací
- mongoose – nástroj pro práci a komunikaci s MongoDB
- body-parser – parsuje těla požadavků do podoby se kterou se pohodlněji pracuje

- cookie-parser – podobně jako body-parser parsuje cookies do podoby pohodlnější pro práci
- bcrypt – nástroj pro hashování
- jsonwebtoken – umožňuje práci s JWT

Node je asynchronní a událostmi řízený. Příkladem asynchronního programování může být práce se soubory. Node odešle požadavek na otevření souboru a na rozdíl od synchronního přístupu nemusí nečinně čekat na odpověď. Node může pokračovat v práci a vykonávat další funkce, než mu systém pošle odpověď s obsahem souboru. [12]

4.1.4 Socket.IO

Socket.IO je javascriptová knihovna umožňující obousměrnou komunikaci mezi klientem a serverem založenou na událostech v reálném čase. Socket.IO využívá komunikačního protokolu WebSocket. Pro fungování Socket.IO je potřeba mít nainstalovanou jeho serverovou implementaci na Node serveru a zároveň klientskou implementaci, která je dostupná pro více jazyků.[13]

Klient a server mezi sebou komunikují na bázi událostí. Je potřeba mít definovaný posluchač na každou očekávanou událost. Posluchač musí obsahovat identifikátor (název události) a funkci, která se při nastání události provede.

Socket.IO podporuje funkcionalitu místností. Je možné mít místnost, v ní mít připojeno několik klientů a při události odeslat informace o události či data jenom klientům připojeným v místnosti. To je vhodné například pro skupinové chaty.[13]

4.1.5 JSON Web Token

JSON Web Token neboli JWT je kompaktní způsob pro bezpečnou komunikaci mezi dvěma stranami v podobě JSON objektu. Struktura tokenu je tvořena ze tří částí oddělených tečkou. Tyto části jsou: hlavička (header), data (payload) a podpis (signature). Hlavička a data jsou JSON objekty, ale v tokenu jsou reprezentovány jako Base64Url, do kterého jsou zakódovány.[14]

Hlavička obvykle obsahuje dva údaje týkající se vytvoření podpisu. Prvním je typ tokenu, tedy JWT. Druhým je název algoritmu, který má být použitý pro

podpis. To může být například HS256, SHA256, RSA apod. Část s daty může obsahovat předdefinované údaje jako třeba čas vypršení nebo předmět. Dále může obsahovat i vlastní data jako kupříkladu jméno držitele tokenu. Podpisová část je tvořena kombinací hlavičkové a datové části převedené do Base64Url a následným použitím algoritmu definovaným v hlavičce a takzvaným tajemstvím, které může být v podobě textového řetězce a musí ho znát obě strany. [14]

Použití JWT může být různé, ale základní myšlenkou je, že po úspěšném přihlášení nebo jiné formě autentizace se uživateli vrátí token, který pak používá pro komunikaci se serverem, když po něm vyžaduje chráněná data. Server musí znát řetězec tajemství, aby token ověřil. Je důležité zdůraznit, že obsah tokenu se dá snadno přečíst. Token se používá jako ověření, že uživatel je tím, za koho se vydává, a nikoliv pro posílání chráněných dat. [14]

4.1.6 MongoDB databáze a Mongoose plugin

MongoDB je univerzální distribuovaná databáze vytvořená pro vývojáře moderních aplikací. MongoDB je dokumentově orientovaná databáze, což znamená že data uchovává v podobě JSON dokumentů. [15]

V dokumentově orientované databázi je koncept řádku nahrazen dokumentem, který je mnohem flexibilnější, dovoluje ukládat vnořené objekty či pole, a tak umožňuje reprezentovat komplexní hierarchické vztahy jediným záznamem. Skupina dokumentů je pak uložená v kolekci. Každý dokument má při vytvoření přiřazený svůj klíč pojmenovaný *_id*, který je v rámci kolekce unikátní.[16]

Kolekce mají dynamická schémata. To znamená že dokumenty v jedné kolekci mohou mít různé podoby. Na rozdíl od relační SQL databáze, kde jsou jasně definované názvy sloupců a jaké datové typy mají obsahovat, dokumentově orientovaná MongoDB může ve stejné kolekci mít zároveň například tyto dva dokumenty:

```
{‘name‘ : ‘John‘},
{‘lottery‘ : [1,2,3]}
```

Tyto dva záznamy mají nejen jiný název klíče, ale dokonce jsou i jiného datového typu – jeden je textový řetězec a druhý je pole čísel. Toto je jedna z vlastností, která

dělá MongoDB tak flexibilní. Nutno podotknout, že míchání struktur v kolekci s myšlenkou, že „je zbytečné vytvářet více kolekcí, když to jde v jedné," není doporučeno. Jenom to, že to databáze umožňuje, neznamená, že to je dobrý nápad. Pro rozdílná datová schémata je vhodné vytvořit příslušné kolekce. [16]

Mongoose je balíček pro Node, který poskytuje funkce usnadňující práci s MongoDB. Umožňuje například práci s MongoDB v podobě objektu nebo definování struktury dokumentů v kolekci pomocí schémat. Definice schématu může mít následující podobu:

```
const userSchema = new Schema({
    name: {
        type: String,
        required: true
    },
    email: {
        type: String,
        required: true,
        unique: true
    },
    phone: {
        type: String,
        required: false,
        default: 'not specified'
    },
})
```

[17]

Takto vytvořené schéma se potom použije pro kompilaci modelu. Model definuje programovací rozhraní pro interakci s databází. Instance modelu pak odpovídá dokumentu v databázi. Nová instance se dá jednoduše vytvořit pomocí klíčového slova *new* a dá se s ní snadno pracovat. Například následující kód vytvoří nového uživatele, dvěma různými způsoby mu nastaví data a pomocí funkce *save* ho uloží do databáze.

```
var userJohn = new User({name: 'John'});
userJohn.email = 'ab@cd.com';
userJohn.save();
```

4.2 Příprava aplikace k vývoji

Před vývojem aplikace je třeba připravit vývojové prostředí. To znamená především instalaci balíčkového manažeru npm, Node.js pro backend a vue/cli pro frontend.

4.2.1 Založení a spuštění projektů

Po instalaci těchto služeb již je možné založit projekt. Pro Node.js k tomu slouží příkaz *npm init,* který spustí krátkého průvodce. Výsledkem je soubor *packages.json*, který obsahuje informace o projektu. Dalším krokem je instalace balíčků, které budou v projektu využity. K tomu slouží příkaz *npm install <balíček>*. Takto nainstalované balíčky budou uloženy do složky *node_modules*, která se vytvoří, pokud ještě neexistuje. Zároveň se informace o balíčku přidá do *dependencies* v souboru *packages.json*. Pokud by nějaké balíčky byly uvedeny v *dependencies* a nebyly nainstalovány, příkaz *npm install* je všechny doinstaluje. Před spuštěním aplikace je ještě nutné v *packages.json* definovat spouštěcí skript, který se spustí příkazem *npm run <skript>* a vytvořit soubor, který bude skriptem spouštěn.

U vue/cli je postup podobný. Založení začíná příkazem *vue create <název-projektu>*, který taktéž spustí krátkého průvodce. Tento příkaz vytvoří kromě souboru *packages.json* celou strukturu složek a souborů s již spustitelným ukázkovým projektem. Spouštěcí skript je taktéž již definovaný. Aplikace se spustí příkazem *npm run serve*, a poběží na portu 8080.

4.2.2 Příprava serverové části

Na rozdíl od klientské části, kde průvodce založením projektu vytvoří již spustitelný projekt, serverovou část je nutné připravit ručně. V souboru, který bude spouštěn spouštěcím skriptem je třeba:

- Importovat všechny potřebné moduly
- Vytvořit instanci express
- Připojit middleware
- Navázat připojení k databázi

- Nastavit obsluhu http požadavků
- Získat nebo vytvořit port http serveru
- Vytvořit http server
- Vytvořit instanci Socket.IO a nastavit obsluhu

Protože aplikace bude obsluhovat větší množství http požadavků, je vhodné využít směrování, které nabízí framework Express. Jeho použití vypadá takto:

```
const users = require('./routes/api/users');
const conversations = require('./routes/api/conversations');

app.use('/api/users', users);
app.use('/api/conversations', conversations);
```

Toto směrování umožňuje rozdělit obsluhu http požadavků do více souborů pro lepší přehlednost kódu. Při příchozím požadavku na adresu */api/users* bude požadavek přesměrován na soubor *users.js* ve složce */routes/api*, který ho obslouží. Stejná logika platí pro požadavky na */api/conversations*.

Samotná obsluha požadavku je díky frameworku Express snadno realizovatelná. Následující ukázka kódu zpracovává http požadavek GET s parametrem *email*. Využívá při tom middleware *authenticateToken*, který kontroluje platnost JWT tokenu. Pokud je token platný, provede vyhledávání uživatele v databázi podle emailu uvedeného v parametru a odešle výsledek vyhledávání jako odpověď.

```
router.get('/:email', authenticateToken, (req, res) => {
    User.findOne({
        email: req.params.email
    }, (err, data)=>{
        if(err)throw err;
        res.send(data);
    })
})
```

Dále stojí za zmínku ukázka vytvoření instance Socket.IO a nastavení jeho obsluhy. Socket.IO vyžaduje jako parametr server, na kterém má poslouchat. V konkrétní ukázce se pro právě připojený socket nastaví obslužná funkce událostí

joinRoom a *leaveRoom*, která zároveň přijímá parametr reprezentující identifikátor místnosti, do které má obsluha obsluhovaný socket připojit či odpojit.

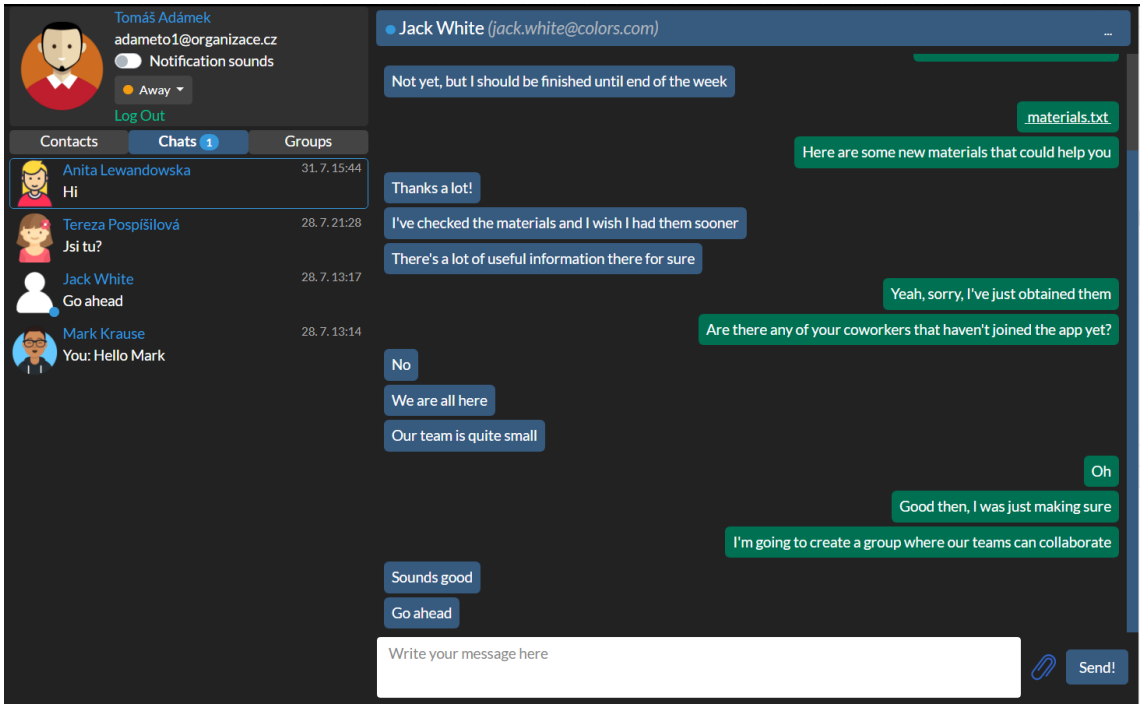
```
const io = require('socket.io')(server);

io.on('connection', function(socket){
    socket.on('joinRoom', function(roomID){
        socket.join(roomID);
    });
    socket.on('leaveRoom', function(roomID){
        socket.leave(roomID);
    });
}
```

4.3 Grafické rozhraní

Grafické rozhraní bylo navrženo s důrazem na přehlednost. Aplikace je realizovaná jako „Single-page application“, což znamená že aplikace se dynamicky přepisuje a data ze serveru získává na pozadí, místo načítání celých stránek znovu. To značně zlepšuje uživatelskou zkušenost, aplikace díky tomu na pohled běží hladce a vypadá moderně. Dynamického přepisování stránky je dosaženo pomocí Vue, které po změně dat vázaných ke zobrazeným komponentám automaticky změnu zjistí a nová data zobrazí. Získávání dat na pozadí pak zajišťuje plugin Axios.

Rozhraní aplikace pro přihlášeného uživatele je tvořeno dvěma základními komponentami – komponenta levého menu a komponenta hlavního obsahu aplikace. V horní části levého menu jsou k nalezení základní informace o přihlášeném uživateli, důležitější je ale část spodní, kde se nachází seznamy uživatelů, konverzací a skupinových konverzací odděleně. Mezi seznamy se dynamicky přepíná kliknutím na štítek s příslušným nadpisem. Toto dynamické přepínání je realizováno pomocí kondicionálního renderování, které poskytuje Vue.



Obr. 3 Ukázka grafického rozhraní aplikace

Grafické rozhraní konverzací (viditelné na obrázku 3) se skládá z hlavičky, těla a patičky. V hlavičce je k nalezení jméno recipienta nebo skupiny a v pravé části také možnosti v podobě dropdown menu, které se dá zobrazit kliknutím na tři tečky.

Tělo je naplněno zprávami. Pro zprávy bylo zvoleno schéma klasického chatu namísto schéma příspěvků, které některé analyzované aplikace preferovaly. Zprávy odeslané přihlášeným uživatelem jsou zarovnané doprava a zprávy odeslané ostatními uživateli jsou zarovnané doleva. U skupinových konverzací se také vyskytují systémové zprávy, které jsou zarovnané na střed a jsou psány červeným písmem. Zprávy se řadí od nejnovějších odspodu a stejně jako u analyzovaných aplikací je jich zobrazeno pouze omezené množství z optimalizačních důvodů. Pro načtení starších zpráv je třeba vyjet posuvníkem nahoru.

Patička stránky obsahuje nástroje pro odeslání nové zprávy. Největší část tvoří textová oblast, po které bylo u návrhu vyžadováno, aby při prohlížení zpráv nezabírala zbytečné místo, ale při psaní zpráv poskytovala dostatek prostoru, aby si uživatel mohl kontrolovat, co píše. Textová oblast sice nabízí uživateli změnu její

velikosti myší, ale z designového hlediska to pro aplikaci vhodné není, protože by uživatel mohl narušit strukturu komponent. Proto byla textová oblast implementována tak, že dynamicky mění výšku vlastní i komponenty nad ní na základě toho, zda do ní uživatel kliknul. Dále se v patičce nachází tlačítka pro poslání souboru a odeslání zprávy.

4.4 Popis klíčových funkcionalit aplikace

4.4.1 Přihlášení, udržování údaje o přihlášení a zabezpečení dat

Po zadání přihlašovacích údajů uživatel odešle požadavek o přihlášení na server. Server pro porovnání zadaného hesla se zahashovaným heslem v databázi musí využít balíčku bcrypt, který je použit i pro hashování hesla při registraci. Pokud kontrola přihlašovacích údajů dopadne pozitivně, server vygeneruje nový JSON Web Token s dobou platnosti 8 hodin a odešle ho zpět klientovi. Klient pak uloží token a email přihlášeného do cookies a nastaví jim platnost rovněž 8 hodin.

Udržování informace o přihlášení uživatele využívá právě tohoto tokenu v cookies. Při načtení aplikace zkontroluje, zda je v cookies nějaký token uložen a pokud ano, považuje uživatele za přihlášeného. Pokud by si někdo do cookies uložil falešný token za účelem imitace přihlášení, k žádným datům by se nedostal. Obsah aplikace po přihlášení totiž obsahuje data z databáze a s každým požadavkem na získání dat se posílá i token, který je ještě před obsloužením požadavku na serveru zkontrolován. Pokud dopadne kontrola negativně, server bez vykonání původního požadavku vrátí http status kód 401 neboli „unauthorized“.

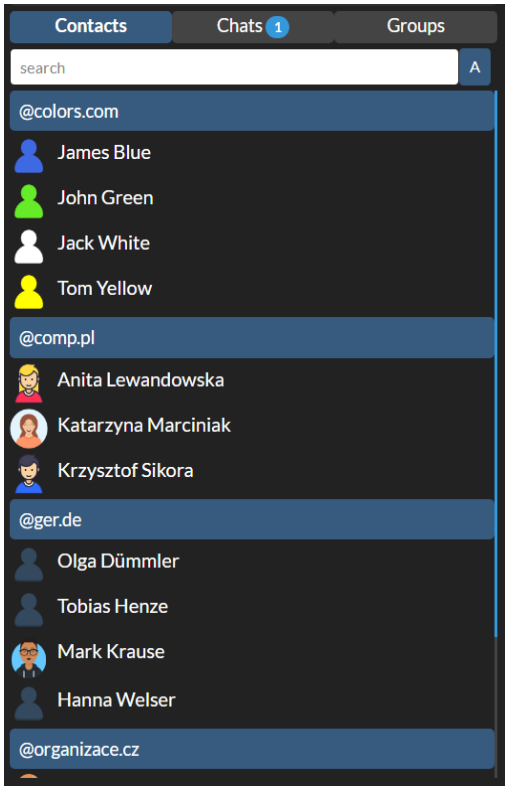
Status 401 je zachytáván interceptorem, který poskytuje plugin Axios. Tento interceptor globálně zachytává příchozí odpovědi na požadavky ještě předtím, než se dostanou na místo v kódu, kde jsou normálně zpracovány a může definovat a převzít obsluhu pro speciální případy. Status 401 je brán jako speciální případ, který může nastat při expiraci tokenu, modifikaci tokenu, smazání cookies nebo uložení falešného tokenu do cookies. Pokud nastane, odpověď je zadržena, z cookies jsou vymazány zbylé údaje o přihlášení a uživatel je přesměrován na přihlašovací formulář. Všechny ostatní odpovědi interceptor nechá projít dál k jejich běžné obsluze. Nastavení interceptoru v kódu vypadá následovně:

```
axios.interceptors.response.use((response) => {
        return response
    }, async (error) => {
        if (error.request.status == 401) {
          if(!this.logged)return;
          this.logged = false;
          VueCookies.remove('accessToken');
          VueCookies.remove('email');
          alert("Session has expired. Please authenticate yourself by log
ging in again.");
        }else{
          throw error
        }
    }
)
```

4.4.2 Evidence kontaktů

Při implementaci evidence kontaktů byla největší snaha o pohodlí uživatele. Bylo žádoucí, aby si uživatel nemusel přidávat kontakty po jednom, a tak je seznam kontaktů globální. Kdokoliv se do aplikace registruje, bude k vidění v seznamu kontaktů všech ostatních uživatelů z vlastnické organizace. Uživatelé z ostatních organizací budou vidět pouze své kolegy a uživatele z vlastnické organizace.

Další formou zpříjemnění uživatelské zkušenosti je zpřehlednění automatickým rozdělením kontaktů do skupin, viditelným na obrázku 4. Je běžné, že každá organizace má svou doménu a přiděluje svým zaměstnancům pracovní emaily s touto doménou. Při registraci musí uživatel použít právě pracovní email, protože registrace je umožněna pouze emailům s doménami uvedenými na whitelistu. Na tomto předpokladu funguje ono automatické rozdělení, které seskupuje kontakty na bázi domén jejich emailů.



Obr. 4 Ukázka způsobu evidence kontaktů

Rozdělení je vizuální a kontakty ve skutečnosti nejsou rozděleny do několika seznamů. Kontakty jsou v jednom seznamu a je třeba je seřadit podle abecedního pořadí jejich příjmení a domén. Toto dvojité řazení ale není ale ve skutečnosti realizováno voláním dvou řadících funkcí po sobě, protože prohlížeče mohou používat jiné algoritmy. Při využití způsobu volání dvou řadících funkcí například v prohlížeči Chrome byl výsledek správný, ale v prohlížeči Mozilla Firefox byly kontakty ve skupinách seřazeny v opačném abecedním pořadí. Řazení je tedy naprogramováno jednou komplexnější řadící funkcí, která vypadá takto:

```
this.contacts.sort((a, b) => {
    if(a.domain.toLowerCase() > b.domain.toLowerCase()) return 1;
    if(a.domain.toLowerCase() < b.domain.toLowerCase()) return -1;
    if(a.lastname.toLowerCase() > b.lastname.toLowerCase()) return 1;
    if(a.lastname.toLowerCase() < b.lastname.toLowerCase()) return -1;
})
```

Vizuální oddělování pak probíhá v *template* části Vue souboru při vypisování kontaktů do html. Díky předchozímu seřazení, stačí pak jednoduchou podmínkou ve smyčce pro vypisování zkontrolovat, zda právě vypisovaný kontakt

má stejnou doménu jako poslední vypsaný kontakt. Pokud ne, vloží se do html oddělovač s názvem domény.

Aplikace také umožňuje přepínání mezi zobrazením jmen a emailů kontaktů. Ke jménu i emailu je vždy zobrazen profilový obrázek uživatele. V obou těchto zobrazeních lze provádět filtraci kontaktů. Defaultně jsou zobrazeny jména kontaktů. Zobrazení emailů může být vhodné hlavně k filtraci, kde uživatel může vyfiltrovat skupinu kontaktů podle domény. Následující kód je ukázkou funkce, která filtruje kontakty podle zvoleného kritéria.

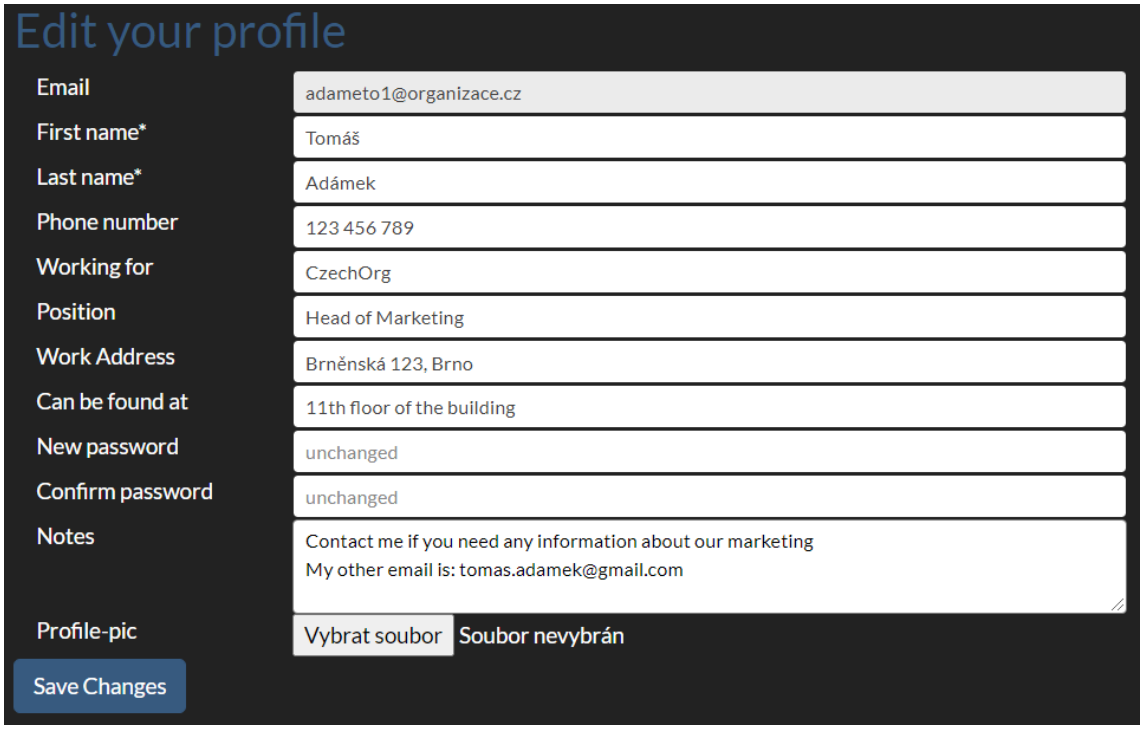
```
getFilteredContactList: function(){
    if(this.filterContactListBy === '@'){
        return this.contacts.filter(contact => contact.email.toLowerCase(
).includes(this.contactSearchText.toLowerCase()));
    }else{
        return this.contacts.filter(contact =>
        (contact.firstname+" "+contact.lastname).toLowerCase().includes(t
his.contactSearchText.toLowerCase()) ||
        (contact.lastname+" "+contact.firstname).toLowerCase().includes(t
his.contactSearchText.toLowerCase())
        );
    }
}
```

Po kliknutí na kontakt ze seznamu se uživateli zobrazí detail kontaktu, který si lze prohlédnout na obrázku 5. Kromě strukturovaně zobrazených informací se na stránce pod profilovým obrázkem nachází dvě tlačítka. První umožňuje začít konverzaci s uživatelem a druhé po kliknutí otevře okno nového emailu s předvyplněným příjemcem v aplikaci, kterou má uživatel defaultně nastavenou jako emailového klienta.

Obr. 5 Ukázka detailních informací kontaktu

4.4.3 Editace osobních dat

Do editace osobních dat se uživatel dostane kliknutím na svůj profilový obrázek v levém horním menu. Rozhraní, které je k vidění na obrázku 6, je tvořeno html elementy *input,* u kterých je využito direktivum *v-model* pro obousměrné svázání s javascriptovými daty. U některých elementů způsoboval prohlížeč Google Chrome problémy svým neodbytným automatickým doplňováním, které i přes nastavení parametru *autocomplete* na různé hodnoty nešlo vypnout. Protože je pro aplikaci důležitá uživatelská zkušenost, a není žádoucí, aby byl uživatel maten tím, že je v poli pro telefonní číslo předvyplněný email, byl tento problém vyřešen nastavením výchozí hodnoty telefonního čísla na „not specified“. Podobný problém u položky hesla, kde rovněž nebylo žádoucí jeho automatické vyplňování, byl vyřešen umístěním elementu *input* typu *password*, mimo zobrazovaný obsah stránky (prosté schování pomocí css problém neřešilo). Prohlížeč Chrome pak vyplnil tento uživatelem nespatřitelný element a viditelné elementy zůstaly nevyplněné.



Obr. 6 Ukázka rozhraní pro editaci osobních údajů

4.4.4 Vytváření konverzací

Vytvořit soukromou konverzaci lze v aplikaci kliknutím na tlačítko „Start conversation“ na profilu kontaktu, se kterým chce uživatel konverzovat. Pokud již konverzace mezi kontaktem a uživatelem existuje, bude uživateli zobrazena.

Tlačítko pro vytvoření skupinové konverzace se nachází v levém menu po kliknutí na štítek „Groups“. Pro vytvoření nové skupiny je nutné zadat její název.

Po vytvoření nového záznamu o konverzaci v databázi je pro konverzaci také vytvořena složka v serverovém úložišti, do které se budou ukládat sdílené soubory. Název složky je shodný s parametrem *_id*, který konverzaci vygenerovala databáze MongoDB.

4.4.5 Zasílání zpráv

Proces zasílání zpráv v reálném čase je realizován pomocí technologie Socket.IO popsané v kapitole o použitých technologiích. V aplikaci je využita jeho funkcionalita místností pro oddělení konverzací. Při otevření konverzace v aplikaci

se klient připojí do místnosti, která používá jako identifikátor parametr *_id* konverzace z MongoDB.
Celý proces pak vypadá takto:

1. Uživatel napíše zprávu a stiskne enter nebo klikne na tlačítko pro odeslání.
2. Vue zkontroluje, zda je text zprávy prázdný a pokud ne, asynchronně odešle http požadavek o uložení zprávy do databáze na server pomocí pluginu Axios.
3. Server zpracuje požadavek a zároveň s uložením zprávy do databáze aktualizuje v dokumentu konverzace seznam uživatelů, kteří nemají poslední zprávu konverzace přečtenou.
4. Vue vyšle událost o nové zprávě serveru pomocí Socket.IO.
5. Serverová část Socket.IO zaznamená událost a vyšle dvě nové události zpět do Vue:
    a. Událost přímo s daty zprávy. Tato událost je vyslaná pouze do místnosti konverzace, tedy uživatelům, kteří mají konverzaci právě otevřenou.
    b. Událost obsahující pouze informaci, že uživatel nějakou zprávu obdržel. Tato událost je dále popsána v kapitole o notifikacích.
6. Pokud má uživatel otevřenou konverzaci (je v socketové místnosti), do které zpráva přišla, Vue ji přidá na konec konverzace a automaticky ji zobrazí bez obnovení stránky.

Při otevření jiné konverzace se socketová místnost staré konverzace opustí a uživatel je připojen do místnosti odpovídající nově otevřené konverzaci. Protože změna obsahu komponenty pracující s konverzací při změně konverzace není spuštěna žádnou funkcí (Vue změní obsah komponenty dynamicky sám), není kam přidat příkazy pro změnu socketové místnosti. Je tedy potřeba v komponentě tuto změnu zaznamenat pomocí funkce Vue watcher, který hlídá změny pro nastavenou proměnnou. V tomto případě je hlídaná proměnná id otevřené konverzace. Když nastane změna proměnné, watcher spustí obslužnou funkci, která má jako parametry hodnotu proměnné před změnou a hodnotu proměnné po změně.

4.4.6 Mazání zpráv

Zprávy v konverzaci se dají smazat kliknutím na ikonku koše, která se zobrazí po najetí myší na řádek s textem zprávy. Tato ikonka se zobrazuje pouze autorovi zprávy. Proces mazání zpráv začíná odesláním http *delete* požadavku na server. Požadavek obsahuje jako parametry id konverzace a identifikátor zprávy, kterým je čas odeslání zprávy v milisekundách. Server pak odešle pokyn k odstranění zprávy z konverzace, který v kódu vypadá takto:

```
Conversation.updateOne({
    _id: new ObjectId(req.params.convID),
},{
    $pull: {
        messages: {
            sender: req.body.emailFromToken,
            time: parseInt(req.params.time)
        }
    }
})
```

V dané konverzaci se pomocí operátoru *pull* odstraní prvek z pole zpráv s odpovídajícím identifikátorem času. V dotazu je dodatečně kontrolován email autora zprávy s emailem uloženém v JSON Web Tokenu, který je odesílán s požadavkem, aby nedošlo k neautorizovanému odstraňování zpráv, pokud by se nějaký uživatel pokusil o podvodný požadavek.

Pokud se jedná o zprávu se souborem, server využije modulu *fs* pro odstranění souboru ze serverového úložiště. Identifikátor souboru se nachází v těle požadavku.

Po odpovědi serveru se ještě odešle událost pomocí Socket.IO všem uživatelům v konverzaci, aby jejich prohlížeč mohl zprávu ihned smazat.

4.4.7 Informace o přijetí zprávy a notifikace

Kromě zobrazování právě přijatých zpráv v otevřené konverzaci je třeba řešit i obecný příjem zpráv pro všechny přihlášené uživatele. Přijímání zpráv i z neotevřených konverzací je nutné řešit, protože je žádoucí, aby uživatel dostal vizuální i zvukové upozornění o nové zprávě.

Vizuálních upozornění je v aplikaci několik. Prvním je změna titulku stránky při příchodu nové zprávy. Toto upozornění je užitečné, pokud se uživatel právě nachází v jiném panelu či okně prohlížeče. Druhým je malé číslo v levém menu u štítků pro zobrazení konverzací, reprezentující počet konverzací, ve kterých má uživatel nepřečtenou zprávu. Toto číslo je získáno z databáze jednoduchým dotazem. Každý záznam o konverzaci v databázi totiž obsahuje seznam uživatelů, kteří mají v konverzaci nepřečtenou zprávu. Součet výskytů uživatele v těchto seznamech je pak výsledným číslem. Třetím vizuálním indikátorem je barevné ohraničení konverzace v seznamu. Poslední dva zmíněné indikátory jsou k vidění na obrázku 3.

Zvukové upozornění je možné ztišit, a to jak globálně, tak separátně pro každou konverzaci. Globální ztišení může uživatel rychle provést kliknutím na malý posuvník v levé horní části aplikace. Informace o ztišení se ukládá do cookies, aby mohla být znovu načtena při dalším přístupu do aplikace. Separátní ztišování lze provést v možnostech každé konverzace. Informace o tomto ztišení se také ukládá do cookies, a to v podobě serializovaného pole. U zvukového upozornění se objevil problém se zásadami webových prohlížečů, které neumožňují přehrát audio, dokud uživatel neprovede interakci se stránkou. V ideálním scénáři, kdy se uživatel přihlásí a nechá stránku otevřenou či začne s někým konverzovat, tato zásada problém nevytváří. Ovšem pokud uživatel například omylem aktualizuje stránku nebo ji otevře a jeho relace přihlášení je stále v platnosti, může zůstat stránka otevřená bez interakce, a v tom případě by tato zásada neumožnila přehrání notifikace. Během psaní aplikace bohužel nebyl nalezen způsob, jak toto opatření obejít.

Předávání informace o nové zprávě probíhá s využitím Socket.IO. Cílem je informovat všechny přihlášené uživatele na příchozí zprávu, bez ohledu na to, zda mají otevřenou konverzaci nebo ne. Logika implementace může být realizována několika způsoby. Jako první se nabízí využití místností zmíněných o kapitolu výše. Místnostmi by se daly reprezentovat všechny konverzace. To by znamenalo vytvořit socketovou místnost pro každou existující konverzaci ve které se přihlášený uživatel nachází a připojit ho do ní. Vytváření velkého počtu dalších místností je ovšem v porovnání s vybranou metodou zbytečné. Tento způsob by

navíc komplikoval již implementované zasílání zpráv do místností za účelem jejich zobrazení v otevřených konverzacích a z těchto důvodů není ideální.

Druhou možností je přenést logiku na stranu klienta. Místo třídění, kterému klientovi patří upozornění a hledání způsobu, jak ho dostat pouze k němu, lze veškeré informace odeslat klientovi a nechat třídění na něm. A to tak, že server najednou odešle všem připojeným na socketu seznam obdržitelů nové zprávy. Každý klient by pak udělal svou část práce tím, že by zjistil, zda seznam obsahuje email přihlášeného uživatele. I když odeslání seznamu všem klientům lze uskutečnit jen jedním příkazem, z hlediska výpočetního výkonu úsporné není. Socket.IO by na pozadí iteroval všemi připojenými klienty a odesílal jim data po jednom, i když to tak z kódu nevypadá. Tato metoda by tedy výpočetní výkon serveru nešetřila, a navíc by přidávala zbytečnou zátěž i všechny klienty.

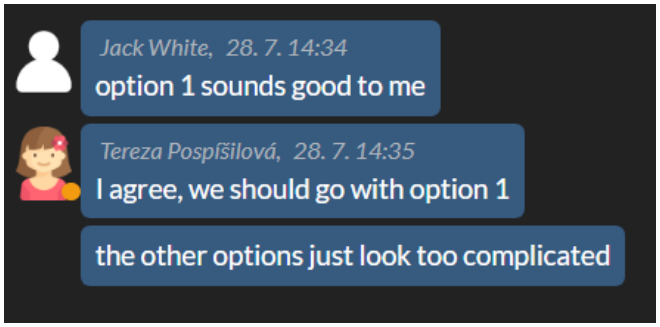
Třetí možnost je využít automaticky vytvořených socketových místností pro každého připojeného uživatele. Tyto místnosti jsou identifikované unikátním nepředvídatelným ID, které Socket.IO při připojení vygeneruje. Je tedy nutné na serveru udržovat rejstřík, kde by byly ID socketů přiřazeny k identifikacím, které používá aplikace (emaily). Tento rejstřík je využit i při udržování stavu uživatelů na serveru. Při události nové zprávy by se touto metodou iterovalo pouze obdržiteli zprávy a odeslalo se upozornění pouze jim, namísto iterování všemi přihlášenými uživateli a další iterací na straně klienta. Dále pro tuto metodu není nutné vytvářet další místnosti, ale využít již automaticky vytvořené. Z těchto důvodů byl zvolen tento způsob jako nejvhodnější.

4.4.8 Skupinové konverzace

Skupinové konverzace se v databázi udržují ve stejné kolekci jako ty soukromé. Jejich databázové schéma je stejné s rozdílem, že skupinové konverzace mají vyplněný parametr *groupName* neboli název skupiny. Kromě toho je vhodné, že MongoDB každé konverzaci v kolekci přiřadí unikátní ID, které pak může být využito jako identifikátor místnosti v Socket.IO. Pokud by byly konverzace oddělené ve dvou kolekcích, mohlo by se stát, že soukromá a skupinová konverzace dostanou stejné ID, což by znemožnilo jeho využití jako identifikátoru v Socket.IO a musel by být vyvinut jiný způsob unikátní identifikace.

Na klientské straně se tedy rozpoznávají skupinové konverzace kontrolou parametru *groupName*. U skupinových konverzací je třeba definovat několik dalších funkcí, jako zobrazení názvu skupiny, rozlišení odesílatelů zpráv a administraci skupiny.

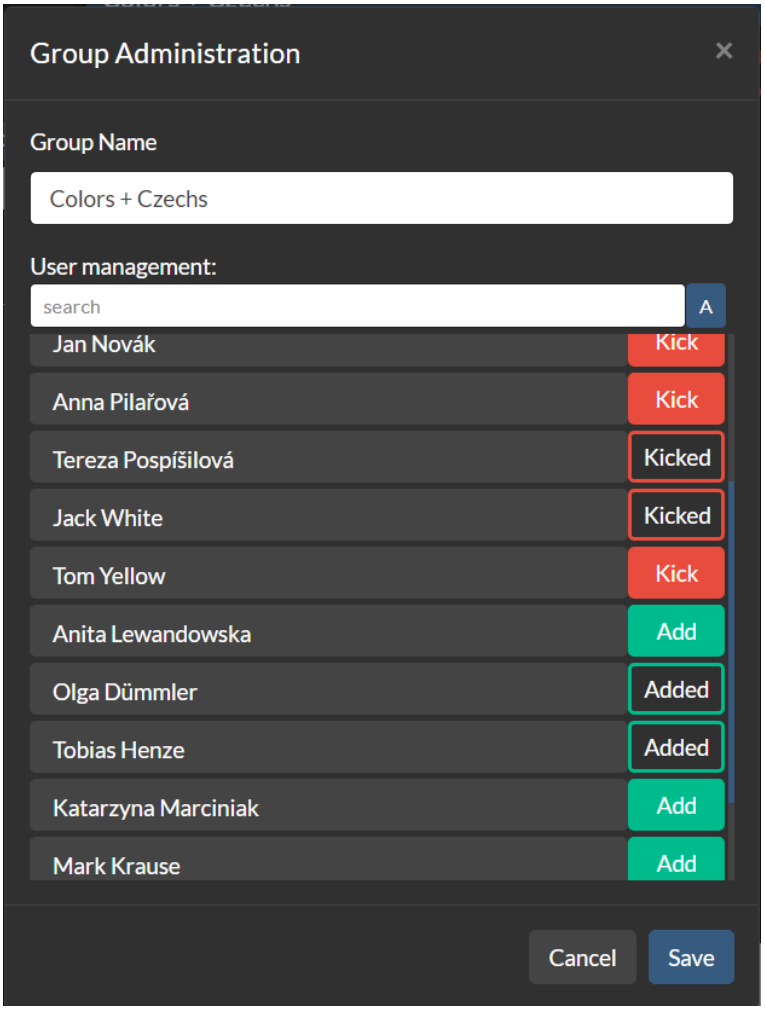
Rozlišováním odesílatelů zpráv je myšlena vizuální identifikace, a to taková, aby uživatel u zprávy ihned poznal, který z členů skupiny je jejím autorem. Všechny analyzované aplikace tuto identifikaci řeší zobrazením profilového obrázku a jména odesílatele u zprávy. Na základě analýzy byla zvolena stejná metoda – viz obrázek 7.



Obr. 7 Ukázka rozlišení odesílatelů ve skupinové konverzaci

Administrace skupiny (viditelná na obrázku 8) slouží ke změně jména skupiny a správě uživatelů. Pro správu uživatelů se v okně administrace nachází seznam uživatelů začínající uživateli, kteří jsou členy skupiny. Podobně jako v seznamu kontaktů je i v tomto seznamu možné přepínat mezi zobrazením jmen a emailů a filtrovat oběma zobrazeními. U členů skupiny se nachází tlačítko pro jejich odstranění a u ostatních uživatelů se nachází tlačítko pro jejich přidání. Po kliknutí se tlačítko dynamicky změní a naznačuje, že s uživatelem bude provedena akce. Všechny změny jsou uloženy až po kliknutí na tlačítko „Save“.

Uživatelé z vlastnické organizace vidí v administraci všechny kontakty, a mají možnost vytvořit spojení mezi více organizacemi, pokud potřebují. Uživatelé z ostatních organizací vidí v seznamu pouze kolegy ze stejné organizace a uživatele vlastnické organizace.



Obr. 8 Ukázka grafického rozhraní pro administraci skupin

Po kliknutí na tlačítko „Save“ se nejprve odešle požadavek na změnu dat v databázi. Server nejprve vytvoří systémové zprávy ke každé změně (například „Uživatel změnil název“ nebo „Uživatel1 přidal uživatele2“) a poté změny i se zprávami do záznamu o konverzaci uloží.

Po zpracování požadavku je nakonec pomocí Socket.IO nutné ihned podat přihlášeným uživatelům informace o změnách. Je třeba:

- Uživatelům s otevřenou konverzací aktualizovat název skupiny a zobrazit nové systémové zprávy
- Znemožnit přístup do konverzace uživatelům, kteří byli odstranění ze skupiny

- Přidat skupinu a vizuální upozornění těm uživatelům, kteří byli do skupiny přidání

4.4.9 Nahrávání souborů

Rozhraní pro nahrávání souborů lze otevřít kliknutím na tlačítko spony v otevřené konverzaci. Je použito standardní rozhraní html elementu *input* typu *file*.

Proces samotného nahrávání souboru začíná načtením souboru v Javascriptu pomocí vestavěného FileReaderu. FileReader soubor načte a uloží ho do proměnné jako base64 řetězec. Tento řetězec je společně s názvem souboru odeslán v podobě JSON objektu na server za použití pluginu Axios.

Server po přijetí požadavku nejprve vytvoří pracovní název souboru. Dále server využije Node modulu „fs“ neboli FileSystem, který vezme base64 řetězec a uloží ho opět v podobě souboru na serverové úložiště pod pracovním názvem. Po úspěšném uložení souboru server ještě přidá do databáze záznam o nové zprávě v konverzaci s objektem obsahujícím pracovní název souboru a původní název souboru. Soubor má evidované dva názvy z těchto důvodů:

- poskytnutí možnosti uživatelům odeslat do konverzace soubory se stejným názvem
- v jedné složce (na serverovém úložišti) nemohou mít dva soubory stejný název, proto je potřeba vygenerovat pracovní název, pod kterým budou soubory uloženy
- soubor má mít při zobrazení a stažení název stejný, jako měl při nahrání

4.4.10 Stahování souborů

Při načítání zpráv klient rozlišuje, jestli se jedná o zprávu obsahující soubor nebo text. Pokud se jedná o soubor tak dále zkoumá, zda se jedná o obrázek, či jiný typ souboru. Když se jedná o obrázek, zobrazí ho přímo pomocí html elementu *img*. Pokud se jedná o soubor jiného typu, zobrazí pouze jeho název v podobě odkazu, který slouží jako tlačítko pro jeho stažení.

Pro implementaci stahování souborů do uživatelova lokálního úložiště se jako první nabízelo využití odkazového html elementu a jeho atributu *download*. Toto jednořádkové řešení však stahovalo soubory pouze pod pracovními názvy. To proto, že prohlížeče nedovolují přejmenování stahovaného souboru, pokud je jeho URL cizího původu.[21] Lokální vývojové servery pro Vue frontend a Node backend mají rozdílné porty, takže se navzájem považují za cizí.

Možnost, jak tento problém vyřešit, je dodat do atributu *href* přímo samotný soubor v podobě BLOB (Binary Large Object) URL, kterou lze vygenerovat pomocí Javascriptu. Použití Javascriptu je tedy nutné a nabízí se možnost v něm vyřešit problém celý. Metoda pro stažení souboru do uživatelova lokálního úložiště vypadá takto:

```
downloadFile: async function(workName, displayName){
    const result = await ServicesConversations.downloadFile(this.convID,
workName);
    const url = window.URL.createObjectURL(new Blob([result.data]), {
        type: result.headers["content-type"]
    });
    const link = document.createElement('a');
    link.href = url;
    link.download = displayName;
    document.body.appendChild(link);
    link.click();
    document.body.removeChild(link);
}
```

V metodě probíhá proces následovně:

1. Pomocí pluginu Axios se stáhnou data souboru ze serveru.
2. Za použití statické funkce *createObjectUrl* se vygeneruje BLOB URL s daty souboru.
3. Javascriptem se vytvoří nový element odkazu.
4. Elementu se Javascriptem nastaví parametry *href* obsahující BLOB URL a *download* obsahující původní název souboru.
5. Javascript přidá element do dokumentu, simuluje kliknutí na něj, a nakonec ho odebere.

Je důležité zmínit, že při vytváření http požadavku v pluginu Axios je nutné specifikovat parametr *responseType* a nastavit mu hodnotu *blob*. Pokud tak není učiněno, některé soubory (například soubory typu pdf) se po stažení a následném otevření budou jevit jako prázdné, nebo jejich obsah bude tvořen nesmyslnými znaky. [22]

4.4.11 Stav aktivity uživatele

Stav uživatelů determinuje server na základě nějaké komunikace s klientem. Nástroj pro komunikaci mezi serverem a klientem v reálném čase se již v aplikaci nachází. Je to Socket.IO a lze ho využít i pro tuto funkcionalitu. Zjišťování stavu uživatelů se dá realizovat dvěma způsoby.

Prvním je způsob založený na událostech odesílaných z klienta. Po přihlášení do aplikace nebo ruční změně stavu by klient odeslal událost s nastavením stavu a server by si ho uložil. Při odpojení by klient opět odeslal vhodnou událost a server by ho ze seznamu vymazal. Každou z těchto změn by server ihned předal všem klientům, aby mohli zobrazit správná data.

Druhý způsob je založen na intervalové kontrole ze strany serveru. Server by v daném intervalu odesílal všem uživatelům dotaz na jejich aktivitu s časovou lhůtou například 1 vteřinu na odpověď. Všichni uživatelé by odpověděli se svým stavem a server by si ji uložil. Po uplynutí časové lhůty by server opět jednorázově všem odeslal aktualizované stavy.

Na první pohled vypadá první způsob jednoznačně úspornější. Pokud by ale aplikace měla velké množství uživatelů, takové, že by se například každých pár vteřin nějaký uživatel připojil, odpojil či měnil stav, aplikace by pokaždé musela aktualizovat seznam a odeslat ho všem klientům. Druhý způsob intervalové kontroly, který by jednorázově dejme tomu jednou za dvě minuty zkontroloval stav všech uživatelů a opět ho jednorázově jedním příkazem všem klientům odeslal, by byl v tomto případě z hlediska výpočetního výkonu úspornější.

Tato aplikace neočekává tak vysoké počty uživatelů, a proto byl vybrán způsob první. Na straně klienta se stav ovládá pomocí dropdown menu umístěným v levé horní části aplikace u profilového obrázku. Zvolený stav se ukládá do cookies, aby mohl být znovu načten při příštím přístupu do aplikace. Stav uživatelů

se zobrazuje barevným puntíkem na několika místech v aplikaci viditelných na obrázku 3 (například v konverzacích nebo v seznamu konverzací) a jeho reaktivní změny jsou zajištěny kondicionálním renderováním a vázáním dat, které poskytuje Vue.

5 Závěr

Cílem aplikace bylo navrhnout a implementovat vlastní aplikaci pro evidenci kontaktů a podporu komunikace. Při návrhu se ukázalo, že funkcionalita komunikace není jenom o zasílání zpráv, ale je doprovázena spoustou dalších funkcionalit jako například vyhledávání ve zprávách, notifikace, implementace stavu uživatelů atd. Právě tyto doprovodné funkcionality hrají velkou roli v tom, jak vysokou úroveň uživatelské zkušenosti aplikace ve finále nabídne.

Při implementaci jsem se seznámil s novými technologiemi jako jsou Vue, Socket.IO či MongoDB. Zatímco MongoDB se prokázala jako jednoznačně praktická volba a potvrdila domněnky uvedené v kapitole o návrhu databáze, Vue neposkytovalo stoprocentní usnadnění práce. Některé věci jako automatickou aktualizaci komponent při změně dat či práci s výpisem dat do html Vue značně ulehčilo, ale na druhou stranu komunikace mezi komponentami se může stát zdlouhavou a nepřehlednou a Vue také působilo problémy se synchronizací při načítání nových dat. Pro další vývoj aplikace bych raději zvolil jednu z nadstaveb Vue jako například Vuex či Nuxt.

Samotná implementace funkcionalit aplikace nebyla tak náročná, jako vymýšlení způsobů, jak implementaci provést a jejich porovnávání ať už vlastní úvahou či s pomocí informací z internetu. Výsledná aplikace je, jak se dalo očekávat, z drtivé většiny frontend. Backendová část aplikace slouží zejména jako prostředník pro komunikaci s databází pomocí REST API. Dalším účelem serverové části je obsluha socketů a správa souborů na serverovém úložišti. Frontendová část ale odvádí většinu práce. Kvůli velkému objemu dat, které musí aplikace zobrazovat za mnoha různých podmínek a vysokým požadavkům na uživatelskou přívětivost se tvorba grafické části frontendu stala až piplavou. Frontend také kromě veškeré práce s daty zajišťuje funkcionality jako dynamické překreslování, kontrolu dat ve formuláři, zvukové i vizuální notifikace, stahování souborů do uživatelova úložiště a mnoho dalších. Všechny funkce aplikace byly testovány v prohlížečích Chrome, Firefox a Edge.

Aplikaci se podařilo vyvinout do stavu, ve kterém splňuje všechny požadavky vytvořené v návrhové části. Aplikace však stále nabízí prostor pro další

vývoj do několika různých směrů. Ve směru komunikace by to byla například implementace hovorů a videohovorů, ve směru kolaborace pak třeba implementace spolupráce na souborech, kterou poskytuje MS Teams. Další možností by bylo aplikaci vyvinout do podoby cloudové namísto on-premises. Protože je pro tento typ aplikace jednou z nejdůležitějších vlastností její uživatelská zkušenost, jsem si jist, že obecný vývoj aplikace by nabral největších obrátek po nasazení do produkce, a to na základě zpětné vazby uživatelů a jejich požadavků na další funkcionality. Do budoucna bych se rád dál věnoval této tématice a snažil se aplikaci rozšířit zmíněnými způsoby.

6 Seznam použité literatury

[1] Facebook Messenger. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2020-08-06]. Dostupné z: https://en.wikipedia.org/wiki/Facebook_Messenger

[2] WIGGERS, Kyle. Facebook Messenger now requires a Facebook account to sign up. *VentureBeat* [online]. 2019 [cit. 2020-08-06]. Dostupné z: https://venturebeat.com/2019/12/26/facebook-messenger-account-sign-up/

[3] *Microsoft Support* [online]. c2020 [cit. 2020-08-06]. Dostupné z: https://support.microsoft.com/en-us/

[4] *Slack* [online]. c2020 [cit. 2020-08-06]. Dostupné z: https://slack.com/

[5] The Most Popular Databases 2019. *Explore-Group* [online]. 2019 [cit. 2020-08-06]. Dostupné z: https://www.explore-group.com/blog/the-most-popular-databases-2019/bp46/

[6] ČÁPKA, David. 1. díl - Úvod do JavaScriptu. *ITnetwork* [online]. c2020 [cit. 2020-08-06]. Dostupné z: http://www.itnetwork.cz/javascript/zaklady/javascript-tutorial-uvod-do-javascriptu-nepochopeny-jazyk

[7] KOĎOUSKOVÁ, Barbora. Vue Js: Výhody, nevýhody a možnosti využití. *Rascasone* [online]. 2020 [cit. 2020-08-06]. Dostupné z: https://www.rascasone.com/cs/blog/co-je-framework-vuejs

[8] *Vue.js* [online]. c2014-2020 [cit. 2020-08-06]. Dostupné z: https://vuejs.org/v2/guide/

[9] ANDERSEN, Bo. Understanding the Virtual DOM. *Coding Explained* [online]. 2017 [cit. 2020-08-06]. Dostupné z: https://codingexplained.com/coding/front-end/vue-js/understanding-virtual-dom

[10] KIESSLING, Manuel. Explained: What are Node.js modules? *The Node Beginner Blog* [online]. 2017 [cit. 2020-07-11]. Dostupné z: https://www.nodebeginner.org/blog/post/nodejs-tutorial-what-are-node.js-modules/

[11] What is npm? *W3Schools* [online]. c1999-2020 [cit. 2020-07-11]. Dostupné z: https://www.w3schools.com/whatis/whatis_npm.asp

[12] Node.js Introduction. *W3Schools* [online]. c1999-2020 [cit. 2020-07-11]. Dostupné z: https://www.w3schools.com/nodejs/nodejs_intro.asp

[13] Socket.io. *Npm* [online]. 2019 [cit. 2020-07-27]. Dostupné z: https://www.npmjs.com/package/socket.io

[14] Introduction to JSON Web Tokens. *Jwt.io* [online]. [cit. 2020-07-11]. Dostupné z: https://jwt.io/introduction/

[15] *MongoDB* [online]. c2020 [cit. 2020-08-06]. Dostupné z: https://www.mongodb.com/

[16] CHODOROW, Kristina. *MongoDB: The Definitive Guide*. 2nd Edition. O'Reilly Media, 2013. ISBN 9781449344689.

[17] HOLMES, Simon. *Mongoose for Application Development*. 2nd Edition. Packt Publishing, 2013. ISBN 9781782168201.

[18] *Mongoose* [online]. c2011 [cit. 2020-08-06]. Dostupné z: https://mongoosejs.com/docs/guide.html

[19] *Vue CLI* [online]. c2018-2020 [cit. 2020-08-06]. Dostupné z: https://cli.vuejs.org/

[20] FERDINANDI, Chris. Sorting an array by multiple criteria with vanilla JavaScript. *Go Make Things* [online]. 2018 [cit. 2020-08-06]. Dostupné z: https://gomakethings.com/sorting-an-array-by-multiple-criteria-with-vanilla-javascript/

[21] The Anchor element. *MDN Web Docs* [online]. c2005-2020 [cit. 2020-08-06]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTML/Element/a

[22] *Stack Overflow* [online]. c2020 [cit. 2020-08-06]. Dostupné z: https://stackoverflow.com/

7 Přílohy

1) **zdrojovy_kod.zip** – archiv obsahující zdrojové kódy celé aplikace (serverové i klientské části)

Univerzita Hradec Králové
Fakulta informatiky a managementu

Zadání bakalářské práce

Autor:	**Vojtěch Navrátil**
Studium:	I1700115
Studijní program:	B1802 Aplikovaná informatika
Studijní obor:	Aplikovaná informatika
Název bakalářské práce:	**Aplikace pro podporu komunikace se zahraničními partnery**
Název bakalářské práce AJ:	Application supporting communication with foreign partners

Cíl, metody, literatura, předpoklady:

Cílem práce je analýza dostupných řešení na trhu, návrh a implementace systému pro evidenci kontaktů zahraničních partnerů.

1. Úvod
2. Analytická část
    1. Analýza řešení existujících aplikací
    2. Souhrn pozitiv a negativ
3. Návrhová část
    1. Návrh nové aplikace na základě výsledků provedné analýzy
4. Implementační část
    1. Popis použitých technologií
    2. Struktura programu
    3. Popis programu
5. Závěr

Garantující pracoviště:	Katedra informatiky a kvantitativních metod, Fakulta informatiky a managementu
Vedoucí práce:	doc. RNDr. Petra Poulová, Ph.D.
Oponent:	Ing. Barbora Tesařová, Ph.D.

Datum zadání závěrečné práce: 14.1.2018