Sem vložte zadání Vaší práce.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

# Webový nástroj pro kolaborativní editaci textů

***Jiří Šimeček***

Katedra softwarového inženýrství
Vedoucí práce: Ing. Petr Špaček, Ph.D.

30. dubna 2018

# Poděkování

Chtěl bych poděkovat...

# Prohlášení

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

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

V Praze dne 30. dubna 2018 ……

### Odkaz na tuto práci

Šimeček, Jiří. *Webový nástroj pro kolaborativní editaci textů*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2018.

# Abstrakt

Tato práce se zabývá problémem kolaborativní editace textů a porovnává jednotlivé známé algoritmy, které tento problém řeší. Dále se zabývá návrhem a implementací prototypu pomocí jednoho z vybraných algortmů.

**Klíčová slova** návrh webové aplikace, kolaborativní editace textů, web v reálném čase, Javascript, ReactJS, NodeJS

# Abstract

Sem doplňte ekvivalent abstraktu Vaší práce v angličtině.

**Keywords** Nahraďte seznamem klíčových slov v angličtině oddělených čárkou.

# Obsah

# Seznam obrázků

# Seznam tabulek

# Úvod

Webové aplikace, které komunikují s uživatelem ve skutečním čase, jsou dnes stále oblíbenější. Uživatel již běžně očekává, že se mu na webových stránkách zobrazují nejrůznější upozornění, či se dokonce aktualizují celé části webové stránky. Nově se začínají objevovat webové nástroje pro kolaborativní spolupráci nad texty (případně nad jinými multimédii), které kombinují myšlenku tvorby obsahu webu uživateli a právě odezvu aplikace ve skutečném čase.

Výstupem této práce je všeobecně nasaditelná komponenta, která bude použita jako jedna z komponent projektu webového IDE s pracovním názvem Laplace-IDE. Komponenta je také určena pro potřeby vývojářů, kteří chtějí vytvořit kolaborativní webový nástroj a nechtějí ho vytvářet od nuly.

Toto téma jsem si zvolil, jelikož většina doposud existujících kolaborativních textových nástojů je postavena nad uzavřeným kódem nebo nad knihovnami, jejichž vývoj byl ukončen. Neexistují tak nástroje, či knihovny, které by bylo možné bez větších problémů použít pro vlastní projekty.

V této práci se zabývám analýzou problému kolaborativní spolupráce nad texty, porovnáním a výběrem vhodných existujících algoritmů a technologií, návrhem znovupoužitelné komponenty a implementací prototypu včetně navržené komponenty.

Tato práce dále pokračuje v následující struktuře: Nejprve se v kapitole 1 zabývám analýzou technologií a použitelných algoritmů, ze které pak přecházím k návrhu architektury komponenty a prototypu v kapitole 2. Navržený prototyp dále v kapitole 3 implementuji a na konec nad výsledným prototypem v kapitole 4 provádím uživatelské testování.

Kapitola 1

# Cíl práce

Cílem rešeršní části práce je analýza požadavků, následné seznámení se zadanými technologiemi a rozbor existujících webových komunikačních protokolů. Dalším cílem je analýza problematiky kolaborativní editace textů a nejčastěji používaných synchronizačních algoritmů. Ale také rozbor existujících aplikací, které umožňují editaci ve skutečném čase, a analýza doménového modelu.

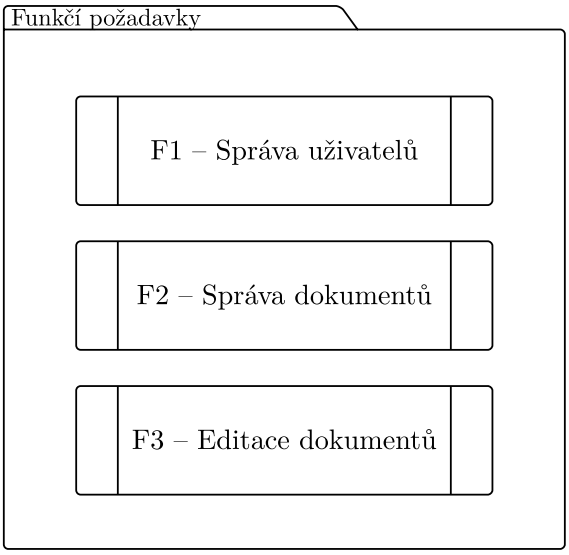
Cílem praktické části je navržení modelu pro uložení dat a prototypu komponenty kolaborativního textového editoru ve skutečném čase. Další cílem je implementace a použití navržené komponenty. A následně uživatelské otestování a vyhodnocení kvality implementovaného řešení.

Kapitola 2

# Analýza

## 2.1 Seznam funkčních požadavků

Před začátkem návrhu každé aplikace je důležité definovat funkční požadavky. Funkční požadavky slouží k vymezení funkcionalit navrhované aplikace. Jednotlivé funkční požadavky jsou dále rozšířeny uživatelskými případy (viz sekce 2.3), které definují použití jednotlivých funkcí aplikace.




Obrázek 2.1: Diagram funkčních požadavků

**F1 – Správa uživatelů** Tento požadavek představuje možnost vytvoření uživatelského účtu a jeho používání. Každý uživatel bude mít uživatelské jméno, které je pro každého uživatele unikátní a slouží jako jeho identifiká-

tor mezi ostatními uživateli. Uživatel si může své přihlašovací i jiné údaje po přihlášení kdykoliv změnit.

**F2 – Správa dokumentů** Přihlášený uživatel si může nechat zobrazit vytvořené dokumenty, vytvořený dokument smazat, či vytvořit dokument nový. Dokumenty jsou vázány na uživatele, který ho vytvořil (dále jen majitel dokumentu), a dokument nelze mezi uživateli přesouvat.

**F3 – Editace dokumentů** Uživatelé mohou upravovat jednotlivé dokumenty ve skutečném čase spolu s ostatními uživateli. U každého dokumentu bude k dispozici komunikační vlákno, kde mohou uživatelé, kteří dokument upravují, komunikovat mezi sebou. Uživatelé mohou pro přístup k dokumentu použít jeho veřejný odkaz nebo mohou být k dokumentu pozváni jednotlivě. Jednotlivý uživatelé, krom majitele, mohou mít možnosti editace dokumentu omezeny pomocí nastavení práv dokumentu.

## 2.2 Seznam nefunkčních požadavků

Pro potřeby projektu Laplace-IDE byly vyhrazeny následující nefunkční požadavky:

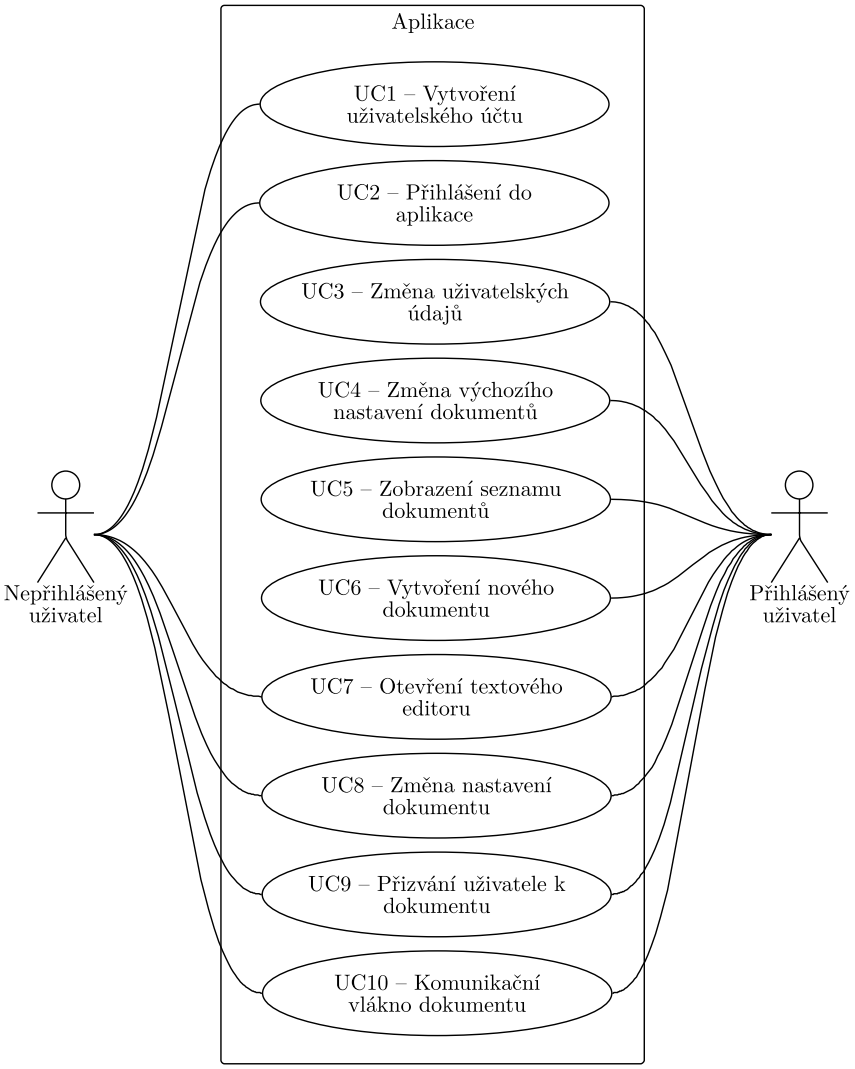
1. validní kód HTML5 (více o jazyce HTML5 v sekci 2.5.1) a Cascading Style Sheets verze 3 (CSS3),
2. programovací jazyk Javascript (více o jazyce JavaScript v sekci 2.5.2),
3. prostředí Node.js (více o prostředí Node.js v sekci 2.5.3) pro server,
4. použití knihovny React (více o knihovně React v sekci 2.5.4) k implementaci komponenty Editoru.

## 2.3 Uživatelské případy

Uživatelské případy blíže rozvádějí funkční požadavky (viz sekce 2.1) a přesněji tak vymezují funkčnosti aplikace. V rámci navrhovaného prototyp této aplikaci nejsou rozlišovány žádné uživatelské role. Rozlišován je pouze přihlášený uživatel od obecného nepřihlášeného uživatele. Z tohoto důvod se v uživatelských případech vyskytují pouze tito dva aktéři.

**UC1 – Vytvoření uživatelského účtu** Nový nepřihlášený uživatel si může pomocí registračního formuláře vytvořit uživatelský účet.

Aplikace zobrazí registrační formulář, který obsahuje vstupní pole uživatelské jméno, email a heslo. Uživatel formulář vyplní a odešle. Aplikace formulář




Obrázek 2.2: Diagram uživatelských případů

zpracuje (vytvoří nového uživatele se zadanými údaji) a uživatele přihlásí. Uživateli je po přihlášení zobrazen seznam dokumentů (nyní prázdný) s tlačítkem vytvořit dokument.

**UC2 – Přihlášení do aplikace** Uživatel, který si již vytvořil uživatelský účet se může k tomuto účtu kdykoliv přihlásit.

Nepřihlášenému uživateli je v horní liště aplikace k dispozici tlačítko `Přihlásit se`. Po kliknutí na tlačítko je uživateli zobrazen přihlašovací formulář obsahující pole uživatelské jméno a heslo. Uživatel je po vyplnění správných

údajů a odeslání formuláře přihlášen. Uživateli je po přihlášení zobrazen seznam dokumentů.

**UC3 – Změna uživatelských údajů** Přihlášený uživatel může kdykoliv provést změnu svých přihlašovacích údajů.

Přihlášený uživatel má v horní liště zobrazené tlačítko `Nastavení`. Po kliknutí na toto tlačítko je uživateli zobrazen formulář pro změnu údajů. Formulář obsahuje pole uživatelské jméno, email, nové heslo a aktuální heslo. Uživatel vyplní údaje, které chce změnit a aktuální heslo. Aplikace po odeslání zobrazí informaci o úspěšné změně údajů.

**UC4 – Změna výchozího nastavení dokumentů** Přihlášený uživatel může změnit výchozí nastavení dokumentů. Toto nastavení je následně aplikováno na uživatelem nově vytvořené dokumenty (u již existujících dokumentů se změny neprojeví).

Přihlášený uživatel má v horní liště zobrazené tlačítko `Nastavení`. Po kliknutí na toto tlačítko je uživateli zobrazen panel s tlačítkem `Nastavení dokumentů`, který se nachází nad formulář pro změnu uživatelských údajů. Kliknutím na tlačítko je uživateli zobrazen formulář pro změnu výchozího nastavení dokumentů. Uživatel upraví pole, které chce změnit, a formulář odešle. Aplikace po odeslání zobrazí informaci o úspěšné změně nastavení.

**UC5 – Zobrazení seznamu dokumentů** Přihlášený uživatel si může zobrazit seznam jím vytvořených dokumentů, seznam dokumentů ke kterým byl přizván a seznam v minulosti otevřených dokumentů.

Přihlášený uživatel má v menu aplikace k dispozici tlačítka `Mé dokumenty`, `Sdílené` a `Poslední`. Po kliknutí na tlačítko aplikace zobrazí korespondující seznam dokumentů.

**UC6 – Vytvoření nového dokumentu** Přihlášený uživatel může kdykoliv vytvořit nový dokument. Počet vytvořených dokumentů není nijak omezený.

Přihlášený uživatel má k dispozici v horní liště tlačítko `Vytvořit nový dokument`. Po kliknutí na tlačítko aplikace dokument vytvoří a aplikuje na něj uživatelovo výchozí nastavení pro dokumenty. Uživatel je přesměrován do textového editoru nově vytvořeného dokumentu.

**UC7 – Otevření textového editoru** Uživatel může přistoupit přímo k veřejnému odkazu dokumentu nebo otevřít editor kliknutím na odkaz v seznamu dokumentů. Po otevření editoru je uživateli zobrazen samotný textový editor a uživatel může začít dokument upravovat (pokud k tomu má pro daný dokument dostatečné oprávnění).

**UC8 – Změna nastavení dokumentu** Uživatel může změnit nastavení dokumentu přímo z textového editoru (pokud k tomu má pro daný dokument dostatečná oprávnění). Aplikace zobrazí nad editorem tlačítko s ikonou ozubeného kola, které po kliknutí otevře panel s jednotlivými poli nastavení.

Nastavení je na editor a dokument aplikováno ve skutečném čase všem upravujícím uživatelům.

**UC9 – Přizvání uživatele k dokumentu** Uživatel může přizvat ostatní uživatele ke spolupráci nad dokumentem a to přímo z textového editoru (pokud k tomu má pro daný dokument dostatečná oprávnění). Aplikace zobrazí nad editorem tlačítko `Sdílet`, které po kliknutí otevře panel s možnostmi sdílení dokumentu.

Uživatel může změnit výchozí práva pro příchozí uživatele pomocí veřejného odkazu a následně odkaz sdílet. Nastavení práv veřejného odkazu se objeví po kliknutí na ikonu ozubeného kola, která se nachází vedle pole s veřejným odkazem. Po odeslání aplikace nastavení uloží a zobrazí potvrzení.

Uživatel také může přizvat jednotlivé uživatele kliknutím na tlačítko `Nová pozvánka`. Po kliknutí na toto tlačítko aplikace zobrazí formulář pro vytvoření pozvánky s poli uživatelské jméno a oprávnění. Uživatel pole vyplní podle potřeby a formulář odešle. Aplikace po odeslání vyhledá uživatele podle uživatelského jména a v případě nalezení vytvoří pozvánku k dokumentu. Aplikace následně obnoví seznam pozvánek pod formulářem a zobrazí informaci o úspěchu, či neúspěchu pozvání.

**UC10 – Komunikační vlákno dokumentu** Uživatel může přispívat do komunikačního vlákna přímo z textového editoru (pokud k tomu má dostatečná oprávnění). Panel s komunikačním vláknem je zobrazen po kliknutí na tlačítko s ikonou konverzační bubliny.

### 2.3.1 Matice pokrytí funkčních požadavků

Při návrhu uživatelských případů je důležité ověřit, zda-li jsme případy pokryli všechny funkční požadavky. Právě k tomuto účelu slouží matice pokrytí funkčních požadavků (viz tabulka 2.1). Matice pokrytí nám ukazuje, které funkční požadavky jsou pokryt v rámci jednotlivých uživatelských případů.

Tabulka 2.1: Matice pokrytí uživatelských požadavků

| | UC1 | UC2 | UC3 | UC4 | UC5 | UC6 | UC7 | UC8 | UC9 | UC10 |
|---|---|---|---|---|---|---|---|---|---|---|
| F1 | × | × | × | × | | | | | | |
| F2 | | | | × | × | × | × | | × | |
| F3 | | | | | | | × | × | × | × |

## 2.4 Doménový model

Pro lepší pochopení dané terminologie je vhodné vytvořit doménový model a popsat jeho entity. Doménový model je také užitečný jako podklad při návrhu databázového schématu (viz sekce 3.2).

### 2.4.1 Uživatel

Entita `Uživatel` nese uživatelovi přihlašovací informace, email, který bude použit v případě zapomenutého hesla, a čas posledního přihlášení. Čas posledního přihlášení může být využit například pro odstranění neaktivních účtů.

### 2.4.2 Dokument

Entita `Dokument` je nositelem informací o dokumentu. Atribut poslední obsah obsahuje poslední text dokumentu potvrzený serverem, který je posílán nově připojeným klientům.

### 2.4.3 Nastavení dokumentu

`Nastavení dokumentu` nese informace o vzhledu a chování editoru pro jednotlivé dokumenty. Vazba na entitu Uživatel umožňuje vytvoření výchozího nastavení dokumentu pro uživatelem nově vytvořené dokumenty.

### 2.4.4 Operace

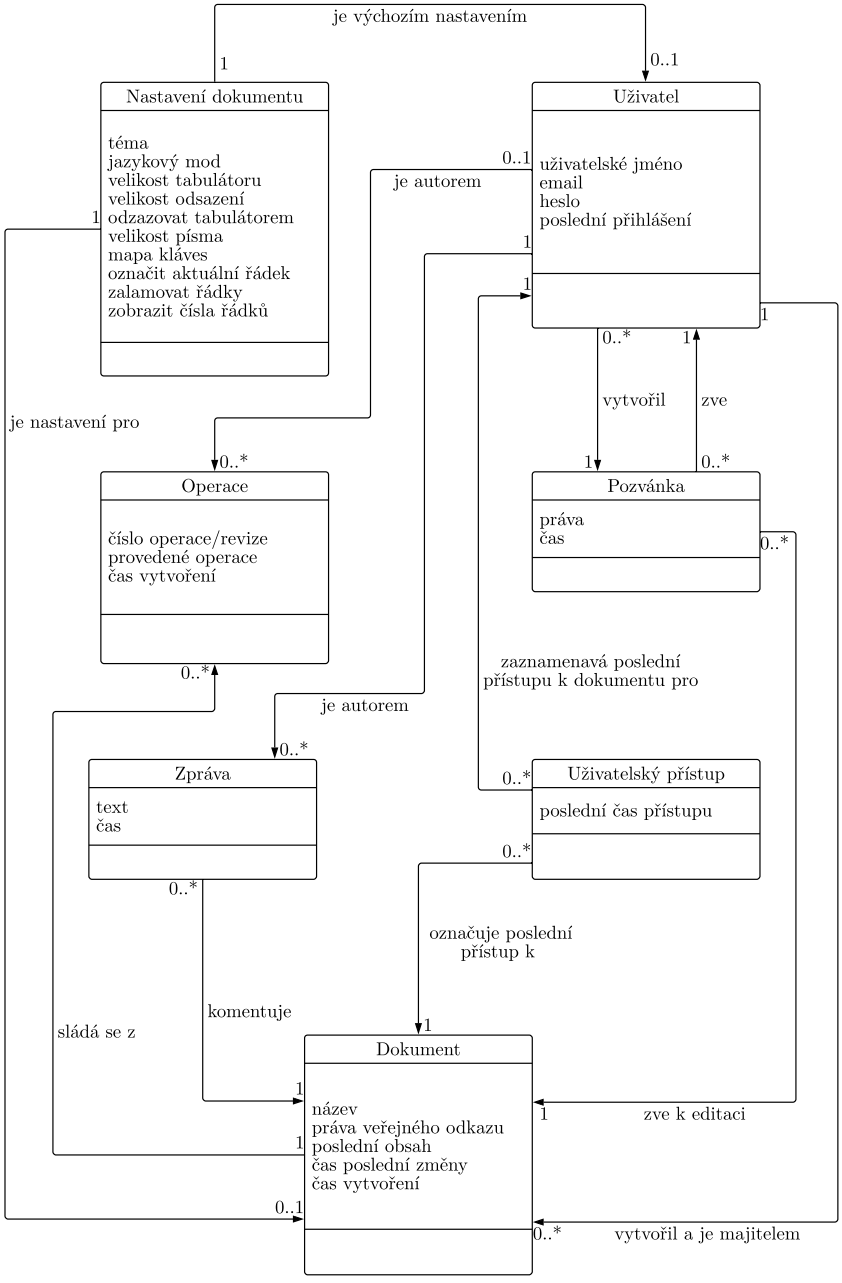
Entita `Operace` označuje jednu revizi dokumentu a obsahuje soubor změn, které byli v dokumentu provedeny od předchozí revize.

### 2.4.5 Další entity

Entita `Pozvánka` umožňuje uživateli pozvat dalšího uživatele k nahlédnutí, či úpravám dokumentu (podle zvoleného atributu práva). `Zpráva` označuje jednu zprávu v komunikačního kanálu dokumentu, z pevné vazby na entitu Uživatel plyne omezení, že zprávu může odeslat pouze přihlášený uživatel. A poslední entita `Uživatelský přístup` udržuje informaci o času posledního přístupu uživatele k dokumentu (pro každou dvojici uživatel a dokument existuje nejvýše jedna její instance).

## 2.5 Technologie

V této sekci popisuji použité technologie vycházející z nefunkčních požadavků, které jsou uvedeny v sekci 2.2.




Obrázek 2.3: Diagram doménového modelu

### 2.5.1 HTML5

HTML5 je značkovací jazyk používaný pro reprezentaci a strukturování obsahu na internetu (přesněji na WWW). Jedná se již o pátou verzi standardu

Hypertext Markup Language (HTML) (doporučená verze podle The World Wide Web Consortium [W3C] v roce 2018 je HTML 5.2 [1]). Tato verze do standartu přidává mimo jiné nové elementy, atributy a funkcionalitu. Pod pojem HTML5 také často zařazujeme rozsáhlou množinu moderních technologií, které umožňují tvorbu více rozmanitých a mocných webových stránek, či aplikací. [2]

HTML vytvořil Tim Berners-Lee a HTML standart byl definován ve spolupráci s organizací Internet Engineering Task Force (IETF) v roce 1993 [3]. Od roku 1996 převzala vývoj HTML standartu organizace The World Wide Web Consortium (W3C) [4]. Roku 2008 se k W3C přidala organizace Web Hypertext Application Technology Working Group (WHATWG) a započali vývoj standartu HTML5, který společně vydali v roce 2014 [5].

### 2.5.2 JavaScript

JavaScript pod pracovním názvem LiveScript vytvořil Brendan Eich v roce 1995, kdy působil jako inženýr ve firmě Netscape. Přejmenování na JavaScript bylo marketingové rozhodnutí a mělo využít tehdejší popularity programovacího jazyka Java od Sun Microsystem a to přesto, že tyto jazyky spolu téměř nesouvisí. Javascript byl poprvé vydán jako součást prohlížeče Netscape 2 roku 1996. Později téhož roku představila firma Microsoft pro svůj prohlížeč Internet Explorer 3 jazyk JScript, který byl JavaScriptu velice podobný. [6]

Roku 1997 vydala organizace European Computer Manufacturer's Association (ECMA) první verzi standartu ECMAScript, který z původního JavaScriptu a JScriptu vycházel [7]. Tento standart prošel v roce 1999 rozsáhlou aktualizací jako ECMAScript 3, která je bez větších změn používána dodnes [6].

Další verze ECMAScript standartu podle [8] jsou:

- ECMAScript 5 z roku 2009,
- ECMAScript 5.1 z roku 2011 (ISO/IEC 16262:2011),
- ECMAScript 2015,
- ECMAScript 2016,
- ECMAScript 2017,
- připravovaný standart ECMAScript 2018.

Dnes pod označením JavaScript běžně chápeme právě standardizovaný ECMAScript [6] a i já ho tak budu dále označovat.

Javascript je navržen k běhu jako skriptovací jazyk v hostitelském prostředí, které musí zajistit mechanismy pro komunikaci mimo toho prostředí. Nejčastějším hostitelským prostředím je webový prohlížeč, ale Javascript můžeme nalézt i na místech jako je Adobe Acrobat, Adobe Photoshop, SVG

vektorová grafika, serverové prostředí NodeJS, databáze Apache CouchDB, nejrůznější vestavěné systémy a další. [6]

Javascript je více paradigmový, dynamický jazyk s datovými typy, operátory, standardními vestavěnými objekty a metodami. Jeho syntaxe je založena na jazycích Java a C. JavaScript podporuje objektově orientované programování pomocí objektových prototypů namísto tříd jako je tomu například u jazyku Java. Dále také podporuje principy funkcionální programování – funkce jsou také objekty. [6]

### 2.5.3 Node.js

Node.js je JavaScriptové běhové prostředí (anglicky runtime environment), které používá událostmi řízenou architekturu umožňující asynchronní přístup k vstupní/výstupní (I/O) operacím. Tato architektura umožňuje optimalizovat propustnost a škálovatelnost webových aplikací s mnoha I/O operacemi, ale také webových aplikací ve skutečném čase (například komunikační programy nebo hry v prohlížeči). [9]

Toto je v kontrastu s dnešními více známými modely souběžnosti, kde se využívají vlákna operačního systému. Síťová komunikace založená na vláknech je relativně neefektivní a její použití bývá velmi obtížné. [10]

Node.js využívá V8 JavaScript interpret vytvořený společností Google pod skupinou The Chromium Project pro prohlížeč Google Chrome a ostatní prohlížeče postavené na Chromium (prohlížeč s otevřeným zdrojovým kódem od společnosti Google) [11]. V8, který je napsaný v C++, kompiluje JavaScriptový kód do nativního strojového kódu namísto jeho interpretace až za běhu programu. To umožňuje vytvořit rychlé běhové prostředí, které nemusí čekat na překlad potřebného kódu. [9]

### 2.5.4 React

React je Javascriptová knihovna pro tvorbu uživatelského rozhraní [12]. React byl vytvořen Jordanem Walkem, inženýrem ve společnosti Facebook, a byl poprvé použit v roce 2011. Původně byl React určen výhradně pro použití na Facebook Timeline, ale Facebook inženýr Pete Hunt se rozhodl React použít i v aplikaci Instagram. Postupně tak React zbavil závislostí na kód Facebooku a tím napomohl vzniku oficiální React knihovny. React byl představen veřejnosti jako knihovna s otevřeným zdrojovým kódem v květnu roku 2013. [13]

Základním prvkem Reactu jsou takzvané komponenty, které přijímají neměnné vlastnosti a mohou definovat vlastní stavové proměnné. Na základě těchto vlastností a stavu, pak komponenta může rozhodnout co bude jejích výstupem pro uživatele (pomocí metody render). Tato vlastnost se nazývá jednosměrný datový tok (anglicky One-way data flow) a architekturu, kterou React implementuje, nazýváme Flux (ta je součástí Reactu už od samého

počátku). [12] Existují však komunitou vytvořené alternativní nástroje, které řeší datový tok v aplikaci, jako je například knihovna Redux [14].

### 2.5.5 Databáze

Databáze je strukturovaný systém souborů určený pro perzistentní uložení dat. Často je ve smyslu slova databáze chápána i množina podpůrných softwarových nástrojů, tato množiny je nazývána jako Systém Řízení Báze Dat (SŘBD), anglicky Database Management System (DBMS).

Zaměřím se pouze na nejoblíbenější zástupce Structured Query Language (SQL) a Not only SQL (NoSQL) databázi, tedy MySQL a MongoDB [15].

**MySQL** je relační SŘBD vyvíjený a podporovaný společností Oracle. Uložená data musejí mít pevnou strukturu, jsou strukturována do tabulek (jednotlivé záznamy jsou pak jejich řádky). MySQL využívá dotazovací jazyk SQL a každá změna struktury dat vyžaduje migrační proceduru, které může způsobit dočasné problémy s dostupností. MySQL databáze nabízí pouze možnost vertikálního škálování, případně replikaci dat mezi více databázemi. [16]

**MongoDB** je nerelační SŘBD s otevřeným zdrojovým kódem vyvíjený společností MongoDB, Inc. Data nemají pevně definovanou strukturu a jsou representována pomocí Binary JavaScript Object Notation (JSON) (BSON) dokumentů. Díky jejich rychlému převodu na klasický JSON není potřeba využívat složité Object-relational mapping (ORM) nástroje pro mapování dat na objekty, jako tomu je například u SQL databází, a tím umožňuje urychlit celkový vývoj aplikací. Nástroje pro mapování dat z dokumentů na objekty jsou nazývány Object-document mapping (ODM). MongoDB nabízí možnost horizontálního škálování a to až do stovek databázových serverů. [16]

## 2.6 Typy síťové komunikace

V této sekci se zaměřuji na rozdílné přístupy ke komunikaci v architektuře klient-server a popisuji jejich výhody, či nevýhody.

### 2.6.1 Pull technologie

Jako pull technologie označuje klasickou strukturu komunikace architektury klient-server. Iniciátorem spojení je výhradně klient a není možné odeslat data ze serveru ke klientovi bez jeho předchozí žádosti.

Příkladem může být běžný protokol Hypertext Transfer Protocol (HTTP), kde klient (většinou prohlížeč) odesílá požadavek na server a ten mu obratem zašle zpět odpověď. [17]

### 2.6.2 Push technologie

Push technologie označuje strukturu komunikace, která je do jisté míry opačná od Pull technologií. Iniciátorem komunikace je server, který tak může odeslat nová data klientovi i bez jeho žádosti.

Tento přístup lze použít například pro textovou komunikaci ve skutečném čase. Klient nemůže dopředu vědět, zda-li na je na serveru k dispozici nová textová zpráva a neví tedy kdy odeslat požadavek pro získání nové zprávy. V případě Push technologií mu stačí počkat a server mu novou zprávu pošle sám. [17]

#### 2.6.2.1 Short a Long polling

Jednou z nejjednodušších method implementace push technologie je takzvaný **Short pooling**. Jedná se o metodu, kdy se klient musí pravidelně dotazovat serveru na nová data, či nové události a tedy o implementaci pomocí opakovaného užití pull technologie. Pokud server žádná nová data nemá, či nedošlo k žádné nové události, odešle klientovi prázdnou odpověď a ukončí spojení.

Druhou možností je držet spojení mezi klientem a serverem otevřené co nejdéle a odpovědět pouze v případě existence nových dat (takzvaný **Long polling**). Výhodou metody Long polling oproti metodě Short polling je nižší počet požadavků, tedy i nižší objem přenesených dat. Otevřené spojení v případě Long pooling také snižuje dobu, za kterou se ke klientu dostanou nová data.

Hlavní výhodou obou zmíněných method je jednoduchost implementace a to jak na klientské, tak i serverové straně komunikace. Mezi hlavní nevýhody patří režijní náklady spojené s HTTP protokolem a jeho hlavičkami, které musí být neustále přeposílány mezi serverem a klientem, a zvyšování doby mezi přijetím dat serverem a jejich přijetím klientem při časté komunikaci. [18]

#### 2.6.2.2 HTTP streaming

Další metodou implementace push technologie je takzvaný HTTP Streaming, který je podobný metodě Long polling s tím rozdílem, že server data posílá jen jako částečnou odpověď a nemusí tedy ukončit spojení. Tato metoda staví na možnosti webové serveru odesílat více částí dat ve stejné odpovědi (například pomocí hlavičky `Transfer-Encoding: chunked` v rámci protokolu HTTP verze 1.1 a novější).

Výhodou metody HTTP Streaming je snížení latence a snížení režijních nákladů s posíláním HTTP hlaviček, které jsou nutné pouze při vytvoření nového spojení. Mezi nevýhody patří chování výchozí vyrovnávací paměti prohlížečů a klientských knihoven, kde není zajištěn přístup k částečným odpovědím od serveru. [18]

#### 2.6.2.3 Server-Sent Events

Server-Sent Events je způsob implementace push technologie přímo webovým prohlížečem. Mimo běžný HTTP protokol může podporovat i jiné komunikační protokoly (záleží na podpoře prohlížeče) [19]. Z pohledu serveru je jeho použití analogické k Long polling, či HTTP Streaming metodě. [20]

Výhodou Server-Sent událostí je jednoduchá implementace pro webové vývojáře, kteří nemusí využívat externí knihovny, ale mohou použít přímo Application Programming Interface (API) prohlížeče. Nevýhodou je relativně nízká podpora mezi webovými prohlížeči a to převážně mezi prohlížeči pro mobilní zařízení. [18]

#### 2.6.2.4 HTML5 Web Socket

HTML5 Web Socket je protokol, který umožňuje plně duplexní oboustrannou komunikaci mezi klientem a server. Jedná se o samostatný protokol, který netíží režijní náklady spojené s HTTP a umožňuje tak velmi efektivní výměnu informací ve skutečném čase. Web Socket využívá HTTP protokol pouze pro navázání spojení, pro které je následně pomocí hlavičky `Connection: Upgrade` změněn protokol z HTTP právě na Web Socket. [21]

Hlavní výhodou používání protokolu Web Socket je již zmíněná oboustranná komunikace, nízká odezva a nízké režijní náklady, jak na klientské, tak i na serverové straně komunikace. Mezi hlavní nevýhody patří slabší podpora ze strany prohlížečů, která se ovšem kvůli vysoké popularitě stále zlepšuje a obecně je stále lepší než podpora Server-Sent Events. [20]

## 2.7 Algoritmy používané pro kolaborativní editaci textů

Synchronizace dvou a více kopií stejného dokumentu ve skutečném čase je komplexní problém. V této sekci popisuji přístupy k synchronizaci textů, ale také dva nejznámější a nejrozšířenější algoritmy pro synchronizaci textu mezi více klienty při použití architektury klient-server [22].

### 2.7.1 Druhy algoritmů

Existují tři nejčastější přístupy k problému synchronizace, metoda zamykání (nebo také vlastnictví), předávání událost a třísměrné sloučení.

Metoda **zamykaní** je nejjednodušší technika. Ve své nejčastější formě může dokument v jednu chvíli editovat pouze jeden uživatel a to ten který si dokument uzamkl. Ostatní uživatele mohou dokument pouze číst. Provádět změny mohou pouze po uvolnění zámku, stažení nové verze dokumentu a přivlastnění jeho zámku.

Některé vylepšené algoritmy na základě uzamykání se pokoušejí automaticky uzamykat pouze upravované části dokumentu. To však zamezuje úzké spolupráci více uživatelů nad dokumentem, protože každý může pracovat pouze na své uzamčené části.

Metoda **předávání událost** spočívá v myšlence zachycení vše změn provedených nad dokumentem a jejich provedení nad všemi kopiemi zároveň. Hlavní představitelem tohoto přístupu je právě operační transformace o které píší níže v sekci 2.7.3.

Hlavním problémem tohoto přístupu je zachycení všech změn dokumentu, které mohou být triviální jako je například napsání znaku, ale také například vložení obrázku, přetažení velkého množství textu, automatická oprava překlepů a mnoho dalších. Přístup předávání událostí není přirozeně konvergentní. Každá změna, která není správně zachycena (nebo ztracena při cestě po síti), vytváří novou verzi dokumentu, kterou již není možné správně obnovit.

Poslední častou metodou je takzvané **třísměrné sloučení**. Uživatel odešle svůj změněný dokument na server, který provede sloučení s aktuální verzí na serveru a uživateli pošle novou sloučenou verzi zpět. Pokud uživatel provedl v době synchronizace v dokumentu změny novou verzi ignoruje a musí se o synchronizaci pokusit znovu později.

Jedná se poloduplexní systém, dokud uživatel píše nedostává žádné aktualizace o dokumentu a ve chvíli co přestane psát zobrazí se všechny změny od ostatních uživatel nebo vyskočí hláška o nastalé kolizi (to samozřejmě záleží na použitém slučovacím algoritmu). [23] [24]

„*Tento system lze přirovnat k automobilu s čelním sklem, které se stane během jízdy neprůhledné. Podívej se na cestu, pak jeď chvilku poslepu, pak zastav a podívej se znovu. Kdyby všichni řídili stejný druh ‚dívej se anebo jeď' automobilů, těžké nehody by byly velmi časté.*“ [23] přeložil Jiří Šimeček

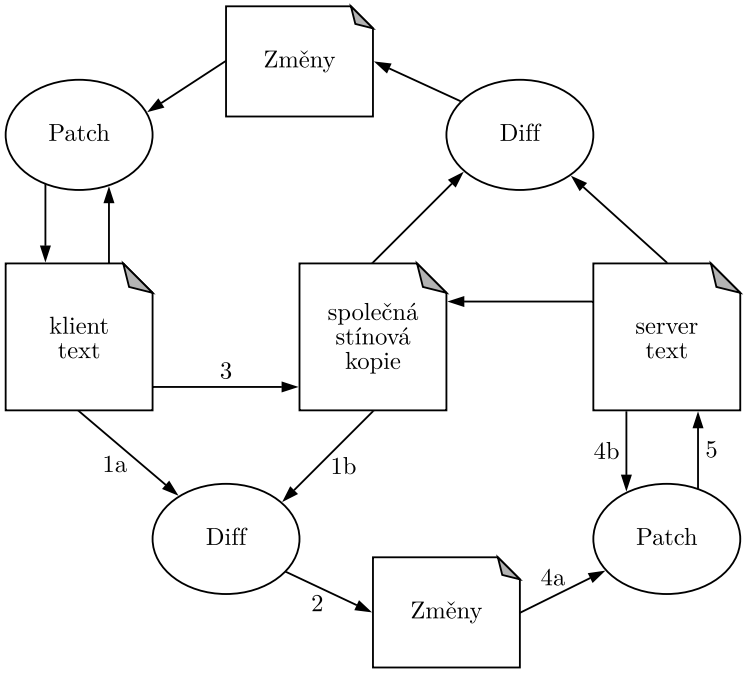
Příkladem systému, který používá třísměrné sloučené je například Apache Subversion (SVN), systém pro správu zdrojových kódů.

### 2.7.2 Diferenciální synchronizace

Diferenciální synchronizace (DS) je algoritmus, který vymyslel Neil Fraser a lze považovat za zástupce třísměrného sloučení. DS řeší problém s průběžnou synchronizací, kterým trpí klasický model třísměrného sloučení, a to použitím stínových kopií a diff-patch algoritmu nejen na serveru, ale i u každé kopie dokumentu.

Na obrázek číslo 2.4 je vidět zjednodušený vývojový diagram DS algoritmu. Diagram předpokládá dva dokumenty (pojmenované server text a klient text), které jsou umístěné na stejném počítači a tedy nepočítá se síťovou dobou odezvy.

Na začátku jsou všechny dokumenty stejné (klient text, server text i společná stínová kopie). Klient text i server text mohou být libovolně upraveny




Obrázek 2.4: Diferenciální synchronizace vývojový diagram [23]

a naším cílem je udržet oba dokumenty neustále co nejvíce podobné.

Algoritmus pokračuje následujícími kroky (viz čísla u jednotlivých přechodů diagramu 2.4):

1. klient text je porovnán oproti společné stínové kopii,
2. výsledkem porovnání je seznam změn, které byly provedeny na klient text kopii dokumentu,
3. klient text je překopírován přes společnou stínovou kopii,
4. seznam změn je aplikován na server text (za použití best-effort match algoritmu),
5. server text je přepsán výsledkem aplikace změn.

Důležité je, že pro správné fungování musí být kroky 4 a 5 atomické, tedy nesmí se stát, že by se server text v tuto chvíli změnil.

Algoritmus předpokládá použití libovolného diff-patch algoritmu, který umožňuje aplikovat změny i na dokument, který se změnil. Lze například použít diff-match-patch algoritmus od společnosti Google. Ten implementuje diff algoritmus od Eugene W. Myers, který je považování za nejlepší obecně použitelný diff algoritmus [25]. [23] [24]

### 2.7.3 Operační transformace

Operační transformace (OT) je algoritmus, který se poprvé objevil ve výzkumném článku s názvem Kontrola souběhu v skupinových systémech od autorů Ellis a Gibbs roku 1989. Jedná se o nejčastěji používaný algoritmus pro kolaborativní spolupráci ve skutečném čase. [26]

Algoritmu je možné použít pro synchronizaci dokumentů různých formátů, ale pro jednoduchost se zaměřím pouze na textové dokumenty bez jakýchkoliv formátovacích značek (například zdrojový kód).

Základní jednotkou celého algoritmu je **operace**. Operace označuje změnu v dokumentu a u čistě textových dokumentů rozlišujeme tři druhy operací:

- přeskoč s parametrem počet znaků, který označuje počet znaků k přeskočení,
- odstraň řetězec s řetězcem jako parametrem, který označuje řetězec k odstranění,
- přidej řetězec s parametrem který označuje řetězec k přidání. [27]

**Transformace dokumentu** Pomocí těchto třech operací jsme schopni popsat jakoukoli změnu, která mohla v dokumentu nastat, a zároveň daný dokument upravit, tak aby odpovídal stavu po této operaci. Tento proces aplikace operace (či jejich souboru) se nazývá transformace dokumentu a umožňuje upravit dokument bez nutnosti jeho uzamčení, či řešení konfliktů.

Transformace má i své omezení, které říká, že součet znaků všech operací transformace se musí rovnat délce dokumentu nad kterým bude transformace provedena. Toto omezení není například u aplikace změn v rámci algoritmu DS vůbec potřeba, protože změny nenesou informaci o délce celého dokumentu.

**Kombinace operací** Operace také můžeme kombinovat, určitě totiž platí, že pokud máme dokument A, soubor operací transformující dokument A na B a soubor operací transformující dokument B na C, tak jistě existuje soubor operací transformující dokument A na dokument C.

**Transformace operací** Dále lze operace transformovat a to pomocí jiných operací. Mějme dokument A, soubory operací z A do B, z A do C, pak aplikací změn, z kterých byl vytvořen druhý soubor, na dokument B získáme nový soubor operací (soubor transformovaný). [28]

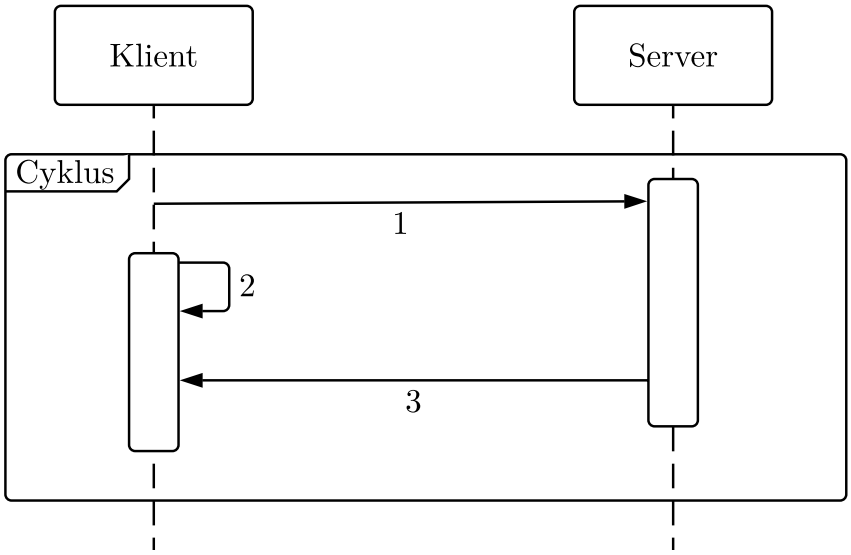
Základní komunikace se skládá z následujících kroků (také znázorněna pomocí sekvenčního diagramu na obrázku číslo 2.5):

1. po změně dokumentu klient změnu popíše pomocí souboru operací, který odešle na server,

2. klient do přijetí potvrzení nesmí na server odeslat další operace, další změny si klient ukládá a kombinuje do jednoho velkého souboru operací,

3. server po přijetí odešle klientu, který soubor vytvořil, potvrzení o přijetí přijetí a pošle soubor všem ostatním klientům,

4. klient po přijetí souboru může odeslat nastřádaný soubor zkombinovaných operací (pokračuje bodem 2) nebo čeká na další změnu a pokračuje znovu bodem 1. [29]




Obrázek 2.5: Operační transformace sekvenční diagram

O krok s čekáním klienta před odesláním dalších operací na potvrzení přijetí posledního souboru serverem se zaručili vývojáři společnosti Google, kteří toto pravidlo poprvé použili pro službu Google Wave (více o službě v sekci 2.8.1). Tento krok snižuje náročnost celého algoritmu omezením počtu současných operací, které by musel server kombinovat. [29]

Pokud klient při čekání na potvrzení od serveru dostane cizí soubor operací musí tento soubor aplikovat na svou kopii dokumentu a transformovat již uložené operace v závislosti na přijatých operacích. Také pokud server dostane od klienta soubor operací, který vznikl před potvrzením posledního souboru, musí soubor transformovat přes všechny soubory operací, která server potvrdil po jejím vytvořem. Z tohoto důvodu jsou soubory operací číslovány a klient s novým souborem operací odesílá na server i poslední pro něj známé číslo souboru.

Implementace obecného algoritmu OT pro různé typy dokumentů není vůbec jednoduchá, což potvrzuje i následující tvrzení tehdejšího vývojáře Google Wave Josepha Gentle:

„*Naneštěstí, implementovat OT není vůbec veselé. Existuje milion algoritmů s různými kompromisy, které jsou však většinou uvězněny ve vědeckých pracích. Tyto algoritmy je opravdu obtížné a časově náročné správně implementovat. […] Jsem bývalý vývojář Google Wave. Napsání Wave trvalo 2 roky a pokud bychom ho dnes chtěli přepsat, napodruhé by to trvalo téměř stejně tak dlouho.*“ [30] přeložil Jiří Šimeček

## 2.8 Existující řešení

V této sekci představuji již existující nástroje pro kolaborativní editaci textů a prozradím, který algoritmus pro sdílení textů používají.

### 2.8.1 Apache Wave

Apache Wave je nástupcem produktu Google Wave od společnosti Google. V lednu roku 2018 byl jeho vývoj pod záštitou Apache Foundation ukončen a není v něm již pokračováno. [31]

Jedná se o serverové řešení komunikace v skutečném čase napsané v jazyce Java, které obsahuje implementaci protokolu Wave Federation. Wave Federation protokol byl navržen jako rozšíření algoritmu OT (více o algoritmu v 2.7.3) společností Google a zasloužil se o důležité vylepšení v podobě potvrzování každé přijaté operace serverem. [31, 32] Projekt dnes není prakticky nasaditelný (nevyšla žádná stabilní verze), ale je považován za důležitý krok k rozšíření kolaborativní editace [28].

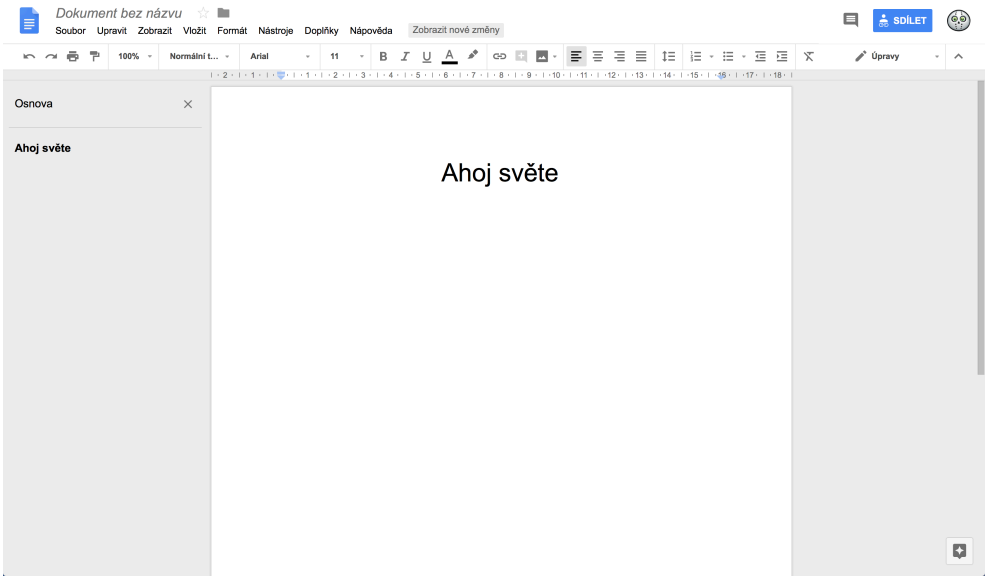
### 2.8.2 Google Docs

Google Docs je textový procesor, který je spolu s dalšími kancelářskými aplikacemi součástí služby Google Drive od společnosti Google. Jedná se editor typu „co vidíš, to dostaneš“ (WYSIWYG), který je podobný ostatním kancelářským textovým procesorům (jako je například Microsoft Word, OpenOffice Writer a další). [33]

Google Docs využívá Google Apps API, které je také postaveno nad algoritmem OT (více o algoritmu v 2.7.3) [34] a to včetně vylepšení se kterým přišel Google při vývoji Google Wave [35].

### 2.8.3 Etherpad

EtherPad jako webová služba pro kolaborativní editaci textů byla odkoupena společností Google roku 2009 za účelem integrace do tehdejší služby Google




Obrázek 2.6: Ukázka aplikace Google Docs

Wave [36]. Google poté zveřejnil zdrojové kódy služby a vznil tak projekt Etherpad, tedy webový textový procesor s otevřeným zdrojovým kódem [37].

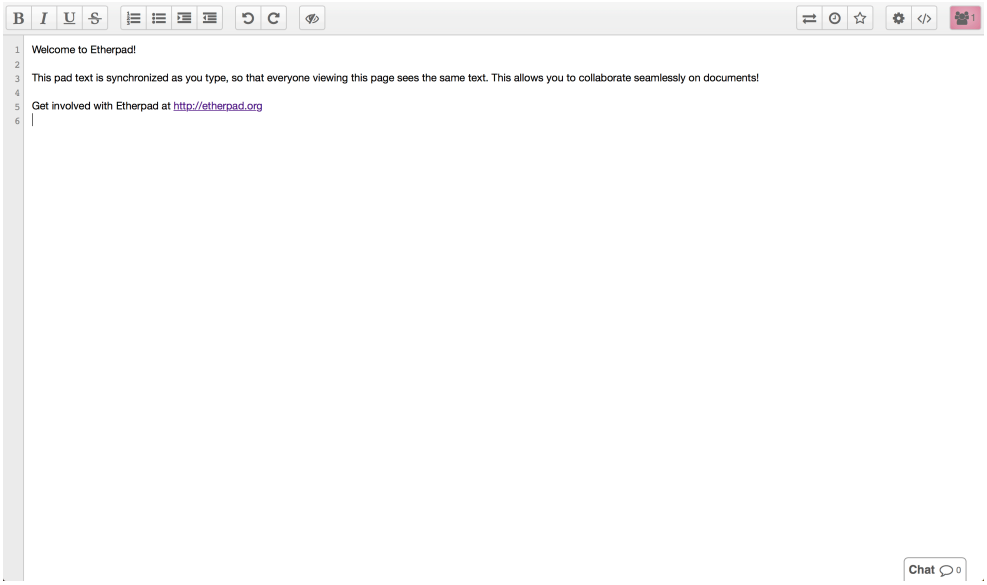
Etherpad byl od zveřejnění otevřeného kódu přepsán z jazyka Scala do JavaScriptu (více o JavaScriptu v 2.5.2) pro serverové prostředí NodeJS (více o NodeJS v 2.5.3), ale původní synchronizační knihovna nazývaná EasySync zůstává nadále stejná [38] [39]. Knihovna EasySync (a tím tedy i Etherpad samotný) využívá principy algoritmu OT (více o algoritmu v 2.7.3) a to včetně vylepšení ve formě čekání klienta po odeslání operace na potvrzení od serveru [39]. Dnes existuje množství nejrůznějších zásuvných modulů, které rozšiřují možnosti nástroje Etherpad a to i včetně modulu pro komunikaci pomocí Web Real-Time Communication (WebRTC), protokol umožňující implementaci audio, či video hovorů přímo ve webové prohlížeči. [40].

### 2.8.4 Codeshare.io

Codeshare.io je textový editor zaměřený na vývojáře a jejich spolupráci ve skutečném čase. Jako hlavní využití aplikace Codeshare.io její autor zmiňuje pohovory s vývojáři a to díky integrovanému video chatu pomocí technologie WebRTC. [41]

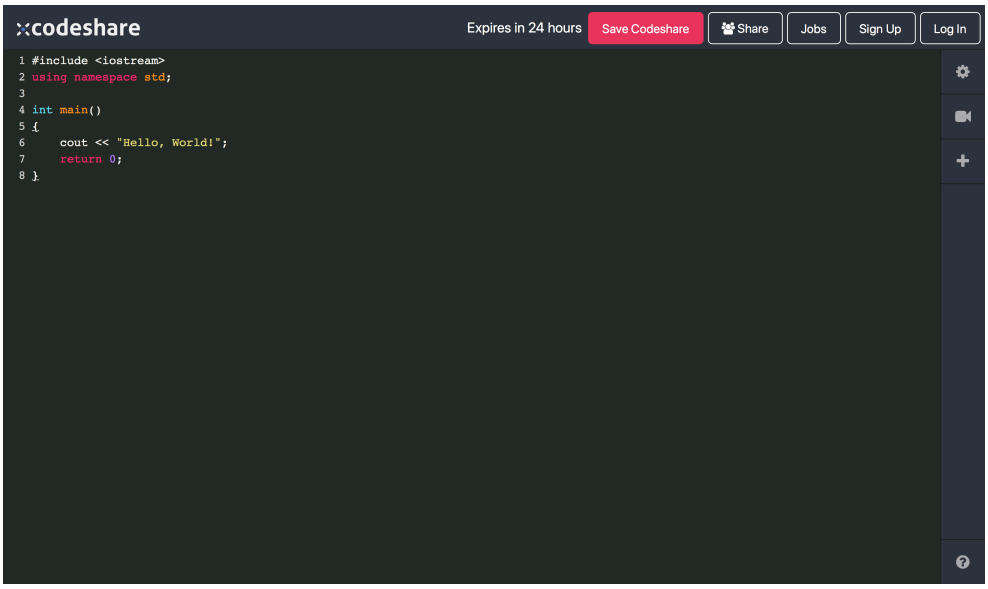
„*Spojil jsem Firebase s Ace editorem a vznikl Codeshare.io, nástroj pro sdílení kódu ve skutečném čase.*“ [42] přeložil Jiří Šimeček

Codeshare.io využívá k synchronizaci textu databázi Firebase Real-time Datebase od společnosti Google. Tato databáze umožňuje jednotlivým klien-




Obrázek 2.7: Ukázka aplikace Etherpad

tům naslouchat, kdy se v databázi změní data a následně tyto data načíst. Jedná se o velmi jednoduchý model synchronizace obsahu na bázi atomických změn a není zde využito žádného pokročilejšího algoritmu pro synchronizaci textu. [42]




Obrázek 2.8: Ukázka aplikace Codeshare.io

Kapitola 3

# Návrh

## 3.1 Architektura

Architektura určuje strukturu a části aplikace. V této sekci vysvětlím jednotlivé části navržené architektury prototypu.

Navrženou architekturu rozdělím na tři část, klientskou, serverovou a databázovou část. Jedná se o model klient-server (viz obrázek 3.1), protože jsou od sebe role klienta a serveru odděleny. Komunikace mezi nimi probíhá pomocí předem definovaných aplikačních rozhraní postavených na protokolu HTTP (více o rozhraní v sekcích 3.5 a 3.6).

### 3.1.1 Klientská část

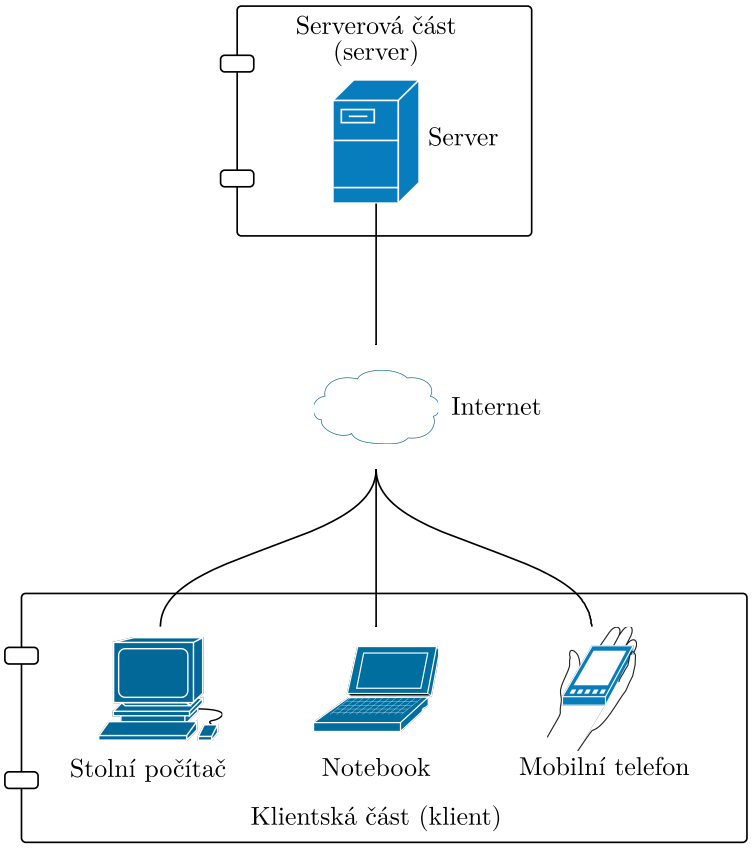
Jedná se o část aplikace, která běží v samotném prohlížeči uživatele. Reaguje na uživatelský vstup, generuje uživatelské rozhraní podle stavu aplikace a pomocí protokolu HTTP komunikuje se serverovou částí za účelem úpravy, či získání dat.

Klientská část je napsána v jazycích HTML5 a JavaScript, ale také využívá knihovny pro tvorbu uživatelského rozhraní ReactJS (více o technologií v sekci 2.5).

### 3.1.2 Serverová část

Serverová část je centrem aplikace, její hlavní zodpovědností je poskytnutí autentizace a autorizace jednotlivých uživatelů. Dále je tato část zodpovědná za zajištění konzistence dat v perzistentním (databázovém) uložišti a poskytuje rozhraní pro komunikaci s klientskou částí aplikace.

Serverová část využívá JavaScriptové běhové prostředí Node.js (více o prostření v sekci 2.5.3) a je navržena podle dvouvrstvé architektury. Datová vrstva zajišťuje přístup k databázové části a za pomoci návrhového vzoru repositář (anglicky repository pattern) vystavuje rozhraní v rámci serverové části, které




Obrázek 3.1: Diagram architektury aplikace (klient-server)

umožňuje přístup k datům. Presenční vrstva zajišťuje komunikaci s klientskou části a validitu přijatých dat, volá jednotlivé funkce datové vrstvy za účelem získání, či uložení dat.

### 3.1.3 Databázová čast

Tato poslední část aplikace je zodpovědná za poskytování rozhraní pro přístup a operace na perzistentním uložištěm.

Rozhodl jsem se pro použití SŘBD MongoDB (více o databázích v sekci 2.5.5). MongoDB umožňuje rychlejší vývoj aplikací a pro problém editace textů se jako zástupce NoSQL hodí lépe, než tradiční SQL databáze. U webové editoru textů lze očekávat stále rostoucí počet dokumentů a jejich úprav, což by mohl být pro SQL databáze problém hlavně z pohledu budoucího škálování aplikace.

MongoDB poskytuje Transmission Control Protocol/Internet Protocol (TCP/IP) rozhraní pro přístup a manipulaci s dokumenty, které následně využívá serve-

rová část aplikace.

## 3.2 Databázové schéma

Jelikož jsem se rozhodl použít NoSQL databázi, nemám jak pevně definovat databázové schéma jako v obvyklé relační databázi. Konzistenci a validitu ukládaných dat musí zajistit sama aplikace a to jak při čtení, tak i při zápisu dat.

Z tohoto důvodu jsem se rozhodl použít ODM nástroj Mongoose, který umožňuje v rámci aplikace definovat struktury jednotlivých dokumentů pro MongoDB a jejich validitu kontroluje před každým uložením. Schéma na obrázku 3.2 není databázovým schématem v pravém slova smyslu, ale jedná se o schéma definované za pomoci právě ODM Mongoose.

Jednotlivé entity schématu přímo reflektují entity z doménového modelu uvedeného v sekci 2.4. Přibyly pouze implementační detaily a atributy jednotlivých relací mezi entitami. Také se změnily názvy entit a jejich atributů z českého do anglického jazyka, aby lepé reflektovali samotný kód aplikace.

Jedinou výraznou změnou oproti doménovému modelu, která zde stojí za zmínku, je entita `User`. Přibyly zde atributy sloužící pro autentizaci pomocí služeb třetích stran (například sociální sítě, či ČVUT heslo). Tyto atributy nesou identifikační údaj pro uživatele v rámci dané autentizační služby třetí strany jako je například facebookId pro sociální síť Facebook, či ČVUT uživatelské jméno pro přihlášení pomocí ČVUT hesla.

Kompletní definice schématu lze nalézt spolu s ostatními zdrojovými kódy na přeloženém CD ve složce `/app/src/model`. Na definici lze i mimo jiné pozorovat validační pravidla pro jednotlivé atributy, jako jsou například číselné rozsahy, maximální délky atd.

## 3.3 Algoritmus synchronizace editovaného textu

Pro účely synchronizace textu ve skutečném čase jsem se rozhodl pro použití algoritmu OT (více o algoritmu v sekci 2.7.3) a to i přes jeho složitější implementaci. Algoritmus je dostatečně rozšířený a otestovaný praxí (příkladem jeho úspěšného použití jsou projekty jako Google Wave a jeho mladší sourozenec Google Docs, viz 2.8).

Samozřejmě jsem nechtěl celý algoritmus implementovat znovu, a tak jsem rozhodl použít knihovnu OT.js. Tato knihovna implementuje základní operace algoritmu OT pro práci s textem a také obsahuje ukázku jak knihovnu použít s knihovnami třetích stran. Bohužel knihovna není od roku 2015 vyvíjena, téměř pro ni neexistuje dokumentace a část jejího kódu jsem musel značně upravit, protože používá až 5 let staré verze knihoven třetích stran, které nejsou kompatibilní s dnešními verzemi těchto knihoven, či dnes již vůbec

neexistují. Ke knihovně OT.js se vracím v části 3.7, v souvislosti s návrhem komponenty editoru.

## 3.4 Autentizace

Autentizace (anglicky authentication) je proces určení skutečné identity uživatele. Pro webové aplikace se nejčastěji jedná o klasické přihlášení uživatele do aplikace pod svým uživatelským účtem. K autentizaci lze nejčastěji rozdělit na dva způsoby, stavovou a bezstavovou autentizace.

### 3.4.1 Stavová autentizace

Stavová autentizace (viz diagram na obrázku 3.3) je nejčastější způsob autentizace na internetu. Tento způsob je velmi jednoduchý na implementaci a díky své oblíbenosti lze jeho implementaci najít v každé větší webové knihovně, či frameworku.

Server si musí pamatovat uložená data pro všechny přihlášené uživatele (nejčastěji v Session), ale také klient si musí pamatovat identifikátor těchto dat pomocí (nejčastěji v Cookies). Pro správné fungování stavová autentizace potřebuje, aby se všechny části aplikace, které potřebují přístup k informacím o přihlášeném uživateli, nacházeli na stejné doméně. Toto omezení vyplývá z použití cookies na straně klienta a bezpečnostním opatřením ze strany prohlížeče ohledně přístupu k nim.

### 3.4.2 Bezstavová autentizace

Bezstavová autentizace (viz diagram na obrázku 3.4) je v poslední době velmi oblíbený způsob autentizace především pro Representational state transfer (REST) API. Autentizační server oproti nejčastěji přihlašovacím údajům vydává takzvaný token (speciální kód, který identifikuje uživatele).

Autentizační token samotný může nést více informací o uživateli, jako je například jeho jméno, ale také jeho oprávnění pro přístup ke zdrojům aplikace. Díky tomu, že token nese informace o oprávnění nemusí, aplikační server při každém dotazu komunikovat s DB a zjišťovat uživatelovo oprávnění. Bezstavová autentizace napomáhá škálování a rychlosti webových aplikací, ale je složitější na implementaci, jak na straně klient, tak na straně serveru.

### 3.4.3 Výběr způsobu autentizace

Pro implementaci autentizace jsem se rozhodl použít jednoduchou stavovou autentizaci za pomoci klientského uložiště cookies a serverového uložiště session, které je perzistentně uloženo v DB. Tento způsob jsem zvolil s ohledem na jednoduchost navrhovaného prototypu.

Díky stavové autentizaci nemusí klientská strana autentizaci vůbec řešit, protože prohlížeč odesílá platné cookies při každém požadavku automaticky. To platí jak pro komunikaci pomocí REST (viz následující sekce 3.5), tak i pro komunikaci ve skutečném čase (viz následující sekce 3.6), kde se cookies odesílají při navázání spojení. Nevýhodou tohoto řešení je, že obě části aplikace (klientská i serverová) musí být umístěny na stejné doméně. V případě navrhovaného prototypu toto ale není problém, jelikož klientská i serverová část jsou sloučeny do jedné aplikace (viz sekce **??** o zveřejnění klientské části) a tudíž sdílí jednu společnou doménu.

## 3.5 REST komunikace

Základním způsobem komunikace mezi klientskou a serverovou části aplikace je REST komunikace.

K implementaci jednotlivých koncových bodů, ale i obecnému zpracování všech HTTP požadavků, jsem se rozhodl použít již připravené řešení v podobě Node.js knihovny, či balíčku knihoven (dále jen framework). Pro prostření Node.js existuje mnoho frameworků pro implementaci REST API, podle [43] jsem výběr zúžil na 3 neoblíbenější Node.js frameworky: Sails.js, Express.js a Hapi.js.

### 3.5.1 Sails.js

Sails.js je plnohodnotný webový Model–view–controller (MVC) framework pro Node.js. Jeho integrovanou součástí je Waterline ORM, který umožňuje použít téměř libovolný SŘBD.

Waterline je, ale také záporem celého frameworku, protože není mezi vývojáři příliš rozšířený a občas ho není jednoduché použít (například mapování vnořených objektů).

### 3.5.2 Express.js

Express.js je velice rozšířený, jednoduchý a lehký framework pro Node.js. V základní konfiguraci obsahuje pouze základní logiku ohledně zpracování HTTP požadavků.

Ale jako každý framework má i své nevýhody. Mezi hlavní nevýhody patří nepříliš propracované zpracování chybových stavů nebo také nedostatečná kódová nezávislost, což může komplikovat další vývoj a znovupoužitelnost částí aplikace.

### 3.5.3 Hapi.js

Hapi.js je framework pro Node.js vyvíjený společností WalMart. Byl vytvořen jako přímá náhrada frameworku Express.js a snaží se řešit jeho nedostatečnou

kódovou nezávislost použitím modulární architektury.

Modulární architektura tento problém sice řeší, ale přidává do frameworku vysokou složitost návrhu. Tato složitost je pro většinu projektů zbytečně vysoká a nevyrovná se přidané hodnotě, která framework přináší oproti použití Express.js.

### 3.5.4 Výběř frameworku

Nakonec jsem se rozhodl použít framework Express.js, převážně kvůli jeho jednoduchosti a rozšířenosti mezi vývojáři, která může být nápomocna hlavně při řešení potenciálního problému. Frameworky Sail.js a Hapi.js se svou velikostí hodí spíše pro větší produkční systémy a nejsou příliš vhodné pro implementaci prototypu této aplikace.

### 3.5.5 Seznam koncových bodů

V této sekci je obsažen kompletní výpis REST koncových bodů. Tyto koncové body přijímají různý počet parametrů, které mohou být jako součást cesty požadavku nebo v těle požadavku ve formátu JSON, a vrací HTTP stavový kód spolu s nepovinnou odpovědí, která je také ve formátu JSON.

#### 3.5.5.1 Registrace uživatele

Koncový bod umožňující registraci nového uživatele. Tento koncový bod přijímá parametry uživatelské jméno, heslo a email.

Více informací o koncovém bodu lze nalézt v tabulce 3.1.

Tabulka 3.1: Koncový bod Registrace uživatele

| | | |
|---|---|---|
| URL: | /api/auth | |
| HTTP metoda: | POST | |
| URL parametry: | žádné | |
| Data parametry: | username | [String] |
| | email | [String] |
| | password | [String] |
| Úspěch: | 200 | Informace o vytvořeném uživateli |
| Neúspěch: | 422 | Popis chyby |

#### 3.5.5.2 Přihlášení uživatele

Koncový bod pro přihlášení již registrovaného uživatele pomocí uživatelské jména a hesla.

Více informací o koncovém bodu lze nalézt v tabulce 3.2.

Tabulka 3.2: Koncový bod Přihlášení uživatele

| URL: | /api/auth/signIn | |
|---|---|---|
| HTTP metoda: | POST | |
| URL parametry: | žádné | |
| Data parametry: | username<br>password | [String]<br>[String] |
| Úspěch: | 200 | Informace o přihlášeném uživateli |
| Neúspěch: | 422 | Popis chyby |

#### 3.5.5.3 Ohlášení uživatele

Koncový bod pro odhlášení aktuálně přihlášeného uživatele.

Více informací o koncovém bodu lze nalézt v tabulce 3.3.

Tabulka 3.3: Koncový bod Ohlášení uživatele

| URL: | /api/auth | |
|---|---|---|
| HTTP metoda: | DELETE | |
| URL parametry: | žádné | |
| Data parametry: | žádné | |
| Úspěch: | 204 | |
| Neúspěch: | 401, 422 | Popis chyby |

#### 3.5.5.4 Odeslání požadavku na obnovu hesla

Koncový bod sloužící k obnově zapomenutého hesla. Přijímá parametry email a uživatelské jméno.

Pokud je nalezen uživatel se přijatými údaji je mu odeslán email obsahující instrukce pro obnovu hesla pomocí unikátního vygenerovaného kódu.

Tabulka 3.4: Koncový bod Odeslání požadavku na obnovu hesla

| URL: | /api/auth/forgotPassword | |
|---|---|---|
| HTTP metoda: | POST | |
| URL parametry: | žádné | |
| Data parametry: | username<br>email | [String]<br>[String] |
| Úspěch: | 200 | Neutrální zpráva o provedení operace |
| Neúspěch: | Nenastává | |

#### 3.5.5.5 Zjištění platnosti kódu pro obnovu hesla

Koncový bod zjišťující existenci kódu pro obnovy hesla předaného pomocí parametru token.

Tabulka 3.5: Koncový bod Zjištění platnosti kódu pro obnovu hesla

| URL: | /api/auth/forgotPassword/:token | |
|---|---|---|
| HTTP metoda: | GET | |
| URL parametry: | token | [String] |
| Data parametry: | žádné | |
| Úspěch: | 204 | |
| Neúspěch: | 404 | Popis chyby |

#### 3.5.5.6 Obnova hesla pomocí kódu

Koncový bod, který umožňuje za podmínky validního kódu pro obnovu hesla (parametr token) provést změnu hesla na nové heslo.

Tabulka 3.6: Koncový bod Obnova hesla pomocí kódu

| URL: | /api/auth/forgotPassword/:token | |
|---|---|---|
| HTTP metoda: | PUT | |
| URL parametry: | token | [String] |
| Data parametry: | newPassword | [String] |
| Úspěch: | 204 | |
| Neúspěch: | 404, 422 | Popis chyby |

#### 3.5.5.7 Informace o přihlášeném uživateli

Koncový bod pro zjištění informací o aktuálně přihlášeném uživateli.

Tabulka 3.7: Koncový bod Informace o přihlášeném uživateli

| URL: | /api/user | |
|---|---|---|
| HTTP metoda: | GET | |
| URL parametry: | žádné | |
| Data parametry: | žádné | |
| Úspěch: | 200 | Informace o přihlášeném uživateli |
| Neúspěch: | 401 | Popis chyby |

#### 3.5.5.8 Změna údajů přihlášeného uživatele

Koncový bod umožňující aktualizaci jednotlivých parametrů přihlášeného uživatele.

Parametry jsou volitelné, stačí tedy odeslat pouze údaje, které mají být změněné. Veškeré změny musí být potvrzeny platným aktuální heslem (parametr password je tedy povinný vždy).

Tabulka 3.8: Koncový bod Změna údajů přihlášeného uživatele

| | | |
|---|---|---|
| URL: | /api/user | |
| HTTP metoda: | PUT | |
| URL parametry: | žádné | |
| Data parametry: | username | [String] |
| | email | [String] |
| | newPassword | [String] |
| | password | [String] |
| Úspěch: | 204 | |
| Neúspěch: | 401, 422 | Popis chyby |

#### 3.5.5.9 Výchozí nastavení dokumentů

Koncový bod pro zjištění výchozího nastavení pro nové dokumenty aktuálně přihlášeného uživatele.

Tabulka 3.9: Koncový bod Výchozí nastavení dokumentů

| | | |
|---|---|---|
| URL: | /api/user/document-settings | |
| HTTP metoda: | GET | |
| URL parametry: | žádné | |
| Data parametry: | žádné | |
| Úspěch: | 200 | Výchozí nastavení dokumentů |
| Neúspěch: | 401 | Popis chyby |

#### 3.5.5.10 Změna výchozího nastavení dokumentů

Koncový bod umožňující změnu jednotlivých polí výchozího nastavení pro dokumenty aktuálně přihlášeného uživatele. Přijímanými parametry jsou všechny pole entity DocumentSettings z databázového schématu na obrázku 3.2 (kromě pole _id).

#### 3.5.5.11 Vytvoření dokumentu

Koncový bod umožňující aktuálně přihlášenému uživateli vytvořit nový dokument.

Tabulka 3.10: Koncový bod Změna výchozího nastavení dokumentů

| URL: | /api/user/document-settings | |
|---|---|---|
| HTTP metoda: | PUT | |
| URL parametry: | žádné | |
| Data parametry: | theme | [String] |
| | mode | [Number] |
| | tabSize | [Number] |
| | indentUnit | [Number] |
| | indentWithTabs | [Boolean] |
| | fontSize | [Number] |
| | keyMap | [String] |
| | styleActiveLine | [String] |
| | lineWrapping | [Boolean] |
| | lineNumbers | [Boolean] |
| Úspěch: | 200 | Výchozí nastavení dokumentů |
| Neúspěch: | 401, 422 | Popis chyby |

Tabulka 3.11: Koncový bod Vytvoření dokumentu

| URL: | /api/document | |
|---|---|---|
| HTTP metoda: | POST | |
| URL parametry: | žádné | |
| Data parametry: | žádné | |
| Úspěch: | 200 | Identifikátor vytvořeného dokumentu |
| Neúspěch: | 401 | Popis chyby |

#### 3.5.5.12 Vytvořené dokumenty

Koncový bod pro získání dokumentů vytvořených aktuálně přihlášeným uživatelem (je jejich vlastníkem).

Tabulka 3.12: Koncový bod Vytvořené dokumenty

| URL: | /api/document | |
|---|---|---|
| HTTP metoda: | GET | |
| URL parametry: | žádné | |
| Data parametry: | žádné | |
| Úspěch: | 200 | Seznam dokumentů |
| Neúspěch: | 401 | Popis chyby |

#### 3.5.5.13 Poslední dokumenty

Koncový bod pro získání dostupných dokumentů, ke kterým v minulosti přistoupil aktuálně přihlášený uživatel.

Tabulka 3.13: Koncový bod Poslední dokumenty

| URL: | /api/document/last | |
|---|---|---|
| HTTP metoda: | GET | |
| URL parametry: | žádné | |
| Data parametry: | žádné | |
| Úspěch: | 200 | Seznam dokumentů |
| Neúspěch: | 401 | Popis chyby |

#### 3.5.5.14 Sdílené dokumenty

Koncový bod pro získání dokumentů, ke kterým byl aktuálně přihlášený uživatel přizván.

Tabulka 3.14: Koncový bod Sdílené dokumenty

| URL: | /api/document/shared | |
|---|---|---|
| HTTP metoda: | GET | |
| URL parametry: | žádné | |
| Data parametry: | žádné | |
| Úspěch: | 200 | Seznam dokumentů |
| Neúspěch: | 401 | Popis chyby |

#### 3.5.5.15 Komunikační vlákno dokumentu

Koncový bod pro získání zpráv komunikačního vlákna dokumentu identifikovaného pomocí parametru documentId. Pro jeho použití musím mít uživatel dostatečná práva v rámci dokumentu pro přístup ke zprávám.

Počet vrácených zpráv lze ovlivnit volitelným Uniform Resource Locator (URL) parametrem number a čas odeslání poslední vrácené zprávy lze určit volitelným URL parametrem lastDate.

#### 3.5.5.16 Odeslání nové zprávy

Koncový bod umožňující vytvoření nové zprávy pro dokument identifikovaný pomocí parametru documentId s textem určeným parametrem message.

Pro jeho použití musím mít uživatel dostatečná práva v rámci dokumentu pro přístup ke zprávám.

Tabulka 3.15: Koncový bod Komunikační vlákno dokumentu

| URL: | /api/document/:documentId/messages | |
|---|---|---|
| HTTP metoda: | GET | |
| URL parametry: | documentId<br>lastDate<br>number | [String]<br>[Date]<br>[Number] |
| Data parametry: | žádné | |
| Úspěch: | 200 | Seznam zpráv |
| Neúspěch: | 403, 404, 422 | Popis chyby |

Tabulka 3.16: Koncový bod Odeslání nové zprávy

| URL: | /api/document/:documentId/messages | |
|---|---|---|
| HTTP metoda: | POST | |
| URL parametry: | documentId | [String] |
| Data parametry: | message | [String] |
| Úspěch: | 204 | |
| Neúspěch: | 401, 403, 404, 422 | Popis chyby |

#### 3.5.5.17 Práva dokumentu

Koncový bod pro získání informací ohledně oprávnění a jednotlivých pozvánek dokumentu identifikovaného pomocí parametru documentId.

Pro jeho použití musím mít uživatel dostatečná práva v rámci dokumentu pro přístup k pozvánkám a sdílení.

Tabulka 3.17: Koncový bod Práva dokumentu

| URL: | /api/document/:documentId/rights | |
|---|---|---|
| HTTP metoda: | GET | |
| URL parametry: | documentId | [String] |
| Data parametry: | žádné | |
| Úspěch: | 200 | Informace o sdílení dokumentu |
| Neúspěch: | 403, 404 | Popis chyby |

#### 3.5.5.18 Změna práv veřejného odkazu dokumentu

Koncový bod pro úpravu oprávnění pro uživatele přistupující k dokumentu (identifikovaného pomocí parametru documentId) pomocí veřejného odkazu.

Pro jeho použití musím mít uživatel dostatečná práva v rámci dokumentu pro přístup k pozvánkám a sdílení.

Tabulka 3.18: Koncový bod Změna práv veřejného odkazu dokumentu

| URL: | /api/document/:documentId/rights | |
|---|---|---|
| HTTP metoda: | PUT | |
| URL parametry: | documentId | [String] |
| Data parametry: | shareLinkRights | [Number] |
| Úspěch: | 204 | |
| Neúspěch: | 403, 404, 422 | Popis chyby |

#### 3.5.5.19 Pozvání uživatele k dokumentu

Koncový bod pro úpravu oprávnění pro přizvaného uživatele k dokumentu (identifikovaného pomocí parametru documentId). Pozvaný uživatel je identifikován pomocí uživatelského jména v parametru.

Pro jeho použití musí mít uživatel dostatečná práva v rámci dokumentu pro přístup k pozvánkám a sdílení.

Tabulka 3.19: Koncový bod Pozvání uživatele k dokumentu

| URL: | /api/document/:documentId/rights/invite | |
|---|---|---|
| HTTP metoda: | PUT | |
| URL parametry: | documentId | [String] |
| Data parametry: | rights | [Number] |
| | to | [String] |
| Úspěch: | 204 | |
| Neúspěch: | 403, 404, 422 | Popis chyby |

#### 3.5.5.20 Odstranění pozvánky k dokumentu

Koncový bod umožňující odstranění pozvánky pro uživatele (identifikovaného pomocí parametru toUserId) k dokumentu (identifikovaného pomocí parametru documentId).

Pro jeho použití musím mít uživatel dostatečná práva v rámci dokumentu pro přístup k pozvánkám a sdílení.

Tabulka 3.20: Koncový bod Odstranění pozvánky k dokumentu

| URL: | /api/document/:documentId/rights/:toUserId | |
|---|---|---|
| HTTP metoda: | DELETE | |
| URL parametry: | documentId | [String] |
| | toUserId | [String] |
| Data parametry: | žádné | |
| Úspěch: | 204 | |
| Neúspěch: | 403, 404, 422 | Popis chyby |

#### 3.5.5.21 Odstranění dokumentu

Koncový bod umožňující trvalé odstranění dokumentu identifikovaného pomocí parametru documentId.

Pro jeho použití musí být aktuálně přihlášený uživatel vlastníkem dokumentu.

Tabulka 3.21: Koncový bod Odstranění dokumentu

| URL: | /api/document/:documentId | |
|---|---|---|
| HTTP metoda: | DELETE | |
| URL parametry: | documentId | [String] |
| Data parametry: | žádné | |
| Úspěch: | 204 | |
| Neúspěch: | 403, 404 | Popis chyby |

#### 3.5.5.22 Překlady webového rozhraní

Koncový bod umožňující stažení textových překladů pro webové rozhraní.

Jazyk vrácených textů je určen pomocí parametru lang (například hodnota cs vrátí českou jazykovou mutaci textů).

Tabulka 3.22: Koncový bod Překlady webového rozhraní

| URL: | /locales/:lang/translation.json | |
|---|---|---|
| HTTP metoda: | GET | |
| URL parametry: | lang | [String] |
| Data parametry: | žádné | |
| Úspěch: | 200 | JSON s textovými překlady |
| Neúspěch: | 404 | Popis chyby |

## 3.6 Komunikace ve skutečném čase

K implementaci komunikace ve skutečném čase existuje více způsoby, ale každý má své kompromisy (více o push technologiích v sekci 2.6.2). Pro komunikaci jsem chtěl použít technologii WebSocket (viz sekce 2.6.2.4), kvůli podpoře full duplexní oboustranné komunikace, ale zároveň jsem chtěl umožnit použít aplikace uživatelům se starším webovým prohlížečem, či mobilním zařízením.

Z tohoto důvodu jsem se rozhodl pro použití knihovny Socket.io, která je je postavena na transportní knihovně Engine.io. Knihovna Socket.io poskytuje ucelené API pro použití technologie WebSocket, pro všechny modelní prohlížeče, ale i pro prohlížeče, které technologii WebSocket dosud nepodporují

a to díky transparentnímu použití záložní komunikace pomocí long pollingu (viz sekce 2.6.2.1).

### 3.6.1 Druhy zpráv

Komunikaci ve skutečném čase využívá komponenta editoru pro synchronizaci textových operací, ale také pro zobrazení nových zpráv v komunikační vlákně dokumentu. Dílčí části komunikace se nazývají zprávy. Každá zpráva má své jméno, podle kterého jsou od sebe zprávy rozeznávány. Zpráva může mít libovolný počet parametrů a nepovinnou funkci zpětného volání.

V této sekci následuje kompletní výpis druhů zpráv zasílaných mezi klientem a serverem.

#### 3.6.1.1 Připojení k dokumentu

Zpráva Připojení k dokumentu je první zprávou, co klient serveru pošle. Server si klienta poznamená, zkontroluje jeho oprávnění, obeznámí ostatní klienty jeho připojením a pomocí zpětného volání vrátí klientu informace o dokumentu, ke kterému se právě připojil.

Tato zpráva existuje ve dvou variantách, jako zpráva pro prvotní připojení k dokumentu a jako zpráva pro opakované připojení po obnovení přerušeného spojení mezi klientem a serverem.

#### 3.6.1.2 Změň jméno

Zprávu Změň jméno odešle server klientům, aby je informoval o nově připojeném klientu. Součástí zprávy je identifikátor nově připojeného klienta a jeho jméno.

#### 3.6.1.3 Klient se odpojil

Zprávu Klient se odpojil odešle server zbylým klientům po odpojení jiného klienta. Zbylí klienti si odpojeného klienta, kterého identifikují pomocí identifikátoru v parametru zprávy, odeberou ze svého seznamu klientů.

#### 3.6.1.4 Operace

Zpráva Operace slouží pro propagaci operací mezi připojenými klienty. Součástí zprávy Operace je číslo další očekávané verze (mezi autorem a serverem), aby server mohl operaci transformovat vůči všem kolizním operacím (více o algoritmu v sekci 2.7.3).

Autor změny odešle zprávu Operace na server, který ji transformovanou uloží a následně odešle všem ostatním klientům, kteří jsou připojeni ke stejnému dokumentu. Nakonec Operace server odešle autorovi zprávu Potvrzení.

#### 3.6.1.5 Potvrzení

Zprávu Potvrzení odešle server klientovi jako odpověď na nově zaslanou zprávu Operace po té co zprávu zpracuje. Klient do doby přijetí zprávy Potvrzení nesmí odeslat na server další zprávu Operace (viz sekce 3.7.1).

#### 3.6.1.6 Vybrání

Zpráva Vybrání slouží pro propagaci změny kurzoru (či vybrání části textu) mezi klienty. Server tuto zprávu nijak nezpracovává a pouze ji propaguje ostatním klientům.

Tato zpráva je použita pouze pro případ, že se změnila pozice kurzoru bez změny textu (změna pozice kurzoru při změně textu je obsažena již ev zprávě Operace).

#### 3.6.1.7 Nastavení

Zpráva Nastavení slouží pro synchronizaci změn nastavení mezi aktuálně připojenými klienty. Klient zprávu odešle spolu se změněným nastavením dokumentu, server nastavení pro daný dokument uloží a následovně jej odešle ostatním připojeným klientům.

#### 3.6.1.8 Zpráva

Tento druh zpráv slouží pro propagaci nových zpráv komunikačního vlákna dokumentu. Klient odešle zprávu s textem na server, ten ji uloží a odešle ostatním připojeným klientům.

Tento způsob propagace je kombinován s REST koncovými body (viz sekce 3.5), které umožňují klientům získat například historii komunikačního vlákna.

#### 3.6.1.9 Chyba

Zpráva Chyba předchází násilnému odpojení klienta od dokumentu. Její příčinou může být například nedostatečné oprávnění uživatele, či nenalezení příslušného dokumentu.

Součástí zprávy Chyba je stavový kód chyby, který může nabývat hodnot 404 nebo 403. Hodnota 404 značí, že nebyl nalezen požadovaný dokument nebo k němu neměl uživatel oprávnění. A hodnota 403 značí, že uživatel provedl operaci, ke které neměl dostatečné oprávnění.

## 3.7 Komponenta editoru

Komponenta editoru je hlavní částí aplikace, která uživatelům umožňuje společně upravovat dokumenty ve skutečném čase a to díky použití algoritmu OT

(více o volbě algoritmu v sekci 3.3). Komponenta tento algoritmus implementuje a poskytuje rozhraní pro jeho použití bez omezení na použité technologii komunikace, či samotné knihovny textového editoru.

Komponentu editoru lze rozdělit na dvě části, část klientská a část serverová, podle částí aplikace kde běží (viz sekce 3.1 o architektuře aplikace).

### 3.7.1 Klientská část komponenty

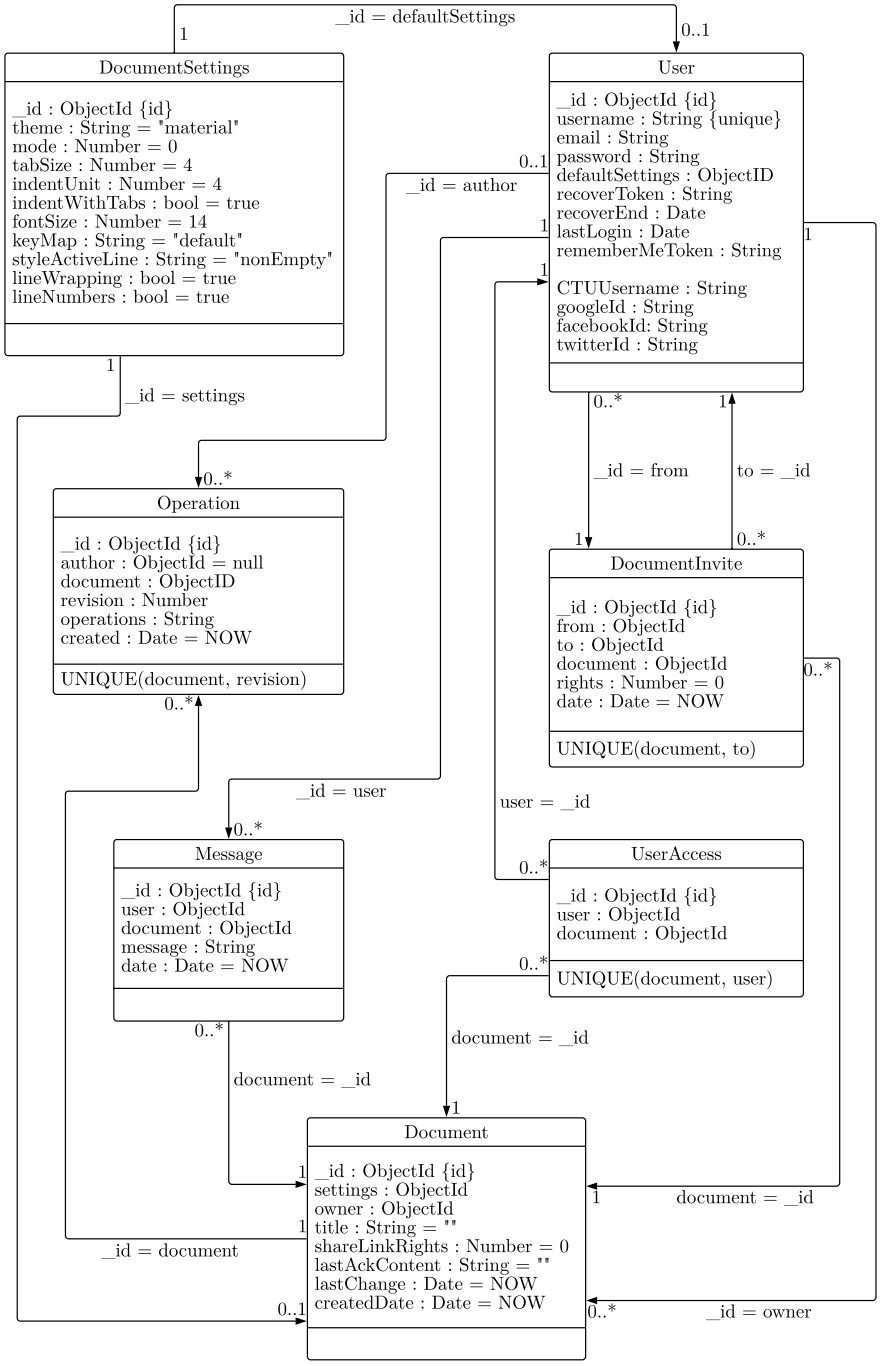
Klientská část komponenty musí komunikovat se zvoleným textovým editorem, zachytávat uživatelský vstup a následně ho převést na abstraktní object operace, který lze dále použít v rámci algoritmu OT. Tato část také musí umět komunikovat pomocí zvolené komunikační technologie s částí serverovou (propagace jednotlivých Operací mezi uživateli).

Struktura klientské části je znázorněna pomocí třídního diagramu na obrázku 3.6. Jádrem celé části je třída `EditorClient`, která dědí od třídy `Client` z knihovny OT.js. Dědí vlastnosti a metody implementující jádro algoritmu OT, jako je například udržování čísla revize a transformace přijatých operací v případě existence nepotvrzené vlastní operace. Třída `Client` a tedy i třída `EditorClient` je navržena podle návrhového vzoru stav, který je popsán v [44, str. 283], a může nabývat 3 stavů (viz stavový diagram na obrázku 3.7).

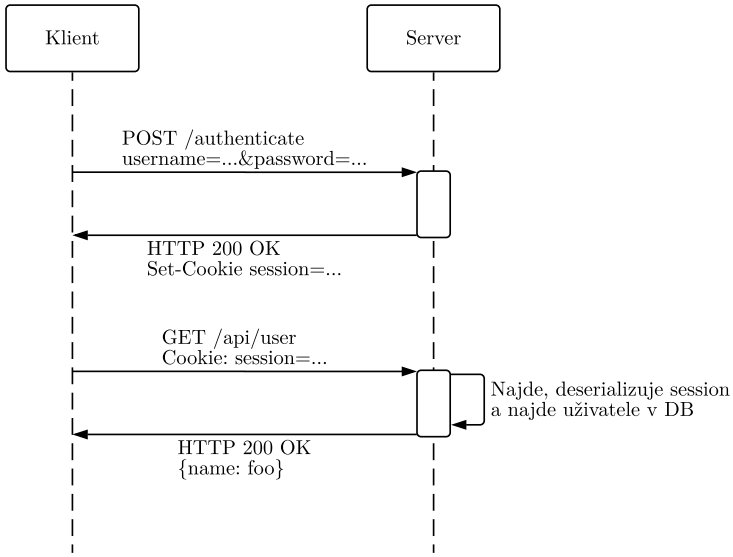
Třída `EditorClient` očekává implementaci třídy `AbstractEditorAdapter` (respektive `AbstractServerAdapter`), která slouží jako rozhraní pro komunikaci s textovým editorem (respektive serverovou částí). Tyto třídy jsou navrženy podle návrhového vzoru adaptér, tak jak je popsán v [44, str. 135], a umožňují spolupráci rozdílných rozhraních (například rozhraní textového editoru) a třídy `EditorClient`. Použití abstraktních tříd umožňuje změnu jednotlivých částí aplikace (změna komunikační technologie, či knihovna textového editoru) a to bez nutnosti zásahu do logiky pro synchronizaci samotných textů.

Tyto abstraktní třídy jsou také potomky třídy `EventEmitter`, která je ustálenou a rozšířenou implementací návrhového vzoru Pozorovatel (anglicky Observer) [44, str. 273] pro jazyk Javascript. Rozhraní `EventEmitter` umožňuje třídě `EditorClient` naslouchat jednotlivým událostem, ke kterým dochází v implementacích zmíněných abstraktních tříd (jako je například změna pozice kursoru, či přijatá operace od serveru). Seznam jednotlivých událostí je možné pozorovat na diagramu 3.8 (respektive 3.9).

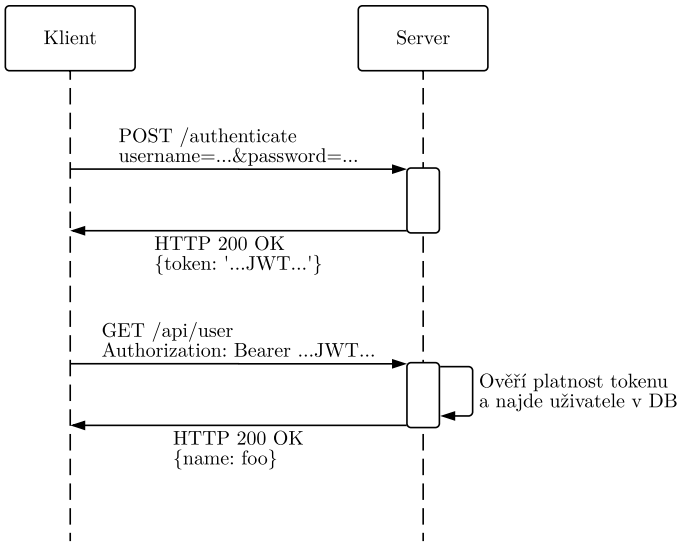
`EditorClient` také využívá třídu `UndoManager` z knihovny OT.js, díky které lze bezpečně použít funkce zpět a vykonat znovu (pokud jejich odchycení podporuje poskytnutá implementace třídy `AbstractEditorAdapter`). Historie je zaznamenávána pomocí jednotlivých operací vykonaných uživatelem a nikoly podle změn samotného textu, jak by tomu bylo běžně. Na změnách textu se totiž může podílet více uživatelů najednou a je nežádoucí, aby například funkce zpět vracela změny provedené jiným než lokálním uživatelem.



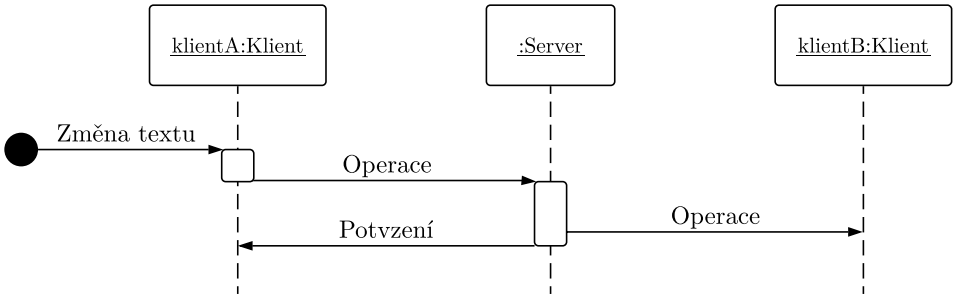

Obrázek 3.2: Databázové schéma v rámci ODM Mongoose



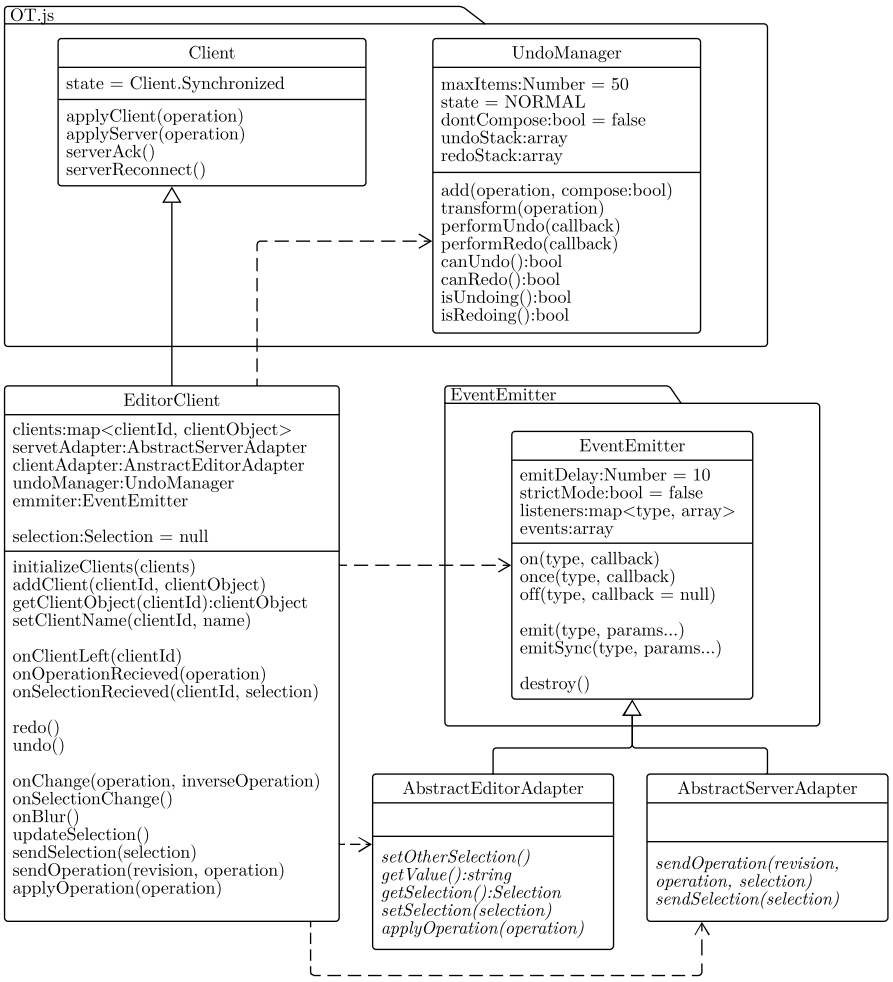

Obrázek 3.3: Sekvenční diagram stavové autentizace



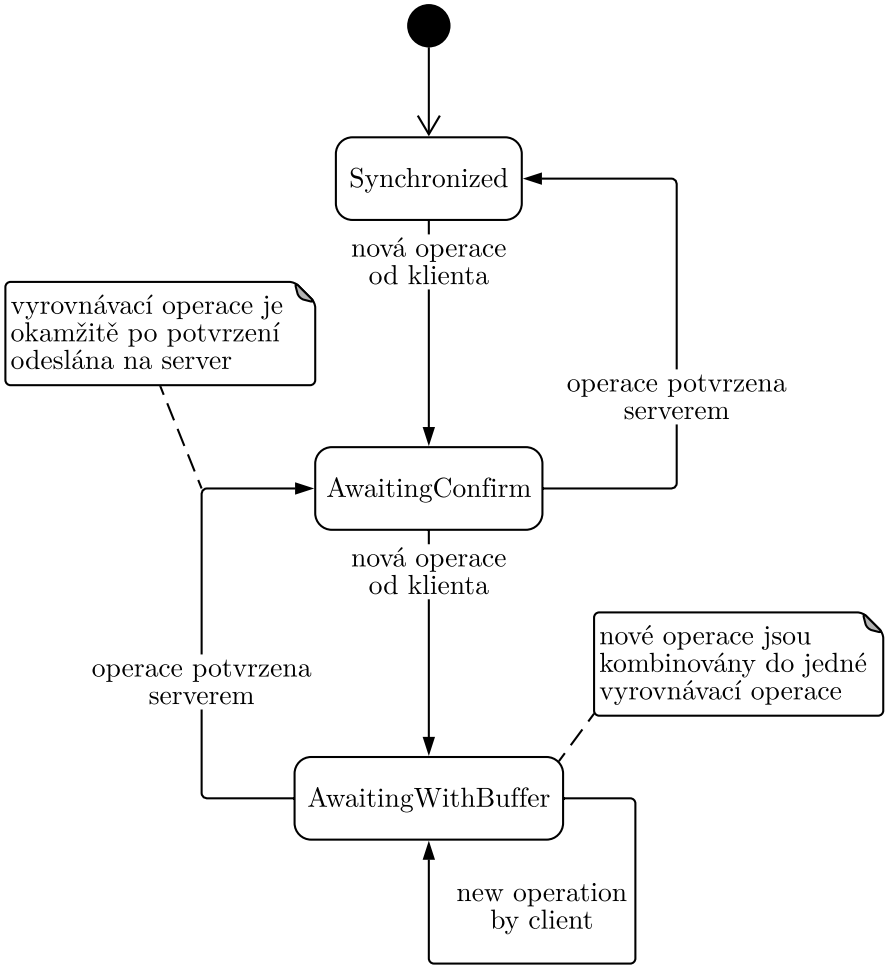

Obrázek 3.4: Sekvenční diagram bezstavové autentizace



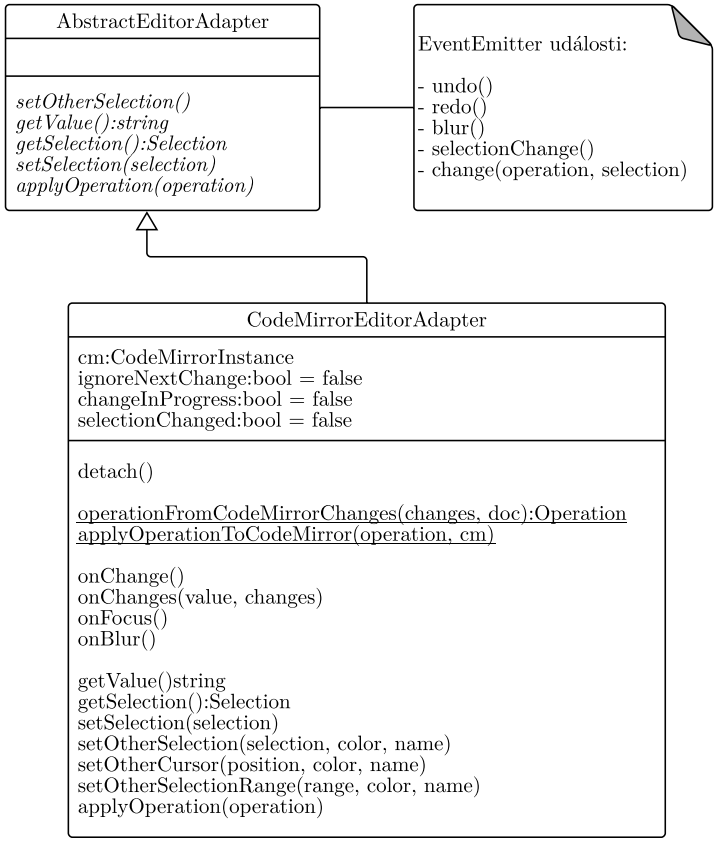

Obrázek 3.5: Sekvenční diagram ukázka komunikace po změně dokumentu



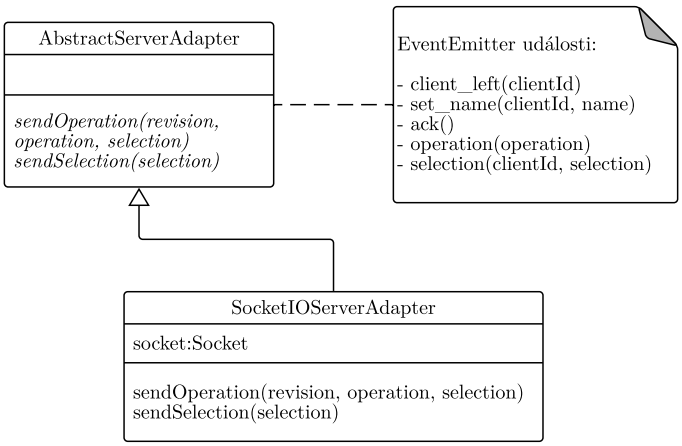

Obrázek 3.6: Třídní diagram klientské části komponenty




Obrázek 3.7: Stavový diagram klientské části komponenty




Obrázek 3.8: Diagram implementace třídy AbstractEditorAdapter




Obrázek 3.9: Diagram implementace třídy AbstractServerAdapter

#### 3.7.1.1 React rozhraní komponenty

Klientská část komponenty v prototypu je zapouzdřena do React komponenty (více o knihovně React v sekci 2.5.4), kterou jsem pojmenoval `RealtimeEditor`. React komponenta umožňuje klientskou část znovu použití kdekoliv v aplikaci a pomocí modulárního návrhu lze upravit i její vzhled.

React komponenta po svém prvním vykreslení vytvoří spojení k serverové části a vytvoří instanci textového editoru CodeMirror, které předá nově vytvořeným instancím tříd `SocketIOServerAdapter` a `CodeMirrorEditorAdapter`. Komponenta dále vytvoří instanci třídy `EditorClient`, které předá vytvořené instance tříd `SocketIOServerAdapter` a `CodeMirrorEditorAdapter`. Komponenta ve svém stavu drží infromace o připojených klientech, nastavení a další informace ohledně aktuálně připojeného dokumentu.

React komponenta `RealtimeEditor` přijímá jak povinné, tak i nepovinné vlastnosti (viz tabulka 3.23).

Tabulka 3.23: Vlastnosti přijímané React komponentou RealtimeEditor

| Jméno: | Povinná: | Datový typ: | Popis: |
|---|---|---|---|
| documentId | ano | String | Identifikátor dokumentu |
| user | ne | Object | Informace o přihlášeném uživateli |
| headerSlot | ne | Node | Komponenta, která se vykreslí na místě hlavičky dokumentu |
| menuSlot | ne | Node | Komponenta, která se vykreslí na místě menu dokumentu |
| errorSlot | ne | Node | Komponenta, která se vykreslí pokud dojde k chybě |

Vlastnost `documentId` označuje identifikátor dokumentu, ke kterému se komponenta má připojit. Vlastnost `user` je objekt s informacemi o přihlášeném uživateli (očekávaná struktura je stejná jako struktura, kterou vrací REST koncový bod 3.5.5.7). Pokud není uživatel přihlášen není nutné vlastnost předávat, či stačí předat hodnotu, která se vyhodnotí jako nepravdivá (`false`, `null` a další). Poslední vlastnosti `headerSlot`, `menuSlot` a `errorSlot` označují modulární React komponenty, které jsem vykresleny na specifickém místě `RealtimeEditor` komponenty. Modulárním komponentám jsou předány vnitřní informace o stavu editoru pomocí jejich vlastností.

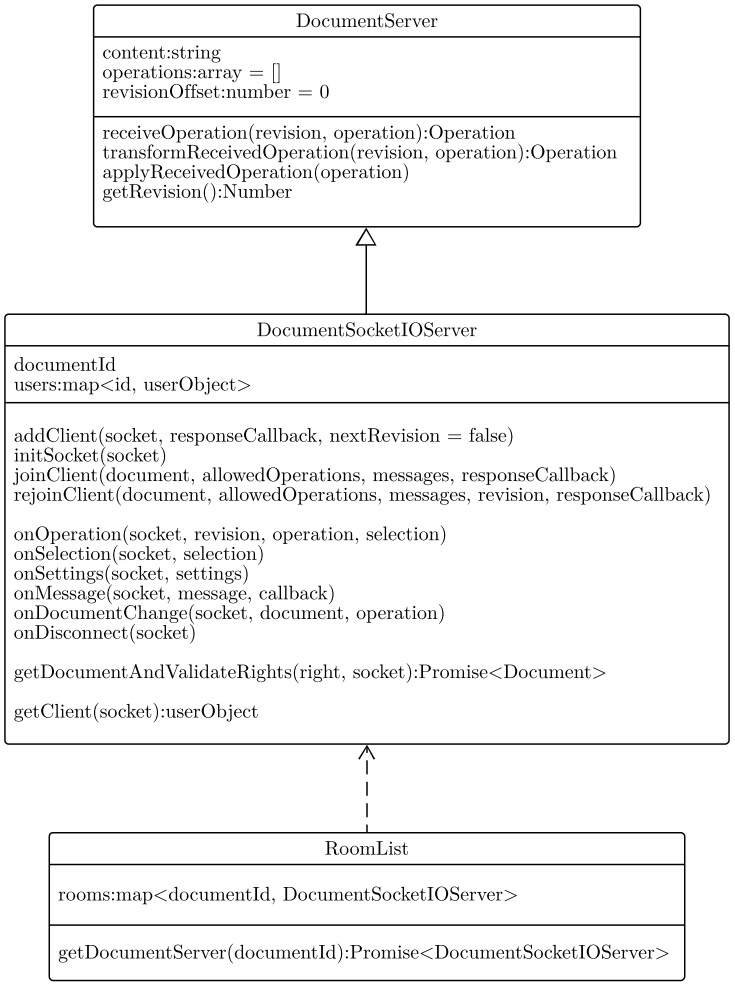
Nejjednodušší použití komponenty je ukázané ve výpisu kódu 1. Tento příklad vykreslí pouze samotný textový editor, který je ovšem aktualizovaný v reálném čase ve všemi uživateli, kteří jsou ve stejnou chvíli připojeni k dokumentu s identifikátorem „helloWorld“.

```
export default () => (
    <RealtimeEditor documentId="helloWorld"/>
)
```

Výpis kódu 1: Příklad použití komponenty RealtimeEditor

### 3.7.2 Serverová část komponenty

Serverová část je odpovědná za propagaci jednotlivých operací mezi klienty připojenými k dokumentu a jejich persistenci pro nově připojené budoucí klienty. Struktura serverové části je znázorněna pomocí třídním diagramu na obrázku 3.10. Základem této části je třída `DocumentServer` (reimplementace třídy Server z knihovny OT.js) a v případě navrhovaného prototypu aplikace i její potomek `DocumentSocketIOServer`.




Obrázek 3.10: Třídní diagram serverové části komponenty

Instance třídy `DocumentServer` (respektive `DocumentSocketIOServer`) představuje jeden document a je zodpovědná o implementaci serverové části algoritmu OT. Stará se o transformaci přijaté operace oproti souběžným, ale již schváleným operacím, a propaguje tuto transformovanou operaci ostatním

uživatelům.

Další třídou, kterou je vhodné zmínit je třída `RoomList`, která udržuje informace o již existujících instancích třídy `DocumentSocketIOServer`, vytváří nové instance v případě, že pro daný dokument dosud neexistuje, a naslouchá novým socket.io spojení s klienty, které dále předává příslušným instancím třídy `DocumentServer`.

Kapitola 4

# Realizace

## 4.1 Autentizace

Autentizace označuje proces ověření identity uživatele. Pro navržený prototyp aplikace se jedná o proces přihlášení uživatele.

K implementaci autentizace jsem využil knihovnu passport.js, která ulehčuje implementaci autentizace pro knihovnu express.js a prostření Node.js (více o prostředí v sekci 2.5.3). Passport.js umožňuje použití autentizačních tříd, které implementují jednotlivé způsoby webové autentizace. Díky modulárnímu návrhu je možné používat různé způsoby autentizace, bez nutnosti změn zbytku aplikace.

Každá autentizační třída přijímá konfigurační objekt a funkci, která volána pro ověření identity. Předaná funkce vyhledá uživatele v DB podle identifikátoru, který získá od autentizační třídy, a vrátí objekt uživatele, který má být přihlášen, nebo chybu.

Instanci knihovny passport.js musí být předány funkce pro serializaci a deserializaci uživatele. V prototypu aplikace jsou uživatelé deserializováni pouze na jejich DB identifikátor a serializováni pomocí jejich opětovného vyhledání v DB.

### 4.1.1 Přihlášení pomocí hesla

Základním způsobem autentizace v navrženém prototypu je přihlášení pomocí uživatelského jména a hesla. Knihovna passport.js pro tento způsob poskytuje připravenou třídu `LocalStrategy`, která požaduje pouze jména parametrů obsahující jméno a heslo přihlašovaného uživatele.

Autorizační funkci jsou předány hodnoty parametrů a uživatel je podle uživatelského jména vyhledán v DB. Následně dojde k validaci přihlašovacího hesla (viz následující sekce 4.1.1.1) a v případě úspěchu je uživatel přihlášen.

#### 4.1.1.1 Uložení hesla

Pro možnost validace přihlašovacího hesla je potřeba nějakým způsobem heslo zaznamenat. Z bezpečnostních důvodů není vhodné v DB uchovávat hesla v čitelné podobě, ani v podobě z které by bylo možné heslo jednoduše získat.

Při implementaci autentizace pomocí hesla jsem se rozhodl pro použití kryptografické hašovací funkce, která poskytuje jednosměrnou transformaci vstupu na téměř náhodný výstup. Kryptografické hašovací funkce se od běžných hašovacích funkcí nezaměřují na rychlost, ale na kryptografické vlastnosti funkce.

Jako hašovací funkci jsou zvolil `bcrypt`, která se řadí mezi pomalé kryptografické hašovací funkce. Tato funkce (na rozdíl od rychlých funkcí standartu SHA-2) má nastavitelný počet iterací, což umožňuje navyšovat její obtížnost vypočtu s rostoucím výkonem počítačů. Vyšší obtížnost výpočtu je vhodná pro hašování hesel z důvodu zpomalení útoku typu brutal force (opakované a náhodné pokusy o uhodnutí hesla).

#### 4.1.1.2 Odhad obtížnosti hesla

Při změně přihlašovacího hesla jsou dnes běžné minimální požadavky na složitost hesla. Tyto požadavky jsou však podle [45] však většinou nedostatečné a neodpovídají skutečné době potřebné pro jejich uhodnutí. Například heslo `P@ssw0rd` oproti `Password` nepřináší téměř žádnou přidanou obtížnost, jelikož bylo odvozenou použitím pouze známých a častých substitucí.

Proto jsem se rozhodl do prototypu přidat odhad obtížnosti hesla, který využívá knihovnu zxcv od společnosti Dropbox. Knihovna umožňuje nastavit minimální časovou hodnotu, za kterou je odhadované prolomení hesla, a podle skóre testovaného hesla rozhodnou, zda-li je dostatečně obtížné. V prototypu aplikace je tato hodnota nastavena, tak aby bylo zabráněno jen použití opravdu jednoduchých hesel. Navíc je také odhadovaná doba prolomení zobrazena přímo u pole pro zadání nového hesla, tak aby si mohl uživatel sám zvolit jaká hodnota je pro něj přijatelná.

### 4.1.2 Ostatní možnosti přihlášení

Prototyp aplikace je díky použití knihovny passport.js připraven na implementaci dalších způsobů autentizace. Dalším způsobem autentizace je například napojení na služby třetích stran jako jsou sociální sítě a podobně. Uživateli pak stačí přihlášení do služby třetí strany a je bez nutnosti zadávání hesla přihlášen i v prototypu aplikace.

Většina služeb třetích stran používá jeden, či více způsobů autentizace. Mezi nejčastěji používané patří způsoby patří autentizace pomocí protokolů OAuth, OAuth2 a OpenID.

Jednotlivé implementace autorizačních tříd vrací unikátní identifikátor uživatele v rámci dané služby, podle kterého lze v aplikace daného uživatele

nalézt a přihlásit. Účty třetích stran je v prototypu aplikace možné spravovat v nastavení uživatelského účtu pod záložkou `Připojené účty`.

#### 4.1.2.1 ČVUT heslo

Jako příklad použití nepříliš rozšířeného způsobu autentizace jsem se rozhodl o implementaci autentizace pomocí školního České vysoké učení technické (ČVUT) hesla. Portál (https://auth.fit.cvut.cz/manager) umožňuje autentizaci pomocí protokolu OAuth2. Pomocí získaného OAuth tokenu umožňuje přistupovat k ostatním školním API rozhraním (jako je například KOS, Usermap a tak další).

Implementovaná autentizační třída `CTUOAuth2Strategy` je potomkem třídy `Strategy` z balíčku `passport-oauth2`, která implementuje základní OAuth2 autentizaci. Ke kompletní implementaci ČVUT autentizace jsem použil identifikaci uživatelů pomocí jejich ČVUT jména, které jsem získal pomocí koncového bodu `https://auth.fit.cvut.cz/oauth/userinfo`. Tento koncový bod vrací v odpovědi informace o předaném OAuth tokenu a jeho majiteli (tedy včetně jeho ČVUT uživatelského jména).

#### 4.1.2.2 Sociální sítě

Jako běžný příklad způsobu autentizace pomocí služeb třetí strany jsem zvolil autentizaci pomocí sociálních sítí. V prototypu aplikace je implementována autentizace pomocí účtů služeb Google, Facebook a Twitter.

Pro implementaci jsem použil připravené autorizační třídy pro jednotlivé sociální sítě. Tyto třídy mají již obsahují komunikaci s koncovými body služby a vrací informace o přihlášeném uživateli. Získané informace se liší napříč službami a je potřeba je vždy vyhledat v příslušné dokumentaci API dané služby.

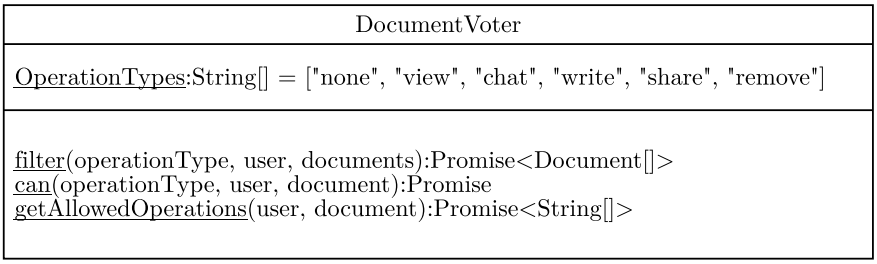
Problémem, na který jsem při implementaci narazil, je nepovinný údaj emailu uživatele u služby Facebook, kde pole email přihlášeného uživatele služba vrací v závislosti na nastavení jeho soukromí i přes nastavení oprávnění mé aplikace. A protože je údaj email v mém schématu pro uživatele povinný (viz sekce 3.2), musel jsem pro službu facebook email generovat z poskytnutých údajů o uživateli (výsledný vygenerovaný email pak může vypadat například jako `1784258414938423@facebook.com`).

## 4.2 Autorizace

Protože v navrženém prototypu aplikace nejsou rozlišovány žádné pokročilejší uživatelské role, je naprosto dostačující jednoduchá kontrola přihlášení uživatele u jednotlivých přístupových bodů. Tuto kontrolu provádí `Middleware` moduly, které validují každý příchozí požadavek od klienta. Pokud se uživatel pokouší přistoupit ke koncovému bodu, který vyžaduje přihlášeného (resp.

nepřihlášeného) uživatele, modul `AuthMiddleware` zkontroluje zda-li je uživatel přihlášen (respȯdhlášen) a pak teprve serverová část aplikace pokračuje ve zpracování požadavku.

Složitější situace nastává v omezení přístupu uživatelů k jednotlivým dokumentům. Uživatel může být k dokumentu přizván a jeho práva, tak nemusí odpovídat globálnímu nastavení práv pro sdílení dokumentu (sdílení pomocí veřejného odkazu). Z tohoto důvodu jsem implementoval statickou třídu `DocumentVoter` (viz třídní diagram na obrázku 4.1). Metody této třídy přijímají objekt uživatele (nebo hodnotu `null`, pokud uživatel není přihlášený) a objekt dokumentu, pro který chceme zjistit uživatelovo oprávnění. Například metoda `DocumentVoter.can()` vrací Promise, která skončí úspěchem pouze v případě, kdy má předaný uživatel dostatečné oprávnění k provedení zvolené akce nad předaným dokumentem.




Obrázek 4.1: Třídní diagram třídy DocumentVoter

Složitým problémem autorizace je omezení přístupu k samotnému editoru dokumentů a to nejen v případě, kdy uživatel nemá k dokumentu oprávnění, ale i v případě, že dokument vůbec neexistuje. Klientská část aplikace nemá přístup k DB a nemůže tedy zjistit, zda-li se má editor vůbec aktuálnímu uživateli zobrazit, či která část má zůstat skryta, protože k ní uživatel nemá dostatečné oprávnění. Proto klientská část pro každý dokument zahájí spojení se serverem, který po přijetí zprávy Připojení k dokumentu (viz sekce 3.6 o komunikaci) rozhodne zda-li uživatele připojí (a vrátí seznam operací, ke kterým má uživatel dostatečné oprávnění) nebo odešle zprávu Chyba a spojení ukončí.

Z tohoto důvodu se může stát, že aplikace před zobrazení chyby o nenalezení dokumentu na zlomek vteřiny zobrazí editor dokumentu. Editor je v uzamčeném stavu a neumožňuje uživateli editaci, ani žádnou jinou operaci nad dokumentem, ale jeho zobrazení je nutné pro správné vytvoření instance textového editoru.

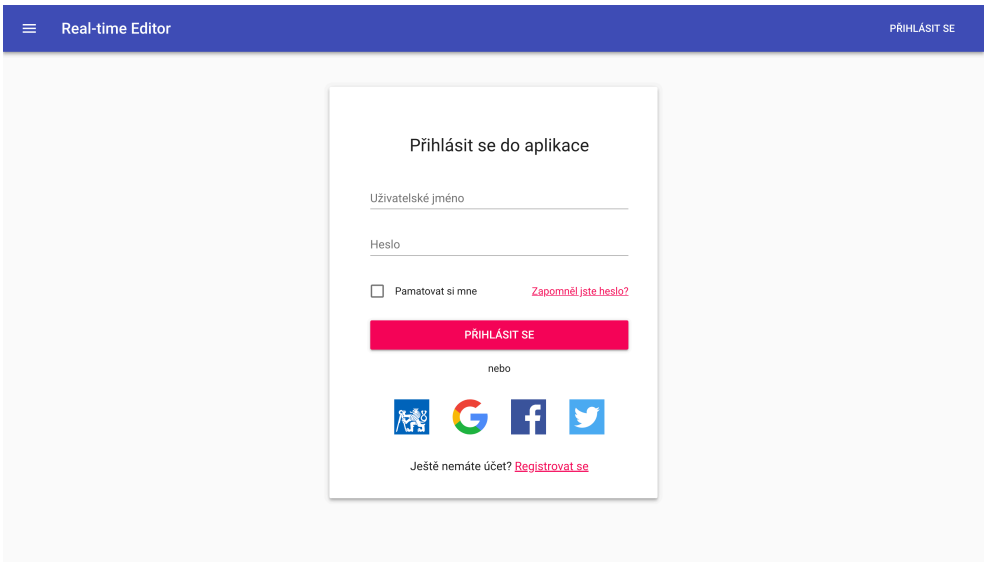
## 4.3 Uživatelské rozhraní

Uživatelské rozhraní a jeho návrh se řadí do odvětví Interakce člověka a počítače (anglicky Human-computer interaction nebo HCI). Jedná se složitou problematiku, která zahrnuje přístupnost, použitelnost a důvěryhodnost uživatelského rozhraní a to nejen pro webové aplikace.

### 4.3.1 Material Design

Pro prototyp aplikace jsem se rozhodl využít již připravené zásady pro tvorbu uživatelského rozhraní od společnosti Google nazvané Material Design. Tyto zásady jsou veřejně dostupné na stránkách společnosti Google a řeší výše zmíněnou problematiku návrhu uživatelského rozhraní. [46]

Jako implementaci těchto zásad jsem využil knihovnu `Material-UI`. Tato knihovna implementuje jednotlivé komponenty zmíněné v zásadách Material design jako React komponenty (více o knihovně React v sekci 2.5.4). Použitím této knihovny vzniklo vizuálně příjemné a intuitivní uživatelské rozhraní (viz náhled rozhraní na obrázku 4.2).

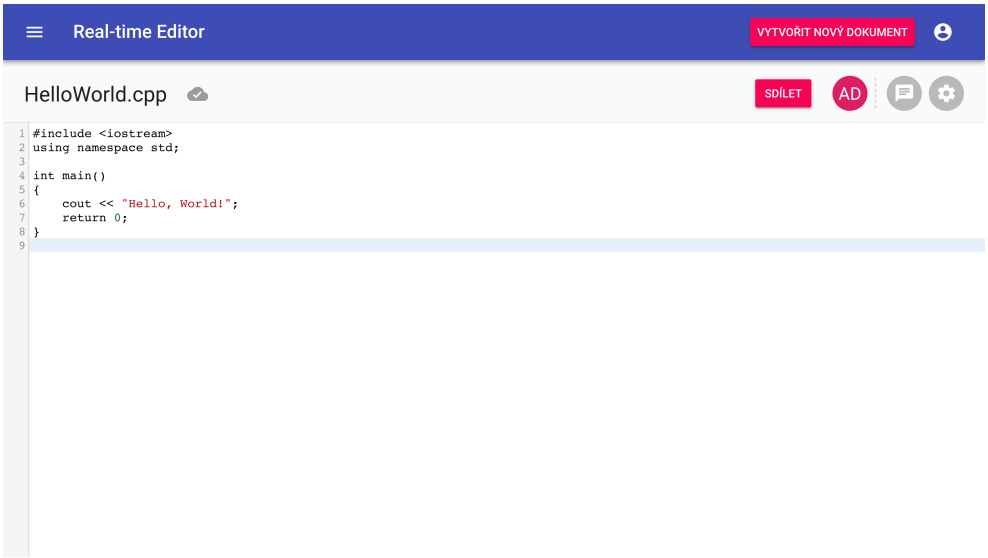



Obrázek 4.2: Náhled přihlašovací obrazovky prototypu aplikace

### 4.3.2 Textový editor

Textový editor je hlavní částí prototypu aplikace. Z tohoto důvodu není vhodné použít jednoduchou textovou oblast a rozhodl jsem se proto použít komplexnější textový editor.

Pro implementaci textového editoru jsem použil JavaScriptovou (více o jazyce v sekci 2.5.2) knihovnu `CodeMirror`. Tato knihovna poskytuje pomocí svého API rozsáhlé možnosti pro manipulaci textu, vzhledu, zvýrazňování syntaxe a celkového chování editoru [47]. Jejím použití vznikl textový editor s rozsáhlými možnostmi nastavení a příjemným vzhledem (viz náhled editoru na obrázku 4.3).




Obrázek 4.3: Náhled editoru prototypu aplikace

## 4.4 Jazykové překlady

Pro implementaci jazykových překladů jsem použil rozšířenou překladový framework `i8next` spolu s modulem `react-i18next`, který obsahuje rozhraní pro React komponenty (více o knihovně v sekci 2.5.4). Tato knihovna umožňuje definovat jazykové překlady textů včetně množných čísel, parametrů a dalšího formátování [48].

K načtení samotných souborů jazykových překladů jsem využil možností modulu `i18next-xhr-backend`, který umožňuje asynchronní načtení překladů, tak aby uživatelé nemuseli stahovat více překladů, než jen pro aktuálně zvolený jazyk. Modul je nastaven, tak aby načítal překladové texty pomocí koncového bodu Překlady webového rozhraní (viz popis koncového bodu v sekci 3.5.5.22).

## 4.5 Sestavení aplikace

Sestavení aplikace označuje postup kompilace, minimalizace a další úpravy zdrojů před jejich vystavením do produkčního prostředí. V implementovaném prototypu aplikace je na sebe klientská a serverová část aplikace vázaná právě po sestavení aplikace a jejím vystavení do produkční prostředí (více v následujících podsekcích).

### 4.5.1 Sestavení klientské části

Klientská část vyžaduje sestavení před každým spuštěním v uživatelském prohlížeči, protože využívá syntaktických prvků jazyka JavaScript, které nejsou prohlížeči podporovány. Pro sestavení jsem využil již připravené konfigurace pro nástroj `Webpack`, která přichází s použitím balíčku `create-react-app` přímo od společnosti Facebook. Balíček `create-react-app` obsahuje mimo nastavení pro produkční sestavení aplikace, i nastavení pro vývoj aplikace a to včetně automatického znovu načtení stránky v prohlížeči po provedení změn ve zdrojovém kódu.

Sestavení klientské části lze spustit pomocí příkazu `yarn build` a sestavená klientská část aplikace je následně umístěně ve složce `client/build-prod`. Takto sestavenou klientskou část aplikace je možné jednoduše vystavil, protože se jedná o statické webové stránky.

### 4.5.2 Serverová část a vystavení klientské části

Serverová část aplikace je připravena na běh v Node.js prostředí (více o prostředí Node.js v sekci 2.5.3), ve kterém byla tato část i vyvíjena. Proto není u serverové části postup sestavení potřebný.

Jediná změna mize produkčním a vývojovým vystavení serverové části je nastavení běhového prostředí Node.js pomocí proměnné prostředí `APP_ENV`, který musí být nastaven na hodnotu `production` (resp. `development` pro vývoj). Tato proměnná zajišťuje správné nastavení session, cookies a protokolování v závislosti na typu vystavení serverové části.

Serverová část aplikace také umožňuje vystavení předem sestavené klientské části. Pokud serverová část aplikace zpracovává požadavek, který nemá předponu spojenou s REST API a ani neodpovídá cestě žádného souboru sestavené klientské části, vrátí serverová část obsah hlavního HTML souboru sestavené klientské části. Klientská část aplikace se následně po spuštění v prohlížeči uživatele rozhodne, zda daná adresa existuje, či nikoliv.

#### 4.5.2.1 Generování HTML metadat

Klientská část aplikace generuje a aktualizuje určité HTML metadata, tak aby jejich obsah odpovídal aktuálnímu obsahu aplikace. Například po otevření

seznamu dokumentů uživatel očekává, že se obsah tagu `title` změní na odpovídající text, stejně jakoby se jednalo běžnou webovou stránku. K dosažení tohoto chování klientské části aplikace jsem použil knihovnu `react-helmet`, která umožňuje aktualizovat metadata podle pořadí vykreslení jednotlivých React komponent.

Tento přístup je dostatečný, dokud aplikaci používá obyčejný uživatel, který nepostřehne prvotní aktualizaci metadat po spuštění aplikace. Bohužel ale není dostatečný v případě, kdy k aplikaci přistupují roboti sociálních sítí a webových vyhledávačů, kteří při procházení a indexování internetu nespouštějí JavaScriptový kód. Pro ně se metadata vůbec nenastaví a při sdílení veřejného odkazu dokumentu na sociálních sítích se objeví pouze jejich původní obsah, který se sdíleným odkazem nemusí souviset.

Já jsem ale chtěl, aby prototyp aplikace při sdílení veřejného odkazu dokumentu zobrazil na sociálních sítích minimálně název tohoto dokumentu. Toho jsem dosáhl pomocí generování metadat na straně serveru, které jsou následně vloženy do hlavního HTML souboru sestavené klientské části a odeslány klientovi. Robot, který takový odkaz navštíví, získá HTML soubor s vygenerovanými metadaty a to bez nutnosti spustit jakýkoliv JavaScript kód klientské části aplikace.

### 4.5.3 Sestavení pomocí Dockeru

Celý proces sestavení prototypu aplikace jsem zapouzdřil použitím vizualizačního nástroje Docker, který umožňuje definovat aplikace jako takzvané Docker obrazy. Tyto obrazy následně běží v takzvaných Docker kontejnerech, které jsou navzájem nezávislé a je možné je jednoduše spravovat (spouštět, duplikovat a tak podobně).

Pro prototyp aplikace jsou připravené dva druhy obrazů, vývojový a produkční. Tyto obrazy obsahují veškeré závislosti klientské i serverové části aplikace. Připravené konfigurace obrazů a kontejnerů aplikace za pomoci nástroje `docker-compose` obsahují mimo samotný prototyp aplikace i DB kontejner s mongoDB (více o DB v sekci 2.5.5).

Vývojový obraz je připraven pro rychlý a jednoduchý vývoj aplikace bez nutnosti instalace jakýchkoliv nástrojů (kromě Dockeru samotného). Tento obraz lze vytvořit pomocí připraveného příkazu `docker-compose up app_dev`, který vytvoří samotný obraz a jeho kontejner, který následně spustí. Kontejner a v něm běžící aplikace naslouchá změnám ve zdrojových souborech aplikace a při jejich uložení aplikaci znovu automaticky sestaví.

Produkční obraz je velmi podobný obrazu vývojovému, liší se především v nastavení běhového prostředí a uzavřenosti kontejneru. Tento obraz lze vytvořit pomocí připraveného příkazu `docker-compose up app_prod`, který vytvoří samotný obraz a jeho kontejner, který následně spustí. Produkční obraz je připraven pro jeho distribuci mezi dalšími stroji a nese s sebou vše potřebné pro běh aplikace (závislosti, nástroje i zdrojový kód). Proto je možné prototyp

aplikace rychle nasadit bez nutnosti instalace prostření a jeho konfigurace na produkčním stroji. Pro aktualizaci aplikace ze změněných zdrojových kódů je potřeba produkční obraz znovu vytvořit (není zde žádné automatické znovu sestavení).

Kapitola 5

# Testování

Zatím se jedná jen o popsanou struktury kapitoly.

### 5.0.1 Uživatelské testování

Průchod 3 uživatelů systémem (scénaře podle uživatelských případů), postřehy z testování a shrnutí (UX není přímým cílem práce, jde především o analýzu problému a návrh jeho řešení).

### 5.0.2 Testování rychlosti

Rychlost synchronizace mezi více uživateli (porovnání rychlosti s přibývajícím počtem uživatelů).

Test prototypu při velkém počtu uživatel (cca 100 připojených uživatel na jeden dokument). Postřehy a shrnutí (možnosti optimalizace a co to znamená pro další škálování).

# Závěr

Zatím se jedná jen o popsanou struktury kapitoly.

Vše z cílů jsem splnil (cíle zopakovat a připomenou způsob splnění). Šlo by to i lépe (něco vyberu jako příklad).

Komponenta bude využita v projektu Laplace-IDE, je ji ale nejdříve potřeba zbavit některých závislostí a exportovat jako samostatný balíček do npm (JavaScript package manager) registru, tak aby mohla být použita opravdu kdekoliv a kýmkoliv.

# Bibliografie


1. LEITHEAD, Travis; EICHOLZ, Arron; MOON, Sangwhan; DANILO, Alex; FAULKNER, Steve. *HTML 5.2* [online]. 2017 [cit. 2018-04-04]. Dostupné z: https://www.w3.org/TR/2017/REC-html52-20171214/. W3C Recommendation. W3C.
2. MOZILLA; INDIVIDUAL CONTRIBUTORS. *HTML5* [online]. 2018 [cit. 2018-04-04]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/HTML5.
3. BERNERS-LEE, Tim; CONNOLLY, Daniel. *Hypertext Markup Language (HTML) A Representation of Textual Information and MetaInformation for Retrieval and Interchange* [online]. 1993-06 [cit. 2018-04-04]. Dostupné z: https://www.w3.org/MarkUp/draft-ietf-iiir-html-01.txt. Internet Draft. IIIR Working Group (IETF).
4. RAGGETT, Dave. *HTML 3.2 Reference Specification* [online]. 1997 [cit. 2018-04-04]. Dostupné z: https://www.w3.org/TR/2018/SPSD-html32-20180315/. W3C Recommendation. W3C.
5. BERJON, Robin; NAVARA, Erika Doyle; LEITHEAD, Travis; PFEIFFER, Silvia; HICKSON, Ian; O'CONNOR, Theresa; FAULKNER, Steve. *HTML5* [online]. 2014 [cit. 2018-04-04]. Dostupné z: http://www.w3.org/TR/2014/REC-html5-20141028/. W3C Recommendation. W3C.
6. MOZILLA; INDIVIDUAL CONTRIBUTORS. *A re-introduction to JavaScript (JS tutorial)* [online]. 2018 [cit. 2018-04-04]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript/A_re-introduction_to_JavaScript.
7. *ECMAScript: A general purpose, cross-platform programming language* [online]. 1997 [cit. 2018-04-04]. Dostupné z: http://www.ecma-international.org/publications/files/ECMA-ST-ARCH/ECMA-262,%201st%20edition,%20June%201997.pdf. ECMA Standard. ECMA.

8. MOZILLA; INDIVIDUAL CONTRIBUTORS. *JavaScript language resources* [online]. 2018 [cit. 2018-04-04]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Language_Resources.
9. ORSINI, Lauren. What You Need To Know About Node.js. *Readwrite* [online]. 2013 [cit. 2018-04-04]. Dostupné z: https://readwrite.com/2013/11/07/what-you-need-to-know-about-nodejs/.
10. NODE.JS FOUNDATION. *About Node.js®* [online]. 2018 [cit. 2018-04-04]. Dostupné z: https://nodejs.org/en/about/.
11. NODE.JS FOUNDATION. *ECMAScript 2015 (ES6) and beyond* [online]. 2018 [cit. 2018-04-04]. Dostupné z: https://nodejs.org/en/docs/es6/.
12. FACEBOOK INC. *A JavaScript library for building user interfaces* [online]. 2018 [cit. 2018-04-05]. Dostupné z: https://reactjs.org.
13. FISHER, Bill. *How was the idea to develop React conceived and how many people worked on developing it and implementing it at Facebook?* [online]. 2015 [cit. 2018-04-05]. Dostupné z: https://www.quora.com/How-was-the-idea-to-develop-React-conceived-and-how-many-people-worked-on-developing-it-and-implementing-it-at-Facebook.
14. INDIVIDUAL CONTRIBUTORS. *Read Me* [online]. 2018 [cit. 2018-04-05]. Dostupné z: https://redux.js.org.
15. STACK EXCHANGE INC. *Stack Overflow Developer Survey 2018* [online]. 2018 [cit. 2018-04-05]. Dostupné z: https://insights.stackoverflow.com/survey/2018/#technology-databases.
16. MONGODB, INC. *MongoDB and MySQL Compared* [online]. 2018 [cit. 2018-04-05]. Dostupné z: https://www.mongodb.com/compare/mongodb-mysql.
17. SPACEY, John. Pull vs Push Technology. *Simplicable* [online]. 2017 [cit. 2018-04-06]. Dostupné z: https://simplicable.com/new/pull-vs-push-technology.
18. LORETO, S.; ERICSSON; SAINT-ANDRE, P.; CISCO; SALSANO, S.; UNIVERSITY OF ROME "TOR VERGATA"; WILKINS, G.; WEBTIDE. *Known Issues and Best Practices for the Use of Long Polling and Streaming in Bidirectional HTTP* [online]. 2011 [cit. 2018-04-06]. Dostupné z: https://tools.ietf.org/html/rfc6202. Request for Comments. IETF.
19. HICKSON, Ian (ed.). *Server-Sent Events* [online]. 2015 [cit. 2018-04-06]. Dostupné z: https://www.w3.org/TR/2015/REC-eventsource-20150203/. W3C Recommendation. W3C.

20. SALVET, Pavel. Komunikace v reálném čase díky Server-Sent Events a Web Sockets. *Interval* [online]. 2015 [cit. 2018-04-06]. Dostupné z: https://www.interval.cz/clanky/komunikace-v-realnem-case-diky-server-sent-events-a-web-sockets/.

21. LUBBERS, Peter; GRECO, Frank; KAAZING CORPORATION. *HTML5 WebSocket: A Quantum Leap in Scalability for the Web* [online] [cit. 2018-04-06]. Dostupné z: http://www.websocket.org/quantum.html.

22. LAFORGE, Guillaume. *Algorithms for collaborative editing* [online]. 2012 [cit. 2018-04-06]. Dostupné z: http://glaforge.appspot.com/article/algorithms-for-collaborative-editing.

23. FRASER, Neil. *Differential Synchronization* [online]. 2009 [cit. 2018-04-07]. Dostupné z: https://neil.fraser.name/writing/sync/.

24. FRASER, Neil. *Google Tech Talks - Differential Synchronization* [online]. 2009 [cit. 2018-04-07]. Dostupné z: https://www.youtube.com/watch?v=S2Hp_1jqpY8.

25. FRASER, Neil. *Home - google/diff-match-patch Wiki* [online]. 2013 [cit. 2018-04-07]. Dostupné z: https://github.com/google/diff-match-patch/wiki.

26. DANIELS, Alden. *Collaborative Editing in JavaScript: An Intro to Operational Transformation* [online]. 2015 [cit. 2018-04-08]. Dostupné z: https://davidwalsh.name/collaborative-editing-javascript-intro-operational-transformation.

27. BAUMANN, Tim. *What is Operational Transformation?* [online]. 2013 [cit. 2018-04-08]. Dostupné z: http://operational-transformation.github.io/what-is-ot.html.

28. SPIEWAK, Daniel. Understanding and Applying Operational Transformation. *Code Commit* [online]. 2010 [cit. 2018-04-05]. Dostupné z: http://www.codecommit.com/blog/java/understanding-and-applying-operational-transformation?utm_source=feedburner&utm_medium=feed&utm_campaign=Feed%3A+codecommit+%28Code+Commit%29.

29. AGARWAL, Srijan. Operational Transformation, the real time collaborative editing algorithm. *Hacker Moon* [online]. 2017 [cit. 2018-04-08]. Dostupné z: https://hackernoon.com/operational-transformation-the-real-time-collaborative-editing-algorithm-bf8756683f66.

30. GENTLE, Joseph. *ShareJS – Live concurrent editing in your app homepage* [online]. 2011 [cit. 2018-04-08]. Dostupné z: https://www.championtutor.com:7004.

31. THE APACHE SOFTWARE FOUNDATION. *Wave Project Incubation Status* [online]. 2018 [cit. 2018-04-05]. Dostupné z: http://incubator.apache.org/projects/wave.html.

32. WANG, David; MAH, Alex. *Google Wave Operational Transformation Whitepaper* [online]. 2009 [cit. 2018-04-05]. Dostupné z: https://web.archive.org/web/20100108095720/http://www.waveprotocol.org:80/whitepapers/operational-transform.

33. The Best Word Processing Software. *Top Ten Reviews* [online]. 2018 [cit. 2018-04-05]. Dostupné z: http://www.toptenreviews.com/business/software/best-word-processing-software/.

34. GOOGLE INC. *Google Apps Realtime - Conflict Resolution and Grouping Changes* [online]. 2016 [cit. 2018-04-05]. Dostupné z: https://developers.google.com/google-apps/realtime/conflict-resolution.

35. CAIRNS, Brian; SIMON, Cheryl. *Google I/O 2013 - The Secrets of the Drive Realtime API* [online]. 2013 [cit. 2018-04-05]. Dostupné z: https://www.youtube.com/watch?v=hv14PTbkIs0.

36. APPJET AND THE GOOGLE PR TEAM. *Google Acquires AppJet* [online]. 2009 [cit. 2018-04-05]. Dostupné z: https://web.archive.org/web/20091206200422/http://etherpad.com/ep/blog/posts/google-acquires-appjet.

37. APPJET AND THE GOOGLE PR TEAM. *EtherPad Open Source Release* [online]. 2009 [cit. 2018-04-05]. Dostupné z: https://web.archive.org/web/20091221023828/http://etherpad.com/ep/blog/posts/etherpad-open-source-release.

38. *Etherpad: Really real-time collaborative document editing* [online]. GitHub, 2018 [cit. 2018-04-05]. Dostupné z: https://github.com/ether/etherpad-lite.

39. APPJET, INC., WITH MODIFICATIONS BY THE ETHERPAD FOUNDATION. *Etherpad and EasySync Technical Manual* [online]. 2011 [cit. 2018-04-05]. Dostupné z: https://raw.githubusercontent.com/ether/etherpad-lite/master/doc/easysync/easysync-full-description.pdf.

40. *Available Etherpad plugins* [online] [cit. 2018-04-05]. Dostupné z: https://static.etherpad.org/plugins.html.

41. MUNROE, Lee; MEHTA, Tejesh. *Share Code in Real-time with Developers* [online]. 2018 [cit. 2018-04-05]. Dostupné z: https://codeshare.io.

42. MUNROE, Lee. *My First Node.js App: CodeShare.io* [online]. 2013 [cit. 2018-04-05]. Dostupné z: https://www.leemunroe.com/codeshare/.

43. SANDOVAL, Kristopher. 13 Node.js Frameworks to Build Web APIs. *Nordic APIs* [online]. 2017 [cit. 2018-04-11]. Dostupné z: https://nordicapis.com/13-node-js-frameworks-to-build-web-apis/.

44. GAMMA, Erich; HELM, Richard; JOHNSON, Ralph; VLISSIDES, John. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1994. ISBN 978-0201633610.

45. WHEELER, Daniel Lowe. *USENIX Security '16 - zxcvbn: Low-Budget Password Strength Estimation* [online]. 2016 [cit. 2018-04-26]. Dostupné z: `https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/wheeler`.

46. GOOGLE INC. *Introduction - Material Design* [online]. 2017 [cit. 2018-04-27]. Dostupné z: `https://material.io/guidelines/`.

47. *CodeMirror* [online] [cit. 2018-04-27]. Dostupné z: `https://codemirror.net`.

48. MÜHLEMANN, Jan. *Introduction - i18next documentation* [online]. 2018 [cit. 2018-04-30]. Dostupné z: `https://www.i18next.com`.

Dodatek A

# Seznam použitých zkratek

**API** Application Programming Interface.

**BSON** Binary JSON.

**CSS3** Cascading Style Sheets verze 3.

**DBMS** Database Management System.

**DS** Diferenciální synchronizace.

**ECMA** European Computer Manufacturer's Association.

**HTML** Hypertext Markup Language.

**HTTP** Hypertext Transfer Protocol.

**I/O** vstupní/výstupní.

**IETF** Internet Engineering Task Force.

**JSON** JavaScript Object Notation.

**MVC** Model–view–controller.

**NoSQL** Not only SQL.

**ODM** Object-document mapping.

**ORM** Object-relational mapping.

**OT** Operační transformace.

**REST** Representational state transfer.

**SQL** Structured Query Language.

**SVN** Apache Subversion.

**SŘBD** Systém Řízení Báze Dat.

**TCP/IP** Transmission Control Protocol/Internet Protocol.

**URL** Uniform Resource Locator.

**W3C** The World Wide Web Consortium.

**WebRTC** Web Real-Time Communication.

**WHATWG** Web Hypertext Application Technology Working Group.

**WWW** World Wide Web.

**WYSIWYG** „co vidíš, to dostaneš“.

**ČVUT** České vysoké učení technické.

Dodatek B

# Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `exe` ..... adresář se spustitelnou formou implementace
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF
  - `thesis.ps` ..... text práce ve formátu PS