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.

14. 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 14. 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, z 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 seznámení se zadanými technologiemi, které budou využity při implementaci prototypu. Dalším cílem je analýza problematiky kolaborativní editace textů a rozbor existujících webových real-time protokolů.

Cílem praktické části je navržení modelu pro uložení dat, implementace prototypu a následné uživatelské otestování a vyhodnocení kvality implementovaného řešení.

Kapitola 2

Analýza

2.1 Technologie

V této sekci popisuji použité technologie vycházející z nefunkčních požadavků, které jsou uvedeny v kapitole 2.6.

2.1.1 HTML5

HTML5 je značkovací jazyk používaný pro reprezentaci a strukturování obsahu na internetu (přesněji na World Wide Web (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 a 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 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].

2.1.2 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 a
- 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]

2.1.3 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]

2.1.4 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].

2.1.5 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 vyvíjena 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]

2.2 Typy síťové komunikace

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

2.2.1 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]

2.2.2 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 zpráva a tak neví kdy odeslat požadavek pro získání nové zprávy, ale nyní mu stačí počkat a server mu novou zprávu pošle sám. [17]

2.2.2.1 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]

2.2.2.2 HTTP streaming

Další metodou implemntace push technologie je takzvaný HTTP Streaming, který je podobný metodě Long polling s tím rozdílem, že 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 (pomocí hlavičky `Transfer-Encoding: chunked` v rámci protokolu HTTP verze 1.1 a starší).

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]

2.2.2.3 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]

2.2.2.4 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 ovšem kvůli vysoké popularitě stále zlepšuje a obecně je stále lepší než podpora Server-Sent Events. [20]

2.3 Algoritmy používané pro kolaborativní editaci textů

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

2.3.1 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 pokouše automaticky uzamykat pouze upravované části dokumentu. To však zamezuje úzké spolupráci více uživatel nad dokumentem, protože každý může pracovat pouze na své části dokumentu.

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 2.3.3.

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 to zkusit 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*

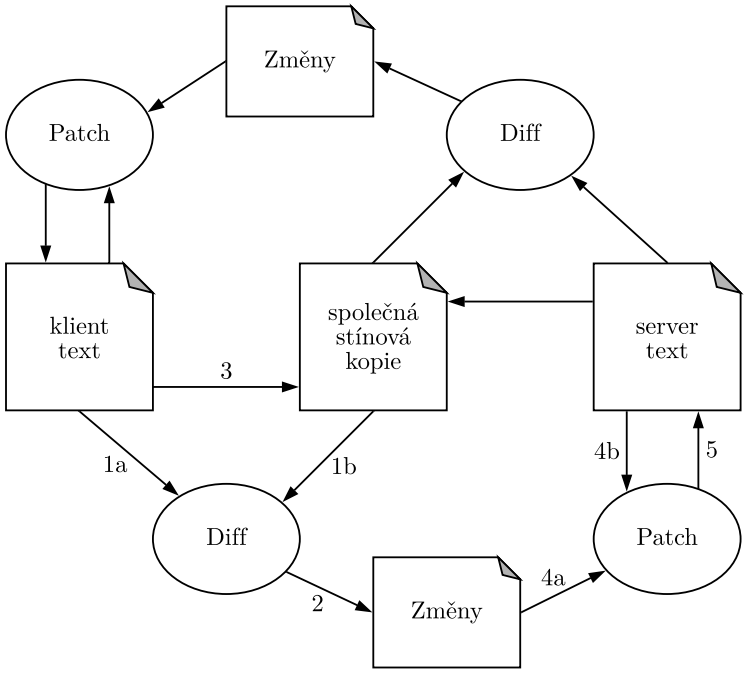
zastav a podívej se znovu. Kdyby všichni řídili stejný druh ‚dívej se a nebo jeď' automobilů, těžké nehody by byly velmi časté." [23] přeložil Jiří Šimeček.

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

2.3.2 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 kopii dokumentu.

Na obrázek číslo 2.1 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.



Obrázek 2.1: Diferenciální synchronizace vývojový diagram [23]

Na začátku jsou všechny dokumenty stejné (klient text, server text i společná stínová kopie). Klient text i server text mohou být libovolně upraveny a naším cílem je udržet oba dokumenty neustále co nejvíce podobné.

Algoritmus pokračuje následujícími kroky (viz čísla u jednotlivých přechodů diagramu 2.1):

1. klient text je porovnán oproti společní 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 math 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]

2.3.3 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 možná použít na 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 řetězce jako parametrem, který označuje řetězec k odstranění a
- 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.

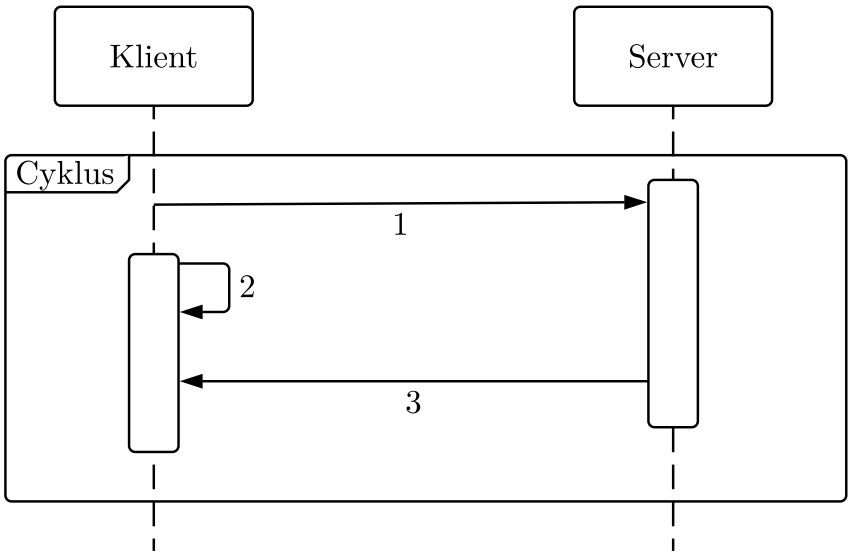
Transformace operací Dále lze operace transformovat a to pomocí jiných operací. Mějme dokument A, soubory operací z A do B, z A do C, pak aplikací změn, z kterých byl vytvořen druhý soubor, na dokument B získáme nový soubor operací (soubor transformovaný). [28]

Základní komunikace se skládá z následujících kroků (také znázorněna pomocí sekvenčního diagramu na obrázku číslo 2.2):

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]

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 (viz 2.4.1). Tento krok snižuje náročnost celého algoritmu omezením počtu současných operací, které by musel server kombinovat. [29]

Pokud klient při čekání na server dostane od serveru 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



Obrázek 2.2: Operační transformace sekvenční diagram

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 věděckých 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.

2.4 Existující řešení

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

2.4.1 Apache Wave

Apache Wave je nástupcem produktu Google Wave od společnosti Google. V lednu roku 2018 byl jeho vývoj pod záštitou Apache Foundation ukončen

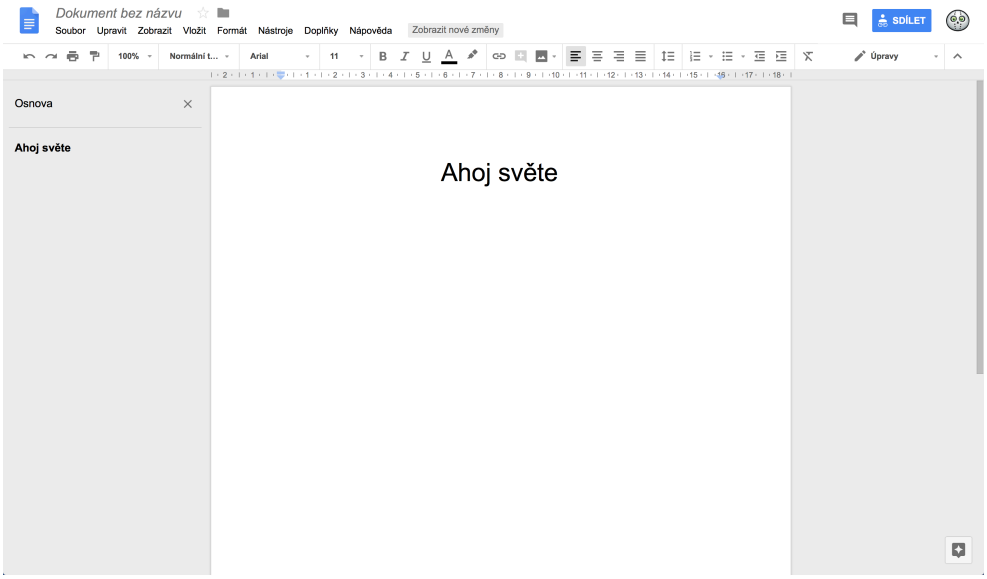
a není v něm již pokračováno. [31]

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 2.3.3) 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].

2.4.2 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 2.3.3) [34] a to včetně vylepšení se kterým přišel Google při vývoji Google Wave [35].



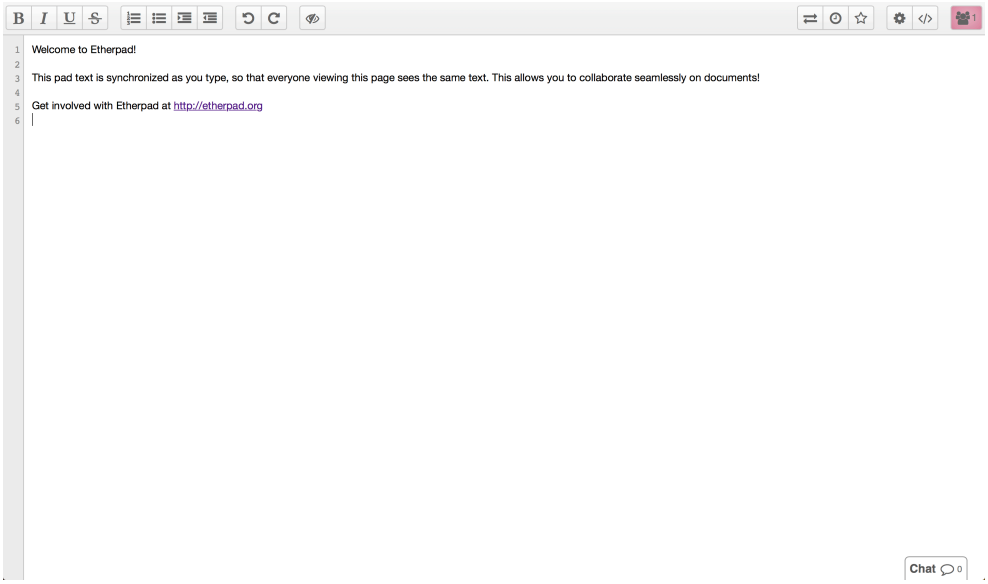
Obrázek 2.3: Ukázka aplikace Google Docs

2.4.3 Etherpad

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

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 2.1.2) a serverové prostředí NodeJS (více o NodeJS v 2.1.3), 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 2.3.3) a vylepšení ve formě včetně čekání klienta po odeslání operace na potvrzení od serveru [39]. Dnes existuje množství nejrůznějších zásuvných modelů, 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].



Obrázek 2.4: Ukázka aplikace Etherpad

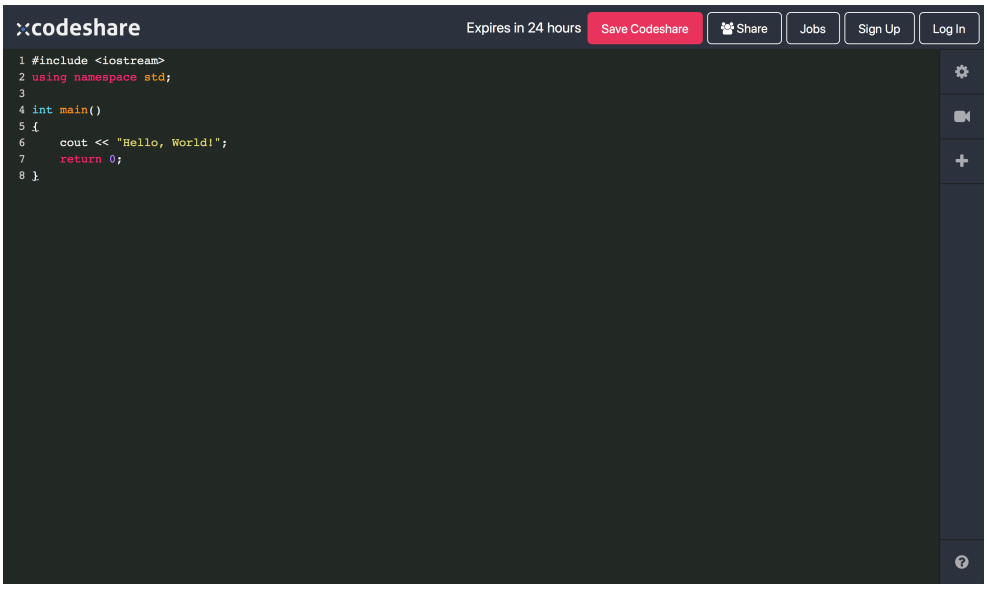
2.4.4 Codeshare.io

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-

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 2.5: Ukázka aplikace Codeshare.io

2.5 Seznam funkčních požadavků

Před začátkem návrh jakékoli aplikace je důležité definovat funkční požadavky. Funkční požadavky slouží k vymezení toho, co má aplikace umět a co už umět nemusí.

2.5.1 Správa uživatelů

Uživatel se může v rámci aplikace zaregistrovat pomocí uživatelského jména a hesla. Uživatelské jméno je pro každého uživatele unikátní a slouží jako jeho identifikátor mezi ostatními uživateli. Pomocí údajů uživatelské jména a heslo, které uvedených při registraci, se může uživatel následně do aplikace přihlásit.

Přihlášený uživatel může své údaje kdykoliv změnit. Dále může kdykoliv změnit vzhled a vlastnosti editoru pro nově vytvořené dokumenty. Toto nastavení však nemá žádný vliv na dokumenty již vytvořené.

V případě, že uživatel zapomene své přístupové heslo, může využít formuláře pro obnovu přístupového hesla. Po odeslání tohoto formuláře aplikace

odešle na emailovou adresu, kterou uživatel zadal při registraci, postup, s jehož pomocí si uživatel může nastavit nové heslo pro přístup do aplikace.

2.5.2 Správa dokumentů

Přihlášený uživatel můžu v aplikaci vytvořit nový dokument. Libovolný jím vytvořený dokument může uživatel také odstranit. Uživatel, pokud k tomu má dostatečné oprávnění, může změnit vzhled a vlastnosti editoru jednotlivých dokumentů.

Uživatel po přihlášení uvidí seznam jím vytvořených dokumentů. Dále si přihlášený uživatel může zobrazit seznam naposledy otevřených dokumentů a seznam dokumentů, ke kterým byl uživatel někým přizván.

Každý dokument má jednoznačně určený veřejný odkaz, který lze sdílet mezi uživateli. Uživatel s dostatečným oprávnění může změnit oprávnění veřejného odkazu. Toto oprávnění získají všichni uživatelé, kteří mají přístup k dokumentu (znají jeho veřejný odkaz). Uživatel s dostatečným oprávněním může přizvat dalšího uživatele ke spolupráci na dokumentu.

Uživatel s dostatečným oprávněním může zobrazit obsah dokumentu a v případě dokument ve skutečném čase editovat. Jednotlivý uživatelé upravující obsah dokumentu jsou od sebe barevně odlišeni. Jejich kurzory (či vybraný text) jsou viditelné ostatními uživateli a označeni stejnou barvou.

Uživatel s dostatečným oprávněním může zobrazit diskuzní vlákno dokumentu a pod je přihlášený může do něj i přispívat. Diskuzní vlákno je aktualizováno ve skutečném čase, tedy uživatel vidí nové zprávy bez nutnosti jakkoliv se stránkou interagovat. Uživatelovi zprávy v diskuzním vláknu jsou označeny stejnou barvou jako jeho kurzor v editoru dokumentu.

2.6 Seznam nefunkčních požadavků

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

1. validní kód HTML5 (více o jazyce HTML5 v sekci 2.1.1) a Cascading Style Sheets verze 3 (CSS3),
2. programovací jazyk Javascript (více o jazyce JavaScript v sekci **??**,
3. prostředí Node.js (více o prostředí Node.js v sekci 2.1.3) pro server a
4. použití knihovny React (více o knihovně React v čsekciásti 2.1.4) k implementaci komponenty Editoru.

2.7 Uživatelské případy

Následující body budu muset rozepsat a popsat jejich cesty.

- Vytvoření nového uživatelského účtu
- Přihlášení a zobrazení vytvořených dokumentů
- Vytvoření nového dokumentu
- Otevření sdíleného dokumentu
- Přizvání nového uživatele ke spolupráci nad dokumentem
- Změna přihlašovacích údajů
- Změna nastavení vzhledu dokumentu
- Změna výchozího nastavení pro nově vytvořené dokumenty
- Nalezení vytvořeného dokumentu a jeho odstranění

2.8 Doménový model

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

2.8.1 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 pro odstranění neaktivních účtů.

2.8.2 Dokument

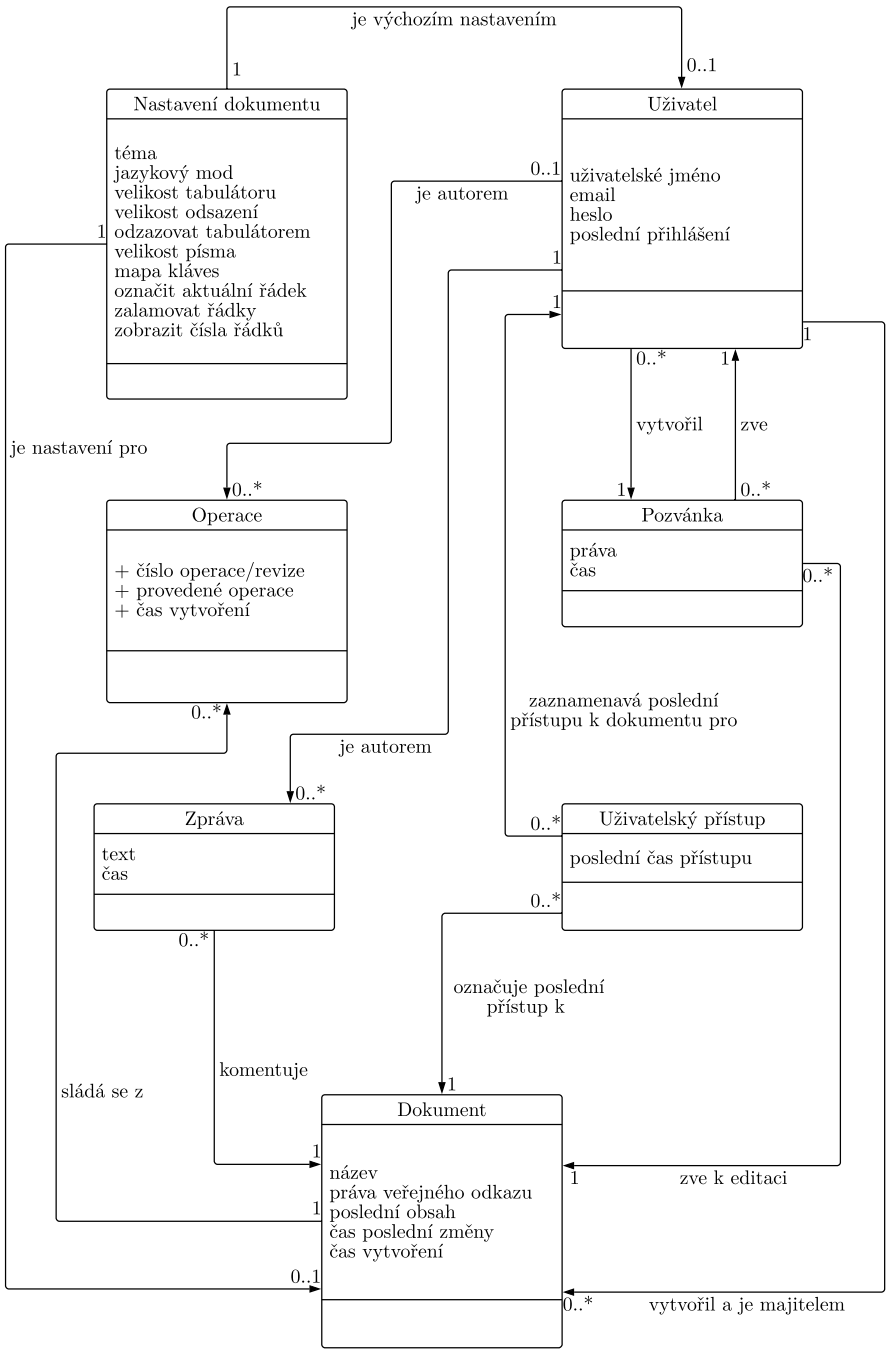
Entita dokument je nositelem informací o dokumentu. Atribut poslední obsah obsahuje poslední text dokumentu potvrzený serverem, který je posílán nově připojeným klientům.

2.8.3 Nastavení dokumentu

Nastavení dokument nese informace o vzhledu a chování editoru pro jednotlivé dokumenty. Vazba na entitu Uživatel umožňuje vytvoření výchozí instance Nastavení dokumentu pro nově vytvořené dokumenty.

2.8.4 Operace

Entita Operace označuje jednotlivou revizi dokumentu, ale také soubor operací v rámci algoritmu OT.



Obrázek 2.6: Diagram doménového modelu

2.8.5 Další entity

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

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, protože je 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 3.4 a 3.5).

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 2.1).

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 2.1.3) 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

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 část

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 2.1.5). 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 serverová 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.

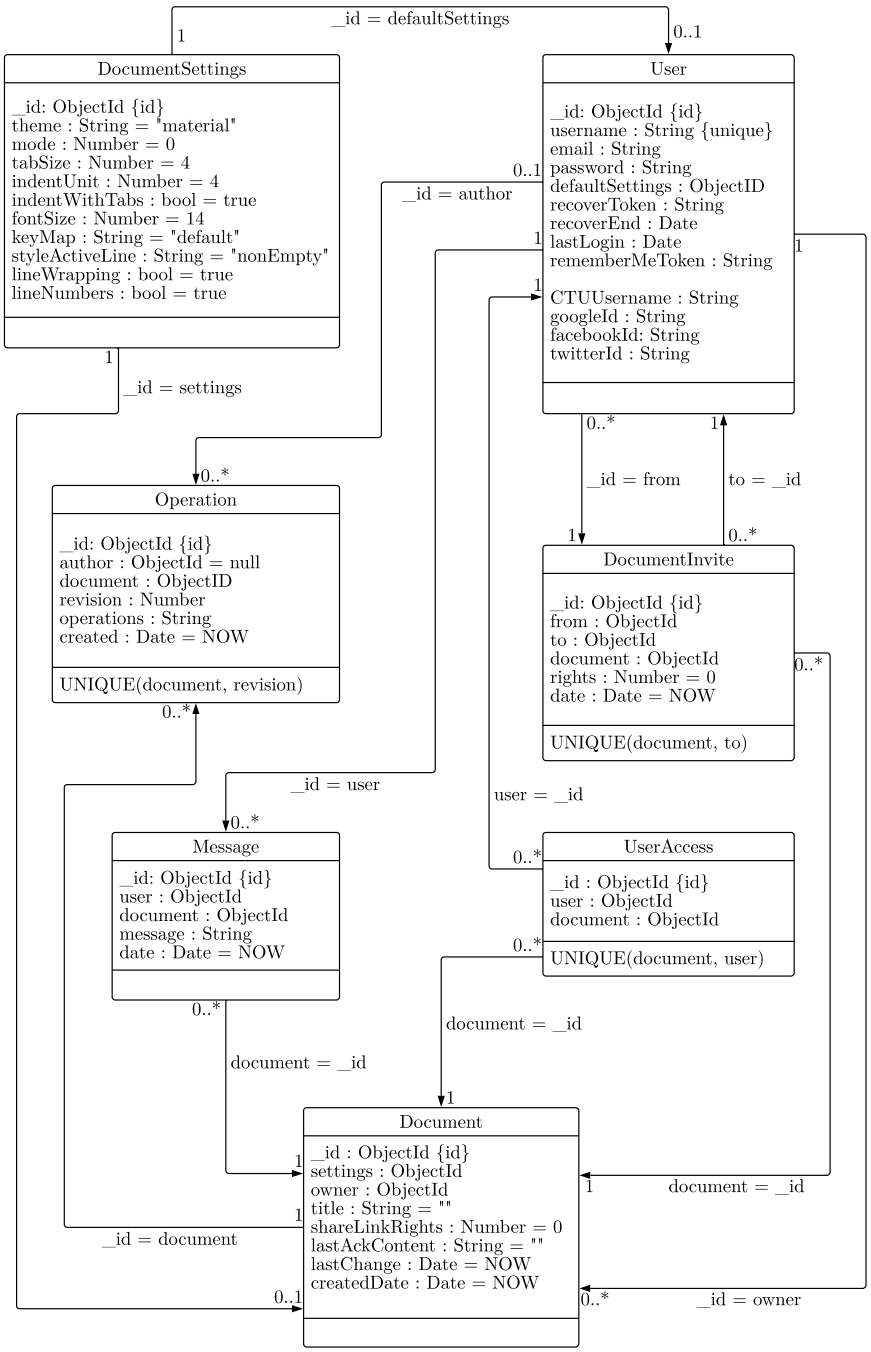
Z tohoto důvodu jsem se rozhodl použít ODM nástroj Mongoose, který umožňuje v rámci aplikace definovat struktury jednotlivých dokumentů pro MongoDB a jejich validitu kontroluje před každým uložením. Schéma na obrázku 3.1 není databázovým schématem v pravém slova smyslu, ale jedná se o schéma definované za pomoci právě ODM Mongoose.

Jednotlivé entity schématu přímo reflektují entity z doménového modelu uvedeného v sekci 2.8. 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.

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 2.3.3) a to i přes jeho složitější



Obrázek 3.1: Databázové schéma v rámci ODM Mongoose

implementaci. Algoritmus je dostatečně rozšířený a otestovaný praxí (příkladem jeho úspěšného použití jsou projekty jako Google Wave a jeho mladší sourozenec Google Docs, viz 2.4).

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 3.6, v souvislosti s návrhem komponenty editoru.

3.4 REST komunikace

Základním způsobem komunikace mezi klientskou a serverovou části aplikace je Representational state transfer (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.

3.4.1 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).

3.4.2 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.

3.4.3 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.

3.4.4 Výběr 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.

3.4.5 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.

3.4.5.1 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

3.4.5.2 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	[String]
	password	[String]
Úspěch:	200	Informace o přihlášeném uživateli
Neúspěch:	422	Popis chyby

3.4.5.3 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

3.4.5.4 POST /api/auth/forgotPassword

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.

3.4.5.5 GET /api/auth/forgotPassword/:token

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

3.4.5.6 PUT /api/auth/forgotPassword/:token

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

3.4.5.7 GET /api/user

Koncový bod pro zjištění informací o aktuálně přihlášeném uživateli.

3.4.5.8 PUT /api/user

Koncový bod umožňující aktualizaci jednotlivých parametrů přihlášeného uživatele. Přijímanými parametry jsou uživatelské jméno, email, nové heslo a aktuální heslo.

Veškeré změny musí být potvrzeny platným aktuální heslem.

3.4.5.9 GET /api/user/document-settings

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

3.4.5.10 PUT /api/user/document-settings

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 3.1 (krom pole _id).

3.4.5.11 POST /api/document

Koncový bod umožňující aktuálně přihlášenému uživateli vytvořit nový dokument.

3.4.5.12 GET /api/document

Koncový bod pro získání dokumentů vytvořených aktuálně přihlášeným uživatelem.

3.4.5.13 GET /api/document/last

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

3.4.5.14 GET /api/document/shared

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

3.4.5.15 GET /api/document

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

3.4.5.16 GET /api/document/:documentId/messages

Koncový bod pro získání zpráv ohledně 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 Uniform Resource Locator (URL) parametrem number a čas odeslání poslední vrácené zprávy lze určit URL parametrem lastDate.

3.4.5.17 POST /api/document/:documentId/messages

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.

3.4.5.18 GET /api/document/:documentId/rights

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.

3.4.5.19 PUT /api/document/:documentId/rights

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.

3.4.5.20 PUT /api/document/:documentId/rights/invite

Koncový bod pro úpravu oprávnění pro přizvaného uživatele k dokumentu (identifikovaného pomocí parametru documentId). Uživatel je identifikován pomocí parametru uživatelské jméno.

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

3.4.5.21 DELETE /api/document/:documentId/rights/:toUserId

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.

3.4.5.22 DELETE /api/document/:documentId

Koncový bod umožňující trvalé odstranění dokumentu identifikovaného pomocí parametru documentId.

Pro jeho použití musí být aktuálně přihlášený uživatel vlastníkem dokumentu.

3.4.5.23 GET /locales/:lang/translation.json

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).

3.5 Komunikace ve skutečném čase

K implementaci komunikace ve skutečném čase existuje více způsoby, ale každý má své kompromisy (více o push technologiích v sekci 2.2.2). Pro komunikaci jsem chtěl použít technologii WebSocket (viz sekce 2.2.2.4), 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 2.2.2.1).

3.5.1 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.

3.5.1.1 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.

3.5.1.2 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.

3.5.1.3 Klient se odpojil

Zprávu Klient se odpojil odešle server zbylým klientům po odpojení jiného klienta. Ostatní klienti si odeberou ze svého seznamu klientů klienta s přijatým identifikátorem, který obdrží v rámci této zprávy.

3.5.1.4 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 2.3.3).

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.

3.5.1.5 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).

3.5.1.6 Nastavení

Zpráva Nastavení slouží pro synchronizaci změn nastavení mezi aktuálně připojenými klienty. Klient zprávu odešle spolu se změněným nastavením do-

kumentu, server nastavení pro daný dokument uloží a následovně jej odešle ostatním připojeným klientům.

3.5.1.7 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 3.4), které umožňují klientům získat například historii komunikačního vlákna.

3.5.1.8 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.

3.6 Komponenta editoru

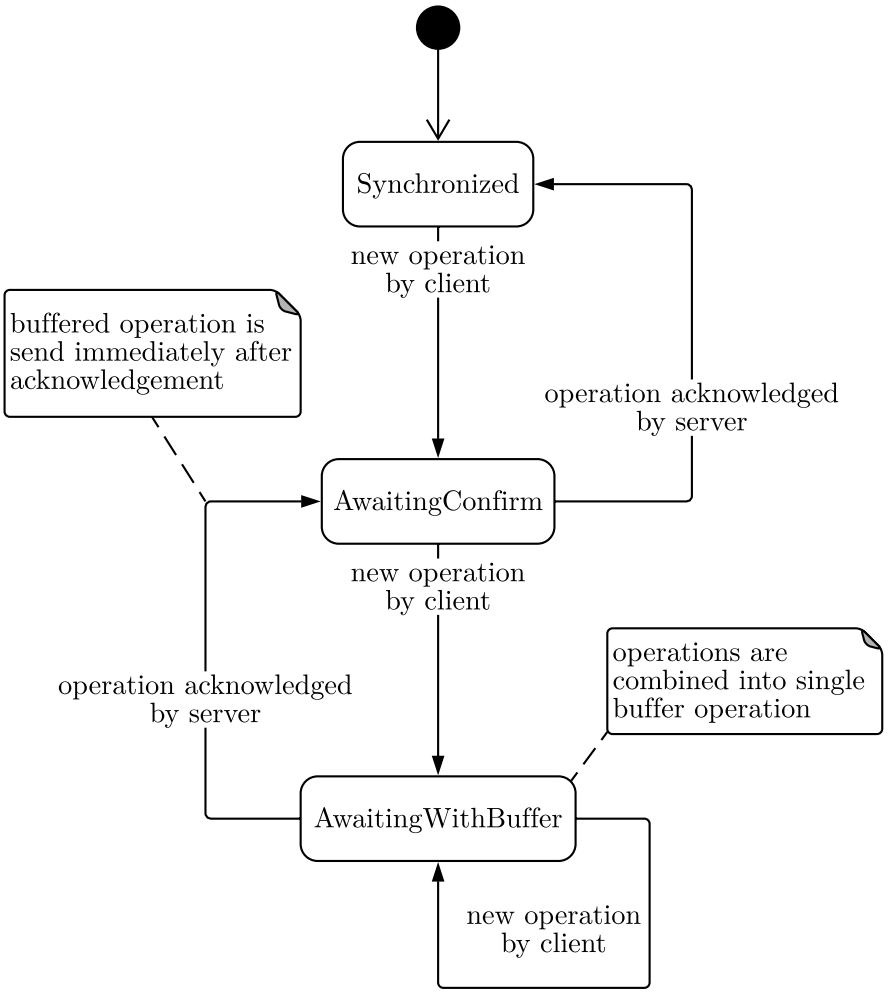
Komponenta editoru je hlavním částí aplikace, umožňuje uživatelům společně upravovat dokumenty ve skutečném čase díky použití algoritmu OT (více o volbě algoritmu v sekci 3.3). Komponenta tento algoritmus implementuje a poskytuje rozhraní pro jeho použití bez omezení na použité technologii komunikace, či samotné knihovny textového editoru.

Komponentu editoru rozdělíme na dvě části, část klientská a část serverová.

3.6.1 Klientská část komponenty

Klientská část musí komunikovat se zvoleným textovým editorem, zachytávat uživatelův 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 technologii s částí serverovou (propagace jednotlivých Operací mezi uživateli).

Jádrem této 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 a může nabývat 3 stavů (viz stavový diagram na obrázku 3.2).

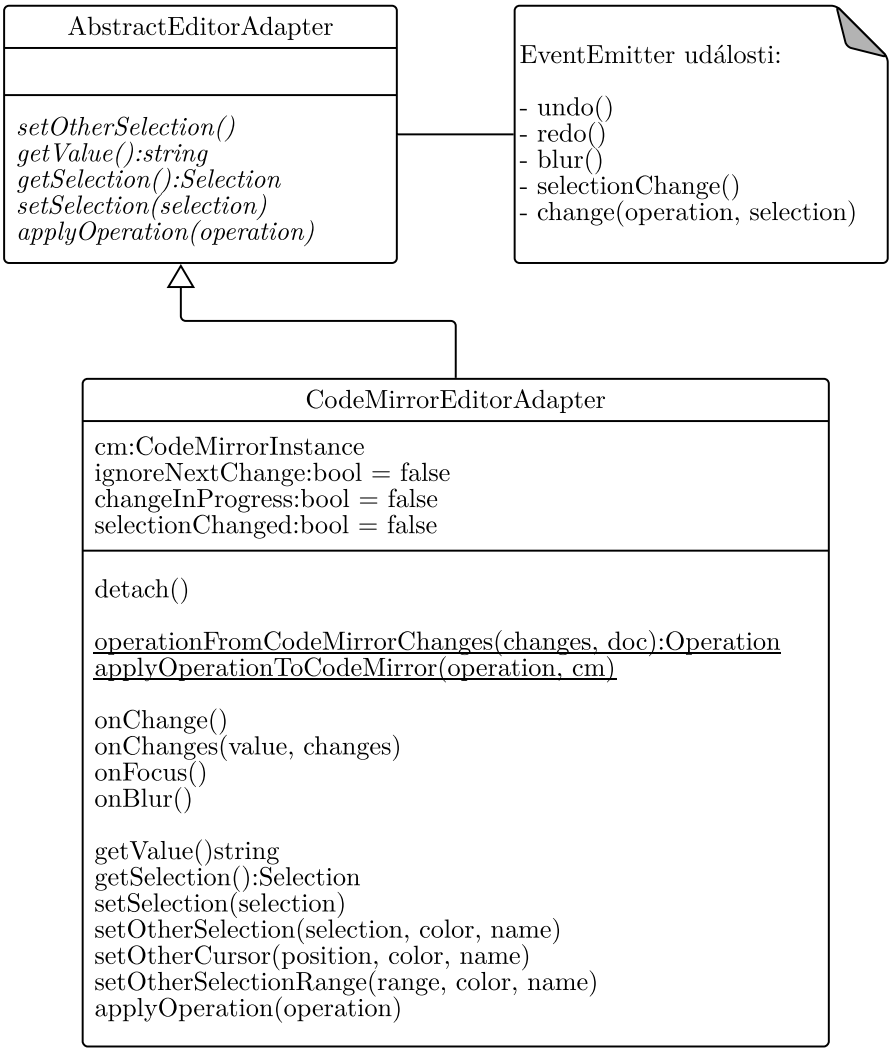


Obrázek 3.2: Stavový diagram klientské části komponenty

Třída EditorClient očekává implementaci třídy AbstractEditorAdapter (respektive AbstractServerAdapter), která slouží jako rozhraní pro komunikaci s textovým editorem (respektive serverovou částí). 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.

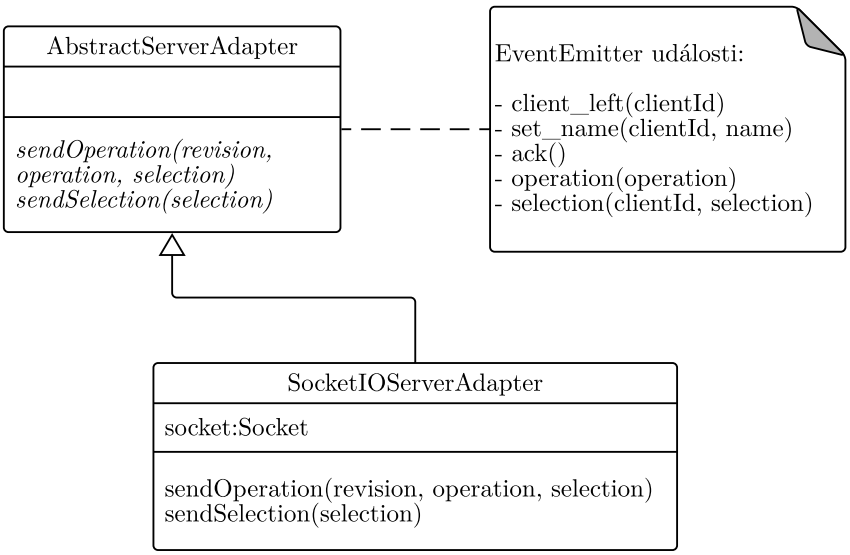
Tyto abstraktní třídy jsou potomky třídy EventEmitter, která je ustálenou implementací návrhového vzory Pozorovatel (anglicky Observer) pro jazyk Javascript. EventEmitter umožňuje třídě EditorClient naslouchat jednot-

livým 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 3.3 (respektive 3.4).



Obrázek 3.3: Diagram implementace třídy AbstractEditorAdapter

EditorClient také využívá třídu UndoManager z knihovny OT.js, díky kterému lze použít bezpečně funkce zpět a vykonat znovu (pokud jejich odchycení podporuje poskytnutá implementace třídy AbstractEditorAdapter). Historie



Obrázek 3.4: Diagram implementace třídy AbstractServerAdapter

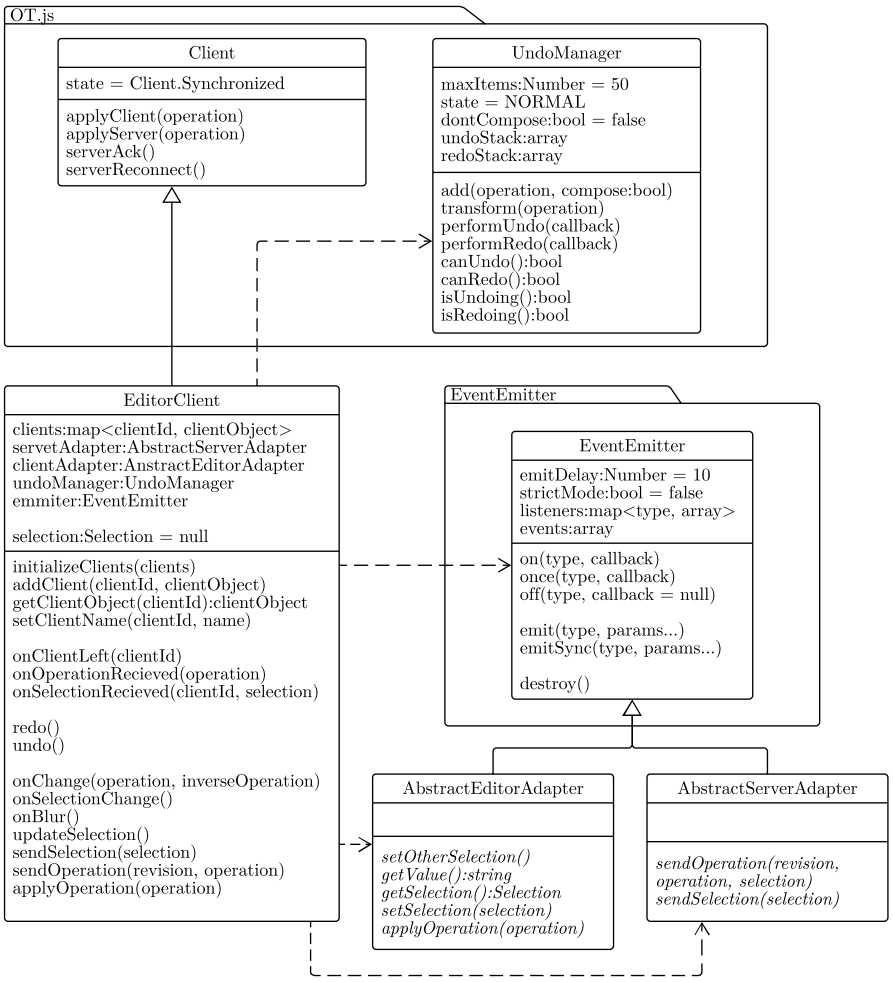
je tak zaznamenávána pomocí jednotlivých operací uživatele a nikoly podle změn samotného textu. Na změnách textu se může podílet více uživatelů najednou a není žádoucí, aby funkce zpět vracela změny provedené jiným než lokálním uživatelem.

3.6.2 Serverová část komponenty

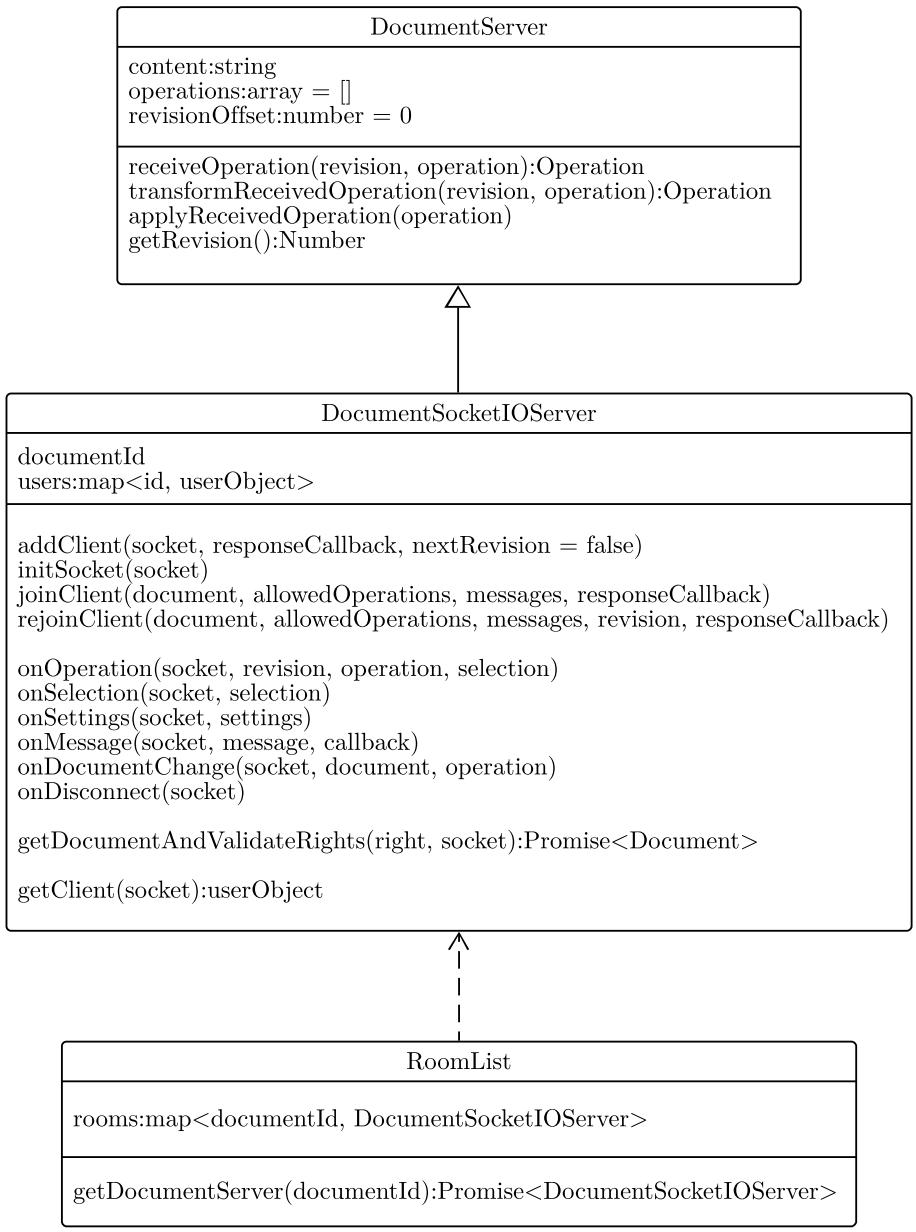
Serverová část je odpovědná za propagaci jednotlivých operací mezi klienty připojenými k dokumentu. Základem této části je třída DocumentServer (reimplementace třídy Server z knihovny OT.js) a v případě navrhovaného prototypu aplikace její potomek DocumentSocketIOServer.

Instance třídy DocumentServer (respektive DocumentSocketIOServer) představuje jeden document a je zodpovědná o implementaci serverové části algoritmu OT. Musí umět transformovat přijaté revize oproti souběžným, ale již schváleným operacím, a propagovat 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.



Obrázek 3.5: Třídní diagram klientské části komponenty



Obrázek 3.6: Třídní diagram serverové části komponenty

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. In: [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. In: [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/.

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“.

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