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.

23. 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 23. 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 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

Návrh

2.1 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 2.1), protože jsou od sebe role klienta a serveru odděleny. Komunikace mezi nimi probíhá pomocí předem definovaných aplikačních rozhraní postavených na protokolu Hypertext Transfer Protocol (HTTP) (více o rozhraní v sekcích 2.5 a 2.6).

2.1.1 Klientská část

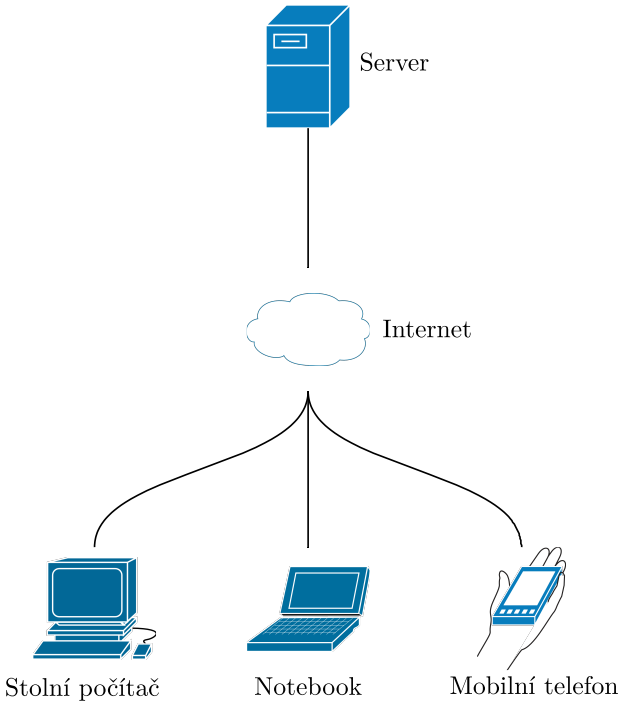
Jedná se o část aplikace, která běží v samotném prohlížeči uživatele. Reaguje na uživatelský vstup, generuje uživatelské rozhraní podle stavu aplikace a pomocí protokolu HTTP komunikuje se serverovou částí za účelem úpravy, či získání dat.

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

2.1.2 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 **??**) 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 2.1: Diagram modelu 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.

2.1.3 Databázová čast

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

Rozhodl jsem se pro použití Systém Řízení Báze Dat (SŘBD) MongoDB (více o databázích v sekci **??**). MongoDB umožňuje rychlejší vývoj aplikací a pro problém editace textů se jako zástupce Not only Structured Query Language (SQL) (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.

2.2 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 Object-document mapping (ODM) nástroj Mongoose, který umožňuje v rámci aplikace definovat struktury jednotlivých dokumentů pro MongoDB a jejich validitu kontroluje před každým uložením. Schéma na obrázku 2.2 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 **??**. 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.

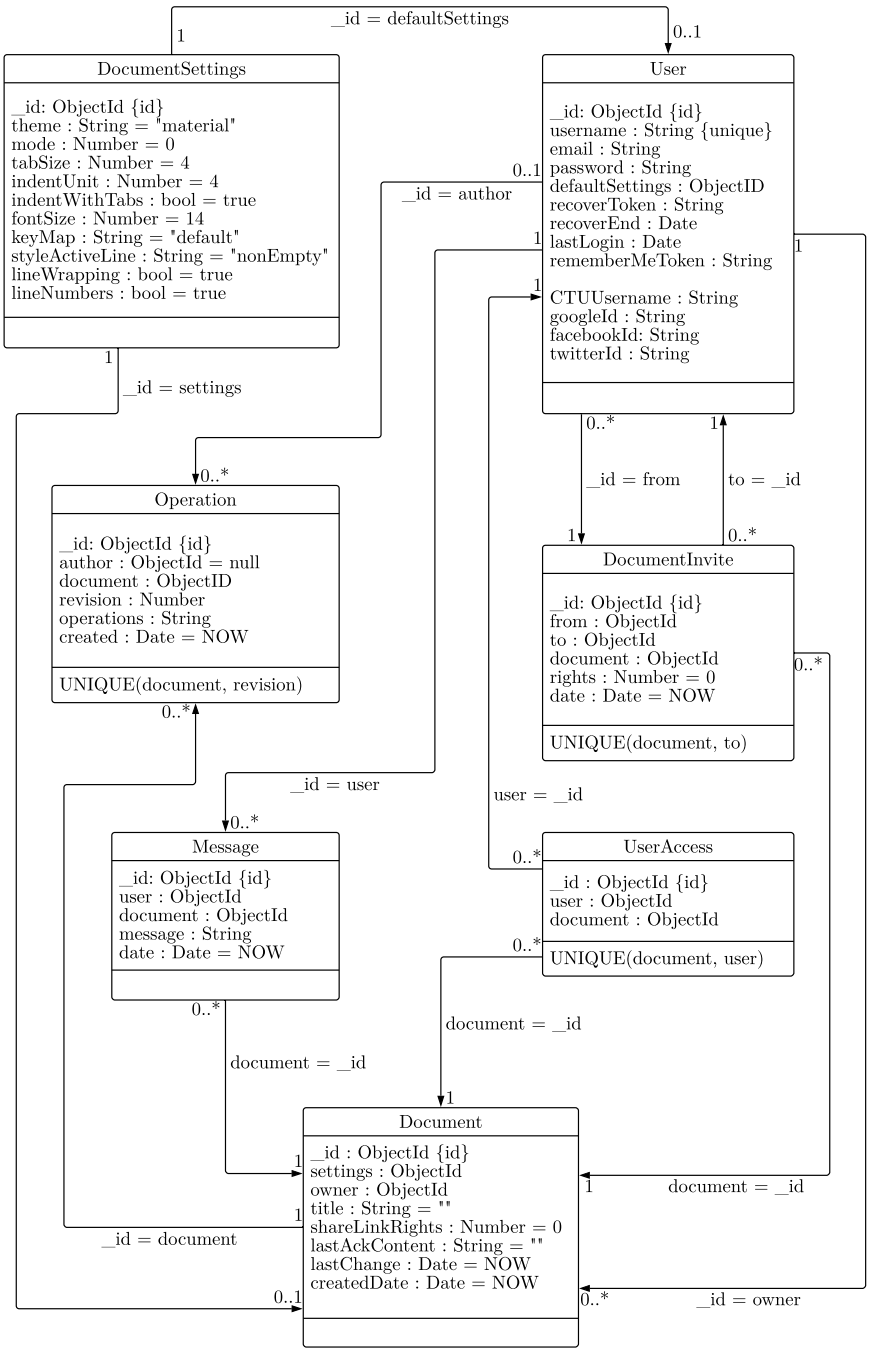
Jedinou výraznou změnou oproti doménovému modelu, která zde stojí za zmínku, je entita `User`. Přibyly zde atributy sloužící pro autentizaci pomocí služeb třetích stran (například sociální sítě, či ČVUT heslo). Tyto atributy nesou identifikační údaj pro uživatele v rámci dané autentizační služby třetí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

2.3 Algoritmus synchronizace editovaného textu

Pro účely synchronizace textu ve skutečném čase jsem se rozhodl pro použití algoritmu Operační transformace (OT) (více o algoritmu v sekci **??**) 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 **??**).

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 2.7, v souvislosti s návrhem komponenty editoru.



Obrázek 2.2: Databázové schéma v rámci ODM Mongoose

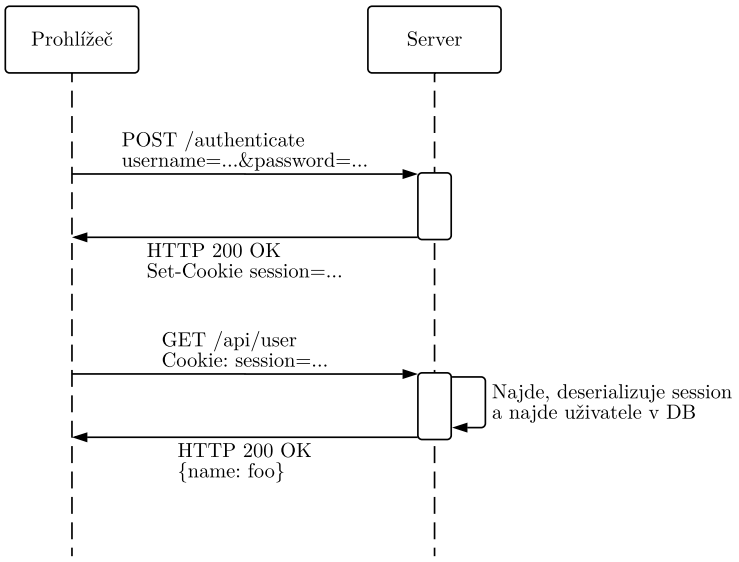
2.4 Autentizace

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.

2.4.1 Stavová autentizace

Stavová autentizace (viz diagram na obrázku 2.3) 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.

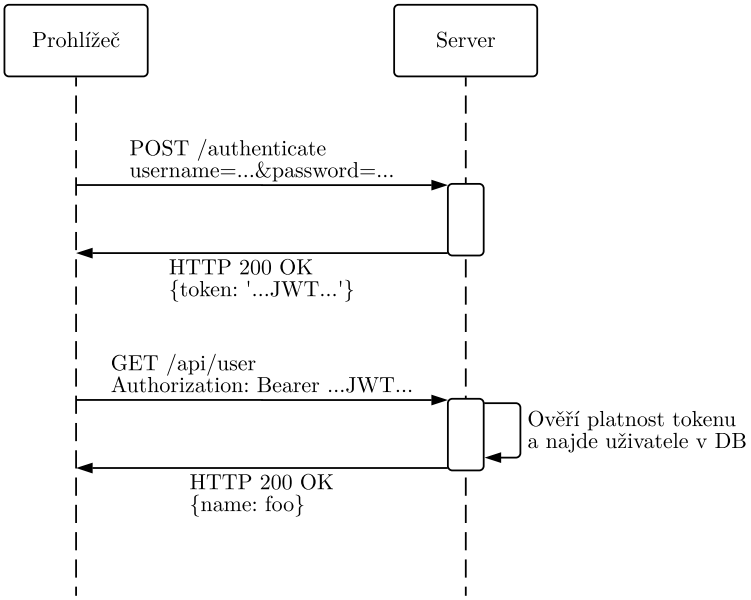


Obrázek 2.3: Sekvenční diagram stavové autentizace

2.4.2 Bezstavová autentizace

Bezstavová autentizace (viz diagram na obrázku 2.4) je v poslední době velmi oblíbený způsob autentizace především pro Representational state transfer (REST) Application Programming Interface (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.



Obrázek 2.4: Sekvenční diagram bezstavové autentizace

2.4.3 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 naprosto automaticky. To platí jak pro komunikaci pomocí REST (viz následující sekce 2.5),

tak i pro komunikaci ve skutečném čase (viz následující sekce 2.6), kde se cookies odesílají při navázání spojení. Jedinou nevýhodou tohoto řešení je, že obě části aplikace (klientská i serverová) musí být umístěny na stejné doméně. V případě navrhovaného prototypu to není problém, jelikož klientská i serverová část jsou sloučeny do jedné aplikace (viz sekce 3.1 o zveřejnění klientské části).

2.5 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 [1] jsem výběr zúžil na 3 neoblíbenější Node.js frameworky: Sails.js, Express.js a Hapi.js.

2.5.1 Sails.js

Sails.js je plnohodnotný webový Model–view–controller (MVC) framework pro Node.js. Jeho integrovanou součástí je Waterline Object-relational mapping (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).

2.5.2 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.

2.5.3 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 vy-

soká a nevyrovná se přidané hodnotě, která framework přináší oproti použití Express.js.

2.5.4 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.

2.5.5 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 JavaScript Object Notation (JSON), a vrací HTTP stavový kód spolu s nepovinnou odpovědí, která je také ve formátu JSON.

2.5.5.1 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 2.1.

Tabulka 2.1: 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

2.5.5.2 Přihlášení uživatele

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

Více informací o koncovém bodu lze nalézt v tabulce 2.2.

2.5.5.3 Ohlášení uživatele

Koncový bod pro odhlášení aktuálně přihlášeného uživatele.

Tabulka 2.2: Koncový bod Přihlášení uživatele

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

Více informací o koncovém bodu lze nalézt v tabulce 2.3.

Tabulka 2.3: Koncový bod Ohlášení uživatele

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

2.5.5.4 Odeslání požadavku na obnovu hesla 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.

Tabulka 2.4: Koncový bod Odeslání požadavku na obnovu hesla

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

2.5.5.5 Zjištění platnosti kódu pro obnovu hesla

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

Tabulka 2.5: 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

2.5.5.6 Obnova hesla pomocí kódu

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

Tabulka 2.6: 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

2.5.5.7 Informace o přihlášeném uživateli

Koncový bod pro zjištění informací o aktuálně přihlášeném uživateli.

Tabulka 2.7: 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

2.5.5.8 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 2.8: 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

2.5.5.9 Výchozí nastavení dokumentů

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

Tabulka 2.9: 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

2.5.5.10 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 2.2 (krom pole _id).

2.5.5.11 Vytvoření dokumentu

Koncový bod umožňující aktuálně přihlášenému uživateli vytvořit nový dokument.

Tabulka 2.10: 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 2.11: 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

2.5.5.12 Vytvořené dokumenty

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

Tabulka 2.12: Koncový bod Vytvořené dokumenty

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

2.5.5.13 Poslední dokumenty

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

Tabulka 2.13: 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

2.5.5.14 Sdílené dokumenty

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

Tabulka 2.14: 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

2.5.5.15 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.

2.5.5.16 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 2.15: Koncový bod Komunikační vlákno dokumentu

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

Tabulka 2.16: 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

2.5.5.17 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 2.17: 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

2.5.5.18 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 2.18: 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

2.5.5.19 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 2.19: 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

2.5.5.20 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 2.20: 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

2.5.5.21 Odstranění dokumentu

Koncový bod umožňující trvalé odstranění dokumentu identifikovaného pomocí parametru documentId.

Pro jeho použití musí být aktuálně přihlášený uživatel vlastníkem dokumentu.

Tabulka 2.21: 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

2.5.5.22 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 2.22: 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

2.6 Komunikace ve skutečném čase

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

2.6.1 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.

2.6.1.1 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.

2.6.1.2 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.

2.6.1.3 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.

2.6.1.4 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 **??**).

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.

2.6.1.5 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).

2.6.1.6 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.

2.6.1.7 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 2.5), které umožňují klientům získat například historii komunikačního vlákna.

2.6.1.8 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.

2.7 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 2.3). 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.

2.7.1 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 2.5).

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.

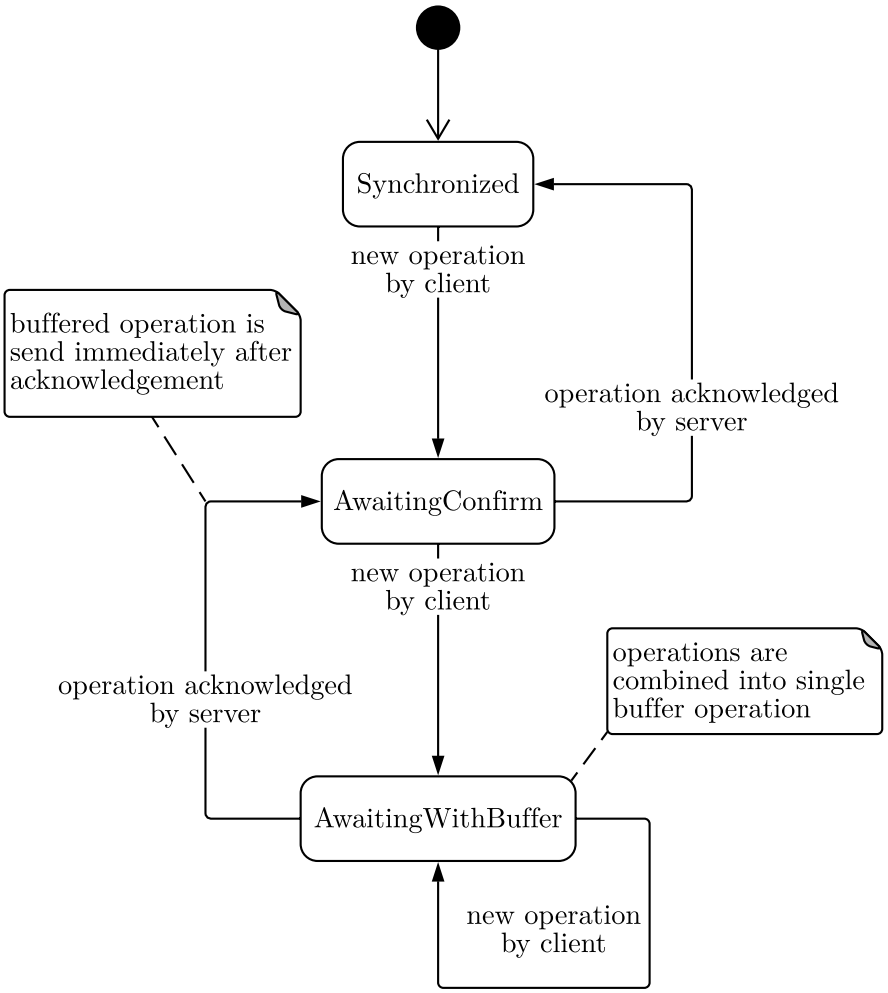
Tyto abstraktní třídy jsou potomky třídy `EventEmitter`, která je ustálenou implementací návrhového vzory Pozorovatel (anglicky Observer) pro jazyk Javascript. Rozhraní `EventEmitter` umožňuje třídě `EditorClient` naslouchat jednotlivým událostem, ke kterým dochází v implementacích zmíněných abstraktních tříd (jako je například změna pozice kursoru, či přijatá operace od serveru). Seznam jednotlivých událostí je možné pozorovat na diagramu 2.6 (respektive 2.7).

`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 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.

2.7.2 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`.

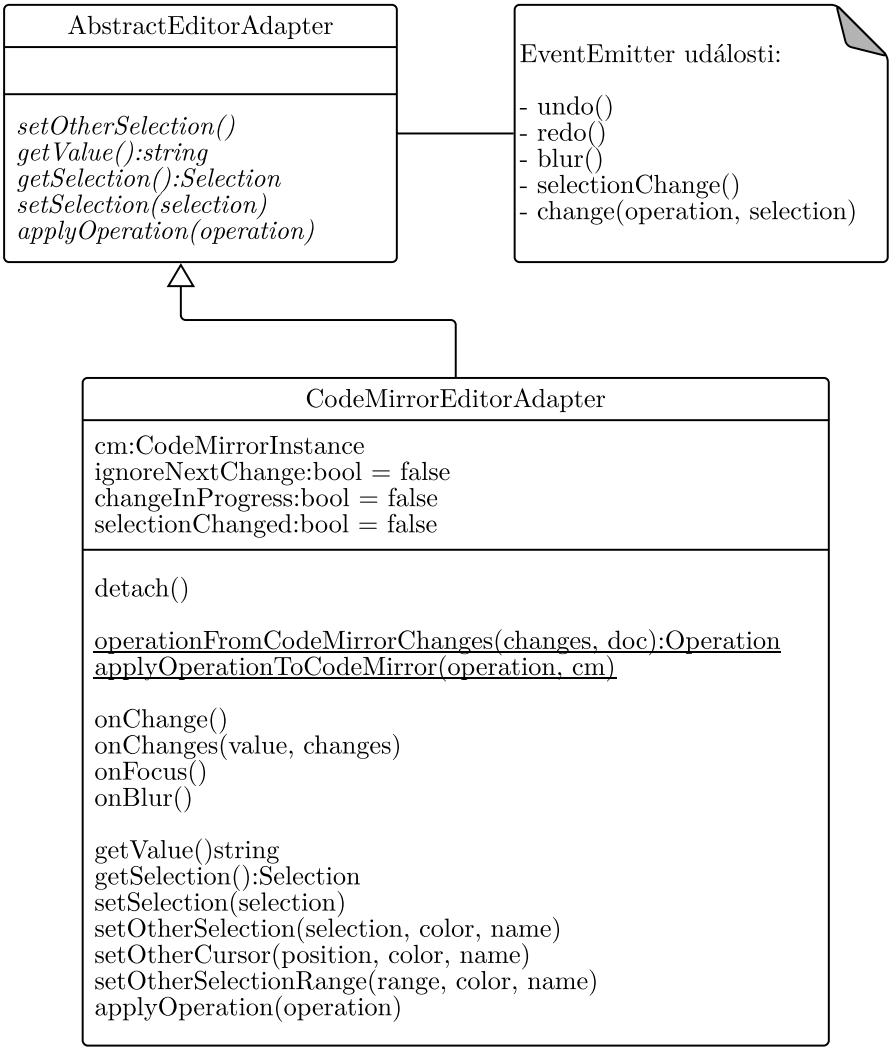
Instance třídy `DocumentServer` (respektive `DocumentSocketIOServer`) představuje jeden document a je zodpovědná o implementaci serverové části algoritmu OT. Musí umět transformovat přijaté revize oproti souběžným, ale již



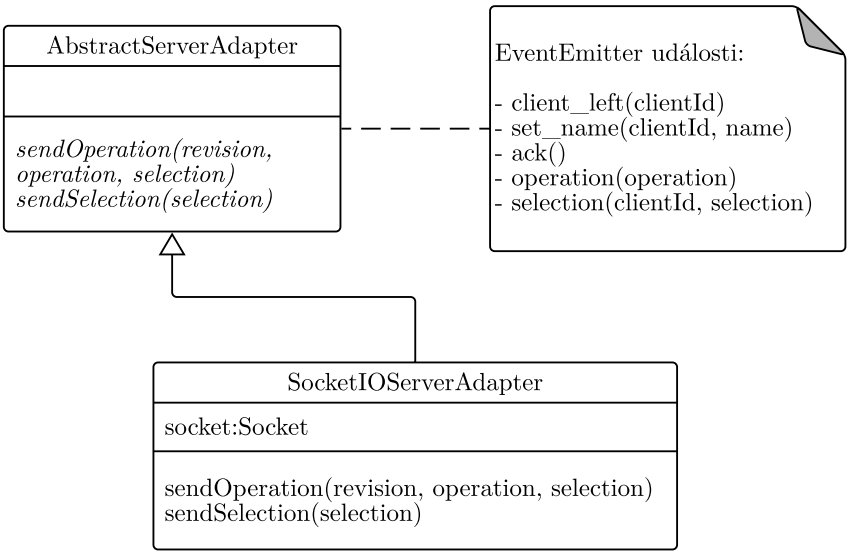
Obrázek 2.5: Stavový diagram klientské části komponenty

schváleným operacím, a propagovat tuto transformovanou operaci ostatním uživatelům.

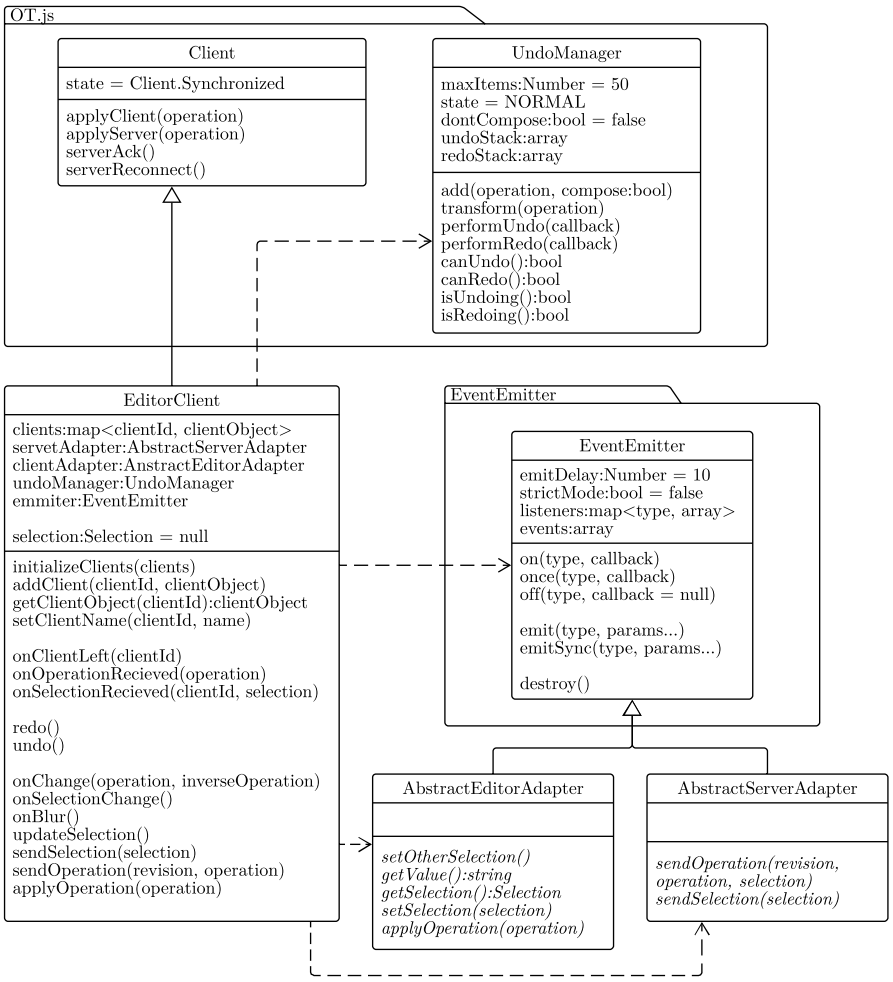
Další třídou, kterou je vhodné zmínit je třída `RoomList`, která udržuje informace o již existujících instancích třídy `DocumentSocketIOServer`, vytváří nové instance v případě, že pro daný dokument dosud neexistuje, a naslouchá novým socket.io spojení s klienty, které dále předává příslušným instancím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



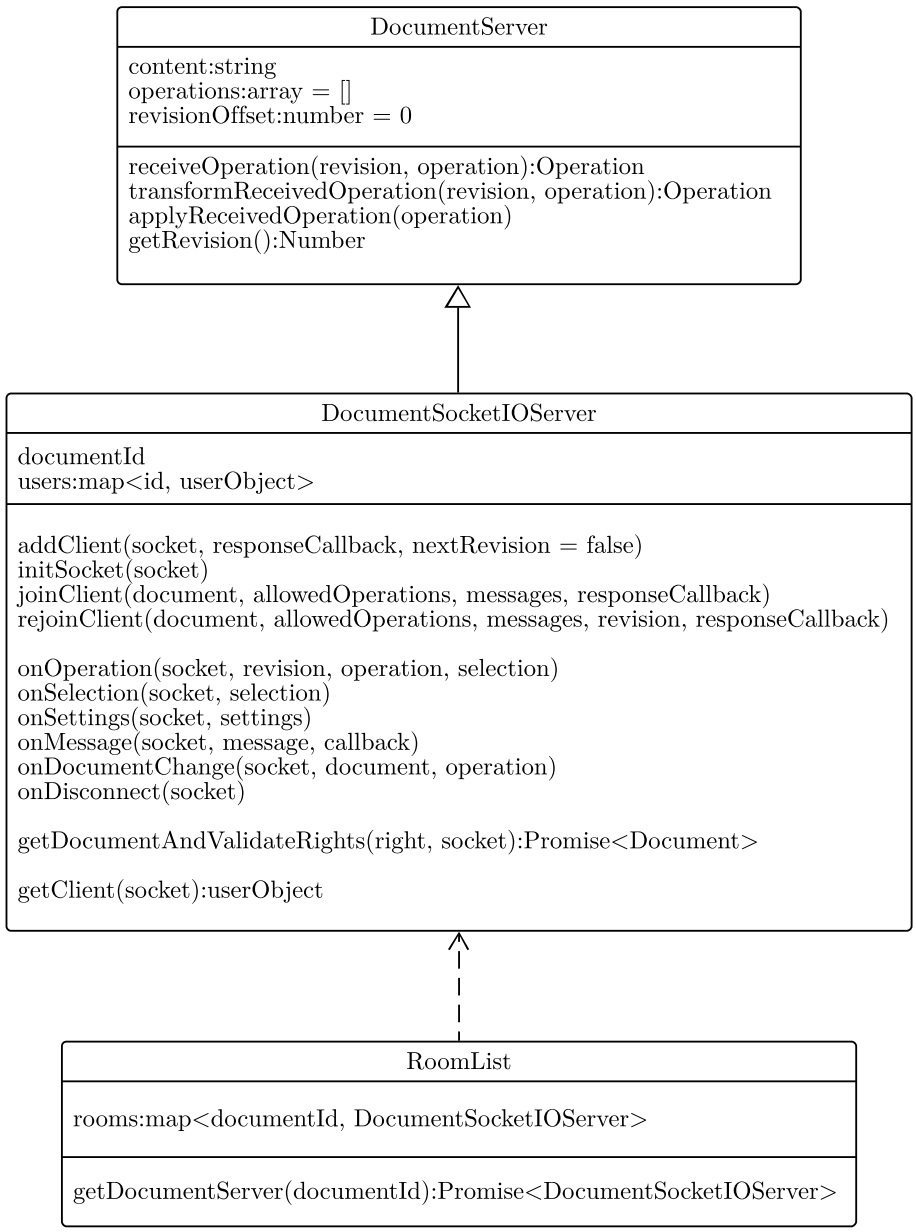
Obrázek 2.6: Diagram implementace třídy AbstractEditorAdapter



Obrázek 2.7: Diagram implementace třídy AbstractServerAdapter



Obrázek 2.8: Třídní diagram klientské části komponenty



Obrázek 2.9: Třídní diagram serverové části komponenty

Kapitola 3

Realizace

Jednotlivé zajímavé části prototypu (zatím se jedná jen o popsanou struktury kapitoly).

3.1 Zveřejnění klientské části

3.1.1 Sdílení veřejných odkazů na dokumenty

Server rendering meta tagů, knihovna react-helmet pro jejich aktualizaci.

3.2 Autentizace

Využití knihovny passport.js. Autentizace pomocí ČVUT hesla a sociálních sítí (OAuth2).

3.2.1 Zapomatování uživatele

Cookie token a jeho konzumace/generace při každém požadavku (ochrana cookie hijacking).

3.2.2 Přihlášení pomocí hesla

3.2.2.1 Uložení hesla

Použití hashovací funkce bcrypt a jeho výhody oproti použití „rychlých“ hashovacích funkcí jako jsou funkce standartu sha (dedikovaný hardware a špatně nastavitelná obtížnost výpočtu).

3.2.2.2 Odhad obtížnosti hesla

Knihovna zxcvb od společnost Dropbox. Problém s vynucování speciálních znaků v heslech (například hesla typu P@ssw0rd, které nemají téměř žádnou přidanou obtížnost díky použití známých substitucí).

3.3 Autorizace

Třída/metody `DocumentVoter` a řešení práv pro přístup k dokumentům.

Problém při zobrazení editoru (klient neví jestli dokument vůbec existuje nebo zda-li k němu má uživatel oprávnění) a jeho částí.

3.4 Překlady

Knihovna i18next a rect-i18next, asynchronní načítání překladů.

3.5 Materail desing

Knihovna material-ui ve verzi 1+, material design doporučení přímo od Google. Co to řeší (konzistence a přístupnost UX), jak jsem to použil.

Kapitola 4

Testování

Zatím se jedná jen o popsanou struktury kapitoly.

4.0.1 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).

4.0.2 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. 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. 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.

HTTP Hypertext Transfer Protocol.

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.

SŘBD Systém Řízení Báze Dat.

TCP/IP Transmission Control Protocol/Internet Protocol.

URL Uniform Resource Locator.

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