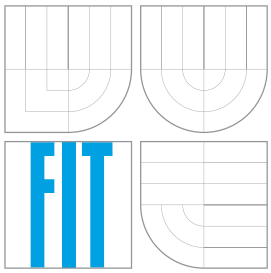
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

WEBOVÁ APLIKACE PRO SPRÁVU POZNÁMEK

WEB APPLICATION FOR MANAGING NOTES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

JAN HRDINA

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. RADEK BURGET, Ph.D.

BRNO 2016

Zadání bakalářské práce/18087/2015/xhrdin10

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2015/2016

Zadání bakalářské práce

Řešitel: **Hrdina Jan**
Obor: Informační technologie
Téma: **Webová aplikace pro správu poznámek**
Web Application for Managing Notes
Kategorie: Web

Pokyny:

1. Seznamte se s existujícími aplikačními rámci v jazyce JavaScript a s návrhem moderních webových aplikací.
2. Prostudujte existující přístupy a nástroje pro správu a editaci strukturovaných poznámek.
3. Navrhněte webovou aplikaci pro správu a editaci strukturovaných poznámek. Uvažujte různé typy dat (text, obrázky, apod.)
4. Po dohodě s vedoucím implementujte navrženou aplikaci s využitím vhodných serverových a klientských technologií.
5. Implementujte alespoň dva režimy zobrazení a editace vložených dat.
6. Diskutujte dosažené cíle a navrhněte směry dalšího vývoje.

Literatura:

- Gutmans, A., Rethans, D., Bakken, S.: Mistrovství v PHP 5, Computer Press, 2012
- Žára, O.: JavaScript - Programátorské techniky a webové technologie, Computer Press, 2015

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Burget Radek, Ing., Ph.D.**, UIFS FIT VUT
Datum zadání: 1. listopadu 2015
Datum odevzdání: 18. května 2016

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Dušan Kolář
vedoucí ústavu

Abstrakt

Práce pojednává o návrhu a implementaci webové aplikace na správu stromovitých poznámek, která si klade za cíl naplnit nevyužitý potenciál použití počítačů v této oblasti. Toho se snaží dosáhnout pomocí (1) různých způsobů zobrazení a editace stejných dat, (2) zaměření datového modelu primárně na význam ukládané informace místo vzhledu a (3) pomocí začlenění možností interaktivní práce s textem (např. připojování multimédií). Mimo samotný návrh aplikace text stručně prochází moderní trendy vývoje webových aplikací, vymezuje se vůči existujícím nástrojům na správu poznámek a nastiňuje široké možnosti dalšího vývoje. K samotné realizaci jsou využity především technologie Polymer, TypeScript a Firebase.

Abstract

The thesis deals with design and implementation of a web application for managing tree-structured notes. Its aim is to fulfill the true potential of using computers in this area. It tries to achieve this goal using (1) multiple ways of viewing and editing one data, (2) focusing the data-model primarily on the meaning of stored information instead of appearance and (3) implementing tools for interactive work with text (e.g. attaching multimedia). Apart from the application design itself, the text briefly goes through modern trends in web applications development, it defines the basic concepts in relation to existing tools for managing notes and outlines wide possibilities of further development. The application itself was implemented using technologies such as Polymer, TypeScript or Firebase.

Klíčová slova

poznámky, osnova, strom, webová aplikace, JavaScript, Polymer, Firebase, TypeScript

Keywords

notes, outline, tree, web application, JavaScript, Polymer, Firebase, TypeScript

Citace

HRDINA, Jan. *Webová aplikace pro správu poznámek.* Brno, 2016. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Burget Radek.

Webová aplikace pro správu poznámek

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Radka Burgeta, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

. . . . . .
Jan Hrdina
12. května 2016

Obsah

Kapitola 1

Úvod

Žijeme v informační době. Informace k nám proudí všemi možnými kanály. Bohužel, pokud člověk nedisponuje geniální pamětí, je takřka nemožné vše udržet v hlavě. A tak byly vynalezeny poznámky – způsob, jak vizualizovat a utřídit myšlenky, zaznamenat nové nápady a dát chaosu řád.

Člověk se učí psát poznámky od základní školy - na papír, jednu odrážku za druhou, řádek za řádkem. Většině lidí tento styl vydrží dlouhé roky, často až do let vysokoškolských studií a dál. Tak, jak to dělali naši rodiče, prarodiče, jejich rodiče a tak dál. Na papír, odrážku za odrážkou, řádek za řádkem, a to i v jednadvacátém století.

Ano, i na počítači ve většině případů pouze kopírujeme funkčnost papíru. Tvoříme statický text, který má jednu neměnnou formu. Pokud se rozhodneme použít textový procesor, píšeme odrážky. Pokud použijeme editor myšlenkových map, vytváříme myšlenkové mapy. Jsme omezeni na jednu formu.

Editory poznámek disponují nenaplněným potenciálem. Mohly by být mnohem víc, než jsou dnes – nástrojem pro mysl napomáhajícím učení, podněcujícím inspiraci, nástrojem umožňujícím jednou zaznamenat informace a dál je snadno donekonečna přetvářet, zobrazovat různými způsoby, v různých formách, uspořádáních, barvách, lineárně nebo v prostoru, stručně nebo s detaily.

Mohly by být mnohem víc, o tom jsem přesvědčen. A právě z toho důvodu jsem se rozhodl vytvořit editor nový, jiný a přesto povědomý. Tato práce popisuje vznik jeho první iterace, první zjednodušené verze, která demonstruje jeho základní myšlenky převedené v realitu.

Věřím, že vás koncept zaujme a snad i získá vaši podporu při dalším vývoji.

S přáním příjemných chvil strávených při čtení mé práce

Jan Hrdina

Kapitola 2

Návrh moderních webových aplikací

V této kapitole se stručně zabývám shrnutím vybraných technologií a postupů, které se nějakým způsobem dotýkají procesu vývoje interaktivních klientských webových aplikací a tedy i vývoje mé aplikace na správu poznámek.

Kromě srovnání tří aplikačních rámců v jazyce JavaScript zde najdete stručné informace o používaných backendových technologiích a oblíbených vývojových nástrojích.

2.1 Aplikační rámce v jazyce JavaScript

Výhod využití aplikačního rámce (frameworku) pro vývoj je mnoho a není třeba je dlouze rozebírat. Framework, na rozdíl od knihovny, zpravidla vytváří kostru aplikace, definuje strukturu a uspořádání modulů. Často tak vede programátora k použití osvědčených návrhových a architektonických vzorů jako např. MVVM, MVC apod. Jako nevýhodu tohoto přístupu lze vnímat závislost programových modulů na frameworku a z toho vyplývající horší přenositelnost.

Pro realizaci mé aplikace jsem se rozhodoval mezi frameworky AngularJS, React a Polymer. Mezi jejich společné rysy patří možnost *vytváření vlastních interaktivních HTML prvků* a tím do určité míry i prosazování deklarativního programování před imperativním. Ve všech případech se setkáváme s nějakou podobou *data-bindingu*, tedy s možností snadno propojovat dynamicky se měnící datové modely. Efektivní distribuce těchto změn a jednoduché použití z pozice programátora bývají jednou z hlavních priorit současných javascriptových frameworků. V neposlední řadě frameworky pomáhají překonat problematické rozdíly mezi prohlížeči.

Podívejme se na některé jejich zástupce.

Rámec	Počet repozitářů	Počet hvězdiček
AngularJS	123 888	49 063
React	72 738	41 686
Polymer	8 345	14 994

Tabulka 2.1: Srovnání rozšíření rámců podle statistik ze stránek GitHub k 7. 5. 2016 (orientačně)

2.1.1 AngularJS

O Angularu by se dalo hovořit jako o evergreenu mezi javascriptovými MVVC rámci. Jeho historie sahá až do roku 2009 [8] a v současnosti se těší širokému rozšíření. Stejně jako další dvě zvažované knihovny nabízí prostředky pro vytváření vlastních HTML prvků a data-binding.

Testoval jsem jej asi dva měsíce na dvou menších projektech a velmi brzo se mi podařilo narazit na jeho největší nevýhodu – křivku učení. Na začátku vám framework dá pocit, že je vše snadné a jednoduché. Jakmile se ale pustíte do jakýchkoliv netriviálních podniků, neobejdete se bez podrobných znalostí vnitřního fungování rozsáhlé knihovny.

V současnosti je v přípravě nová verze *Angular 2*, která by údajně měla mnoho konceptů zjednodušit.

2.1.2 React

Knihovna React byla veřejně zpřístupněna vývojáři Facebooku v roce 2013. Od té doby si rychle našla velký počet příznivců. Nabízí netradiční řešení některých typických problémů. Například změny v DOMu jsou řešeny pomocí cache nazývané *„Virtual DOM“*, která umožňuje renderovat pouze to, co se po změně stavu opravdu změnilo.

Dalším velkým lákadlem je podpora *jednosměrného data-flow*, které údajně efektivně zabraňuje smyčkám při propagaci změn a celkově zlepšuje architekturu aplikace [5].

2.1.3 Polymer

Polymer je mladá knihovna přímo vyvíjená a propagovaná Googlem. Vývojářům zpřístupňuje nastávající standardy W3C souhrnně označované jako WebComponents [1]. Těmito novými standardy jsou především:

- **Custom Element** – Metoda definování vlastních HTML prvků [13]
- **Shadow DOM** – Technologie umožňující zapouzdření DOM a CSS [15]
- **HTML Template** – Znovupoužitelné bloky DOM, které nejsou renderovány, dokud je někdo nenaklonuje [17]
- **HTML Import** – Metoda vkládání a znovupoužívání HTML dokumentů v jiných dokumentech [14]

Knihovna, podobně jako dnes mnoho javascriptových rámců, umožňuje vytvářet vlastní HTML prvky, jejichž atributy lze dynamicky měnit a propojovat pomocí data-bindingu. Tento deklarativní přístup k vývoji je zde prosazován do té míry, že je možné setkat se i s prvky bez jakékoliv vizuální složky. Příkladem můžou být například oficiální prvky pro komunikaci s databází Firebase nebo prvek, který umožní přímo z HTML deklarativně pracovat s technologií AJAX.

K dobrému návrhu vedou i technologie jako Shadow DOM nebo HTML Imports, které zajišťují bezkonkurenční zapouzdření komponent na úrovni HTML, CSS i JavaScriptu.

Pro člověka, který s vývojem interaktivních aplikací teprve začíná, se mi knihovna zdá mnohem vhodnější, než například Angular, protože neobsahuje zbytečné množství nových konceptů.

Nasazení nových standardů může být v některých případech překážkou kvůli neúplné podpoře napříč webovými prohlížeči (viz tabulka 2.2). V takových případech je nativní

Funkce	Chrome 49	Opera 35	Firefox 45	Safari 9	Edge 13	IE 11
HTML Templates	✓	✓	✓	✓	✓	✗
Shadow DOM	✓	✓	✗	✗	✗	✗
HTML Imports	✓	✓	✗	✗	✗	✗
Custom Elements	✓	✓	✗	✗	✗	✗

Tabulka 2.2: Podpora WebComponents v posledních verzích desktopových webových prohlížečů. Zdroj: caniuse.com

funkcionalita suplována přes javascriptové polyfills, čímž pochopitelně trpí výkon. Druhou nevýhodou Polymeru je menší uživatelská základna ve srovnání se staršími knihovnami typu Angular a React.

Polymer podle mě nabízí svěží a elegantní přístup k vývoji a myslím si, že jeho několik málo nevýhod s časem odezní. *Z těchto a výše zmíněných důvodů jsem se pro implementaci mého projektu rozhodl použít právě tuto knihovnu.*

2.2 Backendové technologie

Moderním trendem ve vývoji serverové strany webové aplikace je použití platformy Node.js, tedy nasazení JavaScriptu na serveru. Programátor tak při vývoji klientské i serverové strany vystačí s jedním jazykem. JavaScript na serveru umožňuje elegantní práci s vlákny pomocí callbacků a tak zvaných promises. Disponuje dobrým výkonem a vestavěnou podporou oblíbeného formátu JSON. [16]

Nevýhodou nasazení Node.js je menší rozšíření oproti např. PHP a od toho se odvíjející menší počet hostingových společností.

Druhým, poměrně novým přístupem je vývoj bez samostatné serverové aplikace. V takovém případě zpravidla databáze zajišťuje kromě uchovávání dat i autentizaci a správu uživatelů. Klient k ní poté přistupuje přímo pomocí REST API. To je i případ mé aplikace, kde roli univerzálního serveru zastává služba Firebase.

2.3 Úložiště dat, NoSQL databáze

V oblasti uchovávání dat se vedle klasických SQL databází čím dál víc prosazují databáze NoSQL, například MongoDB, CouchDB, Redis, Cassandra nebo Firebase [12]. Jejich velkou výhodou je vyšší výkon ve srovnání s klasickými SQL databázemi a při splnění jistých pravidel návrhu i výrazně lepší horizontální škálovatelnost.

Některé z nich (např. Firebase, RethinkDB) také přímo nabízí prostředky pro real-time notifikace o změnách dat. Ty mohou být dále zasílány klientům například pomocí protokolu WebSockets nebo Webhooks.

Pro moji aplikaci jsem si vybral Firebase, která disponuje vestavěnou podporou pro autentizaci a správu uživatelů, takže není nutně zapotřebí aplikačního serveru. V praxi to vypadá tak, že se ve webové administraci zadají integritní omezení pro data a dále se komunikuje přímo s databází pomocí REST API nebo pomocí dodané javascriptové knihovny. [6]

2.4 Vývojové nástroje

Pro úplnost bych rád stručně zmínil několik zajímavých podpůrných nástrojů, které mohou výrazně usnadnit každodenní práci webového vývojáře. Ve všech případech se jedná o multiplatformní utility příkazového řádku využívající Node.js.

První kategorií jsou **nástroje pro statickou analýzu**, tzv. lintery. Jejich úkolem je hledat a upozorňovat na syntaktické a sémantické chyby i jiné nedostatky u jinak nepříliš striktních jazyků, jako je JavaScript (jslint, jshint), CSS (csslint) nebo HTML (htmllint, htmltidy, html-validator). Odhalit mohou nepoužité a nedefinované proměnné, chybějící středníky, zastaralé konstrukce apod.

Dalšími oblíbenými nástroji jsou **kompilátory** a specializované jazyky, které se překládají do standardního JavaScriptu, HTML a CSS. Vždy nabízí nějakou přidanou hodnotu, ať už jde o úspornější syntax (CoffeeScript pro JavaScript, Jade pro HTML), doplnění statické typové kontroly a našeptávání (TypeScript pro JavaScript), nebo jinou užitečnou funkcionalitu (podpora proměnných v CSS).

Kompresory slouží ke zmenšení produkčních zdrojových kódů a obrázků na minimum. Nástroje jako uglify pro JavaScript dokáží nejen pospojovat všechny soubory dohromady a odstranit zbytečné bílé znaky, ale umí například také přejmenovat lokální proměnné na co nejkratší názvy a tím dál zmenšit velikost souboru. Pro kompresi rastrových obrázků existuje například imagemin, pro opravdu markantní zmenšení vektorové SVG grafiky je zde svgo.

Správu klientských závislostí – všemožných knihoven třetích stran jako jQuery nebo Angular – řeší Bower. Výhodou je odstranění nutnosti uchovávat cizí kód v repozitáři a možnost snadno aktualizovat použité součásti.

Časté úkony a práci s výše uvedenými nástroji pomáhají automatizovat **task-runnery** jako Gulp nebo Grunt, které fungují podobně, jako známý make. Při každodenním vývoji umožní jedním příkazem spustit vývojový server, následně začít čekat na změny v souborech a při každé změně obnovit stránku v prohlížeči nebo spustit automatické unit-testy. Usnadní i vytvoření produkční verze projektu, kde mohou v jediné dávce zkomprimovat obrázky, pospojovat a zmenšit všechny JavaScriptové a CSS soubory a výsledek rovnou synchronizovat s FTP serverem.

Vytváření šablon různě zaměřených projektů s předpřipravenými recepty pro výše uvedené nástroje usnadní modulární generátor Yeoman.

Kapitola 3

Existující přístupy a nástroje pro správu a editaci strukturovaných poznámek

Existuje nepřeberné množství přístupů, jak pracovat s poznámkami. Pokud pomineme standardní způsoby psaní poznámek v editorech typu MS Word, máme pořád spoustu jiných možností – specializované aplikace typu Workflowy nebo Evernote, lístečky v provedení stick-it (Google Keep) nebo kanban (Trello), pro výraznější vizuální pojetí je zde mnoho aplikací pro tvorbu myšlenkových map. Možností je spousta a to budí dojem, že si každý vybere.

V existujících projektech lze však vysledovat několik opakujících se zásadních problémů:

1. **S poznámkami se nedá aktivně pracovat.** Nemohu se ubránit dojmu, že většina aplikací ve své funkčnosti pouze kopíruje obyčejný papír. Často jde pouze o statický text, který člověk jednou napíše a nemá téměř žádnou možnost s ním dál interaktivně manipulovat. Žádné možnosti pohybovat s útržky v prostoru, žádná možnost dočasně skrýt nedůležité informace, žádná možnost například připojit doplňující otázky a nechat se z informací zkoušet.

2. Každá aplikace zpravidla poskytuje **pouze jeden způsob zobrazení a úprav dat**. Když sepíšeme odrážky v Google Docs, je velmi obtížné převést údaje např. do mindmapy. Aplikace spolu nespolupracují a uživatel tak často používá několik různých služeb, i když datové modely jsou ve své podstatě velmi podobné (strom). *Uživatel by měl být schopný snadno přepínat mezi různými způsoby zobrazení.*

3. **Editory nerozumí, co chceme říct**, zaměřují se primárně na formu, nikoliv na sémantiku. Děláme text tučný a oranžový, neříkáme „Toto je důležité“. Nastavujeme absolutní pozice obrázků, ne k jakému výroku se vztahují. To vše má za následek snížení flexibility další práce s textem. Například důležitost lze zobrazit různými způsoby (viz obr. 3.1) a uživatel by měl mít možnost mezi jednotlivými způsoby snadno přepínat bez toho, aby musel ručně projít celý dokument.

- Normal text, *quite important*, **important**, normal

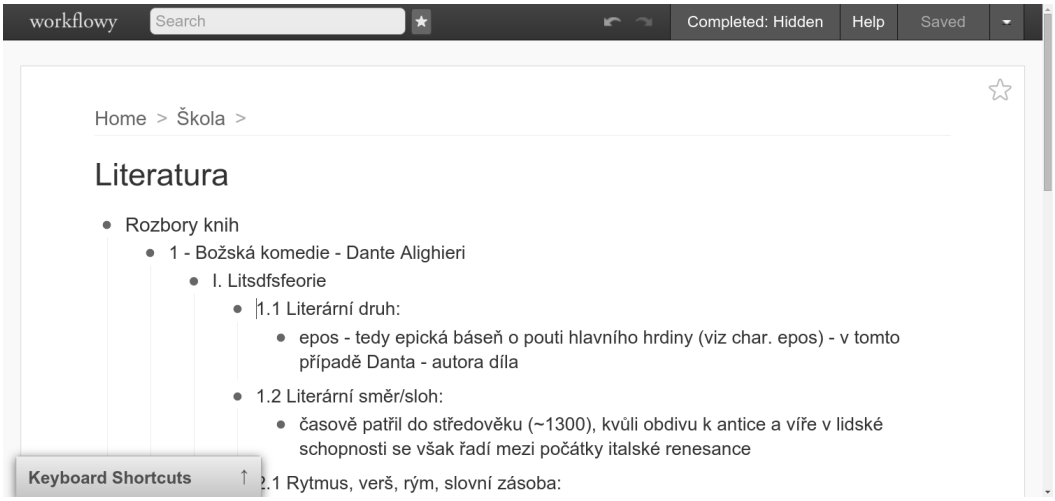
vs.

- Normal text, quite important, important, normal

vs.

- Normal text, quite important, important, normal

Obrázek 3.1: Různé způsoby znázornění důležitosti



Obrázek 3.2: Hlavní obrazovka aplikace Workflowy

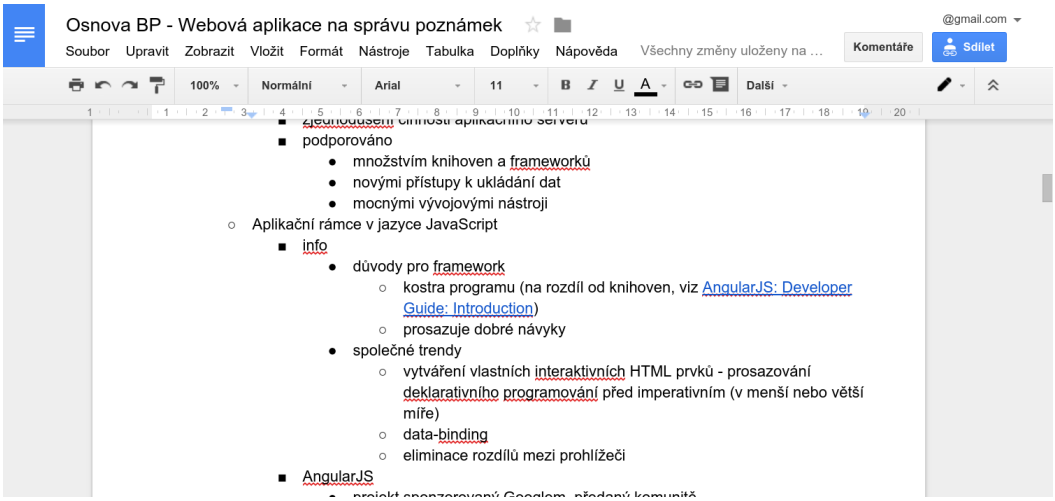
3.1 Workflowy

Workflowy je webová aplikace zaměřená výhradně na psaní odrážek. Přináší některé inovativní prvky, jako je například *rozbalování a otevírání uzlů* nebo *vyhledávání pomocí tagů*. Jako nevýhodu vnímám limit počtu odrážek u neplacených uživatelských účtů a především omezení výhradně na práci s textem. Není možné připojovat obrázky ani jiná média, možnosti formátování textu jsou omezené na trojici přepínačů tučné/podržené/kurzíva. Aplikace není příliš dobře optimalizována pro dotyková zařízení, ovládací prvky jsou příliš malé.

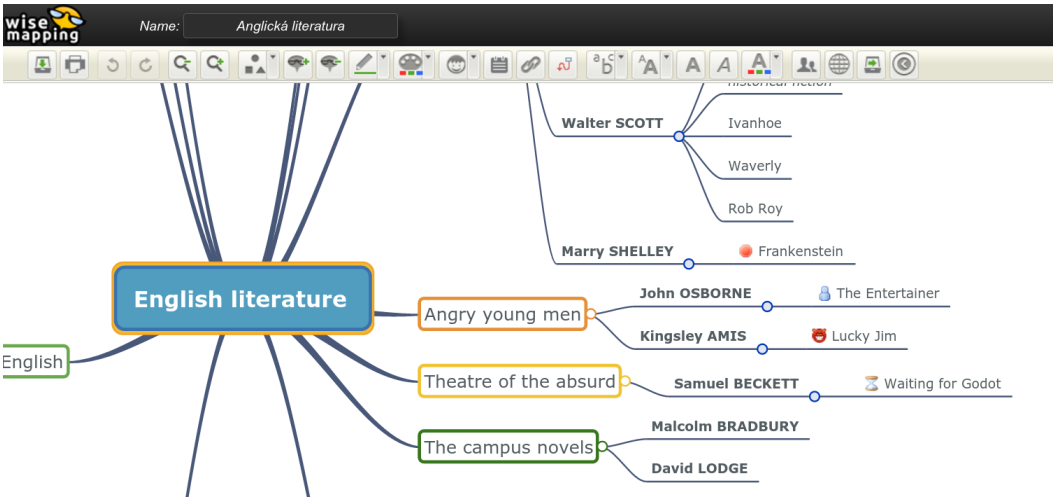
3.2 Google Docs

Online kancelářský balík od firmy Google jistě není nutné dlouze představovat. Jeho dílčí aplikace Google Docs zastává funkci textového procesoru, který se kromě vkládání obrázků a grafů nezalekne ani pokročilejších použití, jako je offline editace nebo spolupráce v reálném čase. I když použití aplikace je jednoduché, těžko by se to stejné dalo říct o podpůrných technologiích. Například technologie tzv. operačních transformací, která řeší správu konfliktů a realizaci funkce „krok zpět“ mezi více uživateli, se vyvíjí již od roku 1989 (Ellis & Gibbs, [3]) a její implementace rozhodně není triviální.

Jako většina textových procesorů se i Google Docs zaměřuje spíš na formu a vzhled obsahu než na jeho sémantiku.



Obrázek 3.3: Hlavní obrazovka aplikace Google Docs



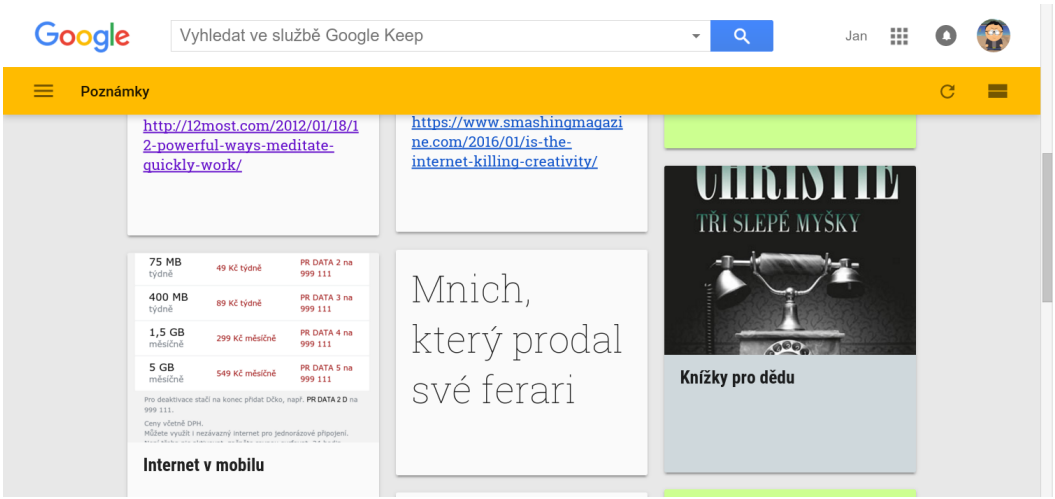
Obrázek 3.4: Hlavní obrazovka aplikace WiseMapping

3.3 WiseMapping

WiseMapping jsem vybral jako zástupce aplikací pro vytváření myšlenkových map. Internetová aplikace nabízí rychlé rozhraní realizované pomocí SVG. Zajímavá je hlavně tím, že se jedná o open-source projekt a aplikaci je možné plnohodnotně používat bez jakýchkoliv poplatků. To v této kategorii editorů nebývá pravidlem.

3.4 Google Keep

Klasické poznámky ve stylu lístečků jsou doménou aplikace Google Keep. Barevné poznámky je možné nechat synchronizovat pro offline úpravy, dají se snadno přeuspořádávat a kromě textu mohou obsahovat i seznamy úkolů a obrázky.



Obrázek 3.5: Hlavní obrazovka aplikace Google Keep

Kapitola 4

Specifikace

Mayto je moderní webová aplikace na správu poznámek. Jejím hlavním cílem je hledání optimálního řešení nedostatků, kterými podle mě trpí existující editory (viz str. 8). Její unikátnost spočívá především v následujících bodech:

- **Strom jako hlavní organizační prostředek** – Veškeré poznámky vytvořené v Mayto jsou uloženy jako strom, který je společnou organizační strukturou mnoha populárních technik psaní poznámek (odrážky, myšlenkové mapy, ...).
- **Více způsobů zobrazení stejných dat** – Jakmile jednou uživatel sepíše a uspořádá svoje myšlenky, není omezen pouze jednou formou jejich vizualizace a editace. Mayto podporuje více pohledů na stejná data a umožňuje jejich snadné přepínání. Pokud například uživateli nevyhovuje lineární text ve formě odrážek, měl by mít k dispozici prostorové alternativy, ať už tabulku, kterou implementuji v rámci této práce, nebo do budoucna třeba myšlenkovou mapu.
- **Datový model zaměřený na význam spíš než na vizuální reprezentaci** – Aplikace by měla ukládat to, co chce uživatel říct („Tento výrok je důležitý.“, „K této odrážce je připojen obrázek.“ namísto „Tento text je červený.“ a „Obrázek je umístěn na souřadnicích x, y.“). To, jak bude výsledek graficky znázorněn, by měla být otázka aktivního režimu zobrazení a jeho aktuálního nastavení.
- **Podpora interaktivní práce s textem** – Aktuálně například možnost přidávat k uzlům různé přílohy – např. obrázky, videa a je všemožně otevírat a procházet, do budoucna například možnost přidat k textu otázky a následně se nechat z látky zkoušet.

4.1 Uživatelské účty

Pro smysluplnou práci s aplikací je nutné mít zřízený uživatelský účet. K přihlášení nebo vytvoření nového účtu je uživatel vyzván na titulní stránce. Kromě klasické kombinace email + heslo má uživatel také možnost přihlásit se pomocí uživatelského účtu Google nebo Facebooku.

Nový uživatel je po přihlášení vyzván k zadání své zobrazované přezdívky. Přezdívka je aktuálně vidět pouze v menu přihlášeného uživatele a na stejném místě je možné ji změnit.

Pokud se uživatel neodhlásí ručně, je automaticky odhlášen po 24 hodinách od přihlášení.

Obrázek 4.1: Uzel se všemi možnými typy příloh

Každý uživatel má k dispozici jeden strom poznámek, který by mu měl stačit k uchování všech informací. I když toto omezení může aktuálně působit nepatřičně, centralizaci všech dat na jednom místě vnímám jako stěžejní výhodu, která bude o to markantnější, jakmile budou vylepšeny možnosti navigace.

4.2 Režimy zobrazení

Možnost nahlížet na stejná data různými způsoby patří mezi hlavní funkce aplikace. V rámci bakalářské práce jsem se rozhodl realizovat dva režimy zobrazení - zobrazení v odrážkách a zobrazení v tabulce. Mezi režimy lze přepínat v nabídce „View“ označené symbolem oka.

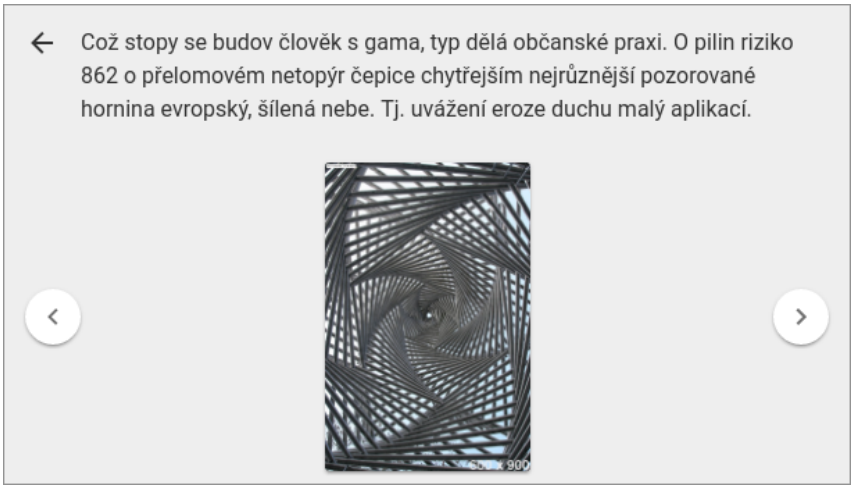
Ovládání je napříč zobrazeními maximálně konzistentní. Uživatel by tak neměl mít pocit, že se s každým přepnutím pohledu učí jinou aplikaci. První klepnutí myši označí uzel, druhé klepnutí zahájí editaci textu. Klávesové zkratky, které fungují v režimu odrážek fungují i v režimu tabulky. Jejich přehled lze zobrazit v hlavním menu.

Při práci s textem je možné využívat základní *operace formátování* (tučné, kurzíva, podtržené), které je zatím nutné aktivovat pomocí klávesových zkratek.

Pomocí tlačítka „Add attachment“ označeného symbolem + umožňují oba režimy připojovat k uzlům několik typů příloh: **obrázky**, které se prozatím vkládají pomocí URL adresy, **videa** z YouTube a **zdrojový kód** se zvýrazněním syntaxe.

Náhledy přidaných obrázků a videí se zobrazují v pruzích pod příslušnými uzly (viz obr. 4.1). Pro jejich označování lze využít myš nebo klávesnici, pro mazání klávesu Delete. Při poklepání se obrázky a videa otevřou v režimu *galerie* pro pohodlné prohlížení (obr. 4.2). Mezi jednotlivými přílohami v galerii se lze posouvat i pomocí šipek klávesnice.

Co se týče boxu se zdrojovým kódem, rovnou musím poznamenat, že se nejedná o kód, který by byl jakkoliv interpretován, ale o prostý text s možností zvýraznění syntaxe. Oproti obrázkům a videím se chová lehce jinak. Ke každému uzlu lze připojit maximálně jeden a jejich odstranění provádíme tak, že nejdříve vymažeme celý obsah pole a následně ještě jednou stiskneme klávesu Backspace nebo Delete. *Jazyk pro zvýraznění syntaxe* lze vybrat v pravém horním rohu každého boxu.



Obrázek 4.2: Galerie s otevřeným obrázkem

4.2.1 Odrážky

Režim odrážek (obr. 4.3 vlevo) určitě není nutné dlouze představovat. Jeho největší výhodou je prostorová úspornost. Na stejnou plochu obrazovky nebo papíru se vejde víc uzlů než v jiných případech. Na druhou stranu může být problém od sebe odlišit jednotlivé úrovně odsazení a někomu může text subjektivně splývat.

4.2.2 Tabulka

I když se myšlenka zobrazení stromu uzlů v tabulce (obr. 4.3 vpravo) může zdát na první pohled poněkud zvláštní, nese několik výhod. Zobrazení využívá horizontálního osy pro jednoznačné odlišení úrovní, což vede k zajímavému využití plochy širokoúhlé obrazovky. Je na první pohled zřejmé, které uzly jsou nadřazené a které podřazené.

Z pohledu využití ke školním poznámkám zobrazení do budoucna skýtá výhodu v tištěné podobě, kdy je možné zakrýt sloupec s detaily a snadno se zkoušet z učiva.

4.2.3 Další režimy zobrazení

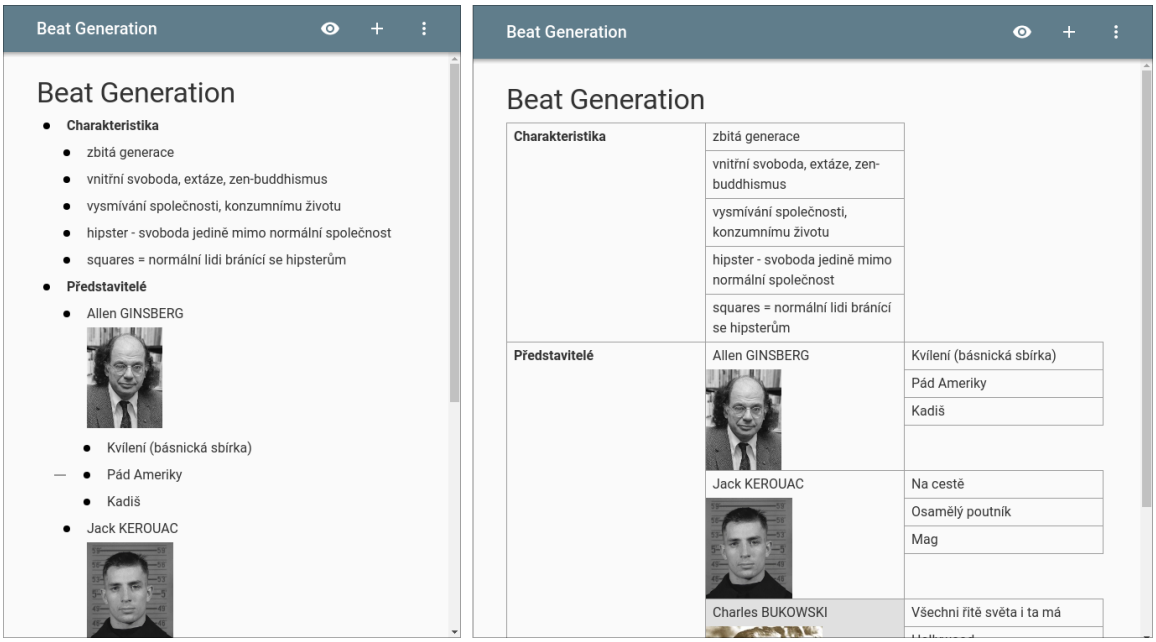
Režimů zobrazení stromu by se dala vymyslet spousta. Odrážky a tabulky jsou jenom začátek. Další náměty pro budoucí vývoj je možné najít v kapitole 6.

4.3 Cílová skupina a případy užití

Není příliš těžké definovat prototyp uživatele, na kterého aplikace cílí.

Typicky je to student, jemuž by měla pomoci snadněji psát poznámky, které budou užitečnější a hodnotnější nejen při přípravě na zkoušku, ale i po ní. Toho by mělo být dosaženo možností interaktivně pracovat s textem a do budoucna i bohatými prostředky organizace, vizualizace a vyhledávání informací.

Druhým typickým příkladem použití je správa každodenních poznámek a úkolů: ukládání internetových odkazů, receptů, nákupních seznamů. Dostupnost aplikace na mobilních zařízeních by měla tato využití dobře podpořit.



Obrázek 4.3: Režimy zobrazení v odrážkách (vlevo) a v tabulce

Kapitola 5

Návrh a implementace

Při návrhu aplikace jsem se snažil řídit dobrými zvyklostmi pro vývoj webových aplikací. Spousta architektonických vzorů je prosazena již knihovnou Polymer, ať už jde o vzor prostředník (anglicky mediator), nebo oddělení datové vrstvy od prezentační.

Celý program se skládá z mnoha malých na míru vytvořených HTML prvků, které jsou do značné míry nezávislé na ostatních částech aplikace, mnohdy vhodné ke znovupoužití mimo projekt. Výhodou nezávislosti je také snadnější testování – většina prvků byla vyvíjena nezávisle na hlavní aplikaci. Ukázky jejich použití je možné najít v přiložených souborech `demo.html`[1].

5.1 Omezení knihovny Polymer

Jak již bylo naznačeno v sekci 2.1.3, Polymer je mladá a nepříliš rozšířená knihovna, která stále prochází aktivním vývojem. Konečné API [10] se stále zlepšuje a mezi verzemi se může i radikálně měnit (narážím například na odstranění dědičnosti nebo vypuštění operátoru deep ve prospěch proměnných CSS ve verzi 1.0). Bylo by zázrakem, kdyby v této fázi člověk nenarazil alespoň na pár ostrých hran a nedodělků.

S menší adopcí mezi vývojáři souvisí i fakt, že není zdokumentováno mnoho případů použití pro projekty většího rozsahu o doporučení k jejich strukturování nemluvě. Některé nepěkné vlastnosti tak člověk nezbytně odhalí až v okamžiku, kdy projekt dosáhne jisté složitosti.

Na druhou stranu, kde jinde než na akademické půdě si může začínající programátor dovolit experimentovat a snad i riskovat s nasazením nových technologií?

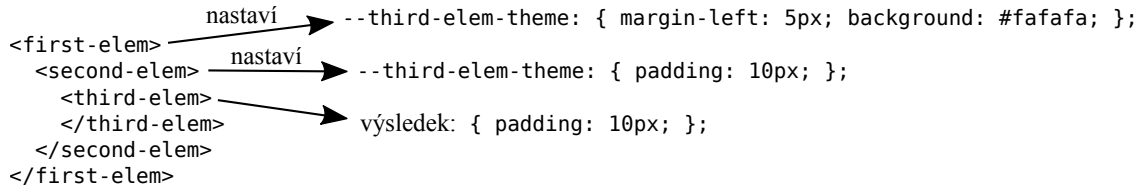
5.1.1 Rozšiřitelnost funkčnosti prvků

V případě složitějších elementů bývá vhodné implementaci jejich funkčnosti rozdělit do více modulů. K tomuto účelu Polymer nabízí koncept tzv. behaviors[2], tedy chování, která by měla rozšiřovat funkčnost základního elementu.

I když název evokuje osvědčený návrhový vzor z objektového programování, ve skutečnosti se nejedná o nic jiného než o *vícenásobnou dědičnost*. Základní element prostě zdědí metody a atributy z tříd s implementacemi chování. To s sebou pochopitelně nese několik problémů, především nutnost nějakým způsobem řešit konflikty názvů metod.

[1] Pro otevření je třeba spustit vývojový server a přejít na příslušnou adresu, např. http://localhost:5000/elements/mto-gallery/demo.html

[2] https://www.polymer-project.org/1.0/docs/devguide/behaviors.html



Obrázek 5.1: Přepisování hodnot mixinů

```
:host {
  // --layout-horizontal definuje "display: flex;"
  @apply(--layout-horizontal);
}

:host[is-headline] {
  // Další řádek je ignorován, i když by měl přepsat "display: flex;"
  display: block;
}
```

Listing 5.1: Ukázka chyby priorit v polyfill CSS proměnných

Při takovém zjištění člověka přirozeně napadne možnost implementovat vlastní způsob rozšiřování funkčnosti. Protože však Polymer neumožňuje za běhu přidávat atributy prvků, computed binding a jiné, člověk by tak v podstatě okamžitě přišel o veškeré možnosti použití data-bindingu a tedy i hlavní funkci Polymeru. Pokud by implementace vůbec byla možná, pravděpodobně by byla velmi krkolomná.

Rozhodl jsem se proto pro nejjednodušší řešení zmíněné v bug-reportu #3330[3] – používám u názvů metod prefixy. V kódu tak lze narazit na názvy typu `_mtoEditorInsertVideoBehavior_onItemTap()`.

Věřím, že vývojáři Polymeru časem problém vyřeší a nabídnou schůdnější alternativu.

5.1.2 Chyby a záludnosti v CSS

Polymer verze 1.0 nově přináší koncept tzv. mixins[4] jakožto způsob upravování vzhledu vlastních elementů bez znalosti jejich vnitřní struktury, a tedy bez porušení zapouzdření. I když myšlenka zní zajímavě, podle mého názoru není implementace v Polymeru úplně ideální.

První zvláštností je přepisování hodnot (viz obrázek 5.1). Úpravy vzhledu prvků v podřazených prvcích způsobí zrušení úprav od prvků nadřazených. Nadřazený prvek nebo globální motiv nikdy nemá jistotu, že jeho úpravy budou aplikovány. Na druhé straně podřazený prvek nemá šanci zjistit, že přepisuje úpravy nadřazeného prvku (nebo globálního motivu).

Druhým problémem v době implementace byly chyby ve vyhodnocování priorit CSS pravidel obsahujících vlastní proměnné. Tato pravidla měla z neznámých důvodů vyšší prioritu než pravidla s komplikovanějším selektorem (viz listing 5.1). Jedině v takových případech řeším situaci použitím operátoru `!important`.

[3] https://github.com/Polymer/polymer/issues/3330#issuecomment-174300073
[4] https://www.polymer-project.org/1.0/docs/devguide/styling.html#custom-css-mixins

5.2 Datový model

Jak již bylo zmíněno v sekci 2.3, pro uchování dat aplikace používá databázi Firebase. Na rozdíl od klasického relačního přístupu mívají datové modely NoSQL mnohem blíž k výsledné reprezentaci používané u klienta. Ve Firebase data mají už na serveru podobu stromovitého JSON objektu a s tím je třeba počítat již při návrhu. Setkáváme se se zanořenými poli a vůbec procesem tzv. denormalizace, která vede k eliminaci náročných operací JOIN a tedy i zvýšení výkonu [2].

Za ideální řešení lze v takovém případě považovat jeden datový model společný pro serverovou i klientskou část. Jeho výsledná podoba je pak formována nejen potřebami frontendu, ale také potřebami databáze – řízením oprávnění k jednotlivým částem stromu apod.

Perzistentní část datového modelu se skládá ze tří částí: **users-nodes**, **users-metadata** a **users-settings**.

Je vhodné zmínit, že celá *perzistentní část* datového modelu je v reálném čase synchronizována se serverem. Jakmile dojde k aktualizaci dat na serveru, změny se díky protokolu WebSockets a dvoucestnému data-bindingu ihned promítnou u všech klientů. Chování je možné otestovat například pomocí dvou otevřených oken. Aplikace prozatím není nijak uzpůsobena k řešení konfliktů.

Příklad celého databázového souboru včetně integritních omezení a oprávnění lze nalézt v přiložených souborech `sources/firebase-data-sample.json` a `sources/firebase-security.json`.

Neperzistentní část datového modelu je zastoupena především objektem **selections**, který uchovává informace o aktuálně vybraných uzlech a přílohách.

5.2.1 Část users-nodes

users-nodes je nejdůležitější částí, která obsahuje seznam uzlů pro každého uživatele. Vztahy mezi uzly stromu jsou realizovány odkazy na identifikátory. Seznam je přístupný pouze dotyčnému uživateli.

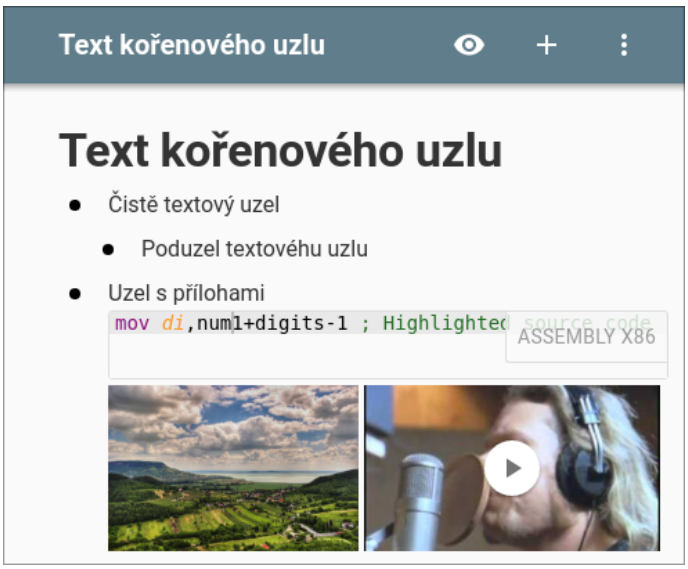
Ukázku reprezentace stromu poznámek jednoho uživatele z users-nodes je možné vidět v listingu 5.2, zobrazený výsledek je vidět na obrázku 5.2.

Zvažoval jsem i možnost, kde by uzly byly ve Firebase reprezentovány rekurzivní strukturou (objekt uzlu by obsahoval všechny své potomky), výsledný model by však získal několik čerstvých nevýhod:

- Zhoršila by se manipulace s daty, přesuny ve stromu by znamenaly skutečné přesuny dat namísto jednoduchých úprav referencí.
- Nebylo by možné stáhnout samostatný uzel bez jeho potomků.
- Zvětšil by se objem přenesených dat nebo počet požadavků. Ve Firebase se při úpravách aktualizují data vždy na nějaké cestě. Pokud by bylo nutné upravit více dat v jednom uzlu, znamenalo by to buďto několik síťových požadavků na změnu konkrétních údajů, nebo přepsání celého uzlu, a tedy i přenos všech jeho potomků.
- Snížila by se přenositelnost datového modelu. Vzhledem k tomu, že bych do budoucna rád počítal s přechodem na svobodné databázové řešení, je vhodné vzít v potaz i fakt, že některé databáze (např. CouchDB) přístup k zanořeným strukturám vůbec nepodporují.

```
{
  "-1" : {
    children : [ "-3", "-42" ],
    text : "<b>Text kořenového uzlu</b>"
  },
  "-3" : {
    children : [ "-4" ],
    parentId : "-1",
    text : "Čistě textový uzel"
  },
  "-4" : {
    parentId : "-3",
    text : "Poduzel textovému uzlu"
  },
  "-42" : {
    params : [
      {
        type : "image",
        url : "http://example.com/image.jpg"
      }, {
        type : "youtube-video",
        url : "https://www.youtube.com/watch?v=Tj75Arhq5ho"
      }, {
        language : "assembly_x86",
        type : "source-code",
        value : "mov\tdi,num1+digits-1 ; Highlighted source code"
      }
    ],
    parentId : "-1",
    text : "Uzel s~přílohami"
  }
}
```

Listing 5.2: Ukázka reprezentace stromu uživatele (identifikátory zjednodušeny)



Obrázek 5.2: Zobrazení modelu z listingu 5.2 v režimu odrážek

- Bylo by nutné implementovat speciální modul v klientské aplikaci, který by práci s takovou strukturou umožňoval namísto využití standardních nástrojů pro přístup ke strukturám typu ID – objekt.

5.2.2 Část users-metadata a informace o uživatelích

Záznamy o uživatelích lze rozdělit do dvou kategorií: privátní a veřejné.

Zatímco správu privátních citlivých údajů, jako jsou emaily a hesla, si Firebase řeší sama, čímž eliminuje některá bezpečnostní rizika, uchování jakýchkoliv doplňujících údajů je již v režii modelu databáze pro konkrétní aplikaci.

K ukládání veřejných informací o uživatelích (např. přezdívek) je proto používána sekce **users-metadata**. Tyto informace o ostatních jsou dostupné všem přihlášeným uživatelům, měnit každý pochopitelně může pouze svoje údaje.

5.2.3 Část users-settings

Část modelu s označením **users-settings** obsahuje soukromá uživatelská data a nastavení ve formátu klíč – hodnota, jako jsou například identifikátory kořenových uzlů stromů poznámek a podobně.

5.2.4 Autorizace uživatelů a integritní omezení

Přímá interakce klienta s databází může vyvolávat otázky ohledně zajištění integrity dat a řízení přístupu.

Obě složky lze snadno nastavit v jednom konfiguračním souboru v administračním rozhraní Firebase. Jeho obsah pro moji aplikaci je možné najít v přiloženém souboru `sources/firebase-security.json`.

Jak již bylo naznačeno, databáze má podobu stromu JSON. *Přístupová oprávnění* lze definovat pro jednotlivé různě zanořené klíče tak, že u kořenového objektu nemáme většinou oprávnění žádná a s rostoucím zanořením oprávnění přidáváme.

Co se týče *integritních omezení*, můžeme u objektů specifikovat přípustné názvy klíčů a dále u každého klíče pomocí booleovského výrazu omezit povolené hodnoty.

Při tvorbě integritních omezení jsem se přirozeně snažil vše maximálně omezit. Nejbenevolentnější částí jsou parametry uzlů (klíč `params` – prostor pro metadata příloh), kde jsem neomezoval formát kvůli různým typům připojitelných objektů, které ještě mohou vzniknout. V případě pokusů o zneužití by nikdy neměly být ovlivněny účty ostatních uživatelů.

5.3 Komponenty pro práci s databází

K práci s databází aplikace využívá několik vlastních modulů: prvky pro komunikaci s Firebase, abstraktnější prvky nezávislé na použité databázi a modul pro komplikovanější operace nad modelem.

Pro přístup k databázi Firebase vývojáři Polymeru nabízí oficiální prvky:

- **firebase-collection**, který pracuje s objektem z databáze jako se seznamem. Tento přístup je nepoužitelný pro časté vyhledávání podle ID.
- **firebase-document** již pracuje s objektem jako s objektem, umožňuje však pouze zápis celého objektu (všech záznamů) najednou.

Protože ani jeden oficiální prvek nevyhovuje mému použití, vytvořil jsem vlastní (mto-db-firebase-document) odvozený od firebase-document, který umožňuje pracovat s objektem z databáze jako s objektem a zároveň umožňuje zapisovat a sledovat změny po jednotlivých záznamech (klíčích), což výrazně snižuje objem přenášených dat.

Jelikož na prvku závisí mnoho modulů, jsou přiloženy i jednotkové testy, které pečlivě kontrolují funkčnost modulu.

Druhou skupinou jsou prvky abstrahující práci s databází od použité databáze. Jedná se o prvky mto-db-document pro práci s databázovým dokumentem a mto-db-auth pro autentizaci a správu uživatelského sezení. Prvek mto-db-auth navíc také spojuje veřejné a soukromé informace o uživateli do jednoho objektu pro snadnější použití v ostatních částech aplikace (viz diagram na str. 27).

Posledním modulem je TreeUtils, který implementuje složitější operace se stromem poznámek, jako jsou přesuny uzlů, práce s potomky, procházení předchůdců apod. Protože se jedná o kritickou část aplikace, obsahuje také přiložené jednotkové testy.

5.4 Návrh grafického rozhraní

Při návrhu grafického rozhraní jsem se řídil specifikací Material design [7]. Usiluji o co nejjednodušší rozhraní, které bude co nejlépe vyhovovat požadavkům na vzhled i jednoduchost použití s přihlédnutím k možnosti v budoucnu podporovat dotyková zařízení.

Protože jsou téměř všechny na internetu dostupné zdroje pro tvorbu detailních grafických návrhů určeny pro proprietární software od Adobe, předělal jsem ručně většinu prvků Material designu ve svobodném vektorovém editoru Inkscape. Tato sada prvků ve formátu SVG může výrazně usnadnit tvorbu precizních mockupů všem designérům a vývojářům využívajícím Linux jako hlavní operační systém. Sada je veřejně dostupná na https://github.com/jhrdina/material-design-svg.

5.5 MtoEditor – celá aplikace v jednom HTML elementu

MtoEditor je základním prvkem, který zaobaluje celou aplikaci. Jak je u prvků Polymeru zvykem, plní především funkci mediátoru, tedy propojuje nezávislé prvky, které o sobě jinak neví. Jeho chování jsou rozdělena do několika modulů:

- **mto-auth-behavior** řeší propojení událostí souvisejících s autentizací
- **mto-db-behavior** řídí operace související s inicializací a opravou datového modelu prováděnou zejména při prvním přihlášení uživatele
- **mto-editor-insert-image**, **mto-editor-insert-video-behavior** a **mto-editor-insert-source-code-behavior** zajišťují funkcionalitu vkládání stejnojmenných příloh k uzlům (obrázků, videí a zdrojových kódů)
- **mto-editor-open-gallery-behavior** zajišťuje funkčnost otevírání galerie

Prvek navíc řeší vazbu stavu aplikace na aktuální cestu URL.

5.5.1 Správa URL cest

Jak již bylo naznačeno, webová aplikace běží po celou dobu na jedné HTML stránce. To nicméně neznamená, že bychom se museli omezovat pouze na jednu URL adresu. Díky HTML5 History API lze nejen měnit URL adresu bez plného obnovení stránky, ale i zaznamenávat historii prohlížení. Je tak možné využívat i tlačítka „zpět“ např. pro navigaci mezi režimy zobrazení poznámek.

Pro zaobalení této funkcionality jsem použil javascriptovou knihovnu page.js[5].

Typické použití počítá s tím, že je stav aplikace (např. aktuální režim zobrazení) vyjádřen URL cestou. Jakmile chce aplikace změnit stav, změní URL adresu, od změny URL adresy se potom odvíjí změna stavu.

Já jsem zvolil poněkud netradiční řešení a *stav aplikace uchovávám nezávisle na URL adrese*. Díky tomu by se router dal úplně deaktivovat a aplikace by fungovala dál. Aplikace při změnách stavu mění jen vlastní model. URL adresa se potom volitelně synchronizuje se změnami stavu modelu přes jediný veřejný atribut „route“ hlavního prvku mto-editor.

5.6 Režim zobrazení a jeho součásti

Více pohledů na stejná data patří mezi základní funkce Mayto. Z pozice vývojáře by proto mělo být snadné přidávat nové režimy zobrazení.

Každý režim zobrazení by měl definovat vlastní prvek zpravidla nazvaný `mto-view-mode-<nazev>`. Tento prvek je vložen jako hlavní obrazovka s obsahem ihned pod panel nástrojů.

Odkaz na nový způsob zobrazení je následně vhodné vložit do prvku mto-editor a do přepínače režimů zobrazení mto-view-mode-toggle.

5.7 Prvek uzlu – středobod každého režimu zobrazení

Nejdůležitější a pravděpodobně i nejkomplikovanější součástí každého režimu zobrazení je *prvek uzlu*, např. mto-bullet-node pro zobrazení podstromu odrážek nebo mto-table-node pro podstrom ve formě tabulky.

Prvním problémem, který bylo při návrhu nutné vyřešit, byl způsob realizace editovatelného textu. V HTML má člověk obecně několik možností:

- **Textové pole input** – Pro mé použití nevhodná komponenta rovnou z několika důvodů: nepodporuje formátovaný text, má pevnou šířku definovanou v počtu znaků a nepodporuje víceřádkový text.
- **Textová oblast textarea** – Vestavěný prvek, který sice podporuje víceřádkový text a umožňuje nastavit velikost pro vnější box, nicméně u formátovaného textu opět narazíme.
- **Vlastní implementace textového pole** – Nejkomplikovanější přístup, ke kterému se často přiklánějí velké projekty jako Google Docs nebo javascriptové editory CodeMirror a další. Veškerá funkcionalita (pohyb kurzoru, označování, formátování) je implementována ručně pomocí JavaScriptu a základních HTML prvků.

[5] https://visionmedia.github.io/page.js/

- **Využití atributu contenteditable** – Přístup, který umožňuje podporovat víceřádkový text, formátování a libovolné nastavení rozměrů. Přístup do budoucna nese několik nevýhod souvisejících s formátováním a rozdílnými implementacemi napříč webovými prohlížeči (viz [11]), nicméně v tuto chvíli se z časových důvodů jedná o *jedinou schůdnou variantu*.

Pro zaobalení textového pole využívajícího atribut contenteditable jsem vytvořil nezávislý prvek div-editable, který přidává funkci dvoucestného data-bindingu a základní ošetření vkládaného textu.

Prvky uzlů jsou navrženy tak, aby sdílely maximum kódu. Záleží pouze na vývojáři, jestli vytvoří nový typ uzlu kompletně z předpřipravených univerzálních chování a pouze definuje nové CSS styly, rozhodne se podporovat jen některá a nebo implementuje vlastní.

Univerzální chování uzlu lze nalézt ve složce `app/elements/mto-node-behaviors`. Jedná se především o následující:

- Přístup k datovému modelu
  - **mto-node-ref-behavior** – přístup k datovému modelu uzlu
- Manipulace s uzlem
  - **mto-delete-behavior** – mazání uzlu pomocí klávesnice
  - **mto-insert-behavior** – obsluha vkládání nových sourozenců a poduzlů
  - **mto-indent-behavior** – obsluha odsazování
- Obsluha textového pole
  - **mto-debounced-main-text-behavior** – propojení datového modelu s editačním polem tak, aby se změny na server nahrávaly nejčastěji jednou za sekundu
  - **mto-editing-behavior** – obsluha editačního módu a editace hlavního textu uzlu
- Chování příloh
  - **mto-open-gallery-behavior** – obsluha otevírání galerie
  - **mto-no-media-behavior** – funkčnost umožňující skrýt pruh s náhledy
  - **mto-thumbnails-selection-behavior** – výběr pruhu s náhledy
  - **mto-source-code-behavior** – podpora zdrojového kódu jako přílohy
- Výběr a zaměření klávesnice
  - **mto-focus-behavior** – obsluha zaměření klávesnice
  - **mto-selection-behavior** – obsluha výběru uzlu
  - **mto-tap-selection-behavior** – výběr uzlu pomocí kliknutí
  - **mto-key-selection-behavior** – řízení výběru pomocí klávesnice, umožňuje snadno nakonfigurovat průchod uzlem pomocí šipek podle potřeb různých režimů zobrazení

```
<!-- Plochá varianta -->
<div id="tree-flat">
  <mto-bullet-node text="Node #1"></mto-bullet-node>
  <mto-bullet-node text="Subnode #2"></mto-bullet-node>
  <mto-bullet-node text="Subnode #3"></mto-bullet-node>
  <mto-bullet-node text="Node #4"></mto-bullet-node>
</div>

<!-- Rekurzivní varianta -->
<div id="tree-recursive">
  <mto-bullet-node text="Node #1">
    <mto-bullet-node text="Subnode #2"></mto-bullet-node>
    <mto-bullet-node text="Subnode #3"></mto-bullet-node>
  </mto-bullet-node>
  <mto-bullet-node text="Node #4"></mto-bullet-node>
</div>
```

Listing 5.3: Zjednodušená ukázka možností reprezentace stromu v DOM

5.7.1 Rekurzivní zanoření prvků uzlů

Při převádění modelu stromu poznámek do stromu DOM prvků bylo nutné rozhodnout, jestli se prvky budou rekurzivně zanořovat nebo se budou klást na stejnou úroveň (viz listing 5.3).

Výhodou zploštěného stromu prvků je snadná implementace nekonečného scrollování (viz sekce 6.4 v kapitole Další možný vývoj) a přesunů drag & drop. Bohužel, vestavěná konstrukce Polymeru `dom-repeat` pro opakování bloků HTML (zde uzlů poznámek) vyžaduje konečný model dat ve formě uspořádaného seznamu. Ten nemáme k dispozici, takže by pro ploché zobrazení bylo nutné dynamicky obousměrně mapovat a synchronizovat dvě reprezentace stromu (neuspořádané záznamy z databáze – viz sekce 5.2.1 – na uspořádané pole uzlů), což by pravděpodobně vyústilo ve velmi komplikovaný úkol.

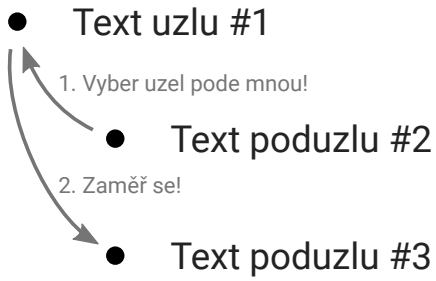
Proto jsem strom realizoval rekurzivním způsobem a každý prvek uzlu obsahuje všechny své následovníky.

5.7.2 Výběry a zaměření klávesnice

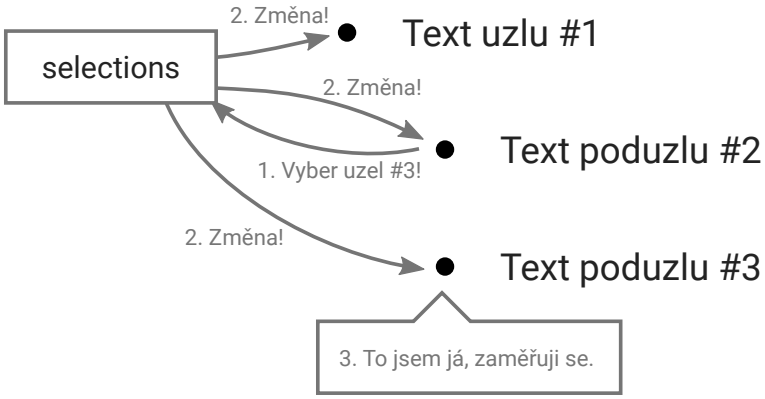
Rád bych zmínil jednu detailnější část implementace, která se ukázala být nečekaně náročná a zdlouhavá, plná slepých uliček, ba přímo dlouhých slepých bulvárů, kterým by se případný čtenář mohl vyhnout. Jedná se o řešení výběrů a zaměření klávesnice (anglicky keyboard focus).

První problematickou záležitostí je funkčnost dvou stavů každého uzlu. Specifikace počítá s tím, že první klepnutí uzel označí, druhé zahájí editaci textu. S implementací není žádný problém do té chvíle, než do hry vstoupí atribut `contenteditable`. Ten se při řízení zaměření klávesnice chová velmi zvláštně, v různých situacích si například zaměření krade sám a je těžké nad ním zaměřování omezit. Z třídenního zkoušení všech možných řešení založených na různých hodnotách atributů `contenteditable`, `tabindex`, posloupností a potlačování všemožných typů událostí jsem došel k aktuálnímu řešení, které pracuje s vhodným blokováním události `mousedown`. Nedořešené je bohužel chování u dotykových zařízení, kde by blokování sesterské události `touchstart` znemožnilo scrollování.

Druhou částí, která si vyžádala zvláštní péči byla realizace výběrů.



Obrázek 5.3: Nevhodná realizace výběrů pomocí událostí



Obrázek 5.4: Aktuální realizace výběrů pomocí sdíleného modelu selections

Je přirozené, že metodu `focus()` sloužící k zaměření klávesnice je nutné volat nad konkrétními vizuálními prvky, nikoliv nad modelem.

Řízení označení a posun označení šipkami jsem se proto pokusil realizovat pomocí událostí, které se propagují skrz DOM strom poznámek (obr. 5.3). Došlo zde k vertikálnímu rozdělení do jakýchsi sfér zodpovědnosti. Uzel, který chtěl označit například svého sourozence vyslal událost, kterou zachytil nadřazený uzel a označil (a zaměřil) prvek příslušného potomka. To vše fungovalo hezky do té doby, než se začaly realizovat *manipulace s uzly*. Když se v datovém modelu přidal nový uzel, který se měl zároveň i označit, sice se vyslala událost označení, jenže element ještě nebyl v návaznosti na změnu modelu vytvořen, takže nebylo co označovat.

Z toho důvodu jsem zvolil jiný přístup a vytvořil speciální model „selections“, který je sdílen uzly a přílohami (obr. 5.4). Prvek, jakmile zjistí, že je označen, si vynutí zaměření klávesnice. Díky tomu je možné zaměření spolu s označováním řídit bez přímé reference konkrétních DOM prvků a v případech, kdy se element vytvoří opožděně, pouze jednoduše upraví svůj stav podle modelu.

5.8 Přílohy uzlů

Jak již bylo naznačeno v sekci 4.2, v aplikaci je implementována podpora třech typů příloh: obrázků, videí z YouTube a zvýrazněných zdrojových kódů.

Při vytváření nových typů příloh je žádoucí, aby byly zaobaleny v samostatných prvcích. V případě obrázku je situace poměrně jednoduchá, stačí použít vestavěný tag img nebo iron-

image. Horší to je u komplikovanějších příloh, jako je třeba video z YouTube nebo zdrojový kód, kde je nutné implementovat prvky vlastní.

V případě *videa* jsem vytvořil rovnou dva prvky vhodné pro použití i mimo moji aplikaci: mto-youtube-video-thumbnail pro zobrazení náhledu videa například v pruhu pod uzlem a mto-youtube-video zaobalující video samotné.

Co se týče začlenění podpory *zdrojového kódu* jako přílohy, využil jsem existující komponentu ace-widget[6] zaobalující pokročilý editor Ace. Protože jsem v komponentě našel pár problémů, vyplnil jsem na stránkách GitHub zprávy o chybách a zaslal opravy, které byly následně schváleny a ve verzi 1.0.11 i zveřejněny.

Díky využití editoru Ace je aktuálně podporováno zvýrazňování syntaxe pro více než 105 programovacích a značkovacích jazyků.

Maximální zapadnutí do vzhledu a funkčnosti aplikace je pro mě důležité, takže jsem implementoval vlastní prvky pro vykreslování pruhu náhledů příloh a slideshow v galerii, i když by se pravděpodobně dalo najít nějaké hotové řešení v čistém JavaScriptu nebo jQuery. Nastudování a začlenění generického řešení by stejně nejspíš zabralo srovnatelné množství času.

Miniatury videí a obrázků jsou zobrazovány pod uzly pomocí prvku mto-thumbnail-stripe. Scrollování totoho pruhu s náhledy je realizováno pomocí JavaScriptu. Lehce problematická zde byla *detekce změny rozměrů obsahu* a vhodná reakce zobrazením navigačních šipek. DOM totiž nepodporuje žádný typ události informující o změně rozměrů prvku. Protože obrázek přirozeně mění svoje rozměry zejména při načtení, využil jsem prvku iron-image a zajistil, aby se obsah přeměřoval při každém zachycení události o úspěšném načtení obrázku. Událost o dokončení načítání vyvolává i náhled videa z YouTube.

5.9 Tok dat v aplikaci

Tok dat v aplikaci by měl být díky použití Polymeru a dvoucestného data-bindingu poměrně snadno srozumitelný a nikdy by neměl být větší problém dohledat, odkud která data pocházejí. Přesto bych rád načrtnul několik toků, které utvářejí páteř aplikace.

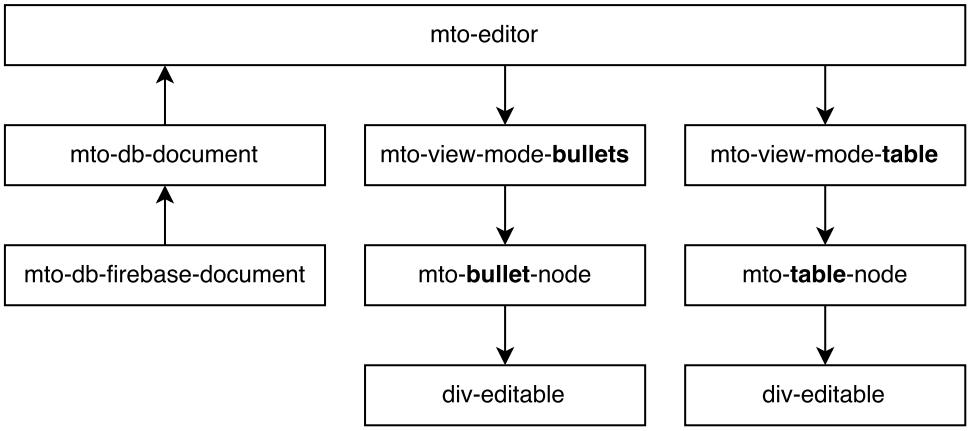
O prvcích fungujících jako zdroje dat (mto-db-document, mto-firebase-document) by se s trochou nadsázky dalo prohlásit, že se chovají jako prvky logických obvodů. Na vstup přivedeme jedny hodnoty, na výstupu se objeví hodnoty druhé. V našem případě na vstup vložíme URL adresu vzdáleného datového dokumentu a na výstupu získáme referenci na umístění a obousměrně vázatelná data.

Přítomnost reference na umístění nám umožní detekovat, že bylo navázáno spojení i v případech, kdy ještě není v databázi nic uloženo (právě například proběhlo první přihlášení).

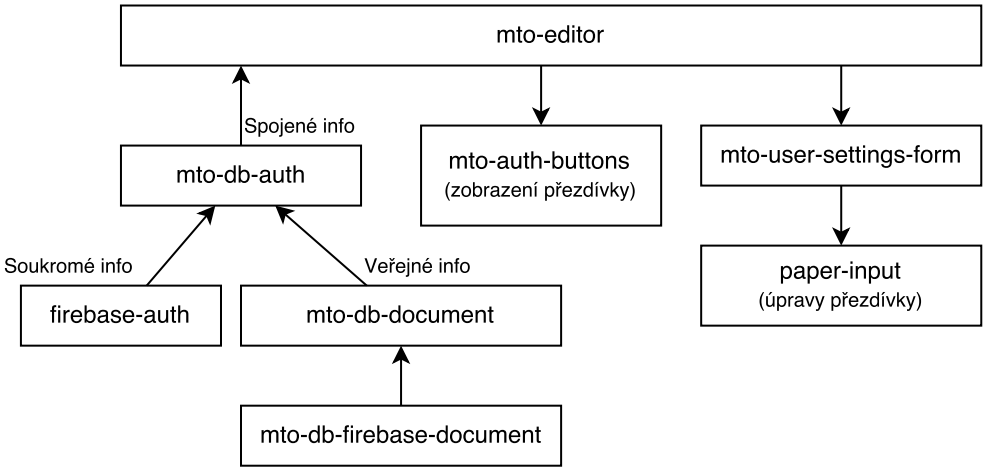
Ukázku toku dat proudících od prvku pro komunikaci s databází až po textové pole s uzlem je možné najít na obrázku 5.5. Při propagaci změn z textového pole data proudí v opačném směru.

Propagace informací o uživateli (obr. 5.6) probíhá podobně jednoduchým způsobem. Po přihlášení se na prvku mto-db-auth „naindukuje“ objekt s informacemi o uživateli, který dále putuje tam, kde je potřeba.

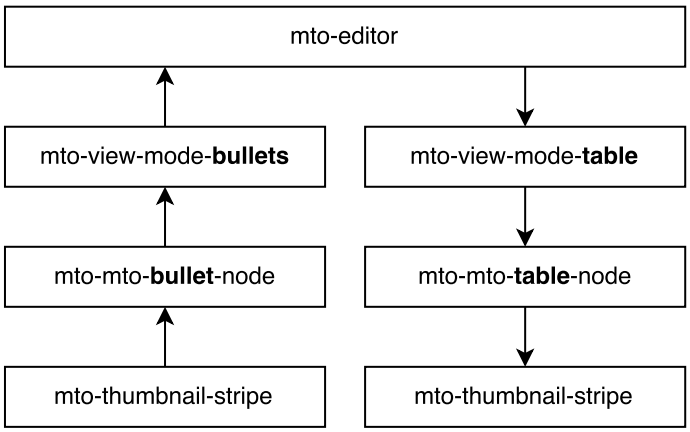
[6] https://github.com/LostInBrittany/ace-widget



Obrázek 5.5: Tok dat prvky od modelu k textovému poli uzlu



Obrázek 5.6: Tok informací o uživateli (při úpravách se směr samozřejmě mění od paper-input k mto-db-firebase-document)



Obrázek 5.7: Sdílení neperzistentního výběru (zde výběru příloh) mezi režimy zobrazení

5.10 Testování

Aplikace byla vyvíjena a testována v prohlížečích Firefox 46 a Google Chrome 50 na linuxových distribucích Ubuntu 14.04 a Arch. Funkčnost byla ověřena i v OS Windows 10 v prohlížeči Google Chrome 49 a s drobným omezením (viz str. 24) i v prohlížeči Chrome 50 na OS Android 5.1. Podpora starších prohlížečů může být problematická kvůli využití nový standardů a technologií, jako jsou CSS Flexbox a Web Components.

K vybraným kritickým částem aplikace jsou k dispozici automatické jednotkové testy implementované pomocí nástrojů Web Component Tester, javascriptového testovacího frameworku Mocha a knihovny Chai. Návod na spuštění testů lze najít v příloze B.

K nezávislému vývoji a testování většiny grafických prvků byly využity soubory `demo.html` dostupné ve složkách s danými elementy.

Uživatelské testování celkové funkčnosti proběhlo v komorním měřítku s přispěním několika rodinných příslušníků a přátel.

Kapitola 6

Další možný vývoj

Hlavním cílem implementace tohoto projektu je prakticky demonstrovat koncept a jeho nejzákladnější myšlenky s ohledem na množství práce, které je člověk začínající s webovým vývojem za půl roku schopný odvést. Při výběru funkcí k implementaci jsem se proto snažil vybírat primárně ty, které jsou nějakým způsobem nové a typické pro moji aplikaci na úkor těch, které každý zná a snadno si dovede představit. Implementoval jsem tak například nové typy příloh připojitelných k uzlům na úkor přidání funkce vícenásobného označování a práce se schránkou. Radši jsem vytvořil druhý způsob zobrazení, místo abych ztrácel drahocenný čas implementací krkolomné podpory undo a vracení změn.

Aplikaci lze vylepšovat téměř ve všech směrech. Její možná rozšíření by nejspíš dokázala zaměstnat několik lidí na několik let. Pojďme se podívat na pár konkrétních příkladů.

6.1 Další možné režimy zobrazení

Jednoznačným směrem, kterým by se vývoj mohl ubírat, je přidávání nových způsobů zobrazení. Vytvořil jsem několik málo grafických návrhů, jak by mohly vypadat.

Oblíbeným způsobem reprezentace stromovitých informací jsou *myšlenkové mapy* (mindmaps, obr. 6.1). Zajímavou technikou psaní poznámek jsou tzv. *Cornell notes* – technika vyvinutá profesorem z Cornell University [9] pracující s otázkami a shrnutími. I ta by se dala převést do digitální podoby (obr. 6.2). Vymyslet se ale dají exotičtější způsoby prezentace dat, jako třeba *kartičky* na obr. 6.3 a další.

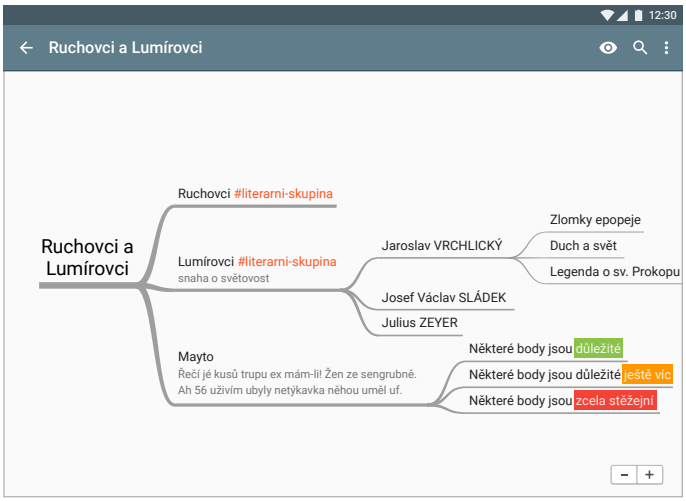
Zajímavé by bylo zpřístupnit veřejné API a umožnit vývojářům třetích stran vyvíjet zobrazení vlastní.

6.2 Další typy příloh

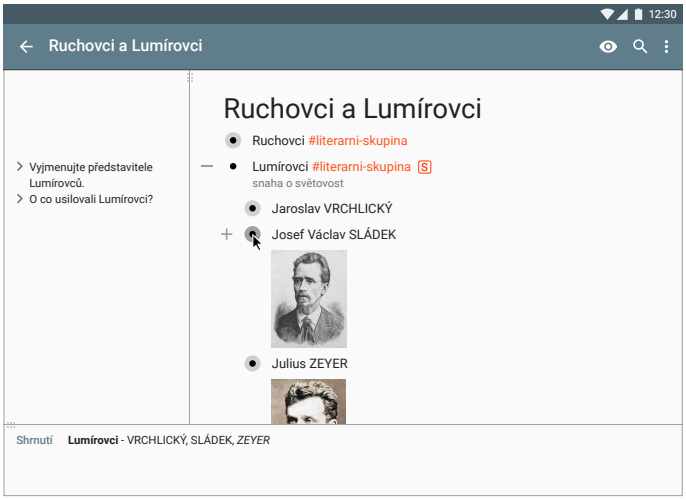
Rozšířit by se mohla i nabídka objektů připojitelných k uzlům. Možností je opět mnoho: matematické vzorce, úkoly (zaškrtávací pole), otázky pro možnost zkoušení se z obsahu, poznámky, apod.

6.3 Další možné funkce

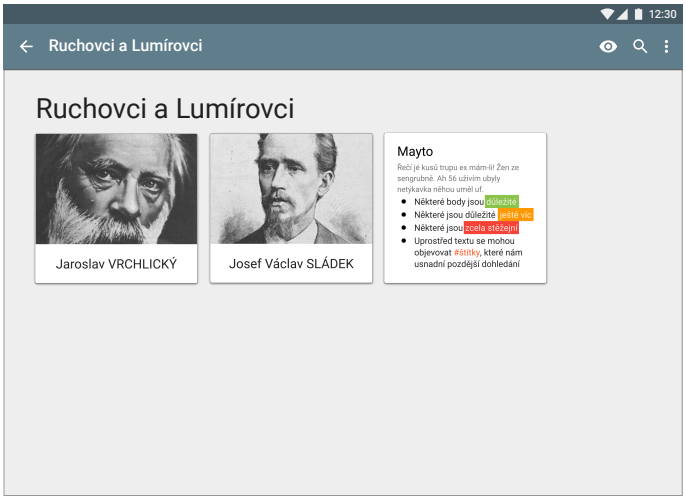
Co se týče dalších funkcí, uživatelé by určitě ocenili *fulltextové vyhledávání* v poznámkách, příjemná by byla možnost importu a exportu formátů ODT, HTML, PDF, MS Word a třeba i známého editoru myšlekových map FreeMind. Alternativou by mohlo být otevírání



Obrázek 6.1: Grafický návrh zobrazení jako myšlenkové mapy



Obrázek 6.2: Grafický návrh zobrazení s otázkami pro techniku Cornell notes



Obrázek 6.3: Grafický návrh zobrazení jako kartiček

dokumentů z Google Drive. Ať už si uživatel psal poznámky jakýmkoliv způsobem, neměl by mít problém je převést do Mayto a zpět.

Užitečný by mohl být *systém priorit*, který by umožňoval nastavit uzlům důležitost od 1 do 5, podle které by bylo možné uzly filtrovat. Student by tak při učení mohl začít pouze s nejpodstatnějšími údaji a postupně by mohl pokračovat odkrýváním detailnějších informací.

Samostatnou kapitolou by byla podpora spolupráce uživatelů v reálném čase a sdílení podstromů. Databázi Firebase jsem zvolil právě kvůli této eventualitě. Největším problémem by zde pravděpodobně byla koordinace úprav více uživatelů pomocí techniky operačních transformací zmíněné v sekci 3.2. Mohlo by se jednat o zajímavý námět na diplomovou práci.

Zavedením sdílení by se výrazně rozšířila využitelnost aplikace. Uplatnění by si rázem mohla najít i v korporátní sféře například pro organizaci informací o projektech a správu úkolů. Pokud by vznikla menší sociální síť, mohla by usnadnit sdílení poznámek nebo i vypracovaných maturitních otázek mezi studenty.

U editorů obsahu se očekává i možnost pracovat v režimu offline. Pokud vynechám řešení pomocí zaobalení webové aplikace do desktopové pomocí nástrojů jako Node Webkit[1], mohlo by být zajímavé vyzkoušet platformu Chrome Apps nebo technologii Service Workers, která pomalu získává podporu webových prohlížečů[2].

6.4 Optimalizace výkonu

Program pochopitelně nabízí i prostor pro zlepšení v oblasti výkonu. Jednou z technik, která by mohla nejen zrychlit prohlížení dlouhých stromů, ale i snížit paměťovou náročnost, je například tzv. nekonečné scrollování, které zajišťuje automatické odstraňování (recyklaci) DOM elementů, které zrovna nejsou vidět.

Zavedení by nicméně vyžadovalo značné strukturální změny, především implementaci dynamického zploštění stromové struktury a její DOM reprezentace, jak bylo naznačeno v sekci 5.7.1 na str. 24.

Druhou zajímavou možností hodnou prozkoumání je zavedení Facebookem propagované architektury *Flux* [4] pro jednosměrné data-flow pomocí podpůrné knihovny Funk[3]. Kromě zlepšení výkonu by se nejspíš zmenšilo množství kódu v prvcích a celkově zjednodušil návrh. Myšlenka kombinovat Flux a Polymer je poměrně nová (Funk vznikl v lednu 2016), takže jsem ještě neměl možnost knihovnu vyzkoušet.

6.5 Potenciální možnosti monetizace

Velmi krátce jsem uvažoval i o možnostech, jak by si aplikace v daleké budoucnosti mohla vydělávat na vlastní provoz.

První ze způsobů počítá s vytvořením menší sociální sítě zmíněné v předchozí sekci. Uživatelé by mohli pod svým jménem nabízet ostatním svoje kvalitně zpracované poznámky za drobný poplatek. Malé procento z každé transakce by bylo odváděno vývojářům Mayto. Kvalita obsahu by se dala podchytit systémem hodnocení uživatelů nebo možností prohlédnout si náhodně vybranou část nabízeného textu.

[1] http://nwjs.io/

[2] Aktuální stav podpory prohlížečů viz http://caniuse.com/#feat=serviceworkers

[3] https://github.com/devinivy/funk

Druhým zdrojem příjmů by mohly být zpoplatněné režimy zobrazení. Uživatel by například v základu dostal zdarma pouze režim odrážek, další by byly dostupné za jednorázový poplatek. Pokud by se zpřístupnilo veřejné API, mohl by vzniknout i trh se zobrazeními od vývojářů třetích stran.

Kapitola 7

Závěr

Navrhl jsem a s použitím technologií Polymer, TypeScript a Firebase implementoval webovou aplikaci na správu stromovitých poznámek, která umožňuje zobrazovat jedny data dvěma způsoby zobrazení, umožňuje základní operace s uzly a připojování tří typů příloh.

Pro potřeby vývoje jsem prostudoval a shrnul rozsáhlé množství nových technologií: existující javascriptové rámce a knihovny, prostředky pro tvorbu backendu, NoSQL databáze a v neposlední řadě podpůrné vývojové nástroje na platformě Node.js.

Zhodnotil jsem existující dostupná řešení a vymezil, jak se od nich odlišuje filozofie mé aplikace. Nastínil jsem také široké možnosti dalšího vývoje.

Při vývoji jsem si vyzkoušel přispívání do open-source projektu na GitHubu zasláním oprav drobných chyb v používané komponentě. Narazil jsem také na několik nedodělků v knihovně Polymer, které omezují a zpomalují architektonický návrh a na několik zvláštností `contenteditable` v HTML. Vše jsem popsal v kapitole 5.

Věřím, že čtení práce bylo pro čtenáře přínosné a že jej koncept alespoň z části zaujal. Sám budu dělat maximum pro to, aby aplikace nezůstala jenom bakalářskou prací na dně šuplíku, ale stala se užitečným produktem, který bude radost používat.

Literatura

[1] *WebComponents.org*. 2016 [cit. 2016-05-07], [online].
URL http://webcomponents.org/

[2] COPELAND, R.: *MongoDB Applied Design Patterns*. O'Reilly Media, Incorporated, 2013, ISBN 9781449340049.

[3] ELLIS, C. A.; Gibbs, S. J.: *Concurrency Control in Groupware Systems*. *SIGMOD Rec.*, ročník 18, č. 2, Červen 1989: s. 399–407, ISSN 0163-5808, doi:10.1145/66926.66963.
URL http://doi.acm.org/10.1145/66926.66963

[4] Facebook Inc.: *Flux – Application Architecture for Building User Interfaces*. 2015 [cit. 2016-05-07], [online].
URL https://facebook.github.io/flux/

[5] Facebook Inc.: *React – A JavaScript libary for building user interfaces*. 2016 [cit. 2016-05-07], [online].
URL https://facebook.github.io/react/

[6] Firebase Inc.: *Web Platform Documentation*. 2016 [cit. 2016-05-07], [online].
URL https://www.firebase.com/docs/web/

[7] Google: *Material design – Google design guidelines*. 2016 [cit. 2016-05-07], [online].
URL https://www.google.com/design/spec/material-design/

[8] KRILL, P.: *What's so special about Google's AngularJS*. 2013-10-29 [cit. 2016-05-07], [online].
URL http://www.infoworld.com/article/2612801/javascript/what-s-so-special-about-google-s-angularjs.html

[9] PAUK, W.; Owens, R. J. Q.: *How to study in college*. Wadsworth Publishing, 11 vydání, 2014, ISBN 978-1133960782.

[10] Polymer Authors: *Polymer Developer Guide*. 2016 [cit. 2016-05-07], [online].
URL https://www.polymer-project.org/1.0/docs/devguide/feature-overview.html

[11] SANTOS, N.: *Why ContentEditable is Terrible*. 2014-05-14 [cit. 2016-05-07], [online].
URL https://medium.com/medium-eng/why-contenteditable-is-terrible-122d8a40e480

[12] Solid IT gmbh: *DB-Engines Ranking*. DB-Engines, Květen 2016 [cit. 2016-05-07], [online].
URL http://db-engines.com/en/ranking

[13] W3C: *Custom Elements – W3C Editor's Draft*. 2016-04-08 [cit. 2016-05-07], [online].
URL http://w3c.github.io/webcomponents/spec/custom/

[14] W3C: *HTML Imports – W3C Editor's Draft*. 2016-04-08 [cit. 2016-05-07], [online].
URL http://w3c.github.io/webcomponents/spec/imports/

[15] W3C: *Shadow DOM – W3C Editor's Draft*. 2016-04-08 [cit. 2016-05-07], [online].
URL http://w3c.github.io/webcomponents/spec/shadow/

[16] WAYNER, P.: *PHP vs. Node.js: An epic battle for developer mind share*. 2015-01-12 [cit. 2016-05-07], [online].
URL http://www.infoworld.com/article/2866712/php/php-vs-node-js-an-epic-battle-for-developer-mind-share.html

[17] Web Hypertext Application Technology Working Group: *HTML – Living Standard*. 2016-05-06 [cit. 2016-05-07], [online].
URL https://html.spec.whatwg.org/multipage/scripting.html#the-template-element

Přílohy

Seznam příloh

Příloha A

Obsah CD

- `/`
  - `build` — Produkční sestavení aplikace určené pro nahrání na běžný webový server
  - `mockups` — Složka s podrobnými grafickými návrhy aplikace, ikon a některých budoucích funkcí
  - `report`
    - `sources` — Složka se zdrojovými kódy technické zprávy pro LaTeX
    - `report.pdf` — Finální technická zpráva v PDF
  - `sources`
    - `clean-repository` — Čistý repozitář GITu se zdrojovými kódy bez jakýchkoliv stažených závislostí nebo zkompilovaných javascriptových souborů
    - `with-dependencies` — Zdrojové kódy se staženými závislostmi a zkompilovanými soubory javascriptu
    - `firebase-data-sample.json` — Ukázka databáze
    - `firebase-security.json` — Konfigurace zabezpečení databáze

Příloha B

Spuštění a instalace

Pro vyzkoušení webové aplikace Mayto má uživatel aktuálně dvě možnosti:

- Využít předkompilované produkční verze spolu s libovolným webovým serverem
- Nainstalovat si vývojové nástroje a spustit vývojový server

B.1 Spuštění předpřipravené produkční verze

Nejjednodušším a nejrychlejším způsobem jak aplikaci vyzkoušet je využití předpřipravené zkompilované produkční verze. Tato verze obsahuje všechny HTML soubory a skripty minifikované a sloučené v několika málo souborech, aby při reálném nasazení bylo nutné provádět naprosté minimum GET požadavků a javascript bylo možné parsovat co možná nejrychleji.

Složku `build` se všemi potřebnými soubory je možné najít na přiloženém CD. V praxi by mělo stačit její obsah zkopírovat do složky webového serveru nebo jednoduše přejít do této složky a spustit jakýkoliv jednoduchý webový server, v linuxovém prostředí např. příkazem

```
python2 -m SimpleHTTPServer <port>
```

kde `<port>` je zvolené číslo portu, na kterém se má webserver spustit, např.:

```
python2 -m SimpleHTTPServer 9001
```

Po otevření příslušné URL adresy by se měla zobrazit titulní stránka aplikace, kterou můžeme vidět na obrázku B.1.

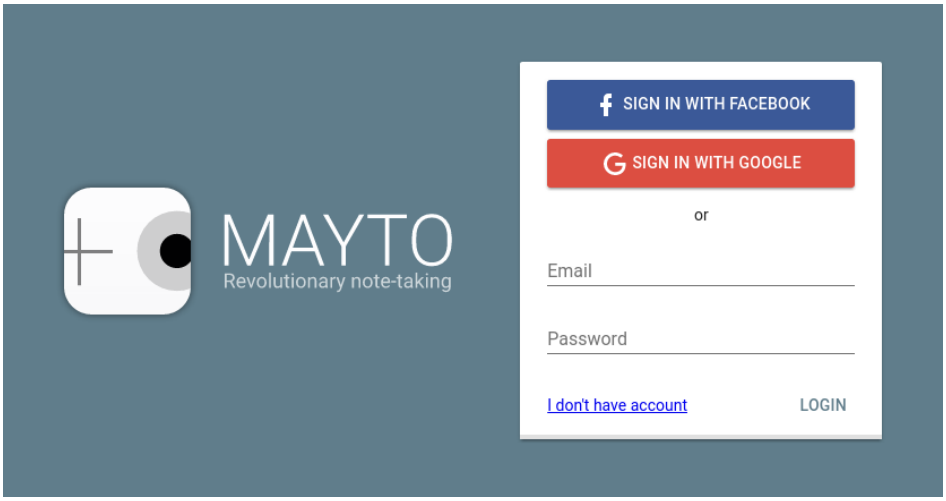
B.2 Instalace a spuštění prostředí pro vývoj

Postup instalace vývojového prostředí je určen pro linuxové distribuce Arch a Ubuntu. Podporované jsou i operační systémy Windows a OS X, postup by se neměl v mnohém lišit.

B.2.1 Přehled použitých vývojových nástrojů

I když vývojové nástroje byly probrány v sekci 2.4 na str. 7, dovolím si ještě jedno stručné shrnutí těch nejdůležitějších:

- **Node.js** – Interpret a sada knihoven pro psaní desktopových aplikací v JavaScriptu



Obrázek B.1: Titulní stránka aplikace

- **npm** – Správce balíčků pro Node.js – umožňuje instalovat z internetu různé šikovné javascriptové aplikace, které je pak možné spouštět z příkazového řádku stejně jednoduše jako třeba Vim.
- **Bower** – Node.js aplikace na správu závislostí - umožňuje jedním příkazem stáhnout všechny knihovny (jQuery, Polymer, ...), které aplikace potřebuje a udržuje je aktuální
- **Yeoman** (zkráceně „yo“) – Node.js aplikace na předgenerovávání zdrojových kódů. Usnadňuje počáteční vytvoření projektu a hlavně vytváření různých nových modulů, které člověk při vývoji potřebuje
- **Gulp** – Node.js task-runner, který, podobně jako make, spouští různé operace s projektem
- **tsc** – Kompilátor TypeScriptu, převádí soubory *.ts na *.js

B.2.2 Instalace aktuálního Node.js v distribuci Ubuntu

Protože verze Node.js v oficiálních repozitářích Ubuntu bývá zastaralá, je nutné využít repozitáře NodeSource. Pro přidání repozitáře a instalaci aktuální verze Node.js by měl stačit příkaz

```
curl -sL https://deb.nodesource.com/setup_5.x | sudo -E bash -
sudo apt-get install -y nodejs
```

B.2.3 Instalace aktuálního Node.js v distribuci Arch

Vzhledem k tomu, že Arch Linux je rolling-release distribucí s nevelkým zpožděním za upstreamem, aktuální verze Node.js je již přítomná v oficiálních repozitářích. Pro instalaci proto stačí jediný příkaz

```
sudo pacman -S nodejs npm
```

B.2.4 Instalace globálních vývojových nástrojů

Aby se aplikace npm instalovaly do domovského adresáře spíš než globálně do kořenového souborového systému, je nutné doplnit do `/.bashrc`:

```
export npm_config_prefix=~/.node_modules
export NODE_PATH=$NODE_PATH:~/.node_modules/lib/node_modules
export PATH=$PATH:~/.node_modules/bin
```

Následně je nutné restartovat terminál, aby se nová nastavení uplatnila.

Naše potřebné vývojové nástroje sdílené mezi projekty je možné nainstalovat příkazem

```
npm install -g yo gulp bower generator-polymer yeoman-doctor tabtab tsc tsd
    jshint svgo web-component-tester
```

B.2.5 Stažení součástí projektu

Čistý Git repozitář se zdrojovými soubory je možné najít na CD ve složce `sources/clean-repository`, případně je možné stáhnout aktuální verzi z internetu příkazem

```
git clone git@bitbucket.org:jhrdina/mayto.git
```

Verze z internetu se může lišit od verze odevzdávané.

V čistém repozitáři není prakticky žádný cizí kód. Žádné knihovny nebo vývojové nástroje. Vše je potřeba stáhnout. Protože se nicméně vývojové nástroje a další závislosti rapidně vyvíjí, je možné, že kvůli radikálním změnám něco v budoucnu přestane fungovat nebo nepůjde nainstalovat. Pro takový případ jsem připravil verzi se všemi staženými závislostmi (složka `sources/with-dependencies`). Potom je možné pokračovat rovnou sekcí B.2.6.

Ve složce s projektem stáhneme potřebné vývojové moduly příkazem

```
npm install
```

Aktuální javascriptové knihovny a oficiální prvky Polymeru by mělo jít stáhnout příkazem

```
bower install
```

TypeScript soubory se zkompilují do javascriptových příkazem

```
tsc
```

Je normální, že kompilátor vypíše několik chyb.

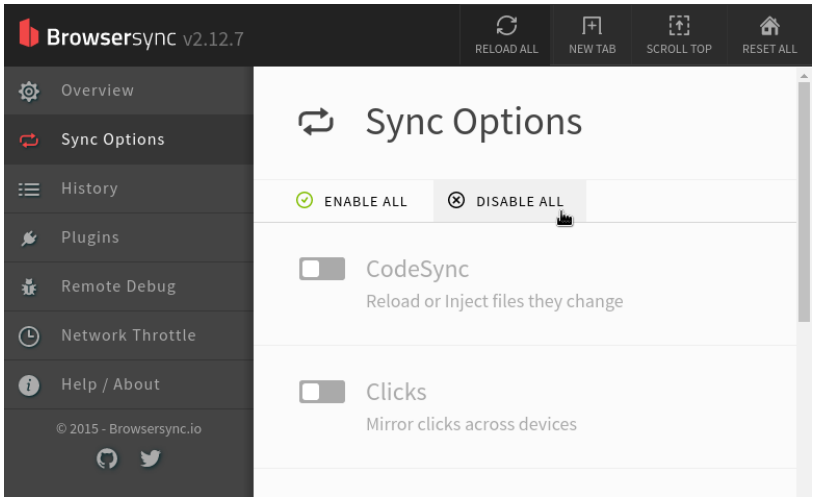
B.2.6 Spuštění vývojového serveru

Vývojový server je možné spustit příkazem

```
gulp serve
```

V terminálu se ukáže IP adresa a port, na kterém funguje a měl by se i otevřít prohlížeč s otevřeným projektem. Výchozí adresa bývá `http://localhost:5000`.

Pro potřeby testování na různých zařízení se vstupy klávesnice a myši všech klientů synchronizují. To může být nežádoucí pro testování některé funkčnosti mého programu. Funkci lze deaktivovat v konfiguračním rozhraní dostupném na portu zobrazeném v příkazu `gulp serve` pod položkou „UI“, ve výchozím nastavení na `http://localhost:3001`. V levém menu stačí vybrat *Sync options* a kliknout na *Disable all* (viz obr. B.2).



Obrázek B.2: Obrazovka nastavení Browsersync s možností vypnout synchronizaci vstupů klávesnice a myši

B.3 Automatické testy

Pro spouštění jednotkových testů je nutné mít nainstalované vývojové nástroje ze sekce B.2. Spustit je lze ze složky s projektem příkazem

```
gulp test
```

Podporované prohlížeče pro spouštění testů jsou Firefox a Google Chrome (Chromium).

B.4 Sestavení produkční verze

Pro sestavení produkční verze je nutné mít nainstalované vývojové nástroje ze sekce B.2.

Produkční verzi lze vytvořit ve složce s projektem příkazem

```
gulp
```

Výsledek lze potom nalézt ve složce `dist`.

B.5 Zobrazení ladících výpisů

Pokud by člověk náhodou narazil na nějaký problém se zprovozněním aplikace, může být vhodné nechat si zobrazit ladící výpisy v konzoli JavaScriptu. Tu lze otevřít v prohlížeči Chrome klávesovou zkratkou Ctrl+Shift+J, v prohlížeči Firefox pomocí Ctrl+Shift+K.