Chtěl bych poděkovat Ing. Tomáši Novotnému za odborné vedení a všestrannou pomoc. Dále bych rád poděkoval rodině a přátelům za jejich podporu.

České vysoké učení technické v Praze
Fakulta elektrotechnická
katedra počítačů

# BAKALÁŘSKÁ PRÁCE

## Webová platforma pro vzdálenou editaci souborů

**Autor:** Jan Fabián

**Vedoucí práce:** Ing. Tomáš Novotný

Praha, 2012

**Název práce:** Webová platforma pro vzdálenou editaci souborů

**Autor:** Jan Fabián

**Katedra (ústav):** Katedra počítačů

**Vedoucí bakalářské práce:** Ing. Tomáš Novotný

**e-mail vedoucího:** tomas.novotny@fel.cvut.cz

**Abstrakt** Tématem této bakalářské práce je implementace online editoru sloužícímu k úpravě souborů typu CSV, JSON a formátů podporovaných editorem Ace. Práce obsahuje serverovou část schopnou poskytovat interakci se souborovým systémem.

**Klíčová slova:** Online editor, CoffeeScript, Node.js

---

**Title:** Web platform for remote file editing

**Author:** Jan Fabián

**Department:** Department of Computer Science and Engineering

**Supervisor:** Ing. Tomáš Novotný

**Supervisor's e-mail address:** tomas.novotny@fel.cvut.cz

**Abstract** The topic of this thesis is an implementation of an online editor, which will be able to edit file formats as CSV, JSON and those supported by editor Ace. This work contains server part capable of providing interaction with file system.

**Keywords:** Online editor, CoffeeScript, Node.js

Prohlašuji, že jsem svou bakalářskou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce a jejím zveřejňováním.

V Praze dne 17. srpna 2012 Jan Fabián

# OBSAH

KAPITOLA
# PRVNÍ

# ÚVOD

Ve své bakalářské práci mám za úkol implementovat webovou aplikaci schopnou editovat vzdálené soubory[1] online. Moduly, které budou mít za úkol editaci, se dají rozdělit do dvou skupin.

1. Mnou definované knihovny pro editaci souborů JSON a CSV
2. Editor ACE

Ve výsledku bych tedy měl mít možnost číst soubory typu JSON, CSV a ty, které jsou podporovány editorem ACE.

Uživatel bude mít kromě volby otevírání, editace a ukládání souborů, také možnost definovat vlastní moduly schopné interakce s otevřeným souborem. Každý modul bude poskytovat nejnutnější API poskytující tuto možnost.

Celá aplikace bude klást velký důraz na JavaScript na klientské části, který ale bude i na serveru díky platformě Node.js. JavaScript nebude psán přímo, ale bude generován preprocesorem CoffeeScript. Ten poskytne aplikaci dynamický styl psaní kódu, zároveň přispěje k lepší čitelnosti.

Celý program by měl fungovat v prohlížeči bez nutnosti obnovování stránky, je proto zapotřebí zajistit asynchronní komunikaci mezi serverem a klientem.

[1] Na systému, kde je aplikace spuštěna.

KAPITOLA
# DRUHÁ

# POŽADAVKY

## 2.1. Funkční

Funkční požadavky definují úlohy aplikace. Co má a musí umět.

1. Procházení souborového systému
   a) Nahlížení do složek (Unixový ekvivalent příkazu ls)
   b) Změna pracovního adresáře (Unixový ekvivalent příkazu cd)
   c) Otevíraní souborů typu (Unixový ekvivalent příkazu open)
      i. CSV
      ii. JSON
      iii. Ostatní textové soubory podporované Ace editorem
2. Editace souborů
3. Ukládání změn souborů
4. Umožnit uživatel vytváření vlastní skriptů, které budou moci
   a) komunikovat s otevřeným souborem
   b) programovatelně editovat soubor

## 2.2. Nefunkční

Požadavky, které určují podmínky a omezení, jaké musí aplikace splňovat.

1. Aplikace bude psaná v jazyce CoffeeScript
2. Požadavky na server bude vyřizovat platforma Node.js
3. Aplikace zvládne otevírat a upravovat větší soubory[1]

[1] U souboru typu CSV 1000+ řádků, JSON 100+ uzlů, ostatní textové soubory 5000+ řádků

KAPITOLA
# TŘETÍ

# POPIS NÁSTROJŮ POUŽITÝCH V APLIKACI

Podle zadaní má být aplikace plně funkční ve webovém prohlížeči, hlavním implementačním jazykem bude JavaScript.

Na straně serveru musí běžet platforma schopná zpracovávat HTTP požadavky, odpovídat na ně a pracovat se soubory na lokálním disku. Kromě mnoha dalšího toto poskytuje Node.js3.1. Ke komfortnější práci s Nodem jako webovým serverem bude sloužit Zappa framework.

Pro urychlení komunikace mezi klientem a serverem při pohybu po souborovém systému aplikace použije knihovnu Socket.io, která je implementována v Zappě[1].

Na klientské části bude potřeba masivně procházet a měnit DOM, což zajistí jQuery knihovna. Hlavní nástroj pro psaní udržitelného kódu na front-endu ale bude Backbone.js.

S rozvojem JavaScriptových enginů v prohlížečích je možné psát mnohem rozsáhlejší a náročnější aplikace, což by bez řádné modularizace vedlo ke spaghetti kódu[2]. Na klientské části mé aplikace proto bude zaveden MVC[3] model, implementován právě pomocí Backbone.js. Ten jde ruku v ruce s Underscore.js, knihovnou šikovných pomocných metod, které v JavaScriptu chybí, nebo jsou zavedeny, až ve verzi ECMAScript 6.

## 3.1. Node.js

Node.js umožňuje práci s adresářovými i síťovými I/O[4] událostmi. V praxi znamená spuštění aplikace v Node rozběhnutí jediného vlákna, které reaguje na vstupy a výstupy. U webového serveru by takový přístup byl celkem problém, pokud by se připojil uživatel A, tedy vyslal by svůj HTTP request, nemohl by server odpovídat jiným uživatelům, dokud by nezpracoval a neposlal odpověď uživateli A.

Node ale funguje na principu asynchronous, non-blocking I/O, jinými slovy tok programu je určován tzv. callbacky[5]. Pro ilustraci následující příklad[4].

Dejme tomu, že bychom chtěli v PHP zavolat MySQL dotaz, vypadalo by to přibližně takto

[1] Jediný význam je čistě estetický, tvůrci Zappy implementovaly methody Socket.IO do svých vlastních, není potřeba načítat další knihovny.

[2] Zdrojový kód s komplexní a spletitou kontrolní strukturou

[3] Controller je zahrnut už v samotném View, proto tato architektura bývá spíše označována jako MV*

[4] Input/Output

[5] Název callback napomáhá k pochopení významu, jako bychom říkali 'call back, when you are done'

```
result = query('SELECT * FROM Table');
// dále pracujeme s proměnnou result
```

s tím, že na dalším řádku očekáváme v proměnné result výsledky našeho dotazu. Celý běh aplikace se tedy musel na tomto řádku zastavit, čekat až MySQL server vrátí odpověď a teprve pak pokračovat ve vykonávání kódu. Což je vcelku nevýhodné, vezmeme-li v potaz, že všechny zbývající operace se vykonávají na úrovni CPU, tedy v několika milisekundách, ale program zkrátka musí čekat (klidně i několik sekund), dokud mu MySQL server neodpoví.

Node.js k danému problému přistupuje následovně.

```
query('SELECT * FROM Table', function(result) {
        // dále pracujeme s proměnnou result
});
```

V čem spočívá výhoda? Na úrovni kódu, kde voláme methodu query, pokračuje hlavní smyčka programu dál. Tedy vše co následuje za metodou query se hned vykoná[6]. Teprve ve chvíli, kdy server obdrží výsledky našeho MySQL dotazu, s nimi dál pracujeme.

### 3.1.1. NPM

Node package manager, tedy balíčkovací nástroj pro správu knihoven použitých v Node.js projektu.

#### Instalace

```
curl http://npmjs.org/install.sh | sh
```

### 3.1.2. CoffeeScript

Na začátku kapitoly jsem se zmínil, že aplikace má být psaná v JavaScriptu. To není úplně pravda, ačkoliv výsledný kód opravdu JS bude. Při psaní totiž použijeme preprocesor CoffeeScript.

CoffeeScript je JavaScript, jen používá jinou syntaxi k vyjádření toho samého[**?** ]. Například v JavaScriptu napsaný kontruktor[7]

```
var Foo;
Foo = (function() {
  function Foo() {}
  Foo.prototype.contructor = function(bar) { this.bar = bar; };
  return Foo;
})();
```

je to samé jako v CoffeeScriptu

```
class Foo
  constructor: (@bar)->
```

[6] Například náročný algoritmus, který nesouvisí s proměnnou result.

[7] Metoda, která při zavolání s klíčovým slovem ‘new‘ vytváří nový objekt

Syntaxe se snaží být co nejvíce podobná dynamickým jazyku Ruby nebo Python. K oddělení kontextu kódu [8] používá namísto složených závorek odřádkování, které musí být konstantní v rámci jednoho CoffeeScript souboru.

#### Instalace

```
npm install coffee-script
```

### 3.1.3. Zappa

Zappa je CoffeeScript interface pro frameworky Express a Socket.IO. Jinými slovy je to knihovna metod určených pro vyřizování dotazů na server, což ulehčuje práci, jelikož aplikace poběží ve webovém prohlížeči.

#### Instalace

```
npm install zappa
```

### 3.1.4. Socket.IO

Knihovna obalující technologie WebSocket[9], Adobe® Flash® Socket, AJAX a JSONP polling. Díky tomu jsou podporovány prakticky všechny verze používaných prohlížečů. Jakmile není možné využít jednu službu zajišťující přenos socketů, Socket.IO má na výběr z několika dalších.

#### Instalace

```
npm install socket.io
```

## 3.2. Klient

### 3.2.1. jQuery

Knihovna pro výběr, modifikaci a pohyb po uzlech v DOM[10]. Dovoluje vytvářet nové objekty v dokumentu, má API pro vytváření a odchytávání událostí. V neposlední řadě poskytuje metody pro asynchronní HTTP komunikaci AJAX[11].

### 3.2.2. Backbone.js

MV* Framework použitý k modularizaci JavaScriptového kódu na frontendu. Zkratka výchází z model-view, ale chybí controllerová část. Tím, že veškeré operace probíhají na klientovi, v prostředí prohlížeče, přebírá funkcionalitu controlleru view, které kromě zobrazování dat také odchytává a reaguje na události (click, hover etc.).

Tato knihovna obsahuje i routovací třídu Router, umožňuje proto vytvářet single-page aplikace, bez nutnosti znovu přehrání celé stránky.

[8] Scope B

[9] Nejnovějšími prohlížeči podporovaný způsob oboustranné komunikace mezi klientem a serverem.

[10] Document Object Model

[11] Asynchronous JavaScript and XML

### 3.2.3. Underscore.js

Menší knihovna pomocných funkcí - helperů. Poskytuje esenciální metody pro práci s objektu, poli i funkcemi, které buď vůbec nejsou v aktuální verzi JavaScriptu definovány nebo se jejich implementace vyskytuje až v novějším vydání. Underscore nám zajišťuje funkčnost metod jako each, map, reduce a mnoha dalších, které při spuštění na starší verzi ECMAScriptu nemusejí být vůbec definovány.

### 3.2.4. Compass

CSS platforma poskytující možnost psaní kaskádových stylů v preprocesoru SASS. Hlídá změny ve zdrojovém kódu a automaticky exportuje výsledný soubor do formátu CSS. Kód, se kterým přijde designer do styku, podporuje DRY[12] díky využití proměnných a znovupoužitelných kusů kódu, tzn. mixinů. Odpadá také nutnost definovat CSS3 vlastnosti pro jednotlivé enginy prohlížečů zvlášť, o to se postará Compass při exportu dat do CSS formátu.

[12]Don't Repeat Yourself, styl psaní kódu, při kterém nedochází k zbytečné duplikaci textu

KAPITOLA

# ČTVRTÁ

# ANALÝZA

V aplikaci budou tři základní moduly, které uživateli dovolí editovat obsah. Každý z nich bude mít své UI, vyhovující požadavkům pro práci s daným souborem. Jeden z těchto modulů, ACE editor, má vlastní interface, u kterého není potřeba nic měnit. Zbylé dva, CSV a JSON, je potřeba analyzovat z hlediska možných případů použití. Analýza CSV editoru spolu s rešerží současných řešení je popsána v kapitole 4.2, JSON editor v kapitole 4.3.

Vedle modulů pro práci se soubory se bude v aplikaci vyskytovat ještě jeden sloužící k obsluze adresářového stromu, který bude jednotlivé soubory otevírat, jak je popsáno v kapitole 4.1.

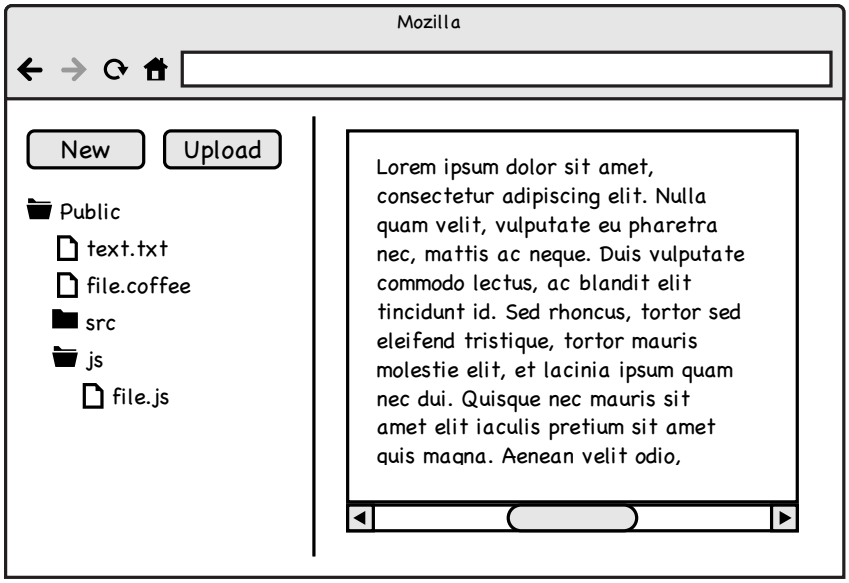
## 4.1. File-system

File-sytem je část aplikace odpovědná za práci se souborovým systémem, měla by mít chování podobné průzkumníkům v operačních systémech.

### 4.1.1. Procházení složek

Při pohybu po adresářové struktuře serveru bude uživatel moct otevírat a zavírat složky ve smyslu náhledu. Adresářový strom tak bude narůstat s počtem otevřených složek viz. návrh 4.1 levá část okna. K zpřehlednění bude uživatel moc změnit pracovní složku, tedy zobrazené soubory a podsložky.

Kliknutí na odkaz složky se tedy dá připodobnit k unixovému příkazu `ls`, zatímco dvojí kliknutí znamená příkaz `cd`.

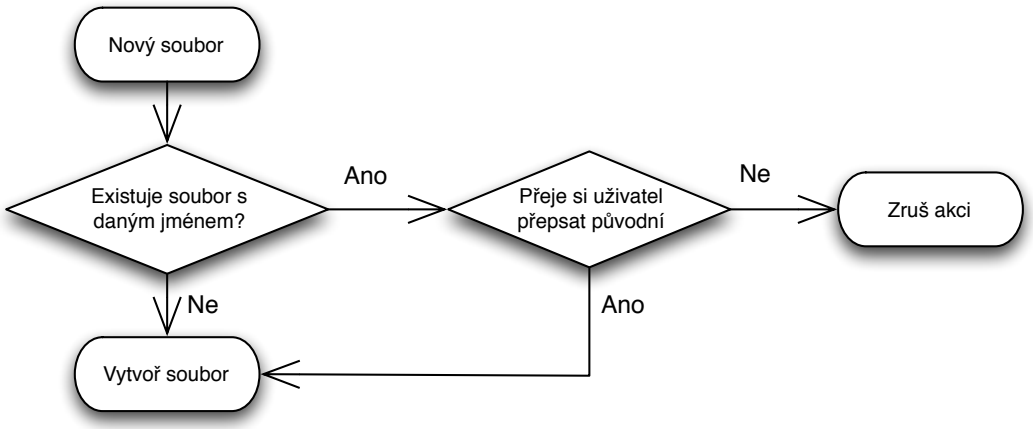



Obrázek 4.1.: UC Procházení složek

### 4.1.2. Práce se soubory

Po kliknutí na odkaz souboru bude aplikace umožňovat otevření daného souboru. Ten se zobrazí v editační části okna, tj. pravá část v návrhu 4.1.

Uživatel bude mít možnost vytvořit nový soubor. Po kliknutí na příslušné tlačítko, v návrhu 4.1 s popiskem New, se spustí údalost popsaná v diagramu 4.2.




Obrázek 4.2.: UC Práce se soubory

Aplikace bude umožňovat nahrání vlastního souboru na server. Tento soubor se ihned po nahrání zobrazí uživateli bez nutnosti obnovy prohlížeče.

Editovaný soubor bude moct být uložen, obsah editoru se odešle na server, kde proběhne zápis na disk, o čemž bude uživatel informován notifikací.

## 4.2. CSV

V kapitole 4.2.1 uvedu srovnání současného řešení CSV editoru, poté provedu samotnou analýzu problému vzhledem k zadání a vyvíjené aplikaci.

### 4.2.1. Srovnání současných řešení

Vzhledem k rozsahu mé aplikace budou srovnávány pouze možnosti editace tabulky. Editory tabulek mají širší možnost využití, které ale není pro účely mé aplikace potřeba analyzovat.

#### Microsoft Excel 2010 (mac 2011)

##### Výběr oblasti

- Výběr řádku se provádí kliknutím na číslo řádku v levé straně, popřípadě klávesovou zkratkou shift+space.
- Výběr sloupce se provede kliknutím na hlavičkovou buňkou označenou standartně písmenem, možné je použít klávesovou zkratku ctrl+space
- Výběr oblasti se provede stiskem levé myši na buňku a tahem ve směru cílové oblasti. Klávesovou zkratkou je shift+šipka ve směru, kde má být oblast výběru zvětšena.

##### Přidání, odebírání oblastí

Přidat oblasti je možné řádek nebo sloupce je možné přes horní menu, položka Insert. Klávesová zkratka je cmd+i (PC ctrl+'+') s výběrem možností Entire row pro přidání řádku a Entire column. K tomuto dialogovému oknu je možné se dostat pravým kliknutím na buňku řádku nebo sloupce, který má být odstraněn, a výběrem Insert.

Odstranění oblasti je možné provést přes horní menu Edit -> Delete..., klávesovou zkratkou ctrl+'-' nebo pravým kliknutím na buňku a volbou Delete. Zobrazí se dialogové okno s těmito možnostmi

- Shift cells up - odstraní vybranou oblast a posune buňky pod oblastí nahoru
- Shift cells down - odstraní oblast a posune buňky vpravo od oblasti doleva
- Entire row - odstraní celé řádky vybrané oblasti
- Entire column - odstraní celé sloupce

Přidání nebo odebrání řádku či sloupce je možné zjednodušit a přeskočit tak nutnost vyplnit dialogové okno. Stačí nejprve vybrat celý řádek či sloupec a až poté zvolit přidání či odebrání oblasti.

#### Vyjmutí, kopírování oblasti

Před vyjmutím či kopírováním je nutné provést výběr oblasti. Vyjmutí i kopírování se provádí buď přes kontextové menu zobrazené po kliknutí pravým tlačítkem myši a po výběru Cut pro vyjmutí, respektive Copy pro kopírování, nebo klávesovou zkratkou cmd+x (vyjmutí), cmd+c (kopírování).

#### Vkládání

Po vyjmutí nebo kopírování je možné oblast opět do tabulky vložit. Vložení se provede pomocí kliknutí pravého tlačitka a volbou Paste, případně klávesovou zkratkou cmd + v. Pokud takto klikneme na jedinou buňku, zkopírovaný obsah se vloží jen jednou. Pro vícenásobnou kopii je možné označit celý řádek, popřípadě sloupec, obsah schránky se poté vloží po celé délce označení. Vložit kopii vícenásobně je možné i do oblasti vybrané tahem, šířka zkopírované oblasti musí ale v tom případě beze zbytku dělit šířku vybrané oblasti. Totéž platí i pro výšku.

#### Přesouvání

Po výběru libovolné oblasti se po krajích zobrazí modrý okraj, ten je možné metodou drag-and-drop přemístit na cílovou pozici.

Další možný postup je složen z těchto kroků

1. Kopírování oblasti (popsáno výše)

2. Přidání oblasti (popsáno výše) na požadované místo

Při těchto krocích se originální obsah posune doprava, či dolů v závislosti na naší volbě v zobrazeném kontextovém menu.

### 4.2.2. Analýza CSV

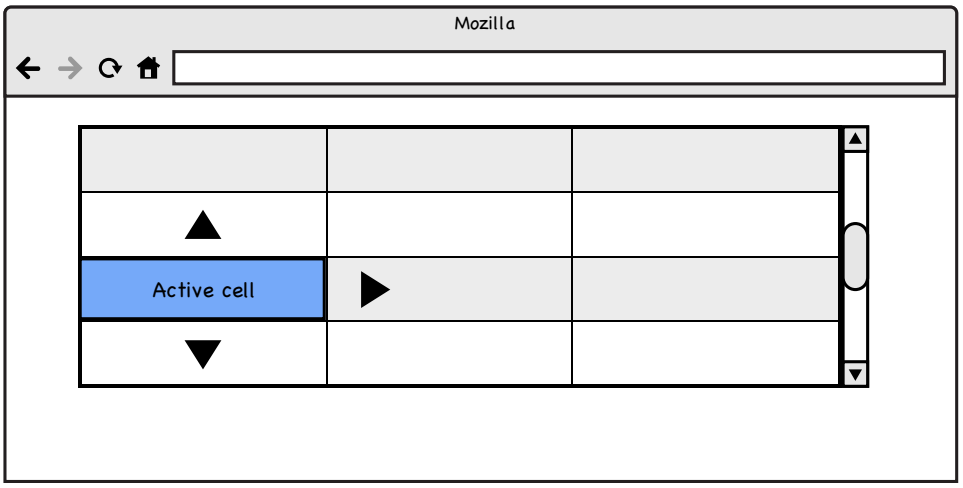
Aplikace umožní editaci tabulek ve formátu CSV, v následující kapitole bude popsána funkčnost a způsob ovládání tohoto modulu.

Tabulkový formát Comma-separated values má následující gramatiku - hodnoty na stejném řádku jsou oddělené čárkou, řádky symbolem `\n` pro nový řádek. Hodnota jedné buňky se může ohraničit uvozovkami pro escapování obsahu, což se hodí ve chvíli, kdy chceme v jedné buňce mít víceřádkový záznam.

```
00001, User 1, "Some notes, with commas but in one cell"
00002, User 2, "Some notes, with commas and new lines \n"
```

### 4.2.3. Pohyb po buňkách

Editor bude umožňovat pohyb do čtyř směrů. Tento pohyb bude vykonáván pomocí klávesových zkratek. Při dosažení krajních buněk bude možné pokračovat v pohybu jen zpětným směrem nebo po hraně, viz obrázek 4.3.




Obrázek 4.3.: Pohyb po buňkách

### 4.2.4. Editace buňky

Editovaný obsah se uloží do modelu tabulky při DOM události focusout, tedy ve chvíli, kdy uživatel opustí editaci buňky. Uživatel bude mít možnost vrátit obsah, který buňka obsahovala před začátkem editace.

### 4.2.5. Editace řádků a sloupců

Uživatel bude moct přidat řádek nad a pod právě editovaný řádek. V takovém případě se změní aktivní buňka, stane se jí buňka na nově přidaném řádku.

Smazání řádku, stejně jako přidání, vyvolá změnu aktivní buňky. Tou se stane buňka pod právě smazaným řádkem, pokud takový existuje. V opačném případě byl smazán poslední řádek a aktivní buňka se tak bude nacházet v řádku nad právě smazaným.

Stejně jako řádek může být přidán, případně smazán, i sloupec. Opět dojde ke změně aktivní buňky.

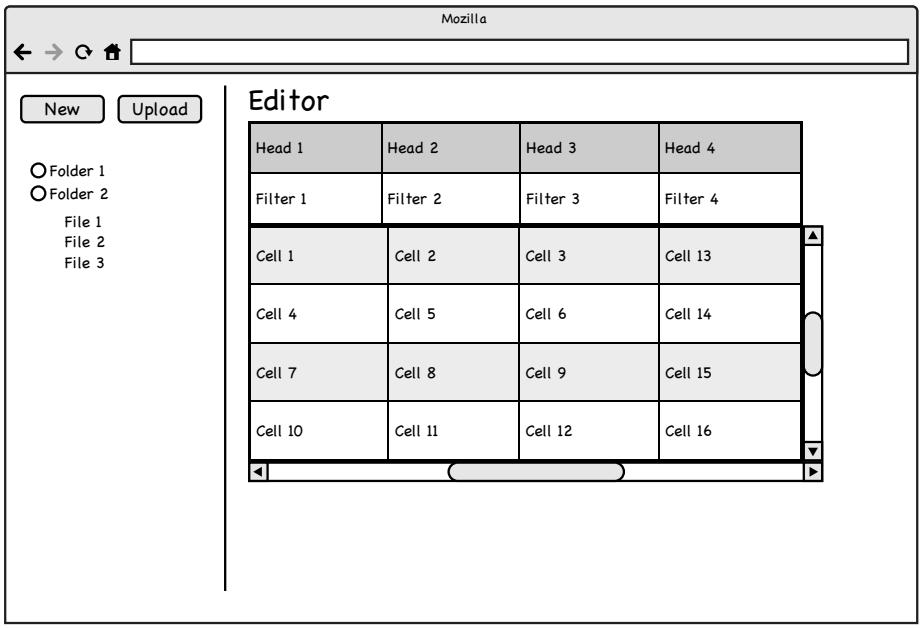
### 4.2.6. Filtrace řádků

Uživatel bude mít možnost filtrovat a zobrazovat jen ty řádky, které obsahují v daném sloupci zadaný řetězec. Během filtrace se bude uživatel moct pohybovat po tabulce, nicméně vstup pro filtrační řetězec zůstane aktivní. Postup kroků během filtrace bude následující

- Uživatel stiskne klávesovou zkratku pro zadání filtrace
- Vstup pro filtraci se stane aktivní
- Buňka pod filtrací, tedy prvního řádků, se stane aktivní

- Pohyb po tabulce bude možný, kurzor ale zůstává na filtraci
- Po zadání nové hodnoty se aktivní stane opět první řádek

V návrhu bylo potřeba vzít v potaz, že filtrační vstup musí být neustále vidět, to znamená i během scrollování. Proto ho není možné umístit jako jeden řádek tabulky. Vhodnější je vytvořit separatní tabulku o stejném počtu sloupců a jednom řádku, viz. obrázek 4.4.




Obrázek 4.4.: Návrh filtrace

## 4.3. JSON

Uživatel bude schopen editovat soubory typu JSON.

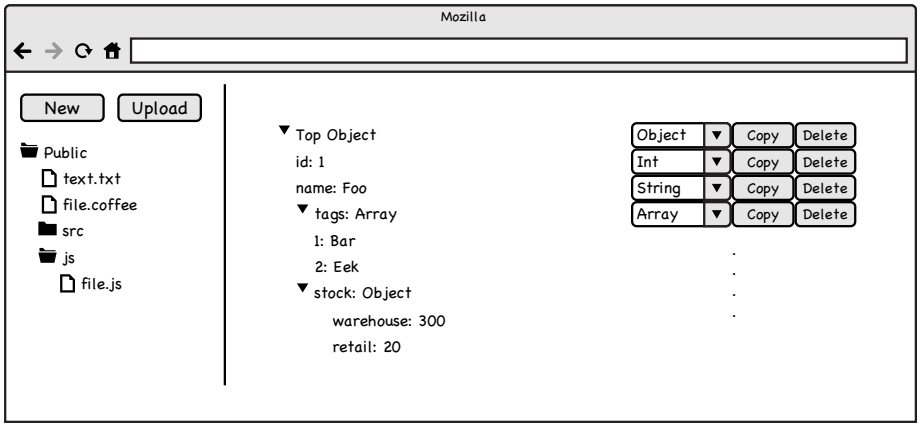
Formát JSON byl vyvinut Douglasem Crockfordem za účelem zaznamenat objekty do notace, která by odpovídala syntaxi JavaScriptu. Objekt v JavaScriptu je ve tvaru klíč: hodnota. Klíčem se rozumí řetězec znaků, unikatně identifikující hodnotu pro daný objekt. Přiřazená hodnota může být primitivní typ (string, number, null), array nebo další objekt.

JSON je standartní návratový typ při práci s REST API, oficiální MIME type je „application/json“.

```
{
        "id"     : 1,
        "name"   : "Foo",
        "tags"   : ["Bar","Eek"],
        "stock"  : { "warehouse":300, "retail":20 }
}
```

Id je v předchozím příkladu primitivním datovým type stejně jako name. Tags je řetezcem (Array) a stock objektem.

Návrh JSON editoru je uveden na obrázku 4.5.




Obrázek 4.5.: Návrh JSON editoru

### 4.3.1. Přidání nové větve

Uživatel bude mít možnost vložit pomocí klávesových zkratek nebo kliknutím myši novou hodnotu do stromu JSON objektu.

Aplikace bude umožňovat zduplikovat již existující větev. Pokud se bude jednat o složenou větev, provede se hluboká kopie, tedy duplikát bude obsahovat i všechny potomky.

### 4.3.2. Pohyb po větvích

Program bude uživateli umožňovat přeskakování po jednotlivých větvích. Je potřeba vzít v potaz, že při editaci by měl program podporovat rychlý přesun jenom po hodnotách a klíčích. To umožní uživateli rychlou editaci, zároveň však bude zpřístupněna možnost klávesového ovládání prvků na řádku (v návrhu 4.5 to jsou tlačítka manuálního výběru datového typu, Copy a Delete).

### 4.3.3. Editace větví

Uživatel bude moci editovat hodnotu a klíč větve. U primitivních to bude znamenat přímo editaci dané hodnoty, u složených objektů, případně řetězců, znamená editace úpravu potomků.

Kromě editace hodnoty bude aplikace umožňovat změnu datového typu větve.

Každý atribut v JSON objektu bude možné úplně odstranit. V návrhu 4.5 je počítáno s tlačítkem sloužícímu ke smazání větve s popiskem Delete.

KAPITOLA
# PÁTÁ

# IMPLEMENTACE

V následující kapitole jsou popsány způsoby implementace a řešení problémů, které se objevily během vytváření jednotlivých modulů.

V kapitole 5.1 bude popsán způsob vytváření balíků kódu na klientské straně, tedy v prohlížeči. Dále v kapitole 5.2 bude zpracováno zobrazovaní souborového systému, tedy ta část aplikace zodpovědná za otevírání souborů a procházení složek. Kapitole 5.3 bude věnován největší prostor, modul CSV editoru je totiž hlavní částí zadání celé práce. V kapitole 5.4 je popsáno implementační řešení editoru JSON souborů.

V samostatné sekci 5.5 bude věnován prostor vysvětlení způsobu komunikace se serverem tedy to, jak jsou zpracovávány dotazy a jak na ně server odpovídá.

## 5.1. Způsob modularizace na klientské straně aplikace

Klientská část aplikace se sestává z modulů určených pro práci s jednotlivými formáty souborů. K modularizaci kódu v prohlížeči je v aplikaci použita knihovna Stitch [1], která implementuje metodu `require` stejně jako je použita v serverové části kódu. To umožňuje rozdělit kód do více souborů, psát objektově a znovu používat některé kusy kódu zavoláním `require`.

V praxi tedy v jednom souboru (např. s názvem lib.coffee) definujeme objekt

```
module.exports =
        key1: 'Value'
        key2: -> console.log 'function'
```

případně funkci

```
module.exports = (arg) -> console.log args
```

Důležité je zmínit použití klíčového slova `module` a jeho atributu `exports`. A ve chvíli, kdy chceme použít danou metodu, případně dostat objekt, zavoláme

```
variable = require './lib'
```

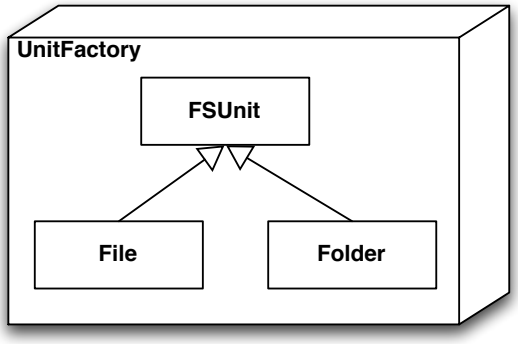
V tom případě nám program vrátí právě obsah proměnné `module.exports`, proto `variable` bude v prvním případě objekt s klíči key1 a key2, v druhém případě funkce vypisující do konzole její argumenty.

[1] https://github.com/sstephenson/stitch/

## 5.2. File-system

Tento modul má na starosti zobrazování a orientaci po souborovém systému. Jeho hlavním prvkem je třída `FS`. Udržuje si veškerá data, kterými jsou objekty třídy `File` nebo `Folder`. Jejich rodičovská třídy nese název `FSUnit` viz. obrázek 5.1.

Objekty třídy `File` znázorňují soubory, `Folder` potom složky. Hlavní objekt třídy `FS` volá při vytváření objektů třídu `UnitFactory`, konkrétně její statickou metodu `create()`.




Obrázek 5.1.: Hierarchie tříd v modulu File-system

Při vytváření adresáře zavolá objekt `FS` pro každý soubor nebo podsložku metodu `write-Unit`, která dále zavolá `UnitFactory.create()` a výsledek uloží do seznamu souborů. Tak se v paměti vytvoří interní reprezentace adresářové struktury. Ke konkrétnímu souboru v adresáři se dostaneme zavoláním metody `getUnitById`.

O zobrazení dat v prohlížeči se stará instance třidy `View`, kterou má objekt `FS`, i potomci třídy `FSUnit`. Každý tento objekt třídy view implementuje metodu `displayData`, která zajišťuje zobrazení dat.

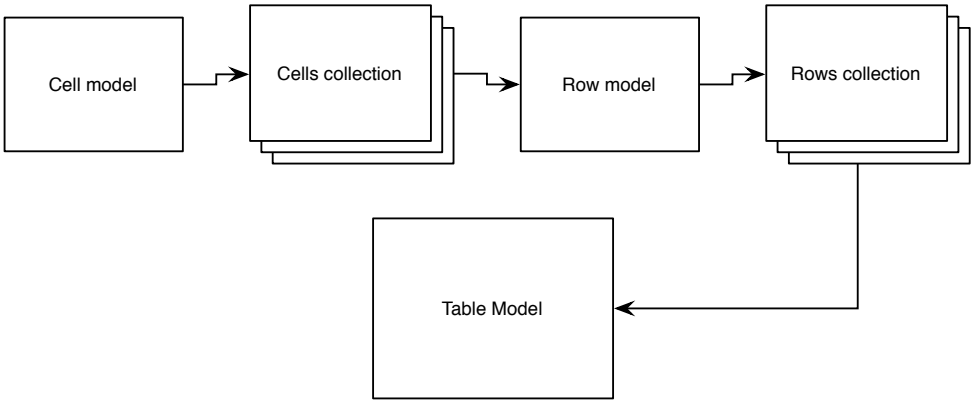
## 5.3. CSV

Hlavní část zadání této práce byl Editor CSV. Vzhledem k tomu, že měl být schopen zobrazovat tabulky v řádech tisíců řádků, rozhodl jsem se použít framework pro psaní klientských aplikací Backbone.js 3.2.2.

Nejprve v sekci 5.3.1 bude popsán způsob využití Backbone.js v aplikaci. V odstavci 5.3.2 bude nastíněn problém zobrazování tabulek s velkým množstvím řádků, možné způsoby řešení a nakonec finální způsob implementaci v aplikaci. Dále kapitola 5.3.3 popisuje možné způsoby přidávání elementů do html dokumentu, přičemž tyto způsoby jsou i otestovány a je změřena jejich časová výkonnost. V poslední části 5.3.4 se text věnuje způsobu změny ovládacích klávesových zkratek.

### 5.3.1. Backbone.js v modulu CSV

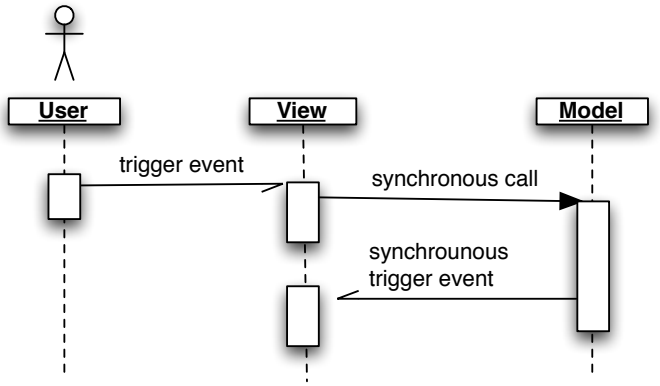
Modul CSV je postaven na Backbone.js frameworku. Modelová část je rozdělena podle následujícího diagramu 5.2.




Obrázek 5.2.: Class diagram CSV modelu

Hlavním stavebním prvkem jsou modely buněk a řádků, které jsou dále roztříděny do kolekcí. Stále se ale jedná jen o data, která nejsou nijak zobrazena. Zobrazení dat má na starost třída View.

View se ve frameworku Backbone.js stará nejen o vizualizaci dat, ale také o interakci s nimi, jinými slovy se jedná i o controller, tak jak je definován v MVC patternu. Komunikace mezi uživatelem, view a modelem probíhá podle diagramu 5.3.




Obrázek 5.3.: Vztah View-Model

Ve chvíli, kdy uživatel spustí událost, kterou sleduje View, například přidá nový řádek tabulky, proběhne synchronní volání Modelu. View už má v tu dobu nastaven event listener, takže ve chvíli, kdy Model dokončí změny, View je může bezpečně synchronně zobrazit. V našem případě tedy model zajišťuje jen validaci, stahování a uspořádání dat, zatímco View se stará o zobrazení. Otázku, jak zobrazit rozsáhlejší tabulku, řeší následující kapitola 5.3.2.

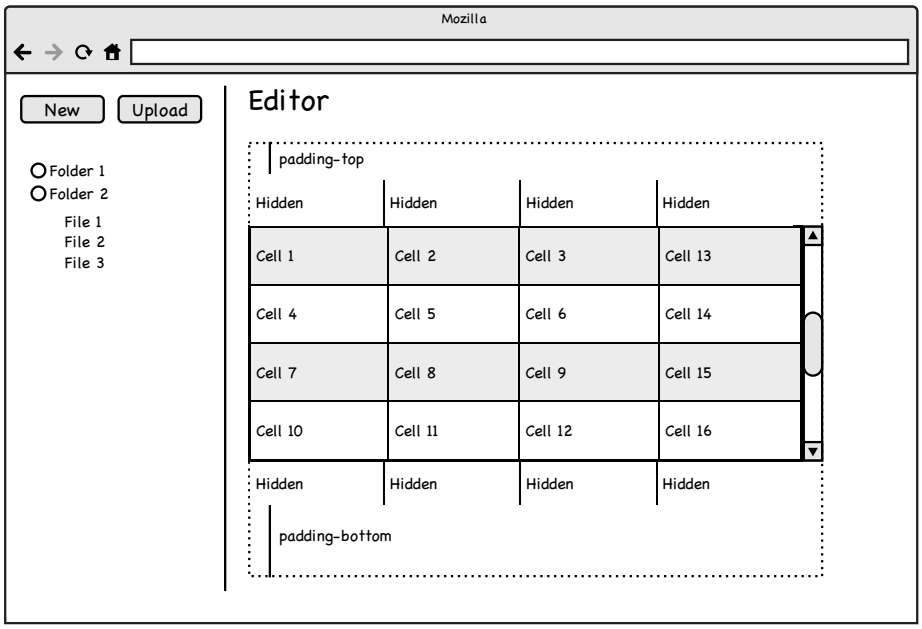
### 5.3.2. Zobrazení víceřádkových tabulek

U obsáhlejších záznamů čítajících mnoho řádků, řádově v tisících, nastává problém s načtením takto velké tabulky v prohlížeči. K problému lze přistupovat následujícími způsoby

1. renderování celé tabulky s nastavením pevné šířky buněk
2. postupným přidáváním řádků do tabulky
3. renderování a zobrazování jen těch řádků, které uživatel může skutečně vidět, jak je uvedeno v návrhu 5.4

První i druhý způsob vyžadují řešit jen prvotní inicializaci tabulky, dále už se o strukturu tabulky v DOMu není třeba starat. Budeme vycházet z předpokladu, že problémová část je právě prvotní zobrazení a jednotlivé enginy prohlížečů zvládnou správné a plynulé zobrazení při skrolování.

Oproti tomu třetí způsob s dynamicky se měnící strukturou DOM vyžaduje reagovat na událost scroll. Tento způsob nese v rámci této práce název dynamické skrolování.




Obrázek 5.4.: Návrh dynamického zobrazování řádků tabulky

### Dynamické skrolování

Pokud uživatel bude posouvat tabulku musí aplikace vykonat tyto kroky

- Odstranit z DOMu ty řádky, které nejsou vidět v divu obalujícím tabulku
- Přidat řádky, které se posunutím mají zobrazit
- Nahradit výšku odstraněných řádků paddingem tabulky a naopak odečíst z paddingu výšku přidaných řádků, tak aby zůstala výška scrollbaru konzistentní

### Způsob implementace

První popsaný způsob, tedy zobrazit celou tabulku najednou, nefunguje správně. U velkých tabulek se aplikace zasekává a přestává reagovat. Třetí způsob, tedy dynamické skrolování, je

rychlý, na druhou stranu přináší problémy při vyvolání callbacku během skrolování, někdy se řádky vůbec nezobrazily. Toto řešení také vedlo k problémům s posuvným skrolbarem, způsobených dynamickým nastavováním css atributu padding.

Aplikace využívá způsob postupného přidávání řádků do tabulky. V prvotní inicializaci tabulky je nastaven cyklus s voláním metody `setTimeout`, na které je zavěšeno zpětné volání zobrazení řádku. S postupnou iterací roste doba prodlevy u setTimeout, proto jsou řádky přidávány postupně v dávkách, zatímco uživatel je o tomto informován nahrávacím pruhem v horní části obrazovky. Velikost dávky a doba prodlevy se definuje v konfiguračním souboru `app/table/config.coffee`.

Tento přístup aplikaci zajišťuje plynulé načítání tabulek s velkým množstvím řádků. Při implementaci bylo potřeba zejména dbát na následující

- Zvolit pevnou šířku buněk, aby engine prohlížeče mohl přidávat řádky ihned.
- Nepoužívat atribut contenteditable ihned při vykreslení, nýbrž až po kliknutí na buňku.

### 5.3.3. Stavba stromových UI

Při větším množství elementů DOM je potřeba správným způsobem stavět stromovou strukturu těchto prvků. Je několik možností, jak k problému přistupovat.

První způsob je použití metody `appendChild`, kterou nabízí každý element. Toto řešení je elegantní, ale také značně neefektivní, protože každá změna DOMu je výkonnostně velice náročná.

Jiný způsob doporučuje připravit si přidávané elementy a vložit je do DOMu najednou. Tato metoda vyžívá metody `innerHTML` a můžeme rozlišit použití stringu nebo seznamu k vytvoření elementu. V prvním případě, tedy u stringu, budeme k přípravě elementu používat operátor `+=`, zatímco u seznamu metodu `push`, a budeme porovnávat rychlost těchto operací.

Poslední popisovaný způsob je použití tzv. fragmentů. Jedná se odlehčené a minimalistické objekty DOM API, které umožňují předpřipravit si sadu elementů a vložit je najednou jako u předchozí metody. Fragment je vytvořen metodou `document.createDocumentFragment()`.

Při srovnání výsledků je vidět, že použití metody fragmentů je nejefektivnější.

| | appendChild | innerHTML (String) | innerHTML (Array) | fragment |
|---|---|---|---|---|
| OS X, Chrome 21 | 70 ms | 58 ms | 47 ms | 19 ms |
| OS X, Firefox 12 | 907 ms | 41 ms | 44 ms | 32 ms |
| OS X, Safari | 21 ms | 25 ms | 22 ms | 11 ms |

Tabulka 5.1.: Srovnání průměru doby vytvoření tabulky 100 krát 100

#### Způsob implementace

Stavba řádků vychází z výše uvedené tabulky a používá fragmenty k sestavování a předpřipravení dokumentových elementů.

### 5.3.4. Způsob změny klávesového ovládání

Vzhledem k zadání, kdy během filtrace dochází k rozdílnému způsobu ovládání pomocí klávesových zkratek, musí aplikace umožňovat tuto změnu ovládání.

Třída vracející funkce vyvolané po stisknutí klávesových zkratek se nazývá `CellViewActions` a je umístěna v `app/table/cell.coffee`. Instance této třídy obsahují jako atributy jednotlivé reakce na události po stisknutí kláves, například metodu `move()`, která se stará o změnu aktivní buňky, tedy o pohyb po tabulce. Instance `CellViewActions` vrací tyto funkce jako objekt po zavolání `giveMeAPI()`.

Pokud chceme nadefinovat odlišné chování pro danou sadu klávesových zkratek, stačí implementovat novou třídu, v případě naší aplikace nazvanou `FilterStateActions`, která přepíše původní funkce novými. Zároveň může přidávat nové chování, stejně jako se děje ve `FilterStateActions`. Je ale potřeba přepsat i metodu `giveMeAPI()` do následujícího tvaru

```
giveMeAPI : ->
  _.extend super(),
    typeFilter : @typeFilter
```

Tato definice volá metodu extend knihovny Underscore.js 3.2.3, která jako parametr dostane objekt, v našem případě zavoláním rodičovské metody `super()` a objekt atributů, který přidá k prvnímu objektu. Na uvedeném příkladu přidáváme objekt s atributem `typeFilter`, který obsahuje stejně pojmenovanou metodu (ta je definovaná ve `FilterStateActions`).

Ve této chvíli už zbývá jenom do statické proměnné `CellView.keypressCbs` uložit objekt funkcí reagujících na klávesové zkratky

```
CellView.keypressCbs = new FilterStateActions().giveMeAPI()
```

a při stisku klávesy definovat, jakým callbackem budeme reagovat na kterou klávesu

```
keypress: (e) =>
  code = e.keyCode || e.which
  callbacks = CellView.keypressCbs
  cb =
    if code >= K_LEFT and code <= K_DOWN then callbacks.arrow
    else if code is K_DEL then callbacks.K_DEL
    else if code is K_ENTER then callbacks.K_ENTER
    # etc.
```

Tato implementace umožňuje lehce spravovat, přidávat a odebírat klávesové zkratky podle zadání.

## 5.4. JSON

Další část zadané práce byl editor JSON.

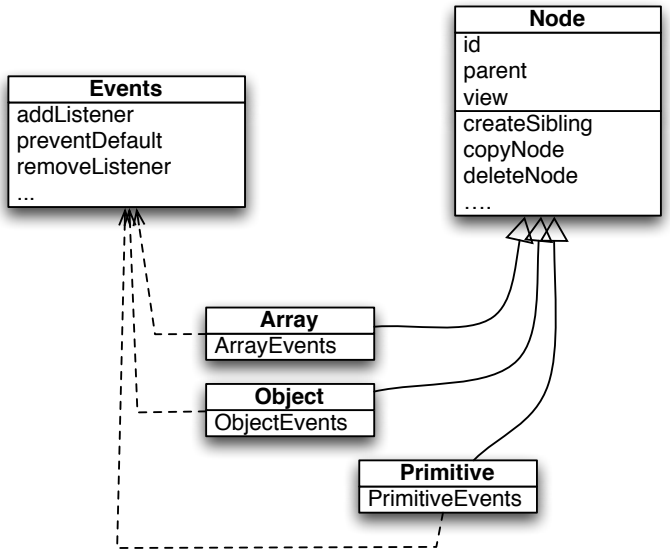
Aplikace si udržuje data a s nimi i view, tedy DOM elementy znázorňující tato data, což je výkonnostně efektivnější, než se na ně neustále dotazovat DOM API.

> Every time your ECMAScript needs access to the DOM, you have to cross this bridge and pay the performance toll fee. The more you work with the DOM, the more you pay.[6]

Nejenom, že je to efektivnější, ale při práci s větším JSON dokumentem i nezbytné. Pokud by měl editovaný JSON například 10 objektů jako hodnoty, každý z nich zase dalších 10 objektů, přičemž bychom se takto zanořili desetkrát, výsledný dokument v prohlížeči by měl 10^10 DOM elementů, pokud bychom je měli zobrazit všechny. Mnohem účinnější pro aplikaci je mít v paměti klienta pouze model, tedy data, a ta zobrazovat ve chvíli, kdy uživatel opravdu rozklikne rodičovskou větev.

### Vztahy tříd

Každá větev je potomkem třídy `Node` viz. obrázek 5.2. Instance odvozených tříd se liší ve vlastní implementaci reakcí na události - eventy, což aplikaci umožňuje reagovat jinak při obsluze událostí na různých typech větví, například dvojklik na větev typu objekt způsobí jiné chování aplikace, než ta samá akce na větev primitivního typu. Zároveň je žádoucí, aby některé vlastnosti a metody měly tyto objekty sdílené, například přidávání událostí bude u všech stejné, proto je aplikace obaluje rodičovskou třídou Events, kterou pak každá větev implementuje v proměnné pojmenované podle svého názvu s příponou Events.




Obrázek 5.5.: Diagram tříd JSON modulu

Každá větev má přiřazeno unikátní id, má právě jednoho rodiče a view, starající se o zobrazení dat. Instanční metody umožňují vytvořit sourozence (`createSibling`), zkopírovat větev (`copyNode`), případně danou větev smazat (`deleteNode`). Naopak některé metody jsou určené jen některým typům větví, například větev typu Array či Object musí mít metodu umožňující vytvoření potomka (`createChild`), naopak větev primitivního typu takovou metodu mít nemůže.

Toto byl poslední modul definovaný na klientské straně, v další kapitole 5.5 se již budu věnovat serverové části aplikace.

## 5.5. Server

Serverová část je postavená Node.js. Reaguje na požadavky klientů především přes sockety. V kapitole 5.5.1 uvedu, jak tento druh komunikace probíhá.

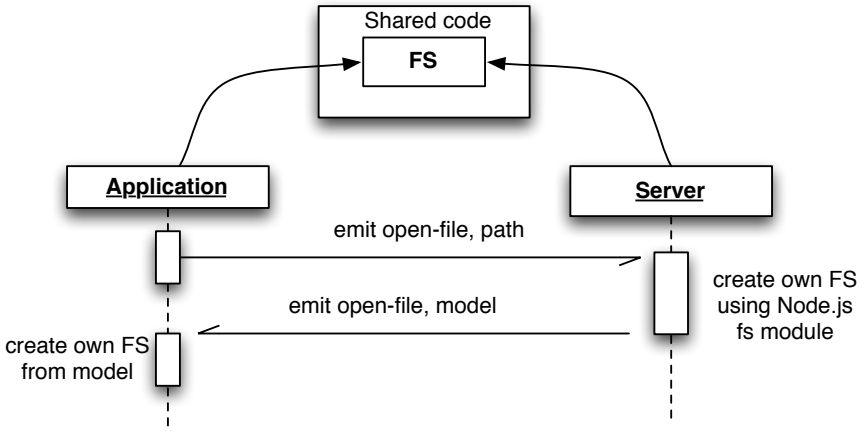
Server odpovídá i na HTTP požadavky. V kapitole 5.5.2 jsou podrobněji popsány. V poslední kapitole 5.5.3 této sekce ukáži srovnání obou metod komunikace a důvod své volby.

### 5.5.1. Socket.IO

Každý klient se při zobrazení aplikace v prohlížeči připojí a vznikne spojení mezi ním a serverem pomocí Socket.IO 3.1.4. Navzájem si vyměňují požadavky, na které reagují. Tyto požadavky si posílají metodou `emit` a odchytávají metodou `on`. Server má možnost poslat zprávu všem připojeným uživatelům pomocí funkce `broadcast`.

Klientská část aplikace komunikuje se serverem pomocí zpětných volání, na základě odpovědí serveru pak mění obsah okna prohlížeče. Toto chování zaručuje asynchronní komunikaci mezi klientem a serverem, aplikace může bežet v prohlížeči bez nutnosti obnovování stránky.

Na příkladu 5.6 je vidět způsob komunikace serveru a aplikace. Aplikace zavolá metodu emit s parametry open-file a path. Tedy žádá server o otevření složky s názvem uloženým v proměnné path. Server žádost odchytí, vytvoří model dané složky a pošle ho klientovi zpět, ten si z něj sestaví vlastní reprezentaci dat a zobrazí je uživateli.




Obrázek 5.6.: Socketová komunikace serveru a klienta

Na daném příkladu je vidět ještě jedna vlastnost. Jak klientská, tak serverová část aplikace využívá sdíleného kódu třídy FS.

Způsob komunikace je detailněji popsán ve výpisu zdrojového kódu. Ve chvíli, kdy uživatel pošle požadavek na server, například otevře složku, aplikace zavolá funkci `emit` a čeká na odpověď serveru (řádek 5 ve vypsaném kódu). Na serveru aplikace využívá Node.js knihovnu `fs` sloužící k procházení file systemu. Server vytvoří model adresářové struktury a pošle ho opět pomocí funkce `emit` zpátky dotazujícímu se klientovi (řádek 19). Ten tuto událost odchytne a data následně zobrazí uživateli (řádek 10).

```
## KLIENT ##
# Po kliknuti na odkaz slozky
$("#fs").on "click", ".directory", ->
  # Odesli na server udalost open-dir
  socket.emit('open-dir', id: that.id)

# Kdyz se od serveru vrati udalost open-dir
socket.on "open-dir": (data) ->
  # zobraz data
  new FSView(data).display()

## SERVER ##
# Jakmile se pripoji novy klient
socket.on 'connection': (client) ->
  # Nastav posluchac na udalost open-dir
  socket.on 'open-dir': (data) ->
    # V odpovedi na tuto udalost priprav File-system
    # a posli ho klientovi zpet
    client.emit('open-dir', 'FS': fileSystem.createFS data.
        path)
```

### 5.5.2. HTTP

Jediné HTTP požadavky, na které aplikace vrací odpověď, jsou následující

- GET na root adresu '/'
- POST dotaz na '/upload' při vzdáleném nahrávání souborů.

Díky frameworku Zappa 3.1.3, na kterém je aplikace postavena, stačí zavolat http metodu v daném rámci, viz. ukázka kódu.

```
require('zappajs').app ->
  @get '/': ->
    @render 'index'
  @post '/upload': ->
    writeFile @request.file, @response
```

### 5.5.3. Srovnání Socket.IO a HTTP

Jak je uvedeno v úvodu kapitoly, hlavní způsob komunikace probíhá pomocí Socket.IO.

Socket.IO ale ze své podstaty představuje špatné řešení pro mobilní zařízení. Tím, že si server s klientskou aplikací neustále vyměňuje pakety a dotazuje se, jestli je klient připojen, dochází k přenosu dat a tedy i plýtvání datového tarifu mobilního zařízení.

Na druhou stranu s ohledem na možnou spolupráci více lidí při editaci souborů touto aplikací, má Socket.IO velikou výhodu - umožňuje informovat o změnách provedených jedním uživatelem zároveň i všechny zbývající uživatele. Takový případ se může hodit ve chvíli, kdy někdo v aplikaci vytvoří nový soubor. Pokud bude mít více uživatelů danou složku otevřenou, tento nový soubor se zobrazí všem bez nutnosti přehrání stránky.

KAPITOLA
# ŠESTÁ

# TESTOVÁNÍ

Následující kapitola obsahuje průběh a výsledky akceptačního testování. Jednotlivé testy jsou strukturovány podle modulů tak, aby odpovídaly požadavkům uvedeným v kapitole 4. Cílem této kapitoly je ověřit funkčnost navrhované aplikace.

**Systémové požadavky**
Testování aplikace probíhalo v prostředí operačního sytému Mac OS X 10.7.3 při použití prohlížeče Google Chrome 19.

## 6.1. Testy

Aplikace je rozdělena do jednotlivých modulů, testy proto budou rozděleny do odpovídajících kategorií - CSV a JSON.

### 6.1.1. CSV

Všechny testy v následující části budou předpokládat, že uživatel má otevřen dokument s příponou csv.

**Test 1 - změna obsahu buňky**

| | |
|---|---|
| Vstup | Kliknutí na buňku a změna obsahu |
| Výstup | Změněná buňka |
| Výsledek | Aplikace splňuje zadání. |
| Komentář | Oproti tabulkovým kalkulátorům typu excel se editovaná buňka neoznačí celá, kurzor je umístěn na místo kliknutí. |

**Test 2 - návrat původní hodnoty buňky**

| | |
|---|---|
| Vstup | Stisknutí klávesy escape během editace buňky |
| Výstup | Editovaná buňka s původním obsahem |
| Výsledek | Aplikace splňuje zadání. |
| Komentář | Kurzor se automaticky umístí na začátek textu v editované buňce. |

### Test 3 - pohyb po buňkách

| | |
|---|---|
| Vstup | Stisknutí kláves ctrl + shift + šipka. |
| Výstup | Nově označená a editovatelná buňka ve směru stisknuté šipky. |
| Výsledek | Aplikace splňuje zadání. |
| Komentář | Při dosažení hraniční buňky aplikace nedovolí uživateli pokračovat. |

### Test 4 - nastavení prvního řádku jako hlavičkového

| | |
|---|---|
| Vstup | Stisknutí tlačítka Header v menu editoru |
| Výstup | První řádek se stane hlavičkovým řádkem a bude zafixován po celou dobu skrolování. |
| Výsledek | Aplikace splňuje zadání. |
| Komentář | Hlavičkovým řádkem se může stát pouze první řádek a to i tehdy, když je vyfiltrován. |

### Test 5 - smazání právě editovaného řádku

| | |
|---|---|
| Vstup | Stisknutí kláves ctrl + shift + backspace během editace buňky |
| Výstup | Aktuální řádek bude smazán. |
| Výsledek | Aplikace splňuje zadání. |
| Komentář | Po smazání kurzor automaticky přeskočí na řádek pod právě smazaným řádkem. Při smazání posledního řádku přeskočí kurzor na předposlední řádek. |

### Test 6 - smazání právě editovaného sloupce

| | |
|---|---|
| Vstup | Stisknutí kláves ctrl + backspace během editace buňky |
| Výstup | Aktuální sloupec bude smazán. |
| Výsledek | Aplikace splňuje zadání. |
| Komentář | Po smazání kurzor automaticky přeskočí na buňku vlevo od právě smazaného sloupce. Při smazání sloupce nejvíce vlevo přeskočí kurzor na sloupec vpravo. |

### Test 7 - přidání řádku pod editovaný řádek

| | |
|---|---|
| Vstup | Stisknutí kláves ctrl + shift + enter během editace buňky |
| Výstup | Nový řádek bude přidán nad editovaný řádek. |
| Výsledek | Aplikace splňuje zadání. |
| Komentář | Po přidání kurzor automaticky přeskočí na nový řádek. |

### Test 8 - přidání řádku nad editovaný řádek

| | |
|---|---|
| Vstup | Stisknutí kláves ctrl + enter během editace buňky |
| Výstup | Nový řádek bude přidán pod editovaný řádek. |
| Výsledek | Aplikace splňuje zadání. |
| Komentář | Po přidání kurzor automaticky přeskočí na nový řádek. |

### Test 9 - přidání sloupce vlevo od editovaného sloupce

| | |
|---|---|
| Vstup | Kliknutí na ikonu se symbolem + (plus) nalevo v horní části sloupce |
| Výstup | Nový sloupec bude přidán vlevo od editovaného sloupce. |
| Výsledek | Aplikace splňuje zadání. |
| Komentář | Po přidání kurzor automaticky přeskočí na nový sloupec.<br>Ikona pro přidání je vidět i za předpokladu, že je první řádek hlavičkový. |

**Test 10 - přidání sloupce napravo od editovaného sloupce**

| | |
|---|---|
| Vstup | Kliknutí na ikonu se symbolem + (plus) napravo v horní části sloupce |
| Výstup | Nový sloupec bude přidán napravo od editovaného sloupce. |
| Výsledek | Aplikace splňuje zadání. |
| Komentář | Po přidání kurzor automaticky přeskočí na nový sloupec. |

**Test 11 - zobrazení filtru**

| | |
|---|---|
| Vstup | Stisknutí kláves ctrl + f během editace buňky vyvolá zobrazení vstupu pro filtr |
| Výstup | Zobrazení filtračního řádku |
| Výsledek | Aplikace splňuje zadání. |
| Komentář | Aktivní buňka je právě ta ve stejném sloupci, který uživatel editoval, když stiskl klávesovou zkratku. |

**Test 12 - filtrace**

| | |
|---|---|
| Vstup | Zadávání hodnot do filtrační buňky |
| Výstup | Řádky tabulky, které nevyhovují zadání jsou odfiltrovány - skryty |
| Výsledek | Aplikace splňuje zadání. |
| Komentář | Během filtrace je uživateli umožněn pohyb po tabulce, aktivní stále zůstává filtrační buňka. |

### 6.1.2. JSON

Všechny testy v následující části budou předpokládat, že uživatel má otevřen dokument s příponou json.

**Test 1 - editace klíče**

| | |
|---|---|
| Vstup | Kliknutí na klíč u atributu objektu a změna obsahu |
| Výstup | Změněný klíč |
| Výsledek | Aplikace splňuje zadání. |

**Test 2 - editace hodnoty**

| | |
|---|---|
| Vstup | Kliknutí na hodnotu atributu větve primitivního typu |
| Výstup | Změněná hodnota |
| Výsledek | Aplikace splňuje zadání s připomínkou. |
| Komentář | Pokud uživatel smaže kompletně celý obsah hodnoty a ztratí focus, není možné kliknout na větev a znovu vyvolat editaci. |

**Test 3 - pohyb na další editovatelnou položku**

| | |
|---|---|
| Vstup | Stisknutí klávesy enter během editace |
| Výstup | Další editovatelná větev se stane aktivní. |
| Výsledek | Aplikace splňuje zadání. |
| Komentář | Po přesunu je kurzor umístěn na začátek editované hodnoty |

**Test 4 - pohyb na předchozí editovatelnou položku**

| | |
|---|---|
| Vstup | Stisknutí kláves shift + enter během editace |
| Výstup | Předchozí editovatelná větev se stane aktivní. |
| Výsledek | Aplikace splňuje zadání. |

### Test 5 - smazání větve

| | |
|---|---|
| Vstup | Kliknutí na tlačítko se znakem '-' (minus) |
| Výstup | Větev bude smazána |
| Výsledek | Aplikace splňuje zadání. |

### Test 6 - přidání nové větve (sourozenec)

| | |
|---|---|
| Vstup | Dvojí kliknutí na větev (v případě objektu, nebo seznam se předpokládá, že je větev zavřená) |
| Výstup | Nová větev |
| Výsledek | Aplikace splňuje zadání. |
| Komentář | Nová větev je přidána za tu, na kterou bylo dvakrát kliknuto |

### Test 7 - přidání nové větve (potomek)

| | |
|---|---|
| Vstup | Dvojí kliknutí na otevřenou větev typu objekt nebo seznam |
| Výstup | Nová větev |
| Výsledek | Aplikace splňuje zadání s připomínkou. |
| Komentář | Nová větev je přidána jako potomek větve, na kterou bylo dvakrát kliknuto. Testování ukázalo, že při přidávání potomků do větve typu seznam, se aplikace chová nestandartně v prohlížeči Mozilla Firefox. Jedná se o špatné číslování indexů. |

KAPITOLA
# SEDMÁ

# UŽIVATELSKÁ PŘÍRUČKA

V této kapitole je stručný návod k ovládání aplikace.

## 7.1. File system

### 7.1.1. Základní operace

- Zobrazit/skrýt obsah složky - **levý klik** na odkaz složky
- Změnit pracovní adresář - **dvojitý klik** na odkaz složky
- Otevřít soubor - **levý klik** na odkaz souboru

### 7.1.2. Přidání dokumentu

Přidání se provede vždy do aktuálního pracovního adresáře.

#### Nový soubor

Pro vytvoření nového dokumentu uživatel klikne na tlačítko `New file`. Zobrazí se dialogové okno, kam zadá název souboru. Pokud už takový soubor existuje, zobrazí se potvrzovací okno. Pokud uživatel potvrdí volbu, prázdný dokument přepíše původní.

#### Nahrát soubor

Po kliknutí na tlačítko Upload se zobrazí formulář se vstupem typu file. Pokud soubor již existuje, aplikace vrátí chybovou hlášku, v opačném případě se soubor uloží na server.

## 7.2. JSON


Obrázek 7.1.: JSON editor

### 7.2.1. Základní operace

- Zobrazit/skrýt obsah objektu (Object) nebo sbírky (Array) - levý klik na ikonu šipky vedle názvu větve
- Editace názvu/obsahu větve - levý klik na klíč/hodnotu
- Změna datového typu větve - změna se provede po výběru nového datového typu v pravé části větve
- Smazání větve - kliknutí na ikonu se znakem minus v pravé části větve

### 7.2.2. Přidání větve

#### Nová větev

Uživatel přidá větev buď jako potomka nebo sourozence. Dvojí kliknutí na větev primitivního typu, případně na zavřenou větev objektu nebo řetězce, přidá za tuto větev novou primitivního typu. Ta samá akce na otevřenou větev způsobí přidání nové větve na konec této otevřené větve jako potomka.

Pokud uživatel během dvojkliku drží **alt**, přidaná větev bude typu řetezec, **alt+shift** přidá novou objektovou větev.

#### Kopie větve

Plná kopie, tzv. deep clone, tedy kopie všech potomků se provede kliknutím na tlačítko s písmenem C v pravé části větve.

### 7.2.3. Pohyb po větvích

Uživatel skáče mezi všemi editovatelnými hodnotami stiskem klávesy **enter**. Pro pohyb zpět stačí vedle stisku enteru držet **shift**.

Prohlížeč dovoluje i pohyb po všech ovládacích prvcích stiskem **tabu**. **Shift+tab** je opět reverzní pohyb.

## 7.3. CSV

### 7.3.1. Základní operace

- Editace buňky - **levý klik** na buňku
- Potvrzení editace - stiskem klávesy **enter**
- Návrat k původní hodnotě v buňce - stiskem klávesy **escape**

### 7.3.2. Pohyb po buňkách

Aplikace umožňuje uživateli pohyb do všech čtyř stran pomocí kláves **ctrl + shift + směr pohybu**, tedy nahoru, dolů, vlevo či vpravo.

### 7.3.3. Editace řádků

Nový řádek nad aktuálně editovaný se přidává pomocí klávesové zkratky **ctrl + enter**. Pokud uživatel zároveň drží **ctrl + shift + enter**, nový řádek se přidá pod aktuálně editovaný.

Smazání aktuálního řádku provede uživatel stiskem **ctrl + shift + backspace**.

### 7.3.4. Smazání řádku

Aplikace umožňuje uživateli pohyb do všech čtyř stran pomocí kláves **ctrl + shift + směr pohybu**, tedy nahoru, dolů, vlevo či vpravo.

KAPITOLA
**OSMÁ**

# ZÁVĚR

Ve své bakalářské práci jsem se snažil implementovat větší CoffeeScriptovou aplikaci schopnou editovat soubory v prohlížeči. Vzhledem k tomu, že CoffeeScript, potažmo JavaScript, má mnoho specifik a unikátních vlastností[3], bylo potřeba kód neustále přepisovat a refaktorovat, aby se na něm dalo dál stavět. Já jsem díky tomu poznal, že psát jiný, než objektově orientovaný JavaScript, není dlouhodobě udržitelné a že, co se týká DOMu, musí člověk neustále optimalizovat[6].

Na otázku, jestli má JavaScript na serveru budoucnost, částečně odpovídá počet uživatelů služby Github sledující repozitář Node.js. Určitě se nejedná o všelék, ale na některé případy využití se asynchronní přístup JavaScriptu hodí velice. Ačkoliv je styl programování ve frameworku Node.js odlišný, podařilo se mi i díky schůzkám s Ing. Tomašem Novotným prostoupit do světa paralelního vykonávání příkazů.

Ze začátku programování této aplikace jsem se snažil vybudovat si všechny komponenty svépomocí, což by bylo pro další vývoj naivní a zbytečné. Na druhou stranu jsem měl možnost docenit při psaní pozdějších modulů užitečnost již napsaných knihoven a frameworků jako jsou Backbone.js a Socket.io.

Aplikace, zejména CSV modul, by měla mít další využití v projektu ExtBrain[1]. Už během vývoje vyvstaly další návrhy na vylepšení, například filtrace řádků v tabulce CSV podle zadaného kritéria. Tyto a jiné podněty jsou buď částečně zpracovány nebo na nich budu dále pracovat.

[1] http://extbrain.felk.cvut.cz/

# LITERATURA


[1] *Async Javascript* [online]. [cit. 20. 4. 2012]. Dostupné z: `http://leanpub.com/asyncjs`.

[2] *Zappa framework* [online]. [cit. 14. 4. 2012]. Dostupné z: `http://zappajs.org/`.

[3] CROCKFORD, D. *JavaScript: The Good Parts*. O'Reilly Media / Yahoo Press, 2008. ISBN 978-0596517748.

[4] HERRON, D. *Node Web Development*. Packt Publishing, 2011. ISBN 978-1-849515-14-6.

[5] STEFANOV, S. *JavaScript Patterns*. O'Reilly Media, 2010. ISBN 978-0596806750.

[6] ZAKAS, N. C. *High Performance JavaScript*. O'Reilly Media / Yahoo Press, 2010. ISBN 978-0-596-80279-0.

# Přílohy

PŘILOHA
# A

# ZKRATKY

| Zkratka | Význam |
|---|---|
| DOM | Document Object Model |
| JSON | JavaScript Object Notation |
| MVC | Model View Controller |
| CSS | Cascading Style Sheets |
| DRY | Don’t Repeat Yourself |
| AJAX | Asynchronous JavaScript and XML |
| API | Application Programming Interface |
| IE | Internet Explorer |
| WYSIWYG | What You See Is What You Get |
| W3C | World Wide Web Consortium |
| REST | Representational State Transfer |
| MIME | Multipurpose Internet Mail Extensions |

PŘILOHA
# B

# VYSVĚTLIVKY

| Kontext | Výraz | Význam |
|---|---|---|
| Asynchronní aplikace | emit | Vyslat událost, v asynchronních systémech vyvolá spuštění všech posluchačů naslouchajících dané události |
| | event | Spustitelná událost. |
| | listener | Funkce, čekající na spuštění události, kdy se provede. |
| JavaScript | scope | Každá proměnná s prefixem this se odkazuje na rámec, ve kterém je zavolána. Tento rámec určuje, kde bude program tuto proměnnou hledat, a nazývá se anglicky scope. |
| | třída | Javascript je objektově orientovaný jazyk, nicméně neobsahuje třídy. Dědičnost a vlastnosti objektu jsou určovány pomocí tzn. prototypů[5]. V textu používám výraz třída, protože v objektově orientovaném kódu se jedná o standartní označení. |

PŘILOHA
# C

# OBSAH CD

- text/
  - src/ - zdrojové kódy textu bakalářské práce ve formátu LyX
  - pdf/ - tisknutelný formát pdf bakalářské práce
- editor/
  - app/ - zdrojové kódy klientské části aplikace
  - lib/ - zdrojové kódy serverové části aplikace
  - node_module/ - balíčky použité v aplikaci
  - public/ - statické soubory (CSS, obrázky)
  - views/
- readme.txt