


Informační systém vydavatelství

Martin Zajíc

Diplomová práce
2014

 Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
akademický rok: 2013/2014

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Bc. Martin Zajíc**
Osobní číslo: **A11526**
Studijní program: **N3902 Inženýrská informatika**
Studijní obor: **Počítačové a komunikační systémy**
Forma studia: **prezenční**

Téma práce: **Informační systém vydavatelství**
Téma anglicky: **A Publishing House Information System**

Zásady pro vypracování:

1. Provedte analýzu informačního systému vydavatelství.
2. Navrhněte funkcionalitu informačního systému a vytvořte jeho model.
3. Navrhněte MongoDB databázi informačního systému.
4. Vytvořte prototyp Frontendu informačního systému pomocí AngularJS,HTML5,CSS3.
5. Vytvořte prototyp Backendu informačního systému v NodeJS.
6. Implementujte funkcionalitu editace a tvorby knih pomocí značkovacího jazyka TeX.
7. Věnujte pozornost zabezpečení aplikace.

Rozsah diplomové práce:

Rozsah příloh:

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

1. Ng-book: The Complete Book on AngularJS [online]. Fullstack io, 2013 [cit. 2014-02-01]. ISBN 978-0991344604. Dostupné z: <https://www.ng-book.com/>.
2. Mastering Node.js [online]. Packt Publishing, 2013 [cit. 2014-02-01]. ISBN 978-1782166320. Dostupné z: <http://visionmedia.github.io/masteringnode/>.
3. POYNTER, Dan. Dan Poynters self-publishing manual: how to write, print and sell your own book. 15th ed., completely rev. Santa Barbara, Calif.: Para Pub., 2006, 463 p. ISBN 978-156-8601-342.
4. DUCKETT, Jon. HTML: design and build websites. Indianapolis, IN: Wiley, c2011, 490 p. ISBN 11-180-0818-9.
5. OETIKER, Tobias, Hubert PARTL, Irene HYNA a Elisabeth SCHLEGL. The Not So Short Introduction to LATEX [online]. [cit. 2014-02-01]. Dostupné z: <http://www.ctan.org/tex-archive/info/lshort/english/>.

Vedoucí diplomové práce:

doc. Ing. Zdenka Prokopová, CSc.

Ústav počítačových a komunikačních systémů

Datum zadání diplomové práce:

7. února 2014

Termín odevzdání diplomové práce:

27. května 2014

Ve Zlíně dne 7. února 2014



prof. Ing. Vladimír Vašek, CSc.
děkan



prof. Ing. Karel Vlček, CSc.
ředitel ústavu

ABSTRAKT

Tato diplomová práce v teoretické části definuje základní pojmy, jako je Informační systém, dále pak Coffeescript, mean stack, typografický systém LaTeX a verzovací systém GIT. Úvodní část je věnována rozboru možností informačních systémů pro knihkupectví. Dále pak obsahuje stručný úvod do jazyků CoffeeScript, JADE a LESS a také do frameworků NodeJS, AngularJS, ExpressJS a HapiJS. Na závěr teoretické části je popsán verzovací systém GIT a typografický systém LaTeX.

V praktické části je rozebrána funkcionalita výsledného systému, struktura aplikace v oblasti databáze, backendu a frontendu. Jsou také popsány způsoby zabezpečení aplikace a jejího nasazení do produkčního prostředí.

Klíčová slova: kniha, knihkupectví, JavaScript, CoffeeScript, framework, NodeJS, AngularJS, LESS, LaTeX, Cloud, Gulp, HapiJS, MongoDB, Jade, API, frontend, backend

ABSTRACT

This thesis work includes the terms such as information system in it's theoretical part. Further, the terms such as Coffeescript, mean stack, typography system LaTeX and version control system GIT. The introductory part is dedicated to the analysis of bookstore information systems. Then it also includes brief introduction to CoffeeScript, JADE, LESS and frameworkrs NodeJS, AngularJS, ExpressJS and HapiJS. At the end of the theoretical part, there is described typography system LaTeX and version control system GIT.

In the practical part, there is analysed functionality of upcoming system and it's structure on database, frontend and backend. And at the end of this part, there are mentioned ways how to secure aplication and prepare to production envirimnt including deploying.

Keywords: book, bookstore, JavaScript, CoffeeScript, framework, NodeJS, AngularJS, LESS, LaTeX, Cloud, Gulp, HapiJS, MongoDB, Jade, API, frontend, backend

Chtěl bych poděkovat především své vedoucí doc. Ing. Zdence Prokopové, CSc., za její cenné rady a připomínky. Dále bych chtěl poděkovat celé své rodině a přítelkyni za podporu po celou dobu mého studia. A také Ing. Petru Šilhavému, Ph.D. a Ing. Oto Dočkalovy za rady a pomoc při tvorbě diplomové práce.

”Nejlepší knihy jsou takové, které člověku říkají, co už sám ví.”

GEORGE ORWELL

Prohlašuji, že

- beru na vědomí, že odevzdáním bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo –bakalářskou práci nebo poskytnout licenci k jejímu využití jen s předchozím písemným souhlasem Univerzity Tomáše Bati ve Zlíně, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše);
- beru na vědomí, že pokud bylo k vypracování bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně

.....
podpis diplomanta

OBSAH

| | |
|---|-----------|
| ÚVOD | 11 |
| I TEORETICKÁ ČÁST | 13 |
| 1 INFORMAČNÍ SYSTÉM | 14 |
| 1.1 EXISTUJÍCÍ INFORMAČNÍ SYSTÉMY PRO KNIHOVNY A KNIHKU- PECTVÍ..... | 14 |
| 1.2 TRENDY V INFORMAČNÍCH SYSTÉMECH | 14 |
| 1.3 PROCES VYDÁVÁNÍ KNIH | 15 |
| 2 POUŽITÉ TECHNOLOGIE | 16 |
| 3 COFFEE-SCRIPT | 16 |
| 3.1 NĚKOLIK UKÁZEK KÓDU V COFFEESCRIPU/JAVASCRIPTU | 17 |
| 3.2 ALTERNATIVY | 19 |
| 3.3 BROWSERIFY | 20 |
| 4 NODEJS | 21 |
| 4.1 EVENT-LOOP | 21 |
| 4.2 BALÍČKOVACÍ SYSTÉM NPM | 22 |
| 4.3 EXPRESSJS | 23 |
| 4.4 MONGOOSE ODM | 24 |
| 4.5 HAPIJS (HTTP API)..... | 24 |
| 4.5.1 Error handler | 25 |
| 4.5.2 Pack | 25 |
| 4.5.3 Composer | 25 |
| 4.5.4 Plugin | 26 |
| 4.5.5 Cyklus requestu | 27 |
| 5 MONGODB | 28 |
| 6 ANGULARJS | 29 |
| 6.1 STRUKTURA | 30 |
| 6.2 ROUTER..... | 32 |
| 6.3 SCOPE | 33 |
| 6.4 CONTROLLER | 33 |
| 6.5 DIRECTIVE | 34 |
| 6.6 FACTORY | 34 |
| 6.7 FILTER..... | 35 |

| | | |
|-----------|--|-----------|
| 6.8 | ANIMACE | 35 |
| 7 | LESS | 36 |
| 7.1 | TWITTER BOOTSRAP | 37 |
| 7.2 | LESSHAT..... | 38 |
| 8 | JADE | 39 |
| 9 | BOWER | 40 |
| 10 | GULPJS | 41 |
| 11 | SYSTÉM PRO SPRÁVU VERZÍ GIT | 42 |
| 11.1 | ZÁKLADNÍ PŘÍKAZY A GITHUB..... | 42 |
| 11.1.1 | Nastavení gitu..... | 42 |
| 11.1.2 | Vytvoření repozitáře a první commit..... | 43 |
| 11.1.3 | Správa repozitáře | 44 |
| 12 | LATEX | 45 |
| 12.1 | ONLINE LATEX EDITORY | 46 |
| 12.2 | ACE EDITOR | 47 |
| II | PRAKTICKÁ ČÁST | 49 |
| 13 | FUNKCIONALITA | 50 |
| 13.1 | MODULÁRNÍ APLIKAČNÍ FUNKCIONALITA..... | 50 |
| 13.2 | UŽIVATELSKÁ FUNKCIONALITA | 51 |
| 13.3 | PROCES SCHVALOVÁNÍ KNIHY | 51 |
| 14 | ADRESÁŘOVÁ STRUKTURA PROJEKTU | 51 |
| 15 | DATABÁZE | 54 |
| 15.1 | NÁVRH/SCHÉMA | 54 |
| 15.1.1 | Users | 54 |
| 15.1.2 | Books | 57 |
| 15.1.3 | Orders..... | 59 |
| 15.1.4 | Project..... | 61 |
| 15.1.5 | Branches | 62 |
| 15.1.6 | Config..... | 64 |
| 15.1.7 | Troubleshoot..... | 64 |
| 15.1.8 | Pages | 65 |
| 15.1.9 | Logs..... | 66 |
| 15.2 | BUDOUCÍ VÝVOJ..... | 67 |
| 15.3 | ZABEZPEČENÍ | 68 |

| | | |
|-----------|---|-----------|
| 15.3.1 | Nastavení zabezpečení MongoDB | 68 |
| 16 | BACKEND | 69 |
| 16.1 | STRUKTURA | 69 |
| 16.2 | API..... | 69 |
| 16.2.1 | /me | 70 |
| 16.2.2 | /book..... | 71 |
| 16.2.3 | /user..... | 71 |
| 16.2.4 | /branch..... | 71 |
| 16.2.5 | /order | 72 |
| 16.2.6 | /project..... | 72 |
| 16.2.7 | /page | 73 |
| 17 | FRONTEND | 73 |
| 17.1 | STORE (E-SHOP)..... | 73 |
| 17.2 | CREATE (SPRÁVA VLASTNÍCH KNIH) | 73 |
| 17.3 | EDIT (EDITOR KNIH) | 74 |
| 17.4 | ADMIN (ADMINISTRACE) | 74 |
| 17.5 | DALŠÍ ČÁSTI..... | 74 |
| 18 | ZABEZPEČENÍ..... | 75 |
| 18.1 | JWT (JSON WEB TOKEN) | 75 |
| 18.1.1 | Implementace backend (NODEJS)..... | 76 |
| 18.2 | DALŠÍ ZRANITELNOSTI | 76 |
| 18.3 | PŘIHLÁŠENÍ A SOUKROMÍ UŽIVATELŮ | 77 |
| 19 | GULP | 77 |
| 19.1 | STRUKTURA | 77 |
| 19.2 | DEVELOPMENT/PRODUCTION..... | 78 |
| 20 | NASAZENÍ APLIKACE (SERVER,CLOUD) | 80 |
| 20.1 | MONGODB | 81 |
| 20.1.1 | Cloudové služby | 82 |
| 20.2 | NODEJS..... | 83 |
| 20.2.1 | Vlastní server..... | 83 |
| 20.2.2 | Cloud..... | 84 |
| | ZÁVĚR..... | 86 |
| | SEZNAM POUŽITÉ LITERATURY | 88 |
| | SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK | 93 |

| | |
|----------------------|----|
| SEZNAM OBRÁZKŮ | 93 |
| SEZNAM PŘÍLOH | 95 |

ÚVOD

Knihkupectví, knihy, jejich prodej a tisk jsou tu s námi již nějakou dobu, vždyť nejstarší knize je přisuzováno stáří 2600 let a je celá ze zlata[6][8]. Od té doby, ale knihy prošly dlouhým vývojem a od zlaté vazby se upustilo. Největším mylníkem pro knihy, a možná i jedním z nejdůležitějších vynálezů pro rozvoj lidstva, se stal v 15.století knihtisk, se kterým je spojeno především jméno **Johanna Gutenberga**[9]. Jeho vynález vytváření kopie tiskem z výšky, se pro tisk používá dodnes.

Vývoj knih, potřeba organizace a distribuce dala za vznik prvním knihkupectvím, nejstarší dosud fungující knihkupectví, má za sebou téměř 300 let existence[7]. Knihkupectví, ostatně jako většina prodejních míst měla, a v některých případech stále má "*informační systém*" v papírové podobě. Tato nutnost se však změnila s dobou počítačů a výpočetní techniky, která dala možnosti rychlé a efektivní organizace pomocí různých informačních systémů.

Změnilo se také postavení knihkupectví. Zatímco dříve bylo knihkupectví místem, kde se lidé scházeli nejen kvůli koupi knih, ale také za účelem výměny informací, je dnes knihkupectví místo, které má především generovat zisky. A tuto úlohu zjednodušují právě informační systémy.

Ukázkou kam se až může takový informační systém a celé knihkupectví v průběhu let dostat je především veřejně známý Amazon, který se dostal na vrchol díky využití moderních technologií a neustálé inovaci. Vdyť již v roce 1997 nabídl objednávky přes internet na jedno kliknutí *1-ClickTM*[10]. Dnes Amazon prodává široký sortiment zboží a distribuuje jej po celém světě. Potřeba růstu Amazonu dala za vznik také množství služeb z nichž je v dnešní době velmi slyšet o EC2¹⁾, která způsobila poprask mezi cloudovými službami, přitom její účel byl původně pouze využít prostředků serverů Amazonu v době kdy knihkupectví nemělo tak velkou návštěvnost. Dnes se jedná o předního poskytovatele cloudových služeb[11].

K vytvoření moderního informačního systému je třeba dobrý návrh a kvalitní technologie. Pokud se jedná o informační systém, který je určen především pro běh ve webovém prostředí, což odpovídá moderním standardům a většinou také základním požadavkům, nabízí se řada technologií a programovacích jazyků.

Při tvorbě moderních aplikací se mění důležitost programovacího jazyka a velký zřetel se bere na ekosystém, který kolem sebe programovací jazyk vytváří. Kvalitní ekosystém a široká podpora knihoven může ušetřit nezanedbatelné množství času a peněz při vývoji. Vývoj nejen webových informačních systémů, ale také webů a webových aplikací se v dnešní době rozděluje na dvě samostatně vyvíjené části a to client a server. U obou

¹⁾EC2 - Elastic cloud

částí existuje velké množství technologií, které je možné pro jejich vývoj použít. Vzhledem k tomu, že pro vývoj klientské části se využívá většinou programovací jazyk JavaScript, je nanejvýš přívětivá možnost použití stejného jazyka na serveru. A javascript byl také vybrán jako jazyk pro vytvoření prototypu aplikace, která bude spolu s použitými technologiemi popsána v následující diplomové práci.

I. TEORETICKÁ ČÁST

1 INFORMAČNÍ SYSTÉM

Přesná definice pojmu Informační systém neexistuje a ani ji nelze jednoduše vytvořit, neboť každý uživatel či tvůrce Informačního systému používá různé terminologie a zdůrazňuje jiné aspekty. Můžeme však říci, že Informační systém (IS) lze chápat jako systém vzájemně propojených informací a procesů, které s těmito informacemi pracují. Přičemž pod pojmem procesy rozumíme funkce, které zpracovávají informace do systému vstupující a transformují je na informace ze systému vystupující. Zjednodušeně můžeme říci, že procesy jsou funkce zabezpečující sběr, přenos, uložení, zpracování a distribuci informací. Pod pojmem informace pak rozumíme data, která slouží zejména pro rozhodování a řízení v rozsáhlejších systémech[12].

Informační systémy obecně se stávají velmi komplexním řešením pro kompletní vedení firem jakýchkoliv rozměrů. Zajišťují širokou škálu služeb, od komunikace se zákazníky přes ekonomické systémy až po prodej a tvorbu obsahu nebo pro evidenci skladu. Takové systémy jsou vyvíjeny většinou dlouhodobě, na míru zákazníkovi a rozvíjeny společně s jeho růstem a potřebami. Může se také jednat o různá spojení stávajících informačních systémů, která jsou často spravována odděleně, nenabízí přehled a porovnání údajů jednotlivých částí.

1.1 Existující informační systémy pro knihovny a knihkupectví

Velmi častou aplikací, která se používá v České republice je kombinace účetního systému Pohoda <http://www.ucetni-systemy.cz/pohoda-standard/c-4468/>, která obsahuje kompletní správu firmy včetně pokladen, skladů a účetnictví. A dokáže tato data synchronizovat s vybranými e-shopy, např. PrestaShop <http://www.prestashop.com/>. E-commerce a komunikace se zákazníkem je pak na e-shopu, který většinou slouží pro klasický prodej. Komunikace mezi zaměstnanci probíhá prostřednictvím e-mailů nebo IM, komunikace se zákazníkem, buď pomocí e-mailů nebo "chatu" přímo v okně prohlížeče.

Další systémy jsou na tom podobně, co ovšem chybí, je např. komunikace s tvůrcem a přehled tvůrce o prodeji jeho díla. Či přímo rozhraní pro tvorbu knihy online.

1.2 Trendy v informačních systémech

Svět informačních systémů se stále rychle vyvíjí. Dříve byl vyspělý informační systém *DOSový* program, který se připojil na server, mnohdy ani to ne, a obsluha jej ovládala klávesnicí. Dnes jsou požadavky na informační systémy mnohem větší a zákazníci ve

většinu případů vyžadují dostupnost kdekoli na světě pomocí webového prohlížeče v mobilu nebo tabletu.[15]

Dalším trendem jsou sociální sítě jako Facebook, LinkedIn, Twitter apod. Informační systémy v dnešní době zahrnují podporu pro sociální sítě. Ale nejedná se pouze o dnes velmi populární přihlašování bez nutnosti "registrace", ale i např. zkoumání povědomí o značce nebo komunikaci se zákazníky pomocí sociálních sítí a realtime chatu.[15]

1.3 Proces vydávání knih

Proces schvalování a vydávání knih je u každého vydavatelství jiný, různě také přístupují jak vydavatelé, tak i tvůrci ke kvalitě vydávaného díla, jeho formátování, kontroly pravopisu a jinými aspekty spojenými s vydáním knihy.

Pokud chce autor vydat knihu, je častým krokem využít některého vydavatelství. Těch je velké množství a často se specializují na vydávání určitého spektra knih. Avšak pokud by o autorovu knihu nemělo žádné vydavatelství zájem, případně nemůže-li autor akceptovat podmínky, které si vydavatelství klade, máme dvě možnosti. První z nich je zhostit se práce vydavatele sám. I přesto, že to není jednoduchá práce (musí si zajistit sazbu, tvorbu obálky, tisk a vazbu, registraci ISBN identifikace knihy a z ní plynoucí povinnosti vůči státu, a hlavně distribuci), vydává se touto cestou značné množství autorů.[43]

Je-li autor schopen vykonat všechny kroky od napsání rukopisu po slavnostní zahájení distribuce zvládnout sám, vykonává tzv. **self-publishing**¹⁾[3].

Mnoho autorů nemá potřebné znalosti, technické zázemí a čas se složitému vydavatelstvímu procesu věnovat. Právě pro ně existuje možnost profesionální odvětví, kterému se říká **Vanity Press** - česky bychom řekli asi „zakázkové vydavatelství“[43]. Pod pojmem vanity press je možné si představit vydavatelství, která pracují stejně jako ta běžná, ale na podnikatelské riziko autora. Poskytují více či méně komplexní služby v závislosti na tom, co autor požaduje. Na konci jejich výrobního procesu je však v každém případě hotová a profesionálně vypadající kniha.[43] Možnost vydání knihy online formou vanity press, poskytuje dnes velké množství vydavatelství v České republice např. Librix www.knihovnicka.cz nebo Tiskový Express www.tiskovyexpress.cz. Žádné neposkytuje možnost psaní knihy přímo online na webu vydavatele.

Celý proces vydání vlastní knihy lze rozložit na jednotlivé části.[42]

- Prvním krokem by mělo být zamyšlení nad tím, co by chtěl autor psát a zda se mu vynaložené úsilí vrátí.

¹⁾Self-publishing - případ procesu vydání knihy, kdy vše obstarává autor

- Sepsání knihy
- Představa o výsledné podobě knihy a poptávka v tiskárně
- Stanovení rozpočtu
- Získání ISBN
- Dokončení podkladů
- Oznamovací povinnost
- Povinné výtisky
- Prodej

Na konci procesu přichází část, kdy autor získává za své dílo peníze a která teprve ukáže zda se úsilí vyplatilo.

Diplomová práce je zaměřena na proces, kdy od počátku tvorby může vydavatel zasahovat do díla autora, protože autor začíná dílo tvořit přímo v systému vydavatele. Výhodou je, že je práce autorovi dostupná všude kde se dostane k počítači, může být od počátku tvořena v šabloně vydavatele a dodržovat tak vysoký standard kvality. Samozřejmě je možné dílo nahrát již hotové podobě.

2 POUŽITÉ TECHNOLOGIE

K vývoji bylo využito aktuálních nástrojů používaných pro vývoj moderních webových aplikací, které usnadňují práci programátorům a výrazně zrychlují tvorbu aplikace. Důraz je kladen na rychlost a škálovatelnost takové aplikace.

3 COFFEE-SCRIPT



Obr. 1. Logo coffeescript

Coffee-script je malý a jednoduchý jazyk, který se transpiluje ¹⁾ do javascriptu. Přispívá k jednoduchosti kódu a urychluje jeho psaní, protože vypouští závorky, přidává někte-

¹⁾transpilovat znamená převádět kód z jednoho programovacího jazyka do druhého

rou funkcionalitu ES6 ²⁾ výrazně zjednodušuje cykly a mnoho dalších funkcí, které jsou často používány.

3.1 Několik ukázek kódu v coffeescriptu/javascriptu

Přiřazení

```
#coffeescript
number = 42
opposite = true
#javascript
var number = 42;
var opposite = true;
```

Podmínky

```
#coffeescript
number = -42 if opposite
#number je -42 pokud je opposite true
date = if friday then sue else jill
#pokej je friday tak date je sue jinak jill
#javascript - stejný kód
if (opposite) {
  number = -42;
}
date = friday ? sue : jill;
```

Funkce není třeba psát se složenými závorkami vše s indentací patří do funkce.

```
#coffeescript
square = (x) ->
  x * x
#javascript
var square = function(x) {
  return x * x;
};
```

Objekty - objekty není třeba uzavírat do složených závorek.

```
#coffeescript
math =
  root: Math.sqrt
  square: square
  cube: (x) -> x * square x
#javascript
var math = {
  root: Math.sqrt,
  square: square,
  cube: function(x) {
    return x * square(x);
  }
};
```

Testování existence proměnné - otazník za proměnnou testuje její existenci a pokud elvis existuje vyskočí alert.

²⁾ES6 neboli ECMAScript6 je nástupce současného standardu ECMAScript5 používaného většinou prohlížečů, přináší podporu pro třídy, moduly, iterátor, yield, a další.

```
#coffeescript
alert "I knew it!" if elvis?
#javascript
if (typeof elvis !== "undefined" && elvis !== null) {
  alert("I knew it!");
}
```

Cykly

```
#coffeescript
eat =(food)->
  console.log(food)
foods = ['broccoli', 'spinach', 'chocolate']
eat food for food in foods when food isnt 'chocolate'
#eat(food) je zavolána pokud položka v poli není chocolate
#Výstup:
broccoli
spinach
#javascript
var eat = function(food){
  console.log(food);
}
var food, foods, _i, _len;
foods = ['broccoli', 'spinach', 'chocolate'];
for (_i = 0, _len = foods.length; _i < _len; _i++) {
  food = foods[_i];
  if (food !== 'chocolate') {
    eat(food);
  }
}
```

Třídy a dědičnost - jsou v CoffeScriptu velmi podobné standardu ES6 a jsou přeloženy na javascriptové prototypy.

```
#coffeescript
class Animal
  constructor: (@name) ->
    console.log @name
  move: (meters) ->
    console.log meters
class Snake extends Animal
  move: ->
    console.log "Syčí..."
    super 5
slepis = new Snake("Slepíš")
slepis.move()
#Výstup
Slepíš
Syčí...
5
```

```
#javascript
var Animal, Snake,
    __hasProp = {}.hasOwnProperty,
    __extends = function(child, parent) {
    for (var key in parent) {
        if (__hasProp.call(parent, key)) child[key] = parent[key];
    }
    function ctor() { this.constructor = child; }
    ctor.prototype = parent.prototype;
    child.prototype = new ctor();
    child.__super__ = parent.prototype;
    return child;
};
Animal = (function() {
    function Animal(name) {
        this.name = name;
    }
    Animal.prototype.move = function(meters) {
        return console.log(meters);
    };
    return Animal;
})();
Snake = (function(_super) {
    __extends(Snake, _super);
    function Snake() {
        return Snake.__super__.constructor.apply(this, arguments);
    }
    Snake.prototype.move = function() {
        console.log("Syčí...");
        return Snake.__super__.move.call(this, 5);
    };
    return Snake;
})(Animal);
```

3.2 Alternativy

Alternativ coffeescriptu respektive javascriptu, je hned několik. Mezi nejzajímavější patří především Type-script a DART. Oba razí odlišnou ideologii a jsou financovány softwarovými giganty Microsoftem a Googlem. **Typescript** - je, co se ideologie týče, podobný více coffeescriptu, protože se jedná pouze o transpiler. S tím rozdílem, že nijak zvláštně neupravuje vzhled kódu nebo syntaxi jazyka, ale přidává především datové typy, které javascript samotný nemá. Tato vlastnost je ovšem následně ztrácena překladem do javascriptu.[45] Oproti tomu **DART** je zcela nový jazyk určený pro běh v prohlížeči, který má možnost kompilace do javascriptu.[36] Momentálně není DART implementován v žádném z majoritních prohlížečů, pouze v projektu dartium, což je fork projektu chromium určený speciálně pro běh DARTu. DART je velkým příslibem pro budoucnost vývoje webových aplikací.



Obr. 2. Logo Browserify

3.3 Browserify

ES6 přinese dlouho čekanou vlastnost a to budou **Moduly**. Moduly umožní v javascriptu importovat funkcionalitu jiných modulů a svoji naopak zpřístupňovat.[24][25] Takto bude vypadat modul:

```
# bod.coffee
Bod=(x, y)->
  @x = x
  @y = y
export = Bod
# app.coffee
import "bod" as Bod
new Bod(0, 0)
```

Modul bod exportuje funkci Bod a modul app importuje modul bod. App pak pomocí operátoru new vytvoří instanci modulu Bod se vstupními proměnnými x,y.

Ale protože bude ještě chvíli trvat než bude ES6 hotov a podporován napříč prohlížeči, byly vytvořeny různé implementace některých jeho nových funkcí pomocí ECMAScript 5.1, jednou z funkcí, které byly implementovány jsou **Moduly**.

Těchto implementací existuje hned několik. NodeJS používá CommonJS moduly na straně serveru a pro prohlížeč jsou k dispozici AMD³⁾ moduly, které používá RequireJS. Další možností jsou však BrowserifyJS moduly, které mají stejnou syntaxi jako CommonJS moduly využívané v NodeJS a umožňují tak importovat CommonJS moduly přímo do prohlížeče.[26]

```
# obsah.coffee
module.exports = (x, y) ->
  return x*y
# app.coffee
obsah = require("./obsah")(2,2)
```

V souboru obsah.coffee exportujeme funkci, která přijímá parametry x,y a vrací obsah obdélníku. V souboru app.coffee voláme soubor obsah.coffee, který je ve společné složce, to je však funkce takže jí můžeme předat vstupní parametry 2,2. v proměnné obsah pak bude uložen výsledek 4.

Moduly umožňují asynchronní načítání svých závislostí, které se načtou vždy pouze

³⁾AMD - Asynchronous Module Definition

jednou, vytváří hierarchii aplikace a strom závislostí jednotlivých komponent. Také umožňuje snadnější rozdělení aplikace na jednotlivé soubory.[26][27]

4 NODEJS

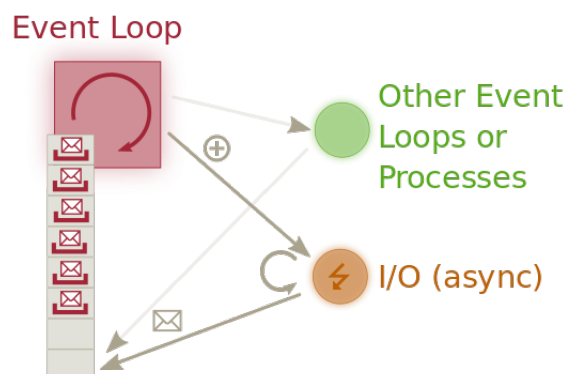


Obr. 3. Logo NodeJS

NodeJS je framework postavený na javascriptu a enginu V8¹⁾ od společnosti Google. Jednou z hlavních výhod nodejs je **event-driven** architektura a vlastní balíčkový systém **npm**. [2]

4.1 Event-loop

Event-loop je jeden ze způsobů zpracování programu. Oproti více zažitým způsobům, kdy voláme funkce a jejich návratové hodnoty posíláme do dalších funkcí. V JavaScriptu máme seznam eventů, které když se stanou (v prohlížeči dojde ke stisknutí tlačítka, na serveru dojde k příchodu HTTP požadavku atd.), tak se zavolá nastavená obsluhující funkce (callback). V zásadě tak můžeme nastavit callbacky na vstup z terminálu, příchod dat na síťový port a periodicky spouštěný event (pomocí setInterval) najednou, to vše bez vláken či řešení synchronizace. Pokud pak někdo bude psát do terminálu a zároveň dojde k připojení přes síť, zpracuje se nejdříve jeden event, poté druhý, atd.. [13][2]



Obr. 4. Grafické znázornění Event-Loop

¹⁾V8 je javascript engine používaný ve webovém prohlížeči Chrome.

4.2 Balíčkovací systém NPM

Zejména v posledních letech začali být velmi populární balíčkovací systémy pro různé programovací jazyky či frameworky inspirované balíčkovacími systémy z operačního systému Linux, respektive z různých distribucí OS Linux. Vše začal jazyk Ruby a především framework Ruby On Rails se systémem tzv. Gemů. PHP dostalo balíčkovací systémy composer (aplikační moduly) a pecl (systémové moduly). Python má pip atd.. NodeJS obsahuje svůj vlastní balíčkovací systém NPM (Node Package Manager). Disponuje velmi jednoduchým ovládáním a přináší programátorům řešení závislostí jejich aplikací a knihoven napříč operačními systémy.[2]

Důležitým prvkem je konfigurační soubor *package.json*, obsahuje nejen informace o závislostech programu, ale také třeba autora, verzi, popis, repozitář a mnoho dalších údajů využívaných dalšími moduly aplikace.

Ukázkový package.json

```
{
  "name": "moje-aplikace",
  "version": "0.0.1",
  "dependencies": {
    "coffee-script": "^1.7.1",
    "express": "3.5.1",
    "mongoose": "^3.8.8",
  },
  "devDependencies": {
    "angular": "=1.2.16",
    "angular-ui-router": "^0.2.10",
  },
  "engines": {
    "node": ">=0.8.0"
  },
  "browserify": {
    "transform": [
      "coffeeify",
      "brfs",
      "envify"
    ]
  },
  "scripts": {
    "dev": "gulp dev",
    "prod": "gulp prod"
  }
}
```

Ze souboru můžeme vyčíst, že jméno aplikace je "moje-aplikace", že se nachází ve verzi "0.0.1", jejími závislostmi jsou "coffee-script,express a mongoose", vývojovými závislostmi pak "angular a angular-ui-router", k běhu potřebuje nodejs ve verzi 0.8.0 a vyšší. Modul browserify používá transformace: "coffeeify, brfs a envify" a na závěr jsou deklarované scripty.

NPM také obsahuje interaktivní shell pro vytváření nových projektů, který vytvoří

package.json za uživatele. Nově instalované moduly jsou při přidání přepínače `-save` nebo `-save-dev` přidávány také automaticky.

Základní příkazy:

install nainstaluje balíčky v package.json

install <balíček> nainstaluje balíček

update aktualizuje balíčky v package.json

search vyhledávání balíčků

4.3 ExpressJS

ExpressJS je webový framework inspirovaný webovým frameworkem Sinatra napsaným v ruby. Jedná se o malý flexibilní a robustní framework pro psaní single-page, multi-page a hybridních stránek postavený na NodeJS.[14]



Obr. 5. Logo ExpressJS

Express je ve stylu Sinatra designován jako velmi jednoduchý a obsahuje tak velmi malé množství funkcí, které se rozšiřuje pluginy a knihovnamy. Express tak umí především zpracovávat routy (url cesty), html požadavky (POST,GET,PUT,PATCH,OPTION) a tělo requestů. Umí také zpracovávat a poskytovat statické soubory a cookies.[14]

Ukázka velmi jednoduché express aplikace:

```
express = require 'express'
server = express()
port = 3000;
app.get "/", (req, res, next)->
  res.render "views/root"
app.get "/hello", (req, res, next) ->
  res.send('world')
app.listen port, ->
  console.log("Listening on #{port}")
```

Po spuštění kódu se vytvoří http server na portu 3000, který má dvě definované routy: `/` - root aplikace vrátí soubor `views/root.html` a `/hello`, který vrátí řetězec "world".

4.4 Mongoose ODM

V posledních letech se stali velmi populární ORM frameworky nad MySQL databází jako je Doctrine. Pro MongoDB databázi v prostředí NodeJS je velmi populární ODM framework Mongoose. V MongoDB databázi neexistují schémata jako je tomu u SQL databází. Mongoose přináší podporu pro databázové schémata, validaci, virtuální položky, funkce pro práci se schématem a rozšiřitelné API.

Jednoduché schéma v Mongoose:

```
mongoose = require 'mongoose'
Schema = mongoose.Schema
User = new Schema
  email:
    type: String
    required: true
    unique: true
    index: true
  name:
    type: String
    required: false
  address:
    city: String
    state: String
  bought_books:
    [
      type: Schema.Types.ObjectId
      ref: "Book"
    ]
  updated_at:
    type: Date
    default: Date.now
  created_at:
    type: Date
    default: Date.now
module.exports = mongoose.model 'User', User
```

Schéma obsahuje také definované typy objektů a jejich validaci. Konkrétně toto schéma uživatele má objekty: **email**- indexovaný, unikátní a povinný, **name**- jméno-řetězec, **address**- který má vnořený objekt, který obsahuje město a stát, **bought_books**-pole referencí na knihy, které uživatel koupil, **updated_at, created_at**-data vytvoření a poslední úpravy uživatele.

4.5 HapiJS (HTTP API)

Pro NodeJS existuje opravdu obrovské množství nejrůznějších frameworků. Framework použitý pro tvorbu tohoto projektu se jmenuje **HapiJS** <http://spumko.github.io/>. Jedná se o jednoduše konfigurovatelný framework s podporou validací, cachování, autentizace atd. Zaměřuje se na psaní znovupoužitelného aplikačního kódu a eliminuje zdlouhavé vytváření infrastruktury.[23]

Ukázka vytvoření základního serveru s jednou routou `"/`:

```
Hapi = require("hapi")
# Vytvoří http server
server = Hapi.createServer("localhost", 8000)
# Root route vrátí text "hello word"
server.route
  method: "GET"
  path: "/"
  config:
    handler: (request, reply) ->
      reply "hello word"
# Spustí http server
server.start ->
  console.log "Server started at port " + server.info.port
return
```

4.5.1 Error handler

Hapi obsahuje build-in funkci pro vyvolání chybových odpovědí pomocí modulu `"boom"`. Následující vyvolá HTTP výjimku se stavovým kódem `"499"` a zprávou špatný dotaz.[35]

```
(request, reply) ->
  error = Hapi.error.badRequest("Špatný dotaz")
  error.output.statusCode = 499
  error.reformat()
  error.output.payload.custom = "abc_123"
  reply error
```

4.5.2 Pack

Pack je funkcionalita, která umožňuje sdružování množství serverů do jedné logické jednotky. Takový pack může sdružovat několik serverů, které mohou mít vlastní pluginy.[35]

```
Hapi = require("hapi")
pack = new Hapi.Pack()
pack.server 8000,
  labels: ["web"]
pack.server 8001,
  labels: ["admin"]
```

Vytvoření dvou serverů `"web"` a `"admin"`

4.5.3 Composer

Composer slouží k jednoduché konfiguraci `"Packu"` v jenom jediném objektu včetně registrace pluginů[35].

```
Hapi = require("hapi")
manifest =
  pack:
    cache: "catbox-memory"
  servers: [
    {
      port: 8000
      options:
        labels: ["web"]
    }
    {
      host: "localhost"
      port: 8001
      options:
        labels: ["admin"]
    }
  ]
  plugins:
    yar:
      cookieOptions:
        password: "secret"
composer = new Hapi.Composer(manifest)
```

Vytvoření dvou serverů "web" a "admin" s pluginem yar a jeho nastavením.

4.5.4 Plugin

Plugin pomáhá rozdělit aplikaci na malé části, které mohou mít vlastní routy, závislosti, modely, logiku apd. V různých kombinacích tak, aby byli použitelné v různých prostředích (development, testování, produkce).[35]

```
Hoek = require("hoek")
internals = defaults:
  version: "/version"
internals.version = 0.1
exports.register = (plugin, options, next) ->
  settings = Hoek.applyToDefaults(internals.defaults, options)
  if settings.version
    plugin.route
      method: "GET"
      path: settings.version
      handler: (request, reply) ->
        reply internals.version
    return
listPlugins = (server) ->
  plugins = []
  Object.keys(server.pack.list).forEach (name) ->
    item = server.pack.list[name]
    plugins.push
      name: item.name
      version: item.version
  plugin.expose "plugins", listPlugins
  next()
```

Jednoduchá ukázka pluginu, který přidává routu "/version", která vrátí jeho verzi a metodu listPlugins, která vypíše všechny pluginy.

4.5.5 Cyklus requestu

Každý příchozí request musí projít sérií předefinovaných kroků spolu s dalšími kroky přidanými programátorem.[35]

'**onRequest**' do objektu procházejícího tímto bodem jsou přidány metody "request.setUrl(url)" a '

Tyto metody mají vliv na routování a mohou být využity pro přepsání pravidel routování. Router není ještě inicializován. Jedná se o bod, kterému je možné přiřadit vlastní metody a změnit tak chování aplikace.

Procházení rout pomocí request path

Parsování cookies

'**onPreAuth**' další rozšiřující, bod, kterým je možné aplikaci rozšířit

Authenticate request

Čtení a parsování payload

Authenticate request payload

'**onPostAuth**' další rozšiřující, bod, kterým je možné aplikaci rozšířit

Validace parametrů cesty

Zpracování query (JSONP)

Validace query

Validace payload

'**onPreHandler**' další rozšiřující, bod, kterým je možné aplikaci rozšířit

Přednastavení routeru

Router

'**onPostHandler**' další rozšiřující, bod, kterým je možné aplikaci rozšířit. V tomto kroce může být modifikována odpověď serveru.

Validace payload odpovědi

'**onPreResponse**' další rozšiřující, bod, kterým je možné aplikaci rozšířit

Odeslání odpovědi (může vyvolat 'internalError')

Emits 'response' event

Emits 'tail' event

5 MONGODB



Obr. 6. Logo MongoDB

MongoDB je dokumentová NoSQL databáze s dokumenty ve formátu json a dynamic-kým schématem. Počátek jejího vývoje spadá do roku 2007 a je vyvíjena společností 10gen, momentálně se nachází ve verzi 2.6.

MongoDB funguje jako klasický databázový server, ke kterému se klienty připojují přes síť (obvykle na portu 27017) a komunikují s ním speciálním protokolem (ovšem na portu 28017 najdete jednoduché HTTP rozhraní). Součástí instalace je knihovna pro C++, jako součásti projektu jsou, ale vyvíjeny a podporovány i knihovny pro Javu, Python, Ruby, PHP a další jazyky.

MongoDB navíc obsahuje mnoho pokročilých funkcí, jako je replikace databáze, balancování zátěže (load balancing), ukládání souborů (file storage), indexace a další,...

Základním rozhraním pro MongoDB je javascriptová konzole, kterou lze spustit příkazem **mongo**.^[16]

Vkládání záznamů:

```
db.users.insert(  
  {  
    name: "Franta",  
    email: "franta@franta.cz",  
    address: {  
      city: "zlín"  
    }  
  }  
)
```

Vytvoří záznam v kolekci uživatelé se jménem "Franta", e-mailem "franta@franta.cz" a adresou "zlín".

Hledání:

```
db.users.find()  
#vrátí všechny záznamy  
db.bands.find({name: "Franta"})  
#vrátí záznamy se jménem Franta  
db.bands.find({name: /^Fra.*})  
#regulérní výrazy v~dotazech  
#vrátí všechny záznamy začínajících na Fra
```

Update:

```
db.users.update(  
  {  
    _id: ObjectId("4b76ef5780b71853fe6c89b6")  
  }, {  
    $set: {  
      email: "franta2@seznam.cz"  
    }  
  })
```

provede update uživatele id 4b76ef5780b71853fe6c89b6 a změní jeho email na "franta2@seznam.cz"

Mazání:

```
db.users.remove({name: "Franta"})
```

smaže všechny uživatele se jménem "Franta"

reference:

```
#přidáme uživateli koupenou knihu  
db.users.update(  
  {  
    _id: ObjectId("4b76ef5780b71853fe6c89b6")  
  }, {  
    $push: {  
      bought_books: new DBRef("books",  
        ObjectId("4b7b1d6bb345126b62fb2230"))  
    }  
  })  
#vyhledáme uživatele  
db.users.findOne({_id: ObjectId("4b76ef5780b71853fe6c89b6")})  
#výstup  
{  
  "_id" : ObjectId("4b76ef5780b71853fe6c89b6"),  
  "name" : "Franta",  
  "email" : "franta@seznam.cz",  
  "books" : [  
    {  
      "$ref" : "books",  
      "$id" : ObjectId("4b7b1d6bb345126b62fb2230")  
    }  
  ],  
  "address" : {  
    "city" : "zlín"  
  }  
}  
#získáme knihy uživatele  
db.users.findOne({_id: ObjectId("4b76ef5780b71853fe6c89b6")})  
  .books.fetch()  
#výstup je stejný jako v předchozím případě jen v poli books  
# se místo reference objeví objekt konkrétní knihy.
```

6 ANGULARJS

Anglický originál popisu angularJS z repozitáře na portále github.com



Obr. 7. Logo AngularJS

AngularJS lets you write client-side web applications as if you had a smarter browser. It lets you use good old HTML (or HAML, Jade and friends!) as your template language and lets you extend HTML's syntax to express your application's components clearly and succinctly. It automatically synchronizes data from your UI (view) with your JavaScript objects (model) through 2-way data binding. To help you structure your application better and make it easy to test, AngularJS teaches the browser how to do dependency injection and inversion of control. Oh yeah and it also helps with server-side communication, taming async callbacks with promises and deferreds; and makes client-side navigation and deeplinking with hashbang urls or HTML5 pushState a piece of cake. The best of all: it makes development fun![17]

Překlad:

AngularJS umožňuje vytvářet tenké klientské aplikace, jako bychom měli chytřejší prohlížeč. Přitom můžete používat HTML (HAML, JADE a podobné) jako šablonovací systém a rozšířit jeho syntaxi tak, aby komponenty vyjadřovaly aplikaci jasně a stručně. Automaticky synchronizuje data v uživatelském rozhraní s vaším javascriptovým modelem, pomocí obousměrné datové vazby. Aby byla vaše aplikace lépe strukturovaná a snáze testovatelná, učí AngularJS váš prohlížeč jak na dependency injection a inversion of control. A abych nezapomněl, pomůže vám také s komunikací se serverem, zkrotit asynchronní callbacky, promises a deferreds; a také dělá navigaci v prohlížeči, pomocí deeplinking s hashbang url nebo html5 pushState snadnou věcí. A to nejlepší na konec: dělá vývoj webových aplikací zábavou.[17]

6.1 Struktura

Struktura kódu aplikace by měla mít nějaká pravidla, tak aby se v kódu vyznal i někdo jiný než autor samotný. Proto například google poskytuje online doporučenou strukturu kódu pro aplikace napsané v AngularJS. Online verze je dostupná na adrese: <https://google-styleguide.googlecode.com/svn/trunk/angularjs-google-style.html> Vzhledem k tomu, že se jedná o obsáhlejší dokument nebude nijak komentován. Struktura aplikace je ale z větší části ovlivněna právě tímto návrhem.

Co se týče struktury adresářové, existuje několik návrhů, které jsou určeny pro různě velké aplikace.

Velmi jednoduchá aplikace v AngularJS vypadá nějak takto:

```
css/  
img/  
js/  
  app.js  
  controllers.js  
  directives.js  
  filters.js  
  services.js  
lib/  
partials/
```

Vše je rozděleno do několika javascriptových souborů, které reprezentují jednotlivé moduly AngularJS, je o něco málo lepší, než jedno souborové aplikace tak, jak to dělá velké množství uživatelů jQuery. Jednotlivé soubory ale nerepresentují konkrétní funkcionalitu, takže se v projektu hůře orientuje.[18]

Tento problém řeší rozdělení jednotlivých modulů do několika sub-modulů, které již reprezentují svoji funkcionalitu.

```
controllers/  
  LoginController.js  
  RegistrationController.js  
  ProductDetailController.js  
  SearchResultsController.js  
common/  
  cartDirective.js  
  productFilter.js  
models/  
  CartModel.js  
  ProductModel.js  
  SearchResultsModel.js  
  UserModel.js  
services/  
  CartService.js  
  UserService.js  
  ProductService.js
```

V případě, že vytváříme, větší aplikaci, stane se pro nás toto schéma opět nedostatečným. A to proto, že ve větších aplikacích je už většinou třeba shlukovat veškeré části dané funkcionality u sebe, takže je ještě třeba strukturu ještě přeskupit.

```
cart/  
  index.less  
  CartModel.js  
  CartService.js  
common/  
  cartDirective.js  
  productFilter.js  
product/  
  search/  
    SearchResultsController.js  
    SearchResultsModel.js  
  ProductDetailController.js  
  ProductModel.js  
  ProductService.js  
  index.less  
  show.html  
user/  
  show.html  
  edit.html  
  index.less  
  LoginController.js  
  RegistrationController.js  
  UserModel.js  
  UserService.js
```

6.2 Router

Router je další důležitá přednost AngularJS. Jak již bylo v úvodu zmíněno routování umožňuje přepínat mezi url "hashtangem" a "html5.pushState". Router v AngularJS je navíc modulem, který lze vyměnit a je tedy možné místo něj použít třeba ui-router, který je jedním z nejpoužívanějších modulů pro AngularJS. Oproti angular-routeru je ui-router stavový, takže každá změna může být reprezentována svým stavem, který může být jednoduše vnořen a výsledkem je pak přehledná hierarchie routeru a url.

Ukázka jednoduché routeru v modulu angular-router

```
phonecatApp.config [  
  "$routeProvider"  
  ($routeProvider) ->  
    $routeProvider.when("/phones",  
      templateUrl: "partials/phone-list.html"  
      controller: "PhoneListCtrl"  
    ).when("/phones/:phoneId",  
      templateUrl: "partials/phone-detail.html"  
      controller: "PhoneDetailCtrl"  
    ).otherwise redirectTo: "/phones"  
]
```

Zde jsou nastaveny routy **/phones** , které mají controller: "PhoneListCtrl" a template: partials/phone-list.html a **/phones/:id**, která slouží pro získání jednoho telefonu. Pokud není routa v seznamu, je přesměrována na routu "/phones"

Ukázka ui-router

```
phonecatApp.config [
  "$stateProvider", "$urlRouterProvider",
  ($stateProvider, $urlRouterProvider) ->
    $urlRouterProvider.otherwise "/phones"
    $stateProvider.state("phones",
      url: "/phones"
      templateUrl: "partials/phones-list.html"
    ).state("phones.one",
      url: "/:id"
      templateUrl: "partials/phone-detail.list.html"
      controller: "PhoneDetailCtrl"
    )
]
```

Stejný případ jako u normálního angular-routery, ale tentokrát s ui-routerem. Z ukázky je vidět, že stavy v ui-routeru, jsou hierarchické a lépe se v nich orientuje.

6.3 Scope

Scope je základním stavebním kamenem AngularJS aplikací. Scope reprezentují model aplikace. Jsou také místem, kde dochází k vyhodnocení výrazů `{{}}`, definujeme zde bussines logic aplikace, metody našich controllerů a vlastnosti views. Scope svazuje controller a view, těsně před tím než se view zobrazí uživateli, je template *prolinkován* se scope a DOM nastaven tak, aby o všech změnách informoval AngularJS.[1]

Scope také tvoří hierarchické struktury, téměř každá direktiva vytváří vlastní scope.

6.4 Controller

Controller je funkcí, která přidává další funkcionalitu scope a je volán při jejím vytvoření.[1]

Ukázka jednoduchého controlleru

```
#VIEW:
<div ng-app="myApp">
  <div ng-controller="Controller" ng-init="init()">
    Jmeno: {{user}}
  </div>
</div>
#CONTROLLER:
app.controller "Controller", ["$scope", "api", ($scope, api) ->
  $scope.user = ""
  $scope.init =>
    $scope.user = api.getUser()
#výsledek
Jmeno: Franta
```

Jednoduché vypsaní jména uživatele, které je získáno z factory "api" funkcí "getUser()".

6.5 Directive

Directivy jsou místem, kde je v AngularJS zpracováván DOM. AngularJS je plný *directive*¹⁾, už pro vytvoření AngularJS aplikace je třeba využít *directive* *ng-app*.^[1]

Zpracování *directive* má 3 fáze, *compile*, *link* a *controller*, které jsou vykonány v tomto pořadí. *Directive* je možné *bindovat* na DOM element několika způsoby^[1]:

E Element `<mojeDirectiva></directiva>`

A Atribut `<div mojeDirectiva="" ></div>`

C Třída `<div class="mojeDirectiva" ></div>`

M Komentář `<!-- directive: mojeDirectiva exp -->`

Directiva může mít vlastní HTML template, vlastní *scope*, *controller*, můžeme určit prioritu jejího vykonání, apod. Ve všech ohledech je *directive* velmi dobře znovuvyužitelný komponent aplikace.^[1]

Ukázka *directive*:

```
app.directive "loading",->
  restrict: "E"
  replace:true
  scope:{}
  link: (scope) ->
    scope.$on "LOADING",->
      scope.loading=true
    scope.$on "LOADING_END",->
      scope.loading=false
  template:'<div class="loading" ng-if="loading"></div>'
VIEW:
<loading></loading>
```

Directive "loading" je možné "bindovat" na DOM element s názvem "loading". Pokud *directive* obdrží "\$broadcast" "LOADING", zobrazí se DOM element s načítáním a pokud obdrží "\$broadcast" "LOADING_END" skryje načítání.^[1]

6.6 Factory

Obecně lze říci, že *service* *provider*y, jakým je i *factory*, jsou k tomu, aby pro všechny *controller*y aplikace poskytovaly stejné metody a data po dobu existence aplikace.

Jedná se vždy o objekt typu **singleton**, který je instancován jednou za život aplikace

¹⁾seznam build-in *directive* pro AngularJS je k dispozici v dokumentaci: <https://docs.angularjs.org/api/ng/directive>

pomocí *Injector* service.[1]

Časté je použití factory jako helper pro komunikaci s backendem aplikace, localStorage, cookieStorage apod.

Ukázka jednoduché factory pro generování náhodných čísel:

```
app.factory "random", ->
  randomNumber:->
    Math.floor(Math.random())
  randomNumberBetween:(min,max)->
    Math.random() * (max - min) + min
  randomFromArray:(array)->
    array[Math.floor(Math.random() * array.length)]
```

Factory v příkladu je spíše takový "helper"²⁾. Má tři metody, které vrací buď náhodné číslo, náhodné číslo mezi dvěma hodnotami nebo náhodný prvek pole.

6.7 Filter

Filter jak už název napovídá je určen pro filtraci dat. Je možné ho použít formou service v controlleru nebo přímo ve view pomocí "roury" `{{ data | filter }}`. AngularJS má několik buildin filtrů jako je *date*, *lowercase*,[1] Je možné si nadefinovat také vlastní filtry. Následující filter dělá jednoduchou věc a to, že drží daný počet míst čísla tak, že před číslo doplňuje nuly:

```
app.filter "numberpad", ->
  (input, places) ->
    out = ""
    if places
      placesLength = parseInt(places, 10)
      inputLength = input.toString().length
      i = 0
      while i < (placesLength - inputLength)
        out = "0" + out
        i++
      out = out + input
    out
#použití:
<div>{{minutes | numberpad:2}}</div>
#výsledek
$scope.minutes = 1
<div>01</div>
```

6.8 Animace

Od verze 1.2 disponuje AngularJS přepracovanými animacemi v samostatném modulu *angular-animate*. Definovat animace je díky tomuto modulu velmi jednoduché, snadno pochopitelné a velmi příjemné. K animacím jsou používány běžné css animace, které jsou definovány pomocí css selectorů. [19]

²⁾helper - pomocná funkce, třída nebo knihovna

Většina AngularJS directive jako *ngRepeat*, *ngShow*, *ngHide*... má vlastní *hooks*, které jsou spouštěny v určitých fázích cyklu directive.[1]

Ukázka *ng-hide* animace:

```
#DOM
<div class="anim" ng-hide="checked" ng-click="checked=false">Hide</div>
#CSS
.anim.ng-hide-add, .anim.ng-hide-remove {
  transition:all linear 0.5s;
  display:block!important;
}
.anim.ng-hide-add.ng-hide-add-active,
.anim.ng-hide-remove {
  opacity:0;
}
.anim.ng-hide-add,
.anim.ng-hide-remove.ng-hide-remove-active {
  opacity:1;
}
```

Tato animace je určena pro directive "ngShow". Z kódu lze vyčíst, že při přidání a odebrání elementu, respektive při přidání třídy "ng-hide-add" a "ng-hide-remove", se provede lineární animace trvající 0.5s a po tuto dobu bude trvat přechod z průhlednosti 1 do průhlednosti 0. Tak může být docíleno zmizení jakéhokoliv DOM elementu.[1] Další informace, jak vytvářet vlastní animace, je možné získat v dokumentaci projektu: <https://docs.angularjs.org/guide/animations>

7 LESS



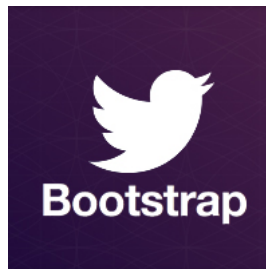
Obr. 8. Logo Less

LESS je pre-processor CSS stylů a jeho účelem je rozšířit funkcionalitu CSS jazyka. CSS je velmi jednoduché avšak nenabízí dynamičnost, která je dnes velmi potřebná a je často vyžadována projektem samotným. Právě k tomuto účelu se používá pre-processor. Velmi populární jsou zejména pre-processory *LESS*, *SASS* a *Stylus*. Ve své podstatě jsou si podobné a nabízejí i podobnou funkcionalitu. Všechny mají stejné vnořené selectory, svůj systém proměnných, cykly, mixin, funkce a mnoho dalších nástrojů usnadňujících práci především kodérům.[20]

Jednoduchá ukázka less:

```
#proměnná
@color: #f938ab;
#mixin
.box-shadow(@style, @c) when (iscolor(@c)) {
  box-shadow: @style @c;
}
#mixin
.box-shadow(@style, @alpha: 50%) when (isnumber(@alpha)) {
  .box-shadow(@style, rgba(0, 0, 0, @alpha));
}
.box {
  color: saturate(@color, 5%);
  #vnořený selector
  div { .box-shadow(0 0 5px, 30%) }
}
#výsledek
.box {
  color: #fe33ac;
}
.box div {
  box-shadow: 0 0 5px rgba(0, 0, 0, 0.3);
}
```

7.1 Twitter Bootstrap



Obr. 9. Logo Twitter Bootstrap

Twitter Bootstrap je jeden z prvních a určitě nejpoužívanější frontend framework¹⁾ současné doby.

Twitter bootstrap disponuje responzivním grid systémem, vlastní sadou CSS a JS komponent, modulární strukturou, standardizovaným API, výbornou dokumentací a mnoho dalšími funkcemi.[4][28]

Twitter Bootstrap podporuje od verze 3 CSS preprocesory, a je dostupný pro LESS i SASS. Čímž umožňuje v LESS stylech aplikace importovat části Bootstrapu takovýmto způsobem.

¹⁾**Frontend frameworky** v určitých segmentech zpřístupňují nástroje webového frontendu masám podobně, jako to udělalo jQuery s JavaScriptem[?, ?, bootstrap] vlastně spojují hodně vlastností do jedné, organizace kódu, CSS preprocesory, JS knihovny,... z toho všeho vytváří sadu CSS a JS knihoven, které mají standardizované API a zaručují funkčnost na podporovaných prohlížečích

```
@import "vendor/bootstrap/less/variables";
@import "variables";
@import "vendor/bootstrap/less/mixins";
@import "vendor/bootstrap/less/normalize";
@import "vendor/bootstrap/less/scaffolding";
@import "vendor/bootstrap/less/grid";
```

Poskytuje tak možnost v LESS stylech využívat Bootstrap mixiny, měnit proměnné a přímo tak ovlivňovat výslednou podobu Bootstrap knihovny[29].

Ukázka použití Twitter bootstrap v html kódu:[30]

```
<div class="row">
  <div class="col-xs-12 col-md-8">
    Levý sloupec 8/12 šířky řádku při rozlišení vyšším než 992px.
    Celý sloupec na zařízeních než 768px.
  </div>
  <div class="col-xs-6 col-md-4">
    Pravý sloupec 4/12 šířky řádku při rozlišení vyšším než 992px.
    Levá polovina řádku při rozlišení nižším než 768px.
  </div>
</div>
```

Popis dalších funkcí a vlastností Twitter Bootstrap je možné nalézt na oficiálním webu <http://getbootstrap.com/css/>.

7.2 LessHat



Obr. 10. Logo LessHat

LessHat je specializovaná knihovna javascriptových mixínů pro less, která velmi usnadňuje práci kodérům. Konkrétně obsahuje 87 mixínů a převezme tak starost o vendor prefixy a nekompatibilitu mezi prohlížeči za kodéra.[32][33]

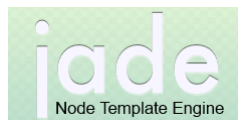
Příklad výstupu mixínu pro gradient:

```
//LESS
.lh-background-image(linear-gradient(
  to right, #6EDDFF 0%,#1AB0DD 100%
));
//CSS
background-image: url(data:image/svg+xml;base64,PD94...L3N2Zz4=);
background-image: -webkit-linear-gradient(
  left, #b0ecff 0%, #72d8f8 100%);
background-image: -webkit-gradient(
  linear, left top, right top, from(#b0ecff), to(#72d8f8));
background-image: linear-gradient(
  to right, #b0ecff 0%, #72d8f8 100%);
```

Less hat vytvořil nejen gradient s vendor prefixy, ale i svg gradient.

Když se k této knihovně přidá ještě sprite generátor, který všechny obrázky vkládá pomocí base64 kódování přímo do souboru, tak můžeme značně snížit počet requestů potřebných při načtení stránky.

8 JADE



Obr. 11. Logo Jade

Jade je velmi podobný CoffeeScriptu nebo YAMLu, jedná se o template engine velmi často používaný v NodeJS aplikacích. Zjednodušuje syntaxi HTML jazyka tím, že odstraňuje párovost HTML tagů a hierarchii elementů určuje indentace řádků.

Jednoduchá ukázka HTML dokumentu napsaného v JADE mluví za vše. HTML tagy jsou jasně patrné, syntaxe je ovšem odlišná. Kód je především jednodušší a HTML tagy připomínají spíše funkce.

```
#JADE
doctype html
html(lang="en")
  head
    title= pageTitle
    script(type='text/javascript').
      if (foo) {
        bar(1 + 5)
      }
  body
    h1 Jade - node template engine
    #container.col
      if youAreUsingJade
        p You are amazing
      else
        p Get on it!
    p.
      Jade is a terse and simple
      templating language with a
      strong focus on performance
      and powerful features.

#HTML
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Jade</title>
    <script type="text/javascript">
      if (foo) {
        bar(1 + 5)
      }
    </script>
  </head>
  <body>
    <h1>Jade - node template engine</h1>
    <div id="container" class="col">
      <p>You are amazing</p>
      <p>Jade is a terse and simple
        templating language with a
        strong focus on performance
        and powerful features.</p>
    </div>
  </body>
</html>
```

[21]

9 BOWER



Obr. 12. Logo Bower

Bower je velmi podobný NodeJS balíčkovacímu systému NPM. Bower, ale slouží k určení balíčkových závislostí pro různé javascriptové knihovny, jako je AngularJS, JQuery, apod. Usnadňuje především plnění závislostí na externích javascriptových knihovnách

a odlehčuje repozitářům, protože odpadá verzování knihovních závislostí a zůstávají pouze reference na ně, které stáhne bower při buildu aplikace.[22]

Bower disponuje stejně jako NPM interaktivním scriptem pro vytváření projektů a má podobnou comand-line syntaxi:

install nainstaluje balíčky v bower.json

install <balíček> nainstaluje balíček

update aktualizuje balíčky v bower.json

search vyhledávání balíčků

10 GULPJS



Obr. 13. Logo GulpJS

Gulp je task runner stejně jako známější Grunt, který je ve světě NodeJS aplikací zřejmě nejpoužívanějším task runnerem. Ve své podstatě jsou si oba programy velmi podobné, ale gulp je především rychlejší, což je při překladu složitějších aplikací velmi výtané. Gulp používá datové streamy a roury (pipes), které jsou stejným konceptem jako "|" (roura) v linuxovém shellu¹⁾. Gulp má také jednodušší konfiguraci a všechny tasky probíhají asynchronně, což přispívá na zvýšení rychlosti nejvíce.

Ukázka gulp tasku pro zpracování LESS. Pro tento úkol by byly třeba alespoň 3 Grunt tasky.

¹⁾Roura pracuje tak, že předává výstup jednoho procesu druhému přímo v paměti.

```
gulp.task('LESS_DEV', ["SPRITES_DEV"], function() {
  return gulp.src("./src/less/main.less")
    .pipe(debug({verbose: true}))
    .pipe(plumber())
    .pipe(less())
    .pipe(autoprefixer(confAutoPrefix))
    .pipe(gulp.dest("./build"))
    .pipe(livereload())
});
```

Postupně jsou provedeny tyto kroky: **src** - vybere soubory a vytvoří stream, **debug** - spustí ladění scriptu, **plumber**-zajistí, že při chybě nedojde k pádu gulpu, **less** - přeloží less zdrojové soubory, **autoprefixe** - vyřeší vendor prefixy prohlížečů na základě konfiguračního objektu, **dest** - uloží stream do souboru, **livereload** - obnoví livereload server.[21]

11 SYSTÉM PRO SPRÁVU VERZÍ GIT

Projekt git byl založen v roce 2005 Linusem Torvaldsem, původně pro vývoj jádra Linuxu, a jsou do něj vloženy zkušenosti, které Linus získal při vedení rozsáhlého projektu Linuxového jádra. [50]

Git je zcela zdarma, open source, **distribuovaný systém pro správu verzí** navržený rychle a účinně zvládnout vše od malých až po velmi rozsáhlé projekty. Každý klon Git je full-fledged (česky plně rozvinuté) úložiště s kompletní historií a veškerými schopnostmi sledování revizí a nezávislostí na přístupu k síti nebo centrálnímu serveru. Větvení (branching) a slučování (merging) jsou rychlé a snadné. [51] Velmi důležité je pak slovo distribuovaný, protože git neukládá každou změnu na centrální úložiště, ale každý klon repozitáře obsahuje i kompletní databázi změn projektu.

11.1 Základní příkazy a sociální web github

Ke správě jednoduchých a malých projektů jako je např. tato bakalářská práce stačí umět opravdu jenom pár příkazů. A díky webu github.com se stává správa malého projektu opravdu jednoduchou záležitostí.

11.1.1 Nastavení gitu

Základní nastavení gitu spočívá pouze v nastavení jména, příjmení a emailu. Pokud používáme git pouze lokálně a nenahráváme data někam na server, bude nám to stačit a můžeme již vytvořit repozitář.

Nastavení jména, příjmení a emailu. [52]

```
git config --global user.name "Jméno Příjmení"  
git config --global user.email "VášEmail@seznam.com"
```

Pokud nahráváme data na server je třeba autentizace. K účelu autentizace se často používá ssh klíč. Ten je možné vygenerovat následujícím způsobem [52]

```
ssh-keygen -t rsa -C "VášEmail@seznam.com"
```

pak je třeba zadat dvakrát heslo pro otevírání klíče a klíč je vygenerován.

Vygenerovaný klíč je k nalezení v domovském adresáři uživatele ve složce `.ssh`, která obsahuje soubory `id_rsa` a `id_rsa.pub`. `id_rsa` je váš privátní klíč a `id_rsa.pub` je klíč veřejný. Pro použití githubu je třeba veřejný klíč nahrát do nastavení na webu “*Account Settings*” > *Click “SSH Public Keys”* > *Click “Add another public key”*. Také je potřeba nahrát do nastavení gitu bezpečnostní token githubu. Bezpečnostní token lze nalézt na webu “*Account Settings*” > *Click “Account Admin.”* položku API token, která obsahuje jakýsi HASH, je třeba vložit do nastavení git na počítači. [52]

```
git config --global github.user uzivatelskeJmeno  
git config --global github.token 0123456789yourf0123456789token
```

11.1.2 Vytvoření repozitáře a první commit

Vytvoření repozitáře je velmi jednoduché a skládá se ze dvou kroků: Vytvoření složky a inicializace repozitáře.

```
mkdir projekt  
cd projekt  
git init
```

Změny v repozitáři jsou reprezentovány commity. Commit je vytvořen přidáním jednoho nebo více souborů a přidáním popisku. Jako první by měl projekt obsahovat soubor README.

```
touch README  
git add README  
git commit -m 'přidáno README'
```

Pokud chceme použít github, je třeba vytvořit repozitář na webu github. Po vyplnění názvu a popisu repozitáře, github vypíše kroky nutné k vytvoření repozitáře. Vypíše tedy, že je třeba vytvořit složku, inicializovat git, přidat první commit (nejčastěji soubor README), přidat adresu repozitáře na serveru github a odeslání změn v repozitáři. [52]

```
git remote add origin git@github.com:uzivatelskeJmeno/projekt.git  
git push origin master
```

11.1.3 Správa repozitáře

Každodenní správa repozitáře jednoduchého projektu je většinou o přidávání commitů a odesílání změn. Takže si většinou lze vystačit s příkazy *add*, *rm*, *diff*, *commit*, *push* a *pull*. Ale git obsahuje také spoustu jiných užitečných funkcí, které usnadní správu i malého projektu.

Commit není pouze přidávání souborů, ale i mazání a změny v souborech.

```
git rm soubor #smaže soubor
git add #přidá soubor, pokud soubor změníme je třeba ho také přidat
git diff #ukáže změny v repozitáři
git commit #vytvoří commit
git push #odešle změny
git pull #stáhne změny ze serveru
```

Jednou z funkcí je **tag**. Tag vytvoří značku za určeným commitem ke které se lze jednoduše vrátit. Je tedy užitečný pokud chce vytvořit např. release programu. Commity a jejich kódy si lze prohlédnout příkazem *git log*

```
git tag v1.0 <kód commitu>
```

K tagu se lze jednoduše vrátit příkazem *checkout*. (příkaz *checkout* také mění větev)

```
git checkout v1.0
```

Další funkcí, která je užitečná je *branch* (větve). Branch můžeme např. použít pokud se chystáme udělat změnu, o které nevíme jistě, zda bude fungovat. Často je branch také k oddělení prací uživatelů či částí projektu, na kterých se pracuje.

Branch lze vytvořit příkazem

```
git branch <název větve>
```

Repozitář je automaticky přepnut na vytvořenou větev a změny už se neprovádí v hlavní větvi *master*, ale ve větvi nové. Vytváření commitů a odesílání změn je poté normální. Nastane ale čas, kdy je třeba části větve nebo větve celou začlenit do větve hlavní. K tomuto je několik příkazů: první z nich je *rebase*, který stáhne změny z hlavní větve a aplikuje je před změny ve vytvořené větvi. Pokud v projektu nevznikají změny souběžně v hlavní větvi i větvi nové, není *rebase* třeba. Změny aplikujeme pomocí změny větve zpět na *master* příkazem *checkout* a odeslání změn z vytvořené větve pomocí *pull* (takto lze aplikovat změny mezi libovolnými větvemi).

```
git checkout -f master
git pull . <název větve>
```

Samozřejmě může nastat čas, kdy větev již není třeba a chceme ji spojit s větví hlavní nebo smazat.

```
git merge <název větve> #spojí aktuální větev se zadanou
git branch -D <název větve> #smaže větev
```

Pokud souběžně vznikají změny ve větvi hlavní i nové a chceme provést příkazy jako *rebase*, *pull*. <větev> nebo *merge* mohou nastat konflikty ve změnách, které je třeba vyřešit (git některé konflikty řeší sám, ale nedokáže všechno). V takovém případě přepne git aktuální větev do speciálního režimu a předá uživateli informace k vyřešení daných konfliktů. [53]

12 LATEX

L^AT_EX je typografickým systémem, který je určen k sazbě vědeckých a matematických dokumentů vysoké typografické kvality. Systém je rovněž vhodný pro tvorbu různých druhů dokumentů, od jednoduchých dopisu po složité knihy. Systém LaTeX je postaven na typografickém formátovacím programu T_EX Donalda E. Knutha.

Jedná se o balík maker, který umožňuje autorům sazbat [34] a tisknout jejich díla v nejvyšší možné typografické kvalitě, přičemž autor používá profesionály před definovaného vzhledu dokumentu. LaTeX byl původně napsán Leslie Lamportem a užívá TeXu jako sazečícího stroje.

X_{La}TeX je dalším rozšíření LaTeXu, které nativně používá Unicode znaky a podporuje moderní formáty fontů (OpenType a Apple Advanced Typography). XeLaTeX především řeší problémy spojené s kódováním českých znaků v PDF dokumentech, které jsou velmi nepříjemné.¹⁾

Pro publikaci předává autor nakladateli obvykle rukopis psaný na stroji. Typograf nakladatelství pak rozhodne o úpravě písemnosti (délka řádku, druh písma, odstupy před a za kapitolou atd.) a napíše sazeči k tomu nezbytné příkazy a sazeč podle těchto příkazů tiskovinu vysází. Analogicky přebírá roli typografa LaTeX a TeX přebírá úlohu sazeče. [5]

Každé LaTeX makro začíná obráceným lomítkem "\", následuje klíčové slovo a jako poslední libovolný počet parametrů ve složených závorkách "{ }" nebo hranatých závorkách.

```
\section{\upc{LaTeX}}
\LaTeX je typografickým systémem, ... možných
...které jsou časté s~\LaTeX em.\footnote{Tato diplomová ...}}
```

Ukázka LaTeXu se zkráceným zdrojem předchozího textu.

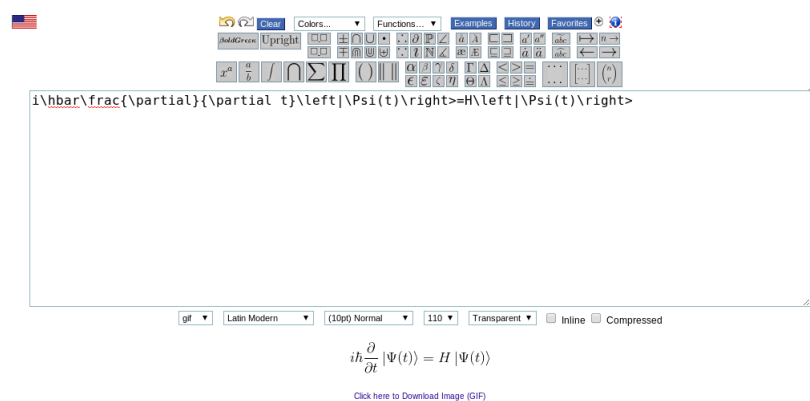
¹⁾Tato diplomová práce je také napsána v XeLaTeXu

12.1 Online LaTeX editory

Na internetu je dostupné značné množství pokročilých $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ editorů, které nabízí funkce na různé úrovni. Od jednoúčelových editorů na úpravu rovnic nebo vytváření tabulek až po komplexní systémy umožňující real-time editaci v týmu uživatelů, s velkým množstvím podpůrných funkcí, které dokáží usnadnit velké množství úkonů[39].

Z kategorie online editorů rovnic vyčnívá především editor **Codecogs**

www.codecogs.com/latex/eqneditor.php. Zvládá komplexní editace rovnic s real-time náhledem.



Obr. 14. Screenshot online editoru rovnic Codecogs

Další kategorií jsou komplexní online editory, ty dosahují velmi různé úrovně nabízených služeb.

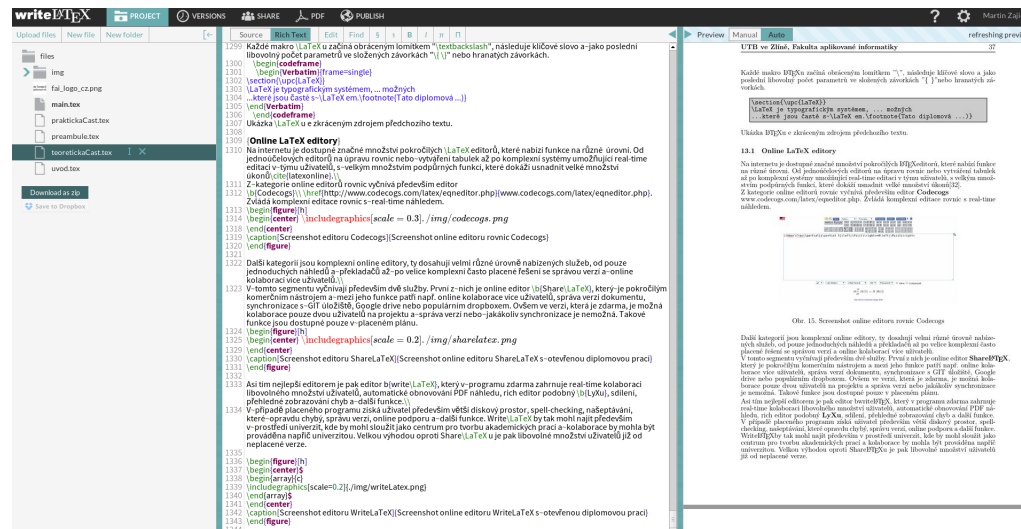
V tomto segmentu vyčnívají především dvě služby. První z nich je online editor **Share $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$** , který je pokročilým komerčním nástrojem a mezi jeho funkce patří např. online kolaborace více uživatelů, správa verzí dokumentu, synchronizace s GIT úložiště, Google drive nebo populárním dropboxem. Ovšem ve verzi, která je zdarma, je možná kolaborace pouze dvou uživatelů na projektu a správa verzí nebo jakákoliv synchronizace je nemožná. Takové funkce jsou dostupné pouze v placeném plánu.

Dalším editorem je pak **write $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$** , který v programu zdarma zahrnuje real-time kolaboraci libovolného množství uživatelů, automatické obnovování PDF náhledu, rich editor podobný **LyX**u, sdílení, přehledné zobrazování chyb a další funkce.

V případě placeného programu získá uživatel především větší diskový prostor, spell-checking, našeptávání, správu verzí, online podporu a další funkce. Write $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ by tak mohl najít uplatnění především v prostředí univerzit, kde by mohl sloužit jako centrum pro tvorbu akademických prací a kolaborace by mohla být prováděna napříč univerzitou. Velkou výhodou oproti Share $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ u je pak libovolné množství uživatelů již od neplacené verze.



Obr. 15. Screenshot online editoru ShareLaTeX s otevřenou diplomovou prací



Obr. 16. Screenshot online editoru WriteLaTeX s otevřenou diplomovou prací

WriteLaTeX navíc disponuje API, kterým je možné otevřít libovolný projekt v editoru přímo z cizího webu[40], což umožňuje snadnou integraci do stávajících služeb, bez potřeby programování složitého editoru a celého ekosystému kolem, což by stálo obrovské úsilí a také velké množství peněz.

12.2 ACE editor

Důležitým prvkem použitým v projektu je vlastní implementace jednoduchého LaTeX editoru. O vlastní funkci editoru se stará ACE editor, který je opensource a byl vytvořen pro potřeby, velmi kvalitní služby Cloud9 c9.io, která poskytuje pokročilý IDE pro tvorbu aplikací přímo ve webovém prohlížeči. Editor podporuje široké množství ja-

zyků, mezi nimiž je i LaTeX včetně podpory našeptávání a je možné přidat i kontrolu pravopisu.

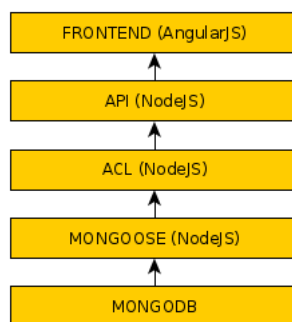
II. PRAKTICKÁ ČÁST

13 FUNKCIONALITA

Na funkcionalitu aplikace lze nahlížet ze dvou úhlů. Prvním je úhel z pohledu modularnosti aplikace a její škálovatelnost a druhým pohledem z hlediska uživatelského.

13.1 Modulární aplikační funkcionalita

V případě, že je řeč o škálovatelnosti aplikace lze dělit funkcionalitu na Databázi (viz. kapitola 15/s54), Backend (viz. kapitola 16/s69) a Frontend (viz. kapitola 17/s73). Každá z těchto komponent může běžet z vlastních alokovaných zdrojů na odlišných místech, navíc rozdělena na několik dalších částí.



Obr. 17. Diagram struktury aplikace

Databázovou funkcionalitu obstarává MongoDB server, který je sám o sobě škálovatelný a může běžet na několika serverech.

Backend zajišťuje NodeJS aplikace, kterou lze dělit na části dle funkcionalit, které většinou reprezentuje nějaké rozhraní (API) a mohou běžet na vlastních serverech. Např. překlad TeX dokumentů je uskutečněn na samostatném serveru, jenž poskytuje rozhraní pro kompilaci TeX dokumentů. Také zajišťuje integritu dat ukládaných do databáze a restrikce uživatelského přístupu.

Frontend je pak rozdělen do několika single-page aplikací, aby se klientovi, který přijde na e-shop nestahoval zbytečně také editor pro tvorbu knih, nebo části aplikace, které by neměly být přístupné. Frontend používá framework AngularJS a jeho kód je plně vykonáván na klientské stanici, tedy v prohlížeči. Data se stahují pomocí asynchronních javascriptových requestů.

13.2 Uživatelská funkcionalita

Uživatelská funkcionalita spočívá pouze na frontendu. Ten zajišťuje, že uživatel může pracovat se systémem. Prakticky je frontend rozdělen to těchto single-page aplikací, kde každá z nich poskytuje určitou funkcionalitu pro určité spektrum uživatelů.

Store Jedná se o jednoduchý uživatelsky přívětivý e-shop. (viz. kapitola 17.1/s73)

Create Rozhraní pro správu vlastních knih. (viz. kapitola 17.2/s73)

Edit Editor knih. (viz. kapitola 17.2/s73)

Admin Administrace. (viz. kapitola 17.4/s74)

Jednotlivé aplikace budou rozebrány ve svých kapitolách. Za zvážení pak stojí vytváření dalších rozhraní, jako je samostatný skladový systém, který pro jednodušší aplikaci postačí jako součást kolekce knih.

13.3 Proces schvalování knihy

V teoretické části byl popsán proces vydání knihy, který nelze nijak obejít a jeho kroky jsou podobné v každém nakladatelství. V aplikaci je možnost knihy vytvářet od prvního písmene a podle toho je upraven i proces jejího schválení. Hlavní myšlenka systému implementovaného v aplikaci spočívá v tom, že je na nakladateli, kdy je kniha hotová a schválí ji k publikaci.

Nakladatel přímo určuje, jak probíhá vývoj, může se vyjadřovat k aktuálnímu průběhu, každá kniha má svoji historii, kterou upravuje uživatel i nakladatel a vzájemně si tam mohou vyměňovat názory a myšlenky.

Jakmile se dostane nově vznikající kniha k určitému mylníku, může být ke zprávě, přidělen status a volitelná zpráva k statusu samotnému, seznam statusů je pak uložen v kolekci config.

Po schválení projektu může proběhnou akce vytvoření knihy. Kdy je z projektu vytvořena kniha a zní odkázáno zpět na projekt. I po publikaci je možné vytvořit opravy přímo pro již existující knihu. Nebo vydat další vydání ať už jako aktualizaci nebo novou knihu.

14 ADRESÁŘOVÁ STRUKTURA PROJEKTU

Statické soubory, do této složky probíhá build FrontEndu aplikace.

```
/build
```

Konfigurační soubory backendu. Nachází se zde 3 soubory pro jednotlivá prostředí.

```
/config
```

appDev - Development prostředí.

appTest - Testovací prostředí.

appProd - Produkční prostředí.

Soubory a funkce související s databází.

```
/database
```

Mongoose pluginy.

```
/database/plugins
```

Databázové schémata.

```
/database/schema
```

Soubory pro build systém.

```
/gulp
```

Jednotlivé tasky v samostatných souborech.

```
/gulp/tasks
```

Pomocné scripty pro build systém.

```
/gulp/util
```

Knihovny pro NodeJS.

```
/node_modules
```

Routery pro jednotlivá API.

```
/routes
```

Zdrojové kódy s frontendovou aplikací.

```
/src
```

Společné funkce.

```
/src/common
```

Překladačové soubory.

```
/src/i18n
```

Obrázky.

```
/src/images
```

LESS styly.

```
/src/less
```

Každá další složka v src pak obsahuje určitou funkcionalitu např. složka *Admin* obsahuje vše související s administrací, uspořádané do modulů. Hlavní modul je popsán souboru *index.coffee* nachází se zde také soubor se styly *index.less*, šablony **.tpl.html* a další CoffeeScript soubory s funkcionalitou. *Index.coffee* *linkuje* dohromady všechny své moduly, funkce, directive, controllery a další své podřazené části včetně společných (*common*).

```
/src/*
```

Dočasné soubory, především *templates.js* a *sprites.less*

```
/tmp
```

Bower componenty.

```
/vendor
```

Hlavní jade soubory, které jsou načítány klientem.

```
/views
```

Další soubory v root aplikace.

```

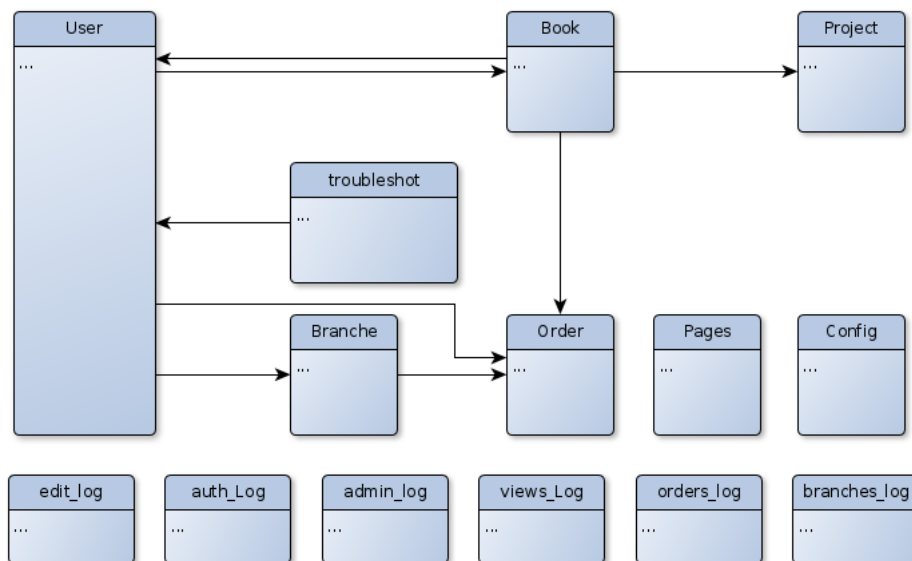
/app.js
#vstupní soubor nodejs aplikace, pouze otevírá index.coffee
/bower.json
#nastavení bower
/coffeelint.json
#nastavení coffeelint
/config.coffee
#konfigurační funkce aplikace
/gulpfile.js
#spouštěč build systému
/LICENCE
#licence aplikace
/package.json
#nastavení NPM
/README.md
#soubor s informacemi o aplikaci

```

Z výše uvedeného vyplývá, že projekt je fragmentován na co nejmenší komponenty tak, aby se v něm dalo snadno orientovat i za předpokladu, že jej programátor vidí poprvé.

15 DATABÁZE

15.1 Návrh/schéma



Obr. 18. Diagram kolekcí v databázi aplikace.

15.1.1 Users

Kolekce uživatelů je nedílnou součástí každého informačního systému. Slouží k přihlašování uživatelů, ke správě zaměstnanců i určování práv v aplikaci.

```
email:
  type: String
  required: true
  unique: true
  index: true
img:
  type: Schema.Types.Mixed
locale:
  type: String
name:
  type: String
  required: true
surname:
  type: String
  required: true
passwordSalt:
  type: String
  required: true
password:
  type: String
  required: true
address:
  [
    state: String
    city: String
    street: String
    postal: Number
    home_number: Number
    type: String
  ]
billing_address: [
  state: String
  city: String
  street: String
  postal: Number
  home_number: Number
  type: String
]
tel:
  type: String
```

První část dokumentu uživatele se zabývá klasickými uživatelskými informacemi.

```

orders:
  [
    type: Schema.Types.ObjectId
    ref: "Order"
  ]
projects:
  [
    type: Schema.Types.ObjectId
    ref: "Project"
  ]
whished_books:
  [
    type: Schema.Types.ObjectId
    ref: "Book"
  ]
payed_e_books:
  [
    type: Schema.Types.ObjectId
    ref: "Book"
  ]
payed_books:
  [
    type: Schema.Types.ObjectId
    ref: "Book"
  ]
favourite_books:
  [
    type: Schema.Types.ObjectId
    ref: "Book"
  ]

```

Další část dokumentu se zabývá především referencemi na knihy. Uživatel může mít velké množství objednávek, které samostatně obsahují všechny uživatelské informace. Každý uživatel může také tvořit knihy tudíž mu náleží kolekce projektů. Může si přát knihy, které mu mohou kupovat známí a může mít označeny své oblíbené knihy. Velmi důležitou položkou jsou pak zaplacené knihy, podle tohoto pole systém pozná, které knihy může uživatel stáhnout ve formátu PDF nebo jestli si je koupil v knižním vydání.

```

roles:
  type: [String]
  default: ['anonymous']
special:
  allow: [String]
  deny: [String]

```

Následuje část zabývající se právy uživatele, uživatel může mít definovány role a zvláštní práva pro přístup k funkcionalitám.

```

User-Agent:String
remoteAddress:String
X-Forwarded-For:String
last_login:
  type:Date
  default: Date.now

```

Důležité je také uchovávat informace o posledním přihlášení uživatele a místě, kde přihlášení proběhlo. Tato skutečnost je důležitá zejména kvůli bezpečnosti uživatele. Uži-

vatel nebo i systém může automaticky některé přihlášení vyhodnotit jako falešné, především na základě geolokačních dat IP adresy. Více údajů o předchozích přihlášeních, lze pak vyčíst z Auth logu (viz. kapitola 15.1.9/s66).

```
meta: [  
  key: String  
  value: String  
]  
updated_at:  
  type: Date  
  default: Date.now  
created_at:  
  type: Date  
  default: Date.now
```

Poslední část, zabývající se především dalšími popisy dokumentu, časy vytvoření a posledními úpravami, se bude opakovat ve všech kolekcích, proto již nebude dále uváděna.

15.1.2 Books

Tou nejdůležitější částí databáze knihkupectví jsou knihy. Pro ty je určena kolekce *book*, která zajišťuje kompletní informace o knize - od jejího popisu až po informace o historii ceny, prodeji a autorovi.

```
name:  
  type: String  
  required: true  
  index: true  
author:  
  name:  
    type: String  
    required: true  
  description: String  
  img:  
    data: Buffer  
    contentType: String  
isbn: String  
url:  
  type: String  
owner:  
  type: Schema.Types.ObjectId  
  ref: "User"  
img:  
  small:  
    data: Buffer  
    contentType: String  
  big:  
    data: Buffer  
    contentType: String  
genres: [  
  name:  
    type: String  
    index: true  
]  
keywords:  
  type: String  
  required: false  
pages:  
  type: Number  
  required: true
```

První část dokumentu se zabývá popisem knihy a jejím autorem. Přesto, že by se po zvyklostech z SQL databází mohlo zdát výhodnější, mít autory v samostatné kolekci není tomu tak. V tomto případě se stáhnou informace o autorovi společně s knihou a informace o autorovi se mohou u každé knihy měnit, např. podle období kdy knihu napsal.

Dotaz na vyhledání autora v kolekci podle jména není nijak složitý. Následující vrátí všechny knihy, které mají ve jméně autora řetězec *George*.

```
name = 'George'
book.find(
  author.name: new RegExp('^'+name+'$', "i"), (err, doc)->
    console.log doc
)
```

U náročnějšího systému by pak bylo vhodné vytvořit pro autory ještě samostatnou kolekci. Údaje o autorovi by, ale i tak měly být zanechány také v kolekci knih, a kolekci autorů používat pouze pro vyhledání knih, které napsal konkrétní autor.

```
commets:[
  {
    author:
      type: String
      require: true
    coment:
      type: String
      require: true
    user:
      type: Schema.Types.ObjectId
      ref: "User"
  }
]
rating:[
  {
    value:
      type: Number
      require: true
    user:
      type: Schema.Types.ObjectId
      ref: "User"
  }
]
review:[
  {
    title:
      type: String
      require: true
    user:
      type: Schema.Types.ObjectId
      ref: "User"
    content: String
  }
]
```

Další v kolekci jsou pak data, kterými přispívají uživatelé. Je možné přidávat komentáře, hodnotit knihy a psát recenze.

```
buyers:[
  type: Schema.Types.ObjectId
  ref: "User"
]
number_of_sales:
  type:Number
  required:false
available:Boolean
delivery_msg:String
price:
  type:Number
released:Date
public_from:Date
buy_allowed_from:Date
description:String
publisher:String
price_hystory:[
  {
    date:
      type:Date
      default: Date.now
    value:
      type:Number
  }
]
```

Poslední část dokumentu je pouze statistická, je zde výčet uživatelů, kteří si knihu koupily, historie ceny, dostupnost knihy a informace o vydání.

15.1.3 Orders

Objednávka je v SQL databázích většinou předmětem velkého množství relací na uživatele adresy, schvalování, dopravu apod. Všechny tyto aspekty NoSQL databáze řeší velmi elegantně a to v jediném dokumentu, bez relací které by vyžadovaly další odkazování. Navíc pokud je objednávka již jednou vytvořena, neměl by její obsah být pozměněn referencemi. Samozřejmě kolekce také obsahuje, reference na zdrojové objekty, ale protože jsou informace již obsaženy přímo v kolekci, není třeba vyhodnocovat tyto reference pomocí *populate*.

```
user:
  name:
    type:String
    required: true
  surname:
    type:String
    required: true
  tel:
    type:String
    required: true
  email:
    type:String
    required: true
  user:
    type: Schema.Types.ObjectId
    ref: "User"
price:
  type:Number
  required: true
```

První část obsahuje informace o kupujícím včetně reference na něj. V případě, že v budoucnu dojde ke smazání kupujícího, data o obchodu s ním jsou stále zaznamenána a konzistentní.

```
items:[
  name:
    type: String
    required: true
    index: true
  author:
    type: String
    required: true
  price:
    type: Number
    required: true
  released: Date
  description: String
  pages:
    type: Number
  book:
    type: Schema.Types.ObjectId
    ref: "User"
]
```

Velmi důležitý je seznam zakoupených položek. Opět je část základních dat zkopírována do dokumentu, aby nedošlo ke změnám v objednávkách následkem změny reference.

```
confirmed: Boolean
states:[
  title: String
  description: String
  date: Date
  default: Date.now
  user:
    type: Schema.Types.ObjectId
    ref: "User"
]
```

Objednávka má samozřejmě svou historii - od chvíle jejího vytvoření až do chvíle jejího vyřízení nebo zániku. Proto jsou v této části definovány změny v objednávce. Pokaždé když objednávka postoupí schvalovacím procesem, nebo změnou zapíše se zpráva co se stalo a kdo akci vyvolal.

```
custom_id:String
custom_description:String
payed:Boolean
billing_address:
  state: String
  city: String
  street: String
  postal: Number
  home_number: Number
  type: String
destination_address:
  state: String
  city: String
  street: String
  postal: Number
  home_number: Number
  type: String
pay_method:
  name:
    type:String
    required: true
  card_number:
    type:Number
delivery_method:
  name:
    type:String
    required: true
price:
  type:Number
  required: true
branche:
  name: String
  city: String
  street: String
  postal: Number
  home_number: Number
  tel: Number
```

Poslední část se zabývá obchodem samotným. Uživatel má možnost zvolit vlastní identifikátor objednávky, metodu platby, místo zaslání objednávky, místo fakturace a zda byla objednávka z e-shopu nebo z kamenné pobočky.

15.1.4 Project

Pro potřebu vytváření knih je zde kolekce Project, která zajišťuje vlastnosti potřebné pro vytvoření publikace. Kromě informací o uživateli obsahuje, konverzaci ohledně projektu, seznam souborů, seznam, tex souborů a doprovodné nastavení překladače.

```
name:
  type: String
  required: true
  index: true
owner:
  type: Schema.Types.ObjectId
  ref: "User"
publicated: Boolean
genres: [
  {
    type: String
    index: true
  }
]
progress: [
  {
    author:
      type: String
      require: true
    comment:
      type: String
      require: true
    user:
      type: Schema.Types.ObjectId
      ref: "User"
    status: String
  }
]
compiler:
  type: String
  default: "xelatex"
rootResourcePath:
  type: String
  default: "main.tex"
renderPath: String
files: [
  {
    name: String
    path: String
  }
]
tex_files: [
  {
    path: String
    content: String
  }
]
```

15.1.5 Branches

Knihkupectví většinou nemá pouze internetový obchod, ale také kamenné pobočky. Pro držení informací o pobočkách slouží kolekce Branches.

```
name:
  type: String
  required: true
address:
  state: String
  city: String
  street: String
  postal: Number
  home_number: Number
  type: String
contacts: [
  {
    name: String
    description: String
    tel: String
    fax: String
    email: String
  }
]
employes: [
  {
    name: String
    position: String
    paychecks: [
      priod_from: Date
      priod_to: Date
      sended: Date
      sum: Number
      info: [
        title: String
        score: string
      ]
    ]
    user:
      type: Schema.Types.ObjectId
      ref: "User"
  }
]
```

Kolekce poboček musí samozřejmě obsahovat informace o pobočce samotné, jejích zaměstnancích a platech zaměstnanců.

```
books_on_sale: [
  {
    type: Schema.Types.ObjectId
    ref: "Book"
  }
]
books_to_arrive: [
  {
    order_date: Date
    arrival_date: Date
    state: String
    book:
      type: Schema.Types.ObjectId
      ref: "Book"
  }
]
orders: [
  {
    type: Schema.Types.ObjectId
    ref: "Order"
  }
]
```

Každá pobočka si navíc musí vést informace o knihách dostupných na pobočce, knihách, které má objednány a uskutečněných prodejkách.

15.1.6 Config

Config je pouze jednoduchá key:value databáze určená ke změnám v prostředí aplikace, tyto změny pak nepotřebují restart nodejs serveru. Vhodným řešením takovéto kolekce, je pak cachování do jiné NoSQL databáze jako je Redis či Memcache.

```
key:String  
value:String
```

```
config.groups: [  
  name:String  
  allowed:[String]  
]
```

Klíč *groups* zajišťuje nastavení skupin a přístup k jednotlivým akcím.

```
config.actions:[String]  
config.actionsuserDefined:[String]
```

Seznam akcí backendu, tento klíč slouží pouze seznam funkcí, ze kterých je možné volit při nastavení práv uživatelů a skupin. Jedna z možností implementace takového seznamu je, že by každý plugin exportoval seznam svých funkcí, které nabízejí omezení práv uživatele a seznamy jednotlivých pluginů by se sjednocovaly v některé z databází běžících v paměti.

```
config.publicSteps: [  
  name:String  
]
```

Klíč *publicSteps* obsahuje seznam kroků, které je možné volit při komentování tvorby knihy.

```
config.genres:[String]
```

Seznam žánrů ze, kterých jde volit při vytváření knihy.

15.1.7 Troubleshoot

V případě, že má uživatel portálu dotaz, je k dispozici troubleshoot kolekce, která zajišťuje komunikaci uživatelů s provozovatelem služby.

```
title:
  type: String
  required: true
msg:
  type: String
user_started:
  name:String
  user:
    type: Schema.Types.ObjectId
    ref: "User"
user_involved:[
  name:String
  user:
    type: Schema.Types.ObjectId
    ref: "User"
]
alert_users:[
  type: Schema.Types.ObjectId
  ref: "User"
]
stage:
  type: String
priority:
  type: String
conversation:[
  msg:
    type: String
  stage_change:Boolean
  priority_change:Boolean
  user:
    name:String
    user:
      type: Schema.Types.ObjectId
      ref: "User"
]
```

Každé vlákno problémů má svůj titulek, úvodní zprávu, jméno uživatele, který ho vytvořil, seznam uživatelů zapojených do řešení a seznam uživatelů, kteří jsou informováni o změnách ve vláknech. Každé vlákno má navíc prioritu a fázi, ve které se nachází. Samozřejmě je možné přidávat zprávy, které mohou být navíc označeny jako změna priority nebo změna stavu.

15.1.8 Pages

Kolekce statických stránek. Jedná se o kolekci editovatelných webových stránek. Např. informace o obchodu, kontakty a další.

```
title:
  type: String
  required: true
url:
  type: String
  index: true
content:String
files:[String]
meta: [
  {
    key: String
    value: String
  }
]
```

Kolekce odpovídá struktuře běžné webové stránky. Obsahuje odkaz na stránku titulek, obsah, obrázky a meta tagy velmi důležité především pro SEO optimalizace.

15.1.9 Logs

Důležitým aspektem u každého webového systému jsou logy a logování co největšího množství událostí. Nejenom pro statistické údaje, ale i pro případ stížností ze strany klienta, ať už kvůli ztrátě dat nebo nefunkčnosti webu.

Pro logování má informační systém několik logů, každý určený pro jiný typ události.

Edit log slouží pro zaznamenávání událostí spojených s úpravou a vytvářením knih.

Auth log log pro zaznamenávání událostí souvisejících s přihlašováním. Např. uživatel se přihlásil, odhlásil, registroval, zadal špatné heslo, přišel do aplikace apod.

Admin log v tomto logu se zaznamenávají události spojené s administrací. Pokud uživatel s vyšším oprávněním něco změní, zobrazí se tato událost zde.

Views log tento log slouží především pro statistiky. Zaznamenávají se zde informace o tom, který uživatel kdy navštívil kterou stránku.

Orders log v tomto logu se zaznamenávají události související s objednáváním knih jako, uživatel vytvořil objednávku, přidal položku, zaplatil objednávku, zrušil objednávku atd.

Branches log záznam událostí spojených s pobočkami.

Logovací kolekce vypadají velmi podobně, proto bude popsána obecná struktura záznamu.

```
_user_id
```

Reference na uživatele, který vyvolal akci, pokud není uvedena je uživatel anonymní.

```
_page_id
```

Reference na stránku (používá se u view logu aby bylo možné získat statistiku navštívené stránky a auth logu aby bylo možné zjistit na, kterou stránku se uživatel pokusil přihlásit).

```
url
```

URL adresa kde byla událost vyvolána.

```
message
```

Popis události.

```
event
```

Název události.

```
type
```

Typ události (ERROR, INFO, DEBUG, WARN).

```
User-Agent
```

Pro lepší vyhodnocení události a případný troubleshoot jsou zaznamenány informace o prohlížeči.

```
remoteAddress
```

Záznam IP adresy, ze které záznam vyvolala.

```
X-Forwarded-For
```

Stejný záznam jako předchozí sloužící pro získání IP adresy v případě, že se aplikace nachází za proxy serverem.

15.2 Budoucí vývoj

MongoDB a jeho absence schémat zajišťuje snadnější rozšiřitelnost aplikace, není třeba nijak zvláště řešit relace mezi kolekcemi a už vůbec ne schéma. A to z důvodu, že MongoDB žádné schéma nemá. Prakticky stačí změnit schéma v mongoose, které je pouze aplikační a MongoDB o něm neví, a restartovat aplikace (nebo provést deploy).

Z funkcionálního hlediska pak budoucí vývoj záleží na požadavcích zákazníka, prakticky by pak další vývoj pravděpodobně zahrnoval rozšíření dat, které jsou zaznamenávány o uživateli tak, aby mu pak aplikace sama nabízela knihy a informovala o novinkách, o které by mohl mít zájem, na základě procházených, koupených knih nebo i informací o procházení jiných webových stránek.

Dalším bodem rozšíření je reklama, jak už reklama na položky portálu samotného tak i placená inzerce dalších portálů a služeb.

15.3 Zabezpečení

MongoDB má vlastní systém zabezpečení, který spočívá v několika systémových rolích vlastním ACL a omezení přístupu uživatelů k databázím a kolekcím. Aplikace reálně potřebuje přístup ke čtení i k zápisu do své databáze, není tedy na místě řešit jakákoliv složitější práva, která by zahrnovala oddělení uživatele pro čtení a zápis, apod. Ta by mohla přijít vhod v případě rozdělení aplikace na několik samostatných rozhraní, které by nepotřebovali např. přístup do některých kolekcí nebo zápis do nich.

Dalším faktorem zabezpečení je pak zabezpečení serverové. V případě zabezpečení databází obecně, je na snadě zamezit maximálně přímé interakci uživatele a databáze. Pokud není databáze sdílená a nepotřebují přístup další externí uživatelé, což je případ spíše multihostingů, kde je většinou otevřený port přímo do databáze či je nabízeno rozhraní pro databázi (PHPMyAdmin pro MySQL, pro MongoDB např. RockMongo). Proto je většinou přístupový bod do databáze možný pouze z místní sítě aplikace tj. z jejího aplikačního clusteru. V tomto velmi vhodném případě je pak přístup do databáze buď přímý přes databázový port nebo přes nějaké rozhraní pro databázi (RockMongo). K tomu je třeba vytvořit v místní síti aplikace server pro vzdálený přístup, jako je OpenVPN nebo jiný SW pro vzdálený přístup nebo připojení počítače do vzdálené místní sítě v Internetu.

15.3.1 Nastavení zabezpečení MongoDB

Zabezpečení MongoDB databáze je velmi jednoduché a lze jej zprovoznit pomocí MongoDB cli. Nejprve je třeba spustit program *"mongo"*, který je standardní součástí balíčku mongoDB pro většinu distribucí. Po spuštění se zobrazí interaktivní *"mongo shell"*ve, kterém je třeba nastavit administrátorské účty v kolekci admin (bez nastavení účtů pro administrátory by se nedalo po zapnutí autentizace do databáze dostat).[41]

Vytvoření uživatele admin s právy upravovat jakoukoliv databázi.

```
use admin
db.createUser(
  {
    user: "admin",
    pwd: "password",
    roles:
      [
        {
          role: "userAdminAnyDatabase",
          db: "admin"
        }
      ]
  }
)
```

Uživatelů je možné přidávat různá oprávnění např. readAnyDatabase, readWriteAny-

Database, dbOwner, userAdmin, userAdminAnyDatabase atd.

Jakmile je administrátor vytvořen je možné zapnout autentizaci. Zapnutí lze provést v souboru `/etc/mongo.conf` a to přepsáním property `"auth"` na `"true"`. Pak je třeba restartovat mongoDB server a poté je možné se přihlásit např. následujícím příkazem:

```
mongo --port 27017 -u admin -p pass --authenticationDatabase admin
```

Nyní není možné se na MongoDB server dostat bez přihlášení. K vytvoření databáze a slouží klíčové slovo `USE`, to říká kterou databázi budeme používat a v případě, že databáze neexistuje ji vytvoří.

```
use mojeDatabaze
```

Po založení databáze je již nutné pouze vytvořit uživatele s přístupem k ní, což může být provedeno stejně jako při vytvoření systémového administrátora. Pokud je vyžadováno zabezpečit aplikaci ještě více, je možné využívat dvou spojení do databáze, jednoho pro čtení a jednoho pro zápis. Taková aplikace může mít např. i dvě api. Jedno právě pro čtení a druhé pro zápis.

16 BACKEND

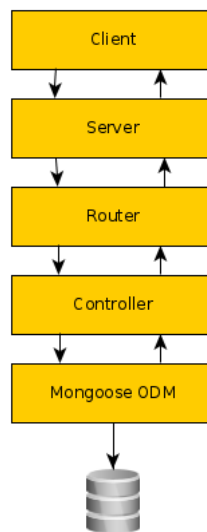
O část aplikace na straně serveru se stará NodeJS server postavený frameworku HapiJS spolu ODM mongoose pro práci s databází.

16.1 Struktura

Struktura backendu odpovídá struktuře celého projektu a je tvořena jednotlivými částmi popsány v následujícím diagramu. Na hoře je klient, což je angularJS frontend aplikace. Request od klienta putuje na server, kde jej zpracovává router frameworku HapiJS, který se stará o validaci requestů (požadavku), autentizaci, přidělení funkcionality, validaci response (odpovědi) a odeslání odpovědi zpět klientovi. Každý request tedy projde routerem, kde je mu přidělena funkce controlleru, provedena potřebná akce např. získání záznamů z databáze pomocí Mongoose ODM apod., odpověď je pak controllerem vrácena zpět klientovi.

16.2 API

API aplikace je velmi jednoduché, další funkcionality může být snadno přidána vytvořením modulu aplikace, který může mít vlastní router, view atd. API je navíc od počátku verzované, což umožňuje snazší orientaci napříč funkcionalitami v pozdějším vývoji aplikace. Aktuální verze routeru má na počátku cesty vždy prefix `"/api/v1"`, který nebude v názvu rout uváděn.



Obr. 19. Diagram prostupu struktury backendu aplikace.

16.2.1 /me

Základní sada API volání orientovaná na přihlášeného uživatele. Pro snadnější oddělení uživatele jako samostatné části aplikace slouží routa ”/me”, která má ve verzi API v1 následující routy:

GET /me vrátí informace o přihlášeném uživateli.

GET /me/all vrátí kompletního uživatele včetně referencí.

PUT /me update uživatele.

GET /me/book vrátí knihy, které uživatel vlastní (jedná se o knihy vytvořené uživatelem).

PUT /me/book/:id update knihy, jíž je uživatel vlastníkem a která má konkrétní ID.

GET /me/project vrátí projekty, jehož je uživatel vlastníkem.

PUT /me/project/:id update projektu, jehož je uživatel vlastníkem a který má konkrétní ID.

GET /me/project/:id/render vrátí render konkrétního projektu.

POST /me/project vytvoří nový projekt.

PUT /me/project aktualizuje projekt.

GET /me/order/:id vrátí objednávku.

POST /me/order vytvoří novou objednávku.

PUT /me/order update objednávky.

16.2.2 /book

API pro knihy slouží pro práci s knihami, z uživatelského hlediska se jedná především o hledání a výpisy.

GET /book vrátí pole knih na základě query stringu.

GET /book/:id vrátí kompletní informace o knize.

GET /book/:id/all vrátí kompletní informace o knize včetně populace.

PUT /book/:id update knihy.

POST /book vytvoření knihy.

DELETE /book/:id smaže knihu.

16.2.3 /user

Tato sada rout slouží pro práci s uživateli a pro přístup je potřeba být přihlášen jako administrátor.

GET /user vrátí pole uživatelů na základě query stringu.

GET /user/:id vrátí kompletní informace o uživateli včetně referencí.

PUT /user/:id update uživatele

POST /user vytvoření uživatele (nejedná se o registraci).

DELETE /user/:id smaže uživatele.

16.2.4 /branch

Routy slouží pouze pro práci s pobočkami, uživatelé nemají k těmto informacím přístup a je možné pouze vypsát pobočky.

GET /branch vrátí pole poboček na základě query stringu.

GET /branch/:id vrátí kompletní informace o pobočce.

PUT /branch/:id update pobočky.

POST /branch vytvoření pobočky.

PUT /branch vytvoření pobočky.

POST /branch/employe vytvoření zaměstnance.

PUT /branch/employe update zaměstnance.

POST /branch/order vytvoření objednávky.

PUT /branch/order update objednávky.

DELETE /branch/:id smaže branch.

16.2.5 /order

Routy slouží pouze pro práci s objednávkami a jsou přístupné pouze oprávněným uživatelům.

GET /order vrátí pole objednávek na základě query stringu

GET /order/:id vrátí kompletní informace o objednávce včetně referencí

PUT /order/:id update objednávky

POST /order vytvoření objednávky

DELETE /order/:id smaže objednávku

16.2.6 /project

Routy slouží pro práci s projekty, renderování správa apod.

GET /project vrátí pole projektů na základě query stringu.

GET /project/:id vrátí kompletní informace o projektu.

GET /project/:id/render vrátí render projektu.

DELETE /project/:id smaže projekt.

PUT /project/:id update projektu.

POST /project vytvoření projektu.

POST /project/:id/file vložení souboru.

DELETE /project/:id/file/:fileId smazání souboru.

16.2.7 /page

Routy slouží pro práci se statickými stránkami.

GET /page vrátí pole stránek na základě query stringu.

GET /page/:id vrátí kompletní informace o projektu.

DELETE /page/:id smaže stránku.

PUT /page/:id update stránky.

POST /page/:id/file nahraní souboru.

DELETE /page/:id/file/:fileId smazání souboru.

17 FRONTEND

Frontend pohání mnohokrát zmiňovaný AngularJS spolu s Browserify. Pro jednoduchost, oddělení funkcionalit a zrychlení načítání je aplikace rozdělena do několika samostatných SPA¹⁾ aplikací. Kde každá má svůj význam a určení.

17.1 Store (e-shop)

Store je jednoduchý obchod zaměřený na prodej knih orientovaný především na tablety. Nabízí funkci košíku internacionalizaci a nastavení uživatele. Velmi zajímavou funkcí je asynchronní carousel knih, který je možné velmi snadno modifikovat. Jedná se vlastně o stream, který slouží pro vyhledávání, filtraci knih reklamu apod. Uživatel si může v každém carouselu nastavit filtry knih.

Výhoda tohoto carouselu je v tom, že je jej možné do budoucna rozšířit, aby uživatel mohl zadávat složité filtry, vyhledávat aj. Carousel by mohl také nést reklamní informace tvářící se jako běžné knihy, záleželo by už jen na provozovateli jak by s takovými možnostmi naložil.

Důraz je kladen především na jednoduchost, tu je možné si dovolit především pro jednostrannost aplikace. V případě, že by bylo nabízeno více produktů, mohl by se obsah carouselů měnit podle druhu zboží atd.

17.2 Create (správa vlastních knih)

Velmi důležitou částí aplikace je část nazvaná "Create", která slouží pro správu vlastních knih. Tato aplikace nabízí vytváření projektů, správu a schvalování knihy. Při

¹⁾SPA - Single Page Application

budoucím vývoji by měla umět například počítat náklady a zisky projektu, zobrazovat informace o koupi, vyhodnocovat zájem uživatelů o knihu nebo by se zde mohli také spravovat kampaně a obchodní strategie prodeje knihy.

17.3 Edit (editor knih)

Část edit slouží pouze jako webový editor projektu, je možné zde editovat LaTeX soubory a zobrazovat stahovat knihy. Pro překlad zdrojových souborů je využívám SW CLSI²⁾, který podporuje různé překladače souborů a vrací formátovaný PDF výstup.

17.4 Admin (administrace)

Administrační část by měla sloužit pro kompletní správu celého systému, nejen pro tvorbu obsahu, ale měla by být také částí, která spravuje pobočky. Měla by zobrazovat statistiky. A v neposlední řadě by měla sloužit pro komunikaci se zákazníkem. Aby byl systém kompletně editovatelný musela by být veškerá menu a další prvky uložena do databáze a cachována v paměti pro rychlý přístup nebo vkládána do jade šablon a cachována i s šablonou.

Součástí implementace jsou i real-time statistiky implementované pomocí HTTP 1.1 websocketů.

17.5 Další části

Další části aplikace jsou už na požadavcích jednotlivých klientů aplikace. Částí, jež každý podnik potřebuje je správa financí, faktur platů atd. Takový systém je sám o sobě tak komplexní a je takového rozsahu, že zahrnout jej by vydalo na samostatnou práci. V každém případě je možné systém napojit na některý ze stávajících účetních systémů, jako je např. Pohoda, jež je v České republice velmi známá.

Další možností je pak napojení na pokladny přímo na pobočce, takové řešení je také velmi komplexní záležitostí, protože nelze jen nainstalovat SW na pokladní PC přidat čtečku a přes internet vyřizovat komunikaci. Pobočka musí se servery komunikovat bezpečně a musí být schopná fungovat i při výpadku připojení. Proto by každá pobočka musela mít vlastní server, který by dělal prostředníka mezi pobočkou a centrálními servery. Takový server by při běžném provozu pouze přeposílal data z pokladen na server, pomocí nějakého zabezpečeného spojení (VPN, SSH aj.) a v případě výpadku by tyto data shromažďoval a odeslal až při obnovení spojení. Zároveň by musel pravidelně aktualizovat databázi knih a další informace.

²⁾CLSI - Common LaTeX Service Interface

18 ZABEZPEČENÍ

Mezi api a frontendem dochází http komunikaci, která je sama o sobě náchylná na řadu útoků, jenž je třeba eliminovat již při návrhu aplikace. Klasické webové aplikace používají k autentizaci uživatele cookies, což není zcela bezpečné. Je nutné je zabezpečit proti útoků jako je CSRF, XSS a další. Návrh této aplikace obsahuje autentizaci pomocí JSON tokenu zvanou JWT¹⁾[47]

18.1 JWT (JSON Web Token)

Výhody:

- + JWT umožňuje cross domain API volání, nenastane tedy běžný cross domain problém, který nastává při cookies, jelikož cookies je vázaná k doméně.
- + JWT obsahuje všechny potřebné údaje pro identifikaci uživatele i mezi různými servery.
- + Je jednoduché realizovat klientské aplikace, jako nativní mobilní aplikace atd.[48]
- + JWT také řeší útoky typu CSRF, které hrozí při cookies autentizaci

Jak to funguje?

JWT token jsou tři base64 textové řetězce za sebou oddělené tečkami.

Ukázka JWT:

```
<header>.<payload>.<hash>  
eyJ0eX...JIUzI1NiJ9.eyJ1c...DU2Nzg5fQ.KG...1yB2xAwc_E
```

Header a payload jsou pouze jednoduché JSON objekty, jenž vypadají jako v následující ukázce:[48] **header:**

```
{ "typ": "JWT", "alg": "HMACSHA256" }
```

Typ je vždy JWT a alg je hash algoritmus použitý pro podepsání tokenu.[48]

payload: může být jakýkoliv validní JSON objekt, ale vyžaduje některé standardizované klíče.

```
{ "user": "Franta", "exp": 123456789}
```

User je definovaný klíč a exp je standardní UNIX time s časem, do kterého token platí. To, že payload může být libovolný je skvělá věc, protože je možné přenášet jakékoliv informace o uživateli. Důležité je vědět, že payload není šifrovaný, jedná se pouze

¹⁾JWT-JSON web token

o base64 zakódování, a není tedy určen k přenosu citlivých informací.

Celý token je pak sestaven tak, že se tečkou spojí base64 header a payload, a přidá hash vygenerovaný konkrétním algoritmem.

18.1.1 Implementace backend (NODEJS)

Implementace na straně serveru řeší doplněk *hapi-auth-jwt*, kterému stačí zadat master klíč aplikace a validační funkci tokenu, jež dostává dekodovaný payload tokenu a může tak ověřit informace o uživateli. Token je pak možné vygenerovat pomocí jednoduché funkce "jwt.sign":

```
token = jwt.sign(  
  name: this.name  
  surname: this.surname  
  email: this.email  
  roles: this.roles  
  special: this.special  
,  
  config.privateKey  
)
```

Stejně jako bylo uvedeno v předchozí kapitole, do části tokenu zvané payload je možné uložit jakékoliv informace, záleží jen na programátorovi jak velké množství dat je únosné. Protože token je přenášen v každém requestu, není vhodné to s přenášenými daty nijak přehánět. Token použitý v mé aplikaci, přenáší základní data o uživateli, včetně jeho práv.

18.2 Další zranitelnosti

Mezi další zranitelnosti patří např. Json zranitelnost, která může vést k tomu, že cizí stránka odcizí data uživatele[37]. Tomuto útoku se před lety neubrání ani gmail[38]. Útok spočívá v obejití cross-domain HTTP hlaviček, které chrání javascript kód před čtením informací z jiných stránek než byl určen. Přesto jde obejít pomocí atributu "src" v tagu <script>. Nejhorší část útoku spočívá v tom, že request na adresu, která je v atributu "src" je proveden s přihlašovacími údaji uživatele, takže je možné takto získat informace, které jsou chráněny některou formou autentizace. Moderní prohlížeče sice tuto zranitelnost odstiňují samy, ale existují sofistikovanější variace, které fungují i na moderní prohlížeče[38]. V oficiální dokumentaci AngularJS je možné nalézt záznam o tomto útoku, jehož řešením je ochrana JSON objektů prefixem, který je činí nevalidní. Tento prefix pak AngularJS sám automaticky odstraňuje. Prefix má následující formát: ")]}',\n"[31]

```
#původní JSON objekt
['one', 'two']
#s prefixem
)]}',
['one', 'two']
```

18.3 Přihlášení a soukromí uživatelů

Uživatelská data jsou samozřejmě odděleny navzájem, aby se uživatel nedostal k datům jiných uživatelů. Pokročilou možností by bylo šifrování obsahu, bohužel takovou vlastnost mongoDB dosud nemá přímo implementovanou a je otázkou, zda by se něco takového vyplatilo, především vzhledem k velkým nárokům na šifrování a dešifrování. Příkladem, že takový systém lze vytvořit, je například datové úložiště MEGA, které šifruje všechny data implicitně.

Heslo uživatele je uloženo klasicky, jako ve většině webových aplikací, v šifrované podobě, v databázi spolu s generovaným saltem pro uživatele. Vytvoření takového hesla pak probíhá tak, že se heslo uživatele přeneso na server, vytvoří se náhodný salt a salt s heslem se zašifrují bezpečnou hash funkcí. Pokud se pak uživatel přihlásí vytáhne se z databáze salt hesla a šifrované heslo uživatele, na salt a příchozí heslo se aplikuje stejná hash funkce jako při vytvoření hesla a výstup se porovná s šifrovaným heslem v databázi, pokud se řetězce shodují je ověření hesla úspěšné a dojde k vygenerování JWT tokenu.

19 GULP

Gulp je používán pro sestavení frontendu aplikace, možné je sestavit aplikaci pro produkční a vývojové účely. Gulp má pro to dva základní tasky *dev* a *prod*. Gulp konkrétně sestavuje javascript pomocí browserify, vytváří sprity¹⁾, optimalizuje obrázky, překládá LESS styly na CSS, převádí html soubory do angularové templateCache, takže se všechny views stáhnou spolu s aplikací, obnovuje prohlížeč při změnách v projektu.

19.1 Struktura

Gulp čte v adresáři, kde je spuštěn soubor *Gulpfile.js*, ten v tomto projektu pouze načítá soubor *gulp/index.js*. Načtou se všechny soubory ve složce *gulp/tasks* a následně spustí task určený parametrem příkazu *gulp*.

¹⁾sprite se používá u webu pro snížení počtu requestů při načítání. Spojuje obrázky do jednoho velkého obrázku, ze kterého jsou obrázky v CSS referencovány pomocí souřadnic a velikosti

19.2 Development/Production

Development prostředí oproti produkčnímu nastavuje některé podrobnější výpisy a především neprovádí optimalizaci kódu a obrázků.

Tasky:

Browserify - Jak již bylo výše uvedeno Browserify překládá zdrojové kódy v coffeescriptu na javascript. Navíc má aplikace několik prostředí, kde má každé vlastní kód. Proto je vytváření prostředí ve smyčce.

```
var enviroments = ["editor","store","admin"]
enviroments.forEach(function (element, index, array) {
  return browserify({
    entries: ['./src/app/'+element+'.coffee'],
    extensions: ['.coffee'],
    transform: ["coffeesify","brfs","envify","browserify-shim"]
  })
  .bundle({debug: true})
  .on('error', handleError)
  .pipe(source('.'+element+'.js'))
  .pipe(gulp.dest('./assets/js/'))
  .pipe(livereload());
})
```

Clean - Clean pouze čistí projekt před buildem, aby ve složce kam bude buildováno nezůstaly staré soubory.

```
gulp.task('clean', function() {
  return gulp.src(["./build", "./tmp/**/*"], {
    read: false
  }).pipe(clean());
});
```

Images - Images především kopíruje obrázky se zdrojové složky, v produkčním prostředí pak obrázky ještě optimalizuje. *pozn. cesta k souboru `*._NS.png` je přítomna, protože všechny png soubory jsou automaticky převáděny do spritu, kromě souborů se sufixem `"_NS"`, které jsou zpracovány právě zde.*

```
gulp.task('images', function() {
  return gulp.src('./src/images/**/*{.ico,.svg,.gif,_NS.png}')
    .pipe(changed(dest))
    .pipe(imagemin())
    .pipe(gulp.dest('./build/images'));
});
```

Less - Task Less při svém volání vždy spustí task SPRITES. Je provedena kontrola pomocí *recess* a css kód je automaticky prefixován pro prohlížeče nastavené v konfiguraci. Při produkčním buildu se pak soubor, ještě minifikuje nastavuje se se

mu přípona *.min* a modul *rev* vygeneruje do názvu souboru náhodný řetězec, který zaručí, že pokud proběhne změna aplikace na severu, klient stáhne html soubor, a protože je jiný název css souboru než má uloženo prohlížeč v cache tak prohlížeč stáhne novou verzi. Pokud by se název souboru nezměnil, načetl by prohlížeč soubor z paměti a stránka by byla zastaralá a třeba i nefunkční.

```
gulp.task('LESS', ["SPRITES"], function() {
  return gulp.src("./src/less/main.less")
    .pipe(recess())
    .pipe(less())
    .pipe(autoprefixer(confAutoPrefix))
    .pipe(minifycss())
    .pipe(rename({suffix: ".min"}))
    .pipe(rev())
    .pipe(gulp.dest("./assets/css"))
});
```

Sprites - Sprity především snižují počet requestů při načítání aplikace a zjednodušují vkládání obrázků při kódování. Weby mají často množství obrázkových ikon a obrázků potřebných pro fungování grafiky, které pokud se načítají samostatně, je na každý obrázek třeba jeden request. Sprite tedy sdružuje všechny obrázky v jenom velkém obrázku, který je pak z LESS referencován pomocí mixinu. Pokud se tedy například ve složce *src/images* nachází soubor *ikona.png*, je vložen do spritu a je možné jej referencovat kdekoliv v LESS dokumentu, pokud includeje soubor *tmp/sprites.less* jednoduše pomocí mixinu a proměnné: *.sprite(@ikona)*;

```
var spriteConf={
  base64: true,
  name: 'sprite.png',
  style: 'sprites.less',
  cssPath: 'images/',
  processor: 'less'
};
gulp.task('SPRITES', function() {
  return gulp.src(["./src/images/**/*.png",
    "!./src/images/**/*.ns.png"])
    .pipe(plumber())
    .pipe(sprite(spriteConf))
    .pipe(gulpif '*.png', gulp.dest("./assets/images")))
    .pipe(gulpif '*.less', gulp.dest("./src/less"));
});
```

Templates - Tento task se stará o převod šablon ze zdrojových souborů. Možné je psát šablony jak v HTML tak v JADE. Další možností šablon je pak templateCache, kdy se html převede na javascript, spojí se do jednoho souboru a stáhnou aplikací při prvním načtení. Stejně jako sprity, převod html šablon do jednoho javascriptu snižuje také značně počet requestů. Pokud je použit templateCache

musí, vývojář počítat s tím, že veškeré zobrazování a skrývání obsahu, např. z důvodu oprávnění, musí být ošetřeno ve frontendu pomocí AngularJS. Nejedná se ale o velkou překážku a v případě potřeby může být část šablon ponechána v html a stáhnuta pomocí xhr requestu.

```
gulp.src(["./src/**/*.html"])
  .pipe(gulp.dest("./assets/partials"));
gulp.src(["./src/**/*.jade"])
  .pipe(jade({
    pretty:true,
    debug:false
  }))
  .pipe(gulp.dest("./assets/partials"));
```

Watch - Watch pouze hlídá změny v souborech, pokud se změní soubor, který hlídá nebo přibude nový, který odpovídá jeho nastavení spustí nastavený task.

```
gulp.watch('src/**/*.coffee', ['browserify_dev']);
gulp.watch(['!src/images/**/*.NS.png',
  'src/images/**/*.png', 'src/**/*.less'], ['LESS_DEV']);
gulp.watch('src/images/**/*.{ico,.svg,.gif,.NS.png}',
  ['images_dev']);
gulp.watch(['src/**/*.html', 'src/**/*.jade'], ['templates']);
```

Konečné řešení pro spojování souborů za účelem snížení počtu requestů by měl přinést chystaný protokol **HTTP 2.0**, který přinese multiplexování requestů a respons, lepší kompresi a mnoho jiného.[49]

20 NASAZENÍ APLIKACE (SERVER,CLOUD)

V současné době, kdy jsou moderní cloudové služby, které umožňují s minimální počáteční investicí spustit projek prakticky jakékoliv velikosti. Je velmi dobré zvážit cloudové nasazení buď samostatné databáze nebo většinou kompletního projektu. Samozřejmě záleží především na finanční situaci provozovatele naší služby, jejím aktuálním stavu a návštěvnosti.

Cloud řeší především velké počáteční náklady na zprovoznění služby. Díky škálovatelnosti cloudových služeb nemusíme nijak zvlášť řešit jak velké množství prostředků bude potřeba. Není třeba kupovat fyzické stroje nebo pronajímat dedikované servery. Není třeba najímat zaměstnance, kteří by zajišťovali chod serverů nebo platit za manage řešení.

Cloud také dává možnosti rozdělení služeb a jejich chod u několika poskytovatelů ať už kvůli dalšímu šetření nákladů (např. pokud bude MongoDB hosting levnější u jiného poskytovatele bude u jednoho aplikační část a u druhého databázová část apod.) nebo dostupnosti z globálního hlediska v různých zemích či kontinentech.

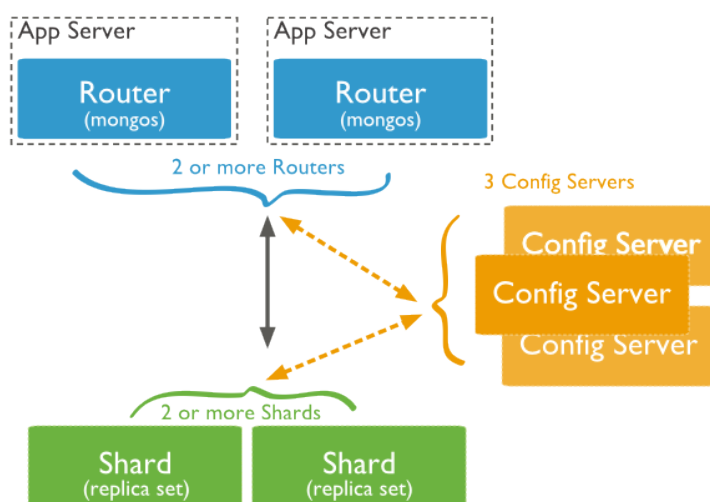
Díky cloudu je také možné odhadnout náklady a nároky na koupi vlastní serverové infrastruktury, protože po dobu provozu v cloudu, by bylo nasbíráno velké množství metrik a následně odhadnuty nároky na vlastní řešení.

Velmi vhodným je pak hybridní řešení, kdy je část aplikace provozována na vlastním serveru a část v cloudu. Často je v cloudu provozována především část databázová.

V případě koupi vlastní serverové infrastruktury, je třeba myslet na vysokou dostupnost služby tzv. HA¹⁾. Nestačí tedy koupit jeden fyzický stroj a něm provozovat službu. Je třeba koupit minimálně dva stroje a pokud jeden vypadne zastoupí jej server druhý. Takové řešení má, ale své nevýhody, ta největší je především to, že druhý server běží a spotřebovává energii naprosto zbytečně přičemž vyžaduje stejnou údržbu a hrozí u něj stejná pravděpodobnost HW chyby jako u serveru, který je využíván. Proto je vhodné před zmíněné (nebo více serverů) umístit load balancer, který by rovnoměrně rozdělil zátěž mezi ně. Díky load balanceru, jsou servery rovnoměrně využívány a v případě výpadku převzou automaticky všechny požadavky funkční servery.

20.1 MongoDB

K rozdělení zátěže a lepšího škálování služeb přispívá i samotné MongoDB, které umožňuje sharding několika MongoDB serverů a replikaci data. Sharding, je ale velmi náročný a k zprovoznění shardingu je třeba alespoň 7mi serverů, 3 konfigurační, 2 a více routerů a 2 a více replik.[46] Sharding je třeba vhodný především pokud množství pa-

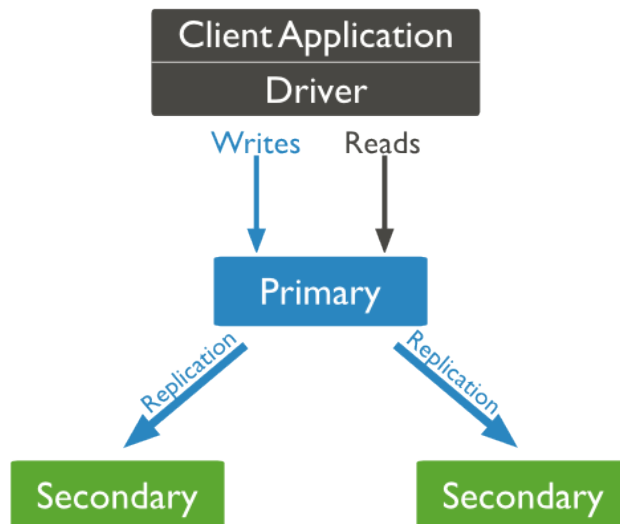


Obr. 20. Diagram produkčního clusteru pro MongoDB sharding.

¹⁾HA - High Availability - dva nebo více serverů jsou vzájemnými klony, a v případě výpadku jednoho, nebo více, serverů výpadek koncový uživatel nezaznamená, maximálně může dojít ke zpomalení služby v případě, že by výkon po výpadku byl nedostatečný.

měti RAM nedostačuje MongoDB požadavkům, nebo nedostačuje diskový prostor.[46]

Jednodušší možnosti škálovatelnosti MongoDB je replikace, která je třeba i v případě že je použit sharding. Jedná se o podobný koncept jako je load balancování requestů pro webový server. Je jeden primární MongoDB server, který přijímá požadavky a preposílá je pro zpracování sekundárním serverům, které vrací výsledky.



Obr. 21. Diagram jednoduché replikace MongoDB serverů.

Škálovatelnost databáze samozřejmě řešíme pouze pokud využíváme vlastní infrastrukturu nebo některé cloudové IaaS²⁾ služby jako je Amazon EC2. V případě, že využíváme pro databázi některou ze služeb SaaS³⁾ neřešíme škálovatelnost, záleží totiž na tom, jaké podpůrné služby si zaplatíme. V jedné ceně je možné mít replikovaný cluster včetně záloh.

20.1.1 Cloudové služby

Mezi známější služby nabízející MongoDB patří např. mongodirector.com nebo mongolab.com obě služby jsou podobné, nabízejí MongoDB service v IaaS službách jiných poskytovatelů např. Amazon AWS, Windows Azure atd. Infrastruktura je dle zvoleného plánu během několika minut aktivní a použitelná, bez jakékoliv znalosti nastavení serverů a řešení jejich zabezpečení. Přičemž ceny vhodné pro provoz poměrně velkých webů začínají kolem 100\$, kdy je možné získat dva replikované mongoDB servery 2GB

²⁾Infrastructure as service - cloudová služba kdy uživatel získá jako by přímo vlastní diskový a výpočetní výkon a chová se k němu jako vlastnímu dedikovanému serveru

³⁾Service as Service - uživateli je pronajímána pouze služba s omezeným diskovým prostorem a výpočetním výkonem

RAM, 15GB SSD, dvě procesorové jádra a možnost kdykoliv zvýšit výkon pouhým kliknutím.

20.2 NodeJS

V oblasti nasazení NodeJS aplikací, je opět možnost nasazení buď na vlastní server nebo do cloudu. V případě vlastního serveru spočívá nasazení v deploy kódu na server a spuštění nebo restart nodeJS aplikace.

20.2.1 Vlastní server

Častou kombinací aplikací zajišťující běh nodeJS aplikace je Proxy server NGINX, Monit pro trvalý běh aplikace, JenkinsCI pro deploy kódu a například Upstart pro spuštění aplikace samotné.

Nastavení NGINX včetně websocketů a SSL certifikátu

```
upstream server {
    server 127.0.0.1:8080;
}
server {
    listen 8085 ssl;
    server_name mojeaplikace.cz;
    ssl_certificate /etc/ssl/mojeaplikace/mojeaplikace.cz.crt;
    ssl_certificate_key /etc/ssl/mojeaplikace/mojeaplikace.cz.key;
    access_log /var/log/nginx/mojeaplikace.log;
    location / {
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header Host $http_host;
        proxy_set_header X-NginX-Proxy true;
        proxy_pass http://server;
        proxy_redirect off;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
    }
}
```

Monit se pak stará o trvalý běh aplikace, její resart po deploy pomocí změny souboru *restart*

```
check process mojeaplikace with pidfile "/var/run/node...app.pid"
    start program = "/sbin/start mojeaplikace"
    stop program = "/sbin/stop mojeaplikace"
    if failed port 8085
        type tcpSSL protocol http
        request /
        with timeout 10 seconds
        then restart
    depends on mojeaplikace_restart
check file mojeaplikace_restart
    with path /media/data/nodejs/mojeaplikace/restart
    if changed timestamp then restart
```

Upstart Script pro spuštění NodeJS aplikací:

```
#!/upstart
description "nodejs mojeaplikace"
start on started mongodb
stop on shutdown
env LOG="/var/log/nodejs/mojeaplikace.log"
env PID="/var/run/nodejs-mojeaplikace.pid"
env nodejsexec="/usr/bin/node"
env app="/media/data/nodejs/mojeaplikace/app.js"
script
    export HOME="/home/nodejs"
    echo $$ > $PID
    exec sudo -u nodejs $nodejsexec $app >> $LOG 2>&1
end script
pre-start script
    # Date format same as (new Date()).toISOString() for consistency
    echo "[`date -u +%Y-%m-%dT%T.%3NZ`] (sys) Starting" >> $LOG
end script
pre-stop script
    rm $PID
    echo "[`date -u +%Y-%m-%dT%T.%3NZ`] (sys) Stopping" >> $LOG
end script
```

JenkinsCI pak jenom zkopíruje kód z GIT repozitáře a zapíše náhodný řetězec do restart souboru, který zkontroluje monit a restartuje upstart script.

Ještě je důležité zmínit, že nodejs běží v jenom vlákně, pokud tedy chceme využít více jader procesoru musíme spustit více instancí a balancovat mezi nimi, jak už bylo výše zmíněno, nebo je možné také použít nodeJS cluster, který vytvoří proxyserver a o balancování se stará sám.[44]

```
cluster = require("cluster")
http = require("http")
numCPUs = require("os").cpus().length
if cluster.isMaster
    i = 0
    while i < numCPUs
        cluster.fork()
        i++
    cluster.on "exit", (worker, code, signal) ->
        console.log "worker " + worker.process.pid + " died"
        return
else
    # HTTP SERVER
    http.createServer((req, res) ->
        res.writeHead 200
        res.end "hello world\n"
        return
    ).listen 8000
```

Kód vytvoří nodejs server na portu 8000 a pro každé jádro procesoru vytvoří jeden worker.[44]

20.2.2 Cloud

V oblasti cloudu je možností také nepočítaně. Mezi nejznámější služby patří např. průkopník v této oblasti Heroku (<http://heroku.com>), NodeJitsu <https://www.nodejitsu.com/>

nebo relativně nová služba od společnosti Red Hat Openshift <https://www.openshift.com/>. Všechny služby jsou velice jednoduché na používání nabízejí jak webové rozhraní pro vytváření aplikací tak jednoduchý deployment pomocí příkazové řádky. Navíc všechny poskytují zároveň MongoDB hosting takže není třeba při startu aplikace využívat externí cloudové služby pro databázi.

Vzhledem k tomu, že je deploy pro všechny cloudové služby prakticky stejný bude uveden příklad pro OpenShift.

Pro instalaci a využití Openshiftu je třeba mít nainstalováno ruby, pak stačí jednoduše nainstalovat gem rhc.

```
sudo gem install rhc
```

Nastavit program rhc, jedná se pouze o přihlášení a nastavení SSH klíče.

```
rhc setup
```

Pak stačí vytvořit aplikaci, příkaz nejenom vytvoří aplikační cartridge v cloudu, ale i git repozitář v místě spuštění příkazu.

```
rhc app create MyApp nodejs-0.10 mongodb-2.6
```

Deploy už je pak pouze vytvoření commitu v GITu a deploy pomocí příkazu PUSH.

```
$cd myapp  
$nano app.js  
$git commit -a -m "Deploy"  
$git push
```

Webovou adresu aplikace je pak možné získat po přihlášení na webu openshift. Openshift navíc nabízí i deploy pomocí JenkinsCI nebo Cloud9.

ZÁVĚR

Cílem této diplomové práce bylo vytvořit návrh budoucího webového informačního systému, optimalizovaného pro potřeby knihkupectví s možností tvorby knih online pomocí typografického systému L^AT_EX. Výsledná aplikace měla být vytvořena na základě technologií: MongoDB jako databáze, NodeJS jako serverové části a AngularJS pro realizaci klientské části. Všechny uvedené technologie se ženou kupředu až neuvěřitelnou rychlostí, která odpovídá potřebám a vývoji dnešních webových technologií. Tentok fakt sebou nese a také přinesl řadu obtíží, se kterými se každý programátor při vývoji setká.

První problém, na který narazí je výběr potřebných technologií. Ve sféře webových javascript frameworků existují desítky řešení, které řeší stejné problémy různými cestami, AngularJS byl zvolen vzhledem ke své současné popularitě a příjemném řešení provázanosti s html pomocí directives. Práce s ním se tak stává příjemnou a jednoduchou. AngularJS se také stal mou osobní volbou mezi javascriptovými frameworky.

To samé jako na u volby technologií pro frontend platí i pro NodeJS aplikace jen je situace ještě mnohem více složitá, protože množství balíčků, mezi kterými je možné vybrat, a které často nesou stejnou funkcionalitu, je nepřehledné a balíčky jsou často dlouho nevyvíjeny nebo je jejich vývoj ukončen dříve než vůbec programátor stihne dokončit aplikaci. Tento fakt je pravděpodobně největším úskalím počátku projektu a při špatné volbě balíčků, hrozí situace ,kdy se projekt pozdrží, kvůli výměně závislostí. Na druhou stranu NodeJS balíčky nenesou většinou nijak komplexní funkcionalitu a lze je nahradit poměrně snadno.

Důležité je nejprve zvolit kvalitní framework, který by měl mít aktivní komunitu, větší množství vývojářů a v nejlepším případě by za ním měla stát nějaká firma. Po dlouhém zvažování a zkoušení různých frameworků, jsem zvolil pro realizaci diplomové práce framework HapiJS za který stojí gigantický obchodní řetěze Walmart. Serverová část pro mě byla osobně největším přínosem, velmi se zajímám o nové technologie ve vývoji webů a NodeJS jsem chtěl již nějakou dobu vyzkoušet, ale nenašel jsem vhodný čas ani projekt, na kterém bych mohl reálně vyzkoušet jeho možnosti.

Po zvolení základních technologií, je třeba připravit vývojové prostředí, které by mělo automatizovat úkoly, které zdržují vývojáře od skutečných problémů, k tomuto účelu byl zvolen task manager Gulp, který automatizuje všechny často se opakující úkoly. V rámci práce byla vytvořena sada tasků, které automatizují vše co je třeba k efektivnímu psaní aplikace. Gulp se stal také velkým problémem při vývoji, vzhledem k rozsahu aplikace, začal občas nespolehlivě spracovávat data a byl třeba znovu spustit, což se stalo velmi nepříjemným.

Navržena byla komplexní databáze v MongoDB. Návrh databáze nebyl zdaleka tak

jednoduchý, jak vypadá výsledný diagram. Oproti klasickým SQL databázím jsou dokumentové NoSQL databáze odlišné téměř ve všem. Především je třeba úplně zapomenout na zásady, pro vytváření SQL databází. V SQL je snaha především o rozdělení databáze na menší celky podle normálních forem, v NoSQL je snaha vměstnat do jednoho dokumentu co možná nejvíce data a pokud jsou data referencována je vhodné ukládat jejich části také k referenci. MongoDB jsem stejně jako NodeJS nikdy u reálného projektu nevyužil, ale práce sním je natolik pohodlná a rychlá, že nelze než jej doporučit ať už jako hlavní databázový server tak v kombinaci s jinými.

Aplikace samotná byla logicky rozdělena do několik menších celků vzhledem k funkcionalitě a velikosti případných knihoven tak, aby načtení e-shopu, který má nejvíce potenciálních uživatelů trvalo co nejkratší dobu. Eshop je realizován tak, aby poskytoval navíc zajímavou funkci v podobě carouselu, který slouží pro prezentaci knih. Carousel je navíc optimalizován také pro "touch"zařízení, jako jsou tablety.

Dále pak byl vytvořen jednoduchý editor pro tvorbu a překlad knih psaných v $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ u. Editor a možnost editace se podařila realizovat úspěšně, nicméně význam je čistě edukativní, protože pokročilé editory dostupné na internetu mají množství funkcí jejich jednotlivé implementace by pravděpodobně bylo možné realizovat v samostatných pracích nejlépe týmu vývojářů.

Další částí, kterou se práce zabývala, byla bezpečnost aplikace a to zajištěním bezpečnosti přenosu informací i ochranou dat a systému jako takového. Byly rozebrány nejčastější formy útoků na webové aplikace a možnosti jak se před nimi chránit. Byly také krátce popsány možnosti nastavení základního zabezpečení databáze MongoDB. V poslední části byly pak rozebrány možnosti nasazení výsledné aplikace na dedikované servery i do cloudu. Byly představeny některé cloudové služby, které mohou usnadnit vývoj a nasazení výsledné aplikace. Byly také představeny konfigurační soubory potřebné k zprovoznění obecně jakékoliv NodeJS aplikace na vlastním serveru.

Musím říci, že jsem byl z možnosti volby vlastních technologií pro práci nadšen a toto nadšení trvá. V rámci rozšíření znalostí byla práce velkým přínosem. Škoda je jen to, že jsem vynaložil hodně času na přípravu platformy pro projekt a zkoušením velkého množství knihoven a frameworků. Kdyby se dal tento čas eliminovat bylo by možné jej vložit do rozšíření funkcí aplikace samotné. V každém případě jsou přínosit práce s výukového hlediska velmi hodnotné a odnáším si množství znalostí, které se určitě neztratí a budou využity již při tvorbě příštího projektu.

SEZNAM POUŽITÉ LITERATURY

- [1] Ng-book: The Complete Book on AngularJS [online]. Fullstack io, 2013 [cit. 2014-02-01]. ISBN 978-0991344604. Dostupné z: <https://www.ng-book.com/>. [e-kniha]
- [2] Mastering Node.js [online]. Packt Publishing, 2013 [cit. 2014-02-01]. ISBN 978-1782166320. Dostupné z: <http://visionmedia.github.io/masteringnode/>.
- [3] POYNTER, Dan. Dan Poynters self-publishing manual: how to write, print and sell your own book. 15th ed., completely rev. Santa Barbara, Calif.: Para Pub., 2006, 463 p. ISBN 978-156-8601-342.
- [4] DUCKETT, Jon. HTML: design and build websites. Indianapolis, IN: Wiley, c2011, 490 p. ISBN 11-180-0818-9.
- [5] OETIKER, Tobias, Hubert PARTL, Irene HYNA a Elisabeth SCHLEGL. The Not So Short Introduction to LATEX [online]. [cit. 2014-02-01]. Dostupné z: <http://www.ctan.org/tex-archive/info/lshort/english/>.
- [6] Unique book goes on display. BBC NEWS | Europe [online]. 2003 [cit. 2014-05-21]. Dostupné z: <http://news.bbc.co.uk/2/hi/europe/2939362.stm>
- [7] Nejstarší knihkupectví na světě. Bývalé centrum umělců a intelektuálů. Světy literatury [online]. 2014 [cit. 2014-05-21]. Dostupné z: <http://svetyliteratury.cz/?p=3424>
- [8] HAYMOND, Bryce. Authentic Ancient Metal Plates. LDS Temples, Mormon Temples, Study Blog [online]. [cit. 2014-05-21]. Dostupné z: <http://www.templestudy.com/2011/04/07/authentic-ancient-metal-plates/>
- [9] PETERSON, Valerie. Gutenberg - Johann Gutenberg and the Invention of the Printing Press. In: About.com [online]. 2014 [cit. 2014-05-21]. Dostupné z: <http://publishing.about.com/od/Books/a/Gutenberg-Johann-Gutenberg-And-The-Invention-Of-The-Printing-Press.htm>
- [10] Timeline History Amazon.com. In: Amazon-genius [online]. 2012, 11-12-2012 [cit. 2014-05-21]. Dostupné z: <http://amazongenius.com/timeline-history-amazon-com/>
- [11] CLARK, Jack. How Amazon exposed its guts: The History of AWS's EC2. ZDNet [online]. 2012 [cit. 2014-05-21]. Dostupné z: <http://www.zdnet.com/how-amazon-exposed-its-guts-the-history-of-awss-ec2-3040155310/>

- [12] ŠMÍD, Vladimír. Management informačního systému [online]. [cit. 2014-05-22]. Dostupné z: <http://www.fi.muni.cz/smid/mis-infosys.htm>
- [13] JŮNA, Jan. Co je to event-driven architektura?. In: Node.js [online]. 2014 [cit. 2014-05-21]. Dostupné z: <http://www.nodejs.cz/co-je-event-driven-architektura/>
- [14] Express - api reference. Express [online]. 2014 [cit. 2014-05-21]. Dostupné z: <http://expressjs.com/3x/api.html>
- [15] Trendy v informačních systémech očima analytiků. In: BusinessIT [online]. 2011 [cit. 2014-05-21]. Dostupné z: <http://www.businessit.cz/cz/analytici-informacni-systemy-cloud-socialni-crm-business-intelligence.php>
- [16] THON, Ladislav. Lehký úvod do MongoDB. ABC Linuxu [online]. 2010 [cit. 2014-05-21]. Dostupné z: <http://www.abclinuxu.cz/clanky/programovani/lehky-uvod-do-mongodb>
- [17] Angular.js. Github [online]. 2014 [cit. 2014-05-21]. Dostupné z: <https://github.com/angular/angular.js>
- [18] MEYERS, Cliff. Code Organization in Large AngularJS and JavaScript Applications. Cliff Meyers [online]. 2013 [cit. 2014-05-21]. Dostupné z: <http://cliffmeyers.com/blog/2013/4/21/code-organization-angularjs-javascript>
- [19] Animations. AngularJS: Developer Guide [online]. 2014 [cit. 2014-05-21]. Dostupné z: <https://docs.angularjs.org/guide/animations>
- [20] Getting started. Less.js [online]. 2014 [cit. 2014-05-21]. Dostupné z: <http://lesscss.org/>
- [21] Tutorial. Jade - Template Engine [online]. 2014 [cit. 2014-05-21]. Dostupné z: <http://jade-lang.com/tutorial/>
- [22] Bower A package manager for the web. Bower [online]. 2014 [cit. 2014-05-21]. Dostupné z: <http://bower.io/>
- [23] Server Framework for Node.js. Hapi [online]. 2014 [cit. 2014-05-21]. Dostupné z: <http://spumko.github.io/>
- [24] ES6 Module Transpiler - Tomorrow's JavaScript module syntax today. In: Github [online]. 2014 [cit. 2014-05-21]. Dostupné z: <http://corner.squareup.com/2013/02/es6-module-transpiler.html>

- [25] ECMAScript 6 Modules: What Are They and How to Use Them Today. In: InfoQ [online]. 2013 [cit. 2014-05-21]. Dostupné z: <http://www.infoq.com/news/2013/08/es6-modules>
- [26] Browserify. Browserify [online]. 2014 [cit. 2014-05-21]. Dostupné z: <http://browserify.org/>
- [27] SUURONEN, Esa-Matti. Journey From RequireJS to Browserify. In: Esa-Matti Suuronen [online]. 2013 [cit. 2014-05-21]. Dostupné z: <http://esa-matti.suuronen.org/blog/2013/03/22/journey-from-requirejs-to-browserify/>
- [28] MICHÁLEK, Martin. K čemu je dobrý Bootstrap a frontend frameworky?. In: Zdroják [online]. 2013 [cit. 2014-05-21]. Dostupné z: <http://www.zdrojak.cz/clanky/k-cemu-je-dobry-bootstrap-frontend-frameworky/>
- [29] Using LESS with Bootstrap. Bootstrap, from Twitter [online]. 2014 [cit. 2014-05-21]. Dostupné z: <http://bootstrap.lesscss.ru/less.html>
- [30] Getting started. Bootstrap [online]. 2014 [cit. 2014-05-21]. Dostupné z: <http://getbootstrap.com/getting-started/>
- [31] \$http. AngularJS API [online]. 2014 [cit. 2014-05-21]. Dostupné z: [https://docs.angularjs.org/api/ng/service/\\$http](https://docs.angularjs.org/api/ng/service/$http)
- [32] LESS Hat 3.0. Github [online]. 2014 [cit. 2014-05-21]. Dostupné z: <https://github.com/madebysource/lesshat/blob/master/README.md>
- [33] JOHNSON, Joshua. Battle of the LESS Mixin Libraries: LESS Elements vs. LESS Hat vs. Bootstrap. Design shack [online]. 2012 [cit. 2014-05-21]. Dostupné z: <http://designshack.net/articles/css/battle-of-the-less-mixin-libraries-less-elements-vs-less-hat-vs-bootstrap/>
- [34] MARTINEK, David. CsLaTeX a balík czech. In: Jak sázet česky [online]. 2008, 2-10-2008 [cit. 2014-05-21]. Dostupné z: <http://www.fit.vutbr.cz/~martinek/latex/czech.html>
- [35] 4.0.x API Reference. Github [online]. 2014 [cit. 2014-05-21]. Dostupné z: <https://github.com/spumko/hapi/blob/v4.1.4/docs/Reference.md>
- [36] Dart: Structured web apps. Dart [online]. 2014 [cit. 2014-05-21]. Dostupné z: <https://www.dartlang.org/>

- [37] Anatomy of a Subtle JSON Vulnerability. You've Been Haacked [online]. 2008 [cit. 2014-05-21]. Dostupné z: <http://haacked.com/archive/2008/11/20/anatomy-of-a-subtle-json-vulnerability.aspx/>
- [38] GROSSMAN, Jeremiah. Advanced Web Attack Techniques using Gmail. Jeremiah Grossman [online]. 2006 [cit. 2014-05-21]. Dostupné z: <http://jeremiahgrossman.blogspot.cz/2006/01/advanced-web-attack-techniques-using.html>
- [39] ČERNÝ, Michal. Online LaTeX editory - plnohodnotná náhrada?. Lupa.cz [online]. 2010 [cit. 2014-05-21]. Dostupné z: <http://www.lupa.cz/clanky/online-latex-editory-plnohodnotna-nahrada/>
- [40] One-click compiling of code in the forum with writeLaTeX. Latex community [online]. 2013 [cit. 2014-05-21]. Dostupné z: <http://latex-community.org/forum/viewtopic.php?f=37&t=22340#>
- [41] Create a User Administrator. MongoDB Manual 2.6.1 [online]. 2014 [cit. 2014-05-21]. Dostupné z: <http://docs.mongodb.org/manual/tutorial/add-user-administrator/>
- [42] Úvod. Jak vydat knihu [online]. [cit. 2014-05-21]. Dostupné z: <http://jakvydatknihu.wikina.cz/>
- [43] BEDNÁŘ, Vojtěch. Vydejte si vlastní knihu. Přes Internet. In: Lupa.cz [online]. 2010 [cit. 2014-05-21]. Dostupné z: <http://www.lupa.cz/clanky/vydejte-si-vlastni-knihu-pres-internet/>
- [44] CANNADAY, Brandon. Planning for the Horizontal: Scaling Node.js Applications. Slideshare [online]. 2013 [cit. 2014-05-21]. Dostupné z: <http://www.slideshare.net/OnModulus/planningforthehorizontal-scalingnode-js-18572644>
- [45] Quick Start. TypeScript [online]. 2014 [cit. 2014-05-21]. Dostupné z: <http://www.typescriptlang.org/Tutorial>
- [46] Sharding Concepts. MongoDB Manual 2.6.1 [online]. 2014 [cit. 2014-05-21]. Dostupné z: <http://docs.mongodb.org/manual/core/sharding/>
- [47] JONES, Michael B., John BRADLEY a Nat SAKIMURA. JSON Web Token (JWT): draft-ietf-oauth-json-web-token-20. Self-issued [online]. 2014 [cit. 2014-05-21]. Dostupné z: <http://self-issued.info/docs/draft-ietf-oauth-json-web-token.html>

- [48] JSON Web Tokens, OWIN, and AngularJS. Code rant [online]. 2014 [cit. 2014-05-21]. Dostupné z: <http://mikehadlow.blogspot.cz/2014/04/json-web-tokens-owin-and-angularjs.html>
- [49] BELSHE, Mike, Roberto PEON a Martin THOMSON. Hypertext Transfer Protocol version 2: draft-ietf-httpbis-http2-latest. Github [online]. 2014 [cit. 2014-05-21]. Dostupné z: <http://http2.github.io/http2-spec/index.html#StreamsLayer>
- [50] TORVALDS, Linus. *LKML.ORG - the Linux Kernel Mailing List Archive* [online]. 2005-04-07 [cit. 2011-05-26]. Linus Torvalds: Re: Kernel SCM saga. Dostupné z WWW: <http://lkml.org/lkml/2005/4/8/9>.
- [51] CHACON, Scott. *Git* [online]. 2005 [cit. 2011-05-26]. Git - Fast Version Control System. Dostupné z WWW: <http://git-scm>
- [52] GitHub Inc. *Help.GitHub* [online]. 2011 [cit. 2011-05-26]. Set Up Git (Linux). Dostupné z WWW: <http://help.github.com/linux-set-up-git/>.
- [53] Linux Kernel Organization, Inc. *The Linux Kernel Archives* [online]. 2006 [cit. 2011-05-26]. Git User's Manual (for version 1.5.3 or newer). Dostupné z WWW: <http://www.kernel.org/pub/software/scm/git/docs/user-manual.html#how-to-merge>.

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

| | |
|------|------------------------------------|
| API | Application Programming Interface |
| EC2 | Elastics cloud |
| ISBN | International Standard Book Number |
| ES6 | ECMAScript6 |
| AMD | Asynchronous module definition |
| JS | JavaScript |
| CS | CoffeeScript |
| CSS | Cascading Style Sheets |
| HTML | HyperText Markup Language |
| LAMP | Linux,Apache,MySQL,PHP |
| MEAN | MongoDB,ExpressJS,AngularJS,NodeJS |
| OS | Opravní systém |
| PHP | Hypertext Preprocessor |
| URL | Uniform Resource Locator |
| ODM | Object Document Mapper |
| ORM | Object Relation Mapper |
| SQL | Structured Query Language |
| JSON | JavaScript Object Notation |
| UI | User Interface |
| UX | User eXperience |
| DOM | Document Object Model |
| SASS | Syntactically Awesome Style Sheets |
| SVG | Scalable Vector Graphics |
| NPM | Node Package Manager |
| PDF | Portable Document Format |
| ACE | Ajax.org Cloud9 Editor |
| SPA | Single Page Application |
| SW | Software |
| PC | Personal Computer |
| VPN | Virtual Private Network |
| SSH | Secure Shell |
| CSRF | Cross Site Request Forgery |
| JWT | JSON web token |
| AWS | Amazon Web Storage |
| SSL | Secure Sockets Layer |
| CI | Continuous Integration |

SEZNAM OBRÁZKŮ

| | |
|--|----|
| Obr. 1. Logo coffeescript | 16 |
| Obr. 2. Logo Browserify | 20 |
| Obr. 3. Logo NodeJS | 21 |
| Obr. 4. Event-Loop..... | 21 |
| Obr. 5. Logo ExpressJS..... | 23 |
| Obr. 6. Logo MongoDB..... | 28 |
| Obr. 7. Logo AngularJS | 30 |
| Obr. 8. Logo Less | 36 |
| Obr. 9. Logo Twitter Bootstrap..... | 37 |
| Obr. 10. Logo LessHat | 38 |
| Obr. 11. Logo Jade..... | 39 |
| Obr. 12. Logo Bower | 40 |
| Obr. 13. Logo GulpJS..... | 41 |
| Obr. 14. Screenshot editoru Codecogs..... | 46 |
| Obr. 15. Screenshot editoru ShareLaTeX | 47 |
| Obr. 16. Screenshot editoru WriteLaTeX | 47 |
| Obr. 17. Diagram struktury aplikace | 50 |
| Obr. 18. Diagram databáze | 54 |
| Obr. 19. Diagram API | 70 |
| Obr. 20. Diagram MongoDB sharding | 81 |
| Obr. 21. Diagram MongoDB replikace | 82 |

SEZNAM PŘÍLOH

- P I. PDF verze diplomové práce a zdrojový kód v jazyce LaTeX
- P II. Zdrojové kódy prototypu aplikace informačního systému knihkupectví

PŘÍLOHA P I. PDF VERZE DIPLOMOVÉ PRÁCE A ZDROJOVÝ KÓD V JAZYCE LATEX

PDF je dostupné na CD nosiči spolu s diplomovou prací, na portále github.com ve vlastním GIT repozitáři a na webu [writelatex.com](https://www.writelatex.com).

Pro nahlédnutí do poslední verze PDF.

Git repozitář se zdrojovými kódy:

<https://github.com/zajca/DP>

Online verze na webu Write \LaTeX .com:

<https://www.writelatex.com/read/qphsjggmntmv>

PŘÍLOHA P II. ZDROJOVÉ KÓDY PROTOTYPU APLIKACE INFORMAČNÍHO SYSTÉMU KNIHKUPECTVÍ

Zdrojové kódy aplikace jsou dostupné na CD nosiči spolu s diplomovou prací a na portále github.com ve vlastním git repozitáři.

Git repozitář se zdrojovými kódy: <https://github.com/zajca/ISK2>

Návod na zprovoznění aplikace se nachází v souboru README.md v kořenovém adresáři aplikace.