Jihočeská univerzita v Českých Budějovicích
Přírodovědecká fakulta

Současný ekosystém Rich Internet Applications v HTML5

Bakalářská Práce

Jakub Vacek, DiS.

Ing. Martin Čížek, MBA

České Budějovice 2017

Jihočeská univerzita v Českých Budějovicích
Přírodovědecká fakulta

ZADÁVACÍ PROTOKOL BAKALÁŘSKÉ PRÁCE

Student: Jakub Vacek DiS.
(jméno, příjmení, tituly)

Obor – zaměření studia: Aplikovaná informatika

Katedra/ústav, kde bude práce vypracovávána: Ústav aplikované informatiky

Školitel: Ing. Martin Čížek, MBA
(jméno, příjmení, tituly, u externího š. název a adresa pracoviště, telefon, fax, e-mail)

Garant z PřF: RNDr Libor Dostálek
(jméno, příjmení, tituly, katedra – jen v případě externího školitele)

Školitel – specialista, konzultant: .....
(jméno, příjmení, tituly, u externího š. název a adresa pracoviště, telefon, fax, e-mail)

Téma bakalářské práce: Současný ekosystém Rich Internet Applications v HTML5

Cíle práce: Cílem práce je seznámit čtenáře se současnou podobou a trendy v ekosystému frameworků, knihoven a nástrojů používaných pro tvorbu tzv. rich internet applications (RIA). Student v rámci práce zvolí vhodnou sadu technologií (technology stack) a problematiku prakticky demonstruje na referenční aplikaci.
Dílčí cíle práce jsou:

1. V teoretické části vymezit pojem RIA, popsat přístupy používané historicky a současné trendy.
2. V teoretické části dále popsat „umožňující technologie" současných trendů – zejména HTML5 a jeho komponenty jako local storage; jazyky a standardy jako JavaScript, EcmaScript, TypeScript; rozšíření protokolu HTTP a nástroje na správu závislostí a sestavení ve světě JavaScriptu.
3. V rešeršní části stručně porovnat alternativní RIA frameworky a popsat podmínky, pro něž jsou vhodné.
4. Na rešeršní část navázat volbou sady technologií, s jejíž pomocí bude navržena a vytvořena referenční aplikace demonstrující principy popsané v teoretické a rešeršní části.
5. Vyhodnotit rozdíly v tomto přístupu k tvorbě aplikací ve srovnání s klasickým přístupem reprezentovaným tradičními web MVC frameworky.

Základní doporučená literatura:

[1] BOZZON, Alessandro, Sara COMAI, Piero FRATERNALI a Giovanni Toffetti CARUGHI. Capturing RIA concepts in a web modeling language. DOI: 10.1145/1135777.1135938. ISBN 10.1145/1135777.1135938. Dostupné z: http://portal.acm.org/citation.cfm?doid=1135777.1135938
[2] DUHL, Joshua. Rich Internet Applications [online]. , 1-33 [cit. 2016-11-30]. Dostupné z: https://www.adobe.com/platform/whitepapers/idc_impact_of_rias.pdf

Financování práce: .....

Vedoucí práce: Martin Čížek podpis: .....

U externích vedoucích fakultní garant práce: .....podpis: .....

Garant oboru bak. studia, pokud je obor zajišťován jinou katedrou/ústavem, než ze které je školitel (nepožaduje se u oboru biologie): .....podpis: .....

Vedoucí katedry/ústavu, kde bude práce vypracovávána: .....podpis: .....

Případný souhlas vedoucího ústavu AV: .....podpis: .....

V Českých Budějovicích dne ...6.1.2017.....Podpis studenta: .....

Bibliografické údaje

Vacek J., 2017: Současný ekosystém Rich Internet Applications v HTML5 [Current Rich Internet Applications ecosystem in HTML5 Bc. Thesis, in Czech.] – 83 p., Faculty of Science, The University of South Bohemia, České Budějovice, Czech Republic.

Anotace

Tato bakalářská práce se zabývá specifickým druhem webových aplikací zvaných Rich Internet Applications (RIA). V první části práce je vymezen pojem RIA a popsány technologie, které se při tvorbě RIA aplikací využívají. V rešeršní části práce jsou porovnány alternativní frameworky. Poté je zvolena vhodná sada technologií, s jejíž pomocí je vytvořena aplikace demonstrující přístup z předchozích částí. Na závěr jsou vyhodnoceny rozdíly v tomto přístupu k tvorbě aplikací ve srovnání s klasickým přístupem reprezentovaným tradičními web MVC farmeworky.

Klíčová slova: Rich Internet Applications, RIA, MVC, TypeScript, JavaScript, Angular, React, HTML5.

Annotation

This bachelor thesis deals with specific type of web applications called Rich Internet Applications (RIA). In first part of thesis, definition of RIA is created and technologies that are used during RIA application development, are described. After that technology stack is chosen and used to create application, which demonstrates approach from previous parts of thesis. In conclusion, differences between selected approach and classical approach, represented by traditional web MVC framework, are evaluated.

Key words: Rich Internet Applications, RIA, MVC, TypeScript, JavaScript, Angular, React, HTML5.

Prohlášení

Prohlašuji, že svoji bakalářskou práci jsem vypracoval samostatně pouze s použitím pramenů a literatury uvedených v seznamu citované literatury. Prohlašuji, že v souladu s § 47b zákona č. 111/1998 Sb. v platném znění souhlasím se zveřejněním své bakalářské práce, a to v nezkrácené podobě elektronickou cestou ve veřejně přístupné části databáze STAG provozované Jihočeskou univerzitou v Českých Budějovicích na jejích internetových stránkách, a to se zachováním mého autorského práva k odevzdanému textu této kvalifikační práce. Souhlasím dále s tím, aby stejnou elektronickou cestou byly v souladu s uvedeným ustanovením zákona č. 111/1998 Sb. zveřejněny posudky školitele a oponentů práce i záznam o průběhu a výsledku obhajoby kvalifikační práce. Rovněž souhlasím s porovnáním textu mé kvalifikační práce s databází kvalifikačních prací Theses.cz provozovanou Národním registrem vysokoškolských kvalifikačních prací a systémem na odhalování plagiátů.

V Českých Budějovicích dne _____ _____

podpis

Poděkování

Děkuji především vedoucímu absolventské práce Ing. Martinu Čížkovi, MBA za trpělivé vedení, cenné rady a čas, který věnoval mé práci. Dále děkuji tvůrcům LaTeX 2$_\varepsilon$[1] za vytvoření systému, který mi značně ulehčil samotné sepsání práce. Děkuji mým blízkým za podporu během studia.

[1] LaTeX 2$_\varepsilon$ je rozšíření systému LaTeX což je kolekce maker pro TeX. TeX je ochranná známka American Mathematical Society.

Obsah

Seznam použitých zkratek

Zkratka	Význam
CSS	Cascading Style Sheets
DOM	Document Object Model
HTML	Hyper Text Markup Language
HTTP	Hyper Text Transfer Protocol
MVC	Model-view-controller
AJAX	Asynchronous JavaScript and XML
NPM	Node package mannager
GUI	Graphical User Interface
XML	Extensible Markup Language
JSON	JavaScript Object Notation
API	Application Programming Interface
REST	Representational State Transfer
POM	Project Object Model
URL	Uniform Resource Locator
Sass	Syntactically Awesome Style Sheets

Kapitola 1

Úvod

V dnešní době je možné najít na internetu čím dál více webových aplikací, jejichž vzhled a chování připomíná desktopové aplikace. Tyto webové aplikace jsou výsledkem dlouhé doby vývoje a standardizace technologií, které značně rozšířily možnosti využití internetu. Internet byl původně navržen pro pouhý přenos dokumetů. Technologie, na kterých byl internet postaven, neumožňovaly vytvářet aplikace jež dnes nazýváme Rich Internet Applications (RIA). To se v posledních dvaceti letech změnilo, byly představeny technologie jako CSS, DOM, JavaScript. Tyto a další technologie byly postupem času standardizovány a implementovány ve většině moderních prohlížečů. To umožňuje dnešním vývojářům vytvářet bohaté, uživatelsky atraktivní aplikace, které je možné spouštět na celé řadě zařízení. Předkládaná bakalářská práce se zabývá právě tímto typem aplikací.

Rich Internet Applications jsem si jako téma své bakalářské vybral z důvodu mého zájmu o vývoj webových aplikací, které využívají moderní technologie a poskytují uživatelům pokročilé funkce a zajímavý vzhled.

Nejprve je nutné přesně vymezit termín RIA, je tedy provedena analýza definic dostupných v odborné literatuře. Na základě této analýzy je vymezen pojem RIA. Poté je čtenář seznámen s některými technologiemi historických přístupů k vývoji RIA. Dále se práce věnuje jazykům a standardům HTML5, JavaScript, EcmaScript, TypeScript a nástrojům jako je NPM, Bower, Gulp, Grunt. V další části jsou srovnány frameworky AngularJS a React. Poté je, pomocí zvolené sada technologií, navržena demonstrační aplikace. Na závěr práce jsou vyhodnoceny rozdíly v přístupu, který byl použit při návrhu aplikace ve srovnání s přístupem tradičních web MVC frameworků.

Cílem práce je seznámit čtenáře se současnou podobou a trendy v ekosystému frameworků, knihoven a nástrojů používaných pro tvorbu tzv. Rich Internet Applications (RIA). V rámci práce bude zvolena vhodná sada technologií (technology stack) s jejíž pomocí bude vytvořena demonstrační aplikace.

Dílčí cíle práce jsou:

- V teoretické části vymezit pojem RIA, popsat přístupy používané historicky a současné trendy.
- V teoretické části dále popsat „umožňující technologie“ současných trendů – zejména HTML5 a jeho komponenty jako local storage, jazyky a standardy jako JavaScript, EcmaScript, TypeScript, rozšíření protokolu HTTP a nástroje na správu závislostí a sestavení ve světě JavaScriptu.
- V rešeršní části stručně porovnat alternativní frameworky a popsat vhodné podmínky, v nichž nacházejí uplatnění.
- Na rešeršní část navázat volbou sady technologií, s jejíž pomocí bude navržena a vytvořena aplikace demonstrující principy popsané v teoretické a rešeršní části.
- Vyhodnotit rozdíly v tomto přístupu k tvorbě aplikací ve srovnání s klasickým přístupem reprezentovaný tradičními web MVC frameworky.

Struktura této bakalářské práce je následující. V kapitole 2 je vymezen pojem RIA, popsány některé historicky používané technologie. V této kapitole jsou dále popsány technologie současných trendů. V kapitole 3 jsou stručně představeny a porovnány alternativní frameworky AngularJS a React a dále popsány vhodné podmínky, v nichž nacházejí uplatnění. Na rešeršní část navazuje praktická část práce. V kapitole 4 je zvolena sada technologií, s jejíž pomocí je navržena a vytvořena aplikace demonstrující principy popsané v teoretické a rešeršní části. Dále je v této kapitole postupně popsán návrh a vývoj demonstrační aplikace. V kapitole 5 jsou vyhodnoceny rozdíly v popsaném přístupu k tvorbě aplikací ve srovnání s klasickým přístupem reprezentovaný tradičními web MVC frameworky. V závěru práce, tedy kapitole 6 jsou vyhodnoceny cíle práce, naznačen další možný rozvoj práce a nastíněny možnosti dalšího zkoumání.

Kapitola 2

Rich Internet Applications

První část této kapitoly se věnuje analýze dostupných definic a vymezení termínu Rich Internet Applications. V současné době existuje na internetu a v odborných publikacích velké množství definic, ale spousta z nich pouze nepřesně porovnává RIA aplikace s desktopovými aplikacemi. Déle jsou stručně popsány některé historicky používané technologie k tvorbě Rich Internet Applications. V poslední části této kapitoly je čtenář seznámen s technologiemi, které se využívají při vývoji Rich Internet Applications.

2.1 Co to jsou Rich Internet Applications?

Paul a Harvey Deitel například definují RIA aplikace:

Rich Internet Applications (RIA) jsou webové aplikace nabízející responzivnost, „rich" vlastnosti a funkcionalitu desktopových aplikací. Předchozí internetové aplikace podporovaly pouze základní HTML grafické uživatelské rozhraní (GUI). Přestože dokázaly obsloužit jednoduché funkce, neměli vzhled a chování desktopových aplkací. [..]RIA aplikace jsou výsledkem dnešních pokročilejších technologií, které umožňují vyšší responzivnost a pokročilejší GUI. Přeloženo z [4]

Příkladem vlastností deskopových aplikací může být užití klávesových zkratek, využití lokálního výpočetního výkonu a okamžitá odezva při interakci uživatele s aplikací bez nutnosti načítání celé stránky. Definice od Santiago Meliá a IEEE jsou velmi podobné, pouze porovnávají RIA aplikace s původními webovými aplikacemi.

Původní webové aplikace trpěly značným omezením v použitelnosti a interaktivitě jejich uživatelských rozhraní. K překonání těchto omezení vznikl nový typ webových aplikací zvaný Rich Internet Application (RIA). RIA aplikace poskytují bohatší a efektivnejší grafické komponenty podobné deskopovým aplikacím. Přeloženo z [1]

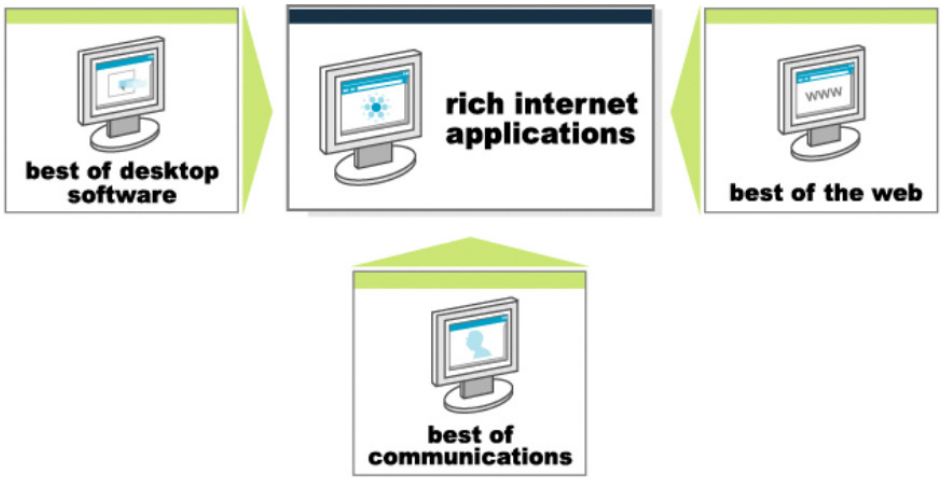
RIA aplikace kombinují snadnou distribuci webových aplikací s interaktivitou uživatelského prostředí a výpočetní silou desktopových aplikací. Výsledná kombinace vylepšuje všechny prvky webové aplikace. Přeloženo z [3]

Alessandro Bozzon zdůrazňuje roli klienta v klient-server architektuře a popisuje jeho chování.

RIA aplikace jsou variantou webových aplikací. Poskytují sofistikované rozhraní pro reprezentaci komplexních procesů a dat. Minimalizují přenos dat mezi klientem a severem a přesouvají prezentační vrstvu ze serveru na klienta. Typicky je RIA aplikace načtena klientem spolu s počátečnimi daty. Klient zajišťuje renderování dat, zpracování událostí a komunikaci ze serverem pokud uživatel požaduje další informace nebo odesílá data. Přeloženo z [2]

Joshua Duhl ve své definici a obrázku(2.1) poukazuje na důležitost komunikace. Zde je klíčová asynchroní komunikace - klient nemusí, při každé změně dat, znovu načítat celou stránku. Význam asynchroní komunikace zdůrazňuje i Marianne Bush [10].

Macromedia definuje RIA aplikace jako kombinaci funkcionality uživatelského prostředí desktopové aplikace s velkým dosahem, nenáročným nasazením webové aplikace a tím nejlepším z interaktivní, mutimediální komunikace. Ve výsledku aplikace uživateli poskytuje intuitivnější, více responzivní zážitek. [..] To nejlepší z komunikace znamená, že klient v RIA aplikacích je schopen více než renedrování stránek. Je schopen provádět výpočty, asynchroně odesílat a získávat data na pozadí, překreslovat části obrazovky. To vše nazávisle na severu ke kterému je připojen. Přeloženo z [5]



Obrázek 2.1: RIA aplikace kombinují to nejlepší z desktopových, webových aplikací a komunikace.Obrázek byl převzat z [5]

Na základě výše uvedeného byl vytvořen vlastní popis pojmu RIA, který podle názoru autora této práce, nejlépe shrnuje vlastnosti RIA aplikací a rozdíly oproti tradičním webovým aplikacím.

Rich Internet Applications (RIA) jsou webové aplikace, které kombinují vlastnosti webové, desktopové aplikace a využívají asynchroní komunikaci. Svým vzhledem a chováním (bohaté uživatelské prostředí s rychlou odezvou) připomínají desktopové aplikace. Klient v RIA aplikacích zajišťuje nejenom renderování stránek a komunikaci se severem, ale také zpracování událostí, fitrování a třídění dat. Výměna dat mezi klientem a serverem probíhá asynchroně, klient tedy zůstává responzivní zatímco neustále přepočítává nebo mění části uživatelského prostředí.

2.2 Historicky používané technologie

Následující část popisuje některé, dříve používané technologie k vývoji RIA. Termín Rich Internet Application byl poprvé představen v roce 2002 firmou Macromedia [6]. Práce, ve které byl termín představen, popisuje vlastnosti RIA aplikací a představuje technologii Macromedia Flash MX. Tato technologie vychází z technologií Remote Scripting firmy Microsoft [7] a X Internet firmy Forrester Research z roku 2001 [8]. Cílem těchto technologií bylo zlepšit uživatelský zážitek při práci s webovou aplikací a přiblížit je tak desktopové aplikaci. Firma Macromedia byla koupena společností Adobe a z Flash MX vznik Adobe Flash.

Adobe Flash je platforma pro tvorbu animací, RIA a jednoduchých her, které běží uvnitř prohlížeče nebo přímo na konkrétním zařízení. K přístupu k aplikacím vytvořeným pomocí Adobe Flash uvnitř prohlížeče slouží plug-in Adobe Flash Player. Tento plug-in je podporován většinou hlavních prohlížečů a operačních systémů. Při tvorbě RIA Adobe Flash využívá programovací jazyk ActionScript, který vychází z jazyka JavaScript a vývojové prostředí Adobe Flash Builder. Nevýhodou je nutnost použití správné verze plug-inu pro zobrazení webové stránky, která využívá Adobe Flash. Flash v současné době používá 6.8 % webových stránek [11].

Další technologie, která byla využívána k tvorbě RIA aplikací je technologie Java Appletů. Applety umožňují vytvářet interaktivní webové aplikace, které nebylo možné tvořit pomocí pouhého HTML. Applet je program napsaný v jazyce Java, který běží uvnitř prohlížeče. Uživatel jej spouští z webové stránky, applet je poté stažen do zařízení uživatele a spouštěn uvnitř JVM sandboxu. Applet, narozdíl od klasické Java aplikace, dědí ze třídy `java.applet.Applet` a neobsahuje metodu `main()` [12]. Nevýhodou je pomalá rychlost prvního spuštění appletu a problémy různých verzí Javy pro spouštění těchto appletů.

K tvorbě interaktivních webových aplikací bylo využíváno také technologie ActiveX, která byla představena firmou Microsoft. Obsahuje kolekci předvytvořeného softwaru, který může být implementován uvnitř aplikace nebo webové stránky. Aplikace vytvořené pomocí ActiveX jsou nazývány ActiveX Controls a umožňují vývojářům rozšířit webové stránky o interaktivní obsah. ActiveX Controls jsou podobné Java Appletům, pro jejich spuštění uživatel musí mít nainstalované ActiveX. Na rozdíl od Java Appletů mohou ActiveX Controls přistupovat k místnímu systému souborů a měnit nastavení operačního systému. ActiveX byly podporovány pouze v prohlížeči InternetExporer a jejich používaní představovalo velké bezpečnostní riziko.

Společnost Microsoft vytvořila technologii pro tvorbu RIA nazývanou Microsoft Silverlight. Microsoft Silverlight je aplikační platforma, která umožňuje vytvářet RIA jejichž uživatelské rozhraní je definováno pomocí programovacího jazyka XAML. První verze byla zveřejněna v roce 2007 a vývoj této platformy byl zastaven v roce 2012 [13].

Pro vývoj RIA se také využívá skupina technologií AJAX. AJAX je zkratka pro Asynchronous JavaScript and XML. Tento termín byl poprvé použit v článku Jesse James Garreta [9]. Jedná se o přístup k vývoji webových aplikací, které mění obsah svých stránek bez nutnosti jejich kompletního znovunačítání. To je možné díky asynchronímu zpracování stránky. AJAX zahrnuje následující technologie:

- HTML a CSS pro prezentaci dat uživateli
- XMLHttpRequest pro asynchroní výměnu dat
- XML, JSON pro výměnu a manipulaci s daty
- DOM pro dynamické zobrazování stránky
- JavaScript svazující vše dohromady

Interakce uživatele s klasickou webovou aplikací probíhá následovně. Uživatel provede nějakou akci, například potvrzení formuláře, tím je na server odeslána HTTP žádost. Server tuto žádost zpracuje a odesílá kompletní HTML dokument. Uživatel tedy musí čekat než proběhne tato výměna dat, tento problém řeší AJAX. AJAX představuje prostředníka mezi serverem a klientem a tím odstraňuje start-stop povahu interakce uživatele se stránkou. Zdá se, že přidání prostředníka mezi server a klienta se aplikace stane méně responzivní, ale opak je pravdou [9]. AJAX umožňuje plynulou interakci uživatele s aplikací, nezávisle na komunikaci aplikace se serverem. V aplikaci využívající AJAX je na server odesílán objekt XMLHttpRequest vytvořený pomocí jazyka JavaScript. Server jej zpracuje a odesílá data ve formátu XML, případně jiném fromátu (JSON, plain text). Klientská aplikace zpracuje, pomocí jazyka JavaScript, odpověď serveru a překreslí, s využitím DOM, stránku. Díky tomu může uživatel používat aplikaci zatímco klient na pozadí komunikuje se serverem. Pokud klient obdrží nová data vykreslí se pouze ta část stránky, která souvisí s obdrženými daty.

2.3 Současné technologie

Následující část této kapitoly popisuje technologie, které se využívají při vývoji RIA. Jedná se především o komponenty HTML5 a jazyk JavaScript, jeho standardy a rozšíření. Déle jsou popsány nástroje na správu závislostí a sestavení ve světě JavaScriptu.

2.3.1 HTML5

HTML5 je další generací HTML, nahrazuje HTML 4.01, XHTML 1.0, XHTML 1.1. HTML5 poskytuje nové funkce, také standardizuje mnoho, již známých, funkcí. Tato verze je prvním pokusem formálně zdokumentovat mnoho „de facto“ standardů, které webové prohlížeče podporují již roky [15]. HTML5 představuje velké množství technologií, včetně:

- Nové HTML elementy
- API pro Geolokaci
- API pro Drag-and-drop
- API pro lokální data
- Forms 2.0
- Podpora videa a audia
- SVG a Canvas grafika
- CSS3
- 2D a 3D animace
- JavaScript 2.0

V následující části budou popsány vybrané funkce HTML5.

2.3.1.1 Nové HTML elementy

HTML5 obsahuje nové elementy, které umožňují vývojářům lépe strukturovat kód. Tyto nové elementy mají následující funkce:

- Rozdělení obsahu stránky do bloků
- Práce s médii
- Práce s formuláři

Rozdělení obsahu stránky do bloků

Rozdělení stránky do bloků je většinou řešeno pomocí `div` elementu opatřeného jménem třídy pro případnou stylizaci. Element `div` nemá žádnou souvislost s částí stránky, kterou zpracovává. Výsledný kód tedy není příliš přehledný. Namísto `div` elementů pro každý element struktury, nabízí HTML5 významově smysluplnější elementy. Obvyklé části stránky jako je záhlaví nebo menu dostaly vlastní elementy. Díky tomu se kód stránky stává přehlednější [18]. To, mimo přehlednější struktury stránky, přináší i další výhodu - internetové vyhledávače dokáží lépe zpracovat strukturu stránky a přesněji ohodnotit její obsah. Nové HTML5 tagy tedy rozdělují stránku do logických bloků jako je záhlaví, zápatí. Mezi tyto elementy patří:

- `header`
- `section`
- `article`
- `aside`
- `footer`
- `nav`

Práce s médii

HTML5 zavádí elementy `audio` pro vkládání zvukových stop a `video` pro vkládání videa. Na příkladu 2.1 je zobrazen kód pro vložení videa. Atributy `height` a `width` určují velikost rámce, atribut `autoplay` spouští přehrávání videa ihned po jeho načtení.

Listing 2.1: HTML5 vložení videa

```
<video width="320" height="240" autoplay >
  <source src="movie.mp4" type="video/mp4">
  <source src="movie.ogg" type="video/ogg">
  Your browser does not support the video tag.
</video>
```

Element `audio` funguje velmi podobně.

Práce s formuláři

Jednou z hlavních změn v HTML5 formulářích je rozšíření elementu `input` novými atributy. Příkladem může být atribut range, který vytvoří z daného elementu posuvník pro výběr čísla. HTML5 disponuje atributy pro výběr barvy, URL adresy, telefoního čísla, data atd. Další změnou je přidání atributů, kteří déle upravují vlastnosti `input` elementu. Příkladem může být například atribut `autofocus`, ten označuje položku formuláře, která bude při načtení formuláře aktivní.

2.3.1.2 API pro lokální data

Local storage

Local storage umožňuje webové aplikaci ukládat data lokálně v prohlížeči uživatele. Před příchodem HTML5 musela být aplikační data ukládana v cookies a při všech HTTP požadavcích odesílána na server, což limitovalo jejich velikost a zvyšovalo objem přenášených dat. Další nevýhodou cookies je jejich poměrně malá kapacita - 4kb pro všechny cookies na dané doméně [14]. Narozdíl od cookies má Local storage kapacitu větší - 5MB [17] a nikdy není přenášena na server. Local storage má využití v případě, kdy je potřeba uživateli uložit data, která není potřeba přenášet na server.

Příkladem může být průběžné ukládání obsahu formulářů, které se díky lokálnímu úložišti může provádět velmi často. V případě nějakého problému tedy uživatel přijde pouze o pár znaků.
Použití Local storage je snadné, na příkladu 2.2 je zobrazen kód pro vložení hodnoty.

Listing 2.2: Vložení hodnoty do Local storage

```
localStorage.setItem("key", "value");
```

Na příkladu 2.3 je zobrazen kód pro získání hodnoty. Velmi podobný je i způsob pro odstranění položky, smazaní celé Local storage.

Listing 2.3: Získání hodnoty z Local storage

```
 var value = localStorage.getItem("key");
```

V některých případech je vhodné ukládat do Local storage celé JSON objekty. Local storage podporuje ukládání pouze řetězců [16], je tedy nutné JSON objekty převést na řetězce.

Listing 2.4: Vložení JSON objektu do Local storage

```
var object = {
  "key" : "value",
  "key 2" : "value 2"
}
localStorage.setItem(
  "object-name", JSON.stringify(object)
);
```

Při převodu JSON objektu na řetězec lze využít funkci JSON.parse()

Listing 2.5: Získání JSON objektu z Local storage

```
var data = localStorage.getItem("object-name");
if (data) {
  var object = JSON.parse(data);
}
```

Session storage

Session storage funguje podobně jako Local storage, líší se kapacitou - u Session storage není limitovaná a životnost Session storage je omezena životností session.

2.3.2 Java Script

JavaScript je objektově orientovaný skriptovací jazyk, který umožňuje umístit kód do webové stránky, tento kód bude tedy proveden na straně klienta. JavaScript využívá 94.5% webových aplikací [20]. JavaScript není odvozen od programovacího jazyka Java, ale mnoho jeho rysů je podobných Javě. Podobnost jmen těchto jazyků je čistě marketingový tah společností Natscape a Sun [19]. JavaScript je nejčastěji používán ve webových prohlížečích, při tomto použití je jádro JavaScriptu často rozšířeno pomocí knihoven, která umožňují interagovat s uživatelem, měnit DOM(Document Object Model). Příkladem JavaScript knohovny může být JQuery. JQuery je jedna z nejpoužívanějších JavaScript knihoven [23]. JQuery pomáhá řešit nekompatibilitu mezi prohlížeči a poskytuje metody pro práci s:

- Manipulace s DOM - datová struktura uložená v prohlížeči, reprezentující strukturu stránky
- Manipulace s CSS
- Zpracování událostí
- Efekty a animace
- AJAX

2.3.3 Ecma Script

EcmaScript je standardizační jméno jazyka JavaScript. EcmaScript je tedy oficiální název standardu a JavaScript je název jedné z jeho implementací. Ve snaze o rozšiřování vlastností jazyka a udržení kompatibility standardu s ostatními standardizačními organizacemi je standard postupně měněn. Následuje stručný popis jednotlivých verzí a změn ve standardu:

EcmaScript 1
První verze standardu, vydaná v červnu 1997 [26].

EcmaScript 2
Tato verze byla publikována v roce 1998 [27] kdy byl jazyk standardizován ISO/IEC. Změny mezi první a druhou verzí jsou minimální [30].

EcmaScript 3
V roce 1999 [28] byl standard rozšířen o mnoho vlastností mezi nejvýznamnější patří [30]:

- Regulární výrazy
- Nové kontrolní výrazy
- Zpracování výjimek

EcmaScript 4
Prestože vývoji čtvrté verze věnovala TC39 (skupina pro vývoj standardu EcmaScript [31]) mnoho času, nebyla čtvrtá verze nikdy publikována.

EcmaScript 5
Pátá verze z roku 2009 [29] přidala podporu pro vlastnosti, které se objevily od vydání třetí verze, mezi nejvýznamější patří [30]:

- Iterační finkce (map, reduce, filter, forEach)
- Podpora JSON
- Getter a Setter funkce

EcmaScript 5.1
Stejně jako v případě EcmaScript 2 byla verze 5.1 vydána pro sjednocení standardu EcmaScript se standardem ISO/IEC .

EcmaScript 6
Tato verze z roku 2015 přinesla několik zásadních změň [30]:

- Moduly
- Deklarace proměných v rámci bloku (let)
- Třídy
- REST parametry
- Symboly

Tato verze ještě není plně podporována webovými prohlížeči proto existují nástroje jako například Babel.js [32], které konvertují kód EcmaScript 6 na starší EcmaScript kód.

2.3.4 TypeScript

Při vývoji rozsáhlých RIA je možné použít jazyk TypeScript, který byl vytvořen firmou Microsoft v roce 2004 [33]. Jedná se o nadstavbu jazyka JavaScript, která přináší atributy známé z objektově orientovaného programování. Cílem TypeScriptu je usnadnění vývoje rozsáhlých JavaScript aplikací. Kód napsaný v TypeScriptu se kompiluje do jazyka JavaScript. TypeScript doplňuje JavaScript o:

- Anotace typů
- Třídy
- Rozhraní
- Genericita
- Moduly

Vzhledem k tomu, že TypeScript byl vytvořen firmou Microsoft má velmi dobrou podporu ve vývojářském nástroji Visual Studio, ale i další vývojářské nástroje přidali podporu pro TypeScript. Mezi tyto nástroje patří: Sublime Text, Atom, Eclipse, Emacs, WebStorm, Vim [34]. TypeScript kompilátor lze, do příkazové řádky, nainstalovat pomocí NPM příkazem `npm install -g typescript`. Příkaz `tsc název_souboru.ts` přeloží soubor `název_souboru.ts` napsaný v jazyce TypeScript do souboru `název_souboru.js`, který obsahuje pouze jazyk JavaScript. Následuje stručný popis vybraných vlastností jazyka TypeScript.

Anotace typů

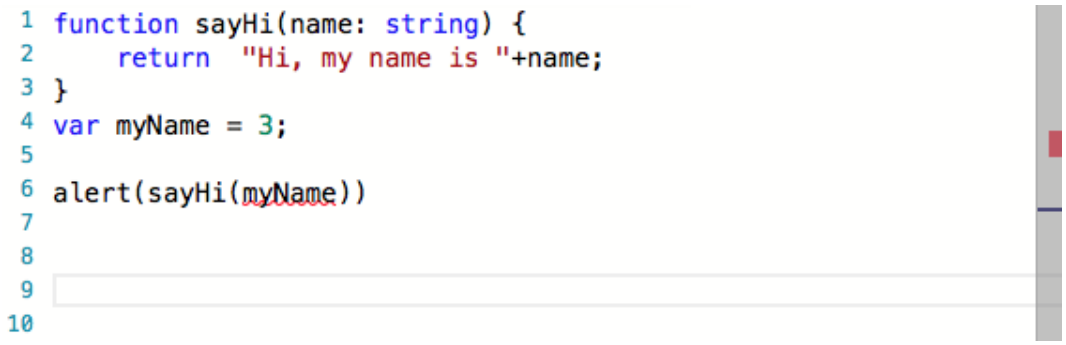
Anotace typů v TypeScriptu umožňují zaznamenat zamýšlený kontrakt funkce nebo proměnné. Na příkladu je zobrazeno použití anotace typů, kód je běžný JavaScript lišící se pouze v hlavičce funkce (`name: string`). V tomto případě byla vytvořena funkce `sayHi` s jedním parametrem typu String.

Listing 2.6: Anotace typu v jazyce TypeScript

```
function sayHi(name: string) {
    return  "Hi, my name is "+name;
}
var myName = "franta";

alert(sayHi(myName))
```

Pokud programátor do proměnné `myName` přiřadí číselnou hodnotu vývojové prostředí jej upozorní, že se snaží do proměnné typu String přiřadit číslo (viz. obrázek 2.2). I přes toto upozornění proběhne kompilace bez problémů, protože anotace typů se při kompilaci do JavaScriptu ztrácí. Upozornění tedy programátora pouze informuje o tom, že vytvořený kód pravděpodobně nebude fungovat tak jak zamýšlel.



Obrázek 2.2: Statická typová kontrola

Rozhraní

Rozhraní v TypeScriptu může být použito jako abstraktní typ, který poté implementují konkrétní třídy, ale také k definování struktury uvnitř programu [33]. Na příkladu 2.7 rozhraní popisuje objekt, který má atributy `firstName` a `lastName` typu String.

Listing 2.7: Rozhraní v jazyce TypeScript

```
interface Person {
    firstName: string;
    lastName: string;
}
function sayHi(person: Person) {
    return "Hi, my name is " + person.firstName + " " + person.
       lastName;
}
var user = { firstName: "Franta", lastName: "Lala" };
document.body.innerHTML = sayHi(user);
```

V jazyce TypeScript jsou dva typy kompatibilní pokud je kompatibilní jejich vnitřní skrutktura [34], to umožňuje implementovat rozhraní bez nutnosti použít klíčové slovo `implements`.

Třídy

Na příkladu 2.8 byla vytvořena třída `Person`, která implementuje rozhraní `ifPerson`, obsahuje konstruktor, několik proměnných a metodu `sayHi()`.

Listing 2.8: Třídy v jazyce TypeScript

```
interface ifPerson {
    firstName: string;
    lastName: string;
    sayHi():string;
}
class Person implements ifPerson{
    constructor(public firstName: string,public lastName: string)
    { }
    sayHi() {
        return "Hi, my name is " + this.firstName + " " + this.
            lastName;
    }
}
var user = new Person("franta", "lala");
document.body.innerHTML = user.sayHi();
```

V příkladu nejsou parametry konstruktoru namapovány na proměnné třídy, to je další vlastnost jazyka TypeScript. Pokud je před parametr konstruktoru vložen modifikátor přístupu (například `public`), tak bude tento parametr automaticky namapován na proměnnou třídy. Na tyto parametry konstruktoru lze odkazovat stejně jako by byly deklarovány jako proměnné třídy, například `this.firstName`. To umožňuje zkrátit deklaraci třídy.

2.3.5 Nástroje na správu závislostí

Rich Internet Applicatons obvykle obsahují velké množství závislostí, je proto vhodné využít nástroj pro správu závislostí. Nástroje na správu závislostí, neboli package managers, slouží k instalaci a správě zavislostí v rámci aplikace. Tento nástroj umožňuje vývojářům sdílet kód, který vytvořili k řešení specifického problému. Tyto části znovupoužitelného kódu se nazývají balíčky nebo moduly. Balíček je obyčejná složka s jedním nebo více soubory, obsahuje také soubor popisující balíček. Typická aplikce, například webová stránka, závisí na desítkách nebo stovkách balíčků.

Tyto balíčky jsou často malé. Základní myšlenkou je vytvoření malých stavební bloků (balíčků), které řeší jeden problém a řeší jej dobře [21]. To umožňuje sestavovat větší, komplexnější řešení sestavené z těchto malých, sdílených stavebních bloků.

2.3.5.1 NPM

NPM je zkratka pro Node Package Manager. Jak jméno naznačuje, NPM je nástroj na správu závislostí pro Node.js. To znamená, že NPM potřebuje pro běh Node.js. NPM je součástí instalace Node.js. NPM používá soubor zvaný `package.json`. Tento soubor obsahuje informace o aplikaci, například:

Listing 2.9: Soubor package.json

```
{
  "name": "my_package",
  "version": "1.0.0",
  "dependencies": {
    "my_dep": "^1.0.0"
  },
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "jakub_vacek",
  "license": "ISC",
  "repository": {
    "type": "git",
    "url": "https://github.com/jakub_vacek/my_package.git"
  },
  "bugs": {
    "url": "https://github.com/jakub_vacek/my_package/issues"
  },
  "homepage": "https://github.com/jakub_vacek/my_package"
}
```

K vytvoření tohoto souboru slouží příkaz `npm init`. Tento příkat spustí jednoduchý dotazník, který vede k vytvoření `package.json` v adresáři kde byl příkaz spuštěn. Stahování jednotlivých balíčků probíhá pomocí příkazu `npm install jméno_balíčku`.

Tento příkaz vytvoří složku `node modules` (pokud již neexistuje) a stáhne balíček do této složky. Příkaz `npm install jméno_balíčku --save` vytvoří složku, stáhne balíček a přidá balíček do `"dependencies"`. Instalace závislostí, uvedených v `"dependencies"` případně `"devDependencies"`, na jiném počítači (například při předávání projektu mezi členy týmu) probíhá pomocí příkazu `npm install`.

2.3.5.2 Bower

Bower je nástroj na správu závislostí vytvořený společností Twitter, slouží ke spravování komponent obsahující HTML, CSS, JavaScript, fonty a dokonce obrázky [22]. Je využíván ke správě závislostí na straně klienta. Bower je Node.js modul, instaluje se pomocí příkazu:

```
npm install bower
```

Příkaz `bower install bootstrap` vytvoří (pokud již neexistuje) složku `bower_components` a nainstaluje do ní balíček `bootstrap`. Příkaz `bower init` vytvoří soubor `bower.json` obsahující informace o aplikaci, případně webové stránce. Další závislosti je možné přídávat příkazem `bower install jméno_balíčku--save`. Na příkladu 2.10 je zobrazen obsah souboru `bower.json`:

Listing 2.10: Soubor bower.json

```
{
  "name": "my_package",
  "homepage": "https://github.com/jakub-vacek/my-package",
  "description": "",
  "main": "index.html",
  "license": "MIT",
  "private": true,
  "ignore": [
    "**/.*",
    "bower_components",
    "test",
  ],
  "dependencies": {
    "angular": "^1.5.8"
  }
}
```

2.3.6 Nástroje pro správu sestavení

Rich Internet Applications často obsahují vetší množství JavaScript souborů, které je vhodné minifikovat (odstranění všech nepotřebných znaků ve zdrojovém kódu). Déle RIA často obasahují soubory css preprocesorů. Tyto soubory je nutné přeložit do ccs. K tomu lze využít nástroj pro správu sestavení (task runner), který se používá pro automatické provádění často prováděných úloh jako například minifikace, kompilace, testování. Tato podkapitola se zabývá nástroji Grunt a Gulp.

2.3.6.1 Grunt

Grunt je task runner pro JavaScript, používá příkazovou řádku pro spouštění uživatelem vytvořených úloh, které jsou definované v souboru `gruntfile.js`. Grunt vytvořil Ben Alman v roce 2012 [24], je napsán v Node.js a je distribuován pomocí NPM. Instalace probíhá podobně jako instalace ostaních NPM balíčků, pomocí příkazu:

```
npm install -g grunt-cli
```

Po instalaci `grunt-cli` je v příkazové řádce k dispozici příkaz `grunt`, tento příkaz hledá lokálně nainstalovaný Grunt. Pokud je lokalní instalace nalezena, `grunt.cli` načte `gruntfile.js` a provede ulohy. Každý `gruntfile.js` obsahuje:

- Funkci `module.exports = function(grunt)` uvnitř této funkce je uveden veškerý kód související s Grunt.
- Konfigurace projektu a úloh
- Načtení Grunt pluginů
- Uživatelem definované úlohy

V následující ukázce je zobrazen obsah souboru `gruntfile.js`. Do tohoto souboru je importována konfigurace projektu ze souboru `package.json`, dále je použit plugin `grunt-contrib-uglify` pro konfiguraci úlohy `uglify`, která mimifikuje zdrojové kódy.

Listing 2.11: Příklad souboru gruntfile.js

```
module.exports = function(grunt) {

  // konfigurace projektu.
  grunt.initConfig({
    pkg: grunt.file.readJSON('package.json'),
    uglify: {
      build: {
        src: 'src/<\%= pkg.name \%>.js',
        dest: 'build/<\%= pkg.name \%>.min.js'
      }
    }
  });
  // načetní pluginu který zajišťuje úlohu "uglify"
  grunt.loadNpmTasks('grunt-contrib-uglify');

  // úloha default
  grunt.registerTask('default', ['uglify']);

};
```

Grunt a pluginy s ním související je nutné přidat do sekce `devDependencies` v souboru `package.json`. To lze provést pomocí příkazu `grunt-init`, který vytvoří `package.json` obsahující nutné závislosti. Přidání Gruntu do již existujícího `package.json` lze provést pomocí:

Listing 2.12: Přidání nástroje Grunt do souboru package.json

```
npm install grunt --sev-dev
```

2.3.6.2 Gulp

Gulp je stejně jako Grunt napsán v Node.js a distribuován pomocí NPM. Grunt a Gulp se liší jazykem použitým v konfiguračním souboru (`gruntfile.js` případně `gulpfile.js`). Grunt používá konfigurační soubory, které jsou napsány v jazce podobnému JSON. Gulp používá soubor `gulpfile.js`, tento soubor je napsán v JavaScriptu.

Instalace Gulp probíhá velmi podobně jako instalace Grunt, nejprve je, pomocí příkazu `npm install --global gulp-cli` nainstalován `gulp-cli`, který umožní používat příkaz `gulp` v příkazové řádce. Dále je nutné přidat Gulp a s ním související pluginy do sekce `devDependencies` v souboru `package.json` pomocí příkazu:

Listing 2.13: Instalace nástroje Gulp

```
npm install grunt --sev-dev
```

Déle je nutné vytvořit soubor `gulpfile.js` ve kterém jsou načteny pluginy a definovány úlohy. Následující `gulpfile.js` využívá pluginy `gulp` a `gulp-sass`, dále obsahuje úlohu `styles`, která překládá `sass` soubory do `css`. Tato úloha je spouštěna uvnitř úlohy `default` pomocí příkazu `gulp`. Soubor `gulpfile.js` dále obsahuje úlohu `watch`, která sleduje změny v souborech na zadané cestě a případně spouští úlohu `styles`.

Listing 2.14: Příklad souboru gulpfile.js

```
//načtení pluginů
var gulp = require('gulp')
var sass = require('gulp-sass');
//úloha default
gulp.task('default', ['styles']);
//kompilace sass souborů
gulp.task('styles', function() {
    return gulp.src('src/main/resources/sass/*.scss')
    .pipe(sass())
    .pipe(gulp.dest('src/main/build/resources/css'));
});
//sledování změn sass souborů
gulp.task('watch', ['styles'], function() {
    gulp.watch('src/main/resources/**/*.scss', ['styles']);
});
```

V následující části je čtenář seznámen s rozšířením HTTP protokolu, konkrétně s konceptem Server Push.

2.3.7 HTTP a jeho rozšíření

HTTP/1.1 je bezstavový protokol, každý požadavek je tedy jedinečný a nezávislý. To přináší určité výhody, server si nemusí ukládat informace o sezení. Webové aplikace potřebují informace o stavu, k tomu se často využívají session. To však znamená, že s každým požadavkem se posílají redundantní informace. Komunikaci vždy zahajuje klientská aplikace. To je u některých typů aplikací (aplikace pro real-time komunikaci, hry) problematické. Proto vznikl přístup zvaný Server Push. Při tomto přístupu server odesílá prostředky klientské aplikaci ještě předtím než o ně strana klienta požádá. Mezi standardizovaná řešení patří WebSocket a Server-sent events. Server Push řeší následující techniky:

2.3.7.1 Polling

Velmi jednoduchá technika kdy se klient neustále (po určitém časovém intervalu) dotazuje serveru zda nedošlo ke změně v datech. Tuto techniky vhodné použít pouze v případě, že server získává nová data opakovaně, po určitém časovém úseku. V opačném případě dochází k plýtvání zdroji.

Variace této techniky se nazývá Long Polling. Klient se opakovaně dotazuje serveru zda nedošlo ke změně dat. V případě, že server nemá žádná data k odeslání se odpoveď neodesílá a spojení zůstává otevřené. Spojené zůstává otevřené dokud není třeba odeslat data nebo do vypršení časového limitu. Po vypršení tohoto limitu se ihned otevírá nové spojení.

2.3.7.2 HTTP/2 Server Push

Zařazení konceptu Server Push do specifikace HTTP/2 má za cíl urychlit komunikaci mezi serverem a klientskou aplikací. Moderní webové aplikace (včetně RIA) využívají velké množství závislostí, které je nutné získat od serveru. Tento proces často zabere větší množství času, uživatel tedy musí čekat, což zhoršuje zážitek uživatele. Server Push umožňuje, při správné implementaci, zrychlit načítání webových aplikací.

Příkladem může hypotetická webová stránka. Tato stránka využívá tři prostředky: `index.htlm`, `script.js` a `style.css`. Uživatel se pomocí prohlížeče připojí ke stránce a získá tak od serveru soubor `index.html`. Po obdržení tohoto souboru prohlížeč zjistí, že bude potřebovat také soubory `script.js` a `style.css`. Prohlížeč odešle požadavky pro získání těchto souborů. Prohlížeč tedy při sestavení stránky postupně odhaluje potřebné soubory a zatěžuje síť získáváním těchto souborů.

Při použití Server Push server obsahuje sadu pravidel, které mohou v určitých momentech spustit Server Push. Tímto momentem například může být obdržení požadavku o soubor `index.html`. Server tedy, mimo `index.html` odešle také rámec typu `PUSH PROMISE`. Tím klientské aplikaci oznámí, že bude odesílat soubory `script.js` a `style.css`. Nečeká tedy na žádost od prohlížeče. Klientská aplikace neodesílá žádost o soubory `script.js` a `style.css`, pouze čeká až je server doručí. Specifikace HTTP/2 je stručné popsána níže.

HTTP/2 je revize HTTP protokolu, nejedná se o celkovou změnu protokolu. HTTP metody, stavové kódy jsou stejné jako u HTTP/1.1. Cílem revize je zlepšení výkonu, především snížení prodlevy a zlepšení využití síťových prostředků. HTTP/2 v současné době využívá 12.6 % webových stránek [35]. Mezi hlavní rozdíly mezi HTTP/1.1 a HTTP/2 patří [36]:

- Formát zpráv - HTTP/2 je, na rozdíl od svých předchůdců, binární.
- Komprese hlaviček - Hlavičky u HTTP/2 jsou stejné jako u HTTP/1.1 a jsou komprimovány pomocí algoritmu HPACK [37].
- Multiplexing - HTTP/2 umožňuje odesílat případně přijímat více požadavků a odpovědí v rámci jednoho TCP spojení.
- Server Push - koncept Server Push v rámci HTTP/2 je popsán výše.

2.3.7.3 HTTP streaming

Mezi serverem a klientem je navázáno spojení, které se neuzavírá. Tento přístup využívá HTML element `Iframe`, který vkládá do HTML stránky další HTML stránku. Pokud nastane na serveru určitá událost, například změna dat, je vnitřní HTML stránka naplněna HTML elementem `script`. Tyto skripty jsou poté postupně spouštěny na straně klienta.

2.3.7.4 Server-sent events

Server-sent events je technologie, která se využívá v případech kdy není nutné aby klient reagoval na data získané ze serveru [39]. Na rozdíl od WebSocket Server-sent events využívají standardní HTTP a rozhraní zvané EventSource. Zajímavostí je, že tato technologie neni podporována v prohlížeči Internet Explorer [40]. Příkladem použití může být stahování předpovědi počasí.

2.3.7.5 WebSocket

WebSocket je technologie, která umožňuje obousměrnou komunikaci mezi serverem a klientem [38]. Klientská aplikace vytvoří WebSocket spojení pomocí procesu, který je označovaný jako WebSocket handshake. Na počátku tohoto procesu klient odešle HTTP požadavek s havičkou `Upgrade`. Pokud server podporuje WebSocket protokol odesílá odpoveď opět s `Upgrade` hlavičkou. HTTP protokol je využíván pouze k sestavení WebSocket spojení, další komunikace se HTTP protokol neúčastní.

V RIA se Server Push využívá při tvorbě aplikací, kde je nutné ze serveru odesílat klientem nevýžadaná data. Příkladem mohou být výše zmíněné aplikace pro real-time komunikaci. Při vývoji aplikací, které využívají Server Push je možné použít Node.js [41] společně s technologiemi jako je Express.js [42] nebo Socket.io [43] na straně serveru. Na straně klienta je poté možné využít celou řadu technologií, včetně AngularJS a React, které budou posány dále.

Kapitola 3

Frameworky pro vývoj webových aplikací

Cílem této kapitoly je stručné srovnání frameworků pro vývoj webových aplikací. Konkrétně budou srovnány frameworky AngularJS a React. Nejprve jsou popsány vlastnosti jednotlivých frameworků. Poté jsou na základě popsaných vlastností, frameworky porovnány.

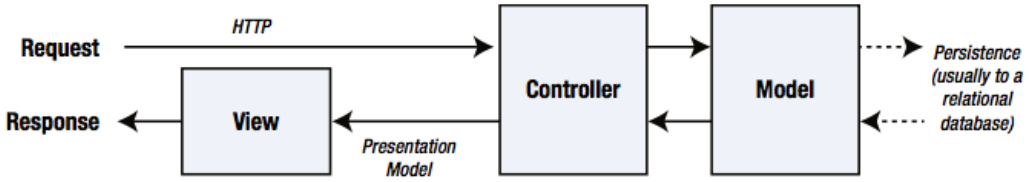
3.1 AngularJS

AngularJS je open-source JavaScript framework, který se zaměřuje na tvorbu dynamických webových aplikací. Tento framework umožňuje rozšířit HTML o značky, které určují chování a obsah daných elementů. Vlastnosti AngularJS jako je data-binding a dependency injection nahrazují mnoho kódu, který by musel tvořit vývojář. Vývojář se tedy může soustředit na aplikační logiku aplikace. AngularJS se používá v prohlížeči, na straně klienta, je tedy možné jej využít s velkým množstvím serverových technologií. V následující části jsou popsány nejdůležitější vlastnosti frameworku AngularJS.

3.1.1 MVC

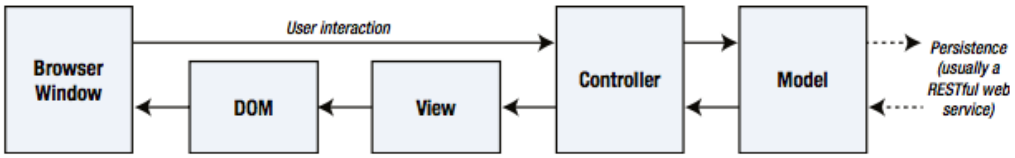
Jedním ze základních konceptů frameworku AngularJS je návrhový vzor MVC. MVC, neboli Model-View-Controller, představuje způsob dělení aplikací na logické částí, odděluje datový model od aplikační a prezentační logiky. Dělení aplikace na logické části usnadňuje vývojářům vývoj, testování i případné rozšíření aplikace. MVC dělí aplikaci na tři modulární části:

- Model obsahuje data se kterými uživatel pracuje. Dále obsahuje logiku těchto dat, tedy funkce pro tvoření a úpravu dat. Model poskytuje metody pro přístup k datům a operacím nad nimy, ale nezobrazuje jakým způsobem jsou data spravována.
- View je uživatelské rozhraní, které vidí uživatel. Generuje se na základě aktuálního modelu.
- Controller spojuje model a view, rozhoduje jakým způsobem budou prezentována data z modelu.



Obrázek 3.1: Imlementace MVC architektury v serverové aplikaci. Obrázek byl převzat z [46]

Na obrázku (3.1) je zobrazena implementace MVC architektury v serverové aplikaci, model je získán z databáze a cílem aplikace je obsluha HTTP požadavků od webového prohlížeče.



Obrázek 3.2: Imlementace MVC architektury v klientské aplikaci. Obrázek byl převzat z [46]

Vzhledem k tomu, že AngularJS běží v prohlížeči je jeho implementace MVC odlišná od implementace MVC v serverové aplikaci. U aplikací na straně klienta (viz. obrázek 3.2) MVC oděluje data, logiku zpracovávající data a HTML elementy, které data zobrazují.

3.1.2 Dependency Injection

Dependency injection je návrhový vzor, který umožňuje definovat závislosti objektu v rámci konfigurace. Tento návrhový vzor umožňuje vývojáři vyhnout se manuálnímu vytvoření instancí závislostí objektu, přenáší tedy zodpovědnost za vytvoření závislostí z objektu na framework. Dependency injection není nový koncept a využívá jej mnoho různých aplikačních frameworků, například populární framework Spring. AngularJS využívá dependency injection k načtení závislostí při tvorbě jednotlivých objektů. Na příkladu níže je zobrazen způsob jakým se v AngularJS definují závislosti.

```
var app = angular.module('App', ['ngRoute','Controllers']);
```

V příkladu jsou, při vytvoření modulu, definovány dvě závislosti. První z nich je `ngRoute`, která slouží ke směrování v rámci AngularJS aplikace. Druhou závislostí je modul `Controllers`, tedy modul obsahující controllery, definovaný vývojářem.

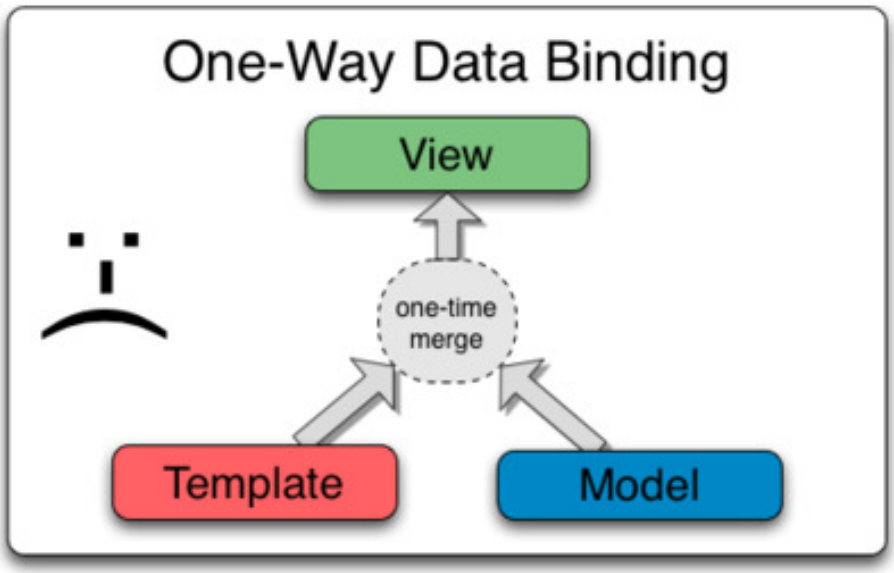
Dependency injection také usnadňuje testování aplikace. Je možné nahradit závislosti „mock“ závislostmi. Například místo závislosti `HttpService`, která komunikuje se serverem lze použít závislost `MockHttpService` pracující pouze s daty v paměnti.

3.1.3 Data Binding

Data binding se v AngularJS využívá k automatické synchronizaci dat mezi modelem a view. Způsob, jakým AngularJS implementuje data binding, umožňuje přistupovat k view jako k aktuální projektci dat modelu. Pokud je změněn model, view zobrazí změnu a naopak.

One way binding

V aplikaci využívající tradiční MVC framework, je uživatelské rozhraní vytvořeno sloučením HTML s lokálními daty. To znamená že, pokud je nutné změnit část uživatelského prostředí musí se znovu vytvořit a poslat celé HTML (viz. obrázek 3.3).



Obrázek 3.3: One way data binding. Obrázek byl převzat z [44]

Pokud vývojář využívá HTML šablony na straně klienta server pouze posílá data (často pomocí JSON nebo XML). Klient, ale stále musí obnovit uživatelské rozhraní a zobrazit tak nově získaná data. To přidává velké množství kódu, který vypadá podobně jako následující příklad:

Listing 3.1: One way binding pomocí JQuery

```
<body>
        Hello, <span id="Name"></span>
    <script>
        var updateNameInUI = function (Name) {
            $('#Name').text(Name);
        };
        <!--Nutno volat při prvním načtení dat -->
        updateNameInUI(user.Name);
        <!-- Nutno volat při každé změně dat-->
        updateNameInUI(updatedName);
    </script>
</body>
```

Metoda `updateNameInUI` pouze předává jméno uživatele v parametru elementu s id `name`. Vývojář musí volat metodu `updateNameInUI` při každé změně dat. Tento kód se týká pouze jednoho políčka. Takových políček má běžná aplikace mnoho. AngularJS tento problém řeší a stejná funkcionalita jako v příkladu výše vypadá, při použití AngularJS, takto:

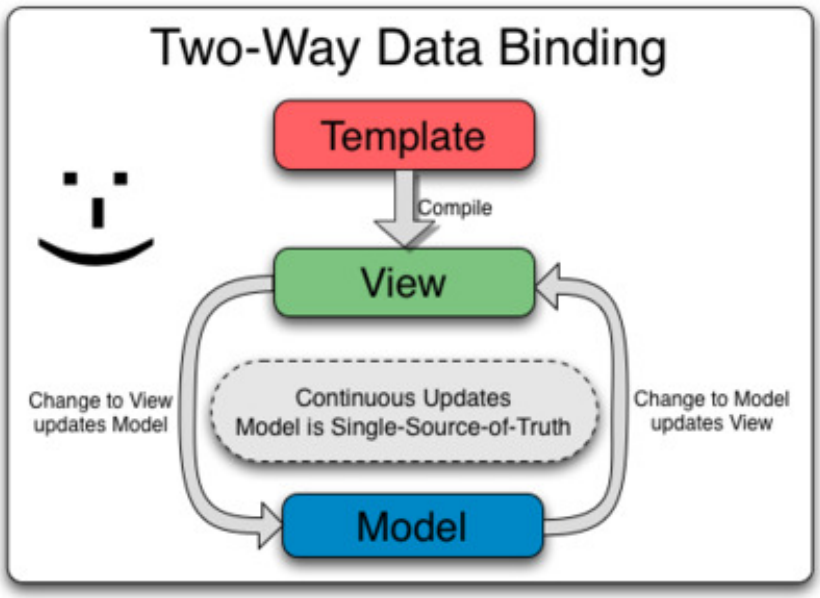
Listing 3.2: One way binding pomocí AngularJS

```
<body>
        Hello, <span>{{Name}}</span>
</body>
```

Vývojař nyní pouze v jazyce JavaScript nastaví hodnotu proměnné `name`. AngularJS sleduje zda se proměnné nezměnila a případně upraví uživatelské rozhraní.

Two way binding

V předchozí části byl čtenář seznámen s one way binding kde jsou data, získáná se serveru, zobrazena v uživatelském rozhraní a když se data v modelu změní, změní se i uživatelské rozhraní. Pokud je nutné získat data od uživatele poté je zpracovat a poslat na server musí vývojář zajistit jak odnovování uživatelského rozhraní tak i modelu. To opět přidává velké množství kódu. AngularJS poskytuje two way binding, který snižuje množství tohoto kódu. (viz. obrázek 3.4).



Obrázek 3.4: Two way data binding. Obrázek byl převzat z [44]

Bez použití AngularJS by řešení při využití klasických formulářů vypadalo takto:

Listing 3.3: Two way binding pomocí JQuery

```
<body>
    <form name="myForm" onsubmit="submitData()">
        <input type="text" id="nameField"/>
        <input type="text" id="emailField"/>
    </form>
    <script>
        <!--  Nastavení dat ve formuláři -->
        function setUserDetails(userDetails) {
            $('#nameField').value(userDetails.Name);
            $('#emailField').value(userDetails.email);
        }
        <!--Získání dat z promuláře -->
        function getUserDetails() {
            return {
                name: $('#nameField').value(),
                email: $('#emailField').value()
            };
        }
        <!-- Potvrzení formuláře-->
        var submitData = function () {
            <!-- Zde pravděpodobně odesíláme data na server,
               data přístupná až voláním getUserDetails() -->
        };
    </script>
</body>
```

Metoda `setUserDetails` opět pouze nastavuje hodnoty ve formuláři. Tuto metodu je nutné volat při potvrzení formuláře (vyčištění obashu formuláře). Metoda `getUserDetails` pouze získává data z formuláře.

Two way binding zajišťuje, že controller a uživatelské rozhraní sdílí stejný model. Pokud se změní data v uživatelském rozhraní nebo v controlleru AngularJS to rozporná a upraví data. Stejná funkcionalita jako v příkladu výše vypadá, při použití AngularJS, takto:

Listing 3.4: Two way binding pomocí AngularJS - HTML

```
<body>
    <form name="myForm" onsubmit="ctrl.submitData()">
        <input type="text" ng-model="user.name"/>
        <input type="text" ng-model="user.email"/>
    </form>
</body>
```

Každý element `input` je navázán na AngularJS model. Po potvrzení formuláře je spuštěna metoda `submitData`, která je definována uvnitř controlleru. JavaScript vypadá takto:

Listing 3.5: Two way binding pomocí AngularJS - JavaScript

```
// Uvnitř controlleru
this.submitData = function() {
 //Zde pravděpodobně odesíláme data na server, data přístupná
    přes this.user
};
```

3.1.4 Direktivy

Direktivy v AngularJS rozšiřují standardní knihovnu HTML tagů, umožňují upravit chování existujících HTML elementů. AngularJS také umožňuje vytvořit vlastní, vývojářem definované direktivy. Díky tomu jsou direktivy jednou z nejdůležitějších vlastností frameworku AngularJS. Direktvy umožňují vytváření znovupoužitelných komponent v rámci aplikace.

Na příkadu níže je zobrazen způsob využití předefinovaných direktiv a také způsob vytvoření elementů pomocí direktiv. Příklad obsahuje HTML kód s direktivou `ng-app` a `ng-controller`. Kód také obsahuje element `user`, který je definovaný pomocí direktiv.

Listing 3.6: Direkivy v AngularJS - HTML

```
<body ng-app="App">
    <div ng-controller="Controller">
        <user></user>
    </div>
</body>
```

Direktiva `na-app` definuje kořenový element applikace a direktiva `ng-controller` definuje controller pro daný element. JavaScript k předchozímu příkladu vypadá takto:

Listing 3.7: Direkivy v AngularJS - JavaScript

```
'use strict';
var app = angular.module('App', []);
app.controller('Controller', ['$scope', function($scope) {
  $scope.user = {
    firstName: 'Franta',
    lastName: 'Lala',
    address: 'Marveniště'
  };
}])
.directive('user', function() {
  return {
    templateUrl: 'user.html'
  };
});
```

V controlleru je definovaný objekt `user` a jeho atributy, déle je zde definována direktiva `user`, která odkazuje na šablonu `user.html`. Na příkladu níže je zobrazen obsah souboru `user.html`

Listing 3.8: Direkivy v AngularJS - HTML šablona

```
First Name: {{user.firstName}} Last Name: {{user.lastName}}
   Address: {{user.address}}
```

3.2 React

React je open-source JavaScript knihovna, která se zaměřuje na tvorbu uživatelských rozhraní. React byl vytvořen společností Facebook v roce 2013 a jeho cílem je tvorba komplexních uživatelských rozhraní, která reperezentují v čase měnící se data [47]. K tomu využívá objektů zvaných komponenta. React se využívá v prohlížeči, na straně klienta, je tedy možné jej využít s velkým množstvím serverových technologií. V následující části jsou popsány nejdůležitejší vlastnosti knihovny React.

3.2.1 Komponenta

Komponenta je základním stavebním prvkem React aplikce. Tyto stavební prvky je možné skládat a tvořit tak složitější uživatelské rozhraní. Do každé komponenty mohou vstupovat data označované jako Properties, a komponenta si může dále udržovat vnitřní data označované jako State. React komponenty mají pouze jednu povinnou metodu a to metodu `render()`. V této metodě vývojář popíše strukturu dané komponenty, včetně jejích závislostí na data. Metoda `render()` vrací HTML element, například `div` jak je zobrazeno na příkladu níže.

Listing 3.9: Komponenty v React

```
class HelloClass extends React.Component {
        render() {
                return <div>Hello world!</div>;
      }
}
```

Výše uvedený příklad používá syntaxi EcmaScript 6 a rozšíření jazyka JavaScript zvané JSX, které popsáno v následující části.

3.2.2 JSX

JSX je rozšíření, které umožňuje používat HTML elementy uvnitř jazyka JavaScript. JSX zápis je poté přeložen na volání JavaScript funkcí knihovny React. Na příkladu níže je zobrazeno srovnání kódu v jazyce JSX a stejného kódu, přeloženého kód do jazyka JavaScript.

Listing 3.10: React JSX

```
React.render( // JSX verze
    <div><h1>Hello world!</h1></div>
);
React.render( // Přeložná JSX verze
    React.createElement('div', null,
        React.createElement('h1', null, 'Hello world!')
    );
);
```

3.2.3 Virtual DOM

Pravděpodobně nejdůležitější částí knihovny React je koncept virtuálního DOM (Document Object Model). Vývojář v React komponentách deklaruje, pomocí JSX, strukturu HTML elementů. Vývojář tedy popisuje jak bude vypadat výsledná stránka na základě příchozích dat. React z tohoto popisu tvoří virtuální DOM, který poté porovnává se skutečným DOM, pokud najde rozdíly tak aktualizuje pouze nutnou část skutečného DOM. Vývojář pouze dodá nová data do jednotlivých komponent a o aktualizaci uživatelského rozhraní se již stará knihovna React. V prohlížeči tak uživatel vždy uvidí aktuální pohled vzhledem k dodaným datům.

3.2.4 Properties

React komponenty lze vnořovat do dalších React komponent. Je tedy nutný, prostředek pro předávání dat mezi jednolivými komponentami. K tomu se využívají objekty označované jako Properties. Na příkladu níže je zobrazeno použití Properties k předání proměnné `name`.

Listing 3.11: React Properties

```
class HelloClass extends React.Component {
  render() {
    return <div>Hello {this.props.name}</div>;
  }
}
ReactDOM.render(<HelloClass name="Jakub" />, mountNode);
```

Properties jsou v knihovně React označovány jako `this.props`, protože to je nejčastější způsob pro přístup k těmto objektům. Properties je skupina JavaScript objektů, které drží React komponenta. Tato skupina objektů se nemění v průběhu celého životního cyklu komponenty.

3.2.5 State

Komponenta může nejenom přijímat vstupní data (pomocí `this.props`), ale také udržovat vnitřní, stavová data. K tomu se využívá objektu State, který je dostupný pomocí `this.state`. Pokud se stavová data komponenty změní, knihovna React opět zavolá metodu `render()` dané komponenty. Na příkladu níže je zobrazeno použití objektu State.

Listing 3.12: React State

```
class Timer extends React.Component {
  constructor(props) {
    super(props);
    this.state = {secondsElapsed: 0};
  }
  tick() {
    this.setState((prevState) => ({
      secondsElapsed: prevState.secondsElapsed + 1
    }));
  }
  componentDidMount() {
    this.interval = setInterval(() => this.tick(), 1000);
  }
  componentWillUnmount() {
    clearInterval(this.interval);
  }
  render() {
    return (
      <div>Seconds Elapsed: {this.state.secondsElapsed}</div>
    );
  }
}
ReactDOM.render(<Timer />, mountNode);
```

3.3 Srovnání frameworků

V předchozích částech této kapitoli byly popsány vlastnosti frameworku AngularJS a knihovny React. Následující tabulka obsahuje srovnání nejvýraznějších vlastností těchto technologií.

Vlastnost	**AngularJS**	**React**
Vývojář	Google	Facebook
Datum zveřejnění	2009	2013
Jazyk	JavaScript, TypeScript	JSX, JavaScript
Architektura	Kompletní MVC framework	Knihovna pro tvorbu UI
Struktura	HTML, JS a CSS	Integruje HTML, JS pomocí JSX
Data binding	Two-way data binding	One-way data binding
HTML šablony	Ano	Ne
Správa závislostí	Ano, dependency injection	Ne
DOM	DOM v prohlížeči	Virtuální DOM
Obtížnost zvládnutí	Komplexní	Po pochopení konceptu snadné
Model	Ano	Ne
View	Ano	Ano
Controller	Ano	Ne

Tabulka 3.1: Srovnání AngularJS a React

AngularJS je kompletní framework, obsahující implementaci návrhového vzoru MVC, dependency injection, moduly pro komunikaci ze serverem a další. Vývojář využívající tento framework má všechny tyto vlastnosti ihned k dispozici, nemusí využívat žádné další knihovny. To odraží i heslo AngularJS „One framework“. React je „pouze“ knihovna, která je zaměřena na jednu činnost, tedy tvorba uživatelských rozhraní. V některých zdrojích je React označován jako V z návrhového vzoru MVC [47]. Poskytuje vývojáři možnost vytvářet uživatelské rozhraní, nikoliv implementaci MVC nebo systém pro práci ze závislostmi. Pokud potřebuje vývojář vytvářet komplexní aplikaci, která komunikuje se serverem, spravuje množství závislostí a zároveň využívá React bude nutné použít další knihovny. Toto lze považovat za nevýhodu. Na druhou stranu, nutnost použít další knihovnu poskytuje vývojáři možnost si zvolit knihovnu podle svých představ.

Dalším rozdílem je čas potřebný k osvojení si dané technologie. Knihovna React není zdaleka tak rozsáhlá a její osvojení tedy nezabere tolik času. Přístup této knihovny k tvorbě uživatelských rozhraní je však, oproti AngularJS, značně rozdílný. Autorovy této práce zabralo pochopení tohoto přístupu mnoho času. Jak bylo zmíněno výše AngularJS je framework, obsahující velké množství dalších konceptů (MVC, dependency injection). Vývojář si musí osvojit i tyto koncepty, což zabere čas. Nicméně AnguarJS má dobře zpracovanou dokumentaci, která tento proces ulehčí.

Následující část popisuje, podle názoru autora této práce, vhodné použití jednotlivých technologií. Pokud vývojář tvoří aplikaci, která vykresluje menší množství dat, komunikuje se serverem a využívá návrhový vzor MVC je vhodné použít framework AngularJS. Vývojáři stačí pouze tento framework, který snadno stáhne a v podstatě ihned může začít pracovat.

Pokud vývojař tvoří rozsáhlý, dlouhotrvající projekt, který vykresluje velké množství dat, spolehá na velké množství knihoven a pracuje na něm více vývojářů je vhodné použít knihovnu React. Vývojář si tak může zvolit sadu knihoven, která bude přesně vyhovovat jeho zvyklostem a zamýšlenému použití. Tato volba, stejně jako stažení závislostí a rozjetí projektu však zabere čas.

Kapitola 4

Praktická část

Tato kapitola se zabývá návrhem a tvorbou aplikace demonstrující principy popsané v teoretické a rešeršní části. Postupně popisuje specifikaci, návrh a tvorbu aplikace. V první části jsou popsány funkce, které bude aplikace poskytovat a její předpokládané použití. Poté jsou zvoleny technologie použité při tvorbě aplikace. Tato část je rozdělena na dvě části, část zabývající se technologiemi na straně klienta a část zabývající se technologiemi na straně serveru. Další části se zabývají návrhem a vývojem aplikace.

4.1 Specifikace aplikace

Aplikace použitá k demostraci principů, popsaných v předchocích kapitolách bude sloužit jako úkolníček s možností sledovat aktivitu uživatele a čas strávený prací na jednotlivých úkolech. Každý uživatel bude mít své projekty, které budou obsahovat jednotlivé úkoly nutné ke splnění projektu. Projekty bude možné přidávat a mazat, projekt bude dále obsahovat informaci o svém dokončení. Tato informace bude dynamicky počítána podle počtu splněných úkolů. Jednotlivé úkoly bude možné mazat, přidávat, upravovat a počítat čas, který uživatel prací na úkolu strávil. Aplikace bude sledovat aktivitu uživatele, a umožňovat uživateli prohlížet si svou aktivitu v rámci celé aplikace. Aplikace bude obsahovat dva typy rolí uživatelů, standardní roli uživatele a roli administrátora. Role administrátora bude uživateli umožňovat přidávat a mazat uživatelské účty. Administrátor bude mít dále přístup k projektům, úkolům a aktivitám ostatních uživatelů. Aplikace bude zpracována jako webová single page aplikace. Data budou uložena v relační databázi.

4.2 Volba sady technologií

Tato část se zabývá volbou vhodné sady technologií, která bude použita k tvorbě demonstrační aplikace. Jednotlivé technologie byly vybrány na základě předpokládaného využití v rámci aplikace s přihlédnutím na zkušenosti autora této práce. Nejprve jsou popsány zvolené technologie na straně serveru a poté na straně klienta.

4.2.1 Volba sady technologií na straně serveru

Základní technologií na straně serveru je framework Spring Boot [48]. Spring Boot je convention over configuration řešení aplikačního frameworku Spring. Convention over configuration je návrhový vzor jehož základní myšlenkou je zjednodušení návrhu aplikace zjednodušením konfigurace projektu - vývojář musí konfigurovat pouze nestandardní části aplikace. Další výhodou je vestavěný server (Tomcat nebo Jetty).

Pro správu sestavení a závislostí na straně serveru je použit Apache Maven. Apache Maven je nástroj pro správu, řízení a automatizaci sestavení aplikací. Maven využívá jednoduchou XML strukturu nazvanou Project Object Model. Tento model popisuje softwarový projekt z pohledu závislostí na externích knihovnách, popisu procesu sestavení a testování.

Pro práci z relační databází je použit ORMLite, což je jednoduchý open-source framework pro objektově relační mapování mezi SQL databází a objekty v jazyce Java. K ukládaní dat v rámci demostrační aplikace je použita open-source relační databáze Apache Derby.

Následuje seznam technologií použitých na straně serveru:

- Spring Boot ve verzi 1.5.1.
- Apache Maven ve verzi 3.3.9
- ORMLite ve verzi 5.0
- Apache Derby ve verzi 10.13.1.1

4.2.2 Volba sady technologií na straně klienta

Základní technologií na straně klienta je framework AngularJS, který je blíže popsán v 3.1. Pro tvorbu animací je použit balíček Angular-animate. Na straně klienta je také použit směrovací framework UI-router, který se používá pro směrování v rámci single page aplikací. UI-router poskytuje směrování pomocí stavů, stav je reprezentován konkrétním URL a pohledem (view). Pohled představuje uživatelské rozhraní stavu, stavy i pohledy je možné vnořovat a tvořit tak stromovou strukturu.

Pro správu závislostí při vývoji, je použit nástroj na správu závislostí NPM, který je blíže popsán v 2.3.5.1. Pro správu závislostí je použit Bower, který je popsán v 2.3.5.2.

K tvorbě uživatelského rozhraní je použit open-source frontend framework UIkit. Aplikace využívá UIkit komponenty jako `uk-flex` pro rozvržení elementů na obrazovce, `uk-icon` pro tlačítka a další. Pro zjednodušení tvorby Css stylů byl použit Css preprocesor. Při volbě preprocesoru byly zvažováni preprocesory Sass a Less. Na základě zkušeností autora této práce byl zvolen Sass.

Poslední technologií je Gulp, který je blíže popsán v 2.3.6.2. Při tvorbě demonstrační aplikace je Gulp použit pro překlad, sloučení a minifikaci scss souborů. K tomu využívá balíčky gulp-concat, gulp-sass a gulp-minify-css.

Následuje seznam technologií použitých na straně klienta:

- NPM
- Bower
- AngularJS ve verzi 1.6.2
- UI-router ve verzi 0.4.2
- UIkit ve verzi 2.27.2
- Sass
- Gulp ve verzi 3.9.1

4.3 Návrh aplikace

Tato část se zabývá návrhem demonstrační aplikace. Jako vývojové prostředí bylo zvoleno Netbeans IDE ve verzi 8.2. Ke správě verzí byl využit Git. Strana serveru a klienta jsou zpracovány jako samostané Netbeans projekty, server je založen na Netbeans Maven projektu a klient na Netbeans HTML5 projektu. Server poskytuje data z relační databáze pomocí REST rozhraní (viz. tabluka 4.1) ve formátu JSON.

URL	**HTTP metoda**	**Obsah dotazu**	**Akce**
/activities	POST	Objekt Activity	Vytvoření nové aktivity
/activities	GET	Id uživatele	Získání kolekce aktivit
/activities	GET	Id objektu	Získání detailu aktivity
/login	GET	údaje uživatele	Přihlášení uživatele
/projects	GET	Id uživatele	Získání kolekce projektů
/projects	POST	Objekt Project	Vytvoření projektu
/projects	GET	Id objektu	Získání detailu projektu
/projects	DELETE	Id objektu	Smazání projektu
/todos	GET	Id projektu	Získání kolekce úkolů
/todos	GET	Id objektu	Získání detailu úkolu
/todos	DELETE	Id objektu	Smazání úkolu
/todos	PUT	Objekt Todo	Úprava úkolu
/todos	POST	Objekt Todo	Vytvoření úkolu
/users	GET	Id uživatele	Získání kolekce uživatelů
/users	POST	Objekt User	Vytvoření uživatele
/users	DELETE	Id uživatele	Smazání uživatele
/users	GET	Id objektu	Získání detailu uživatele

Tabulka 4.1: REST API

4.3.1 Návrh serveru

Jak je uvedeno výše server poskytuje data pomocí REST rozhraní, tomu je podřízena struktura balíčků v projektu, která je popsána níže.

Core
Obsahuje třídu `Application.java`, která obsahuje metodu `main()`. Balíček Core dále obsahuje třídu `StatusException`, která předává informace o vyhozené výjimce jako HTTP status.

Core.Security
Obsahuje třídu `SecurityConfig.java`, která zajišťuje nastavení bezpečnosti v rámci aplikace. Nastavuje především přístup k jednotlivým URL, například uživatel s rolí `USER` nemá přístup k URL `\user`. Třída dále nastavuje cross origin přístup a způsob šifrování hesel.

Mock
Obsahuje třídu `MockDataInitializer`, která v databázi vytvoří testovací data. Déle obsahuje třídy impementující rozhraní z balíčku Service a poskytující testovací data, nahrazují tedy relační databázi při vývoji. Následuje seznam tříd v balíčku Mock.

- `MockActivityServiceImpl.java`
- `MockTodoServiceImpl.java`
- `MockProjectServiceImpl.java`
- `MockUserServiceImpl.java`

Model
Obsahuje třídy jednotlivých entit. Atributy těchto tříd jsou označeny anotacemi pro framework ORMLite. Déle poskytují metody pro nastavení a získání hodnoty jednotlivých atributů, metodu `toString()`, `hashCode()` a `equals()`. Následuje seznam tříd v balíčku Model.

- `Activity.java`
- `Todo.java`
- `Project.java`
- `User.java`

Model.TransferObject

Obsahuje třídy jejichž instance jsou serializovány do formátu JSON a odesílany na stranu klienta. Třídy z tohoto balíčku jsou konstruovány z instancí tříd v balíčku Model a obsahují pouze základní informace o entitách. Tyto informace jsou, na straně klienta, použity v tabulkách. Pokud klient zažádá o detail entity, odesílá se instance třídy doplněná o veškerá data související s entitou. Následuje seznam tříd v balíčku TransferObject.

- `ActivityTo.java`
- `TodoTo.java`
- `ProjectTo.java`
- `UserTo.java`

Service

Obsahuje třídy pro práci s daty k tomu využívají třídy Dao (Data access object) z frameworku ORMLite. Jednotilvé třídy zajištují práci vždy s jedním typem entity a jsou označené anotacemi `@Service`. Jejich instance jsou do tříd z balíčku Controlller vložené pomocí anotace `@Resource`, tedy pomocí dependency injection. Balíček dále obsahuje rozhraní popisující třídy z tohoto balíčku.

- `ActivityService.java`
- `ActivityServiceImpl.java`
- `TodoService.java`
- `TodoServiceImpl.java`
- `ProjectService.java`
- `ProjectServiceImpl.java`
- `UserService.java`
- `UserServiceImpl.java`

Controller
Obsahuje třídy označené anotacemi `@Controller` a `@RequestMapping` pro framework Spring Booot. Tyto třídy zpracovávají žádosti ze strany klienta a odesílají data ve formátu JSON. K získání dat používají instance tříd z balíčku Service. Následuje seznam tříd v balíčku Controller.

- `ActivityController.java`
- `LoginController.java`
- `TodoController.java`
- `ProjectController.java`
- `UserController.java`

4.3.2 Návrh klienta

Klient komunikuje se serverem pomocí rozhraní z tabulky 4.1 a využívá JavaScript objekty zvané `state`, které poskytuje framework UI-router. K jednotlivým stavům jsou přiřazeny direktivy z frameworku AngularJS zvané komponenty. Tomuto přístupu je přispůsobena struktura projektu. Složka `Site Root` obsahuje soubory:

index.html tento soubor je vstupním bodem aplikace, obsahuje definici závislostí celé aplikace.

bower.json tento soubor obsahuje definici závislostí nutných pro běh aplikace, příklad souboru `bower.json` lze nalézt v 2.3.5.2.

gulpfile.js je konfigurační soubor nástroje pro správu sestavení Gulp, obsahuje definici úloh pro práci s Sass soubory. Obsah souboru je velmi podobný 2.3.6.2.

package.json tento soubor obsahuje definici závislostí nutných pro vývoj aplikace, příklad souboru `package.json` lze nalézt v 2.3.5.1.

Core
Obsahuje soubor `App.js` který obsahuje deklaraci modulu `App`

Conroller

Obsahuje soubor `appContoller.js`, který deklaruje controller `appController`. Tento controller obsahuje metody `login()` a `logout()`, zajišťuje tedy přihlášení a odhlášení uživatele.

Component

Složka obsahuje soubory, které definují jednotlivé stavy a s nimi korespondující komponenty. Složka Component dále obsahuje soubory:

- `activitiesComponent.js`
- `homeComponent.js`
- `loginComponent.js`
- `todosComponent.js`
- `projectsComponent.js`
- `usersComponent.js`

Template

Obsahuje HTML šablony pro jednotlivé stavy. Šablony jsou rozděleny do složek podle toho jakého objektu se týkají. Složka Template obsahuje soubory:

- Složku `Activity`
- Složku `Todo`
- Složku `Project`
- Složku `User`
- `header.html`
- `login.html`
- `home.html`

Service
Obsahuje soubory, které přistupují k datům serveru pomocí AngularJS `$http` objektů. Každý soubor obsahuje funkce pro práci s jedním typem objektů, tedy soubor `todoService` obsahuje funkce pro práci s objektem `Todo`. Složka Service obsahuje soubory:

- `activityService.js`
- `todoService.js`
- `projectServicejs`
- `userService.js`

bower_modules
Obsahuje složky závislostí definovaných v souboru `bower.json`

node_modules
Obsahuje složky závislostí definovaných v souboru `package.json`

css
Obsahuje soubor `style.css` do kterého se překládají Sass soubory ze složky sass.

sass
Obsahuje soubory s koncovkou `scss`, které definují styly použité v rámci aplikace.

4.4 Vývoj aplikace

Tato část popisuje samotný vývoj demostrační aplikace. Nejprve je popsán vývoj serveru. Postupně popisuje tvorbu tříd a jejich využití na straně serveru. Déle je nastíněno využití mock objektů při vývoji serveru. Následně je rozebrána tvorba klientské aplikace, tedy tvorba AngularJS komponent, jejich využití ve spoluráci s UI-router stavy a vytvoření HTML šablon pro jednotlivé stavy. Jako poslední je popsán vývoj uživatelského prostředí, rozmístění jednotlivých komponent a využití frameworku UIkit.

4.4.1 Vývoj serveru

Při vývoji serveru byly nejprve přidány potřebné závislosti do souboru `pom.xml`. Poté byly naprogramovány třídy z balíčku Model, tedy objekty reprezentující uživatele, úkol, projekt a aktivitu. Déle byly vytvořeny třídy z balíčku TransferObject, které se odesílají na stranu klienta a obsahují informace o objektu. Poté byly naprogramovány třídy implementující rozhraní z balíčku Service, tyto třídy měly ve svém názvu předponu `Mock` a nahrazovaly relační databázi při vývoji aplikace. Déle byly postupně vytvořeny třídy ze souboru Controller, jejichž metody obsluhují žádosti ze strany klienta. Na příkladu níže je zobrazena metoda `getTodosOfProject`, která na HTTP GET požadavek s parametrem `projectId`, odesílá `ArrayList` objektů `TodoTo` ve formátu JSON.

Listing 4.1: Metoda getTodosOfProject

```
@CrossOrigin(origins = "http://localhost:8383")
@RequestMapping(value = "/todos", method = RequestMethod.GET,
    params="projectId")
public ResponseEntity<ArrayList<TodoTo>> getTodosOProject(
    @RequestParam(value = "projectId") int projectId) {
     LOG.log(Level.INFO, "Geting todos of project with id: " +
        projectId);
     ArrayList<TodoTo> todos;
     try {
         //Get items and map them to transfer objects
         todos = todoService.getTodosOfProject(projectId)
                 .stream()
                 .map(item -> new TodoTo(item,false))
                 .collect(Collectors.toCollection(
                    ArrayList::new));
         return new ResponseEntity<>(todos, HttpStatus.OK);
     } catch (StatusException ex) {
         LOG.log(Level.SEVERE, null, ex);
         return new ResponseEntity<>(ex.status);
     }
}
```

V těle metody je nejprve pomocí `todoService` vytvořen `ArrayList` objektů `Todo`, ten je přemapován na `ArrayList` objektů `TodoTo`. V případě vyhození výjimky `StatusException` metoda vrací objekt `ResponseEntitiy` se statusem výjimky.

Po naprogramování tříd v balíčku Controller bylo přistoupeno k otestování komunikace mezi serverem a klientem. Byl vytvořen jednoduchý HTML5 klient odesílající HTTP požadavky a zobrazující odpovědi serveru ve formátu JSON.

Déle byly atributy tříd `Activity, Todo, User, Project` (tedy třídy z balíčku Model) opatřeny anotacemi pro framework ORMLite. V balíčku Service byly vytvořeny třídy, impementující rozhraní z téhož balíčku. Tyto třídy slouží pro přístup k datům v relační databázi, k tomu využívají ORMLite objektů `Dao`. Následně byla naprogramována třída `MockDataInitializer` v balíčku Mock, který vkládá do databáze data použitá pro testování.

Poté byla, v balíčku Security naprogramována třída `SecurityConfig`. Tato třída zajišťuje přístup k jednotlivým URL, způsob šifrování hesel a způsob autentizace.

4.4.2 Vývoj klienta

Po vytvoření fungujícího serveru bylo přistoupeno k programování klientské aplikace. Nejprve byl, ve složce Core, vytvořen soubor `App.js`, který definuje modul `App`. Tento modul představuje AngularJS aplikaci.

Déle byl vytvořen soubor `appController.js` který zajišťuje přihlášení a odhlášení uživatele. S tvorbou tohoto souboru souvisí naprogramování šablon `login.html`, `home.html` a `header.html`. Šablona `login.html` představuje přihlašovací obrazovku aplikace. Šablona `home.html` obsahuje tři tlačítka, která uživatele zavedou na stav představující tabulku aktivit, projektů a uživatelů (pokud má přihlášený uživatel roli `ADMIN`). Šablona `header.html` představuje horní panel obrazovky aplikace, mění se podle toho v jakém stavu se aplikace nachází, zda je uživatel přihlášen. Následně byl vytvořen soubor `index.html`, který je vstupním bodem klientské aplikace, načítá veškeré závislosti. Na následující ukázce je obsah elementu `body` souboru `index.html`

Listing 4.2: Soubor index.html

```
<body ng-app="App" ng-controller="appController" >
    <div ng-include src="'Template/header.html'"> </div>
    <div ui-view > </div>
</body>
```

Element `body` je rozšířen o direktivu `ng-app`, která definuje AngularJS aplikaci. Déle je rozšířen o direktivu `ng-controller`, která definuje controller pro daný element. Direktiva `ng-include` vlákdá do HTML dokumetu další HTML dokument, v tomto případě soubor `header.html`. Poté byly postupně naprogramovány jednolivé stavy a komponenty. Příkladem je níže zobrazený obsah souboru `activitiesComponent.js`

Listing 4.3: Stav activities

```
App.config(function ($stateProvider) {
    $stateProvider.state('activities', {
        url: '/activities',
        template: '<activities loged-user="$resolve.logedUser"
           activities="$resolve.activities" selected-user="
           $resolve.selectedUser"></activities>',
        component: 'activities',
        params: {
            selectedUser: null,
            logedUser: null
        },
        resolve: {
            activities: function($activityService, $stateParams){
                return $activityService.getActivitiesOfUser(
                   $stateParams.selectedUser)
            },
            selectedUser: function ($stateParams) {
                return $stateParams.selectedUser;
            },
            logedUser: function ($stateParams) {
                return $stateParams.logedUser;
            }
        },
        controllerAs: '$resolve'
    }).state('activityDetail', {
        parent: 'activities',
        url: '/activityDetail',
        templateUrl: 'Template/Activity/activityDetail.html'
    });
});
```

V ukázce je definován stav `activities`, jeho URL, šablona, parametry stavu (předávané pomocí URL), vstupní data a jméno komponenty, která daný stav obsluhuje. Déle je definován vnořený stav `activityDetail`, který představuje detail aktivity. Na ukázce níže je zobrazena definice komponenty `activities` v souboru `activitiesComponent.js`.

Listing 4.4: Deklarace komponenty v activitiesComponent.js

```
angular.module('App').component('activities', {
    bindings: {activities: '=', selectedUser: '=', logedUser: '='
       },
    templateUrl: 'Template/Activity/activityTable.html',
    controller: function ($activityService) {
        var self = this;
        this.getActivityDetail = function (id) {
            $activityService.getActivityDetail(id).then(function (
               response) {
                self.activity = response;
            });
        }
    }
})
```

V ukázce výše je definována komponenta `activities`, její vstupní data, šablona a controller, který obsahuje pouze funkci `getActivityDetail()`. Tato komponenta využívá objekt `$activityService`, který zajišťuje komunikaci se serverem. Tento objekt je definován v souboru `activitityService`, jehož obsah je zobrazen níže:

Listing 4.5: Obsah activityService.js

```
App.service('$activityService', ['$http', function ($http) {
        return { getActivityDetail: function (id) {
                return $http.get('http://localhost:8080/
                   activityDetail', {params: {"id": id}})
                        .then(function (response) {
                            return response.data;
                        });
                }
         };
    }]);
```

V příkladu výše je definována funkce `getActivityDetail`, která komunikuje se serverem pomocí objektu `$http`. Návratovou hodnotou této funkce je objekt `promise`, funkce tedy nečeká až server pošle požadovaná data. Podobně jsou naprogramovány i ostaní soubory ze složky Service.

Po naprogramování komponent a souborů pro komunikaci se serverem byly, v jednotlivých komponentách, postupně přidány vnořené stavy. Poté byly vytvořeny HTML šablony k těmto stavům. V tabulce 4.2 jsou uvedeny stavy v klientské aplikaci.

Název stavu	**Komponenta stavu**	**Popis stavu**
login	loginComponent	Stav pro přihlášení uživatele, počáteční stav
home	homeComponent	Stav úvodní obrazovky
users	usersComponent	Stav pro obrazovku uživatelů
createUser	usersComponent	Stav pro vytvoření uživatele
userDetail	usersComponent	Stav pro detail uživatele
todos	todosComponent	Stav obrazovku úkolů
createTodo	todosComponent	Stav pro vytvoření úkolu
updateTodo	todosComponent	Stav pro úpravu úkolu
trackTodo	todosComponent	Stav pro sledování času stráveného prací na úkolu
todoDetail	todosComponent	Stav pro detail úkolu
projects	projectsComponent	Stav pro obrazovku projektů
createProject	projectsComponent	Stav pro vytvoření projektu
projectDetail	projectsComponent	Stav pro detail projektu
activities	activitiesComponent	Stav pro obrazovku aktivit
activityDetail	activitiesComponent	Stav pro detail aktivity

Tabulka 4.2: Stavy klientské aplikace

Následně bylo přistoupeno k tvorbě Sass stylů pro úpravu vzhledu šablon. Dále bylo nutné vytvořené soubory s koncovkou `scss` přeložit do css, minifikovat a sloučit do souboru `style.css`.

4.4.3 Vývoj uživatelského rozhraní

Ke každému stavu z tabulky 4.2 byla vytvořena HTML šablona. Šablony stavů, které obsluhují další, vnořené stavy obsahují element `<div ui-view></div>`. Tento element označuje místo kam bude vložena šablona vnořeného stavu. Téměř všechny šablony jsou vloženy do elementu s id `container`. K tomuto id jsou přiřazeny Sass styly upravující vzhled elementů. Na příkladu níže je zobrazena část souboru `container.scss`, která definuje styly pro elementy s id `container` a `containerWithBorder`. Při tvorbě těchto souborů bylo využito vlastností preprocesoru Sass.

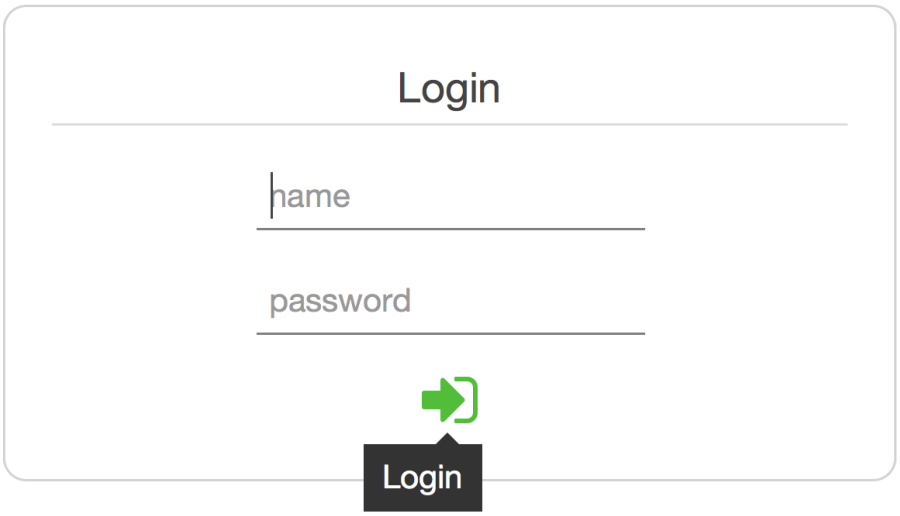
Listing 4.6: Styl pro elementy s id container

```
@import "common.scss";

#container{
    margin-top: 10px;
    margin-left: auto;
    margin-right: auto;
    background: $bg_color;
    color: $text_color;
    font-size: $font-size;
}
#containerWithBorder{
    @extend #container;
    border: $border;
    border-radius:  $rounded_corners_br;
    padding: $rounded_corners_pad;
}
```

Při tvorbě uživatelského rozhraní byly použity komponenty z frameworku UIkit, především komponenty pro úpravu rozvržení objektů na obrazovce, například `uk-flex`. HTML elementy `table` používají komponenty `uk-table` a pro dynamickou úpravu šířky jednotlivých elementů jsou použity komponenty `uk-width`. Pro výběr data, případně času byly použity komponenty `uk-datepicker` a `uk-timepicker`.

Místo standarních tlačítek byly použity komponenty `uk-icon` opatřené stylem pro úpravu barvy a velikosti. Vzhledem k tomu, že komponenty `uk-icon` neobsahují žádný popis, byly využity komponenty `data-uk-tooltip`. Tyto komponenty zobrazí popis akce ikony po najetí kurzorem na ikonu. Na následujícím obrázku je zobrazeno využití komponent `uk-icon` a `data-uk-tooltip`.



Obrázek 4.1: Přihlašovací formulář

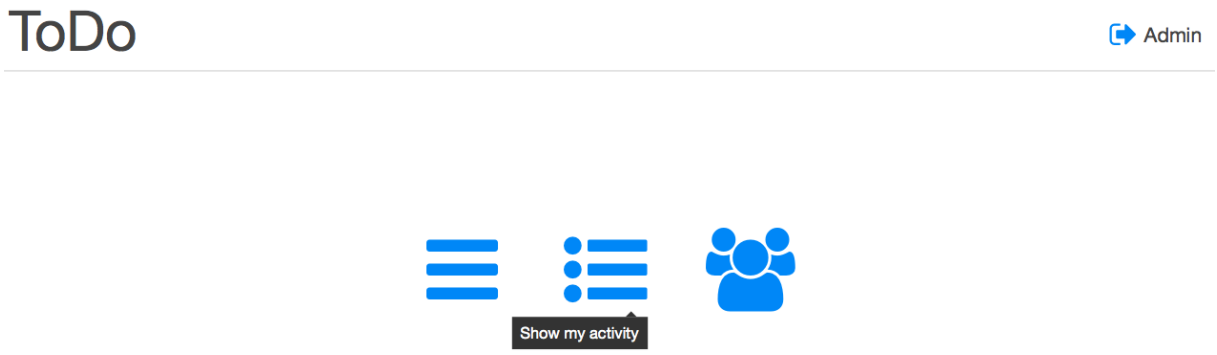
Při tvorbě uživatelského rozhraní byla podsatná snaha o co nejjednoduší, přehledné rozhraní. Aplikace je rozdělena na čtyři hlavní stavy:

- `home`
- `users`
- `projects`
- `activities`

Nehledě na právě aktivní stav aplikace vždy zobrazuje záhlaví aplikace, tedy šablonu s názvem `header.html`. Tato šablona umožňuje uživateli přecházet mezi stavy bez nutnosti vracet se na úvodní obrazovku (stav `home`). Následuje bližší popis výše zmíněných stavů a jejich obrazovek.

home

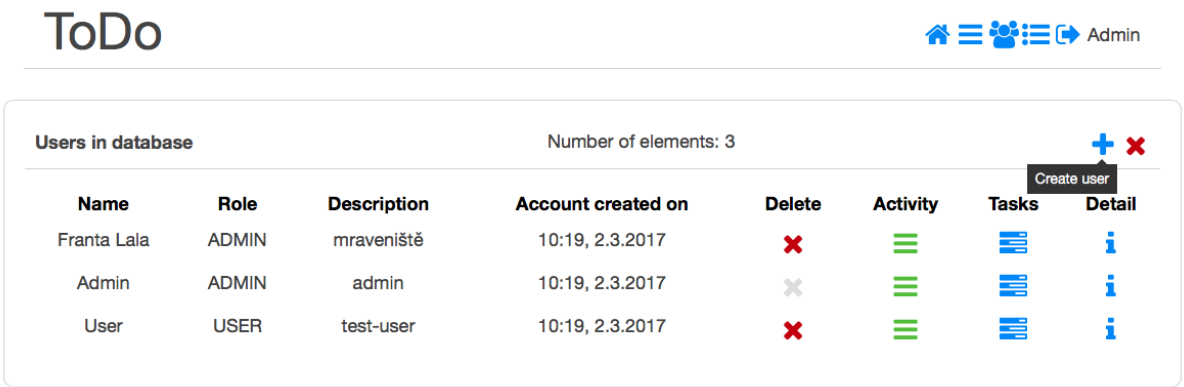
Úvodní obrazovka, která se zobrazí po přihlášení, obsahuje tlačítka vedoucí na obrazovky zobrazující informace o uživatelích, projektech a aktivitách. Dále obsahuje tlačítko pro odhlášení uživatele. Vzhled úvodní obrazovky je zobrazen na následujícím obrázku.



Obrázek 4.2: Úvodní obrazovka aplikace

users

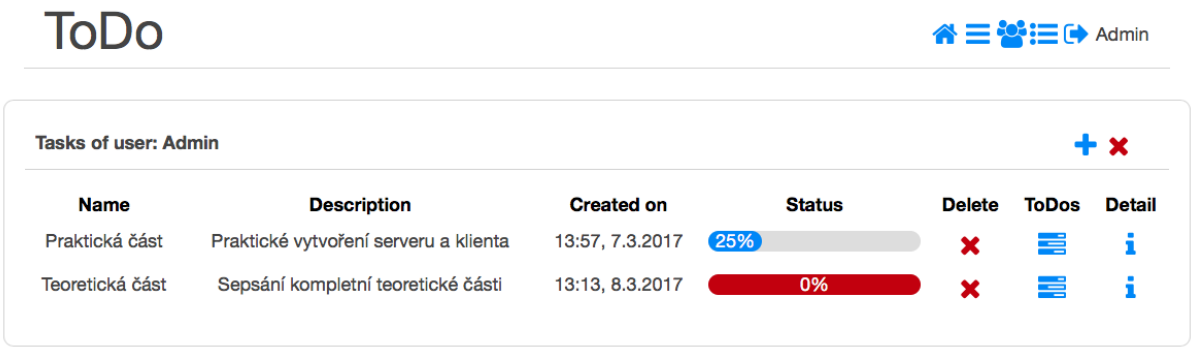
Stav `users` představuje obrazovku uživatelů. Tato obrazovka je přístupná pouze uživateli s rolí `ADMIN` a zobrazuje informace o uživatelích aplikace, umožňuje přidávat a mazat jednotlivé uživatele. Déle umožňuje uživateli spravovat projekty a úkoly jednotlivých uživatelů. Obrazovka uživatelů je zobrazena na následujícím obrázku.



Obrázek 4.3: Obrazovka uživatelů

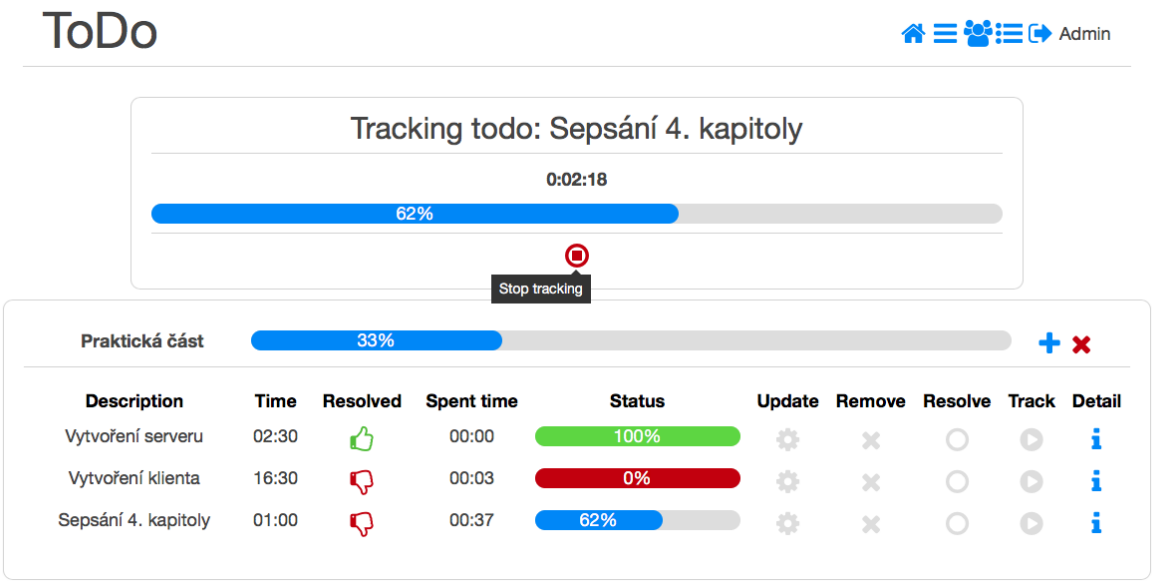
projects

Tato obrazovka zobrazuje projekty přihlášeného případně vybraného uživatele, umožňuje mazat, přidávat projekty a prohlížet si detaily projektů. Obrazovka projektů je zobrazena na následujícím obrázku.



Obrázek 4.4: Obrazovka projektů

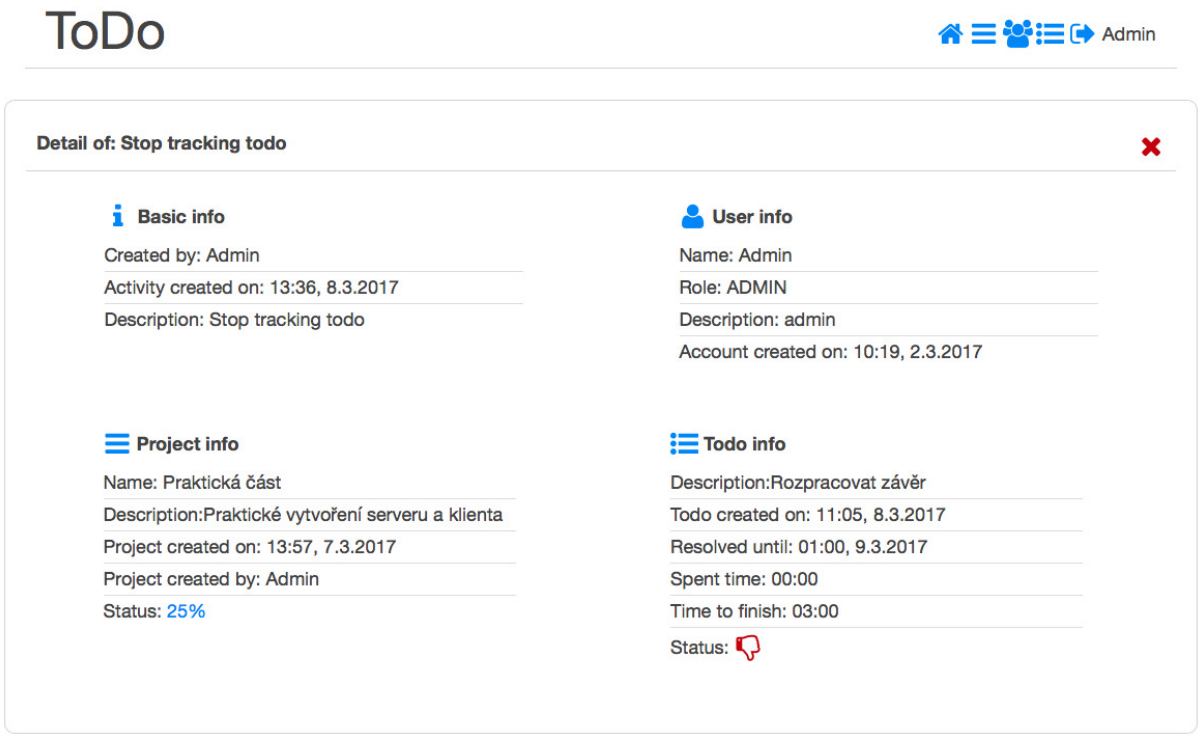
Z tohoto stavu je možné přejít na obrazovku úkolů (stav `todos`) vybraného projektu. Obrazovka úkolů umožňuje přidávat, mazat, upravovat jednotlivé úkoly. Dále umožňuje sledovat čas strávený prací na vybraném úkolu (viz. obrázek 4.5).



Obrázek 4.5: Obrazovka úkolů

activities

Obrazovka aktivit umožňuje sledovat aktivity přihlášeného uživatele. Pokud má přihlášený uživatel roli `ADMIN` tak může sledovat také aktivity a projekty ostaních uživatelů. Obrazovka aktivit s oteveřeným detailem aktivity je zobrazena na následujícím obrázku.



Obrázek 4.6: Obrazovka aktivit

Kapitola 5

Vyhodnocení praktické části

V předchozích kapitolách byl popsán zvolený přístup a tvorba demonstrační aplikace. Tato kapitola se zabývá vyhodnocením rozdílů v tomto přístupu ve srovnání s klasickým přístupem reprezentovaným tradičními web MVC farmeworky.

Demonstrační aplikace je rozdělena na dvě samostatné aplikace. První aplikace představuje stranu serveru a druhá aplikace představuje stranu klienta. Strana serveru komunikuje s klinetem pomocí REST API. Tyto vlastnosti umožňují nahradit aplikaci na straně klienta za jinou, využívající rozdílnou technologii. Je také možné nahradit aplikaci na straně serveru. Nová aplikace může být napsána v jiném jazyce musí, ale umožňovat komunikaci pomocí stejného REST API a poskytovat data se stejnou struktutrou. Důvody pro změnu technologií mohou být různé. V praxi je často nutné změnit sadu technologií na straně klienta z důvodu nahrazení zastaralé technologie, úprav funkcí klientské aplikace a další. Tato modulární struktura umožňuje měnit klientskou aplikaci bez nutnosti měnit aplikaci na straně serveru, což usnadňuje vývoj.

Dalším rozdílem může být vykreslování HTML šablon. Ve zvoleném přístupu je využit framework AngularJS, který vytváří HTML stránky sloučením HTML šablony a dat získaných se serveru. Tento proces probíhá na straně klienta a umožňuje měnit pouze určitou část stránky, není nutné znovu načítat celou stránku. Klientská aplikace přijímá od serveru data ve formánu JSON. Na druhou stranu, pokud dojde při vykreslování šablony k nějaké chybě, o které by měl být informován server, je nutné informaci o této chybě odeslat na server.

Klasické MVC frameworky vykreslují HTML šablony na straně serveru. Na stranu klienta je tedy odesílán HTML dokument. Klient tento dokument pouze zpracuje a zobrazí uživateli. Pokud dojde při vykreslování šablony k nějaké chybě, server tuto chybu zachytí a zpracuje.

Často diskutovaná je také rychlost načítání ve vztahu k vykreslování HTML šablon. Pokud je při prvním načtení stránky nutné načíst větší počet objektů se serveru je vhodnější použít vykreslování HTML šablon na straně serveru. Stránka je získána pouze jedním požadavkem. Při použití vykreslování šablon na straně klinta je nejprvé nutné získat objekty ze serveru, to může představovat mnoho požadavků. Mezi těmito přístupy existuje kompromis, vykreslování šablon na straně serveru je použito pouze pro počáteční načtení stránky a další šablony už jsou vykreslovány na straně klienta. Tento přístup nabízí například React s využitím Node.js na straně serveru.

Zvolený přístup přenáší část aplikační logiky ze strany serveru na stranu klienta. Příkladem může být například výpočet času stráveného prací na úkolu u demonstrační aplikace. Tento výpočet se provádí ihned po ukončení sledování práce na úkolu, konkrétně jej provádí metoda `stopTracking()` modulu `timeService`. Na server je odeslán úkol s již vypočítaným stráveným časem. Na straně serveru je tento požadavek zpracován pomocí metody `updateTodo`. Není tedy potřeba žádné další URL, které by řešilo výpočet stráveného času. Tento přístup umožňuje zjednodušení komunikačního rozhraní mezi serverem a klinetem. Dalším příkladem může být mazaní objektu projekt. Tuto operaci obsluhuje na straně klienta metoda `deleteProject()`, nejprve je odeslán požadavek o smazání projektu a poté je smazán projekt z množiny projektů na straně klienta. Není tedy nutné znovu načítat množinu projektů ze serveru. Přenos aplikační logiky na klientskou aplikaci snižuje zátěž serveru, je využit výpočetní výkon zařízení na kterém běží klientská aplikace. Server je tak schopný obsloužit více klientů najednou.

Listing 5.1: Metoda deleteProject

```
this.deleteProject = function (project) {
  $projectService.deleteProject(project).then(function (response)
     {
          //deleting on client
          for (var i = 0; i < self.projects.length; i++){
            if (self.projects[i].id === project.id)
                  {
                    self.projects.splice(i, 1);
                  }
           }
        });
};
```

Kapitola 6

Závěr

V bakalářské práci byly splněny všechny požadavky zadání. Nejprve byla provedena analýza definic dostupných v odborné literatuře. Na základě této analýzy byl vymezen termín Rich Internet Applications. Poté byl čtenář seznámen s některými technologiemi současných přístupů k vývoji RIA. Nejprve byly posány některé komponenty HTML5. Následně byly stručně představeny jazyky a standardy JavaScript, EcmaScript a TypeScript. Déle byl čtenář seznámen s některými nástroji na správu závislostí, tedy NPM a Bower. Zde by bylo možné představit i další nástroje na správu závislostí, například JSPM [49]. Následně práce popisuje nástroje na správu sestavení, konkrétně Gulp a Grunt. Těchto nástrojů existuje velké množství, seznámení se všemi by však přesáhlo rámec této práce.

V kapitole 3 byly stručně představeny a porovnány technologie AngularJS a React. Framework AngularJS byl zvolen na základě zkušeností autora této práce. Knihovna React byla zvolena pro její odlišný přístup k tvorbě uživatelských rozhraní. Obě technologie mají specifické vlastnosti a podmínky v nichž nacházejí uplatňení. Tyto rozdíly jsou stručně rozebrány v závěru kapitoly 3.

Na začátku kapitoly 4 byla zvolena sada technologií, která byla použita pro tvorbu demonstrační aplikace. Zde bylo možné vybírat z velkého množství technologií, bylo by možné použít například knihovnu React s využitím návrhového vzoru Flux [51]. Jednotlivé technologie byly vybrány na základě předpokládaného použití, s přihlédnutím na zkušenosti autora této práce. Nejprve byly popsány zvolené technologie na straně serveru a poté na straně klienta. Kapitola 4 se dále věnuje specifikaci a návrhu demonstrační aplikace. Následně byla popsána zamýšlená struktura a funkce jednotlivých částí aplikce.

Po návrhu demostrační aplikace bylo přistoupeno k samotnému vývoji. Nejprve byl popsán vývoj serveru, který poskytuje data pomocí REST rozhraní ve formátu JSON. K ukládání dat byla využita relační databáze.

Poté je čtenář seznámen s vývojem klientské aplikace s využitím frameworku AngularJS a knihovny UI-router. Tato kombinace umožňuje k jednotlivým stavům definovat komponenty, které daný stav obsluhují a tím zjednodušují a zpřehledňují celý vývoj. Při vývoji klientské aplikace bylo nutné vyřešit několik problémů, které souvisely se zpracováním času. Tyto problémy byly vyřešeny vytvořením objektu `timeService`, který poskytuje metody pro vytvoření JavaScript objektu Date a výpočet uplynulého času. Dalším možným řešením by bylo nahrazení UIkit komponent `uk-datepicker` a `uk-timepicker` jinými komponentami pro výběr data a času, například `md-datepicker` z balíčku Angular Material [50]. Aplikace využívá pouze základní autentizaci z balíčku Spring Security. Aplikaci by bylo možné rozšířit o pokročilejší způsob autentizace a práci rozšířit o část věnující se zabezpečení RIA.

V kapitole 5 byly vyhodnoceny rozdíly mezi přístupem použitým k tvorbě demonstrační aplikace a přístupem reprezentovaným tradičními web MVC farmeworky. Samotný kód demonstrační aplikace je dostupný na přiloženém CD, případně na adrese: https://github.com/Jakub-Vacek.

Na tuto práci by bylo možné navázat a zaměřit se na vývoj RIA, které využívají koncept Server Push (viz. 2.3.7) a technologie jako Express.js [42] nebo Socket.io [43].

Literatura

[1] MELIA, Santiago, Jaime GOMEZ, Sandy PAREZ a Oscar DIAZ. A Model-Driven Development for GWT-Based Rich Internet Applications with OOH4RIA. *Web Engineering*. IEEE, 2008. DOI: 10.1109/ICWE.2008.36. ISBN 10.1109/ICWE.2008.36. Dostupné z: http://ieeexplore.ieee.org/document/4577865/

[2] BOZZON, Alessandro, Sara COMAI, Piero FRATERNALI a Giovanni Toffetti CARUGHI. *Capturing RIA concepts in a web modeling language*. 2006, , 907 - 908. DOI: 10.1145/1135777.1135938. ISBN 10.1145/1135777.1135938. Dostupné z: http://portal.acm.org/citation.cfm?doid=1135777.1135938

[3] FRATERNALI, Piero, Gustavo ROSSI a Fernando S??NCHEZ-FIGUEROA. Rich Internet Applications. *IEEE Internet Computing*. IEEE, 2010, (3), 9 - 12. DOI: 10.1109/MIC.2010.76. ISBN 10.1109/MIC.2010.76. ISSN 1089-7801. Dostupné z: http://ieeexplore.ieee.org/document/5481362/

[4] DEITEL, Paul J. a Harvey M. DEITEL. *Ajax, rich Internet applications, and web development for programmers*. Upper Saddle River, NJ: Prentice Hall, 2008. ISBN 0131587382.

[5] DUHL, Joshua. *Rich Internet Applications* [online]. 2002, 1-33 [cit. 2016-11-30]. Dostupné z: https://www.adobe.com/platform/whitepapers/idc_impact_of_rias.pdf

[6] ALLAIRE, Jeremy. *Macromedia Flash MX: A next-generation rich client* [online]. Macromedia, 2002 [cit. 2016-11-30]. Dostupné z: http://download.macromedia.com/pub/flash/whitepapers/richclient.pdf

[7] Remote Scripting. *Microsoft MSDN* [online]. Redmond: Microsoft, 1999 [cit. 2017-03-20]. Dostupné z: https://msdn.microsoft.com/en-us/library/ms974566.aspx

[8] HOWE, Carl. *The X Internet* [online]. Cambridge: Forrester Research, 2001, , 1-20 [cit. 2016-12-01]. Dostupné z: http://ebusiness.mit.edu/sponsors/common/2001-Nov-BdMtg/TheXInternetGeorgeColony.pdf

[9] GARRETT, Jesse James. *Ajax: A New Approach to Web Applications* [online]. In: . 2005 [cit. 2016-12-01]. Dostupné z: http://adaptivepath.org/ideas/ajax-new-approach-web-applications/

[10] BUSCH, Marianne a Nora KOCH. *Rich Internet Applications: State-of-the-Art* [online]. Mnichov: Ludwig-Maximilians-Universitat, 2009 [cit. 2017-04-01]. Dostupné z: http://uwe.pst.ifi.lmu.de/publications/maewa_rias_report.pdf

[11] Usage Statistics of Flash. *W3techs* [online]. Maria Enzersdorf: W3 Techs, c2009 [cit. 2017-04-02]. Dostupné z: https://w3techs.com/technologies/details/cp-flash/all/all

[12] Applet (Java Platform SE 8). *Java Platform Standard Edition 8 Documentation* [online]. [cit. 2017-03-31]. Dostupné z: https://docs.oracle.com/javase/8/docs/api/java/applet/Applet.html

[13] *Microsoft Silverlight* [online]. Redmond: Microsoft, 2007 [cit. 2017-03-13]. Dostupné z: https://www.microsoft.com/silverlight/

[14] Browser Cookie Limits. *Browser Cookie Limits* [online]. [cit. 2016-12-02]. Dostupné z: http://browsercookielimits.squawky.net

[15] PILGRIM, Mark. *HTML5*. Sebastopol: O'Reilly Media, 2010. ISBN 14-493-9966-5.

[16] HTML - storage interface: Living Standard. *WHATWG* [online]. https://whatwg.org: WHATWG [cit. 2016-12-03]. Dostupné z: https://html.spec.whatwg.org/multipage/webstorage.html#the-storage-interface

[17] HTML5 Local storage. *Sitepoint* [online]. [cit. 2016-12-03]. Dostupné z: https://www.sitepoint.com/html5-local-storage-revisited/

[18] FRAIN, Ben. *Responsive web design with HTML5 and CSS3 learn responsive design using HTML5 and CSS3 to adapt websites to any browser or screen size*. Birmingham: Packt Publishing Ltd, 2012. ISBN 18-496-9319-6.

[19] FLANAGAN, David. *JavaScript: the definitive guide*. Fifth edition. Sepastopol, CA: O'Reilly, 2006. ISBN 05-961-0199-6.

[20] Usage statistics of JavaScript. *W3techs* [online]. Maria Enzersdorf: W3 Techs, c2009 [cit. 2017-03-18]. Dostupné z: https://w3techs.com/technologies/details/cp-javascript/all/all

[21] *NPM* [online]. Oakland: NPM, 2009 [cit. 2016-12-06]. Dostupné z: https://www.npmjs.com

[22] *Bower: oficiální stránky* [online]. San Francisco: Twiter, 2012 [cit. 2016-12-08]. Dostupné z: https://bower.io

[23] W3techs: využití jQuery. *W3techs* [online]. Maria Enzersdorf: W3 techs, c2009 [cit. 2016-12-08]. Dostupné z: https://w3techs.com/technologies/details/js-jquery/all/all

[24] PILLORA, Jaime. *Getting Started with Grunt: The JavaScript Task Runner*. Livery Place 35 Livery Street Birmingham B3 2PB, UK.: Packt Publishing, 2014. ISBN 978-1-78398-062-8.

[25] Standard Ecma-262. *Ecma international* [online]. Rue du Rhóne: Ecma International, 1997 [cit. 2016-12-15]. Dostupné z: http://www.ecma-international.org/publications/standards/Ecma-262.htm

[26] Standard Ecma-262 1. *Ecma international* [online]. Rue du Rhóne: Ecma International, 1997 [cit. 2016-12-15]. Dostupné z: http://www.ecma-international.org/publications/files/ECMA-ST-ARCH/ECMA-262,%201st%20edition,%20June%201997.pdf

[27] Standard Ecma-262 2. *Ecma international* [online]. Rue du Rhóne: Ecma International, 1997 [cit. 2016-12-15]. Dostupné z: http://www.ecma-international.org/publications/files/ECMA-ST-ARCH/ECMA-262,%202nd%20edition,%20August%201998.pdf

[28] Standard Ecma-262 3. *Ecma international* [online]. Rue du Rhóne: Ecma International, 1997 [cit. 2016-12-15]. Dostupné z: http://www.ecma-international.org/publications/files/ECMA-ST-ARCH/ECMA-262,%203rd%20edition,%20December%201999.pdf

[29] Standart Ecma-262 5. *Ecma international* [online]. Rue du Rhóne: Ecma International, 1997 [cit. 2016-12-15]. Dostupné z: http://www.ecma-international.org/publications/files/ECMA-ST-ARCH/ECMA-262%205th%20edition%20December%202009.pdf

[30] Standard Ecma-262 7. *Ecma international* [online]. Rue du Rhóne: Ecma International, 1997 [cit. 2016-12-15]. Dostupné z: http://www.ecma-international.org/ecma-262/7.0/index.html

[31] TC39. *Ecma international* [online]. [cit. 2016-12-15]. Dostupné z: https://www.ecma-international.org/memento/TC39.htm

[32] *Babeljs* [online]. 2012 [cit. 2016-12-15]. Dostupné z: http://babeljs.io

[33] FENTON, Steve. *Pro TypeScript: Application-Scale JavaScript Development*. 1. New York: Apress, 2014. ISBN 9781430267911.

[34] *TypeScript* [online]. Redmond: Microsoft, 2012 [cit. 2016-12-19]. Dostupné z: https://www.typescriptlang.org/index.html

[35] Usage statistics of HTTP2. *W3techcs* [online]. Maria Enzersdorf: W3 Techs, c2009 [cit. 2017-03-17]. Dostupné z: https://w3techs.com/technologies/details/ce-http2/all/all

[36] *HTTP/2* [online]. Fremont: IETF [cit. 2017-03-17]. Dostupné z: https://http2.github.io

[37] HPACK: Header Compression for HTTP/2. *HTTP/2* [online]. Fremont: IETF, 2014 [cit. 2017-03-17]. Dostupné z: http://httpwg.org/specs/rfc7541.html

[38] HTML - WebSocket. *HTML Standart* [online]. WHATWG, 2004 [cit. 2017-03-17]. Dostupné z: https://html.spec.whatwg.org/multipage/comms.html#network

[39] HTML Standartd - Server-sent events. *HTML Standartd* [online]. WHATWG [cit. 2017-03-31]. Dostupné z: https://html.spec.whatwg.org/#server-sent-events

[40] Can I use server-sent events. *Can I use* [online]. [cit. 2017-03-31]. Dostupné z: http://caniuse.com/#feat=eventsource

[41] *Node.js* [online]. Sausalito: Joyent, 2009 [cit. 2017-03-18]. Dostupné z: https://nodejs.org/en/

[42] *ExpressJS* [online]. San Mateo: Strong Loop, c2016 [cit. 2017-03-18]. Dostupné z: https://expressjs.com

[43] *Socket.io* [online]. 2014 [cit. 2017-03-18]. Dostupné z: https://socket.io

[44] *AngularJS* [online]. Mountain View: Google, 2009 [cit. 2017-01-21]. Dostupné z: https://angularjs.org

[45] *React* [online]. Palo Alto: Facebook, 2013 [cit. 2017-01-21]. Dostupné z: https://facebook.github.io/react/

[46] FREEMAN, Adam. *Pro AngularJS*. New York: Apress, 2014. ISBN 1430264489.

[47] GACKENHEIMER, Cory. *Introduction to React*. 1. New York: Apress, 2015. Expert's voice in Web development. ISBN 14-842-1246-0.

[48] *Spring Boot* [online]. San Francisco: Pivotal, 2012 [cit. 2017-03-15]. Dostupné z: https://projects.spring.io/spring-boot/

[49] *Jspm.io* [online]. [cit. 2017-03-11]. Dostupné z: http://jspm.io

[50] *Angular Material* [online]. Mountain View: Google, 2009 [cit. 2017-03-11]. Dostupné z: https://material.angularjs.org/latest/

[51] *Flux* [online]. Palo Alto: Facebook, c2014 [cit. 2017-03-20]. Dostupné z: http://facebook.github.io/flux/

Seznam obrázků

Seznam úryvků kódu

Seznam tabulek

Příloha A

Obsah přiloženého CD/DVD

K této práci je přiloženo CD/DVD s následující adresářovou strukturou.

- `Praktická část`: Netbeans projekty klienta a serveru
  - `Client`: Netbeans projekt klientské aplikace
  - `Server`: Netbeans projekt serveru, v složce `Target` se nachází kompletní sestavený projekt
- `Vacek_BP_2017`– Bakalářská práce ve formátu LaTeX
- `Vacek_BP_2017.pdf`– Bakalářská práce ve formátu PDF

Příloha B

Použitý software

LaTeX ⟨http://www.miktex.org/⟩

TeXShop ⟨http://pages.uoregon.edu/koch/texshop/⟩

NetBeans IDE ⟨https://netbeans.org⟩

Apache Maven ⟨https://maven.apache.org⟩

Spring Boot ⟨http://projects.spring.io/spring-boot/⟩

ORMLite ⟨http://ormlite.com⟩

Apache Derby ⟨https://db.apache.org/derby/⟩

NPM ⟨https://www.npmjs.com⟩

Bower ⟨https://bower.io⟩

AngularJS ⟨https://angularjs.org⟩

UI-router ⟨https://ui-router.github.io/about/⟩

UIKit ⟨https://getuikit.com⟩

SASS ⟨http://sass-lang.com⟩

Gulp ⟨http://gulpjs.com⟩

Příloha C

Stažení a spuštění demonstrační aplikace

Tato příloha popisuje stažení projektu z adresy: https://github.com/Jakub-Vacek. Déle je popsáno spuštení aplikace pomocí příkazové řádky. Stažení serverové a klientské aplikace probíhá pomocí příkazové řádky a nástroje Git. Git je možné jej stáhnout na adrese: https://git-scm.com/downloads.

C.1 Server

Stažení serverové aplikace je možné provést příkazem:

```
git clone https://github.com/Jakub-Vacek/BcAppServer.git
```

Následně je nutné sestavit stažený projekt, k tomu je použit nástroj Maven. Maven lze stáhnout na adrese: https://maven.apache.org/download.cgi. Samotné sestavení se provádí příkazem `mvn package` v kořenové složce projektu. Poté již stačí aplikaci spustit příkazem `java -jar target/BcApp-1.0-SNAPSHOT.jar`. Databáze obsahuje dva vytvořené uživatelské účty, první se standartní rolí a přihlašovacími údaji User/Use, druhý s administrátorskou rolí a přihlašovacími údaji Admin/Admin. Plnění databáze testovacími účty lze zrušit zakomentováním řádku 13. v souboru `Application.java`.

C.2 Klientská aplikace

Stažení klientské aplikace je možné provést příkazem:

```
git clone https://github.com/Jakub-Vacek/BcApClient.git
```

Následně je nutné nainstalovat závislosti klientské aplikace, k tomu je použit nástroj NPM. NPM lze stáhnout na adrese: https://www.npmjs.com. Samotné stažení závislostí se provádí příkazem `nmp instal` a následně `bower install` v kořenové složce projektu. Poté je možné příkazem `gulp serve` spustit jednoduchý server. Klinetská aplikace běží na adrese: http://localhost:8383. Pro přihlášení je nutný běžící server.