Sem vložte zadání Vaší práce.

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra softwarového inženýrství

Bakalářská práce

Webová aplikace HashStory

Alexandr Marinko

Vedoucí práce: Ing. Jaroslav Kuchař

10. května 2015

Poděkování

Chtěl bych poděkovat vedoucímu práce Ing. Jaroslavu Kuchařovi za ochotu, korektnost a vynaložený čas při vedení mé bakalářské práce. Dále bych rád poděkoval své rodině a mým přátelům za veškerou podporu během studia.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V Buštěhradě dne 10. května 2015 .....

Odkaz na tuto práci

Marinko, Alexandr. *Webová aplikace HashStory*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2015.

Abstrakt

Tato práce popisuje vývoj webové aplikace HashStory. Cílem práce je vytvořit aplikaci umožňující uživatelům sociálních sítí sestrojit prezentační timeline příběhu z vlastních příspěvků. Pozornost je věnována přístupu k datům sociálních sítí, asynchronnímu zpracování importu a návrhu aplikačního rozhraní (API) pomocí architektonického stylu REST (Representational state transfer). Výsledkem je prototyp aplikace, který poslouží k budoucímu rozvoji.

Klíčová slova Webová aplikace, prezentační timeline příběhu, API sociálních sítí, REST API, RabbitMQ, Symfony, Backbone.js

Abstract

This thesis describes the development of web application HashStory. The main goal is to allow social network users to construct presentational timeline of story based on their own posts. The focus is put on social networks data access, asynchronous processing of import event and designing application programming interface (API) using architectural style REST (Representational state transfer). The result is a prototype of application that will serve for the future development.

Keywords Web application, presentational timeline of story, social network APIs, REST API, RabbitMQ, Symfony, Backbone.js

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Příchod sociálních sítí a běžně dostupných mobilních zařízení ovlivnil lidskou společnost. Lidé, zejména aktivní uživatelé, sdílí každým dnem různé informace počínaje politikou a konče fotografií, co zrovna bylo k večeři. Stal se z toho lidový obyčej, že dokumentujeme každý moment a každý detail. Při takto častém sdílení mají příspěvky tendenci rychle stárnout a uživatel při hledání příspěvku staršího data může ocitnout v nekonečném scrollování. Uživatele přitom často nezajímá daný příspěvek v minulosti, ale příběh který prožil – tedy příspěvky, které spolu tématicky souvisí. Tento příběh tedy žije v minulosti a uživatele s nim často pojí osobní vzpomínky. I z nostalgického hlediska vzniká u uživatele potřeba si tento příběh příležitostně připomínat.

Cílem bakalářské práce je navrhnout, implementovat a otestovat webovou aplikaci, která umožní uživatelům sociálních sítí sestrojit prezentační timeline příběhu z vlastních příspěvků. K vyjádření souvislosti mezi příspěvky je možné použít hashtag. Uživatelé tak díky této webové aplikaci získají možnost zobrazit příběh v prezentační podobě a obnovit vzpomínky z minulosti. Aplikace může taktéž sloužit jako prostor pro prohlížení inspirativních příběhů na jednom místě.

V první kapitole této práce je popsán současný stav problematiky a upřesněny požadavky kladené na aplikaci. V druhé kapitole je věnována pozornost API sociálních sítích, zejména přístupu k datům v rámci dané sítě. Třetí kapitola popisuje návrh aplikace a její důležité součásti. V kapitole o implementaci jsou popsány vybrané části a zajímavosti, které se vyskytly v průběhu implementace. Další kapitola popisuje způsob testování aplikace. Závěrečná kapitola pak shrnuje poznatky a popisuje budoucí vývoj.

Kapitola 1

Analýza

Tato kapitola se nejdříve zabývá rešerší současné problematiky. V rešerši jsou popsány jednotlivé subjekty a jejich odlišnosti v přístupu řešení. Z provedené rešerše bylo následně zjištěno, že daná problematika existuje. V dalších částech kapitoly jsou specifikovány požadavky kladené na aplikaci a případy užití.

1.1 Současný stav problematiky

1.1.1 Google Plus Stories

Google se v rámci sociální sítě Google Plus zaměřuje na multimediální obsah. Při splnění určitých podmínek [1], hlavně poskytnutí fotografií a videií, je uživateli automaticky vytvořena prezentační timeline příběhu. Příběh je navíc doplněn o geolokační informace. Joseph Smarr, Google social web engineer, uvádí [2], že doplnění výsledného příběhu o geolokační informace je založeno na 3 principech:

- získání geoinformací ze vstupních dat,
- získání dat při používání služeb Google Now [3] nebo Google Maps [4],
- rozpoznání vstupu na základě počitačového vidění.

Google řeší problém, kdy fotoalbum obsahuje velké množství fotografií, proto přidává asociaci mezi fotoalbem a příběhem – tedy shrnutí toho nejdůležitějšího v podobě retrospektivních vzpomínek.

1.1.2 Twitter Search

Twitter Search [5] umožňuje vyhledávat příspěvky podle hledaného výrazu. Pro zobrazení příspěvků od daného uživatele s daným hashtagem je možné zadat následující výraz:

```
from:@spazef0rze AND #Heartbleed
```

Výstup je informačního charakteru, chybí tomu prezentační podoba. Navíc mezi jednotlivými příspěvky není možné přidávat či editovat příspěvky.

1.1.3 Storify

Hlavním posláním Storify [6] je zachytit či zdokumentovat nedávné významné události ve světě. Storify předpokladá, že nejlepší reportér je ten, který je na místě činu ve správný čas. V době, kdy jsou mobilní zařízení běžně dostupné, se z každého člověka může stát potenciální reportér. Po zachycení určitého okamžiku pak reportéři přirozeně sdílí na různé sociální sítě. Jakmile existuje podklad na sociálních sítí, uživatelé Storify mají možnost z této události vytvořit příběh – shrnutí či zdokumentování významných bodů v rámci proběhlé události.

Vytváření příběhu je umožněno přes minimalistické WISIWIG prostředí, které je doplněno o možnost přidání příspěvků z různých sociálních sítí.

1.1.4 RebelMouse

RebelMouse [7] je zástupným příkladem aplikací, které umožňují prezentovat agregovaný obsah z různých sociálních sítí. Obsahuje uživatelsky přívětivé rozhraní pro přidávání a revidování nových příspěvků. V RebelMouse je sice možné vytvořit timeline příběhu z obsahu sociálních sítí, avšak výstup slouží spíše pro marketingové účely.

1.1.5 Share A Story

Share A Story [8] je sociální iniciativa, jíž vizí je shromažďování inspirativních příběhů lidí z celého světa. Daný příběh může představovat pro čtenáře funkci motivační až poučnou. Reprezentace příběhu je vyjádřena pouze v textové podobě, což v případě dlouhého souvislého textu přináší značné nevýhody. Sdílení příběhu je řešeno poměrně zastaralým způsobem přes e-mail.

1.1.6 Shrnutí

Z provedené rešerše bylo zjištěno, že daná problematika skutečně existuje, a dokonce se tímto tématem zabývá i samotný Google. Z uvedených řešeních sdílí podobnou vizi Google Plus Stories a Share A Story. Hlavní rozdíl spočívá ve vstupu a prezentaci příběhu.

Posledně jmenovaný Share A Story je spíš určen pro čtenáře, jelikož příběh je prezentován pouze v textové podobě. Naopak Google se primárně soustředí na generování prezentační timeline příběhu z fotoalba. Uživatel tak při vytváření příběhu věnuje minimální úsílí a minimální čas.

V aplikaci HashStory je víc kladen důraz na manuální vytváření příběhu a import obsahu ze sociálních sítí. Obsah lze navíc importovat na základě hashtagů z vlastních příspěvků. Uživateli je taktéž umožněno prohlížet příběhy ostatních uživatelů na jednom místě.

1.2 Specifikace požadavků

1.2.1 Funkční požadavky

F1. Lokální registrace a přihlášení do aplikace

- Uživatel při registraci poskytuje uživatelské jméno, e-mail a heslo. Po vyplnění registračního formuláře je uživateli odeslán e-mail s aktivačním odkazem, po přejití na odkaz je účet aktivován.
- Uživatel při přihlášení poskytuje e-mail nebo uživatelské jméno a heslo. V případě zapomenutého hesla je uživateli odeslán e-mail s odkazem k obnovení hesla, po přejití na odkaz a vyplnění formuláře je heslo obnoveno.

F2. Registrace a přihlášení do aplikace prostřednictvím sociálních sítí

- Uživatel se registruje/přihlašuje prostřednictvím služeb sociálních sítí Facebook, Twitter, Instagram.
- V případě již registrovaného uživatele propojit externí účet s lokálním.

F3. Sestrojení timeline příběhu

- Zobrazení příspěvků podle data od nejstaršího po nejnovější.
- Zobrazení mapy navštívených míst v rámci příběhu.

F4. Import příspěvků ze sociálních sítí

- Podpora importu příspěvků z Facebooku, Twitteru, Instagramu.
- Importovat na základě vstupu od uživatele: nedávné neoznačené příspěvky, nedávné příspěvky označené daným hashtagem.

1.2.2 Nefunkční požadavky

N1. Webová aplikace

- Dostupnost aplikace přes web v moderních prohlížečích Google Chrome 41.0 [9], Mozilla Firefox 37.0 [10], Opera 29.0 [11], Internet Explorer 11.0 [12].

N2. Aplikace rozšiřitelná o novou sociální síť

- Architektura aplikace umožní snadné přidání nové sociální sítě a začlenění do aplikace.

N3. Asynchronní zpracování importu ze sociálních sítí

- Událost zpracování importu příspěvků nebude zbytečně zdržovat uživatele. Událost se zpracuje asynchronně na pozadí.

1.3 Případy užití

1.3.1 Příběh

UC1. Zobrazit seznam příběhů

- Uživatel vidí pouze veřejné přiběhy.

UC2. Zobrazit příběh

- Privátní příběh je zobrazen pouze autorovi. Chráněný příběh je zobrazen uživatelům, kteří na něj mají odkaz a není zobrazen ve veřejném seznamu příběhů. Veřejný příběh je zobrazen všem uživatelům.

UC3. Vytvořit nový příběh

- Nový příběh je vytvořen ve výchozím nastavení jako privátní příběh.

UC4. Upravit příběh

UC5. Smazat příběh

1.3.2 Příspěvky příběhu

UC6. Zobrazení příspěvků v rámci příběhu

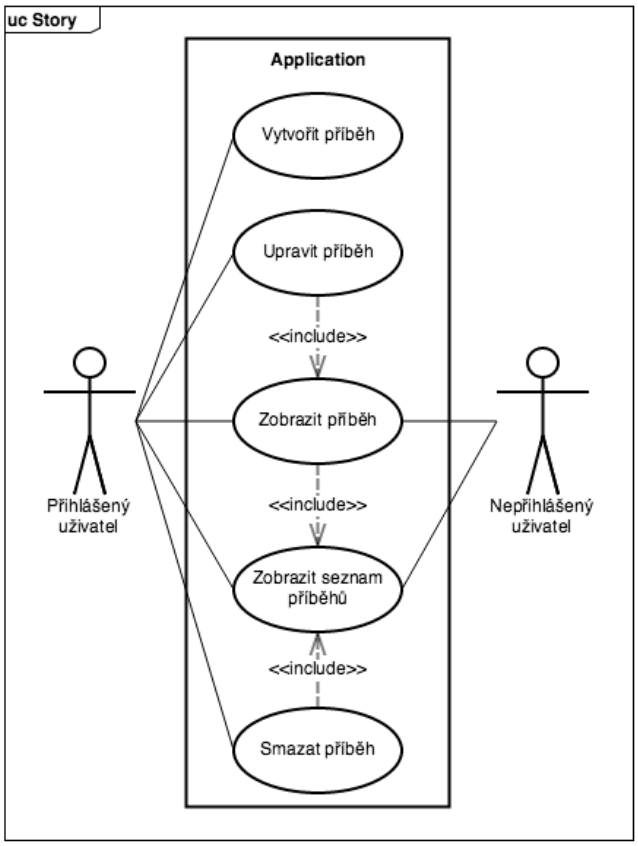
UC7. Vytvořit příspěvek v rámci příběhu

UC8. Upravit příspěvek v rámci příběhu

UC9. Smazat příspěvek v rámci příběhu

1.3.3 Import

UC10. Zobrazit importy ke konkrétní sociální síti



Obrázek 1.1: Diagram případů užití příběhu

1.3.4 Revidování importovaných příspěvků

UC11. Zobrazit importované příspěvky

UC12. Zobrazit náhled importovaného příspěvku

UC13. Výběr importovaných příspěvků podle typu nebo zdroji dat

UC14. Přidat jednotlivý importovaný příspěvek do příběhu

UC15 Dávkové přidání importovaných příspěvků do příběhu

UC16 Dávkové mazání importovaných příspěvků

Kapitola 2

API sociálních sítí

Dnes je již standartem, že webová API nabízejí přístup ke svým datům prostřednictvím HTTP pomocí REST rozhraní. V REST rozhraní je potřeba uvést jak a ke kterému zdroji dat se bude přistupovat. Zdrojem je myšleno koncový bod, který má vlastní identifikátor URI. Způsob, jak přistupovat ke zdroji, určují následující HTTP dotazovací metody [13]:

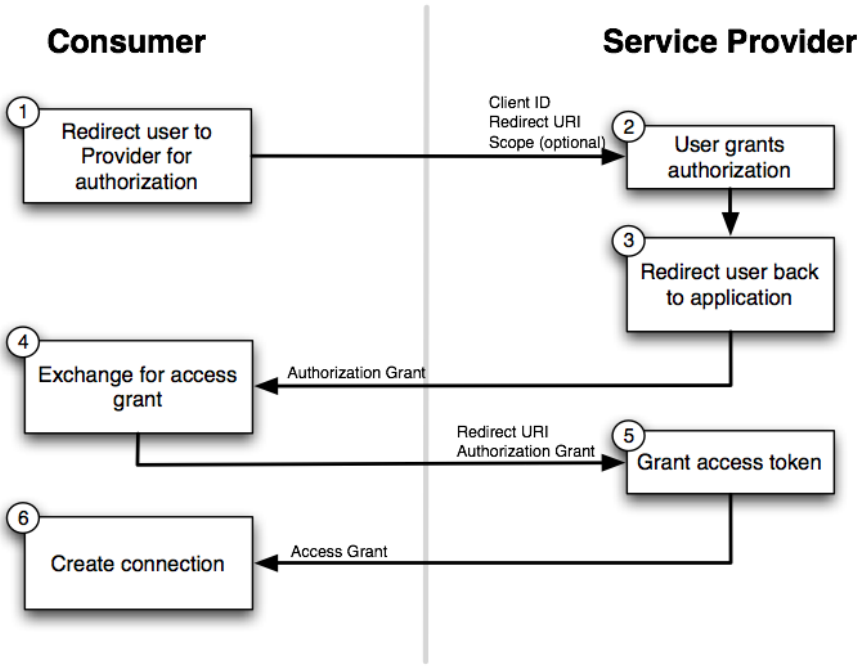
- **Get** – metoda pro získání zdroje,
- **Post** – metoda pro vytvoření nového zdroje,
- **Delete** – metoda pro smazání zdroje,
- **Put** – metoda pro úpravu celého zdroje,
- **Patch** – metoda pro částečnou úpravu zdroje.

Webová služba pak zpracuje požadavek, připraví data – pokud je potřeba, a odpoví příslušným HTTP stavovým kódem.

2.1 Autentizace

Většina služeb v současnosti, Facebook, Twitter a Instagram nevyjímaje, používá k zabezpečení svého API autentizační protokol OAuth. Oproti lokální autentizaci, kdy uživatel poskytne své přihlašovací údaje aplikaci, je uživatel přesměrován na danou službu, u které se autentizuje.

V terminologii OAuth je nutné rozlišovat Consumera a Service providera. Consumer reprezentuje aplikaci, která přistupuje ke zdrojům služby. Service provider reprezentuje službu, která tyto zdroje obsahuje. Před samotnou komunikací je ovšem potřeba zaregistrovat Consumera u Service providera – v našem případě aplikaci HashStory na webu dané služby. Při registraci se často uvádí název aplikace, adresa webu a *redirect URL*. Service provider poté



Obrázek 2.1: OAuth2 - prvotní komunikace [16]

vygeneruje *Client ID* a *Client secret*, které je potřeba v aplikaci uchovat. *Client ID* se využije při prvotní autorizaci a *Client secret* následně při získání *access tokenu*.

V momentě, kdy registrace u Service providera proběhla úspěšně, je možné začít komunikovat. Prvotní komunikace začíná autorizací. Uživatel je přesměrován na danou službu, kde se přihlásí a povolí/odmítne přístup aplikaci s určitými oprávněnimi. Poté je přesměrován zpátky na *redirect URL*, které uvedl v požadavku nebo při registraci Consumera. Aplikace, v případě povolení přístupu uživatelem, následně požádá o *access token* a pokud vše proběhlo úspěšně, je možné se začít dotazovat na API dané služby. Prvotní komunikaci znázorňuje obrázek 2.1, detailnější informace o OAuth lze nalézt ve specifikaci RFC 6749 [14] nebo na přednášce předmětu MI-W20 [15].

2.2 Dotazování

Při dotazování na API webových služeb je potřeba mít platný *access token*. *Access token* nejdříve vzniká při prvotní komunikaci a má omezenou platnost. Je nutné počítat s tím, že *access token* se může kdykoli zneplatnit – například po vypršení platnosti, odebráním aplikace uživatelem a teoreticky nepopsanou anomálií na straně Service providera. V případě vypršení platnosti je potřeba obnovit token, v ostatních případech bude potřeba absolvovat znovu prvotní

Tabulka 2.1: Facebook Graph API v2.2 - uvažované zdroje

Zdroj	Oprávnění	Typ příspěvku
/user/feed	read_stream	status, odkaz, fotografie, video
/user/links	read_stream	odkaz
/user/statuses	user_status	status
/user/photos	user_photos	fotografie
/user/photos/tagged	user_photos	fotografie
/user/videos	user_videos	video
/user/videos/tagged	user_videos	video

komunikaci. Při samotném dotazování pak aplikace ke každému požadavku přikládá *access token*, díky čemuž se identifikuje uživatel a daná služba pak vrací příslušná data.

2.3 Přístup k datům sociálních sítí

Pro sestrojení timeline příběhu je žádoucí nabídnout možnost importovat uživatelské příspěvky ze sociálních sítí. V následujících částech je popsáno, jak toho docílit v rámci konkrétní sociální sítě.

2.3.1 Facebook Graph API

Facebook Graph API [17] podporuje autentizační protokol OAuth 2.0. Z pohledu dat se budeme zajímat o uživatelské příspěvky umístěné na jeho zdi. Jedná se o veškeré statusy, odkazy, fotografie a videa. Tabulka 2.1 popisuje tyto uvažované zdroje.

Získat *read_stream* oprávnění ve schvalovacím procesu je prakticky nemožné. V dokumentaci je uvedeno [18], že toto oprávnění slouží pro „Facebook-branded“ aplikace běžící na platformě, kde Facebook momentálně nefiguruje. Tato možnost tedy odpadá. Zbývajícím řešením je importovat příspěvky typu status, fotografie, video s příslušnými oprávněními. Tímto ovšem přijdeme o odkazy. Na druhou stranu příspěvky budou proporciálně rozloženy, protože se bude dotazovat na více koncových bodů. I z tohoto důvodu bude vhodné se dotazovat dávkově. Zahrnout veškeré operace do jednoho požadavku a ten pak následně odeslat. Facebook pak tyto operace zpracuje paralelně a výsledná data vrátí v jedné odpovědi.

V době implementace byla dostupná verze Graph API 2.2. Později byla představena verze 2.3, která umožňuje využít oprávnění *user_posts*. Tímto lze získat všechny typy uživatelských příspěvků včetně odkazů. Veškeré změny provedené v Graph API lze dohledat v changelogu [19].

2.3.2 Twitter API

Twitter používá k zabezpečení svého API [20] autentizační protokol OAuth 1.0A. Uživatelské přispěvky lze získat z koncového bodu */statuses/user_timeline.json*. Příspěvek se jeví jako status, může ovšem obsahovat i dodatečná média. Prozatím Twitter poskytuje pouze fotografie.

2.3.3 Instagram API

Instagram nabízí poměrně jednoduché API [21], které je zabezpečeno pomocí autentizačního protokolu OAuth 2.0. Pro získání uživatelských příspěvků je určen koncový bod */users/self/media/recent*. K tomu vystačí výchozí oprávnění *basic* pro čtení. Příspěvkem může být fotografie nebo video.

2.3.4 Knihovny

Pro API sociálních sítí existují jak oficiální, tak i neoficiální knihovny třetích stran. Tyto knihovny většinou řeší autentizaci, přístup k datům, životnost *access tokenu* a další funkcionality. Za předpokladu, že v budoucnu se počet sociálních sítí v aplikaci může zvyšovat, byla zvolena jiná varianta. Autentizaci uživatelů oproti známým sociálním sítím bude zajišťovat knihovna HWIOAuthBundle [22]. Dotazování na API sociálních sítí bude probíhat pomocí HTTP klienta Guzzle [23]. Výhoda této varianty spočívá v tom, že odpadá nutnost neustále přidávat a nastudovávat nové knihovny.

Je potřeba zmínit, že knihovna HWIOAuthBundle se nestará o životnost *access tokenu*. Knihovna slouží k jednorázové autentizaci a získání *access tokenu*. Pokud se *access token* zneplatní, jsou dvě možnosti jak to rešit. Buď obnovit *access token* pomocí *refresh tokenu*, nebo absolvovat znovu prvotní komunikaci. Podle neoficiálních informací Instagram a Twitter životnost nenastavují. Facebook nastavuje platnost *access tokenu* na 60 dní [24]. S ohledem na tyto skutečnosti bylo zvoleno řešení, kdy před každým importem se nejprve otestuje spojení – a pokud se jedná o neplatný *access token*, bude nutné absolvovat znovu prvotní komunikaci.

Kapitola 3

Návrh

Návrh je důležitou součástí softwarového procesu. V této fázi procesu jsou zahrnuty poznatky z analýzy a obsah návrhu slouží pro pozdější implementaci. Tato kapitola nejdříve popisuje zvolenou architekturu aplikace, návrhový model tříd a návrh REST API. V dalších částech kapitoly je popsán způsob zpracování importu ze sociálních sítí a rozšiřitelnost architektury o novou sociální síť.

3.1 Architektura a technologie

Webové aplikace jsou v poslední době čím dál víc komplexní. Velká část logiky se přesouvá ze serverové části na klientskou s cílem nabídnout uživateli lepší prožitek a vyšší míru interaktivity. Je tedy nutné se zabývat architekturou jak v serverové části, tak i v klientské.

Jedním z návrhových rozhodnutí bylo použití přístupu multi-page aplikací. Tímto přístupem lze využít výhod jak tradičního přístupu, tak i přístupu single-page aplikací. Tradičním přístupem se rozumí klasické generování HTML obsahu, kdy při každém požadavku je překreslována celá stránka. Tradiční přístup má své uplatnění v případech, kdy je nutné zajistit indexovatelnost některých částí aplikace.[1] Naopak některé části aplikace pro indexaci tolik důležité nejsou a vyplatí se tyto části strukturovat jako mikro aplikace s využitím single-page přístupu [25]. Kombinací obou přístupů lze přispět k indexovatelnosti a interaktivnosti aplikace.

3.1.1 Serverová část

Při psaní serverové části je zcela běžné zvolit architektonický vzor Model-View-Controller. Díky tomu lze aplikaci rozdělit do nezávislých logických celků,

[1] Indexovatelnosti lze dosáhnout i v single-page aplikacích. Tímto ovšem roste složitost a nároky na aplikaci.

a tím zvýšit přehlednost a znovupoužitelnost kódu. Dalším zvoleným návrhovým rozhodnutím bylo použití servisně orientované architektury, která umožní strukturovat specifickou aplikační logiku do jednotlivých služeb. Této vlastnosti lze využít zejména při komunikaci s API sociálních sítích, kdy jednotlivé služby budou zapouzdřovat logiku pro danou sociální síť.

S ohledem na architekturu a předchozí zkušenosti byl jako implementační jazyk zvolen PHP spolu s frameworkem Symfony [26]. O databázovou vstvu aplikace se bude starat ORM framework Doctrine [27], který je založen na architektonickém návrhovém vzoru Data Mapper a umožňuje mapování entitních objektů do relační databáze. Databázovým systémem bylo zvoleno MySQL [28].

3.1.2 Klientská část

V klientské části se při psaní webových aplikací využívá programovací jazyk JavaScript, který je interpretován přímo v prohlížeči. Ačkoliv existují určité standardy jazyka, implementace JavaScriptu se ve webových prohlížečích přeci jen v několika detailech různí. I díky tomu vznikla knihovna jQuery [29], která poskytuje podporu pro majoritní prohlížeče. Knihovna jQuery exceluje zejména při práci s DOM elementy, událostmi nebo AJAX voláním. Avšak samotné použití knihovny jQuery při práci s komplexnějšími aplikacemi může přinést víc škody než úžitku. V této situaci se tedy vyplatí zabývat klientskou strukturou aplikace a jako jedním z řešení může být použití MVC frameworku.

S ohledem na absenci předešlých zkušeností s MVC frameworky v klientské části, byl zvolen framework Backbone.js [30] spolu s knihovnou RequireJS [31]. Touto kombinací lze docílit rozdělení klientské struktury aplikace do jednotlivých modulů. Díky RequireJS je pak možné tyto moduly načítat asynchronně a ve správném pořadí. Správnost pořadí je zaručena určením závislostí.

3.2 Návrhový model tříd

3.2.1 Uživatel (User)

Entita *User* reprezentuje uživatele aplikace. Entita je potomkem knihovní třídy *FOS\UserBundle\Model\User*, která obsahuje atributy nutné pro lokální přihlášení. U uživatele je kromě osobních údajů evidován identifikátor účtu a *access token* dané sociální sítě. Uživatel může být autorem více příběhů.

3.2.2 Příběh (Story)

Entita *Story* reprezentuje příběh. Každý příběh musí mít název a přirazeného autora. Příběh obsahuje kolekci příspěvků a kolekci příslušných importů. U příběhu je rozlišována viditelnost, která určuje komu je příběh zobrazen. Veřejný

příběh - všem uživatelům, chráněný příběh - uživatelům vlastnící odkaz, soukromý příběh - pouze autorovi. Při vytvoření příběhu je viditelnost nastavena ve výchozím stavu jako privátní. Metody *isPublic()*, *isProtected()*, *isPrivate()* pomáhají k rozlišení viditelnosti. Příběhu lze nastavit atribut *featured*, který slouží jako doporučení na základě výběru redakce.

3.2.3 Import příběhu (StoryImport)

Entita *StoryImport* reprezentuje import daného příběhu. Import obsahuje kolekci importovaných příspěvků dané sociální sítě. Při vytvoření je importu zpočátku přiřazen stav *Processing* a v závislosti na úspěšnosti je pak přiřazen stav *Successful* nebo *Failed*. U importu jsou evidovány hashtagy, podle kterých se importují příspěvky ze sociálních sítí, a chybová hláška *errorMessage*, která se nastaví v případě neúspěšného importu. Metody *isSuccessful()*, *isProcessing()*, *isFailed()* rozlišují stav importu.

3.2.4 Abstraktní příspěvek (Post)

Abstraktní rodičovská třída *Post* slouží jako společný perzistentní základ pro své potomky. Jelikož příspěvek daného příběhu může vzniknout z importovaného příspěvku – tudíž sdílí společné atributy i chování, jeví se abstraktní třída jako ideálním řešením. Příspěvek je v aplikaci rozlišen podle typu. Typem příspěvku může být:

Status - příspěvek obsahující text.

Link - příspěvek obsahující text i odkaz.

Image - příspěvek obsahující text i obrázek.

Video - příspěvek obsahující text i video.

Na základě typu příspěvku jsou nastaveny atributy *message* a *content*, který je typu *ContentStorage* a slouží k uchování meta-informací o obsahu příspěvku ve formě JSON. Dále příspěvek obsahuje atribut *location*, který slouží k uchování geoinformací příspěvku ve formě JSON. V neposlední řadě obsahuje příspěvek atributy *created* a *cursor*. Atribut *cursor* je použit při stránkování a má tvar řetězce `"{createdTimestamp}.{idPost}"`.

3.2.5 Importovaný příspěvek (BucketPost)

Entita *BucketPost* reprezentuje importovaný příspěvek z dané sociální sítě. Entita je potomkem abstraktní třídy *Post*. Importovaný příspěvek je přiřazen danému importu.

3.2.6 Příspěvek příběhu (StoryPost)

Entita *StoryPost* reprezentuje příspěvek daného příběhu. Entita je potomkem abstraktní třídy *Post*. Příspěvek je přirazen danému příběhu a oproti rodiči obsahuje navíc atributy *caption* a *canEdit*. Neperzistentní atribut *canEdit* určuje, zda je možné daný příspěvek editovat.

3.2.7 Abstraktní třída ContentStorage

Abstraktní třída *ContentStorage* poskykuje společný základ pro uložiště meta-informací o obsahu příspěvku. Obsahem příspěvku může být odkaz, fotografie nebo video. S ohledem na obsah příspěvku je uložiště rozlišeno podle typu: *Link*, *Image*, *Video* nebo *Album*.

Třída definuje abstraktní metody *toArray()* a *fromArray($data)*, které musí potomci implementovat. Tyto metody slouží k serializaci/deserializaci objektu do pole. Doctrine následně toto pole zkonvertuje do JSON řetězce, který je poté v databázi MySQL uložen do datového typu *LongText*. Je potřeba dodat, že klasická serializace/deserializace objektu pomocí PHP funkcí *serialize()/unserialize()* je nedostačující. Vzhledem k budoucímu vystavení REST API je potřeba zajistit přehledný a lidsky čitelný formát struktury. Proto každý potomek definuje svůj vlastní interní formát. Díky tomu lze spravovat veškerou logiku konzistentně na jednom místě.

3.2.8 Třída ImageContentStorage

Třída *ImageContentStorage* dědí z abstraktní třídy *ContentStorage*. Tato třída slouží pro uložení meta-informací obrázku. Třída nabízí možnost ukládat stejný obrázek podle typu rozlišení a poskytuje rozhraní pro přidávání či odebírání obrázku.

Listing 3.1: Ukázka uložené struktury ImageContentStorage ve formátu JSON

```
{
  "type": "image",
  "data": {
    "standard": {
      "type": "image",
      "width": 468,
      "height": 580,
      "resolution": "standard",
      "link": "https://example.com/example.jpg"
    },
    "thumbnail": {..}
  }
}
```

3.2.9 Třída VideoContentStorage

Třída *VideoContentStorage* dědí z abstraktní třídy *ContentStorage*. Třída je dost podobná třídě *ImageContentStorage* s tím rozdílem, že ukládá meta-informace videa.

Listing 3.2: Ukázka uložené struktury VideoContentStorage ve formátu JSON

```
{
  "type": "video"
  "data": {
    "standard": {
      "type": "video",
      "preview": {
        "type": "image",
        "width": 720,
        "height": 480,
        "resolution": "standard",
        "link": "https://example.com/example.jpg"
      },
      "width": 720,
      "height": 480,
      "embedHtml": "<iframe ..></iframe>",
      "resolution": "standard",
      "link": "https://example.com/example.mp4"
    }
  }
}
```

3.2.10 Třída LinkContentStorage

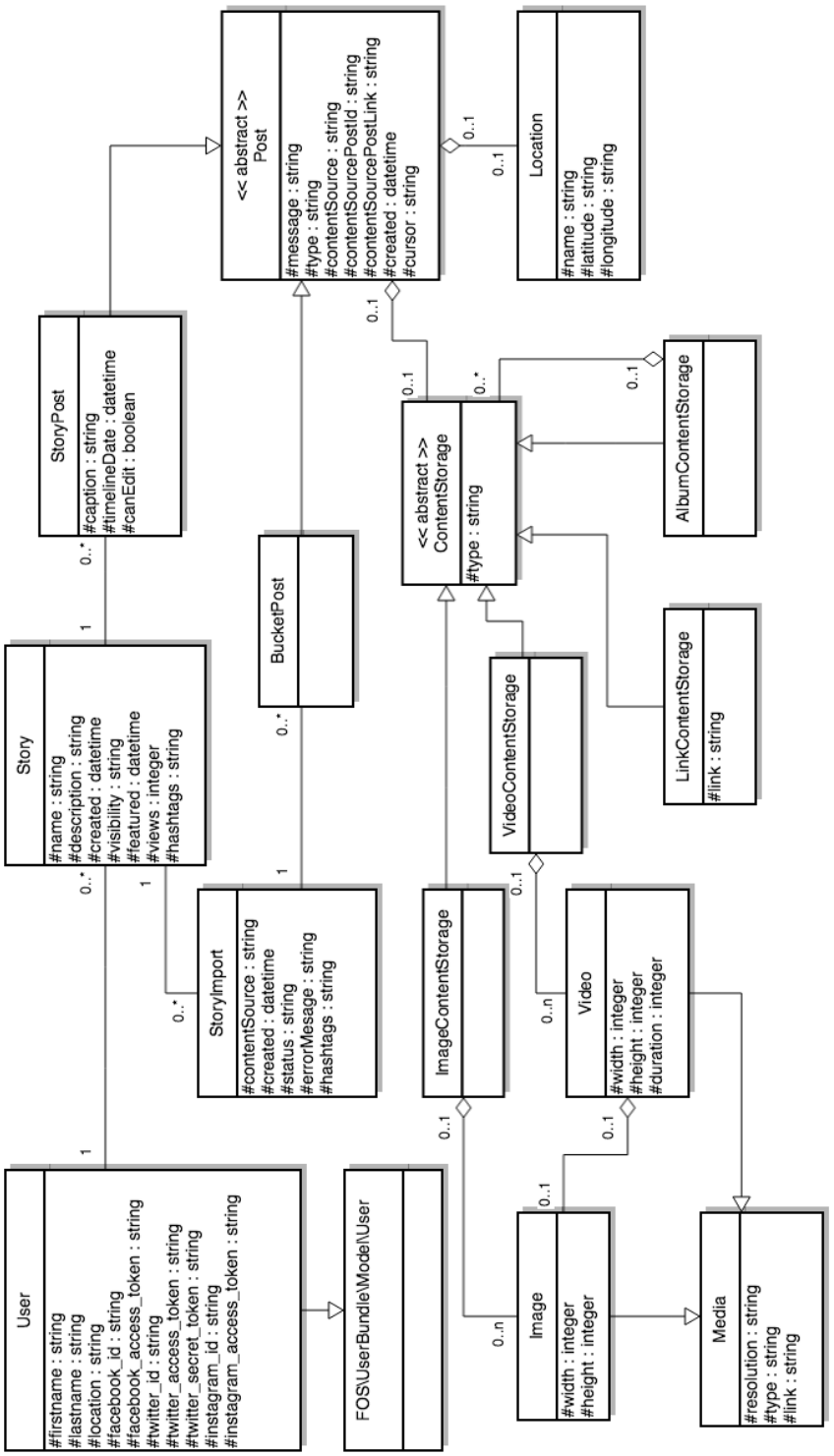
Třída *LinkContentStorage* dědí z abstraktní třídy *ContentStorage*. Třída obsahuje rozhraní pro manipulaci s odkazem.

3.2.11 Třída AlbumContentStorage

Třída *AlbumContentStorage* dědí z abstraktní třídy *ContentStorage*. Třída obsahuje kolekci typu *ContentStorage*. Díky tomu může album obsahovat různé typy uložišť: odkazy, obrázky, videa, a dokonce i „albumy uvnitř albumu“.

3.2.12 Abstraktní třída Media

Abstraktní třída Media představuje společný základ pro své potomky. Médium se rozlišuje podle typu a rozlišení. Typem média může být *Image* nebo *Video*. Dále jsou podporovány rozlišení typu *thumbnail*, *low*, *standard*. Atribut *link* určuje fyzické umístění daného média. Třída obsahuje metody *toArray()*, *fromArray($data)* určené k serializaci/deserializaci objektu do pole.



Obrázek 3.1: Diagram tříd

3.2.13 Třída Image

Třída *Image* dědí z abstraktní třídy *Media* a reprezentuje jednotlivý obrázek. Třída obsahuje atributy *width* a *height*, které popisují rozměry obrázku.

3.2.14 Třída Video

Třída *Video* dědí z abstraktní třídy *Media* a reprezentuje jednotlivé video. Třída obsahuje atributy pro rozměry videa. Dále obsahuje atribut *duration*, který specifikuje délku videa v sekundách, a atribut *embedHtml*, který obsahuje HTML embedovaného videa.

3.3 REST API

Komunikace mezi serverovou a klientskou části aplikace bude probíhat prostřednictvím REST API. Pomocí REST je možné v serverové části aplikace vystavit rozhraní, které bude úzce spolupracovat s datovým modelem aplikace a bude konzumováno webovým klientem. Pro výměnu dat bude výhradně použit formát JSON, ačkoliv je možné přistupovat k zdrojům i pomocí XML. Autentizace bude probíhat pouze základním způsobem pomocí *cookies*.[2]

Vzhledem k případům užití UC15 a UC16, zejména kvůli hromadnému mazání a přidávání, je nutné v REST API poskytnout dávkové operace. Dávkováním bude možné provést více operací v jednom požadavku, které by se jinak prováděly samostatně po jednom požadavku a vedlo by ke zpomalení doby provádění. K dávkování bude použita HTTP metoda *Patch*, která bude odkazovat na zdroj typu kolekce.

Při návrhu REST API je třeba dbát na intuitivní rozhraní a řídit se osvědčenými postupy [32]. Tabulky 3.1, 3.2, 3.3 a 3.4 popisují návrh REST rozhraní aplikace.[3] API je rovněž vhodné verzovat, proto bude v aplikaci použit URL prefix */api/v1*. K usnadnění psaní REST rozhraní slouží v Symfony balíček *FOSRestBundle* [33]. Veškerou logiku spojenou s daným REST zdrojem aplikace bude obsluhovat *controller* pomocí metod zvaných akcí. Ke každé metodě se uvádí anotace, které dodatečně specifikují chování metody. Většinou se v anotacích uvádí použitá HTTP metoda, relativní URL a volitelně parametry dotazu. Pro lepší manipulaci s entitami budou v *controlleru* využívány servisní třídy.

Při vracení dat bude použita obálka. Díky tomu je možné vracet kromě požadovaných dat i doplňující informace. Obálka bude reprezentována třídou *Envelope* a k požadovaným datům bude připojovat meta-informace popisující vrácenou odpověď.

[2] V budoucnu je v plánu použití SSL certifikátu spolu s OAuth2.

[3] Kompletní dokumentaci lze nalézt na přiloženém CD.

Listing 3.3: Ukázka vrácené odpovědi pomocí obálky

```
{
  "meta": {
    "http_status_code": 200,
    "http_status_message": "OK",
    "info_message": ""
  },
  "data": {..}
}
```

Tabulka 3.1: REST API zdroje - Příběh (Story)

Metoda	Zdroj	Popis
GET	/stories	Vrátí kolekci veřejných příběhů.
POST	/stories	Vytvoří nový příběh.
GET	/stories/featured	Vrátí kolekci veřejných příběhů na základě atributu *featured*.
GET	/stories/search/term	Vrátí kolekci veřejných příběhů odpovídajících hledanému termínu *term*.
GET	/stories/id	Vrátí daný příběh.
PUT	/stories/id	Provede změny daného příběhu.
DELETE	/stories/id	Smaže daný příběh.
GET	/stories/id/locations	Vrátí kolekci lokací v rámci daného příběhu.

Tabulka 3.2: REST API zdroje - Importovaný příspěvek (BucketPost)

Metoda	Zdroj	Popis
PATCH	/bucket_posts	Dávkové operace s importovanými příspěvky.
GET	/bucket_posts/id	Vrátí daný importovaný příspěvek.
DELETE	/bucket_posts/id	Smaže daný importovaný příspěvek.
GET	/imports/id/bucket_posts	Vrátí kolekci importovaných příspěvků daného importu.
GET	/stories/id/bucket_posts	Vrátí kolekci všech importovaných příspěvků daného příběhu.

Tabulka 3.3: REST API zdroje - Import (StoryImport)

Metoda	Zdroj	Popis
GET	/imports/id	Vrátí daný import.
DELETE	/imports/id	Smaže daný import.
GET	/stories/id/imports	Vrátí kolekci importů daného příběhu.

Tabulka 3.4: REST API zdroje - Příspěvek příběhu (StoryPost)

Metoda	Zdroj	Popis
PATCH	/posts	Dávkové operace s příspěvky příběhu.
GET	/posts/id	Vrátí daný příspěvek příběhu.
PUT	/posts/id	Provede změny daného příspěvku příběhu.
DELETE	/posts/id	Smaže daný příspěvek příběhu.
GET	/stories/id/posts	Vrátí kolekci příspěvků daného příběhu.
POST	/stories/id/posts	Vytvoří nový příspěvek daného příběhu.

3.3.1 Stránkování

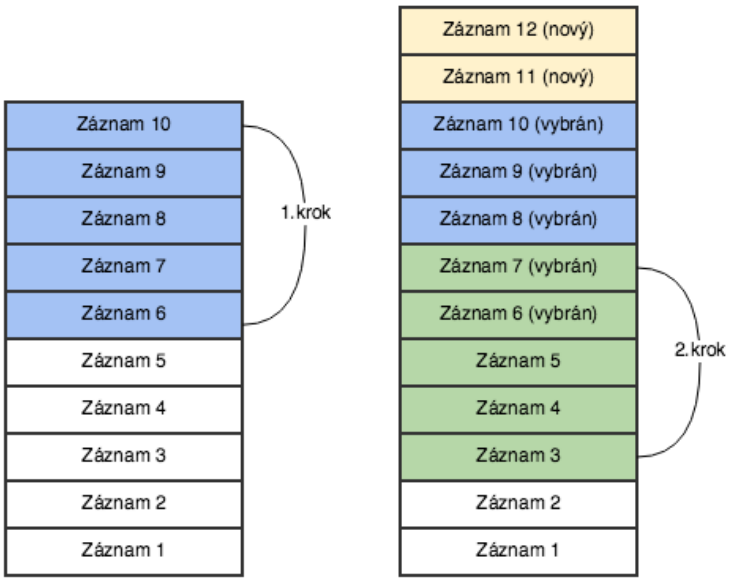
Při běžném stránkování REST API, kdy neočekáváme měnící se data, se využívá technika pomocí URL parametrů *limit* a *offset*. Pomocí parametru *limit* lze omezit počet zobrazovaných dat a pomocí parametru *offset* lze posunout začátek zobrazovaných dat. Parametr *offset* není nijak svázán s již načtenými daty, proto u často měnících se dat je tato technika nedostačující.

Jedním z problémů, které mohou nastat při běžném stránkování, je redundance dat. Uvažujme, že v prvním kroku vybereme posledních 5 záznamů. V druhém kroku budeme chtít vybrat taktéž 5 záznamů. Mezi prvním a druhým krokem však přibudou 2 nové záznamy. Tím pádem záznamy s pořadovým číslem 7 a 6 budou ve druhém kroku redundantní. Obrázek 3.2 znázorňuje redundanci při běžném stránkování. Mazání dat naopak při běžném stránkování může vést k nezobrazení určitých záznamů.

Řešením, jak stránkovat u často měnících se dat, je použití techniky *Cursor-based pagination* [34]. Tato technika je založena na kurzoru, který slouží k jednoznačné identifikaci záznamu. Při stránkování je tedy nutné si pamatovat kurzor posledního načteného záznamu. Při dalším načtení slouží tento kurzor jako počateční bod, od kterého lze načíst následující nebo předešlé záznamy. V terminologii REST k tomu budou sloužit parametry *after* a *before* doplněné o parametr *limit*.

Kandidátem na kurzor by mohl být atribut obsahující časovou informaci ve formě *timestamp*. Avšak vzhledem k tomu, že v aplikaci některé importované příspěvky mohou mít stejný *timestamp*, ztrácí tento atribut unikátnost. Řeše-

ním je tedy použití kurzoru ve tvaru řetězce `"{timestamp}.{id}"`. Záznamy se stejným *timestamp* budou rozlišeny podle uloženého *id*, čímž bude zajištěna unikátnost záznamu. Při tvorbě SQL dotazů pak bude nutné záznamy řadit podle kurzoru. Stránkování pomocí kurzoru tak odstraňuje uvedené problémy, které souvisí s běžným stránkováním.



Obrázek 3.2: Stránkování - redundance dat

3.4 Asynchronní zpracování

Asynchronní zpracování události se hodí na místě, kde mohou nastat blokující vstupní/výstupní operace. Událost importu příspěvků ze sociální sítě se dá považovat za blokující. Tato událost nejdříve čeká přes síť na požadovaná data a poté v případě potřeby tato data ukládá do databáze. V konečném důsledku může tato událost trvat v jednotkách sekund a z pohledu aplikace by bylo velmi neetické, kdyby se tím uživatel zbytečně zdržoval.

Vzhledem k nefunkčnímu požadavku N3. lze asynchronní zpracování řešit dvěma způsoby – použitím posixových threadů nebo posíláním zpráv pomocí message brokeru. PHP sice od verze 5.5 podporuje posixové thready, avšak dokumentace a množství příkladů z reálného světa je nedostačující. Navíc proč znovu vynalézat kolo, když asynchronicitu řeší message broker a v případě potřeby existuje i možnost škálování v distribuovaném prostředí.

Existuje celá řada message brokerů [35], které se věnují dané problematice. Při výběru byla zohledněna ustálenost, výstižná dokumentace, množství

reálných příkladů a podpora pro PHP spolu s frameworkem Symfony. Konečné rozhodnutí bylo učiněno ve prospěch message brokera RabbitMQ [36] s použitím Symfony balíčku RabbitMqBundle [37].

Z pohledu vývojáře se RabbitMQ skládá ze čtyř částí: *producer*, *exchange*, *queue* a *consumer*. Producent pouze zasílá vytvořené zprávy, které následně putují do exchange. V exchange jsou zprávy na základě routovacích pravidel umístěny do front typu FIFO. Konzument pak naslouchá a vyzvedává zprávy z front. Po vyzvednutí zprávy konzument tuto zprávu zpracuje a v případě úspěchu odešle potvrzení.

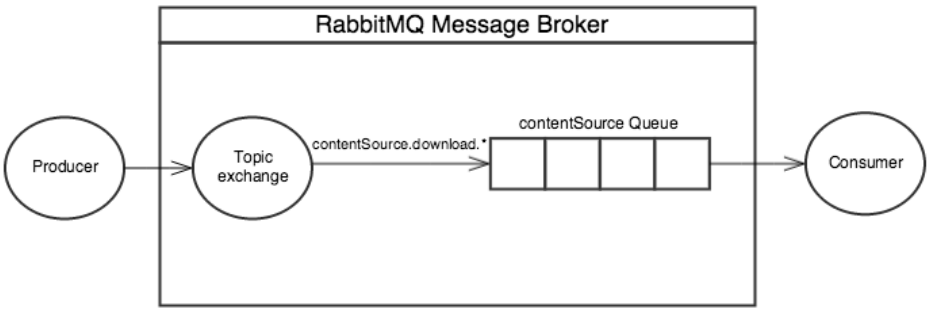
3.4.1 Asynchronní zpracování importu

Událost import příspěvků ze sociálních sítích bude vyvolána uživatelem po stisknutí tlačítka. Nejdříve je ovšem vhodné určit strategii, jak budou zprávy odesílany/konzumovány a která data budou obsahem. V exchange je tedy nutné určit, jak budou příchozí zprávy umístěny do front. Možnosti jsou následující:

- *Fanout* - broadcasting zprávy do všech front.
- *Direct* - umístění zprávy přímo do fronty podle *routing key*.
- *Headers* - umístění zprávy podle hlavičky.
- *Topic* - umístění zprávy do jedné nebo více front podle *routing key*.

Kvůli lepší flexibilitě bude nastaven typ exchange na *topic*. Zprávy, které budou určeny k odeslání, je vhodné rozlišovat podle sociální sítě. Na straně producenta bude nutné nastavit *routing key* ve formátu `"contentSource.download.{socialNetwork}"`, kde socialNetwork značí název sociální sítě. Na straně konzumenta je pak určitá volnost. Je možné naslouchat na vybrané fronty nebo naslouchat na všechny pomocí *routing key* `"contentSource.download.*"`. Návrh exchange je znázorněn obrázkem 3.3.

Producent bude v aplikaci pouze odesílat zprávy, proto bude reprezentován knihovní třídou *OldSound\RabbitMqBundle\RabbitMq\Producer*, která obsahuje metodu *publish()*. Tato metoda slouží k odeslání zprávy a jako argument přijímá obsah zprávy a daný *routing key*. Konzument bude v aplikaci reprezentován třídou *ContentSourceDownloadWorker*. Po vyzvednutí zprávy bude zavolána callback metoda *execute()*, která přečte obsah zprávy a následně tuto zprávu zpracuje. Jelikož zprávy budou putovat přes síť, je vhodné před samotným přenosem obsah zprávy serializovat. Na druhém konci před čtením je pochopitelně nutné tento obsah deserializovat a po úspěšném zpracování odeslat potvrzení.



Obrázek 3.3: RabbitMQ - topic exchange

3.5 Rozšiřitelnost architektury o novou sociální síť

Vzhledem k nefunkčnímu požadavku N2. je nutné, aby architektura aplikace umožňovala snadné přidání nové sociální sítě. Architektura v serverové části je z velké míry určena frameworkem Symfony, který podporuje servisně orientovanou architekturu. Díky tomu je možné jednotlivé třídy zajišťující komunikaci s API sociálních sítí deklarovat jako služby.

Společný základ bude poskytovat abstraktní servisní třída *ContentSourceService*. Tato třída definuje abstraktní metody *createClient($params)*, *testConnection()*, *getProfileInfo()*, *download()*, které musí potomci implementovat. Implementováním abstraktních metod a nastavením příslušné konfigurace lze snadno přidávat nové třídy, které budou zajišťovat komunikaci s danou sociální sítí.

Kapitola 4

Implementace

Při implementaci bylo vycházeno z návrhu a specifikace požadavků. Při samotném implementování byla dodržována navržená architektura a zažité konvence při psaní kódu. V této kapitole jsou popsány vybrané zajímavosti, které se vyskytly během vývoje.

4.1 Serverová část

4.1.1 Registrace a přihlašování uživatelů

Pro lokální registraci a přihlašování uživatelů je použita knihovna *FOSUserBundle* [38]. Pro externí autentizaci oproti sociálním sítím je využita knihovna *HWIOAuthBundle* [22]. V případě již existujícího účtu je nutné propojit lokální účet s externím. K tomu slouží třída *FOSUBUserProvider* obsahující metodu *connect()*, která tyto účty propojí. V případě ještě neexistujícího lokálního účtu, je možné s pomocí metody *loadUserByOAuthUserResponse()* vytvořit lokální účet na základě externího. Kombinací obou knihoven a jejich propojením je uživateli poskytnuta volnost při výběru způsobu přihlášení.

4.1.2 Event listeners

V aplikaci jsou obsaženy entitní *event listeners*, kteří naslouchají na určité události. V Doctrine tyto události zpravidla spouští *Event manager*. Při vyvolání události je pak zavolána registrovaná callback metoda příslušného posluchače.

Posluchači *BucketPostListener* a *StoryPostListener* obsahují metodu *postPersist()*, která je zavolána po uložení dané entity do databáze. Po uložení je pak možné získat hodnotu atributu *id* a nastavit kurzor kvůli stránkování. Posluchač *StoryPostListener* dále obsahuje metodu *postLoad()*, která nastaví atribut *canEdit* po načtení z databáze.

4.1.3 Formuláře

V aplikaci jsou vytvořeny speciální třídy pro práci s formuláři. Touto cestou lze částečně přesunout logiku, která by byla obsažena v controllerech, do samostatných tříd, a tím přispět k znovupoužitelnosti.

Pro práci s příběhem k tomu slouží formulář *StoryType*, k práci s příspěvky příběhu slouží formulář *StoryPostType*. Dále jsou v aplikaci obsaženy formuláře pro práci s importem ze sociálních sítí: *FacebookSourceType*, *TwitterSourceType*, *InstagramSourceType*. Tyto formuláře se od sebe příliš neliší. Ovšem za předpokladu, že v budoucnu bude nutné uživateli nabídnout volbu příspěvků k importování podle typu příspěvku, je vhodnější formuláře rozdělit do jednotlivých tříd.

4.1.4 Servisní třídy pro komunikaci s API sociálních sítích

Komunikaci s API sociálních sítích zajišťují jednotlivé servisní třídy, které dědí z abtraktní třídy *ContentSourceService*. Tato třída obsahuje kromě abstraktních metod[4] i atributy pro nastavení komunikačních limitů. Jedná se o nastavení celkového počtu iterovaných/importovaných příspěvků, počtu příspěvků v jednom požadavku a nastavení času vyhrazeného pro zpracování požadavku. Zmíněné limity pak lze přizpůsobit v potomcích.

Chování potomků *FacebookService*, *TwitterService* a *InstagramService* je velmi podobné. Každý potomek před komunikací nastaví parametry pro danou sociální síť a vytvoří HTTP klienta. Za požadované parametry lze označit koncový bod URL a uživatelův *access token*. Importování dat pak probíhá prostřednictvím metody *download()*. Tato metoda iterativně vytváří/odesílá požadavky na daný koncový bod a přijatá data následně při splnění podmínek ukládá do databáze. Importování dat končí v momentě překročení komunikačních limitů nebo při přijetí prázdných dat.

4.2 Klientská část

Klientská část je částečně určena serverovým návrhovým modelem. Jelikož klient konzumuje serverové REST API, jsou klientské modely podobné serverovým a liší se pouze v syntaxi. K tvorbě uživatelského rozhraní byl použit framework Bootstrap [39]. Zástupné obrázky renderuje knihovna Holder.js [40]. Při psaní HTML a CSS se dodržovalo oddělení struktury od prezentace.

4.2.1 Prezentační timeline příběhu

Prezentační timeline je z pohledu uživatele výsledek veškerého úsílí, které věnoval při sestavení příběhu. Timeline příběhu slouží pouze pro prezentační

[4] Abstraktní metody jsou uvedeny v sekci 3.5.

účely. Rozhodně nebylo cílem vytvořit komplexní timeline s velkou hromadou funkcionalit.

Prezentační formou byla zvolena vertikální timeline s jednosloupcovým layoutem. Vertikální timeline je tvořena příspěvky příběhu a je doprovázena svislou čárou. Tato čára je realizována pomocí CSS pseudo elementu *:before* a slouží spíš k estetickým účelům. Příspěvky jsou načítany po částech s využitím techniky nekonečného scrollování. Díky tomu je možné při dosažení konce stránky načíst AJAXově další část příspěvků. Při scrollování jednotlivých příspěvků jsou využity CSS animace. Prezentační timeline je znázorněna na obrázku 4.1.

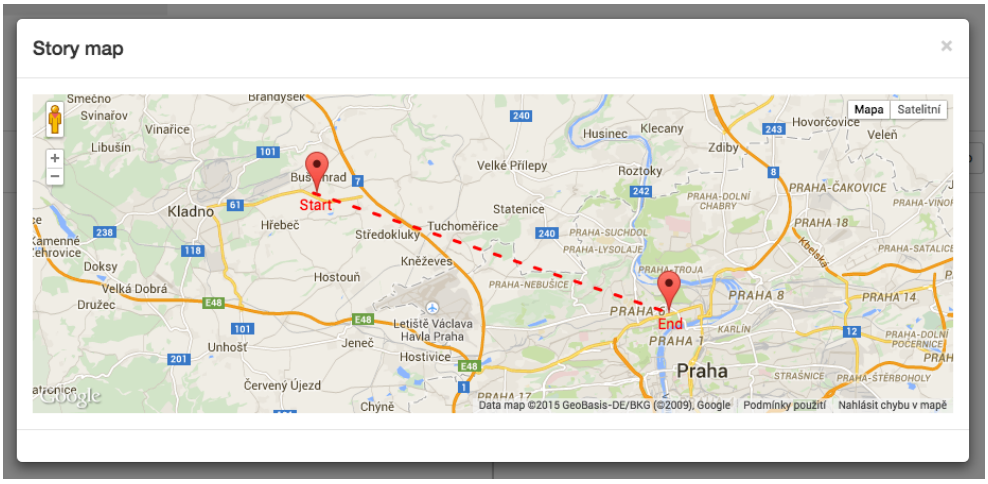
Obrázek 4.1: Ukázka prezentační timeline

4.2.2 Zobrazení mapy

Prezentace příběhu je doplněna o mapu příběhu. Mapa je tvořena z příspěvků obsahujících geolokační informace a zobrazena pomocí Google Maps [4]. K vy-

tvoření mapy jsou v aplikaci vyžadovány minimálně dvě lokace značící začátek a konec příběhu. Tyto lokace jsou na mapě označeny jako body pomocí markeru a pomocí labelů *Start* a *End*. Tyto body jsou pak doprovázeny přerušovanou čárou značící cestu příběhu.

Pro zobrazení mapy je nejdříve nutné zaregistrovat aplikaci na webu Google Console [41] a získat *apiKey*, který bude identifikovat aplikaci. Získání mapy je realizováno prostřednictvím Google Maps API [42]. K požadavku pak aplikace přikládá lokace pomocí zeměpisných souřadnic *latitude* a *longitude*. Vrácená mapa příběhu je poté v aplikaci zobrazena v modálním okně 4.2.



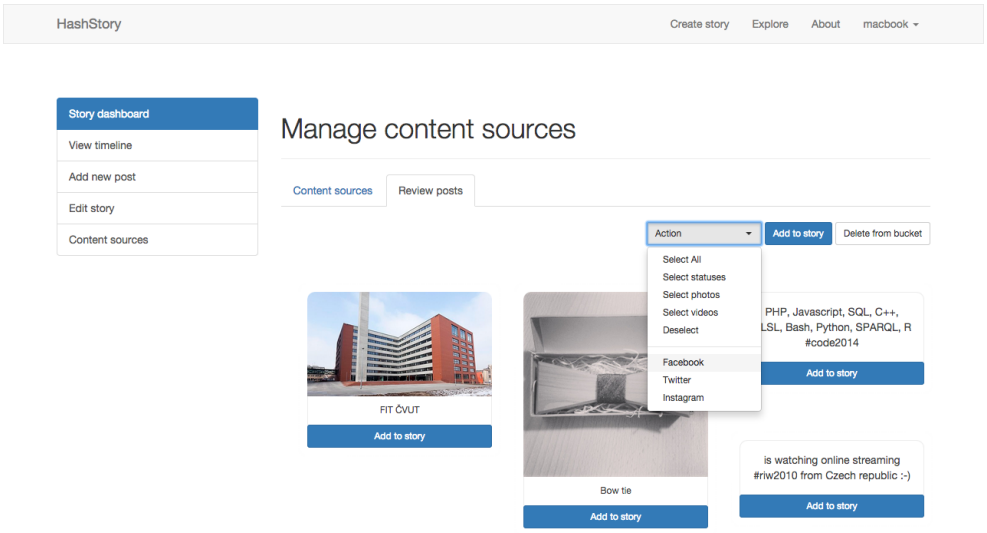
Obrázek 4.2: Ukázka mapy příběhu

4.2.3 Revidování příspěvků

Po úspěšném importu se importované příspěvky objeví v takzvaném *Bucketu*. Jelikož se může jednat o velký počet importovaných příspěvků, je vhodné před samotným přidáním do příběhu tyto příspěvky v *Bucketu* revidovat. K zobrazení importovaných příspěvků je použit třísloupcouvý layout. Jednotlivé příspěvky lze zobrazit v modálním okně jako náhled. Dále je možné tyto příspěvky vybírat podle typu příspěvku nebo sociální sítě, a následně mazat nebo přidávat do příběhu. Obrázek 4.3 ukazuje revidování importovaných příspěvků.

4.2.4 Prohlížení příběhů

Aktivních uživatelů je zpravidla méně než pasivních [43]. Pasivního uživatele je nutné zaujmout, aby se stal později aktivním. Aplikace proto umožňuje prohlí-



Obrázek 4.3: Ukázka revidování importovaných příspěvků

žet veřejné příběhy široké veřejnosti. Uživateli je umožněno listovat podle nedávných příběhů nebo podle příběhu označených jako *featured*. Uživatel může taktéž vyhledávat veřejné příběhy podle hledaného výrazu.

Kapitola 5

Testování

Testování softwaru je nedílnou součástí softwarového procesu. Testováním lze odhalit nedostatky a přispět k lepší kvalitě výsledné aplikace. V této kapitole je popsán způsob testování aplikace. Nejdříve byla aplikace otestována jednotkovými (Unit) testy a poté bylo otestováno REST API pomocí funkčních testů.

5.1 Unit testy

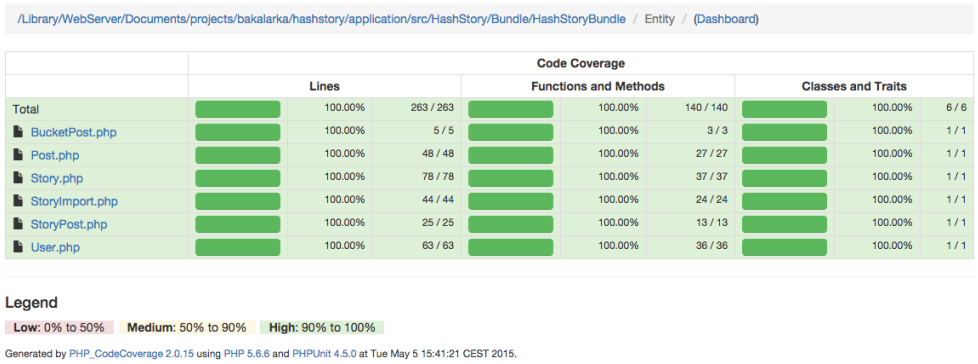
Jednotkové testy patří ke skupině testů, které provádí sám vývojář. Tyto testy jsou zaměřeny na samostatné jednotky programu – třídy, metody, proměnné. Testy jsou prováděny oproti veřejnému rozhraní tříd a zjišťuje se, zda jednotlivé vstupy/výstupy jednotek odpovídají očekávaným hodnotám či chování.

V aplikaci byly otestovány jednotlivé entity pomocí testovacího frameworku PHPUnit [44]. Pro každou entitu byly vytvořeny testovací třídy. Nestandartním případem byla abstraktní třída *Post*. K otestování této abstraktní třídy byl vytvořen potomek *TestablePost*. Vytvoření potomka bylo nutné z důvodu instancování třídy, kterou pak lze testovat běžným způsobem.

Při samotném testování entit bylo odhaleno pár nedostatků. Jednalo se o prázdné hodnoty parametrů funkcí a nesprávné určení inverzní strany v asociacích. Odhalené nedostatky byly následně odstraněny. Pokrytí kódu entit je znázorněno na obrázku 5.1. Z pokrytí kódu nelze dělat závěry o kvalitě testů, jedná se pouze míru udávající z kolika procent je kód skutečně pokryt testy.

5.2 Testování REST API

K testování REST API byly použity funkční testy. Pomocí funkčních testů lze otestovat celkové chování proti REST rozhraní. V Symfony pro tyto účely slouží třída *Symfony\Bundle\FrameworkBundle\Test\WebTestCase*, která je potomkem klasické *TestCase* třídy z PHPUnit. Z důvodu lepší testovatelnosti



Obrázek 5.1: Testování entit - pokrytí kódu

byla vytvořena abstraktní rodičovská třída *ApiV1BaseClass*. Tato třída nastavuje HTTP klienta, testovacího uživatele a obsahuje metody pro testování odpovědi. Potomci této třídy pak testují jednotlivé zdroje REST rozhraní.

V aplikaci byly provedeny pouze základní testy REST rozhraní. Tyto testy mají velmi podobný průběh. Nejdříve se v přípravné fázi vytvoří potřebná data, instancuje se HTTP klient a samotné testování pak probíhá formou požadavek – odpověď. V odpovědích se kromě vracených dat vždy kontroluje HTTP stavový kód a případně hlavičky obsahující *Content-Type* jako *application/json*. Během testování bylo zjištěno pár nedostatků. Jednalo se o prázdné hodnoty v parametrech požadavku. Dost závažným nedostatkem byla absence hlavičky *Location* po vytvoření nového zdroje metodou *Post*. Tuto hlavičku bylo nutné nastavit na URI nově vytvořeného zdroje. Uvedené nedostatky byly následně odstraněny.

Závěr

Cílem práce bylo vytvořit webovou aplikaci, která umožní uživatelům sociálních sítí sestrojit prezentační timeline příběhu z vlastních příspěvků. Zpočátku bylo nezbytně nutné provést rešerši existujících řešení. Tato rešerše ukázala objektivní pohled na danou problematiku a posloužila jako podklad pro další směřování aplikace. Poté byly specifikovány požadavky a případy užití kladené na aplikaci s přizpůsobením rozsahu práce. Na základě analýzy byl vytvořen návrh aplikace. V této fázi byla navržena architektura aplikace, návrhový model tříd a RESTové rozhraní. Dále byl navržen způsob, jak řešit asynchronní zpracování importu, a rozšiřitelnost architektury o novou sociální síť. Vytvořený návrh byl následně ověřen implementací a testováním. Výsledkem práce je prototyp aplikace, který poslouží pro budoucí rozvoj.

Tato práce mě obohatila o mnoho poznatků. Prozkoumal jsem API sociálních sítí a způsob autentizace přes OAuth. Rozšířil jsem své znalosti v Symfony a vyzkoušel MVC v klientské části. Poznal jsem způsob, jak lze časově náročné úlohy zpracovávat asynchronně pomocí message brokera, a prohloubil znalosti spojené s tvorbou REST rozhraní. Nabyté znalosti a zkušenosti považuji za přínosné.

V průběhu práce jsem přišel na řadu nápadů a vylepšení. V budoucnu bude nevyhnutelně nutné vylepšit zabezpečení REST API. V úvahu připadá použití SSL certifikátu spolu s autentizačním protokolem OAuth2. Dále bude nutné umožnit uživateli vyhledávat podle hashtagu, nahrávat fotografie a celkově pojmout prezentační timeline příběhu kreativněji. U tohoto typu projektu je totiž velmi tenká hranice mezi přijetím a odmítnutím uživatelem. Bude tedy dost záležet jak uživatele zaujme prezentační timeline a nabízený obsah.

Literatura

[1] GOOGLE INC.: Relive and share your adventures using Stories. [online], [cit. 2015-04-01]. Dostupné z: https://support.google.com/plus/answer/6029803?hl=en

[2] MADRIGAL, A. C.: How to Teach Google What a Story Is. [online], [cit. 2015-04-01]. Dostupné z: http://www.theatlantic.com/technology/archive/2014/07/how-does-google-teach-a-computer-to-tell-stories/373270/

[3] GOOGLE INC.: Google Now. [online], [cit. 2015-05-04]. Dostupné z: https://www.google.com/landing/now

[4] GOOGLE INC.: Mapy Google. [online], [cit. 2015-05-04]. Dostupné z: https://www.google.cz/maps

[5] TWITTER INC.: Twitter Search. [online], [cit. 2015-04-01]. Dostupné z: https://twitter.com/search-home

[6] STORIFY INC.: Storify. [online], [cit. 2015-04-02]. Dostupné z: https://storify.com

[7] REBELMOUSE INC.: RebelMouse. [online], [cit. 2015-04-02]. Dostupné z: https://www.rebelmouse.com

[8] SHARE A STORY: Share your story. [online], [cit. 2015-04-01]. Dostupné z: http://www.shareastory.org/submit-a-story

[9] GOOGLE INC.: Google Chrome. [software], [přístup 2015-05-04]. Dostupné z: https://www.google.cz/chrome/browser/desktop/

[10] MOZILLA CORPORATION: Mozilla Firefox. [software], [přístup 2015-05-04]. Dostupné z: https://www.mozilla.org/cs/firefox/desktop/

[11] OPERA SOFTWARE: Opera. [software], [přístup 2015-05-04]. Dostupné z: http://www.opera.com/cs

[12] MICROSOFT CORPORATION: Internet Explorer. [software], [přístup 2015-05-04]. Dostupné z: http://windows.microsoft.com/cs-cz/internet-explorer/download-ie

[13] MALÝ, M.: REST: architektura pro webové API. [online], [cit. 2015-05-04]. Dostupné z: http://www.zdrojak.cz/clanky/rest-architektura-pro-webove-api/

[14] HARDT, D.: The OAuth 2.0 Authorization Framework. [online], [cit. 2015-05-04]. Dostupné z: http://www.ietf.org/rfc/rfc6749.txt

[15] VITVAR, T.: Lecture 7: Security in REST. [online], [cit. 2015-05-04]. Dostupné z: http://humla.vitvar.com/slides/w20/lecture7.html

[16] WALLS, C.; DONALD, K.; BAIA, B.: Chapter 2. Service Providers. [online], [cit. 2015-05-04]. Dostupné z: http://www.springframework.net/social/refdoc/serviceproviders.html

[17] FACEBOOK INC.: The Graph API. [online], [cit. 2015-05-04]. Dostupné z: https://developers.facebook.com/docs/graph-api

[18] FACEBOOK INC.: Permissions with Facebook Login. [online], [cit. 2015-05-04]. Dostupné z: https://developers.facebook.com/docs/facebook-login/permissions/v2.3#reference-read_stream

[19] FACEBOOK INC.: Facebook Platform Changelog. [online], [cit. 2015-05-04]. Dostupné z: https://developers.facebook.com/docs/apps/changelog

[20] TWITTER INC.: REST APIs. [online], [cit. 2015-05-04]. Dostupné z: https://dev.twitter.com/rest/public

[21] INSTAGRAM INC.: Instagram Developer Documentation. [online], [cit. 2015-05-04]. Dostupné z: https://instagram.com/developer/

[22] HARDWARE.INFO: HWIOAuthBundle. [online], [cit. 2015-05-04]. Dostupné z: https://github.com/hwi/HWIOAuthBundle

[23] DOWLING, M.: Guzzle. [online], [cit. 2015-05-04]. Dostupné z: https://github.com/guzzle/guzzle

[24] FACEBOOK INC.: Access Tokens. [online], [cit. 2015-05-04]. Dostupné z: https://developers.facebook.com/docs/facebook-login/access-tokens#termtokens

[25] EDDY, B.: Backbone.js Organizational Patterns. [online], [cit. 2015-05-04]. Dostupné z: http://www.foraker.com/backbone-js-organizational-patterns/

[26] SENSIO LABS: Symfony2 Framework. [software], [přístup 2015-05-04]. Dostupné z: http://symfony.com

[27] DOCTRINE TEAM: Doctrine Project. [software], [přístup 2015-05-04]. Dostupné z: http://www.doctrine-project.org

[28] ORACLE CORPORATION: MySQL. [software], [přístup 2015-05-04]. Dostupné z: https://www.mysql.com

[29] JQUERY FOUNDATION: jQuery. [software], [přístup 2015-05-04]. Dostupné z: https://jquery.com

[30] ASHKENAS, J.: Backbone.js. [software], [přístup 2015-05-04]. Dostupné z: http://backbonejs.org

[31] BURKE, J.: Require.js. [software], [přístup 2015-05-04]. Dostupné z: http://requirejs.org

[32] SAHNI, V.: Best Practices for Designing a Pragmatic RESTful API. [online], [cit. 2015-05-04]. Dostupné z: http://www.vinaysahni.com/best-practices-for-a-pragmatic-restful-api

[33] FRIENDS OF SYMFONY: FOSRestBundle. [online], [cit. 2015-05-04]. Dostupné z: https://github.com/FriendsOfSymfony/FOSRestBundle

[34] NIMESH, R.: Paginating Real-Time Data with Cursor Based Pagination. [online], [cit. 2015-05-04]. Dostupné z: http://www.sitepoint.com/paginating-real-time-data-cursor-based-pagination

[35] STRZALKOWSKI, L.: Queues. [online], [cit. 2015-05-04]. Dostupné z: http://queues.io

[36] PIVOTAL SOFTWARE INC.: RabbitMQ. [software], [přístup 2015-05-04]. Dostupné z: https://www.rabbitmq.com

[37] VIDELA, A.: RabbitMqBundle. [online], [cit. 2015-05-04]. Dostupné z: https://github.com/videlalvaro/RabbitMqBundle

[38] FRIENDS OF SYMFONY: FOSUserBundle. [online], [cit. 2015-05-04]. Dostupné z: https://github.com/FriendsOfSymfony/FOSUserBundle

[39] OTTO, M.; THORNTON, J.: Bootstrap. [software], [přístup 2015-05-04]. Dostupné z: http://getbootstrap.com/

[40] MALOPINSKY, I.: Holder. [online], [cit. 2015-05-04]. Dostupné z: https://github.com/imsky/holder

[41] GOOGLE INC.: Google Developers Console. [online], [cit. 2015-05-04]. Dostupné z: https://console.developers.google.com

[42] GOOGLE INC.: Google Maps Javascript API v3. [online], [cit. 2015-05-04]. Dostupné z: https://developers.google.com/maps/documentation/javascript

[43] NIELSEN, J.: The 90-9-1 Rule for Participation Inequality in Social Media and Online Communities. [online], [cit. 2015-05-04]. Dostupné z: http://www.nngroup.com/articles/participation-inequality

[44] BERGMANN, S.: PHPUnit framework. [software], [přístup 2015-05-05]. Dostupné z: https://phpunit.de

Příloha **A**

Seznam použitých zkratek

AJAX Asynchronous JavaScript and XML

API Application Programming Interface

CSS Cascading Style Sheets

DOM Document Object Model

FIFO First In First Out

HTML HyperText Markup Language

HTTP HyperText Transfer Protocol

JSON JavaScript Object Notation

MVC Model-View-Controller

ORM Object-relational mapping

PHP PHP: Hypertext Preprocessor

REST Representational State Transfer

SQL Structured Query Language

SSL Secure Sockets Layer

URI Uniform Resource Identifier

URL Uniform Resource Locator

WYSIWYG What you see is what you get

XML Extensible markup language

Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `src`
  - `impl` ..... zdrojové kódy implementace
    - `install.md` ..... instalační příručka
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `BP_Marinko_Alexandr_2015.pdf` ..... text práce ve formátu PDF