FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Systém pro Wikimedia Česká republika
Student:	Ivo Kořínek
Vedoucí:	Ing. Jiří Hunka
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2023/2024

Pokyny pro vypracování

Cílem práce je tvorba vhodné open source aplikace pro tvorbu a hlavně následné zpracování dat z pořádaných akcí Wikimedia ČR, dále jen akcí. Výstupem pořádaných akcí je komplexní vyhodnocení v optimálním formátu pro pracovníky Wikimedia ČR.

Postupujte v těchto krocích:
1. Proveďte řádnou analýzu průběhu a vyhodnocení akcí včetně možností automatizace.
2. Na základě analýzy proveďte řádný návrh.
3. Dle návrhu implementujte výslednou aplikaci.
4. Navrhněte a následně aplikujte vhodné sady testů.
5. Zajistěte aby byla open source aplikace dobře přístupná budoucím vývojářům.
6. Zhodnoťte dosažené výsledky a navrhněte možná budoucí vylepšení.

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 3. února 2023 v Praze.

Bakalářská práce

SYSTÉM PRO WIKIMEDIA ČESKÁ REPUBLIKA

Ivo Kořínek

Fakulta informačních technologií
Katedra softwarového inženýrství
Vedoucí: Ing. Jiří Hunka
11. května 2023

České vysoké učení technické v Praze
Fakulta informačních technologií
© 2023 Ivo Kořínek. Všechna práva vyhrazena.
Tato práce vznikla jako školní dílo na Českém vysokém učení technickém v Praze, Fakultě informačních technologií. Práce je chráněna právními předpisy a mezinárodními úmluvami o právu autorském a právech souvisejících s právem autorským. K jejímu užití, s výjimkou bezúplatných zákonných licencí a nad rámec oprávnění uvedených v Prohlášení, je nezbytný souhlas autora.

Odkaz na tuto práci: Kořínek Ivo. *Systém pro Wikimedia Česká republika.* Bakalářská práce. České vysoké učení technické v Praze, Fakulta informačních technologií, 2023.

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů kódu

Chci poděkovat svému vedoucímu, Ing. Jiřímu Hunkovi, bez kterého by práce nemohla vzniknout a jež mi dal potřebnou volnost pro její dokončení, Martinu Urbancovi za pomoc s orientací v rozsáhlém světě Wikimedia a Mgr. Kláře Joklové za zpětnou vazbu a pomoc s testováním aplikace z uživatelského pohledu. Dále chci poděkovat svým rodičům za poskytnutí péče a střechy nad hlavou, našim kočkám za poskytnutí společnosti a nepřevrhování sklenic do klávesnic, a přátelům z blízka i daleka za toleranci mého velmi vystresovaného já v procesu psání této práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při 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. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisů, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené.

V Praze dne 11. května 2023 .....

Abstrakt

Práce se zabývá návrhem a implementací softwarového systému pro správu událostí pořádaných spolkem Wikimedia Česká republika. Ten také získává data o dopadu těchto událostí na obsah spravovaný nadací Wikimedia Foundation.

Výsledkem práce je základ systému, který využívá aplikační rozhraní projektů Wikimedia pro sběr dat o editacích stránek provedených v rámci spolkem pořádaných událostí. Pro správu těchto událostí a zobrazení agregovaných metrik je použitá webová aplikace.

V práci vytvářený systém může již nyní částečně nahradit aktuálně používané zastaralé postupy sběru dat, a po jeho dokončení tyto postupy nahradí úplně, čímž sníží zátěž pracovníků spolku.

Klíčová slova Wikimedia, sběr metrik, Spring Framework, vícevrstvá architektura, klient-server

Abstract

The work concerns the design and implementation of a software system for managing events organized by the Wikimedia Czech Republic association. It also acquires data regarding the impact of these events on content managed by the Wikimedia Foundation.

The result of this work is the foundation of a system which uses Wikimedia projects' application interface to gather data about page edits made at events organized by the association. A web application is used for managing the events, as well as displaying the aggregate metrics.

The system created in this work can already partially replace the obsolete processes used presently, and upon completion will replace them entirely, reducing the workload of the association's employees.

Keywords Wikimedia, gathering metrics, Spring Framework, multilayer architecture, client-server

Seznam zkratek

API	Application Programming Interface
GLAM	Galleries, Libraries, Archives and Museums
HTTP	HyperText Transfer Protocol
JPA	Jakarta Persistence API
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
MVC	Model-View-Controller
npm	Node Package Manager
REST	Representational State Transfer
SPA	Single-page Application
URL	Uniform Resource Locator
WM	Wikimedia
WMČR	Wikimedia Česká republika

Úvod

Spolek Wikimedia Česká republika (dále WMČR) je neziskovou organizací s dobrovolnickou členskou základnou, jejíž účelem je zpřístupňování informací široké veřejnosti skrze projekty spravované celosvětovou nadací Wikimedia Foundation. V rámci své činnosti pořádá WMČR různé akce, výzvy, a soutěže (dále jen „události“), většinou právě za účelem rozšíření obsahu přístupného skrze projekty Wikimedia.

Spolek pro své potřeby sbírá metriky týkající se dopadu pořádaných událostí, ale proces sběru těchto dat je aktuálně pouze evolucí procesu vzniklého při samém počátku spolku, a s rostoucí aktivitou organizace se stává pro její pracovníky čím dál více náročným. Proto vyvstala potřeba po novém systému správy událostí, jež by většinu tohoto procesu zautomatizoval a poskytl jednotné prostředí pro správu těchto dat, a tím urychlil, zjednodušil a zpřesnil proces agregace těchto metrik.

Výsledný systém má potenciál výrazně ulehčit práci pracovníkům WMČR, a v případě úspěchu být rozšířen mezi další pobočky Wikimedia po celém světě.

Cílem práce je analyzovat aktuální proces sběru metrik a možnosti jeho automatizace, na základě požadavků vyvstalých z této analýzy navrhnout vhodný systém, ten následně implementovat, řádně otestovat, a v závěru výsledek zhodnotit a navrhnout další možnosti rozvoje.

Práce samotná je rozdělena do částí podle jednotlivých kroků vývoje: první částí je analýza současného procesu sběru metrik ve spolku a průzkum možností jeho automatizace. Druhá část pojednává o návrhu samotného systému na základě analýzy z první části. Třetí část popisuje samotnou implementaci systému na základě návrhu. Čtvrtá část popisuje proces testování systému, a pátá část zhodnocuje výsledky práce a možnosti jejího budoucího uplatnění a rozvoje.

Kapitola 1
Analýza

Tato kapitola se zabývá průzkumem prostředí projektů Wikimedia, analýzou aktuálně zavedených procesů pořádání událostí a sběru metrik ve spolku Wikimedia Česká republika včetně možnosti automatizace těchto procesů, a formulováním požadavků pro výsledný systém.

1.1 Wikimedia Foundation a její ekosystém

Wikimedia Foundation je nadace, jejíž cílem je sbírat, rozvíjet, a zpřístupňovat vzdělávací obsah [1]. Toho dosahuje mj. i pomocí práce dobrovolníků v místních pobočkách (mezi něž patří i Wikimedia Česká republika), které jsou z právního hlediska entitami nezávislými na Wikimedia Foundation [2].

Wikimedia Foundation pro plnění své mise provozuje rozličné projekty. Mezi nejznámější patří Wikipedie, otevřená encyklopedie, ale existuje i mnoho dalších[1]. Většina těchto projektů je dostupná ve velkém množství jazykových variant, které jsou z technického hlediska od sebe oddělené[2]. Mezi výjimky patří mj. Wikimedia Commons, sloužící jako úložiště souburů, obzvláště pak obrázků a fotek[3], a Wikidata, sloužící k organizaci velkého množství dat v podobě, která je čitelná strojově i lidsky; oba tyto projekty nemají jazykové varianty a jsou společné pro celý svět.

Všechny projekty Wikimedia Foundation jsou hostovány skrze MediaWiki, otevřenou volně dostupnou softwarovou platformu vyvinutou nadací [3].

1.2 Současný stav ve Wikimedia Česká republika

Tato sekce popisuje současný proces pořádání událostí a agregace dat z nich v rámci spolku Wikimedia Česká republika.

1.2.1 Pořádané události

Události pořádané WMČR se pro účely práce dají shrnout do dvou základních kategorií:

[1] Seznam všech projektů Wikimedia Foundation je dostupný zde:
`https://meta.wikimedia.org/wiki/Complete_list_of_Wikimedia_projects`

[2] Například česká Wikipedie je jiný projekt, než Wikipedie anglická; toto bude později důležité.

[3] Tyto jsou pak používány napříč projekty; všechny obrázky na Wikipedii jsou ve skutečnosti uloženy právě na Wikimedia Commons.

Události krátkodobé, které se konají na jednom místě[4] v jeden čas, s účastníky registrovanými dopředu. Příkladem takové akce je „Editaton Sucho 2022“, který se odehrával 31. 5. 2022 v Bubenči a jenž se týkal mj. úprav článků o pitné vodě a jejím spoření [4].

Události dlouhodobé, které mají zpravidla delší dobu trvání a účastníci se neregistrují dopředu. Místo společného setkání, jako u krátkodobých událostí, mají události dlouhodobé většinou formu výzvy či soutěže — účelem události je vytvořit či upravit co nejvíce článků o určitém tématu. Účastníci své editace označují dohodnutým hashtagem, za pomoci nějž se dá identifikovat dopad události.

Příkladem takové akce je soutěž „Československo 1948–1989“. Ta se odehrála od 28. 10. do 18. 12. 2022, týkala se článků o daném období a souvisejících tématech, a účastníci své editace označovali hashtagem `#soutezceskoslovensko2022` [5].

Krátkodobé události jsou často součástí událostí dlouhodobých; v rámci výše zmíněné soutěže WMČR pořádala vícero krátkodobých událostí, jejichž náplň se se soutěží překrývala. Toto bude muset být zohledněno při agregaci metrik.

1.2.2 Sběr metrik

1.2.2.1 Dlouhodobé události

U dlouhodobých událostí (viz sekce 1.2.1) je sběr metrik relativně přímočarý. Dohodnutý hashtag je vyhledán skrze nástroj „Wikimedia hashtag search[5]“, obecně známý jako Hashtags. Hashtags umožňuje mj. vyhledávání podle konkrétního hesla, projektu, a časového rozsahu v řádu dní.

Poté většinou dochází k následnému dočištění skrze nástroj Outreach Dashboard[6], kam účastníci mohou přidat své úpravy pokud hashtagu nevyužili, a výsledná data z obou nástrojů se poté ručně kobinují dohromady.

1.2.2.2 Krátkodobé události

Krátkodobé události bývají rozmanitější a ze své podstaty se tedy různí i způsob sběru dat. Nejčastější scénář však vypadá přibližně následovně:

Pracovník vytvoří Formulář Google, přes který se účastníci registrují. Po skončení události pracovník sesbírá data z formuláře, přepíše je do nástroje dle svého výběru (nejčastěji Event Metrics[7] či Outreach Dashboard[6], ale i jiné), ze kterého následně metriky sesbírá.

1.2.2.3 Agregace

Takto sesbíraná data se následně přepisují do Tabulek Google, jichž spolek hojně využívá. Je spíše pravidlem než výjimkou, že data z jedné události je potřeba postupně přepsat do několika různých tabulek, než jsou výsledná data agregována do konečné autoritativní tabulky metrik. Vzorce jsou v rámci tabulek používány spíše rudimentárním způsobem a mnohokrát dochází k ručnímu kopírování či přepisu, ne ojediněle i s použitím tužky a papíru v rámci procesu, což výrazně zvyšuje riziko zavedení chyby do dat.

[4] Místo nemusí být nutně konkrétní fyzická lokace; tyto akce se, obzvláště během pandemie, konají i formou dálkovou. Toto je však spíše výjimka než pravidlo, a stále platí jednorázovost a krátká doba trvání dané události.

[5] Dostupný na `https://hashtags.wmcloud.org/`

[6] Dostupná na `https://outreachdashboard.wmflabs.org`

[7] Dostupná na `https://eventmetrics.wmflabs.org/`

1.3 Systémy Wikimedia

Tato sekce se věnuje problémům vyvstalým během analýzy systémů Wikimedia, které ovlivňují práci, a jejich možným řešením.

1.3.1 Identifikace uživatel

Jedním problémem pro systém je jednoznačná identifikace jednoho uživatele napříč projekty. Jednotlivé projekty mají vlastní číselné ID pro jednotlivé uživatele, které není nijak koordinované napříč projekty, tj. znalost ID uživatele v rámci jednoho projektu neposkytuje žádné užitečné informace o jeho ID v rámci ostatních projektů. Historicky byly jednotlivé projekty absolutně oddělené, a jednotlivé uživatelské účty nebyly nijak synchronizovány[8], což prakticky znemožňovalo jakoukoli globální koordinaci [6, 7].

Toto bylo v letech 2014–15 řešeno zavedením jednotného loginu a koordinace uživatelských jmen napříč projekty, což však problém řeší zdánlivě pouze částečně, vzhledem k tomu, že uživatelské jméno, byť identifikátorem jednoznačným, není neměnné. Centrální autentifikační server CentralAuth však každému uživateli přiřazuje unikátní neměnné globální ID, které lze získat za pomoci uživatelského jména (a jméno za pomoci globální ID) API dotazem na kterýkoli projekt skrze parametr `globaluserinfo` [8].

1.3.2 Hashtagy

Projekty WM prakticky nemají žádnou přímou podporu specificky pro hashtagy. Při editaci stránek mohou uživatelé vyplnit shrnutí, a do něho často hashtagy umisťují, ale není možno vyhledávat ani podle textu shrnutí editace, natož podle hashtagů v něm obsažených. Protože je ale mnohdy žádoucí podle hashtagů vyhledávat (mj. i pro účely této práce), vznikl nástroj *Hashtags*[9] (zmíněný i v sekci 1.2.2.1). Hashtags monitoruje editace na všech projektech s výjimkou Wikidat[10] a umožňuje vyhledávání. [10]

Hashtags využívá streamu `recentchanges`[11], který ukazuje všechny změny v daném projektu za posledních 30 dní. V tomto streamu Hashtags vyhledává editace, které ve svém shrnutí obsahují validní hashtag, a ukládá si je do vlastní databáze, ve které pak vyhledává. [10]

Dokumentace tohoto nástroje není podrobná [11] a nezmiňuje se o možnosti používání nástroje automatizovaně. V issue tracking systému se však dá dohledat, že existuje API, které umí zpracovat stejné dotazy, jako uživatelské rozhraní, s výsledkem ve strojově zpracovatelné podobě [11].

Nástroj je hojně využívaný, relativně dobře udržovaný, obsahuje již několikaletou prakticky nereplikovatelnou historii záznamů a má strojově čitelné API, nemá tedy smysl vyvíjet vlastní řešení.

[8] Včetně například možnosti toho, aby různí uživatelé měli stejné jméno, pokud byli registrováni v různých projektech, či aby jeden uživatel měl v různých projektech jiná jména; obojí bylo relativně častým jevem [6].

[9] Dostupný na `https://hashtags.wmcloud.org/`.

[10] Editace na Wikidatech obsahují přibližně o jeden až dva řády více hashtagů než všechny ostatní projekty WM dohromady a většina jich je strojových a dohledatelných jinými způsoby [9].

[11] Dostupný i pro běžné uživatele pro každý projekt pod adresou `/wiki/Special:RecentChanges`

1.4 Požadavky řešení

Požadavky a jejich sběr jsou jednou z nejdůležitějších částí raných fází softwarového vývoje. Určují trajektorii výsledného systému a jejich kvalita výrazně ovlivňuje jeho úspěšnost. Za požadavek se dá požadovat jakékoli tvrzení o cílích, kterých má výsledný systém dosahovat, a podmínek pro dosažení těchto cílů, či o vlastnostech, které výslednému systému přináší hodnotu. [12]

Mimo jiné požadavky fungují jako „most“ mezi vývojáři a výslednými uživateli systému, a zajišťují tím společnou představu o vlastnostech výsledného systému. Správný proces sběru požadavků zahrnuje dialog mezi oběma stranami.[12]

Správné požadavky by měly[12] být [12]:

Kompletní — požadavky by měly pokrývat veškerou funkcionalitu výsledného produktu.

Korektní — požadavky by měly správně a přesně popisovat budovanou funkcionalitu[13].

Proveditelné — požadavky by mělo být možno uskutečnit, a to jak samy o sobě, tak v rámci celého systému požadavků, a toto uskutečnění by nemělo být neúnosně náročné.

Nutné — požadavky by měly popisovat pouze skutečně potřebnou funkcionalitu.

Prioritizované — požadavky by měly mít určenou prioritu relativně k ostatním požadavkům. Toto umožní větší flexibilitu a schopnost upravovat specifikaci v případě neočekávaných dějů.

Jednoznačné — požadavek by měl být sepsán s jediným možným významem, ale zároveň být jednoduše pochopitelný; případné technické výrazy nebo možné nejednoznačné termíny by měly být dovysvětleny.

Ověřitelné — splnění požadavku by mělo být možné ověřit. Bez ověřitelnosti se určení správnosti implementace mění z objektivní analýzy na subjektivní názor. [12]

Tyto vlastnosti byly zohledněny při tvorbě požadavků, jejichž výčet se nachází níže v této sekci. Pro zjednodušení čtení se pro korektnost, proveditelnost, nutnost, jednoznačnost a ověřitelnost u jednotlivých požadavků předpokládá zřejmost těchto vlastností. Prioritizovanost je určena dvěma číselnými ukazateli: prioritou, kde vyšší číslo značí vyšší prioritu, a odhadovanou implementační náročností, kde vyšší číslo značí vyšší náročnost. Volba čísel je zdůvodněna komentářem.

V práci jsou dále požadavky děleny na **funkční** a **nefunkční**:

Funkční požadavky specifikují funkcionalitu, kterou musí systém vykazovat, aby ho jeho uživatelé mohli využít ke správnému plnění svých úkolů. Tyto požadavky popisují pozorovatelné chování systému, podmínky, pod kterými systém toto chování projevuje, a akce, které systém jeho uživatelům umožní. [12]

Nefunkční požadavky popisují vlastnosti a charakteristiky, které má systém vykazovat, nebo omezení, která musí dodržovat, která ale nejsou pozorovatelným chováním systému (tj. funkčním požadavkem). Nefunkční požadavek se tedy může týkat například výkonu systému, jeho dostupnosti, spolehlivosti, či hardwarové náročnosti, ale i jiných vlastností. [12]

Požadavky systému pro účely práce jsou vypsány v následujících dvou podsekcích.

[12] Tato část využívá podmiňovací „měly by“ místo imperativnějších form textu ze dvou důvodů; prvním je ten, že názory ohledně vlastností správného požadavku se různí; druhým, důležitějším, je prostý fakt, že ve většině případů nelze zajistit všechny vypsané vlastnosti najednou, což ale neznamená, že je daný požadavek špatný. [12]

[13] Posouzení této „správnosti“ nemusí být vždy jasné, ale většinou vychází ze zdrojů daného požadavku: proč je požadavek zahrnut? Jaký je jeho účel?

1.4.1 Funkční požadavky řešení

1.4.1.1 F1 — Správa uživatel

Systém bude evidovat uživatele WM projektů, kteří se účastnili událostí. Uživatele bude možno vytvářet (na základě existujících účtů v rámci projektů WM) a upravovat jejich údaje.

Priorita: 3 — základní funkcionalita systému, bez které jeho vývoj nemá smysl.
Náročnost: 2 — je potřeba sbírat data o uživatelských účtech z projektů WM a zajišťovat jejich validitu.

1.4.1.2 F2 — Kategorizace uživatel

Uživatele je potřeba různě kategorizovat. Pro účely agregace je například důležité, zda je daný uživatel „nováček“ (tj. byl nově zaregistrován na nějaké události), pohlaví, apod. [13]

Místo zanesení těchto informací přímo do modelu bylo rozhodnuto zavést pro uživatele tagovací systém. Jednotliví uživatelé budou moci být přiřazeni k různým tagům v závislosti na tom, do jaké kategorie se řadí. Tagy mají pro jednodušší práci stromovou[14] strukturu (např. tagy pro jednotlivé věkové skupiny mohou být „podtagem“ tagu pojmenovaného „Věk“).

Výhodou tagů je jednoznačně flexibilita — pro případ zavedení nové kategorie stačí vytvořit nový tag a není tedy třeba žádného zásahu do vnitřního fungování systému. Tagy bude možno vytvářet, přiřazovat a odebírat jim uživatele a upravovat jejich jméno a místo ve stromové struktuře (přiřazovat a odebírat „nadtag“ či „podtagy“).

Priorita: 2 — bez splnění požadavku je systém výrazně ochuzen, ale v případě drastického skluzu lze implementaci pozdržet.
Náročnost: 2 — Je potřeba zajistit validitu stromové struktury, obzvláště pak zamezit vzniku cyklů.

1.4.1.3 F3 — Správa událostí

Systém bude evidovat události, kterých se uživatelé účastní či účastnili. Mezi evidované údaje o události patří počáteční a koncové datum, projekty, jichž se událost týká, a její účastníci. Události bude možno vytvářet a měnit výše zmíněné údaje.

Pro události krátkodobé (viz sekce 1.2.1) budou účastníci a údaje o nich vkládány do systému ručně, pro události dlouhodobé budou sbírány automaticky s možností ručního dočištění.

Priorita: 2–3 — alespoň částečné splnění (samotná tvorba událostí a alespoň jedna forma přiřazení uživatel) tvoří základní funkcionalitu systému, bez které jeho vývoj nemá smysl. Kompletní splnění má též vysokou prioritu a systém je bez něj výrazně ochuzen, ale v případě drastického skluzu lze implementaci krátkodobě pozdržet.
Náročnost: 1–3 — události s ručním vkládáním uživatel jsou relativně nenáročné na implementaci, automatický sběr potřebuje komunikaci s vnějšími servery a zajištění validity dat.

[14] Graf tagů ve skutečnosti tvoří les, protože může mít libovolné množství kořenů; v práci se přesto struktura nazývá „stromovou“ vzhledem k jednoduššímu pochopení textu.

1.4.1.4 F4 — Kategorizace událostí

Podobně jako uživatele je třeba kategorizovat i události. Wikimedia ČR mj. rozlišuje mezi programy vzdělávacími, komunitními a partnerskými [13]. Tagovací systém bude tedy využit i pro události, i s podobnou stromovou strukturou, jednotlivé tagy však budou striktně odděleny od těch uživatelských[15]. Možnosti práce s tagy událostí odpovídají tagům uživatelským.

Priorita: 2 — bez splnění požadavku je systém výrazně ochuzen, ale v případě drastického skluzu lze implementaci pozdržet.
Náročnost: 2 — Je potřeba zajistit validitu stromové struktury, obzvláště pak zamezit vzniku cyklů.

1.4.1.5 F5 — Správa projektů

Systém bude evidovat projekty WM, kterých se týkají jednotlivé události. Mezi evidované údaje patří název a webová adresa projektu. Projekty bude možno zavádět do systému.

Priorita: 3 — základní funkcionalita systému, bez které jeho vývoj nemá smysl.
Náročnost: 1 — pravděpodobně samostatně stojící modelová třída a datové operace s ní.

1.4.1.6 F6 — Agregace metrik

Na základě informací sesbíraných dle požadavků **F1–4** bude agregovat a zobrazovat metriky o jednotlivých akcích, skupinách akcí na základě tagů, a obdobně účastníků těchto akcí. Mezi agregované metriky bude mj. patřit počet účastníků, a počet vytvořených a upravených stránek, s možností dělení i dle projektu.

Priorita: 3 — základní funkcionalita systému, bez které jeho vývoj nemá smysl.
Náročnost: 3 — je potřeba komunikovat s vnějšími servery, pravděpodobně pomocí API, získat data a zpracovat je do užitečné podoby. Je možné, že pouze API nebude postačovat, a bude potřeba přistoupit k replikám databází projektů WM či hromadných výpisů jejich dat.

1.4.1.7 F7 — Role pracovníků

Pracovníci budou rozřazeni do rolí s různými pravomocemi, týkajícími se možností provádět ostatní akce v rámci systému. Pravomoce bude možno konfigurovat.
Priorita: 1 — splnění funkcionality není urgentní pro první verzi, ale dlouhodobě bez ní nemá systém smysl.
Náročnost: 2–3 — Záleží na požadované granularitě pravomocí; v případě vysoké granularity může být náročné implementovat dostatečně modulární systém pravomocí.

1.4.2 Nefunkční požadavky řešení

1.4.2.1 N1 — Přístup k systému

K systému bude možno přistupovat formou webové aplikace. Tato aplikace bude optimalizována pro použití na stolním počítači a předpokládá použití i osobami bez velkých technických znalostí.

Priorita: 3 — bez možnosti uživatelsky přívětivého přístupu k systému nemá jeho vývoj smysl; nelze bez něj mj. zajistit uživatelské testování.
Náročnost: 4 — Je potřeba vyvinout celou webovou aplikaci.

[15] Nelze tedy přiřadit uživatelský tag události a naopak.

1.4.2.2 N2 — Simultánní práce

Systém umožňuje bezkonfliktní práci více pracovníkům zároveň.

Priorita: 3 — kompletní splnění požadavku není urgentní pro první verzi systému, ale systém je bez něj dlouhodobě nepoužitelný a dodatečné přidávání funkcionality není uskutečnitelné bez neúměrných zásahů do systému.
Náročnost: 1–3 — Pokud je systém už od začátku návrhu připravován na simultánní práci více uživatel, není třeba jí věnovat přílišnou pozornost navíc; dodatečná implementace by však znamenala zásah do prakticky všeho zdrojového kódu.

1.4.2.3 N3 — Autorizovaný přístup

Systém musí být zabezpečen před neautorizovaným přístupem k citlivým datům.

Priorita: 2 — kompletní splnění požadavku není urgentní pro první verzi systému, ale jako webovou aplikaci dle **N1** ho nelze používat v ostrém provozu bez alespoň nějaké formy zabezpečení.
Náročnost: 1–2 — Bezpečnostním prvkům je třeba věnovat zvláštní pozornost, jinak může být systém zranitelný, ale existují řešení, jejichž promyšlená implementace není příliš náročná.

1.5 Aktéři a případy užití

Aktéři jsou uživatelé, systémy (včetně systému samotného), či hardwarová zařízení, kteří interagují se systémem za účelem dosažení nějakého užitečného cíle. Případ užití je pak diskrétní samostatná aktivita, kterou v systému aktér vykonává a která má hodnotný výsledek. [12]
Případy užití zasazují již dříve zformulované požadavky do lidského kontextu, čímž zjednodušují validaci požadavků budoucími uživateli a usměrňují návrh a testování systému [14]. Mnohdy je však nemožné jimi pokrýt všechny funkcionality systému, obzvláště u velkých projektů, či pokud uživatel se systémem interaguje minimálně [12].

Aktéři a případy užití[16] sestavené pro účely práce na základě funkčních požadavků ze sekce 1.4 jsou vypsány v následujících dvou podsekcích, a jejich vzájemné pokrytí je vyobrazeno v tabulce 1.1. Lze pozorovat, že požadavky **N1** a **N2** nejsou pokryty žádným případem užití, což je však vzhledem k jejich povaze přijatelné.

1.5.1 Aktéři

1.5.1.1 Nepřihlášený pracovník

Nepřihlášený pracovník je „nejslabší“ role v rámci systému. Jediný případ užití nepřihlášeného pracovníka je přihlášení.

1.5.1.2 Pracovník

Pracovník je základním aktérem systému. Spadá pod něj většina případů užití.

1.5.1.3 Administrátor

Administrátor má v rámci systému největší množství pravomocí. Spadají pod něj všechny případy užití pracovníka, ale též dodatečné administrativní pravomoci.

[16] Případy užití jsou označovány jako UC[číslo] podle anglického názvu termínu, „use case“.

1.5.1.4 Systém

Systém samotný též provádí některé případy užití, a je tedy také aktérem.

1.5.2 Případy užití

1.5.2.1 UC1 — Přihlásit se

Aktér: Nepřihlášený pracovník
Pokryté požadavky: N3

Pracovník se přihlásí pro přístup do systému.

1.5.2.2 UC2 - Zobrazit seznam uživatel

Aktér: Pracovník
Pokryté požadavky: F1

Pracovník si zobrazí seznam uživatel, které systém eviduje. Seznam lze filtrovat podle uživatelského jména či tagů.

1.5.2.3 UC3 — Přidat uživatele

Aktér: Pracovník, systém
Pokryté požadavky: F1, F2, F3

Aktér zadá do systému nového uživatele. Během procesu zadávání aktér vyplní jeho uživatelské jméno odpovídající jménu účtu na projektech WM. Dodatečně může uživateli přiřadit tagy a události, jichž se uživatel účastní.

1.5.2.4 UC4 — Spravovat uživatele

Aktér: Pracovník, systém
Pokryté požadavky: F1, F2, F3

Aktér upravuje informace o uživateli, konkrétně přiřazuje a odebírá jeho tagy a události, kterých se uživatel účastní.

1.5.2.5 UC5 — Zobrazit seznam událostí

Aktér: Pracovník
Pokryté požadavky: F3

Pracovník si zobrazí seznam evidovaných událostí. Seznam lze filtrovat podle názvu události či jejích tagů.

1.5.2.6 UC6 — Přidat událost

Aktér: Pracovník
Pokryté požadavky: F1, F3, F4

Pracovník do systému vkládá novou událost. Během procesu vytváření události musí uvést její název, počáteční a koncové datum, projekty, jichž se událost týká, a její typ (krátkodobá bez

hastagu, dlouhodobá s hashtagem). V případě dlouhodobé události uvede její hashtag. Volitelně může události přiřadit tagy a účastníky.

1.5.2.7 UC7 — Spravovat událost

Aktér: Pracovník (částečně systém)
Pokryté požadavky: F1, F3, F4

Pracovník upravuje informace o události, konkrétně počáteční a koncové datum, přiřazuje a odebírá účastníky, projekty události, a tagy. Systém dlouhodobým událostem přiřazuje dodatečné uživatele.

1.5.2.8 UC8 — Zobrazit seznam tagů

Aktér: Pracovník
Pokryté požadavky: F2, F4

Pracovník si zobrazí seznam evidovaných tagů (tagy uživatelské odděleně od tagů událostí) včetně stromové struktury. Seznamy lze filtrovat dle názvu tagu.

1.5.2.9 UC9 — Přidat tag

Aktér: Pracovník
Pokryté požadavky: F2, F4

Pracovník přidá tag. Povinně vybere unikátní[17] jméno tagu, volitelně pak vybere „nadtag“ a „podtagy“ v rámci stromové struktury a přiřadí uživatele či události, ke kterým tag náleží.

1.5.2.10 UC10 — Spravovat tag

Aktér: Pracovník
Pokryté požadavky: F2, F4

Pracovník tagu přidává a odebírá účastníky či události. Může měnit stromovou strukturu tagů přidáváním a odebíráním „nadtagu“ či „podtagů“.

1.5.2.11 UC11 — Zobrazit seznam projektů

Aktér: Pracovník
Pokryté požadavky: F5

Pracovník si zobrazí seznam systémem evidovaných projektů WM.

1.5.2.12 UC12 — Přidat projekt

Aktér: Pracovník
Pokryté požadavky: F5

Pracovník zaeviduje projekt. V rámci procesu vyplní název a webovou adresu projektu.

[17] Unikátní pro konkrétní druh tagu; je možné mít uživatelský tag a tag události se stejným názvem.

1.5.2.13 UC13 — Sbírat editace

Aktér: Systém
Pokryté požadavky: F6

Systém automaticky sbírá informace o editacích náležících k jednotlivým událostem. V závislosti na typu události se bude strategie sběru měnit; v případě krátkodobých událostí bude editace sbírat na základě dostupných informací o účastnících, v případě událostí dlouhodobých dle hashtagu. V obou případech zohlední projekty a časové trvání události. V případě dlouhodobých editací může systém vytvořit nové účastníky a přiřadit je události na základě **UC3** a **UC7**. Editace jsou sbírány pravidelně u všech událostí, jejichž koncové datum ještě nenastalo, a i po něm bude vyvoláno změnou účastníků či projektů dle **UC4, UC7**.

1.5.2.14 UC14 — Dočistit editace

Aktér: Pracovník
Pokryté požadavky: F6

V případě potřeby pracovník sesbíraná data z **UC13** dočistí, tj. přidá či odebere editace jednotlivých účastníků události za účelem zpřesnění výsledku výpočtu.

1.5.2.15 UC15 — Agregovat metriky

Aktér: Systém
Pokryté požadavky: F6

Na základě editací sesbíraných a dočištěných skrze **UC13** a **UC14** systém agreguje metriky o jednotlivých událostech a kategoriích událostí definovaných tagy.

1.5.2.16 UC16 — Zobrazit metriky

Aktér: Pracovník
Pokryté požadavky: F6

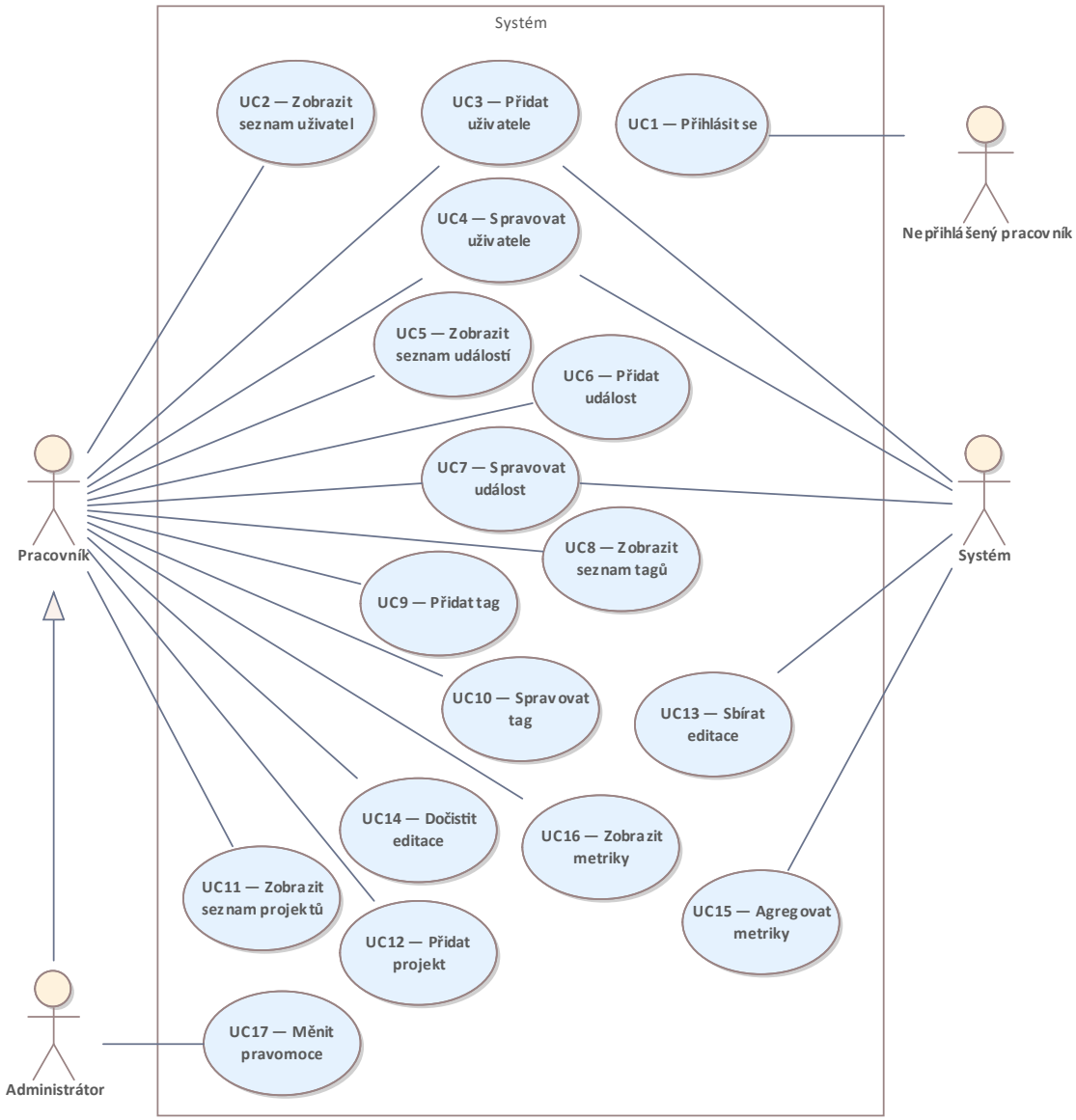
Pracovník si zobrazí metriky agregované skrze **UC15**.

1.5.2.17 UC17 — Měnit pravomoce

Aktér: Administrátor
Pokryté požadavky: F7

Administrátor pracovníkům přiřazuje a odebírá pravomoce — akce, které jsou v rámci systému pracovníkovi povoleny vykonávat.

Obrázek 1.1 UML Diagram případů užití.



Tabulka 1.1 Pokrytí požadavků případy užití.

	F1	F2	F3	F4	F5	F6	F7	N1	N2	N3
UC1										+
UC2	+									
UC3	+	+	+							
UC4	+	+	+							
UC5			+							
UC6	+		+	+						
UC7	+		+	+						
UC8		+		+						
UC9		+		+						
UC10		+		+						
UC11					+					
UC12					+					
UC13						+				
UC14						+				
UC15						+				
UC16						+				
UC17							+			

Kapitola 2
Návrh

Tato kapitola se věnuje samotnému návrhu systému. Využívá analýzy z předchozí kapitoly a sestavuje na základě něj doménový model, který tvoří základ výsledného systému. Dále popisuje použitou architekturu a zvolené technologie pro serverovou a klientskou část systému, a zdůvodňuje jejich zvolení.

2.1 Doménový model

Doménový model je široce používaným nástrojem v oblasti systémového návrhu. Je tvořen dekompozicí zájmové domény — problému, který je řešen — na konceptuální třídy — jednotlivé části, které jsou pro řešení problému důležité. Samotný model pak ukazuje tyto konceptuální třídy, jejich vlastnosti, a vztahy mezi nimi. Doménový model popisuje zájmovou doménu z perspektivy „reálného světa“ a nezachází do implementačních detailů. [15]

Diagram doménového modelu systému se nachází na obrázku 2.1 a jednotlivé konceptuální třídy jsou vysvětlené v následující podsekci.

2.1.1 Konceptuální třídy

2.1.1.1 User

Uživatel projektů Wikimedia. Jeho vlastnostmi jsou jeho uživatelské jméno a globální ID (viz podsekce 1.3.1).

2.1.1.2 Event

Pořádaná událost. Jejími vlastnostmi jsou její jméno, počáteční a koncové datum, druh události, a její hashtag, pokud nějaký má. Jedné události se účastní libovolný počet uživatel, a jeden uživatel se může účastnit libovolného počtu událostí.

2.1.1.3 Project

Projekt Wikimedia. Jeho vlastnostmi jsou jeho název a webová adresa. Jedna událost se týká libovolného množství projektů, a jednoho projektu se může týkat libovolné množství událostí.

2.1.1.4 UserTag, EventTag

Tag uživatele, respektive události. Jeho vlastností je jeho název. Jeden tag má libovolný počet uživatel nebo událostí, a jeden uživatel nebo událost má libovolný počet tagů. Jeden tag má nejvýše jeden „nadtag“ a libovolné množství „podtagů“.

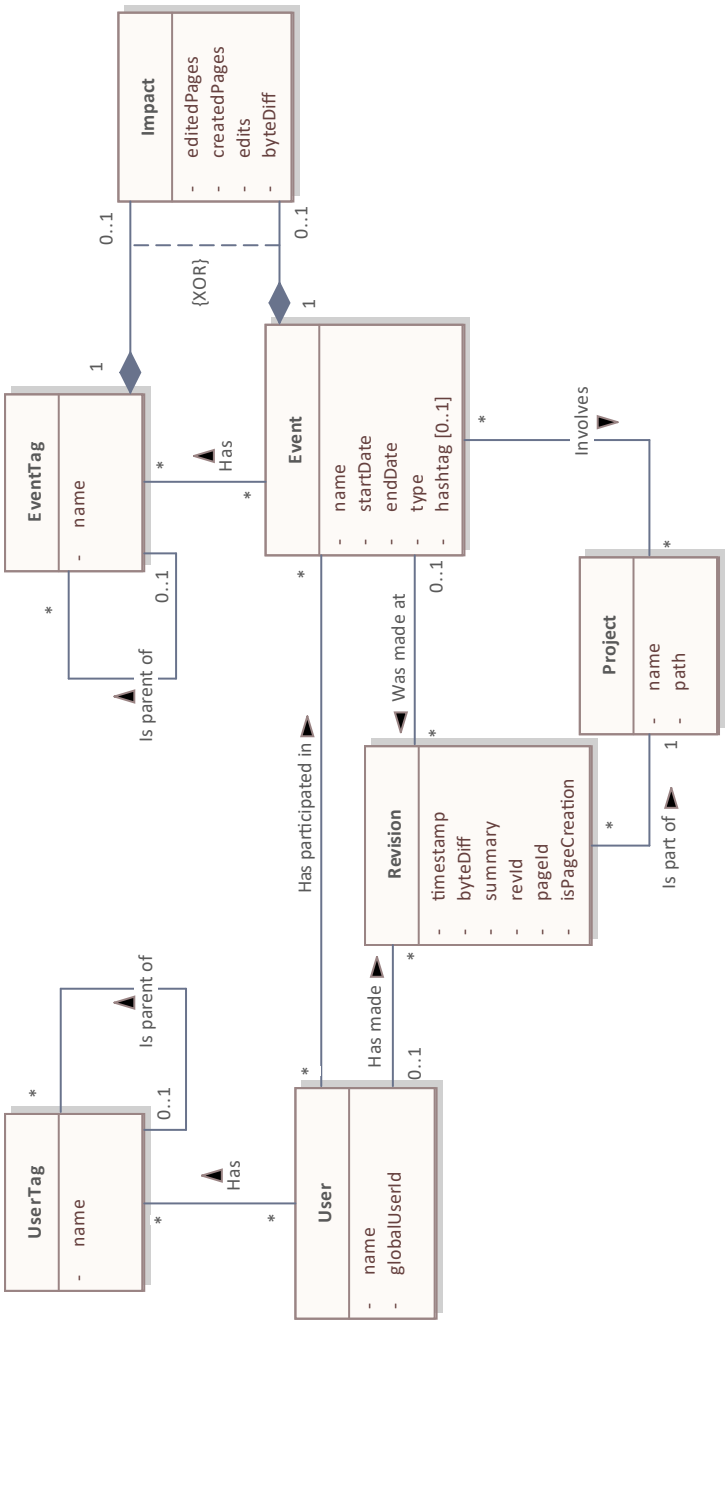
2.1.1.5 Revision

Editace článku v rámci ekosystému WM. Jejími vlastnostmi jsou čas editace, množství změněných dat v bajtech, slovní shrnutí, ID revize v rámci projektu, ID editovaného článku, a zda daná revize daný článek vytvořila. Jedna editace byla provedena v právě jednom projektu, v jednom projektu je provedeno libovolné množství editací. Editace je provedena nejvýše jedním uživatelem[1], jeden uživatel provádí libovolné množství editací. Editace se týká libovolného množství událostí, jedna událost obsahuje libovolné množství editací.

2.1.1.6 Impact

Dopad události (či všech událostí s daným tagem) na projekty WM. Jejími vlastnostmi jsou počet editovaných stránek (včetně nově vytvořených) v rámci události, počet nově vytvořených stránek, celkový počet editací, a počet přidaných bajtů[2] do systémů WM. Tyto vlastnosti jsou vypočtené na základě konceptuálních tříd Revision ve vztahu s danou událostí (či všemi událostmi náležejícími danému tagu). Dopad náleží právě jedné události či právě jednomu tagu, nikoli však události i tagu zároveň.

[1] Editace může být provedena i nepřihlášeným, v takovém případě nelze navázat na konkrétního uživatele.
[2] Tyto vlastnosti jsou dále přesněji definovány v sekci 3.1.3.5.



Obrázek 2.1 UML diagram doménového modelu

2.2 Serverová část

Serverová část systému je samostatně funkční jednotka, v níž probíhají veškeré výpočty, získávání dat z vnitřní databáze a vzdálených serverů a jejich zpracování . Přístup k jednotlivým funkcím serverové části je umožněn skrze API — aplikační rozhraní, které je jednoduše strojově dostupné, ale není uzpůsobeno pro lidský přístup. Klientská část pro přístup k datům využívá API serverové části.

2.2.1 Architektura

Server využívá relativně běžné vícevrstvé architektury. Systém je, podobně jako dort, rozdělen do několika vrstev, každá s konkrétním účelem [16]:

Prezentační vrstva přijímá požadavky od klienta, za pomoci nižších vrstev je zpracovává, a odesílá odpovědi,

Aplikační vrstva provádí téměř veškerou logiku aplikace; probíhají v ní veškeré výpočty a zpracování dat,

Datová vrstva se stará o přístup k datům, ať už z vnitřní databáze, či ze vzdálených aplikačních rozhraní.

Základním principem tohoto rozvrstvení je iniciace „shora dolů“ — vyšší vrstvy pro své potřeby volají vrstvy pod nimi, nikoli však naopak. Jinak by se též dalo říci, že každá vrstva pro své správné fungování závisí pouze na vrstvách pod ní, a je nezávislá na vrstvách nad ní. [16]

Výraznou výhodou takového návrhového vzoru je relativně nízká provázanost, vysoká míra nezávislosti a jasné rozdělení zodpovědností mezi jeho jednotlivými částmi, což systém zpřehledňuje, zvyšuje jeho modularitu a rozšiřitelnost, a zjednodušuje případné zásahy do jeho fungování.

Funkce a struktura jednotlivých vrstev v rámci serveru je podrobněji popsána níže.

2.2.1.1 Prezentační vrstva

Prezentační vrstva je tvořena **API controllery**, což jsou rozhraní, která přijímají požadavky zvně[3] a předávají je ke zpracování. Controller obecně „hlídá“ několik API endpointů — adres, na které mohou přicházet HTTP požadavky — a proces jejich zpracování má následující kroky:

Controller přijme od klienta HTTP požadavek. Pokud požadavek obsahuje tělo, konvertuje ho na validní objekt, a zavolá příslušnou servisní třídu (viz aplikační vrstva) ke zpracování požadavku. Po dokončení zpracování controller od servisní třídy převezme výsledný objekt[4], zkonvertuje ho do potřebného formátu odpovědi, je-li to potřeba, a pošle odpověď nazpět. Pokud během tohoto procesu nastane výjimka, controller ji zachytí a odešle odpověď, která klienta o chybě obeznámí.

Seznam API endpointů je dostupný v příloze A.

2.2.1.2 Aplikační vrstva

Aplikační vrstva je tvořena **servisními třídami**), které jsou dále děleny na **vnitřní servisní třídy** a **servisní třídy klientů**.

Ve vnitřních servisních třídách probíhá téměř veškerá logika aplikace. Servisní třída je zavolána controllerem, zpracuje daný požadavek, během nějž volá data access objekty (viz datová

[3] Pro účely práce jsou zvažovány pouze požadavky od webového klienta popisovaného v následující sekci, ale do budoucna lze u některých endpointů uvažovat i o veřejném přístupu.

[4] Pokud je účelem zpracování mazání objektu, není vrácen žádný objekt a požadavek je považován za úspěšně zpracovaný, pokud nenastane výjimka.

vrstva) pro přístup k datům z databáze, jejich úpravu, či ukládání nových dat, a volá servisní třídy klientů pro získání dat ze vzdálených rozhraní.

Servisní třídy klientů napomáhají vnitřním servisním třídám, zpracovávají jejich požadavky a posílají je klientům (viz datová vrstva) pro získání vzdálených dat, interpretují příchozí odpovědi a posílají je nazpět vnitřním servisním třídám.

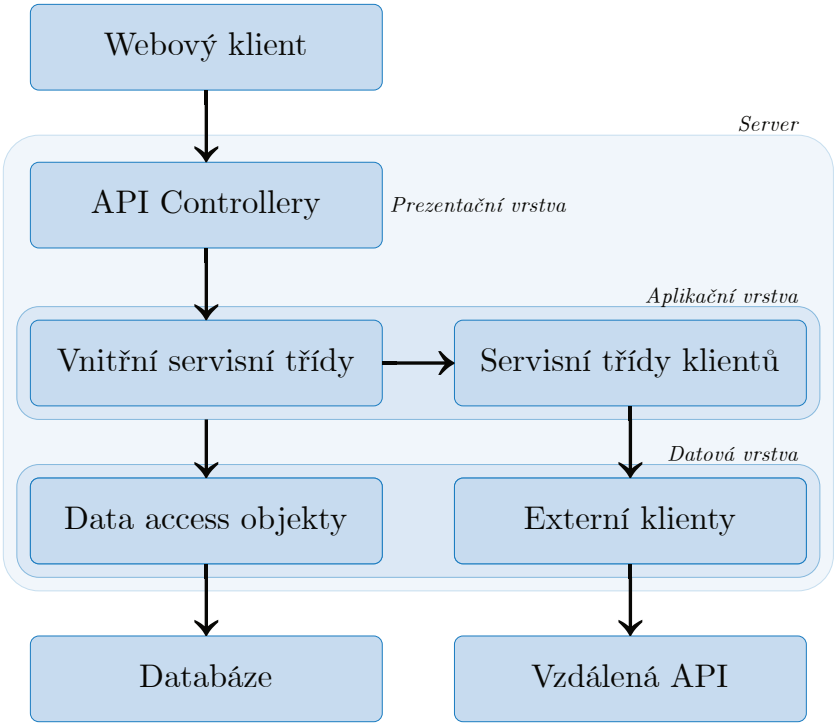
2.2.1.3 Datová vrstva

Datová vrstva je tvořena **data access objekty** a **externími klienty**.

Data access objekty tvoří rozhraní pro komunikaci s databází. Starají se o přístup k datům v databázi, jejich ukládání, a jejich konverzi na objekty, se kterými server pracuje, a naopak.

Externí klienty podobně na základě volání ze servisních tříd odesílají požadavky na vzdálené API endpointy a čekají na odpověď, kterou konvertují na validní objekt a předávají zpět servisním třídám. Na rozdíl od data access objektů v rámci práce externí klienty pouze data ze vzdálených databází získávají a nemohou ukládat nová.

Obrázek 2.2 Diagram rozvrstvení serveru, jeho jednotlivých komponent, a iniciačního toku.



2.2.2 Využité technologie

Tato podsekce popisuje programovací jazyk a framework zvolený pro vývoj serverové časti systému, a zdůvodňuje jejich zvolení.

Serverová část funguje prakticky nezávisle na již existujících systémech, respektive přistupuje k nim pouze za pomoci technologicky nezávislých API, a volba tedy není těmito faktory ovlivněna.

2.2.2.1 Java

Java je vysoko-úrovňový objektově orientovaný silně a staticky typovaný programovací jazyk [17]. Používá garbage collection a neobsahuje „nebezpečné“ chování, jako například přístup k poli bez kontroly indexu. Zdrojový kód psaný v Javě je kompilován do bytekódu, který je nezávislý na zařízení, na kterém byl kompilován a na kterém bude spouštěn. Pro účely spuštění je poté využit virtuální stroj (dále JVM), který za účelem běhu programu abstrahuje hardwarové specificity reálného stroje, čímž umožňuje spouštění stejného kódu bez úprav na téměř kterémkoli zařízení [18].

Výraznou výhodou Javy je její popularita. Dle Stack Overflow Survey je 6. nejpoužívanějším programovacím jazykem [19], pravidelně pro ni vychází nové verze [20], existuje pro ni velké množství knižních i elektronických zdrojů, a je na ní postaveno velké množství frameworků, včetně toho, jehož využívá tato práce.

2.2.2.2 Spring

Spring je framework[5] vyvíjený od roku 2003 za účelem zjednodušení tvorby enterprise aplikací v programovacím jazyce Java [21]. Obecně se těší velké oblibě; v dnešní době to je nejpoužívanější framework v jazyce Java a celkově čtvrtý nejpoužívanější ne-webový framework [19]. V práci je využita i nadstavba *Spring Boot*, jejíž účelem je mj. abstrahovat velkou část konfigurace a poskytnout další nástroje umožňující zrychlení vývoje [22].

Zásadní výhodou využití Springu je právě ono dramatické zrychlení a zjednodušení vývoje aplikace. Základní *Spring Framework* se postará o vkládání závislostí, *Spring Data JPA* téměř kompletně abstrahuje přístup k databázi, *Spring Web MVC* tvorbu API controlleru abstrahuje na několik málo anotací (viz výpis 2.1). Spring též automaticky podporuje vícevláknové procesy a ve výchozím nastavení vytváří nové vlákno pro každý příchozí požadavek, což umožňuje zpracování mnoha požadavků zároveň.

Spring je open-source a publikován pod licencí Apache 2.0, což umožňuje jeho využití v práci [23].

Výpis kódu 2.1 Jednoduchý controller se Spring Web MVC

```
@RestController
class HelloController {

    @GetMapping("/hello")
    String getHello() {
        return "Hello, world!"
    }
}
```

[5] Ve skutečnosti je v dnešní době Spring celá rodina frameworků, postavená nad základním frameworkem zvaným *Spring Framework*. Pro zjednodušení bude však v textu Spring nadále považován za jeden framework s více částmi.

2.3 Klientská část

Klientská část systému je uživatelsky přívětivým webovým rozhraním pro přístup k funkcím serverové části. Sama nemá užitečnou funkcionalitu, pouze ji zprostředkovává a umožňuje využití systému kýmkoliv.

2.3.1 Architektura

Klient je navržen jako takzvaná „single-page“ aplikace — na rozdíl od tradičních webových stránek, které při většině uživatelských akcí musí překreslit a načíst celou stránku znovu, single-page aplikace pouze postupně upravuje jednotlivé části stránky podle potřeby. Oproti tradičním desktopovým aplikacím však zároveň nabízí stejnou flexibilitu a jednoduchost přístupu, jakou mají webové stránky [24]. Tímto kombinuje několik výhod obou „tradičních[6]“ možností:

Rychlost. Obzvláště u větších „tradičních“ webových stránek může být proces opětovného stahování mnohdy redundantních dat a jejich opětovného vykreslování časově náročný. Uživatel musí opětovně čekat, a nedozví se, zda stránka např. zamrzla, či se stále načítá. Oproti tomu single-page aplikace potřebuje stahovat a překreslovat pouze změněné části stránky, čímž se může pro uživatele výrazně snížit doba čekání, a pro server i prohlížeč míra jeho zátěže [24].

Oznámení stavu. Single-page aplikace dokáže během čekání na odpověď serveru dynamicky vykreslit načítací listu či kolečko, a ujistit tím uživatele o jejím stavu, případně zpracovat chybu připojení během posílání požadavku na server či přijímání odpovědi a obeznámit s tím uživatele i bez kompletního přepsání stránky chybovou hláškou, případně zobrazit jiný, „fallbackový“ obsah [24].

Přístupnost. Na rozdíl od tradičních desktopových či mobilních aplikací je single-page webová aplikace jednoduše a bezpracně přístupná z téměř jakéhokoli zařízení s přístupem k internetu [24].

Okamžitá aktualizace. Na rozdíl od tradičních desktopových aplikací je aktualizace single-page aplikace záležitostí jednoho opětovného načtení stránky v prohlížeči. Není neobvyklé vidět single-page aplikace, které jsou aktualizovány i několikrát během jednoho dne, téměř[7] bez problému [24].

Single-page aplikace mají samozřejmě i nevýhody oproti tradičnějším řešením; jedním příkladem je větší množství celkového kódu, který musí prohlížeč načíst a zpracovat, což znamená, že první načtení aplikace bývá pomalejší, než u tradičních webových stránek. Toto je sice relativně rychle vynahrazeno právě tím, že výsledná stránka se již nikdy znovu nenačítá celá, a mění se pouze její části, přesto je však dobré na toto myslet v případě užití na nevýkonných zařízeních, případně na místech s pomalejším připojením.

2.3.2 Využité technologie

Tato sekce popisuje programovací jazyk a frameworky zvolené pro vývoj klientské části systému, a zdůvodňuje jejich zvolení.

Klientská část přistupuje pouze k serverové části systému za pomoci technologicky nezávislého API, a volba tedy není tímto ovlivněna.

[6] V dnešní době většina velkých webových i desktopových aplikací uplatňuje alespoň do nějaké míry koncept single-page, „tradiční“ tedy může být poněkud zavádějící pojem; pro účely této sekce se tím myslí klienty, které těchto konceptů nevyužívají.

[7] Některé velké změny, obzvláště ty měnící specifikaci komunikace mezi klientem a serverem, jsou nadále problematické, ale obecně je náročnost zajištění fungování systému pro všechny uživatele během těchto úprav výrazně méně náročná, než u „tradičních“ aplikací, kde uživatelé relativně běžně používají i vícero starších verzí zároveň.

2.3.2.1 JavaScript

JavaScript je prototypový, slabě a dynamicky typovaný programovací jazyk založený na standardu ECMAScript, který podporuje vícero programovacích paradigmat [25]. Je to již více než dekádu nejpoužívanější programovací jazyk [19]. Je využíván téměř všemi nestatickými webovými stránkami [26], ale jeho využití přesahuje i dále, mj. i do oblasti serverů [25].

JavaScript má na interaktivní webové aplikace praktický monopol, a je tedy v podstatě bez alternativy.

2.3.2.2 Vue

Vue.js (dále jen Vue) je javascriptový framework pro tvorbu uživatelských rozhraní vyvíjený od roku 2014 Evanem Youem. Je založen na systému komponent, které mohou být používány znovu a znovu, čímž umožňuje stavět výkonná reaktivní uživatelská rozhraní využívající moderní webové technologie. Vue je vysoce flexibilní a oproti jiným frameworkům má relativně nízkou režii a krátkou křivku učení [27, 28, 29].

Vue je open-source a publikován pod licencí MIT, což umožňuje jeho využití v práci [28].

2.3.2.3 Quasar

Quasar je javascriptový framework vybudovaný nad Vue.js za účelem urychlení a zkvalitnění vývoje multiplatformních webových aplikací, vyvíjený od roku 2015 Razvanem Stoenescuem. Mimo jiné obsahuje velkou knihovnu výkonných přizpůsobitelných Vue komponent navržených v souladu s široce používanou designovou filozofií Material Design [30].

Quasar je open-source a publikován pod licencí MIT, což umožňuje jeho využití v práci [30].

Kapitola 3
Implementace

Tato kapitola se věnuje implementaci systému. Využívá analýzy a návrhu z předchozích kapitol a přetváří doménový model v modelové třídy tvořící základ výsledného systému, a zdůvodňuje změny oproti původnímu návrhu. Dále popisuje proces získávání dat a způsob jejich prezentace a manipulace.

3.1 Serverová část

3.1.1 Modely

Modely jsou třídy reprezentující jednotlivé části řešené domény, a jsou tedy přenesením konceptuálních tříd doménového modelu ze sekce 2.1 do reálné implementace. Instance těchto tříd jsou základními nosiči dat v rámci serveru, a jednotlivé vrstvy si je předávají, tvoří nové, a mění existující, aby dosáhly vytyčených cílů.

Vazby mezi jednotlivými modelovými třídami jsou reprezentované referencí v případě vztahu s maximálně jedním povoleným prvkem, a kolekcí referencí v případě vícenásobného vztahu.

Jedinou změnou u většiny modelových tříd oproti jejich respektivním koncepčním třídám je přidání jednoznačného neměnného číselného identifikátoru, který zjednodušuje a urychluje práci s jednotlivými instancemi. Tento identifikátor je generován automaticky při prvním uložení instance do databáze a není tedy třeba ho předávat systému dopředu. Výjimkami jsou třída *User*, kde je tímto identifikátorem globální ID zmiňované v sekcích 1.3.1 a 2.1, a generované ID tedy nepotřebuje, a *Impact*, která není ukládána (více níže). Zbylé změny jsou popsané níže.

3.1.1.1 User

U modelové třídy uživatele přibyl oproti doménovému modelu atribut `localId`, indikující místní ID designovaného „globálního projektu“ (viz podsekce 3.1.3.2).

Uživatelské jméno je pak považováno za nestálé, a je tedy pro správné fungování systému důležité se před jeho využitím ujišťovat, zda nebylo změněno, a tuto změnu správně reflektovat v systému.

3.1.1.2 Revision

U modelové třídy je vlastnost indikující vytvoření stránky danou editací přeměněna z pravdivostní hodnoty na číselné ID předchozí editace projektu. V případě, že editace stránku vytvořila, má toto ID hodnotu 0.

3.1.1.3 Impact

U modelové třídy dopadu oproti doménovému modelu *ubyla* existenční závislost na události či tagu; vzhledem k povaze třídy, kde všechny její atributy jsou derivované z existujících vlastností, je vytvářena a předávána podle potřeby a není jakkoli ukládána.

3.1.2 Entity a jejich správa

Entity jsou záznamy o jednotlivých modelových třídách v rámci databáze. Spring práci s entitami prakticky celou abstrahuje, ale přesto je důležité mít povědomí o jejich funkci.

Specifikace jednotlivých entit (s výjimkou třídy Impact, která není v databázi uchovávána, a tedy entitu nemá) se téměř neliší od modelových tříd; jedinou změnou je reprezentace vztahů, která je v databázi reprezentovaná standardní formou:

U vazeb typu 1:N je na jedné straně vazby vždy nejvýše jedna entita, a vazba je tedy reprezentována odkazem na tuto entitu na druhé straně. Příkladem je vazba „nadtag-podtag“, kde jeden tag může mít mnoho podtagů, ale maximálně jeden nadtag. Vazba je tedy reprezentována u záznamu podtagu odkazem na jeho nadtag.

U vazeb N:N může být na obou stranách vazby libovolné množství entit, a je tedy nutné vztah dekomponovat do další tabulky, která udržuje záznamy o jednotlivých propojeních.

Jednou vlastností Springu, které je potřeba věnovat zvláštní pozornost, je *vlastnictví vazeb*. U vazby je vždy jedna strana označena jako vlastnická, a Spring danou vazbu „hlídá“ pouze z této strany. Je-li potřeba přidat či odebrat vazbu, tato změna musí být provedena právě z vlastnické strany, jinak není uložena.

Pro účely implementace bylo považováno za žádoucí, aby změny probíhaly bez ohledu na vlastnictví vazby; tato funkcionalita byla tedy implementována na aplikační úrovni. Jednotlivé servisní třídy rozpoznají, je-li požadovaná změna vazby, kterou daná třída nevlastní, a vyvolají ji na vlastnické straně vazby.

Výpis kódu 3.1 Část kódu definující třídu User s anotacemi určujícími chování odpovídající entity.

```
@Entity                                        // třída specifikuje entitu
public class User implements IdAble<Long> {

    @Id                                        // klíč entity
    private Long id;

    private Long localId;

    private String username;

    private Instant registration;

    @ManyToMany                                // vlastněný vztah typu N:N
    private Set<UserTag> tags;

    @ManyToMany(mappedBy = "participants") // nevlastněný vztah typu N:N
    private Set<Event> events;

    @OneToMany(mappedBy = "user")              // nevlastněný vztah typu 1:N
    private Set<Revision> revisions;

    (...)
}
```

3.1.3 Procesy využívající vzdáleně získaná data

Tato podsekce se věnuje popisu procesů serverové části systému, jež v rámci svého vykonávání získávají data ze vzdálených API a konvertují je na validní instance modelových tříd, či těchto získaných dat využívají, a vysvětluje jejich význam v rámci systému.

3.1.3.1 Standardní dotaz na API Wikimedia projektu

Vzhledem k faktu, že téměř všechny API dotazy směřují na některý z projektů Wikimedia, mají jisté společné vlastnosti. Pro stručnost jsou tyto vypsány zde, a dále již nejsou zmiňovány.

Všechny dotazy probíhají na URL `[cestaprojektu]/w/api.php` a konkrétní dotaz je pak předán formou dotazovacího řetězce v rámci URL. Například dotaz na českou Wikipedii zjišťující informace o revizi s číselným ID 21282405 má následující formu:

`https://cs.wikipedia.org/w/api.php?action=query&prop=revisions&revids=21282405`.

Parametr `action=query` indikuje dotaz na nějakou existující informaci, `prop=revisions` upřesňuje, že jde o editace, a `revids=21282405` specifikuje konkrétní editaci.

Všechny dotazy serveru na API WM projektu mají následující parametry[1]:

Tabulka 3.1 Parametry dotazu na WM API

Název a hodnota parametru	Popis významu
`action=query`	Účelem volání API je dotaz.
`format=json`	Odpověď je ve strojově čitelném formátu JSON.
`formatversion=2`	Určení konkrétního formátu odpovědi.
`maxlag=3`	Je-li lag WM serveru nad 3 sekundy, dotaz není proveden.

Účelem využití parametru `maxlag` je zamezení provádění ne nezbytných volání v případě vysokého zatížení WM serveru.

Odpovědi na tyto dotazy jsou ve formátu JSON, a jejich jednotlivé klíče jsou vypsány v tabulkách pro každý dotaz zvlášť. V těchto výpisech jsou některé klíče vynechány, pokud nemají žádný význam pro tuto práci.

Po přijetí serverovou částí systému jsou odpovědi převedeny externími klienty na ekvivalentní Java objekty. Tyto objekty pak externí servisní třídy převádějí na validní instance modelových tříd, a toto převedení je popsáno pro každý případ zvlášť.

3.1.3.2 Tvorba a správa uživatele

Uživatel je jednou z nejzákladnějších modelových tříd systému a je z definice propojen se systémy Wikimedia. Není tedy žádným překvapením, že nemalá část API dotazů se týká tvorby instancí těchto tříd a udržování jejich validity v rámci ekosystému Wikimedia.

K zavedení uživatele do systému je potřeba znát pouze aktuální uživatelské jméno; zbylá data systém sbírá z API projektu. Při tvorbě uživatele je API zavoláno s dvěma různými dotazy:

První dotaz získává globální uživatelské info standardním API dotazem s dodatečnými parametry `meta=globaluserinfo`, a `guiuser` následovaný uživatelským jménem. Odpověď se nemění bez ohledu na dotazovaný projekt, v implementaci byl proto určen jeden konkrétní „designovaný globální projekt," na který tyto projektově agnostické dotazy směřují[2]. Odpovědí je záznam s následujícími informacemi:

[1] Kompletní dokumentace API je dostupná pro každý projekt na adrese `/w/api.php?action=help`

[2] V aktuální implementaci je tímto designovaným globálním projektem Meta-Wiki, wiki o projektech Wikimedia, dostupná na `meta.wikimedia.org`.

Tabulka 3.2 Struktura záznamů odpovědi dotazu na globální info uživatele

Klíč	Hodnota
id	globální ID uživatele
name	uživatelské jméno uživatele
registration	čas registrace uživatele

Tyto informace jsou sice všechny, které systém dle návrhu potřebuje, vyvstává však následující problém: uživatelské jméno je nestálé a může se měnit. Je tedy třeba ho pravidelně aktualizovat pro správné fungování systému, ideálně pak pokaždé, kdy je tento údaj potřeba.

Záměnou parametru `guiuser` za `guiid` lze ten samý záznam získat z globálního ID, toto však naráží na limitaci dotazu na globálního uživatele: lze provádět pouze pro jednoho uživatele najednou. Důsledkem tohoto by při širším použití dotazu bylo neúnosné zpomalení systému s přibývajícím počtem uživatel[3]. Tohoto globálního dotazu je tedy využito pouze při tvorbě uživatele, případně při práci s jediným uživatelem.

Jako alternativa byl do uživatelské modelové třídy byl tedy zaveden nový údaj, kterým je lokální ID designovaného globálního projektu. Tento údaj je získán během vytváření uživatele standardním dotazem na API projektu s dodatečnými parametry `list=users` a `ususers` následovaný uživatelským jménem. Odpovědí je série záznamů s následujícími informacemi:

Tabulka 3.3 Struktura záznamů odpovědi dotazu na lokální info uživatele

Klíč	Hodnota
userid	lokální ID uživatele
name	uživatelské jméno uživatele

Záměnou parametru `ususers` za `ususerids` se lze tedy dotazovat na proměnné uživatelské jméno skrze neměnné lokální ID, a tento dotaz lze provádět i pro více uživatel zároveň[4] (jednotlivá ID jsou pak oddělena svislicí).

3.1.3.3 Získávání editací krátkodobých událostí

Získávání editací je spuštěno po vytvoření a každé úpravě události. Tento proces probíhá v jiném vlákně, než zpracování samotného požadavku. Hlavním důvodem je pouze tangenciální vztah tohoto procesu s prováděnou úpravou — systém tímto pouze reaguje na prováděnou úpravu, není součástí úpravy samotné, a jeho úspěch či neúspěch též nic neříká o úspěchu úpravy.

Pro krátkodobé události je systému předán časový rozsah, uživatelé a projekty události. Pro každý projekt zvlášť je pak podán standardní dotaz s následujícími dodatečnými parametry:

Tabulka 3.4 Parametry dotazu na uživatele

Klíč a hodnota parametru	Popis
`list=usercontribs`	Seznam editací uživatel
`ucuser=[uživatelská jména oddělená svislicí]`	Seznam uživatel
`ucstart=[čas do]`	Nejzažší čas editace
`ucend=[čas od]`	Nejdřívější čas editace
`ucprop=ids|title|timestamp|comment|sizediff|flags`	Specifikace odpovědi

[3] Ve starší verzi systému využívající pouze dotaz na globální uživatelské info docházelo k čekání přibližně 3 sekundy při pouhém načítání seznamu obsahujícího 17 uživatel, což je při přibližně o dva řády vyšším očekávaném počtu evidovaných uživatel absolutně neakceptovatelná hodnota.

[4] Základní limit je 50 uživatel na jeden dotaz, tuto hodnotu lze však relativně nenáročně navýšit až na 500 v případě potřeby.

Tento dotaz vrátí sérii záznamů o editacích uživatel s následujícími informacemi:

Tabulka 3.5 Struktura záznamů odpovědi dotazu na editace uživatel

Klíč	Hodnota
userid	lokální ID uživatele
user	uživatelské jméno uživatele
pageid	ID stránky v rámci projektu
revid	ID editace v rámci projektu
parentid	ID předchozí editace, nebo 0, pokud neexistuje
timestamp	čas editace
comment	slovní shrnutí editace
sizediff	počet editací přidaných bajtů obsahu stránky

K vytvoření validní instance modelové třídy editace již není potřeba žádných dalších informací, a lze je tedy správně uložit a navázat vazbu na uživatele, událost a projekt. Tohoto seznamu editací je pak dále využito při výpočtu dopadu události či jejího tagu.

3.1.3.4 Získávání editací dlouhodobých událostí

Oproti krátkodobým událostem se editace prováděné v rámci událostí dlouhodobých získávají nikoli ze seznamu účastníků, ale z hashtagu události. Kvůli limitacím systémů Wikimedia je pro jejich získávání využit nástroj Hashtags (více viz podsekce 1.3.2) a jeho JSONový endpoint na adrese `hashtags.wmcloud.org/json/`. Dotaz je prováděn pro každý projekt události zvlášť. V rámci dotazu mu je předán hledaný hashtag, adresa prohledávaného projektu, a počáteční a koncové datum. Odpovědí je série záznamů o editacích s následujícími informacemi:

Tabulka 3.6 Struktura záznamů odpovědi dotazu na Hashtags

Klíč	Hodnota
Domain	webová doména projektu editace
Timestamp	čas provedení editace
Username	uživatelské jméno autora editace
Page_title	název editované stránky
Edit_summary	slovní shrnutí editace
Revision_ID	číselné ID editace v rámci projektu

Pro vytvoření validních instancí modelové třídy editace chybí zbylé informace o autorovi (obzvláště pak globální ID využívané jako klíč), počet přidaných bajtů, číselné ID stránky a ID předchozí editace stránky.

Informace o uživateli jsou získány skrze uživatelskou servisní třídu, která v závislosti na uživatelském jméně buď najde odpovídající záznam v databázi, nebo vytvoří nový záznam stejně jako v případě ručního vytvoření uživatele (viz Tvorba a správa uživatele). Tento postup je též vyobrazen na diagramu v příloze C.

Zbylé informace jsou získány skrze dvojici dotazů na API daného projektu s dodatečnými parametry `prop=revisions`, `rvprops=ids`, případně `rvprops=ids|size`, `rvslots=*`[5], a `revids` určený seznamem ID čísel editací, pro které je dotaz prováděn (maximálně 50), oddělených svislicemi.

Odpovědí je série záznamů o stránkách s následujícími informacemi:

[5]Parametr `rvslots` je potřeba explicitně specifikovat z důvodu zpětné kompatibility aktuální verze API s dřívějšími verzemi bez tohoto parametru.

Tabulka 3.7 Struktura záznamů odpovědi dotazu na editace stránky

Název parametru	Popis významu hodnoty
pageid	číselné ID stránky v rámci projektu
title	název stránky
revisions	série záznamů o editacích dané stránky

V rámci série editací se pak v jednotlivých záznamech nachází následující informace:

Tabulka 3.8 Struktura záznamů editací stránky

Název parametru	Popis významu hodnoty
revid	číselné ID editace v rámci projektu
parentid	číselné ID předchozí editace stránky
size	velikost stránky v bajtech po provedení editace (pouze při specifikaci parametru `size` v `rvprops`)

První dotaz je volán pouze s hodnotou parametru `rvprops=ids`, a pro každou editaci je jím získáno ID předchozí editace stránky, v druhém volání s hodnotou parametru `rvprops=ids|size` je mu pak předána ID editace i editace jí předcházející. Z odpovědi je pak získána velikost stránky v bajtech po provedení těchto revizí, z čehož je již výpočet rozdílu těchto velikostí triviální.

Po tomto druhém volání je tedy již možné vytvořit validní instanci modelové třídy identifikace, správně ji navázat na ostatní instance modelových tříd, a využít k výpočtu dopadu stejně jako editace krátkodobých událostí.

3.1.3.5 Výpočet dopadu

Výpočet dopadu jako takový již nepotřebuje vzdálený přístup; všechna potřebná data již byla získána v rámci předchozích podsekcí.

Vlastní proces je pak výpočetně relativně nenáročný; dopadové servisní třídě je předána množina všech editací, pro které je dopad zjišťován, a v této množině je pak vypočten:

počet upravených stránek výběrem unikátních stránek nacházejících se v množině editací dle ID a projektu stránky,

počet nově založených stránek výběrem prvků množiny editací s ID předchozí editace 0,

počet provedených editací zjištěním velikosti množiny, a

počet přidaných bajtů sečtením počtu přidaných bajtů každé editace v množině.

Zvláštní pozornost je věnována „sečtení počtu přidaných bajtů“. Ne každá editace totiž bajty přidává, některé je i ubírají, a existují různé strategie započítávání těchto negativních editací. Dopadová servisní třída tedy umožňuje počítání podle tří různých strategií:

`NET` — matematický součet $(2 + (-2) = 0)$,

`ABSOLUTE` — součet absolutních hodnot $(2 + (-2) = 4)$,

`POSITIVE` — součet ignorující záporné hodnoty $(2 + (-2) = 2)$.

Existují argumenty pro využití všech tří strategií; odstranění zvandalizovaného odstavce stránku sice zkrátí, a tradičně ji tedy nelze považovat za „přidání“, ale je to zajisté prospěšná změna, a názory ohledně započítání takové editace se přirozeně různí.

V aktuální implementaci součet používá strategii `ABSOLUTE`, ale v případě potřeby je možno jednoduše strategii změnit či vytvořit novou.

Výpis kódu 3.2 Část kódu dopadové servisní třídy definující strategii sčítání a její použití v kódu.

```
private final DiffStrategy strategy = DiffStrategy.ABSOLUTE;
enum DiffStrategy {
    NET,
    POSITIVE,
    ABSOLUTE
}

private Function<Revision, Long> getDiffFunc() {
    return switch (strategy) {
        case NET      -> Revision::getDiff;
        case ABSOLUTE -> r -> Math.abs(r.getDiff());
        case POSITIVE -> r -> (r.getDiff() > 0 ? r.getDiff() : 0);
    };
}

private Long getDiff(Set<Revision> revs) {
    return revs.stream().map(getDiffFunc()).reduce(0L, Long::sum);
}
```

3.1.4 Zajištění validity ukládaných dat

Pro většinu modelových tříd je serverová část systému vysoce permisivní a umožňuje uložení téměř jakékoli kombinace vlastností instance (například nezabraňuje uložit událost, která začíná až po svém skončení). Tato sekce se zabývá výjimkami, kdy je validita dat systémem vynucována a opak by mohl vést k zásadním chybám.

3.1.4.1 Stromová struktura tagů

V grafu všech tagů nesmí vzniknout cykly. Není možné, aby jeden tag byl sám sobě nadtagem, či aby nadtag byl zároveň podtagem. Tato vlastnost je vynucována na úrovni servisních tříd při každém vytváření či úpravě tagů jednoduchým algoritmem, který kontroluje, zda ukládáná změna nevede ke vzniku cyklu v grafu.

3.1.4.2 Editace událostí

Při změně vlastností události (datum konání, projekty, účastníci) jsou automaticky události v rámci zpracování v servisní třídě odebrány všechny editace, které neodpovídají vlastnostem události po jejich změně. V případě dlouhodobých akcí, kde jsou účastníci sbíráni automaticky, jsou v případě nulového počtu editací účastníků v rámci dané události po této úpravě odebráni i oni.

3.1.5 Nástroje využité v implementaci

3.1.5.1 Gradle

Gradle je řídící program určený k automatizaci sestavení softwaru [31]. Gradle běží na JVM, takže jeho logika může používat standardní „javovské“ API. Je též podporován mnoha vývojovými prostředími, což zjednodušuje práci s ním.

V rámci serverové části systému je Gradle využit k jeho sestavování a kompilaci, správě jeho závislostí (konkrétně právě Spring Frameworku), a samotnému spouštění skrze Docker (viz níže).

3.1.5.2 Docker

Docker je platforma vyvinutá pro účel zjednodušení a automatizace procesů v rámci vývoje a nasazování aplikací. Toho dosahuje oddělováním aplikací do „kontejnerů“, víceméně[6] izolovaných prostředí, která ne nepodobně virtuálním strojům poskytují jednotné podmínky pro běh aplikace bez ohledu na fyzický hardware stroje. Kontejnery jsou na rozdíl od virtuálních strojů sice méně mocným nástrojem, protože nejsou plnou simulací stroje včetně veškerého hardwaru, toto však vynahrazují menší velikostí, větší flexibilitou, a obecně jednodušším použitím a vyšší škálovatelností. I proto je Docker v současnosti jedním z nejpoužívanějších a nejoblíbenějších nástrojů [19, 32, 33].

Docker je využit k automatizaci sestavení serverové části systému, sestavení databáze a napojení serveru na ni, a samotnému spouštění serveru.

3.1.5.3 Spring Initializr

Spring Initializr[7] (sic) je utilita, jejíž účelem je generování počáteční struktury souborů pro projekty využívající Spring Frameworku. Mimo jiné do struktury zasazuje i Gradle (viz výše).

Spring Initializr byl využito na samém počátku vývoje k vytvoření struktury repozitáře, ve kterém se implementace systému nachází.

3.2 Klientská část

Klient je podle návrhu ze sekce 2.3 implementován jako single-page aplikace s využitím Vue komponent jak nativních, tak těch poskytovaných frameworkem Quasar. Klient tvoří grafické rozhraní pro zobrazování a manipulaci dat uložených v serverové části systému, k čemuž využívá API endpointů, vypsaných v příloze A.

Většina dat, která přichází odpověďmi ze serveru, neprochází výraznou transformací, s výjimkou tagů, které se v odpovědích ze serveru nachází v běžném plochém poli, a do stromové struktury jsou sestaveny až v klientu.

Design je uzpůsoben pro práci na stolním počítači či laptopu; použití aplikace na mobilním zařízení je možné, ale netestované. Jednotlivé komponenty jsou responzivní, a aplikace se dobře adaptuje pro různé šířky obrazovek, ale pro obrazovky užší než přibližně 450 px začínají být některé prvky neergonomické. Bude-li potřeba, je možno aplikaci připravit pro použití na mobilních zařízeních s pouze menšími úpravami.

Ukázky obrazovek se nachází v příloze D.

3.2.1 Nástroje využité v implementaci

3.2.1.1 Vue I18n

Vue I18n je knihovna pro Vue, jejíž účelem je zavádění jazykové lokalizace do aplikací v něm vytvářených. Místo zasazování jednotlivých řetězců textu přímo do jednotlivých komponent jsou obsaženy ve speciálních JSONových objektech (jeden pro každý jazyk), a na jejich místě v komponentě je pak pouze odkaz na daný řetězec. Za běhu se pak do komponenty vkládají řetězce ve zvoleném jazyce.

Klientská část aktuálně podporuje český a anglický jazyk.

[6] Míra izolovanosti lze měnit, chceme-li například zajistit komunikaci mezi kontejnery či hostitelským systémem [32]. V rámci práce je toto využito k propojení databáze a serverové části a zpřístupnění API endpointů.

[7] Dostupný na `https://start.spring.io/`.

Knihovna podporuje i možnost přepínání jazyku za běhu; toto sice v současné implementaci není umožněno a jednotlivé stránky jsou vždy zobrazovány v českém jazyce, ale aplikace je plně připravena pro budoucí přidání této funkcionality.

Výpis kódu 3.3 Kód definující jednotlivé položky komponenty zobrazující dopad, využívající Vue I18n pro zobrazení názvu položky a formátování číselné hodnoty. Kód je pro přehlednost upraven.

```
<q-item>
  <q-item-section>
    {{ val.toLocaleString($i18n.locale) }}
  </q-item-section>
  <q-item-section>
    {{ $t("impact." + name) }}
  </q-item-section>
</q-item>
```

3.2.1.2 Vue Router

Vue Router je oficiální knihovna pro Vue.js. Jejím základním účelem je zjednodušení tvorby single-page aplikací (viz sekce 2.3.1) skrze mapování Vue komponent na jednotlivé adresy a místa v aplikaci [34].

Vue Router při kliknutí na odkaz v rámci aplikace (například při otevření detailu uživatele, zobrazení seznamu událostí, či kliknutí na tlačítko vytváření nového tagu) překreslí změněné části stránky a příslušně upraví prohlížečem zobrazovanou URL bez nutnosti načítat celou stránku znovu, čímž implementuje principy single-page aplikace.

3.2.1.3 Axios

Axios je jednoduchý javascriptový asynchronní HTTP klient s knihovnou vybudovaný za účelem zjednodušení posílání a příjímání webových požadavků a odpovědí. S více než 42 miliony stažení týdně[8] na správci balíčků npm a více než 99 tisíci hvězdičkami[8] na hostovací platformě GitHub se Axios řadí k nejoblíbenějším balíčkům současnosti [35, 36].

V aplikaci je Axios využit k posílání požadavků na serverovou část a přijímání odpovědí z ní.

[8] K březnu 2023.

Kapitola 4
Testování

Tato kapitola se věnuje testování systému a popisu a zdůvodnění jednotlivých typů testů. Dále též zhodnocuje výsledky tohoto testování a důsledky z něj vyplývající.

4.1 Automatizované testování

Obě části systému jsou pokryty několika druhy automatizovaných testů, které jsou blíže popsány v této sekci.

4.1.1 Statická analýza kódu

Statická analýza kódu spočívá ve zkoumání formy a obsahu kódu bez jeho spuštění. Jejím účelem je poukázat na vzorce ve zdrojovém kódu, které často značí přítomnost chyby. [37]

Mezi takové vzorce můžou patřit například nedosažitelné větve, deklarace proměnných, které dále nejsou nikde využity, přístup k objektu, který potenciálně není inicializovaný, apod. Tyto vzorce samy o sobě nejsou *nutně* špatně, jsou-li správně využity a ošetřeny, ale obvykle poukazují na přítomnost chyby v kódu, pokud jejich využití není dostatečně promyšleno a odůvodněno.

V rámci práce je využita statická analýza poskytovaná vývojovým prostředím IntelliJ IDEA[1], a specificky pro klientskou část je navíc využit nástroj ESLint[2] s oficiálním pluginem pro detekci zásadních chyb ve Vue.js kódu[3].

4.1.2 Jednotkové testy

Jednotkové testy zkoumají správnou funkčnost nejmenších částí kódu, typicky jedné třídy či i pouze jedné funkce [38]. Jejich malý rozsah znamená, že jsou jednotlivé testy prováděny velmi rychle, a v případě nalezení chyby je možno velmi přesně lokalizovat její příčinu. Na druhou stranu může být jejich psaní náročné, obzvláště u velkých projektů, a nijak neověřují funkčnost větších celků a komunikace mezi nimi.

Testování jednotlivých tříd je prováděno za pomoci tzv. „mockování“ — protože není vhodné, aby byl test ovlivňován ostatními třídami, se kterými by testovaný komponent běžně komunikoval, jsou tyto nahrazeny jejich maketami, které tuto komunikaci simulují. K tomuto účelu je využit framework Mockito[4], který tuto funkcionalitu poskytuje.

[1] Dostupný na adrese `https://www.jetbrains.com/idea/`.
[2] Dostupný na adrese `https://eslint.org/`.
[3] Dostupná na `https://eslint.vuejs.org/`; v práci je využita priorita A: „Essential“.
[4] Dostupný na `https://site.mockito.org/`

V rámci práce jsou jednotkové testy nasazeny v serverové části, a aktuálně testují správnost funkce modelových a vnitřních servisních tříd, které jsou pro správnou funkci systému nejdůležitější. Modelové třídy jsou pokryty z 99 % , vnitřní servisní třídy pak z 91 %. V celé serverové části je aktuálně pokryto 52 % řádků. Do budoucna je plánováno jednotkové testy dále rozšiřovat.

4.1.3 Systémové testy

Systémové testy simulují procesy probíhající v celém, plně integrovaném systému, a systém je tedy testován jako funkční celek [38]. Oproti běžnému fungování systému jsou pouze dva rozdíly:

Prvním rozdílem je použitá databáze — místo produkční databáze systém interaguje se zvláštní databází, která data ukládá pouze po dobu průběhu testu.

Druhý rozdíl se projevuje při dotazu na vzdálené API endpointy: místo skutečných dotazů na skutečné vzdálené servery se dotaz a jeho odpověď, podobně jako při jednotkových testech, mockují pro zajištění konzistence testů napříč běhy a jejich nezávislost na stavu vzdálených serverů a kvalitě připojení.

Aktuálně systém žádné takové testy nemá, je je však plánováno do budoucna zařadit do sestavy.

4.2 Manuální testování

Systém byl též testován manuálně, a to jak autorem práce, tak částečně i Martinem Urbancem[5], který mj. výrazně pomáhal při nasazování systému na testovací server. Na nesystematickém manuálním testování nemůže spolehlivost systému záviset, ale obzvláště v raných fázích implementace pomáhá odhalit největší chyby.

Mezi nedostatky objevené manuálním testováním patří například nezohledňování vlastnických vazeb v rané verzi serverové části (viz 3.1.2), které způsobovalo, že se tyto vazby v některých případech neukládaly, či výrazné zdržení způsobované opakovaným jednotlivým voláním API endpointu `globaluserinfo` (zmiňovaného v sekci 3.1.3.2).

4.3 Uživatelské testování

Uživatelské testování, jak již název napovídá, spočívá v testování budoucími uživateli systému. Účelem tohoto typu testování je v rámci práce převážně sběr dat ohledně přívětivosti a ergonomičnosti klienta — zda se dobře používá, zda chování systému odpovídá představám jeho výsledných uživatel, zda je uživatelské rozhraní sestaveno tak, aby práci napomáhalo a naopak jí nebránilo, apod.

Poslední sezení k uživatelskému testování proběhlo 27. dubna 2023. V rámci přípravy na toto (a budoucí) testování byla aplikace nasazena na testovací server a její veřejný přístup byl zabezpečen skrze základní HTTP autentifikaci[6]. Testující byli instruováni k použití vlastních zařízení k přístupu k aplikaci na testovacím serveru, kde následně ověřovali její funkcionalitu.

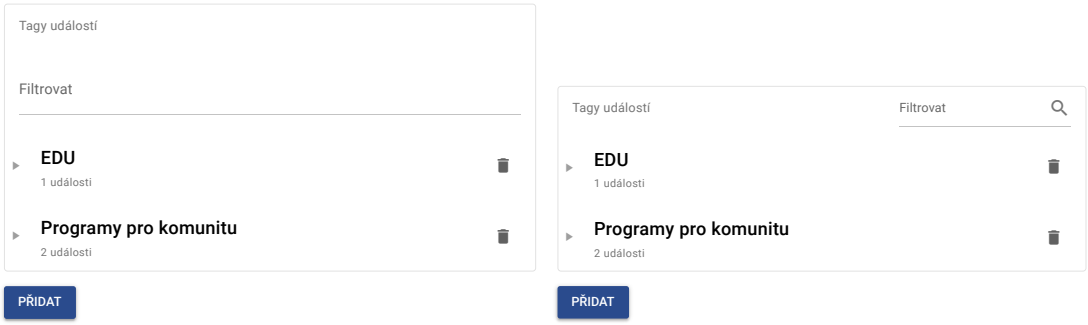
Z tohoto sezení vzestaly následující poznatky:

Pole pro filtrování seznamu je umístěno nevhodně. Jeho rozsah přes celou šířku seznamu v kombinaci s výrazným tlačítkem „Přidat“ (které je, obzvláště při menším počtu položek v seznamu, umístěno blízko onomu poli) a jeho vizuální podobností polím ve formulářích při vytváření nových položek, vedlo testující k jeho mylnému použití, když byli instruováni přidat novou položku.

[5] Martin Urbanec je mj. jedním z administrátorů české Wikipedie a radní Wikimedia ČR.

[6] Toto řešení je pouze dočasné do implementace autorizace skrze OAuth2, ale vzhledem k tomu, že aplikace nepracuje s citlivými daty, je toto řešení prozatím dostatečné.

Toto bylo zohledněno v následující iteraci zmenšením pole, přidáním ikony lupy, a jeho posunutím do pravého horního rohu (viz obrázek 4.1). Tímto je design podobnější designu vyhledávacích polí v jiných aplikacích a očekává se vymizení či výrazné zmírnění tohoto problému (což bude ověřeno při příštím sezení uživatelského testování).



Obrázek 4.1 Seznam tagů událostí s dvěma kořenovými položkami před (vlevo) a po (vpravo) změně designu filtrační kolonky vyplývající z uživatelského testování.

Tagy událostí ve svém dopadu nezapočítávají dopad jejich podtagů. Toto bylo během implementace nechtěně opomenuto, a funkcionalita byla dodána společně s menším refaktorováním kódu dopadové servisní třídy.

Otevřené rozbalovací seznamy zakrývají důležité prvky. Testující toto poznamenali obzvláště u formulářů pro vytváření jednotlivých prvků, kde takto rozbalený seznam často překrýval tlačítko pro odeslání formuláře.

Jako odpověď na tuto zpětnou vazbu bylo upraveno základní chování těchto seznamů tak, že se rozbalují směrem nahoru. Předpokladem je, že pracovníci většinou formulář vyplňují sekvenčně odshora dolů, a tedy takto rozbalený seznam bude zakrývat pouze již vyplněná pole.

V některých případech (například je-li nad polem málo místa) se může seznam nadále rozbalovat směrem dolů, ale toto chování je nyní víceméně ojedinělé.

V rozbalovacích seznamech nelze rozeznat strukturu tagů. Pro jednotlivé tagy se aktuálně v rozbalovacích seznamech při vytváření prvků zobrazují pouze jejich jména. To ztěžuje výběr, jelikož nelze poznat, zda je tag ve stromové struktuře listem či vnitřním uzlem.

Sami testující navrhli například zobrazení stromové struktury v rozbalovacím seznamu, podobně jako je tomu již teď u seznamu všech tagů, či zvýraznění tagů, které nejsou listy, například tučným textem. Zvýraznění by bylo jednoduché implementovat, ale má relativně nízkou vypovídající hodnotu bez potřebného kontextu; zobrazení stromové strukutry by bylo jasnější, ale složitější implementovat, a pravděpodobně by trpělo nízkou orientovatelností v případě většího množství tagů ve struktuře. Mezi další možnosti patří zobrazení informací v dodatkovém textu pod názvem tagu či vedle něj, jako například jméno nadtagu či počet podtagů. Aktuálně není rozhodnuto, které řešení problému použít.

Mnoho těchto problémů již bylo adresováno (viz výše), zbylé jsou brány v úvahu a jejich řešení je v plánu zapracovat do implementace co nejdříve. Do budoucna jsou plánovaná další sezení k uživatelskému testování se zvyšující se frekvencí s blížícím se dokončením aplikace.

Kapitola 5
Závěr

V práci vytvořený systém tvoří solidní základ, který lze dále efektivně rozvíjet. Byly splněny všechny požadavky ze sekce 1.4 s nejvyšší prioritou 3 a střední prioritou 2. **N3** (autorizovaný přístup) je prozatím splněn pouze částečně autorizací skrze základní HTTP autentifikaci, vzhledem k absenci práce s citlivými daty je toto však dostatečné pro účely rozšířeného testování. V blízké budoucnosti bude tento požadavek splněn plně implementací autorizace dle OAuth protokolu. Jediný požadavek priority 1, **F7** (role pracovníků), nebyl splněn, implementace je však též plánovaná v blízké budoucnosti.

Co se týče případů užití, mimo těch pokrývajících splnění výše zmíněných požadavků N3 a F7 nebyl prozatím implementován případ užití **UC14** (dočistit editace), jeho implementace je však též plánována co nejdříve.

Systém využívá aplikační rozhraní jednotlivých projektů Wikimedia pro sběr dat o editacích stránek provedených v rámci spolkem pořádaných událostí. Zpracováním těchto dat agreguje užitečné metriky, které zobrazuje pracovníkům skrze webovou aplikaci, která umožňuje i správu těchto událostí.

Aplikace má do budoucna velký potenciál. Mezi navrhovaná rozšíření po dokončení základní implementace patří specializovaná podpora pro multimediální soubory Wikimedia Commons a datové soubory Wikidat, generování reportů a jejich export do tabulek, tvorba registračních formulářů pro účastníky událostí, které by se automaticky přepisovaly do systému, veřejně přístupné aplikační rozhraní pro získání souhrnných anonymizovaných dat, a další.

Zdrojový kód je veřejně dostupný na `https://github.com/wmcz/statistics-tool`.

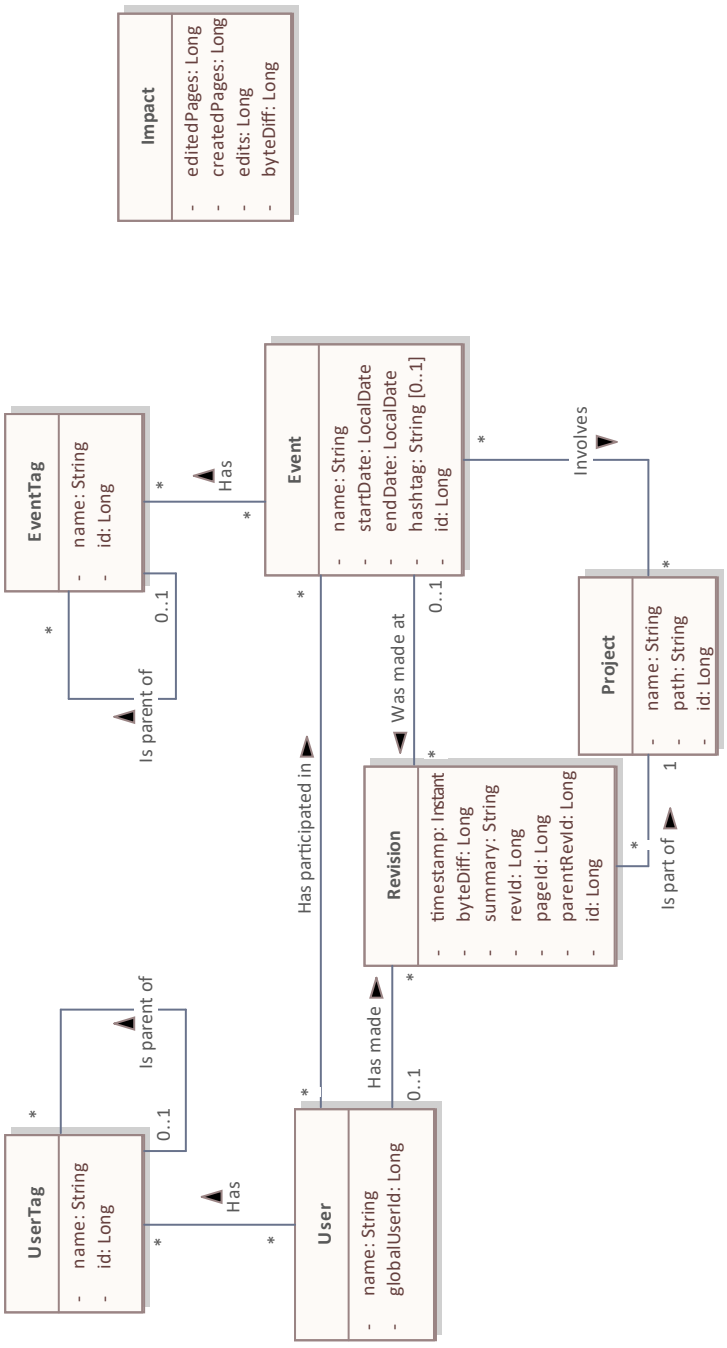
Příloha A

Seznam API endpointů

Endpoint	Metoda	Popis funkce
/users	GET	Získat seznam evidovaných uživatel.
	POST	Zaevidovat nového uživatele.
	PUT	Změnit data existujícího uživatele.
/users/{id}	GET	Získat uživatele, jehož id je {id}.
	DELETE	Smazat uživatele, jehož id je {id}.
/events	GET	Získat seznam evidovaných událostí.
	POST	Zaevidovat novou událost.
	PUT	Změnit data existující události.
/events/{id}	GET	Získat událost, jejíž id je {id}.
	DELETE	Smazat událost, jejíž id je {id}.
/events/{id}/revisions	GET	Získat všechny události s id {id}.
	PUT	Změnit editace náležící k události s id {id}.
/events/{id}/impact	GET	Získat dopad události s id {id}.
/tags/user-tags	GET	Získat seznam evidovaných uživatelských tagů.
	POST	Zaevidovat nový uživatelský tag.
	PUT	Změnit data existujícího uživatelského tagu.
/tags/user-tags/{id}	GET	Získat uživatelský tag, jehož id je {id}.
	DELETE	Smazat uživatelský tag, jehož id je {id}.
/tags/event-tags	GET	Získat seznam evidovaných tagů události.
	POST	Zaevidovat nový tag události.
	PUT	Změnit data existujícího tagu události.
/tags/event-tags/{id}	GET	Získat tag události, jehož id je {id}.
	DELETE	Smazat tag události, jehož id je {id}.
/tags/event-tags/{id}/impact	GET	Získat dopad všech událostí s tagem s id {id}.
/projects	GET	Získat seznam evidovaných projektů.
	POST	Zaevidovat nový projekt.
	PUT	Změnit data existujícího projektu.
/projects/{id}	GET	Získat projekt, jehož id je {id}.
	DELETE	Smazat projekt, jehož id je {id}.

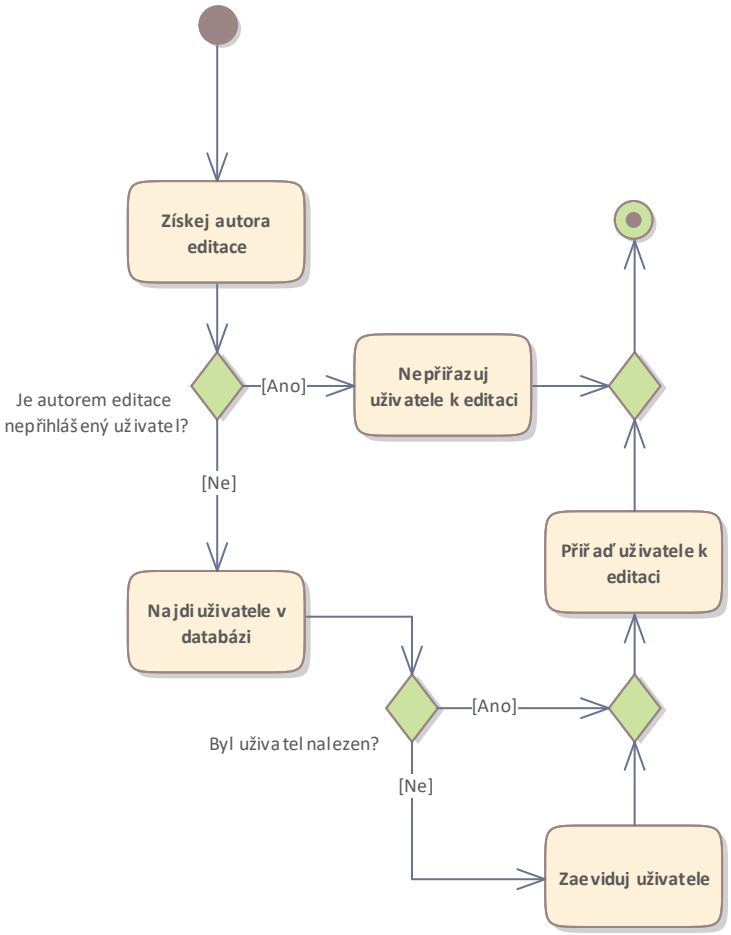
Příloha B

Modelové třídy serverové části


Impact
- editedPages: Long
- createdPages: Long
- edits: Long
- byteDiff: Long
EventTag
- name: String
- id: Long
Has
Event
- name: String
- startDate: LocalDate
- endDate: LocalDate
- hashtag: String [0..1]
- id: Long
Involves
Is parent of
Was made at
Project
- name: String
- path: String
- id: Long
Has participated in
Revision
- timestamp: Instant
- byteDiff: Long
- summary: String
- revId: Long
- pageId: Long
- parentRevId: Long
- id: Long
Is part of
Is parent of
Has made
UserTag
- name: String
- id: Long
Has
User
- name: String
- globalUserId: Long

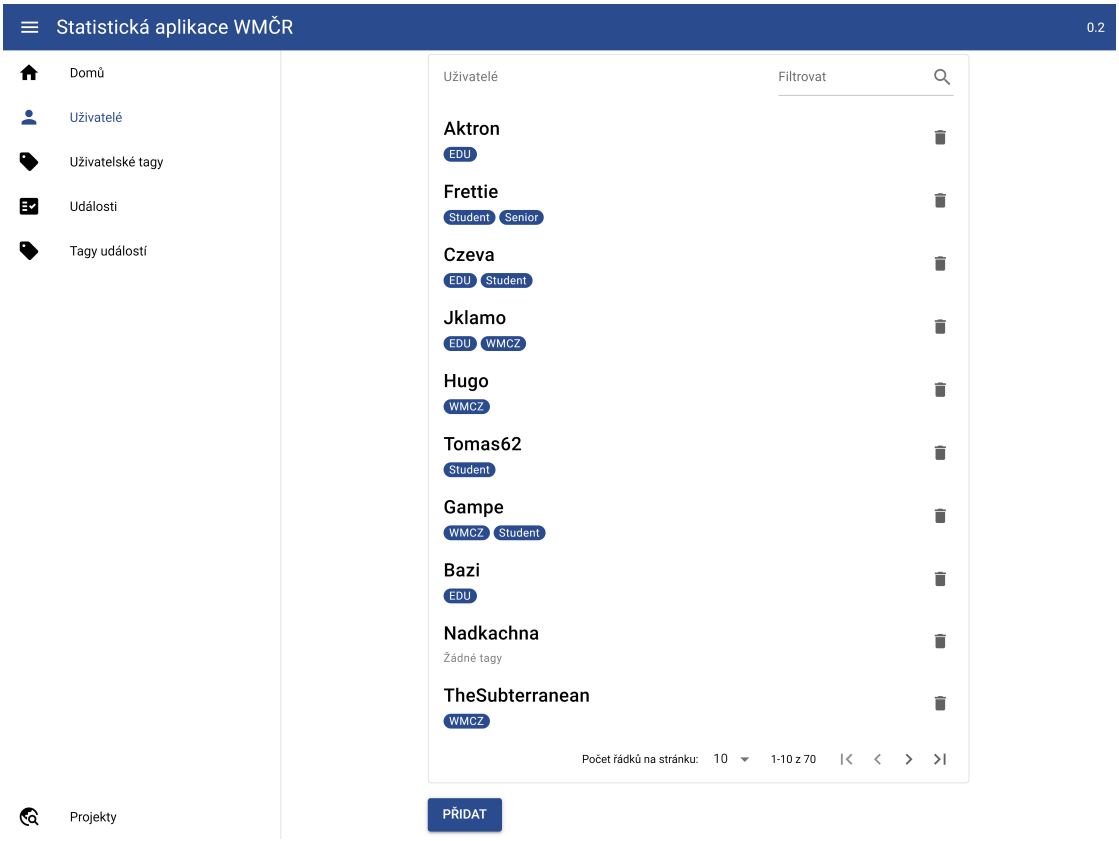
Příloha C

Diagram přiřazení uživatele k editaci dlouhodobé události

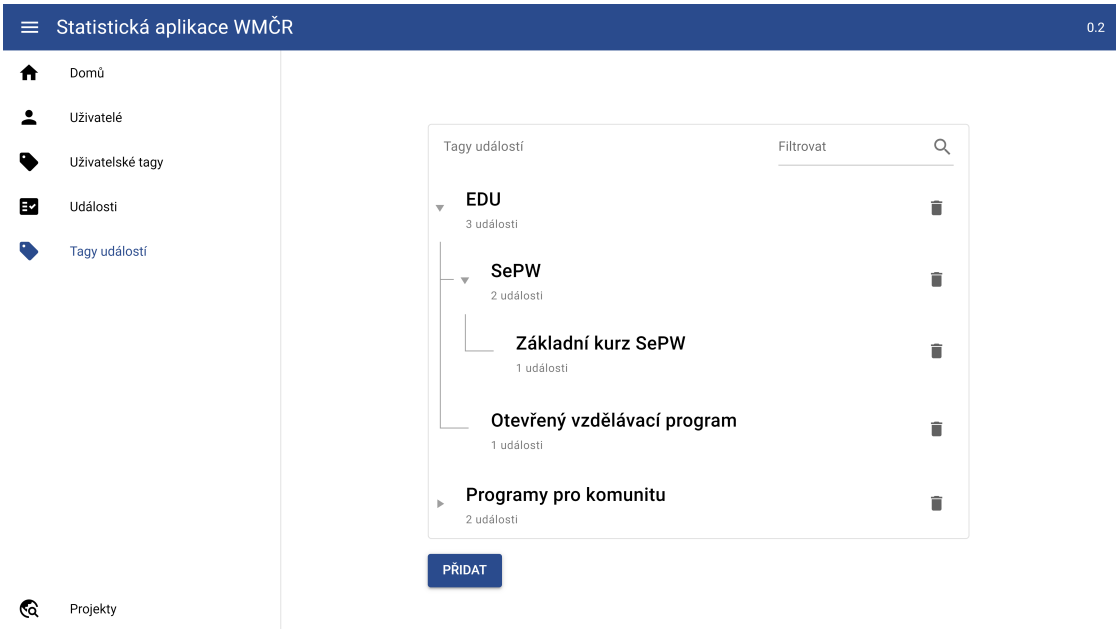


Příloha D

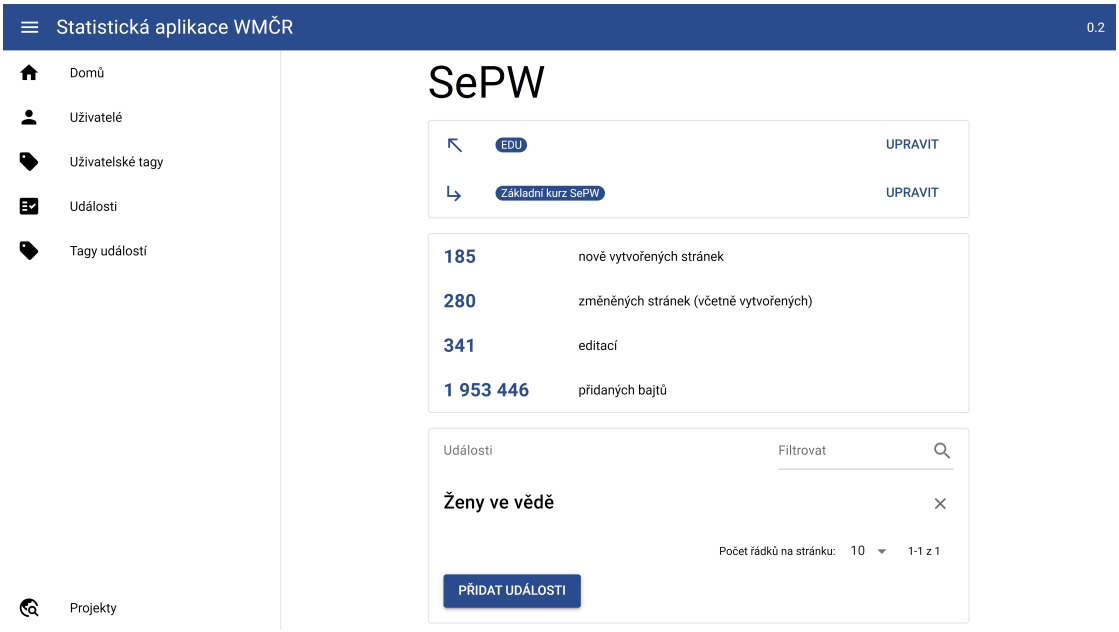
Ukázka obrazovek klientské části



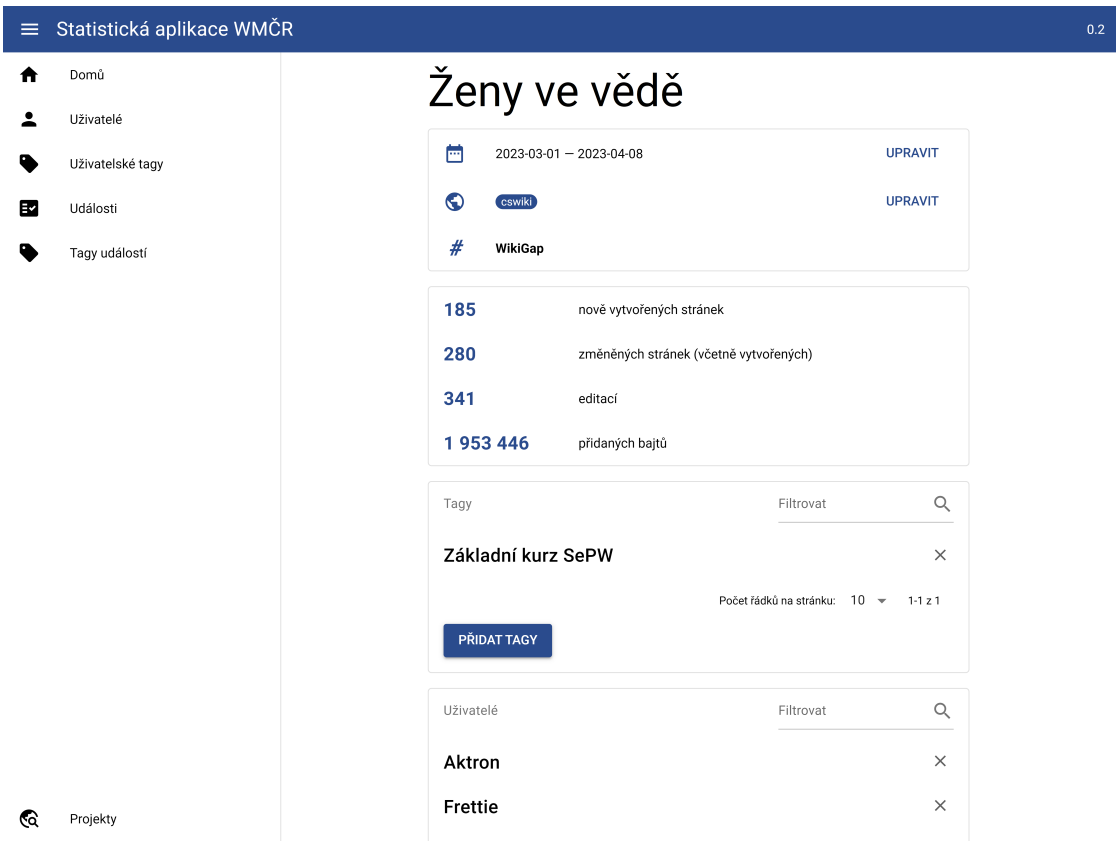
Obrázek D.1 Obrazovka zobrazující seznam uživatel včetně tagů jim přiřazených.



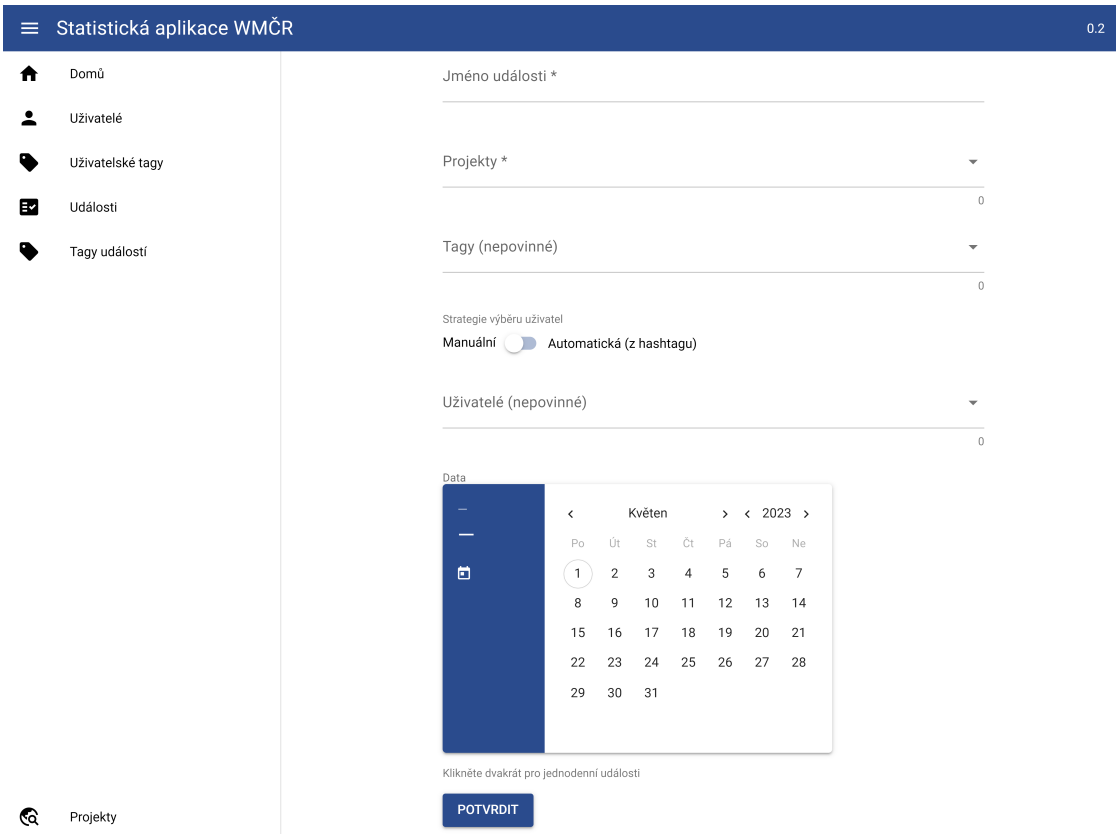
Obrázek D.2 Obrazovka zobrazující seznam tagů ve stromové struktuře. Tagy pod kořenem EDU jsou všechny rozbalené, tag Programy pro komunitu zůstává sbalený.



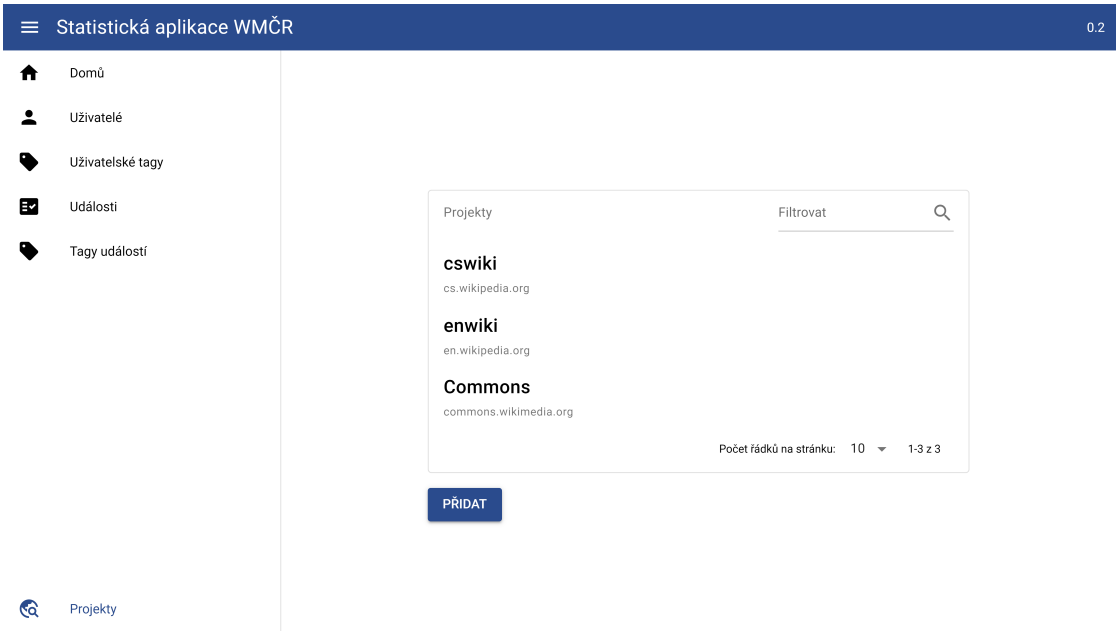
Obrázek D.3 Obrazovka ukazující detail tagu SePW. Vyobrazeny jsou jeho nadtag a podtag, události jemu přímo náležející, a dopad událostí náležícím jemu a jeho podtagům.



Obrázek D.4 Obrazovka zobrazující detail události. Vyobrazeny jsou jeho datum konání, projekty, hashtag, dopad, a přiřazené tagy a uživatelé (částečně mimo obrazovku, scrollovatelné).



Obrázek D.5 Obrazovka zobrazující formulář vytvoření události.



Obrázek D.6 Obrazovka zobrazující seznam projektů včetně jejich adres.

Bibliografie

1. *Poslání* [online]. Wikimedia Foundation, 2021 [cit. 2023-03-09]. Dostupné z: https://meta.wikimedia.org/wiki/Mission.
2. *Místní organizace Wikimedia* [online]. Wikimedia Foundation, 2023 [cit. 2023-03-09]. Dostupné z: https://meta.wikimedia.org/wiki/Wikimedia_chapters.
3. *Sites using MediaWiki/Wikimedia* [online]. Wikimedia Foundation, 2023 [cit. 2023-03-09]. Dostupné z: https://www.mediawiki.org/wiki/Sites_using_MediaWiki/Wikimedia.
4. *Wikipedie:Editaton Sucho v ČR 2022* [online]. Wikimedia Foundation, 2022 [cit. 2023-03-19]. Dostupné z: https://cs.wikipedia.org/wiki/Wikipedie:Editaton_Sucho_v_%C4%8CR_2022.
5. *Wikipedie:WikiProjekt Československo 1948–1989/Soutěž* [online]. Wikimedia Foundation, 2023 [cit. 2023-03-19]. Dostupné z: https://cs.wikipedia.org/wiki/Wikipedie:WikiProjekt_%C4%8Ceskoslovensko_1948%E2%80%931989/Sout%C4%9B%C5%BE.
6. PETERZELL, Keegan. *Single-User Login provides access to all wikis* [online]. Wikimedia Foundation, 2015 [cit. 2023-03-12]. Dostupné z: https://diff.wikimedia.org/2015/04/14/single-user-login-for-all-wikis/.
7. *Help:Unified login* [online]. Wikimedia Foundation, 2023 [cit. 2023-03-12]. Dostupné z: https://meta.wikimedia.org/wiki/Help:Unified_login.
8. *Extension:CentralAuth/API* [online]. Wikimedia Foundation, 2023 [cit. 2023-03-12]. Dostupné z: https://www.mediawiki.org/wiki/Extension:CentralAuth/API.
9. WALTON, Sam; FAUCONNIER, Sandra. *Consider addition of Wikidata tracking* [online]. Wikimedia Foundation, 2019 [cit. 2023-03-19]. Dostupné z: https://phabricator.wikimedia.org/T207029.
10. WALTON, Sam et al. *Hashtags — Meta* [online]. Wikimedia Foundation, 2022 [cit. 2023-03-25]. Dostupné z: https://meta.wikimedia.org/wiki/Hashtags.
11. MUSIKANIMAL; WALTON, Sam; CUMMINGS, John. *Add JSON API endpoint* [online]. Wikimedia Foundation, 2020 [cit. 2023-03-19]. Dostupné z: https://phabricator.wikimedia.org/T256061.
12. WEIGERS, Karl E. *Software Requirements, Second Edition*. Microsoft Press, 2003. ISBN 978-0-7356-1879-4.
13. WIKIMEDIA ČESKÁ REPUBLIKA. *Výroční zpráva 2021* [online]. 2022. [cit. 2023-03-09]. Dostupné z: https://upload.wikimedia.org/wikipedia/commons/f/f6/Vyrocni_zprava_WM_CZ_2021_online.pdf.

14. HAM, Gary A. Four Roads to Use Case Discovery: There Is a Use (and a Case) for Each One. *CrossTalk: The Journal of Defense Software Engineering* [online]. Prosinec 1998, roč. 11, č. 12 [cit. 2023-04-06]. ISSN 2160-1577, ISSN 2160-1593. Dostupné z: https://web.archive.org/web/20120805100621/http://www.crosstalkonline.org/storage/issue-archives/1998/199812/199812-Ham.pdf. Archiv zaniklé stránky v podobě z 2012-08-05.
15. LARMAN, Craig. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process, 2nd edition*. Prentice Hall, 2001. ISBN 0-13-092569-1.
16. RICHARDS, Mark. *Software Architecture Patterns, 2nd Edition*. O'Reilly Media, Inc., 2022. ISBN 978-1-098-13427-3.
17. GOSLING, James et al. *The Java® Language Specification, Java SE 19 Edition* [online]. Oracle Corporation, 2022 [cit. 2023-03-19]. Dostupné z: https://docs.oracle.com/javase/specs/jls/se19/jls19.pdf.
18. LINDHOLM, Tim et al. *The Java® Virtual Machine Specification, Java SE 19 Edition* [online]. Oracle Corporation, 2022 [cit. 2023-03-19]. Dostupné z: https://docs.oracle.com/javase/specs/jls/se19/jls19.pdf.
19. *Stack Overflow 2022 Software Developer Survey* [online]. Stack Exchange, Inc., 2022 [cit. 2023-03-14]. Dostupné z: https://survey.stackoverflow.co/2022/#technology-most-popular-technologies.
20. *Oracle Java SE Support Roadmap* [online]. Oracle Corporation, 2023 [cit. 2023-04-15]. Dostupné z: https://www.oracle.com/java/technologies/java-se-support-roadmap.html.
21. *Spring Framework Overview* [online]. VMware, Inc., 2023 [cit. 2023-03-14]. Dostupné z: https://docs.spring.io/spring-framework/docs/current/reference/html/overview.html.
22. *Spring Boot Overview* [online]. VMware, Inc., 2023 [cit. 2023-03-14]. Dostupné z: https://docs.spring.io/spring-boot/docs/current/reference/html/getting-started.html.
23. *Repozitář Spring Framework* [online]. Github, Inc., 2023 [cit. 2023-03-19]. Dostupné z: https://github.com/spring-projects/spring-framework.
24. MIKOWSKI, Michael S.; POWELL, Josh C. *Single Page Web Applications: JavaScript end-to-end*. Manning Publications Company, 2013. ISBN 978-1-617-29075-6.
25. *JavaScript reference* [online]. Mozilla Corporation, 2023 [cit. 2023-03-24]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript.
26. Q-SUCCESS DI GELBMANN GMBH. *Usage statistics of JavaScript as client-side programming language on websites* [online]. 2023. [cit. 2023-03-24]. Dostupné z: https://w3techs.com/technologies/details/cp-javascript/.
27. YOU, Evan et al. *Introduction — Vue.js* [online]. 2023. [cit. 2023-03-24]. Dostupné z: https://vuejs.org/guide/introduction.html.
28. YOU, Evan et al. *Frequently Asked Questions — Vue.js* [online]. 2023. [cit. 2023-03-24]. Dostupné z: https://vuejs.org/about/faq.html.
29. JOSHI, Mohit. *Angular vs React vs Vue: Core Differences* [online]. 2022. [cit. 2023-03-24]. Dostupné z: https://www.browserstack.com/guide/angular-vs-react-vs-vue.
30. STOENESCU, Razvan et al. *Introduction to Quasar* [online]. PULSARDEV SRL, 2023 [cit. 2023-03-24]. Dostupné z: https://quasar.dev/introduction-to-quasar/.
31. *Gradle User Manual* [online]. Gradle, Inc., 2022 [cit. 2023-03-19]. Dostupné z: https://docs.gradle.org/8.0.2/userguide/userguide.html.

32. *Docker overview* [online]. Docker, Inc., 2023 [cit. 2023-03-24]. Dostupné z: https://docs.docker.com/get-started/overview/.

33. *What's The Difference Between Containers And Virtual Machines?* [Online]. Amazon Web Services, Inc., 2023 [cit. 2023-03-24]. Dostupné z: https://github.com/axios/axios.

34. MOROTE, Eduardo San Martin et al. *Getting Started — Vue Router* [online]. 2023. [cit. 2023-03-24]. Dostupné z: https://router.vuejs.org/guide/.

35. *Axios* [online]. npm, Inc., 2023 [cit. 2023-03-24]. Dostupné z: https://www.npmjs.com/package/axios.

36. *Axios* [online]. GitHub, Inc., 2023 [cit. 2023-03-24]. Dostupné z: https://github.com/axios/axios.

37. LOURIDAS, Panagiotis. Static code analysis. *IEEE Software*. 2006, roč. 23, č. 4, s. 58–61. Dostupné z DOI: 10.1109/MS.2006.114.

38. BLACK, Rex. *Managing the Testing Process: Practical Tools and Techniques for Managing Hardware and Software Testing, 2nd Edition*. Wiley, 2002. ISBN 978-0471223986.

Obsah přiloženého archivu

```
readme.txt ..... stručný popis obsahu média
src
    impl ..... zdrojové kódy implementace
    thesis ..... zdrojová forma práce ve formátu LaTeX
text ..... text práce
    thesis.pdf ..... text práce ve formátu PDF
```