Vysoká škola ekonomická v Praze

Fakulta informatiky a statistiky

Vývoj aplikace s využitím frameworku Vaadin

DIPLOMOVÁ PRÁCE

Studijní program: Aplikovaná informatika

Studijní obor: Informační systémy a technologie

Autor: Bc. David Galaš

Vedoucí diplomové práce: Ing. Jarmila Pavlíčková, Ph.D.

Praha, duben 2019

Prohlášení

Prohlašuji, že jsem diplomovou práci Vývoj aplikace s využitím frameworku Vaadin vypracoval samostatně za použití v práci uvedených pramenů a literatury.

V Praze dne 29. dubna 2019 . . . . . .

Bc. David Galaš

Poděkování

Rád bych poděkoval Ing. Jarmile Pavlíčkové, Ph.D. za její cenné rady a velice ochotný přístup během vedení mé diplomové práce. Poděkovat bych chtěl také své rodině za inspiraci a podporu při psaní této práce.

Abstrakt

Diplomová práce je zaměřena na zmapování možností frameworku Vaadin pro komunikaci s REST API a zpracování získaných dat. Schopnosti frameworku jsou ověřeny získáním zdrojových dat z vybraných veřejných REST API a jejich následným vizuálním zpracováním v rámci vyvinuté ukázkové aplikace. Teoretická část práce vymezuje jednotlivé komponenty Vaadinu, které jsou využity pro získání příslušných dat, a dále popisuje aspekty a parametry zvolených veřejných REST API. V analytické části práce je provedena kompletní analýza ukázkové aplikace, která v rámci práce slouží k praktickému ověření zkoumaných vlastností frameworku Vaadin. Analýza zahrnuje specifikaci funkčních a nefunkčních požadavků na aplikaci, souvisejících případů užití, globální a detailní architekturu ukázkové aplikace, testovací případy a také akceptační kritéria. Praktickou část představuje samotný vývoj ukázkové aplikace s využitím jazyka Java a související konfigurace aplikačního serveru. Daná aplikace je vyvinuta na základě provedené analýzy, přičemž pro komunikaci s vybranými REST API a následné zpracování získaných dat využívá poznatky z teoretické části práce. Závěr práce následně shrnuje získané poznatky z oblasti využití frameworku Vaadin, určuje, jak byly splněny jednotlivé stanovené dílčí cíle práce a nastiňuje možnosti budoucího využití této práce.

Klíčová slova

Vaadin, Java, framework, REST API, NASA, Mapbox.

Abstract

The aim of this thesis is to verify the possible use of the Vaadin framework for internet communication with REST APIs and for user data procession. The Vaadin framework abilities are verified by data querying from selected REST APIs and by consecutive procession of queried data within the given sample application. The theoretical part describes necessary components for building a web user interface and for providing the data to the user interface. There are also mentioned some of the most important aspects and parameters of REST APIs selected to provide user data. The analytical part contains a thorough analysis for the sample application which includes a functional and non-functional requirements specification and related use cases, global and detail architecture for the sample application and the set of test cases with acceptance criteria. The practical part is focused on development of the sample application including necessary configuration of chosen application server which was made to run the server. The sample application is based on carried out analysis while the communication with selected REST APIs and consecutive data procession is made using the knowledge from theoretical part of the thesis. The conclusion of this thesis summarizes the findings made in theoretical part, shows how the intermediate objectives have been achieved and outlines the possible future use of this thesis.

Keywords

Vaadin, Java, framework, REST API, NASA, Mapbox.

Obsah

1 Úvod ... 11
1.1 Struktura práce ... 12
1.2 Cíle práce ... 12
1.3 Metody dosažení cíle ... 13
1.4 Důvod výběru tématu a vymezení práce ... 13
1.5 Předpoklady a omezení ... 13
1.6 Výstupy práce a očekávaný přínos ... 14
1.7 Komentovaná rešerše použitých zdrojů ... 14
2 Framework Vaadin ... 16
2.1 Historie verzí ... 16
2.2 Možnosti využití ... 18
2.3 Vlastnosti frameworku ... 19
2.3.1 Architektura webové aplikace ... 20
2.3.2 Bindování dat ... 23
2.3.3 Kompatibilita s programovacími jazyky ... 25
2.3.4 Podporované prohlížeče ... 25
2.4 Klíčové komponenty ... 26
2.4.1 Grid ... 26
2.4.2 Field ... 27
2.4.3 Layout ... 29
2.5 Motivy ... 30
2.6 Recenze odborníků a uživatelů ... 31
2.7 Porovnání s konkurenčními frameworky ... 32
2.7.1 Spring ... 33
2.7.2 Play ... 34
2.7.3 Wicket ... 35
2.7.4 Grails ... 36
2.7.5 Vert.X ... 37
2.7.6 Shrnutí porovnávaných frameworků ... 38
3 Komunikace s REST API ... 39
3.1 Základní vlastnosti ... 39
3.2 Kritéria výběru ... 40
3.3 NASA API ... 40
3.4 Mapbox API ... 41
4 Analýza a návrh aplikace ... 43
4.1 Technická charakteristika aplikace ... 43

4.2 Uživatelská charakteristika aplikace ... 43
4.3 Použitá metodika ... 43
4.4 Požadavky na aplikaci ... 46
4.4.1 Funkční požadavky ... 46
4.4.2 Nefunkční požadavky ... 47
4.5 Případy užití ... 49
4.6 Architektura aplikace ... 52
4.6.1 Globální architektura ... 52
4.6.2 Detailní architektura ... 53
4.7 Testování aplikace ... 56
4.8 Podmínky akceptace ... 56
5 Vývoj aplikace ... 58
5.1 Postup vývoje aplikace ... 58
5.2 Konfigurace aplikačního serveru ... 59
5.3 Struktura aplikace ... 61
5.4 Logování ... 61
6 Testování aplikace ... 63
6.1 Sumarizace provedených testovacích případů ... 63
6.2 Vyhodnocení akceptačních kritérií ... 65
Závěr ... 66
Seznam citované literatury ... 68
Příloha A: Testovací případy ... 74
Příloha B: Použité zkratky v názvech testovacích případů ... 87

Seznam obrázků

Obrázek 1: Časový plán vydávání nových verzí Vaadinu (Vaadin Ltd., 2019c) 18
Obrázek 2: Architektura vaadinovské aplikace (Vaadin Ltd., 2019e) 20
Obrázek 3: Přístup komponent k DOM (Vaadin Ltd., 2019e) 21
Obrázek 4: Grid (Vaadin Ltd., 2019ch)..... 26
Obrázek 5: Field komponenty (Vaadin Ltd., 2019j) 28
Obrázek 6: Implementace rozhraní HasValue (Vaadin Ltd., 2019j)..... 28
Obrázek 7: Schéma komponent typu Layout (Vaadin Ltd., 2019k)..... 30
Obrázek 8: Životní cyklus projektu v metodice OpenUP (zdroj: Balduino, 2007, s. 3) 45
Obrázek 9: Diagram případů užití pro ukázkovou aplikaci (zdroj: autor) 50
Obrázek 10: Globální architektura ukázkové aplikace (zdroj: autor) 53
Obrázek 11: Detailní architektura ukázkové aplikace (zdroj: autor)..... 54

Seznam tabulek

Tabulka 1: Podporované prohlížeče a jejich verze (zdroj: Vaadin Ltd., 2019h) 25
Tabulka 2: Parametry dotazu na NASA API (zdroj: NASA, 2019b) 41
Tabulka 3: Parametry dotazu na Mapbox Geocoding API (zdroj: Mapbox, 2019c) 42
Tabulka 4: Funkční požadavky (zdroj: autor) 46
Tabulka 5: Nefunkční požadavky (zdroj: autor) 47
Tabulka 6: Případ užití FP01 (zdroj: autor) 51
Tabulka 7: Případ užití FP02 (zdroj: autor) 51
Tabulka 8: Případ užití FP03 (zdroj: autor) 52
Tabulka 9: Stupně závažnosti chyb pro akceptaci (zdroj: IEEE, 2010, s. 8) 56
Tabulka 10: Akceptační podmínky pro ukázkovou aplikaci (zdroj: autor) 57
Tabulka 11: Přehled provedených testovacích případů (zdroj: autor) 63
Tabulka 12: Vyhodnocení definovaných akceptačních kritérií (zdroj: autor) 65

Seznam ukázek kódu

Ukázka kódu 1: Komponenta nejvyšší úrovně abstrakce (Vaadin Ltd., 2019e) 22
Ukázka kódu 2: Příklad komponenty nižší úrovně abstrakce (Vaadin Ltd., 2019e) 22
Ukázka kódu 3: Šablona komponenty nejnižší úrovně abstrakce (Vaadin Ltd., 2019e) 23
Ukázka kódu 4: Třída pro zapouzdření šablony z ukázky kódu 3 (Vaadin Ltd., 2019e) 23
Ukázka kódu 5: Komponenta binder pro data (Vaadin Ltd., 2019f) 24
Ukázka kódu 6: Binder pro Plain Old Java objekt (Vaadin Ltd., 2019g) 24
Ukázka kódu 7: Seznam osob jako zdroj dat pro grid (Vaadin Ltd., 2019ch) 27
Ukázka kódu 8: Data provider jako zdroj dat pro grid (Vaadin Ltd., 2019i) 27
Ukázka kódu 9: Validator (Vaadin Ltd., 2019j) 29
Ukázka kódu 10: Aplikace motivu (Vaadin Ltd., 2019n) 31
Ukázka kódu 11: Routing (Vaadin Ltd., 2019o) 31
Ukázka kódu 12: Obsah konfiguračního souboru tomcat-users (zdroj: autor) 59
Ukázka kódu 13: Plugin pro vzdálený deployment na Apache Tomcat (zdroj: autor) 60
Ukázka kódu 14: Konfigurace pro vzdálené debugování (zdroj: autor) 60
Ukázka kódu 15: Obsah konfiguračního souboru pro Log4j (zdroj: autor) 62

1 Úvod

Pro analýzu a zpracování dat existuje mnoho různých nástrojů a frameworků. Pro práci s daty byl v rámci této práce vybrán framework Vaadin, což je moderní webový framework, který podporuje jazyk Java. Tento framework, vytvořený stejnojmennou společností, je dle (Vaadin Ltd., 2019a) určen k vytváření přehledných UI pro webové aplikace s využitím jednotlivých komponent, které lze vzájemně kombinovat a vytvářet tak uživatelsky přívětivá uživatelská rozhraní, která jsou schopna pracovat s rozsáhlou množinou dat, získaných z externích zdrojů. Jedná se tak o moderní framework, který využívá moderní programátorské přístupy a technologie a který zároveň upřednostňuje jednoduchost a použitelnost před robustností, což byl také jeden z důvodů jeho výběru pro účely této práce. Pro získání zdrojových dat, zobrazovaných ve výsledné aplikaci, disponuje tento framework speciálními backendovými komponenty, které umožňují komunikaci s databází či s externími REST API, čehož je v této práci využíváno. Vzhledem k tomu, že technologie REST je v oblasti webových služeb již velmi rozšířena a ustálena, byla pro potřeby této práce uvažována pouze taková veřejná API, která pracují na principu REST. Za účelem získání zdrojových dat pro zobrazení a následné zpracování byla v této práci na základě definovaných kritérií vybrána dvě veřejná REST API, ze kterých byla daná zdrojová data prostřednictvím dotazů získávána. Prvním je veřejné REST API od společnosti NASA, které poskytuje detailní vědecká data o Zemi blízkých objektech. Druhé vybrané REST API je provozováno společností Mapbox a umožňuje na základě zadaného názvu konkrétního města získat podrobné geokódovací údaje, včetně doplňujících informací o daném místě. Data získaná z těchto uvedených REST API jsou v aplikaci pomocí frameworku Vaadin dále zpracovávána a převáděna do čitelné podoby pro běžného uživatele. Celkově tak vzniká jednoduchá a přehledná ukázková aplikace, využívající technologií jazyka Java, která pracuje s daty od vybraných REST API.

1.1 Struktura práce

Práce je dle obsahu rozdělena na teoretickou, analytickou a praktickou část. V teoretické části je nejprve popsán zvolený framework Vaadin, včetně stručné historie jeho vývoje, možností jeho využití a klíčových komponent, které jsou v praktické části pro vytvoření aplikace využívány. Kromě uvedených vlastností je v teoretické části obsaženo také kritické porovnání s konkurenčními frameworky na základě předem vybraných vlastností, jehož účelem je zařadit framework Vaadin do širšího kontextu v rámci jednotlivých dostupných frameworků pro jazyk Java. Dále jsou v této části detailně popsána obě zvolená REST API včetně jednotlivých parametrů potřebných pro získání dat a také kritérií, podle kterých byla daná API vybrána. Celkově tak teoretická část práce představuje znalostní podklad pro následující části práce a uvádí potřebné technické pojmy. Analytická část práce je zaměřena na specifikaci jednotlivých funkčních a nefunkčních požadavků na vyvíjenou aplikaci, dále na definování souvisejících případů užití pro danou aplikaci a návrh podoby její globální a detailní architektury. Dále je v analytické části obsažena také sada testovacích případů, která slouží k verifikaci vytvořené aplikace, aby bylo dosaženo potřebné úrovně kvality aplikace. Společně s testovacími případy byla stanovena také akceptační kritéria stanovující maximální přípustný počet chyb odstupňovaných podle závažnosti. Praktickou část pak představuje samotná aplikace, včetně popisu postupu jejího vývoje, provedených konfigurací vybraného aplikačního serveru, jejího průběžného testování pomocí vytvořené sady testovacích případů a závěrečného vyhodnocení testů na základě definovaných akceptačních kritérií.

1.2 Cíle práce

Cílem této práce je zmapovat možnosti využití frameworku Vaadin pro komunikaci s REST API a následné vizuální zpracování získaných dat v rámci ukázkové aplikace. Práce je rozdělena na teoretickou, analytickou a praktickou část, přičemž každá z uvedených částí obsahuje jednotlivé dílčí cíle, které slouží k naplnění celkového zadaného cíle práce. Teoretická část slouží především k seznámení s frameworkem Vaadin, představení jeho hlavních komponent a zařazení frameworku do širšího kontextu. Druhým dílčím cílem teoretické části práce je představení jednotlivých vybraných veřejných REST API a jejich klíčových vlastností pro tuto práci. Cílem analytické části práce je vytvořit podklady pro vývoj ukázkové aplikace v podobě detailní specifikace funkčních a nefunkčních požadavků, souvisejících případů užití, základní architektury z globálního a detailního pohledu a sady testovacích případů k aplikaci, včetně definování akceptačních kritérií. Praktická část má následně za cíl vytvoření ukázkové webové Java aplikace, využívající definovaných komponent z teoretické části a analytických podkladů z analytické části práce, která slouží k demonstraci uvedených vlastností v teoretické části. Aplikace tak kombinací poznatků z teoretické a analytické části práce celkově naplňuje definovaný cíl této práce.

1.3 Metody dosažení cíle

Základní metodou pro získání informací o frameworku Vaadin je studium oficiální dokumentace vydané k frameworku Vaadin, včetně odborných článků a publikací, zabývajících se daným frameworkem. Dále je pro splnění zadaného cíle klíčová analýza dokumentace k jednotlivým zvoleným REST API, aby bylo možné s nimi pracovat a získávat zdrojová data pro vizuální zpracování. Pro následnou analýzu aplikace je využito specifikované, modifikované metodiky, popsané v práci, přičemž její aplikace byla provedena s využitím osobních odborných znalostí získaných praxí v oboru. Vývoj samotné ukázkové aplikace je proveden podle specifikované metodiky za opětované aplikace vlastních odborných znalostí v oboru.

1.4 Důvod výběru tématu a vymezení práce

Důvodem výběru daného tématu byla především několikaletá pracovní zkušenost s programováním v Javě a frameworkem Vaadin, včetně obecného zájmu o analýzu a vývoj aplikací a souvisejících metodik. Framework Vaadin byl vybrán z důvodu jeho vlastností, které jsou zcela vhodné pro účely této práce, a také kvůli zatím poměrně malému pokrytí tohoto frameworku v závěrečných pracích, které se týkají programování v jazyku Java. Vzhledem k rozsáhlým možnostem, které framework Vaadin nabízí, bylo nutné omezit se pouze na komponenty, které se starají o vytváření přehledného uživatelského rozhraní obsahující formuláře a tabulky, a také na komponenty, které zajišťují potřebná data zobrazená na frontendu. Co se týče výběru veřejných REST API, bylo nutné vybrat taková REST API, která poskytují data, která jsou relevantní pro využití v akademické práci a která tato data poskytují bezplatně v souladu se stanovenými licenčními podmínkami.

1.5 Předpoklady a omezení

Po teoretické stránce vyžaduje práce alespoň základní znalosti objektového programování, ideálně jazyka Java. Jednotlivé komponenty jsou sice jasně definovány v teoretické části práce, nicméně jsou předpokládány určité základní znalosti ohledně programování. Analytická část vyžaduje znalost objektového modelování pro pochopení architektury aplikace a její funkcionality, a také základní znalost v oboru softwarového testování pro testování vytvořené aplikace pomocí přiložené sady testovacích případů. Praktická část počítá se základními znalostmi konfigurace aplikačního serveru Apache Tomcat a také se znalostmi konfigurace aplikačního logování s využitím fasády Slf4j.

Hlavním omezením této práce je především nedostatek odborné literatury, zabývající se frameworkem Vaadin, a tudíž nutnost čerpání informací z oficiální dokumentace k frameworku, vydávané autorskou společností frameworku. Dalším omezením této práce je nutnost demonstrace uvedených vlastností frameworku pouze na vybraných komponentách, jelikož rozsáhlé možnosti, jimiž framework Vaadin disponuje, dalece přesahují potřeby a cíle této práce. Za omezení lze také považovat zúžení výběru veřejných

REST API pro získávaní dat, která budou zobrazována na frontendu, jelikož by z hlediska implementace bylo velice časově a technicky náročné propojit větší počet daných REST API s ukázkovou aplikací. Také v rámci samotných vybraných REST API bylo při vývoji aplikace nutné provést určitá opatření, omezující možnosti těchto REST API na úroveň, která odpovídá zadaným cílům této práce.

1.6 Výstupy práce a očekávaný přínos

Primárním výstupem této práce je funkční ukázková aplikace, napsaná v jazyce Java, využívající framework Vaadin k získávání a zpracování dat od vybraných REST API. Sekundárním výstupem je sada testovacích případů, sloužících k verifikaci vytvořené aplikace. Očekávaným přínosem této práce je především zmapování možností frameworku Vaadin při získávání dat od externích REST API a jeho využití k dalšímu zpracování těchto dat. Dalším přínosem práce je také výběr veřejných REST API na základě specifikovaných kritérií, která jsou vhodná pro interakci s vybraným frameworkem Vaadin. Možným budoucím přínosem této práce je také možnost rozšíření či úpravy vyvinuté aplikace pro její využití v odborných pracích na příbuzná témata, která jsou realizována s využitím frameworku Vaadin.

1.7 Komentovaná rešerše použitých zdrojů

Rešerše dostupných akademických zdrojů ukázala, že v rámci Vysoké školy ekonomické existují pouze dvě závěrečné práce, které obsahují klíčové slovo Vaadin. První prací je bakalářská práce od autora (Kozák, 2016), která se zabývá rozborem expertního systému NEST a Vaadin využívá jako grafické uživatelské rozhraní nad bází znalostí pro daný expertní systém. Framework Vaadin je v této práci rozebírán pouze povrchově, přičemž je nastíněna jeho architektura, hierarchie komponent pro uživatelské rozhraní a možnosti práce s CSS. Druhá nalezená bakalářská práce od autora (Černý, 2013) se zabývá problematikou sémantického webu a souvisejícími transformacemi ontologií a framework Vaadin využívá pro vytvoření základu uživatelského rozhraní k navrhovaného webovému editoru transformačních zdrojů. V rámci této práce je opět stručně popsána architektura vaadinovské aplikace, dále je stručně vysvětlena tvorba uživatelského rozhraní a je provedeno základní zhodnocení rozebíraného frameworku. Obě bakalářské práce se tak věnují Vaadinu pouze okrajově a nezacházejí příliš do detailů, jelikož využívají Vaadin pouze jako doplněk k naplnění stanovených cílů. Co se týče přístupu REST, bylo nalezeno poměrně velké množství diplomových i bakalářských prací, které alespoň částečně rozebírají jeho problematiku a principy fungování. Také bylo nalezeno několik závěrečných prací, které se věnují návrhu či vývoji REST služby a které rovněž obsahují popis základních HTTP metod pro komunikaci s REST API. Příkladem takové práce je diplomová práce od (Koch, 2013), kde se autor věnuje katalogizaci webových služeb v rámci architektury REST. V práci mimo jiné rozebírá uvedené metody pro komunikaci s REST API a uvádí jejich porovnání s CRUD operacemi. Celkově je tak téma přístupu REST z akademického hlediska v současné době velmi dobře pokryto.

Odborná literatura se bohužel zabývá pouze tématem architektury REST, jak je patrné u odborných knižních publikací od (Richardson a Amundsen, 2013) a (Webber, Parastatidis a Robinson, 2010), které podrobně rozebírají stavbu REST API, uvádějí důležité vlastnosti, na které je třeba brát zřetel a výše uvedené doplňují příklady z praxe. Z knihy od autorů (Richardson a Amundsen, 2013) bylo využito kapitoly zabývající se metodami protokolu HTTP a jejich významu v souvislosti s REST API, a také úvodní kapitoly pro definování přístupu REST. Druhá kniha od kolektivu autorů (Webber, Parastatidis a Robinson, 2010) byla využita především pro vytvoření analogie jednotlivých HTTP metod s CRUD operacemi. Pro doplnění informací o průběhu komunikace s REST API byl využit odborný článek od autorky (Monus, 2019), ve kterém jsou uvedeny jednotlivé souborové formáty, které lze k této komunikaci využít, a také článek od (Liew, 2018), ve kterém autor uvádí postup vytváření dotazu na REST API.

Ohledně frameworku Vaadin je situace s odbornou literaturou poněkud komplikovanější, jelikož kromě (Vaadin Ltd., 2017), představující online dokumentaci v knižní podobě, která navíc není pro účely této práce ve všech ohledech dostatečně aktuální, neexistuje vysloveně odborná vědecká literatura, zabývající se novějšími verzemi Vaadinu. Z toho důvodu bylo za účelem získání informací o základních vlastnostech frameworku, historii jeho verzí, možnostech využití, architektuře aplikace a stavby jednotlivých komponent využito aktuální online dokumentace, která představuje citované zdroje (Vaadin Ltd, 2019a-o). Tato dokumentace je plně aktuální a slouží jako cenný zdroj pro vývojáře, pracující s frameworkem Vaadin. Pro vytvoření kritického porovnání konkurenčních frameworků byla provedena analýza odborných článků (Javapipe, 2019), (Koblentz, 2013) (Malhotra, 2018) a (Monus, 2018), obsahujících informace o nejvyužívanějších frameworkách pro Javu, doplněných o informace přímo od autorů jednotlivých frameworků, ze kterých bylo čerpáno v případě porovnávaných vlastností daných frameworků.

Pro získání potřebných informací o využívaných veřejných REST API bylo čerpáno přímo z oficiálních webových stránek těchto REST API, konkrétně (NASA, 2019a) a (Mapbox, 2019a). Z těchto webů byly následně čerpány veškeré potřebné technické informace pro komunikaci s danými REST API, jako jsou přístupové tokeny, parametry pro vytváření dotazů a použitelné HTTP metody.

Finální aplikace, vytvořená na základě teoretických podkladů získaných z výše popsaných zdrojů a analytických podkladů, čerpajících převážně z vlastní předchozí praxe v oboru, využívá metodiku uvedenou v (Balduino, 2007), která byla pro potřeby této práce upravena a přizpůsobena. Vzhledem k tomu, že pro plné zprovoznění aplikace bylo nutné nakonfigurovat také příslušný aplikační server, bylo v rámci vývoje aplikace čerpáno také z oficiální dokumentace k serveru Apache Tomcat, dostupné na webu (Apache Software Foundation, 2019c).

2 Framework Vaadin

Vaadin je moderní framework zaměřený především na tvorbu přehledných UI k webovým aplikacím. Původně byl součástí open-source knihovny s názvem Millstone verze 3, vyvinuté stejnojmenným týmem v roce 2002. Postupně byl dále rozvíjen, až v roce 2009 došlo k přejmenování na framework Vaadin (Vaadin Ltd., 2017, s. 8). Kompletní podrobná historie verzí tohoto frameworku je uvedena v následující podkapitole.

Tento framework byl vybrán z důvodu jednoduchosti práce s připravenými komponentami při vývoji aplikací, kdy v případě nutnosti provádění úprav v aplikaci lze jednotlivé komponenty mezi sebou jednoduše nahrazovat, a také kvůli obecné přehlednosti vytvářeného uživatelského rozhraní, které má moderní vzhled a působí velmi intuitivně. K Vaadinu je dle (Vaadin Ltd., 2017) dále dostupná přehledná dokumentace, obsahující kromě obvyklého popisu jednotlivých komponent a principu jejich fungování také tutoriály a naučná demo videa pro rychlejší pochopení základních principů. Také zde působí rozsáhlá a velice aktivní komunita, pořádající konference a webináře a poskytující podporu vývojářům na fórech. Výhodou je také plná integrace technologie Maven, která umožňuje snadný přístup k potřebným komponentám. Jedná se tak o framework, který využívá moderní přístupy a zároveň poskytuje maximální podporu svým uživatelům, což je z hlediska vývojáře velice přínosné.

Co se týče licencování, Vaadin je dle (Vaadin Ltd., 2019b) obecně šířen pod licencí Apache License verze 2.0, kromě dokumentace, která spadá pod licenci Creative Commons CC-BY-ND verze 2.0. Komponenty frameworku se v základě dělí na dva druhy a mají odlišný způsob licencování. První jsou dle (Vaadin Ltd., 2019b) takzvané Core komponenty, šířené pod licencí Apache License, verze 2.0, které jsou vedeny jako open-source. Těch je možné libovolně využívat, dále je rozšiřovat či vylepšovat a distribuovat jako součást vlastního softwaru. Druhým typem komponentů jsou dle (Vaadin Ltd., 2019b) Pro komponenty, které jsou vydávány pod licencí Commercial Vaadin Add-on License. Ta zjednodušeně označuje tyto komponenty jako placené a jejich využívání se řídí podle stanoveného ceníku. Vzhledem k poměrně vysokým cenám potřebných licencí jsou pro účely této práce využívány pouze Core komponenty, které jsou volně dostupné.

Obecně se jedná o poměrně rozsáhlý framework, jehož možnosti dalece přesahují potřeby této práce. Pro naplnění zadaného cíle práce tak byly využity pouze některé komponenty, které jsou v práci dále podrobněji rozebrány.

2.1 Historie verzí

Následující seznam zachycuje stručnou historii verzí frameworku Vaadin, včetně krátkého popisu každé verze, seřazenou chronologicky (Vaadin Ltd., 2019b).

2002 – První verze, vydaná jako součást knihovny Millstone verze 3, vyvinuté společností IT Mill.

2006 – Vydání další generace knihovny pod názvem IT Mill Toolkit verze 4, obsahující zcela nový AJAX engine, což umožnilo tvorbu ajaxových Single-Page aplikací.

2007 – Vydání IT Mill Toolkit verze 5, přinášející další možnosti pro tvorbu ajaxových aplikací. Klientská strana vytvářeného UI byla přepsána pomocí nástroje Google Web Toolkit, což umožnilo použití Javy jak na straně klienta, tak na straně serveru. To výrazně zjednodušilo vytváření nových komponent a úpravu těch stávajících.

2009 – IT Mill Toolkit přejmenován na Vaadin Framework a vydán jako verze 6, společnost IT Mill přejmenována na Vaadin Ltd. Byl vytvořen speciální plugin pro Eclipse, což usnadnilo vytváření vaadinovských projektů.

2010 - Byl vytvořen Vaadin Directory neboli místo pro hromadnou správu jednotlivých komponent a pluginů, což mělo za následek velký nárůst v počtu komponent, dostupných vývojářům. Celkově došlo také k nárůstu uživatelské komunity.

2013 – Vydání Vaadin verze 7, přinášející plnohodnotnou integraci nástroje Google Web Toolkit do Vaadinu, umožňující snadnější tvorbu widgetů. Také byla zavedena podpora Sass pro rozšíření možnosti stylizace webových uživatelských rozhraní (Ekblad, 2013).

2017 – Vydání Vaadin verze 8, obsahující rozsáhlé změny v implementaci formulářů. Data jsou od této verze bindována k formulářovým polím prostřednictvím komponenty Binder, která je v práci dále popsána, a výsledná data jsou zobrazována v Gridu neboli komponentě, vytvářející multifunkční, interaktivní tabulku, rovněž v práci detailně popsané. Došlo také k výkonnostní optimalizaci procesu bindování dat, umožňující efektivnější využívání paměti a výkonu procesoru. (Tahvonen, 2017).

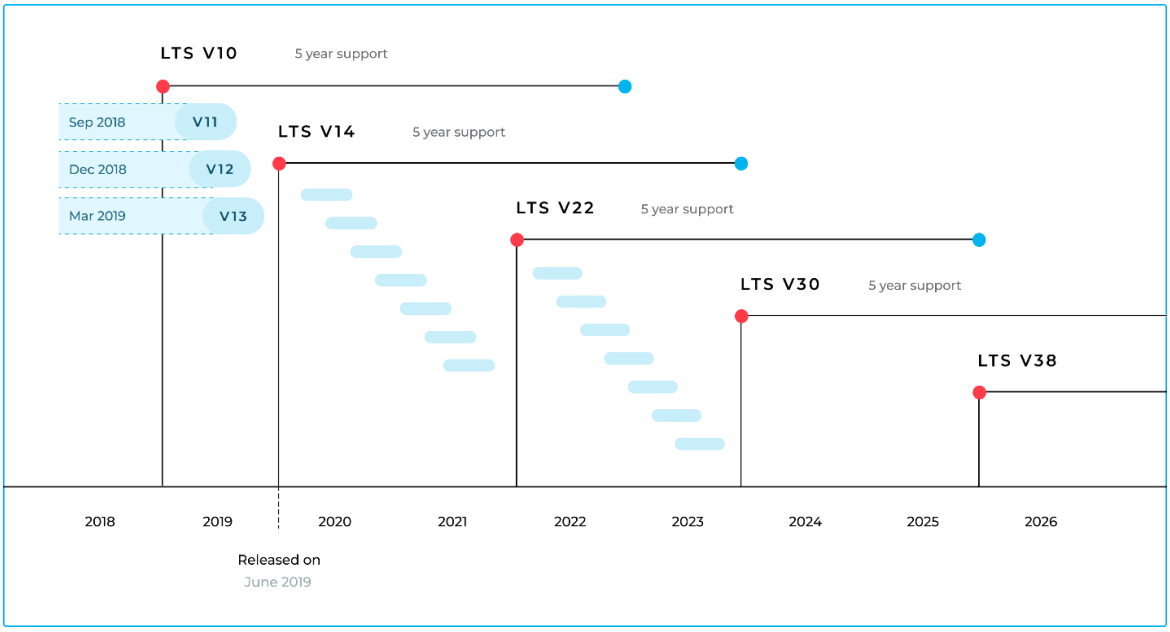
červenec 2018 - Vaadin verze 10, představující nejrozsáhlejší změny v historii tohoto frameworku. Dochází k přejmenování na Vaadin Flow, zahrnující zvýšení modularity jednotlivých komponent a přidání speciálního Element API pro manipulaci s modelem DOM. Byl také přidán nástroj Vaadin Designer, umožňující vytváření UI pomocí drag and drop. (Duarte, 2018). Dle (Vaadin Ltd., 2019c) se jedná o zatím poslední vydanou long-term support verzi tohoto frameworku.

září 2018 – Vaadin verze 11, přinášející pouze drobné změny, přičemž hlavním přínosem je integrace nástroje Gradle do Vaadinu. Součástí jsou také dvě nové komponenty. Jedna pro zobrazení souhlasu s užíváním cookies, druhá pro potvrzovací dialogové okno. (Jansson, 2018).

prosinec 2018 – Vaadin verze 12, obsahující vylepšení několika stávajících komponent, a zcela novou komponentu Vaadin CRUD, umožňující snadnější provádění CRUD operací nad příslušnou databází. Tato komponenta je však přístupná pouze v rámci placené Pro verze (Häyry, 2018).

březen 2019 – Vaadin verze 13, která obsahuje drobné aktualizace některých komponent, dále zavádí nová API pro práci s vybranými komponenty a také představuje několik vylepšení pro práci se základním motivem Lumo, popsaným dále v práci (Nguyen, 2019).

Pro úplnost je vhodné dodat, že dle (Häyry, 2018) došlo od verze 10 ke změně v modelu releasů nových verzí frameworku. Na základě (Vaadin Ltd., 2019c) je vždy k předem stanovenému datu vydána tzv. LTS verze, což je zkratka pro long-term support verzi, označující regulérní, neexperimentální verzi Vaadinu, pro kterou je poskytována podpora následujících 5 let od jejího vydání. Jedná se verzi frameworku, obsahující větší množství změn, která neobsahuje experimentální komponenty. Jak potvrzuje také (Kronquist, 2018), poslední vydanou LTS verzí je verze 10, verze 11, 12 a 13 jsou pouze krátkodobé verze, vydávané každé 3 měsíce v rámci releasového plánu, které slouží k experimentování s novými prvky. U všech výše uvedených verzí také platí, že jsou považovány za platné a aktuální v době psaní této práce. Na následujícím obrázku je pak zachycena časová osa, která představuje časový harmonogram vydávání jednotlivých verzí frameworku Vaadin v budoucích letech.



Obrázek 1: Časový plán vydávání nových verzí Vaadinu (Vaadin Ltd., 2019c)

Z obrázku vychází, že framework je v pravidelných intervalech aktualizován, a zároveň je u každé verze zajištěna několikaletá podpora. Framework Vaadin je tak velmi udržovaným a pravidelně aktualizovaným frameworkem, který disponuje aktivní komunitou vývojářů, starající se o životní cyklus jeho jednotlivých verzí.

2.2 Možnosti využití

Jak již bylo nastíněno v úvodu, framework Vaadin je zaměřen na tvorbu jednoduchých a přehledných UI k webovým aplikacím. Dle (Vaadin Ltd., 2019a) poskytuje tento framework

velké množství komponent, ze kterých je následně možné složit přehledné a moderně zpracované UI k vyvíjené aplikaci. Uvedené komponenty také umožňují responsivní design aplikace, což tento framework činí využitelným jak pro webové, tak pro mobilní aplikace, které mají potenciál být velice flexibilní a uživatelsky přívětivé.

Z hlediska využití Vaadinu v praxi uvádí (Vaadin Ltd., 2019d) tři hlavní příklady. Prvním je využití k vytvoření kompletního designu uživatelského rozhraní u informačního systému AXPM One společnosti AXPM, což je americká zdravotnická společnost, působící v oboru stomatologie. Druhým příkladem je využití pro vytvoření uživatelského rozhraní webového nástroje USP Secure Entry Server, který funguje jako webový aplikační firewall, provozovaný švýcarskou společností United Security Providers. Třetí příklad uvádí využití tohoto frameworku pro vytvoření frontendu k informačnímu systému KemConnect finské společnosti Kemira, který slouží pro sběr a analýzu zákaznických dat v oblasti chemického průmyslu.

Z uvedených příkladů je tak patrné, že framework je využíván především jako frontendová vrstva nad komplexními webovými nástroji a rozsáhlými informačními systémy velkých mezinárodních firem, ale i menších lokálně působících společností. Framework Vaadin se tak jeví jako univerzální řešení pro frontend pro rozsáhlejší podnikové aplikace.

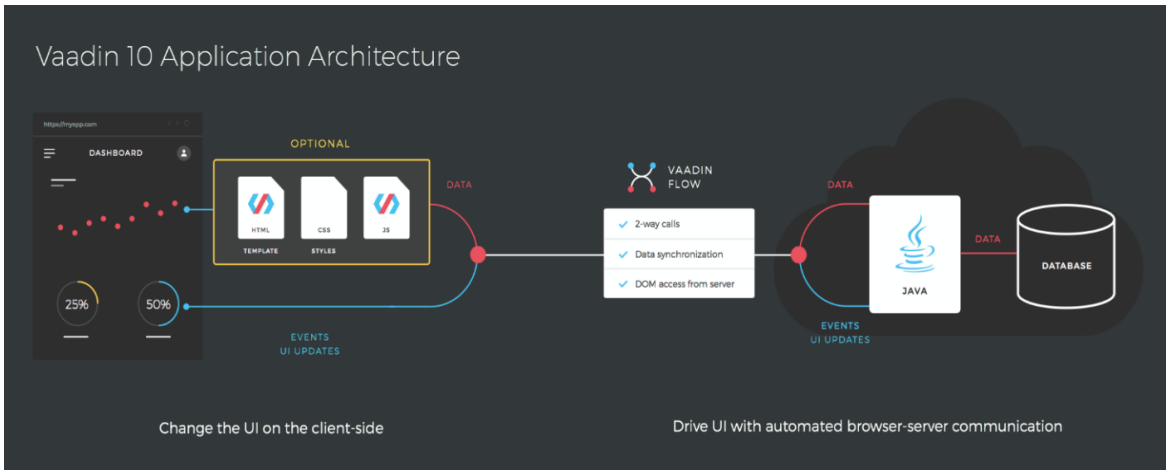
2.3 Vlastnosti frameworku

Z technického hlediska se dle (Vaadin Ltd., 2019a) jedná o framework, který na rozdíl od konkurenčních frameworků nedisponuje vysloveně vlastním ekosystémem, avšak umožňuje integraci s ostatními frameworky, jako je například Spring, prostřednictvím pluginů, dostupných ve Vaadin Directory (Koblentz, 2013). Výhodou Vaadinu je způsob zjednodušení vývoje aplikací tím, jak pracuje s dostupnými prostředky, kdy dochází k automatizaci veškeré komunikace, probíhající mezi klientem a serverem. Velkým přínosem je také podpora více jazyků, běžících na JVM, jako jsou Groovy či Kotlin, což poskytuje vývojářům značnou volnost.

Před rozebráním jednotlivých technologických aspektů je důležité zmínit, že s vydáním Vaadin verze 10, neboli Vaadin Flow, došlo k mnoha změnám způsobujícím značnou odlišnost Vaadin Flow od Vaadin verze 8. V některých případech došlo pouze k převzetí dosavadní technologie a k jejímu využití v rámci Vaadin Flow, avšak v určitých oblastech byly představeny modernější přístupy. Veškeré uvedené technické vlastnosti se tak vztahují k Vaadin Flow verze 10, která byla v době psaní této práce poslední vydanou LTS verzí a pro kterou je v rámci této práce používán základní zkrácený název Vaadin. Vzhledem k rozsáhlým možnostem frameworku byla pro každou oblast vyhrazena samostatná podkapitola.

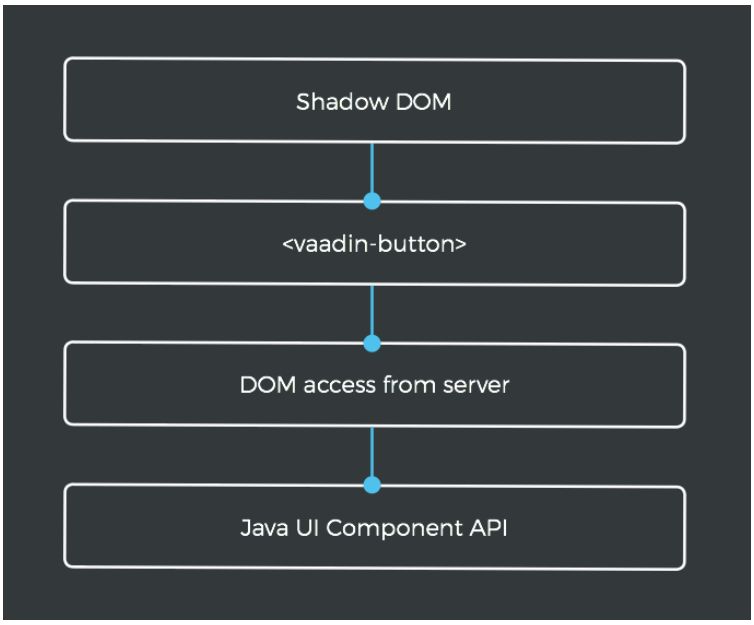
2.3.1 Architektura webové aplikace

Při vytváření webové aplikace dochází dle (Vaadin Ltd., 2019e) k převedení jednotlivých UI elementů, jako jsou tlačítka, checkboxy, vyhledávací pole a další v jednotlivé webové komponenty, reprezentované jednotlivými třídami. Tím je dosaženo oddělení a rozlišení jednotlivých komponent, což výrazně ulehčuje vývoj aplikací. S nástupem Vaadin Flow neboli Vaadin 10, došlo k implementaci speciálního server-side JAVA API, které umožňuje práci se zmíněnými komponenty. Tento zdroj dále tvrdí, že Vaadin Flow umožňuje obousměrnou komunikaci mezi serverem a webovým prohlížečem, což výrazně zjednodušuje propojení uživatelského rozhraní s backendem, využívajícím Javu. Kompletní architektura vaadinovské webové aplikace je zachycena na obrázku 2.



Obrázek 2: Architektura vaadinovské aplikace (Vaadin Ltd., 2019e)

Z obrázku je patrné, že veškeré změny na straně serveru se díky výše zmíněné obousměrné komunikaci a datové synchronizaci ihned projeví na straně klienta v podobě změny v některé z UI komponent. (Vaadin Ltd., 2019e) dále uvádí, že díky Vaadin Flow je možné přistupovat k elementům DOM přímo ze strany serveru, využívajícím Javu, jak je zachyceno na následujícím obrázku 3.



Obrázek 3: Přístup komponent k DOM (Vaadin Ltd., 2019e)

Framework dle (Vaadin Ltd., 2019e) umožňuje práci s DOM ve webovém prohlížeči pomocí vlastního stínového DOM, který je identický s tím skutečným a ve kterém jsou pomocí komponent prováděny změny. Ty jsou následně automaticky synchronizovány se skutečným DOM ve webovém prohlížeči, jak zjednodušeně zachycuje obrázek 3. Kořenovým elementem je v případě stínového DOM element instance komponenty UI, dostupné skrze metodu `ui.getElement()`. Tento element reprezentuje tag `<body>` ve skutečném DOM. Jednotlivé elementy jsou implementovány pomocí návrhového vzoru flyweight, díky čemuž dle (Baeldung, 2018) dochází při práci s těmito elementy k efektivnímu využívání kapacity dostupné operační paměti. Webová aplikace je tak dle (Vaadin Ltd., 2019e) strukturována jako strom jednotlivých elementů s instancí elementu UI v kořeni. Jednotlivé elementy lze přidávat jako potomky jiných elementů buď na poslední danou pozici v seznamu potomků, nebo na jakoukoli libovolnou pozici za kořenovým elementem. Pro vytváření jednoduchých webových aplikací tak díky abstrakci webových elementů není nutné disponovat detailními znalostmi ohledně principů komunikace mezi klientem a serverem a lze se soustředit přímo na práci s webovými komponenty, které pracují na vyšší úrovni abstrakce vzhledem k DOM.

Ve vztahu k jednotlivým úrovním abstrakce lze dle (Vaadin Ltd., 2019e) vytvářet několik druhů webových komponent, tvořících výsledné uživatelské rozhraní. Na nejvyšší úrovni abstrakce lze jednoduše vytvářet komponenty vytvářením nových instancí příslušných tříd. Díky tomu není nutné psát jakýkoliv HTML nebo javascriptový kód, jelikož veškerá potřebná funkcionalita je již implementována a obsažena v dané komponentě, reprezentované příslušnou Java třídou.

```
TextField textField = new TextField();
Label greeting = new Label("Hello stranger");
```

Ukázka kódu 1: Komponenta nejvyšší úrovně abstrakce (Vaadin Ltd., 2019e)

Na následující úrovni abstrakce je dle (Vaadin Ltd., 2019e) možné vytvářet nové komponenty modifikací těch stávajících s využitím dědičnosti. Díky možnosti přístupu k DOM ze strany serveru, naznačeném na obrázku 3 je tak možné provádět úpravy jednotlivých komponent a zároveň při provádění změn zůstat na straně serveru, čímž je zachována přirozená automatická komunikace mezi klientem a serverem, zajištěná rozebíraným frameworkem. Příklad takovéto komponenty je zachycen v ukázce kódu 2.

```
@Tag("my-label")
    public class MyLabel extends Component {
        public void setText(String text) {
            getElement().setText(text);
        }

        public String getText() {
            return getElement().getText();
        }
    }
```

Ukázka kódu 2: Příklad komponenty nižší úrovně abstrakce (Vaadin Ltd., 2019e)

Ukázka kódu 2 zachycuje tvorbu upravené komponenty s názvem MyLabel, která je rozšířením předka s názvem Component, a která obsahuje metody setter a getter pro nastavení a získání textu, obsaženém v labelu. Poslední možností je dle (Vaadin Ltd., 2019e) vytváření komponent na nejnižší úrovni abstrakce, kde je možné převzít kontrolu nad DOM a komunikací mezi klientem a serverem prostřednictvím HTML šablon pro jednotlivé komponenty a příslušné Java třídy, která tuto šablonu pro danou komponentu zapouzdřuje do podoby komponenty vyšší úrovně abstrakce.

```
<template>
    <vaadin-vertical-layout>
        <vaadin-text-field id="textField"></vaadin-text-field>
        <label id="greeting">Hello stranger</label>

        <input type="color" on-input="updateFavoriteColor">
        <label>Favorite color: </label>
    </vaadin-vertical-layout>
</template>
```

Ukázka kódu 3: Šablona komponenty nejnižší úrovně abstrakce (Vaadin Ltd., 2019e)

V ukázce kódu 3 je zachycena HTML šablona pro komponentu VerticalLayout, obsahující komponenty TextField a Label. K této šabloně se váže třída, zachycená v ukázce kódu 4.

```
private @Id("textField") TextField textField;
private @Id("greeting") Label greeting;

// Setting things up in the component's constructor
textField.addValueChangeListener(event ->
      greeting.setText("Hello " + event.getValue()));

// Instance method in the component published to the client
@EventHandler private void updateFavoriteColor(
      @EventData("event.target.value") String color) {
    getModel().setColorCode(color);
}
```

Ukázka kódu 4: Třída pro zapouzdření šablony z ukázky kódu 3 (Vaadin Ltd., 2019e)

V ukázce kódu 4 je třída, která umožňuje práci s danou HTML šablonou. Jejím prostřednictvím lze nastavit potřebné prvky v konstruktoru komponenty a zavolat metodu obsaženou v HTML šabloně dané komponenty.

Celkově tak Vaadin nabízí poměrně širokou škálu možností práce s jednotlivými komponenty, což jednoznačně zlepšuje potenciál při tvorbě webových aplikací.

2.3.2 Bindování dat

Jelikož je v aplikacích využívajících Vaadin poměrně běžné, že uživatel vyplňuje určitá strukturovaná data do formulářových polí, je nutné zajistit, aby byla uživatelem vyplňovaná data čitelná pro danou aplikaci. (Vaadin Ltd., 2019f) uvádí, že vyplňovaná data jsou reprezentována v podobě instance určitého business objektu, za který lze dle (Techopedia, 2019) považovat jakýkoliv objekt, představující množinu proměnných, které se vztahují k

jedné instanci dané třídy, přičemž s tímto objektem je možné dále pracovat. Dle tohoto zdroje existuje za tímto účelem speciální třída s názvem Binder, která se stará o čtení dat z business objektu, reprezentujícím uživatelem vyplněná data a jejich konverzí mezi formátem, který očekává komponenta formulářového pole a formátem, který očekává business objekt. Třídu Binder tak lze označit jako propojovací prvek mezi hodnotou formulářového pole a proměnnou, do které bude tato hodnota uložena. V následující ukázce kódu je zachycen jednoduchý příklad využití třídy Binder.

```
Binder<Person> binder = new Binder<>();

TextField titleField = new TextField();

// Start by defining the Field instance to use
binder.forField(titleField)
  // Finalize by doing the actual binding to the Person class
  .bind(
    // Callback that loads the title from a person instance
    Person::getTitle,
    // Callback that saves the title in a person instance
    Person::setTitle);
```

Ukázka kódu 5: Komponenta binder pro data (Vaadin Ltd., 2019f)

V kódu je znázorněno vytvoření nových instancí Binderu a formulářového pole a následné propojení tohoto formulářového pole s metodami getteru a setteru třídy Person, které se vyplňované údaje týkají. V praxi dojde k tomu, že díky binderu bude do proměnné s pravděpodobným názvem „title“ uložena hodnota, která byla vyplněna do formulářového pole.

Pro případ, že jsou výše zmíněné business objekty reprezentované pomocí Plain Old Java objektů neboli zkráceně POJO, nebo pomocí Java beanů, je pro ně dle (Vaadin Ltd., 2019g) možné vytvořit instanci binderu, jak je znázorněno v následující ukázce kódu.

```
Binder<Person> binder = new Binder<>(Person.class);

// Bind based on property name
binder.bind(nameField, "name");
```

Ukázka kódu 6: Binder pro Plain Old Java objekt (Vaadin Ltd., 2019g)

Znázorněná instance Binderu je speciálně vytvořená pro POJO třídu Person. Proměnná instance pak využívá reflexi k propojení formulářového pole s názvem nameField a

proměnné s názvem name, což jak tvrdí (Vaadin Ltd., 2019g), zmenšuje objem kódu, který je za účelem bindování potřeba vytvořit.

2.3.3 Kompatibilita s programovacími jazyky

Dle (Vaadin Ltd., 2019h) běží Vaadin na JVM, což v praxi znamená, že podporuje širokou škálu jazyků a frameworků, které jsou na bázi JVM založeny a jejichž výhodou je dle (Smith, 2018) podpora na mnoha typech zařízení, otevřenost a široká komunita. Mezi nejčastěji využívané programovací jazyky ve spojení s rozebíraným frameworkem patří především:

- Java,
- Kotlin,
- Scala,
- Gradle.

Je vhodné dodat, že do uvedené skupiny jazyků patří samozřejmě další, méně rozšířené jazyky a frameworky a výše uvedený výčet lze považovat pouze za orientační.

V souvislosti s využívanými JVM programovacími jazyky jsou pak nejčastěji využívány následující aplikační servery (Vaadin Ltd., 2019h):

- Apache Tomcat,
- JBoss,
- Glassfish.

2.3.4 Podporované prohlížeče

Vzhledem k tomu, že je dle (Vaadin Ltd., 2019a) Vaadin určen pro vývoj webových aplikací, je důležitá podpora široké škály webových prohlížečů. Výčet podporovaných prohlížečů a jejich verzí je zachycen v následující tabulce.

Tabulka 1: Podporované prohlížeče a jejich verze (zdroj: Vaadin Ltd., 2019h)

Webový prohlížeč	**Verze**
Internet Explorer	Verze 11
Microsoft Edge	Nejnovější
Firefox	Nejnovější
Chrome	Nejnovější
Safari	iOS9

2.4 Klíčové komponenty

Vaadin nabízí svým uživatelům širokou škálu komponent pro tvorbu webových aplikací. Poměrně velká část je dostupná zdarma, některé komplexnější a lépe vypadající komponenty jsou však placené. Jak již bylo zmíněno v úvodu, pro účely této práce je využito pouze volně dostupných komponent, které jsou pro splnění zadaného cíle dostačující. Vzhledem k velkému množství dostupných komponent jsou v této kapitole popsány pouze hlavní, volně dostupné komponenty, využité pro naplnění cíle praktické části této práce.

2.4.1 Grid

Hlavní komponenta, která je součástí téměř každé vaadinovské aplikace, je nazývána Grid. Dle (Vaadin Ltd., 2019ch) se jedná o server-side komponentu v podobě multifunkční tabulky, která slouží k zobrazování a editaci tabulkových dat, zorganizovaných do řádků a sloupů. Obsah jednotlivých sloupců je možné řadit či filtrovat, jednotlivé řádky je možné označovat a editovat a šířku sloupců i řádků lze libovolně měnit. Příklad gridu je zachycen na následujícím obrázku.

	Names		Years		
	First Name ▲ 2	Last Name ▲ 1	Born In	Died In	Lived For
	Filter	Filter	Filter	Filter	Filter
☐	Tycho	Brahe	1546 AD	1601 AD	55 years
☐	Giordano	Bruno	1548 AD	1600 AD	52 years
☐	Nicolaus	Copernicus	1473 AD	1543 AD	70 years
☐	Galileo	Galilei	1564 AD	1642 AD	78 years
☐	Christiaan	Huygens	1629 AD	1695 AD	66 years
			1555,17	1618,50	63,33

Obrázek 4: Grid (Vaadin Ltd., 2019ch)

Co se týče plnění gridu daty, (Vaadin Ltd., 2019ch) tvrdí, že je zde implementována podpora pro lazy-loading, takže jsou v tabulce zobrazena pouze data, která jsou viditelná na stránce, což je výhoda zvláště při zobrazování většího objemu dat na méně výkonných zařízeních. Jako zdroj dat pro grid defaultně slouží určitý seznam položek, zpravidla reprezentovaný instancí třídy List, jak dokládá následující ukázka kódu.

```
// Have some data
List<Person> people = Arrays.asList(
    new Person("Nicolaus Copernicus", 1543),
    new Person("Galileo Galilei", 1564),
    new Person("Johannes Kepler", 1571));

// Create a grid bound to the list
Grid<Person> grid = new Grid<>();
grid.setItems(people);
grid.addColumn(Person::getName).setCaption("Name");
grid.addColumn(Person::getBirthYear).setCaption("Year of
birth");

layout.addComponent(grid);
```

Ukázka kódu 7: Seznam osob jako zdroj dat pro grid (Vaadin Ltd., 2019ch)

Pro rozsáhlejší aplikace, které komunikují s určitým API za účelem získání, případně aktualizování zobrazovaných dat, je výše uvedený způsob značně nepraktický. Pro tyto aplikace umožňuje Vaadin dle (Vaadin Ltd., 2019i) napojení gridu na DataProvider, což je komponenta, která zajišťuje komunikaci s daným API v pozadí a poskytuje výsledná data gridu.

```
Grid<Person> grid = new Grid<>();

grid.setDataProvider(dataProvider);
```

Ukázka kódu 8: Data provider jako zdroj dat pro grid (Vaadin Ltd., 2019i)

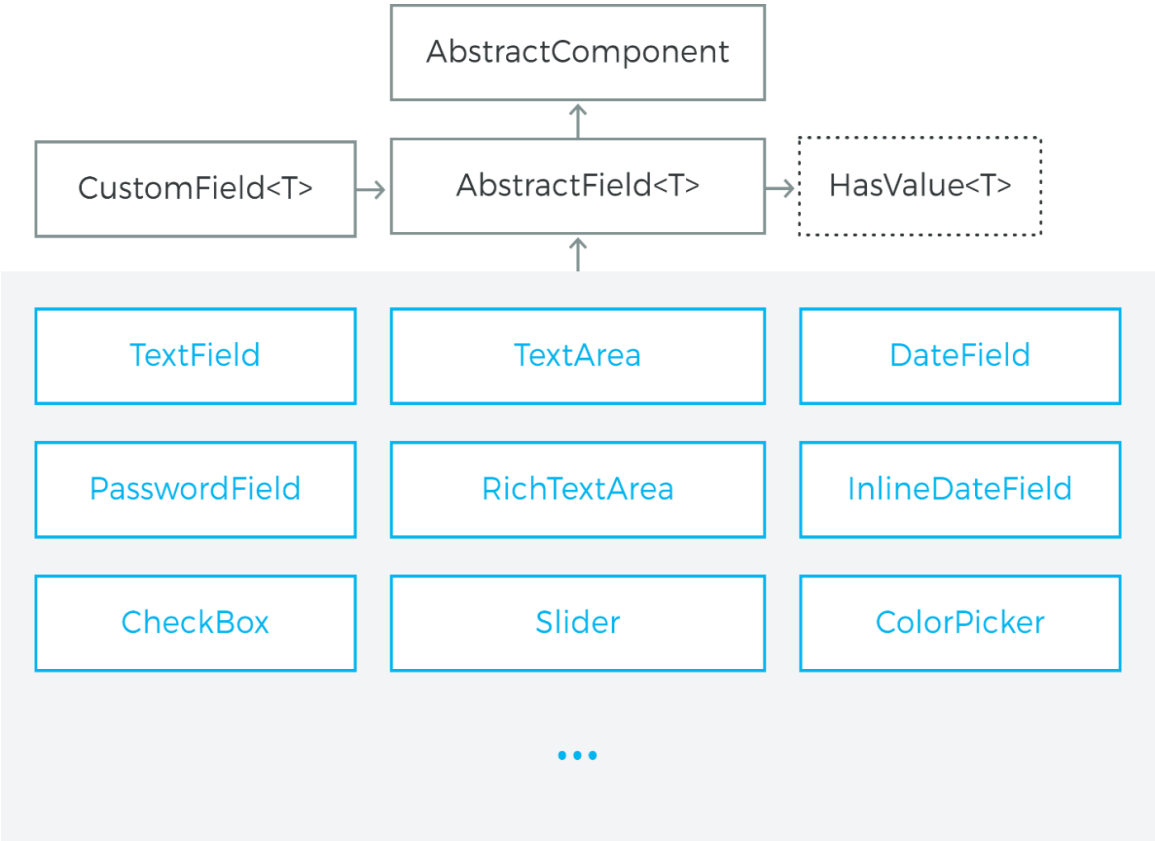
Uvedený příklad přiřazuje gridu proměnnou data provider, která představuje instanci třídy DataProvider, sloužící k předávání dat gridu. Tento způsob plnění gridu potřebnými daty je také využíván v praktické části práce při komunikaci s vybranými REST API.

Co se týče vizuální stránky gridu, je možné buď ponechat defaultní styly, nebo vytvořit libovolné styly pomocí CSS pro celý grid, či jeho jednotlivé elementy, jak uvádí (Vaadin Ltd., 2019ch).

2.4.2 Field

Field je další důležitou komponentou, bez které se neobejde téměř žádná webová aplikace. Jak uvádí (Vaadin Ltd., 2019j), u této komponenty platí, že nese určitou hodnotu, kterou může uživatel měnit skrze uživatelské rozhraní. Jedná se tak o typicky formulářové komponenty. Je důležité zdůraznit, že název Field je obecné označení pro celou rodinu

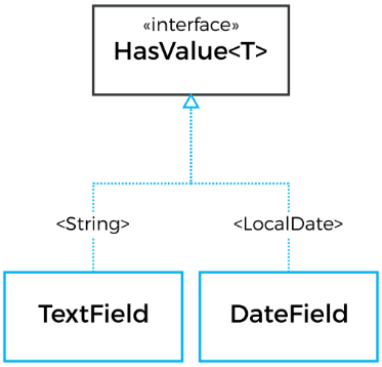
komponent, které dědí od společného předka s názvem AbstractField, jak dokládá následující obrázek.



Obrázek 5: Field komponenty (Vaadin Ltd., 2019j)

Kromě komponent, z jejichž názvu je zřejmé, že se jedná o určitá vyplňovací pole, využívají předka AbstractField i další komponenty, které slovo Field ve svém názvu explicitně neobsahují.

Všechny field komponenty s výjimkou těch, které umožňují uživateli zvolit hodnotu například zaškrtnutím určité možnosti, implementují dle (Vaadin Ltd., 2019j) rozhraní HasValue, které se stará o práci s uživatelem zadanou hodnotou, kterou daná field komponenta nese.



Obrázek 6: Implementace rozhraní HasValue (Vaadin Ltd., 2019j)

Mezi nejčastěji využívané field komponenty patří především komponenta TextField, která představuje jednoduché vyplňovací pole, do kterého je možné zadat libovolný text, či DateField, kam je možné vyplnit určité datum. Pro tento typ komponent je také typické, že jsou využívány ve spojení s dalšími komponentami Vaadinu, jako jsou kontextová menu, ikony, tlačítka a další, které rozšiřují jejich základní funkcionalitu.

Komponenta Binder dle (Vaadin Ltd., 2019j) umožňuje také implementaci jednoto či více validátorů, který slouží k ověření platnosti uživatelem zadaného vstupu do dané field komponenty před jeho uložením do objektu, jak zobrazuje následující ukázka kódu.

```
binder.forField(nameField)
    .withValidator(new StringLengthValidator(
        "Name must be between 2 and 20 characters long",
        2, 20))
    .bind(Person::getName, Person::setName);
```

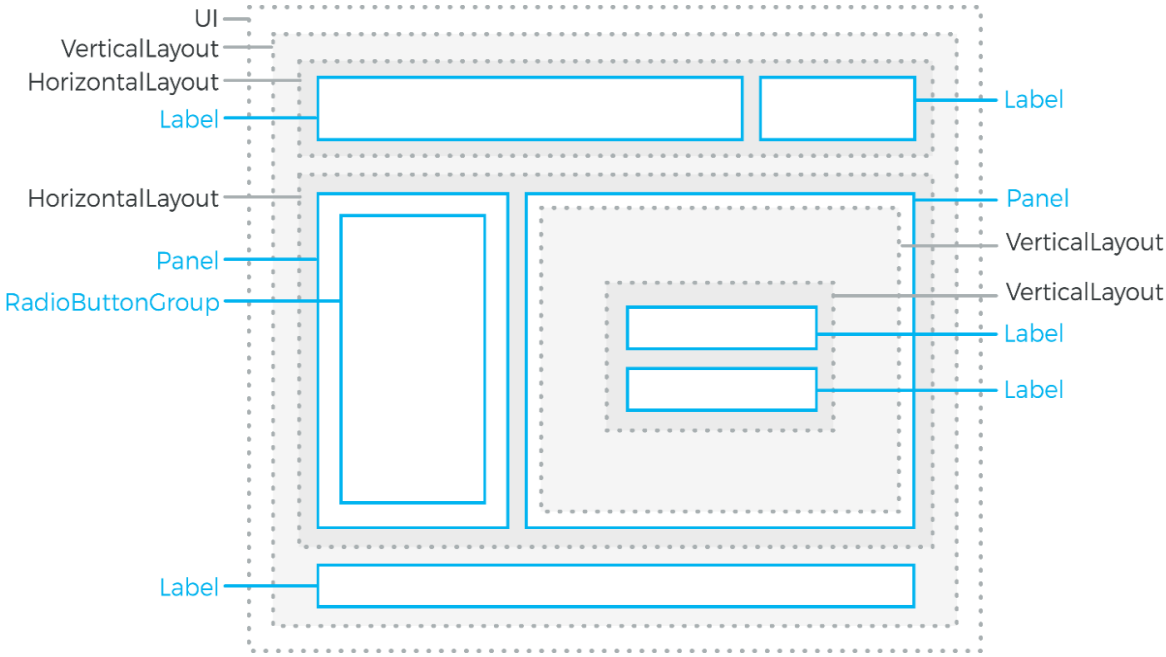
Ukázka kódu 9: Validator (Vaadin Ltd., 2019j)

Uvedený validátor umožňuje ověřit, zda je v poli nameField vyplněn text o délce 2-20 znaků. První část pak představuje chybovou hlášku, která je uživateli zobrazena při nesplnění stanovené podmínky, přičemž platí, že dokud není uvedená podmínka naplněna, nejsou vložené hodnoty uloženy. Kromě již vytvořených validátorů lze skrze implementaci rozhraní Validator vytvořit vlastní customizovaný validátor, ověřující potřebné nároky na vstupní data. Obecně pak platí, že validátor lze implementovat pro všechny field komponenty.

Samozřejmostí je také úprava stylů jednotlivých field komponent, kdy je možné pomocí CSS libovolně upravovat defaultní styl, či vytvořit nový.

2.4.3 Layout

Oproti komponentám z kapitol 2.4.1 a 2.4.2, které slouží k interakci s uživatelem, slouží dle (Vaadin Ltd., 2019k) layoutové komponenty k umisťování výše zmíněných komponent na stránce v rámci uživatelského rozhraní. Celkové schéma layoutových komponent zachycuje následující obrázek.



Obrázek 7: Schéma komponent typu Layout (Vaadin Ltd., 2019k)

Z obrázku je patrné, že možností uspořádání jednotlivých komponent na stránce je celá řada. Základní vrstvu tvoří vždy komponenta UI, na níž jsou dle obrázku kladeny další vrstvy v podobě vertikálních či horizontálních layoutů, obsahujících další komponenty. Jak již vyplývá z názvů a jak také tvrdí (Vaadin Ltd., 2019l), komponenty VerticalLayout a HorizontalLayout uspořádávají obsažené komponenty vertikálně respektive horizontálně na stránce. Oba zmíněné layouty představují základní komponenty každé vaadinovské aplikace, přičemž je obvyklé, že kořenovým layoutem v UI bývá vertikální layout a do něj jsou následně vložené horizontální layouty. Společně s komponentou FormLayout, která představuje rozvržení pro formulářová pole a tlačítka, dědí dle (Vaadin Ltd., 2019l) zmíněné dvě layoutové třídy od společného předka s názvem AbstractOrderedLayout a tvoří tak základní stavební prvky webových aplikací, přičemž jejich praktické využití je demonstrováno v praktické části práce.

Závěrem této kapitoly je vhodné zdůraznit, že se jedná pouze o obecný výčet základních klíčových komponent pro webové aplikace využívající rozebíraný framework. Celkové množství nabízených komponent je daleko vyšší a přesahuje rámec této práce.

2.5 Motivy

U webových aplikací využívajících Vaadin dochází k separaci vzhledu uživatelského rozhraní od související logiky prostřednictvím HTML motivů. Na základě (Vaadin Ltd., 2019m) je veškerá stylizace realizována pomocí CSS, které je v rámci aplikace odděleno od balíčků s aplikační logikou. V rámci Vaadinu existují dvě možnosti aplikace motivů. První je tvorba stylů na globální úrovni nad celou aplikací, které ovlivňují vzhled a umístění všech prvků aplikace. Druhou možností je aplikace individuálních stylů pro každou komponentu, což poskytuje prostor pro detailnější customizaci. Jak dále tvrdí (Vaadin Ltd., 2019n),

zmíněné motivy lze aplikovat několika způsoby. Pokud je na kořenové úrovni aplikace specifikována anotace *@Theme*, je daný motiv aplikován na celou aplikaci. Pokud je na kořenové úrovni uvedeno *@NoTheme*, použití motivu je deaktivováno. Zároveň platí, že defaultním motivem je vždy motiv Lumo, který dodává dané aplikaci jednoduchý a přehledný vzhled a který byl také použit pro ukázkovou aplikaci. Příklad aplikace jednotného motivu pro celou aplikaci zachycuje ukázka zdrojového kódu.

```
@Route(value = "")
@Theme(Lumo.class) // can be omitted for Lumo
public class Application extends Div {
}
```

Ukázka kódu 10: Aplikace motivu (Vaadin Ltd., 2019n)

Uvedenou anotací je k aplikaci přidělen defaultní motiv Lumo. Pro danou aplikaci z ukázky kódu 10 je specifikována také anotace *@Route*, která dle (Vaadin Ltd., 2019o) umožňuje v rámci aplikace určit libovolnou doplňující komponentu jako cílovou cestu pro daný URL fragment. V praxi to znamená, že je možné volat určité komponenty v závislosti na zobrazené stránce, jak také zachycuje následující ukázka kódu.

```
@Route("some/path")
public class SomePathComponent extends Div {
  public SomePathComponent() {
    setText("Hello @Route!");
  }
}
```

Ukázka kódu 11: Routing (Vaadin Ltd., 2019o)

Na základě specifikované route neboli cesty je v dané aplikaci možné zobrazit komponentu SomePathComponent na stránce, uvedené v route. V případě, že hodnota route je rovna prázdnému řetězci jako v ukázce kódu 10, je dle (Vaadin Ltd., 2019o) hodnota anotace *@Route* odvozena od názvu dané třídy. V případě ukázky kódu 10 tak bude mít *@Route* hodnotu „application".

2.6 Recenze odborníků a uživatelů

Z hlediska oblíbenosti figuruje obvykle Vaadin mezi deseti nejpoužívanějšími webovými frameworky pro Javu, jak dokládají například (Malhotra, 2018) a (Javapipe, 2018). Pro hlubší pochopení zájmu o tento framework je nutné analyzovat články odborníků na dané téma a pro zjištění názorů veřejnosti také největší programátorská fóra.

Odborné názory

Odborných článků rozebírajících framework Vaadin verze 10, na který je práce primárně zaměřena, bohužel není mnoho. Většina dostupných recenzí totiž zohledňuje pouze Vaadin verzi 8, což není pro potřeby této práce dostatečné. Ohledně Vaadin 10 byla nalezena pouze jediná recenze v podobě (Artamonov, 2018), zaměřená speciálně na Vaadin 10, která byla podrobně analyzována. Uvedený zdroj chválí framework za jednoduchost použití jednotlivých komponent, včetně jejich znovupoužitelnosti a customizace, dále vyzdvihuje automatickou synchronizaci uživatelského rozhraní po změnách provedených na straně serveru a celkovou integraci javascriptového frontendu s backendem běžícím na Javě. Na druhou stranu kritizuje Vaadin 10 za přílišnou nezávislost jednotlivých komponent, způsobující velmi obsáhlou strukturu externích knihoven u projektu. Také poukazuje na značné problémy při přechodu z Vaadin verze 8 na Vaadin Flow neboli Vaadin verze 10. Vzhledem k tomu, že se z technického hlediska jedná o zcela nový framework, není k dispozici žádný oficiální migrační postup, což může způsobit obrovské komplikace při přechodu na novější verzi frameworku u dané aplikace.

Názory veřejnosti

Na rozdíl od odborných recenzí je Vaadin často zmiňován na nejrůznějších programátorských fórech a portálech. Na nejpopulárnějším programátorském fóru Stack Overflow je možné nalézt dohromady přes 1000 otázek označených tagem „vaadin“. Jak je patrné z (Bourque, 2018), hodnotí uživatelé Vaadin Flow jako zcela odlišný od předchozí verze a obtížný na migraci. Zároveň vyzdvihují možnost vytváření uživatelského rozhraní pomocí Javy bez nutnosti práce s Javasciptem či CSS. Na fóru G2 Crowd uživatelé dle (G2 Crowd, 2018) kladně hodnotí rychlost vývoje aplikace, možnost využití šablon pro tvorbu vlastních frontendových komponent a celkovou jednoduchost práce s frameworkem. Jako negativum uvádějí především delší čas načítání a kompilace projektů a vysoké náklady na placené komponenty. Pro doplnění uživatelských recenzí byly analyzovány také příspěvky z fóra Quora, kde dle (Quora, 2018) uživatelé opět kladně hodnotí především rychlost vývoje aplikací a vhodnost využití pro rozsáhlejší aplikace na enterprise úrovni. Z hlediska negativ pak uživatelé uvádějí zejména celkovou rozsáhlost frameworku.

2.7 Porovnání s konkurenčními frameworky

Vaadin patří v současnosti mezi nejpopulárnější frameworky pro Javu, nicméně zde existuje celá řada dalších frameworků, které zásadním způsobem usnadňují vývoj aplikací v Javě. Pro hlubší zasazení Vaadinu do kontextu Java frameworků bylo na základě analýzy (Malhotra, 2018), (Monus, 2018) a (Javapipe, 2019) vybráno několik v současnosti populárních frameworků, které byly porovnány na základě dále uvedených kritérií. Vybrány byly následující frameworky:

- Spring,
- Play,
- Wicket,

- Grails,
- Vert.X.

Pro tyto frameworky byla za účelem kritického porovnání a posouzení vybrána na základě (Koblentz, 2013) jednotlivá srovnávací kritéria, na jejichž základě bylo dané porovnání provedeno:

- ekosystém frameworku,
- škálovatelnost,
- dokumentace a komunita,
- udržovatelnost kódu,
- jednoduchost použití.

2.7.1 Spring

Spring je dle (Tutorialspoint, 2019) v současnosti jedním z nejpoužívanějších a zároveň nejstarších open-source frameworků pro Javu. Vývojářům poskytuje kompletní sadu nástrojů pro vytváření webových aplikací a jejich konfiguraci, včetně potřebných bezpečnostních prvků, jak tvrdí také (Malhotra, 2018). Spring je dle (Pivotal Software Inc., 2019) rozdělen na zhruba 20 různých modulů, seskupených do větších celků, které vždy odpovídají příslušné části funkcionality vyvíjené aplikace. To vývojářům umožňuje udržovat přehledný a snadno přístupný zdrojový kód. Spring také využívá moderních přístupů, jako je Inversion of Control nebo Dependency Injection, umožňujících efektivnější práci s frameworkem. Z hlediska výše stanovených srovnávacích kritérií se framework Spring vyznačuje následujícími vlastnostmi:

Ekosystém

Framework Spring disponuje poměrně rozsáhlým, ale zároveň velice přehledným ekosystémem vzájemně propojených modulů, rozdělených do částí Web layer, Common layer, Service layer, Data layer a Foundation layer. Tyto části odpovídají jednotlivým vrstvám funkcionality vyvíjené aplikace. Jednotlivé moduly lze snadno získat skrze technologii Maven, což činí tento framework velice snadno přístupným (SpringTutorials, 2015).

Škálovatelnost

Podle (Koblentz, 2013) je s využitím Springu možné vytvářet snadno škálovatelné webové aplikace, které mohou využívat například dávkového zpracování příkazů či rozdělení jednotlivých částí aplikace na moduly a jejich vzájemné propojení pomocí služby obstarávající komunikaci mezi komponenty. Samozřejmostí je také podpora více vláknových aplikací a možnosti optimalizace práce s pamětí.

Dokumentace a komunita

Co se týče dokumentace, dle (Malhotra, 2018) existuje ke Springu velké množství nejrůznějších tutoriálů, příkladů a video návodů, včetně cvičné aplikace s názvem Petclinic. Pro vývojáře jsou tak veškeré potřebné informace a návody dobře dostupné. Spring také disponuje silnou komunitou, která je aktivní na různých programátorských fórech, kde se řeší individuální problémy jednotlivých vývojářů a dle (Koblentz, 2013) jsou také pořádány každoroční konference, nazývané SpringOne/2GX, které jsou zaměřené na aktuální trendy v oblasti vývoje.

Udržovatelnost kódu

Údržba zdrojového kódu vyžaduje v případě Springu pokročilejší znalost frameworku a jeho ekosystému. Základním prvkem pro podporu udržovatelnosti je široká škála Java anotací, které usnadňují práci se zdrojovým kódem v případě, že je potřeba jej aktualizovat (Koblentz, 2013).

Jednoduchost použití

Podle (Monus, 2018) je tento framework určen primárně pro aplikace na enterprise úrovni, což v praxi znamená velký objem potřebných znalostí pro využití jeho potenciálu. Jedná se tak spíše o složitější a technicky náročnější framework.

2.7.2 Play

Play představuje dle (Lightbend, 2018) jednoduchý open-source framework, který integruje všechny potřebné komponenty pro vývoj moderních webových aplikací, jako je integrovaný HTTP server, zpracovávání formulářů, či ochrana proti CSRF útokům. Využívá MVC architektury pro vývoj aplikací a podporuje využití návrhových vzorů. (Malhotra, 2018) dále tvrdí, že samotný framework je postaven na Akka toolkitu, což je rozšířený open-source nástroj, běžící na JVM. Framework Play umožňuje vývojářům udržovat jednoduché workflow a klade důraz na produktivitu práce.

Ekosystém

Framework Play má plnohodnotný ekosystém, který dle (Koblent, 2013) obsahuje Simple Build Tool pro buildování vyvíjených aplikací, dále využívá jazyka Scala jako šablonovací systém a nabízí podporu pro asynchronní I/O operace. Celý ekosystém je navržen tak, aby co nejvíce zvyšoval produktivitu práce při současném zachování potenciálu využití.

Škálovatelnost

Z hlediska škálovatelnosti je tento framework dle (Lightbend, 2018) na velice slušné úrovni, jelikož využívá lehkou, bezstavovou architekturu, vycházející ze zmíněného Akka toolkitu, který umožňuje bezproblémové vertikální i horizontální škálování vyvíjených webových aplikací.

Dokumentace a komunita

Na webu vydavatelské společnosti existuje velmi přehledná dokumentace, obsahující všechny potřebné podklady pro práci s tímto frameworkem. Kromě manuálů ke komponentám frameworku Play pokrývá tato dokumentace také funkcionality Akka toolkitu. Komunita je zde velice aktivní, některé nadnárodní společnosti, jako například LinkedIn využily tento framework při vývoji své hlavní stránky a v případě LinkedInu také dochází k pravidelnému publikování dokumentace pro komunitu frameworku Play (Koblentz, 2013).

Udržovatelnost kódu

Zdrojový kód jednotlivých komponent je dle (Koblentz, 2013) lehce čitelný a pochopitelný, a jelikož je využíváno principu konvence nad konfigurací, tak nevzniká žádný zbytečný zdrojový kód. Všechny projekty stavěné na tomto frameworku mají obdobnou strukturu, což má také pozitivní vliv na udržovatelnost.

Jednoduchost použití

Dle (Koblentz, 2013) lze základy tohoto frameworku pochopit velice rychle, nicméně využití jeho plného potenciálu naopak vyžaduje hlubší znalosti a je tak časově značně náročné. Problematická je zejména integrace s již existujícími nástroji a prostředím, jelikož například místo Mavenu je využíván již zmíněný Simple Build Tool. (Koblentz, 2013) dále tvrdí, že další problém spočívá také v migraci na novější verze tohoto frameworku, konkrétně verze 1.x na 2.x.

2.7.3 Wicket

Wicket přestavuje dle (Malhotra, 2018) jednoduchý open-source framework od společnosti Apache pro vytváření elegantních, responsivních webových aplikací. Podle (Monus, 2018) se primárně jedná o server-side framework, který využívá komponentovou architekturu, kde jsou jednotlivé komponenty baleny do znovupoužitelných balíčků, ke kterým je možné připojit vlastní javascriptový kód či CSS. (Malhotra, 2018) dále tvrdí, že tento framework je také lehce integrovatelný s HTML, což z něj činí nástroj pro puristy, kteří preferují provádění úprav v HTML.

Ekosystém

Co se týče ekosystému, dle (Monus, 2018) hrají zásadní roli jednotlivé, znovupoužitelné komponenty tvořící balíčky, nicméně jak uvádí (Koblentz, 2013), počítá tento framework také s využitím funkcionalit z ostatních frameworků. Sám o sobě tak nepokrývá veškerou potřebnou funkcionalitu pro tvorbu webové aplikace, jako například framework Spring, popsaný v předchozí podkapitole.

Škálovatelnost

Jelikož se jedná o server-side framework, kdy je výsledná stránka, která je zobrazována uživateli, renderována na straně serveru, klade Wicket určité nároky na výpočetní zdroje.

Vzhledem k výborné podpoře AJAX requestů je možné s ohledem na škálovatelnost využít tento framework k vytvoření základního skeletonu hlavní stránky a zbylou komunikaci provádět pomocí ajaxových dotazů. Jinak budou nároky na serverové zdroje zbytečně vysoké (Koblentz, 2013).

Dokumentace a komunita

Jak je patrné z (Apache Software Foundation, 2019a), k Wicketu je dostupná velmi obsáhlá a přehledná dokumentace, která pokrývá veškeré potřebné oblasti. K dispozici jsou také cvičné ukázkové aplikace, umožňující lepší pochopení problematiky. Framework také disponuje poměrně rozsáhlou a aktivní komunitou, která se stará o podporu vývojářů na nejrůznějších programátorských fórech a na sociálních sítích.

Udržovatelnost kódu

Díky separaci logiky a HTML je možné bezpečně provádět změny v designu a upravovat zdrojový kód bez nutnosti obav o dopady na logiku. Využívané HTML šablony jsou velice jednoduché na úpravu a celkově je tak údržba zdrojového kódu v rámci tohoto frameworku velmi snadnou záležitostí (Koblentz, 2013).

Jednoduchost použití

Díky jasnému oddělení Javy a HTML je pro vývojáře, specializující se na danou oblast funkcionality, velice snadné soustředit se pouze na svoji část aplikace (Koblentz, 2013). Dle tohoto zdroje se také při použití Wicketu téměř nevyskytuje jakýkoliv opakující se zdrojový kód a je využíváno anotací pro další zjednodušení práce s frameworkem.

2.7.4 Grails

Grails představuje dle (Monus, 2018) dynamický, webový open-source framework, využívající jazyka Groovy, který platí za objektově orientovaný jazyk, běžící na JVM. Tento framework využívá moderních programovacích principů, je dobře kompatibilní zejména s frameworky Spring a Hibernate a celkově umožňuje rychlý a efektivní vývoj webových aplikací, jak dále uvádí také (Javapipe, 2019). Pro Grails také existuje velké množství pluginů pro populární IDE, jako je IntelliJ IDEA, či Eclipse, což činí tento framework velice flexibilním.

Ekosystém

Tento framework je podle (Koblentz, 2013) označován jako tzv. full-stack framework a kromě MVC obsahuje velké množství nástrojů pro činnosti související s vývojem. Tento zdroj však dále tvrdí, že Grails tyto nástroje vysloveně nevynucuje a v souvislosti s tímto frameworkem je tak možné využít již zavedených nástrojů, na které jsou vývojáři zvyklí. Přínosem je také již zmíněné množství existujících pluginů.

Škálovatelnost

Co se týče škálovatelnosti, (Koblentz, 2013) uvádí, že vzhledem k častému propojení tohoto Frameworku se Springem a Hibernatem lze velice snadno škálovat a rozšiřovat vyvíjené webové aplikace.

Dokumentace a komunita

Jak je patrné z (Apache Software Foundation, 2019b), disponuje Grails rozsáhlou a velice přehlednou dokumentací, obsahující všechny potřebné informace pro práci s tímto frameworkem. Obsaženy jsou také nejrůznější tutoriály a vzorová aplikace. Uvedený zdroj dále tvrdí, že komunita je velmi aktivní a rozvinutá, pravidelně jsou pořádány konference a stejně jako u předchozích frameworků dochází k poskytování podpory vývojářům na jednotlivých fórech.

Udržovatelnost kódu

Projekty vytvořené v rámci Grailsu mají obdobnou strukturu. Jazyk Groovy, na kterém je tento framework postaven, je obecně úspornější, co se objemu kódu týče, a je implementována podpora pro tzv. hot code reloading, což má za následek výraznou úsporu času při údržbě zdrojového kódu (Koblentz, 2013).

Jednoduchost použití

Grails je dle (Javapipe, 2019) určen pro rychlý vývoj aplikací a provedení potřebných konfigurací je díky implementované konzoli a definovaným příkazům velice snadnou záležitostí. Framework je tak z hlediska použitelnosti uživatelsky velice přívětivý.

2.7.5 Vert.X

Vert.X je dle (Monus, 2018) jednoduchý webový framework od společnosti Eclipse Foundation, který je kromě Javy kompatibilní také s dalšími jazyky, jako jsou Groovy, Ruby či Scala, jak dokazuje také (Javapipe, 2019). V zásadě se jedná o jednu velkou knihovnu modulárních komponent, což v praxi umožňuje aplikaci libovolných dalších nástrojů a komponent z ostatních knihoven dle potřeby. Framework je tak velmi flexibilní a dle (Javapipe, 2019) také velmi jednoduchý na nastavení.

Ekosystém

Tento framework poskytuje dle (The Eclipse Foundation, 2019a) kromě základních core komponent také pokročilé komponenty pro práci s cloudem, testování či komunikaci s webovými službami. Všechny komponenty jsou navíc plně modulární a kompatibilní s komponenty z ostatních knihoven pro Javu.

Škálovatelnost

Z hlediska škálovatelnosti záleží dle (Javapipe, 2019) na rozsahu a složitosti škálovaného systému. Framework je totiž vhodný pro škálování zejména malých systémů, v případě rozsáhlejších systémů se jedná o velice pracnou záležitost.

Dokumentace a komunita

K frameworku je dostupná přehledná dokumentace, která kromě technických informací obsahuje také odkazy na ukázkové aplikace, jak je patrné z (The Eclipse Foundation, 2019a). Komunita, stejně jako v případě předchozích frameworků, působí na programátorských fórech, vede vlastní wiki stránku a jeví se jako velmi aktivní (The Eclipse Foundation, 2019b).

Udržovatelnost kódu

Pro udržování zdrojového kódu je vhodné využít knihovnu *Awesome Vert.x*, která dle (The Eclipse Foundation, 2019c) doplňuje základní funkcionalitu například o možnosti provádění konfigurací skrze anotace. Nástroje pro práci s kódem, které lze využívat v rámci IDE, jsou taktéž dostupné ve formě dodatečných pluginů.

Jednoduchost použití

Podle (The Eclipse Foundation, 2019d) se jedná o framework, který je z hlediska použitelnosti velmi jednoduchý a flexibilní a zároveň umožňuje vytvářet aplikace s ohledem na výkon. Vyznačuje se také značnou volností v konvencích pro vytváření aplikace, což vývojářům poskytuje značnou volnost.

2.7.6 Shrnutí porovnávaných frameworků

Na základě provedeného porovnání je patrné, že nejrozsáhlejším a nejrozšířenějším frameworkem je Spring. Jeho jedinou slabinou v rámci porovnávaných kritérií je jednoduchost jeho použití, jelikož vzhledem ke svému rozsahu klade na vývojáře značné nároky. Z hlediska jednoduchosti je opakem Springu framework Wicket, který je díky zmíněnému oddělení Javy a HTML velice přívětivý a jednoduchý na pochopení. Z hlediska škálovatelnosti vyhrává opět Spring, který díky své modularitě umožňuje škálovat jak jednoduché, tak rozsáhlejší aplikace. Co se týče dokumentace a komunity, všechny uvedené frameworky disponují velmi slušnou a dostupnou dokumentací a jejich komunita je velmi aktivní. Poslední kritérium udržovatelnosti zdrojového kódu nemá v tomto případě jednoznačného vítěze, jelikož všechny uvedené frameworky tuto vlastnost implementují na dostatečné úrovni. Vzhledem k tomu, že Spring se značně liší od zbývajících čtyř frameworků, je možné vybrat jako hlavní konkurenci pro Vaadin frameworky dva. Prvním je bezesporu Spring, druhým je framework Grails, který si při sečtení porovnávaných kritérií drží lehký náskok nad zbývajícími frameworky. Výsledek však vychází pouze na základě vybraných kritérií a nelze vyloučit, že v případě jinak volených kritérií by porovnání uvedených frameworků dopadlo odlišně.

3 Komunikace s REST API

Za účelem prokázání schopnosti frameworku zprostředkovat komunikaci s REST API a daným způsobem zpracovat a zobrazit uživateli výsledky této komunikace byla na základě pečlivé analýzy vybrána dvě veřejná REST API, poskytující veřejnosti přístupná data. Pro pochopení průběhu komunikace je však nejprve nutné uvést základní vlastnosti REST API a možnosti, které tyto API svým uživatelům poskytují.

3.1 Základní vlastnosti

Jak uvádějí (Richardson a Amundsen, 2013, s. 29-32), REST představuje architektonický styl tvořící sadu vymezených pravidel pro vytváření webových služeb, jejichž význačným prvkem je bezstavovost neboli absence uchovávání informací na stranách odesílatele či příjemce v rámci komunikace. (Liew, 2018) dále uvádí, že v rámci přístupu REST jsou za účelem získání určité části dat ze zdroje využívány jednotlivé dotazy na API, složené z několika definovaných parametrů, přičemž daný zdroj vždy disponuje specifickou URL adresou. Vrácená data tak představují odpovědi na zaslané dotazy, které dle (Richardson a Amundsen, 2013, s. 33) z technického hlediska využívají metod protokolu HTTP a mohou být následující:

- GET – získání dat ze zdroje,
- POST – vytvoření nových záznamů na zdroji,
- PUT – kompletní aktualizace záznamů na zdroji,
- DELETE – vymazání zdrojových dat,
- HEAD – získání metadat, odesílaných současně se zdrojovými daty,
- OPTIONS – získaní přehledu HTTP metod, které zdroj podporuje.

Kromě výše uvedených typů dotazů existují ještě dotazy využívající metody, které dle (Richardson a Amundsen, 2013, s. 34) nejsou oficiální součástí HTTP standardu. Jedná se o následující metody:

- PATCH – modifikace části záznamů na zdroji,
- LINK – připojení dalšího zdroje k již existujícímu zdroji,
- UNLINK – zrušení propojení daného zdroje s jiným zdrojem.

Některé z výše uvedených metod tak dle (Webber, Parastatidis a Robinson, 2010, s. 57) a (Ola, 2018) představují CRUD operace nad danou množinou dat. Konkrétně operaci Create představuje metoda POST, operaci Read odpovídá metoda GET, operaci Update ztělesňují metody PUT a PATCH, přičemž rozdíl mezi těmito metodami spočívá v rozsahu změn prováděných nad zdrojovými daty. Zatímco metoda PUT umožňuje pouze kompletní aktualizaci daných dat, metoda PATCH umožňuje aktualizovat pouze část těchto dat. Poslední CRUD operaci Delete pak zajišťuje stejnojmenná metoda DELETE.

Restová architektura umožňuje dle (Monus, 2019) v rámci odpovědí zasílat data ve formátech plain-text, HTML, XML, JSON a YAML. Z uvedených formátů je dle tohoto zdroje v současnosti nejoblíbenější formát JSON, jelikož je jednoduchý, lidsky čitelný a umožňuje velice rychlou výměnu dat. Vzhledem k možnostem vybraných REST API, uvedených v následujících kapitolách, jsou v této práci získávána a dále zpracovávána data ve formátu JSON.

3.2 Kritéria výběru

Na základě předchozích zkušeností s vývojem aplikací, doplněných o poznatky z článku od (Freitas, 2018), bylo vybráno několik kritérií, na jejichž základě byl proveden výběr vhodných veřejných REST API pro účely této práce. Uvedená kritéria jsou následující:

- licenční podmínky,
- limit počtu odesílaných požadavků,
- relevantnost zasílaných dat.

Pro naplnění prvního požadavku bylo nutné najít taková veřejná REST API, která zdarma poskytují alespoň určitý demo klíč nebo demo token. API s placeným přístupem nejsou vzhledem ke stanoveným omezením této práce uvažována. S přístupovým klíčem se pojí druhé důležité kritérium, kterým je omezení množství odesílaných požadavků na dané API za určitou jednotku času. Pro účely této práce tak bylo nutné najít taková REST API, která v rámci licenčních podmínek umožňují přijmout alespoň 50 požadavků za den. Poslední uvedený požadavek na relevantnost zasílaných dat představuje nárok na úroveň získávaných dat, kdy je vzhledem k akademické povaze této práce nutné, aby stahovaná data byla na odborné úrovni a pocházela od důvěryhodného a relevantního zdroje.

3.3 NASA API

NASA API představuje veřejné REST API, které dle (NASA, 2019a) umožňuje získat prostřednictvím zasílaných požadavků data, která souvisejí s odbornou činností americké společnosti NASA. Tato data se týkají především prováděného výzkumu planety Země, Marsu, Slunce a nejznámějších objevených asteroidů, nacházejících se poblíž Země. Prostřednictvím tohoto API je možné získat jak surová data ve formátu JSON určená k dalšímu zpracování, tak fotografie ve formátu JPG pořízené satelity NASA. Pro tuto práci byla z tohoto API získávána data, týkající se oblasti označované jako Near Earth Objects (NEO) neboli Zemi blízké objekty, která se vždy vztahují k zadanému rozmezí platných kalendářních dat, přičemž z důvodu zjednodušení bylo implementováno vyhledávání Zemi blízkých objektů pouze k jednomu konkrétnímu kalendářnímu datu (NASA, 2019b).

Licenční podmínky

Na základě (NASA, 2019b) existuje několik režimů práce s tímto API, které vycházejí ze stanovených licenčních podmínek. První možností je využití DEMO_KEY, což je

přístupový token, který uživatele opravňuje k provedení maximálního počtu 30 dotazů za hodinu a celkově 50 dotazů z jedné unikátní IP adresy za den. To uživateli umožňuje alespoň základní práci s tímto API na elementární úrovni. Pro získání plného přístupu slouží stručný formulář, po jehož vyplnění je uživateli vygenerován NASA developer key, opravňující k provedení 1000 dotazů za hodinu. Pro práci s NASA API byl za akademickým účelem získán uvedený developer key, jehož možnosti jsou pro účely této práce zcela dostačující.

Parametry pro získaní dat

Pro získání zvolených dat od NASA API jsou v práci využity následující specifikované parametry a jejich hodnoty:

Tabulka 2: Parametry dotazu na NASA API (zdroj: NASA, 2019b)

Parametr	Hodnota
api_key	G5RtgrVQak8TBnysXstyRbd1MmJiOal23aUurQh9
start_date	Vybrána uživatelem z kalendáře
end_date	Vždy stejná jako hodnota start_date

Přístupový token api_key byl získán registrací na webové stránce k NASA API (NASA, 2019c). Parametr start_date, představující datum, od kterého jsou Zemi blízké objekty vyhledávány, je zadáván uživatelem. Hodnota parametru end_date, představujícího konec datového intervalu pro vyhledávání daných vesmírných objektů, je pak z důvodu zjednodušení vyhledávání na jedno konkrétní datum vždy rovna hodnotě parametru start_date.

3.4 Mapbox API

Mapbox dle (Mapbox, 2019a) představuje platformu pro lokaci a navigaci pro mobilní a webové aplikace. Svým uživatelům poskytuje geolokační data, API a SDK jako podklady a nástroje při vývoji aplikací na nejrozšířenějších platformách. Kromě výše uvedených prvků umožňuje Mapbox svým uživatelům vytvářet individuální mapy, dlouhodobě se zaměřuje na optimalizaci navigačních tras na základě dat dostupných v reálném čase a v poslední řadě podporuje také využití technologie rozšířené reality pro volnočasové aktivity. Co se týče poskytovaných API, Mapbox dle (Mapbox, 2019b) nabízí uživatelům celkem tři druhy API zaměřené vždy na určitou oblast týkající se lokace či navigace. Těmito druhy jsou: Maps API, umožňující práci s mapami a geografickými daty, Navigation API pro poskytování navigačních dat a Geocoding API, které umožňuje na základě určeného místa získat jeho geografické koordináty. Z uvedených API bylo pro účely této práce vybráno Geocoding API, které dle (Mapbox, 2019c) na základě zadaného názvu konkrétního místa, města, či státu poskytuje základní popisné geografické údaje o daném místě a včetně jeho přesné

zeměpisné šířky a délky. Dle uvedeného zdroje umožňuje toto API také retrospektivní vyhledávání míst na základě zadaných koordinát, které však z důvodu složitosti validace uživatelem zadávaných koordinát nebylo v práci implementováno.

Licenční podmínky

Na všechna API od Mapboxu se dle (Mapbox, 2019d) vztahují licenční podmínky typu pay-as-you-go, kdy je uživatel oprávněn zdarma provést prvních 50 000 dotazů na výše uvedená API. Každých následujících 1000 dotazů je pak zpoplatněno paušální částkou. Přístup k API je zajištěn pomocí tokenu, který je možné vygenerovat po registraci na webu společnosti Mapbox. K tomuto tokenu se pak vztahují uvedená omezení na počet dotazů. Vybrané API je tak z hlediska licence zcela vyhovující pro účely této práce.

Parametry pro získaní dat

Pro získání vybraných dat od Mapbox Geoding API byly využity parametry a jejich hodnoty zachycené v následující tabulce.

Tabulka 3: Parametry dotazu na Mapbox Geocoding API (zdroj: Mapbox, 2019c)

Parametr	Hodnota
access_token	pk.eyJ1Ijoic2hhZG93MTUiLCJhIjoiY2pzYTk4YnVuMTd6djQ0b2tldWxjNWY4eSJ9.ggoq5tLa1gx-U14rndkDFA
search_text	Vybrána uživatelem ze seznamu

Přístupový token access_token byl získán vytvořením uživatelského účtu na webové stránce společnosti Mapbox (Mapbox, 2019b). Parametr search_text pak představuje hodnotu formuláře s hlavními městy evropských zemí, vybranou uživatelem ze seznamu.

4 Analýza a návrh aplikace

Za účelem praktického ověření vlastností frameworku Vaadin, popsaných v kapitole 2, byla navržena a vyvinuta jednoduchá ukázková aplikace, která umožňuje komunikaci s vybranými veřejnými REST API, popsanými v předchozí kapitole. Pro její vývoj bylo však nejprve nutné provést detailní analýzu, která je obsažena v této kapitole.

4.1 Technická charakteristika aplikace

Účelem navržené aplikace je především prokázat, že prostřednictvím frameworku Vaadin je možné jednoduše komunikovat s veřejnými REST API, vybranými na základě předem stanovených kritérií, a získaná data od těchto REST API dále zpracovávat. Navrhovaná aplikace se soustředí zejména na technickou stránku zpracování přijímaných dat od vybraných REST API a jejich zpracování a zpřístupnění koncovému uživateli. Z toho důvodu je v aplikaci preferována celková funkčnost a použitelnost nad vzhledem a grafickým designem. Kromě funkčnosti je kladen důraz také na nefunkční požadavky, které se výrazným způsobem podílejí na celkové použitelnosti aplikace a které jsou obsaženy v následující kapitole.

Pro spuštění a následný provoz aplikace byl využit aplikační server Apache Tomcat, na kterém byly provedeny potřebné konfigurace, které jsou v práci dále popsány. Pro vývoj samotné aplikace byla využita částečně upravená metodika OpenUp, jejíž upravená podoba je součástí této části práce.

4.2 Uživatelská charakteristika aplikace

Z uživatelského hlediska se jedná o jednoduchou aplikaci, která umožňuje v zobrazeném menu vybrat příslušné REST API a následně z něj prostřednictvím zadaných dat do formuláře získávat potřebná data. Tato data jsou uživateli zobrazena v multifunkční tabulce označované jako grid, která umožňuje řazení hodnot v jednotlivých sloupcích, a kromě toho také barevné odlišení jednotlivých řádků pro větší přehlednost. Zde je důležité podotknout, že vzhledem ke značné složitosti a objemu stahovaných dat je v gridu vždy zobrazena pouze jejich hlavní a nejdůležitější část, jelikož by navrhovaná aplikace jinak mohla klást značné nároky na výpočetní zdroje. Pro získání kompletní odpovědi od daného REST API, včetně zaslaných parametrů, umožňuje aplikace stisknutím tlačítka stáhnout kompletní odpověď ve formátu JSON, která může posloužit k další strojové analýze. Kromě formátu JSON umožňuje aplikace také export zobrazeného gridu do formátu XLSX pro Microsoft Excel, který umožňuje manažerskou analýzu získaných dat.

4.3 Použitá metodika

Pro vývoj aplikace byla využita metodika OpenUP, která však byla pro potřeby této práce výrazně zeštíhlena a přizpůsobena. Metodika OpenUp je dle (Balduino, 2007, s. 1) agilní metodikou, dostupnou pod Eclipse Public License, která představuje zeštíhlení metodiky

Unified Process a aplikuje obdobné principy. Těmito principy jsou především iterativní a inkrementální přístup v rámci životního cyklu, využívání scénářů a případů užití, řízení rizik a důraz na architekturu aplikace. Jak dále tvrdí také (Management byznys informatiky, 2019), tato metodika je definována jako „minimálně dostatečná“, ale zároveň kompletní a poskytuje prostor pro přizpůsobení a rozšíření v závislosti na projektu, na který je aplikována. (Balduino, 2007, s. 1-2) dále uvádí, že metodika OpenUp obecně stojí na čtyřech základních principech, z nichž každý odpovídá jednomu z prohlášení v Agilním manifestu. Tyto principy jsou:

- spolupráce za účelem sladění zájmů a sdílení znalostí,
- maximalizace hodnoty pro zájmové strany srovnáním priorit,
- zaměření na architekturu již od počátku projektu za účelem minimalizace rizik a organizace vývoje,
- postupné zlepšování na základě obdržené zpětné vazby.

Uvedené čtyři principy odpovídají v pořadí, v jakém jsou výše uvedeny, následujícím prohlášením Agilního manifestu:

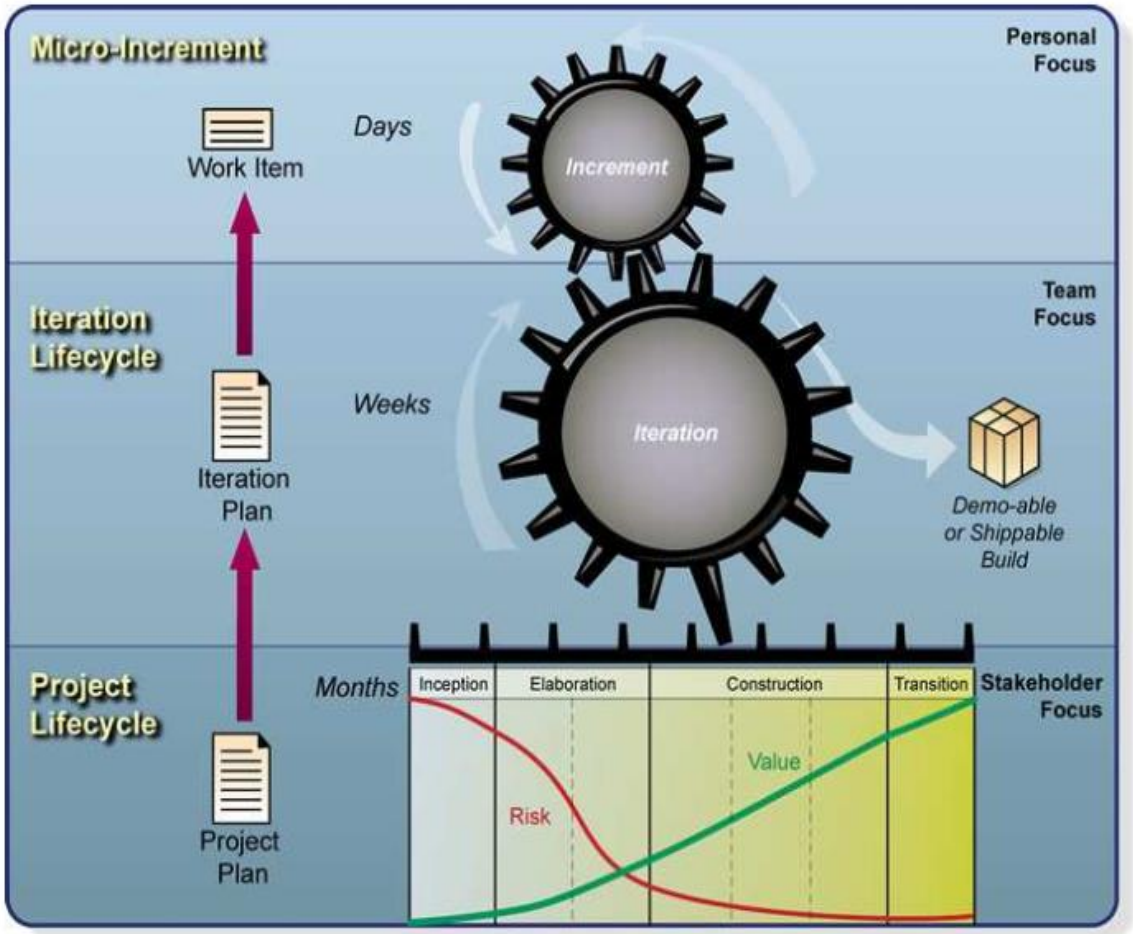
- jednotlivci a interakce před procesy a nástroji,
- spolupráce se zákazníky před smluvním vyjednáváním,
- funkční software před rozsáhlou dokumentací,
- schopnost reakce na změny před dodržováním plánu.

Jak je dále uvedeno v (Balduino, 2007, s. 3-5), metodika OpenUP je celkově členěna na dvě vzájemně korelující dimenze, jmenovitě metodickou náplň, kde jsou definovány jednotlivé role, disciplíny a artefakty, a procesní náplň, ve které jsou jednotlivé metodické elementy aplikovány. Uvedené role jsou následující:

- zájmová strana,
- analytik,
- architekt,
- vývojář,
- tester,
- projektový manažer,
- libovolná.

Role odpovídají běžným rolím, vyskytujícím se na softwarových projektech, libovolná role představuje v praxi jakéhokoliv člena týmu, schopného provádět obecné úkoly. V metodice jsou dle (Balduino, 2007, s. 4) uvedeny také jednotlivé základní disciplíny. Konkrétně se jedná o definici požadavků, tvorbu architektury, vývoj, testování, projektový management a konfigurační management. Uvedené disciplíny se projevují napříč životním cyklem a jsou obstarávány výše definovanými rolemi. Z hlediska životního cyklu je daný projekt rozdělen do čtyř částí na zahájení, elaboraci, konstrukci a přechod. Vývoj probíhá po mikro-

inkrementech, které zpravidla představují výstup práce za několik pracovních hodin či dní, které postupně tvoří výsledný celek. Životní cyklus projektu včetně mikro inkrementů je zachycen na následujícím obrázku.



Obrázek 8: Životní cyklus projektu v metodice OpenUP (zdroj: Balduino, 2007, s. 3)

Obrázek dokazuje uvedený iterativní a inkrementální přístup, který dle (Balduino, 2007, s. 1) představuje jeden ze základů agilního přístupu. Kromě výše uvedených principů, rolí a dimenzí jsou v rámci OpenUP stanoveny také potřebné základní definice případu užití, požadavků a dalších artefaktů, využitých při vývoji ukázkové aplikace.

Vzhledem k rozsahu ukázkové aplikace a jejího účelu byla výše popsaná metodika upravena a přizpůsobena tak, aby odpovídala účelům této práce. Na základě možností modifikace, uvedených v (The Eclipse Foundation, 2009) byly provedeny tyto úpravy:

- Byly definovány pouze dvě základní role, konkrétně vývojář a zájmová strana. Vývojář zastává v rámci této práce zároveň roli analytika, architekta, testera a projektového manažera. Zájmová strana pak definuje požadavky a stanovuje časové termíny dokončení jednotlivých částí práce.
- Délka jednoho mikro-inkrementu byla stanovena jako jedna hodina, jedna iterace představuje jeden pracovní den.

- Životní cyklus projektu byl celkově zkrácen na dobu jednoho měsíce.

Výše uvedené úpravy metodiky OpenUP se vztahují pouze na tuto práci s ohledem na její cíle. V případě rozšíření práce či zcela jiné práce, jejíž součástí je vývoj aplikace, by bylo nutné provést revizi uvedených úprav s ohledem na stanovené cíle dané práce.

4.4 Požadavky na aplikaci

Pro zajištění potřebné úrovně funkčnosti a celkové kvality navržené aplikace bylo zapotřebí definovat požadavky, které jsou z hlediska jejich podstaty rozděleny na funkční a nefunkční. Jelikož je cílem navrhované aplikace především zajištění komunikace s externím veřejným REST API, kde je spolehlivost komunikace a získávaných dat upřednostňována před celkovým designem aplikace, převažují v oblasti požadavků nefunkční požadavky.

4.4.1 Funkční požadavky

V následující tabulce jsou zachyceny všechny definované funkční požadavky na navrženou aplikaci. Jednotlivé požadavky jsou odstupňovány podle priority, vyjádřené celočíselnou stupnicí 1-5, kde stupeň 1 představuje nejnižší prioritu a stupeň 5 prioritu nejvyšší. Každému funkčnímu požadavku také odpovídá příslušný případ užití, reprezentovaný odpovídajícím kódovým označením.

Tabulka 4: Funkční požadavky (zdroj: autor)

ID	Název požadavku	Obsah požadavku	Priorita požadavku	Související případ užití
FP01	Možnost výběru REST API v menu	Uživatel si může v hlavním menu vybrat příslušné REST API, ze kterého si přeje stahovat data.	5	UC01
FP02	Volba parametrů dotazu na REST API	Uživatel může zvolit parametry odesílaného dotazu na vybrané REST API prostřednictvím formuláře.	5	UC02
FP03	Možnost odeslat dotaz na REST API	Uživatel může pomocí tlačítka vedle formuláře odeslat dotaz na REST API s parametry, které zadal do formuláře.	5	UC03
FP04	Zobrazení dat v gridu	Uživatel si může zobrazit stažená data od REST API v gridu.	5	UC04

ID	Název požadavku	Obsah požadavku	Priorita požadavku	Související případ užití
FP05	Aktualizace zobrazených dat v gridu	Uživatel může aktualizovat zobrazená data v gridu při opětovném odeslání dotazu či zadání nového dotazu.	4	UC05
FP06	Možnost stažení kompletní odpovědi od REST API	Uživatel si může stáhnout kompletní odpověď od vybraného REST API.	3	UC06
FP07	Možnost exportu gridu do excelu	Uživatel si muže vyexportovat aktuální obsah gridu do formátu pro MS Excel.	2	UC07

4.4.2 Nefunkční požadavky

Za účelem pokrytí nároků na nefunkční aspekty navrhované aplikace byly kromě funkčních požadavků definovány také nefunkční požadavky. V následující tabulce jsou zachyceny všechny stanovené nefunkční požadavky, které jsou stejně jako v případě funkčních požadavků ohodnoceny stupni priority 1-5.

Tabulka 5: Nefunkční požadavky (zdroj: autor)

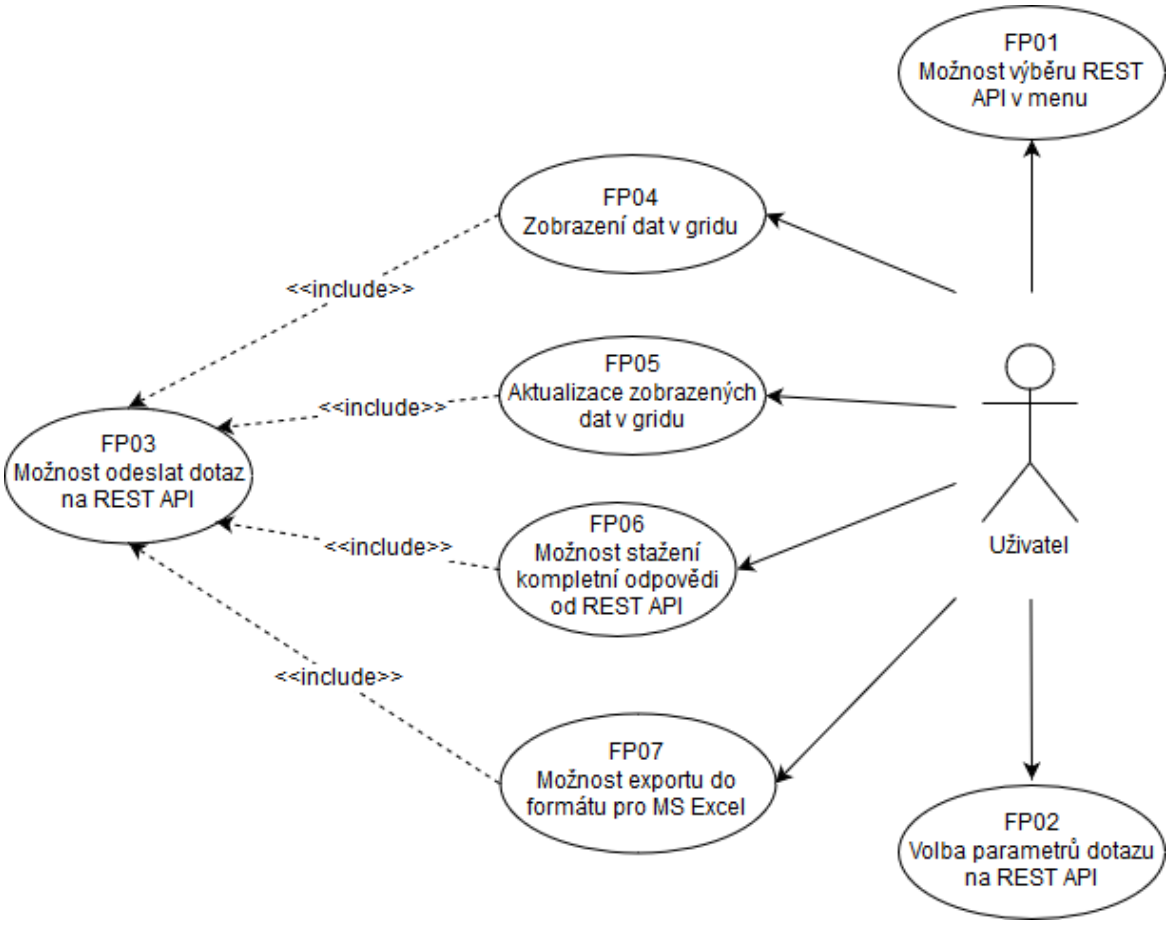
ID	Kategorie požadavku	Název požadavku	Obsah požadavku	Priorita požadavku
NP01	Výkon	Doba načtení aplikace	Aplikace je při svém prvním spuštění načtena v čase maximálně 3 vteřin.	5
NP02	Výkon	Doba zpracování dotazu na vybrané REST API	Doba zpracování jednoho dotazu na vybrané REST API nepřesahuje 15 vteřin.	5
NP03	Spolehlivost	Spolehlivost přenosu dat	Při přenosech dat směrem od aplikace k danému REST API a od REST API k aplikaci nedochází k žádné ztrátě či poškození přenášených dat.	5
NP04	Použitelnost	Jazyková lokalizace aplikace	Uživatelské rozhraní aplikace je vytvořeno kompletně v českém jazyce.	3

ID	Kategorie požadavku	Název požadavku	Obsah požadavku	Priorita požadavku
NP05	Použitelnost	Intuitivnost ovládání aplikace	Ovládání celé aplikace je na první pohled jasné a intuitivní.	4
NP06	Použitelnost	Přehlednost uspořádání funkčních prvků aplikace	Veškeré funkční prvky aplikace, jako jsou formuláře a tlačítka, jsou přehledně uspořádány.	3
NP07	Použitelnost	Přehlednost zobrazení výsledných dat uživateli	Výsledná data, zaslaná jako odpověď od vybraného REST API jsou přehledně zobrazena a zformátována v gridu.	4
NP08	Použitelnost	Design uživatelského rozhraní	Celé uživatelské rozhraní disponuje jednotným a graficky sladěným designem.	2
NP09	Použitelnost	Přehlednost vyexportovaných dat do formátu pro MS EXCEL	Data vyexportovaná do formátu pro MS Excel jsou přehledně zformátována a jsou čitelná.	4
NP10	Systémová omezení	Vývojové prostředí	IntelliJ IDEA	5
NP11	Systémová omezení	Programovací jazyk	Oracle Java, verze 1.8	5
NP12	Systémová omezení	Aplikační server	Apache Tomcat verze 8	5
NP13	Systémová omezení	Framework pro komunikaci s REST API	Vaadin verze 8, Vaadin verze 10 (Vaadin Flow)	5
NP14	Systémová omezení	Kompatibilita s operačními systémy	Aplikace je kompatibilní s operačním systémem Microsoft Windows 10.	4
NP15	Systémová omezení	Kompatibilita s webovými prohlížeči	S aplikací lze pracovat ve webových prohlížečích Mozilla Firefox a Google Chrome.	4

ID	Kategorie požadavku	Název požadavku	Obsah požadavku	Priorita požadavku
NP16	Systémová omezení	Knihovna pro zajištění logování	Pro logování je využito fasády SLF4J v kombinaci s frameworkem Log4j.	3
NP17	Systémová omezení	Formát pro ukládání stažených dat	Data zobrazená v gridu je možné stáhnout jako soubor ve formátu JSON.	4
NP18	Systémová omezení	Formát pro exportování stažených dat	Data vyexportovaná z gridu pro použití v MS Excel jsou exportována ve formátu XLSX.	4
NP19	Zabezpečení	Odolnost formulářových polí vůči útokům	Všechna formulářová pole jsou koncipována tak, aby nebylo možné provádět XSS útoky či vkládat škodlivé skripty do formulářových polí.	4

4.5 Případy užití

Pro detailnější popis funkcionality aplikace byl na každý funkční požadavek navázán jeden případ užití. Celkové znázornění všech případů užití je zachyceno na následujícím obrázku.



Obrázek 9: Diagram případů užití pro ukázkovou aplikaci (zdroj: autor)

Z obrázku vyplývá, že existují tři hlavní případy užití, které jsou pro funkčnost celé aplikace klíčové. Prvním klíčovým případem užití je výběr daného REST API, ze kterého si uživatel přeje stahovat data a který představuje první krok uživatele po spuštění aplikace. Druhým klíčovým případem užití je volba parametrů pro odesílání samotného dotazu na dané REST API, bez kterých by nebylo možné daný dotaz na REST API odeslat. Třetím a zároveň nejkomplexnějším případem užití je možnost odeslání dotazu na REST API, na který jsou dále navázány další případy užití, obsahující zobrazování dat v gridu, jejich aktualizace a možnosti stažení kompletní odpovědi od REST API ve formátech JSON a XLSX.

Pro větší přehlednost byly výše zmíněné tři klíčové případy užití rozepsány do detailnější podoby ve formě tabulek s vybranými atributy na základě analýzy (TemplateLab, 2019).

Tabulka 6: Případ užití FP01 (zdroj: autor)

Název	Možnost výběru REST API v menu
ID	FP01
Popis	Uživatel může v základním menu aplikace vybrat příslušné REST API, ze kterého si přeje stahovat data
Aktér	Uživatel
Vyvolání akce	Kliknutí na položku z menu
Počáteční podmínky	Aplikace je spuštěna
Koncové podmínky	Je vybráno příslušné REST API
Primární průběh	1. Uživatel si vybere z menu příslušné REST API kliknutím na položku z menu.
Alternativní průběh	Není
Výjimky	Nejsou

Tabulka 7: Případ užití FP02 (zdroj: autor)

Název	Volba parametrů dotazu na REST API
ID	FP02
Popis	Uživatel může zvolit parametry odesílaného dotazu na vybrané REST API prostřednictvím formuláře
Aktér	Uživatel
Vyvolání akce	Vyplnění formulářových polí
Počáteční podmínky	Je vybráno příslušné REST API
Koncové podmínky	Jsou vyplněna všechna příslušná formulářová pole
Primární průběh	1. Uživatel vyplní zobrazená formulářová pole výběrem příslušné hodnoty z nabídky či kalendáře. 2. Uživatel zkontroluje, že nenechal žádné ze zobrazených formulářových polí nevyplněné.
Alternativní průběh	Není
Výjimky	Nejsou

Tabulka 8: Případ užití FP03 (zdroj: autor)

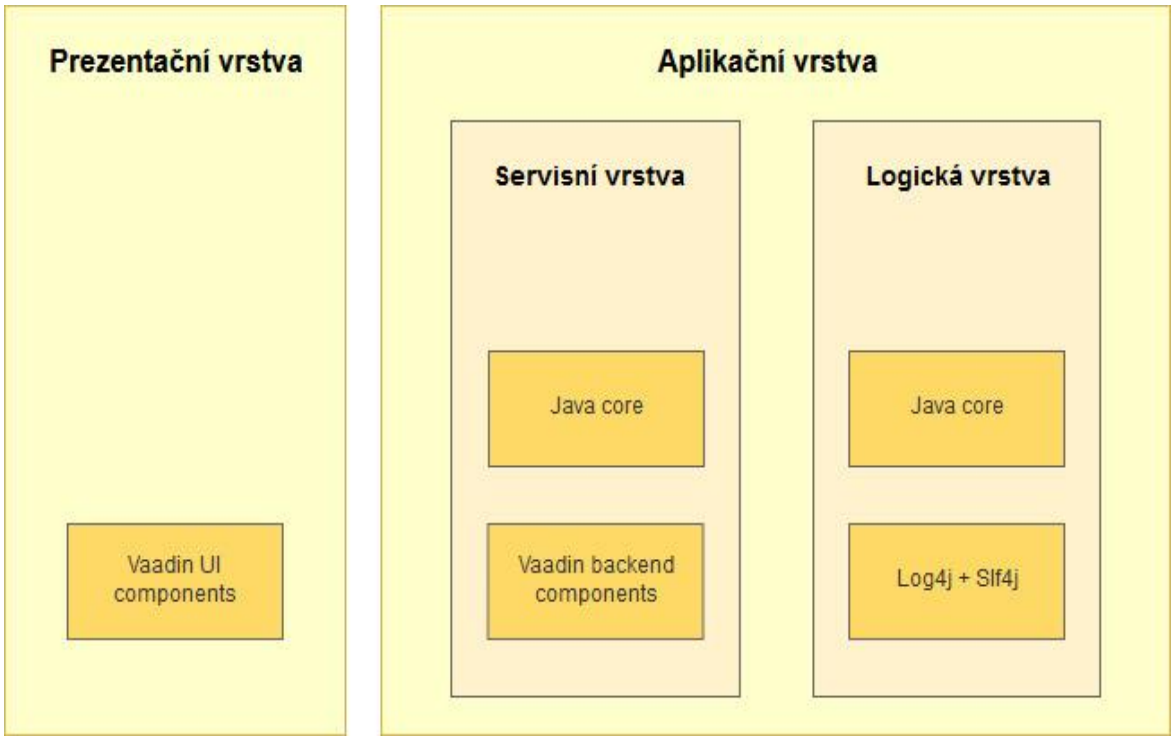
Název	Možnost odeslat dotaz na REST API
ID	FP03
Popis	Uživatel může pomocí tlačítka vedle formuláře odeslat dotaz na REST API s parametry, které zadal do formuláře
Aktér	Uživatel
Vyvolání akce	Stisknutí tlačítka Odeslat dotaz
Počáteční podmínky	Jsou vyplněna příslušná formulářová pole
Koncové podmínky	Koncové REST API přijme dotaz
Primární průběh	1. Uživatel klikne na tlačítko „Odeslat dotaz“ vedle vyplněných formulářových polí.
Alternativní průběh	Není
Výjimky	Nejsou

4.6 Architektura aplikace

Navrhovaná aplikace pro naplnění stanovených funkčních a nefunkčních požadavků využívá kromě základních knihoven Javy a příslušných knihoven frameworku Vaadin také fasádu Slf4j, která v kombinaci s frameworkem Logback umožňuje zaznamenávat probíhající aktivity v aplikaci. S ohledem na úroveň podrobnosti návrhu byla vytvořena architektura aplikace ve dvou úrovních detailu pohledu.

4.6.1 Globální architektura

Z globálního hlediska je architektura aplikace vnímána z hlediska třívrstvého modelu architektury aplikace, obsahujícího jednotlivé vrstvy dle (Guru99, 2019). Globální architektura navrhované aplikace je zachycena na následujícím obrázku.

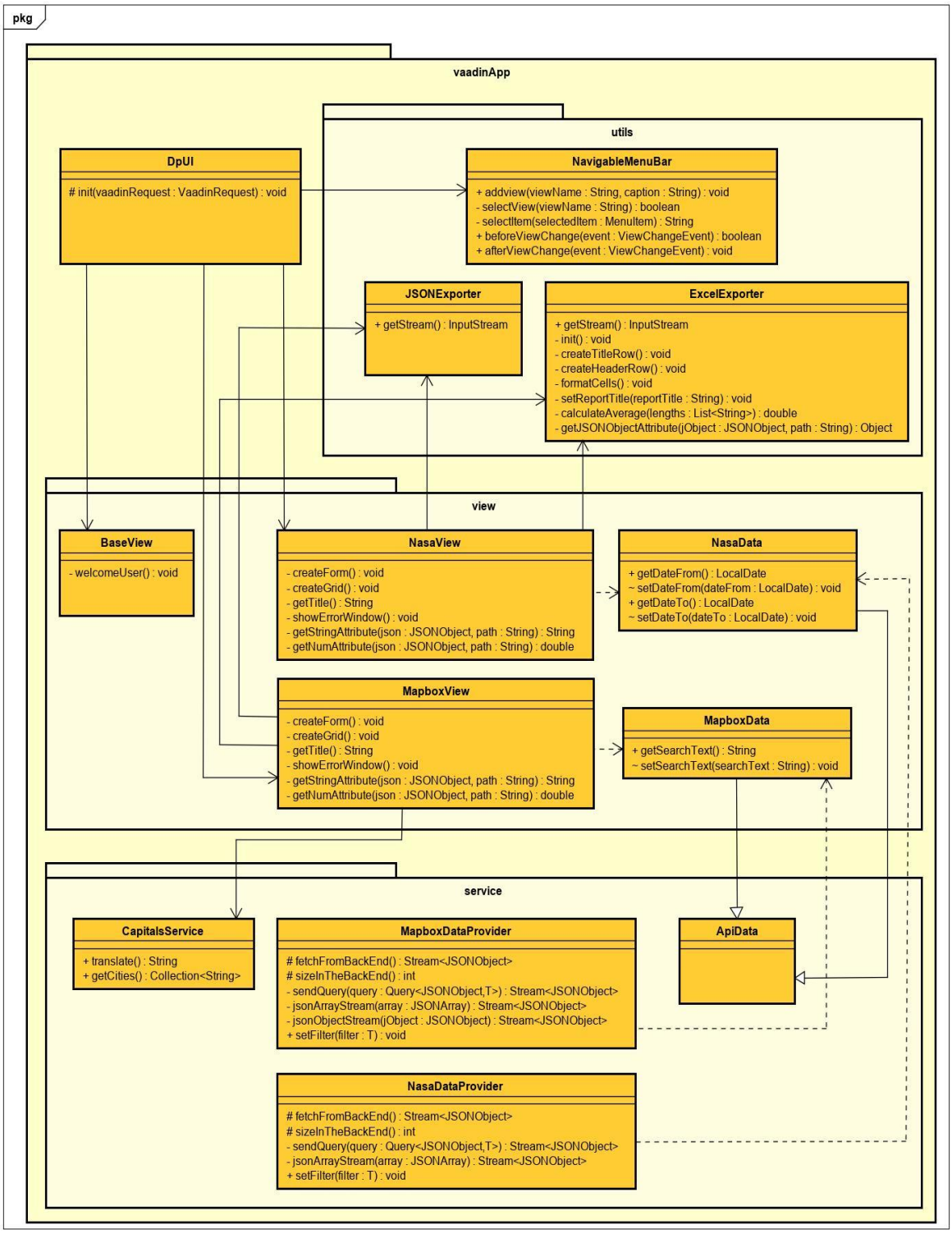


Obrázek 10: Globální architektura ukázkové aplikace (zdroj: autor)

Z obrázku je patrné, že v aplikaci zcela chybí datová vrstva, což je zapříčiněno tím, že v pozadí aplikace neběží žádná databáze a stahovaná data jsou ukládána pouze v cache a zobrazována uživateli. Prezentační vrstva je tak tvořena pouze příslušnými frontendovými komponentami Vaadinu. Aplikační vrstva je pro větší přehlednost rozdělena na servisní a logickou vrstvu, kde servisní vrstva představuje třídy, které obstarávají veškerou komunikaci s externími REST API a je tvořena jak backendovými komponentami Vaadinu, tak základními knihovnami Javy. Logická vrstva naopak zahrnuje potřebnou doplňující logiku pro funkčnost aplikace, postavenou na základních knihovnách Javy, a také logování, prováděné pomocí frameworku Log4j s rozšířením o fasádu Slf4j. Využití fasády Slf4j má tu výhodu, že v kombinaci s ní lze využít i jiné frameworky umožňující logování, jako jsou například Logback či Log4j2, což umožňuje výběr příslušného logovacího frameworku dle aktuální potřeby bez nutnosti provádění změn ve zdrojovém kódu.

4.6.2 Detailní architektura

Detailní architektura se oproti globální architektuře zabývá návrhem jednotlivých tříd a jejich hlavních metod pro implementaci definovaných funkčních a nefunkčních prvků. Detailní architektura navrhované aplikace je zachycena na následujícím obrázku.



Obrázek 11: Detailní architektura ukázkové aplikace (zdroj: autor)

Z obrázku detailní architektury je patrné rozdělení do balíčků, které z velké části odpovídá jednotlivým vrstvám globální architektury aplikace. Základní třídou celé aplikace je třída s názvem DpUI, která rozšiřuje třídu UI, představující základní komponentu pro každou aplikaci postavenou na frameworku Vaadin a která v případě webové aplikace zastupuje klasikou hlavní třídu s názvem Main. V rámci třídy DpUI jsou vytvářeny instance tříd pro

jednotlivá view, kde třída BaseView představuje úvodní obrazovku aplikace, obsahující uvítání uživatele, úvodní text, a dále pak třídy NasaView a MapboxView, které se starají o korektní zobrazování získaných dat od NASA API respektive Mapbox Geocoding API. Tyto dvě třídy v sobě dále obsahují vnořené třídy, které slouží jako POJO třídy pro přenášení dat mezi formuláři, kam uživatelé zadávají parametry pro jednotlivé dotazy, a data providery. Z toho důvodu jsou tyto dvě třídy, konkrétně NasaData a MapboxData, součástí balíčku *view*, který by jinak obsahoval pouze jednotlivá view představující jednotlivé obrazovky. Všechny tři třídy představující jednotlivá view jsou rozšířením komponenty View, která je jednou ze základních frontendových komponent Vaadinu. Dále je v diagramu obsažen balíček *utils*, který z hlediska globální architektury spadá pod logickou vrstvu aplikace. Tento balíček obsahuje třídu NavigableMenubar, která vytváří základní menu aplikace, tvořené tlačítky pro jednotlivá REST API, rozšířené o možnost přepínání jednotlivých view pomocí vaadinovské komponenty Navigator. Stejně jako u jednotlivých view je instance této třídy rovněž vytvářena v rámci základní třídy DpUI. Balíček *utils* dále obsahuje třídu JSONExporter, díky které si uživatel může stáhnout kompletní odpověď od daného REST API stisknutím jediného tlačítka, která může posloužit pro hlubší strojovou analýzu získaných dat. Kromě JSONExporteru je v *utils* obsažen také ExcelExporter, který umožňuje vyexportovat data, která jsou právě zobrazena v gridu, do XLSX souboru pro MS Excel, opět stisknutím jediného tlačítka. Takto vyexportovaná data pak mohou sloužit například jako podklad pro manažerskou analýzu. Instance obou tříd pro exporty dat jsou pak vytvářeny v rámci view pro data od NASA API a Mapbox API. Posledním balíčkem je balíček *service*, který z hlediska diagramu globální architektury odpovídá servisní vrstvě aplikace. Komponenty z tohoto balíčku zajišťují přímou komunikaci s vybranými veřejnými REST API a starají se o přenos získaných dat směrem k uživateli. Za tímto účelem jsou v balíčku obsaženy třídy NasaDataProvider a MapboxDataProvider, které rozšiřují základní vaadinovskou komponentu DataProvider, sloužící jako zdroj dat pro grid. Uvedené data providery jsou naprogramovány tak, aby přesně splňovaly konfigurační požadavky a parametry pro komunikaci s uvedenými REST API a aby dále poskytovaly potřebná data gridům v jednotlivých view v čitelné podobě. Pro předávání parametrů využívají oba data providery příslušné, výše zmíněné POJO třídy, které rozšiřují rodičovskou třídu ApiData, vytvořenou z důvodu nutnosti předávání daných parametrů v rámci JSON dat pro zobrazení v jednotlivých gridech, přičemž tyto parametry jsou vždy specifické pro odpověď od vybraného REST API. Poslední třídou v rámci balíčku *service* je třída CapitalsService, která poskytuje názvy jednotlivých hlavních měst a jejich anglické ekvivalenty formuláři pro zadávání dotazů na Mapbox API a jejíž instance je vytvářena pouze v rámci třídy MapboxView. Pro úplnost je třeba zmínit ještě logování, které je obsaženo v příslušných třídách napříč jednotlivými balíčky. Z hlediska globální architektury by však příslušelo k logické vrstvě aplikace. Celkově se tak z hlediska detailní architektury jedná o aplikaci malého rozsahu, která slouží zejména jako ukázková aplikace, prokazující uvedené vlastnosti rozebíraného frameworku.

4.7 Testování aplikace

Za účelem verifikace definovaných požadavků na aplikaci byla navržena a vytvořena sada testovacích případů, které byly nad navrženou aplikací postupně prováděny. Vytvořené testovací případy jsou rozděleny do jednotlivých kategorií podle funkční části, kterou testují. V rámci stanovených kategorií pak testy kompletně pokrývají definovanou funkcionalitu aplikace, aby jejich prostřednictvím došlo k úplnému funkčnímu otestovaní aplikace. K vytvořené sadě testů byla definována také akceptační kritéria, která zajišťují odpovídající úroveň kvality aplikace. Jednotlivé testovací případy jsou vzhledem k jejich množství a rozsahu součástí příloh. Celkově je tak pomocí testovacích případů a akceptačních kritérií zajištěna potřebná úroveň kvality všech částí navržené aplikace.

4.8 Podmínky akceptace

Pro dosažení požadované úrovně kvality ukázkové aplikace byly k sadě vytvořených testovacích případů stanoveny také podmínky její akceptace, aby aplikace dokázala splnit zadané cíle práce. Na základě (IEEE, 2010, s. 8) byly identifikovány následující stupně závažnosti chyb, které se mohou vyskytnout v aplikaci.

Tabulka 9: Stupně závažnosti chyb pro akceptaci (zdroj: IEEE, 2010, s. 8)

Stupeň závažnosti chyby	Popis
Stupeň 1 – Blokující chyba	Testování aplikace není vůbec možné z důvodu její nedostupnosti, neexistuje způsob, jak aplikaci otestovat.
Stupeň 2 – Kritická chyba	Základní funkcionality aplikace jsou zcela nedostupné.
Stupeň 3 – Velmi významná chyba	Základní funkcionality jsou určitým způsobem ovlivněny, ale jsou dostupné.
Stupeň 4 – Méně významná chyba	Jsou ovlivněny vedlejší, méně významné funkcionality aplikace.
Stupeň 5 – Bezvýznamná chyba	Není identifikován žádný dopad na funkcionality aplikace.

Na základě definované stupnice závažnosti chyb v aplikaci byly stanoveny akceptační podmínky, zanesené do tabulky 10.

Tabulka 10: Akceptační podmínky pro ukázkovou aplikaci (zdroj: autor)

Stupeň závažnosti chyby	Přípustný počet chyb
Blokující chyba	0
Kritická chyba	0
Velmi významná chyba	0
Méně významná chyba	1
Bezvýznamná chyba	5

Ze stanovených akceptačních podmínek vyplývají poměrně přísné nároky na ukázkovou aplikaci, které jsou však pro splnění stanovených cílů práce nezbytné.

5 Vývoj aplikace

Na základě specifikovaných funkčních a nefunkčních požadavků, případů užití a vytvořené architektury v globálním a detailním měřítku byla vytvořena ukázková aplikace, prokazující výše popsané vlastnosti Vaadinu a jeho schopnost komunikace s veřejnými REST API a následného zpracování dat, obdržených v odpovědích na dotazy na dané REST API. Při vývoji aplikace byl kladen důraz především na funkčnost a intuitivnost, které byly upřednostněny před grafickým zpracováním uživatelského rozhraní aplikace. Vzhledem k tomu, že se jedná o webovou aplikaci, bylo nutné zprovoznit a nastavit také aplikační server, aby bylo možné tuto aplikaci nasadit a dále s ní pracovat. Všechna potřebná nastavení aplikačního serveru, včetně logování a dalších informací vztažených k vývoji ukázkové aplikace jsou obsaženy dále v této kapitole.

5.1 Postup vývoje aplikace

Při vývoji ukázkové aplikace bylo postupováno podle modifikované metodiky OpenUP, popsané v kapitole 4.3. Nejprve byly definovány jednotlivé funkční a nefunkční požadavky, kterým byly přiřazeny jednotlivé stupně priority podle jejich významu pro splnění zadaných cílů práce. Na funkční požadavky byly navázány také odpovídající případy užití pro detailnější zachycení požadované funkcionality. Dále byl vytvořen hlavní diagram případů užití, který na aplikaci pohlíží z pozice uživatele jako aktéra. K tomuto diagramu byly následně na ukázku vytvořeny tři detailně rozepsané případy užití pro tři nejsložitější funkcionality, které kromě obecného popisu zachycují také akci, která danou funkcionalitu vyvolává, počáteční a koncové podmínky a primární průběh případu užití. Po definici požadavků a vytvoření případů užití následovala tvorba globální architektury, která na aplikaci pohlíží z hlediska jednotlivých vrstev a technologií, které jsou pro vytvoření daných vrstev využity. Na globální architekturu navazuje detailní architektura, která na aplikaci pohlíží z hlediska jednotlivých tříd, seskupených do balíčků podle funkcionality a metod, které dané třídy obsahují. Kromě toho zachycuje detailní architektura také vztahy mezi jednotlivými třídami, ze kterých následně vyplývá postup tvorby aplikace. Po vytvoření globální a detailní architektury následovala definice testovacích případů, vztahujících se k jednotlivým funkčním oblastem aplikace, které mají za cíl verifikovat vytvořenou aplikaci vůči definovaným funkčním požadavkům a zajistit tak potřebou úroveň kvality aplikace. Na závěr byly definovány akceptační podmínky, které definují závažnost chyb, které mohou být potenciálně objeveny v aplikaci, a jejich maximální přípustný počet, odhalený během testování vytvořené aplikace, čímž byla zkompletována analýza, potřebná pro tvorbu aplikace.

Po dokončení analýzy následoval samotný vývoj aplikace, na jehož začátku bylo nejprve nutné zprovoznit a nakonfigurovat aplikační server, na který bude při vývoji aplikace nasazována. Veškerá provedená konfigurace aplikačního serveru je pro větší přehlednost popsána v samostatné kapitole 5.2. včetně potřebných parametrů pro jeho spuštění. Po zprovoznění aplikačního serveru byla vyvíjena aplikace v jednotlivých krátkých iteracích,

které odpovídají agilnímu přístupu, popsaném v modifikované verzi metodiky OpenUP. Na konci každé iterace vždy probíhalo testování, kde byly odhalovány nedostatky a chyby v naprogramovaných funkcionalitách. Na závěr pak byla provedena akceptace vytvořené aplikace na základě stanovených kritérií v rámci provedené analýzy.

5.2 Konfigurace aplikačního serveru

Jelikož je ukázková aplikace webovou aplikací, bylo pro její uvedení do provozu nutné nainstalovat a nakonfigurovat také aplikační server. S ohledem na předchozí zkušenosti z praxe byl jako aplikační server vybrán Apache Tomcat. Jak uvádí (Apache Software Foundation, 2019c), Tomcat je open-source nástroj, který je oficiálně vydávaný pod licencí Apache License version 2. Tento aplikační server implementuje technologie Java, Java Servlet, JavaServer Pages a Java WebSocket a v současnosti je využíván jako prostředí pro provoz rozsáhlých aplikací nejrůznějších organizací napříč odvětvími. Server Apache Tomcat tak lze považovat za zavedenou a ozkoušenou technologii, vhodnou pro účely této práce, která zároveň svými možnostmi přesahuje rámec funkcionality, potřebný pro naplnění cílů práce.

V rámci konfigurace byl nejprve stažen a rozbalen instalační archív obsahující Apache Tomcat verze 8. Před prvním spuštěním serveru bylo nutné nastavit požadované proměnné prostředí v systému Windows tak, aby je mohl Tomcat správně využívat. První potřebnou systémovou proměnnou je proměnná CATALINA_HOME, jejíž hodnotou je systémová cesta ke složce, ve které je aplikační server Tomcat nainstalován. Druhou systémovou proměnnou potřebnou pro spuštění serveru je proměnná JAVA_HOME, která ukazuje na složku, kde je nainstalována Java JDK. Po specifikaci proměnných prostředí bylo nutné nakonfigurovat samotný aplikační server tak, aby bylo v rámci vývoje aplikace možné vzdáleně nasazovat aktuální verzi aplikace na server a také ji vzdáleně debugovat. Pro vzdálené nasazování byl nejprve upraven konfigurační soubor *tomcat-users* do podoby, jaká je zachycena v následující ukázce kódu.

```
<tomcat-users>

    <role rolename="manager-gui"/>
    <role rolename="manager-status"/>
    <role rolename="manager-script"/>
    <role rolename="manager-jmx"/>

    <user username="admin" password="" roles="manager-gui, manager-status, manager-script, manager-jmx"/>

</tomcat-users>
```

Ukázka kódu 12: Obsah konfiguračního souboru tomcat-users (zdroj: autor)

Tomcat-users je konfigurační soubor, ve kterém jsou definovány jednotlivé role pro nástroj Tomcat manager, který dle (Apache Software Foundation, 2019d) umožňuje vzdálené nasazování webových aplikací na server. Pro jednoduchost práce s tímto managerem byly

v konfiguračním souboru definovány všechny základní role, definované v oficiální dokumentaci. Tyto role a jejich význam jsou:

- manager-gui: přístup do HTML rozhraní,
- manager-status: přístup ke stránce poskytující informace o stavu serveru,
- manager-script: přístup k plain-text rozhraní, umožňující vzdálené nasazování aplikací,
- manager-jmx: přístup k JMX proxy rozhraní a stránce o stavu serveru.

Výše uvedeným rolím byl v rámci konfigurace vytvořen účet admin, který byl vzhledem k povaze aplikace, která je určena ke spouštění pouze na localhostu, ponechán bez hesla. Dále byl do mavenovského projektu aplikace za účelem vzdáleného deploymentu přidán plugin, který umožňuje provádět základní operace související s nasazováním webové aplikace na vzdálený server. Uvedený plugin má tyto parametry:

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-release-plugin</artifactId>
    <version>2.4.2</version>
</plugin>
```

Ukázka kódu 13: Plugin pro vzdálený deployment na Apache Tomcat (zdroj: autor)

Uvedený plugin vzdáleně komunikuje s nástrojem Tomcat manager a díky tomu je možné nasazovat aplikaci na Apache Tomcat přímo z prostředí IntelliJ IDEA.

Pro zprovoznění vzdáleného debugování bylo nutné vytvořit dávkový soubor *setenv.bat* a přidat ho do složky s nainstalovaným Tomcatem, ke které byla dříve nastavena cesta v rámci systémové proměnné CATALINA_HOME. Do uvedeného dávkového souboru byl následně přidán následující konfigurační řádek:

```
set CATALINA_OPTS="-agentlib:jdwp=transport=dt_socket,server=y,suspend=n,address=8000"
```

Ukázka kódu 14: Konfigurace pro vzdálené debugování (zdroj: autor)

Uvedený konfigurační řádek obsahuje nastavení, odpovídající potřebné konfiguraci pro vzdálené testování, získané z prostředí IntelliJ IDEA z možností nastavení vzdálených aplikační serverů. Řádek zavádí novou proměnnou CATALINA_OPTS, ve které jsou specifikována doplňující nastavení pro Apache Tomcat a která je v tomto případě definována nikoliv na úrovni operačního systému, ale na úrovni instalace aplikačního serveru. Tato proměnná specifikuje způsob komunikace mezi aplikačním serverem a debuggerem, kterou je v tomto případě, jak uvádí také (Paraschiv, 2017), možnost *dt_socket* neboli přenos dat prostřednictvím tzv. socketového rozhraní, dále pomocí parametru *server=y* stanovuje, že se virtuální stroj Javy bude chovat jako debugovací server a samotný debugger se bude připojovat jako klient. Parametr *suspend=y* určuje, že činnost virtuálního stroje bude

pozastavena ještě před načtením základní třídy a jeho činnost bude znovu pokračovat po připojení debuggeru. Závěrečný parametr *address=8000* pak definuje port Tomcatu, ke kterému se bude debugger připojovat.

Po nastavení a přidání výše uvedených konfigurací je následně přes vybrané IDE, kterým je v případě této práce IntelliJ IDEA, možné libovolně zkoušet a nasazovat vyvíjenou aplikaci na aplikační server Apache Tomcat a provádět její vzdálené debugování.

5.3 Struktura aplikace

Vyvinutá aplikace disponuje základní balíčkovou strukturou, která využívá zavedené konvence pro aplikace, postavené na jazyku Java. Základní balíček, ve kterém se nacházejí vytvořené knihovny, byl pojmenován *cz.xgald01.dp*, na základě jazykové lokalizace aplikace, xname autora a zkratce pro diplomovou práci. Tento balíček dále obsahuje tři vnořené balíčky, rozdělené podle oblasti funkcionality, kterou obsažené třídy pokrývají, a dále obsahuje také základní třídu celé aplikace, stojící samostatně mimo vnořené balíčky. Zmíněné tři vnořené balíčky jsou *service*, *utils* a *view*. Balíček *service* obsahuje třídy, které komunikují s externími REST API a které následně obstarávají data pro zobrazení uživateli v gridu. Kromě toho obsahuje také třídu poskytující názvy evropských hlavních měst a jejich anglických ekvivalentů ve formuláři pro dotazy na Mapbox API. Následuje balíček *utils*, jehož náplní jsou třídy, které představují speciální nástroje pro export dat a vytvoření základního menu, které dokáže přepínat jednotlivé obrazovky dle zvoleného REST API. Nástroje pro export zahrnují třídy JSONExporter a ExcelExporter, které umožňují export stažených, respektive zobrazených dat do formátů JSON a XLSX. Třetí balíček *view* obsahuje třídy, které vytvářejí jednotlivé obrazovky v aplikaci. Konkrétně se jedná o základní obrazovku s uvítacím nápisem pro uživatele a obrazovky pro interakci s oběma vybranými REST API. Co se týče souborů obsahujících kaskádové styly, které se starají o celkové uspořádání a vzhled frontendu, tyto soubory jsou při použití frameworku Vaadin automaticky generovány do odděleného balíčku s názvem *webapp*, který se nachází na stejné úrovni, jako balíček *java*, který je určen pro veškeré vytvořené knihovny v rámci aplikace a který v případě této aplikace obsahuje již zmíněný balíček *cz.xgald01.dp* a konfigurační soubor pro logování prostřednictvím fasády Slf4j. Celkově tak daná struktura odpovídá běžné struktuře webové aplikace malého rozsahu.

5.4 Logování

Vytvořená aplikace podporuje kromě specifikovaných funkcionalit také logování všech důležitých událostí v aplikaci do konzole. Správce aplikace tak může v případě nenadálých chyb v aplikaci sledovat posloupnost provedených akcí a rychleji odhalit zdroj dané chyby. Pro implementaci logování byl využit logovací framework Log4j v kombinaci s fasádou Slf4j. Jak uvádí (Paraschiv, 2018), hlavním motivem pro využití fasády Slf4j je možnost oddělení aplikace od souvisejícího loggeru a napojení na logovací framework až v době nasazování aplikace, což poskytuje vývojářům možnost změny logovacího nástroje bez

nutnosti úpravy stávajícího zdrojového kódu, jelikož zejména na začátku vývoje nemusí být vždy jasné, který z dostupných logovacích frameworků je pro danou aplikaci nejvhodnější. Uvedené rozpojení aplikace a logovacího frameworku následně implikuje značnou výhodu Sl4fj, kterou je možnost volby mezi více frameworky pro logování ve spojení s touto fasádou, jelikož Slf4j představuje pouze abstraktní vrstvu. Nejčastěji využívanými logovacími frameworky ve spojení se Slf4j jsou frameworky Log4j, využitý v ukázkové aplikaci, Log4j2 a Logback. (Paraschiv, 2018) dále tvrdí, že mezi další výhody fasády Slf4j patří celková jednoduchost použití, podpora parametrizace logovaných zpráv, dostupnost přehledné dokumentace a také celková nenáročnost na výpočetní zdroje.

Pro implementaci logování byl vytvořen konfigurační soubor s názvem *logging.properties*, který byl přidán do mavenovského projektu s aplikací. Obsah konfiguračního souboru je zachycen v následující ukázce kódu.

```
# Nastaveni rootLoggeru na info
log4j.rootLogger=INFO, stdout
# Logy do konzole
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.Target=System.out
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %-5p %c{1} - %m%n
```

Ukázka kódu 15: Obsah konfiguračního souboru pro Log4j (zdroj: autor)

Z uvedeného kódu je patrné, že jsou logovány základní události se stupněm závažnosti označeným jako INFO, které zpravidla informují o základních operacích, provedených v aplikaci. Vzhledem k cíli práce a rozsahu aplikace nebyly jiné stupně závažnosti logovaných zpráv definovány. Z konfigurace dále vyplývá, že logování je prováděno pouze do konzole aplikačního serveru, na kterém aplikace běží, a také že součástí každé logované zprávy je časové razítko, zachycující přesné datum a čas, kdy byla daná akce v aplikaci provedena. Aplikace tak díky logování poskytuje alespoň základní nástroj pro monitorování událostí, které na jejím pozadí na základě provedených uživatelských akcí probíhají.

6 Testování aplikace

Vyvinutá aplikace demonstrující schopnosti frameworku Vaadin byla komplexně otestována pomocí vytvořené sady testovacích případů za účelem verifikace správnosti vytvořené aplikace. Jednotlivé testovací případy byly vytvořeny tak, aby pokrývaly specifikované funkční požadavky a jejich provedením je tak zajištěno, že ukázková aplikace tyto požadavky splňuje a disponuje náležitou úrovní kvality.

6.1 Sumarizace provedených testovacích případů

Při testování aplikace bylo v rámci jednotlivých iterací postupně provedeno všech 28 vytvořených testovacích případů, zaměřených na jednotlivé funkční části aplikace. Celkový výsledek provedených testů je zachycen v následující tabulce.

Tabulka 11: Přehled provedených testovacích případů (zdroj: autor)

ID	Název	Stav
TC_01	Hlavní menu - tlačítka	Úspěšný
TC_02	NASA - formulář	Úspěšný
TC_03	Mapbox - formulář	Úspěšný
TC_04	NASA – formulář - tlačítko	Úspěšný
TC_05	Mapbox – formulář - tlačítko	Úspěšný
TC_06	NASA – tlačítka pro export	Úspěšný
TC_07	Mapbox – tlačítka pro export	Úspěšný
TC_08	Hlavní stránka – doplňující informace	Úspěšný
TC_09	NASA – formulář – invalidní hodnoty	Úspěšný
TC_10	NASA – formulář – validní hodnoty	Úspěšný
TC_11	Mapbox – formulář – invalidní hodnoty	Úspěšný
TC_12	Mapbox – formulář – validní hodnoty	Úspěšný
TC_13	NASA – validní dotaz	Úspěšný

ID	Název	Stav
TC_14	Mapbox – validní dotaz	Úspěšný
TC_15	NASA – řádky gridu - oddělení	Úspěšný
TC_16	NASA – řádky gridu - editace	Úspěšný
TC_17	NASA – aktualizace gridu	Úspěšný
TC_18	Mapbox – řádky gridu - oddělení	Úspěšný
TC_19	Mapbox – řádky gridu - editace	Úspěšný
TC_20	Mapbox – aktualizace gridu	Úspěšný
TC_21	NASA – export do JSON - export	Úspěšný
TC_22	NASA – export do JSON - kontrola	Úspěšný
TC_23	NASA – export pro MS Excel - export	Úspěšný
TC_24	NASA – export pro MS Excel - kontrola	Úspěšný
TC_25	Mapbox – export do JSON - export	Úspěšný
TC_26	Mapbox – export do JSON - kontrola	Úspěšný
TC_27	Mapbox – export pro MS Excel - export	Úspěšný
TC_28	Mapbox – export pro MS Excel - kontrola	Úspěšný

Z tabulky je patrné, že všech 28 testovacích případů skončilo úspěchem. Výsledek testování lze interpretovat tak, že vytvořená aplikace nevykazuje chyby v rámci oblastí testovaných vytvořenými testovacími případy a dosahuje tak potřebné úrovně kvality. Jelikož na základě tvrzení od (Morgan a kolektiv, 2010, s. 16) není možné jednou sadou testů odhalit všechny možné chyby, které se mohou v aplikaci vyskytnout, a zároveň není vzhledem ke stanoveným cílům a rozsahu práce možné pokrýt testovacími případy kompletně všechny kombinace dostupných možností, lze aplikaci označit nikoliv za bezchybnou, nýbrž za dostatečně kvalitní pro splnění stanovených cílů této práce.

6.2 Vyhodnocení akceptačních kritérií

V souvislosti s provedenými testovacími případy je nutné provést také vyhodnocení definovaných akceptačních kritérií ohledně povoleného množství nalezených chyb v aplikaci. Skutečné počty chyb, nalezených vytvořenými testovacími případy, jsou zachyceny níže v tabulce.

Tabulka 12: Vyhodnocení definovaných akceptačních kritérií (zdroj: autor)

Stupeň závažnosti chyby	Přípustný počet chyb	Nalezený počet chyb
Blokující chyba	0	0
Kritická chyba	0	0
Velmi významná chyba	0	0
Méně významná chyba	1	0
Bezvýznamná chyba	5	0

Z výše uvedené tabulky je zřejmé, že provedením daných testovacích případů nebyly nalezeny žádné chyby. Na základě toho je možné prohlásit, že vytvořená aplikace naplňuje definované funkční požadavky a lze ji akceptovat.

Závěr

Hlavním cílem této práce bylo prokázat možnosti využití frameworku Vaadin pro komunikaci s vybranými veřejnými REST API a následovnou práci s daty, získanými prostřednictvím dotazů na jednotlivá REST API v rámci ukázkové aplikace. Toho bylo dosaženo definováním tří dílčích cílů, z nichž každý přispívá k naplnění hlavního zadaného cíle této práce.

Prvním dílčím cílem práce bylo zajistit veškeré potřebné teoretické podklady pro práci s frameworkem Vaadin, které v rámci práce slouží jako znalostní báze pro následující části práce. V rámci teoretické části práce byla nejprve uvedena stručná historie jednotlivých verzí Vaadinu, která uvádí jednotlivé novinky a aktualizace, které jednotlivé verze přinášejí. Dále jsou v teoretické části uvedeny základní vlastnosti rozebíraného frameworku, které slouží jako technický znalostní podklad. Po vlastnostech následuje definice jednotlivých komponent frameworku, potřebných pro vytvoření ukázkové aplikace, popsané v praktické části práce, která zadanou komunikaci s vybranými REST API umožňuje. Kromě komponent byly v rámci teoretické části zjištěny také odborné a laické názory na práci s frameworkem Vaadin, které reprezentují názor veřejnosti na rozebíraný framework, a také bylo provedeno porovnání současných konkurenčních frameworků pro jazyk Java, které na základě určených kritérií srovnává jednotlivé frameworky a následně určuje nejlepší alternativy k Vaadinu. Z provedeného srovnání vyšly nejlépe frameworky Spring a Grails, přičemž výsledek u frameworku Spring je značně ovlivněn jeho velikostí v porovnání s ostatními uvedenými frameworky, která má vliv na jeho hodnocení v rámci porovnávaných kritérií. Z toho důvodu byl jako druhý vítěz zvolen framework Grails, který naopak představuje spíše jednodušší framework a svými vlastnostmi vyniká nad zbývajícími frameworky. K provedenému srovnání je však nutné dodat, že je zde výrazná závislost na definovaných srovnávacích kritériích a případné změny v daných kritériích by mohly mít značný vliv na výsledek provedeného srovnání. V poslední řadě obsahuje teoretická část také základní teoretické podklady pro práci s vybranými REST API od společností NASA a Mapbox, které jsou dále využity v analytické a praktické části práce.

Druhým dílčím cílem této práce bylo zajistit veškeré potřebné analytické podklady pro vývoj ukázkové aplikace, čehož bylo dosaženo v analytické části práce. V rámci analýzy byla stručně charakterizována vytvářená aplikace včetně metodiky, podle které byla vyvíjena, přičemž z praktických důvodů byla daná metodika upravena a přizpůsobena pro potřeby této práce. Dále byly definovány jednotlivé funkční a nefunkční požadavky na aplikaci a související případy užití, které musela ukázková aplikace naplnit. Po specifikaci požadavků následovalo v rámci analýzy vytvoření diagramů globální a detailní architektury ukázkové aplikace, které zachycují architekturu aplikace z hlediska různých úrovní detailu. Na závěr analytické části práce byla vytvořena sada testovacích případů, sloužících k verifikaci ukázkové aplikace, k níž byla připojena také akceptační kritéria zajišťující, že výsledná aplikace bude disponovat potřebnou úrovní kvality.

Třetí dílčí cíl představoval vývoj samotné ukázkové aplikace, ověřující získané teoretické poznatky v praxi a demonstrující schopnost vybraného frameworku komunikovat s vybranými REST API. Vývoj aplikace probíhal s využitím definovaných vlastností a komponent Vaadinu z teoretické části práce a na základě analytických podkladů z analytické části práce. V rámci praktické části práce byla provedena také konfigurace aplikačního serveru Apache Tomcat, který umožňuje spuštění a provoz dané aplikace. Veškerá potřebná konfigurace vybraného aplikačního serveru je tak součástí praktické části práce. Na závěr bylo provedeno vyhodnocení testování aplikace pomocí vytvořené sady testovacích případů a byla provedena její akceptace. Vzhledem k naplnění zadaných požadavků a úspěšnému vývoji aplikace skončily všechny definované testovací případy úspěchem a aplikace tak mohla být akceptována. Zároveň tím došlo k naplnění třetího zadaného cíle práce, čímž byl také naplněn hlavní cíl práce.

Práci je možné dále rozšířit o další komponenty Vaadinu a následně s jejich využitím také rozšířit vyvinutou ukázkovou aplikaci. Definovaná sada testovacích případů přitom může sloužit jako podklad pro testování základních funkcionalit dané aplikace. Celkově také může tato práce sloužit jako podklad pro další výzkum, prováděný za pomoci frameworku Vaadin.

Seznam citované literatury

APACHE SOFTWARE FOUNDATION, 2019a. Documentation. *Apache Wicket* [online]. [cit. 2019-02-08]. Dostupné z: https://wicket.apache.org/learn/

APACHE SOFTWARE FOUNDATION, 2019b. The Grails Framework. *Grails* [online]. [cit. 2019-02-09]. Dostupné z: http://docs.grails.org/latest/guide/single.html

APACHE SOFTWARE FOUNDATION, 2019c. *Apache Tomcat* [online]. [cit. 2019-03-24]. Dostupné z: http://tomcat.apache.org/

APACHE SOFTWARE FOUNDATION, 2019d. Manager App HOW-TO. *Apache Tomcat* [online]. [cit. 2019-03-24]. Dostupné z: https://tomcat.apache.org/tomcat-8.5-doc/manager-howto.html#Introduction

ARTAMONOV, Yuriy, 2018. Vaadin Flow: A Dev's Thoughts on the Future of Vaadin. *DZone* [online]. [cit. 2019-02-13]. Dostupné z: https://dzone.com/articles/vaadin-flow-a-marvelous-deer

BAELDUNG, 2018. Flyweight Pattern in Java. *Baeldung* [online]. [cit. 2018-12-30]. Dostupné z: https://www.baeldung.com/java-flyweight

BALDUINO, Ricardo, 2007. *Introduction to OpenUP (Open Unified Process)* [online]. The Eclipse Foundation [cit. 2019-03-17]. Dostupné z: https://www.eclipse.org/epf/general/OpenUP.pdf

BOURQUE, Basil, 2018. Understanding Vaadin Flow / Vaadin 10. *Stack Overflow* [online]. [cit. 2019-02-13]. Dostupné z: https://stackoverflow.com/questions/51100108/understanding-vaadin-flow-vaadin-10

ČERNÝ, Ján. *Webový grafický editor transformačních vzorů* [online]. Praha, 2013 [cit. 2019-02-27]. Dostupné z: https://vskp.vse.cz/64326_webovy_graficky_editor_transformacnich_vzoru. Bakalářská práce. Vysoká škola ekonomická v Praze.

DUARTE, Alejandro, 2018. Vaadin 10: The Largest Update in the History of Vaadin. *DZone* [online]. [cit. 2018-12-16]. Dostupné z: https://dzone.com/articles/vaadin-10-the-largest-update-in-the-history-of-vaa

EKBLAD, Sami, 2013. The next reindeer is here - Vaadin 7.0.0 is out. *Vaadin* [online]. [cit. 2018-12-16]. Dostupné z: https://vaadin.com/blog/the-next-reindeer-is-here-vaadin-7-0-0-is-out

FREITAS, Vitor, 2018. How to Use RESTful APIs with Django. *SimpleIsBetterThanComplex* [online]. [cit. 2019-02-17]. Dostupné z: https://simpleisbetterthancomplex.com/tutorial/2018/02/03/how-to-use-restful-apis-with-django.html

G2 CROWD, 2018. Vaadin Reviews. *G2 Crowd* [online]. [cit. 2019-02-13]. Dostupné z: https://www.g2crowd.com/products/vaadin/reviews

GURU99, 2019. *N Tier(Multi-Tier), 3-Tier, 2-Tier Architecture with EXAMPLE* [online]. [cit. 2019-02-24]. Dostupné z: https://www.guru99.com/n-tier-architecture-system-concepts-tips.html

HÄYRY, Johannes, 2018. Vaadin 12 Brings New Components and Improved Performance. *Vaadin* [online]. [cit. 2018-12-17]. Dostupné z: https://vaadin.com/blog/vaadin-12-brings-new-components-and-improved-performance

IEEE, 2010. *IEEE Standard Classification for Software Anomalies* [online]. [cit. 2019-03-15]. Dostupné z: http://www.ctestlabs.org/neoacm/1044_2009.pdf

JANSSON, Jens, 2018. Vaadin 11 is now available with Gradle support and new components. *Vaadin* [online]. [cit. 2018-12-17]. Dostupné z: https://vaadin.com/blog/vaadin-11-is-now-available-with-gradle-support-and-new-components

JAVAPIPE, 2019. 10 Best Java Web Frameworks to Use in 2019. *Javapipe* [online]. [cit. 2019-02-06]. Dostupné z: https://javapipe.com/blog/best-java-web-frameworks/

KOBLENTZ, Adam, 2013. The Ultimate Java Web Frameworks Comparison: Spring MVC, Grails, Vaadin, GWT, Wicket, Play, Struts and JSF. *ZeroTurnaround* [online]. [cit. 2019-02-05]. Dostupné z: https://zeroturnaround.com/rebellabs/the-curious-coders-java-web-frameworks-comparison-spring-mvc-grails-vaadin-gwt-wicket-play-struts-and-jsf/

KOCH, Ondřej. *Katalogizace webových služeb založených na architektuře REST* [online]. Praha, 2013 [cit. 2019-02-27]. Dostupné z: https://vskp.vse.cz/34757_katalogizace_webovych_sluzeb_zalozenych_na_architekture_rest. Diplomová práce. Vysoká škola ekonomická v Praze.

KOZÁK, Martin. *Webový grafický editor báze znalostí pro expertní systém NEST* [online]. Praha, 2016 [cit. 2019-02-27]. Dostupné z: https://vskp.vse.cz/48725_webovy_graficky_editor_baze_znalosti_pro_expertni_system_nest. Bakalářská práce. Vysoká škola ekonomická v Praze.

KRONQUIST, Jonatan, 2018. Stability in production – Bleeding edge in development. *Vaadin* [online]. [cit. 2018-12-17]. Dostupné z: https://vaadin.com/blog/the-revaadin-platform

LIEW, Zell, 2018. Understanding And Using REST APIs. *Smashing Magazine* [online]. [cit. 2019-02-15]. Dostupné z: https://www.smashingmagazine.com/2018/01/understanding-using-rest-api/

LIGHTBEND, 2018. What is Play?. *PlayFramework* [online]. [cit. 2019-02-07]. Dostupné z: https://www.playframework.com/documentation/2.7.x/Introduction

MALHOTRA, Mantra, 2018. Top 10 Java Frameworks For Startups 2019. *Medium* [online]. [cit. 2019-02-06]. Dostupné z: https://medium.com/swlh/top-java-web-development-frameworks-for-startups-2019-f02ce1fca3db

MANAGEMENT BYZNYS INFORMATIKY, 2019. M524: OpenUP. *MBI VŠE* [online]. [cit. 2019-03-17]. Dostupné z: https://mbi.vse.cz/public/cs/obj/METHOD-89

MAPBOX, 2019a. About Mapbox. *Mapbox* [online]. [cit. 2019-02-17]. Dostupné z: https://www.mapbox.com/about/

MAPBOX, 2019b. *Mapbox* [online]. [cit. 2019-02-17]. Dostupné z: https://www.mapbox.com/

MAPBOX, 2019c. Geocoding. *API Documentation* [online]. [cit. 2019-03-11]. Dostupné z: https://docs.mapbox.com/api/search/#geocoding

MAPBOX, 2019d. Pricing. *Mapbox* [online]. [cit. 2019-02-18]. Dostupné z: https://www.mapbox.com/pricing/

MONUS, Anna, 2018. 17 Popular Java Frameworks [2018 edition]: Pros, cons, and more. *Raygun* [online]. [cit. 2019-02-07]. Dostupné z: https://raygun.com/blog/popular-java-frameworks/

MONUS, Anna, 2019. SOAP vs REST vs JSON comparison. *Raygun* [online]. [cit. 2019-02-17]. Dostupné z: https://raygun.com/blog/soap-vs-rest-vs-json/

MORGAN, Peter, SAMAROO, Angelina, THOMPSON, Geoff a WILLIAMS, Peter. *Software testing: an ISTQB-ISEB Foundation Guide*. 2nd ed. London: British Computer Society, 2010. ISBN 978-190-6124-762.

NASA, 2019a. NASA APIs. *NASA Open APIs* [online]. [cit. 2019-02-17]. Dostupné z: https://api.nasa.gov/index.html

NASA, 2019b. Neo - Feed. *Nasa Open APIs* [online]. [cit. 2019-03-11]. Dostupné z: https://api.nasa.gov/api.html#neows-feed

NASA, 2019c. Authentication. *Nasa Open APIs* [online]. [cit. 2019-02-17]. Dostupné z: https://api.nasa.gov/api.html#authentication

NGUYEN, Tien, 2019. 13.0.2 Including latest maintenance releases. *Github* [online]. [cit. 2019-03-31]. Dostupné z: https://github.com/vaadin/platform/releases/tag/13.0.2

OLA, Segun, 2018. RESTful API Design—PUT vs PATCH. *Medium* [online]. [cit. 2019-02-16]. Dostupné z: https://medium.com/backticks-tildes/restful-api-design-put-vs-patch-4a061aa3ed0b

PARASCHIV, Eugen, 2018. SLF4J: 10 Reasons Why You Should Be Using It. *Stackify* [online]. [cit. 2019-03-23]. Dostupné z: https://stackify.com/slf4j-java/

PARASCHIV, Eugen, 2017. Java Remote Debugging in the Real World. *DZone* [online]. [cit. 2019-03-24]. Dostupné z: https://dzone.com/articles/a-practical-guide-to-java-remote-debugging-in-the

PIVOTAL SOFTWARE INC., 2019. Introduction to Spring Framework. *Spring docs* [online]. [cit. 2019-02-06]. Dostupné z: https://docs.spring.io/spring/docs/3.0.x/spring-framework-reference/html/overview.html

QUORA, 2018. What is your evaluation of Vaadin framework in 2018?. *Quora* [online]. [cit. 2019-02-15]. Dostupné z: https://www.quora.com/What-is-your-evaluation-of-Vaadin-framework-in-2018

RICHARDSON, Leonard a AMUNDSEN, Michael. *RESTful Web APIs* [online]. Beijing: O'Reilly, 2013 [cit. 2019-02-15]. ISBN 978-144-9358-068.

SMITH, Tom, 2018. 2018 Java Ecosystem Executive Insights. *DZone* [online]. [cit. 2019-01-04]. Dostupné z: https://dzone.com/articles/2018-java-ecosystem-executive-insights

SPRINGTUTORIALS, 2015. Spring Framework Ecosystem – Introduction to Spring Projects. *SpringTutorials* [online]. [cit. 2019-02-06]. Dostupné z: http://springtutorials.com/spring-ecosystem/

TAHVONEN, Matti, 2017. Vaadin Framework 8 is out!. *Vaadin* [online]. [cit. 2018-12-16]. Dostupné z: https://vaadin.com/blog/vaadin-framework-8-is-out

TECHOPEDIA, 2019. Business Object. *Techopedia* [online]. [cit. 2019-01-04]. Dostupné z: https://www.techopedia.com/definition/25982/business-object-bo

TEMPLATELAB, 2019. 40 Use Case Templates & Examples. *TemplateLab* [online]. [cit. 2019-02-23]. Dostupné z: http://templatelab.com/use-case-templates/

THE ECLIPSE FOUNDATION, 2009. Resources for Modifying Methods. *OpenUpWeb* [online]. [cit. 2019-03-18]. Dostupné z: http://www.utm.mx/~caff/doc/OpenUPWeb/index.htm

THE ECLIPSE FOUNDATION, 2019a. Documentation. *VertX* [online]. [cit. 2019-02-10]. Dostupné z: https://vertx.io/docs/

THE ECLIPSE FOUNDATION, 2019b. Community. *VertX* [online]. [cit. 2019-02-11]. Dostupné z: https://vertx.io/community/

THE ECLIPSE FOUNDATION, 2019c. Awesome Vert.x. *Github* [online]. [cit. 2019-02-11]. Dostupné z: https://github.com/vert-x3/vertx-awesome

THE ECLISPE FOUNDATION, 2019d. *VertX* [online]. [cit. 2019-02-11]. Dostupné z: https://vertx.io/

TUTORIALSPOINT, 2019. Spring Framework - Overview. *Tutorialspoint* [online]. [cit. 2019-02-06]. Dostupné z: https://www.tutorialspoint.com/spring/spring_overview.htm

VAADIN LTD., 2017. *Book of Vaadin* [online]. [cit. 2018-12-15]. Dostupné z: https://vaadin.com/book

VAADIN LTD., 2019a. *Vaadin Platform* [online]. [cit. 2019-01-02]. Dostupné z: https://vaadin.com/

VAADIN LTD., 2019b. Vaadin Licenses. *Vaadin* [online]. [cit. 2019-01-02]. Dostupné z: https://vaadin.com/license

VAADIN LTD., 2019c. Vaadin Roadmap. *Vaadin* [online]. [cit. 2019-01-04]. Dostupné z: https://vaadin.com/roadmap

VAADIN LTD., 2019d. Success Stories. *Vaadin* [online]. [cit. 2019-01-06]. Dostupné z: https://vaadin.com/success-stories

VAADIN LTD., 2019e. Introduction to Vaadin Flow. *Vaadin* [online]. [cit. 2019-01-08]. Dostupné z: https://vaadin.com/docs/v10/flow/introduction/introduction-overview.html

VAADIN LTD., 2019f. Binding Data to Forms. *Vaadin* [online]. [cit. 2019-01-03]. Dostupné z: https://vaadin.com/docs/v10/flow/binding-data/tutorial-flow-components-binder.html

VAADIN LTD., 2019g. Binding Beans to Forms. *Vaadin* [online]. [cit. 2019-01-03]. Dostupné z: https://vaadin.com/docs/v10/flow/binding-data/tutorial-flow-components-binder-beans.html

VAADIN LTD., 2019h. Flow. *Vaadin* [online]. [cit. 2019-01-04]. Dostupné z: https://vaadin.com/flow

VAADIN LTD., 2019ch. Grid. *Vaadin* [online]. [cit. 2019-01-05]. Dostupné z: https://vaadin.com/docs/v8/framework/components/components-grid.html

VAADIN LTD., 2019i. Showing Many Items in a Listing. *Vaadin* [online]. [cit. 2019-01-05]. Dostupné z: https://vaadin.com/docs/v8/framework/datamodel/datamodel-providers.html#datamodel.dataproviders

VAADIN LTD., 2019j. Field Components. *Vaadin* [online]. [cit. 2019-01-06]. Dostupné z: https://vaadin.com/docs/v8/framework/components/components-fields.html

VAADIN LTD., 2019k. Overview. *Vaadin* [online]. [cit. 2019-01-06]. Dostupné z: https://vaadin.com/docs/v8/framework/layout/layout-overview.html

VAADIN LTD., 2019l. VerticalLayout and HorizontalLayout. *Vaadin* [online]. [cit. 2019-01-06]. Dostupné z: https://vaadin.com/docs/v8/framework/layout/layout-orderedlayout.html

VAADIN LTD., 2019m. Theming Overview. *Vaadin* [online]. [cit. 2019-03-30]. Dostupné z: https://vaadin.com/docs/v10/flow/theme/theming-overview.html

VAADIN LTD., 2019n. Using Component Themes. *Vaadin* [online]. [cit. 2019-03-30]. Dostupné z: https://vaadin.com/docs/v10/flow/theme/using-component-themes.html

VAADIN LTD., 2019o. Defining Routes With @Route. *Vaadin* [online]. [cit. 2019-03-30]. Dostupné z: https://vaadin.com/docs/v10/flow/routing/tutorial-routing-annotation.html

WEBBER, Jim, PARASTATIDIS, Savas a ROBINSON, Ian. REST in Practice: Hypermedia and Systems Architecture. United States of America: O'Reilly, 2010. ISBN 978-0-596-80582-1

Příloha A: Testovací případy

TC_01	Hlavní menu - tlačítka	UI
Krok	**Popis kroku**	**Očekávaný výsledek**
Počáteční podmínka	Není	
Krok 1	Otevřete aplikaci.	Aplikace spuštěna.
Krok 2	Zkontrolujte, že uživatelské menu obsahuje tlačítka s volbami příslušných REST API.	Tlačítka se v menu nacházejí.

TC_02	NASA - formulář	UI
Krok	**Popis kroku**	**Očekávaný výsledek**
Počáteční podmínka	Není	
Krok 1	Otevřete aplikaci.	Aplikace spuštěna.
Krok 2	Vyberte v menu tlačítko s REST API od NASA.	Tlačítko vybráno, je zobrazen formulář pro odesílání dat na NASA API.

TC_03	Mapbox - formulář	UI
Krok	**Popis kroku**	**Očekávaný výsledek**
Počáteční podmínka	Není	
Krok 1	Otevřete aplikaci.	Aplikace spuštěna.
Krok 2	Vyberte v menu tlačítko s REST API od společnosti Mapbox.	Tlačítko vybráno, je zobrazen formulář pro odesílání dat na Mapbox API.

TC_04	NASA – formulář - tlačítko	UI
Krok	**Popis kroku**	**Očekávaný výsledek**
Počáteční podmínka	Není	
Krok 1	Otevřete aplikaci.	Aplikace spuštěna.
Krok 2	Vyberte v menu tlačítko s REST API od NASA.	Tlačítko vybráno, je zobrazen formulář pro odesílání dat na NASA API.
Krok 3	Zkontrolujte, že se vedle formuláře nachází tlačítko „Odeslat dotaz“ pro odeslání dotazu na NASA API.	Tlačítko se vedle formuláře nachází.

TC_05	Mapbox – formulář - tlačítko	UI
Krok	**Popis kroku**	**Očekávaný výsledek**
Počáteční podmínka	Není	
Krok 1	Otevřete aplikaci.	Aplikace spuštěna.
Krok 2	Vyberte v menu tlačítko s REST API od společnosti Mapbox.	Tlačítko vybráno, je zobrazen formulář pro odesílání dat na Mapbox API.
Krok 3	Zkontrolujte, že se vedle formuláře nachází tlačítko „Odeslat dotaz“ pro odeslání dotazu na Mapbox API.	Tlačítko se vedle formuláře nachází.

TC_06	NASA – tlačítka pro export	UI
Krok	**Popis kroku**	**Očekávaný výsledek**
Počáteční podmínka	Není	
Krok 1	Otevřete aplikaci.	Aplikace spuštěna.
Krok 2	Vyberte v menu tlačítko s REST API od NASA.	Tlačítko vybráno, je zobrazen formulář pro odesílání dat na NASA API.
Krok 3	Zkontrolujte, že se vedle formuláře nacházejí tlačítka pro export do formátů JSON a XLSX.	Tlačítka se vedle formuláře nacházejí.

TC_07	Mapbox – tlačítka pro export	UI
Krok	**Popis kroku**	**Očekávaný výsledek**
Počáteční podmínka	Není	
Krok 1	Otevřete aplikaci.	Aplikace spuštěna.
Krok 2	Vyberte v menu tlačítko s REST API od společnosti Mapbox.	Tlačítko vybráno, je zobrazen formulář pro odesílání dat na Mapbox API.
Krok 3	Zkontrolujte, že se vedle formuláře nacházejí tlačítka pro export do formátů JSON a XLSX.	Tlačítka se vedle formuláře nacházejí.

TC_08	Hlavní stránka – doplňující informace	UI
Krok	**Popis kroku**	**Očekávaný výsledek**
Počáteční podmínka	Není	
Krok 1	Otevřete aplikaci.	Aplikace spuštěna.
Krok 2	Ujistěte se, že aplikace obsahuje dole na hlavní stránce informace o autorovi a roce svého vzniku.	Aplikace dané informace obsahuje dole na stránce.

TC_09	NASA – formulář – invalidní hodnoty	FORM
Krok	**Popis kroku**	**Očekávaný výsledek**
Počáteční podmínka	Není	
Krok 1	Otevřete aplikaci.	Aplikace spuštěna.
Krok 2	Vyberte REST API od NASA.	Vybráno, je zobrazen formulář pro odesílání dotazů na NASA API.
Krok 3	Vyplňte formulář invalidními hodnotami, například zadejte několik neplatných znaků nebo vyberte datum ve vzdálené budoucnosti.	Neplatné hodnoty nelze do formuláře zadat. Podstata formuláře to uživateli neumožňuje.

TC_10	NASA – formulář – validní hodnoty	FORM
Krok	**Popis kroku**	**Očekávaný výsledek**
Počáteční podmínka	Není	
Krok 1	Otevřete aplikaci.	Aplikace spuštěna.
Krok 2	Vyberte REST API od NASA.	Vybráno, je zobrazen formulář pro odesílání dotazů na NASA API.
Krok 3	Vyplňte formulář validními hodnotami, například výběrem současného kalendářního data.	Formulář vyplněn validními hodnotami.
Krok 4	Klikněte na tlačítko „Odeslat dotaz“.	Dotaz na NASA API je úspěšně odeslán.

TC_11	Mapbox – formulář – invalidní hodnoty	FORM
Krok	**Popis kroku**	**Očekávaný výsledek**
Počáteční podmínka	Není	
Krok 1	Otevřete aplikaci.	Aplikace spuštěna.
Krok 2	Vyberte REST API od společnosti Mapbox.	Vybráno, je zobrazen formulář pro odesílání dotazů na Mapbox API.
Krok 3	Vyplňte formulář invalidními hodnotami, například zadejte několik neplatných znaků.	Neplatné hodnoty nelze do formuláře zadat. Podstata formuláře to uživateli neumožňuje.

TC_12	Mapbox – formulář – validní hodnoty	FORM
Krok	**Popis kroku**	**Očekávaný výsledek**
Počáteční podmínka	Není	
Krok 1	Otevřete aplikaci.	Aplikace spuštěna.
Krok 2	Vyberte REST API od společnosti Mapbox.	Vybráno, je zobrazen formulář pro odesílání dotazů na Mapbox API.
Krok 3	Vyplňte formulář validními hodnotami, například výběrem konkrétního hlavního města z nabídky.	Formulář vyplněn validními hodnotami.
Krok 4	Klikněte na tlačítko „Odeslat dotaz“.	Dotaz na Mapbox API je úspěšně odeslán.

TC_13	NASA – validní dotaz	KOM
Krok	**Popis kroku**	**Očekávaný výsledek**
Počáteční podmínka	Není	
Krok 1	Otevřete aplikaci.	Aplikace spuštěna.
Krok 2	Vyberte REST API od NASA.	Vybráno, je zobrazen formulář pro odesílání dotazů na NASA API.
Krok 3	Vyplňte formulář validními hodnotami, například výběrem současného kalendářního data.	Formulář vyplněn validními hodnotami.
Krok 4	Klikněte na tlačítko „Odeslat dotaz“.	Dotaz na NASA API je úspěšně odeslán.
Krok 5	Zkontrolujte, že v gridu je zobrazena odpověď od NASA API.	Odpověď je zobrazena v gridu.

TC_14	Mapbox – validní dotaz	KOM
Krok	**Popis kroku**	**Očekávaný výsledek**
Počáteční podmínka	Není	
Krok 1	Otevřete aplikaci.	Aplikace spuštěna.
Krok 2	Vyberte REST API od společnosti Mapbox.	Vybráno, je zobrazen formulář pro odesílání dotazů na Mapbox API.
Krok 3	Vyplňte formulář validními hodnotami, například výběrem konkrétního hlavního města z nabídky.	Formulář vyplněn validními hodnotami.
Krok 4	Klikněte na tlačítko „Odeslat dotaz“.	Dotaz na Mapbox API je úspěšně odeslán.
Krok 5	Zkontrolujte, že v gridu je zobrazena odpověď od Mapbox API.	Odpověď je zobrazena v gridu.

TC_15	NASA – řádky gridu - oddělení	GRID
Krok	**Popis kroku**	**Očekávaný výsledek**
Počáteční podmínka	Není	
Krok 1	Otevřete aplikaci.	Aplikace spuštěna.
Krok 2	Vyberte REST API od NASA.	Vybráno, je zobrazen formulář pro odesílání dotazů na NASA API.
Krok 3	Vyplňte formulář validními hodnotami a odešlete dotaz.	Dotaz na dané REST API úspěšně odeslán.
Krok 4	Zkontrolujte, že v gridu je zobrazena odpověď od daného REST API.	Odpověď je zobrazena v gridu.
Krok 5	Zkontrolujte, že jednotlivé řádky v gridu jsou barevně odděleny.	Řádky v gridu jsou od sebe barevně odděleny.

TC_16	NASA – řádky gridu - editace	GRID
Krok	**Popis kroku**	**Očekávaný výsledek**
Počáteční podmínka	Není	
Krok 1	Otevřete aplikaci.	Aplikace spuštěna.
Krok 2	Vyberte REST API od NASA.	Vybráno, je zobrazen formulář pro odesílání dotazů na NASA API.
Krok 3	Vyplňte formulář validními hodnotami a odešlete dotaz.	Dotaz na dané REST API úspěšně odeslán.
Krok 4	Zkontrolujte, že v gridu je zobrazena odpověď od daného REST API.	Odpověď je zobrazena v gridu.
Krok 5	Klikněte do libovolné buňky libovolného řádku gridu.	Kliknuto, zobrazená data není možné nijak editovat.

TC_17	NASA – aktualizace gridu	GRID
Krok	**Popis kroku**	**Očekávaný výsledek**
Počáteční podmínka	Není	
Krok 1	Otevřete aplikaci.	Aplikace spuštěna.
Krok 2	Vyberte REST API od NASA.	Vybráno, je zobrazen formulář pro odesílání dotazů na NASA API.
Krok 3	Vyplňte formulář validními hodnotami a odešlete dotaz.	Dotaz na dané REST API úspěšně odeslán.
Krok 4	Zkontrolujte, že v gridu je zobrazena odpověď od daného REST API.	Odpověď je zobrazena v gridu.
Krok 5	Vyplňte formulář novými validními hodnotami a odešlete nový dotaz.	Nový dotaz úspěšně odeslán.
Krok 6	Zkontrolujte, že v gridu je zobrazena nová odpověď od daného REST API.	Nová odpověď je zobrazena v gridu.

TC_18	Mapbox – řádky gridu - oddělení	GRID
Krok	**Popis kroku**	**Očekávaný výsledek**
Počáteční podmínka	Není	
Krok 1	Otevřete aplikaci.	Aplikace spuštěna.
Krok 2	Vyberte REST API od společnosti Mapbox.	Vybráno, je zobrazen formulář pro odesílání dotazů na Mapbox API.
Krok 3	Vyplňte formulář validními hodnotami a odešlete dotaz.	Dotaz na dané REST API úspěšně odeslán.
Krok 4	Zkontrolujte, že v gridu je zobrazena odpověď od daného REST API.	Odpověď je zobrazena v gridu.
Krok 5	Zkontrolujte, že jednotlivé řádky v gridu jsou barevně odděleny.	Řádky v gridu jsou od sebe barevně odděleny.

TC_19	Mapbox – řádky gridu - editace	GRID
Krok	**Popis kroku**	**Očekávaný výsledek**
Počáteční podmínka	Není	
Krok 1	Otevřete aplikaci.	Aplikace spuštěna.
Krok 2	Vyberte REST API od společnosti Mapbox.	Vybráno, je zobrazen formulář pro odesílání dotazů na Mapbox API.
Krok 3	Vyplňte formulář validními hodnotami a odešlete dotaz.	Dotaz na dané REST API úspěšně odeslán.
Krok 4	Zkontrolujte, že v gridu je zobrazena odpověď od daného REST API.	Odpověď je zobrazena v gridu.
Krok 5	Klikněte do libovolné buňky libovolného řádku gridu.	Kliknuto, zobrazená data není možné nijak editovat.

TC_20	Mapbox – aktualizace gridu	GRID
Krok	**Popis kroku**	**Očekávaný výsledek**
Počáteční podmínka	Není	
Krok 1	Otevřete aplikaci.	Aplikace spuštěna.
Krok 2	Vyberte REST API od společnosti Mapbox.	Vybráno, je zobrazen formulář pro odesílání dotazů na Mapbox API.
Krok 3	Vyplňte formulář validními hodnotami a odešlete dotaz.	Dotaz na dané REST API úspěšně odeslán.
Krok 4	Zkontrolujte, že v gridu je zobrazena odpověď od daného REST API.	Odpověď je zobrazena v gridu.
Krok 5	Vyplňte formulář novými validními hodnotami a odešlete nový dotaz	Nový dotaz úspěšně odeslán.
Krok 6	Zkontrolujte, že v gridu je zobrazena nová odpověď od daného REST API.	Nová odpověď je zobrazena v gridu.

TC_21	NASA – export do JSON - export	EXP
Krok	**Popis kroku**	**Očekávaný výsledek**
Počáteční podmínka	Není	
Krok 1	Otevřete aplikaci.	Aplikace spuštěna.
Krok 2	Vyberte REST API od NASA.	Vybráno, je zobrazen formulář pro odesílání dotazů na NASA API.
Krok 3	Vyplňte formulář validními hodnotami a odešlete dotaz.	Dotaz na dané REST API úspěšně odeslán.
Krok 4	Zkontrolujte, že v gridu je zobrazena odpověď od daného REST API.	Odpověď je zobrazena v gridu.
Krok 5	Klikněte na tlačítko Uložit jako, nacházející se nad gridem.	Kliknuto, vyskočí okno s možností stažení souboru ve formátu JSON.

TC_22	NASA – export do JSON - kontrola	EXP
Krok	**Popis kroku**	**Očekávaný výsledek**
Počáteční podmínka	Není	
Krok 1	Otevřete aplikaci.	Aplikace spuštěna.
Krok 2	Vyberte REST API od NASA.	Vybráno, je zobrazen formulář pro odesílání dotazů na NASA API.
Krok 3	Vyplňte formulář validními hodnotami a odešlete dotaz.	Dotaz na dané REST API úspěšně odeslán.
Krok 4	Zkontrolujte, že v gridu je zobrazena odpověď od daného REST API.	Odpověď je zobrazena v gridu.
Krok 5	Klikněte na tlačítko Uložit jako, nacházející se nad gridem.	Kliknuto, vyskočí okno s možností stažení souboru ve formátu JSON.
Krok 6	Stáhněte a otevřete soubor.	Soubor otevřen.
Krok 7	Zkontrolujte, že formátování odpovídá formátu JSON, například pomocí online validátoru.	Formátování odpovídá.

TC_23	NASA – export pro MS Excel - export	EXP
Krok	**Popis kroku**	**Očekávaný výsledek**
Počáteční podmínka	Není	
Krok 1	Otevřete aplikaci.	Aplikace spuštěna.
Krok 2	Vyberte REST API od NASA.	Vybráno, je zobrazen formulář pro odesílání dotazů na NASA API.
Krok 3	Vyplňte formulář validními hodnotami a odešlete dotaz.	Dotaz na dané REST API úspěšně odeslán.
Krok 4	Zkontrolujte, že v gridu je zobrazena odpověď od daného REST API.	Odpověď je zobrazena v gridu.
Krok 5	Klikněte na tlačítko Excel, nacházející se nad gridem.	Kliknuto, vyskočí okno s možností stažení souboru ve formátu pro MS Excel.

TC_24	NASA – export pro MS Excel - kontrola	EXP
Krok	**Popis kroku**	**Očekávaný výsledek**
Počáteční podmínka	Není	
Krok 1	Otevřete aplikaci.	Aplikace spuštěna.
Krok 2	Vyberte REST API od NASA.	Vybráno, je zobrazen formulář pro odesílání dotazů na NASA API.
Krok 3	Vyplňte formulář validními hodnotami a odešlete dotaz.	Dotaz na dané REST API úspěšně odeslán.
Krok 4	Zkontrolujte, že v gridu je zobrazena odpověď od daného REST API.	Odpověď je zobrazena v gridu.
Krok 5	Klikněte na tlačítko Excel, nacházející se nad gridem.	Kliknuto, vyskočí okno s možností stažení souboru ve formátu pro MS Excel.
Krok 6	Stáhněte a otevřete soubor.	Soubor otevřen.
Krok 7	Zkontrolujte, že vyexportovaná data jsou řádně zformátována.	Vyexportovaná data jsou zformátována.

TC_25	Mapbox – export do JSON - export	EXP
Krok	**Popis kroku**	**Očekávaný výsledek**
Počáteční podmínka	Není	
Krok 1	Otevřete aplikaci.	Aplikace spuštěna.
Krok 2	Vyberte REST API od společnosti Mapbox.	Vybráno, je zobrazen formulář pro odesílání dotazů na Mapbox API.
Krok 3	Vyplňte formulář validními hodnotami a odešlete dotaz.	Dotaz na dané REST API úspěšně odeslán.
Krok 4	Zkontrolujte, že v gridu je zobrazena odpověď od daného REST API.	Odpověď je zobrazena v gridu.
Krok 5	Klikněte na tlačítko Uložit jako, nacházející se nad gridem.	Kliknuto, vyskočí okno s možností stažení souboru ve formátu JSON.

TC_26	Mapbox – export do JSON - kontrola	EXP
Krok	**Popis kroku**	**Očekávaný výsledek**
Počáteční podmínka	Není	
Krok 1	Otevřete aplikaci.	Aplikace spuštěna.
Krok 2	Vyberte REST API od společnosti Mapbox.	Vybráno, je zobrazen formulář pro odesílání dotazů na Mapbox API.
Krok 3	Vyplňte formulář validními hodnotami a odešlete dotaz.	Dotaz na dané REST API úspěšně odeslán.
Krok 4	Zkontrolujte, že v gridu je zobrazena odpověď od daného REST API.	Odpověď je zobrazena v gridu.
Krok 5	Klikněte na tlačítko Uložit jako, nacházející se nad gridem.	Kliknuto, vyskočí okno s možností stažení souboru ve formátu JSON.
Krok 6	Stáhněte a otevřete soubor.	Soubor otevřen.
Krok 7	Zkontrolujte formátování souboru, například pomocí JSON validátoru.	Formátování odpovídá.

TC_27	Mapbox – export pro MS Excel - export	EXP
Krok	**Popis kroku**	**Očekávaný výsledek**
Počáteční podmínka	Není	
Krok 1	Otevřete aplikaci.	Aplikace spuštěna.
Krok 2	Vyberte REST API od společnosti Mapbox.	Vybráno, je zobrazen formulář pro odesílání dotazů na Mapbox API.
Krok 3	Vyplňte formulář validními hodnotami a odešlete dotaz.	Dotaz na dané REST API úspěšně odeslán.
Krok 4	Zkontrolujte, že v gridu je zobrazena odpověď od daného REST API.	Odpověď je zobrazena v gridu.
Krok 5	Klikněte na tlačítko Excel, nacházející se nad gridem.	Kliknuto, vyskočí okno s možností stažení souboru ve formátu pro MS Excel.

TC_28	Mapbox – export pro MS Excel - kontrola	EXP
Krok	**Popis kroku**	**Očekávaný výsledek**
Počáteční podmínka	Není	
Krok 1	Otevřete aplikaci.	Aplikace spuštěna.
Krok 2	Vyberte REST API od společnosti Mapbox.	Vybráno, je zobrazen formulář pro odesílání dotazů na Mapbox API.
Krok 3	Vyplňte formulář validními hodnotami a odešlete dotaz.	Dotaz na dané REST API úspěšně odeslán.
Krok 4	Zkontrolujte, že v gridu je zobrazena odpověď od daného REST API.	Odpověď je zobrazena v gridu.
Krok 5	Klikněte na tlačítko Excel, nacházející se nad gridem.	Kliknuto, vyskočí okno s možností stažení souboru ve formátu pro MS Excel.
Krok 6	Stáhněte a otevřete soubor.	Soubor otevřen.
Krok 7	Zkontrolujte, že vyexportovaná data jsou řádně zformátována.	Vyexportovaná data jsou zformátována.

Příloha B: Použité zkratky v názvech testovacích případů

Zkratka	**Popis**
UI	Test uživatelského rozhraní
FORM	Test vlastnosti formuláře
KOM	Test komunikace s REST API
GRID	Test vlastnosti gridu
EXP	Test exportu