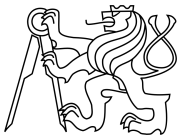
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Monitoring realitního trhu v Praze
Student:	Bc. Marek Dorda
Vedoucí:	Ing. Ivo Lašek, Ph.D.
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2016/17

Pokyny pro vypracování

Cílem diplomové práce je vytvořit nástroj pro monitoring cen realit v Praze. Vyvinuté řešení by mělo být schopno sbírat automaticky data o prodejních cenách nemovitostí z realitních serverů (např. Sreality.cz) a prezentovat je v uživatelsky přívětivé podobě, která umožní další analýzu posbíraných dat.

Pokyny:
* Navrhněte, implementujte a otestujte nástroj pro automatický sběr informací o aktuální nabídce realit v Praze. Informace se budou stahovat z vybraného realitního serveru (např. Sreality.cz).
* Navrhněte, implementujte a otestujte webovou aplikaci, která umožní sesbíraná data přehledně prezentovat ve formě cenové mapy.
* Navrhněte a nasaďte rozhraní pro hlubší analýzu poskytnutých dat. Rozhraní by mělo umožňovat ad-hoc filtrování dat, počítání základních statistických údajů (např. „průměrná cena bytů 2+kk na Praze 6“) a jejich vizualizaci.

Seznam odborné literatury

Dodá vedoucí práce.

L.S.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 13. února 2016

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

Diplomová práce

Monitoring realitního trhu v Praze

Bc. Marek Dorda

Vedoucí práce: Ing. Ivo Lašek, Ph.D.

8. května 2016

Poděkování

Děkuji vedoucímu této práce Ing. Ivu Laškovi, Ph.D. za připomínky, cenné rady, četné nápady a metodické vedení při psaní této práce. Děkuji také Bc. Tereze Rybářové za korekturu stylistickou i gramatickou.

Prohlášení

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

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

V Praze dne 8. května 2016 .....

Odkaz na tuto práci

Dorda, Marek. *Monitoring realitního trhu v Praze.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2016.

Abstrakt

Cílem této práce je vytvořit automatický nástroj pro monitoring cen realit v Praze. Hlavním cílem je vyvinout software, který automaticky sbírá data o cenách prodejů a pronájmů nemovitostí z vybraného realitního serveru a umožňuje další uživatelsky přívětivou analýzu posbíraných dat.

Klíčová slova monitoring realit, vývoj realitního trhu, Praha, analýza dat, webová aplikace

Abstract

The goal of this master's thesis is to develop a software that has the ability to automatically monitor real estate data, including market trends for both renting and owning real estate from a selected server. The software also provides a user-friendly data analysis, and web application.

Keywords real estate monitoring, real estate market trends, Prague, data analysis, web application

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Realitních serverů inzerujících nemovitosti v České republice je mnoho a neustále vznikají další. Trh s realitami zažívá díky mnoha politickým a ekonomickým faktorům velký rozmach, který posiluje pozici realitních společností. Z tohoto důvodů existuje několik softwarových řešení, které jim usnadňují orientaci na realitním trhu. Neexistuje ale žádný veřejně dostupný projekt, který by situaci usnadňoval těm, kteří chtějí v České republice nemovitost koupit, nikoliv prodat. Přestože povědomí o tom, co konkrétně je u realit cenotvorný prvek, má většina lidí, kteří se o koupi nemovitosti zajímají, neexistuje žádný veřejně dostupný nástroj, který by o tom poskytoval jasné důkazy.

Cílem práce tedy je vytvořit nástroj, který poskytne nejen nakupujícím komplexní informace o tom, které prvky jsou cenotvorné, které lokality jsou drahé a které naopak levné, jaká dispozice bytu je nejvýhodnější a jaká naopak nejméně výhodná, zda jsou byty v panelových domech opravdu nejlevnější, zda bydlení v centru je opravdu dražší než na okraji města, která roční doba je nejvýhodnější ke koupi nemovitost a mnoho dalších informací. Dalším cílem je umožnit lidem, aby si mohli vyfiltrovat nemovitosti podle požadovaných parametrů a analyzovat data prostřednictvím grafů či cenových map. Cenová mapa by měla umožňovat přehledné zobrazení cenových kategorií nemovitostí na geografické mapě.

Osobní motivací k vypracování této práce je především fakt, že trh s realitami v České republice poskytuje příležitost pro prosazení nových nástrojů. A to z toho důvodu, že patří v západním světě mezi ty mladší a levnější, a čeká jej pravděpodobně další rozmach. To je zapříčiněno historickým vývojem v České republice. Vhledem k tomu, jak rychle v České republice vzrostly ceny nemovitostí za posledních 20 let, lze předpokládat, že i nadále porostou. To naznačují i ceny nemovitostí v západní Evropě, Severní Americe a Austrálii, kde byl vývoj velmi podobný, jen o několik desítek let dříve. Český realitní trh je stále dobrou investiční příležitostí a podobný nástroj, jehož vytvořením se práce zabývá, může usnadnit mnohdy těžká rozhodnutí související s nákupem nemovitostí.

Kapitola 1

Existující možnosti

1.1 Hotová řešení

Na internetu existuje mnoho projektů, které monitorují realitní trh. Obvykle se jedná o nástroje pro realitní makléře a obchodní společnosti, které potřebují rychle reagovat na nový inzerát nebo správně odhadnout cenu nemovitosti v určené lokalitě. Hlavním účelem těchto nástrojů je rychle podat informaci o novém inzerátu nemovitosti nebo o změně v katastru nemovitostí. Projekty, které se zabývají analýzou realitního trhu, jeho vývojem a zpracováním statistik, jsou především zahraniční. Zahraniční servery mají navíc k dispozici data z delšího časového úseku než servery české.

Pro účely této práce byly vybrány celkem 3 české a 3 zahraniční projekty. Popis funkcionality jednotlivých projektů vychází pouze z toho, co uvádějí na svých internetových stránkách, neboť všechny služby jsou pouze za poplatek. Konkrétně se jedná o tyto projekty:

Infoexe monitoring - český projekt, který je svou funkcionalitou zaměřen především na realitní makléře. Zajímavou službou ze všech, které projekt nabízí, je například sledování změn v katastru nemovitostí. [1]

Octopus Pro - další z českých projektů, který je opět určen především pro realitní makléře. Mimo jiné nabízí například cenovou mapu realit. [2]

ADOL Monitoring Realit - poslední z porovnávaných českých projektů, který je opět určen přednostně pro realitní makléře. K dispozici je rovněž cenová mapa realit. [3]

JLR Land Title Solutions - kanadský projekt s více než 30 letou tradicí. Jedná se především o poradenskou firmu, která kromě poradenských služeb nabízí také online aplikaci umožňující například odhady cen nebo interaktivní prohlížení statistik. [4]

PropertyRadar - americký projekt, který kromě cenové mapy a poskytování detailních informací k jednotlivým parcelám, umožňuje také analýzu investic, která zohledňuje parametry nemovitosti, její lokalitu a cenu, na jejichž základě spočítá návratnost investice. [5]

APM PriceFinder - australský projekt, který má k dispozici databázi s daty nasbíranými za více než 30 let. Jedná se především o poradenskou firmu, která ve své online aplikaci poskytuje například možnost filtrovat statistiky na základě zvýraznění libovolné oblasti na mapě. [6]

1.1.1 Infoexe monitoring

Jedná se o český server, který je zaměřen především na potřeby realitních makléřů. Jeho hlavní funkcí je upozornění uživatele na nový soukromý inzerát v rozmezí od 1 do 3 minut od jeho přidání. Tato funkce makléři umožní včas reagovat na nabídku a nabídnout zadavateli inzerátu své služby. Server stahuje údaje z více než 10 serverů, včetně některých zahraničních. Taktéž zvládá detekci stejných inzerátů, které jsou uvedeny na více serverech zároveň. [1]

Mezi další poskytované služby patří například vyhledávání nemovitostí podobných některé konkrétně zvolené, sledování vývoje ceny u nemovitosti, vyhledávání dalších inzerátů od stejného inzerenta, možnost přidávání uživatelských poznámek k jednotlivým nemovitostem, filtrování nemovitostí podle různých parametrů (stáří inzerátu, lokality, dispozice bytu apod.). Server rovněž umožňuje hlídání zvolených objektů v katastru nemovitostí a případné nahlášení každé změny uživateli. [1]

1.1.2 Octopus Pro

Octopus Pro je další z českých serverů, který je rovněž zaměřen na potřeby realitních makléřů. Stěžejní funkcionalitou je notifikace uživatele do jedné minuty od doby, kdy se na některém z monitorovaných inzertních serverů objeví nový soukromý inzerát. Server stahuje údaje z českých a slovenských realitních webů. Také provádí zpracování vybraných tištěných inzertních novin. [2]

Octopus Pro nabízí i další služby, jakými je například cenová mapa, která reflektuje za kolik je možné v určité lokalitě nemovitost prodat či pronajmout. Mapa slouží i k výběru cenově vhodné lokality k investici, ke sledování změny cen v závislosti na dostupnosti občanské vybavenosti, případně k zobrazení odchylek od normálních parametrů typických pro danou lokalitu (například příliš levný byt v drahé lokalitě apod.). Služba rovněž nabízí predikce cen na základě dostupných údajů a odhady cen podle adresy. [2]

Data jsou sbírána jak z nabídek uvedených v inzerátech, tak z realizovaných prodejů, což nabízí možnost porovnání mezi nabídkou a skutečnou prodejností. [2]

1.1.3 ADOL Monitoring Realit

ADOL Monitoring Realit je český projekt, který je opět určen především pro realitní makléře. Informace o nově dostupné nabídce je uživateli oznámena do jedné minuty od jejího zveřejnění. Server stahuje data ze všech českých serverů (podrobnosti neuvádí) a má k dispozici archiv s daty nashromážděnými za dobu 3 let. [3]

Server umožňuje sledovat vývoj cen nejen jednotlivých nabídek bytů a domů, ale i pozemků, dražeb a exekucí. K dispozici je i cenová mapa a možnost exportovat data do uživatelem definovaného XML formátu. [3]

1.1.4 JLR Land Title Solutions

Jedná se o kanadskou poradenskou firmu zaměřující se především na oblast města Quebec. Omezený sortiment služeb nabízí i pro Ontario a Spojené státy americké. Kromě poradenství nabízí i online služby, kde je k dispozici databáze s daty o prodeji nemovitostí a hypoték, nashromážděnými za dobu více než 30 let. [4]

Z online služeb stojí za zmínku například odhad ceny nemovitosti na základě jejich parametrů a lokality, kde se nemovitost nachází. Při odhadu je možné měnit rádius okolí nemovitosti pro výpočet odhadované ceny v rozmezí 500m až 10km. Taktéž je k dispozici notifikace při jakékoliv změně ve vlastnictví nemovitosti nebo denní reporty ohledně proběhlých transakcí v dané lokalitě. JLR rovněž nabízí službu, kdy na základě zadaných parametrů nemovitosti jsou vyhledány podobné prodeje ve zvoleném časovém období. K dispozici je i procházení statistik a interaktivní práce s nimi. [4]

1.1.5 PropertyRadar

PropertyRadar je americká společnost, která nabízí nejen možnost prohledávání nemovitostí podle parametrů a vyhledávání aukcí či exekucí, ale také nástroj pro analýzu investice. Tento nástroj odhadne cenu zvolené nemovitosti na základě jejich parametrů a vývoje ceny v lokalitě, kde se nemovitost nachází, spočítá, za jakou dobu se investice vrátí pronájmem (cena pronájmu je stanovena opět odhadem), případně kdy by bylo nejvýhodnější nemovitost opět prodat. [5]

K dalším službám patří například mapy, které zobrazují jak ceny, tak i zvolené parametry nemovitostí (například nejžádanější dispozice bytu v oblasti). Je možné si nastavit hlídání nemovitosti, kdy je uživatel upozorněn na jakoukoliv změnu, ve vlastnictví nebo v inzerátu, týkající se dané nemovitosti. K dispozici je ke zvolené nemovitosti i historie prodeje a změn ve vlastnictví, včetně uvedené ceny, za kterou byly veškeré prodeje realizovány. PropertyRadar nabízí i týmovou spolupráci, kdy si uživatelé mohou společně vytvářet seznamy nemovitostí, přiřazovat jim štítky apod. [5]

1.1.6 APM PriceFinder

APM PriceFinder je společnost působící v Austrálii, která nabízí především poradenské služby, při kterých vychází zejména z dat nastřádaných za dobu více než 30 leté existence. Některé služby nabízí prostřednictvím webových stránek, jako například pravidelné reporty proběhlých transakcí nebo funkci pojmenovanou manažer rizik (detaily neuvádí). [6]

K dalším službám patří mapové vizualizace cen a různých jiných parametrů nemovitostí. Unikátní funkcí je možnost zvolit na mapě libovolnou oblast, na jejímž základě aplikace zobrazí statistiky. K dispozici je také API, pomocí kterého lze validovat údaje o nemovitosti a přistupovat ke statistikám. Ke konkrétní adrese lze vyhledat nemovitost, ke které existuje archiv inzerátů, prodejních cen a také odhad aktuální ceny určené na základě oblasti a parametrů nemovitosti. [6]

1.2 Průzkum realitních serverů

Realitních serverů, které uvádějí ve své nabídce nemovitosti na území Prahy, existuje celá řada. Většinou se inzeráty na jednotlivých serverech opakují, což je poměrně očekávané, neboť v zájmu realitních makléřů či společností je nemovitost prodat. Inzerce stejné nabídky nemovitosti napříč několika realitními servery není nijak náročná a šanci na prodej zvyšuje. Existují i nástroje k tomu přímo určené. Například software Realman, který umožňuje spravovat nabídku na jednom místě a následně ji exportovat na více než 40 realitních serverů [7].

Z výše popsaného se jeví jako rozumná volba vybrat si jeden rozsáhlý realitní server, ze kterého lze snadno strojově dolovat a zpracovávat data. Zajistí se tím to, že data o nemovitostech nebudou obsahovat duplicity již existujících nabídek z jiných realitních serverů. Rovněž se předejde různým kolizím s neaktuálními nabídkami z méně navštěvovaných serverů apod. Na druhou stranu bude nutné vybrat takový realitní server, který obsahuje dostatečné množství inzerátů, ze kterých bude možné získat data se statisticky vypovídající hodnotou. Díky tomu bude možné z takto získaných dat vyvodit závěry odpovídající skutečnosti.

Kritériem pro výběr nejvhodnějšího serveru pro získávání dat není pouze množství inzerátů na něm uvedených, ale též kvalita informací, jejich množství, a v neposlední řadě i možnost strojového zpracování dat.

Ze všech serverů byly do užšího výběru zvoleny čtyři takové, které jsou vlastněny významnými internetovými společnostmi v Čechách. U takových serverů lze předpokládat, že jsou společnostmi, které je vlastní, mediálně propagovány. Díky tomu je pravděpodobné, že lidé tyto servery používají a znají. Konkrétně se jedná o následující inzertní servery:

Tabulka 1.1: Přehled počtu inzerátů u vybraných serverů v říjnu 2015

	byty		domy	
	prodej	pronájem	prodej	pronájem
reality.idnes.cz	4 600	2 600	1 100	300
hyperreality.cz	2 100	1 500	500	100
realitymix.cz	4 600	3 400	1 000	400
sreality.cz	5 400	3 600	1 400	500

- **reality.idnes.cz** společnosti MAFRA a.s. [8]
- **hyperreality.cz** společnosti HyperMedia, a.s. (člen NetMonitor) [9]
- **realitymix.cz** společnosti DALTEN media s.r.o. (člen Economia, a.s.) [10]
- **sreality.cz** společnosti Seznam.cz, a.s. [11]

1.2.1 Počet nemovitostí v nabídce

Prvním důležitým kritériem byl počet nemovitostí, které jednotlivé servery nabízejí. V říjnu roku 2015 vypadala nabídka pro Prahu tak, jak je uvedeno v tabulce 1.1. Hodnoty jsou uvedeny zaokrouhleně v řádu stovek.

Jak lze z tabulky vyčíst, nejvíce inzerovaných nemovitostí v říjnu roku 2015 bylo uvedeno na serveru sreality.cz, kdy nejen v celkovém součtu, ale i u jednotlivých položek překonal ostatní porovnávané servery. Celkovým počtem zaujaly i servery reality.idnes.cz a realitymix.cz, které se s počtem inzerátů pohybovaly přibližně na 80% nabídky sreality.cz.

1.2.2 Kvalita dostupných informací

Dalším z důležitých kritérií bylo množství a kvalita informací uvedených na jednotlivých realitních serverech. Některé parametry byly uvedeny u všech porovnávaných serverů, jako je například:

- počet podlaží budovy,
- celková plocha,
- třída energetické náročnosti,
- stav budovy,
- dostupnost výtahu v budově,

Tabulka 1.2: Rozdíly v dostupnosti informací u inzerátů vybraných serverů v říjnu 2015

	reality.idnes.cz		hyperreality.cz		realitymix.cz		sreality.cz	
	byty	domy	byty	domy	byty	domy	byty	domy
Lokalita objektu	✓	✓			✓	✓	✓	✓
Umístění objektu				✓		✓		✓
Vodní zdroj	✓	✓			✓	✓	✓	✓
Topení	✓	✓		✓	✓	✓	✓	✓
Odpad	✓	✓			✓	✓	✓	✓
GPS			✓	✓	✓	✓	✓	✓
Vlastnictví			✓		✓		✓	
Body zájmu			✓	✓			✓	✓

- dispozice objektu (3+1, 2+kk, ...),
- dostupnost sklepu,
- a mnohé další...

Porovnávány tedy byly pouze vybrané parametry, kterými se servery od sebe odlišují. Nutno dodat, že množství uvedených informací se liší inzerát od inzerátu, neboť jeho vyplnění je pouze v kompetenci prodejce. Porovnání je z tohoto důvodů řešeno tak, že pokud se vybraný údaj vyskytl u většiny inzerátů, je pro příslušný server uveden jako dostupný. Přehled dostupných informací u inzerátů, kterými se od sebe servery odlišují, zobrazuje tabulka 1.2.

Lokalita objektu je upřesnění, zda se objekt nachází v klidné části obce, v rušné části obce, v centru, na sídlišti, na samotě apod.

Umístění objektu upřesňuje, zda je objekt součástí řadové zástavby, zda stojí samostatně, je rohový apod.

Topení uvádí, jakým způsobem je zajištěno vytápění objektu. Například lokální plynové topení, ústřední dálkové apod.

Odpad znamená, jakým způsobem je řešen v objektu odpad. Například veřejná kanalizace, jímka, septik apod.

Vlastnictví může být například družstevní, osobní, státní, ...

Body zájmu jsou místa v okolí, která mohou ovlivnit zájem o nemovitost. Například blízkost restaurace, školy, lékařského zařízení, autobusové zastávky, stanice metra, bankomatu, pošty, obchodu a jiné občanské vybavenosti.

Jak je z tabulky 1.2 patrné, nejvíce informací je uvedeno na realitním serveru sreality.cz. Dalším kvalitním zdrojem by mohly být stránky realitymix.cz, nicméně postrádají body zájmu, což je poměrně velký nedostatek při snaze zjistit, které prvky v okolí mohou být cenotvorné. Stránky reality.idnes.cz nepřipadají v úvahu vůbec, protože kvůli absenci GPS souřadnic je zcela znemožněno vytvářet cenové mapy.

1.2.3 Možnost strojového zpracování dat

V neposlední řadě bylo potřeba porovnat i možnost strojového zpracování dat, jelikož práce předpokládá, že data budou získávána automaticky v pravidelných intervalech. Počet požadavků pro získání veškerého obsahu je uváděn pro situaci, kdy by bylo potřeba stáhnout data o celkem 5 000 nemovitostech.

Realitní server reality.idnes.cz vypisuje údaje přímo do HTML stránky. Přehled se seznamem nemovitostí je možné řadit dle aktuálnosti a ceny, kdy lze vypsat 20 nebo 30 položek na stránku (což znamená pro 5 000 položek v nejlepším případě 167 stránek pro získání kompletního přehledu). V HTML kódu je seznam nemovitostí identifikován jednoznačným ID a každá položka reprezentující nemovitost je označena třídou, která je pro všechny nemovitosti stejná. Získání adres s detaily nemovitostí by tedy nebyl problém. Na stránce s detailem nemovitosti je obsah taktéž jednoznačně identifikován atributem ID. Nejdůležitější parametry jsou uvedeny v seznamu tagu <dl>, speciální znaky jsou kódovány HTML entitami, takže po základním převodu a drobné sanitaci by bylo možné získat jejich hodnoty. Problém by mohl nastat u specifikace městské části a dispozice nemovitosti, jelikož tyto údaje jsou uvedeny pouze v titulku stránky. V takovém případě by bylo nutné výraz patřičně naparsovat a údaje získat.

Téměř totožná je situace i na webu realitymix.cz. Přehled se seznamem nemovitostí je možné řadit dle aktuálnosti a ceny, kdy je možné na stránku vypsat 25 položek (což odpovídá 200 požadavkům pro získání kompletního seznamu 5 000 položek). V HTML kódu přehledu nemovitostí je seznam jednoznačně identifikován atributem ID a jednotlivé nemovitosti jsou pojmenovány

třídou, která je stejná pro všechny nemovitosti. Není proto problém získat odkazy na detaily nemovitostí. Na stránce s detailem nemovitosti jsou některé nejdůležitější parametry uvedeny v tabulce, která má jednoznačný identifikátor, takže po drobné sanitaci by bylo možné vše potřebné získat. Avšak některé jiné údaje jsou uvedeny pouze v nestrukturovaném textu a jejich získání by bylo velmi komplikované. Navíc mapa je zobrazena na základě vyhledávání konkrétní poštovní adresy na Google Maps, souřadnice GPS proto nejsou na stránce uvedeny.

Na stránkách hyperreality.cz je situace stále podobná, ale s nepatrnými odlišnostmi. Přehled se seznamem nemovitostí je možné řadit dle aktuálnosti a ceny, kdy je možné vypsat 36 položek na stránku (což pro 5 000 položek znamená 139 požadavků pro získání kompletního seznamu). V HTML kódu ovšem není seznam nemovitostí nijak jednoznačně identifikován a jednotlivé prvky v něm obsahují pouze obecnou třídu, která je používána i u jiných prvků na stránce, nicméně podle HTML elementů umístěných uvnitř jednotlivých prvků by bylo možné dobrat se k adresám s detaily nemovitostí. Stejně tak nejsou jednoznačně identifikovány ani parametry uváděné v detailu nemovitosti, jsou pouze umístěny v tabulce s obecnou třídou, která je používána i pro jiné prvky na stránce. Podle jednotlivých textů v prvním sloupci tabulky by ale bylo možné po drobné sanitaci správně určit příslušné hodnoty jednotlivých parametrů.

Na realitním serveru sreality.cz je situace zcela odlišná než u předchozích serverů. Celý web běží na JavaScript frameworku AngularJS [11], pomocí něhož jsou data získávána asynchronně ve formátu JSON po načtení HTML kódu. Požadavky pro získání dat jsou odesílány na konkrétní adresu s příslušnými parametry. U JSON souboru, který reprezentuje přehled se seznamem nemovitostí, je možné na jeden dotaz získat 20, 40 nebo 60 položek (což pro 5 000 položek znamená v nejlepším případě 84 požadavků pro získání kompletního seznamu). JSON soubory reprezentující jak přehled se seznamem nemovitostí, tak detaily jednotlivých nemovitostí, jsou uzpůsobeny ke strojovému zpracování, lze tedy poměrně snadno získat všechny potřebné údaje.

1.3 Cenová mapa

Na internetu je mnoho společností, které poskytují mapové podklady. Například Google Maps společnosti Google, Bing Maps od společnosti Microsoft, komunitou vytvářené OpenStreetMaps nebo Mapy.cz od společnosti Seznam.cz, a.s. Pro podrobný průzkum byly vybrány Google Maps, jelikož mají velmi podrobně dokumentované API, včetně mnoha konkrétních příkladů, a širokou uživatelskou základnu [12].

1.3.1 Google Maps

Google Maps je nástroj vyvíjený firmou Google, který poskytuje mapy celého světa, které mohou být jak satelitní, tak základní topografické. Kromě mapových podkladů je možné mapu rozšířit pomocí API i o další vrstvy, jako například: [13]

- dopravní vrstvu,
- vrstvu veřejné dopravy,
- vrstvu cyklostezek,
- vrstvu s heatmapou,
- a další.

Vrstva s heatmapou umožňuje do mapy vkládat body s určitou váhou [14], čehož by se dalo teoreticky použít pro zobrazení cenové mapy. Problémem je, že tato heatmapa je určena pro zobrazování hustoty bodů, kde váha znamená pouze zvýšení hustoty. Při snaze zobrazit takto průměrnou cenu v lokalitě dochází k tomu, že do zbarvení vrstvy se promítá nejen cena nemovitosti, ale i hustota nabídek v dané lokalitě. Což vede ke zkreslení výsledku.

Možným řešením by bylo nepoužívat heatmapu, ale využít geometrické tvary různých barev, které lze do mapy vložit [15]. Na straně serveru by se spočítala průměrná cena nemovitostí za metr čtvereční pro určitou lokalitu a na základě této ceny by se do mapy vložil kruh nebo čtverec odpovídající barvy. Taktéž by se při různém přiblížení mapy dala oblast průměrování zmenšovat až do takové míry, kdy by se zobrazily jednotlivé nemovitosti samostatně.

1.3.2 Ostatní mapy

Bing Maps taktéž umožňují do mapy přidat vrstvu s heatmapou. Nastává zde ovšem totožná situace jako v případě Google Maps, tedy že do heatmapy se projevuje nejen cena nemovitosti, ale i hustota bodů na mapě. [16]

Pro OpenStreetMaps existuje řada pluginů, které umožňují zobrazovat v mapě vrstvu s heatmapou, nicméně mnoho z nich je zastaralých, jiné nepodporují přidávání bodů s váhou a další mají již několikrát zmíněný problém s tím, že do vrstvy přidávají i hustotu bodů. [17]

V případě serveru Mapy.cz není možnost rozšíření mapy o vrstvu heatmapy vůbec podporována [18].

1.4 Nástroje pro vizualizaci dat

Nástroje, které dokáží vizualizovat data do uživatelsky interaktivního prostředí tak, aby bylo možné data filtrovat, je celá řada. Například nástroj Splunk společnosti Splunk Inc., Grafana společnosti Torkel Ödegaard & Coding Instinct AB, Sumo Logic stejnojmenné firmy, Loom Systems stejnojmenné firmy nebo Kibana od společnosti Elasticsearch.

Loom Systems spolu s aplikacemi Sumo Logic a Splunk vyžadují placenou licenci [19] [20] [21]. Grafana je určena především pro vizualizaci časových řad, jako jsou například záznamy internetového provozu, logy aplikací, průmyslové testování nebo vývoj počasí [22]. Kibana je univerzálně použitelná na téměř libovolná data [23], z tohoto důvodu byla vybrána k podrobnému prozkoumání.

1.4.1 Elasticsearch a Kibana

Společnost Elasticsearch vyvíjí stejnojmenný produkt, který slouží pro vyhledávání a analyzování dat v reálném čase. Jedná se o platformu, která poskytuje data pro další nástroje a pluginy, které s ní komunikují pomocí RESTful API. Mezi jeho hlavní přednosti patří především: [24]

- rychlost, se kterou pracuje s velkým objemem dat,
- možnost filtrování podle libovolných parametrů,
- integrace analytických nástrojů,
- dokumentově orientované uložiště dat.

Pomocí speciálních nástrojů je možné převést relační databázi do uložiště Elasticsearch. Jedním z těchto nástrojů je například Elasticsearch JDBC Importer vyvíjený vývojářem Jörg Prante. Nástroj umožňuje spustit libovolný SQL dotaz nad zvolenou relační databází a výsledek dotazu převést do JSON formátu, který je odeslán do uložiště Elasticsearch. Je možné nastavit i mapování jednotlivých polí z výstupu SQL dotazu do zvoleného datového typu na straně Elasticsearch. Stejně tak je možné nastavit pravidelné exporty relační databáze v předdefinovaný čas pomocí Quartz cron syntaxe. [25]

Pomocí nástroje Kibana, který rovněž vyvíjí společnost Elasticsearch, lze data analyzovat prostřednictvím webového prohlížeče. Kibana poskytuje webové rozhraní pro analýzu dat uložených v uložišti Elasticsearch a nabízí jejich vizualizaci například ve formě: [23]

- sloupcových grafů,
- řádkových a rozptylových grafů,
- histogramů,

- koláčových grafů,
- geografických map,
- matematických transformací

Rovněž nabízí možnost data rozčleňovat a dělat jimi průřezy na základě zvolených hodnot. Řešením by tedy mohlo být posbíraná data exportovat do uložiště Elasticsearch pomocí Elasticsearch JDBC Importer a s využitím rozhraní Kibana data analyzovat. Kibana navíc podporuje u mapových vizualizací širokou nastavitelnost, jako například nastavení toho, který parametr se má na mapě zobrazovat, zdali se má vizualizovat jeho součet, maximum, minimum, medián nebo průměr z okolí. Také lze nastavit, zda se mají při přibližování mapy přepočítávat hodnoty na menších blocích až do úrovně jednotlivých nabídek nebo zda má být ponechána hodnota vypočítána z pevně zvoleného okolí. [23]

1.5 Katastr nemovitostí

Údaje získané pouze z nabízených inzerátů mohou být zavádějící, jelikož ceny jsou nastaveny tak, aby byly výhodné především pro prodávajícího. Skutečných prodejních cen by bylo možné se dobrat přes katastr nemovitostí, neboť od 1. ledna 2014 je povinností uvádět do katastru i cenu, za kterou byla nemovitost prodána [26].

Katastrální úřad nabízí celkem 3 způsoby online nahlížení do katastru nemovitostí [27]:

- bezplatné nahlížení (omezený rozsah informací),
- dálkový neomezený přístup,
- přístup pomocí aplikace Webové služby Dálkového přístupu.

Bezplatný přístup je určen výhradně pro interakci s uživatelem. Veškeré vytěžování nebo automatické dolování dat není dovoleno. Navíc zobrazení detailu o parcele vyžaduje opsání CAPTCHA kódu. [28]

Dálkový neomezený přístup je zdarma přístupný pouze samosprávným orgánům obcí, měst, krajů, organizačním složkám státu, notářům a soudním exekutorům [29]. Všichni ostatní mají přístup zpoplatněn dle platného ceníku, tzn. získání podrobnosti o parcele za 10Kč, získání informací o dosažené ceně při prodeji za 50Kč [30].

Aplikace Webové služby Dálkového přístupu je API rozhraní, které rozšiřuje dálkový neomezený přístup o možnost sběru dat strojově a jejich následného strojového zpracování. Pro použití je vyžadován zákaznický účet. Získání údajů se opět řídí platným ceníkem a ceny jsou totožné jako u dálkového neomezeného přístupu. Údaje lze získat ve strojově čitelném XML formátu. [31]

1.6 Souhrn

Pro účely této práce nelze použít žádné z hotových existujících řešení monitorování realitního trhu. Všechny projekty jsou čistě komerční záležitostí, jejichž provozovatelé se většinou zaměřují na potřeby realitních makléřů a obchodníků. Je ale možné se jimi inspirovat. Mezi zajímavou funkcionalitu, kterou prozkoumané servery nabízejí, lze zařadit například:

- sledování vývoje ceny u nemovitostí,
- vyhledávání ceny u podobných nemovitostí,
- filtrování nemovitostí podle parametrů,
- cenová mapa,
- mapy dominantních parametrů nemovitostí v lokalitě,
- upozornění na výrazně dražší nebo levnější nemovitosti než je běžné v dané lokalitě,
- cenové odhady na základě parametrů nemovitosti a lokality,
- vyhledávání prodejů všech podobných nemovitostí ve zvolené lokalitě,
- nástroj pro analýzu návratnosti investice.

Z průzkumu českých realitních serverů vyšel nejlépe server sreality.cz. Dosahuje největšího množství nabízených nemovitostí v Praze, uvádí i nejpodrobnější a nejkomplexnější informace o nemovitostech a navíc data z něj získaná jsou i nejlépe strojově zpracovatelná. Například server realitymix.cz sice dosahuje počtem inzerátů přibližně 80% sreality.cz a nabízí srovnatelné množství informací k jednotlivým nemovitostem, ale strojové zpracování je již komplikovanější a například není možné se v kódu dobrat GPS souřadnic. Podobně je na tom i server reality.idnes.cz, který neuvádí na svých stránkách GPS souřadnice a ani nezobrazuje nemovitosti na mapě. Posledním z uvažovaných serverů byly stránky hyperreality.cz, které ale dosahovaly počtem inzerátu necelých 40% nabídky sreality.cz. Navíc množství uváděných informací nebylo tak komplexní jako u všech ostatních porovnávaných serverů.

Cenovou mapu je možné vytvořit pomocí nástroje Google Maps nebo Kibana. Řešení přes Google Maps by vyžadovalo kompletní implementaci algoritmu pro výpočet hodnoty, která by zastupovala ceny všech nemovitostí v určité lokalitě. To z toho důvodu, že by nebylo možné použít nachystanou vrstvu heatmapy, která v případě Google Maps slouží k jiným účelům. Řešením by tedy bylo data vizualizovat pomocí geometrických tvarů vkládaných do mapy. V případě nástroje Kibana je možné použít cenovou mapu bez jakýchkoliv úprav.

Pro vizualizaci a hlubší analýzu posbíraných dat je možné využít open-source projekt Kibana fungující na platformě Elasticsearch. Vizualizovaná data je možné filtrovat, vytvářet cenové mapy a rovněž vytvářet náhledy různých řezů daty. Navíc je možné rychle a plynule zobrazovat data za velké časové období, protože platforma Elasticsearch slouží především k analýze velkého objemu dat a je tomu tedy uzpůsobena.

Další poměrně užitečnou funkcionalitou by mohlo být získávání dat z katastru nemovitostí a porovnávání skutečných prodejních cen nemovitostí s těmi inzerovanými. Stejně tak by bylo možné přímo ke konkrétní parcele vytvářet historii prodeje, avšak získání potřebných údajů ze strany katastrálního úřadu je natolik nákladné, že znemožňuje realizaci v rámci této práce.

Kapitola 2

Analýza a návrh aplikace

Následující kapitola je věnována podrobné analýze budoucího řešení výsledné aplikace. Je zde uvedeno, jakým způsobem bude aplikace rozdělena a jak budou jednotlivé komponenty spolu komunikovat. Jsou v ní taktéž sepsány funkční a nefunkční požadavky, uživatelské role a případy užití. Rovněž je v kapitole popsáno, jakým způsobem budou data získána a následně zpracována. V závěru kapitoly je vysvětlena volba programovacího jazyka, databáze a dalších technologií.

2.1 Specifikace požadavků

2.1.1 Funkční požadavky

Z důvodu velké rozsáhlosti požadavků vyplývajících z průzkumu existujících řešení a inzertních serverů, nebudou v rámci této práce realizovány všechny uvedené požadavky. Požadavky, které jsou zamýšleny do budoucna a nebudou součástí vyvíjené aplikace, jsou uvedeny kurzívou.

- Mapové vizualizace
  - změna data, ke kterému se vztahují nabídky
  - posouvání, přibližování a oddalování mapy
  - přepínání mezi pronájmem a prodejem, domy a byty
  - *volba vizualizovaných parametrů (cena, typ budovy, ...)*
- Analýza dat
  - změna data, ke kterému se vztahují nabídky
  - přepínání mezi pronájmem a prodejem, domy a byty
  - základní statistické údaje
  - vizualizace zkoumaných údajů pomocí grafů

- *Pokročilé analytické nástroje*
  - *vývoj ceny v závislosti na době zveřejnění inzerátu*
  - *za jakou dobu se vrátí investice do nemovitosti jejím pronájmem (na základě parametrů nemovitosti a její lokality)*
  - *průměrná doba, po kterou je inzerát zveřejněn v nabídce (na základě parametrů nemovitosti a její lokality)*
  - *vyhledávání lokalit, kde ceny vzrostly nejstrměji*
  - *vyhledávání levných nemovitostí v drahé lokalitě*

Implementované části (mapové vizualizace a analýza dat) budou mít navíc k dispozici filtrování podle různých parametrů nemovitostí, konkrétně podle těchto:

- přítomnost balkonu, sklepu, terasy, parkovacího stání, garáže, vybavení a výtahu
- typ vlastnictví
- městská část
- typ budovy (panelová, cihlová, ...)
- umístění domu (pouze pro domy; rohový, řadový, ...)
- varianta domu (pouze pro domy; přízemní, patrový, ...)
- stav nemovitosti (novostavba, k demolici, po rekonstrukci, ...)
- energetická třída
- dispozice (2+kk, 3+1, vila, rodinný dům, ...)
- způsob vytápění
- zdroj vody
- řešení odpadu
- lokalita objektu (rušná část, samota, ...)
- interval plochy nemovitosti
- interval plochy pozemku (pouze pro domy)
- interval velikosti zahrady (pouze pro domy)
- interval poschodí

- interval ceny
- časové období dostupnosti
- *zvolená oblast na mapě*

U mapových vizualizací by měl mít uživatel možnost si sám zvolit, zda chce zobrazovat údaje o bytech nebo domech, případně pronájmech nebo prodejích. Taktéž by měl mít možnost s mapou interaktivně pracovat, tedy mapu přibližovat, oddalovat a posouvat. Dále by měl mít uživatel možnost filtrovat zobrazené nemovitosti podle různých parametrů (městská část, dispozice, časové období, ze kterého chce zobrazit nabídky, cenovou kategorii, ...). Další variantou rozšíření funkcionality by v budoucnu mohla být možnost zobrazovat kromě cenové mapy i mapy dalších parametrů, které dominují v určité oblasti (například dispozice bytu, velikost pozemku náležící k rodinnému domu, zda je v domě výtah nebo ne apod.).

Analyzování dat by rovněž mělo mít k dispozici možnost volby mezi domy a byty, případně mezi pronájmy a prodeji. Kromě toho by měl mít uživatel možnost filtrovat data na základě časového období, kdy byly nabídky nemovitostí inzerovány, intervalu ceny nemovitostí a dle mnoha dalších parametrů uvedených ve výčtu výše.

Aplikace by mohla v budoucnu nabízet i další nástroje, jako je například sledování vývoje ceny nemovitosti v závislosti na tom, jak dlouho je inzerát zveřejněn, a jaká je průměrná doba zveřejnění inzerátu. Dále by mohla poskytovat analýzu, za jakou dobu se vrátí investice do nemovitosti jejím pronájmem (v závislosti na lokalitě a zvolených parametrech). Užitečným nástrojem by mohlo být vyhledávání lokalit, kde ceny vzrostly nejstrměji za určitý časový úsek, případně vyhledávání levných nemovitostí v drahých lokalitách.

2.1.2 Nefunkční požadavky

- dostupnost uživatelské aplikace přes webový prohlížeč
- generování vizualizací dat a jejich filtrování by mělo probíhat v rámci jednotek sekund, bez ohledu na množství dat v databázi
- uživatel aplikace nesmí mít možnost jakkoliv měnit údaje v databázi
- možné chyby při aktualizaci nabídek by se neměly projevit do uživatelské aplikace
- případné úpravy spojené se změnou struktury zdroje dat nesmí ovlivnit funkci uživatelské aplikace
- možnost případného rozšíření o další zdroje dat

2.2 Případy užití

Aplikace je cílená především na běžné uživatele, kteří si mohou interaktivně prohlížet nasbíraná data, filtrovat je podle různých parametrů, zobrazovat na mapě apod. Není potřeba nijak řešit různá uživatelská oprávnění, neboť v aplikaci není nic, co by vyžadovalo dostupnou administraci. Jedná se o veřejně přístupnou aplikaci, která umožňuje pouze prohlížení, bez možnosti editace posbíraných dat. Na obrázku 2.1 je zobrazen diagram případů užití, který vychází přímo z definice funkčních požadavků z kapitoly 2.1.1. Diagram zobrazuje pouze funkcionalitu, která je zamýšlená v rámci této práce. Kompletní diagram případů užití, včetně funkcionality zamýšlené v budoucnu, je uveden v příloze C.1.

2.3 Komponenty

Pro vyšší udržitelnost, spolehlivost a do budoucna i rozšiřitelnost, bude aplikace rozdělena na samostatné komponenty. Pokud tedy jedna z nich přestane fungovat, ostatní to neovlivní. Aplikace se bude skládat celkem ze čtyř částí:

- datové uložiště,
- crawler,
- uživatelská aplikace,
- nástroj Kibana a platforma Elasticsearch.

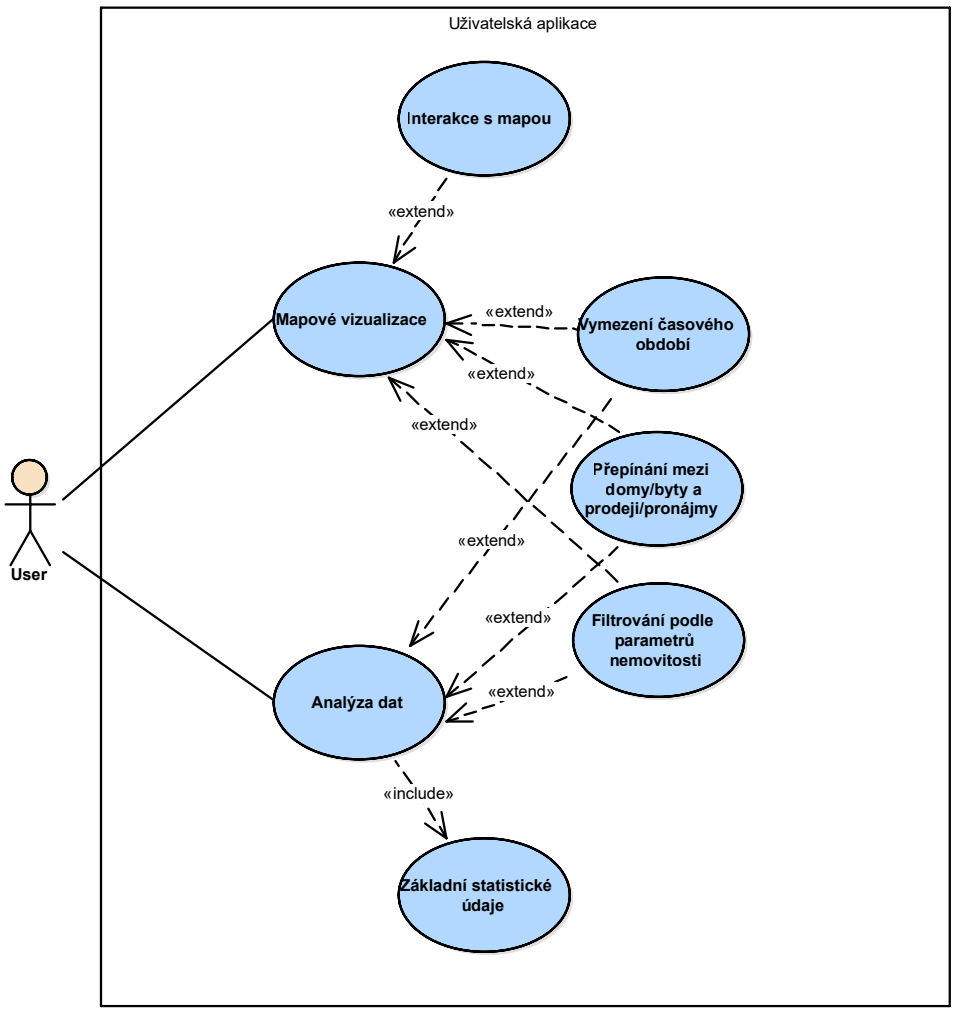
Na obrázku 2.2 je zobrazeno, jak budou jednotlivé komponenty nasazeny v rámci této práce. Na obrázku 2.3 je zobrazeno, jak by mohla situace například vypadat v ideálním případě.

2.3.1 Datové uložiště

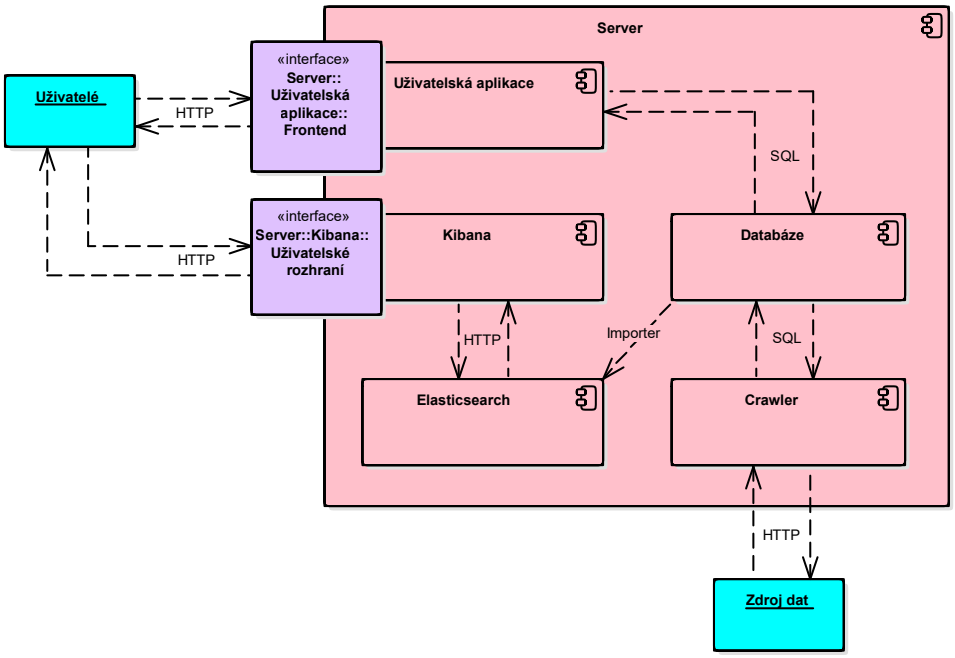
Vzhledem k tomu, že data určená k uložení do databáze jsou tabulková (seznam nemovitostí a jejich parametrů), lze použít relační databázi. Tato relační databáze bude společným uzlem pro všechny další komponenty. Jedná se tedy o nejzranitelnější prvek celého systému, který může narušit spolehlivost aplikace a způsobit nefunkčnost všech komponent. Lze hovořit o obecném problému centralizovaného způsobu řízení systémů a je proto potřeba zajistit tomuto prvku větší stabilitu a dostupnost.

Ze všech existujících řešení jsou zde uvedena dvě nejběžnější, která se v praxi používají:

- Server, na kterém by byla databáze nasazena, by byl High Availability Cluster. Jedná se o systém, který je navržen tak, aby mohl garantovat



Obrázek 2.1: Diagram případů užití



Obrázek 2.2: Diagram nasazení

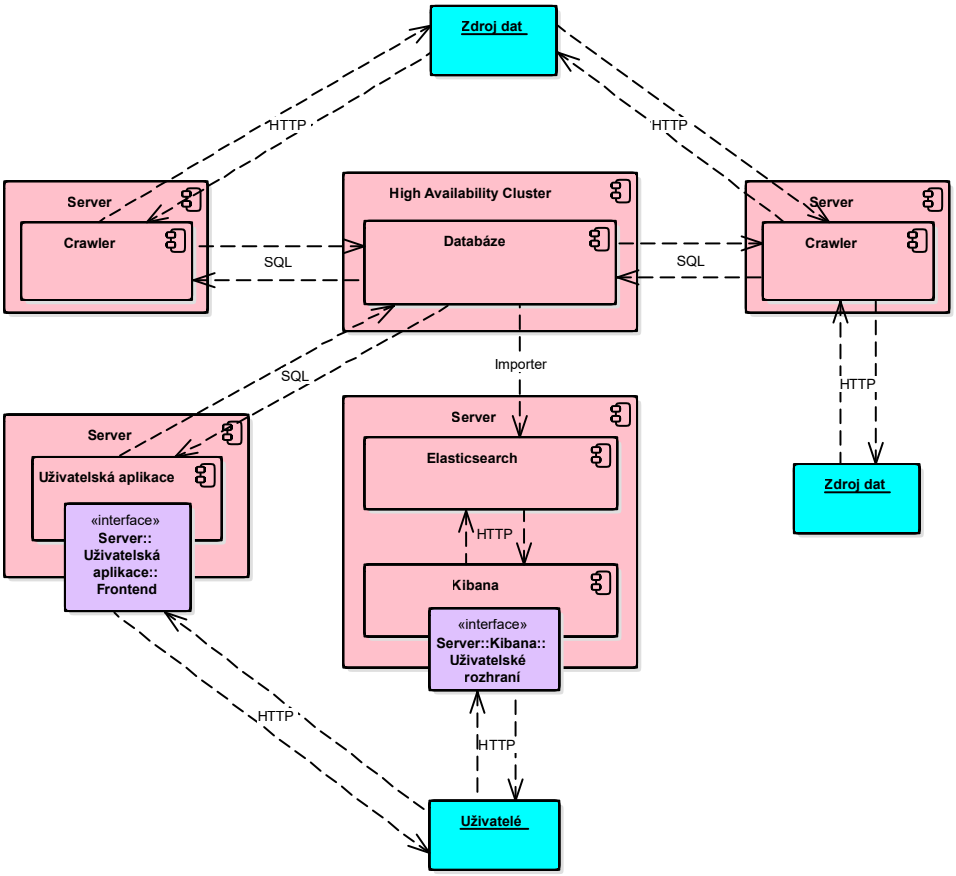
téměř absolutní dostupnost [32]. High Availability Cluster je nabízen za paušální poplatek například poskytovateli hostingových služeb.

- Databáze by byla nasazena nezávisle na alespoň dvou serverech. Jednotlivé instance databáze by se vzájemně replikovaly a synchronizovaly. Všechny komponenty, které by přistupovaly k databázi, by znaly údaje potřebné k připojení ke všem spuštěným instancím databáze a měly by nastavené priority připojení k jednotlivým instancím. Pokud by instance s nejvyšší prioritou nebyla dostupná, komponenta by se automaticky pokusila navázat spojení s jinou instancí databáze, u které má nastavenou druhou nejvyšší prioritu.

První zmíněné řešení je v podstatě bez starostí, jelikož systémové řešení má na starosti poskytovatel hostingu. Nevýhodou však je, že měsíční poplatky za provoz High Availability Cluster systému jsou poměrně vysoké. U druhého uvedeného řešení by bylo nutné vyřešit synchronizaci jednotlivých instancí databáze, k tomu ale existují již hotová řešení [33] [34].

2.3.2 Crawler

Crawler bude mít na starosti stahování dat z inzertního serveru. Dále bude data zpracovávat, parsovat z nich požadovaná data a odesílat je do databáze



Obrázek 2.3: Příklad diagramu nasazení v ideálním případě

v datovém uložišti. Jeden konkrétní crawler může mít na starosti zpracování jen určité části všech dat (například pouze prodej bytů), jiný zase libovolnou jinou část (pronájem bytů) atd. Což umožňuje poměrně velkou možnost distribuce stahování a zpracování dat mezi mnoho nezávislých uzlů. Stejného principu by bylo možné využít i v případě, kdy by bylo žádoucí získávat data z více serverů.

2.3.3 Kibana a Elasticsearch

Nástroj Kibana, fungující na platformě Elasticsearch, bude použit ke generování jak cenové mapy, tak k vytváření grafů vizualizujících jednotlivé parametry nemovitostí. Nevýhodou nástroje Kibana je, že nepodporuje uživatelské účty a uživatelská oprávnění. Existuje sice řešení v podobě nástroje Shield, ale ten není zdarma a je vyžadována licence s každoročním poplatkem [35]. Důsledkem toho by kdokoliv z uživatelů mohl měnit nastavení nástroje Kibana,

mazat vizualizace v něm vytvořené apod. Řešením by bylo využít možnosti vložení map nebo vizualizací vytvořených v nástroji Kibana do jiné stránky. Veškerý obsah, který je vložen do jiné stránky, Kibana automaticky poskytuje pouze ke čtení [36].

Kibana nabízí možnost libovolného filtrování dat, nicméně v uživatelsky přívětivé podobě, kdy si uživatel může pouhým klikáním myši navolit filtry, je filtrování omezeno pouze na možnost volit jednu hodnotu z každého parametru (například pouze dispozice bytu „2+kk“, variantu „2+kk nebo 2+1“ již není možné pohodlně zvolit) [37]. Řešením je poskytnout uživatelsky přívětivé filtrování na stránce, ve které jsou vizualizace z nástroje Kibana vloženy, a výsledek zvolených filtrů předat nástroji Kibana ke zpracování.

2.3.4 Uživatelská aplikace

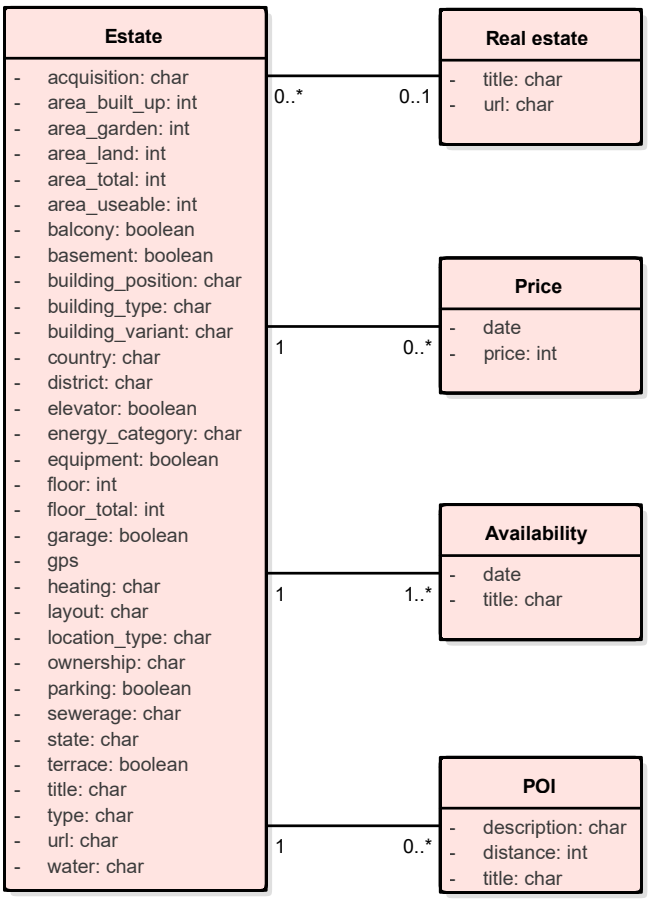
Uživatelská aplikace bude poskytovat rozhraní mezi uživatelem a nástrojem Kibana. Bude nabízet uživatelsky přívětivé rozhraní, které bude obsahovat vlastní možnost filtrování dat a vložené vizualizace a mapy z nástroje Kibana. Uživatelsky navolené filtrování bude předáno nástroji Kibana, který filtrování zohlední ve vizualizacích a v cenové mapě. Aplikace bude k databázi přistupovat pouze za účelem získání hodnot pro filtrování.

2.4 Doménový model

Na obrázku 2.4 je znázorněn doménový model. Struktura jednotlivých tříd vyplývá z průzkumu realitních serverů. Model je tedy sestaven tak, aby obsáhl všechny potřebné údaje pro pozdější analýzu a statistiky.

Model obsahuje 5 základních tříd, kdy hlavní třídou je třída *Estate*, která reprezentuje nemovitost a bude obsahovat kolekce tříd *Price* (údaj o ceně v čase), *Availability* (údaj o dostupnosti inzerátu nemovitosti v čase) a *POI* (bod zájmu v okolí nemovitosti). Taktéž bude obsahovat odkaz na třídu *Real estate*, která reprezentuje realitní kancelář, kterou je nemovitost nabízena k prodeji. Význam jednotlivých atributů je následující:

- **Estate** - konkrétní nemovitost,
  - **title** - stručný popis nemovitosti,
  - **url** - adresa, ze které byl inzerát stažen (je možné ji použít pro spárování údaje v systému s údaji na inzertním serveru),
  - **gps** - GPS souřadnice nemovitosti,
  - **area_built_up** - výměra zastavěné plochy pozemku,
  - **area_garden** - výměra zahrady,
  - **area_land** - celková výměra parcely,



Obrázek 2.4: Doménový model

– **area_total** - celková výměra nemovitosti,
– **area_useable** - výměra užitné plochy nemovitosti,
– **balcony** - zda je součástí nemovitosti balkón,
– **terrace** - terasa,
– **basement** - sklep,
– **parking** - parkovací místo,
– **garage** - garáž,
– **equipment** - vybavení,
– **elevator** - zda je budova vybavena výtahem,
– **floor** - podlaží, ve kterém se nemovitost nachází,
– **floor_total** - celkový počet podlaží v budově,

  - **ownership** - v jakém vlastnictví je nabízená nemovitost; například osobní, družstevní, státní, ...,
  - **country** - kraj, ve kterém se nemovitost nachází,
  - **district** - okres nebo městská část, ve které se nemovitost nachází,
  - **building_type** - typ budovy; například skeletová, cihlová, panelová, kamenná, ...,
  - **state** - stav budovy; například novostavba, velmi dobrý, před rekonstrukcí, ve výstavbě, ...,
  - **building_position** - zda je objekt součástí řadové zástavby, zda stojí samostatně, je rohový apod.,
  - **building_variant** - zda je budova patrová nebo přízemní (především u rodinných domů),
  - **energy_category** - energetická náročnost nemovitosti,
  - **layout** - dispozice bytu a typ domu; například byt 2+1, byt 3+kk, vila, chata, ...
  - **type** - zda se jedná o dům nebo byt,
  - **acquisition** - zda je nemovitost určena k prodeji nebo pronájmu,
  - **heating** - jakým způsobem je zajištěno vytápění objektu; například lokální plynové topení, ústřední dálkové, apod.,
  - **water** - jak je v nemovitosti vyřešen zdroj vody; například dálkový vodovod, lokální zdroj, ...,
  - **sewerage** - jakým způsobem je v objektu vyřešen odpad; například veřejná kanalizace, jímka, septik apod.,
  - **location_type** lokalita, kde se objekt nachází; například v klidné části obce, v rušné části obce, v centru, na sídlišti, na samotě apod.,

- **Real estate** - jaká realitní kancelář nabízí nemovitost k prodeji či pronájmu, včetně URL adresy realitní kanceláře,
- **Price** - cena nabízené nemovitosti v čase; datum uvádí, od kdy je uvedená cena platná,
- **Availability** - dostupnost inzerátu v čase; datum uvádí, od kdy je uvedená dostupnost platná,
- **POI** - místa zájmu v okolí, která mohou ovlivnit zájem o nemovitost; například restaurace, autobusová zastávka, stanice metra, bankomat, pošta, obchod, ...,
  - **title** - obecný popis bodu zájmu; například stanice metra, restaurace, ...,

- **description** - konkrétnější popis bodu zájmu; například: Dejvická, pizzerie, ...,
- **distance** - vzdálenost v metrech od nemovitosti k bodu zájmu.

U tříd *Price* a *Availability* je uvedeno pouze jedno datum, které uvádí, od kdy je uložený údaj platný. Ukončení jeho platnosti je vyřešeno tak, že existuje další záznam s pozdějším datem. Toto řešení bylo zvoleno z toho důvodu, že žádná nemovitost nemůže být inzerována na jednom serveru se dvěma různými cenami a stejně tak nemůže být jeden inzerát dostupný i nedostupný zároveň. V případě, kdy by v budoucnu přibyly inzeráty i z jiných serverů, a každý z nich by uváděl ve stejný čas odlišnou cenu nebo dostupnost pro totožnou nemovitost, stačí přidat ke třídě *Price* či *Availability* identifikátor zdroje.

2.5 Databázový model

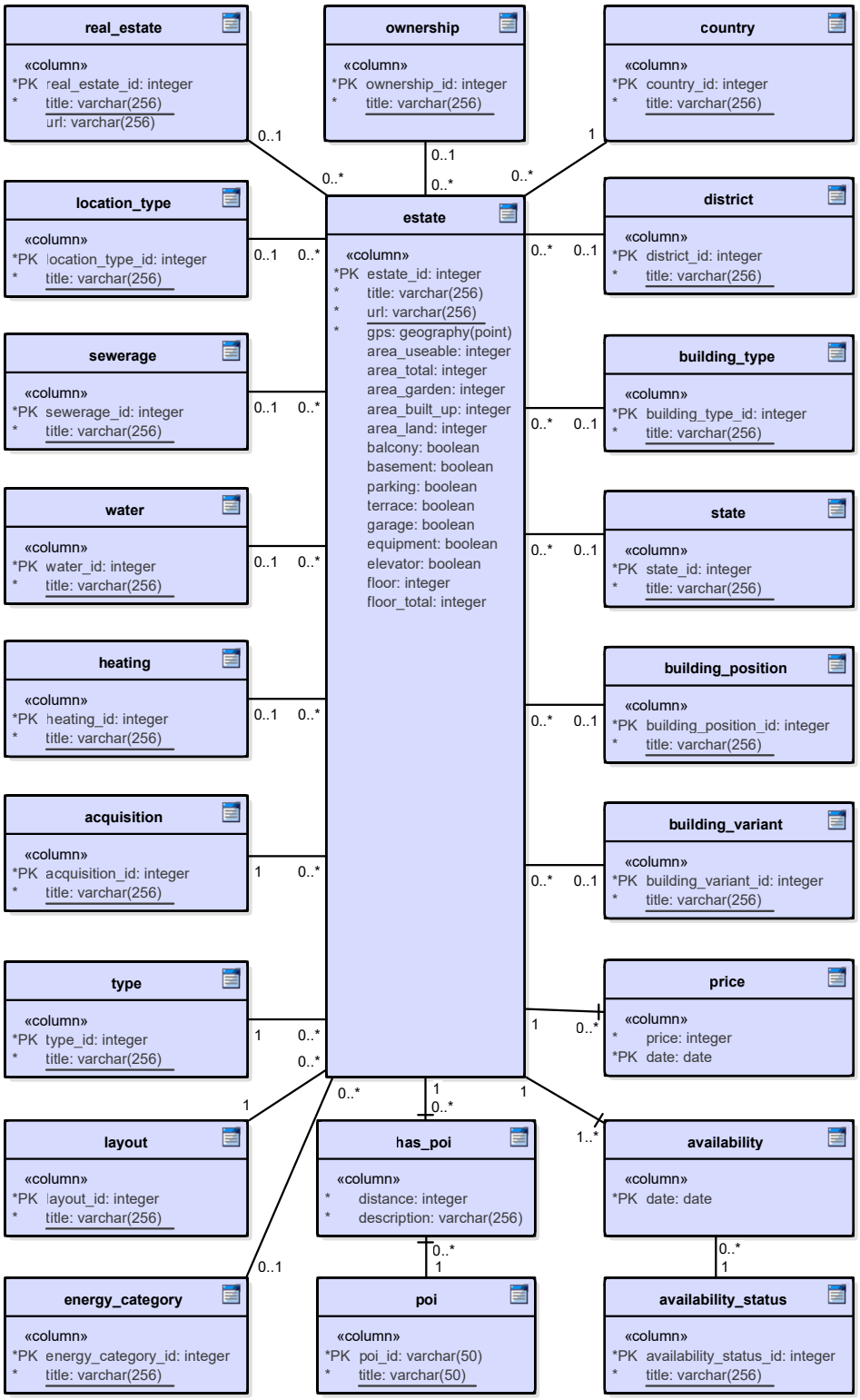
Aplikace bude mít jednu databázi, ke které budou přistupovat všechny komponenty. Jak již bylo zmíněno v kapitole 2.3.1, data budou ukládána do relační databáze. Na obrázcích 2.5 a 2.6 je zobrazeno logické schéma této databáze, kompletní skript pro její vytvoření je uveden v příloze E.1.

Databázový model vychází přímo z doménového modelu. Oproti doménovému modelu byly některé atributy třídy *Estate* vyřešeny samostatnou tabulkou, neboť u jejich hodnot lze předpokládat poměrně často se opakující hodnoty, které je z hlediska údržby a pozdějšího filtrování lepší mít uvedeny zvlášť. Taktéž vztahy many-to-many jsou řešeny extra tabulkou. Význam jednotlivých atributů je stejný jako u doménového modelu z kapitoly 2.4.

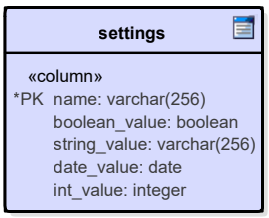
Tabulka *settings* obsahuje různé systémové hodnoty, jako například do databáze naposledy přidaný byt k pronájmu, dům k prodeji apod. Hodnoty uložené v této tabulce slouží především k porovnávání údajů na inzertním serveru a v databázi aplikace, aby bylo možné během aktualizace dat správně rozhodnout o tom, které inzeráty byly změněny nebo nově přidány. Tabulku je možné použít i k ukládání libovolných jiných systémových dat, jelikož je navržena univerzálně pro hodnoty typu boolean, string, date a int.

2.6 Řešení omezení strojového dolování

Jelikož se databáze inzerátů bude pravidelně aktualizovat, lze očekávat velký počet dotazů směrovaných na inzertní server. Pokud by bylo na inzertním serveru 10 000 inzerátů a aktualizace by se nedala nijak optimalizovat, znamenalo by to v nejlepším případě více než 10 000 dotazů každý den ze stejné IP adresy. V takovém případě by se mohlo stát, že by server začal takovouto adresu blokovat, popřípadě omezovat počet dotazů dostupných v určitém časovém úseku, či vyžadovat opisování CAPTCHA kódu.



Obrázek 2.5: Logické schéma databázového modelu nemovitostí



Obrázek 2.6: Logické schéma databázového modelu systémových hodnot

Server sreality.cz počet dotazů nijak neomezuje, ale i přesto bude v aplikaci řešeno preventivní opatření. Crawler bude mít k dispozici seznam proxy serverů. Data bude stahovat v několika etapách:

- prodej bytů,
- pronájem bytů,
- prodej domů,
- pronájem domů.

Před stahováním libovolné etapy crawler náhodně zvolí proxy server, přes který data stáhne. Pokud by během stahování došlo k libovolné chybě a pokus byl neúspěšný, crawler náhodně vybere jiný proxy server a pokusí se pokračovat ve stahování dat od místa, kde došlo k chybě. Navíc se každá etapa začne stahovat v náhodně zvolenou dobu nezávisle na ostatních tak, aby se co nejvíce omezila pravidelnost stahování.

2.7 Způsob získání a zpracování dat

Realitní server sreality.cz nahrává obsah HTML stránek asynchronně pomocí JavaScript kódu, proto není v jejich zdrojovém kódu uveden obsah reprezentující nemovitosti či jejich přehled [11]. Nicméně monitorováním síťové komunikace lze vypozorovat, na jakou adresu a s jakými parametry asynchronní HTTP požadavky odcházejí. Stejně tak lze vypozorovat, jaké odpovědi jsou na požadavky vráceny.

Data jsou získávána ve dvou krocích. Napřed je nutné získat přehled se seznamem všech nemovitostí. Na základě tohoto seznamu se lze dostat k jednotlivým detailům nemovitostí. Situace je naprosto stejná, jako v případě, kdy si stránky prohlíží běžný uživatel. Na detail konkrétní nemovitosti se bez znalosti její URL adresy lze dostat pouze z přehledu se seznamem nemovitostí.

2.7.1 Přehled se seznamem nemovitostí

Přehled se seznamem nemovitostí je stránka, která dokáže zobrazit maximálně 60 položek naráz. K dalším záznamům se lze dobrat pomocí stránkování. Funguje to tedy tak, že na první stránce je 1. až 60. položka, na druhé stránce 61. až 120. položka atd. Stejným způsobem je k tomu potřeba přistupovat i z hlediska získávání dat.

2.7.1.1 HTTP request

Veškeré požadavky pro získání přehledu jsou směrovány na jednu URL adresu (*http://www.sreality.cz/api/cs/v1/estates*), která se liší pouze parametry k ní přidanými [11]. Níže nejsou uvedeny všechny parametry ani jejich hodnoty, ale pouze takové, které mají praktický význam pro zamýšlenou aplikaci. Kompletní seznam hodnot vybraných parametrů je uveden v příloze D.1. V rámci práce mají význam tyto vybrané:

- category_main_cb
  - 1 - byty
  - 2 - domy
- category_type_cb
  - 1 - prodej
  - 2 - pronájem
- locality_region_id
  - 10 - Praha
- page - stránka, od které se stahují data
- per_page - počet vrácených nemovitostí od určité stránky; výchozí hodnota je 20, další alternativy jsou 40 a 60
- sort
  - 0 - nejnovější
- tms - čas ve vteřinách od 1. 1. 1970, slouží ke kontrole cache
- locality_district_id
  - 5001 - Praha 1
  - 5002 - Praha 2
  - 5003 - Praha 3

- 5004 - Praha 4
- 5005 - Praha 5
- 5006 - Praha 6
- 5007 - Praha 7
- 5008 - Praha 8
- 5009 - Praha 9
- 5010 - Praha 10

Žádný z uvedených parametrů není povinný, ale umožňuje velmi efektivně filtrovat dotazovaná data. Je tedy možné, pokud by to bylo potřeba, se například dotázat pouze na prodej bytů po rekonstrukci na Praze 6 s dispozicí 2+kk, které budou seřazeny podle ceny (viz podrobnější přehled v příloze D.1).

2.7.1.2 HTTP response

Odpověď na požadavek se vrací ve formátu JSON. Struktura odpovědi je uvedena ve zdrojovém kódu 2.1. Ne všechny atributy uvedené v kódu jsou potřeba. Atributy dostačující k získání potřebných dat jsou následující:

- **result_size** - celkový počet nabídek dle zvolených kritérií (například prodej bytů),
- **_embedded.estates** - pole se stručnými informacemi ohledně jednotlivých nemovitostí,
- **filter** - parametry z URL adresy požadavku,
- **_links.self.href** - URL stránky, ze které byl JSON získán,
- **per_page** - počet nemovitostí uvedených v JSON souboru,
- **page** - číslo stránky přehledu.

Pole **_embedded.estates**, které obsahuje náhledy jednotlivých nemovitostí, sestává z objektů, které mají stejnou strukturu, jaká je uvedena ve zdrojovém kódu 2.2. K získání potřebných údajů jsou potřeba opět jen některé vybrané atributy. Konkrétně tyto:

- **name** - stručný popis nemovitosti,
- **price_czk.value_raw** - strojově čitelná cena nemovitosti (tzn. bez mezer, oddělovačů řádů, měny apod.),
- **price_czk.unit** - měna, ve které je cena uvedena,
- **_links.self.href** - URL na JSON soubor s detailem nemovitosti,
- **gps** - GPS souřadnice nemovitosti.

Zdrojový kód 2.1: Struktura odpovědi na požadavek přehledu se seznamem nemovitostí

```
{
    "meta_description": "...",
    "result_size": 123,
    "_embedded": {
        "estates": [],
        "is_saved": {},
        "not_precise_location_count": {}
    },
    "title": "...",
    "locality": "...",
    "filter": {},
    "_links": {
        "self": {
            "href": "..."
        },
        "clusters_with_bounding_box_of_first_10": {
            "href": "..."
        },
        "rss": {
            "href": "..."
        }
    },
    "locality_dativ": "...",
    "logged_in": false,
    "per_page": 60,
    "category_instrumental": "...",
    "page": 1
}
```

2.7.2 Detail nemovitosti

Detail nemovitosti je stránka, na kterou se lze bez znalosti konkrétní URL adresy dostat z přehledu se seznamem všech nemovitostí. Obsahuje podrobné a kompletní informace o jedné konkrétní nemovitosti.

2.7.2.1 HTTP request

URL adresa, na kterou je třeba odeslat HTTP požadavek, je uvedena na přehledu se seznamem nemovitostí, konkrétně v atributu **_links.self.href** pole **_embedded.estates[]**. K této adrese již není třeba přidávat žádné parametry [11].

Zdrojový kód 2.2: Struktura JSON objektu reprezentujícího jednu nemovitost v přehledu se seznamem nemovitostí

```
{
    "attractive_offer": 0,
    "_embedded": {
        "favourite": {},
        "note": {}
    },
    "name": "..."
    "locality": "...",
    "region_tip": 0,
    "price": 123,
    "price_czk": {
        "value_raw": 123
        "unit": "...",
        "name": "..."
    },
    "has_video": false,
    "_links": {
        "images": [],
        "self": {
            "href": "..."
        },
        "image_middle2": [],
        "iterator": {}
    },
    "rus": false,
    "seo": {},
    "new": false,
    "paid_logo": 0,
    "hash_id": 123,
    "gps": {}
}
```

Zdrojový kód 2.3: Struktura odpovědi požadavku na detail nemovitosti

```
{
    "meta_description": "...",
    "map": {},
    "logged_in": false,
    "_embedded": {
        "note": {},
        "favourite": {},
        "calculator": null,
        "images": [],
        "seller": {}
    },
    "name": {},
    "locality": {},
    "text": {},
    "price_czk": {
        "unit": "...",
        "name": "...",
        "value": "...",
        "value_raw": 123
    },
    "is_topped": false,
    "_links": {
        "self": {
            "profile": "...",
            "href": "...",
            "title": "..."
        },
        "local_search": {},
        "broader_search": {}
    },
    "poi": [],
    "seo": {},
    "rus": false,
    "is_topped_today": false,
    "items": []
}
```

2.7.2.2 HTTP response

Odpověď na požadavek je, stejně jako v případě přehledu se seznamem nemovitostí, ve formátu JSON. Struktura této odpovědi je uvedena ve zdrojovém kódu 2.3. Řada z uvedených atributů v kódu je pro účely práce nepotřebná. Důležité atributy pro získání požadovaných informací jsou následující:

Zdrojový kód 2.4: Struktura JSON objektu reprezentujícího bod zájmu v detailu nemovitosti

```
{
    "distance": 123,
    "description": "...",
    "url": "...",
    "lon": 0,
    "lat": 0,
    "imgUrl": "...",
    "name": "..."
}
```

- **map** - GPS souřadnice nemovitosti,
- **__embedded.seller** - podrobné informace o prodejci,
- **name** - stručný popis nemovitosti,
- **price__czk.value__raw** - strojově čitelná cena nemovitosti (tzn. bez mezer, oddělovačů řádů, měny apod.),
- **price__czk.unit** - měna, ve které je cena uvedena,
- **__links.self.href** - URL, ze které byl detailní popis nemovitosti stažen,
- **poi** - pole se všemi body zájmu v okolí nemovitosti,
- **seo** - parametry z URL adresy požadavku,
- **items** - pole obsahující podrobnější informace o nemovitosti (například výměra, dispozice, umístění objektu, typ budovy apod.).

Pole **poi**, které obsahuje veškeré body zájmu v okolí nemovitosti, je tvořeno objekty, které mají stejnou strukturu, jaká je uvedena ve zdrojovém kódu 2.4. Pro účely vyvíjené aplikace postačí následující atributy:

- **distance** - vzdálenost místa k nemovitosti,
- **name** - obecný popis bodu zájmu,
- **description** - konkrétnější popis bodu zájmu.

Podrobnější informace o nemovitosti, jako je například výměra, dispozice, typ budovy apod., jsou reprezentovány objekty, jejichž struktura je uvedena ve zdrojovém kódu 2.5. Každý prvek pole **items** je jednoznačně identifikován polem **name**, jehož hodnota **value** je datového typu **type**.

Zdrojový kód 2.5: Struktura JSON objektu reprezentujícího podrobnější informace v detailu nemovitosti

```
{
    "type": "...",
    "name": "...",
    "value": "..."
}
```

2.7.3 Souhrn

Pro získání přehledu se seznamem nemovitostí pro Prahu naprosto postačí přidat do URL adresy následující parametry:

- category_main_cb,
- category_type_cb,
- locality_region_id,
- page,
- per_page.

Kombinace těchto parametrů umožní stahovat data odděleně pro pronájem a prodej, stejně tak pro byty a domy. Všechny ostatní potřebné údaje je možné získat přímo z detailu bytu, proto bude využití přehledu se seznamem nemovitostí sloužit pouze k získání URL adres s jejich detaily.

2.8 Aktualizace dat

Data ze stránek sreality.cz je potřeba stahovat v pravidelných intervalech, aby bylo možné pozorovat vývoj realitního trhu v čase. Je tedy potřeba nějakým způsobem určit, které inzeráty byly aktualizovány, které přidány a které naopak smazány oproti aktuálnímu stavu v databázi.

Řešení by mohlo být využití RSS kanálu, který server sreality.cz nabízí. Umožňuje dokonce i filtrování nabídek podle parametrů v URL adrese [38]. Toto filtrování funguje stejně, jako je tomu v případě získání dat přehledu se seznamem nemovitostí uvedeném v kapitole 2.7.1.1. Nevýhodou tohoto řešení je fakt, že tímto způsobem je možné kontrolovat pouze aktualizované a nové inzeráty, nikoliv ty, které byly z nabídky odstraněny.

Další variantou by mohl být postup, při kterém je stažen kompletně celý přehled se seznamem nemovitostí, vytvořen list všech aktuálně dostupných

nemovitostí na stránkách sreality.cz a ten porovnán se stavem v databázi. Nemovitosti, které v databázi nejsou, jsou nově přidány. Naopak ty, které v databázi přebývají, jsou od poslední kontroly odstraněny z nabídky. Nemovitosti aktualizované od poslední kontroly by bylo možné identifikovat tak, že by byl stažený seznam ze stránek sreality.cz seřazen podle aktuálnosti inzerátů. To lze zařídit parametrem **sort** v adrese požadavku (viz kapitola 2.7.1.1). Jednoduše by algoritmus vyhodnocující, které nemovitosti jsou aktualizovány od poslední kontroly, postupoval v procházení seznamu do té doby, než by narazil na URL nejaktuálnějšího prvku v databázi. V případě že by byl inzerát ukončující procházení z nabídky odstraněn, zaktualizují se všechny dostupné inzeráty. Výhodou tohoto řešení je především to, že detaily nemovitostí jsou staženy pouze pro ty inzeráty, které jsou nově přidány nebo aktualizovány. Počet požadavků na server sreality.cz bude s největší pravděpodobností výrazně nižší, než kdyby se stahovaly všechny detaily. Pochopitelně záleží na počtu změn od poslední aktualizace, ale lze předpokládat, že většina inzerátů zůstane nezměněných.

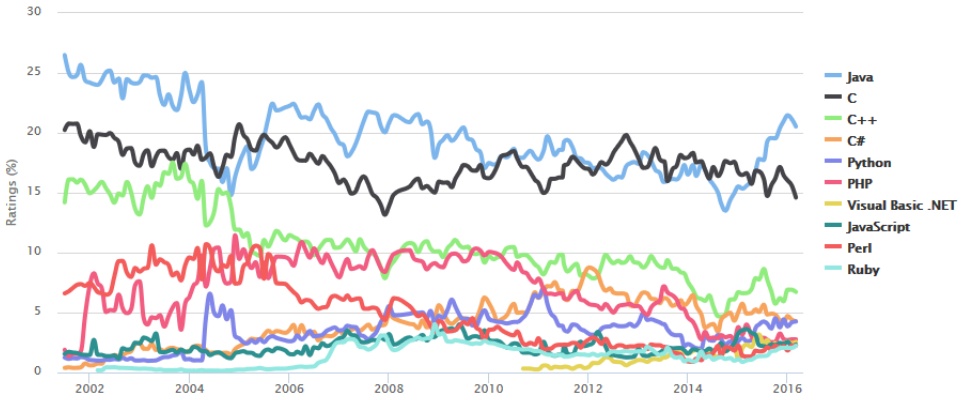
Aktualizace bude probíhat tak, že v případě, kdy se změnila cena nemovitosti, je nová cena uložena s aktuálním datem do databáze. Totéž se týká i dostupnosti inzerátu. Je-li inzerát nově nedostupný, je mu přiřazen status nedostupnosti s aktuálním datem. Pokud je naopak starý inzerát nově dostupný, bude mu přiřazen status dostupnosti. Žádné jiné změny není potřeba sledovat v čase, takže všechny ostatní údaje k nemovitosti jsou přeuloženy aktuálními, jelikož lze předpokládat, že ten, kdo inzerát zaktualizoval, pravděpodobně opravil a doplnil údaje, než aby některé záměrně odstranil.

2.9 Specifikace použitých technologií

2.9.1 Programovací jazyk

Při výběru programovacího jazyka je potřeba vzít v úvahu hned několik věcí. Například na jaké platformě má aplikace běžet, jakým způsobem se bude aplikace v budoucnu nadále rozvíjet a v neposlední řadě i jaké možnosti daný programovací jazyk poskytuje. Aplikace bude v rámci práce nasazena na serveru s operačním systémem Debian Linux, ale vzhledem k velké rozmanitosti operačních systémů, které jsou obecně instalovány na serverech [39], bude při výběru zvolen multiplatformní programovací jazyk. Na obrázku 2.7 je zobrazen graf oblíbenosti programovacích jazyků, podle kterého se jeví jako nejvhodnější programovací jazyk Java, který nabízí i značné množství frameworků [40].

Pro programovací jazyk Java existuje projekt Spring, vyvíjený společností Pivotal Software, Inc., který nabízí několik dílčích frameworků, které lze výhodně použít pro vyvíjenou aplikaci. Například Spring Framework podporuje MVC webové aplikace a RESTfull webové služby [42], čehož lze využít při programování uživatelské aplikace. Jeho další výhodou je, že zjednodušuje psaní



Obrázek 2.7: Oblíbenost programovacích jazyků, převzato z TIOBE software BV [41]

kódu a také následné testování, například odstraněním těsných vazeb mezi třídami, využíváním tzv. POJO objektů, vlastní podporou AOP a mnoha dalšími vlastnostmi [43].

Rovněž by bylo možné použít další z frameworků projektu Spring, konkrétně Spring Boot. Ten obsahuje funkce jako například metriky, kontroly dostupnosti aplikace, externí konfigurační soubory apod. Taktéž umožňuje projekt zkompilovat do balíku, který už v sobě obsahuje Tomcat, Jetty nebo Undertow server, takže ho lze spustit na libovolném počítači s nainstalovanou platformou Java bez toho, aniž by byl vyžadován nainstalovaný server [44]. Těchto vlastností lze velmi výhodně využít v případě crawleru, který je navržen tak, aby mohl být spuštěn na externích serverech.

K testování lze výhodně použít frameworky JUnit, Mockito a PowerMock, které umožňují například testování aplikací psaných v rozšíření Spring Framework, simulaci jejich tříd během testů či dokonce ohlídání volání konstruktorů těchto tříd [45] [46] [47].

2.9.2 Databáze

Podle serveru DB-Engines patří mezi čtyři nejoblíbenější relační databáze tyto [48]:

- Oracle,
- MySQL,
- Microsoft SQL Server,
- PostgreSQL.

Zdarma a bez omezení jsou pouze PostreSQL a MySQL. Vhledem k tomu, že databáze bude v aplikaci sdílena všemi komponentami, je rozhodujícím kritériem výběru především její stabilita a propustnost. Nicméně v tomto ohledu je k dispozici velké množství testů, které jsou si vzájemně protichůdné. Pravděpodobně výsledky závisejí na tom, pomocí jakých aplikací je měření prováděno a na konkrétních verzích databáze, které byly k testování použity. Proto byla zvolena taková databáze, která nabízí větší funkcionalitu. V tomto ohledu je PostgreSQL vybavenější nežli MySQL [49] [50].

2.9.3 Webové rozhraní

Webové rozhraní mezi aplikací a uživatelem (tzv. frontend) bude vyvíjeno ve standardu HTML5, který je podporován všemi moderními rozšířenými internetovými prohlížeči. HTML5 zpřístupňuje funkcionalitu jako například element canvas, úložiště local storage a další, které mohou být použity pro vykreslování grafů ve vektorovém formátu nebo pro ukládání uživatelského nastavení stránky. [51]

K nastylování webové stránky bude použit standard CSS3, jehož vlastnosti jsou taktéž podporovány všemi moderními rozšířenými internetovými prohlížeči. CSS3 umožňuje například animace přechodů a libovolných CSS vlastností, snadnější responzivní řešení stránky a další. [52]

Pro snadnější vývoj a psaní CSS stylů bude použit jazyk SASS, který umožňuje strukturování stylů během vývoje do více souborů, psaní funkcí pro nejčastěji používanou funkcionalitu, vnořování elementů do sebe a další možnosti usnadňující vývoj. [53]

K vývoji uživatelské aplikace bude použit AngularJS framework. Pro tento framework existuje celá řada pluginů třetích stran, které mohou být použity k realizaci filtrování. AngularJS navíc spolupracuje s jQuery frameworkem [54] a v kombinaci s RequireJS frameworkem je možné frontend aplikaci rozdělit do souborů po logických celcích, které jsou následně automaticky spojeny do jednoho souboru [55] [54].

Kapitola 3

Uživatelské rozhraní

Uživatelské rozhraní aplikace bylo navrženo tak, aby uživateli poskytovalo přehledné a intuitivní prostředí. Napřed byly navrženy wireframy pro jednotlivé stránky a následně byly převedeny na hi-fi prototyp, který byl otestován na několika uživatelích. Uživatelské rozhraní bylo v několika iteracích postupně měněno podle výsledku testování s uživateli. Během celého procesu byl kladen důraz na principy Nielsenova desatera [56].

3.1 Wireframy

Wireframe je obsahová kostra výsledné stránky, která má znázornit, jakým způsobem budou jednotlivé prvky na stránce rozloženy. Kromě toho wireframe poskytuje při počátečním návrhu představu o tom, jak bude řešena navigace na stránce a co vše bude součástí stránek. Wireframe je vhodný pro prvotní návrh uživatelského rozhraní, protože veškeré nedostatky při návrhu rozhraní, které jsou během jeho vytváření zjištěny, lze velmi snadno změnit. [57]

Z analýzy a návrhu aplikace vyplývá, že její součástí bude cenová mapa a stránka s analýzou dat. Rovněž by měl uživatel mít možnost si zvolit, která kategorie jej zajímá, zda prodej nebo pronájem bytů či domů. Při vstupu na stránku by měl být uživatel seznámen s tím, kde se nachází a co mu nástroj nabízí. Toto vše lze shrnout do tří různých stránek:

- domovská stránka se základním seznámením s aplikací a vysvětlením, co nástroj umožňuje,
- stránka s cenovou mapou s možností přepínat mezi byty a domy, prodejem a pronájem,
- stránka s analýzou dat s možností přepínat mezi byty a domy, prodejem a pronájem.

Obrázek 3.1: Wireframe domovské stránky

3.1.1 Domovská stránka

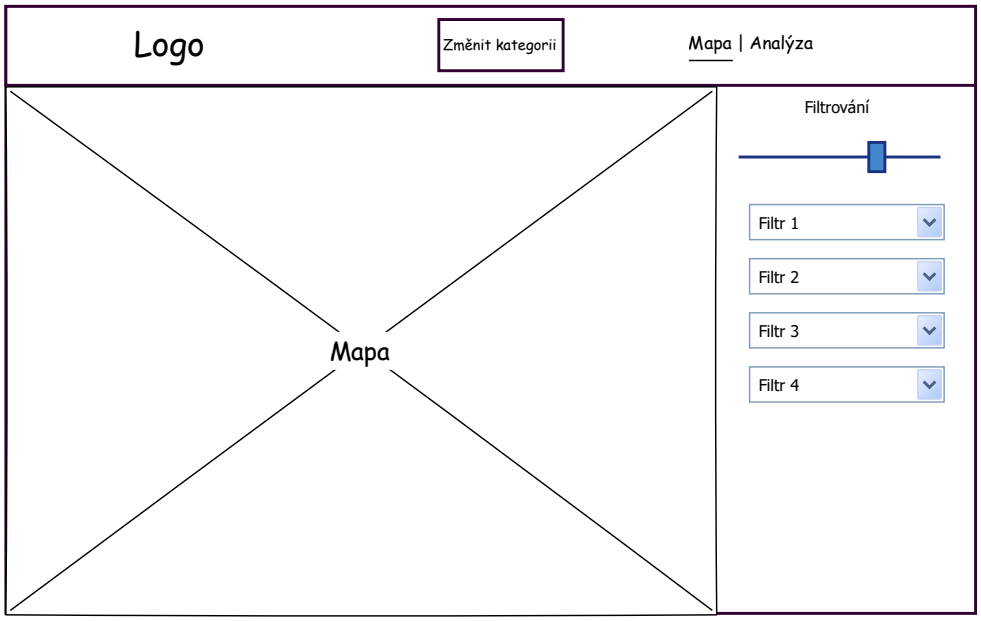
Při návrhu domovské stránky bylo potřeba zohlednit to, že návštěvník stránek nebude pravděpodobně vědět, k čemu nástroj slouží. Z tohoto důvodu byla domovská stránka navržena tak, aby velmi stručně uvedla uživatele do způsobu ovládání aplikace a dala mu jasný pokyn, kam má na stránce pokračovat.

Úvodní text na stránce proto uživatele seznámí s aplikací a v závěru tohoto textu jsou uvedeny navigační tlačítka, která uživatele přesměrují na příslušné stránky. Výsledný wireframe je zobrazen na obrázku 3.1.

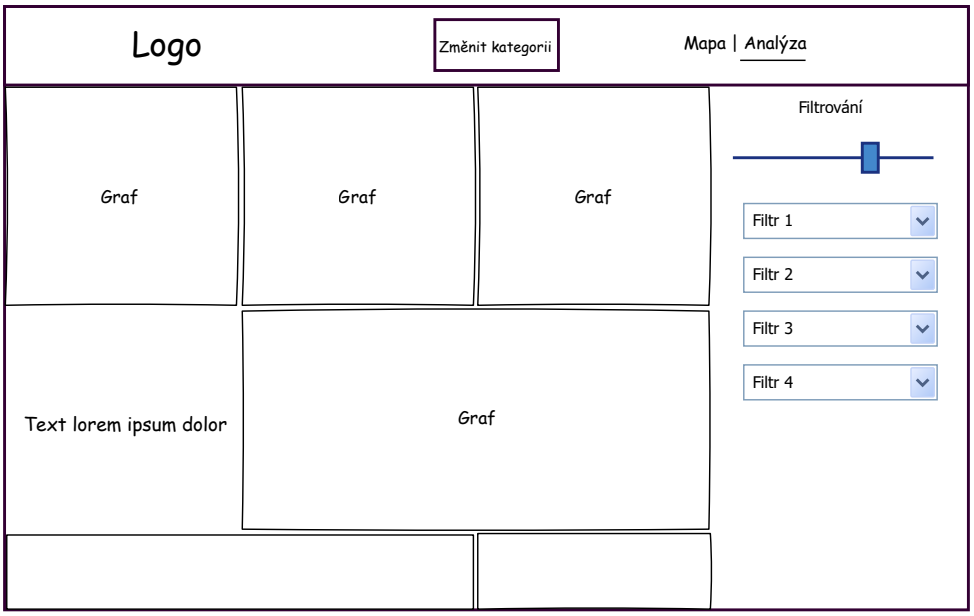
3.1.2 Cenová mapa a analýza dat

Stránky s cenovou mapou a analýzou dat mají sloužit uživateli, který již ví, proč je na konkrétní stránce. Má tedy nějaký záměr, který mu již není potřeba vysvětlovat. Z tohoto důvodu byly obě stránky pojaty minimalisticky, aby uživatele nerozptylovaly od činnosti zbytečnými informacemi a prvky na stránce.

V pravé části stránky jsou umístěny filtry, pomocí kterých může uživatel libovolně filtrovat data, v levé části poté obsah, který odpovídá zvoleným filtrům. V záhlaví stránky je možné změnit kategorii, o kterou má uživatel zájem. Po kliknutí na změnu kategorie se zobrazí modální okno s výpisem všech kategorií. Po kliknutí na libovolnou kategorii dojde ke změně obsahu, ale uživatel zůstane na stejné stránce. Obrázek 3.2 zobrazuje stránku s cenovou mapou a obrázek 3.3 potom stránku s analýzou dat.



Obrázek 3.2: Wireframe stránky s cenovou mapou



Obrázek 3.3: Wireframe stránky s analýzou dat

3.2 Převod na hi-fi prototyp a testování s uživateli

Navržené wireframy byly následně převedeny do hi-fi prototypu, což je funkční verze wireframe, která obsahuje i designové prvky [57]. Tento prototyp byl testován s uživateli různé úrovně a znalosti počítačů. Během testování byl prototyp několikrát měněn na základě zpětné vazby uživatelů, konkrétně byly změněny následující prvky:

- způsob filtrování,
- skrolování na stránce,
- zobrazení procesu nahrávání stránky,
- tlačítko pro přepínání kategorií.

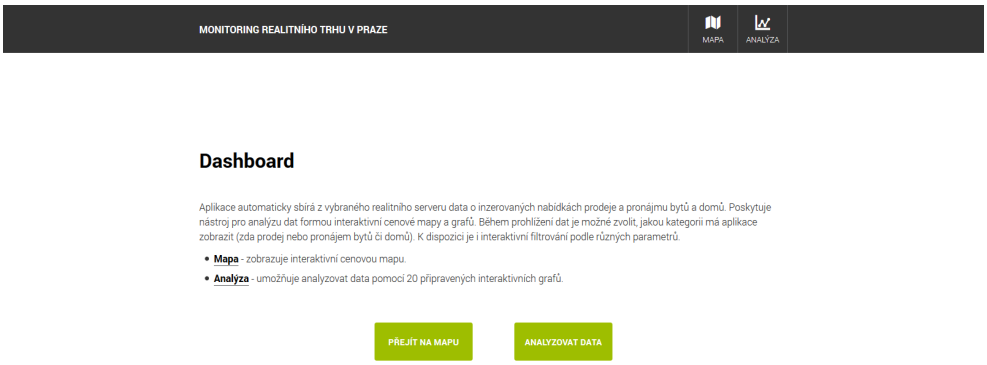
Uživatelé v první verzi prototypu nevěděli, že u volby filtrů je možné zvolit více položek jednoho parametru. Z tohoto důvodu byl změněn box, ve kterém probíhá výběr filtrů konkrétního parametru. Při výběru jedné položky se box nezavírá a označená položka se zvýrazní. Tato změna vedla k tomu, že uživatelé začali vybírat více položek k jednomu parametru.

Během testování si uživatelé stěžovali, že při odskrolování obsahu stránky se odskrolovala i pravá část stránky s filtry. Když chtěli například sledovat změnu jednoho konkrétního grafu při odlišném filtrování, museli odskrolovat zpět nahoru, navolit filtry, potvrdit aktualizaci a opět odskrolovat stránku dolů k příslušnému grafu. Tento proces měl za následek, že uživatel neviděl změnu grafu bezprostředně a nevnímal změnu filtrování na konkrétní graf tak, jak potřeboval. Proto byla stránka rozdělena do dvou částí. Filtrování v pravé části stránky a obsah v levé části stránky jsou od sebe odděleny tak, že lze s nimi skrolovat nezávisle na zbytku stránky.

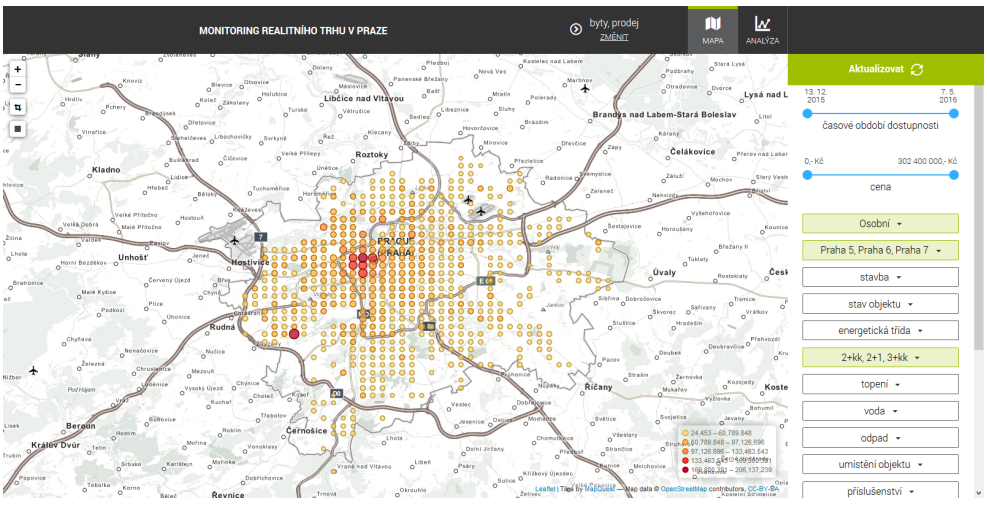
Při pozorování uživatelů nastávala občas situace, kdy při kliknutí v menu uživatel kliknutí opakoval, jelikož se v očekávaném čase nevykreslila nová stránka. Z tohoto důvodu byla do stránky přidána animace načítání stránek, která dává uživateli zpětnou vazbu, že jeho kliknutí je systémem zaznamenáno a probíhá zpracování.

Poslední úpravou bylo zvýraznění tlačítka na změnu kategorie. Uživatelé během testování nevěděli, kde mohou změnit například prodej bytů na pronájem bytů. Proto byla k tlačítku přidána ikona, která navigační prvek zvýraznila, čímž byl problém vyřešen.

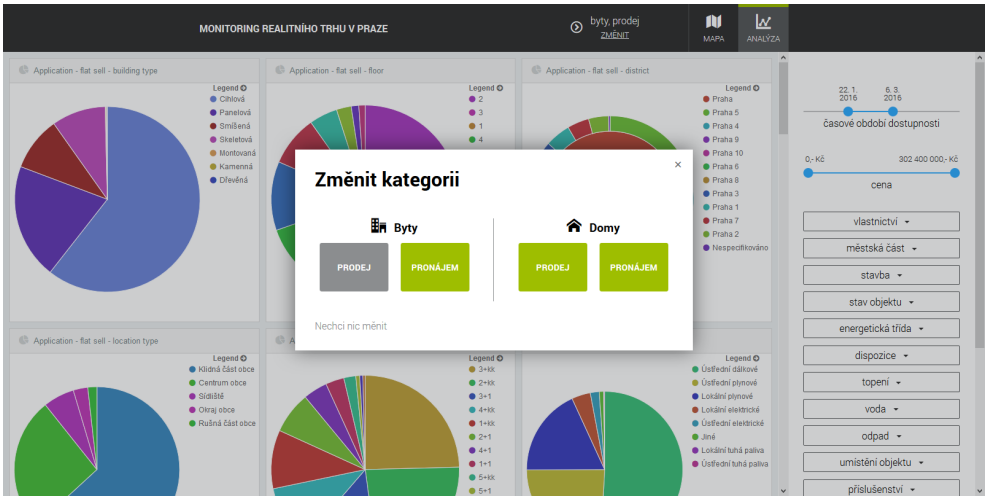
Výsledná podoba domovské stránky je znázorněna na obrázku 3.4. Obrázek 3.5 znázorňuje konečnou podobu stránky s cenovou mapu a obrázek 3.6 s analýzou dat.



Obrázek 3.4: Screenshot uživatelské aplikace, domovská stránka



Obrázek 3.5: Screenshot uživatelské aplikace, cenová mapa



Obrázek 3.6: Screenshot uživatelské aplikace, stránka s analýzou dat

Kapitola 4

Implementace

Implementace probíhala v celkem 4 krocích:

- **Konfigurace serveru** byla nezbytným krokem pro to, aby mohly být všechny komponenty nasazeny a testovány. Proces konfigurace probíhal průběžně podle toho, jaké požadavky se postupně objevovaly během implementace aplikace a nástrojů třetích stran.
- **Společná datová vrstva** byla naprogramována nejdříve, jelikož všechny komponenty používají stejnou databázi a tudíž i stejnou datovou vrstvu.
- **Crawler** byl zprovozněn jako první ze všech komponent, jelikož bylo vhodné sbírat data co nejdříve a zároveň tak mít obsah pro další vývoj aplikace.
- **Kibana** spolu s platformou **Elasticsearch** a **uživatelskou aplikací** byly spuštěny na závěr nad již plněnou a pravidelně aktualizovanou databází.

Společná datová vrstva, crawler a uživatelská aplikace jsou postaveny nad frameworky Spring. Podrobnosti jsou popsány v jednotlivých kapitolách. Pro kompilování zmíněných komponent je použit nástroj Maven společnosti The Apache Software Foundation, který umožňuje širokou nastavitelnost finálního výstupu aplikace a přidávání externích knihoven z veřejného repozitáře, které lze snadno aktualizovat na jejich nejnovější verzi. Taktéž umožňuje vyhledávání chybějících knihoven na základě již napsaného kódu a mnoho dalších funkcí pro usnadnění vývoje. [58]

4.1 Konfigurace serveru

K vývoji nástroje pro monitoring realitního trhu v Praze byl k dispozici server s operačním systémem Debian 7.10 Wheezy, který běží na jádru 3.2.0-4-amd64 GNU/Linux. Na systému byl již nainstalován nginx server ve verzi 1.2.1.

Na server bylo nezbytné nainstalovat další nástroje, aby bylo možné všechny komponenty spustit. Konkrétně bylo nutné nainstalovat následující nástroje:

- prostředí Java 8,
- server Tomcat 8,
- databáze PostgreSQL 9.5.2,
- Elasticsearch a nástroj Kibana.

Instalace prostředí Java 8, serveru Tomcat 8 a databáze PostgreSQL 9.5.2 nevyžadovala žádné speciální konfigurace ani během ní nebyl řešen žádný problém. Instalaci Elasticsearch a nástroje Kibana je věnována samostatná kapitola 4.4.

Jelikož je nginx spuštěn jako hlavní server (tedy ten, který komunikuje s klienty), bylo nezbytné jej nastavit tak, aby přeposílal požadavky na služby běžící v pozadí. Aby bylo možné jednoznačně identifikovat, který požadavek má být kam směrován, byla vytvořena následující struktura:

- veškeré požadavky směrovány na adresář /server/* jsou předávány serveru Tomcat,
- crawler nevyžaduje žádné příchozí požadavky, ale pro otestování jeho funkčnosti jsou všechny požadavky na adresář /crawler/* přeposílány na něj,
- všechny ostatní požadavky jsou předávány nástroji Kibana, který používá Elasticsearch.

Konfigurační soubor nginx serveru, který zařídí výše popsané chování, je uveden v kódu 4.1.

4.2 Společná datová vrstva

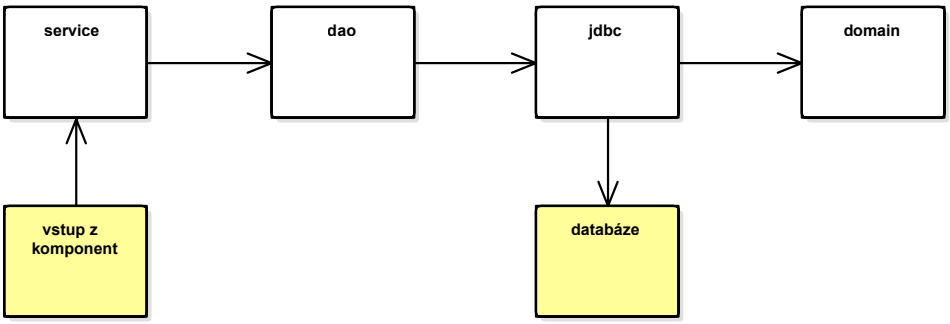
Jelikož pro všechny komponenty existuje pouze jedna databáze, kterou společně sdílejí, sdílejí i datovou vrstvu, která zprostředkovává mapování databáze do Java objektů. K tomuto mapování je použita komponenta Spring JDBC, která je součástí frameworku Spring. Spring JDBC umožňuje spustit nad zvolenou databází libovolný SQL dotaz a jeho výsledek převést pomocí speciálních tříd *ResultSetExtractor* a *RowMapper* na libovolný Java objekt [59].

Datová vrstva je rozdělena do několika skupin (tzv. *package*, což je soubor jedné a více tříd, které tvoří logický celek). Každá z těchto skupin poskytuje konkrétní funkcionalitu. Na obrázku 4.1 jsou znázorněny jednotlivé skupiny, včetně jejich vzájemné závislosti. Jak lze z diagramu vypozorovat, je mezi skupinami respektována hierarchie, proto žádná z hierarchicky nižších skupin

Zdrojový kód 4.1: Konfigurace nginx serveru

```
server {
    # dalsi nastaveni, jako treba listen, server_name a root

    location /server/ {
        gzip on;
        gzip_types "application/json;charset=UTF-8" application/json
            "application/javascript;charset=UTF-8"
            application/javascript text/javascript;
        proxy_set_header X-Forwarded-Host $host;
        proxy_set_header X-Forwarded-Server $host;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_pass http://127.0.0.1:8080/server/;
        proxy_connect_timeout 18000;
        proxy_send_timeout 18000;
        proxy_read_timeout 18000;
        send_timeout 18000;
    }
    location /crawler/ {
        proxy_set_header X-Forwarded-Host $host;
        proxy_set_header X-Forwarded-Server $host;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_pass http://127.0.0.1:8080/crawler/;
        proxy_connect_timeout 18000;
        proxy_send_timeout 18000;
        proxy_read_timeout 18000;
        send_timeout 18000;
    }
    location / {
        proxy_set_header X-Forwarded-Host $host;
        proxy_set_header X-Forwarded-Server $host;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_pass http://127.0.0.1:5601/;
        proxy_connect_timeout 18000;
        proxy_send_timeout 18000;
        proxy_read_timeout 18000;
        send_timeout 18000;
        auth_basic "Restricted Access";
        auth_basic_user_file /etc/nginx/htpasswd.users;
    }
}
```



Obrázek 4.1: Závislost jednotlivých package v rámci datové vrstvy (šipky znázorňují vztah „zdrojový package používá funkcionalitu cíle“, žlutou barvou jsou odlišeny vstupy a externí zdroje)

nepoužívá funkce ze skupin jim nadřazených. Díky tomu je zaručena nezávislost hierarchicky nižších skupin na těch vyšších a zvyšuje se tím udržitelnost a rozšiřitelnost. Skupiny jsou následující:

- **domain** - hierarchicky nejnižší skupina, která reprezentuje modifikovaný doménový model z kapitoly 2.4. Podrobnosti o změnách v doménovém modelu jsou uvedeny v kapitole 4.2.1.
- **jdbc** - hierarchicky druhá nejnižší skupina, která implementuje třídy *ResultSetExtractor* a *RowMapper* ze Spring JDBC frameworku. Zajišťuje převod SQL dotazů na Java objekty (například na třídy z domain vrstvy nebo na jiné datové typy).
- **dao** - skupina hierarchicky nadřazená předchozí, která vytváří SQL dotazy, které jsou následně odeslány do databáze.
- **service** - nejnadřazenější skupina, která volá funkce DAO vrstvy. Má na starosti kontrolu, zda jsou již objekty v databázi, jestli požadavek do databáze proběhl správně nebo je-li objekt ke vložení do databáze správný apod.

4.2.1 Změny doménového modelu

Při implementaci se ukázalo, že návrh doménového modelu z kapitoly 2.4 je sice navržen správně, ale komplikuje proces přidávání nové nemovitosti. Problém byl ten, že některé údaje o nemovitosti jsou ve stažených datech uvedeny pouze formou ID. Chybí v nich textový popis toho, co ID reprezentuje. Konkrétně tato situace nastává u dispozice a městské části, kdy například namísto 2+kk a Praha 6 jsou uvedeny pouze ID 4 a 5006. Doménový model ale předpokládá řetězce s konkrétní hodnotou.

Řešením by bylo se vždy dotázat do databáze na textovou hodnotu, která je zastupována konkrétním ID, nicméně by tím byl proces zpracování nových nemovitostí zpomalen a zbytečně by se zatěžovala databáze, jelikož by s každou nemovitostí probíhaly dva dotazy do databáze navíc.

Dalším řešením by mohla být převodová tabulka, která by se načetla z databáze před začátkem zpracování, a podle ní by se posléze dosazovaly příslušné hodnoty.

Zvolené řešení je ale nakonec takové, že všechny objekty, které mohou být reprezentovány jak ID, tak i hodnotou, jsou řešeny samostatnou třídou, která obsahuje obě hodnoty. Výhodou takového řešení je, že se pracuje se stejným objektem napříč celou aplikací a není potřeba zjišťovat, zda se pracuje s ID nebo textovou hodnotou, jelikož jsou obě dvě hodnoty k dispozici.

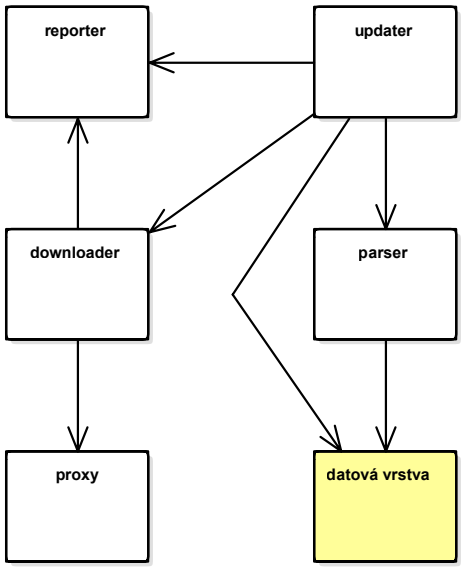
4.3 Crawler

Crawler slouží ke sběru informací z inzertního serveru sreality.cz. Je postaven na frameworku Spring Boot, který poskytuje nástroje pro měření dostupnosti aplikace, možnost externího konfiguračního souboru a další. Především ale nabízí funkcionalitu zkompilovat aplikaci do jednoho souboru, který již obsahuje server Tomcat, Jetty či Undertow. Díky tomu je možné jej spustit na libovolném počítači, který ani nemusí mít nainstalovanou serverovou aplikaci. Díky možnosti zkompilování do souboru *jar* je jedinou podmínkou nasazení crawleru na zcela libovolném počítači přítomnost JVM. [44]

Funkce crawleru lze rozdělit do několika skupin (tzv. *package*, což je soubor jedné a více tříd, které tvoří logický celek), které spolu úzce spolupracují. Na obrázku 4.2 jsou jednotlivé skupiny zobrazeny, včetně vzájemných závislostí. Skupiny jsou následující:

- **downloader** má na starosti stažení veškerých potřebných dat,
- **updater** určuje, které údaje mají být staženy, kdy mají být staženy a následně které z nich mají být uloženy či aktualizovány,
- **parser** zpracovává stažená data a převádí je do Java objektů,
- **proxy** nastavuje proxy připojení, aby nedošlo k omezení či zablokování IP adresy, kterou crawler používá,
- **reporter** odesílá emailem reporty, v nichž je uveden souhrn a výsledek každodenní aktualizace.

Všechny součásti jsou navrženy tak, aby byly co nejvíce odolné vůči chybám. Jelikož crawler pracuje výhradně s daty třetí strany, není možné garantovat správnost vstupu. Taktéž není možné garantovat funkčnost proxy serveru



Obrázek 4.2: Závislost jednotlivých package v rámci crawleru (šipky znázorňují vztah „zdrojový package používá funkcionalitu cíle“, žlutou barvou jsou odlišeny externí zdroje)

nebo dostupnost inzertního serveru sreality.cz. Bližší popis toho, jak se jednotlivé části vypořádávají s případnými chybami či nečekaným vstupem, je uveden v příslušných podkapitolách.

4.3.1 Downloader

Downloader umožňuje stáhnout JSON odpovědi inzertního serveru sreality.cz na disk. Mezi hlavní funkcionalitu patří stažení přehledu se seznamem všech nemovitostí a stažení všech nebo pouze vybraných detailů nemovitostí. A to vždy v rámci zvolené kategorie (byty/domy, prodej/pronájem). V případě, kdy již nejsou stažená data potřeba, downloader je zkomprimuje do formátu zip a archivuje pro případ, kdy by bylo potřeba rekonstruovat databázi ze zálohy dat.

Samotné stahování dat probíhá tak, že je poslán HTTP GET požadavek na adresu serveru sreality.cz a odpověď je uložena na disk ve formátu JSON. Pokud při stahování dojde k chybě nebo server do 10 sekund neodpoví, downloader zažádá o změnu proxy a pokus opakuje. Toto se děje celkem desetkrát, a pokud neuspěje ani na desátý pokus, adresu přeskočí a uloží ji do seznamu neúspěšných stažení. Po ukončení downloadu všech dat se pokusí opět stejným způsobem stáhnout všechny neúspěšné adresy. Pokud ani tehdy neuspěje, adresy definitivně vynechá a dále s nimi v aktuálním procesu nepracuje.

4.3.2 Updater

Updater má za úkol aktualizovat data v databázi podle aktuálního stavu inzerátů na serveru sreality.cz. Ze staženého přehledu se seznamem všech nemovitostí rozhodne, která nemovitost má být označena za nedostupnou, která zaktualizována a která nově přidána. Aktualizace probíhají ve 4 etapách (prodej bytů, pronájem bytů, prodej domů, pronájem domů) v náhodně zvolené časy, aby se vyloučila jakákoliv pravidelnost vedoucí k odhalení strojového stahování.

Proces aktualizace funguje tak, že přehled se seznamem všech nemovitostí je stažen již seřazený podle nejaktuálnější nabídky. Toto řazení nabízí přímo server sreality.cz a na přední místa umisťuje buďto nemovitosti aktualizované anebo nově přidané. Updater získá z tohoto seznamu URL jednotlivých detailů ve stejném pořadí, v jakém byly staženy, a porovnává je s databází.

Všechny URL adresy, které v databázi ještě nejsou, jsou přidány jako nové nemovitosti. Všechny URL, které jsou v databázi vedeny jako aktuálně nabízené nemovitosti, a zároveň již nejsou na staženém seznamu, jsou zarchivovány jako již nedostupné. Všechny ostatní jsou potenciálně zaktualizované. O tom, u které URL byl její obsah skutečně změněn a u které ne, je rozhodnuto na základě nejaktuálnější nemovitosti z předchozí aktualizace. Jelikož je stažený seznam seřazen podle aktuálnosti, jakmile dorazí algoritmus k naposledy nejaktuálnější nemovitosti, je jasné, že vše za ní je již v databázi aktuální.

Mohou nastat dva případy, které výše uvedenému rozporují. A to když nejaktuálnější nemovitost z poslední aktualizace:

- na staženém seznamu není,
- byla rovněž aktualizována.

V prvním uvedeném případě by došlo pouze k tomu, že by se zaktualizoval kompletně celý stažený seznam, jelikož by nebyla nalezena URL adresa, která proces zastaví. To ale ničemu nevadí. V druhém případě by mohlo nastat to, že byla vynechána aktualizace několika nemovitostí. Pro takový případ je porovnáván kompletní obsah nejaktuálnější nemovitosti z poslední aktualizace s jejím aktuálním obsahem na inzertním serveru. Pokud se obsahy od sebe libovolně odlišují, jsou opět zaktualizovány všechny nemovitosti.

Primitivním řešením by mohlo být, že vždy, při každém spuštění updateru, dojde k aktualizaci všech nemovitostí. To ale v případě, kdy je v nabídce v průměru 10 000 nemovitostí každý den, výrazně zvedá počet požadavků a zatěžuje server sreality.cz. Díky zvolenému řešení, které stahuje pouze změněné nemovitosti, se v průměru aktualizuje každý den přibližně 3 000 nemovitostí, což je 30% z celkového počtu.

4.3.3 Parser

Parser slouží k převodu stažených dat ve formátu JSON do Java objektů, se kterými se dále pracuje. Získání jednotlivých informací do finálního objektu je vždy řešeno samostatnou funkcí, jelikož v případě, kdy dojde k chybě (neexistující část souboru, chybějící informace, ...) je výjimka odchycena v rámci funkce a navrácena hodnota *null*. Tato hodnota je předána do výsledného objektu a proces dále pokračuje bez přerušení.

4.3.4 Proxy

Proxy slouží k nastavení proxy připojení. Důvodem je to, aby každodenní aktualizace nemohly být jednoduše zablokovány na základě IP adresy. Proxy má vlastní seznam několika IP adres funkčních a ověřených proxy serverů, který je čas od času zaktualizován ručně. Při požadavku na nastavení proxy je náhodně vybrána libovolná IP adresa ze seznamu, která je použita. Ihned po nastavení proxy serveru je proveden pokus o stažení stránky example.org. Pokud se tento pokus nepovede, je nastavena jiná náhodná IP adresa.

Důvod, proč proces získávání IP adres proxy serverů nebyl zautomatizován (například dostupné proxy listy či použití projektu Tor), je ten, že by nebylo možné zajistit kvalitní server s rychlou odezvou, vysokou dostupností a dobrou rychlostí připojení. Při aktualizaci nemovitostí se průměrně stahuje 60MB dat, což je výrazně urychleno, pokud je k dispozici kvalitní a ověřený proxy server.

4.3.5 Reporter

Reporter slouží k odeslání informačního emailem o každodenní aktualizaci. Je implementován jako singleton, protože je nepravidelně volán updaterem a downloaderem po celou dobu denní aktualizace. Reporter poskytuje následující funkcionalitu:

- založení nového logu,
- zapsání předané zprávy do logu,
- založení nového souboru se statistikami,
- zapsání předané zprávy do souboru se statistikami,
- vytvoření emailu s předaným obsahem a předmětem,
- připojení logu k emailu ve formě přílohy,
- odeslání emailu na zadanou emailovou adresu.

Celý cyklus začíná tím, že updater zažádá o založení logu a statistického souboru. Downloader a updater zapisují do logu veškeré chyby, které nastaly

při procesu aktualizace. Po skončení aktualizace je do statistického souboru updaterem zapsáno, kolik nemovitostí bylo v databázi změněno (včetně identifikačních URL) a odeslán email se souhrnem, kde je v příloze připojen log s chybami, které nastaly. Součástí emailu i statistického souboru je rovněž informace o tom, které adresy se nepodařilo během aktualizace stáhnout.

4.4 Kibana a Elasticsearch

Nástroj Kibana, fungující na platformě Elasticsearch, je použit pro veškeré vizualizace v aplikaci. Kibana umožňuje vizualizovat data formou hodnot, grafů a map. Rovněž je možné data libovolně filtrovat.

K tomu, aby bylo možné spustit nástroj Kibana jako serverovou službu, bylo potřeba udělat postupně několik kroků:

- nainstalovat a zprovoznit Elasticsearch jako serverovou službu,
- nainstalovat a zprovoznit nástroj Kibana jako serverovou službu,
- nakonfigurovat export relační databáze do uložiště dat Elasticsearch,
- nakonfigurovat nástroj Kibana pro zobrazení dat z Elasticsearch uložiště.

První dva body byly bez jakýchkoliv komplikací, stačilo postupovat přesně podle oficiálního návodu [60]. Stejně tomu bylo i v případě konfigurace nástroje Kibana. Komplikace nastaly pouze při převodu relační databáze do uložiště dat Elasticsearch.

4.4.1 Převod PostgreSQL do uložiště Elasticsearch

K převodu relační databáze do dokumentově orientovaného uložiště dat Elasticsearch byl použit nástroj Elasticsearch JDBC Importer. Ten umožňuje libovolným SQL dotazem vytvořit JSON objekt, který je následně uložen do uložiště Elasticsearch. [25]

Aby bylo možné v Elasticsearch použít string hodnoty k vizualizaci, musejí být v uložišti zaindexovány jako *not_analyzed*. V opačném případě je Elasticseaech považuje za klasický full text, nad kterým lze vyhledávat. Stejně tak je potřeba GPS souřadnice uložit jako typ *geo_point*, jinak by nebylo možné zobrazovat vizualizace na geografické mapě. [61]

Kompletní JSON objekt pro nastavení namapování výsledků SQL dotazu do Elasticsearch uložiště je uveden v kódu 4.2. Tímto nastavením se zajistí, že se při požadavku na vložení dat do Elasticsearch uložiště použijí definovaná pravidla.

Aktualizace relační databáze probíhá každý den. Bylo by proto vhodné, aby se stejně často aktualizovalo i uložiště Elasticsearch. Nástroj Elasticsearch

Zdrojový kód 4.2: JSON objekt s nastavením mapování SQL dotazu do Elasticsearch uložiště

```
{
    properties: {
        acquisition: {type: "string", index: "not_analyzed"},
        area_garden: {type: "long"},
        area_built_up: {type: "long"},
        area_land: {type: "long"},
        area_total: {type: "long"},
        area_useable: {type: "long"},
        available_from: {type: "date"},
        available_to: {type: "date"},
        balcony: {type: "boolean"},
        basement: {type: "boolean"},
        building_type: {type: "string", index: "not_analyzed"},
        building_position: {type: "string", index: "not_analyzed"},
        building_variant: {type: "string", index: "not_analyzed"},
        country: {type: "string", index: "not_analyzed"},
        district: {type: "string", index: "not_analyzed"},
        elevator: {type: "boolean"},
        energy_category: {type: "string", index: "not_analyzed"},
        equipment: {type: "boolean"},
        floor: {type: "long"},
        floor_total: {type: "long"},
        garage: {type: "boolean"},
        gps: {type: "geo_point"},
        heating: {type: "string", index: "not_analyzed"},
        layout: {type: "string", index: "not_analyzed"},
        location_type: {type: "string", index: "not_analyzed"},
        ownership: {type: "string", index: "not_analyzed"},
        parking: {type: "boolean"},
        price: {type: "long"},
        price_per_square_meter: {type: "long"},
        real_estate: {type: "string", index: "not_analyzed"},
        real_estate_url: {type: "string", index: "not_analyzed"},
        state: {type: "string", index: "not_analyzed"},
        terrace: {type: "boolean"},
        title: {type: "string", index: "not_analyzed"},
        type: {type: "string", index: "not_analyzed"},
        url: {type: "string", index: "not_analyzed"},
        water: {type: "string", index: "not_analyzed"},
        sewerage: {type: "string", index: "not_analyzed"}
    }
}
```

Zdrojový kód 4.3: JSON objekt s nastavením aktualizace dat v Elasticsearch uložišti

```
{
    type : "jdbc",
    jdbc : {
        url : "jdbc:postgresql://",
        user : "",
        password : "",
        sql : "",
        autocommit : true,
        schedule : "0 0 20 ? * *",
        index : "",
        type : ""
    }
}
```

JDBC Importer podporuje syntaxi Quartz cron expression formátu [25], pomocí kterého lze nastavit téměř libovolně kdy a jak často se má aktualizace spouštět. Výsledný konfigurační JSON objekt, který spouští aktualizaci, je uveden v kódu 4.3.

Atribut *type* určuje typ zdrojové databáze a objekt *jdbc* její konfiguraci. Atributy *jdbc.url*, *jdbc.user* a *jdbc.password* slouží k přístupu do zdrojové databáze. SQL dotaz, který se má vykonat, je uveden v atributu *jdbc.sql*. Každý sloupec výstupu tohoto dotazu, jehož jméno se shoduje s názvem atributu v nastavení mapování (kód 4.2), bude namapován dle uvedených pravidel. Důležité je nastavení *jdbc.autocommit* na hodnotu true, neboť bez toho by se neaktualizovaly již existující prvky v uložišti Elasticsearch, ale přidávaly by se pouze nové, jejichž ID ještě nejsou v uložišti uvedena. Atribut *jdbc.schedule* slouží k naplánování automatického spuštění (v tomto konkrétním případě každý den ve 20:00). Pole *jdbc.index* a *jdbc.type* mohou být libovolná a určují pouze to, pod jakými názvy budou v Elasticsearch uložišti dohledatelná.

4.4.2 Řešené problémy

Během procesu zprovozňování Elasticsearch a rozhraní Kibana bylo nutné vyřešit mnoho drobných a tři poměrně zásadní problémy. Dále jsou uvedeny pouze ty zásadní. Rovněž je uvedeno, jaké řešení bylo zvoleno.

4.4.2.1 Vizualizace vnořených objektů

Elasticsearch podporuje, stejně jako například klasické databáze, odkazy na jiné objekty pomocí cizích klíčů. Toho by se dalo využít například u ceny

nemovitosti, její dostupnosti nebo bodů zájmu v okolí nemovitosti, kdy typicky na jednu nemovitost připadá hned několik záznamů. Provedení by bylo poměrně snadné. Objekt s nemovitostí by měl unikátní ID a objekty reprezentující její cenu, dostupnost nebo bod zájmu by měly uveden atribut *parent*, jehož hodnota by byla rovna hodnotě tohoto ID. Nebo by bylo možné rovnou SQL dotazem vytvořit objekt k uložení do Elasticsearch uložiště, který by obsahoval další vnořené objekty. [62]

Problém nastává v momentě, kdy je potřeba v prostředí Kibana vizualizovat nebo přistupovat k těmto vnořeným či odkazovaným objektům. Tuto funkcionalitu Kibana nemá a jednoduše takové objekty ignoruje. [63]

Tento problém byl vyřešen kompromisem. Ukládána je pouze poslední aktuální cena. Dostupnost je uváděna ve dvou parametrech - od kdy je nemovitost inzerována a do kdy byla inzerována. V případě, že je inzerát stále aktuální, je parametr uvádějící konec inzerce prázdný. Pokud byl inzerát ze stránek odstraněn a později znovu zveřejněn, je tento fakt ignorován a je uvedeno pouze datum prvního zveřejnění a v případě, že inzerát již není dostupný, je uvedeno datum posledního dne inzerce. Body zájmu nejsou uváděny vůbec.

4.4.2.2 Úniky paměti (memory leaks)

Při spuštění prostředí Kibana nastala situace, kdy se čas od času služba sama zastavila a bylo nutné ji restartovat. Při prozkoumání logu aplikace nic nenaznačovalo tomu, že by byla ukončena z důvodu jakékoliv chyby. Při bližším zkoumání a pozorování bylo zjištěno, že bezprostředně po spuštění služby zabírala 90MB paměti RAM. Po dvou hodinách to bylo již 160MB a po dalších několika hodinách již 270MB RAM.

Ukázalo se, že tento problém má více uživatelů. Podle všech indicií tato situace vzniká při používání nástroje Kibana ve verzi 4.1 a vyšší v kombinaci s určitou verzí Node.js. Při takovémto sestavení nastávají úniky paměti (tzv. memory leaks), které zahltí RAM paměť a služba je poté ukončena. [64]

Tento problém byl vyřešen tak, že během spouštění nástroje Kibana, jakožto serverové služby, je zároveň i nastaven maximální limit alokované paměti (parametr *–max-old-space-size=250*). Při dosažení tohoto limitu je spuštěn automaticky garbage collector, který uvolní alokovanou paměť od nedostupných bloků [64]. Díky tomu není služba zastavena a funguje bez potíží.

4.4.2.3 Hromadná aktualizace Elasticsearch databáze

Při automatické aktualizaci dat v Elasticsearch uložišti se spouštějí celkem 4 SQL dotazy. A to z toho důvodu, že v Elasticsearch uložišti jsou jednotlivé kategorie (byty/domy, prodej/pronájem) uloženy pod různými indexy. Toto rozdělení umožňuje prohlížet a spravovat data z jedné kategorie zcela nezávisle na všech ostatních. Ideální stav by tedy byl, aby konfigurace aktualizace

Zdrojový kód 4.4: JSON objekt s náznakem nastavení aktualizace několika indexů Elasticsearch uložiště zároveň

```
{
    type : "jdbc",
    schedule : "0 0 20 ? * *",
    jdbc : [{}, {}, {}, {}]
}
```

vypadala tak, jak je naznačeno v kódu 4.4. Podle oficiální dokumentace je takový zápis možný a funkční [25].

Problém ovšem nastává v případě, kdy se zápis atributu *jdbc* formou pole kombinuje s atributem *schedule*. Tato funkcionalita, kdy je v jeden čas spuštěno několik JDBC sekvencí, je podle vývojáře nástroje Elasticsearch JDBC Importer velmi problematická a byla z něj dokonce úplně odebrána [65].

Problém byl vyřešen tak, že na serveru neběží jeden, ale rovnou čtyři procesy, které mají na starost aktualizaci dat v Elasticsearch uložišti. Každý z těchto procesů aktualizuje jednu kategorii, a aby nedocházelo ke konfliktům, spouští se postupně s 5 minutovými odstupy.

4.4.3 Uživatelské rozhraní

Uživatelské rozhraní nástroje Kibana není příliš vhodné pro účely této práce. Kibana nepodporuje uživatelské účty ani uživatelská oprávnění, proto každý návštěvník může být zároveň administrátorem. Rovněž nepodporuje uživatelsky přívětivé filtrování, protože se jedná o univerzální nástroj, který poskytuje širokou funkcionalitu.

Kdokoliv z uživatelů by tedy měl možnost měnit systémové nastavení nástroje Kibana, měnit či mazat v něm vytvořené vizualizace a definované dashboardy. Rovněž by měl uživatel možnost libovolně manipulovat s daty v uložišti v rozsahu, jaký Kibana prostřednictvím frontendu umožňuje.

Kibana podporuje téměř libovolné filtrování dat. Nicméně v uživatelsky přívětivé podobě, kdy není po uživateli vyžadována žádná znalost speciální syntaxe, lze filtrovat data pouze na úrovní logického součinu. Například je možné filtrovat všechny nemovitosti, které splňují:

- 3. podlaží a zároveň
- novostavba a zároveň
- Praha 6

Pokud by ale uživatel vyžadoval pokročilejší filtrování, kde kromě logického součinu je potřeba i logický součet, je nutné znát syntaxi, která toto zařídí. Například výpis všech nemovitostí, pro které platí:

- 3. nebo 4. podlaží a zároveň
- novostavba nebo po rekonstrukci a zároveň
- Praha 6

Tohoto vyfiltrování nelze docílit pouhým klikáním na parametry, ale lze jej poměrně snadno dosáhnout pomocí následující syntaxe:

```
floor: (3 OR 4) AND state: ("Novostavba" OR "Po rekonstrukci") AND
    district: "Praha 6"
```

Problém s uživatelskými účty je vyřešen tak, že vizualizace vytvořené v nástroji Kibana jsou vloženy do jiné stránky pomocí *iframe*. Díky tomuto je veškerý vložený obsah do jiné stránky (tzv. *embedded content*) poskytnut nástrojem Kibana pouze ke čtení. Navíc takto vložený obsah postrádá veškerou navigaci a znemožňuje uživateli jakýkoliv pohyb v rámci nástroje Kibana. [36]

Stejným způsobem je vyřešen i problém s uživatelskou přívětivostí filtrování. Je-li obsah vložený ve stránce pomocí *iframe* na stejné doméně jako je stránka obsahující *iframe*, může JavaScript libovolně přistupovat k oběma dokumentům. Je tedy možné ovládání filtrů přesunout na stránku obsahující *iframe* a strojově vytvořený, syntakticky správný, výraz filtrování lze předat do dokumentu uvnitř *iframe*.

4.4.4 Vizualizace

Nástroj Kibana poskytuje vizualizace dat v několika formách:

- prohlížení dat v uložišti Elasticsearch,
- vytvoření jedné vizualizace, která může představovat mapu, graf nebo prvek (text, tabulka, ...) obsahující statistické údaje,
- vytvoření dashboardu, což je množina libovolných vizualizací.

Prohlížení dat není možné sdílet a tudíž ani vložit do jiné stránky. Jedná se o přehled, kde je kompletní výpis všech záznamů s kompletním výpisem všech jejich hodnot.

Vizualizace je možné sdílet a vkládat do jiných stránek. Vizualizace může být například geografická mapa, sloupcový graf, koláčový graf, čárový graf, plošný graf, tabulka s údaji, obyčejný text nebo číselná hodnota. Při tvorbě

vizualizace je možné libovolně volit, které parametry nemovitostí mají být promítnuty na kterou osu, podle jakých hodnot se mají data k vizualizaci sdružovat (například graf reprezentující velikost pozemku vykreslovat v intervalech po 10 m^2) apod.

Dashboard je množina jedné či více vizualizací, proto je možné je rovněž vkládat do jiných stránek. Dashboardy jsou v rámci práce používány pro vizualizaci všech grafů, protože k jedné sadě vyfiltrovaných dat je potřeba zobrazit více grafů. Například pro prodej bytů je zobrazen koláčový graf dispozice (2+kk, 3+1, ...), čárový graf vývoje ceny, sloupcový graf velikosti bytu a další. Při aplikaci filtru jsou poté přegenerovány všechny grafy naráz.

V rámci analýzy dat a prezentace základních statistických údajů byly vytvořeny cenové mapy nemovitostí a grafy zobrazující:

- poměr:
  - typů budov,
  - podlaží, ve kterých se nemovitosti nacházejí,
  - městských částí,
  - lokalit, ve kterých se nemovitosti nacházejí,
  - dispozic nemovitostí,
  - způsobů vytápění,
  - zdrojů vody,
  - řešení odpadu,
  - typů vlastnictví,
  - stavů, v jakých nemovitosti jsou,
  - energetických tříd
  - umístění rodinných domů vzhledem k zástavbě,
- průměrnou cenu za metr čtvereční v závislosti na:
  - výměře nemovitostí,
  - typu budov,
  - umístění rodinných domů v rámci zástavby,
  - městských částech,
  - dispozicích nemovitostí,
  - zdrojích vody,
  - typech vlastnictví,
  - lokalitách, kde se nemovitosti nacházejí,

- porovnání toho, kolik nemovitostí má výtah, sklep, balkon, výbavu, garáž nebo parkovací stání,
- prvních 10 realitních kanceláří, které nabízejí nejvíce nemovitostí,
- medián ceny podle výměry nemovitostí,
- vývoj průměrné ceny za metr čtvereční v čase,
- histogram četnosti nabídky v závislosti na užitné ploše nemovitostí.

4.5 Uživatelská aplikace

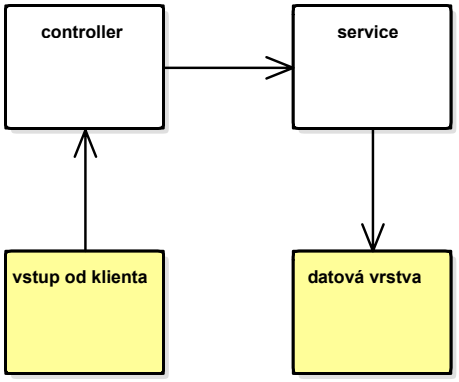
Uživatelská aplikace je externí nástavbou nástroje Kibana, zobrazuje jeho výstup v HTML elementu *iframe* a poskytuje uživateli intuitivní a přívětivé prostředí. Jedná se v podstatě o rozhraní mezi uživatelem a nástrojem Kibana. Aplikace poskytuje možnost nastavování filtrů dle různých parametrů nemovitostí a časového období, ve kterém byla inzerovaná nemovitost dostupná. Výsledky filtrování jsou aplikací předány nástroji Kibana, který na jeho základě přegeneruje vizualizace.

Přestože funkcionalita samotné aplikace je velmi elementární, protože téměř vše poskytuje Kibana, je aplikace řešena pomocí frameworku Spring MVC. Toto řešení bylo zvoleno z toho důvodu, že v budoucnu budou přidávány další nástroje analýzy (viz funkční požadavky v kapitole 2.1.1), které budou přímo součástí aplikace. Proto aplikaci tvoří několik skupin (tzv. *package*, což je soubor jedné a více tříd, které tvoří logický celek). Na obrázku 4.3 jsou zobrazeny jednotlivé skupiny včetně vzájemných závislostí.

- **controller** - zpracovává požadavky od klienta a poskytuje API rozhraní aplikace.
- **service** - poskytuje data, o které zažádá controller. Datová vrstva je použita pouze k získání hodnot potřebných k vygenerování filtrů. Všechna data k nemovitostem jsou získávána nástrojem Kibana z Elasticsearch uložiště.

4.5.1 Skripty na straně klienta

Aplikace je napsána pomocí JavaScript frameworku AngularJS. Kód je pro přehlednost dělen do složek a souborů po logických celcích a s využitím frameworku RequireJS je následně spojen do jednoho JavaScript souboru. V aplikaci je využito několik open-source rozšíření, které umožňují interaktivní posuvníky pro filtrování, boxy pro výběr více hodnot, modální okna sloužící k navigaci atd.



Obrázek 4.3: Závislost jednotlivých package v rámci uživatelské aplikace (šipky znázorňují vztah „zdrojový package používá funkcionalitu cíle“, žlutou barvou jsou odlišeny vstupy a externí zdroje)

4.5.1.1 AngularJS

AngularJS se prezentuje jako „Superheroic JavaScript MVW Framework“, kde MVW je zkratka pro Model-View-Whatever, což je parafráze pro zkratku MVC (Model-View-Controller) [66]. To přesně charakterizuje, k čemu a jak se AngularJS používá. Je navržen tak, aby oddělil model a view vrstvu od sebe tím, že mezi ní vloží cokoliv, co programátor zrovna potřebuje. Typicky se jedná o JavaScript v takové podobě, v jaké se běžně používá. Rozdíl je pouze v tom, že má přístup k datům model vrstvy a při libovolné změně v této vrstvě se automaticky aktualizuje i view vrstva [54].

Je-li v době načítání knihovny dostupný i framework jQuery, je automaticky integrován do AngularJS. V jednotlivých funkcích whatever vrstvy je poté možné používat plnou funkčnost jQuery frameworku [54].

V rámci uživatelské aplikace je AngularJS používán prakticky na veškerou funkcionalitu. Od prvního načtení stránky, kdy je zároveň načten i AngularJS, už probíhá veškerá navigace skrze jeho rozšíření ngRoute, které na základě změny adresy dokáže zavolat příslušný controller a stáhnout správnou HTML šablonu. Naznačení konfigurace ngRoute je uvedeno v kódu 4.5.

Funkce *when* reprezentuje jeden stav, ve kterém se může uživatel stránek nacházet. Je volána se dvěma parametry:

- adresa stránky,
- konkrétní nastavení pro danou stránku.

V adrese lze používat zástupné parametry, které jsou používány pro identifikaci toho, jakou kategorii si uživatel zvolil (*:type* pro byty nebo domy, *:acquisition* pro prodej nebo pronájem). Atributy *controller* a *templateUrl*

Zdrojový kód 4.5: Naznačení konfigurace ngRoute frameworku AngularJS

```
$routeProvider.
    when('/', {
        controller: 'DashboardController',
        templateUrl: 'views/dashboard.html'
    }).
    when('/mapa/:type/:acquisition', {
        controller: 'MapController',
        templateUrl: 'views/map.html',
        resolve: {
            filter: function (FilterLoader) {
                return FilterLoader();
            }
        }
    }).otherwise({
        redirectTo: '/'
    });
```

určují, jaký controller a šablona mají být načteny, a atribut *resolve* slouží k předání definované služby, například takové, která stáhne hodnoty filtrů pro danou kategorii.

4.5.1.2 RequireJS

RequireJS je JavaScript framework, který umožňuje rozdělit JavaScript kód do více souborů tak, že soubory respektují vzájemné závislosti a jsou načítány ve správném pořadí [55]. Taktéž poskytuje nástroj pro spojení všech souborů do jednoho.

Použití tohoto frameworku na stránkách poté probíhá tak, že je načtena knihovna RequireJS a do HTML atributu *data-main* je zadána adresa konfiguračního skriptu. V uživatelské aplikaci je struktura konfiguračního skriptu uvedena v kódu 4.6.

Sekce *paths* je volitelná a umožňuje definovat synonyma pro cesty ke skriptům (uvádí se bez koncovky .js), které lze poté používat kdekoliv. Sekce *shim* slouží k definování vzájemných závislostí mezi jednotlivými soubory. Ve funkci *require* je uveden seznam všech souborů, které mají být načteny. Lze použít jak cestu ke skriptům (opět bez koncovky .js), tak i dříve definovaná synonyma. Po načtení všech souborů je spuštěn kód funkce, která je předána jako poslední parametr volání *require*. V tomto konkrétním případě tedy k nastavení AngularJS.

Jednotlivé JavaScript soubory jsou před publikací na serveru spojeny do jednoho souboru a s využitím Closure compiler zkomprimovány. K tomu je

Zdrojový kód 4.6: Struktura konfiguračního skriptu RequireJS

```
require.config({
    paths: {
        angular: 'vendor/angular.min',
        angularRoute: 'vendor/angular-route.min',
        angularSanitize: 'vendor/angular-sanitize.min',
        //... a dalsi volitelne zadefinovani cest
    },
    shim: {
        angular: {
            exports: 'angular',
            deps: ['jquery']
        },
        angularMultiselect: {
            deps: ['angular']
        },
        angularSlider: {
            deps: ['angular']
        },
        //... a dalsi definice vzajemnych zavislosti jednotlivych
            souboru
    }
});

require([
    'app',
    'angular',
    'jquery',
    //nacteni vsech JavaScript souboru
],
function (app, angular) {
    //kod, ktery bude spusten po nacteni vsech souboru
    //zde se definuje nastaveni AngularJS
});
```

použito prostředí Grunt, které umožňuje spustit sadu úkolů, které mohou spustit libovolný z nainstalovaných Grunt pluginů [67]. Kromě komprimace JavaScript kódu je Grunt v aplikaci použit taktéž pro zvýšení kompatibility CSS stylů napříč webovými prohlížeči.

4.5.1.3 Filtrování dat

Jedna ze stěžejních funkcionalit uživatelské aplikace je možnost filtrovat data podle různých parametrů. Filtr je řešen jako direktiva frameworku AngularJS (kombinace JavaScript kódu a HTML šablony, které jsou vyvolány vlastním HTML tagem). K jeho použití na stránce poté stačí použít následující HTML tag:

```
<filter filter="filter"></filter>
```

Atributem *filter* je direktivě předán objekt, který obsahuje seznam všech parametrů a jejich hodnot, podle kterých mají být vytvořeny filtry.

Filtr umožňuje výběr více hodnot. A to jak v rámci jednoho parametru (kde jsou hodnoty mezi sebou spojeny logickým součtem, tedy alespoň jedna z vybraných hodnot musí být uvedena v nemovitosti), tak napříč všemi parametry (kde jsou hodnoty mezi sebou spojeny logickým součinem, tedy všechny parametry musí vyhovovat v rámci nemovitosti). Například lze vyfiltrovat všechny nemovitosti, které jsou:

- 2+kk, 2+1 nebo 3+kk
- a zároveň jsou v osobním vlastnictví
- a zároveň jsou po rekonstrukci nebo se jedná o novostavby.

Kromě toho lze nemovitosti filtrovat i podle data. Například všechny nemovitosti, které byly nabízeny 1. 1. 2016, nebo všechny nemovitosti 2+kk, které byly nabízeny od 1. 1. 2016 do 1. 2. 2016. Rovněž je možné vyfiltrovat nemovitosti podle ceny, užitné plochy, velikosti pozemku, poschodí, ve kterém se nemovitost nachází a podle dalších parametrů, které jsou u jednotlivých nemovitostí uvedeny.

Pro uživatelský vstup, na základě jakých parametrů a hodnot se má obsah filtrovat, jsou použity dva externí pluginy:

- **AngularJS MultiSelect** od vývojáře Ignatius Steven. Plugin funguje podobně jako klasický HTML select box s možností zvolit si více hodnot. Navíc je plně nastavitelný a je možné jej libovolně nastylovat. [68]
- **AngularJS Slider** od skupiny několika vývojářů. Plugin umožňuje určovat hodnotu či rozsah hodnot pomocí posuvníků. Taktéž je plně nastavitelný a lze jej libovolně nastylovat. [69]

Zdrojový kód 4.7: Získání scope frameworku AngularJS vytvořeného nástroje Kibana v iframe

```
//ziskani obsahu iframe s atributem id="kibana"
var kibana = document.getElementById("kibana").contentWindow

//ziskani scope u embed dashboardu
var scopeDashboard =
    kibana
    .angular.element(".application.ng-scope.tab-dashboard.theme-light")
    .scope()

//ziskani scope u embed vizualizace
var scopeVisualize =
    kibana
    .angular.element(".application.ng-scope.tab-visualize")
    .scope()
```

4.5.2 Vložení a ovládání prvků nástroje Kibana

Vizualizace a dashboardy nástroje Kibana lze vložit do stránky pomocí HTML tagu *iframe*, jehož zdrojovou adresu je možné získat v nástroji Kibana. Je-li dokument v *iframe* (dále jen potomek) provozován na stejné doméně, jako stránka která jej obsahuje (dále jen rodič), může JavaScript přistupovat kompletně k obsahu obou dokumentů. Je tedy možné libovolný kus kódu nebo funkcionality z rodiče předat potomkovi.

Toho se v aplikaci využívá pro filtrování. Díky tomu, že je nástroj Kibana open-source, lze vypozorovat, jak je která funkcionalita zajištěna. Kibana používá pro základní funkcionalitu JavaScript framework AngularJS. Pomocí něj Kibana získává uživatelem zadaný výraz k filtrování a spustí funkci, která zajistí aktualizaci vizualizací. Pokud je však Kibana v pozici potomka ve stránce, pole pro zadání výrazu k filtrování je znepřístupněno.

Řešením je tento výraz předat z rodiče a rovněž z pozice rodiče vyvolat akci pro aktualizaci dat v potomkovi. Ze všeho nejdříve je potřeba získat od potomka tzv. *scope* frameworku AngularJS. Jedná se objekt, který reprezentuje určitou funkcionalitu aplikace. Těchto objektů může být ve stránce několik a každý může mít na starost jinou funkcionalitu. Který *scope* zajišťuje filtrování dat je možné určit ze zdrojových kódů nástroje Kibana. JavaScript kód k jeho získání je uveden v kódu 4.7.

Předání výrazu k vyfiltrování dat a následné aplikování tohoto filtru se liší podle toho, zda je ve stránce vložený obsah vygenerovaný nástroje Kibana dashboard (množina vizualizací) nebo pouze jedna vizualizace. Jakým způsobem je potřeba postupovat je opět možné vypozorovat ze zdrojových kódů nástroje

Zdrojový kód 4.8: Aplikování filtru u dashboardu nástroje Kibana v iframe

```
//predani hodnoty filtru
scopeDashboard.state.query.query_string.query = "*"
//aktualizace dashboardu
scopeDashboard.filterResults();
```

Zdrojový kód 4.9: Aplikování filtru u vizualizace nástroje Kibana v iframe

```
//predani hodnoty filtru
scopeVisualize.$$childHead.$$childHead.state.query.query_string.query
    = "*"
//aktualizace vizualizace
scopeVisualize.$$childHead.fetch();
```

Kibana. Kód 4.8 uvádí případ pro dashboard a v kódu 4.9 je uveden případ pro jednu vizualizaci.

4.6 Souhrn

V rámci implementace byl nakonfigurován server a na něm spuštěny všechny komponenty. Na serveru je tedy zprovozněna databáze PostgreSQL, crawler, uživatelská aplikace a platforma Elasticsearch s nástrojem Kibana.

Crawler každý den aktualizuje data v databázi. Ke stahování dat volí různé časy a náhodné proxy servery. Po skončení aktualizace jsou všechny získané soubory zkomprimovány a archivovány. Výsledek každé aktualizace je reportován emailem a rovněž zapsán do logu na serveru.

Nástroj Kibana, který používá platformu Elasticsearch, vizualizuje data formou:

- cenových map,
- grafů zobrazujících poměr a vývoj jednotlivých parametrů nemovitostí,
- souhrnných statistických údajů.

Databáze je každý den převáděna z PostgresSQL do dokumentově orientovaného uložiště Elasticsearch pomocí nástroje Elsticsearch JDBC Importer.

Uživatelská aplikace poskytuje intuitivní a přívětivé prostředí pro uživatele a zobrazuje výstupy z nástroje Kibana. Umožňuje libovolné filtrování napříč

všemi parametry nemovitostí a převádí uživatelem zvolené filtrování do syntaxe nástroje Kibana. Rovněž vyvolává přegenerování dat v nástroji Kibana s ohledem na uživatelem zvolené filtrování.

Kapitola 5

Testování

Testování aplikace je pokryto manuálními i automatickými testy. Manuální testy slouží k realizaci integračních testů, výkonnostních testů a k testování z pohledu uživatele. Taktéž byly manuální testy používány během vývoje a po nasazení aplikace, kdy byla porovnávána databáze se stavem realitního serveru.

Automatickými testy je pokryta klíčová funkcionalita aplikace, tedy komponenta crawler. U této komponenty je nezbytné, aby fungovala bezchybně, jelikož jakákoliv chyba by mohla způsobit nesprávnou změnu v databázi a ovlivnění celé analýzy dat. Jelikož je crawler závislý na vstupních datech třetí strany, předpokládá se u něj, že v budoucnu může být refaktorován z důvodu libovolných změn na straně realitního serveru. Automatické unit testy tedy usnadňují ověření funkcionality po každém refaktoru.

5.1 Unit testy

Testy jsou automaticky spouštěny při každé kompilaci zdrojového kódu crawleru, tedy při každé jeho případné změně. Během návrhů testovacího scénáře byl kladen důraz na to, aby testování pokrylo co největší funkcionalitu crawleru a otestovalo ji v situacích, kdy jsou na vstupu jak validní, tak i nevalidní data. Jelikož crawler pracuje s daty třetí strany, jejichž formát se může kdykoliv změnit, je v některých situacích během testování vyvolána chyba, aby se ověřila jeho odolnost vůči nečekaným vstupům.

5.1.1 Testovací frameworky

Unit testování crawleru je realizováno pomocí spolupráce celkem 3 frameworků, protože každý z nich nabízí určitou část funkcionality:

- **JUnit** je nezbytnou součástí k tomu, aby bylo možné plnohodnotně testovat aplikace využívající Spring framework [70]. Taktéž poskytuje

nástroj pro vyhodnocování očekávaného výstupu a skutečného výstupu. Dále slouží k určení toho, které funkce jsou automatickými testy a mají být spuštěny během kompilace zdrojových kódů, a které funkce mají být spuštěny ještě před samotnými testy či bezprostředně po nich. [45]

- **Mockito** umožňuje simulovat libovolné třídy během testů (tzv. *mock*). Takto simulovaným třídám lze nastavit pravidla, jaké hodnoty mají být vráceny v případě volání některé z veřejných metod simulované třídy. Taktéž poskytuje nástroj pro sledování skutečné instance třídy (tzv. *spy*). Rozdíl oproti *mock* třídě je ten, že u *spy* objektu skutečně existuje instance třídy a během testu plně funguje. Pokud je například u *mock* třídy zavolána metoda, která nemá definovanou návratovou hodnotu v rámci testu, vůbec nic se nastane. V případě *spy* objektu by byla spuštěna skutečná metoda instance. [46]
- **PowerMock** rozšiřuje funkcionalitu frameworku Mockito. U *mock* a *spy* objektů umožňuje kontrolovat i privátní nebo statické metody. Dále poskytuje nástroj pro odchytávání objektů předávaných jiným metodám. Taktéž nabízí kontrolu vytváření nových instancí tříd. Pokud je během probíhajícího testu zavolán v aplikaci konstruktor, umožňuje PowerMock namísto vytvoření nové instance podstrčit vlastní objekt (například *mock* nebo *spy*). [47]

5.1.2 Testovací scénář

5.1.2.1 Průběh stahování dat bez výpadku

Účel: Ověření toho, zda proběhne podle očekávání stahování dat, během kterého nenastane žádný výpadek nebo chyba.

Podmínky: Odpovědi na HTTP požadavky jsou v testu nastaveny správně.

Očekávaný výsledek: Proxy server byl nastaven pouze jednou, jelikož k žádné chybě při stahování nedošlo. Funkce ke stažení dat byly volány přesně tolikrát, kolik se stahovalo souborů. Požadavky k zápisu na disk odpovídají počtu stažených souborů.

Poznámka: Počet stránek s přehledem nemovitostí, které mají být staženy, je určen automaticky na základě odpovědi, která je na HTTP požadavek vrácena testovacím prostředím. Stejně tak počet stránek s detailem nemovitostí určuje velikost pole obsahující URL adresy, které je předáno aplikaci testovacím prostředím.

5.1.2.2 Průběh stahování dat s výpadky

Účel: Ověření toho, zda proběhne podle očekávání stahování dat, během kterého nastane několik výpadků či chyb v různých fázích procesu.

Podmínky: Odpovědi na HTTP požadavky jsou v testu nastaveny správně.

Očekávaný výsledek: Proxy server byl nastaven při každém neúspěšném pokusu ke stažení dat. Celkový počet požadavků ke stažení odpovídá počtu stahovaných souborů navýšených o počet neúspěšných pokusů. Požadavky k zápisu na disk odpovídají počtu úspěšně stažených souborů.

Poznámka: Počet stránek s přehledem nemovitostí, které mají být staženy, je určen automaticky na základě odpovědi, která je na HTTP požadavek vrácena testovacím prostředím. Stejně tak počet stránek s detailem nemovitostí určuje velikost pole obsahující URL adresy, které je předáno aplikaci testovacím prostředím. Chyby jsou vyvolávány jak během stahování dat, tak i během jejich zpracování do JSON formátu. Je simulována jak dočasná, tak i naprostá nedostupnost zdroje.

5.1.2.3 Umístění stažených dat na disku

Účel: Ověření toho, zda stažená data z příslušné kategorie (byty/domy, prodej/pronájem) jsou uložena do správných složek na disku.

Očekávaný výsledek: Umístění složky, která je použita v průběhu stahování dat, odpovídá očekávanému umístění v závislosti na aktuálně stahované kategorii. Jak pro přehled se seznamem nemovitostí, tak pro detaily nemovitostí.

5.1.2.4 Získání URL detailů nemovitostí

Účel: Ověření toho, zda získání URL detailů nemovitostí z přehledu se seznamem všech nemovitostí proběhne správně.

Podmínky: Obsahy simulovaných souborů načítaných z disku jsou v testu nastaveny správně.

Očekávaný výsledek: Získané URL adresy ze souborů jsou shodné s očekávanými.

Poznámka: Obsah každého souboru, který je aplikací načítán, je dodán testovacím prostředím. Simulována jsou jak validní data, prázdný JSON, JSON zcela odlišného obsahu, tak i null na vstupu namísto souboru.

5.1.2.5 Získání obsahu pro nemovitost

Účel: Ověření toho, zda parsování JSON souboru s detailem nemovitosti proběhne správně.

Podmínky: Obsahy simulovaných souborů načítaných z disku jsou v testu nastaveny správně.

Očekávaný výsledek: Vytvořené Java objekty reprezentující nemovitost a její další části (cena, body zájmu, ...) jsou shodné s očekávanými.

Poznámka: Obsah každého souboru, který je aplikací načítán, je dodán testovacím prostředím. Simulována jsou jak validní data, prázdný JSON, tak i null na vstupu namísto souboru.

5.1.2.6 Stanovení průběhu aktualizace dat

Účel: Ověření zda aplikace správně rozhoduje o tom, které nemovitosti mají být v databázi archivovány, aktualizovány nebo do ní nově přidány.

Očekávaný výsledek: Na základě vstupních dat a stavu databáze je rozhodnutí o průběhu aktualizace správné. Nastane-li chyba při získávání URL adres ze stránek, aktualizace neproběhne vůbec.

Poznámka: Obsah databáze i stav realitního serveru je aplikaci dodán testovacím prostředím. Celkem je testováno 7 situací, které by mohly nastat. Pochopitelně situací může být v reálném provozu mnohem více, ale cílem bylo otestovat takové kombinace, které pokryjí především nejběžnější a problematické stavy. Testováno je i ověřování toho, zda byla nejnovější nemovitost dané kategorie v databázi od poslední aktualizace změněna na realitním serveru nebo ne.

Níže jsou popsány situace, které jsou testovány. Pojmem *breakpoint* je označena URL adresa nemovitosti, která je v databázi nejnovější v rámci kategorie. Používá se k určení toho, které nemovitosti mají být aktualizovány a které už ne (více v kapitole 4.3.2).

- **1. situace**
  - breakpoint je uveden na stránkách a nebyl od poslední aktualizace změněn
  - žádná z archivovaných nemovitostí v databázi se na stránkách nevyskytuje
  - na stránkách jsou nové nemovitosti, které nejsou v databázi
  - v databázi není žádná dostupná nemovitost, která by chyběla na stránkách (jinými slovy není co archivovat)
- **2. situace**
  - totéž jako 1. situace, rozdíl je pouze v tom, že breakpoint byl od poslední aktualizace změněn
- **3. situace**
  - breakpoint je uveden na stránkách a nebyl od poslední aktualizace změněn

  - na stránkách se vyskytují nemovitosti, které jsou v databázi vedeny jako archivované
  - na stránkách nejsou žádné nové nemovitosti, které by nebyly v databázi
  - v databázi jsou dostupné nemovitosti, které již nejsou na stránkách

- **4. situace**
  - breakpoint na stránkách chybí
  - na stránkách se vyskytují nemovitosti, které jsou v databázi vedeny jako archivované
  - na stránkách se vyskytují nemovitosti, které jsou v databázi vedeny jako archivované
  - v databázi jsou dostupné nemovitosti, které již nejsou na stránkách

- **5. situace**
  - totéž jako 4. situace, rozdíl je pouze v tom, že se z databáze nepodařilo načíst informaci o nejnovější nemovitosti v rámci dané kategorie

- **6. situace**
  - záznam o nejnovější nemovitosti v rámci dané kategorie je načten v pořádku
  - k analýze je odeslán prázdný seznam URL adres, jelikož se ho nepodařilo načíst

- **7. situace**
  - totéž jako 6. situace, rozdíl je pouze v tom, že namísto prázdného seznamu je odeslán null, jelikož při získávání URL adres došlo k chybě

5.1.2.7 Zpracování dat v průběhu aktualizace

Účel: Ověření toho, zda na základě stanovení průběhu aktualizace jsou příslušné nemovitosti opravdu aktualizovány, nově přidány nebo archivovány.

Očekávaný výsledek: Počet volání funkce pro aktualizaci dat odpovídá počtu nemovitostí k tomu určených, včetně jejich URL adres. Stejně tak pro funkce sloužící k archivaci nebo přidávání nemovitostí.

Poznámka: Obsah databáze i stav realitního serveru je aplikaci dodán testovacím prostředím. Ověřuje se, které funkce byly kolikrát volány a s jakými parametry.

5.1.2.8 Průběh aktualizace dat

Účel: Ověření toho, zda jsou během aktualizace data stažena, aktualizována a zda je odeslán report.

Očekávaný výsledek: Byl stažen přehled se seznamem všech nemovitostí, stanoven průběh aktualizace a staženy detaily pouze pro nemovitosti změněné od poslední aktualizace. Příslušné nemovitosti byly aktualizovány, přidány nebo archivovány. Byl uložen obsah nejnovější nemovitosti pro účely příští aktualizace. Byl vytvořen report a nastaven jeho obsah, taktéž byly zapsány informace do statistického souboru. Na závěr proběhla komprese stažených dat.

Poznámka: Obsah databáze i stav realitního serveru je aplikaci dodán testovacím prostředím. Ověřuje se, které funkce byly kolikrát volány a s jakými parametry.

5.1.2.9 Reportování

Účel: Ověření toho, zda pravidelné reporty zapisují předané informace na disk a reportují je emailem.

Podmínky: Nastavení emailu (server, port, odesílatel, adresát, ...) je uloženo v externím souboru.

Očekávaný výsledek: Obsah emailu je vytvářen postupným přidáváním dalšího obsahu, v předmětu emailu se vyskytuje informace o tom, zda aktualizace proběhla v pořádku nebo s chybou, email je odeslán pouze jednou, ze správné adresy a na zadanou adresu. Jsou vytvořeny příslušné logovací soubory a na závěr jsou uzavřeny pro zápis.

Poznámka: Email je ověřován v momentě, kdy je předán k odeslání, ale k jeho skutečnému odeslání během testu nedojde.

5.2 Integrační testy

Integrační testy byly prováděny manuálně. Pomocí těchto testů bylo ověřeno, zda jednotlivé komponenty aplikace spolupracují správně. Konkrétně bylo ověřeno, jestli crawler ukládá správné hodnoty do databáze a zda se tyto změny projevují v uživatelské aplikaci a v uložišti platformy Elasticsearch.

Stav databáze byl kontrolován několikrát namátkou, kdy byly porovnány URL adresy nemovitostí z databáze, které byly vedeny jako dostupné, se skutečným stavem na stránkách a staženými daty z aktualizace. Při porovnání se staženými daty byla shoda 100%. U kontroly se skutečným stavem na stránkách se nemovitosti drobně odlišovaly od stavu v databázi. To bylo způsobeno tím, že se během doby od aktualizování databáze skutečný stav na realitním

serveru změnil. Nicméně odlišnosti byly minimální a v řádu jednotek nemovitostí, lze tedy předpokládat, že zápis nemovitostí do databáze funguje správně.

Taktéž byly namátkově kontrolovány některé nemovitosti, jestli jejich údaje (cena, dispozice, městská část, body zájmů, ...) odpovídají reálnému stavu na stránkách realitního serveru. Během těchto testů nebyla nalezena žádná nekonzistence. Podle denních reportů nenastal při každodenních aktualizacích žádný problém, například typu odkazování cizího klíče na neexistující prvek, přidávání již existujícího unikátního pole apod. Lze říci, že i v tomto směru funguje zpracování dat crawlerem do databáze bezchybně.

Data uživatelské aplikace jsou získávány přímým dotazem do databáze v době, kdy je stránka načítána. Není proto pravděpodobné, že by se údaje odlišovaly od stavu v databázi. Nicméně bylo potřeba ověřit, zda se získané hodnoty z databáze správně reprezentují na frontendu uživatelské aplikace. Byly porovnány hodnoty uvedené ve filtrování se stavem v databázi. Ani v tomto případě nebyla nalezena žádná nekonzistence.

Do uložiště platformy Elasticsearch jsou data z PostreSQL databáze převáděna každý den. Několikrát byla provedena kontrola, zda stav v uložišti odpovídá stavu v databázi. Při převodu jsou data lehce modifikována pro potřeby Elasticsearch a nástroje Kibana, více je popsáno v kapitole 4.4. I s ohledem na tyto změny byl stav vždy totožný a odpovídal grafickým výstupům nástroje Kibana na frontendu aplikace.

Jediný problém nastává s podporou prohlížeče Internet Explorer a Edge. Tyto prohlížeče mají maximální limit 2083 znaků na délku URL adresy. Tuto délku Kibana v případě vkládání dashboardu do *iframe* překračuje [71]. Z tohoto důvodu není obsah v Internet Explorer a Edge načten.

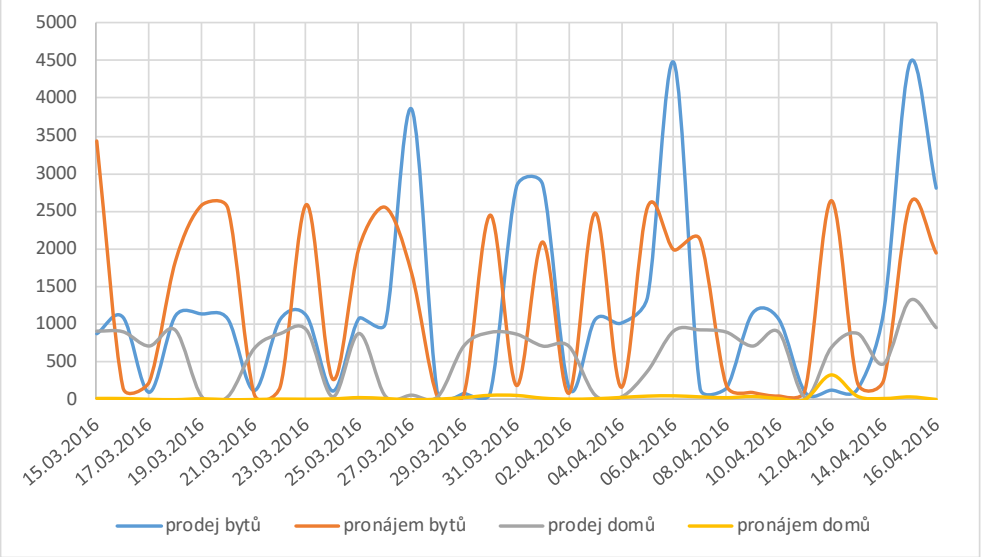
V rámci integračních manuálních testů jinak nebyl nalezen žádný problém a byla potvrzena bezchybná funkčnost aplikace. Při dalším vývoji aplikace a jejím případném rozšiřování už bude pravděpodobně potřeba zautomatizovat i integrační testy, nicméně v současném stavu to není nezbytné.

5.3 Výkonnostní testy

Pro získání představy o tom, kolik uživatelů je schopna aplikace obsloužit naráz nebo kolik dat je možné crawlerem stáhnout a zapsat do databáze za určitý čas, byly realizovány základní výkonnostní testy.

5.3.1 Crawler

Podle statistik denních reportů v období od 15. 3. 2016 do 16. 4. 2016 je každý den staženo průměrně 3 118 stránek s detaily nemovitostí. Na grafu 5.1 je zobrazen poměr jednotlivých kategorií, přesné hodnoty jsou uvedeny v tabulce 5.1. Z grafu je patrné, kdy nastala aktualizace všech nemovitostí v dané kategorii, způsobená odstranění inzerátu z nabídky, který sloužil jako



Obrázek 5.1: Počet stažených detailů nemovitostí během každodenní aktualizace

breakpoint pro aktualizaci. U prodeje bytů nastala tato situace celkem čtyřikrát za měsíc. U pronájmů bytů celkem jedenáctkrát (což svědčí o větším zájmu o pronájmy, tedy i rychlejší obměnou jejich nabídek).

Stránky s přehledem nemovitostí musí být staženy vždy všechny, jejich počet se ale příliš nemění. Pohybuje se v závislosti na počtu nabízených nemovitostí v rozmezí od 135 do 175, průměrně 160.

Průměrná velikost jedné stránky s přehledem nemovitostí je 76kB. U stránky reprezentující detail nemovitosti to je 14kB. Celkem se tedy každý den stahuje průměrně 54,5MB dat. Maximálně bylo za jeden den staženo 149,3MB dat, naopak nejméně bylo staženo 15,4MB dat.

Rychlost stahování se u proxy serverů liší a kolísá podle denní doby. V ideálním případě se rychlost stahování pohybuje okolo 1,5 Mb/s, což znamená, že průměrný denní objem dat je stažen za necelých 5 minut. Nějaký čas navíc vyžaduje nastavení a ověření funkčnosti proxy serveru a taky doba odezvy na odeslaný požadavek. Pro jistotu je možné počítat s trojnásobkem doby downloadu. V ideálním případě jsou tedy data stažena za 15 minut.

V některých případech je proxy server přetížen a poskytuje například rychlost stahování pouze 50kb/s. V takovém případě by čistý čas stahování byl 2,5 hodiny. Nicméně je potřeba brát v úvahu, že před zahájením stahování každé kategorie (byty/domy, prodej/pronájem) je nastaven proxy server náhodně, takže by takto pomalé připojení pravděpodobně nebylo použito pro stažení všech stránek.

Aby nedocházelo k situacím, kdy pomalé proxy připojení narušuje funkčnost aplikace, je zavedeno opatření, že po 10 sekundách bez odezvy serveru

Tabulka 5.1: Počet stažených detailů nemovitostí během každodenní aktualizace (hodnoty jsou uváděny ve formátu *nové* + *aktualizované* nemovitosti)

	byty		domy	
	prodej	pronájem	prodej	pronájem
15. 3. 2016	69 + 809	285 + 3141	53 + 861	5 + 14
16. 3. 2016	42 + 1055	147 + 13	22 + 884	2 + 16
17. 3. 2016	94 + 8	196 + 37	14 + 702	7 + 1
18. 3. 2016	80 + 1036	181 + 1653	15 + 911	2 +1
19. 3. 2016	78 + 1063	65 + 2511	14 + 28	5 + 10
20. 3. 2016	18 + 1054	35 + 2497	5 + 37	1 + 2
21. 3. 2016	84 + 41	31 + 43	10 + 671	2 + 3
22. 3. 2016	46 + 1026	177 + 1	11 + 870	5 + 6
23. 3. 2016	87 + 1026	126 + 2458	20 + 908	7 + 2
24. 3. 2016	83 + 35	156 + 113	12 + 27	7 + 6
25. 3. 2016	90 + 993	204 + 1810	15 + 868	8 + 22
26. 3. 2016	19 + 993	40 + 2507	0 + 48	2 + 16
27. 3. 2016	8 + 3851	18 + 1662	9 + 51	1 + 1
28. 3. 2016	12 + 51	12 + 13	6 + 34	1 + 9
29. 3. 2016	47 + 40	65 + 10	15 + 704	6 + 24
30. 3. 2016	74 + 20	113 + 2331	5 + 892	10 + 50
31. 3. 2016	111 + 2715	151 + 37	9 + 865	18 + 40
1. 4. 2016	103 + 2752	259 + 1829	14 + 701	6 + 17
2. 4. 2016	121 + 42	85 + 0	10 + 702	8 + 2
3. 4. 2016	21 + 1045	42 + 2430	9 + 51	5 + 10
4. 4. 2016	17 + 1003	121 + 43	5 + 31	1 + 31
5. 4. 2016	162 + 1209	101 + 2451	13 + 369	8 + 41
6. 4. 2016	57 + 4418	177 + 1807	16 + 902	12 + 41
7. 4. 2016	124 + 29	292 + 1826	42 + 889	5 + 34
8. 4. 2016	111 + 44	148 + 43	16 + 885	7 + 21
9. 4. 2016	100 + 1051	84 + 11	24 + 691	14 + 30
10. 4. 2016	32 + 1038	44 + 4	7 + 893	2 + 16
11. 4. 2016	62 + 41	122 + 4	4 + 30	2 + 4
12. 4. 2016	111 + 19	86 + 2549	16 + 678	20 + 304
13. 4. 2016	116 + 27	168 + 54	32 + 852	9 + 38
14. 4. 2016	148 + 1028	212 + 49	45 + 434	6 + 12
15. 4. 2016	59 + 4414	103 + 2491	16 + 1309	8 + 31
16. 4. 2016	94 + 2715	135 + 1807	10 + 951	3 + 3

dojde k automatické změněně proxy serveru a pokus je opakován. Takže i v případě, kdy by byla například přetížena polovina proxy serverů ze seznamu, lze odhadovat, že doba stažení by se pohybovala kolem 2 hodin. A to včetně nastavování nových proxy serverů, čekání na odezvu odeslaného požadavku apod. Pokud by tato situace nastala v kombinaci s tím, že by byl stahován větší objem dat, například 150MB, jednalo by se přibližně o 6 hodin.

Analýza logů o průběhu aktualizace dat toto potvrzuje. Nejdéle trvalo stahování téměř 3 hodiny. Došlo k tomu v době, kdy seznam proxy serverů byl dva měsíce neaktualizovaný a bylo v daný den staženo celkem 114MB dat.

Zpracování dat a jejich uložení do databáze trvá pro 3 000 nemovitostí přibližně 20 minut. Ačkoliv se může zdát, že doba je nepřiměřená počtu záznamů, je potřeba brát v úvahu, že před každým zápisem nemovitosti do databáze probíhá několik SQL dotazů. Zápis může probíhat i do více než 15 databázových tabulek, kde je u každé napřed zkontrolováno, zda již hodnota není v tabulce náhodou uvedena. Navíc u tabulek, které obsahují desetitisíce záznamů, je použito indexování, což zápis zpomaluje o nutnost aktualizovat indexy.

Aktualizace (a tedy i stahování dat) je spouštěna po částech ve čtyřech fázích, pro každou kategorii zvlášť. Díky tomu nedochází při zpracování hodnot do databáze k blokaci stahování dat jiné kategorie a naopak.

Současný stav je tedy bez potíží udržitelný jedním crawlerem. Pokud by byl sběr dat rozšířen na celou Českou republiku, celkový počet nemovitostí by vzrostl na šestinásobek. V březnu 2016 na realitním serveru sreality.cz tvořila nabídka pro Prahu z celkového počtu nabídek pro celou republiku následující podíl:

- prodej bytů - 24%,
- pronájem bytů - 36%,
- prodej domů - 4%,
- pronájem domů - 45%.

V takovém případě by bylo zapotřebí mít crawlerů několik. Aby byla aplikace schopna zpracovat všechna data i v případě přetížení proxy serverů a nečekaného množství stahovaných dat, mělo by postačit tolik crawlerů, kolikrát vzrostl objem dat, celkem tedy 6. Úspornějším řešením by mohlo být centralizování rozhodnutí o průběhu aktualizace. Například by jeden hlavní crawler stáhl informace o tom, kolik je potřeba aktualizovat nemovitostí, a práci by rovnoměrně rozdělil mezi více crawlerů. Ty by se mohly hlásit hlavnímu crawleru v momentě, kdy stahování dokončily a na základě toho by mohl hlavní crawler práci ještě dále přerozdělovat. V takovém případě by pravděpodobně stačilo méně crawlerů, ale bylo by nezbytné zmíněné řešení otestovat.

5.3.2 Uživatelská aplikace

Aplikace byla testována několika různými nástroji, aby bylo možné určit, jakou zátěž server, na kterém je spuštěna uživatelská aplikace, snese a za jakou dobu se stáhne kompletní obsah stránek včetně všech zdrojů, které obsahuje. Konkrétně byly použity následující nástroje:

- **Pingdom Website Speed Test** pro identifikaci všech zdrojů, které jsou spolu se stránkou staženy, a pro určení doby, za kterou jsou staženy.
- **Apache Benchmark** pro otestování maximální zátěže, během které ještě server dokáže vrátit uživateli data v přijatelném čase.
- **Google Chrome** pro změření doby, za kterou je vykreslena stránka od momentu, kdy je odeslán první požadavek na server.

Z analýzy nástroje Pingdom Website Speed Test vyplynulo, že při načítání stránky je celkem odesláno 27 HTTP požadavků a uživateli je vráceno celkem 1,7MB dat. Z tohoto objemu dat připadá přibližně 0,2MB na samotnou aplikaci, zbylých 1,5MB na nástroj Kibana. Načtení tohoto objemu dat trvá přibližně 1,6s (jedná se o průměr vycházející z 50 nezávislých spuštění testu v různé denní doby). Přibližně 80% veškerého času vyžaduje nástroj Kibana.

Jelikož nástroj Apache Benchmark nezohledňuje asynchronně nahrávaný obsah, nebylo potřeba testovat samotnou uživatelskou aplikaci, která je čistě statická, jelikož veškerý obsah poskytuje nástroj Kibana. Pro účely testování zátěže nástroje Kibana byl vytvořen jednoduchý skript, který zasílal požadavek do uložiště Elasticsearch, a výsledek tohoto dotazu vypsal do stránky. Požadavek, který byl opakovaně odesílán do uložiště, byl totožný s tím, který je běžně používán nástrojem Kibana k získávání dat pro dashboard prodeje bytů. Kategorie prodeje bytů byla zvolena z toho důvodu, že obsahuje nejvíce položek. Lze proto předpokládat, že zpracování takového požadavku bude oproti jiným požadavkům trvat nejdelší dobu a bude výpočetně nejnáročnější.

Během testování zátěže skriptu, který opakovaně odesílal požadavek do uložiště Elasticsearch a vracel jeho výsledky, nebyla překročena doba odezvy 300ms ani při spuštění testu s parametry pro 1000 paralelních požadavků a o celkovém počtu 5000 požadavků. Další zátěžové testy v tomto směru nebyly realizovány, jelikož nelze předpokládat, že by server během následujících několika let musel zvládat větší zátěž.

Na závěr bylo pomocí aplikace Google Chrome a jejího nástroje Timeline otestováno, za jakou dobu je vykreslena stránka od momentu, kdy byl odeslán první požadavek. Z celkem 50 pokusů, které byly prováděny v různou denní dobu, vycházela průměrná doba renderování kompletního obsahu, jehož součástí je 20 interaktivních grafů, necelých 8 sekund. Což je doba odpovídající povaze aplikace, která slouží k analýze velkého množství posbíraných dat.

Kapitola 6

Další rozvoj aplikace

Výstupem práce je funkční aplikace, která stahuje informace z vybraného realitního serveru a poskytuje nástroj pro základní analýzu dat. Během vývoje aplikace vzniklo mnoho nápadů, jak by bylo možné aplikaci dodatečně vylepšit a rozšířit.

6.1 Rozšíření funkcionality

Data v databázi, která jsou vytvářena na základě každodenních aktualizací, je možné analyzovat mnoha dalšími způsoby. Současný stav poskytuje pouze základní statistické údaje, jejichž kompletní seznam je uveden v kapitole 4.4.4. Nicméně z analýzy existujících řešení vyplynula celá řada dalších možností analýzy dat (podrobnosti jsou uvedeny v kapitole 2.1.1).

Zajímavý pohled na praxi realitních makléřů a společností by mohla přinést analýza vývoje ceny nemovitosti v závislosti na době, po kterou je inzerát zveřejněn. Rovněž by bylo zajímavé sledovat dobu, po kterou je v průměru zveřejněn inzerát s ohledem na lokalitu a parametry nemovitosti.

Pro účely investic by mohly být užitečné nástroje, které by z nashromážděných dat dokázaly určit, za jakou dobu by se přibližně vrátila investice do nemovitosti jejím pronájmem. Popřípadě v jaké lokalitě nejstrměji rostly ceny a kde naopak nejstrměji klesaly.

Jistě by se objevily i další možnosti a nápady, kterými by bylo možné aplikaci obohatit. Nejlepším způsobem pro jejich sběr by bylo veřejné spuštění aplikace s formulářem pro uživatelskou zpětnou vazbu.

6.2 Spolupráce s realitními servery a katastrálním úřadem

V kapitole 5.3.1 je podrobná analýza toho, kolik crawlerů by bylo potřeba k tomu, aby bylo možné rozšířit databázi o všechny další kraje České repub-

liky. Provozování většího množství crawlerů by mohlo být výhodnější nahradit spoluprací s realitními servery. Například domluvou na pravidelném poskytování omezených a určitých informací z jejich databáze. Ať už za určitý paušální poplatek nebo za poskytnutou protislužbu ve formě výsledků analýzy dat.

Jak již bylo popsáno v kapitole 1.5, katastr nemovitostí vyžaduje za každé strojové zpracování informací poplatek. Informace o tom, za kolik se konkrétní nemovitost skutečně prodala a za jakou dobu od zveřejnění inzerátu ze stránek se prodala, by mohla být pro úspěch aplikace klíčová. Z tohoto důvodu by bylo vhodné se pokusit o spolupráci s katastrálním úřadem, ať už za pravidelný poplatek nebo za poskytnuté informace vyplývající z analýzy dat.

6.3 Více nezávislých crawlerů

V případě, že by se nepodařilo dohodnout s realitními servery a stahování dat by muselo pokračovat pomocí crawleru, bylo by nezbytné jejich počet rozšířit. Jak již bylo odhadnuto v kapitole 5.3.1, pro stahování dat pro celou Českou republiku by bylo potřeba přibližně 6 crawlerů. V případě, že by bylo potřeba databázi rozšířit i o další inzertní servery, jejich počet by ještě vzrostl.

K urychlení procesu aktualizace bude pravděpodobně potřeba navrhnout například centrálně řízený způsob aktualizace, aby se zátěž rozložila mezi jednotlivé crawlery rovnoměrně. Další variantou by mohlo být používání soukromých proxy serverů, které by netrpěly přetížením a tím způsobeným zpomalením připojení.

Dalším způsobem, jak omezit množství stahovaných dat, by mohla být snaha o vylepšení aktualizace v případě, kdy byla z nabídky stažena nemovitost, která sloužila jako breakpoint při stahování detailů o nemovitostech (podrobnosti jsou uvedeny v kapitole 4.3.2). Například tak, že by namísto jedné nemovitosti bylo uchováno nemovitostí hned několik, třeba 5. Tím by se zvýšila pravděpodobnost, že by v případě přeskočení jednoho breakpointu byl použit jiný. Jak je uvedeno v kapitole 5.3.1, například u bytů dochází k přeskočení breakpointu téměř každý druhý den.

6.4 Sběr informací o pozemcích

Momentální stav je takový, že jsou stahovány veškeré informace o prodejích a pronájmech bytů a domů. Vzhledem k tomu, jakým způsobem vznikají nové developerské projekty v okolí velkých měst, by mohlo být zajímavé sledovat také ceny pozemků. V kombinaci s tím by bylo pravděpodobně nezbytné získávat i informace z územních plánů, aby bylo možné správně identifikovat, jak se změny v územním plánu projeví na ceně pozemku.

6.5 Modularita analyzátorů dat

V současném řešení aplikace je generován veškerý výstup analýzy dat nástrojem Kibana. Tento nástroj poskytuje vše, co je v rámci práce potřeba, ale do budoucna by pravděpodobně bylo vhodnější dělat analýzu pomocí vlastní aplikace. Jelikož je Kibana externím řešením, které nabízí nástroje k analýze téměř libovolných dat, může nastat situace, kdy pomocí ní nebude možné realizovat velmi specifické požadavky, které se u nemovitostí mohou objevit. Některé takové situace již byly v rámci práce řešeny (viz kapitola 4.4.2.1).

Jednotlivé analýzy by mohly být řešeny pomocí samostatných modulů, které by mohly být přidávány i za běhu aplikace. Do budoucna by bylo možné rozšířit tuto myšlenku i o možnost přidávání analyzátorů třetích stran, které by měly k dispozici veřejné API rozhraní aplikace.

6.6 API rozhraní

Jednou z možností, jak rozšířit povědomí o aplikaci, by mohlo být naprogramování veřejného API rozhraní. Díky tomu by bylo umožněno všem vývojářům přistupovat k metodám, které by vracely požadované údaje, jako jsou například:

- průměrná cena bytů 2+kk na Praze 6,
- medián ceny za metr čtvereční v Brně,
- 3 kraje, kde jsou nejdražší pozemky,
- počet prodaných bytů v určitém časovém úseku,
- a další...

Uplatnění takového API rozhraní by mohlo být například na stránkách realitních kanceláří a společností nebo na stránkách inzertních serverů, které by chtěly uživatelům poskytnout určitou informaci o lokalitě, kterou si případný zájemce zrovna prohlíží. Rovněž by API rozhraní mohlo inspirovat příznivce Linked Open Data, kteří by v konečném důsledku mohli analyzovat vliv demografických a sociálních údajů na prodej nemovitostí a jejich cenu.

Závěr

Cílem práce bylo navrhnout, implementovat a otestovat nástroj pro monitoring cen realit v Praze, který automaticky sbírá data o nemovitostech z vybraného serveru a výsledky následné analýzy dat prezentuje v uživatelsky přívětivé podobě.

Tento cíl se podařilo úspěšně splnit nad rámec požadavků, které byly na projekt kladeny. Výsledkem je webová aplikace, která se skládá z několika nezávislých komponent, které používají společnou databázi.

Crawler, který získává data z vybraného inzertního serveru, je stahuje v několika etapách v náhodné časy a za použití vlastního seznamu proxy serverů. Stažené informace ukládá do databáze, jejíž obsah aktualizuje každý den. Výsledky průběhu každodenní aktualizace jsou ukládány na disk na serveru a rovněž reportovány na zvolenou emailovou adresu. Součástí emailového reportu je i příloha s logem obsahujícím výpis všech chyb, které během aktualizace nastaly. Jelikož je crawler klíčovou komponentou, jejíž bezchybná funkčnost je pro další analýzu dat nezbytná, obsahuje crawler sadu automatických unit testů, které prověřují veškerou jeho funkcionalitu.

K analyzování dat slouží open-source nástroj Kibana, který funguje na platformě Elasticsearch. Kibana prezentuje crawlerem nastřádaná data ve formě cenových map a grafů. Databáze, do které jsou data crawlerem ukládána, je každý den převáděna do uložiště Elasticsearch, ze kterého získává údaje Kibana.

Uživatelská aplikace zobrazuje výstupy nástroje Kibana a poskytuje k nim navržené uživatelsky přívětivé prostředí. Uživatel si může zvolit, zda chce zobrazit údaje pro prodej nebo pronájem bytů či domů. Rovněž má možnost data libovolně filtrovat a zobrazit si tak údaje libovolně zvolené podmnožiny dat.

Aplikace byla otestována v rámci komponent unit testy, stejně tak komplexně pomocí manuálních integračních testů. Výkonnostní testy ukázaly, že množství crawlerů bude potřeba rozšířit, pokud se bude rozšiřovat monitoring i na další kraje v České republice. Výkonnostní testy rovněž ukázaly, že uživatelská aplikace je navržena dobře a že bez větších potíží zvládne i 1 000 pa-

ralelních uživatelů.

Během vývoje aplikace se ukázalo, že možností, které zpracovávané téma poskytuje, je obrovské množství. Vývoj aplikace je tedy v začátcích a bylo by vhodné ji rozšířit například o pokročilé nástroje analýzy (vývoj ceny v závislosti na stáří inzerátu, hledání levných nemovitostí v drahých lokalitách, nástroj pro odhad návratnosti investice a další). Taktéž by bylo vhodné zpracovávat i údaje z katastrálního úřadu a porovnávat skutečnou prodejnost a prodejní ceny s inzerovanými nabídkami.

Z analýzy existujících řešení vyplývá, že takovýchto projektů, které se zaměřují na statistické zpracování dat inzertních serverů v České republice, není mnoho. Proto by další rozvoj projektu mohl být žádoucí a přínosný.

Literatura

[1] infoexe s.r.o.: *infoexe | monitoring*. [cit. 2015-12-27]. Dostupné z: `http://www.exe.cz/`

[2] OctopusPro s.r.o.: *Služby - OctopusPro*. [cit. 2015-12-27]. Dostupné z: `http://www.octopuspro.cz/sluzby/`

[3] ADOL Group s.r.o.: *ADOL^{TM} Monitoring Realit - monitoring inzerce*. [cit. 2015-12-27]. Dostupné z: `http://www.adol.cz/monitoring-inzerce.php`

[4] JLR: *Online Services (Quebec) - JLR Land Title Solutions*. [cit. 2015-12-27]. Dostupné z: `https://www.jlr.ca/online-services.aspx`

[5] PropertyRadar: *PropertyRadar - The leads, data, analysis, and alerts you need*. [cit. 2015-12-27]. Dostupné z: `https://www.propertyradar.com/`

[6] APM PriceFinder: *APM PriceFinder | Property Data & Analytics | Australian Property Monitors | Property Data Solutions*. [cit. 2015-12-27]. Dostupné z: `http://www.apmpricefinder.com.au/`

[7] Realman s.r.o.: *REALMAN | Online realitní systém - Exporty*. [cit. 2015-10-21]. Dostupné z: `http://www.realman.cz/export-na-realitni-servery`

[8] MAFRA a.s.: *Reality.iDNES.cz - Reality a nemovitosti z celé ČR*. [cit. 2015-10-21]. Dostupné z: `http://reality.idnes.cz/`

[9] HyperMedia, a.s.: *HyperReality.cz*. [cit. 2015-10-21]. Dostupné z: `http://www.hyperreality.cz/`

[10] DALTEN media s.r.o.: *Reality, nemovitosti z celé ČR, prodej bytů, pronájem bytů*. [cit. 2015-10-21]. Dostupné z: `http://realitymix.centrum.cz/`

[11] Seznam.cz, a.s.: *Sreality.cz • reality a nemovitosti z celé ČR*. [cit. 2015-10-21]. Dostupné z: http://www.sreality.cz/

[12] Google: *Google Maps APIs | Google Developers*. [cit. 2016-03-02]. Dostupné z: https://developers.google.com/maps/

[13] Google: *Layers | Google Maps JavaScript API | Google Developers*. [cit. 2016-03-02]. Dostupné z: https://developers.google.com/maps/documentation/javascript/layers

[14] Google: *Heatmap Layer | Google Maps JavaScript API | Google Developers*. [cit. 2016-03-02]. Dostupné z: https://developers.google.com/maps/documentation/javascript/heatmaplayer

[15] Google: *Shapes | Google Maps Android API | Google Developers*. [cit. 2016-03-02]. Dostupné z: https://developers.google.com/maps/documentation/android-api/shapes

[16] Microsoft: *Creating Heat Maps with Bing Maps and Dynamics CRM | Microsoft Dynamics CRM Team Blog*. [cit. 2016-03-02]. Dostupné z: https://blogs.msdn.microsoft.com/crm/2012/10/29/creating-heat-maps-with-bing-maps-and-dynamics-crm/

[17] OpenStreetMap Foundation: *Heatmap with OSM - OSM Help*. [cit. 2016-03-02]. Dostupné z: https://help.openstreetmap.org/questions/16537/heatmap-with-osm

[18] Seznam.cz, a.s.: *JsDoc Reference - Seznam tříd*. [cit. 2016-03-02]. Dostupné z: https://api.mapy.cz/doc/index.html

[19] Splunk Inc.: *Operational Intelligence, Log Management, Application Management, Enterprise Security and Compliance | Splunk*. [cit. 2016-03-02]. Dostupné z: http://www.splunk.com/

[20] Loom Systems: *Loom Systems | The First Artificially Intelligent Data Scientist*. [cit. 2016-03-02]. Dostupné z: http://www.loomsystems.com/

[21] Sumo Logic: *Continuous Intelligence, Log Management & Analysis: Sumo Logic*. [cit. 2016-03-02]. Dostupné z: https://www.sumologic.com/

[22] Torkel Ödegaard & Coding Instinct AB: *Grafana - Graphite and InfluxDB Dashboard and graph composer*. [cit. 2016-03-02]. Dostupné z: http://grafana.org/

[23] Elasticsearch: *Kibana: Explore, Visualize, Discover Data | Elastic*. [cit. 2016-03-02]. Dostupné z: https://www.elastic.co/products/kibana

[24] Elasticsearch: *Elasticsearch | Elastic*. [cit. 2016-03-02]. Dostupné z: https://www.elastic.co/products/elasticsearch

[25] Jörg Prante: *GitHub - jprante/elasticsearch-jdbc: JDBC importer for Elasticsearch*. [cit. 2016-03-02]. Dostupné z: https://github.com/jprante/elasticsearch-jdbc

[26] Ministerstvo vnitra: *Portál veřejné správy*. [cit. 2016-01-03]. Dostupné z: http://portal.gov.cz/app/zakony/download?idBiblio=80383&nr=256~2F2013~20Sb.&ft=txt

[27] ČÚZK: *ČÚZK - Poskytování údajů z KN*. [cit. 2016-01-03]. Dostupné z: http://www.cuzk.cz/Katastr-nemovitosti/Poskytovani-udaju-z-KN/Poskytovani-udaju-z-KN.aspx

[28] ČÚZK: *Nahlížení do katastru nemovitostí*. [cit. 2016-01-03]. Dostupné z: http://nahlizenidokn.cuzk.cz/

[29] ČÚZK: *ČÚZK - Zřízení účtu dálkového přístupu*. [cit. 2016-01-03]. Dostupné z: http://www.cuzk.cz/Katastr-nemovitosti/Poskytovani-udaju-z-KN/Dalkovy-pristup/Zrizeni-uctu-dalkoveho-pristupu.aspx

[30] ČÚZK: *ČÚZK - účtování výstupů z KN poskytovaných DP A WSDP*. [cit. 2016-01-03]. Dostupné z: http://www.cuzk.cz/Katastr-nemovitosti/Poskytovani-udaju-z-KN/Dalkovy-pristup/Uctovani-vystupu-z-KN-poskytovanych-DP-A-WSDP.aspx

[31] ČÚZK: *ČÚZK - Webové služby dálkového přístupu*. [cit. 2016-01-03]. Dostupné z: http://www.cuzk.cz/Katastr-nemovitosti/Poskytovani-udaju-z-KN/Dalkovy-pristup/Webove-sluzby-dalkoveho-pristupu.aspx

[32] Schmidt, K.: *High Availability and Disaster Recovery*. Springer, 2006, ISBN 978-3-540-24460-8.

[33] Postgres-XC: *GitHub - postgres-x2/postgres-x2: Master Postgres-XC source repository*. [cit. 2016-03-11]. Dostupné z: https://github.com/postgres-x2/postgres-x2

[34] MySQL: *MySQL 5.7 Reference Manual :: 18.6.10 MySQL Cluster Replication: Multi-Master and Circular Replication*. [cit. 2016-03-11]. Dostupné z: https://dev.mysql.com/doc/refman/5.7/en/mysql-cluster-replication-multi-master.html

[35] Elasticsearch: *Shield | Elastic*. [cit. 2016-04-30]. Dostupné z: https://www.elastic.co/products/shield

[36] Elasticsearch: *Kibana User Guide [4.3]*. [cit. 2015-12-30]. Dostupné z: https://www.elastic.co/guide/en/kibana/current/index.html

[37] Elasticsearch: *Multi-select (OR) dashboard filtering - Issue #3693 - elastic/kibana - GitHub*. [cit. 2015-12-30]. Dostupné z: `https://github.com/elastic/kibana/issues/3693`

[38] Seznam.cz, a.s.: *Sreality.cz RSS*. [cit. 2015-10-21]. Dostupné z: `http://www.sreality.cz/api/cs/v1/estates/rss`

[39] Netcraft Ltd.: *March 2016 Web Server Survey | Netcraft*. [cit. 2016-03-19]. Dostupné z: `http://news.netcraft.com/archives/2016/03/18/march-2016-web-server-survey.html`

[40] Java-source.net: *Open Source Web Frameworks in Java*. [cit. 2016-03-19]. Dostupné z: `http://java-source.net/open-source/web-frameworks`

[41] TIOBE software BV: *TIOBE Index | Tiobe - The Software Quality Company*. [cit. 2016-03-19]. Dostupné z: `http://www.tiobe.com/tiobe_index?page=index`

[42] Pivotal Software, Inc.: *Spring Framework*. [cit. 2016-03-19]. Dostupné z: `http://projects.spring.io/spring-framework/`

[43] Walls, C.: *Spring in Action*. Manning Publications Co., třetí vydání, 2011, ISBN 978-1-935182-35-1.

[44] Pivotal Software, Inc.: *Spring Boot*. [cit. 2016-03-19]. Dostupné z: `http://projects.spring.io/spring-boot/`

[45] JUnit: *JUnit - About*. [cit. 2016-04-30]. Dostupné z: `http://junit.org/junit4/`

[46] Mockito: *Mockito*. [cit. 2016-04-30]. Dostupné z: `http://mockito.org/`

[47] Jayway: *PowerMock is a Java framework that allows you to unit test code normally regarded as untestable.* [cit. 2016-04-30]. Dostupné z: `https://github.com/jayway/powermock`

[48] DB-Engines: *historical trend of the popularity ranking of database management systems*. [cit. 2016-03-19]. Dostupné z: `http://db-engines.com/en/ranking_trend`

[49] PostgreSQL: *Feature Matrix*. [cit. 2016-03-19]. Dostupné z: `http://www.postgresql.org/about/featurematrix/`

[50] MySQL: *MySQL 5.7 Reference Manual :: 1.3.2 The Main Features of MySQL*. [cit. 2016-03-19]. Dostupné z: `http://dev.mysql.com/doc/refman/5.7/en/features.html`

[51] Mark Pilgrim: *Dive Into HTML5*. [cit. 2016-03-19]. Dostupné z: `http://diveintohtml5.info/`

[52] Gasston, P.: *The Book of CSS3: A Developer's Guide to the Future of Web Design*. No Starch Press, druhé vydání, 2014, ISBN 978-1-593275-80-8.

[53] Chris Eppstein & collective: *Sass: Syntactically Awesome Style Sheets*. [cit. 2016-03-19]. Dostupné z: `http://sass-lang.com/documentation/file.SASS_REFERENCE.html`

[54] Green, B.; Seshadri, S.: *AngularJS*. O'Reilly Media, 2013, ISBN 978-1-449344-85-6.

[55] Franko, G.: *Instant Dependency Management with RequireJS How-to*. Packt Publishing, 2013, ISBN 978-1-782169-06-2.

[56] Nielsen, J.: *Usability Inspection Methods*. John Wiley & Sons, 1994, ISBN 978-0-471018-77-3.

[57] Telono SA: *Lo-Fi vs. Hi-Fi Prototyping: how real does the real thing have to be?* [cit. 2016-05-02]. Dostupné z: `http://www.telono.com/fr/nos-articles/lo-fi-vs-hi-fi-prototyping-how-real-does-the-real-thing-have-to-be/`

[58] The Apache Software Foundation: *Maven Features*. [cit. 2016-04-16]. Dostupné z: `https://maven.apache.org/maven-features.html`

[59] Pivotal Software, Inc.: *Data access with JDBC*. [cit. 2016-04-20]. Dostupné z: `https://docs.spring.io/spring/docs/current/spring-framework-reference/html/jdbc.html`

[60] Elasticsearch: *Getting Kibana Up and Running*. [cit. 2016-04-17]. Dostupné z: `https://www.elastic.co/guide/en/kibana/current/setup.html`

[61] Elasticsearch: *Mapping*. [cit. 2016-04-17]. Dostupné z: `https://www.elastic.co/guide/en/elasticsearch/guide/current/mapping-intro.html`

[62] Elasticsearch: *Indexing Parents and Children*. [cit. 2016-04-17]. Dostupné z: `https://www.elastic.co/guide/en/elasticsearch/guide/current/indexing-parent-child.html`

[63] Elasticsearch: *Nested type aggregations - Issue #1084 - elastic/kibana - GitHub*. [cit. 2015-12-13]. Dostupné z: `https://github.com/elastic/kibana/issues/1084`

[64] Elasticsearch: *Memory leak in server? - Issue #5170 - elastic/kibana - GitHub*. [cit. 2015-12-13]. Dostupné z: `https://github.com/elastic/kibana/issues/5170`

[65] Jörg Prante: *Why not support multiple import in a single jdbc river - Issue #746 - jprante/elasticsearch-jdbc - GitHub*. [cit. 2016-02-03]. Dostupné z: https://github.com/jprante/elasticsearch-jdbc/issues/746

[66] Google: *AngularJS — Superheroic JavaScript MVW Framework*. [cit. 2015-12-27]. Dostupné z: https://angularjs.org/

[67] Grunt Development Team: *Grunt: The JavaScript Task Runner*. [cit. 2016-04-16]. Dostupné z: http://gruntjs.com/

[68] Ignatius Steven: *AngularJS MultiSelect*. [cit. 2016-04-16]. Dostupné z: https://github.com/isteven/angular-multi-select

[69] Valentin Hervieu & collective: *AngularJS Slider*. [cit. 2016-04-16]. Dostupné z: http://angular-slider.github.io/angularjs-slider

[70] JUnit: *SpringJUnit4ClassRunner (Spring Framework 4.2.4.RELEASE API)*. [cit. 2016-04-30]. Dostupné z: http://docs.spring.io/autorepo/docs/spring-framework/4.2.4.RELEASE/javadoc-api/org/springframework/test/context/junit4/SpringJUnit4ClassRunner.html

[71] Elasticsearch: *Better error handling for IE11 users accessing large dashboards - Issue #6531 - elastic/kibana - GitHub*. [cit. 2016-04-30]. Dostupné z: https://github.com/elastic/kibana/issues/6531

Příloha **A**

Seznam použitých zkratek

AJAX Asynchronous JavaScript and XML

AOP Aspect Oriented Programming

API Application Programming Interface

CAPTCHA Completely Automated Public Turing test to tell

CSS Cascading Style Sheets

GPS Global Positioning System

GUI Graphical User Interface

HTML HyperText Markup Language

IP Internet Protocol

JDBC Java Database Connectivity

JSON JavaScript Object Notation

JVM Java Virtual Machine

MVC Model-View-Controller

POI Point Of Interest

POJO Plain Old Java Object

RAM Random Access Memory

REST Representational State Transfer

RSS Rich Site Summary

SASS Syntactically Awesome Style Sheets

SDK Software Development Kit

SQL Structured Query Language

URL Uniform Resource Locator

XML Extensible Markup Language Computers and Humans Apart

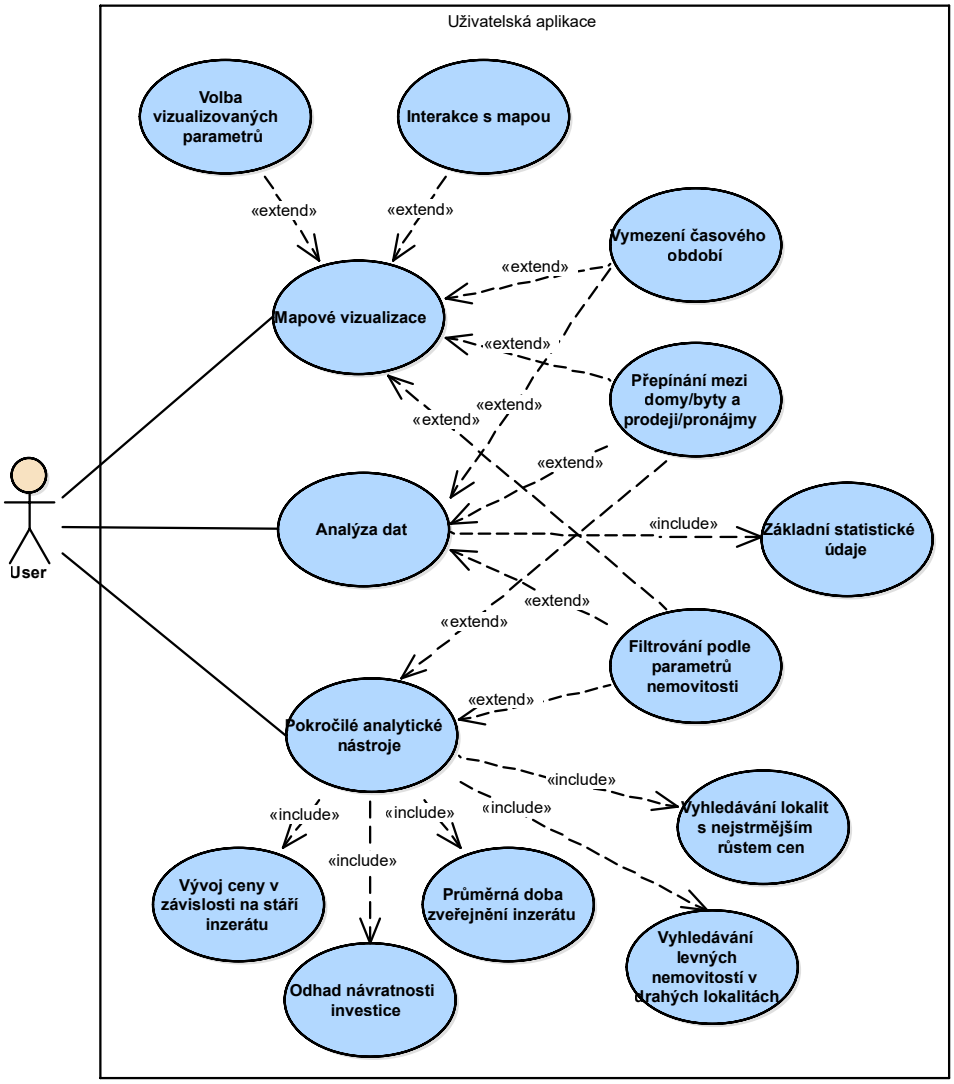
Příloha **B**

Obsah přiloženého CD

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

Příloha C

Obrázky



Obrázek C.1: Kompletní diagram případů užití

Příloha D

Přehledy

D.1 Přehled vybraných parametrů URL adresy pro získání přehledu se seznamem nemovitostí

- category_main_cb
  - 1 - byty
  - 2 - domy
  - 3 - pozemky
  - 4 - komerční
  - 5 - ostatní
- category_type_cb
  - 1 - prodej
  - 2 - pronájem
  - 3 - dražby
- locality_region_id
  - 1 - Jihočeský
  - 2 - Plzeňský
  - 3 - Karlovarský
  - 4 - Ústecký
  - 5 - Liberecký
  - 6 - Královéhradecký
  - 7 - Pardubický
  - 8 - Olomoucký

  - 9 - Zlínský
  - 10 - Praha
  - 11 - Středočeský
  - 12 - Moravskoslezský
  - 13 - Vysočina
  - 14 - Jihomoravský
- page - stránka, od které se stahují data
- per_page - počet vrácených nemovitostí od určité stránky; výchozí hodnota je 20, další alternativy jsou 40 a 60
- sort
  - 0 - nejnovější
  - 1 - nejlevnější
  - 2 - nejdražší
- tms - čas ve vteřinách od 1. 1. 1970, slouží ke kontrole cache
- category_sub_cb
  - 2 - 1+kk
  - 3 - 1+1
  - 4 - 2+kk
  - 5 - 2+1
  - 6 - 3+kk
  - 7 - 3+1
  - 8 - 4+kk
  - 9 - 4+1
  - 10 - 5+kk
  - 11 - 5+1
  - 12 - 6 a více
  - 16 - atypický
  - 33 - chata
  - 35 - památka
  - 37 - rodinný dům
  - 39 - vila
  - 40 - projekt na klíč

  - 44 - zemědělská usedlost
  - 47 - pokoj v objektu

- building_condition
  - 1 - velmi dobrý
  - 2 - dobrý
  - 3 - špatný
  - 4 - ve výstavbě
  - 5 - developerské projekty
  - 6 - novostavba
  - 7 - k demolici
  - 8 - před rekonstrukcí
  - 9 - po rekonstrukci

- locality_district_id
  - 1 - České Budějovice
  - 2 - Český Krumlov
  - 3 - Jindřichův Hradec
  - 4 - Písek
  - 5 - Prachatice
  - 6 - Strakonice
  - 7 - Tábor
  - 8 - Domažlice
  - 9 - Cheb
  - 10 - Karlovy Vary
  - 11 - Klatovy
  - 12 - Plzeň-město
  - 13 - Plzeň-jih
  - 14 - Plzeň-sever
  - 15 - Rokycany
  - 16 - Sokolov
  - 17 - Tachov
  - 18 - Česká Lípa
  - 19 - Děčín

- 20 - Chomutov
- 21 - Jablonec nad Nisou
- 22 - Liberec
- 23 - Litoměřice
- 24 - Louny
- 25 - Most
- 26 - Teplice
- 27 - Ústí nad Labem
- 28 - Hradec Králové
- 29 - Chrudim
- 30 - Jičín
- 31 - Náchod
- 32 - Pardubice
- 33 - Rychnov nad Kněžnou
- 34 - Semily
- 35 - Svitavy
- 36 - Trutnov
- 37 - Ústí nad Orlicí
- 38 - Zlín
- 39 - Kroměříž
- 40 - Prostějov
- 41 - Uherské Hradiště
- 42 - Olomouc
- 43 - Přerov
- 44 - Šumperk
- 45 - Vsetín
- 46 - Jeseník
- 48 - Benešov
- 49 - Beroun
- 50 - Kladno
- 51 - Kolín
- 52 - Kutná hora
- 53 - Mladá Boleslav

- 54 - Mělník
- 55 - Nymburk
- 56 - Praha-východ
- 57 - Praha-západ
- 58 - Příbram
- 59 - Rakovník
- 60 - Bruntál
- 61 - Frýdek-Místek
- 62 - Karviná
- 63 - Nový Jičín
- 64 - Opava
- 65 - Ostrava-město
- 66 - Havlíčkův Brod
- 67 - Jihlava
- 68 - Pelhřimov
- 69 - Třebíč
- 70 - Žďár nad Sázavou
- 71 - Blansko
- 72 - Brno-město
- 73 - Brno-venkov
- 74 - Břeclav
- 75 - Hodonín
- 76 - Vyškov
- 77 - Znojmo
- 5001 - Praha 1
- 5002 - Praha 2
- 5003 - Praha 3
- 5004 - Praha 4
- 5005 - Praha 5
- 5006 - Praha 6
- 5007 - Praha 7
- 5008 - Praha 8
- 5009 - Praha 9
- 5010 - Praha 10

Příloha E

Zdrojové kódy

E.1 SQL skript pro vytvoření databáze

Zdrojový kód E.1: Skript pro vytvoření databáze

```
CREATE TABLE real_estate(
    real_estate_id SERIAL PRIMARY KEY,
    title VARCHAR(256) UNIQUE NOT NULL,
    url VARCHAR(256)
);

CREATE TABLE ownership(
    ownership_id SERIAL PRIMARY KEY,
    title VARCHAR(256) UNIQUE NOT NULL
);

CREATE TABLE country(
    country_id SERIAL PRIMARY KEY,
    title VARCHAR(256) UNIQUE NOT NULL
);

CREATE TABLE district(
    district_id SERIAL PRIMARY KEY,
    title VARCHAR(256) UNIQUE NOT NULL
);

CREATE TABLE building_type(
    building_type_id SERIAL PRIMARY KEY,
    title VARCHAR(256) UNIQUE NOT NULL
);

CREATE TABLE building_position(
    building_position_id SERIAL PRIMARY KEY,
```

```
    title VARCHAR(256) UNIQUE NOT NULL
);

CREATE TABLE building_variant(
    building_variant_id SERIAL PRIMARY KEY,
    title VARCHAR(256) UNIQUE NOT NULL
);

CREATE TABLE state(
    state_id SERIAL PRIMARY KEY,
    title VARCHAR(256) UNIQUE NOT NULL
);

CREATE TABLE energy_category(
    energy_category_id SERIAL PRIMARY KEY,
    title VARCHAR(256) UNIQUE NOT NULL
);

CREATE TABLE layout(
    layout_id SERIAL PRIMARY KEY,
    title VARCHAR(256) UNIQUE NOT NULL
);

CREATE TABLE type(
    type_id SERIAL PRIMARY KEY,
    title VARCHAR(256) UNIQUE NOT NULL
);

CREATE TABLE acquisition(
    acquisition_id SERIAL PRIMARY KEY,
    title VARCHAR(256) UNIQUE NOT NULL
);

CREATE TABLE heating(
    heating_id SERIAL PRIMARY KEY,
    title VARCHAR(256) UNIQUE NOT NULL
);

CREATE TABLE water(
    water_id SERIAL PRIMARY KEY,
    title VARCHAR(256) UNIQUE NOT NULL
);

CREATE TABLE sewerage(
    sewerage_id SERIAL PRIMARY KEY,
    title VARCHAR(256) UNIQUE NOT NULL
);

CREATE TABLE location_type(
```

```
    location_type_id SERIAL PRIMARY KEY,
    title VARCHAR(256) UNIQUE NOT NULL
);

CREATE TABLE estate(
    estate_id SERIAL PRIMARY KEY,
    title VARCHAR(256) NOT NULL,
    url VARCHAR(256) UNIQUE NOT NULL,
    gps POINT,
    area_useable INTEGER,
    area_total INTEGER,
    area_garden INTEGER,
    area_built_up INTEGER,
    area_land INTEGER,
    balcony BOOLEAN,
    basement BOOLEAN,
    parking BOOLEAN,
    terrace BOOLEAN,
    garage BOOLEAN,
    equipment BOOLEAN,
    elevator BOOLEAN,
    floor INTEGER,
    floor_total INTEGER,
    real_estate_id INTEGER,
    ownership_id INTEGER,
    country_id INTEGER NOT NULL,
    district_id INTEGER,
    building_type_id INTEGER,
    building_position_id INTEGER,
    building_variant_id INTEGER,
    state_id INTEGER,
    energy_category_id INTEGER,
    layout_id INTEGER NOT NULL,
    type_id INTEGER NOT NULL,
    acquisition_id INTEGER NOT NULL,
    heating_id INTEGER,
    water_id INTEGER,
    sewerage_id INTEGER,
    location_type_id INTEGER,
    FOREIGN KEY (real_estate_id) REFERENCES
        real_estate(real_estate_id),
    FOREIGN KEY (ownership_id) REFERENCES ownership(ownership_id),
    FOREIGN KEY (country_id) REFERENCES country(country_id),
    FOREIGN KEY (district_id) REFERENCES district(district_id),
    FOREIGN KEY (building_type_id) REFERENCES
        building_type(building_type_id),
    FOREIGN KEY (building_position_id) REFERENCES
        building_position(building_position_id),
    FOREIGN KEY (building_variant_id) REFERENCES
```

```
        building_variant(building_variant_id),
    FOREIGN KEY (state_id) REFERENCES state(state_id),
    FOREIGN KEY (energy_category_id) REFERENCES
        energy_category(energy_category_id),
    FOREIGN KEY (layout_id) REFERENCES layout(layout_id),
    FOREIGN KEY (type_id) REFERENCES type(type_id),
    FOREIGN KEY (acquisition_id) REFERENCES
        acquisition(acquisition_id),
    FOREIGN KEY (heating_id) REFERENCES heating(heating_id),
    FOREIGN KEY (water_id) REFERENCES water(water_id),
    FOREIGN KEY (sewerage_id) REFERENCES sewerage(sewerage_id),
    FOREIGN KEY (location_type_id) REFERENCES
        location_type(location_type_id)
);

CREATE TABLE price(
    price INTEGER NOT NULL,
    date DATE NOT NULL,
    estate_id INTEGER NOT NULL,
    FOREIGN KEY (estate_id) REFERENCES estate(estate_id),
    PRIMARY KEY (date, estate_id)
);

CREATE TABLE availability_status(
    availability_status_id SERIAL PRIMARY KEY,
    title VARCHAR(256) UNIQUE NOT NULL
);

CREATE TABLE availability(
    date DATE NOT NULL,
    availability_status_id INTEGER NOT NULL,
    estate_id INTEGER NOT NULL,
    FOREIGN KEY (availability_status_id) REFERENCES
        availability_status(availability_status_id),
    FOREIGN KEY (estate_id) REFERENCES estate(estate_id),
    PRIMARY KEY (date, estate_id)
);

CREATE TABLE poi(
    poi_id SERIAL PRIMARY KEY,
    title VARCHAR(256) UNIQUE NOT NULL
);

CREATE TABLE has_poi(
    distance INTEGER NOT NULL,
    description VARCHAR(256) NOT NULL,
    estate_id INTEGER NOT NULL,
    poi_id INTEGER NOT NULL,
    FOREIGN KEY (estate_id) REFERENCES estate(estate_id),
```

```
    FOREIGN KEY (poi_id) REFERENCES poi(poi_id),
    PRIMARY KEY (estate_id, poi_id)
);

CREATE TABLE settings(
    name VARCHAR(256) PRIMARY KEY,
    boolean_value BOOLEAN,
    string_value VARCHAR(256),
    date_value DATE,
    int_value INTEGER
);
```