FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Aplikace pro monitoring cen nemovitostí
Student:	Ondřej Malach
Vedoucí:	Ing. Jan Blizničenko
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2020/2021

Pokyny pro vypracování

Cílem práce je vytvořit nástroj dlouhodobě monitorující ceny českých nemovitostí, který bude přístupný uživatelům skrze Single Page webovou aplikaci. Aplikace bude na denní bázi automatizovaně stahovat aktuální nabídku nemovitostí z některého z velkých realitních serverů, zpracovávat ji a ukládat do databáze. Na základě těchto dat bude nástroj počítat index cen českých nemovitostí a kromě toho bude nabízet uživatelům interaktivní rozhraní s možností vykreslování grafů na základě zvolených parametrů. Nedílnou součástí práce je i výběr vhodných technologií a využití již existujících frameworků v co největší možné míře.

- Proveďte rešerši podobných nástrojů.
- Proveďte rešerši relevantních technologií, nástrojů, frameworků a knihoven.
- Vytvořte návrh aplikace a jejích součástí.

- Proveďte implementaci aplikace.

- Otestujte a zdokumentujte své řešení.

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 6. února 2020 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Aplikace pro monitoring cen nemovitostí

Ondřej Malach

Katedra softwarového inženýrství
Vedoucí práce: Ing. Jan Blizničenko

27. června 2021

Poděkování

Děkuji Janu Blizničenkovi za cenné rady a připomínky, své rodině a především drahé ženě za neutuchající pomoc a podporu, tátovi za odborné konzultace a Davidu J. Malanovi za jeho pedagogickou činnost, bez níž bych se v tuto chvíli živil čímkoliv, jen ne programováním.

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. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 27. června 2021 ……

Odkaz na tuto práci

Malach, Ondřej. *Aplikace pro monitoring cen nemovitostí*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2021.

Abstrakt

Tato práce se zabývá návrhem a implementací nástroje, který automatizovaně monitoruje ceny českých nemovitostí a na jejich základě sestavuje **index**.

Hlavní myšlenkou je vytvoření interaktivní webové aplikace, skrze kterou je možné sledovat vývoj ceny českých nemovitostí takřka „v reálném čase“ ať již v obecném rámci napříč českým trhem, či s rozlišením na jednotlivé regiony a typy nemovitostí.

Backendová část je napsána v Javě s použitím moderní platformy Quarkus. Frontend je postavený na javaskriptovém frameworku Oracle JET.

Klíčová slova Monitoring cen nemovitostí, Český realitní index, Český nemovitostní index, Reality, Nemovitosti, Vývoj ceny českých nemovitostí, Český realitní trh, Quarkus, Java, Single Page aplikace

Abstract

This thesis deals with the design and implementation of a tool that automatically monitors prices of Czech real estates and compiles an **index** based on them.

The main idea is to create an interactive web application, through which it is possible to monitor the development of Czech real estate prices almost „in real time“, either in general across the Czech market, or with a distinction to individual regions and types of real estates.

The backend part is written in Java using modern Quarkus platform. The frontend is built on the Oracle JET JavaScript framework.

Keywords Real Estate Price Monitoring, Czech Real Estate Index, Real Estate, Czech Real Estate Price Development, Czech Real Estate Market, Quarkus, Java, Single Page application

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů kódu

ÚVOD

Psal se konec roku 2017 a já jsem pracoval v hypotečním oddělení nejmenované banky. Seděl jsem na jednom z pravidelných celofiremních meetingů a sledoval prezentaci vedoucího analytického oddělení, jak mluví o tom, že „ceny nemovitostí jsou s nejvyšší pravděpodobností nadhodnoceny o 15 až 25 procent“. Pamatuji si, že mi v tu chvíli blesklo hlavou: „V tom případě by nemovitost kupoval jen blázen!“ a investici do vlastní nemovitosti jsem odložil na neurčito. Tou dobou se prodával metr čtvereční rezidenční nemovitosti v Praze průměrně okolo sedmdesáti tisíc Kč [1].

Dnes – o tři a půl roku později – stojí stejný metr již výrazně přes sto tisíc korun [2][3]. Co se naopak nezměnilo, jsou hlasy mluvící o předražených nemovitostech – je to pár týdnů, co guvernér České národní banky uvedl, že „ceny nemovitostí jsou nadhodnoceny o 15 až 25 procent“ [4].

V nedávné anketě serveru *aktuálně.cz* uspořádané mezi třinácti vybranými experty se pro perspektivu dalšího růstu cen vyslovilo stejné množství osobností jako pro její pokles. Nejlépe to podle mého názoru vyjádřil ekonom ČSOB Petr Dufek: „Budoucí vývoj? To si myslím, vůbec nikdo neví [5].“

Začal jsem tedy pátrat, zda by nebylo možné opřít se o „tvrdá data“, spíš než o názory ekonomů a šéfů investičních, analytických, stavebních a realitních společností. Vždyť technické analýzy akciových trhů jsou významným nástrojem používaným pro odhad budoucího vývoje. Kupodivu jsem však zjistil, že ačkoliv již existuje několik entit, které se zabývají mapováním růstu nemovitostí v ČR, žádná z nich to nedělá na denní bázi, natož „v reálném čase“ a především s dostatečně detailním „rozlišením“.

A od toho byl již jen krůček k nápadu, že bych mohl požadovaný nástroj vytvořit sám.

Kapitola 1

CÍL PRÁCE

Cílem této bakalářské práce je vytvořit nástroj dlouhodobě monitorující ceny českých nemovitostí, který bude přístupný uživatelům skrze Single Page webovou aplikaci.

Aplikace bude na denní bázi provádět automatizované stahování aktuální nabídky nemovitostí z některého z velkých českých realitních serverů, která se posléze zpracuje a uloží do databáze. Na základě těchto dat bude počítán index cen českých nemovitostí zvaný **Český realitní index**, který bude dostupný skrze interaktivní webové rozhraní, které bude dále nabízet možnost detailního zobrazování vývoje cen nemovitostí na základě uživatelem zvolených parametrů a kategorií.

Teoretická část se bude zabývat řešerší, návrhem aplikace a vysvětlením jejích hlavních principů; praktická implementací a otestováním na úrovni unit a integračních testů.

Nedílnou součástí práce bude i výběr vhodných technologií a využití již existujících frameworků v co největší možné míře.

Kapitola 2

ANALÝZA

2.1 Realitní index

Zatímco ve světě informačních technologií není třeba termín *index* blíže vysvětlovat, v kontextu této práce jej používám ve významu běžném v ekonomii: jedná se o *ukazatel (relativní) hodnoty určitého souboru aktiv* [6].

Český realitní index je tedy **indikátor vyjadřující hodnotu českých nemovitostí (realit) v čase**.

2.1.1 Rešerše existujících realitních indexů

Indexy měřící cenu nemovitostí nejsou žádnou novinkou. Naopak, existuje jich celá řada [7] a v posledních letech s tím, jak roste význam nemovitostí jako alternativního investičního nástroje, jejich počet dále roste. Zpravidla jsou však zaměřeny na vyspělé a/nebo velké země. Z těch nejznámějších jmenujme např. *Dow Jones U.S. Real Estate* [8] nebo *UK House Price Index* [9], oba mapující ceny nemovitostí více než 20 let.

V rámci České republiky hrál dlouhou dobu velmi významnou roli **Český statistický úřad**, který pravidelně v ročním intervalu v rámci publikace *Ceny sledovaných druhů nemovitostí* zveřejňoval i index cen nemovitostí na základě informací o výběru daně z nabytých nemovitých věcí. Protože však byla tato daň v září 2020 zrušena, ČSÚ oznámil ukončení sledování vývoje cen nemovitostí až do doby, než nalezne „alespoň částečnou náhradu“ zaniklého datového zdroje. Poslední zveřejněná data jsou tak za rok 2019, přičemž „předpokládaná první publikace nového druhu by se měla objevit během roku 2022“ [10].

Dle četnosti citací v médiích tak roli nejvýznamnějšího českého nemovitostního indexu převzal komerční **Deloitte Real Index**. Je postaven na datech projektu *Cenová mapa Společnosti pro Cenové mapy ČR*, který čerpá informace o uskutečněných prodejích přímo z katastru nemovitostí. To sice do jisté míry dává přesnější výsledky, na stranu druhou to má i některé zásadní ne-

výhody. Například zcela chybí údaje o družstevních bytech, neboť ty katastr nemovitostí neeviduje, a především jsou data vydávána zpětně a pouze na kvartální bázi [11]. (Pro představu: K datu 5. května 2021 ještě nejsou k dispozici informace za první čtvrtletí tohoto roku.)

Kvalitní informace ohledně vývoje cen českých nemovitostí jsou tedy k dostání, nicméně nenabízejí to, co bych rád implementoval v rámci této práce: **možnost jejich sledování v reálném čase**.

2.2 České realitní servery

Jednou z nejdůležitějších otázek, které zásadním způsobem ovlivní podobu této bakalářské práce, je, ze kterého realitního serveru budu čerpat data o nemovitostech.

Z hlediska priorit je pro mě nejdůležitější:

- **Počet aktivních inzerátů**, a to z důvodu co nejvyšší možné relevance počítaného indexu.
- **Počet unikátních uživatelů**, neboť jsem přesvědčen o tom, že tento údaj do velké míry vypovídá o tom, jak jsou nabídky relevantní.
- **Podoba veřejného API**, protože aplikace potřebuje zasílat automatizované dotazy a automatizovaně je i zpracovávat.
- **Efektivita z hlediska množství zasílaných *HTTP requestů*** – vzhledem k tomu, že bude očekávaně třeba analyzovat vyšší jednotky tisíc až nižší desítky tisíc nemovitostí každý den, je důležité, aby tak bylo možné učinit s co nejmenší možnou síťovou aktivitou ze strany aplikace.

Dle veřejných zdrojů je jedničkou na trhu jak do počtu aktivních inzerátů tak do počtu návštěvníků server *sreality.cz*. Druhé místo patří taktéž v obou kategoriích portálu *reality.idnes.cz*. Na třetím místě z hlediska návštěv je dle Netmonitoru server *bezrealitky.cz*, který má nicméně výrazně méně aktivních inzerátů než každý z obou předchozích. Podle některých zdrojů třetí místo z pohledu množství nabídek přísluší serveru *realitymix.cz*, který nicméně v oblasti návštěvnosti nepatří ani do první desítky [12].

Množství aktivních inzerátů a měsíční počty unikátních uživatelů, návštěv a zobrazených stránek pro jednotlivé servery shrnuje následující tabulka:

Význam ostatních realitních serverů lze oproti čtyřem výše zmíněným považovat za minoritní, proto se jimi nebudu v rámci této práce hlouběji zabývat.

Pojďme se na jednotlivé realitní servery z hlediska výše zmíněných hledisek podívat blíže:

Tabulka 2.1: Srovnání realitních serverů.

Server	Inzerátů	Návštěvníků	Návštěv	Zobrazení
Sreality [13]	65 235	2 371 496	15 196 213	143 740 457
iDNES Reality [14]	52 196	1 103 544	3 333 012	79 467 914
RealityMIX [15]	15 603	–	343 830	4 194 726
Bezrealitky [16]	7 627	678 616	1 873 638	8 126 050

(Počty inzerátů vychází z oficiálních dat serverů ze dne 24. 4. 2021.)

2.2.1 Sreality

Jedná se o dlouhodobě největší, nejnavštěvovanější a nejznámější realitní server v České republice. Co se týče počtu aktivních inzerátů a unikátních uživatelů, v obou těchto oblastech nemá v České republice konkurenci.

API pro výpis inzerátů je dostupné přes GET request posílaný na adresu `https://www.sreality.cz/api/cs/v2/estates`, ke které se připojují parametry upřesňující lokalitu, požadovaný typ nemovitosti apod. Server vrací odpověď ve formátu JSON, která obsahuje celkový počet inzerátů vyhovujících dotazu, a první stránku výsledků. Pro každou nabízenou nemovitost jsou uvedeny mimo jiné tyto informace:

- název inzerátu obsahující rozlohu nemovitosti,
- cena,
- upřesnění lokality (typicky ulice nebo alespoň městská část),
- ID v Sreality univerzu,
- GPS souřadnice,
- štítky určující vlastnosti dané nemovitosti: např. zda má garáž, balkón, že se jedná o novou nemovitost, z jakého materiálu je postavena atd.

Uveďme příklad, jak vypadá URL dotazu, kterým bychom chtěli získat nabídku bytů k prodeji s dispozicemi 3+kk a 3+1 na Praze 1 a 8:

```
https://www.sreality.cz/api/cs/v2/estates?category_main_cb=
1&category_sub_cb=6|7&category_type_cb=1&locality_district_id=
5001|5008&locality_region_id=10
```

Výpis kódu 1: URL dotazu pro Sreality API

Podoba adresy je srozumitelná a snadno čitelná – stačí rozklíčovat, co konkrétně představují jednotlivé parametry a jejich hodnoty. Pomocí nich je možné podrobněji definovat kritéria nemovitosti. Patří mezi ně například:

- typ nemovitosti (byt, rodinný dům, pozemek, ...),

- dispozice nemovitosti,
- kraj,
- město či městská část,
- počet nemovitostí, které má dotaz vrátit (grafické rozhraní nabízí hodnoty 20, 40 či 60).

Bez nutnosti ptát se individuálně na detaily té či které nemovitosti jsme tak schopni jedním dotazem získat poměrně významné množství informací až pro šedesát nemovitostí. Odpověď lze navíc velmi snadno zparsovat a tudíž i automatizovaně zpracovat.

2.2.2 iDNES Reality

Stabilní česká „dvojka“ se počtem inzerovaných nabídek blíží svému většímu konkurentovi, nicméně v počtu unikátních návštěvníků, návštěv a zobrazených stránek mezi nimi panuje již poměrně značný rozdíl. Kde má však iDNES Reality portál z hlediska záměru této bakalářské práce největší nedostatky, je jeho API, jehož vstupním bodem je tak jako v předchozím případě GET request. Srovnejme, jak by vypadal URL dotazu pro získání stejného seznamu inzerátů jako v předchozím případě:

```
https://reality.idnes.cz/s/byty/?s-l=VUSC-19;MOP-19;MOP-86&s-
qc[subtypeFlat][0]=3k&s-qc[subtypeFlat][1]=31&s-qc[locality][0]=
VUSC-19&s-qc[locality][1]=MOP-19&s-qc[locality][2]=MOP-86
```

Výpis kódu 2: URL dotazu pro iDNES reality API

Na první pohled je patrné, že typ nemovitosti je pevnou součástí odkazu, a to jak jeho základní části, tak příslušných parametrů. Nebude proto možné sestavovat požadavky zcela genericky tak jako v případě Sreality API. Subjektivním poznatkem je, že toto API působí o poznání méně srozumitelným dojmem než v předchozím případě.

Server vrací odpověď ve formě samostatného HTML souboru, který krom seznamu požadovaných inzerátů obsahuje i velké množství druhotných prvků jako jsou reklamy, grafické elementy a JavaScriptový kód. Možnost parsování takového dokumentu je ve srovnání s JSON formátem omezená a představuje netriviální problém.

Součástí odpovědi je stejně jako u Sreality celkový počet odpovídajích inzerátů a první stránka výsledků – ta však obsahuje vždy pouze dvacet nabízených nemovitostí, přičemž součástí každého inzerátu jsou jen tyto tři údaje:

- název inzerátu obsahující rozlohu nemovitosti,
- upřesnění lokality (typicky ulice nebo alespoň městská část)

- a cena.

Pokud bychom se spokojili s těmito základními údaji, znamenalo by to, že efektivita iDNES Reality API z hlediska množství zasílaných požadavků by byla oproti Sreality třetinová. Pakliže bychom chtěli pracovat i s těmi informacemi, které Sreality nabízí v základním přehledu, bylo by nutné posílat zvláštní dotaz pro každou z inzerovaných nemovitostí, což by znamenalo snížení efektivity na méně než jednu šedesátinu.

2.2.3 RealityMIX

Ačkoliv počtem inzerátů uzavírá RealityMIX trojici největších realitních serverů, jeho návštěvnost je natolik nízká, že ani nebývá součástí oficiálních žebříčků, které české realitní servery sledují a vyhodnocují, a tak není možné z oficiálních zdrojů zjistit skutečný počet unikátních návštěvníků. Můžeme však porovnat průměrný počet návštěv, resp. zobrazených stránek za měsíc, což jsou veřejně dohledatelné údaje: 343 840 návštěv, resp. 4 194 726 zobrazených stránek RealityMIX serveru představuje asi 12 %, resp. 8 % provozu na iDNES Reality a přibližně 2 %, resp. 3 % provozu na Sreality serveru.

API opět poskytuje seznam inzerátů přes GET request. URL tohoto dotazu vypadá následovně:

```
https://realitymix.cz/vypis-nabidek/?form[adresa_kraj_id][]=
19&form[adresa_region_id][19][]=19&form[adresa_region_id][19][]=
86&form[cena_mena]=&form[cena_normalizovana__from]=&form[cena_
normalizovana__to]=&form[dispozice][]=11&form[dispozice][]=
4&form[exclusive]=&form[fk_rk]=&form[inzerat_typ]=
1&form[nemovitost_typ]=4&form[plocha__from]=&form[plocha_
_to]=&form[podlazi_cislo__from]=&form[podlazi_cislo__to]=
&form[projekt_id]=&form[search_in_city]=&form[search_in_text]=
&form[stari_inzeratu]=&form[stav_objektu]=&form[top_nabidky]=
```

Výpis kódu 3: URL dotazu pro RealityMIX API

Stejně jako u Sreality základ URL zůstává neměnný a konkretizace požadavku probíhá přidáním příslušných parametrů. Oproti oběma předchozím má však toto API dvě zvláštní specifika. Zaprvé, součástí URL jsou vždy všechny dostupné parametry, a to i v případě, kdy jsou pro daný dotaz zcela irelevantní – pak se jim nepřiřazuje žádná hodnota. A zadruhé, v případě, že pro některý parametr vyžadujeme vícero hodnot, musíme jej multiplikovat, jak je patrné z výše uvedeného příkladu, kde parametry `form[adresa_region_id][19][]` a `form[dispozice][]` jsou v URL zahrnuty dvakrát pokaždé s jinou hodnotou.

Formát a struktura odpovědi i její vizuální vzhled jsou velmi podobné, jako je tomu u iDNES Reality - dokonce do té míry, že si dovoluji vyjádřit domněnku, že se nutně nemusí jednat o podobnost čistě náhodnou. Z technického

hlediska se jedná opět o HTML stránku, která obsahuje celkový počet inzerovaných nemovitostí a první stránku s dvaceti inzeráty. Shodné jsou i uváděné informace pro každý z nich:

- název inzerátu obsahující rozlohu nemovitosti,
- upřesnění lokality,
- cena.

Z hlediska efektivity tedy platí pro RealityMIX to samé, co je uvedeno výše pro iDNES Reality server: v nejlepším případě jsme schopni se dostat na třetinu efektivity Sreality serveru na úkor množství informací; jinak je třeba zasílat separátní dotaz pro každou nemovitost a efektivita je méně než 2 % (jedna šedesátina).

2.2.4 Bezrealitky

Server bezrealitky.cz má svoji identitu založenou na tom, že na něm inzerují přímí majitelé nemovitostí bez zastoupení realitních kanceláří. To se logicky promítá do velikosti nabídky, která je oproti Sreality serveru přibližně o 90 % menší.

API tohoto serveru je dosti odlišné od všech předchozích: seznam inzerátů je dostupný přes POST request posílaný na URL `www.bezrealitky.cz/api/record/markers`, který ve svém těle obsahuje upřesňující parametry. Zajímavým způsobem je řešená definice lokality – protože Bezrealitky server je silně zaměřen na hledání nemovitostí skrze interaktivní mapu, jednotlivé oblasti jsou definovány jako nepravidelné mnohoúhelníky definované množinou GPS souřadnic. (Například pro Prahu je to 154 bodů, pro Brno jich je dokonce 228.) Odpověď se vrací v JSON formátu a obsahuje všechny nemovitosti, které vyhovují požadovaným parametrům, mimo jiné s těmito základními informacemi:

- cena,
- dispozice,
- podlahová plocha,
- GPS souřadnice,
- ID v univerzu Bezrealitky.

Naopak adresa, resp. upřesněná lokalita explicitně zahrnuty nejsou – nicméně to není zásadní překážkou vzhledem k tomu, že máme k dispozici GPS souřadnice.

Případnou alternativou je využití doplňujícího dotazu v GraphQL jazyce, který API Bezrealitky podporuje, a pomocí kterého lze získat na základě ID

nemovitosti (nebo jejich množiny) další podrobnosti včetně adresy nebo štítků upřesňující vlastnosti nemovitosti podobně jako je tomu u Sreality.

Co se týče efektivity, je na tom portál Bezrealitky vůbec nejlépe ze všech zkoumaných realitních serverů. Teoreticky by bylo možné získat všechny nemovitosti z celé ČR jediným dotazem a ten pak zparsovat. Ale i v případě, že bychom posílali dotazy po jednotlivých kategoriích a lokalitách, bude efektivita přinejmenším násobně větší než v případě ostatních konkurentů.

2.2.5 Vyhodnocení realitních serverů

Pokud vyhodnotíme pořadí realitních serverů z hlediska priorit definovaných na začátku této kapitoly, je vítěz jednoznačný:

Tabulka 2.2: Vyhodnocení realitních serverů

Hledisko	Sreality	iDNES Reality	RealityMIX	Bezrealitky
Počet inzerátů	**1.**	2.	3.	4.
Počet uživatelů	**1.**	2.	4.	3.
Podoba API	**1.-2.**	3.-4.	3.-4.	**1.-2.**
Efektivita	2.	3.-4.	3.-4.	**1.**

Zajímavý **alternativní pohled** na relevantnost těchto serverů nabízí realitní makléř Lukáš Hrdina, který paralelně inzeruje na všech těchto serverech. Dle jeho statistik více než 70 % reakcí na nabídky prodeje přichází z `sreality.cz`, necelých 15 % z `reality.idnes.cz` a méně než 5 % z `realitymix.cz` [17]. To však jen podtrhuje výše uvedené výsledky.

Jako primární zdroj dat pro svoji aplikaci volím tedy server **Sreality**.

2.3 Záměr

V úvodu této práce byly formálně definovány cíle. Ty jsou ale poměrně obecné a nevyjadřují nutně, co je hlavním záměrem a těžištěm z hlediska tvorby aplikace. Podívejme se proto blíže na základní myšlenky a koncepty, na kterých chci aplikaci stavět.

Skutečným jádrem tohoto projektu je **sběr dat**, potažmo jejich **databáze**. Index se dá vypočítat zpětně, je dokonce možné jeho výpočet změnit, případně lze vytvářet indexy zcela nové. Ovšem jen za předpokladu, že máme k dispozici relevantní a vhodným způsobem uložená data.

Bude tedy třeba věnovat velkou péči návrhu databázového schématu, neboť jakékoliv případné budoucí změny by mohly způsobit nekompatibilitu s dřívějšími daty.

Velký důraz bude též kladen na **spolehlivost** a **robustnost** systému zodpovědného za získávání dat: v případě, že aplikace spadne nebo její hostitelský

server bude mít výpadek, musí být možné spustit proces sběru dat znovu bez jakýchkoliv negativních efektů jako je například vznik duplicitních záznamů.

Dalším důležitým konceptem je snaha, aby automatizované sbírání dat působilo co nejméně roboticky – v ideálním případě by server, ze kterého budu čerpat data, neměl vůbec poznat, že požadavky na něj zasílané nepochází z grafického rozhraní používaného běžnými uživateli. To s sebou přináší nutnost zaprvé pečlivě hlídat zasílaná metadata a zadruhé randomizovat čas zasílání HTTP requestů a v neposlední řadě je rozložit do delšího časového úseku – zkrátka tak, jak s realitním portálem zachází skutečný člověk.

Naopak samotné zobrazování dat – přesto, že z hlediska koncového uživatele se jedná o nejviditelnější a nejdůležitější část celého projektu – je z konceptuálního pohledu jen jakousi relativně nepodstatnou „třešničkou na dortu“, kterou lze navíc bez jakýchkoli dopadů na datovou bázi libovolně měnit.

2.4 Analýza požadavků

Výchozím bodem při tvorbě jakékoliv aplikace je seznam funkčních a nefunkčních požadavků, abychom měli přesnější představu, jakou aplikaci máme vyvinout, jak ji máme vyvinout a především jakou aplikaci vyvinout *nemáme*.

2.4.1 Funkční požadavky

Jedná se o popis, jak má software fungovat bez vazby na konkrétní technologii či detailní architekturu systému [18]. Čistě z hlediska funkčních požadavků je aplikace poměrně nekomplikovaná. Hlavní otázkou při jejím vývoji není ani tak to, *co má dělat*, ale spíš *jak to má dělat*. Definujme si proto jen stručně několik základních funkčních požadavků a pro bližší podrobnosti jako je způsob, kterým je index počítán, či seznam typů nemovitostí a lokalit, nad kterými má sběr dat probíhat, odkažme čtenáře na sekci Metodika v kapitole Návrh.

- Monitoring nabídky nemovitostí:
  - **F1**: Aplikace musí na denní bázi z realitního serveru stahovat kompletní prodejní nabídku nemovitostí požadovaných Metodikou.
  - **F2**: Aplikace musí nabídku nemovitostí ukládat do databáze.
  - **F3**: Pro každou uchovávanou nemovitostní nabídku musí být evidováno, kdy byla vytvořena, kdy byla naposledy aktivní a jak se vyvíjela cena nemovitosti v čase, pakliže nebyla stejná po celou dobu životního cyklu nabízené nemovitosti.
- Simulace chování reálných uživatelů při komunikaci s realitním serverem:
  - **F4**: Stahování dat z realitního serveru musí být prováděno v pseudonáhodné časy.

  - **F5**: Aplikace musí simulovat přístup z různých zařízení.
  - **F6**: Mezi zasílanými dotazy je třeba dodržovat alespoň několikavteřinovou pauzu.

- Poskytování informací:
  - **F7**: Aplikace musí nabízet REST API, skrze které bude možné získat údaje o nemovitostech.
    - **F7A**: Aplikace musí poskytovat RESTový zdroj pro získání historického vývoje nemovitostního indexu s rozlišením na jednotlivé dny.
    - **F7B**: Aplikace musí poskytovat RESTový zdroj pro získání aktuální hodnoty nemovitostního indexu.
    - **F7C**: Aplikace musí nabízet RESTový zdroj pro získání historického vývoje průměrné ceny za 1 m^2 s rozlišením na jednotlivé dny, a to v závislosti na uživatelem požadovaném typu nemovitosti a jedné či více požadovaných lokalitách.
  - **F8**: Odpovědi zasílané RESTovým rozhraním musí být ve formátu JSON.

- Zobrazování informací:
  - **F9**: Aplikace musí poskytovat grafické webové rozhraní.
  - **F10**: Webové rozhraní musí obsahovat graf zobrazující vývoj realitního indexu v čase.
  - **F11**: Webové rozhraní musí obsahovat interaktivní sekci zobrazující vývoj průměrné ceny nemovitostí za 1 m^2 v čase, a to na základě uživatelem zvoleného typu nemovitosti a jedné či více zvolených lokalitách. V případě, že uživatel požaduje vícero lokalit, graf musí zachytit souběžný vývoj průměrné ceny pro každou lokalitu zvlášť.

2.4.2 Nefunkční požadavky

Definují kvality softwarového systému a jeho omezení [19].

- **N1** - Podporované platformy: Webový portál aplikace bude přizpůsoben pro zobrazení přes PC.
- **N2** - Podporované prohlížeče: Webový portál aplikace bude podporovat následující prohlížeče v příslušných verzích: Chrome 91+, Firefox 89+, Microsoft Edge 88+, Safari 13+.
- **N3** - Propustnost: Aplikace musí umět zpracovat alespoň desítky požadavků za minutu.

- **N4** - Doba odezvy: REST API musí poslat odpověď na požadavek v průměru nejpozději do dvou sekund.
- **N5** - Škálovatelnost: Aplikace musí být škálovatelná tak, aby v budoucnu potenciálně uměla obsloužit i tisíce až desetitisíce požadavků za hodinu.
- **N6** - Zabezpečení: Webový server musí pracovat v režimu HTTPS.

Kapitola 3

SREALITY API

Vzhledem k tomu, že dokumentace aplikačního rozhraní, skrze které je možné z Sreality serveru získávat nabídku nemovitostí, není veřejně dostupná[1], nezbylo mi než kontrakt API „odpozorovat“ z praxe - používal jsem tedy kombinaci **grafického rozhraní** Sreality, přes které jsem postupně volil jednotlivé filtry, a **vývojářské konzole** v prohlížeči, pomocí které jsem následně analyzoval, jak se tato volba promítá do podoby odesílaného požadavku.

Výsledky pozorování shrnuje tato kapitola.

Sreality API je webovou RESTovou službou a umožňuje tedy efektivní práci se zdroji pomocí standardních HTTP požadavků.

Každý takový požadavek musí obsahovat:

- **schéma** (v tomto případě `HTTPS`),
- **kořenovou autoritu** (`www.sreality.cz`),
- **cestu identifikující zdroj** (například `/api/cs/v2/estates`)

a často je třeba připojit též parametry, které mohou být buď součástí URL, nebo schované v těle dotazu.

Všechny dotazy související s nabídkou nemovitostí na Sreality obsahují základní URI `/api/cs/v2` a musí být typu `GET` – s datovými zdroji tedy můžeme pracovat pouze v režimu čtení a není možné je jakkoliv měnit.

Celý dotaz pro Sreality API tedy může vypadat například takto:

```
GET https://www.sreality.cz/api/cs/v2/estates?category_sub_cb=2
```

Výpis kódu 4: Příklad dotazu pro Sreality API

Říkáme tím, že chceme seznam všech nemovitostí s dispozicí 1+kk (neboť

[1] Zde narážíme na známý problém, že „absence důkazů není důkazem absence“. Možná by tedy bylo přesnější konstatovat, že autorovi této práce se dokumentaci API pro získávání nabídky nemovitostí ani po nemalém úsilí nepodařilo nalézt, na rozdíl od dokumentace API pro importování inzerátů na Sreality server, viz [20].

ta je v Sreality univerzu reprezentována právě parametrem `category_sub_cb` rovným hodnotě `2`).

REST endpointů nabízí Sreality vícero (pro větší přehlednost vynechávám společnou část `/api/cs/v2/estates`):

- **/count** – počet nemovitostí v Sreality databázi odpovídající předávaným parametrům,
- **/estates** – nemovitosti odpovídající předávaným parametrům se základními údaji,
- **/clusters** – množina ID nemovitostí v oblasti určené předávanými GPS souřadnicemi,
- **/mortgage** – parametry hypotéky u bank napříč trhem,
- **/userinfo** – informace o uživateli,
- **/pois** – seznam služeb jako jsou restaurace, obchody apod. v okolí předávaných GPS souřadnic.

Pro účel této práce si vystačím pouze s dvojicí zdrojů `/count` a `/estates`, kdy budu nejdříve zjišťovat, zda vůbec v Sreality databázi existují nemovitosti s požadovanými parametry, a až poté se budu případně tázat na jejich seznam. Čistě z hlediska funkcionality by bylo možné využívat rovnou endpoint `/estates`, nicméně vzhledem k tomu, že za normálních okolností z grafického prostředí není možné vyžádat si podrobnosti o nemovitostech takového typu, pro který neexistuje ani jeden zástupce, nevidím důvod, proč by na sebe měla aplikace zbytečně upozorňovat.

3.1 Významné URL parametry

Sreality nabízí širokou škálu parametrů, pomocí kterých lze podrobně definovat, jak má vypadat poptávaná nemovitost i její okolí. Pro potřeby mé aplikace budu však využívat pouze část z nich. Konkrétně mě zajímají tyto parametry (ke kterým uvádím reprezentativní seznam hodnot):

category_main_cb: Určuje základní typ nemovitosti:

- 1: byt,
- 2: dům,
- 3: pozemek,
- 4: komerční nemovitost a nebytový prostor,
- 5: ostatní.

category_sub_cb: Dispozice nemovitosti u bytů, resp. konkrétní typ domu a pozemku:

- 1: N/A,
- 2: 1+kk (kategorie Byty),
- 3: 1+1 (kategorie Byty),
- 4: 2+kk (kategorie Byty),
- 5: 2+1 (kategorie Byty),
- 6: 3+kk (kategorie Byty),
- 7: 3+1 (kategorie Byty),
- 8: 4+kk (kategorie Byty),
- 9: 4+1 (kategorie Byty),
- 10: 5+kk (kategorie Byty),
- 11: 5+1 (kategorie Byty),
- 12: 6 pokoju a vic (kategorie Byty),
- 16: atypický (kategorie Byty),
- 19: stavební parcela (kategorie Pozemky),
- 23: zahrada (kategorie Pozemky),
- 33: chata (kategorie Domy,
- 37: rodinný (kategorie Domy),
- 39: vila (kategorie Domy).

category_type_cb: Typ inzerátu:

- 1: prodej,
- 2: pronájem,
- 3: dražba.

locality_country_id: Země:

- 112: Česká republika,
- 793: Slovensko,
- 10000: obecně všechny zahraniční země.

locality_region_id: Kraj:

- 10: Praha,
- 11: Středočeský,
- 12: Moravskoslezský,

- 14: Jihomoravský.

locality_district_id: Město:

- 56: Praha-východ,
- 57: Praha-západ,
- 65: Ostrava-město,
- 72: Brno-město,
- 73: Brno-venkov.
- (Praha je identifikována jen přes `locality_region_id`).

no_auction: Příznak, zda mají být zahrnuty nemovitosti nabízené v dražbě:

- 1: ignorovat dražené nemovitosti,
- 0: zahrnout dražené nemovitosti.

per_page: Počet výsledků na stránku, přičemž UI nabízí hodnoty:

- 20,
- 40,
- 60.

page: Požadovaná stránka výsledků:

- Pokud parametr není zadán, server defaultně vrací první stránku.

room_count_cb: Počet pokojů u RD:

- 1-4: odpovídající počet pokojů,
- 5: 5 a více,
- 6: atypický.

tms: Timestamp ve formátu známém jako `UNIX time`[2].

3.2 REST endpoint /count

V grafickém rozhraní je dotaz na tento zdroj odesílán při každé manipulaci s parametry poptávané nemovitosti. Pokud je počet nalezených nemovitostí nenulový, výsledek se promítne do popisku na tlačíku `Zobrazit N inzerátů`, kde `N` reprezentuje tuto číselnou hodnotu. Pokud nejsou nalezeny žádné takové nemovitosti, zobrazí se místo tlačítka informace `Nenašli jsme žádný inzerát`.

Odpověď má následující formát:

[2]Neboli počet milisekund od 1. ledna 1970, 00:00:00 UTC

```
{
    logged_in: false
    result_size: 7654
    result_size_auction: 108
    _links: {self: {profile: "/estates/count/doc",…}}
}
```


Výpis kódu 5: Odpověď Sreality serveru pro zdroj /count

Zásadním údajem je hodnota klíče `result_size`, která nám říká, kolik nemovitostí splňuje poptávaná kritéria. Klíč `result_size_auction` je přítomen vždy, a to bez ohledu na to, zda-li jsme požadovali nemovitosti v aukci ignorovat nebo zahrnout.

3.3 REST endpoint /estates

Tento zdroj je volán standardně po kliknutí na tlačítko `Zobrazit N inzerátů`. Kromě odlišného zdroje je podoba požadavku téměř totožná s tím, který byl odeslán na zdroj \`count` jako poslední, a to až na jednu malou výjimku, kterou je parametr `locality_country_id`: zatímco pro zdroj `count` je automaticky připojen, pro `estates` je v případě nemovitostí na území ČR vynechávána.

Odpověď vypadá obdobně jako *Výpis kódu 8* v *Příloze B*. Kromě rekapitulace zvolených parametrů (viz klíč `filter`) dostáváme mimo jiné informaci o počtu nalezených nemovitostí (`result_size`), číslo stránky, která nám byla zaslána (`page`), a především seznam nemovitostí, který je dostupný pod hierarchií klíčů `_embedded -> estates`, jak je patrné z *Výpisu kódu 9* v *Příloze B*.

Nejdůležitější informací je pro nás to, jaké údaje jsou vraceny pro každou nemovitost, viz následující *Výpis kódu 6*. Z nich jsou pro mě nejcennější klíče:

- **name** – název inzerátu obsahující i rozlohu nemovitosti,
- **price** – cena,
- **gps** – souřadnice,
- **hash_id** – unikátní identifikátor v rámci Sreality,
- **labelsAll** – štítky, ze kterých se dají vyčíst podrobné vlastnosti nemovitosti včetně těch, které grafické rozhraní Sreality nepublikuje.

Pokud tyto informace zkombinujeme s vhodnou volbou parametrů, se kterými budeme žádat Sreality API o seznam nemovitostí, jsme schopni se zcela vyhnout nutnosti posílat požadavek na podrobné detaily zvlášť pro každou z nemovitostí.

```
{
   estates: [
      0: {
         attractive_offer: 0
         auctionPrice: 0
         category: 2
         exclusively_at_rk: 1
         gps: {lat: 49.139881, lon: 16.666706}
         has_floor_plan: 1
         has_matterport_url: false
         has_panorama: 0
         has_video: false
         hash_id: 1354607196
         is_auction: false
         labels: ["Novostavba"]
         labelsAll: [
            0: ["new_building", "partly_furnished"]
            1: ["candy_shop", "movies", "natural_attraction",
            ↪  "playground", "small_shop", "tavern", "vet",
            ↪  "sports", ...]
         labelsReleased: [["new_building"], []]
         locality: "Režná, Brno - Tuřany"
         name: "Prodej  rodinného domu 209 m², pozemek 209 m²"
         new: false
         paid_logo: 0
         price: 11990000
         price_czk: {value_raw: 11990000, unit: "", name:
         ↪  "Celková cena"}
         region_tip: 0
         rus: falsevaz
         seo: {category_main_cb: 2, category_sub_cb: 37, ...}
         type: 1
         _embedded: {favourite: {is_favourite: false,…}, note:
         ↪  {note: "",…}}
         _links: {,…}
      },
      1: ...
   ]
}
```


Výpis kódu 6: Detail nemovitosti z Sreality serveru

Kapitola 4

NÁVRH

4.1 Metodika

Odpověď na otázku, jakým způsobem počítat index ceny nemovitostí, je pravděpodobně tím vůbec nejdůležitějším rozhodnutím v rámci této bakalářské práce. Možností je bezpočet a neexistuje žádné *ideální* řešení. Tak jako u jakéhokoliv jiného indexu záleží vždy na tom, která aktiva se rozhodneme zahrnout do výběru, a která nikoliv. Nelze se tedy vyhnout jisté míře subjektivity. O to důležitější ale je, aby existovala zcela jasná metodika, jak je index počítán.

První charakteristickou vlastností je, že do indexu budou zahrnuty pouze nemovitosti pro individuální bydlení - komerční nemovitosti budou tedy zcela záměrně **vynechány**. Z mého pohledu se jedná o natolik odlišné kategorie, že nedává velký smysl snažit se je pokrýt společným jmenovatelem.

Geograficky bude index omezen na nemovitosti ze tří největších českých měst:

- **Prahy**,
- **Brna**,
- a **Ostravy**,

a to jednak z důvodu větší diverzifikace (jedná se o nejvýznamnější zástupce Čech, Moravy a Slezska, tedy tří poměrně odlišných oblastí) a jednak proto, že právě v těchto městech trvale existuje dostatečně velká nabídka na to, aby se zprůměrovaly případné výrazné odchylky na obou krajích Gaussovy křivky.

Množina typů nemovitostí bude omezena na:

- **byty**,
- **rodinné domy**
- a **stavební pozemky**,

jakožto typické reprezentanty v oblasti individuální bydlení.

To nás ale přivádí k problému, jakým způsobem mezi sebou porovnat cenu bytu, cenu domu se zahradou a cenu stavebního pozemku tak, abychom na výstupu dostali jedno jediné číslo. Je zcela zřejmé, že průměrná cena za metr čtvereční nezastavěného stavebního pozemku má zcela jiný hodnotový význam než průměrná cena za metr čtvereční bytu. U rodinných domů se situace navíc komplikuje kvůli faktu, že součástí stavby je nutně i pozemek, na kterém stojí, a zpravidla také pozemek, na kterém stavba nestojí, ale je využíván jako zahrada, místo pro bazén a podobně. Řešením, které jsem zvolil, je **vážený průměr**.

4.1.1 Vážený průměr

Je zobecněním aritmetického průměru a vyjadřuje charakteristiku statistického souboru v případě, že hodnoty v tomto souboru mají různou důležitost [21]. Je definován následujícím způsobem:

Mějme soubor n hodnot $X = x_1, \ldots, x_n$ a k nim odpovídající soubor vah $W = w_1, \ldots, w_n$. Váženým průměrem pak nazveme hodnotu $\bar{x}$ takovou, že:

$$\bar{x} = \frac{\sum_{i=1}^{n} w_i x_i}{\sum_{i=1}^{n} w_i}$$

Zbývá tedy určit, jakou hodnotu přiřadit jednotlivým typům nemovitosti. Ač se cena stavby rodinného domu obvykle přepočítává na krychlový metr, architekti Roman Kučírek a Jan Komárek uvádí přepočet na metr čtvereční, který činí přibližně 35 000 Kč, a to v podstatě bez ohledu na lokalitu v rámci České republiky [22]. Naopak cena pozemků se napříč Českou republikou významně liší. Dle článku Denisy Babkové z roku 2018 [23] se ceny za 1 m^2 stavebního pozemku v tomto roce pohybovaly od 1 400 Kč v Moravskoslezském kraji po více než 7 500 Kč v Praze. Na základě mého pozorování[3] se tento nepoměr s růstem cen v posledních letech ještě podstatně zvýraznil: Zatímco v Ostravě zůstává cena pozemku na podobné úrovni jako ve zmiňovaném článku (zhruba 1700 Kč za 1 m^2), v Brně se pohybuje okolo 8350 Kč za 1 m^2 a v Praze se dokonce dostala přes 14 500 Kč za metr čtvereční. To znamená, že v Praze se zastavěním pozemku jeho hodnota zvýší přibližně třiapůlkrát, v Brně pětkrát a v Ostravě více než jedenadvacetkrát.

Jedná se o příliš velké rozdíly na to, aby je bylo možné objektivně pokrýt společným číselným vyjádřením. Rozhodl jsem se tedy, že v rámci každé ze sledovaných oblastí ohodnotím rodinný dům i pozemek různými vahami, abychom docílili podobného poměru jejich vážených průměrů vůči průměrné ceně bytu, jehož ohodnocení je zafixováno na hodnotě $w = 1$, napříč všemi oblastmi.

[3] Sledování průměrné ceny nemovitostí v Praze, Brně a Ostravě probíhalo na denní bázi mezi 24. dubnem a 28. květnem přes Sreality server. Celkový vzorek činil 8141 nemovitostí.

Na základě těchto vah jsme schopni dle výše uvedeného vzorce spočítat vážený průměr ceny za 1 m^2 rodinného domu s pozemkem, což nám dále umožňuje vyjádřit i příslušnou váhu tohoto typu nemovitosti. Konkrétní hodnoty jednotlivých vah zachycuje následující tabulka:

Tabulka 4.1: Váhy sledovaných typů nemovitostí

Město	w_{byt}	$w_{pozemek}$	$w_{RD+pozemek}$
Praha	1	$\frac{1}{3{,}405}$	$\frac{1}{1{,}632}$
Brno	1	$\frac{1}{5{,}188}$	$\frac{1}{1{,}727}$
Ostrava	1	$\frac{1}{21{,}444}$	$\frac{1}{1{,}392}$

(Sestaveno na základě dat viz pozn. pod čarou č. 3.)

Pro správný výpočet realitního indexu neboli váženého průměru napříč nemovitostmi a lokalitami je nutné zohlednit i počet nemovitostí. Je tedy třeba explicitně zmínit, že hodnota w_i ze vzorce pro výpočet váženého průměru je ve skutečnosti součinem váhy daného druhu nemovitosti v daném městě a počtu takovýchto nemovitostí.

Důležitým charakteristickým rysem pro výpočet indexu je rozhodnutí, že jednotlivé váhy budou konstantní, tj. i do budoucna zůstanou na výše zmíněných hodnotách, aby index věrněji reflektoval význam změny v čase vůči svému počátku.

Tak jak je u indexů běžným jevem, výsledná hodnota bude interpretována jako počet bodů.

Posledním metodickým znakem je, že výsledek váženého průměru, který je v tuto chvíli šestimístný, bude ponížen o dva řády, abychom operovali s „rozumnou velikostí“ čísel. Realitní index tak bude reprezentován číslem v řádu tisíců, což na základě dosavadního pozorování považuji za dostatečně detailní k tomu, abychom zachytili cenové změny v horizontu týdnů a řádech stokorun, ale zároveň ne příliš detailní na to, aby index zachycoval korunové či desetikorunové změny v rámci jednotlivých dnů, a byl tak příliš volatilní.

Podrobně rozepsaný vzorec pro výpočet českého realitního indexu $\bar{I}$ vypadá tedy následovně:

$$\bar{I} = \frac{n_{B_i} w_{B_i} x_{B_i} + n_{P_i} w_{P_i} x_{P_i} + n_{D_i} w_{D_i} x_{D_i}}{n_{B_i} w_{B_i} + n_{P_i} w_{P_i} + n_{D_i} w_{D_i}} \cdot 10^{-2}\ b.,$$

kde n_{B_i}, n_{P_i} a n_{D_i} představují postupně počty zahrnutých bytů, pozemků a rodinných domů v jednotlivých městech, w_{B_i}, w_{P_i} a w_{D_i} jsou postupně váhy bytu, pozemku a rodinného domu s pozemkem v jednotlivých městech (viz Tabulka 4.1) a x_{B_i}, x_{P_i} a x_{D_i} představují postupně cenové průměry za 1 m^2 bytu, pozemku a rodinného domu s pozemkem v jednotlivých městech.

4.2 Základní architektura

Jako základ pro svoji aplikaci jsem zvolil **třívrstvou architekturu** [24]. Ta předepisuje rozdělení na tři logické a zpravidla i fyzické celky:

- **Prezentační vrstva** nabízí uživatelské rozhraní a zprostředkovává komunikaci mezi klientem a aplikací.
- **Aplikační vrstva** označovaná někdy jako „vrstva business logiky“ je srdcem celé aplikace. Sbíhají se v ní data a požadavky z prezentační vrstvy s daty a požadavky pocházejícími z vnitřních procesů, které má za úkol zpracovat a případně na ně zareagovat typicky odesláním patřičné odpovědi prezentační vrstvě. Zajišťuje také veškerou komunikaci s datovou vrstvou.
- **Datová vrstva** ukládá a spravuje veškeré informace perzistentního charakteru.

V prostředí webových aplikací se vžilo používání pojmů **frontend** a **backend** jakožto synonym pro prezentační, resp. aplikační vrstvu – tohoto pojmenování se budu v rámci této práce držet.

4.3 Backend

4.3.1 Technologie

4.3.1.1 Java

Java je high level objektově orientovaný programovací jazyk představený v roce 1995 firmou Sun Microsystems. Dlouhodobě se jedná o jeden ze tří nejpoužívanějších [25] a nejpopulárnějších [26] programovacích jazyků na světě. Mezi jeho hlavní výhody patří platformní nezávislost, vysoká stabilita a rychlost (dnes již srovnatelná s jazyky C/C++). Nejen proto se Java stala takřka nedílnou součástí velkých serverových a webových aplikací a díky Androidu i víc než poloviny všech mobilních aplikací. Nicméně technologie Javy není jen o programovacím jazyku. Existuje též několik paralelních Java platforem, kdy každá z nich představuje obrovský ekosystém open source i komerčních knihoven, technologií a frameworků, který se v průběhu posledního čtvrtstoletí kolem tohoto jazyka vytvořil. Programátor tak může využít prověřených komponent řešících za něj častou problematiku jako jsou relační mapování, REST služby, Dependency Injection, bezpečnost, persistence dat a v neposlední řadě i jejich vzájemné provázání, a díky tomu se soustředit čistě na psaní business logiky.

Tímto způsobem budu postupovat i já v rámci vývoje mé aplikace a jako zastřešující framework k tomu budu využívat Quarkus běžící na Javě SDK 15.

4.3.1.2 Quarkus

Quarkus je moderní rámec[4] z dílny RedHatu[5] pro vytváření „supersonických a subatomických“[27] Java aplikací šitých na míru cloudovým a obecně kontejnerovým prostředím. Co to znamená?

Java se brzy po svém vzniku stala synonymem pro velké synchronní monolitické aplikace běžící na serverech s dedikovaným operačním systémem, CPU a pamětí (byť později na virtuální úrovni). Nicméně dnešní technologický svět vypadá již velmi odlišně – v souladu s heslem „Anything as a Service“ – software (IaaS), platforma (PaaS), infrastruktura (IaaS) atd., to vše dnes funguje na principu mikroslužeb a serverless služeb žijících v separátních kontejnerech, které jsou typicky nasazeny v cloudu a sdílejí společné prostředky [28]. To na jednu stranu přináší velké zefektivnění hardwarového využití a umožňuje vysokou škálovatelnost, ale na stranu druhou to s sebou nese i dosti odlišné požadavky na programovací jazyky – na první místo je kladena rychlost startu aplikace a co nejmenší paměťová náročnost, kdy je zcela běžné, že se musí během několika stovek milisekund nasadit velké množství nových kontejnerů s novými instancemi dané služby. Něco takového bylo ovšem pro Javu běžící na JVM dlouho nepředstavitelné. A právě Quarkus si vzal za cíl toto změnit.

Quarkus může běžet ve dvou módech:

- **klasickém**
- a **nativním**.

V klasickém módu je Java kompilována do bytekódu a poté intepretována pomocí Java Virtual Machine tak, jak je běžným zvykem. Rozdílem je, že tým stojící za tímto projektem vzal nejpoužívanější frameworky zodpovědné za asi 80 % funkcionality, a optimalizoval je tak, že přesunul maximální možné množství úkolů, které jsou obvykle vykonávány až v době, kdy nastartuje aplikační server, již do doby buildu aplikace. Tím došlo k snížení počtu tříd, které je nutné zavádět, zvýšení rychlosti startu aplikace a omezení paměťových nároků [29].

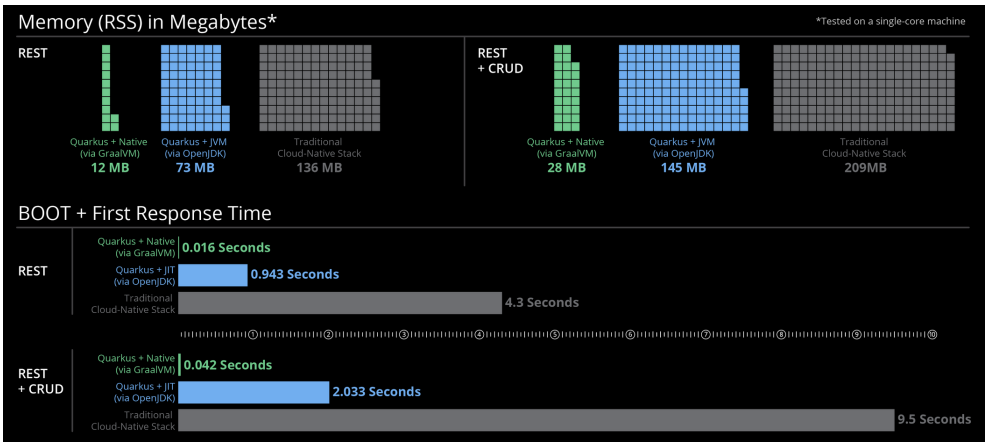
O mnoho zajímavější je druhý mód, při kterém Quarkus za pomoci GraalVM provádí kompilaci nikoliv do bytekódu, nýbrž přímo do nativní spustitelné binárky. Jedná se o náročný proces, při kterém se provádí hluboká statická analýza, jejímž cílem je zjistit všechny potenciální podoby stromu volání, díky čemuž je možné detekovat a zahodit ty části kódu napříč všemi knihovnami, které program ve skutečnosti vůbec nepotřebuje. Binárka pak kromě aplikačního kódu a zredukovaných knihoven obsahuje i nejmenší možnou verzi VM, aby bylo možné provádět „just in time“ kompilaci pro ty části, jež nebylo

[4] V angličtině „stack“ nebo „framework“.

[5] Konkrétně od týmu, který stojí za projektem Hibernate, v čele s Emmanuelem Bernardem, Jasonem Greenem, Bobem McWhirterem a Markem Little.

možné nebo výhodné převést do nativního kódu [30]. Ve spojení s metodou představenou v předchozím odstavci je výsledkem opravdu výrazné snížení doby nutné pro spuštění aplikace a snížení konzumace paměti, jak názorně ilustruje *Obrázek č. 4.1.*

Obrázek 4.1: Doba startu a množství potřebné paměti frameworku Quarkus



A právě možnost nativního módu je hlavním důvodem, proč jsem tento framework zvolil, neboť v případě komerčního nasazení v prostředí, ve kterém se platí za spotřebovaný výkon a paměť, lze tímto způsobem poměrně výrazně snížit náklady na provoz.

Kromě toho se Quarkus usilovně snaží zlepšit programátorský zážitek: například veškerá konfigurace nehledě na množství použitých dílčích frameworků či rozšíření byla sjednocena do jediného souboru. K dispozici je také nadstavba Hibernate frameworku zvaná Panache, která výrazně zjednodušuje práci s entitními třídami a nejběžnějšími databázovými dotazy nad nimi. Co osobně hodnotím velmi kladně, je tzv. „live coding experience“ – jakmile dojde ke změně v kódu, stačí za běhu aplikace obnovit prohlížeč připojený k webového rozhraní, který je nativní součástí mikroslužeb, které framework poskytuje, a Quarkus engine sám identifikuje změněné třídy, které následně překompiluje a provede restart – a to vše zpravidla během několika stovek milisekund. Pro programátory zvyklé na vytváření webové aplikace v PHP nebo Node.js je to vcelku běžná věc; nicméně z hlediska Javy se jedná o malou revoluci [31]. Sám ze své zkušenosti mohu potvrdit, že oproti tradičnímu vývoji v Javě je to velmi příjemný posun vpřed, což je ostatně druhý důvod, proč jsem tento framework zvolil.

4.3.1.3 PostgreSQL

Pro datový management jsem zvolil open source objektově-relační databázový systém PostgreSQL[6] ve verzi 13. Za svou téměř pětadvacetiletou historii[7] získal velmi dobrou pověst stabilního, spolehlivého a bezpečného systému, který je přitom výkonnostně srovnatelný s velkými komerčními řešeními při výrazně subtilnější instalaci [32]. Je také velmi dobře škálovatelný, což je mimo jiné důvodem, proč jej začínají používat i opravdu velké enterprise firmy jako SAP, Comcast nebo Adjust, který v roce 2018 obsluhoval 250 000 databázových požadavků za sekundu nad daty o velikosti více než 4 PB[33].

Z hlediska této práce je zajímavé, že Postgres nabízí také *partitioning* zefektivňující práci nad obrovskými tabulkami rozdělením na menší bloky a procedurální rozšíření PL/pgSQL, které může přijít vhod například v testovacím provozu, kdy jsou častým jevem změny schématu či nutnost opravit velké množství záznamů.

Kromě výše zmíněného existuje ještě jeden velmi prozaický důvod, proč jsem zvolil právě Postgres: je to jeden ze dvou databázových systémů, se kterými mám největší pracovní zkušenosti, přičemž ten druhý je komerční.

4.3.2 Návrh API

Pro získávání dat o ceně monitorovaných nemovitostí bude aplikační vrstva poskytovat následující RESTové rozhraní zasílající odpovědi ve formátu JSON:

- **@GET /rest/oldestdate** vrací nejstarší datum, pro které jsou dostupné statistiky.
- **@GET /rest/latestdate** vrací nejnovější datum, pro které jsou dostupné statistiky.
- **@GET /rest/crei/history** vrací kompletní historii Českého realitního indexu na denní bázi, tj. všechny existující dvojice *datum-příslušná bodová hodnota*.
- **@GET /rest/crei/current** vrací aktuální bodovou hodnotu Českého realitního indexu.
- **@GET /rest/avgprice/history/custom** očekávává dva tzv. `query` parametry:
  - `city`: neprázdná kolekce měst,
  - `estateType`: typ nemovitosti.

[6] zkráceně nazývaný „Postgres“

[7] nepočítaje další desítky let existence Ingresu, z něhož Postgres přímo vychází

Na základě těchto parametrů je vrácena kompletní historie průměrných cen nemovitostí daného typu na denní bázi, a to v závislosti na konkrétním městě. *Výpis kódu 7* ukazuje, jak odpověď na tento dotaz vypadá.

```
[
  {
    "date": "2021-04-25",
    "city": "Praha",
    "value": 120811,
    "id": 0
  },
  ...,
  {
    "date": "2021-04-30",
    "city": "Brno",
    "value": 88513,
    "id": 39
  },
  ...,
  {
    "date": "2021-05-01",
    "city": "Ostrava",
    "value": 42303,
    "id": 115
  },
  ...
]
```

Výpis kódu 7: Odpověď aplikace na dotaz pro získání cenového vývoje

Kromě toho bude pro testovací fázi k dispozici i rozhraní pro práci s nemovitostmi, abych mohl získávat, vytvářet, měnit a mazat v případě potřeby data bez přímého přístupu k databázi – protože ale neplánuji, že by toto API mělo být součástí produkčního kódu, nebudu jej zde podrobně rozebírat.

4.3.3 Návrh backendové architektury

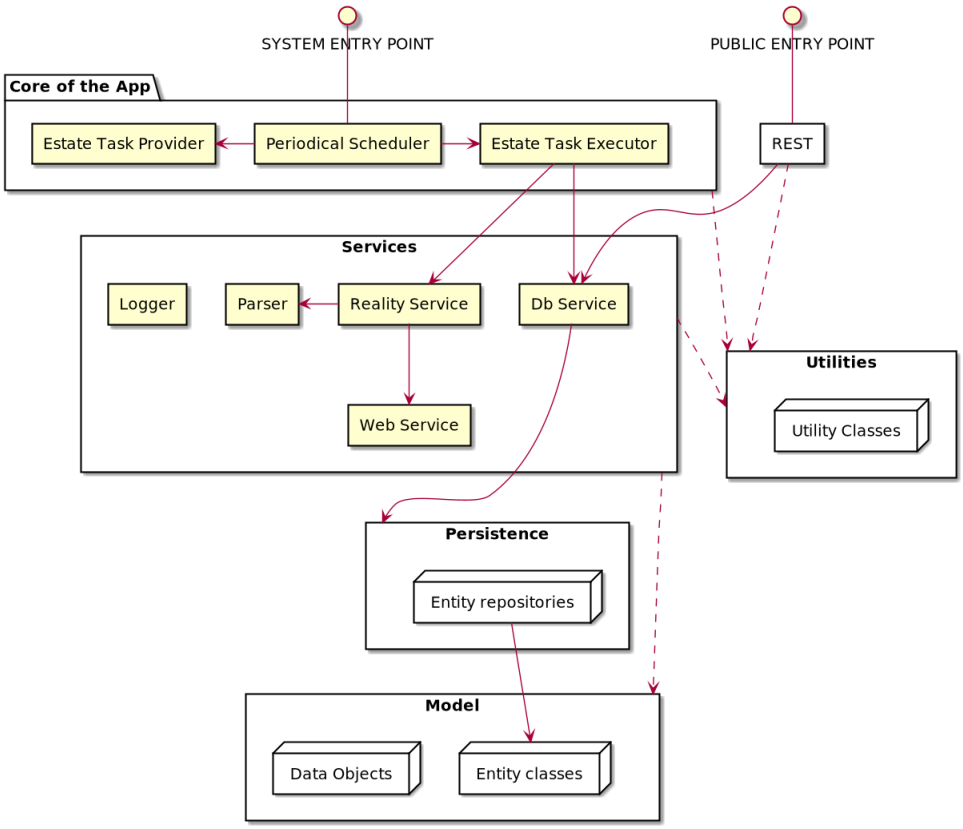
„*Vůbec nejdůležitějším faktorem, který odlišuje dobře navrženou komponentu od mizerně navržené, je to, do jaké míry daná komponenta izoluje interní informace od svého okolí. Dobře navržená komponenta skrývá veškeré implementační detaily a čistě odděluje API od jeho implementace.*“ [34]

Tento postulát Joshua Blocha, mimochodem jednoho z nejvýznamnějších autorů Javy, je základním principem, ze kterého chci při návrhu backendu postupovat. Sám ze své pracovní zkušenosti vím, jak obtížné až nemožné je pracovat s projektem, který vyrostl do obřích rozměrů tomuto principu navzdory. Striktní oddělení API a implementace umožňuje rozdělit implementační logiku do různých modulů bez velkých závislostí mezi nimi, a tím vytvářet skutečně efektivní vrstevnaté a tedy i škálovatelné a mnohem lépe udržovatelné aplikace.

4.3.3.1 Konceptuální model

Než přejdeme k návrhu konkrétních tříd, pojďme se zamyslet nad obecnějším konceptem backendové architektury - viz *Obrázek č. 4.2.*

Obrázek 4.2: Konceptuální model



Jak vidno, počítám s šesti většími celky představující určité obecnější „business domény“:

- **Srdcem aplikace** bude periodicky se spouštějící plánovač, který si vyzvedne seznam všech typů nemovitostních nabídek, které je potřeba

stáhnout z realitního serveru, a pro každý z nich naplánuje nezávislé provedení, které následně zajistí komponenta pracovně zvaná `Estate Task Executor`.

- Další celek budou představovat **servisní komponenty**. Každá z nich má přesně vymezený účel – komponenta `Reality Service` bude znát detaily rozhraní používaného realitního serveru a ve spolupráci s `Webovou` a `Parsovací` komponentou bude schopna získat všechny požadované nemovitostní nabídky. Kromě toho zde bude existovat `DB Service` komponenta, která bude poskytovat abstrahované API pro přístup k datům. Interně bude využívat funkcionality Persistenční komponenty. V neposlední řadě zde bude i komponenta provádějící logování – očekávaně ji budou používat všechny ostatní servisní třídy, což jsem pro větší přehlednost v diagramu záměrně neuvedl.

- Do samostatné komponenty vyčlením **persistenční logiku**, neboť je přeci jen do velké míry závislá přinejmenším na typu zvolené databáze a případně i použitého frameworku a je zcela jistě rozumné odstínit ji od zbytku aplikace, jak je to jen možné.

- **Modelová komponenta** bude sdružovat reprezentaci datových struktur používaných v rámci aplikace - typicky všechny entitní třídy a další datové objekty.

- Separátně definovaná bude též komponenta zajišťující **RESTové služby**. Ze servisních tříd bude mít očekávaně vazbu jen na databázové služby, případně utilitní třídy.

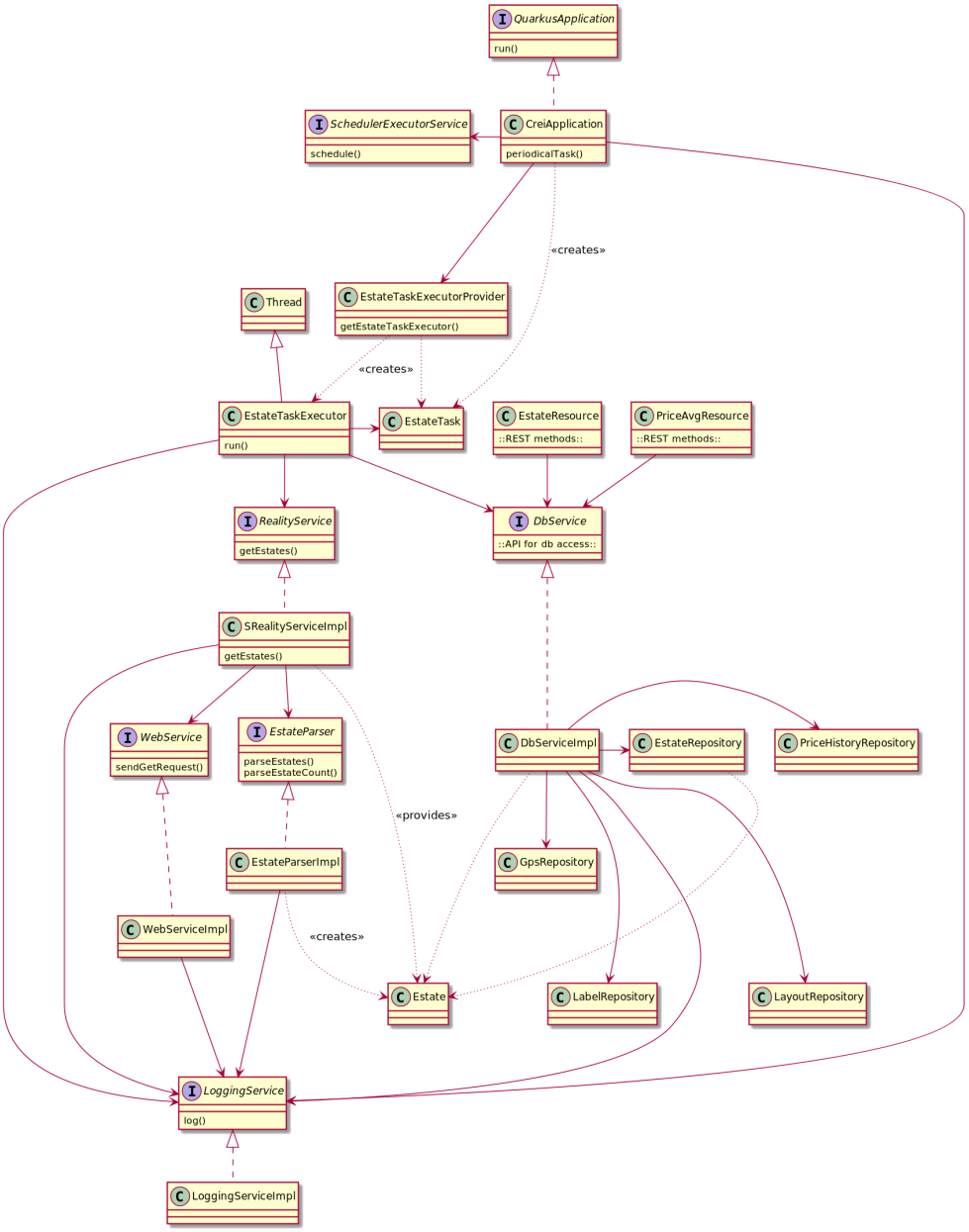
- Poslední jmenovanou je **Utilitní komponenta**, která bude skrze statické metody nabízet drobnou funkcionalitu potřebnou napříč celým systémem, pro kterou nedává smysl vytvářet abstrakci ani samostatnou doménu.

4.3.3.2 UML diagram tříd

Obrázek č. 4.3 zachycuje zjednodušený UML diagram tříd. (Pro větší přehlednost a kompaktnost jsou vynechány některé méně důležité třídy a enumy – například z entitních tříd je zastoupena jen `Estate`).
Vysvětleme si trochu podrobněji s tímto diagramem před očima základní principy fungování nejdůležitějších komponent, potažmo celé aplikace:

- **`Estate`** je entitní třídou implementující stejnojmennou databázovou tabulku, a reprezentuje tedy jednu konkrétní nemovitost (nebo jednu konkrétní nabídku nemovitosti, chcete-li).

Obrázek 4.3: Zjednodušený UML diagram tříd



- **`EstateTask`** představuje jednu konkrétní kategorii nabídky nemovitostí[8], kterou je potřeba z realitního serveru získat.
- **`CreiApplication`**: Jakožto interní vstupní bod celé aplikace musí dle pravidel Quarkus frameworku implementovat rozhraní

[8]neboli unikátní kombinaci **lokality**, **typu** a **dispozice** nemovitosti

`QuarkusApplication`, které předepisuje jedinou metodu `run()`, jenž se spouští právě při startu aplikace.

Středobodem je však metoda `periodicalTask()`, která se každý den ve stejný čas automaticky zavolá a vykoná dvě důležité věci:

- vytvoří seznam všech úkolů, které je potřeba provést, reprezentovaných instancemi třídy `EstateTask`
- a ve spolupráci s třídou `EstateTaskExecutor` reprezentující vlákno a rozhraním `java.util.concurrent.ScheduledExecutorService` zajistí nezávislé provedení každého úkolu v náhodný čas z definovaného intervalu.

- **`EstateTaskExecutor`** v sobě obsahuje právě jeden `EstateTask` a v momentě, kdy je toto vlákno spuštěno, získá přes API `RealityService` všechny nemovitosti (`Estate` objekty) dané kategorie, které následně přes API `DbService` uloží do databáze.

- **`RealityService`** je obecné rozhraní pro zprostředkování komunikace s realitními servery. Myšlenka je taková, že každý realitní server bude reprezentován samostatnou implementací. Momentálně existuje tedy jediná implementaci v podobě `SRealityServiceImpl` třídy. Jak název napovídá, zprostředkovává získání nemovitostí z SReality serveru – potřebuje tedy znát detaily o jeho API, čímž je na něm přímo závislá. Pro svoji činnost využívá API `WebService` a `EstateParser`.

- **`WebService`** nabízí obecné API pro zasílání webových požadavků a má jedinou implementaci `WebServiceImpl`, která závisí pouze na logovací funkcionalitě – jedná se tedy o velmi dobře izolovanou komponentu.

- **`Estate Parser`** poskytuje rozhraní pro extrakci `Estate` objektů z těla HTTP odpovědi. Jedná se o obdobný případ jako u výše zmíněného interface `RealityService`. I zde platí, že každá implementace musí být nutně závislá na zvoleném realitním serveru, neboť každý realitní server má specifický formát odpovědi. Jak název `SRealityEstateParserImpl` naznačuje, momentálně existuje jediná implementace pro parsování odpovědí z SReality serveru.

- **`DbService`** interface nabízí rozsáhlé API pro práci s databází. Kromě ukládání a získávání všech entitních tříd, případně i jejich kolekcí, umožňuje například i získání průměrných cen den po dni.

  Implementace `DbServiceImpl` k databázi nepřistupuje přímo, ale je od ní odstíněná ještě přes jednou vrstvu, kterou zde představují jednotlivé **repozitáře**. Ty jsou závislé na obecném typu databáze (v tomto případě SQL) a také použitém „`Hibernate with Panache`“ ORM frameworku – případné budoucí zásadní změny v oblasti databáze (například přechod

na NoSQL) či změny ORM frameworku se tak budou týkat v ideálním případě jen repozitářů. V tom nejhorším bude třeba provést změny i v `DbServiceImpl`, nicméně samotné rozhraní `DbService` zůstane beze změny.

- `LoggingService` je rozhraní využívané zbytkem systému pro logování a má jednoduchou implementaci `DbService` v tuto chvíli bez závislostí na dalších komponentách systému.

- Posledními stojícími za zmínku jsou tzv. **RESTful Resource třídy**: `EstateResource` a `PriceAvgResource`, které zprostředkovávají práci s RESTovými zdroji. Každá metoda v rámci těchto tříd je anotována unikátní kombinací URL mappingu a typu HTTP requestu, čímž obě třídy definují **REST API**, přes které lze s backendovou aplikací komunikovat z okolního světa – právě přes něj bude získávat potřebná data frontendová část.

4.3.4 Databázové schéma

Databázové schéma (viz *Obrázek č. 4.4*) počítá s existencí následujících databázových objektů:

- `Estate` je vůbec nejdůležitější tabulkou. Reprezentuje klíčové vlastnosti inzerované nemovitosti. Povinnými údaji jsou:

  - město neboli lokalita (`city`),
  - druh nemovitosti (`estateType`),
  - cena (`price`),
  - jednoznačný identifikátor v rámci Sreality serveru (`srealityId`),
  - velikost nemovitosti (`size`),
  - a data, kdy byla nemovitost poprvé zaregistrována a kdy byla naposledy evidovaná v nabídce (`timeCreated`, resp. `timeLastActive`).

    Kromě toho obsahuje řadu nepovinných atributů, z nichž některé jsou dopočítávány – například průměr ceny za čtvereční metr (`pricePerSquareMeter`) nebo vzdálenost od centra (`distanceFromCenter`).

    Specifickou skupinu informací představují samostatné tabulky, se kterými je tabulka `Estate` svázána přes cizí klíče, viz níže. Tuto formu jsem zvolil především z důvodu větší úspornosti databáze – jedná se vesměs o subjekty nabývající omezené množiny hodnot, a tak se vyloženě nabízí ukládat je do zvláštní tabulky a tu provázat s tabulkou `Estate` pomocí vztahu `1 : N`.

- Tabulka **Layout** představuje dispozici nemovitosti. Nejvíce smysluplná je v případě bytů a pak rodinných domů, kde se využívá údaj o počtu místností. Naopak pro pozemky se neuplatňuje vůbec.

- Název tabulky **Gps** mluví sám za sebe – reprezentuje souřadnice, na kterých se nemovitost nachází.

- Tabulka **Label** zaznamenává významné štítky, které se s danou nemovitostí pojí. (Zda jde o novostavbu, zda má garáž či balkón atd.) Protože takových štítků může mít nemovitost mnoho, konceptuálně je tabulka **Label** s tabulkou **Entity** ve vztahu **M : N**. Pro implementaci tohoto vztahu si tedy pomůžeme obvyklou berličkou používanou v relačních databázích: asociační tabulkou **entity_label**, která je vůči oběma tabulkám ve vztahu **N : 1**.

- **Pricehistory** tabulka vznikla jako reakce na poznatek, že cena konkrétní nemovitosti nemusí být fixní – naopak se v průběhu času může hýbat oběma směry. Aby měl cenový index opravdu relevantní výpovědní hodnotu, je tedy nutné změnu ceny sledovat a uchovávat.

4.3.4.1 Databázové indexy

Databázový index je struktura typicky implementovaná jako B-strom, která slouží k rychlejšímu a efektivnějšímu provádění dotazů nad databázovými tabulkami.

Vzhledem k faktu, že v rámci běhu aplikace se bude nad databází provádět velké množství podobných dotazů, rozhodl jsem se hned od začátku vytvořit několik databázových indexů, které přispějí k rychlejšímu běhu nejčastějších z nich.

Pro tabulku **Estate** to bude:

- Index nad sloupečkem **srealityid**, neboť při vytváření či aktualizaci nemovitosti se musí přes tento identifikátor nejdříve zjistit, zda-li již v databázi není.

- Kompozitní index nad sloupečky (**timecreated**, **timelastactive**, **estatetype** a **city**), který pokrývá hned několik typických *n-tic* objevujících se v klauzuli **WHERE**.

A pro tabulku **Pricehistory**:

- Index nad sloupečkem **estate_id**, čímž významně urychlíme zjišťování cenových změn pro danou nemovitost.

4.4 Frontend

Dnešní webový frontend je nejčastěji vytvářen kombinací tří jazyků:

- **HTML**, v němž je psán v zásadě veškerý statický text;
- **CSS** definující grafické styly textu a statických prvků
- a **JavaScript** zajišťující vykreslování dynamického obsahu včetně grafických animací.

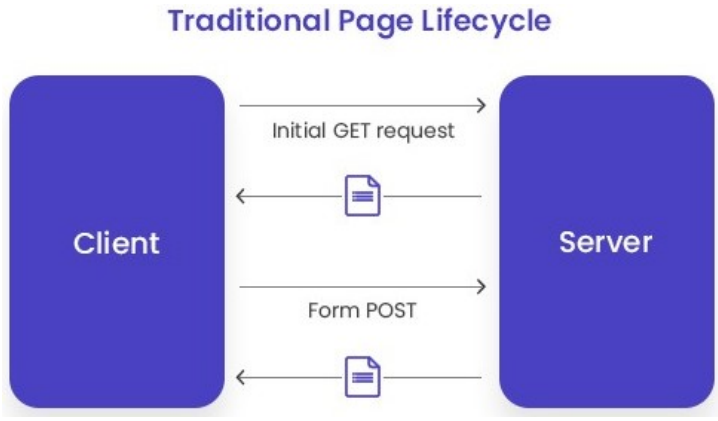
Tyto jazyky budu využívat i pro webovou část této aplikace.

4.4.1 Technologie

4.4.1.1 Single Page vs. Multi Page

Dřívější způsob, kterým fungovaly webové stránky, spočíval v principu, kdy při každé změně vyžadující změnu stránky došlo k zaslání a načtení zcela nové stránky. Takovéto aplikace jsou označovány jako **Multi Page**.

Obrázek 4.5: Tradiční neboli Multi Page webová aplikace



Zdroj: [35]

Naopak trendem posledních let jsou tzv. **Single Page** aplikace. Ty jsou založeny na myšlence, že jako odpověď na úvodní request server zašle všechny šablony a grafické prvky i se skripty (typicky psané v jazyce JavaScript), které jsou nutné pro vykreslení stránky. V případě, že uživatel provede nějakou akci, která vyžaduje nová data (např. rozkliknutí emailu), pošle se asynchronní dotaz, jehož odpověď obsahuje jen konkrétně vyžádané informace nejčastěji ve formátu JSON. Skripty se pak postarají o aktualizování a překreslení příslušného prvku, přičemž zbytek stránky zůstává beze změny.

Obrázek 4.6: Single Page webová aplikace

Zdroj: [35]

4.4.1.2 Oracle JET

Protože jsem backendový vývojář a s grafikou se příliš nepřátelím, hledal jsem nástroj, ve kterém by bylo možné vytvořit moderně vypadající frontend pokud možno co nejefektivnějším způsobem. Řešení jsem nalezl v podobě Oracle JavaScript Extension Toolkitu (zkráceně Oracle JET), což je modulární nástroj založený na kolekci javascriptových knihoven a frameworků jako jsou například Node.js, Knockout.js, jQuery UI, Require.js a další. Jak říká John Brock, jeden ze zakladatelů tohoto projektu: „Cílem bylo poskytnout nástroj, pomocí kterého je možné vyvinout grafické rozhraní s tak velkým, anebo naopak tak malým úsilím, jak programátorovi vyhovuje [36].“

Součástí projektu je i výukový portál s příklady a repozitář předpřipravených komponent, ze kterých je možné grafickou aplikaci vystavět bez nutnosti psát veškerý kód „od nuly“.

Oracle JET je distribuován pod open source Universal Permissive licencí (UPL), a je tedy možné jej používat prakticky neomezeným způsobem i pro komerční účely [37].

4.4.2 Wireframes

Wireframe, do češtiny poněkud otrocky překládané jako „drátěný model“, je primitivní skica webu, která se používá k vytvoření základní představy o tom, jak bude vypadat struktura stránky a rozmístění prvků bez toho, aniž by se řešila konkrétní grafická podoba.

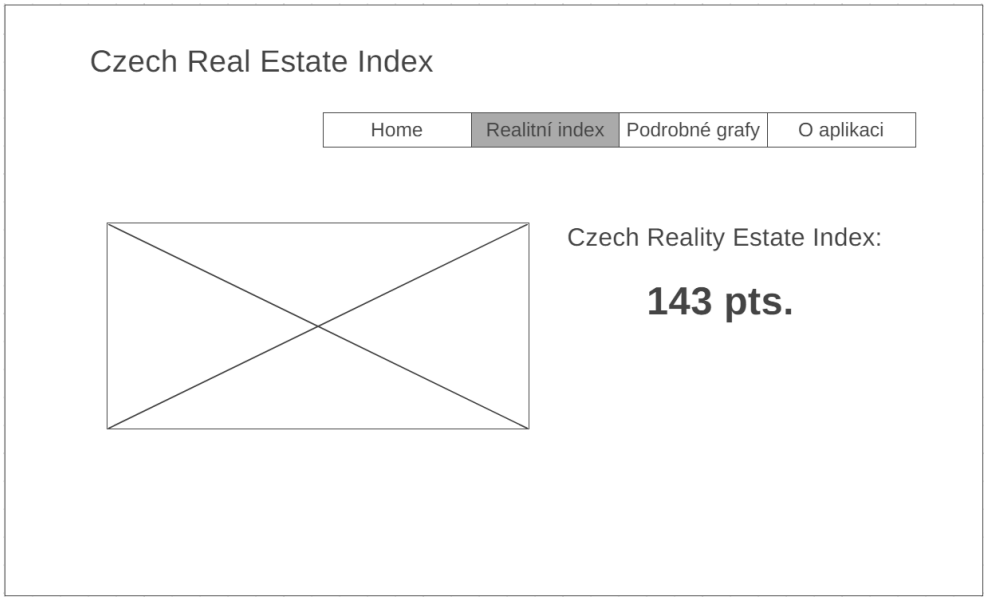
Jednoduchá single page webová aplikace bude obsahovat několik podsekcí.

Obrázek č. 4.7 ukazuje, jak bude přibližně vypadat úvodní obrazovka.

Obrázek 4.7: Wireframe: Úvodní obrazovka

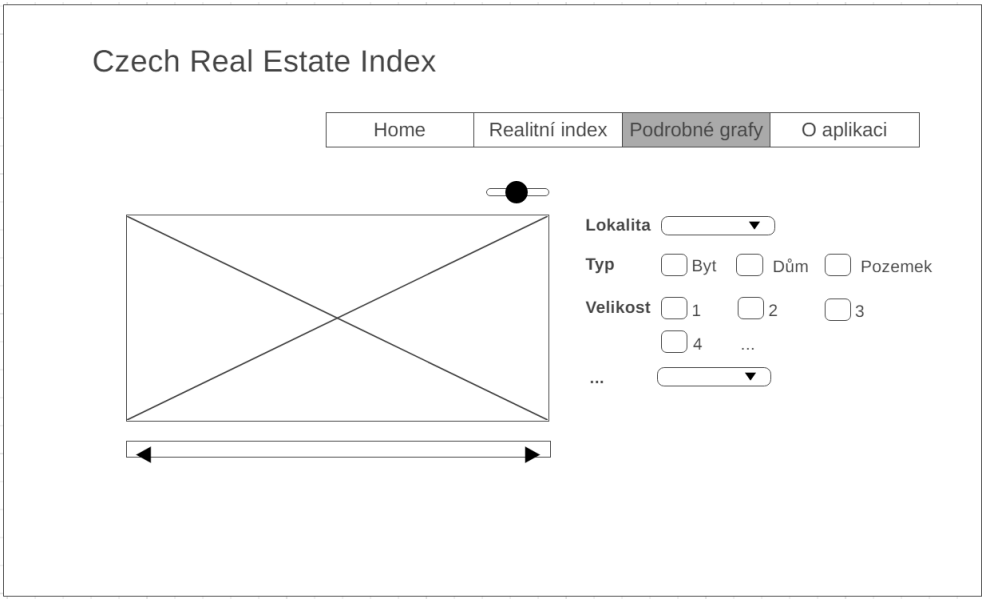
Po kliknutí na „Realitní index“ se vykreslí graf zachycující vývoj indexu v čase a vedle něj bude zobrazena jeho akutální bodová hodnota, viz obr. č. 4.8.

Obrázek 4.8: Wireframe: Realitní index



Poslední wireframe (*Obrázek č. 4.9*) zachycuje interaktivní sekci, která nabízí možnost vykreslit graf vývoje ceny v závislosti na uživatelem zadaných parametrech.

Obrázek 4.9: Wireframe: Customizovaný graf



Konečná podoba se může samozřejmě lišit – bude záležet především na tom, jak bude stránka z hlediska estetiky a designu „fungovat“, až bude oblečena do finálního grafického hávu. Nevylučuji ani alternativu případného přesunu části „O aplikaci“ do zápatí, případně vznik dalších sekcí.

4.5 Slabé stránky navrženého řešení

Achillovou patou aplikace je významná závislost na Sreality serveru. V případě, že se změní API nebo se z nějakého důvodu služba stane nepoužitelnou, bude třeba přepsat část kódu, resp. představit kód zcela nový. To je ale riziko, které nelze jakkoliv ovlivnit; bude zde přítomno vždy bez ohledu na zvolený realitní server. Lze však minimalizovat potenciální negativní následky – a přesně o to jsem se snažil v rámci návrhu softwarové architektury. Díky striktnímu oddělení API od implementace bude změna realitního serveru znamenat víceméně jen novou implementaci `RealityService` a `EstateParser` rozhraní a výčtových tříd, které definují podobu a význam parametrů pro API realitního serveru.

V nejhorším možném případě se tak bude změna týkat přibližně 5 % kódu, jakkoliv důležitého, nicméně bude to změna velmi izolovaná a zbytek aplikace ji nikterak nepocítí.

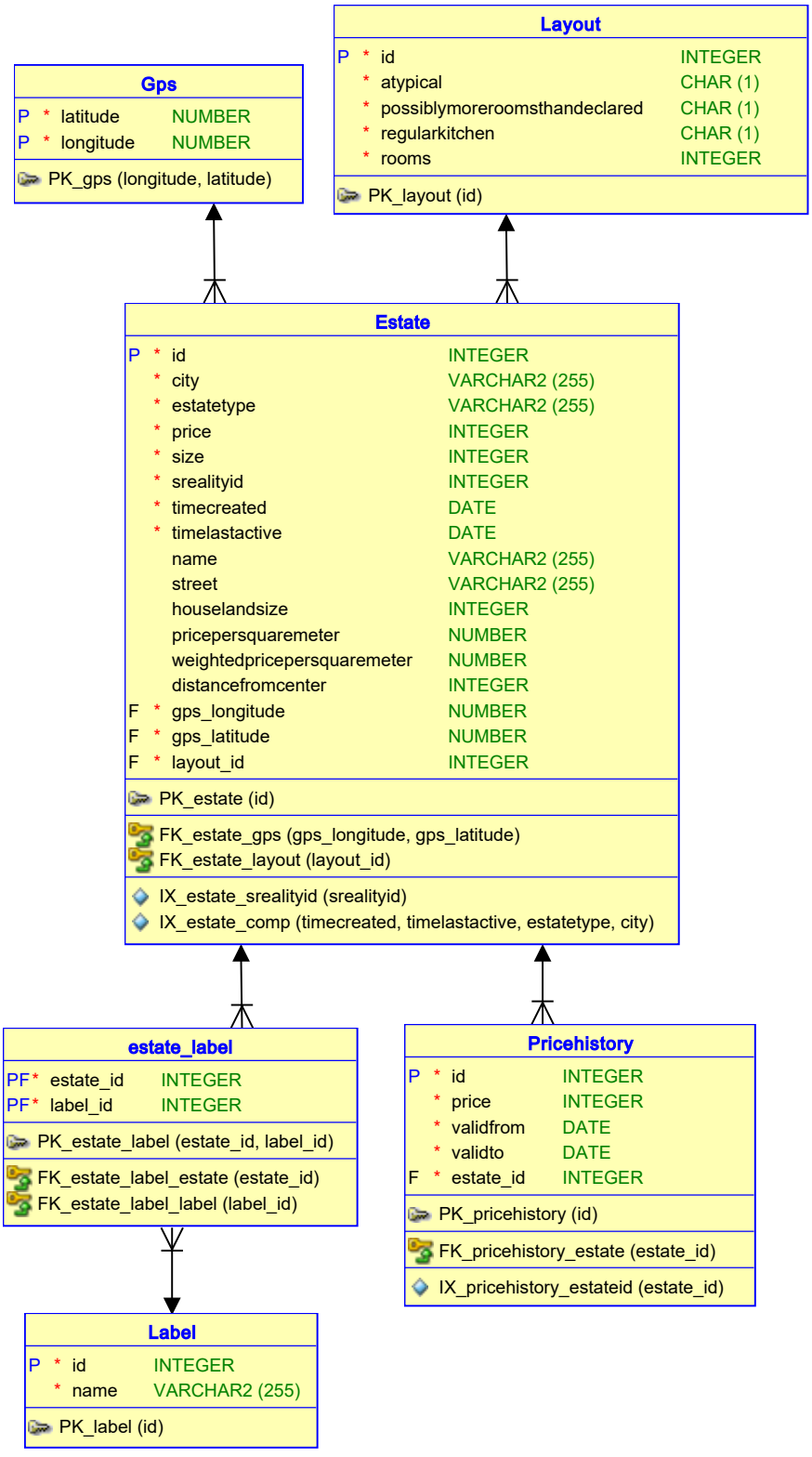
Další slabinou je fakt, že HTTP požadavky na realitní server přichází z jedné a té samé IP adresy. Ačkoliv se jejich počet snažím držet na co nejnižších číslech (vyšší desítky denně náhodně rozložených do několika hodin),

přesto existuje šance, že bude komunikace z této IP adresy zablokována. V tom případě se nabízející minimálně tato čtyři řešení:

- použít VPN, kterou mám k dispozici v rámci svého zaměstnání (eticky problematické),
- využít některý z bezplatných VPN serverů (nepříliš spolehlivé a deterministické řešení),
- použít placenou VPN službu
- nebo v případě provozu na lokálním serveru využít k internetovému připojení služeb některého poskytovatelů, kteří při každém připojení přidělují novou IP adresu (typicky mobilní operátoři).

Nejlevnější placenou variantou se jeví ta předposlední – konkrétně to znamená nízké stokoruny měsíčně, což je pro mě v případě nutnosti přijatelná varianta.

Obrázek 4.4: Relační databázové schéma



Kapitola 5

REALIZACE

5.1 Nasazení

V současné chvíli je aplikace nasazena na mém osobním počítači. Mým plánem je sbírat následujících přibližně 6 měsíců data, a poté ji zpřístupnit široké veřejnosti, neboť ceny nemovitostí mají přeci jen poměrně velkou setrvačnost a pokud bych měl zobrazovat vývoj indexu nyní – za těch několik týdnů, pro které mám nasbíraná data –, graf by ze všeho nejvíc připomínal přímku rovnoběžnou s osou x, což zcela jistě není informace hodná publikování, jak je ostatně vidět na Obrázku č. 5.1.

Za účelem zpřístupnění veřejnosti jsem koupil doménu `crei.cz`[9], na které chci webovou aplikaci provozovat.

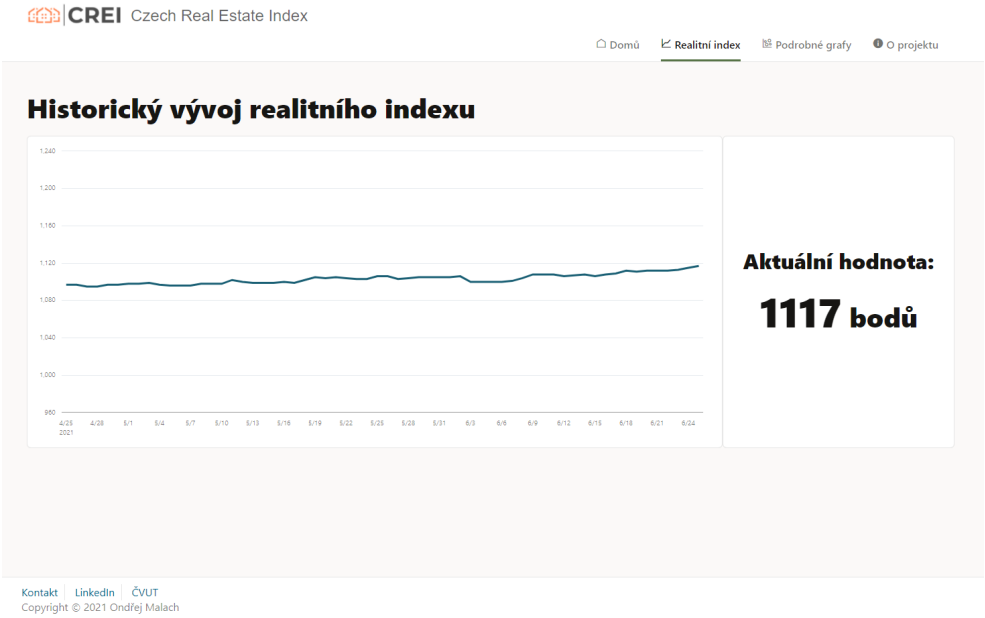
Co se týče hostingu, rád bych využil faktu, že backend, frontend i databáze mohou běžet v kontejnerech a aplikaci nasadil do cloudového prostředí fungujícím na přístup „Pay As You Go“. Existují dokonce i cloudové služby, které jsou zdarma, nicméně i zde jsem ochoten v případě potřeby platit částku ve výši stokorun měsíčně, za kterou lze získat službu výrazně přesahující mé potřeby.

5.2 Testování

Při psaní aplikace jsem se snažil do jisté míry aplikovat metodu zvanou „Test Driven Development“, kdy se ze všeho nejdříve napíšou testy, které musí logicky končit chybou, a až poté vzniká produkční kód v co nejmenším rozsahu, se kterým testy „projdou“, a na tomto základě se staví komplexnější konstrukce. Obzvlášť při psaní modulů zodpovědných za komunikaci z Sreality serverem, resp. parsovací funkcionalitu mi tento postup ušetřil mnoho drahocenného času a přepisování kódu.

[9] CREI je zkratkou pro Czech Real Estate Index

Obrázek 5.1: Grafická podoba finálního řešení



Z významné většiny jsem vytvářel unit testy, které považuji za nutný základ, neboť jak říká Joshua Bloch: „Jestliže je nemáte, zkrátka netušíte, jestli váš kód funguje [38].“

Druhou největší skupinou byly manuální testy, protože v jakékoliv aplikaci, která pracuje s otevřenou množinou dat, vždy existuje větší či menší množství neočekávaných scénářů, které automatizované testy nezachytí a často je zachytit ani nemůžou. V rámci testovacího provozu, který trvá v době dokončování této práce již víc než měsíc, jsem tak intenzivně pracoval s logováním a manuálními dotazy nad databází, abych objevil příčiny chyb, které se během této doby objevily.

Nejmenší užitek z mého pohledu nakonec představovaly integrační testy, a to především z toho důvodu, že bylo nutné použít v určitém rozsahu *mockování*, což z principu snižuje relevanci testování. Přesto jich několik používám k testování komplexní funkcionality produkční implementace veřejného API rozhraní `RealityService`, která komunikuje s několika dalšími moduly. Aby při tomto procesu nedocházelo k obvyklému odesílání HTTP požadavků na realitní server, „podstrkuji“ v testovacím prostředí speciální implementaci `WebService` rozhraní, které vrací předpřipravenou HTTP odpověď.

5.3 Dokumentace

Dokumentace projektu je součástí přiloženého datového nosiče a skládá se z následujících tří kategorií:

5.3.1 Technická dokumentace

Popisuje, jak nainstalovat a spustit backend i frontend. Kromě toho podrobně dokumentuje RESTové rozhraní aplikační vrstvy.

5.3.2 Javadoc

Jedná se o dokumentaci vytaženou přímo ze zdrojového Java kódu backendové aplikace prezentovanou v HTML formátu. Pro všechny veřejné a některé privátní metody popisuje, co je jejich náplní, jaké parametry přijímají a které hodnoty vrací.

5.3.3 Popis grafického rozhraní

Přestože jsem se snažil, aby byl frontend velmi přímočarý, přehledný a intuitivní, v rámci dobrých mravů jsem k dokumentaci připojil i stručný popis grafického rozhraní.

5.4 Vyhodnocení požadavků

5.4.1 Funkční požadavky

- Požadavky F1 a F2 – neboli stahování a ukládání nabídky nemovitostí – jsou fakticky hlavní náplní backendové aplikace. Jejich splnění je technicky zajištěno třídami `SRealityServiceImpl` a `DbServiceImpl`.

- Splnění požadavku F3, který definuje seznam údajů, které je nutné pro každou nemovitostní nabídku evidovat, je principiálně vyřešeno implementací navrženého databázového schématu. O vytváření historie cenových změn se stará `DbServiceImpl` v rámci metody `createOrUpdateEstates()`.

- F4 – přístup k realitnímu serveru v pseudonáhodných časech – je zajišťován poměrně komplexní funkcionalitou. V třídě `CreiApplication` jsou definovány „přípustné“ hodiny v rámci dne, kdy je aplikaci povoleno stahovat nemovitostní nabídky. Periodicky se spouštějící plánovač pak pro každý typ nemovitostní nabídky náhodně vybere z tohoto seznamu jednu položku, k ní vygeneruje náhodný počet minut od 0 do 59 a náhodný počet sekund ve stejném intervalu.

- Požadavek F5 předepisující simulaci přístupu z různých zařízení je naplněn použitím hlavičky „User-Agent“ pro každý HTTP request. Obdobně jako v předchozím případě plánovač dosazuje pro tuto hlavičku náhodnou hodnotu z definovaného seznamu `UserAgentConstants`.

- F6 vyžadující dodržování dostatečných pauz mezi jednotlivými HTTP požadavky je zajištěn jednak uspáváním vlákna po náhodnou dobu z určitého intervalu, a jednak zavedením jednovláknového poolu pro javovský `ScheduledExecutor` využívaný pro stahování nemovitostních nabídek, abychom předešli paralelnímu souběhu dvou různých vláken.

- Skupina požadavků F7 se týká RESTových služeb. Ty jsou v rámci aplikace zajištěny pomocí RESTEasy frameworku, což je jedna z implementací Jakarta RESTful Web Services (JAX-RS) specifikace. Zdroje vyžadované v rámci F7A, F7B a F7C jsou implementovány endpointy `GET crei/history`, `GET crei/current` a `GET avgprice/history/custom`.

- Požadavek F8 na JSON formát je vyřešen za pomocí anotace `@Produces` z výše zmíněné JAX-RS specifikace.

- Požadavek na grafické webové rozhraní F9 byl naplněn přesně dle návrhu frontendu v předchozí kapitole, tj. pomocí frameworku Oracle JET.

- Funkční požadavky F10 a F11 byly implementovány s použitím Oracle JET komponent `oj-chart`, `oj-checkboxset` a `oj-radioset` které pracují s daty poskytovanými RESTovými endpointy definovanými skupinou požadavků F7.

5.4.2 Nefunkční požadavky

- Požadavky N1 a N2 definující podporované platformy a prohlížeče jsou splněny již samotným výběrem frontendového frameworku, neboť Oracle JET aplikace nativně podporují PC platformu a všechny významné prohlížeče, a to i ve výrazně starších verzích, než je vyžadováno [39].

- Požadavky N3 a N4 zabývající se propustností, resp. dobou odezvy spolu do jisté míry souvisí. Pokud chceme zjistit, kolik požadavků za časovou jednotku aplikace dokáže obsloužit, je třeba počítat s nejhorším možným případem, a k tomu je třeba znát časy odezvy jednotlivých požadavků. Zavedením databázových indexů se podařilo ustálit dobu vyřízení nejnáročnějších RESTových požadavků okolo 250 ms. To znamená, že i v nejhorším případě aplikace obslouží 240 požadavků za minutu. Protože však RESTEasy framework nativně podporuje paralelní zpracování, ve skutečnosti je tento limit mnohonásobně vyšší. Ale i v případě sériového zpracování jsou oba požadavky naplněny s velkou rezervou.

- N5 hovoří o požadavku na škálovatelnost. To je z principu ne zcela exaktně měřitelné hledisko, jehož platnost se zpravidla ukáže až v konkrétní praktické situaci. Nicméně obecně lze říci, že dobrá škálovatelnost je v prvé řadě umožněna vhodným návrhem architektury, kde jsou jednotlivé části co nejvíce izolovány – a přesně o to jsem se v rámci této

práce snažil. Dalším důležitým faktorem je nativní schopnost kontejnerizace Quarkus aplikací. V případě budoucího zvýšení zátěže je tak možné například duplikovat některé z komponent do samostatných kontejnerů.

- Požadavek N6 byl naplněn použitím HTTPS protokolu. V tuto chvíli je používán self-signed certifikát, který bude nahrazen certifikátem vydaným některou z respektovaných autorit v momentě, kdy bude aplikace veřejně publikována.

5.5 Problémy v praxi

Během testovacího provozu aplikace jsem narazil na několik zajímavých problémů:

5.5.1 Měnící se údaje

Ukazuje se, že realitní kanceláře spravující své inzeráty na Sreality serveru jsou poměrně kreativní, co se týče přístupu k parametrům, u kterých je přirozené očekávat, že by měly zůstat konstantní. Stává se tak například, že udávaná velikost nemovitosti v průběhu času o pár metrů naroste. Podle mého pozorování se zpravidla správce rozhodne do výměry bytu zahrnout i sklep či balkón nebo dokonce podíl na společných prostorech. Mohu se pouze domnívat, zda je to proto, aby se nemovitost jevila atraktivnější, či z nějakého jiného důvodu.

Protože na základě dat, která v současné chvíli monitoruji, nelze odhadnout, zda jsou korektnější údaje původní nebo upravené, přiklonil jsem se k řešení, kdy při každé změně aktualizuji parametry nemovitosti v databázi tak, aby odpovídaly aktuálnímu stavu. Pozitivem každopádně je, že k takovýmto změnám nedochází příliš často – řádově jednou za několik stovek inzerátů.

5.5.2 Duplicitní nemovitosti

V určitý okamžik jsem si všimnul, že se v databázi hromadí velmi podobné nemovitosti, které se liší jen v drobnostech - např. nepatrně odlišných GPS souřadnicích nebo právě výše zmíněné udávané velikosti. Souviselo to se způsobem, jak jsem definoval, že dvě nemovitosti mají být považovány za shodné. Protože jsem se na počátku implementace domníval, že by nebylo příliš rozumné spolehnout se na to, že parametr `hash_id` používaný serverem Sreality bude opravdu vždy vztažen jen k jediné nemovitosti a nebude nikdy recyklován, rozhodoval jsem o duplicitě na základě několika parametrů: kromě zmiňovaných GPS souřadnic a `hash_id` to byly například i velikost, typ a dispozice nemovitosti.

Nicméně po objevení problému s duplicitami jsem provedl analýzu nasbíraných dat (řadově několik tisíc nemovitostí) a nenašel jsem jediný případ,

kdy by dvě fundamentálně odlišné nemovitosti sdílely stejné `hash_id`. Především jsem ale po podrobném prostudování dokumentace Sreality pro importní rozhraní určené realitním kancelářím narazil na informaci, že identifikátory s koncovkou `_id` jsou „interní id v databázi Seznamu – unikátní v celém univerzu“ [20]. Rozhodl jsem se tedy posuzovat duplicitu nemovitostí jen pomocí tohoto identifikátoru.

Abych zachoval relevantnost dat a zároveň nepřišel o nasbírané inzeráty za několik týdnů, sjednotil jsem duplicitní data do jediného záznamu pomocí PL/pgSQL skriptu spuštěného nad celou tabulkou Estate.

5.5.3 Chybné údaje

Dalším zajímavým problémem se ukázal být fakt, že na některé důležité údaje se nedá bezvýhradně spolehnout: například některá realitní kancelář občas publikuje v sekci *Prodej* i nemovitost nabízenou k pronájmu. Tím dochází k velkému zkreslení průměrných cen kvůli tomu, že se cena myšlená jako měsíční nájem bere jako prodejní cena nemovitosti. Objevuje se ale i opačný extrém způsobený obyčejným překlepem – v nabídce se tak objeví třeba panelákový byt o rozloze 45 m^2 s cenou 193 500 000 Kč[10].

Řešení, které jsem zvolil, je, že před uložením nemovitosti do databáze nejdříve spočítám průměrnou cenu za čtvereční metr – pokud je o víc než 500 % vyšší nebo o 80 % nižší než je průměr v dané lokalitě, takovou nemovitost „zahodím“ a do databáze ji tedy neukládám, pouze ji zaloguji pro případné manuální posouzení.

5.6 Možnosti dalšího rozvoje

Během tvorby aplikace jsem nashromáždil množství nápadů na budoucí rozšíření, a v rámci této kapitolky chci zmínit ty, které se mi i s odstupem času jeví jako smysluplné.

Logickým krokem je **rozšíření počtu sledovaných lokalit** o další významná česká města jako Olomouc, Jihlavu, Zlín, Plzeň, Pardubice, Ústí nad Labem a další.

Dalším možným vylepšením je umožnit sestavování grafů nad specifickými kategoriemi – např. závislost ceny na vzdálenosti od centra či filtrování nemovitostí dle stáří či materiálu, ze kterého jsou postaveny.

Z nasbíraných dat by bylo též možné sestavovat zajímavé statistiky typu jak dlouho průměrně trvá, než nemovitost zmizí z nabídky, či jaké typové vlastnosti mají nejprodávanější nemovitosti v jednotlivých lokalitách.

[10] Ačkoliv v tržním prostředí je zcela legitimní nabídnout libovolnou cenu, tady se nelze ubránit dojmu, že cenovka obsahuje alespoň dvě nuly navíc.

Zásadní rozšíření by představoval systém uživatelských účtů, který by například umožňoval zasílání notifikací v případě, že ceny v určité oblasti dosáhnout konkrétní cenové hladiny.

V hlavě mám i nápady ohledně rozšíření, které se bude týkat čistě interních mechanismů aplikace – přemýšlím o přidání mailové služby, která mi bude pravidelně zasílat hlášení o výsledku monitoringu a pak také odešle informaci v případě závažné chyby.

Kapitola 6

ZÁVĚR

V této bakalářské práci jsem se zabýval analýzou, návrhem a implementací nástroje pro monitorování ceny nemovitostí v reálném čase, na jehož základě je modelován realitní index.

Jsem přesvědčen, že se mi povedlo naplnit všechny cíle definované v úvodu práce. Aplikace je hotová, funkční a již více než měsíc sbírá data v plném rozsahu.

V rešeršní části jsem se zabýval především srovnáním největších českých realitních serverů, ze kterého vzešel natolik jednoznačný vítěz z hlediska potřeb této aplikace, že za něj případně nebude jednoduché najít náhradu.

V rámci návrhu jsem věnoval velkou pozornost výběru technologií a architektuře, aby aplikace obstála z hlediska nefunkčních požadavků na škálovatelnost, výkon a bezpečnost.

Samotná implementace pro mě byla časově náročnější především z toho důvodu, že jsem s některými technologiemi pracoval vůbec poprvé – bylo to nicméně vědomé rozhodnutí a jsem rád, že jsem jej učinil, neboť jsem se přiučil tomu, na co bych si pravděpodobně jinak nikdy nenašel čas.

Co se týče dopadu této práce, je zřejmé, že relevance indexu bude logicky růst ruku v ruce s dobou jeho existence. Plný potenciál se začne projevovat nejspíš až za několik let. Nicméně jsem přesvědčen, že již v horizontu půl roku či roku bude možné vypozorovat určité trendy, které třeba někomu přispějí pozitivním způsobem k rozhodnutí, zda koupit či nekoupit nemovitost. Pokud se tak stane, budu osobně považovat smysl této práce za více než naplněný. To ale ukáže až budoucnost, neboť jak říká přísloví předepisované kdekomu:

„*Cesta dlouhá tisíc mil začíná prvním krokem.*“

Bibliografie

1. ČESKÝ STATISTICKÝ ÚŘAD. *Ceny sledovaných druhů nemovitostí – 2017-2019*. 2020. Dostupné také z: https://www.czso.cz/csu/czso/ceny-sledovanych-druhu-nemovitosti-2017-az-2019. [Přístup 2021-04-20].
2. DELOITTE. *Deloitte Real Index Q4 2020*. 2021. Dostupné také z: https://www2.deloitte.com/content/dam/Deloitte/cz/Documents/real-estate/EN_Real_index_4Q_2020.pdf. [Přístup 2021-04-20].
3. HYPOTEČNÍ BANKA. *HB Index*. 2021. Dostupné také z: https://www.hypotecnibanka.cz/o-bance/pro-media/hb-index/hb-index-zdrazovani-rezidencnich-nemovit1/. [Přístup 2021-05-07].
4. MARKÉTA BIDRMANOVÁ, SEZNAM ZPRÁVY. *Končí éra laciných hypoték? Rozhovor s guvernérem České národní banky Jiřím Rusnokem.* 2021. Dostupné také z: https://www.seznamzpravy.cz/clanek/rusnok-ceny-hypotek-jsou-na-dne-odted-cekejte-mirny-rust-143901. [Přístup 2021-04-22].
5. PAVLA ADAMCOVÁ A JIŘÍ KROPÁČEK, AKTUÁLNĚ.CZ. *Kam zamíří ceny nemovitostí. Mapa bydlení s argumenty expertů pro a proti zdražování*. 2020. Dostupné také z: https://zpravy.aktualne.cz/ekonomika/skutecne-ceny-prodanych-bytu-v-cr/r~e267fb689aa711eaaabd0cc47ab5f122/. [Přístup 2021-04-22].
6. JAMES CHEN, INVESTOPEDIA. *Guide to Index Fund Investing: What Is an Index?* 2021. Dostupné také z: https://www.investopedia.com/terms/i/index.asp. [Přístup 2021-04-25].
7. ETF DATABASE. *List of Real Estate Indexes*. 2021. Dostupné také z: https://etfdb.com/indexes/real-estate/. [Přístup 2021-04-25].

8. S&P DOW JONES INDICES LLC. *Dow Jones U.S. Real Estate Index: Base Dates and History Availability*. 2021. Dostupné také z: https://www.spglobal.com/spdji/en/documents/methodologies/methodology-dj-global-indices.pdf. [Přístup 2021-04-27].
9. HOMETRACK DATA SYSTEMS LTD. *Methodology: How long are the Hometrack House Price Indices series?* 2021. Dostupné také z: https://www.hometrack.com/uk/insight/uk-house-price-index/. [Přístup 2021-04-27].
10. ČESKÝ STATISTICKÝ ÚŘAD. *Ukončení publikace Ceny sledovaných druhů nemovitostí*. 2021. Dostupné také z: https://www.czso.cz/csu/czso/ukonceni-publikace-ceny-sledovanych-druhu-nemovitosti. [Přístup 2021-04-30].
11. SPOLEČNOST PRO CENOVÉ MAPY ČR S.R.O. *Cenová mapa prodejních cen*. 2021. Dostupné také z: https://www.cenovamapa.org/CZE/About/2/. [Přístup 2021-05-05].
12. STAVEBNÍ FÓRUM. *Česko má 35 realitních portálů*. 2019. Dostupné také z: https://www.stavebni-forum.cz/cs/newsroom/cesko-ma-35-realitnich-portalu/. [Přístup 2021-06-07].
13. NETMONITOR. *Online data pro Sreality server*. 2021. Dostupné také z: https://www.netmonitor.cz/online-data-ola. [Přístup 2021-04-04].
14. NETMONITOR. *Online data pro iDNES Reality server*. 2021. Dostupné také z: https://www.netmonitor.cz/online-data-ola. [Přístup 2021-04-04].
15. SIMILARWEB. *SimilarWeb: RealityMIX Traffic Overview*. 2021. Dostupné také z: https://www.similarweb.com/website/realitymix.cz/ [Přístup 2021-04-24].
16. NETMONITOR. *Online data pro Bezrealitky server*. 2021. Dostupné také z: https://www.netmonitor.cz/online-data-ola. [Přístup 2021-04-04].
17. LUKÁŠ HRDINA. *Jaké realitní servery se vyplatí využít při prodeji a kolik stojí inzerce?* 2018. Dostupné také z: https://realitnikonzultace.cz/realitni-servery/. [Přístup 2021-06-07].
18. ALGOMICA. *Algomica: Vývoj a testování softwaru*. 2007-2021. Dostupné také z: http://www.algomica.cz/specifikace_softwaru.htm. [Přístup 2021-05-08].
19. RUTH MALAN, DANA BREDEMEYER. *White paper: Defining Non-Functional Requirements*. 2001. Dostupné také z: http://www.bredemeyer.com/pdf_files/NonFunctReq.PDF. [Přístup 2021-05-08. Z originálu přeloženo autorem této práce.]

20. SEZNAM.CZ, A.S. *Sreality - dokumentace k importnímu rozhraní*. 2015. Dostupné také z: `https://admin.sreality.cz/doc/import.pdf`. [Přístup 2021-05-15].

21. WIKISCRIPTA. *Míry polohy*. 2021. Dostupné také z: `https://www.wikiskripta.eu/w/M%C3%ADry_polohy`. [Přístup 2021-05-09].

22. ING. ARCH. ROMAN KUČÍREK, ING. ARCH. JAN KOMÁREK. *Kolik peněz stojí realizace rodinného domu?* 2020. Dostupné také z: `https://www.estav.cz/cz/8414.kolik-penez-stoji-realizace-rodinneho-domu-je-draha`. [Přístup 2021-05-09].

23. BABKOVÁ, Denisa. *Kolik stojí pozemky pro stavbu domů?* 2018. Dostupné také z: `https://www.stavebni-vzdelani.cz/cena-pozemku/`. [Přístup 2021-05-09].

24. IBM CLOUD EDUCATION. *Three-Tier Architecture*. 2020. Dostupné také z: `https://www.ibm.com/uk-en/cloud/learn/three-tier-architecture`. [Přístup 2021-05-21].

25. JETBRAINS S.R.O. *The State of Developer Ecosystem 2020*. 2020. Dostupné také z: `https://www.jetbrains.com/lp/devecosystem-2020/`. [Přístup 2021-05-16].

26. TIOBE SOFTWARE BV. *TIOBE Index*. 2021. Dostupné také z: `https://www.tiobe.com/tiobe-index/`. [Přístup 2021-05-16].

27. REDHAT, INC. *Quarkus Portal*. 2021. Dostupné také z: `https://quarkus.io/`. [Přístup 2021-06-27].

28. GREENE, Jason. *Introducing Quarkus: a next-generation Kubernetes native Java framework*. 2019. Dostupné také z: `https://developers.redhat.com/blog/2019/03/07/quarkus-next-generation-kubernetes-native-java-framework`. [Přístup 2021-05-18].

29. ZHANGCHENG, Ryan. *Build Time Boot: Refine Java Framework*. 2020. Dostupné také z: `https://dzone.com/articles/build-time-boot-refine-java-framework`. [Přístup 2021-05-18].

30. REDHAT, INC. *Quarkus Production Documentation*. 2021. Dostupné také z: `https://access.redhat.com/documentation/en-us/red_hat_build_of_quarkus/1.7/html/compiling_your_quarkus_applications_to_native_executables/proc-producing-native-executable_quarkus-building-native-executable`. [Přístup 2021-05-18].

31. BERNARD, Emmanuel. *Quarkus: Why, how and what*. Devoxx Belgium 2019, 2019. Dostupné také z: `https://www.youtube.com/watch?v=SQDR34KoC-8`. [Přístup 2020-10-15].

32. STĚHULE, Pavel. *PostgreSQL*. 2021. Dostupné také z: https://postgres.cz/wiki/PostgreSQL. [Přístup 2021-05-19].

33. TRAVERS, Chris. *Odpověď na otázku „What is the largest production deployment of PostgreSQL for online use?“ na serveru Quora*. 2018. Dostupné také z: https://www.quora.com/What-is-the-largest-production-deployment-of-PostgreSQL-for-online-use/answer/Chris-Travers-4.

34. BLOCH, Joshua. In: *Effective Java, Third Edition*. Addison-Wesley Professional, 2017, s. 73. ISBN: 9780134686097. Z originálu přeloženo autorem této práce.

35. SAGAR, Paresh. *What Is a Single Page Application? Meaning, Pitfalls & Benefits*. 2020. Dostupné také z: https://www.excellentwebworld.com/what-is-a-single-page-application/. [Přístup 2021-05-25].

36. BROCK, John. *Building Browser-Based UIs with Oracle JET*. 2017. Dostupné také z: https://www.oracle.com/corporate/features/oracle-jet.html.

37. INITIATIVE, Open Source. *The Universal Permissive License (UPL), Version 1.0*. 2021. Dostupné také z: https://opensource.org/licenses/UPL. [Přístup 2021-05-20].

38. JANICE J. HEISS, JOSHUA BLOCH. *More Effective Java With Google's Joshua Bloch*. 2008. Dostupné také z: https://www.oracle.com/technical-resources/articles/javase/bloch-effective-08-qa.html. Rozhovor. [Přístup 2021-05-30].

39. ORACLE. *Oracle JET Supported Platforms*. 2021. Dostupné také z: https://www.oracle.com/webfolder/technetwork/jet-400/globalSupport-FAQ.html. [Přístup 2021-06-17].

Příloha A

Seznam použitých zkratek

API Application Programming Interface

CPU Central Processing Unit

CSS Cascading Style Sheets

ČSÚ Český statistický úřad

GPS Global Positioning System

GUI Graphical User Interface

HTML HyperText Markup Language

HTTP HyperText Transfer Protocol

HTTPS HyperText Transfer Protocol Secure

JSON JavaScript Object Notation

JVM Java Virtual Machine

ORM Object–Relational Mapping

RD Rodinný dům

REST Representational State Transfer

SDK Software Development Kit

URI Uniform Resource Identifier

URL Uniform Resource Locator

VM Virtual Machine

VPN Virtual Private Network

XML Extensible Markup Language

Příloha B

Ukázky kódu

```
{
    banner_seller_order_id: 40631
    category_instrumental: "výběrem či prodejem domu"
    filter: {category_main_cb: "2", locality_district_id: "72",
    ↪   suggested_regionId: -1, suggested_districtId: -1,…}
    filterLabels: []
    filterLabels2: {something_more3_3310: "elevator",
    ↪   furnished_1: "furnished", furnished_2:
    ↪   "partly_furnished",…}
    locality: "okres Brno-město"
    locality_dativ: "v Brně-městě"
    logged_in: true
    meta_description: "105 realit v nabídce prodej domů
    ↪   Brno-město. Vyberte si novou nemovitost na sreality.cz s
    ↪   hledáním na mapě a velkými náhledy fotografií nabízených
    ↪   domů."
    page: 1
    per_page: 20
    result_size: 105
    title: "Domy na prodej Brno-město"
    _embedded: {,…}
    _links: {self: {,…}, clusters_with_bounding_box_of_first_10:
    ↪   {,…}, rss: {,…}}
}
```


Výpis kódu 8: Odpověď Sreality serveru pro zdroj /estates (makro pohled)

```
{
    (...)
    _embedded: {,…}
    (...)
    estates: [{labelsReleased: [[], []], has_panorama: 0,
    ↪  labels: [], is_auction: false,…},…]
       0: {labelsReleased: [[], []], has_panorama: 0, labels:
       ↪  [], is_auction: false,…}
       1: {labelsReleased: [["basin"], []], has_panorama: 0,
       ↪  labels: ["Bazén"], is_auction: false,…}
       2: {labelsReleased: [["new_building"], []], has_panorama:
       ↪  0, labels: ["Novostavba"], is_auction: false,…}
       3: {labelsReleased: [[], ["tram"]], has_panorama: 0,
       ↪  labels: ["Tramvaj 1 min. pěšky"], is_auction:
       ↪  false,…}
       4: {labelsReleased: [["basin"], ["post_office", "atm"]],
       ↪  has_panorama: 0,…}
       5: {labelsReleased: [["basin"], ["tram", "post_office"]],
       ↪  has_panorama: 0,…}
       (...)
    (...)
}
```

Výpis kódu 9: Odpověď Sreality serveru pro požadavek na zdroj /estates (detail)

Příloha C

Obsah přiloženého nosiče

- `readme.txt` ..... stručný popis obsahu datového nosiče
- `src`
  - `backend` ..... zdrojové kódy implementace backendové aplikace
  - `frontend` ..... zdrojové kódy implementace frontendové aplikace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `doc` ..... dokumentace
  - `descriptionOfGui.md` ..... popis frontendového rozhraní
  - `technicalDocumentation.md` ..... technická dokumentace
  - `javadoc`
    - `index.html` ..... dokumentace zdrojového kódu backendu
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF