FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Mapa verejných zákaziek
Student:	Marek Drozdík
Vedoucí:	Mgr. Martin Mareš
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2023/2024

Pokyny pro vypracování

V posledných rokoch sú štátne verejné zákazky zverejňované pomocou webovej služby "Národní elektronický nástroj". Táto forma umožňuje automatické spracovanie týchto dát a ich následnú analýzu.

Cieľom tejto práce je vytvoriť webovú aplikáciu, ktorá umožní vizualizovať zverejnené dáta pomocou mapy. Na mape budú vhodným spôsobom zobrazení výhercovia a účastníci vo verejných zákazkách s voliteľným filtrovaním cez zadávateľa, účastníka a miesta. Aplikácia bude dáta získavať periodicky na pozadí, pre komunikáciu medzi frontendovou a backendovou časťou sa bude používať REST API. Pre ukladanie dát v databáze bude aplikácia používať vhodnú reprezentáciu dát vzhľadom k zobrazovaniu dát na mape. Aplikácia bude využívať kontajnerizáciu a spĺňať odporúčané postupy softwarového inžinierstva z pohľadu dokumentácie a testovania. Výsledná práca bude funkčným prototypom webovej aplikácie podľa uvedeného zadania.

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

Bakalárska práca

MAPA VEREJNÝCH ZÁKAZIEK

Marek Drozdík

Fakulta informačních technologií
Katedra softwarového inžinierstva
Vedúci: Mgr. Martin Mareš
11. januára 2024

Obsah

Zoznam obrázkov

Zoznam tabuliek

Zoznam výpisov kódu

Chcel by som poďakovať predovšetkým vedúcemu práce Mgr. Martinovi Marešovi a Mgr. Adamovi Szabó za ich cenné rady, venovaný čas a odborné vedenie. Tiež ďakujem svojej rodine a priateľom za ich neustálu podporu a povzbudenie počas celého štúdia.

Vyhlásenie

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

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

V Praha dňa 11. januára 2024

Abstrakt

Verejné zákazky sú jedným z kľúčových nástrojov pre transparentné a efektívne hospodárenie s verejnými financiami. Práca si kladie za cieľ vytvoriť webovú aplikáciu, ktorá získava dáta o verejných zákazkách v Českej republike a následne ich vizualizuje pomocou sady interaktívnych máp. Výsledná aplikácia periodicky získava dáta z portálu Národní elektronický nástroj pomocou webového scrappingu. Aplikácia využíva proces geokódovania na obohatenie získaných dát o geografickú polohu jednotlivých subjektov. Získané a obohatené dáta sú vizualizované na mapách a aplikácia podporuje filtráciu zobrazených dát podľa miesta plnenia, zadávateľa a účastníka. Zobrazovanie dát o verejných zákazkách na mape slúži na identifikáciu českých, ale aj zahraničných miest, do ktorých idú verejné financie Českej republiky, čo môže byť nápomocné pre získanie väčšieho kontextu o súťažiacich stranách.

Klíčová slova verejné zákazky, verejné financie, webová aplikácia, webový scrapping, vizualizácia dát, mapa, Spring Boot, React, Deck.gl

Abstract

Public procurements are one of the key tools for transparent and efficient management of public finances. The aim of this thesis is to create a web application that obtains data on public procurements in the Czech Republic and subsequently visualizes them using a set of interactive maps. The resulting application periodically retrieves data from the National electronic tool using web scraping. The application uses the process of geocoding to enrich the acquired data with the geographical location of each legal entity. The obtained and enriched data are visualized on maps, and the application supports filtering the displayed data by place of performance, contracting authority, and participant. Displaying public procurements data on a map helps to identify Czech as well as foreign locations where public funds from the Czech Republic go, which can help gain a broader context about the competing parties.

Keywords public procurement, public funds, web application, web scrapping, data visualization, map, Spring Boot, React, Deck.gl

Zoznam skratiek

API	Application Programming Interface
CRUD	Create, Read, Update, Delete
DOM	Document Object Model
DTO	Data Transfer Object
FCP	First Contentful Paint
HATEOAS	Hypermedia As The Engine Of Application State
HTTP	Hypertext Transfer Protocol
JPA	Java Persistence API
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
LCP	Largest Contentful Paint
MUI	Material-UI
ORM	Object-Relational Mapping
REST	Representational State Transfer
SOA	Service-Oriented Architecture
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
TTFB	Time To First Byte
UI	User Interface
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
UX	User Experience
WWW	World Wide Web
XML	eXtensible Markup Language

Úvod

Česká republika a ďalšie krajiny strednej a východnej Európy ešte stále nie sú na rovnakej úrovni ako západné či škandinávske krajiny v oblasti hospodárenia s verejnými financiami. Toto potvrdzujú aj rôzne medzinárodné indexy a hodnotenia, ktoré ukazujú existujúce rozdiely v efektívnosti správy verejných financií medzi týmito regiónmi. [1] Verejné zákazky sú jedným z kľúčových nástrojov pre transparentné a efektívne hospodárenie s verejnými financiami. Verejne dostupné a otvorené dáta o verejných zákazkách sú základným predpokladom na vykonávanie kontroly nad hospodárením danej organizácie či štátu. Takáto kontrola môže pozostávať z rôznych analýz alebo vizualizácií, ktoré predstavujú tieto dáta v nových súvislostiach či pohľadoch. Až táto činnosť môže prinášať výsledky v podobe odhaľovania neefektívneho či neúčelného narábania s verejnými financiami.

Mapa ako spôsob vizualizácie dát ponúka nový pohľad na doménu verejných zákaziek v Českej republike, z ktorého môžu čerpať novinári, ale aj neodborná verejnosť. Výsledná aplikácia bude umožňovať prehľadávanie jednotlivých dodávateľov a účastníkov verejných zákaziek na mape. Takéto zobrazenie spoločností môže byť nápomocné pri identifikácií schránkových firiem, ktoré sídlia na virtuálnych adresách v daňových rajoch. Mapa bude umožňovať aj identifikáciu lokalít, do ktorých idú najväčšie investície z verejných zákaziek. Získané dáta touto prácou bude využívať aj nadväzujúca práca zaoberajúca sa dátovou analýzou zameranou na odhaľovanie neúčelného hospodárenia s verejnými financiami či korupcie.

Táto práca si preto kladie za cieľ vytvoriť webovú aplikáciu, ktorá bude získavať dáta o verejných zákazkách v Českej republike a následne ich vizualizovať pomocou mapy. Prvým cieľom je predstavenie domény verejných zákaziek a urobenie rešerše už existujúcich riešení vizualizácie týchto dát. Nasledovným cieľom je inžinierstvo požiadaviek a vytvorenie špecifikácie funkčných a nefunkčných požiadaviek. Súčasťou práce bude aj vytvorenie detailného návrhu užívateľského rozhrania. Nasledovným cieľom je návrh softvéru, v rámci ktorého sa vyberie vhodná softvérová architektúra a vytvorí návrh databázovej reprezentácie dát. Prvým cieľom implementačnej časti práce je získanie a uloženie dát o ukončených verejných zákazkách v Českej republike. Získavanie dát sa bude periodicky spúšťať na doplnenie uložených dát o nové verejné zákazky. Ďalším cieľom je implementácia samotnej webovej aplikácie, ktorá bude získané dáta vizualizovať na mape s možnosťou filtrácie podľa zadávateľa, dodávateľa a miesta realizácie. Posledným cieľom je vytvorenie dokumentácie a otestovanie výslednej aplikácie.

Kapitola 1
Analýza

Táto kapitola popisuje prvú fázu životného cyklu softvérového vývoja, ktorou je analýza. Úvod tejto kapitoly sa venuje základným pojmom v doméne verejných zákazkách a ich definíciám zo zákona. Následne kapitola uvádza už existujúce riešenia zverejňovania dát o verejných zákazkách v Českej republike. Kapitola pokračuje procesom inžinierstva požiadaviek a zadefinuje rozdelenie požiadaviek na funkčné a nefunkčné. V závere zaradí do týchto kategórii požiadavky plynúce zo zadania tejto práce a z konzultácií s vedúcim tejto práce, ktorý zároveň predstavoval aj rolu zákazníka.

1.1 Základné pojmy

Na zadefinovanie základných pojmov z oblasti verejných zákaziek v Českej republike slúži zákon č. 134/2016 o zadávaní verejných zákaziek [2], ktorý uvádza nasledovné:

Zadávateľ

Zadávateľom môže byť:

- Česká republika,
- Česká národní banka,
- samospráva,
- iná právnická osoba spĺňajúca určité vlastnosti.

Dodávateľ

Dodávateľom sa rozumie osoba, ktorá ponúka poskytnutie dodávky, služieb alebo stavebných prác, prípadne viacero takýchto osôb spoločne. Za dodávateľa sa tiež považuje pobočka závodu.

Verejné zákazky

Zo zákona o zadávaní verejných zákaziek plynie: „*Zadáním veřejné zakázky se pro účely tohoto zákona rozumí uzavření úplatné smlouvy mezi zadavatelem a dodavatelem, z níž vyplývá povinnost dodavatele poskytnout dodávky, služby nebo stavební práce.*“ A teda pod verejnou zákazkou sa rozumie poskytnutie dodávky, služby či stavebnej práce dodávateľom na základe uzatvorenej zmluvy so zadávateľom.

1.2 Existujúce riešenia

Žiadny z verejne dostupných portálov nevyužíva mapu ako formu vizualizácie dát o verejných zákazkách. Existuje však množstvo portálov, ktoré sa venujú verejným zákazkám v Českej republike, veľa z nich však len na komerčné účely, a teda len spájajú zadávateľov s dodávateľmi. Pre túto prácu je kľúčová evidencia histórie už ukončených zákaziek s informáciou o ich výsledku, ktorú mnohé z komerčných portálov nezverejňujú. Medzi existujúce riešenia, ktoré evidujú informácie o ukončených verejných zákazkách patria:

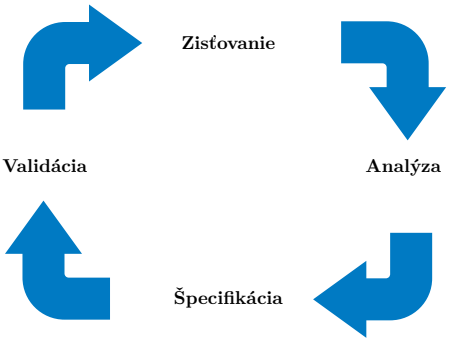
nen.nipez.cz Celým názvom Národní elektronický nástroj (NEN) je webovým portálom, ktorý vytvorilo Ministerstvo pro místní rozvoj. Portál eviduje výsledky verejných zákaziek a aj historické informácie o všetkých ukončených zákazkách. Výsledok ukončených verejných zákaziek obsahuje detailné informácie o dodávateľovi a o cene z uzatvorenej zmluvy medzi zadávateľom a dodávateľom. Vo výsledku sú evidované aj jednotlivé ponuky od uchádzačov vrátane nimi dopytovanej ceny za danú zákazku. Portál eviduje aj presnú adresu sídla zadávateľov, dodávateľov a aj uchádzačov o verejné zákazky. Ďalej portál obsahuje rôzne informácie slúžiace na kategorizáciu verejných zákaziek, akým je napríklad stav v akom sa verejná zákazka aktuálne nachádza alebo druh, ktorý zaraďuje zákazku do dodávky, služby či stavebnej práce. Uvádzajú aj hlavné miesto výkonu práce, aj keď žiaľ len na úrovni krajov a množstvo ďalších informácií. Portál zverejňuje aj grafy zobrazujúce základné štatistické informácie o verejných zákazkách v Českej republike, ale len v kontexte posledného kalendárneho roku. Značná výhoda tohto portálu v porovnaní s alternatívami vyplýva z uznesenia vlády Českej republiky č. 408/2018, ktoré udáva povinnosť využívať práve tento portál organizáciami centrálnej štátnej správy pri verejných zákazkách s predpokladanou hodnotou presahujúcou 500 000 CZK. [3]

tenderarena.cz Portál na svojej stránke uvádza, že je koncipovaný ako plnohodnotná náhrada NEN. [4] Dáta, ktoré tento portál poskytuje, sú tak v mnohom podobné tým z portálu NEN. Portál eviduje historické dáta o ukončených verejných zákazkách a ponúka aj informácie o výsledkoch, ktoré obsahujú dáta o dodávateľoch a aj o účastníkoch obstarávania. Na rozdiel od NEN portál *tenderarena.cz* neeviduje adresy sídiel jednotlivých subjektov a neumožňuje prehliadanie zoznamu všetkých verejných zákaziek súčasne. Portál podporuje len zobrazovanie zoznamu zákaziek za posledných 24 hodín alebo prehliadanie zoznamu zákaziek konkrétneho zadávateľa.

rozza.cz Webový portál vytvorený Ministerstvom pro místní rozvoj, ktorý integruje zákazky zo systému NEN a z *tenderarena.cz* a mnohých ďalších portálov, čím zjednodušuje prístup dodávateľov k verejným zákazkám. [5] Portál eviduje aktuálny stav verejnej zákazky ale na zobrazenie výsledku sú užívatelia nútení prejsť na príslušný portál, na ktorom bola daná verejná zákazka vytvorená. Na rozdiel od ostatných zmienených portálov, portál *rozza.cz* umožňuje stiahnutie otvorených dát o verejných zákazkách pre daný kalendárny rok. Tieto dáta sú vo formáte XML, ale žiaľ informácie o výsledkoch obstarávania sú dostupné opäť len formou odkazu na príslušný zadávací portál danej zákazky.

1.3 Inžinierstvo požiadaviek

Požiadavky na systém popisujú čo má systém robiť, aké služby má poskytovať a aké sú obmedzenia jeho činnosti. Tieto služby odrážajú potreby zákazníka na systém. Proces zisťovania, analýzy, špecifikácie a validácie týchto služieb a obmedzení je označovaný ako inžinierstvo požiadaviek. Ako znázorňuje obrázok 1.1 inžinierstvo požiadaviek je v praxi často iteratívny proces, kde sa navyše jednotlivé procesy často prelínajú. Požiadavky, ktoré vzniknú týmto procesom, by mali byť jednoznačné, jednoducho zrozumiteľné, úplné a konzistentné. Často sú požiadavky delené na funkčné a nefunkčné.[6]



Obr. 1.1 Proces inžinierstva požiadaviek.

Získavanie a presné špecifikovanie požiadaviek je jedeným z kľúčových častí vývoja softvéru, keďže chybná či nedostatočná špecifikácia požiadaviek môže viesť k výraznému predĺženiu, a teda aj predraženiu dodávaného softvéru. Pri istých modeloch životného cyklu vývoja softvéru, akým je napríklad vodopádový model [7] sa detailná a úplná špecifikácia musí vytvárať hneď v úvode vývojového cyklu. Pri tomto modely musí byť každá fáza dokončená skôr, ako sa začne ďalšia fáza. V neskorších fázach sa už ťažko aplikujú prípadné zmeny požiadaviek na dodávaný systém.

Alternatívny prístup prináša agilný model [8], pri ktorom sa vývoj musí prispôsobiť dynamickým a často meniacim sa požiadavkám. Inžinierstvo požiadaviek prebieha počas celého vývojového cyklu a špecifikácia požiadaviek sa musí neustále aktualizovať. Slabé inžinierstvo požiadaviek je jedným z hlavných príčin zlyhania agilných projektov. [9] Vývoj tejto práce najviac pripomínal agilný model a jednotlivé požiadavky sme v spolupráci so zákazníkom (vedúcim práce) vytvárali počas celého vývojového cyklu, ktorý bol rozdelený do menších iterácií.

Pod aktérmi rozumieme ľudí, iné systémy alebo externé zariadenia, ktoré interagujú s vyvíjaným systémom. [10] V tejto práci som identifikoval nasledovných aktérov:

Užívateľ webovej stránky
Ide o bežného užívateľa vytvorenej webovej stránky, ktorá zobrazuje dáta o verejných zákazkách pomocou mapy. Môžu ním byť novinári ale aj neodborná verejnosť, ktorá sa zaujíma o hospodárenie s verejnými financiami.

Dátový analytik alebo systém na dátovú analýzu
Tento aktér môže pracovať s dátami, ktoré sa v rámci tejto práce získali a využívať ich na ďalšiu analýzu.

1.3.1 Funkčné požiadavky

Funkčné požiadavky [6] sú popisy služieb, ktoré by mal systém poskytovať, reakcie systému na určité vstupy a chovanie systému v konkrétnych situáciach. Často je žiadúce aj explicitne definovať, čo by systém robiť nemal. Zo zadania tejto práce alebo z komunikácie so zákazníkom práce vyplývajú nasledovné funkčné požiadavky:

F1 Získavanie dát
Aplikácia bude na pozadí získavať dáta o ukončených verejných zákazkách v Českej republike. Okrem základných informácií, akými je názov a typ zákazky, budú dáta obsahovať aj informácie o výsledku danej zákazky. Výsledok bude obsahovať zmluvnú cenu, informácie o ponukách a informácie o dodávateľovi a účastníkoch obstarávania. Dáta budú tiež obsahovať presné adresy zadávateľov, dodávateľov a účastníkov.

F2 Periodické získavanie dát
Aplikácia sa bude periodicky spúšťať každých 30 dní na získanie nových dát o ukončených

verejných zákaziek.

F3 Prezeranie dodávateľov
Užívateľ si bude vedieť prezerať dodávateľov verejných zákaziek na mape.

F4 Prezeranie účastníkov
Užívateľ si bude vedieť prezerať účastníkov verejných zákaziek na mape.

F5 Filtrovanie podľa zadávateľa
Užívateľ bude môcť voliteľne filtrovať zobrazené dáta podľa zadávateľa.

F6 Filtrovanie podľa dodávateľa
Užívateľ bude môcť voliteľne filtrovať zobrazené dáta podľa dodávateľa.

F7 Filtrovanie podľa miesta
Užívateľ bude môcť voliteľne filtrovať zobrazené dáta podľa miesta.

F8 Negatívne vymedzenie na prácu s dátami
Aplikácia slúži len na vizualizáciu dát a bude podporovať len čítanie dát. Užívatelia tak nebudú môcť vytvárať, meniť alebo vymazávať dáta.

F9 Negatívne vymedzenie na užívateľské účty
Aplikácia nebude podporovať rôzne užívateľské účty alebo prihlasovanie.

1.3.2 Nefunkčné požiadavky

Nefunkčné požiadavky [6] sú kritéria alebo obmedzenia na vyvíjaný systém a týkajú sa systému ako celku a nie jednotlivých funkcií či služieb. Zo zadania tejto práce alebo z komunikácie so zákazníkom vyplývajú nasledovné nefunkčné požiadavky:

N1 Dokumentácia
Bude vytvorená dokumentácia v kóde, z ktorej bude možné vygenerovanie detailnej dokumentácie systému.

N2 Testovanie
Systém bude spĺňať štandardy softvérového inžinierstva z pohľadu testovania. Pre jednotlivé komponenty bude zvolený adekvátny typ testovania vzhľadom na rozsah a funkcionalitu danej komponenty.

N3 Kompatibilita s webovými prehliadačmi
Aplikácia bude kompatibilná s webovým prehliadačom Google Chrome vo verzii od 115.

N4 Responzívny webový dizajn
Webová stránka bude responzívna. Bude použiteľná na rôznych najpoužívanejších rozmeroch obrazoviek vrátane mobilných telefónov.

N5 Rýchlosť webovej stránky
Webová stránka začne vykresľovanie obsahu (metrika FCP[1]) do 2 sekúnd od otvorenia stránky. Vykreslenie najväčšej obsahovej časti obrazovky (metrika LCP[2]) bude do 3 sekúnd.

N6 Bezpečnostné štandardy
Aplikácia bude spĺňať štandardy softvérového inžinierstva z pohľadu bezpečnosti. Tajomstvá nebudú súčasťou verzovacieho systému.

N7 Kontajnerizácia
Aplikácia bude využívať kontajnerizáciu na uľahčenie nasadzovania. Na kontajnerizáciu bude využitý nástroj `docker`.

[1] Viac o metrike FCP na `https://web.dev/articles/fcp`.
[2] Viac o metrike LCP na `https://web.dev/articles/lcp`.

Kapitola 2

Architektúra a technológie

Táto kapitola sa venuje architektúre softvérových aplikácií. Rozoberá rôzne prístupy k architektúre softvéru. Podrobnejšie popisuje a porovnáva monolitnú architektúru a architektúru mikroslužieb. Následne predstavuje a zdôvodňuje výber architektúry pre túto prácu. Záver tejto kapitoly uvádza rozdelenie aplikácie na jednotlivé komponenty a popisuje funkcionalitu každej z nich. Pri každej komponente uvedie a zdôvodní aj výber technológií pre danú komponentu.

Architektúra softvérového systému spočíva v rozdelení systému na komponenty, usporiadania týchto komponentov a určenia spôsobov, akými budú medzi sebou komunikovať. Dôvodom pre toto rozdelenie je uľahčenie vývoja, nasadenia a údržby systému. Existuje množstvo systémov, ktoré fungujú správne a napriek tomu nastávajú problémy pri nasadzovaní, údržbe či prebiehajúcom vývoji. Stratégia návrhu architektúry by mala čo najdlhšie ponechávať čo najviac otvorených možností, aby bol systém vždy schopný prijímať nové požiadavky s čo najmenšou prácnosťou.[11]

Miera kvality návrhu softvéru sa dá určiť na základe vyžadovanej prácnosti na splnenie rôznych požiadaviek od zákazníka. Ak je táto prácnosť nízka a zostane nízka počas celej životnosti systému, tak je návrh s vysokou pravdepodobnosťou dobrý. Ak sa táto prácnosť zvyšuje s každým novým vydaním systému, návrh je zrejme zlý. V prípade zlého návrhu sa softvér môže dostať do bodu, kedy ho nebude možné ďalej vyvíjať (bolo by to príliš drahé) a aplikovať na ňom vyžadované požiadavky. Hlavným cieľom návrhu architektúry je teda minimalizácia celkových nákladov naprieč celým životným cyklom systému a maximalizácia produktivity programátorov. Dôraz na dobrý návrh architektúry dávajú aj Briana Foota a Josepha Yodera v ich citáte: „*If you think good architecture is expensive, try bad architecture.*“ Hovoria, že ak si niekto myslí, že dobrá architektúra je drahá, tak si má skúsiť aká drahá je zlá architektúra. [11]

2.1 Monolitná architektúra

Monolitná architektúra [12], znázornená na obrázku 2.1, je typ softvérovej architektúry, pri ktorej sa musí nasadzovať celá funkcionalita systému ako jeden celok, a teda neumožňuje nasadzovanie jednotlivých komponent samostatne. Monolit býva často interne delený na moduly a vrstvy, ale toto členenie neumožňuje nasadzovanie po častiach. Tak ako každá iná architektúra, aj monolitná architektúra má svoje výhody a nevýhody.

Výhody monolitnej architektúry

Medzi hlavné výhody [13] monolitnej architektúry patria:

Jednoduché nasadzovanie Jediný spustiteľný súbor alebo priečinok uľahčuje nasadzovanie.

Výkon Nakoľko je celá funkcionalita systému v jednom celku, nie je nutné volanie niekoľkých externých API služieb na zaistenie potrebnej funkcionality.

Testovanie Urýchľuje a zjednodušuje najmä proces *End-to-end* testovania (E2E), ktoré overuje funkčnosť celého systému z pohľadu konečného užívateľa. [14]

Nevýhody monolitnej architektúry

Monolitná architektúra býva celkom efektívna pri menších systémoch, nevýhody [13] sa začnú prejavovať najmä pri väčších systémoch, pri ktorých začne byť rozširovanie systému problematické. Medzi hlavné nevýhody monolitnej architektúry patria:

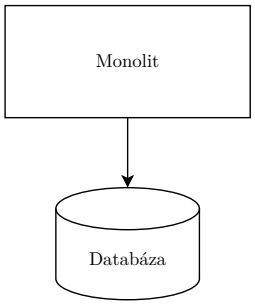
Pomalší vývoj Veľké monolitné systémy robia vývoj komplexnejším a tým aj pomalším.

Obtiažna rozširovateľnosť Monolitná architektúra neumožňuje nezávislé rozširovanie jednotlivých komponent.

Nespoľahlivosť Chyba v nejakom z modulov môže ovplyvňovať, a teda aj zhodiť, celú aplikáciu.

Neflexibilita vo voľbe technológií Monolit je obmedzený technológiami, ktoré sa už používajú v danom systéme. Obtiažna je aj aktualizácia už využívaných technológií.

Časté nasadzovanie celého systému Každá malá zmena v systéme vyžaduje nasadenie celého systému.



Obr. 2.1 Monolitná architektúra [12].

2.2 Architektúra orientovaná na služby

Architektúra orientovaná na služby alebo SOA (Service Oriented Architecture) [15] predstavuje metódu vývoja softvéru, kde sú používané softvérové komponenty nazývané služby na vytváranie komplexných systémov. Každá služba poskytuje konkrétnu biznisovú funkcionalitu a na jej naplnenie môže komunikovať s inými službami, a to aj naprieč rôznymi platformami a programovacími jazykmi. SOA uľahčuje paralelizáciu vývoja a tým urýchľuje dodávky systému a uľahčuje následnú údržbu. Medzi základné zásady SOA patria:

Slabé väzby medzi komponentami
Služby by mali mať čo najmenšie závislosti na externých zdrojoch, ako sú dátové modely alebo informačné systémy. Taktiež by mali byť bezstavové, a teda neuchovávať žiadne informácie z predchádzajúcich relácií.

Abstrakcia
Klienti služby nemusia vedieť logiku a implementačné detaily danej služby. Služby by mali vystupovať ako čierne krabičky a klienti by mali dostať potrebné informácie na využitie danej služby z dokumentácie.

Princíp skladania
Služby by mali mať adekvátnu veľkosť a rozsah, ideálne by mali zaisťovať len jednu konkrétnu biznisovú funkcionalitu. Služby by sa mali vedieť skladať alebo využívať navzájom na zaistenie vykonania komplexných funkcionalít.

2.2.1 Architektúra mikroslužieb

Architektúra mikroslužieb [16] je typ SOA, ktorý definuje rozsah jednotlivých služieb a pre ktorý je kľúčové nezávislé nasadzovanie. Táto architektúra zdieľa so SOA aj jej zásady opísané vyššie, ďalej sú uvedené preto už len tie, ktorými sa architektúra mikroslužieb odlišuje od SOA alebo tie, ktoré sa pri nej musia dodržiavať striktnejšie:

Nezávislé nasadzovanie V mikroslužbách by malo byť možné urobenie a nasadenie zmien bez nutnosti následného nasadenia inej mikroslužby.

Modelovanie podľa biznisovej domény Mikroslužby by mali byť modelované podľa biznisových domén a mali by uprednostňovať kohéziu biznisových funkcionalít pred kohéziou technických funkcionalít. Týmto sa znižuje potreba pre zmeny vyžadujúce zásahy do viacerých mikroslužieb a zároveň zjednodušuje proces integrácie nových mikroslužieb do systému.

Rozsah Rozsah jednotlivých mikroslužieb je čiastočne definovaný modelovaním podľa biznisových domén. Cieľom by však malo byť aj to, aby mikroslužby mali čo najmenšie možné rozhrania.

Vlastnenie vlastného stavu Ak mikroslužba chce získať dáta od inej mikroslužby, mala by si ich od nej vypýtať. Mikroslužby by nemali zdieľať ani databázu, nakoľko sa tým vystavujú riziku nežiadúcich zmien dát inými mikroslužbami a pripravujú sa o možnosť rozhodnutia, aké informácie zo svojho stavu budú skrývať a aké budú zverejňovať.

2.3 Porovnanie architektúr

SOA spolu s architektúrou mikroslužieb predstavujú alternatívu k monolitnej architektúre, a teda riešia aj väčšinu jej nevýhod rozoberaných v 2.1. Architektúra mikroslužieb v porovnaní so SOA však zvyčajne dosahuje týchto výhod do vyššej miery tým, že striktnejšie dodržiava zásady a koncepty SOA. Architektúra mikroslužieb uľahčuje vývoj, nakoľko programátori pracujú na menších ucelených komponentoch. Navyše komponenty majú jasne zadefinované, čo do nich patrí, a čo už nie. Rozdelenie na mikroslužby zjednodušuje aj paralelizáciu práce, čím sa urýchľuje samotný vývoj. Jednotlivé mikroslužby sa môžu škálovať v podstate nezávislo na sebe, čo bolo u monolitnej architektúry obtiažnejšie. V porovnaní s monolitnou architektúrou, je architektúra mikroslužieb robustnejšia, môže pri nej spadnúť nejaká komponenta a ostatné časti systému môžu aj napriek tomu fungovať ďalej. [17]

Architektúra mikroslužieb ale prináša aj značnú komplexitu v návrhu systému, a to najmä vo vhodnom rozdelení na jednotlivé mikroslužby. Dodržanie všetkých vyššie spomenutých zásad architektúry mikroslužieb nie je vždy triviálne a ich porušením sa vývojári pripravujú o niektoré z výhod tohto návrhu a môžu tak skončiť s ešte komplexnejším vývojom či údržbou ako by tomu bolo pri architektúre monolitu. Výber vhodnej architektúry pre konkrétny systém predstavuje pomerne komplexnú úlohu, ktorej záver má následky na celý vývojový cyklus softvéru.

2.4 Komunikácia medzi komponentami

Medzi najčastejšie používané prístupy k implementácii komunikácie medzi jednotlivými komponentami [18] patria:

SOAP
SOAP (Simple Object Access Protocol) jeden z najstarších protokolov slúžiacich na výmenu štruktúrovaných dát medzi systémami a na komunikáciu využíva formát XML nad HTTP. Medzi hlavné výhody tohto protokolu patrí štandardizácia a bezpečnosť.

REST
REST (Representational State Transfer) je architektonický štýl, ktorý definuje sadu architektonických obmedzení. Jeho dodržiavanie by malo viesť k architektúre, ktorá je ľahko použiteľná, škálovateľná a flexibilná. Medzi základné zásady REST-u patria:

Klient-server architektúra REST je založený na separácií klienta od servera. Klient posiela požiadavky na server, ten ich spracováva a naspäť posiela príslušné odpovede.

Práca so zdrojmi Zakladá na práci so zdrojmi. Každý zdroj je jednoznačne určený svojím unikátnym identifikátorom URI (Uniform Resource Identifier).

Jednotné rozhranie Práca so zdrojmi je definovaná štandardizovanou množinou metód a formátom odpovedí.

Bezstavovosť Každý požiadavok od klienta je sebestačný a obsahuje všetky potrebné informácie na jeho spracovanie serverom.

HATEOAS Každá odpoveď poskytnutá serverom by mala obsahovať metadáta s odkazmi na všetky súvisiace informácie o tom, ako daný zdroj používať. Až táto zásada umožňuje skutočné oddelenie klienta od servera, avšak množstvo REST API túto zásadu nedodržiava a to najmä kvôli značnej komplexnosti, ktorú do systému prináša.

REST sa v praxi najčastejšie využíva s formátom JSON a HTTP, ktorý definuje rozhranie na prácu so zdrojmi HTTP metódami. Príklad implementácie REST-u nad HTTP pre všetky CRUD (Create Read Update Delete) operácie je znázornený v tabuľke 2.1.

Tabuľka 2.1 Príklad REST API podporujúceho všetky CRUD operácie.

Operácia	HTTP metóda	Využitie
CREATE	POST/PUT	Vytvorenie nového zdroja na danej URI.
READ	GET	Získanie aktuálneho stavu zdroja z danej URI
UPDATE	PUT/PATCH	Čiastočná aktualizácia zdroja na danej URI
DELETE	DELETE	Vymazanie zdroja na danej URI

GraphQL
GraphQL je dotazovací jazyk pre API, ktorý umožňuje klientom interakciu s jediným *endpointom* na získanie potrebných dát. Takýto prístup eliminuje nutnosť „reťazenia“ požiadaviek a zároveň znižuje veľkosť prenášaných dát, keďže klienti dostávajú len tie dáta, ktoré si vypýtali. Hlavnou výhodou GraphQL API je teda rýchlosť, akou sa klienti dostávajú k dátam.

2.5 Architektúra aplikácie

Táto práca pozostáva z troch hlavných problémov:

- získavanie dát,
- ukladanie dát,
- zobrazovanie dát.

Rozhodol som sa preto rozdeliť aplikáciu na štyri komponenty 2.2, kde každá komponenta predstavuje jednu mikroslužbu:

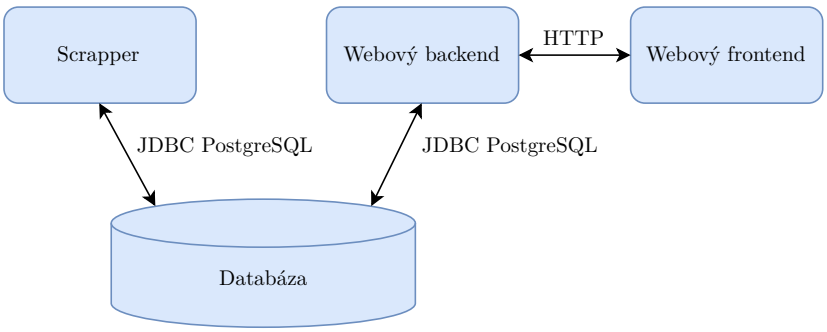
Scrapper Zabezpečuje získavanie dát.

Databáza Slúži ako úložisko dát.

Webový backend Zabezpečuje sprostredkovanie dát webovému frontendu.

Webový frontend Slúži na vizualizáciu dát.

Využitie takéhoto prístupu oddeľuje riešenia jednotlivých problémov a umožňuje ich nezávislé nasadzovanie. Ako je zrejmé z obrázku architektúry 2.2 Backend aj Scrapper zdieľajú jednu databázu, čo by podľa zásady vlastnenia vlastného stavu v architektúre mikroslužieb nemalo nastávať. Táto zásada slúži na zníženie rizika neočakávanej modifikácie údajov inou mikroslužbou, čo v tomto systéme nemôže nastať, nakoľko webový backend implementuje len operácie na čítanie dát.



Obr. 2.2 Architektúra aplikácie.

2.6 Scrapper

Táto komponenta zabezpečuje získavanie, spracovávanie a ukladanie dát do databázy. Pre túto prácu sú kľúčové dáta o výsledkoch ukončených verejných zákaziek, ktoré musia obsahovať:

- cenu zo zmluvy medzi dodávateľom a zadávateľom,
- ceny jednotlivých ponúk od uchádzačov,
- detailné informácie o dodávateľoch a účastníkoch,
- presné adresy sídiel všetkých subjektov.

Všetky tieto informácie ponúka len portál *nen.nipez.cz*, ktorý však nezverejňuje žiadne API na získanie týchto dát a ani neumožňuje stiahnutie otvorených dát, ako to umožňuje portál *rozza.cz*. Komponenta preto využíva webový scrapping na získanie potrebných dát z portálu *nen.nipez.cz*. Pod webovým scrappingom sa rozumie proces získavania dát z WWW serverov s využitím HTTP a následného ukladania týchto dát do databázy. [19]

Portál *nen.nipez.cz* uvádza presnú adresu zadávateľov, dodávateľov a aj uchádzačov o verejné zákazky, neuvádza však ich geografické súradnice, ktoré sú kľúčové pri zobrazovaní dát na mape. Získavanie geografických súradníc z adresy sa nazýva geokódovanie a nasledujúce služby implementujú tento proces:

Google Geocoding API Službu vytvorila spoločnosť Google a umožňuje geokódovanie adries z celého sveta. Na volanie tohto API služba ponúka aj *open source* klientské knižnice pre jazyky Java, Python alebo Go. Komponenta obmedzuje využitie tejto služby len na geokódovanie sídiel zahraničných spoločností z dôvodu spoplatnenia služby podľa počtu požiadaviek na API.

Profinit Geocoding API[1] Táto služba umožňuje geokódovanie len českých adries. Vznikla v rámci interného vývoja firmy Profinit a získal som možnosť ju v rámci tejto práce bezplatne využívať. Služba pri geokódovaní českých adries ponúka podobné výsledky ako Google Geocoding API a zároveň rieši jej limitácie z pohľadu spoplatnenia. Komponenta využíva túto službu na geokódovanie adries sídiel českých spoločností a zadávateľov.

Portál *nen.nipez.cz* uvádza ceny verejných zákaziek v peňažnej mene, v akej bola podpísaná zmluva. Táto nekonzistentnosť peňažných mien komplikuje následnú analýzu či vizualizáciu dát, a preto táto komponenta zamieňa všetky peňažné meny do CZK. Komponenta Scrapper však získava aj verejné zákazky, ktoré sa mohli realizovať pred niekoľkými rokmi, kedy mohli byť kurzy na zámenu mien úplne odlišné tým dnešnými. Z týchto dôvodov som hľadal nástroj, ktorý poskytuje zámenu peňažných mien s historickým pohľadom. Tieto požiadavky spĺňa nástroj *fixer.io*, ktorý je navyše bezplatný pre 1000 požiadaviek mesačne. [20]

2.6.1 Technológie

Táto časť uvádza technológie, ktoré využíva komponenta Scrapper. Zameriava sa na opis a zdôvodnenie výberu technológií, pre ktoré som sa rozhodol už počas analýzy. Ostatné využité technológie budú uvedené v rámci implementácie.

Java

Java je kompilovaný, objektovo orientovaný a silne typovaný programovací jazyk. Jednou z najväčších výhod tohto jazyka je jej nezávislosť od platformy. Túto vlastnosť zabezpečuje kompilačný proces jazyka, ktorý prekladá program do strojovo nezávislého jazyka nazývaného bajtkód. Na rozdiel od jazykov C a C++, ktorými bola Java inšpirovaná, má tento jazyk automatickú správu pamäte pomocou *garbage collectoru* a tým eliminuje nutnosť explicitného uvoľňovania pamäte. [21]

Spring Boot

Spring je *open source* framework slúžiaci na vytváranie produkčných aplikácií nad JVM (Java Virtual Machine).

Spring Boot je balíček nástrojov a otvorených knižníc, ktorý urýchľuje a zjednodušuje vývoj webových aplikácií a mikroslužieb vo frameworku Spring. Toto dosahuje napríklad automatickou konfiguráciou, ktorá zakladá projekt s prednastavenými závislosťami. [22] Vzhľadom k alternatívnym webovým frameworkom pre JVM má Spring Boot najväčšiu užívateľskú základňu [23], čo

má za následok množstvo dostupných materiálov. Navyše považujem aj samotnú dokumentáciu tohto frameworku za veľmi kvalitnú a z týchto dôvodov som sa rozhodol pre jeho využitie v rámci tejto práci.

Táto komponenta bude využívať najmä sadu závislostí `spring-boot-starter-data-jpa`, ktorá slúži na prácu s databázou. Z tejto sady využíva napríklad framework Hibernate, ktorý poskytuje ORM (objektovo-relačné mapovanie) na manipuláciu s dátami z databázy využitím paradigmy objektovo orientovaného programovania. Framework Hibernate implementuje rozhrania JPA (Java Persistence API), čo je špecifikácia, ktorá definuje sadu rozhraní a anotácií pre prácu s relačnými databázami. Toto uľahčuje zmenu konkrétnej implementácie JPA na alternatívy frameworku Hibernate, akými sú napríklad OpenJPA alebo EclipseLink. [24]

JSOUP

JSOUP je knižnica pre jazyk Java, ktorá slúži na dolovanie dát z webu. Ponúka ľahko použiteľné API na získanie HTML kódu webovej stránky pomocou jej URL (Uniform Resource Locator) adresy a uľahčuje aj následnú extrakciu a manipuláciu s údajmi z tejto stránky, načo využíva DOM (Document Object Model) reprezentáciu HTML kódu. [25] Knižnica JSOUP ma v porovnaní s alternatívami zaujala najmä v kontexte jednoduchosti využitia a množstva dostupných selektorov na získavanie dát z HTML.

DOM je API na prácu s HTML alebo XML dokumentami. Definuje logickú štruktúru dokumentu, spôsob akým sa k nemu pristupuje a ako sa s ním manipuluje. Dôležitou vlastnosťou tohto API je garancia, že k všetkému čo je v HTML dokumente sa dá pristúpiť, zmeniť, vymazať alebo pridať pomocou DOM-u. Je založená na princípe objektovej štruktúry, ktorá pripomína štruktúru HTML alebo XML dokumentu, ktorý modeluje. [26]

2.7 Databáza

Úvod tejto časti sa venuje možnostiam ukladania dát a uvádza a zdôvodňuje voľbu databázového systému pre túto prácu. Následne zadefinuje proces konceptuálneho modelovania a predstaví konceptuálnu schému pre doménu verejných zákaziek. Ďalej vysvetlí jednotlivé entity spolu s ich hlavnými položkami a v závere uvedie relačný diagram databázy.

Relačná databáza je spôsob modelovania informácií do tabuliek, riadkov a stĺpcov. Tento typ databázy má schopnosť vytvárať spojenia medzi tabuľkami, čo uľahčuje pochopenie a získanie prehľadu o vzťahoch medzi rôznymi údajmi. Alternatívou k relačným databázam sú nerelačné databázy, častejšie označované ako NoSQL databázy. Rozdiel medzi nimi spočíva v spôsobe, akým sú dáta uložené, organizované a dotazované. Nerelačné databázy môžu ukladať dáta vo forme samostatných dokumentov, úložiska typu kľúč hodnota alebo formou grafových modelov. [27]

Pre účely tejto práce považujem relačný model za lepšiu voľbu, nakoľko umožňuje jednoznačne namodelovať danú doménu a detailne zachytiť jednotlivé vzťahy medzi entitami. Z množstva relačných databázových systémov využívam v aplikácií PostgreSQL, pretože mám s ním najviac skúseností a považujem jeho integráciu s ostatnými využívanými technológiami za veľmi dobrú. Navyše podľa prieskumu portálu *Stack Overflow* je PostgreSQL najpopulárnejším databázovým systémom roku 2023, čo má za následok množstvo dostupných materiálov. [23] Je to *open source* systém, ktorý na dotazovanie dát z databázy využíva jazyk SQL (Structured Query Language) a vyznačuje sa najmä spoľahlivosťou, integritou dát, robustnosťou a rozsiahlym súborom funkcionalít. [28]

2.7.1 Konceptuálna schéma

Konceptuálne modelovanie je proces modelovania reality slúžiaci na jasné zadefinovanie požiadaviek na databázový systém. Z tohto dôvodu by malo byť modelovanie dostatočne jednoduché, aby mu rozumeli aj zákazníci, ktorí nemusia byť vzdelaní v IT sektore. Výsledkom konceptuálneho modelovania je konceptuálna schéma, ktoré môže slúžiť ako dokumentácia požiadaviek a je vstupom do realizácie databázy. Konceptuálna schéma tejto práce je znázornená na obrázku 2.3.

Z konceptuálneho schématu 2.3 je zrejmé, že vzťah medzi verejnou zákazkou (procurement) a spoločnosťou (company) nezodpovedá uvedenej definícií dodávateľa zo základných pojmov 1.1. Tento vzťah je typu M:1, zatiaľ čo definícia hovorí, že dodávateľom môže byť aj viacero spoločností a vzťah by teda mal byť typu M:N. V realite je však verejných zákaziek s viacerými dodávateľmi minimum a preto som zvolil toto zjednodušenie databázového modelu. Zákazky s viacerými dodávateľmi sa ukladajú ako nový záznam v tabuľke `procurement` pre každého dodávateľa. Je to tak jediná situácia, kedy môže nastať, že hodnoty v stĺpci `system_number` nebudú v databáze unikátne, čo umožňuje následnú identifikáciu verejných zákaziek s viacerými dodávateľmi.

2.7.2 Entity

V práci som identifikoval nasledujúce entity, ktoré sa v aplikácií reprezentujú ako tabuľky:

Verejná zákazka (Procurement)
Základná entita, ktorá uchováva všetky informácie o verejných zákazkách. Obsahuje napríklad `system_number`, čo je identifikátor jednotlivých verejných zákaziek využívaný portálom *nen.nipez.cz*. Ďalej sú medzi týmito informáciami dátumy, kedy bola zákazka publikovaná `date_of_publication` a kedy bola uzatvorená zmluva `date_of_contract_close`. Obsahuje aj jednotlivé ceny zo zmluvy, ako je cena bez DPH `contract_price` a cena s DPH `contract_price_vat`. Ukladanie oboch týchto formátov cien môže byť užitočné na odhaľovanie rôznych úľav na daniach, ktoré mohli byť pridelené jednotlivým spoločnostiam. Dôležitým je takisto údaj o mieste plnenia `place_of_performance`, ale žial portál *nen.nipez.cz* poskytuje túto informáciu len na úrovni krajov.

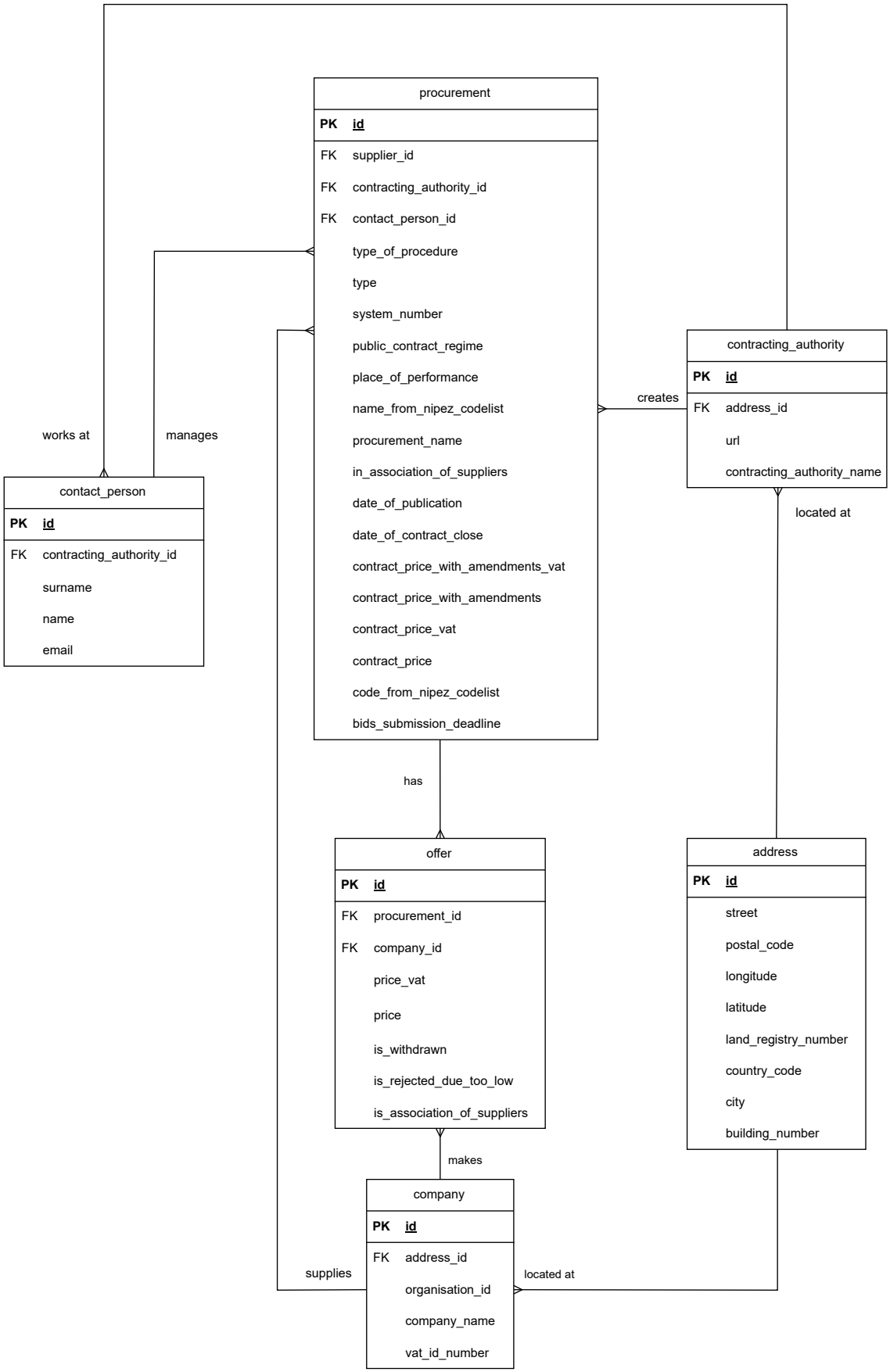
Spoločnosť (Company)
Táto entita slúži na ukladanie údajov o spoločnostiach, ktoré niekedy dodávali alebo sa uchádzali o nejakú verejnú zákazku v Českej republike. Eviduje IČO (Identifikačné číslo osoby) `organisation_id` aj DIČ (Daňové identifikačné číslo) `vat_id_number`, ktorými sa tieto spoločnosti dajú dodatočne vyhľadať aj na iných portáloch.

Zadávateľ (Contracting authority)
Táto entita slúži na ukladanie údajov o zadávateľoch, ktorí už niekedy vytvorili nejakú verejnú zákazku. Okrem názvu `contracting_authority_name` daného zadávateľa obsahuje aj odkaz na portál *nen.nipez.cz*, na ktorom je možné nájsť ďalšie informácie o danom zadávateľovi.

Ponuka (Offer)
Entita reprezentuje ponuku vytvorenú spoločnosťou na nejakú verejnú zákazku. Eviduje cenu `price` a cenu s DPH `price_vat`, za ktorú by daná spoločnosť bola ochotná dodávať verejnú zákazku. Obsahuje aj informácie či daná ponuka bola spoločnosťou stiahnutá `is_withdrawn`, či bola ponuka zamietnutá z dôvodu príliš nízkej dopytovanej ceny `is_rejected_due_too_low`, alebo či daná spoločnosť reprezentuje asociáciu spoločností `is_association_of_suppliers`.

Adresa (Address)
Táto entita predstavuje adresu sídla spoločností alebo zadávateľov.

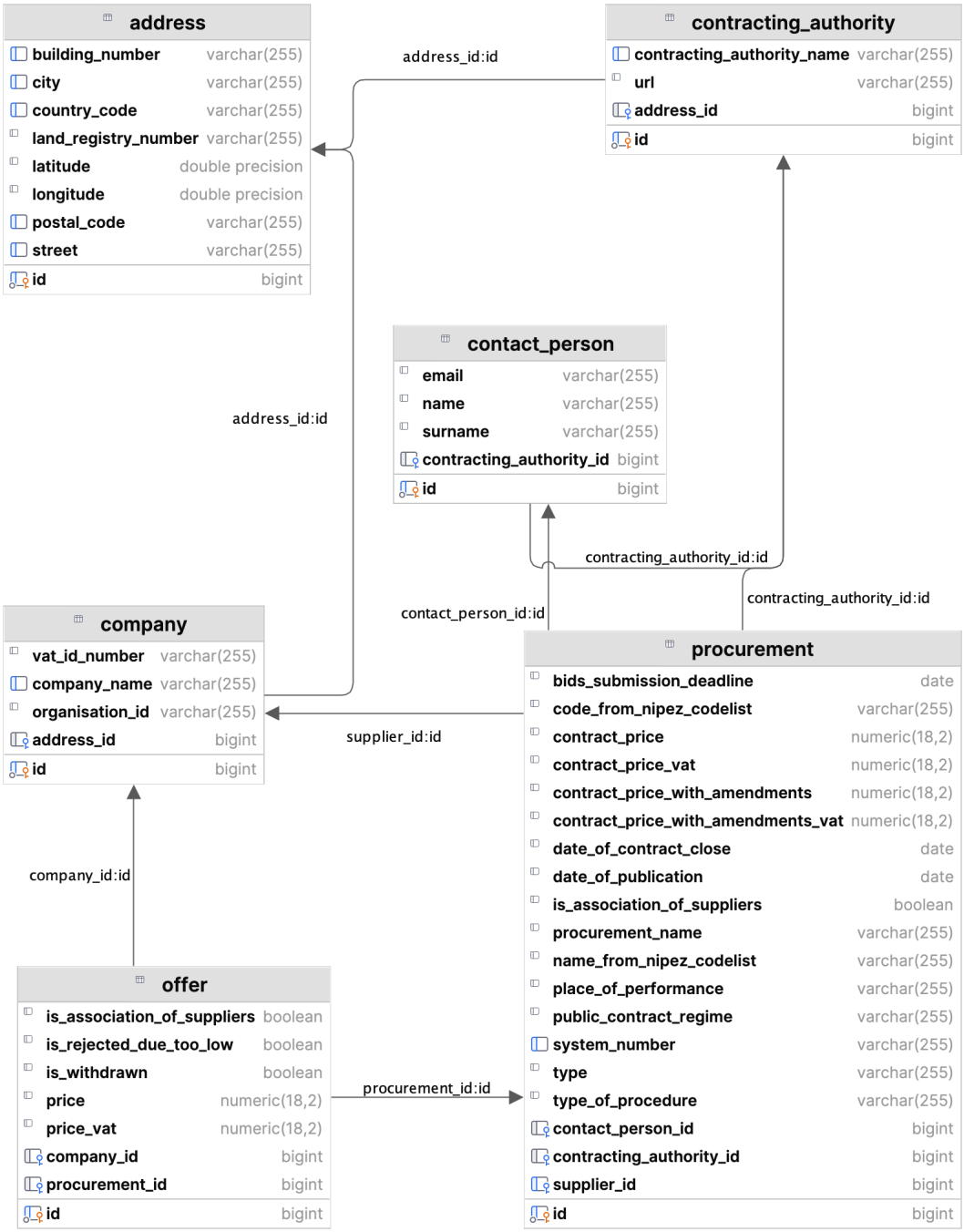
Kontaktná osoba (Contact person)
Kontaktné osoby sú zamestnanci zadávateľa, ktorí sú poverení správou danej verejnej zákazky.



Obr. 2.3 Konceptuálne schéma databázy.

2.7.3 Relačný diagram

Relačný diagram na rozdiel od konceptuálnej schémy znázorňuje reálny databázový systém s konkrétnymi implementačnými detailmi danej databázovej technológie. Medzi tieto detaily patrí napríklad dekompozícia M:N vzťahov alebo konkrétne dátové typy jednotlivých stĺpcov. Relačný diagram je znázornený na obrázku 2.4.



Obr. 2.4 Relačný diagram databázy.

2.8 Webový backend

Komponenta zabezpečuje sprostredkovanie dát z databázy klientom a poskytuje operácie výhradne na čítanie dát. Spomedzi predstavených prístupov k implementácií komunikácie medzi komponentami v 2.4 som pre komunikáciu medzi webovým frontendom a webovým backendom zvolil REST. Alternatívny prístup GraphQL považujem za zbytočne komplexný pre toto pomerne jednoduché API a SOAP som vylúčil z dôvodu využívania formátu XML, ktorý nie je natívne podporovaný zvolenou knižnicou na vizualizáciu dát. Komponenta tak predstavuje REST server, ktorý na komunikáciu využíva HTTP a dáta posiela vo formáte JSON.

Táto komponenta bude, podobne ako Scrapper, využívať programovací jazyk Java v kombinácií s frameworkom Spring Boot. Rovnako ako komponenta Scrapper, bude aj táto komponenta využívať `spring-boot-starter-data-jpa` na prácu s databázou. Na vytvorenie REST serveru bude využívať `spring-boot-starter-web` [29], z ktorej bude používať *open source* webový server `Tomcat`.

2.9 Webový frontend

Táto komponenta slúži na vizualizáciu dát formou webovej stránky. Na získavanie dát komponenta komunikuje výhradne s backendovou komponentou. Komponenta využíva nasledujúce technológie:

JavaScript

JavaScript [30] je interpretovaný programovací jazyk, podporujúci objektovo orientované programovanie, imperatívne programovanie a aj funkcionálne programovanie. Najčastejšie je využívaný ako skriptovací jazyk pre webové stránky, ale používa sa aj mimo prehliadač napríklad v kombinácii s prostredím *Node.js*.

JavaScript využíva programovanie založené na prototypoch [31]. Je to štýl objektovo orientovaného programovania, v ktorom triedy nie sú explicitne definované, ale sú odvodené pridaním vlastností a metód do inštancie inej triedy. Prototypové programovanie teda umožňuje vytváranie objektov bez definovania ich tried.

React

React je JavaScriptový framework, ktorý vytvorila spoločnosť Meta v roku 2013 s hlavným cieľom riešiť komplexnosť tvorby dynamických užívateľských rozhraní s často meniacimi sa dátami. Hlavným stavebným blokom v React aplikáciách sú React komponenty, pomocou ktorých sa stavia celé užívateľské rozhranie. [32] React som pre túto prácu vybral z dôvodu, že je aktuálne najrozšírenejším frameworkom na tvorbu užívateľských rozhraní [23] a jeho popularita, na rozdiel od alternatív, neustále rastie. Navyše na vizualizáciu dát na mape som sa rozhodol využiť framework Deck.gl, ktorý je navrhnutý na prácu s frameworkom React a množstvo príkladov z Deck.gl dokumentácie je napísaných práve v tomto frameworku.

React som využíval v kombinácií s technológiou Vite, ktorá slúži na zostavovanie aplikácie. Jej cieľom je poskytnúť rýchlejší a úspornejší vývoj moderných webových projektov. [33] Alternatívou na zostavovanie React aplikácií je nástroj CRA (*Create React App*), ktorý je síce podporovaný a odporúčaný samotným frameworkom React, ale pri zostavovaní aplikácie je v porovnaní s Vite mnohonásobne pomalší. Okrem produkčných či vývojových zostavovaní nástroj Vite výrazne urýchľuje aj *hot reloading*, čo je vývojárska funkcia umožňujúca okamžité zobrazenie zmien v aplikácii bez potreby manuálneho obnovenia stránky. [34]

Deck.gl

Deck.gl [35] je framework navrhnutý na vizualizácie veľkých sád údajov. Využíva pri tom WebGL [36], čo je JavaScriptové API navrhnuté na vykresľovanie interaktívnych 3D a 2D grafík priamo vo webových prehliadačoch. WebGL využíva grafické akcelerácie poskytované zariadením užívateľa, čím zvyšuje rýchlosť a efektívnosť vykresľovania grafík. Deck.gl mapuje dáta (zvyčajne zoznam JSON objektov) do vrstiev, ktoré následne zobrazuje na pohľadoch. Príkladom takéhoto zobrazenia je vrstva ikon zobrazená nad mapou, ktorá zastáva rolu pohľadov. Framework Deck.gl umožňuje aj efektívne vykresľovanie a kombináciu viacerých vrstiev súčasne.

Material UI

MUI (Material UI) [37] je *open source* knižnica, ktorá implementuje Material design od spoločnosti Google. Obsahuje rozsiahlu kolekciu predpripravených React komponentov, ktoré sa jednoducho modifikujú a sú pripravené na použitie v produkcii.

Kapitola 3
UI/UX návrh

Táto kapitola vysvetľuje dôležitosť UI (užívateľské rozhranie) a UX (užívateľská skúsenosť) návrhu. Približuje proces tvorenia wireframov obrazoviek a v tejto súvislosti predstavuje nástroj Figma, ktorý slúži práve na návrh UI/UX. Záver tejto kapitoly uvádza samotné návrhy jednotlivých obrazoviek.

UI/UX [38] sa skladá z dvoch rozdielnych, ale výrazne prepojených konceptov softvérového dizajnu, ktoré zohrávajú kľúčovú úlohu pri vývoji softvéru, nakoľko priamo ovplyvňujú ako užívatelia interagujú so systémom.

UI sa týka vizuálnych a interaktívnych prvkov. Patrí do neho návrh rozloženia, typografie, farebných schém, ikon, tlačidiel a iných vizuálnych prvkov, ktoré spoločne tvoria užívateľské rozhranie.

Na druhej strane, UX sa zaoberá návrhom celkovej užívateľskej skúsenosti pri práci so systémom. Zaoberá sa najmä zjednodušovaním a zefektívňovaním dosiahnutia cieľov jednotlivých užívateľov. Každý krok vedúci k dosiahnutiu cieľa by mal byť pre užívateľa zmysluplný, užívateľsky prívetivý a mal by spĺňať očakávania užívateľov. Tieto ciele sa pri UX návrhu dosahujú rôznymi metódami, od mapovania ciest užívateľov (*flow diagram*) až po testovanie použiteľnosti.

Na dizajnový návrh tejto práce bol využitý nástroj Figma, ktorý ponúka veľkú sadu funkcionalít, využiteľných pre celý proces UI/UX návrhu. Je to moderný webový nástroj na návrh dizajnu softvérových aplikácií, umožňujúci okrem klasického modelovania jednotlivých obrazoviek aj interakciu medzi nimi. Výsledkom teda môže byť aj veľmi jednoduchý prototyp aplikácie, ktorý umožňuje vykonať testovanie použiteľnosti koncovými užívateľmi pred samotnou implementáciou. Týmto sa Figma odlišuje od bežných nástrojov na kreslenie a to je aj dôvod, prečo som si vybral práve tento nástroj.[39]

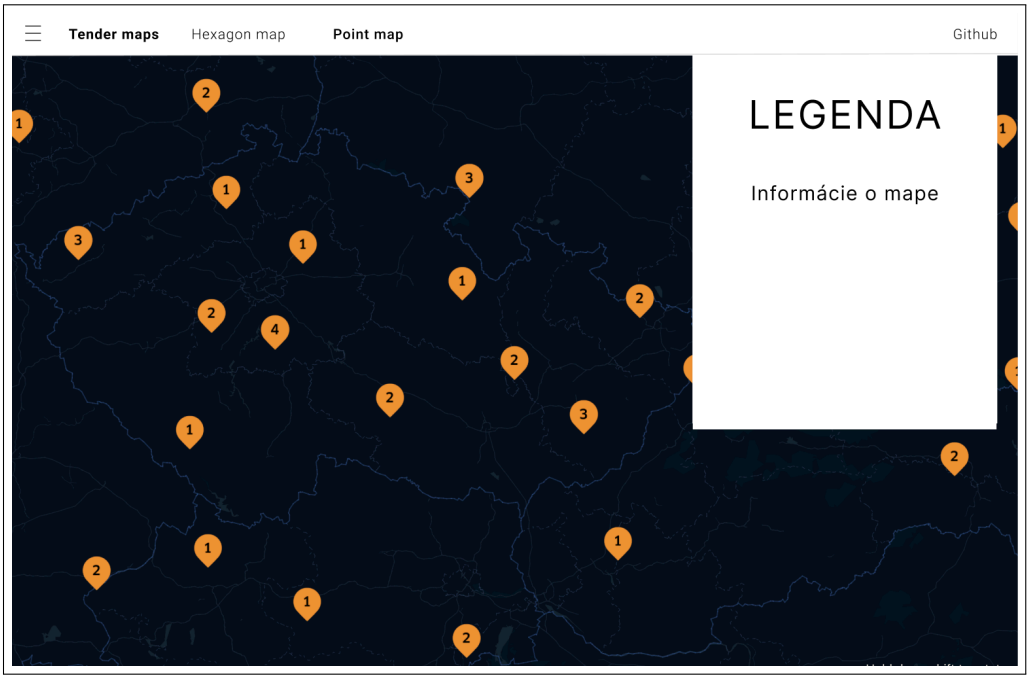
Návrh užívateľského rozhrania bol inšpirovaný užívateľským rozhraním webovej stránky frameworku *Deck.gl*. Inšpiroval sa minimalistickým dizajnom, farebnou schémou ale aj rozložením jednotlivých elementov na obrazovke, akými sú navigácia, bočné menu alebo legendy zobrazených máp.

3.1 Wireframing

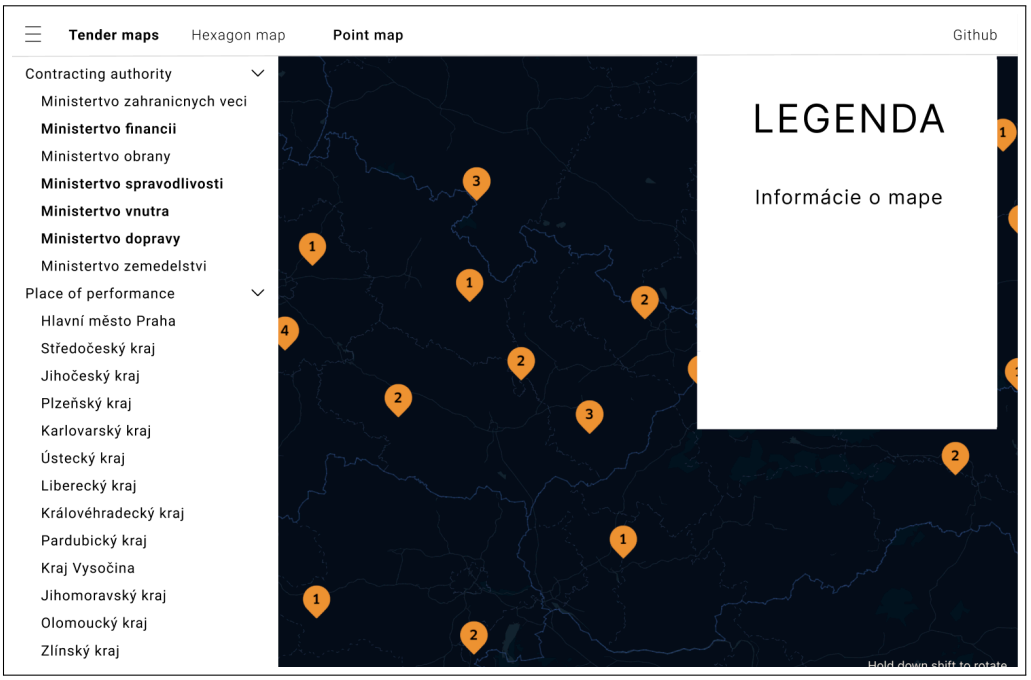
Wireframing predstavuje vizuálnu reprezentáciu základnej architektúry a funkčnosti webovej stránky. Je to len kostra aplikácie, pre ktorú konkrétne dizajnové rozhodnutia o jednotlivých elementoch nie sú prioritou.[40]

Obrázky 3.1 a 3.2 znázorňujú vytvorené wireframy. Oba wireframy zobrazujú rovnakú mapu, ale obrázok 3.2 má otvorené bočné menu, ktoré slúži na filtráciu zobrazených objektov. Týmto návrh adresuje identifikované funkčné požiadavky z 1.3.1 zamerané na filtráciu dát. Konkrétne je

v bočnom menu filtrácia podľa zadávateľa a filtrácia podľa miesta plnenia. Funkčný požiadavok na filtrovanie podľa dodávateľa je splnený samotnou interakciou s mapou, ktorá bude umožňovať približovanie, oddiaľovanie a pohyb po mape, čím efektívne zabezpečuje filtrovanie zobrazených dodávateľov.



Obr. 3.1 Návrh obrazovky zobrazujúcej mapu.



Obr. 3.2 Návrh obrazovky zobrazujúcej mapu s filtrom.

Kapitola 4
Implementácia

Táto kapitola sa zaoberá samotnou implementáciou a implementačnými detailmi aplikácie. Kapitola je rozdelená na podkapitoly podľa jednotlivých komponent, ktoré boli identifikované v rámci analýzy. Každá komponenta opisuje implementačné problémy a zdôvodňuje ich riešenia. V rámci frontend komponenty sú predstavené aj jednotlivé typy máp, ktoré boli zvolené pre vizualizáciu dát.

4.1 Scrapper

Táto časť sa zaoberá implementáciou dolovania a spracovania dát z portálu *nen.nipez.cz*. V úvode sa venuje návrhu implementácie a následne uvedie základnú logiku algoritmu na dolovanie dát. Ďalej predstavuje implementáciu požiadavku na periodické získavanie dát a vysvetľuje algoritmus na exponenciálne oneskorenie pri opakovaní neúspešných požiadaviek. Následne sa venuje problémom, ktoré nastali pri implementácii komponenty, akými bola chybovosť dát či pomalé dolovanie a záver tejto časti sa zaoberá monitoringom komponenty.

Na prácu s databázou využíva Hibernate poskytovaný frameworkom Spring Boot a na získanie a spracovanie HTML súborov využíva knižnicu JSOUP. Okrem samotného dolovania komponenta komunikuje s externými službami na obohatenie dát o geografické súradnice alebo zámenu peňažných mien do CZK.

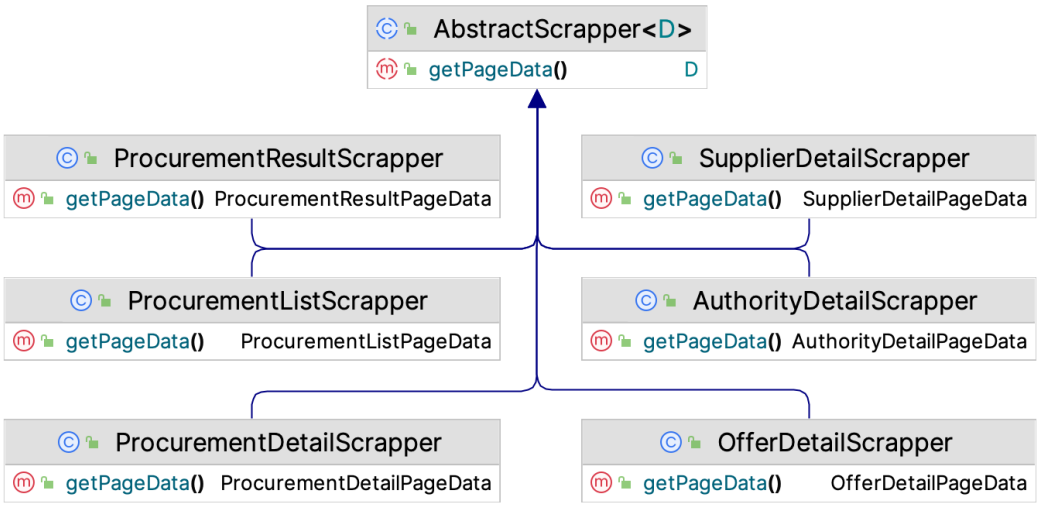
Komponenta je navrhnutá ako sada modulov, kde každý modul zabezpečuje nejakú konkrétnu funkcionalitu. Komponenta implementuje nasledovné moduly:

Doménový modul Model mapujúci Java objekty na relačné tabuľky.

Fetcher Modul zabezpečuje získanie HTML kódu webovej stránky a jej následné spracovanie do formátu DOM. Na zabezpečenie tejto funkcionality využíva knižnicu JSOUP a jej metódy `Jsoup.connect(url).get()`, ktorých návratová hodnota je DOM reprezentácia HTML stránky.

Geocoder Tento modul zabezpečuje geokódovanie adries na geografické súradnice. Rozhranie má aktuálne dve rôzne implementácie, prvou je `ProfinitGeocoder`, ktorá na geokódovanie využíva API od firmy Profinit a druhá je `GooogleGeocoder`, ktorá zas využíva Geocoding API od firmy Google.

Scrapper Modul zabezpečujúci získavanie všetkých potrebných dát z DOM reprezentácie webových stránok. Jednotlivé rozšírenia `AbstractScrapper` sú znázornené na obrázku 4.1, kde každé rozšírenie predstavuje konkrétnu stránku, ktorú komponenta spracováva.



Obr. 4.1 Modul Scrapper.

Exchanger Modul exchanger zabezpečuje zamieňanie peňažných mien s historickým pohľadom. Aktuálne má len jednu implementáciu, ktorá na získanie týchto dát využíva nástroj *fixer.io*.

Builder Z webu získavame množstvo nepovinných atribútov a nie všetky atribúty sa inicializujú v rovnakom čase, čo môže spôsobovať výskyt doménových objektov v nevalidnom stave naprieč aplikáciou. Na zamedzenie takéhoto stavu komponenta implementuje návrhový vzor *Builder*, ktorý umožňuje konštruovať komplexné objekty krok po kroku. [41]

DAO Modul slúžiaci na separáciu logiky perzistencie dát od zvyšku aplikácie. Využíva na to návrhový vzor DAO (Data Access Object), ktorý mimo iné uľahčuje aj prípadnú výmenu konkrétnej implementácie dátového úložiska. [42] Na jeho implementáciu sa používa rozhranie `JpaRepository`, ktoré poskytuje framework Spring Boot. Rozšírenie tohto rozhrania pridáva aj všetky CRUD operácie nad danou entitou.

4.1.1 Základná logika scrappovania dát

Scrappovanie dát pozostáva z načítania DOM reprezentácie webovej stránky a následného získania všetkých potrebných dát. Medzi týmito dátami sú aj odkazy na ďalšie stránky, do ktorých sa program zanoruje. Na vydolovanie jednej verejnej zákazky potrebuje komponenta získať a spracovať minimálne 6 rôznych stránok. Jednotlivé stránky sú popísané v nasledujúcej časti, väzby medzi stránkami sú zobrazené na obrázku 4.2.

Zoznam verejných zákaziek Stránka reprezentujúca zoznam verejných zákaziek. Implementuje stránkovanie, kde na jednej stránke je 50 verejných zákaziek.

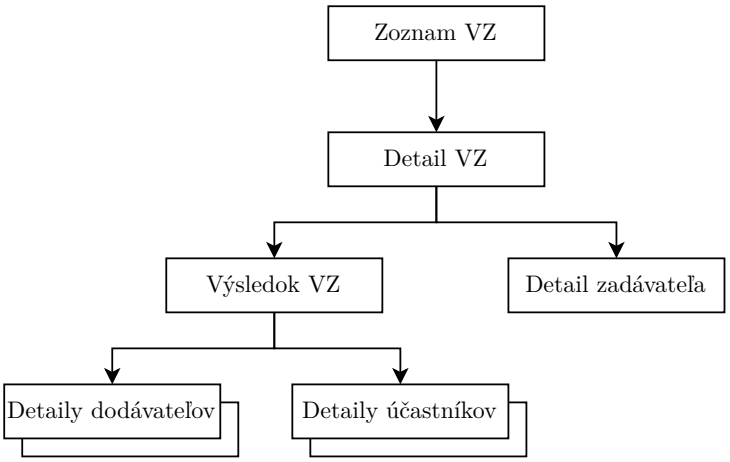
Detail verejnej zákazky Stránka obsahuje detailnejšie informácie o verejnej zákazke, akými sú rôzne kategorizácie alebo dátumy.

Výsledok verejnej zákazky Z tejto stránky sa dolujú informácie o výsledku verejnej zákazky, sú na nej zoznamy dodávateľov a účastníkov danej verejnej zákazky.

Detail zadávateľa Odkaz na túto stránku je dostupný z detailu verejnej zákazky. Obsahuje informácie o zadávateľovi danej verejnej zákazky, akými je napríklad adresa sídla daného zadávateľa.

Detail účastníka Odkaz na túto stránku je dostupný z výsledku verejnej zákazky. Obsahuje informácie o účastníkovi danej verejnej zákazky, akými je adresa sídla danej spoločnosti a cena, za ktorú by účastník dodával danú verejnú zákazku.

Detail dodávateľa Odkaz na túto stránku je dostupný z výsledku verejnej zákazky. Obsahuje informácie o dodávateľovi danej verejnej zákazky, akými je adresa sídla dodávateľa a zmluvná cena. V porovnaní s detailom účastníka obsahuje aj ďalšie informácie, akými sú napríklad ceny s dodatkami.



Obr. 4.2 Väzby medzi stránkami portálu *nen.nipez.cz*.

4.1.2 Periodické získavanie dát

Táto časť sa zaoberá implementáciou funkčného požiadavku na periodické získavanie dát. Tento požiadavok je implementovaný automatickým spúšťaním metódy v komponente Scrapper vo vopred stanovených intervaloch. Na toto sa využíva Spring anotácia `@Scheduled` [43], ktorá umožňuje periodické spúšťanie metód. Anotácia berie ako element `fixedDelay`, ktorým sa dá špecifikovať po akej dobe, po skončení predošlého behu, sa má metóda opäť spustiť. Využitie tejto anotácie je znázornené na ukážke kódu 4.1. Táto metóda sa bude spúšťať vždy 30 dní po dobehnutí predošlého behu programu a získa tak novovzniknuté verejné zákazky.

```
@Scheduled(fixedDelay = 30, timeUnit = TimeUnit.DAYS)
public void run(){
  int page = 1;
  List<String> systemNumbers = procurementController.getPageSystemNumbers(page);
  while(!systemNumbers.isEmpty()){
    scrapeProcurementList(systemNumbers);
    systemNumbers = procurementController.getPageSystemNumbers(++page);
  }
}
```

Listing 4.1 Metóda na periodické spúšťanie scrappingu.

4.1.3 Exponenciálne oneskorenie požiadaviek

Algoritmus exponenciálneho zvyšovania časového oneskorenia sa využíva na implementáciu logiky opakovania požiadaviek, ktoré zlyhali. Komponenta pri každom neúspešnom pokuse čaká exponenciálne dlhšie ako tomu bolo pri predchádzajúcom pokuse. Takýto prístup nepreťažuje portál *nen.nipez.cz* požiadavkami, čo v prípade tohto portálu viedlo ku násobne pomalším odpovediam. Samotné exponenciálne oneskorenie ale nie je dostačujúce. Ak by komponenta posielala požiadavky z viacerých vlákien súčasne, môže nastať situácia, kedy by viacero vlákien čakalo rovnako dlhú periódu a jednotlivé pokusy by tak nastávali v rovnakých časoch, čo by opäť mohlo viesť k preťažovaniu servera. Z tohto dôvodu sa k exponenciálnemu oneskoreniu pridáva *jitter*, ktorý pridáva náhodnosť do oneskorenia jednotlivých pokusov. [44]

V samotnom kóde je algoritmus implementovaný pomocou Spring anotácie `@Retryable` a je znázornený vo výpise kódu 4.2. Táto anotácia zabezpečuje opakované spúšťanie metódy `getDocumentWithRetry()`, v prípade vyhodenia výnimky typu `IOException`. Samotná logika oneskorenia je naimplementovaná Spring anotáciou `@Backoff`, ktorej parametre definujú počiatočné oneskorenie, multiplikátor oneskorenia, maximálne oneskorenie a parameter `random` pridáva *jitter*. Pre oneskorenie pokusu teda platí `backoff = random_between(delay, min(maxDelay, delay * 2 ** attempt)`. Spring ponúka aj anotáciu `@Recover`, ktorá zavolá metódu s touto anotáciou v prípade, že zlyhali všetky opakované pokusy metódy s anotáciou `@Retryable`.

```
@Retryable(retryFor = IOException.class,
  maxAttempts = 10,
  backoff = @Backoff(delay = 1000,
    multiplier = 2,
    maxDelay = 1_024_000,
    random = true
  )
)
public Document getDocumentWithRetry(String url) throws IOException {
    return Jsoup.connect(url).get();
}
```

Listing 4.2 Metóda na získavanie DOM webovej stránky z URL.

4.1.4 Chybovosť dát

Jedným z problémov, ktorý nastával pri implementácii tejto komponenty bola pomerne častá nekonzistentnosť a chybovosť dát na portáli *nen.nipez.cz*. Za najzávažnejšie považujem chyby v zadaných peňažných cenách verejných zákaziek. Príkladom takejto verejnej zákazky je dodávka čiernych tričiek pre Políciu Českej republiky na dobu dvoch rokov za vyše 7 biliónov (10^{12}) CZK[1]. Za podobné chyby boli upravované číselné dátové typy pre peňažné hodnoty v databáze.

Dáta sú dolované z anglickej verzie portálu a množstvo ďalších chýb tak súviselo so zlým prekladom. Napríklad jedným zo stavov v akom sa môže verejná zákazka nachádzať je „ukončení plnění“ čo bolo preložené ako „performance in progress“, čo v preklade znamená pravý opak ukončeného plnenia, a teda prebiehajúce plnenie. Ďalším prekvapením bol fakt, že za celé pôsobenie tohto portálu bolo zákaziek v stave zadaná skoro dva krát toľko ako zákaziek v stave ukončená.

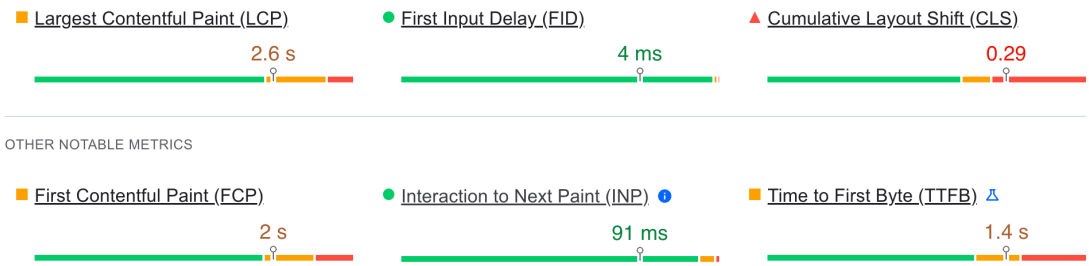
O oboch týchto skutočnostiach som informoval správcov portálu. Zlý preklad bol do pár týždňov opravený a o zjavne chybnom stave pomerne značnej časti verejných zákaziek som dostal

[1] Verejná zákazka je dostupná na https://nen.nipez.cz/en/verejne-zakazky/detail-zakazky/N006-21-V00008639. Cena sa mohla od času písania práce upraviť.

odpoveď, že si ho užívatelia zabúdajú aktualizovať. Po tomto zistení som začal získavať dáta o verejných zákazkách, ktoré boli v jednom z týchto stavov. Boli nájdené aj rôzne menšie chyby, ako napríklad zákazka v stave ukončená ale bez informácií o výsledkoch tohto obstarávania alebo jedného dodávateľa s dvoma rôznymi názvami spoločnosti.

4.1.5 Pomalý scrapping

Ďalej som sa zaoberal riešením pomalého scrappingu dát. Ako vidno z obrázku 4.2, pre každú verejnú zákazku sa musia získať dáta z minimálne piatich stránok portálu *nen.nipez.cz* a k tomu samotný zoznam verejných zákaziek. Odpovede tohto portálu sú však pomerne dosť pomalé, o čom svedčia aj výkonnostné metriky nástroja PageSpeed od spoločnosti Google znázornené na obrázku 4.3. Zaujímavá je najmä metrika TTFB (Time to First Byte) [45], ktorá meria čas medzi odoslaním požiadavku a prvým prijatým bajtom odpovede. Pre porovnanie, alternatívny portál *tenderarena.cz* dosahuje hodnotu TTFB na úrovni 0,4 s. [46]



Obr. 4.3 Výkonnostné metriky portálu *nen.nipez.cz*. [47]

Pomocou nástroja profiler som zistil, že ďalšou metódou, kde komponenta strávila značný čas, bolo geokódovanie českých adries prostredníctvom Profinit API.

Pre lepšie pochopenie a kvantifikáciu dĺžky trvania týchto procesov som sa rozhodol pre vytvorenie telemetrického systému, ktorý by monitoroval beh tejto komponenty. Samotné vytvorenie tohto systému predstavím v nasledujúcej sekcii 4.1.6, tu sa však budem odkazovať na výsledky monitoringu, ktoré sú znázornené na obrázku 4.4. Na tomto obrázku, sú okrem iných informácií, uvedené aj nasledujúce priemery:

- získanie DOM reprezentácie stránky z portálu *nen.nipez.cz* (3,15 s),
- geokódovanie s Profinit API (3,72 s),
- geokódovanie s Google API (176 ms),

Rozdiely medzi metrikou TTFB a dátami z monitoringu sú spôsobené faktom, že táto metrika meria prijatie len prvého bajtu a neparsuje získaný HTML kód do formátu DOM. Navyše do hodnôt z monitoringu sa započítava aj exponenciálne oneskorenie neúspešných požiadavkov.

Pre urýchlenie scrappovania som sa preto rozhodol využiť viacero vlákien, ktoré asynchrónne získavajú a geokódujú dáta. Paralelizmus je implementovaný na úrovni ukladania jednej verejnej zákazky a vytváranie nových záznamov v databáze sa vykonáva synchrónne, po získaní všetkých údajov potrebných na uloženie danej zákazky. Na správu vlákien sa využíva Spring `ThreadPoolTaskExecutor` a na vytváranie nových vlákien anotácia `@Async` v kombinácií s `CompletableFuture`, ktoré ponúka jazyk Java.

Komponentu sa však neoplatí spúšťať na veľa vláknach (optimum je len 5 až 10 vlákien), nakoľko nadmerné preťažovanie portálu *nen.nipez.cz* má za následok násobne pomalšie odpovede. Aktuálna implementácia paralelizmu tak nie je ideálna a možné vylepšenie je popísane v 6.1.

4.1.6 Monitoring komponenty

Pomalé scrappovanie dát spôsobuje, že táto komponenta bude pri počiatočnom získavaní všetkých zákaziek pomerne dlho nepretržite bežať. Z tohto dôvodu je využitá sada nástrojov na monitorovanie komponenty počas jej behu:

Spring Boot Actuator
Spring Boot Actuator slúži na monitorovanie Spring Boot aplikácií. Vystavuje endpoint `/actuator`, na ktorom sú dostupné jednotlivé metriky. [48]

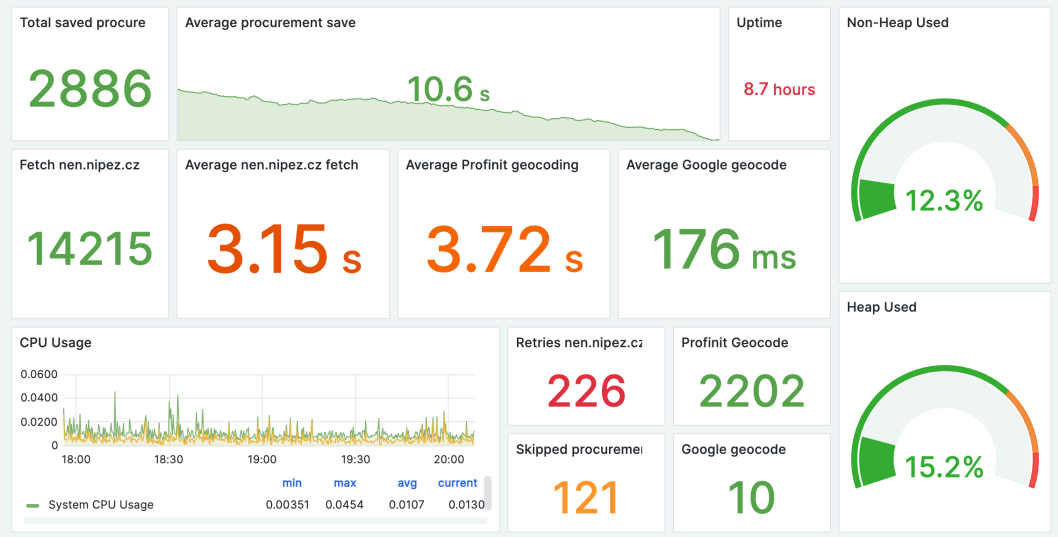
Micrometer
Micrometer ponúka jednoduché rozhranie pre nástroje na monitorovanie aplikácií a podporuje množstvo populárnych monitorovacích systémov, ako sú Atlas, Datadog alebo Prometheus. [49] Tento nástroj ponúka aj anotáciu `@Timed`, ktorú komponenta využíva na vytvorenie časovača pre metódy zabezpečujúce napríklad scrappovanie alebo geokódovanie. Zbierané metriky touto anotáciou sú znázornené na obrázku 4.3 a sú dostupné na endpointe `/actuator/prometheus`.

```
scrapper_profinit_geocode_seconds_count 2208.0
scrapper_profinit_geocode_seconds_sum 8219.086296332
scrapper_profinit_geocode_seconds_max 3.95709325
```

Listing 4.3 Metriky generovaná z anotácie `@Timed`.

Prometheus
Prometheus je *open source* monitorovací systém, ktorý zbiera a ukladá metriky spolu s časovou pečiatkou, ktorá uvádza čas, kedy bola daná metrika zaznamenaná. [50]

Grafana
Na vizualizáciu metrík slúži *open source* nástroj Grafana. Tento nástroj podporuje vytváranie panelov, ktoré môžu obsahovať rôzne typy grafov, meradiel alebo počítadiel. Príklad takejto vizualizácie je znázornený na obrázku 4.4.



Obr. 4.4 Vizualizácia monitoringu v Grafane.

4.2 Webový backend

Táto časť sa venuje implementácii komponenty webový backend, ktorej hlavným cieľom je poskytovanie dát klientom. V úvode predstaví a zdôvodní architektúru komponenty. Následne popíše spôsob implementácie funkčných požiadavkov zameraných na filtráciu dát a v závere uvedie API dokumentáciu.

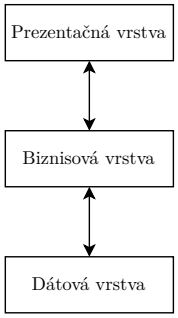
Vzhľadom na to, že sa implementuje celá logika ohľadne získavania a spracovania dát v komponente Scrapper, predstavuje táto komponenta jednoduché REST API, ktoré slúži len na čítanie dát. Aj napriek tomu je architektúra komponenty navrhnutá ako trojvrstvová, aby ju bolo možné efektívne rozširovať a testovať.

Vo vrstvovej architektúre [51] má každá vrstva svoju špecifickú rolu, zodpovednosť a vytvára istú formu abstrakcie pre ostatné vrstvy. Medzi hlavné zásady tejto architektúry patrí vlastnosť, že jednotlivé požiadavky musia vždy prejsť cez každú vrstvu a nemôže tak nastať situácia, kedy by sa napríklad prezentačná vrstva priamo dopytovala dátovej vrstvy. Jednotlivé vrstvy by mali byť od seba izolované, zmeny v jednej vrstve by mali minimálne ovplyvňovať ostatné vrstvy.

Prezentačná vrstva Prezentačná vrstva býva častokrát užívateľské rozhranie aplikácie. V mojom prípade je to komunikačná vrstva, zabezpečujúca interakcie s klientami. V tejto vrstve využívam návrhový vzor DTO (Data Transfer Object) [52], ktorým definujem aké údaje sú zdieľané s klientami.

Biznisová vrstva Táto vrstva zabezpečuje celú biznisovú logiku komponenty.

Dátová vrstva Dátová vrstva zabezpečuje získavanie dát z dátového úložiska.



Obr. 4.5 Trojvrstvová architektúra. [51]

4.2.1 Filtrácia dát

Najkomplexnejšou časťou tejto komponenty bola implementácia jednotlivých dotazov podporujúcich filtráciu dát. Napríklad na mapové zobrazenie spoločností, ktoré sa uchádzali o verejné zákazky, ale nikdy nevyhrali, bolo nutné spájanie štyroch tabuliek a ak by sa k tomu pridali ešte aj všetky formy filtrácie z funkčných požiadaviek 1.3.1, musel by sa výsledok spájať aj s ďalšou tabuľkou. Navyše filtrácia funguje nad množinou zadávateľov a množinou miest plnenia a v prípade ak je niektorá z týchto množín prázdna, tak ju má dotaz ignorovať a vrátiť nefiltrované dáta.

Na implementáciu týchto dotazov som sa rozhodol využiť Spring špecifikácie [53], ktoré umožňujú vytváranie databázových dotazov programovo priamo v Jave a na použitie stačí rozšíriť repozitár o `JpaSpecificationExecutor`. Samotné vytvorenie špecifikácie je znázornené na výpise kódu 4.4, ktorý implementuje dotaz opísaný v predchádzajúcom odseku. Na

zavolanie dotazu so špecifikáciou stačí parametrom poskytnúť vytvorenú špecifikáciu metóde `repository.findAll(specification)`.

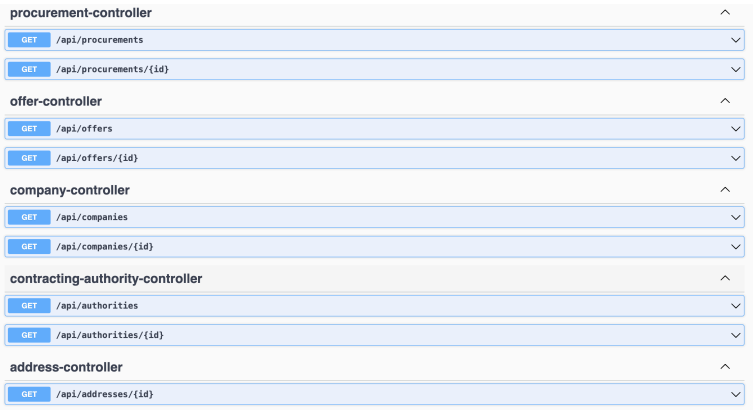
Podľa môjho názoru prináša takýto prístup k implementácii dotazov radu výhod. Komplexné dotazy umožňuje rozdeliť na menšie a jednoduchšie predikáty, ktoré sa následne spájajú na vytvorenie zložitejších dotazov. Navyše tento prístup poskytuje abstrakciu nad prácou s databázovými tabuľkami a špecifikácie sa vytvárajú na úrovni Java objektov. Napríklad na získanie spoločností, ktoré nikdy nedodali žiadnu verejnú zákazku stačí zistiť, či je prázdna množina `suppliedProcurements`, ktorá je klasickou členskou premennou triedy `Company`.

```
public static Specification<Company> getCompanies(
  List<String> placesOfPerformance,
  List<Long> contractingAuthorityIds,
  Boolean hasExactAddress,
  boolean isSupplier) {
    return (root, query, criteriaBuilder) -> {
      query.distinct(true);
      Predicate supplierStatusPredicate = isSupplier
        ? criteriaBuilder.isNotEmpty(root.get("suppliedProcurements"))
        : criteriaBuilder.isEmpty(root.get("suppliedProcurements"));
      return criteriaBuilder.and(supplierStatusPredicate,
        getPlaceOfPerfPred(root, criteriaBuilder, placesOfPerformance),
        getExactAddressPred(root, criteriaBuilder, hasExactAddress),
        getAuthorityPred(root, criteriaBuilder, contractingAuthorityIds));
    };
}
```

Listing 4.4 Príklad metódy na vytvorenie špecifikácie pre JPA dotaz.

4.2.2 API dokumentácia

Na vytvorenie API dokumentácie bol využitý *open source* nástroj Swagger, ktorý vie dokumentáciu vygenerovať z Java kódu a zobrazovať ju vo formáte webovej stránky 4.6. Generácia API dokumentácie do značnej miery uľahčuje aj jej následné udržiavanie.



Obr. 4.6 Swagger API dokumentácia.

4.3 Webový frontend

Táto časť sa venuje implementácii webového frontendu, ktorý slúži na vizualizáciu dát prostredníctvom webovej stránky. V úvode opisuje funkcionálny prístup k vytváraniu React komponent a vysvetľuje jeho výhody. Následne predstaví nástroj Axios a ukáže jeho využitie na získanie dát od webového backendu. V závere opíše a zdôvodní výber druhov máp pre samotnú vizualizáciu dát.

Architektúra tejto mikroslužby spočívala v rozdelení užívateľského rozhrania na React komponenty, ktoré sú základným blokom React aplikácií. Služba využíva funkcionálny prístup k vytváraniu komponent, čo umožňuje použitie aj React hooks. Hooky sú špeciálne funkcie, ktoré je možné volať len na začiatku komponent a umožňujú využívanie rôznych funkcií Reactu. V práci sa najčastejšie využíva stavový hook `useState()`, pomocou ktorého sa vytvárajú nové stavové premenné. Vývoj pomocou funkcionálnych komponent a hookov je aktuálne doporučený prístup k vývoju React aplikácií.[54]

Na komunikáciu s backendom je využitý HTTP klient Axios, ktorý je založený na JavaScript API `Promise`. Objekt `Promise` [55] reprezentuje eventuálne dokončenie alebo zlyhanie asynchrónnej operácie a jeho výslednú hodnotu. Výpis kódu 4.5 znázorňuje využitie Axios klienta. React komponenty jednotlivých máp volajú túto funkciu s príslušnými parametrami v hooku `useEffect()` na získanie dát, ktoré sú potrebné pre danú mapu. Táto funkcionalita je znázornená na výpise kódu 4.6. Tento hook umožňuje definovať aj pole závislostí, v ktorom sa môžu nachádzať reaktívne hodnoty, akými sú napríklad stavové premenné alebo funkcie definované v danej React komponente. Hook `useEffect()` sa zavolá pri prvom vykreslení komponenty a zároveň pri každej zmene nejakej z jeho závislostí. [56] Hook z výpisu kódu 4.6 sa teda zavolá aj pri každej zmene stavových premenných `filterLocations` a `filterAuthorities`, v ktorých je uložený aktuálny stav zvolených filtrov daným užívateľom.

```
async function fetchData(path, setFetchedData) {
  try {
    const url = apiBaseUrl + path;
    const response = await axios.get(url);
    setFetchedData(response.data);
  } catch (error) {
    console.error("Failed to fetch data:", error);
    showErrorPopUp(error);
  }
}
```

Listing 4.5 Funkcia na získanie zadávateľov.

```
useEffect(() => {
  fetchData(
    addFiltersToPath(PROCUREMENTS_PATH,{"supplierHasExactAddress": true}),
    setData
  );
}, [filterLocations, filterAuthorities, fetchData, addFiltersToPath]);
```

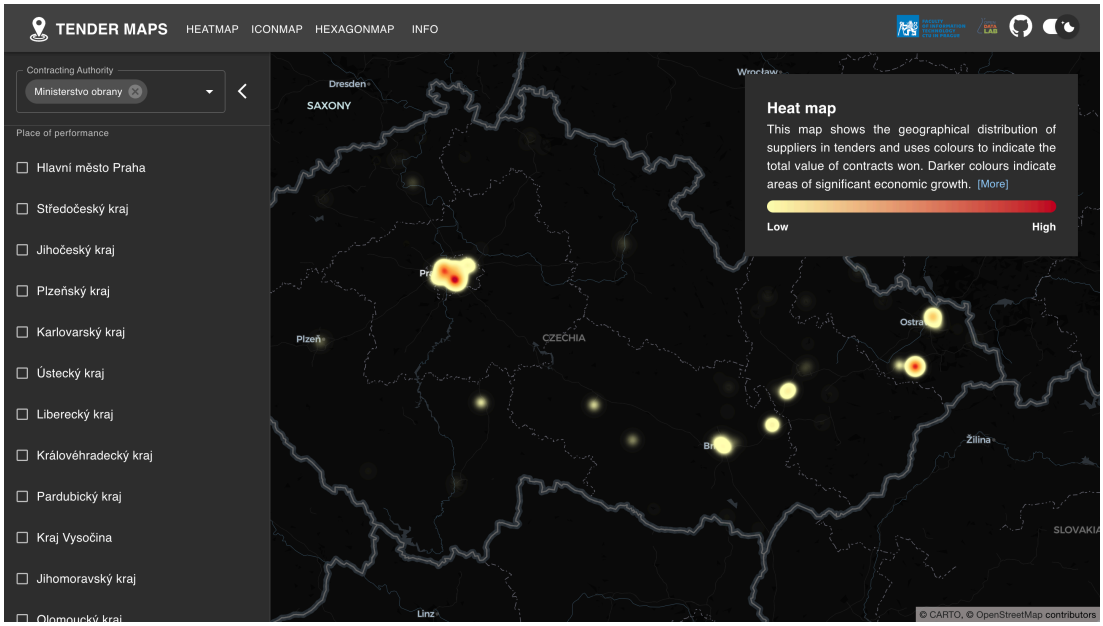
Listing 4.6 Príklad využitia useEffect() hooku.

4.3.1 Mapy

Táto časť sa zaoberá zvolenými metódami vizualizácie dát. Na vizualizáciu sú využité tri rôzne typy interaktívnych máp, ktoré poskytuje framework *Deck.gl*. Dáta na každej z týchto máp je možné filtrovať podľa zadávateľa zákazky a podľa miesta realizácie. Na filtrovanie podľa zadávateľa som využil React komponentu Material UI `AutoComplete`, ktorá implementuje textové vyhľadávanie s ponukou možností.

4.3.1.1 Heat mapa

Heat mapa znázorňuje distribúciu dodávateľov verejných zákaziek, kde je intenzita farby vypočítaná z kumulatívnej ceny dodávaných zákaziek danou spoločnosťou. Farebné spektrum začína na bledej žltej a končí na tmavej červenej. Táto forma vizualizácie môže poskytovať intuitívny prehľad o významných lokalitách ekonomickej aktivity v oblasti verejných obstarávaní v Českej republike. Táto mapa je znázornená na obrázku 4.7.

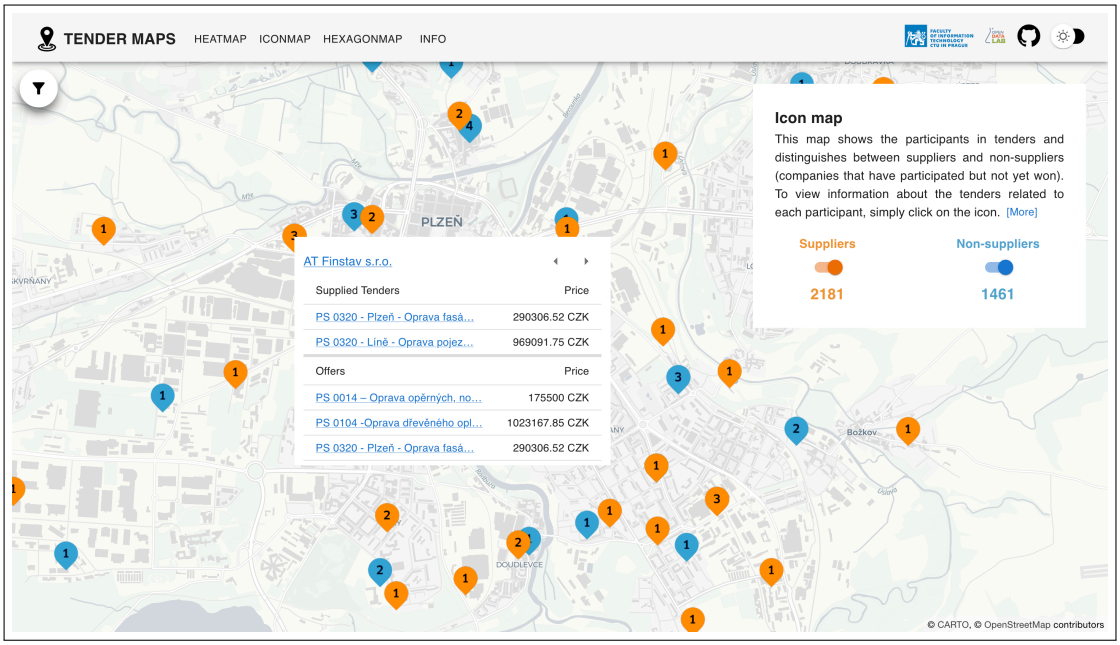


Obr. 4.7 Heat mapa s filtráciou podľa zadávateľa.

4.3.1.2 Ikonová mapa

Táto vizualizácia predstavuje interaktívnu mapu s ikonami, kde každá ikona zobrazuje spoločnosť, ktorá sa niekedy zapojila do verejného obstarávania v Českej republike. Farebne odlišuje medzi dodávateľmi a spoločnosťami, ktoré sa zatiaľ len uchádzali, ale nikdy nedodávali žiadnu zákazku. Užívatelia môžu interagovať s týmito ikonami, aby získali zoznamy dodaných zákaziek a zoznam všetkých ponúk, vytvorených danou spoločnosťou.

Táto mapa využíva funkcionalitu knižnice Deck.gl, ktorá umožňuje efektívne vykresľovať viacero vrstiev na sebe. Mapa vykresľuje dve rôzne `IconLayer`, jednu na vykreslenie dodávateľov a druhú na spoločnosti, ktoré nikdy nedodávali žiadne zákazky. Implementoval som aj dva prepínače, ktorými užívatelia majú možnosť skryť niektorú z týchto vrstiev. Mapa je znázornená na obrázku 4.8

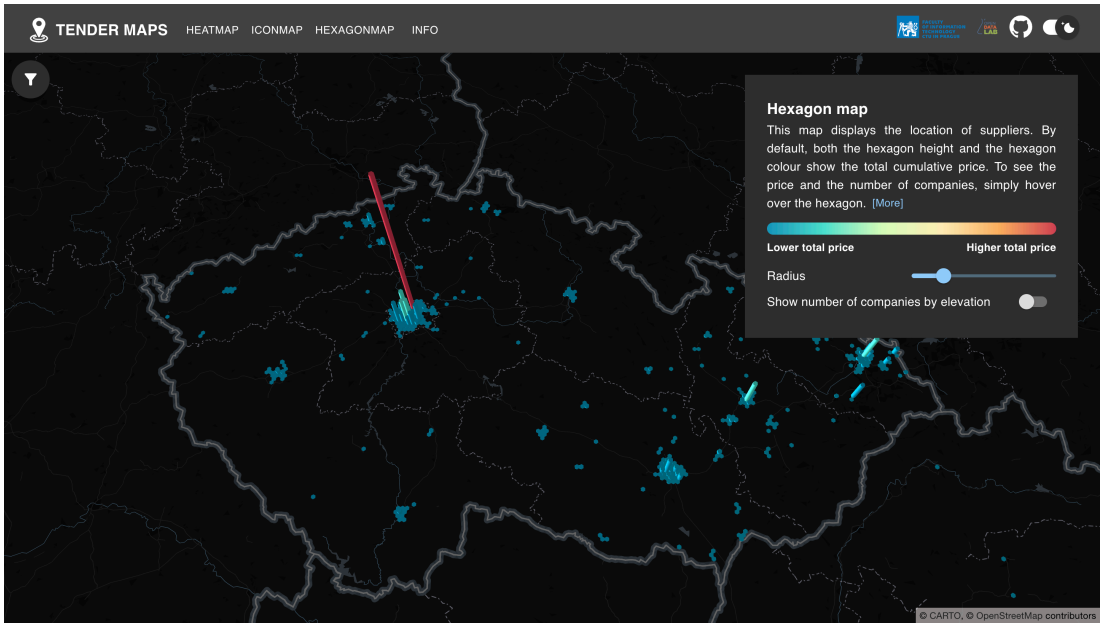


Obr. 4.8 Ikonová mapa s otvoreným detailom spoločnosti.

4.3.1.3 Hexagónová mapa

Hexagónová 3D mapa slúži, podobne ako heat mapa, na vizuálne znázornenie geografickej distribúcie dodávateľov zapojených do verejného obstarávania. Na rozdiel od heat mapy, však tieto informácie zobrazuje pomocou hexagónových stĺpcov. Toto zobrazenie prináša svoje výhody. Napríklad umožňuje zobrazovať dve rôzne informácie súčasne, jednu informáciu výškou stĺpca a druhú farbou stĺpca. Podľa predvolených nastavení mapy aj výška aj farba zobrazujú kumulatívnu cenu všetkých dodávaných zakaziek spoločnostiami v danom hexagóne. Implementoval som ale aj prepínač, ktorým užívatelia vedia zmeniť toto predvolené nastavenia a výškou stĺpca zobrazovať počet spoločností v danom hexagóne. Posúvač zase slúži na zmenu veľkosti jednotlivých hexagónov.

Ďalšou výhodou tejto mapy je možnosť užívateľov interagovať so stĺpcami a získať tak dodatočné údaje o spoločnostiach nachádzajúcich sa v tejto oblasti. Ak užívateľ ukáže na niektorý zo stĺpcov, zobrazí sa okno obsahujúce informácie o celkovej peňažnej hodnote dodávaných zákaziek spoločnosťami v danom hexagóne a počet spoločností v hexagóne. Nevýhodou v porovnaní s heat mapou je nereaktívne približovanie a fakt, že jednotlivé stĺpce prekrývajú samotnú mapu. Mapa je znázornená na obrázku 4.9.



Obr. 4.9 Hexagónová mapa.

Kapitola 5

Testovanie a nasadenie

Táto kapitola sa venuje posledným dvom fázam životného cyklu vývoja softvéru, ktorými je testovanie a nasadenie. Prvá časť sa venuje testovaniu a je rozdelená na testovanie komponenty Scrapper a komponenty Backend, nakoľko pri nich boli využité rôzne typy testovania. V rámci prvej časti sú opísané aj využité testovacie frameworky. Druhá časť kapitoly sa venuje nasadeniu systému. Zameria sa na proces kontajnerizácie a na nástroj Docker. V rámci tejto časti je uvedený aj nástroj Matomo, ktorý je využívaný na získavanie analytických informácií o návštevnosti vytvorenej webovej stránky.

5.1 Scrapper

Komponenta implementuje samotný scrapping, geokódovanie, zamieňanie peňažných mien a operácie nad databázou. Navyše využíva paralelizmus na urýchlenie dolovania dát. Z tohto dôvodu sa na otestovanie tejto komponenty využíva kombinácia unit a integračných testov. Na testovanie komponenta využíva testovací framework JUnit, ktorý uľahčuje implementáciu automatických a opakovateľných testov. Na zistenie pokrytia testov sa využíva knižnica JaCoCo. Ukážka výstupu z tejto knižnice je znázornená na obrázku 5.1.

Na testovanie využíva komponenta *in-memory* databázu H2. Tento typ databáz ukladá dáta len do RAM pamäte. Takýto prístup výrazne skracuje reakčné časy tým, že eliminuje potrebu prístupu k hlavnej pamäti počítača. [57]

Unit (jednotkový) test [58] je automatický test, ktorý izolovane testuje menšiu časť kódu porovnávaním skutočného stavu od očakávaného. Testovanými časťami sú v Jave najčastejšie triedy a jednotkové testy tak pokrývajú jednu alebo viacero metód danej triedy. Na testovanie modulu, ktorý zabezpečuje extrakciu dát z HTML kódu sú vytvorené a verzované pomocné súbory obsahujúce HTML kód testovaných stránok. Tieto súbory sa v prípade vymazania automaticky vytvoria, ale vymazanie prináša riziko zmeny dát, čo môže spôsobiť zlyhávanie testov.

Integračné testovanie [59] sa na rozdiel od unit testovania vykonáva na úrovni modulov. Tento typ testovanie kladie dôraz najmä na interakcie medzi jednotlivými modulmi. V práci sa integračné testy využívajú aj na testovanie externých API služieb. Napríklad API služby na geokódovanie adries, pri ktorých by izolované unit testovanie neodhaľovalo prípadné zmeny v danej API službe. Integračnými testami je pokryté aj ukladanie konkrétnych entít, kde sa testuje väčšina modulov tejto komponenty ako jeden celok.

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
cz.cvut.fit.bap.parser.domain		53%		35%	75	141	129	265	67	131	0	6
cz.cvut.fit.bap.parser.controller.geocoder		79%		75%	6	21	25	87	2	13	1	3
cz.cvut.fit.bap.parser.controller		90%		82%	8	60	14	188	0	34	0	7
cz.cvut.fit.bap.parser.controller.scrapper		94%		78%	13	80	15	200	2	55	0	8
cz.cvut.fit.bap.parser.controller.fetcher		70%		n/a	5	14	12	33	5	14	1	3
cz.cvut.fit.bap.parser.controller.currency_exchanger		97%		78%	5	20	11	220	2	13	1	3
cz.cvut.fit.bap.parser		78%		100%	1	11	8	36	1	9	0	2
cz.cvut.fit.bap.parser.controller.data		99%		50%	4	66	0	22	0	62	0	10
cz.cvut.fit.bap.parser.controller.builder		100%		n/a	0	12	0	50	0	12	0	3
cz.cvut.fit.bap.parser.business		100%		n/a	0	14	0	22	0	14	0	7
Total	630 of 4,952	87%	44 of 164	73%	117	439	214	1,123	79	357	3	52

Obr. 5.1 Pokrytie testov komponenty Scrapper.

5.2 Backend

Backend sa stará o filtráciu a prípravu dát pre frontend. Okrem tejto logiky je komponenta pomerne jednoduchá a neobsahuje množstvo inej logiky. Komponenta opäť využíva in-memory databázu H2. Pre testovacie účely sa databáza naplňuje dátami pomocou SQL skriptu `data.sql`, ktorý sa automaticky spúšťa po vytvorení tabuliek. Komponenta opäť využíva knižnicu JaCoCo na zistenie pokrytia testov. Výstup z tejto knižnice pre túto komponentu je znázornený na obrázku 5.2.

Na testovanie komponenty sa využívajú integračné testy, ktoré testujú jednotlivé endpointy API. Na testovanie API sa využíva testovací framework `MockMvc` od Springu, ktorý mockuje požiadavky a odpovede bez nutnosti bežiaceho webového serveru. [60] Príklad takéhoto testu je znázornený vo výpise kódu 5.1.

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed
cz.cvut.fit.bap.procurements.api.procurements_api.controller.dto		100%		n/a	0	33	0	5	0
cz.cvut.fit.bap.procurements.api.procurements_api.controller		100%		n/a	0	12	0	38	0
cz.cvut.fit.bap.procurements.api.procurements_api.controller.dto.converter		100%		n/a	0	10	0	28	0
cz.cvut.fit.bap.procurements.api.procurements_api.bussiness		100%		n/a	0	11	0	20	0
cz.cvut.fit.bap.procurements.api.procurements_api.utilities.specifications		93%		83%	9	36	6	63	3
cz.cvut.fit.bap.procurements.api.procurements_api		37%		n/a	1	2	2	3	1
cz.cvut.fit.bap.procurements.api.procurements_api.domain		28%		0%	57	90	99	141	47
Total	323 of 1,270	74%	26 of 56	53%	67	194	107	298	51

Obr. 5.2 Pokrytie testov backend komponenty.

```
@Test
public void testReadAllNoParams() throws Exception {
  mockMvc.perform(get("/api/procurements").contentType(MediaType.APPLICATION_JSON))
    .andExpect(status().isOk())
    .andExpect(content().contentTypeCompatibleWith(MediaType.APPLICATION_JSON))
    .andExpect(jsonPath("$", Matchers.hasSize(4)))
    .andExpect(jsonPath("$[0].id", Matchers.is(1)))
    .andExpect(jsonPath("$[1].id", Matchers.is(2)));
}
```

Listing 5.1 Test pomocou MockMvc.

5.3 Nasadenie

Táto časť sa zaoberá nasadením systému, zameria sa pri tom na softvérový proces kontajnerizácie. V tejto súvislosti predstaví platformu Docker a nástroj Docker-compose a opíše proces spustenia aplikácie. V závere uvedie nástroj Matomo, ktorý je využívaný na získavanie analytických informácií o užívateľoch webovej stránky.

5.3.1 Docker

Docker [61] je kontajnerizačná platforma umožňujúca zabaľovanie softvéru do kontajnerov a jeho spúšťanie na rôznych prostrediach.

Kontajner je samostatne spustiteľná softvérová aplikácia alebo služba, obsahujúca všetky potrebné závislosti na beh daného softvéru, čím napomáhajú k izolácií behu od hosťovacieho stroja. Docker kontajnery bežia na rôznych počítačoch alebo virtuálnych strojoch, na ktorých je nainštalovaný *Docker engine*. Kontajnery umožňujú modifikácie počas behu programu, akými sú napríklad zmeny konfigurácie alebo inštalácia softvéru.

Docker *image* je samostatný spustiteľný súbor slúžiaci na vytváranie kontajnerov. Vytvára sa z textových súborov nazývaných *Dockerfile*, ktoré obsahujú inštrukcie na vytvorenie Docker *image*. Narozdiel od kontajnerov, sú Docker *image* nemeniteľné a urobenie zmien tak vždy znamená vytvorenie nového *image*. V tejto práci má každá komponenta naimplementovaný vlastný *Dockerfile*.

Docker-compose [62] je nástroj na vytváranie a spúšťanie Docker aplikácií pozostávajúcich z viacerých kontajnerov. V práci sa Docker-compose využíva na uľahčenie nasadenia celého systému, ktorý pozostáva z viacerých komponentov.

5.3.2 Spustenie aplikácie

Na spustenie aplikácie stačí naklonovať aplikáciu z verzovacieho systému[1], modifikovať súbor `.env` príslušnými tajomstvami, akými sú napríklad API kľúče, a spustiť aplikáciu príkazom `docker-compose up`. Týmto príkazom sa okrem spustenia jednotlivých komponentov vytvárajú aj kontajnery pre Prometheus a Grafanu, ktoré slúžia na monitoring komponenty Scrapper. Detailnejší postup opisujúci proces spustenia aplikácie je rozobraný v súbore `README.md`.

5.3.3 Matomo

Matomo je open-source nástroj pre získavanie analytických informácií, ktorý umožňuje užívateľom sledovať a analyzovať návštevnosť ich webovej stránky. V porovnaní s alternatívami je Matomo *on-premise* systém, čo ponúka lepšiu kontrolu nad zbieranými dátami. Ponúka funkcie ako zisťovanie návštevnosti, analýzu užívateľmi vykonávaných akcií alebo monitorovanie využívaných zariadení či prehliadačov pre prístup k danej webovej stránke. Všetky tieto dáta sú nesmierne dôležité pre zlepšovanie užívateľských skúseností, zvyšovanie návštevnosti či pre samotný vývoj webovej stránky.

[1] Aplikácia využíva Github, repozitár je dostupný na `https://github.com/opendatalabcz/tender-maps`

Kapitola 6
Možné rozšírenia

Táto kapitola sa venuje možným rozšíreniam alebo vylepšeniam webovej aplikácie. Tieto rozšírenia neboli implementované v rámci tejto práce, ale ich implementácia by mohla navýšiť počty užívateľov, zlepšiť ich užívateľské skúsenosti alebo vylepšiť systém samotný.

6.1 Vylepšenie paralelizmu v komponente Scrapper

Aktuálna implementácia paralelizmu urýchľuje ukladanie verejných zákaziek len na úrovni jednej verejnej zákazky, kedy sa asynchrónne dolujú dáta napríklad o všetkých ponukách alebo dodávateľoch. Takéto riešenie však nie je ľahko škálovateľné a lepším riešením by bolo asynchrónne spracovávanie jednotlivých verejných zákaziek. Jednou z možných implementácií tohto vylepšenia je návrhový vzor producent a spotrebiteľ. Úlohu producenta by predstavovali vlákna, ktoré by dolovali a geokódovali dáta. Každý producent by pri tom pracoval na inej verejnej zákazke. Spotrebiteľov by zastupovalo len jedno vlákno, ktoré by ako jediné pristupovalo k databáze a vkladalo tak vydolované dáta. Takýto prístup by efektívne zamedzil asynchrónnym prístupom k databáze. K tomuto riešeniu sa nepristúpilo v rámci tejto práce z dôvodu, že ak je portál *nen.nipez.cz* príliš preťažovaný, začne násobne spomaľovať odpovede, čo môže viesť ku ešte pomalšiemu dolovaniu. Toto správanie portálu by sa dalo obísť vytvorením klasteru strojov s rôznymi verejnými IP adresami, ktorý by vedel dolovať dáta súčasne.

6.2 Vylepšenie heat a hexagónovej mapy

Aktuálne implementácie heat a hexagónovej mapy zobrazujú absolutné kumulatívne hodnoty cien dodávaných zákaziek. Dá sa teda predpokladať, že veľké mestá budú sfarbené do tmavočervenej farby. Zaujímavým rozšírením týchto máp by bol prepínač, ktorý by transformoval dáta podľa určitých koeficientov. Príkladom takého koeficientu je HDP danej lokality či kraja.

6.3 Rozšírenie filtrácií

Zaujímavým rozšírením by mohlo byť pridanie dodatočných filtrácií. Napríklad filtrovanie podľa typu verejnej zákazky, podľa dátumu vzniku alebo dátumu podpísania kontraktu. Pre ikonovú mapu by zas mohla byť prínosná implementácia vyhľadávania jednotlivých spoločností na mape, čo by vylepšovalo užívateľské skúsenosti užívateľov zaujímajúcich sa o konkrétne spoločnosti.

Záver

Cieľom tejto práce bolo vytvorenie webovej aplikácie, ktorá vizualizuje dáta o verejných zákazkách v Českej republike pomocou máp.

Z výsledkov analýzy existujúcich riešení vyplýva, že existujúce portály neponúkajú pohľad na tieto dáta v rôznych súvislostiach a nevyužívajú mapu ako formu vizualizácie dát. V rámci analýzy som pokračoval identifikáciou funkčných a nefunkčných požiadaviek. Následne som navrhol vhodnú architektúru pre túto aplikáciu a pokračoval som návrhom relačnej databázy, ktorá by čo najlepšie modelovala doménu verejných zákaziek. Vytvoril som aj detailný návrh užívateľského rozhrania, ktorý dopomohol aj k ujasneniu samotných požiadaviek na systém.

Výsledná webová aplikácia vizualizuje dáta o verejných zákazkách pomocou sady interaktívnych máp a umožňuje filtrovanie podľa miesta plnenia a zadávateľa. Aplikácia scrappuje dáta o verejných zákazkách z portálu Národní elektronický nástroj a tento proces sa automaticky spúšťa každých 30 dní na zaistenie získania nových zákaziek. Aplikácia využíva proces geokódovania adries na obohatenie získaných dát o geografickú polohu. Pre prácu s cenami verejných zákaziek, aplikácia implementuje aj zámenu tých cien, ktoré portál nezverejňoval v CZK. Počas implementácie dolovania dát som narazil na pomerne častú chybovosť dát na danom portáli, o ktorej som informoval správcov portálu a niektoré závažnejšie chyby boli už opravené. Webový backend je implementovaný ako REST API, ktoré zabezpečuje aj filtráciu dát. Samotná vizualizácia získaných dát pozostáva z heat mapy, ikonovej mapy a hexagónovej mapy. Každá z týchto máp ponúka iný a jedinečný pohľad na dáta o verejných zákazkách v Českej republike. Pre účely získania detailnejšieho prehľadu o behu aplikácie je navrhnutý a implementovaný aj monitorovací systém, ktorý zbiera a vizualizuje rôzne typy metrík.

Vytvorená webová aplikácia je dostupná na doméne `https://tendermaps.opendatalab.cz/` a môže ju tak využívať širšia verejnosť, zaujímajúca sa o hospodárenie s verejnými financiami. Získané dáta touto prácou budú využité aj v ďalšej práci, ktorá sa venuje dátovej analýze a hľadaniu neúčelného či neefektívneho zaobchádzania s verejnými financiami.

Zdrojové súbory aplikácie sú verejne dostupné na portáli GitHub. Vývoj tak môže pokračovať v *open-source* modely, kde sa môžu do neho zapájať aj ďalší vývojári. Osobne plánujem vo vývoji pokračovať implementáciou niektorých z možných rozšírení. Konkrétne by som sa chcel zamerať na vylepšenie a urýchlenie dolovania dát.

Bibliografia

1. TRANSPARENCY INTERNATIONAL. *2022 Corruption Perceptions Index: Explore the results — transparency.org* [online]. 2022. [cit. 2023-11-08]. Dostupné z : `https://www.transparency.org/en/cpi/2022`.
2. ČESKÁ REPUBLIKA. *Zákon 134/2016 Sb. ze dne 19. dubna 2016 o zadávání veřejných zakázek*. 2016.
3. ČESKÁ REPUBLIKA. *Usnesení vlády České republiky č. 408/2018 o uložení povinnosti využívat Národní elektronický nástroj při zadávání veřejných zakázek*. 2018.
4. TENDER SYSTEMS S.R.O. *Tender arena nástroj pro zadávání veřejných zakázek* [online]. [cit. 2023-12-31]. Dostupné z : `https://www.tendersystems.cz/tenderarena.html`.
5. MINISTERSTVO PRO MÍSTNÍ ROZVOJ ČESKÉ REPUBLIKY. *Rozza rozcestník zákazek* [online]. [cit. 2023-12-31]. Dostupné z : `https://www.tendersystems.cz/tenderarena.html`.
6. SOMMERVILLE, Ian. *Softwarové inženýrství*. Ninth. Addison Wesley, 2013. ISBN 978-80-251-3826-7.
7. ADENOWO, Adetokunbo AA; ADENOWO, Basirat A. Software engineering methodologies: a review of the waterfall model and object-oriented approach. *International Journal of Scientific & Engineering Research*. 2013, roč. 4, č. 7, s. 427–434.
8. HOY, Zoe; XU, Mark. Agile Software Requirements Engineering Challenges-Solutions;A Conceptual Framework from Systematic Literature Review. *Information*. 2023, roč. 14, č. 6. ISSN 2078-2489. Dostupné tiež z: `https://www.mdpi.com/2078-2489/14/6/322`.
9. BJARNASON, Elizabeth; UNTERKALMSTEINER, Michael; BORG, Markus; ENGSTRÖM, Emelie. A multi-case study of agile requirements engineering and the use of test cases as requirements. *Information and Software Technology*. 2016, roč. 77, s. 61–79.
10. SOMMERVILLE, Ian; COCKBURN, Alan. *Requirements Engineering: The Good Practice*. Pearson Education, 2015. ISBN 978-0471974444.
11. MARTIN, Robert C. *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Pearson Education, 2017. ISBN 978-0134494166. s. 147–148, 29–38.
12. NEWMAN, Sam. Monolith to Microservices: Evolutionary Patterns to Trasform Your Monolith. In: O'Reilly Media, Inc., 2019, kap. The Monolith. ISBN 9781492047841.
13. ATLASSIAN. *Microservices vs. monolithic architecture* [online]. [cit. 2023-12-02]. Dostupné z : `https://www.atlassian.com/microservices/microservices-architecture/microservices-vs-monolith`.

14. TSAI, W.T.; BAI, Xiaoying; PAUL, R.; SHAO, Weiguang; AGARWAL, V. End-to-end integration testing design. In: *25th Annual International Computer Software and Applications Conference. COMPSAC 2001.* 2001, s. 166–171. Dostupné z DOI: 10.1109/CMPSAC.2001.960613.
15. AMAZON WEB SERVICES. *What is SOA (Service-Oriented Architecture)?* [online]. [cit. 2023-11-27]. Dostupné z : https://aws.amazon.com/what-is/service-oriented-architecture.
16. NEWMAN, Sam. *Building Microservices: Designing Fine-Grained Systems.* Second. O'Reilly Media, 2021. ISBN 978-1492034025.
17. NEWMAN, Sam. Building Microservices: Designing Fine-Grained Systems. In: Second. O'Reilly Media, 2021, kap. Advantages of Microservices. ISBN 978-1492034025.
18. ALTEXSOFT. *Comparing API Architectural Styles: SOAP vs REST vs GraphQL vs RPC* [online]. [cit. 2024-01-06]. Dostupné z : https://www.altexsoft.com/blog/soap-vs-rest-vs-graphql-vs-rpc/.
19. ZHAO, Bo. Web scraping. *Encyclopedia of big data.* 2017, roč. 1. Dostupné z DOI: 10.1007/978-3-319-32001-4_483-1.
20. APILAYER. *Foreign exchange rates and currency conversion JSON API* [online]. [cit. 2024-01-03]. Dostupné z : https://fixer.io/#pricing_plan.
21. GOSLING, James; JOY, Bill; JR., Guy L. Steele; BRACHA, Gilad. *The Java language specification.* Addison-Wesley Professional, 2000. ISBN 978-0201310085.
22. IBM. *What is Java Spring Boot?* [online]. [cit. 2023-12-10]. Dostupné z : https://www.ibm.com/topics/java-spring-boot.
23. STACK OVERFLOW. *2023 Developer Survey* [online]. [cit. 2024-01-05]. Dostupné z : https://survey.stackoverflow.co/2023/.
24. KOCAK, Burak. *JPA, Hibernate And Spring Data JPA* [online]. [cit. 2023-12-10]. Dostupné z : https://medium.com/@burakkocakeu/jpa-hibernate-and-spring-data-jpa-efa71feb82ac.
25. HEDLEY, Jonathan. *jsoup: Java HTML Parser* [online]. [cit. 2023-12-10]. Dostupné z : https://jsoup.org/.
26. WOOD, Lauren; LE HORS, Arnaud; APPARAO, Vidur; BYRNE, Steve; CHAMPION, Mike; ISAACS, Scott; JACOBS, Ian; NICOL, Gavin; ROBIE, Jonathan; SUTOR, Robert et al. Document object model (dom) level 1 specification. *W3C recommendation.* 1998, roč. 1.
27. KLEPPMANN, Martin. Designing Data-Intensive Applications. In: *The Big Ideas Behind Reliable, Scalable, and Maintainable Systems.* O'Reilly Media, 2017, kap. 2. ISBN 978-1449373320.
28. THE POSTGRESQL GLOBAL DEVELOPMENT GROUP. *About* [online]. [cit. 2023-12-09]. Dostupné z : https://www.postgresql.org/about/.
29. VMWARE. *Spring Web* [online]. [cit. 2023-12-10]. Dostupné z : https://docs.spring.io/spring-boot/docs/current/reference/html/web.html.
30. MOZILLA FOUNDATION. *JavaScript* [online]. [cit. 2023-12-10]. Dostupné z : https://developer.mozilla.org/en-US/docs/Web/JavaScript.
31. MOZILLA FOUNDATION. *Prototype-based programming* [online]. [cit. 2023-12-10]. Dostupné z : https://developer.mozilla.org/en-US/docs/Glossary/Prototype-based_programming.
32. GACKENHEIMER, Cory. *Introduction to React.* First. Apress, 2015. ISBN 978-1484212462.

33. YOU, Evan. *Getting Started* [online]. [cit. 2023-12-10]. Dostupné z : https://vitejs.dev/guide/.
34. BISIAKOWSKI, Dominik. *Why we use Vite instead of Create React App* [online]. [cit. 2024-01-05]. Dostupné z : https://makimo.com/blog/why-we-use-vite-instead-of-create-react-app/.
35. OPENJS FOUNDATION. *Introduction* [online]. [cit. 2023-12-10]. Dostupné z : https://deck.gl/docs.
36. MOZILLA FOUNDATION. *WebGL: 2D and 3D graphics for the web* [online]. [cit. 2023-12-10]. Dostupné z : https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API.
37. MUI. *Material UI - Overview* [online]. [cit. 2023-12-17]. Dostupné z : https://mui.com/material-ui/getting-started/.
38. HAMIDLI, Nasrullah. Introduction to UI/UX Design: Key Concepts and Principles. 2023.
39. FIGMA. *The modern interface design tool* [online]. [cit. 2023-12-13]. Dostupné z : https://www.figma.com/ui-design-tool/.
40. MALONE, Emily. *THE PROS AND CONS OF WIREFRAMING IN A WEBSITE REDESIGN* [online]. [cit. 2023-12-13]. Dostupné z : https://www.growthdrivendesign.com/blog/the-pros-and-cons-of-wireframing-in-a-website-redesign.
41. KIWY, Frank. *Exploring Joshua Bloch's Builder design pattern in Java* [online]. [cit. 2023-12-15]. Dostupné z : https://blogs.oracle.com/javamagazine/post/exploring-joshua-blochs-builder-design-pattern-in-java.
42. DIGITALOCEAN. *DAO Design Pattern* [online]. [cit. 2023-12-22]. Dostupné z : https://www.digitalocean.com/community/tutorials/dao-design-pattern.
43. VMWARE. *Scheduling Tasks* [online]. [cit. 2023-12-21]. Dostupné z : https://spring.io/guides/gs/scheduling-tasks/.
44. BROOKER, Marc. *Exponential Backoff And Jitter* [online]. [cit. 2023-12-31]. Dostupné z : https://aws.amazon.com/blogs/architecture/exponential-backoff-and-jitter/.
45. WAGNER, Jeremy; POLLARD, Barry. *Time to First Byte (TTFB)* [online]. [cit. 2023-12-30]. Dostupné z : https://web.dev/articles/ttfb.
46. GOOGLE. *Report from Jan 6, 2024, 8:34:11 PM* [online]. [cit. 2024-01-06]. Dostupné z : https://pagespeed.web.dev/analysis/https-tenderarena-cz-dodavatel/9d98xo0un6?form_factor=desktop.
47. GOOGLE. *Report from Jan 8, 2024, 11:25:03 PM* [online]. [cit. 2024-01-06]. Dostupné z : https://pagespeed.web.dev/analysis/https-nen-nipez-cz-en/3psbueqzvj?form_factor=desktop.
48. VMWARE. *Production-ready Features* [online]. [cit. 2024-01-09]. Dostupné z : https://docs.spring.io/spring-boot/docs/current/reference/html/actuator.html.
49. BROADCOM INC. *Vendor-neutral application observability facade* [online]. [cit. 2024-01-09]. Dostupné z : https://micrometer.io/.
50. PROMETHEUS. *What is Prometheus?* [online]. [cit. 2024-01-09]. Dostupné z : https://prometheus.io/docs/introduction/overview/.
51. O'REILLY MEDIA, INC. *Software Architecture Patterns* [online]. [cit. 2023-12-15]. Dostupné z : https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch01.html.
52. FOWLER, Martin. *Data Transfer Object* [online]. [cit. 2023-12-15]. Dostupné z : https://martinfowler.com/eaaCatalog/dataTransferObject.html.

53. VMWARE. *Specifications* [online]. [cit. 2023-12-15]. Dostupné z : https://docs.spring.io/spring-data/jpa/reference/jpa/specifications.html.

54. META OPEN SOURCE. *Quick Start* [online]. [cit. 2023-12-17]. Dostupné z : https://react.dev/learn.

55. MOZILLA FOUNDATION. *Promise* [online]. [cit. 2023-12-23]. Dostupné z : https://developer.mozilla.org/enUS/docs/Web/JavaScript/Reference/Global_Objects/Promise.

56. META OPEN SOURCE. *useEffect* [online]. [cit. 2024-01-09]. Dostupné z : https://react.dev/reference/react/useEffect.

57. AMAZON WEB SERVICES. *What Is an In-Memory Database?* [online]. [cit. 2023-12-27]. Dostupné z : https://aws.amazon.com/nosql/in-memory/.

58. OLAN, Michael. Unit testing: test early, test often. *Journal of Computing Sciences in Colleges*. 2003, roč. 19, č. 2, s. 319–328.

59. LEUNG, Hareton KN; WHITE, Lee. A study of integration testing and software regression at the integration level. In: *Proceedings. Conference on Software Maintenance 1990*. IEEE, 1990, s. 290–301.

60. VMWARE. *MockMvc* [online]. [cit. 2023-12-27]. Dostupné z : https://docs.spring.io/springframework/reference/testing/spring-mvc-test-framework.html.

61. AMAZON WEB SERVICES. *What's the Difference Between Docker Images and Containers?* [online]. [cit. 2023-12-29]. Dostupné z : https://aws.amazon.com/compare/the-difference-between-docker-images-and-containers/.

62. DOCKER. *Docker Compose overview* [online]. [cit. 2024-01-06]. Dostupné z : https://docs.docker.com/compose/.

Obsah priloženého média

```
parser/ ..................................................... komponenta Scrapper
    Dockerfile ..................................... konfigurácia kontajnerizácie
    README.md ..................................... inštrukcie na spustenie Scrappru
    build.gradle ................................................ Gradle konfigurácia
    gradle/ ........................................... adresár s Gradle skriptami
    gradlew ..................................................... Gradle wrapper skript
    gradlew.bat ............................... Gradle wrapper skript pre Windows
    grafanaDashboard.json ..................................... šablóna pre Grafanu
    prometheus.yml ....................................... konfigurácia Prometheus
    settings.gradle ............................................. nastavenia Gradle
    src/ ........................................... adresár so zdrojovými súbormi
procurements_api/ ..................................... komponenta webový backend
    Dockerfile ..................................... konfigurácia kontajnerizácie
    build.gradle ................................................ Gradle konfigurácia
    gradle/ ........................................... adresár s Gradle skriptami
    gradlew ..................................................... Gradle wrapper skript
    gradlew.bat ............................... Gradle wrapper skript pre Windows
    settings.gradle ............................................. nastavenia Gradle
    src/ ........................................... adresár so zdrojovými súbormi
tender-map-frontend/ ................................ komponenta webový frontend
    Dockerfile ..................................... konfigurácia kontajnerizácie
    index.html ................................................ vstupný HTML súbor
    package.json ..................................... súbor definujúci závislosti
    public/ ........................................... adresár so statickými zdrojmi
    src/ ........................................... adresár so zdrojovými súbormi
    vite.config.js ............................................ konfigurácia Vite
LICENSE ..................................................................... licencia
README.md ..................................... inštrukcie na spustenie aplikácie
docker-compose.yml ................................ konfigurácia docker-compose
```