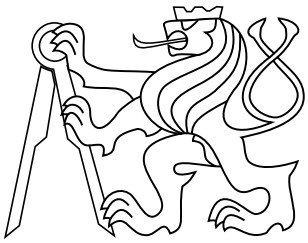
bakalářská práce

Návrh a implementace aplikace REST API v jazyce Java pro práci s databází PostgreSQL

Serhij Horčička



Květen 2017

Ing. Pavel Lafata, Ph.D.

České vysoké učení technické v Praze
Fakulta elektrotechnická, Katedra telekomunikační techniky

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Horčička** Jméno: **Serhij** Osobní číslo: **426063**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra telekomunikační techniky**

Studijní program: **Komunikace, multimédia a elektronika**

Studijní obor: **Síťové a informační technologie**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Návrh a implementace aplikace REST API v jazyce Java pro práci s databází PostgreSQL

Název bakalářské práce anglicky:

Designing and implementation of REST API application in Java for procesing of PostgreSQL databases

Pokyny pro vypracování:

Cílem práce je navrhnout a implementovat REST API, které umožní aplikacím třetích stran zpracovávat data poskytovaná různými ministerstvy prostřednictvím databáze PostgreSQL. Dále bude vytvořen webový portál, který popisuje API a také nabízí základní jednoduché vyhledávání subjektů, faktur, smluv a objednávek. Výsledky poskytuje formou jednoduchého reportu. API bude podporovat throttling, tj. ochranu proti nadměrnému zatěžování a také blacklisting dle IP adres. Součástí řešení bude také scheduler, který zajistí pravidelné stahování dat z webů veřejné správy s využitím existujícího modulu. Tato informace bude uvedena na informační stránce portálu. Řešení bude implementováno v jazyce JAVA.

Seznam doporučené literatury:

[1] Douglas, K.: PostgreSQL (2nd Edition), Sams Publishing; 2. edition, 2005. ISBN: 978-0-67232-756-8.
[2] Gilmore, W. J.: Beginning PHP and PostgreSQL 8: From Novice to Professional, Apress; 1. edition, 2006. ISBN: 978-1-59059-547-3.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Pavel Lafata Ph.D., katedra telekomunikační techniky FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **19.10.2016** Termín odevzdání bakalářské práce: **26.05.2017**

Platnost zadání bakalářské práce: **30.09.2018**

Podpis vedoucí(ho) práce	Podpis vedoucí(ho) ústavu/katedry	Podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

Datum převzetí zadání	Podpis studenta

Poděkování

Děkuji panu Ing. Pavlu Lafatovi, Ph.D. za trpělivost a toleranci, kterou se mnou měl. Také bych chtěl poděkovat kolegům z pracovního prostředí Václavu Legatovi a Jakubovi Jestřábovi.

Prohlášení

Prohlašuji, že jsem zadanou bakalářskou práci zpracoval sám s přispěním vedoucího práce a konzultanta a používal jsem pouze literaturu v práci uvedenou. Dále prohlašuji, že nemám námitek proti půjčování nebo zveřejňování mé bakalářské práce nebo její části se souhlasem katedry.

V Praze dne 26. 5. 2017

Podpis bakalanta

.....

Abstrakt

Cílem této práce bylo navrhnout rozhraní webové aplikace pro konzumování dat poskytovaných různými státními orgány. Výsledná data se zobrazují na webovém portálu.

Klíčová slova

Rest; API; JAVA; HTTP; Internet of Things;

Abstrakt

The goal of this bachelor work was to create interface of web application to consume data provided by state organs. Resulting data are showed on web portal.

Keywords

Rest; API; JAVA; HTTP; Internet of Things;

Obsah

Zkratky

HTTP	Internetový protokol
GUI	Grafické uživatelské rozhraní
JSON	Datový formát
POJO	Jednoduchá třída
Rest	Architektura rozhraní
JDK	Java Vývojový Kit
IoT	Internet věcí
SQL	Standardizovaný strukturovaný dotazovací jazyk
XML	Obecný značkovací jazyk
URI	Jednotný identifikátor zdroje
Cache	Mezipaměť
HATEOAS	Hypermedia jako aplikační stav

Seznam obrázků

1 Úvod

V poslední době se pro mnoho velkých i malých společností stává trendem sdílet svoje data a služby veřejně s možností jejich dalšího použití třetími stranami. Sdílení dat a vytváření rozhraní pro sdílení těchto dat je důležité pro budoucí vývoj internetu věcí, který je spíše znám pod anglickým názvem Internet of Things. Vzhledem k tomu, že veškerá technologie směřuje k „Internet of Things“, pojmu používanému pro model, kdy veškeré objekty, ať jsou nějak technologicky vyspělé („inteligentnější“), či nikoliv („elektronický budík“), vzájemně komunikují a sdílejí nebo stahují data z jiných objektů. Proto je třeba nějakého kanálu, přes který komunikace bude probíhat. Jedním ze způsobů, jak toho lze dosáhnout, je API („Application Programming Interface“). Termín označovaný v informatice jako rozhraní pro programování aplikací. Tento přístup lze aplikovat na veškeré aplikace a objekty. Jde o model, kdy dochází k otevřenému distribuování dat z jedné strany a konzumování těchto dat paralelně mnohými objekty. Ve své podstatě jde o rozhraní, které má nastavené pevné procedury a při každém volání vrací požadovaná data.
Stále více zařízení se zapojuje do tohoto modelu, prostředkem pro komunikaci mezi všemi zařízeními v této síti se může stát Rest. Rest slouží jako architektura pro komunikaci, ve které může být klientem uživatel nebo stroj .
Pro splnění všech požadavků, uvedených v zadání, bude aplikace obsahovat několik částí. Teoretická část, ve které budou rozebrány technologie a stavba celé aplikace Rest, API část, ve které bude implementována webová aplikace poskytující pouze data, a část Webový portál, kde bude implementována webová aplikace, která konzumuje data z Rest API a zobrazuje je koncovému uživateli služby.

2 Motivace

Myšlenkou, která stála za vytvořením daného projektu, je sdílení dat poskytovaných státními organizacemi pro veřejnost. Data poskytovaná jednotlivými organizacemi se budou načítat do jedné databáze, odkud si je přes API budou moci stahovat další portály a používat je na svých stránkách. V brzké době bude stále více státních orgánů zveřejňovat svoje data.
Bohužel pro většinu běžných lidí budou tato data v nečitelné podobě nebo je nenaleznou ve složitě organizovaných stránkách ministerstev. Proto jsem se rozhodl realizovat tento projekt, který lze poměrně snadno rozšiřovat o další moduly a zdroje, přidávat další procedury pro volání či zpřehledňovat konzumovatelný obsah. Proto je výsledná aplikace snadno konfigurovatelná s možností dalšího vývoje a doplnění v budoucnu.

3 Cíl práce

Cílem práce je navrhnout a implementovat Rest API, které umožní aplikacím třetích stran zpracovávat data poskytovaná různými ministerstvy prostřednictvím databáze PostgreSQL.
Dále bude vytvořen webový portál, který popisuje API a také nabízí základní jednoduché vyhledávání subjektů, faktur, smluv a objednávek. Výsledky poskytuje formou jednoduchého reportu.
API bude podporovat throttling, tj. ochranu proti nadměrnému zatěžování, a také blacklisting dle IP adres. Součástí řešení bude také scheduler, který zajistí pravidelné stahování dat z webů veřejné správy s využitím existujícího modulu.
Tato informace bude uvedena na informační stránce portálu. Řešení bude implementováno v jazyce JAVA.

4 Teoretický rozbor

V části teoretického rozboru jsou popsány veškeré použité technologie, které aplikace používá. Dále jsou zde popsány způsoby, jakými dané technologie fungují a jejich použití obecně.

4.1 Rest

Termín Rest („Representational State Transfer“) byl poprvé použit v disertační práci „Architectural Styles and the Design of Network-based Software Architectures“[1], kde Roy Fielding v roce 2000 popsal, jakým způsobem může být navržena architektura rozhraní v distribuovaném systému. Fielding píše, že Rest sám o sobě není architektura, ale „styl“ sady omezení, který se pokouší o zmenšení objemu komunikace, ale přitom zachovává konfigurovatelnost a nezávislost každé komponenty sítě. Distribuovaná architektura v tomto smyslu znamená, že různé části programu běží na různých strojích a pro svou komunikaci využívají síť.

Rest je na rozdíl od XML-RPC či SOAP orientován datově, nikoli procedurálně. Webové služby definují vzdálené procedury a protokol pro jejich volání. Rest určuje, jak se přistupuje k datům. Rozhraní Rest slouží jako jednotný a snadný přístup ke zdrojům. V tomto případě můžou být zdrojem jak data, tak například stavy aplikace. Všechny zdroje v této architektuře mají svůj jedinečný identifikátor URI.

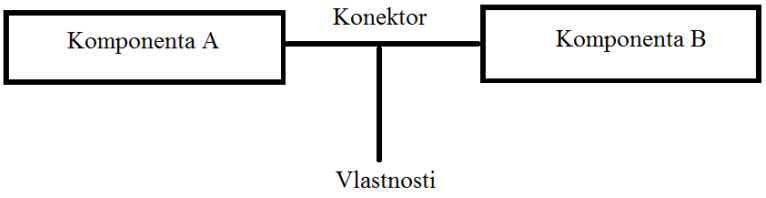
V této části bude uvedena teorie o Rest přístupu, poté bude detailně rozebrán sám pojem Rest a to, jak k němu přistupuje sám Roy Fielding.

4.1.1 Softwarová architektura

„Software architecture“[1] je pojem, který definoval Fielding ve své disertační práci z roku 2000. Jedná se o strukturu softwarové architektury, kde se nacházejí komponenty, které jsou vázány navzájem „connectorem“ mající určité vlastnosti.

Komponenty jsou výpočetní součásti, které tvoří architekturu společně s popisem jejich interakcí. Tyto výpočtové komponenty jsou abstraktními jednotkami softwaru instrukcí a vnitřních stavů, které poskytují transformaci dat přes jeho rozhraní. Pokud si představíme architekturu jako na obrázku 1, tak Uzly představují komponenty a oblouky představují vztahy mezi Uzly (konektory). Některé příklady komponentů v architektuře Implementace mohou být klienty, servery nebo databázemi.

Fielding uvádí novou definici pro konektory. Konektory jsou abstraktním mechanismem, který zprostředkovává komunikaci, koordinaci nebo spolupráci mezi složkami. Každá implementace architektury má podrobnosti o svých komponentech a konektorech skrytých na architekturní úrovni. Příklady konektorů mohou být jakékoli komunikační protokoly, jako třeba protokol klient-server.
Datové elementy jsou definované jako soubor informací přenesených z komponentů nebo přijatých komponentou přes konektor.
Vlastnosti mohou být definovány jako omezení, které poskytují podmínky pro komponenty a vztahy mezi komponenty.



Obrázek 1 Struktura Softwarové Architektury

4.1.2 Softwarové architektonické vzory (styly)

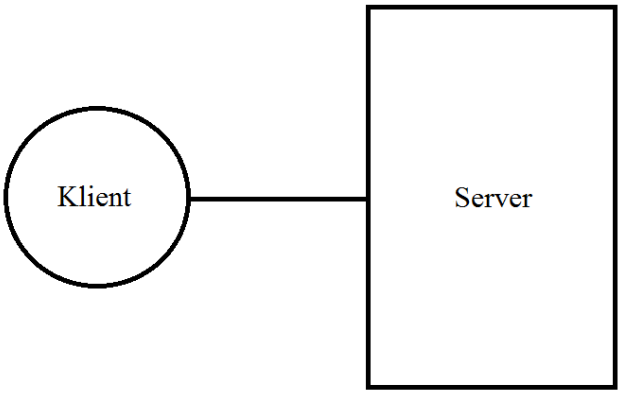
Softwarový styl architektury lze popsat jako sadu omezení, která se uplatňuje na to, jak komponenty vzájemně komunikují.[1]

4.1.3 Rest přístup

Rest je architekturový styl distribuovaných systémů hypermedií. Je to hybridní styl odvozený z různých síťových architekturových stylů a kombinovaný s dalšími omezeními, aby bylo možné definovat jednotné rozhraní pro komunikaci. Zaměřuje se na omezení, která musí být aplikovaná na sémantiku konektorů, zatímco ostatní styly, jako například servisově orientované, se zaměřují na omezení sémantiky komponentů.
Pro odvození Restu začal Roy Fielding postupně identifikovat potřeby systému bez žádných omezení a následně po dokončení zkoumaní postupně aplikoval omezení na jeho elementy. Touto cestou aplikovaní omezení na systém lze dosáhnout detailního pochopení celého kontextu daného systému. Následně se přidávají omezení pro zajištění spolupráce mezi komponenty.

4.2 Rest architektonické styly

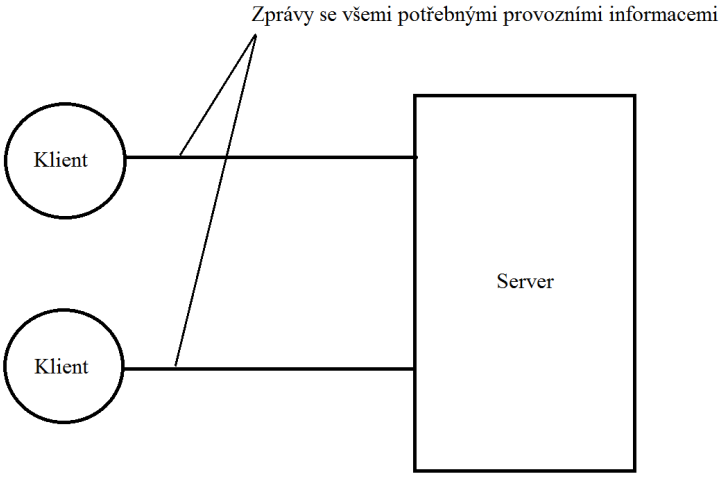
4.2.1 Klient-Server



Obrázek 2 Klient-Server

Pravděpodobně jeden z nejvíce rozšířených stylů architektury je Klient-Server. Tato architektura vynucuje oddělení úkolů klienta od úkolů serveru. Tím se zajišťuje vytvoření základní distribuované architektury, kde probíhá nezávislý vývoj logiky odděleně. Tento styl architektury zároveň vyžaduje, aby server nabízel více funkcí a byl schopný je zpracovat. Klient poté vyvolává konkrétní funkce serveru a pokud byl dotaz zaslaný klientem validní, server zpracuje dotaz a odpoví. Výjimky při zaslání dotazu jsou zkoumány a pokud nějaké nastanou, dotaz se vrátí s chybou klientovi.

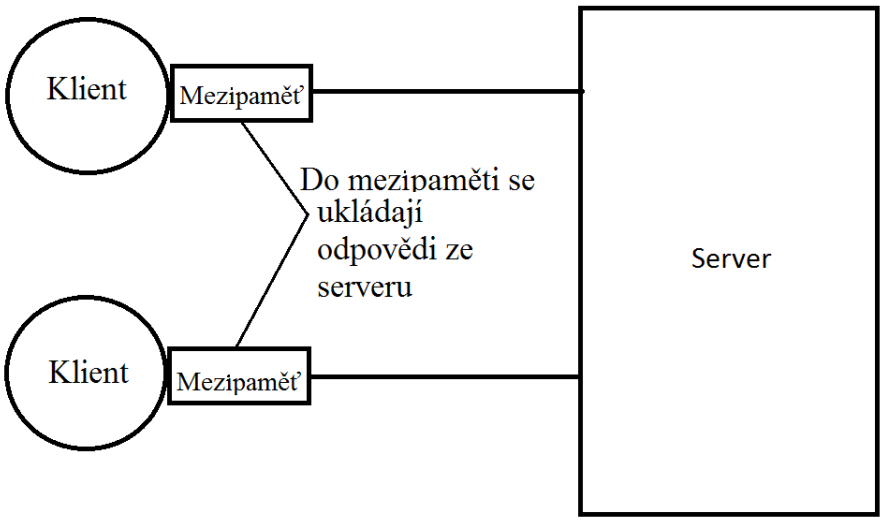
4.2.2 Bezstavový



Obrázek 3 Klient-Bezstavový-Server

Je založený na požadavcích klienta, kdy požadavek obsahuje veškerá potřebná data pro server. Server neukládá žádná data mezi požadavky. Klient ukládá informaci o stavu požadavku u sebe. Tento model interakce má své výhody i nevýhody. Mezi výhody lze považovat jednoduchost, rozšiřitelnost, spolehlivost.
Transparentnost[2] tohoto stylu se zlepšila tím, že na serverové straně není třeba kontrolovat žádosti z důvodu monitoringu. Škálování je zajištěno tím, že není třeba ukládat veškeré stavy mezi žádostmi. Díky tomu může server rychleji uvolňovat zdroje.

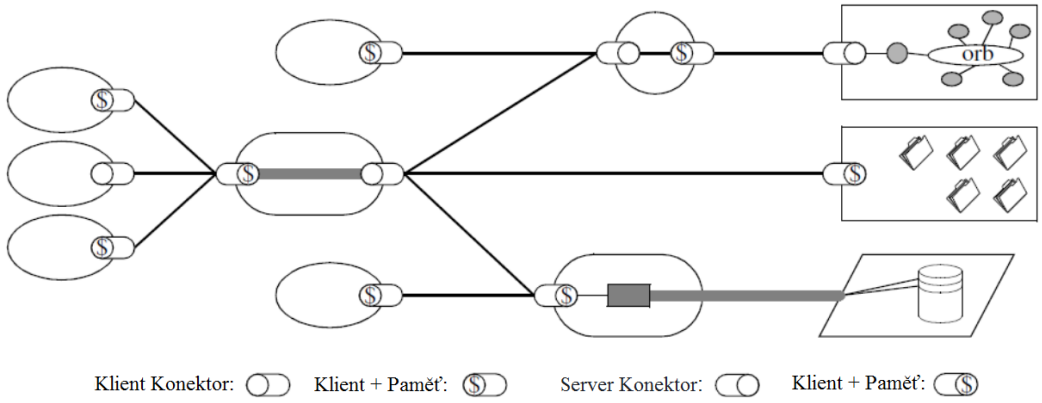
4.2.3 Mezipaměť



Obrázek 4 Klient-Mezipaměť-Bezstavový-Server

Kvůli zmenšení objemu dotazů ze strany klienta na server, které vznikají při Stateless zprávách, se zavádí v Rest stylu architektury Cache[3] (mezipaměť). Cache se tvoří na straně klienta.
Server musí explicitně označovat odpovědi, které budou uložené do paměti (Cachable). Dělá se to z toho důvodu, aby všechny komponenty v této komunikaci věděly, které zprávy lze a které nelze ukládat do paměti. Pokud by mezi klientem byla ještě další vrstva v podobě middleware, potom by i ona mohla automaticky ukládat data do paměti pro další použití.
Následně všechny dotazy vytvořené klientem prochází komponentou pro ukládání do Cache, kde se buď použijí znovu z Cache, nebo se alespoň částečně obnoví ze serveru a klient vždy obdrží ta nejnovější data. Tímto se sníží počet interakcí do sítě a zmenší se latence požadavku, což pro uživatele přináší výkonový rozdíl. Problém ale v tomto modelu nastává, když se data už uložená v paměti liší od těch, které by server při normálním zpracování požadavku vrátil.

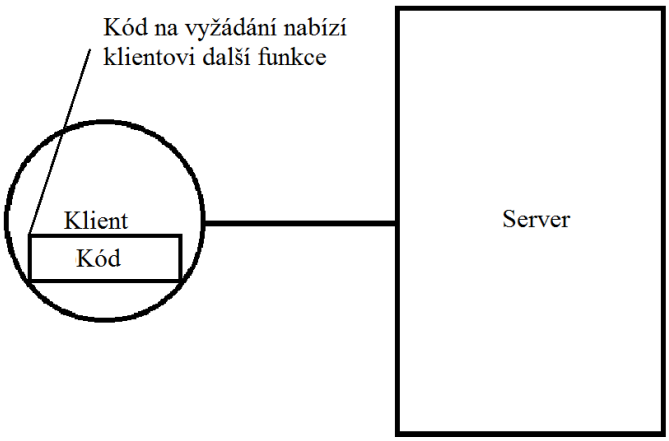
4.2.4 Vrstvový



Obrázek 5 Klient-Mezipaměť-Bezstavový-Vrstvový-Server

Vrstvový model architektury[1] umožňuje hierarchicky skládat komponenty. V tomto modelu jsou komponenty omezeny svojí komunikací pouze na elementy v jejich vrstvě.. Tím se dá dosáhnout zvýšení zabezpečení. Tyto komponenty v mezivrstvě musejí být zařazeny transparentně.
To znamená tak, aby při komunikaci mezi serverem a klientem nevznikala nekonzistence. Pro správnou funkcionalitu tohoto modelu musí platit, že interakce mezi danou službou a klientem je vždy konzistentní bez ohledu na to, zda klient komunikuje se službou umístěnou v mezivrstvě nebo se službou, která představuje konečného příjemce zprávy (server). To samé platí pro opačnou stranu, kdy server nemusí vědět, zda nyní komunikuje s konečným klientem nebo je to další server, který provádí další komunikaci.
Tato forma skrývání informací[3] zjednodušuje distribuovanou architekturu a umožňuje komponentům, které se v ní nacházejí, vyvíjet se samostatně a v úplné nezávislosti na jiných komponentách.

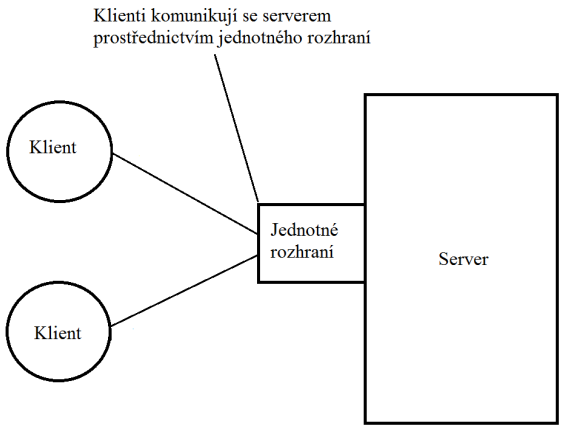
4.2.5 Kód na vyžádání (volitelný)



Obrázek 6 Klient-KódNaVyžádání-Server

V daném modelu jde o přenesení zátěže ze serveru na klienta. Server může posílat klientovi spustitelné části kódů, kterými dosáhne rozšíření funkcionality u klienta. Jako příklad takových spustitelných kódů lze uvést JAVA applets nebo Javascript, které lze spustit na straně klienta. Tento přístup lze ospravedlnit v případě, že službu, kterou klient po serveru vyžaduje, klient sám dokáže vykonat efektivněji než server.
Tento model není nutné aplikovat v architektuře, jelikož to má své výhody a nevýhody v závislosti na případu, ve kterém se tato funkčnost implementuje, tudíž ne vždy je možné tento model implementovat. Mezi nevýhody použití tohoto modelu lze přidat to, že kód na vyžádání způsobuje nepřehlednost samotným kódem, který je pak těžké interpretovat pro klienta.

4.2.6 Jednotné rozhraní



Obrázek 7 Klient-JednotnéRozhraní-Server

Rest architekturní styl říká, že všechny komponenty v této architektuře musejí sdílet právě jedno rozhraní. Jednotné rozhraní je nezbytnou součástí při návrhu jakékoliv Rest služby. Takové rozhraní zjednodušuje a rozděluje architekturu, což umožňuje každé komponentě v této architektuře nezávislý vývoj. Právě jednotné rozhraní a nezávislý vývoj jednotlivých komponent odlišuje Rest architekturní styl[2] od ostatních stylů. Při aplikaci této architektury se vše zjednodušuje a zpřehledňuje. Má to nicméně i své nevýhody. Jednotné rozhraní je standardizované, tudíž pro některé klienty, kteří by mohli mít specifická rozhraní a pracovat tak efektivněji, bude komunikace pomalejší, jelikož musí zjistit požadované informace více dotazy. Pro úspěšnou implementaci této architektury je potřeba dodržet následující pravidla:

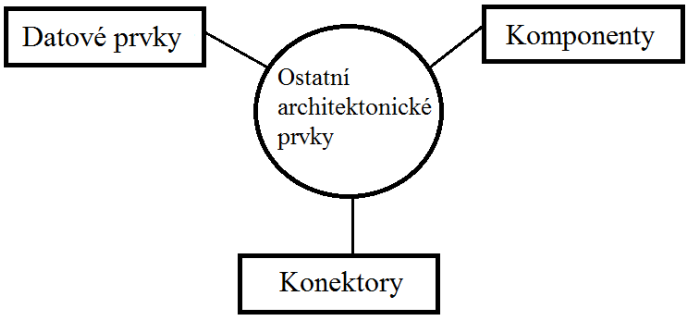
• Jednoznačná identifikace zdrojů
Jednotlivé zdroje jsou identifikované v požadavcich ve webových Rest aplikacích, například pomocí URI. Serverová strana se musí koncepčně oddělit od toho, co posílá klientům. Když server posílá data klientům, pak data klient obdrží v podobě JSON, XML, HTML/TEXT, nikoliv v interní reprezentaci serveru.

• Manipulace zdrojů prostřednictvím zastoupení
Pokud klient disponuje reprezentací zdrojů včetně metadat, může sám mazat nebo měnit zdroj.

• Zprávy obsahující svůj popis
Každá zpráva obsahuje dostatek informací, které popisuji, jak zprávu zpracovávat.

• HATEOAS
Hypermedia as the engine of application state[1]. Po přístupu k počátečnímu identifikátoru URI by měl Rest poskytnout klientům veškeré informace bez ohledu na to, jaké akce a zdroje jsou dostupné. Klient nemusí zkoumat strukturu celého zdroje. Odkazy a akce jsou poskytnuty dynamicky podle toho, kde se klient aktuálně nachází.

4.3 Rest elementy



Obrázek 8 Rest-Architektonické-Elementy

Architekturní styl Rest definuje architekturu klient-server, kde se pro komunikaci používá bezstavový protokol, obvykle HTTP. V Rest architektuře si klienti[1] a servery vyměňují reprezentaci zdrojů přes standardizované rozhraní a protokol.
Každá softwarová architektura se skládá z komponent, konektoru a dat. Rest se na rozdíl od jiných architekturových stylů zaměřuje na to, jak spolupracují komponenty s jinými komponenty a jak komponenty fungují. Naopak se nestará o to, jak jsou jednotlivé komponenty implementovány.

4.3.1 Rest komponenty

V Rest architektuře se různý software, který spolupracuje s jiným softwarem, jmenuje komponenta. Komponenty slouží k navázání spojení. Rest služby mohou používat mnoho různých klientů a aplikací, které běží na různých platformách a přístrojích, jako jsou telefony nebo prohlížeče.
Tyto dotazy od klienta jsou posílaný například přes proxy server, který jako první typ komponenty zajišťuje bezpečnost a základní Cache. Za proxy serverem může následovat celá řada dalších typů komponent, které například ověřují uživatele, starají se o rozložení zatížení na server, ukládaní dotazu do Cache či přihlášení uživatele. Zde je dobře vidět řetězení komponent v této architektuře.

4.3.2 Rest konektory

Konektory představují abstraktní rozhraní pro komunikaci komponent. Zvyšují jednoduchost tím, že poskytují čisté oddělení komponent. Skrývají to, jakým způsobem bylo něco v architektuře implementováno a jakým způsobem probíhá samotná komunikace. Rest zapouzdřuje různé činnosti přístupu a přenosu reprezentací do různých typů konektorů.
Za typy konektoru lze považovat například server na kterém běží Apache, libwww nebo API. Server poslouchá dotazy a odesílá odpovědi. DNS server, na kterém běží překlad adres z URI na adresu, která je srozumitelná pro síť, přidává latence do požadavku. Klient a server jsou primárními konektory v této architektuře. Rozdíl mezi nimi je v tom, že klient požadavky vytváří a server neustále poslouchá a odpovídá na požadavky poskytnutím požadovaných služeb.

4.3.3 Rest data elementy

Klíčovým aspektem systému Rest je stav datových elementů[1], jehož součásti komunikují přenosem reprezentací aktuálního nebo požadovaného stavu datových elementů. Rest identifikuje šest datových elementů: zdroj, identifikátor zdroje, zdroje metadat, reprezentace, metadata reprezentace a řídicí data.

- Zdroj

Jakékoli informace, které lze pojmenovat, jsou zdrojem. Zdrojem je koncepční mapování souboru entit, nikoliv samotné entity. Takové mapování se může časem měnit. Obecně platí, že RESTful zdrojem je vše, co je adresovatelné přes web.

- Identifikátor zdroje

Každý zdroj musí mít název, který ho jednoznačně identifikuje. Pod protokolem HTTP se tyto nazývají URI. Uniformní identifikátor zdrojů (URI) v systému RESTful je hypertextový odkaz na prostředek. Jedná se o jediný prostředek, jakým si klienti a servery vyměňují reprezentace zdrojů. Vztah mezi identifikátory URI a zdroji je mnoho k jednomu. Zdroj může mít více identifikátorů URI, které poskytují různé informace o umístění zdroje.

- Identifikátor zdroje metadat

Metadata zdroje slouží k jeho popisu. Poskytují doplňující informaci o zdroji, jako je například informace o umístění.

- Identifikátor reprezentace

Je to něco, co se posílá navzájem mezi klienty a servery. Takže nikdy nepřenášíme ani nedostáváme zdroje, pouze jejich reprezentace. Reprezentace zachycuje aktuální nebo zamýšlený stav zdroje. Konkrétní zdroj může mít více reprezentací.

- Metadata reprezentace

Popisují reprezentaci dat.

- Řídící data

Definuje účel zprávy mezi komponenty. Například zda je zpráva požadavek nebo odpověď.

4.3.4 HATEOAS

Hypermedia je aplikační stav, pojem, pod kterým je definice říkající, že pro navigaci a hledání cesty musí být použít Hypertext. Tato vlastnost odlišuje Rest architekturní styl od ostatních. Princip spočívá v komunikaci klienta se serverem skrze dynamicky poskytované hypermedium.
Rest používá pro komunikaci bezstavové zprávy, ale zároveň je v definici napsáno Representational state transfer, tento problém řeší HATEOAS. Jako příklad je zde možné použít Rest API. Klient potřebuje jen první inicializační URI adresu k reprezentaci zdroje. Počínaje tímto okamžikem jsou klient nebo stavy aplikace řízené dynamicky podle toho, co klient nebo aplikace volí z poskytnutých možností od serveru.

Lze tedy říci, že každá reprezentace zdroje musí obsahovat dynamicky se měnící odkazy, které reprezentují další stav, kam lze pokračovat. V každé obdržené reprezentaci zdrojů ze serveru musí být odkazy, které odkazují na následující dotaz.

Klient následně pomocí odkazu může manipulovat s reprezentací zdrojů či získávat jiná k němu vztahující se data. Tímto je právě dosaženo RESTful komunikací, která je řízena pouze pomocí hypermedia a jím poskytovanými informacemi o dostupných akcích nad reprezentací zdrojů.

4.4 Rest API

4.4.1 API

API - application programming interface. API je rozhraní, které se skládá z procedur a funkcí, které plní jeden nebo více účelů z důvodu použití jiným softwarem/programem. Tím lze dosáhnout implementování procedur a funkcí, které lze poskytnout jinému software/programu bez nutnosti implementovat stejné funkce a procedury v jiném softwaru/programu.
Roy Fielding byl jedním ze spoluautorů protokolu HTTP. Proto je Rest API knihovna založena na HTTP standardu (Hypertext Transfer Protocol). Rest API plní vždy funkci přidání softwaru a funkcionality k již stávajícímu systému/programu.

4.4.2 Rest přes HTTP protokol

HTTP – Hypertext Transfer Protocol. Je bezstavový aplikační protokol, pomocí kterého se posílají požadavky a odpovědi. HTTP používá zprávy, které v sobě nesou informaci o samotné zprávě, tento způsob zajišťuje flexibilní interakci s webovými hypertextovými systémy.
V RESTfull systémech klienti a servery vyjednávají pomocí reprezentace zdrojů přes HTTP. RESTful API umožňuje vyvinout jakoukoliv webovou aplikaci, která má všechny HTTP operace. Tyto systémy používají čtyři základní funkce persistentního uložiště, a to CRUD[1] (Vytvořit, Číst, Aktualizovat, Odstranit). Tyto operace do HTTP se promítají v následujících metodách:

• GET
GET dotaz se používá pouze pro získání reprezentace zdroje, nikoliv k jeho modifikaci. GET dotazy se považují za bezpečné, jelikož nemění stav zdroje. GET API by měly být vždy idempotentní, což znamená, že vytváření více identických požadavků musí mít vždy stejný výsledek, dokud jiný API dotaz typu POST nebo PUT nezmění stav zdroje na serverové straně. Pokud požadavek URI odkazuje na validní adresu pro produkci dat, jsou klientu vracená data v podobě entity, nikoliv ve zdrojovém kódu.

• POST
POST se používá pro vytvoření zdrojů na straně serveru. Není to bezpečná metoda, jelikož mění stav zdroje.

- PUT

PUT se používá pro aktualizaci stávajícího zdroje, může se ale používat i pro vytvoření, když daný zdroj neexistuje. Proto tato metoda není bezpečná, jelikož může měnit stav zdrojů.

- DELETE

Daná metoda smaže zdroj ze serveru, pokud bude poskytnuto správné ID zdroje.
Bezpečné metody nikdy nemění stav zdrojů, jenom GET je bezpečná metoda a nejčastěji se používá, pokud je účelem API sdílet data bez možností modifikací. Idempotentní metody vždy dosahují stejného výsledku bez ohledu na to, kolikrát jsou volané. Pokud jsou výše uvedené metody CRUD použity správně, potom jsou také idempotentní.

5 Implemetace

V první části této práce byl rozebrán architekturní styl Restu. Tato část se zabývá tím, jak implementovat Rest API s použitím Spring Boot a JAVA. Dále budou vysvětleny použité nástroje a nakonec bude rozebraná samotná implementace.

5.1 Rest API

V dané části bude podrobněji popsána realizace Rest API, použité technologie a řešení zvolené na základě analýzy.
Vzhledem k tomu, že má tato práce za cíl data pouze poskytnout dalším stranám bez možnosti jejich modifikace, bude API podporovat pouze „GET“ dotazy určené jen pro čtení dat.
Pro lepší přehlednost a čitelnost bude reprezentaci zdrojů v API plnit formát JSON.
Pro mapovaní hodnot z databáze do objektu v JAVA bude použít „Mybatis“ framework, jelikož nabízí značnou konfigurovatelnost a jednoduchou práci s SQL[4]. Je samozřejmě možné použít jiné nástroje jako vrstvu mezi databází a aplikaci, například dnes velice populární „Hibernate“.
Pro rychlý start a funkčnost aplikace v krátké době je „Hibernate“ dobré řešení. Pokud má ale aplikace větší nároky a stále stoupající složitost dotazu do databáze, lze narazit na neočekávané chování aplikace a na některé omezení v podobě spojení nebo tvoření dlouhých dotazů v HQL („Hibernate Querry Language“).
V kapitole IP filtr bude popsána implementace filtru IP adres kvůli omezení zatížení v aplikaci. V části Odkazy bude podrobněji rozebráno, který způsob pro poskytnutí odkazu klientovi je nejvhodnější proto, aby data zůstávala v snadno zpracovatelné podobě.
Nakonec bude popsána dokumentace potřebná pro pochopení API dalšími konzumenty..
Po té, co aplikace je vyvinuta a běží, je nutné ji otestovat. Poslední kapitola se zabývá pouze testováním.

5.1.1 Vývojové prostředí

Jako vývojové prostředí bylo použito IntelliJ Idea 2017. Vybráno bylo z důvodu velké nabídky různých pomocných nástrojů zabudovaných přímo uvnitř prostředí. Idea nabízí spoustu pečlivě promýšlených a zajímavých funkcí pro zefektivnění vývoje a testovaní. Jednou z takových funkcí bylo připojení do databáze. Idea se umí přímo ve vývojovém prostředí napojit na databázi, kterou vývojář používá. Následně po konfiguraci si Idea stáhne do lokálního úložiště záznamy z databáze a zpřístupní je. Výsledkem toho je efektivnější práce vývojáře, jelikož už není třeba přepínat mezi nástrojem poskytnutým dodavatelem databáze a vývojovým prostředím. Idea po nastavení databáze nahrazuje plnohodnotné nástroje od dodavatele. Takových příkladů lze uvést pro toto vývojové prostředí ještě mnoho.
Daná práce byla implementována v jazyce JAVA[5]. Tento jazyk patří do skupiny vyšších programovacích jazyků, pojem, který se používá u jazyků s vetší mírou abstrakce. To znamená, že jazyk je postaven tak, aby logika řešení problému byla podobná tomu,

jakým pohledem problém řeší člověk, na rozdíl od nižších programovacích jazyků, které jsou ve způsobu psaní kódů a řešení logiky blíže strojům.
JAVA je jedním z nejvíce populárních jazyků pro vývoj klient-server web aplikací. Je to dané tím, že nabízí velkou škálu nástrojů určené právě pro kvalitní a rychlý vývoj aplikací tohoto typu. Tento programovací jazyk je navržen pro multiplatformní použití, této přenositelnosti je dosaženo díky JVM („Java Virtual Machine“). JVM je sada programů používaná pro spuštění programu ve virtuálním stroji, v tomto virtuálním stroji se pak spouští úplně stejný program napsaný v JAVA na všech platformách. Pro vývoj v JAVA je nutné nainstalovat prostředí JDK („Java Development Kit“). JDK v sobě obsahuje JRE („Java Runtime Enviroment“), ve kterém se nachází již zmíněný virtuální stroj Java Virtual Machine.
V dané práci byl použít JAVA JDK 8. Po nainstalovaní JDK je nutné nastavit systémové proměnné PATH přidáním cesty k bin složce nainstalovaného JDK.

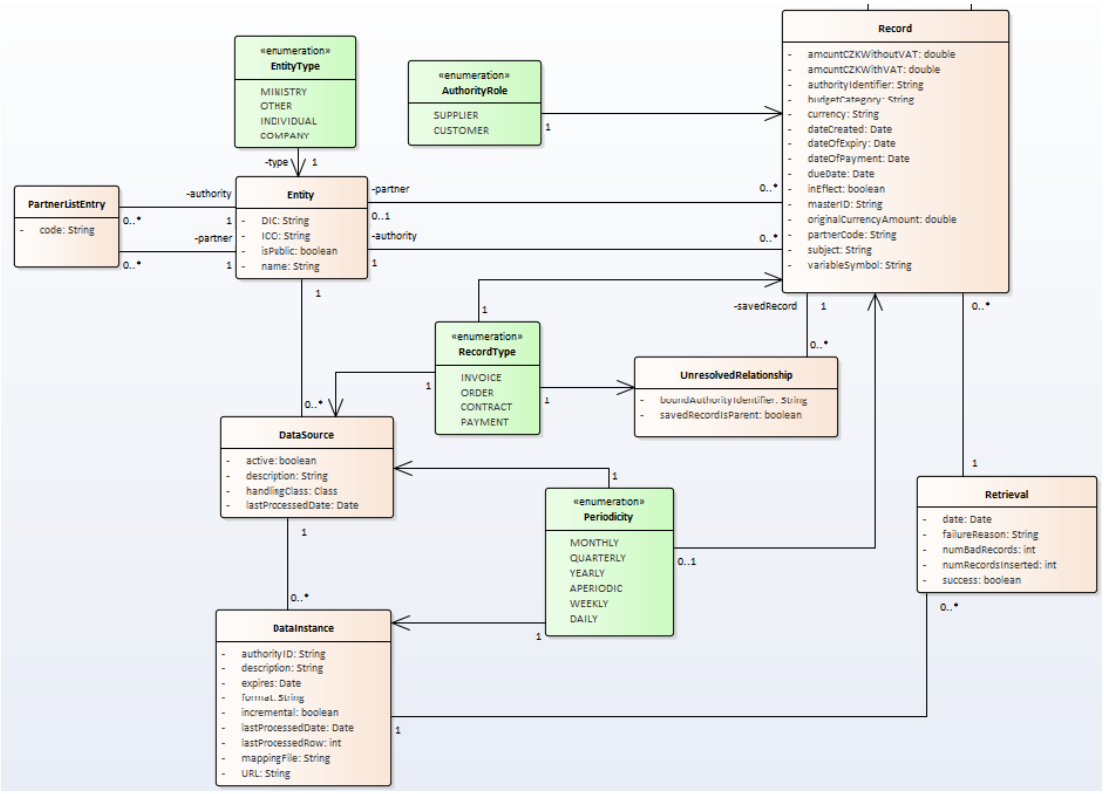
5.1.2 Nástroje

Pro vývoj API byl použit Spring Boot[6], což je nadstavba pro Spring Framework[7]. Spring Framework je platforma pro JAVA poskytující podporu pro vývoj nejrůznějších JAVA aplikací. Spring Boot je nadstavba pro Spring, která zjednodušuje použití frameworku Spring a zrychluje celý proces vývoje tím, že již v sobě obsahuje přednastavení. Výhodou použití Spring Boot je již zmíněná rychlost při startu projektu, snadná konfigurace a zabudovaný server Apache Maven, tudíž není třeba konfigurovat server ve vývojovém prostředí, stačí jen spustit aplikaci z hlavní třídy.
Dalším z použitých nástrojů je Apache Maven. Tento nástroj se používá pro řízení a spravování projektu. Projekt musí obsahovat soubor POM.xml, ve kterém je veškerá konfigurace daného nástroje. Maven řídí sestavení projektu a doplňuje závislosti potřebné pro úspěšnou kompilaci aplikace.

5.1.3 Databáze

V této práci byla použita databáze PostgreSQL[4], která je dostupná pro použití zdarma. Mezi její další výhody patří jednoduchá rozšiřitelnost, multiplatformnost a stabilita.
Pro tento projekt byla už databáze vytvořena včetně dat uvnitř a poskytnuta pro tuto práci ve formě lokálního souboru. Pro zobrazení a práci s touto databází bylo zapotřebí stáhnout software od dodavatele a následně obnovit databázi z poskytnutého souboru.

Struktura databáze je vidět na obrázku 9, v daném nástroji je možné vytvářet SQL dotazy do databáze, mazat nebo sestavovat databázi. Nelze tam však vidět vztahy mezi jednotlivými tabulkami, proto je potřeba dalšího nástroje.
V teto práci byl použít Enterprise Architect Lite (volná licence). V tom jsou jasně viditelné všechny vztahy mezi tabulkami, které se nachází v poskytnuté databázi. Lze pozorovat, že hlavní tabulkou v tomto schématu je „Record“, ve které jsou všechna data o konkrétním záznamu.



Obrázek 9 Vztahy mezi tabulkami

5.1.4 JSON formát

Aplikace produkuje jako výsledek své činnosti data. Tato data musí pro srozumitelnost a další zpracovaní splňovat určitý formát. V dané práci byl jako formát pro reprezentaci zdrojů použít JSON („JavaScript Object Notation").
JSON se v poslední době velice často používá při tvorbě Rest API, je to dané tím, že jeho formátovaní dat je poměrně jednoduché a celkem čitelné pro člověka. Jeho struktura se dá definovat jako seřazený seznam objektů. JSON byl použit v této práci z důvodu jednoduchosti formátu výsledných dat, příklad záznamu z databáze lze vidět na obrázku 10. Na rozdíl od dalších populárních formátů, jako je například XML, nemá JSON zavírací závorky, což ho dělá proti XML čitelnějším a vice přehledným.

Obrázek 10 JSON formát

5.1.5 Implementace API

Jedním z cílů této práce je vytvořit Rest API, které bude následně poskytnuto třetím stranám. V této části budou zjednodušeně a obecně popsány části aplikace, které jsou nutné k její správné funkčnosti.
První krok tohoto procesu lze udělat založením projektu ve vývojovém prostředí, kde bude nakonfigurován Maven nastavením POM souboru a následně vložena první konfigurace pro použití Spring Boot. Nebo lze použít stránku spring inicializer pro ještě pohodlnější nastavení projektu.
Po vytvoření projektu a základního nastavení lze vytvořit kostru. Kostra projektu se bude skládat z logických celků jako je Kontrolér, ve kterém se bude implementovat způsob, jakým se data budou získávat, model, ve kterém bude popsána struktura databáze, a „Mapper", termín používaný pro technologii Mybatis, což je XML soubor, který bude obsahovat konkrétní dotazy do databáze. Vše se podrobněji rozebere v dalších podkapitolách práce.

Model

Pro začátek je potřeba vytvořit mapování databáze. Mapování se provádí z toho důvodu, aby hodnoty, které se nachází v databázi, byly přeneseny na objekty v JAVA. Poté, co se na objekty namapují hodnoty, s nimi lze provádět další operace.
Mapovaní databáze je prováděno do takzvané třídy POJO, takové třídy obsahují pouze proměnné podle těch v databázi. Pro získávaní hodnot proměnných v těchto třídách musí ještě být nastaveno získávaní a nastavení pro každou položku, jak lze vidět níže. Získávaní a nastavení se vytváří proto, aby s proměnnými bylo možné pracovat v rámci celé aplikace a zároveň aby bylo možné hodnoty v nich získat nebo nastavit.

```
@JsonProperty
    private String mappingFile;
    private String format;

    public String getFormat() {
        return format;
    }
    public void setFormat(String format) {
        this.format = format;
    }
    public String getMappingFile() {
        return mappingFile;
    }
    public void setMappingFile(String mappingFile) {
        this.mappingFile = mappingFile;
    }
```

Mybatis technologie

Mybatis[8] je framework ve světě JAVA patřící do skupiny JPA („Java Persistance Framework“), technologie sloužící jako vrstva abstrakce nad databází. Mybatis umožňuje automatizaci ukládaní a čtení dat z databáze.
Tento nástroj zároveň patří do skupiny ORM („Object Relational Mapping“), lze tedy dosáhnout mapovaní objektu v aplikaci na data, která se nachází v databázi.
Pro konfiguraci daného frameworku je nejdříve potřeba přidat knihovny do POM souboru v kořenovém adresáři projektu, tím se stáhnou všechny potřebné knihovny pro použití tohoto nástroje. Následně je potřeba vytvořit dva XML soubory. Jeden slouží pro konfiguraci samotného frameworku. Konfigurace může byt poměrně rozsáhlá, od ukládání dat do mezipaměti až po nastavení vlastností pro jednotlivé dotazy do databáze. V této práci nebylo nutné nastavovat další konfigurace, viz blok kódu níže.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration
        PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
        "http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
    <settings>
        <setting name="logImpl" value="LOG4J"/>
    </settings>
    <mappers>
        <mapper resource="mapper/RecordMapper.xml"/>
    </mappers>
</configuration>
```

Druhý soubor obsahuje sadu SQL dotazů napsaných ve specifické podobě Dynamic SQL tohoto frameworku. Dynamic SQL[8] je jedna ze zajímavých funkcí Mybatis umožňující poměrně snadné psaní SQL dotazu, jejich spojení a podmínění v jednom XML souboru. Lze v něm používat podmínky jako „pokud“, „vyber“, „pro každý“. Všechny tyto podmínky pomáhají k přehlednému tvoření dotazů do databáze. Část Dynamic SQL lze vidět v následujícím kódu.

```
<select id="searchByName" resultMap="recordResultMap">
SELECT * FROM record r JOIN entity e ON r.authority = e.entity_id
WHERE 1=1
        <if test="name != null">
            <bind name="pattern" value="'%' + name + '%'" />
            AND LOWER(r.subject) LIKE LOWER(#{pattern})
        </if>
        UNION
SELECT * FROM record r JOIN entity e ON r.partner = e.entity_id
WHERE 1=1
        <if test="name != null">
            <bind name="pattern" value="'%' + name + '%'" />
            AND LOWER(r.subject) LIKE LOWER(#{pattern})
        </if>
        <if test="name != null">
            ORDER BY subject DESC
        </if>
</select>
```

Kontrolér

Tato vrstva aplikace[7] se stará o zpracování dotazů a odpovědí, které budou přicházet ve formě HTTP. Zde byl použit Spring framework. Spring nabízí užitečné funkce a doporučení při vývoji aplikací. V daném případě byly použity komponenty @RestController a @RequestMapping. Kontrolér vystavuje metody, které byly okomentovány pro mapování jako koncové body protokolu HTTP.
Pokud příchozí dotaz HTTP odpovídá metodě, pro kterou je nastavené mapovaní, potom metoda bude zavolána v kódu a provede se celý řetězec zpracovaní, ověření a vrácení dat včetně odkazu a dalších informací zpět stroji nebo klientu ve formátu JSON.
Zjednodušený příklad konfigurace lze vidět v kódu níže, kde je nastaveno mapování na cestě „/search“, takže pokud se vygeneruje dotaz s tímto URI na portu, na kterém běží

aplikace, pak se použije příslušná metoda. Výstupem budou data a status odpovědi, pokud se data najdou a všechny validace projdou, vrátí se klientovi nebo dalšímu serveru status „OK“.

```
@RestController
public class MainController {

@RequestMapping(value = "/search",
method = RequestMethod.GET, produces = "application/json")
public @ResponseBody ResponseEntity<List<Record>> getByName(
@RequestParam(value = "name", required = false) String name,
@RequestParam(value = "page", required = false, defaultValue = "1")
Integer page,
@RequestParam(value = "size", required = false, defaultValue = "30")
Integer size) {}
```

V této vrstvě aplikace byly použité mnohé další funkcionality Spring frameworku jako například @ResponseBody, která říká, že výstupem konkrétní metody jsou data pro konzumenta služby. @RequestParam slouží pro získaní parametru z URI a jejich mapovaní do objektu v JAVA.

Odkazy

Proto, aby byla splněna podmínka Rest API ohledně HATEOAS[9], musí se do reprezentace zdrojů, která se vrací klientovi, vkládat odkazy na další možné akce, které klient může podniknout. Tudíž klientovi stačí první validní URI dotaz a následně se klient může řídit samostatně přes dynamicky poskytované hypermédium.
V této práci bylo vybráno vkládání odkazů do hlaviček HTTP dotazu. Je to dané tím, že pokud se odkazy vloží přímo do těla JSON odpovědi, musí následně druhá strana tyto odkazy získávat z každého záznamu dalším zpracováním dat.
Pokud se odkazy nachází v hlavičce HTTP odpovědi, klient může snadno získat odkazy na další možné akce poskytované API. Přitom data, která se klientu vrátila, neobsahují žádné další části kódů, pouze reprezentaci zdrojů. Tudíž tato data lze pak rovnou používat pro další účely, například zobrazení na webovém portálu. Příklad odkazu poskytovaných dynamicky je níže.

```
Response Headers
Content-Type:application/json;charset=UTF-8
Date:Wed, 24 May 2017 13:53:18 GMT
Links:<http://localhost:8080/search?name=ekol&page=1&size=1>;rel="first",
<http://localhost:8080/search?name=ekol&page=2&size=1>;rel="next",
,<http://localhost:8080/search?name=ekol&page=213&size=1>;rel="last"
Transfer-Encoding:chunked
X-Forwarded-For:0:0:0:0:0:0:0:1
X-Total-Page-Count:214
X-Total-Records:214
```

IP filtr

Pro řízení zatížení v této práci byl implementován IP filtr, který při každém volání aplikace zjišťuje IP („Internet Protocol“) číselně vyjádřenou adresu klienta, a pokud

daná IP překročí počet volání nastavený v aplikaci, bude přesunuta do Blacklistu. Tento termín se používá pro případy, kdy se zamezuje úplný přístup konkrétní aplikaci nebo klientu tím, že se klient přesune na speciální seznam.
Filtr byl implementován za pomoci Spring frameworku. Z každého dotazu byla získána IP adresa klienta a následně provedena kontrola, zda už daná adresa překročila počet volání. Tato část byla implementována ve třidě IpLimitFilter. Jako příklady, kde se používá „blacklisting“, lze uvést web servery, firewally nebo spam filtry.

Omezení přístupů IP

V dané aplikaci byl implementován kromě IP filtru na „blacklisting“ i filtr na časové omezení dotazu. Anglicky termín pro tuto funkčnost je „Throttling“, používá se pro vyrovnání okamžité zátěže na server. V aplikaci je nastaven pevně limit, po kterém aplikace začne omezovat dotazy na určitý čas. V případě, že někdo bude pokračovat v posílaní dotazu, se adresa zablokuje přesunutím do„blacklistu“. V opačném případě, pokud počet dotazů klesne, aplikace obnoví časový limit zpět na původní hodnotu.

Plánovač stahování

V aplikaci je implementováno pravidelné spouštění úlohy, která volá již existující aplikaci na stahování veřejně dostupných dat z ministerstev a dalších institucí do lokální databáze PostgreSQL.
Github Profinit.
Z tabulky „retrieval“ je možné zjistit, zda k aktualizaci došlo, jestli bylo volání úspěšné a kolik záznamů se podařilo nebo nepodařilo naimportovat.
Na webových stránkách je pak tato informace prezentována uživatelům, aby byli informováni o případném výpadku/chybách v poskytování dat.

Zaznamenávání

Při výběru zaznamenávacího nástroje byl kladen důraz na možné pozdější potřeby aplikace. Použitý zaznamenávací nastroj v této aplikaci je log4j, který nabízí velkou konfigurovatelnost, různé typy zaznamenávání, zaznamenávání určitých typů souborů, zápis chyb do souboru a další.
Pro nastavení zaznamenávání je potřeba stáhnout doplňující knihovny a přidat je v kořenové složce projektu do XML souboru POM. Následně je potřeba nastavit konfiguraci v souboru log4j.properties, viz následující kód.

```
log4j.rootLogger=DEBUG, stdout
#"file" parameter provide logging to file with defined limit
#log4j.rootLogger=DEBUG, stdout, file
# MyBatis logging configuration...
log4j.logger.eu.profinit.opendata.RecorMapper=TRACE
# Console output...
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%5p [%t] - %m%n
```

Po dotažení knihoven a nastavení v konfiguračním souboru lze používat daný nástroj v kódu. Zaznamenávání v aplikaci bylo nastavené primárně v IP filtru, jelikož tam bude pravděpodobně nejčastěji používáno. V budoucnu lze zaznamenávání nastavit všude,

kde procházejí data aplikací, a tak lze snadno odhalovat chyby, protože v aplikačním záznamu bude celý průchod aplikací až do okamžiku, kdy chyba nastala.

Dokumentace API

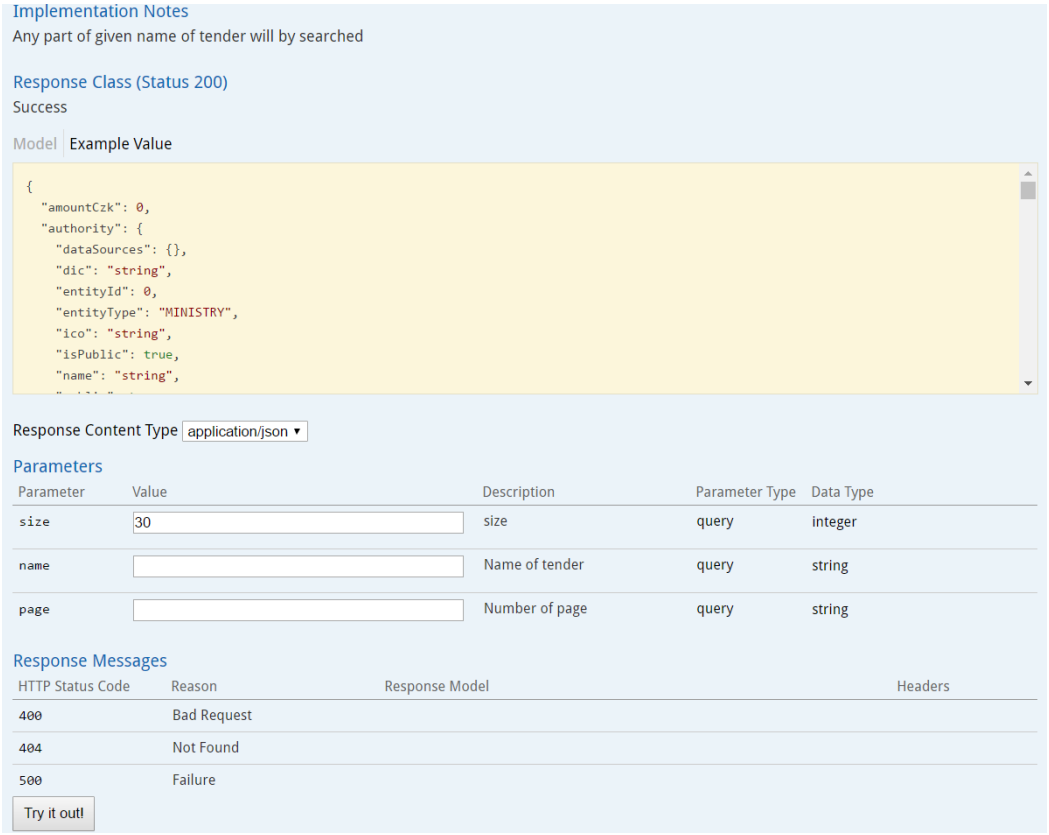
Poměrně důležitou součástí API je dokumentace[10]. Dokumentace slouží především pro někoho, kdo bude chtít dané API integrovat do svých stránek.
V této práci byl použit populární nástroj pro dokumentaci Swagger2. Tento nástroj byl vybrán, protože nabízí podporu pro integraci do Spring Boot aplikací. Největší výhodou této integrace je, že se dá Swagger2 používat pomocí anotací přímo v JAVA.
V této aplikaci se anotace k dokumentaci nachází v Kontroléru, tudíž jsou snadně dostupné z jednoho místa a prostředí a při změně v aplikacích lze taktéž snadno změnit dokumentaci. Dokumentace tudíž zůstává vždy aktuální a směrodatná pro uživatele, kteří chtějí API použít.
Pro nastavení Swagger2 je potřeba pouze přidat závislosti do POM souboru. Další výhodou Swagger2 je to, že jej lze konfigurovat kompletně pomocí kódu. Za tímto účelem byla vytvořena třída přímo pro konfiguraci Swagger2, která je v kódu níže.

```
@Configuration
@EnableSwagger2
public class SwaggerConfig {
@Bean
public Docket openDataApi() {
return new Docket(DocumentationType.SWAGGER_2)
.apiInfo(apiInfo())
.useDefaultResponseMessages(false)
.select()
.build();
}
private ApiInfo apiInfo() {
return new ApiInfoBuilder()
.title("Opendata, veřejné zakázky")
.description("Rest API Powered By Profinit.eu")
.termsOfServiceUrl("http://www-03.ibm.com/software/sla/sladb.nsf/sla/bm?Open")
.license("Apache License Version 2.0")
.licenseUrl("https://github.com/IBM-Bluemix/news-aggregator/blob/master/LICENSE")
.version("1.0")
.build();
 }
}
```

Po nastavení samotného Swagger2 lze nastavit odpovídající části aplikace, jako je uvedeno pro příklad v kódu níže. Po nastavení Kontroléru Swagger2 ví, jaké jsou koncové body aplikace, jaké vracejí stavy a v jakém formátu API poskytuje reprezentaci zdrojů.

```
@ApiOperation(value = "Search by name", notes = "Any part of given name
of tender will by searched", produces = "application/json")
@ApiResponses(value ={
@ApiResponse(code = 200, message = "Success", response = Record.class),
@ApiResponse(code = 400, message = "Bad Request"),
@ApiResponse(code = 404, message = "Not Found"),
@ApiResponse(code = 500, message = "Failure")})
@ApiImplicitParams({
@ApiImplicitParam(name = "name", value = "Name of tender",
required = false, dataType = "string", paramType = "query"),
@ApiImplicitParam(name = "page", value = "Number of page",
required = false, dataType = "string", paramType = "query")})
@Produces(value = "application/json")
@CrossOrigin()
@RequestMapping(value = "/search", method = RequestMethod.GET,
produces = "application/json")
```

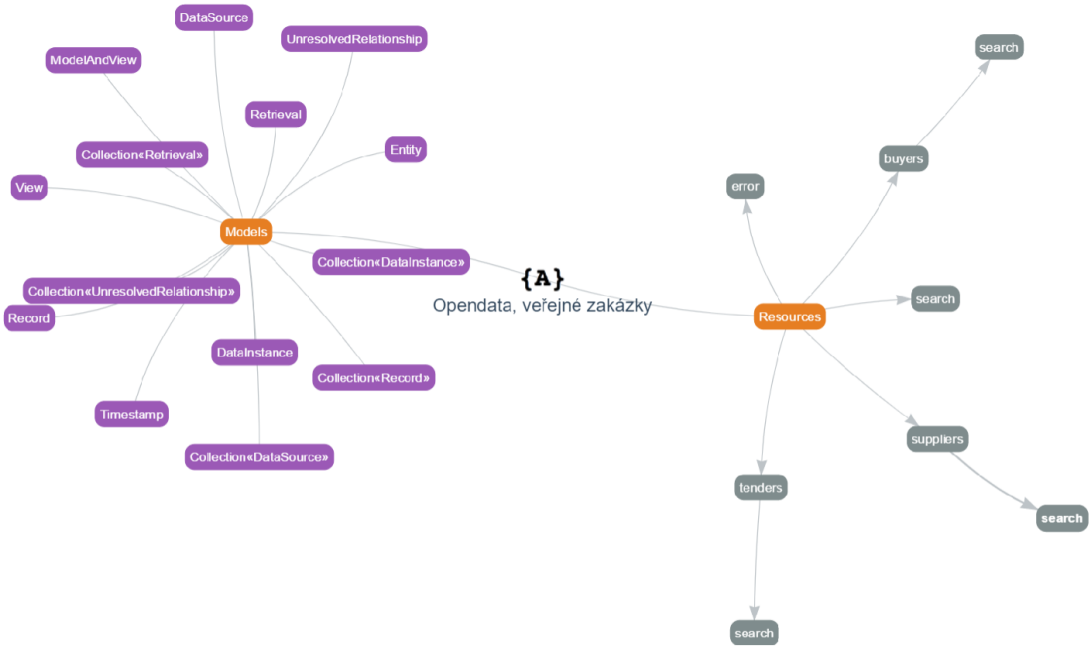
Výsledkem je, že na adrese uvedené v dokumentaci pro tento nástroj „/swagger-ui.html“ lze vidět dokumentaci v podobě dotazů, které lze na API aplikovat. Po rozkliknutí jednotlivých metod lze přímo vyzkoušet, jaká data API vrací, v jakém formátu, případně jaké parametry lze do dotazu vložit. Viz obrázek 11.



Obrázek 11 Swagger dokumentace

Pro ještě lepší grafické znázorněni zdrojů a metod, kterými k nim lze přistupovat, může uživatel použít rozšíření „swagger.ed“ v prohlížeči Google Chrome, který, pokud aplikace běží, nabízí možnost na adrese „/v2/api-docs“ vidět pohyblivou grafickou mapu

koncových bodů systému. V této mapě lze taktéž zkoumat každý koncový bod, jak lze vidět na obrázku 12.



Obrázek 12 Swagger grafický model

5.1.6 Testování

Po dokončeni implementace API je třeba ho otestovat. V této aplikaci má API z důvodu zamezení modifikací dat koncové body nastavené pouze na čtení, tudíž není třeba žádné složité aplikace pro testováni.
V této práci byl pro testování použít nástroj od Google „Postman“. To je jednoduchý a přehledný nástroj, který nabízí základní možnosti testování webových aplikaci.
Po puštění aplikace je potřeba ověřit, zda funguje a vrací reprezentaci zdrojů takovou, jak bylo zamýšleno. V tomto případě nebyla potřeba testovací analýza, jelikož aplikace má jen několik koncových bodů, na kterých lze jenom čist, případně přidat dva parametry navíc. Všechny koncové body aplikace byly otestovány, včetně zadávaní doplňujících parametrů jako je počet stránek nebo počet záznamů na stránku. Příklad testovací URI: `http://localhost:8080/search?name=ekol&page=1&size=1`
Z URI jde vypozorovat, že je testován konečný bod „/search“. Hledání je provedeno podle jména a zároveň obsahuje další parametry jako je stránka a počet záznamů na stránku. Vždy bylo zkontrolováno, že se u konkrétního dotazu nachází odkazy v hlavičce HTTP. V průběhu testování bylo objeveno a opraveno několik chyb tykajících se stránkování a poskytování odkazů pro navigaci klienta.
Největším problémem této části byla pravděpodobně nekonzistence dat, při testování byla objevena chyba související s načítáním záznamů do databáze.
Některé záznamy se zapsaly jen částečně a některé byly například až na jeden atribut prázdné. Tomuto se bohužel nelze vyhnout, jelikož tyto záznamy na jednotlivých ministerstvech zadávají do databáze ručně lidé, tudíž vždy muže nastat chyba lidského faktoru.

5.2 Web portál

V předchozí části práce bylo implementováno Rest API, které poskytuje reprezentaci zdrojů ve formátu JSON. Tento způsob zobrazení dat pro koncového uživatele, pokud není strojem, není úplně vhodný. Proto byl, jako další část práce, vytvořen webový portál, na kterém lze výsledná data vidět v čitelnější podobě pro člověka.

Webový portál bude reagovat na akce uživatele a v případě zadaní vstupních hodnot pro vyhledávaní a potvrzení hledání tlačítkem odešle JSP („Java Server Pages“) dotaz POST do webové aplikace. Dotaz POST se používá pro získaní proměnných z formuláře. To znamená, že uživatel vyhledávacím tlačítkem potvrdí odeslání formuláře a následně Kontrolér v aplikaci zachytí toto volaní a získá z něho vstupní hodnoty.
Pokud URI, ze kterého se uživatel dotazuje, je validní a vstupní hodnoty jsou zadané, provede se zpracovaní dotazu v Kontroléru aplikace. Zde probíhají validace a vrácení stránky s vyplněnými hodnotami.
Před tím, než aplikace vrátí stránku s výsledky, musí získat pro požadované vstupní hodnoty data. V této práci byl použít nástroj Retrofit, populární nástroj pro vývoj Android aplikací. Zde byl zvolen, protože umožňuje poměrně rychlé nastavení a velkou konfigurovatelnost a přitom neztrácí na funkčnosti.
V Kontroléru aplikace zjistí, na které URI a s kterými parametry se volal dotaz. Podle toho vloží parametry do rozhraní Retrofitu a ten zavolá API už se všemi vstupními hodnotami.
Do Kontroléru se následně vrátí seznam hodnot, které se nastaví do JSP, a následně obnovená stránka se vrátí uživateli služby.

5.2.1 Vývojové prostředí

Stejně jako v předchozí části Rest API bylo použito vývojové prostředí IntelliJ Idea. Konfigurace zůstala stejná, tudíž se stále jedná o programovací jazyk JAVA a aplikaci postavenou pomocí Spring Boot frameworku.

Technologie webových stránek

Jako technologie stránek pro webový portál byla zvolena JSP. Výhoda této technologie je v tom, že používá ve své podstatě jednoduché HTML (“Hypertext Markup Language”), které zpracovává a vrací uživateli JAVA Servlet, ve své podstatě je to JAVA třída, která provádí komunikaci na modelu klient-server a většinou zpracovává HTTP požadavky, ačkoliv může fungovat i na jiných protokolech. Pomoci JAVA Servletu lze snadno pracovat s dotazy HTTP protokolu.

5.2.2 Implementace webového portálu

- model
  - AuthorityRole
  - DataInstance
  - DataSource
  - DataSourceHandler
  - Entity
  - EntityType
  - ParentListEntry
  - Periodicity
  - Record
  - RecordType
  - Retrieval
  - UnresolvedRelationship

Aplikace je postavena na stejné technologii Spring Boot frameworku a je implementována pomocí jazyku JAVA. Jako řídicí aplikace pro správné dotažení závislostí a kompilací byl zvolen Maven.

Zjednodušený princip fungování byl popsán v kapitole „Web portál“, zde bude popsaná implementace. Po založeni projektu a prvotní konfiguraci Spring Boot je potřeba vytvořit POJO model, podle kterého se budou mapovat zdroje získané z API na objekty v JAVA, vytvořený model lze vidět na začátku podkapitoly.

Rozhraní pro volání API

Jak již název sekce napovídá, zde se bude implementovat volání API. Pro použití nástroje Retrofit[11] je nutné stáhnout jeho knihovny a přidat je do POM souboru, tímto se zajistí přístup k danému nastroji v aplikaci. Přidání nástroje do projektu lze vidět v následujícím kódu.

```
<dependency>
<groupId>com.squareup.retrofit2</groupId>
<artifactId>retrofit</artifactId>
<version>2.2.0</version>
</dependency>
```

Dále je potřeba Retrofit nastavit. Nastavení probíhá poměrně jednoduše a je k vidění v kódu níže. Je zde vidět i napevno nastavená adresa, na kterou má Retrofit posílat dotazy, v daném případě je nastavena lokální adresa Rest API.

```
@Bean
public Retrofit retrofit(OkHttpClient client) {
Gson gSon=  new GsonBuilder().setDateFormat("yyyy-MM-dd").create();
return new Retrofit.Builder()
.addConverterFactory(GsonConverterFactory.create(gSon))
.baseUrl("http://localhost:8080/")
.client(client)
.build();
}
```

Pro používaní Retrofitu je vše téměř hotové. Nyní lze vytvořit rozhraní, které slouží pro volání API. Zde se vkládají parametry do dotazu, které jsou pak odesílány do API. Příklad volání lze vidět v kódu níže.

```
@Component
public interface OpenDataAPI {
@GET("/search")
Call<List<Record>> getRecordByName(@Query(value = "name", encoded = true)
String name);
```

Struktura stránek

Samotné stránky jsou napsané v HTML, které je vložené do JSP. Právě taková konstrukce umožňuje použití JAVA Servletu[7], který se stará o komunikaci na protokolu HTTP.
Aplikace obsahuje čtyři základní stránky: Domácí stránka, Vyhledávání podle zakázky, Vyhledávání podle zadavatele, Vyhledávání podle dodavatele. Každá z těchto stránek obsahuje formulář, který se odesílá do aplikace po kliknutí na tlačítko hledání. V okamžiku stisku tlačítka se posílá HTTP dotaz POST na aplikaci. Takto se vstupní data dostanou přímo do aplikace přes JAVA Servlet a následně do Kontroléru.

Na webovém portálu se nachází další stránky, které nesou informativní charakter. Stránka „API“ popisuje, kde a jak přistupovat k API, případně kde je k němu dokumentace.
Na stránce „Data“ je uvedeno, kdo data poskytuje pro další zpracovaní a kdy byla data v databázi naposledy obnovena.
Stránka „O projektu“ obsahuje informaci o tom, kdo za projektem stojí, kdo jej realizoval a proč projekt vzniknul.

Kontrolér

I zde byl použít Spring framework, na rozdíl od Rest API je zde ale použitý jiný Kontrolér. V této části práce byl použit Kontrolér, který je určen pro práci s JSP.

Princip fungování je následující: koncový uživatel odešle z webového portálu požadavek ve formě HTTP POST dotazu, pokud URI požadavek bude validní proti tomu, co je napsané v Kontroléru, Kontrolér volání zachytí a provede se veškerá logika spojená s validací vstupu a dotažení zdrojů z API. Po vykonání veškeré logiky spojené se zpracováním dotazu nastaví Kontrolér konkrétní stránce hodnoty a vrátí ji uživateli. Tuto rutinu pak opakuje na veškeré URI, které mé v sobě popsané.

5.2.3 Testování

Testování webového portálu probíhalo ručně. Každá stránka byla otestována na základní funkčnost, to znamená, že bylo ověřeno odesílání formulářů ze stránky či zda zadané vstupní hodnoty na stránce odpovídají těm, které jsou v dotazu do API. Dále po nastavení na nízký limit se testovalo, zda funguje omezení přístupu IP adresy, ze které přichází na API počet dotazů překračující nastavený limit.

6 Návrh dalších vylepšení

6.1 Vylepšení Rest API

Počet vylepšení a rozšíření pro část Rest API může být opravdu hodně. Pro začátek by se dalo API rozšířit o mnoho dalších hledacích kritérií. Například, aby API umělo hledat záznamy podle variabilního symbolu nebo podle data platby, případně dalších. Do API lze poměrně snadno přidat CACHE buď v Spring framevorku nebo v konfiguraci pro Mybatis framework.

Dále lze do aplikace přidat parametr pro každý koncový bod, který bude API říkat, aby vrátilo pouze seznam klíčových údajů pro zadané vyhledávací kritérium. Je to velice užitečná funkčnost, protože když uživatel hledá určitý záznam, zajímá ho většinou několik klíčových identifikací záznamu, jako je například jméno, IČO, DIČ a další.

Tudíž poté, co API vrátí seznam klíčových údajů pro jednotlivé záznamy, si je uživatel projde a zvolí pouze jeden, což znamená, že se znovu zavolá API, ale tentokrát se vrací jeden konkrétní celý záznam.

Tímto se nejen zredukuje objem přenesených dat, ale taktéž je to užitečné pro vývoj webového portálu, jelikož není třeba na jedné stránce zobrazovat velké množství záznamů v nepřehledné podobě. Jako další rozšíření, které by stálo za úvahu, je vrácení pouze jednoho atributu z celého záznamu. Příkladem může být volaní API, kde Klient, ať je to člověk nebo stroj, potřebuje vědět jenom datum, kdy byla zakázka uskutečněna, tudíž zavolá API s kritérii pro vyhledaní a za to přidá identifikátor atributu, který se má vrátit zpátky.

Pro lepší navigaci a řízení Klienta hypertextem by se dalo rozšířit HTTP hlavičku o další stavy, odkazy a možnosti navigace.

Dalším užitečným rozšířením aplikace můžou být jednotkové testy, automatické testy, které se spouštějí při kompilaci programu. Jednotkové testy jsou užitečné z hlediska údržby aplikace, jelikož fungují jako testy jednotlivých funkci aplikace a pokud nastane nějaká nečekaná změna v aplikaci, například při vývoji, jednotkové testy neproběhnou do konce. Takto se dá zjistit, zda nově naimplementovaná funkce aplikace splňuje staré požadavky a zda to tak má být.

6.2 Vylepšení Webového portálu

Jelikož Webový portál je to, co koncový uživatel vnímá jako výsledek poskytované služby, je potřeba, aby byl portál přehledný, čitelný a jasně popsaný. Zároveň aby byl portál moderní, bylo by vhodné jej realizovat s pomocí nových nástrojů jako je třeba AngularJS. Tudíž dalším krokem ve vývoji musí být re-stylizace celého portálu.

Přímo na Webovém portálu lze popsat API a jak jej používat.

Další možností vylepšení je implementovat do portálu Google Analytics, nástroj pro sledování statistik návštěvnosti. Jedná se o užitečný nástroj pro analýzu a budoucí vývoj na základě této analýzy.

7 Závěr

V rámci této bakalářské práce byly postupně teoreticky rozebrány a následně použity při implementaci technologie pro vývoj webových aplikaci. Cílem práce bylo vytvořit Rest API a jednoduchý webový portál, kde uživatelé budou moci hledat záznamy ve státních veřejných zakázkách, ale hlavně bude pro veřejnost vystavené API, rozhraní přes které třetí strany můžou používat data o veřejných zakázkách.

K dosažení tohoto cíle bylo nezbytné naučit se programovací jazyk JAVA, ovládat framework Javy jako je Spring Boot a Spring, pracovat s databází a jazykem SQL, případně Dynamic SQL v frameworku Mybatis, byly nutné částečné znalosti programováni JSP a tím pádem i HTML, znalosti stavby architektury webových aplikaci a znalost architekturního stylu Rest. Všechny tyto technologie a nástroje zmiňuje podrobněji teoretická část práce. Následně, v části implementace, se přistupovalo ke konkrétním metodikám a realizaci samotné práce.

Před tím, než byla vytvořena daná aplikace, byl vyvinut podobný prototyp v rámci této práce s pomocí JAVA frameworku „Hibernate“. Bohužel, až na konci implementace API se zjistilo, že tento framework nenabízí tak dobrou konfigurovatelnost a má výkonové problémy týkající se dotazu do databáze proti „Mybatis“. Pro malé projekty s rychlým startem je Hibernate framework ideálním řešením, ale pro aplikaci, která bude servisovaná a do budoucna rozšiřovaná, je to nevhodné řešení.

Cil práce byl splněn včetně všech položek, které byly uvedené v zadaní. Aplikaci čeká postupný vývoj webového portálu a rozšíření Rest API o další funkce. Následně bude aplikace veřejně dostupná a provozována společnosti „Profinit EU s.r.o“.

Příloha A

CD

Seznam souborů na CD:
serhij horčička.pdf
zdrojový kód aplikace.zip
zdrojový kód projektu sharelatex.zip
návod použití aplikace.txt

Literatura

[1] Roy Thomas Fielding Ph.D. "Architectural Styles and the Design of Network-based Software Architectures". Dipl. University of California, Irvine, 2000. URL: https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm.

[2] *List of software architecture styles and patterns [online]*. URL: https://en.wikipedia.org/wiki/List_of_software_architecture_styles_and_patterns.

[3] Arcitura Education Inc. *List of software architecture styles and patterns [online]*. URL: http://whatisrest.com/rest_constraints/client_server.

[4] Korry Douglas. *PostgreSQL: A Comprehensive Guide to Building, Programming, and Administering PostgresSQL Databases*. 2. vydání. Sams, 2003. ISBN: 978-0735712577.

[5] Herbert Schildt. *Java 7: Výukový kurz*. 1. vydání. Computer Press, 2012. ISBN: 978-80-251-3748-2.

[6] Alex Antonov. *Spring Boot Cookbook*. 1. vydání. Packt Publishing Limited, 2015. ISBN: 978-1785284151.

[7] Tutorials Point. *Spring tutorial [online]*. URL: https://www.tutorialspoint.com/spring/.

[8] MyBatis.org. *Mybatis Configuration [online]*. URL: http://www.mybatis.org/mybatis-3/configuration.html.

[9] GitHub Inc. *Traversing with Pagination [online]*. URL: https://developer.github.com/v3/guides/traversing-with-pagination/.

[10] John Thompson. *SPRING BOOT RESTFUL API DOCUMENTATION WITH SWAGGER 2 [online]*. URL: https://springframework.guru/spring-boot-restful-api-documentation-with-swagger-2/.

[11] Inc. Square. *Retrofit [online]*. URL: http://square.github.io/retrofit/.

[12] Ryan Tomayko. *How I Explained REST to My Wife [online]*. URL: http://www.looah.com/source/view/2284.