SLOVENSKÁ TECHNICKÁ
UNIVERZITA V BRATISLAVE
FAKULTA INFORMATIKY
A INFORMAČNÝCH TECHNOLÓGIÍ

Tím 15

DOKUMENTÁCIA K INŽINIERSKEMU DIELU

Tímový projekt

Študijný odbor: Informačné systémy, Softvérové inžinierstvo
Miesto vypracovania: Ústav informatiky a softvérového inžinierstva, FIIT STU Bratislava
Vedúca tímu: Ing. Nadežda Andrejčíková, PhD.

December 2014

Obsah

Kapitola 1

Big picture

Úvod

Veda je nepochybne oblasťou, kde je potrebné spracovávať obrovské množstvo informácií. Navyše ak je takého spracovávanie realizované v heterogénnom prostredí je tento proces oveľa náročnejší. Nie len pre výskumníka je dôležité získať informácie a všeobecný pohľad na niektoré diela. V existujúcich citačných indexoch sa nachádzajú bibliografické dáta, ktoré slúžia ako zdroj získavania ohlasov k jednotlivým publikáciám. Ak z takýchto zdrojov získame informácie, vhodne ich analyzujeme a poprepájame, môžu s veľkou mierou uľahčiť výskumníkovi zdĺhavú prácu pri získavaní ohlasov najmä na vlastné diela.

Navrhovaný systém bude teda identifikovať a získavať zdroje z iných externých systémov – primárne z WOSu a SCOPUSu - analyzovať takto získané zdroje, ukladať získavané dáta a analyzovať ohlasy k niektorým vybraným publikáciám. Systém bude slúžiť aj ako úložisko takzvaných fulltextov.

Vybudovaním vlastnej databázy bude systém možné postupne rozširovať na základe prichádzajúcich požiadaviek od používateľov. Takto má systém potenciál byť rozšírený o ďalšie funkcionality. V snahe zabrániť multiplicite, budú duplikované dáta ignorované. Použitím vlastnej lokálnej databázy bude systém rýchlejšie pristupovať k dokumentom, ako by to bolo v prípade dopytovania sa do citačných indexov pri prijatí nových požiadaviek. Systém bude implementovaný ako webová aplikácia poskytujúca službu REST, ktorá bude komunikovať s klientom pomocou JSON formátu. Poskytnutím tejto služby umožníme jednoduché rozširovanie systému o ďalších klientov, alebo jednoduché poskytovanie služieb systému ďalším systémov ako API.

Ciele na zimný semester

Získavanie a vyhľadávanie ohlasov

Najdôležitejším cieľom pre náš projekt počas zimného semestra, je umožniť používateľom systému, vyhľadávať dokumenty v citačných indexoch WOS a SCOPUS. Pre niektoré alebo všetky vyhľadané diela získať ohlasy a takýmto spôsobom prezentovať používateľovi ohlasy na diela z viacerých systémov.

Deduplikácia dát

Dôležitou a neoddeliteľnou časťou je riešenie multiplicity dát, keďže sa môže jednoducho stať, že sa minimálne v dvoch citačných databázach nachádzajú rovnaké diela a ohlasy. Ak by sme neidentifikovali a neriešili problém multiplicity, mohli by umelo zvýšiť počet ohlasov a vrátiť nesprávne výsledky.

Používateľské prostredie a API

Ďalším cieľom je poskytnúť používateľovi rozhranie, ktorým jednoducho odošle požiadavku. Z dôvodu, že používateľ nie je povinný ovládať komunikačný formát výmenu dát (JSON), je potrebné implementovať webové rozhranie s jednoduchými interaktívnymi prvkami. Pre iné systémy bude systém poskytovať API REST rozhranie, komunikujúce pomocou formátu JSON.

Systém

Funkcionálne požiadavky

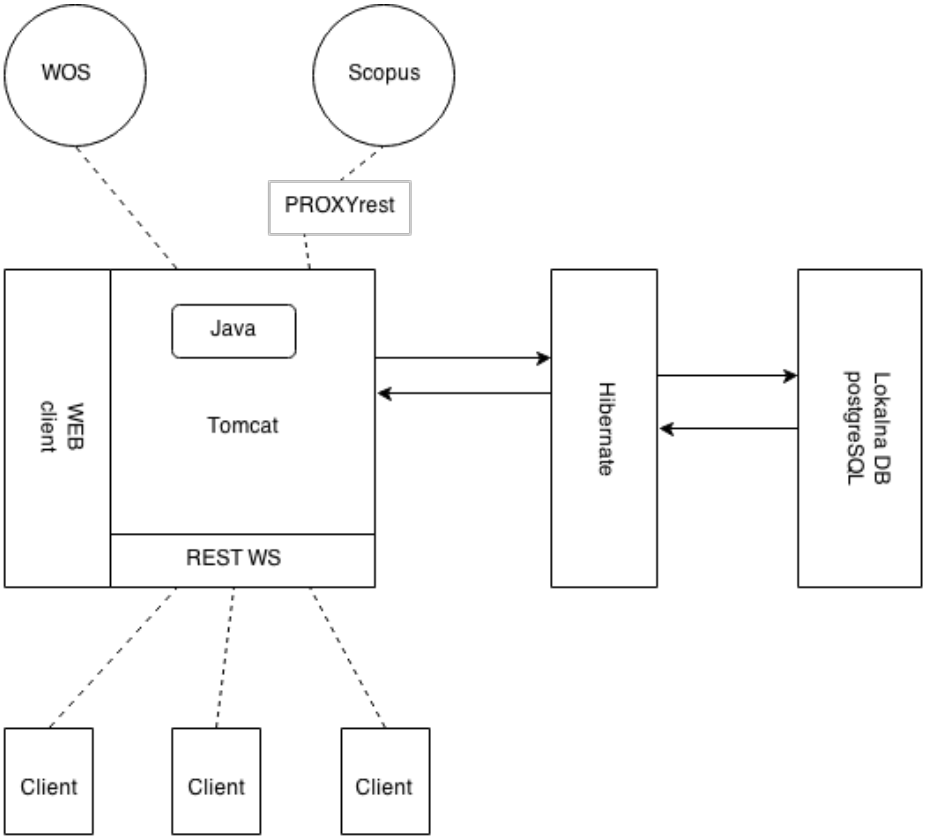
Pre systém sú zadefinované hlavné funkcionálne požiadavky, ktorými sú:

- Webová aplikácia
- Rozšíriteľnosť systému s vlastnou databázou
- Poskytovanie API pre iné systémy
- Jednoduché grafické webové prostredie

Podrobnejšie sú tieto požiadavky rozobrané spolu s opisom architektúry systému.

Ako sme už spomenuli v úvode, systém je navrhovaný ako webová aplikácia v podobe balíčku WAR. Na obrázku nižšie sa nachádza znázornená architektúra systému.

Základný servlet je nasadený na serveri Tomcat verzie 7. Servlet je implementovaný ako služba REST, ktorá komunikuje na základe protokolu HTTP, so štandardnými HTTP metódami GET a POST. Ako lokálna databáza je použitá postgresql, do ktorej sa pristupuje pomocou objektovo relačného mapovača HIBERNATE. Logika servletu je implementovaný v objektovo orientovanom jazyku JAVA, a teda pomocou JPA a HIBERNATE sa vytvárané objekty serializujú do databázy.



Obr. 1.1: Logický model

Prístup do citačných indexov WOS a Scopus je realizovaný štandardne prostredníctvom jednoduchých HttpClient-ov. WOS poskytuje svoje služby prostredníctvom api, ktorá komunikuje na základe SOAP správ v xml formáte(SOAP WS/ wsdl). Rovnako výsledky dopytov vracia v podobe XML správ. Scopus poskytuje svoje api prostredníctvom REST služby a výsledky dopytov vracia vo formáte JSON respektíve XML. Prístup do týchto citačných databáz je možný len z IP adries, ktoré majú k nim povolený prístup. Naša REST aplikácia je nasadená na serveri, ktorý má IP prístup do WOS-u, no pre Scopus je použitá proxyRest služba, ktorej IP adresa má povolený prístup.

WebClient je jednoduché grafické používateľské rozhranie, ktoré umožní vizualizovať výsledky zo systému vo formáte JSON a neskôr aj v zoznamoch. Na obrázku môžeme vidieť, že systém je jednoducho rozšíriteľný ďalšími klientmi, ktorý pristupujú cez REST WS.

Dátové modely

Mapovanie entít:

nasa databaza	WOS	Scopus
nazov_dokumentu	TI - Document title	Title
zdroj (len WOS/SCOPUS)	WOS	Scopus
zdroj_id	UT - Accession number	EID
typ dokumentu	PT - Publication Type*	Document type
autori	AU - Authors	Authors

data - ulozit cely zaznam o dokumente ako text.....pre wos aj scopus *(J=Journal; B=Book; S=Series; P=Patent)

1.1 Globálne ciele

Identifikovali sme nasledovné ciele, ktoré chceme dosiahnuť v rámci riešenia projektu počas zimného semestra:

1. **Oboznámenie sa s problematikou**
   Pred začatím práce na projekt je oblasť projektu pre väčšinu členov tímu neznáma, preto je potrebné sa s ňou oboznámiť. Cieľom je sa oboznámiť s možnosťami získavania informácií z webových citačných databáz WOS a SCOPUS zahŕňajúc spôsob prístupu do databáz a formou v akej budú dáta získané.

2. **Vytvorenie funkčného prototypu aplikácie**
   Cieľom je vytvoriť funkčnú REST služby pomocou jazyka JAVA, ktorá bude poskytovať knižničným systémom sa dopytovať do citačných databáz WOS a SCOPUS. Služba bude umožňovať vyhľadávať buď diela na základe mena autora a názvu diela alebo ohlasy na vybrané dielo na základe identifikátora vybranej databázy. Taktiež chceme vytvoriť jednoduché testovacie rozhranie pre zjednodušenie testovania a možnosti testovania treťou osobou.

3. **Vybudovanie databázy**
   Za účelom obmedzenia potreby prístupov do citačných databáz a za účelom neskoršej práce nad získanými dátami sme sa rozhodli vytvárať si vlastnú, lokálnu databázu zo získavaných dát. V tejto databáze budú uložené deduplikované záznamy diel a ich ohlasy, ktoré sa získajú pri používaní nášho servletu.

4. **Získanie schopnosti pracovať s podpornými nástrojmi pre prácu v tíme**
   Podporné nástroje sú pre prácu v tíme veľmi užitočné a ich ovládanie každým členom tímu je potrebné pre zvýšenie efektivity práce. Aktívnym a správnym používaním týchto nástrojov sa centralizujú informácie ohľadne vykonávaných

činností na projekte a taktiež sa vďaka nim dá vyhnúť závažným rizikám. V našom tíme sme sa rozhodli používať nástroj Jira.

5. **Získanie schopnosti pracovať v tíme**
   Cieľom je aby sa každý člen naučil zásadám spolu práce v tíme a aktívne sa podieľal na práci v tíme. Veľmi dôležité je komunikovať s ostatnými členmi tímu podľa dohodnutých pravidiel a spolu riešiť vzniknuté problémy.

1.2 Celkový pohľad na systém

1.2.1 Architektúra

- systém je z architektonického hľadiska servlet poskytujúci dopytovacie rozhranie pre osoby a iné systémy,
- fungovanie servletu podporuje aplikačný server Apache Tomcat v.7
- metódy rozhrania, ktoré servlet poskytuje pre komunikáciu sú HTTP GET alebo POST kompatibilné
- rozhranie poskytované servletom je REST, prijíma aj vracia správy vo formáte JSON
- triedy servletu sú implementované v jazyku Java s využitím J2EE frameworku

1.2.2 Scenáre dopytov (Workflows)

Vyhľadanie dokumentu

- **Prijatie JSON dopytu s menom autora a názvom dokumentu**
  - dopyt vo formáte JSON:

Listing 1.1: Dopyt vo formáte JSON

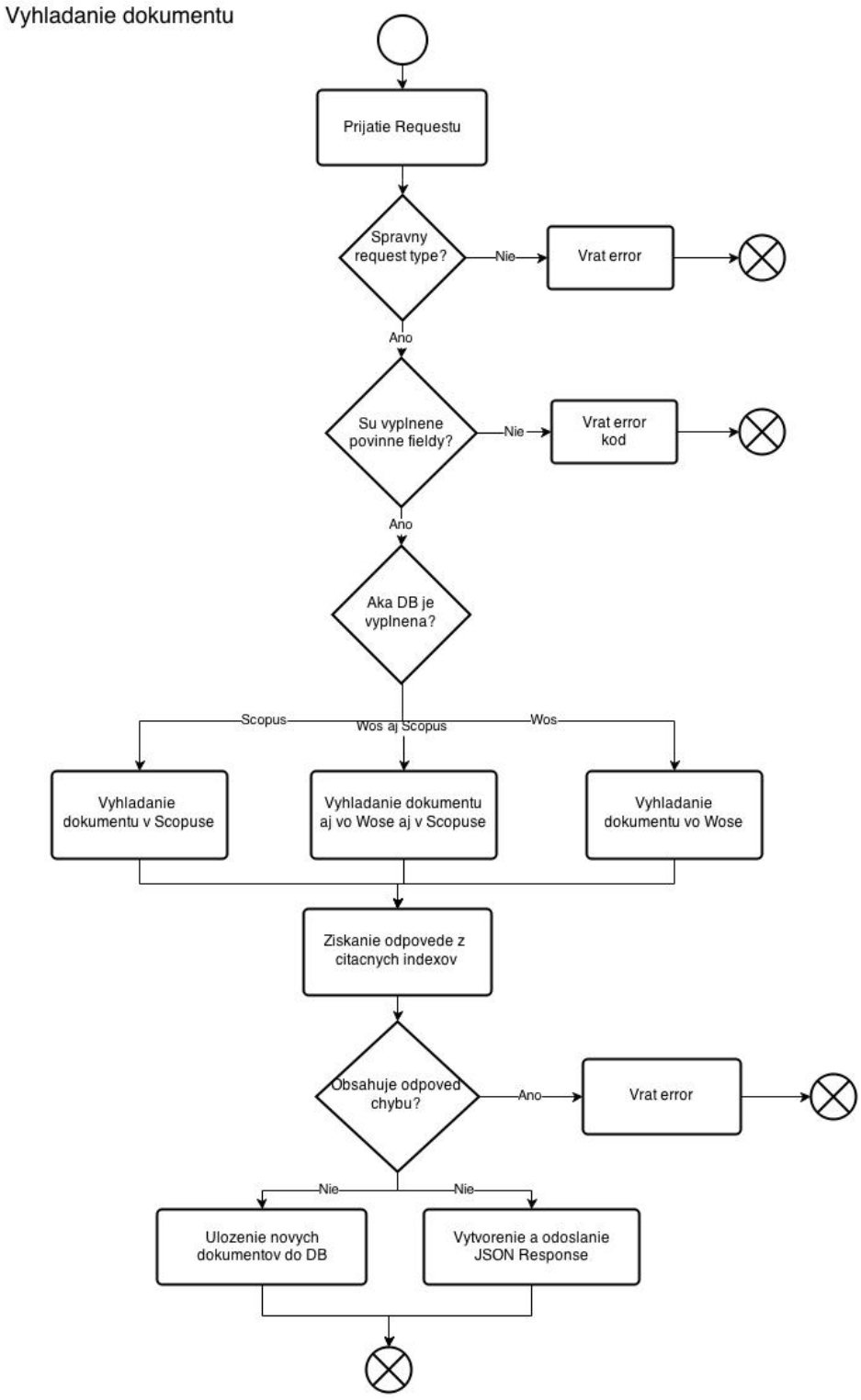
```
{
    "request_type": "search",
    "db": ["wos","scopus"],
    "title": "",
    "author": {
        "name": "",
        "surname": ""
    }
}
```

- **Príjem dopytu**
  - všeobecne pre všetky metódy

- **Správny request type?**
  - všeobecne pre všetky metódy
  - pre vyhľadanie dokumentov musí byť request_type "search"
- **Sú vyplnené povinné polia?**
  - musí byť vyplnený aspoň jeden záznam databázy, v ktorej sa má vyhľadávať
  - musí byť vyplnený autor dokumentu
  - musí byť vyplnený názov dokumentu
  - ak jeden z povinných parametrov chýba, vrátime chybovú hlášku hovoriacu o tom, ktoré parametre nie sú zadané
- **Aký záznam databázy je vyplnený?**
  - ak je vyplnený len SCOPUS, vyhľadávame len v SCOPUSe
  - ak je vyplnený len WOS, vyhľadávame len vo WOSe
  - ak sú vyplnené obe, vyhľadávame v oboch databázach
- **Vyhľadanie dokumentov v citačných indexoch**
  - vyhľadanie dokumentov aktuálne na základe mena autora a názvu dokumentu:
  - sformulovanie dopytu na API citačného indexu
  - odoslanie dopytov
- **Získanie odpovede z citačných indexov**
  - príjem odpovede z citačného indexu
- **Obsahuje odpoveď chybu?**
  - ak odpoveď z citačných indexov obsahuje chybu, vrátime chybu aj my
- **Uloženie nových dokumentov do databázy**
  - prejdeme všetky dokumenty a ak sa niektorý nenachádza v našej databáze (zistíme podľa SCOPUS/WOS ID), uložíme ho do nej
- **Vytvorenie a odoslanie JSON odpovede**
  - odpovede z WOS a SCOPUS sú vo formáte XML, no my ich potrebujeme previesť do formátu JSON a vybrať si štandard
  - odpoveď vo formáte JSON

Listing 1.2: JSON odpoveď

```
{
        "response_type":"search",
    "recordsWosCount": ,
    "searchQueryWos": "",
    "recordsWos":
    [
        {
            "wosID": "",
            "title": "",
            "authors": "",
            "source":""
        },
        {
            "wosID": "",
            "title": "",
            "authors": "",
            "source":""
        }
    ],
    "searchQueryScopus": "",
    "recordsScopusCount": ,
    "recordsScopus":
    [
        {
            "scopusID": "",
            "auhor": "",
            "title": "",
            "source": ""
        },
        {
            "scopusID": "",
            "auhor": "",
            "title": "",
            "source": ""
        }
    ]
}
```



Obr. 1.2: Workflow diagram vyhľadania dokumentu

Vyhľadanie ohlasov na dielo

- **Prijatie JSON dopytu**
  - dopyt vo formáte JSON:

Listing 1.3: Dopyt vo formáte JSON

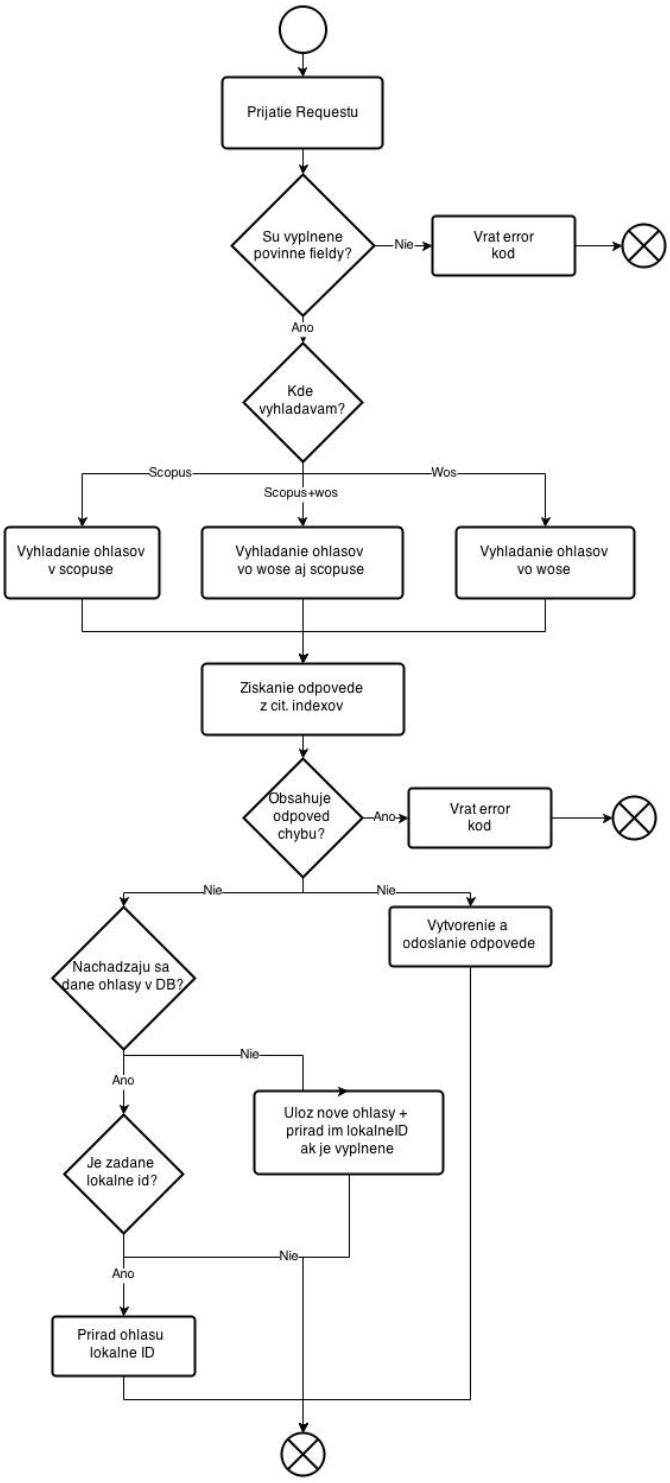
```
{
      "request_type": "citations",
      "idWos": "",
      "idScopus":""  ,
      "institution":"",
      "institutionidrecord":""
}
```

- **Príjem dopytu**
  - kontrola vstupných polí
- **Sú vyplnené povinné polia?**
  - request type pre túto metódu je "citations"
  - musí byť vyplnené buď WOS ID alebo SCOPUS ID
- **Miesto hľadnia**
  - podľa parametrov prebehne vyhľadávanie v jednom alebo v oboch citačných indexoch
- **Vyhľadanie ohlasov**
  - vo WOS sa vyhľadá podľa WOS ID dokumentu
  - SCOPUS nepodporuje vyhľadávanie ohlasov podľa ID, tak sa v ňom vyhľadá podľa iných parametrov, ktoré sa získajú z našej DB alebo po vyhľadaní dokumentu podľa ID v Scopuse
- **Získanie odpovede z citačných indexov**
  - príjem odpovede z citačného indexu
- **Obsahuje odpoveď chybu?**
  - ak odpoveď z citačných indexov obsahuje chybu, vrátime chybu aj my
- **Nachádzajú sa získané ohlasy v databáze?**
  - ak sa niektoré zo získaných záznamov nenachádzajú v databáze, uložia sa tam. Ak obsahoval dopyt aj userID, pridá sa k záznamu

- **Je zadané lokálne ID**
  - ak sa záznam nachádza v lokálnej databáze a nebolo zadané lokálne ID, pokračuje sa ďalej
- **Priradenie ohlasu k lokálnemu ID dokumentu**
  - ak sa záznam nachádza v lokálnej databáze a bolo zadané lokálne ID, priradí sa k danému záznamu
- **Vytvorenie a odoslanie odpovede**
  - odpoveď obsahuje získané ohlasy na dielo

Listing 1.4: JSON odpoveď

```
{
        "response_type":"citations",
        "searchQueryWos": "",
        "recordsWosCount": ,,
        "recordsWos":
        [{
                        "wosID":"",
                        "title":"",
                        "author":"",
                        "source":""
                },
                {
                        "wosID":"",
                        "title":"",
                        "author":"",
                        "source":""
        }],
        "searchQueryScopus": "",
        "recordsScopusCount": ,
        "recordsScopus":
        [{
                        "scopusID": "",
                        "title":"",
                        "authors":"",
                        "source":""
                },
                {
                        "scopusID":"",
                        "title":""
                        "authors":"",
                        "source":""
}]}
```



Obr. 1.3: Workflow diagram vyhľadania ohlasov na dielo

Uloženie ID dopytovateľa k lokálnemu záznamu

- **Prijatie JSON dopytu**
  - dopyt vo formáte JSON:

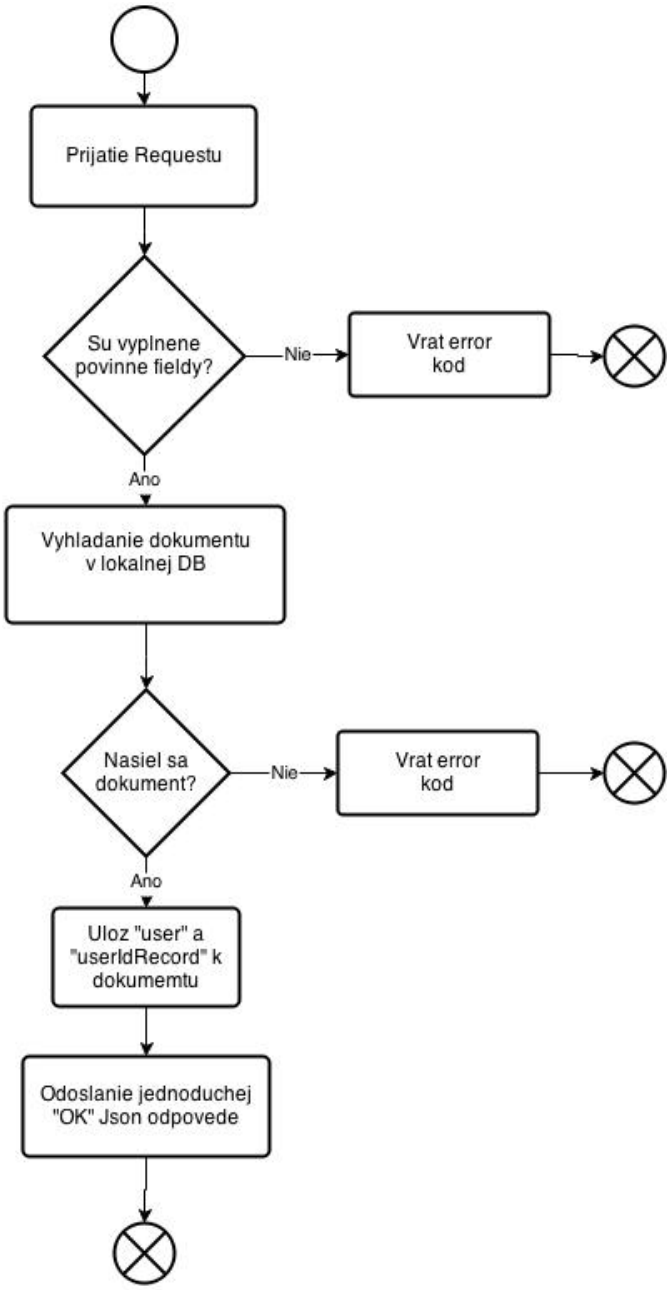
Listing 1.5: Dopyt vo formáte JSON

```
{
        "request_type": "addUserId",
        "institution": "",
        "institutionidrecord": "",
        "idWos": "",
        "idScopus": "",
}
```

- **Sú vyplnené povinné polia?**
  - request type pre túto metódu musí byť äddUserID"
  - musí byť vyplnený dopytovateľ - skratka inštitúcie
  - musí byť vyplnené institutionIdRecord - lokálne ID diela v rámci dopytujúcej sa inštitúcie
  - musí byť vyplnené WOS alebo SCOPUS ID
- **Vyhľadanie dokumentu v lokálnej databáze**
  - vyhľadanie na základe WOS a SCOPUS ID z dopytu
  - ak sú vyplnené obe ID, vyhľadanie záznamu dokumentu z oboch citačných indexoch
  - ak záznam nebol nájdený, vrátime chybovú hlášku a neskôr záznam dohľadáme v citačných indexoch
- **Uloženie používateľa a userIdRecord k dokumentu**
  - používateľom je napríklad STU
  - institutionIdRecord je napríklad STU ID dokumentu
- **Odoslanie odpovede**
  - jednoduchá potvrdzujúca odpoveď - OK

Listing 1.6: JSON odpoveď

```
{
    "status": "", "1-ok" alebo "2-err"
}
```

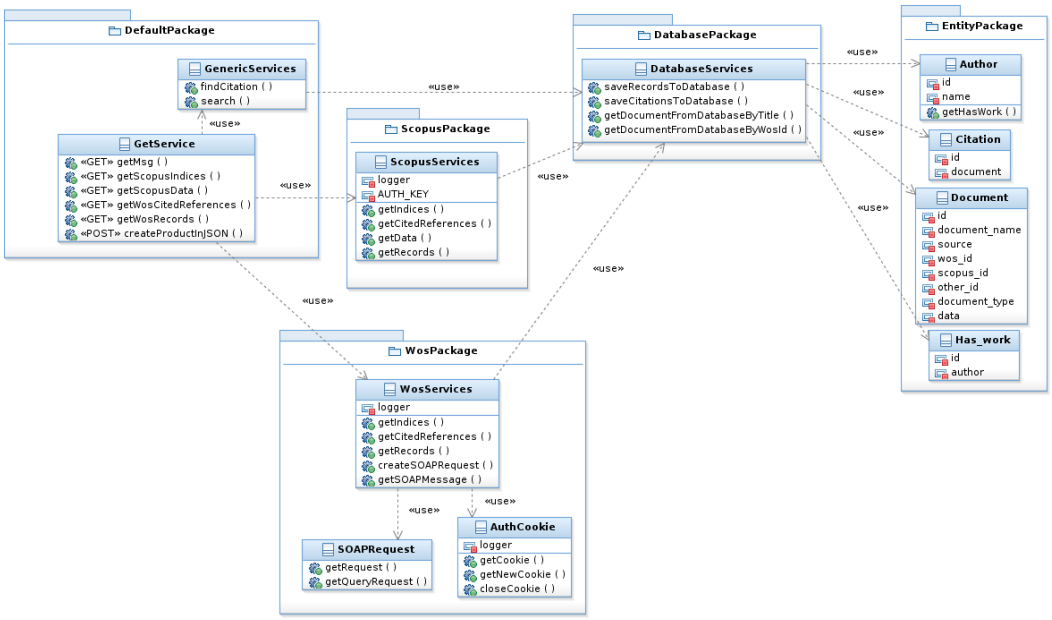


Obr. 1.4: Workflow diagram uloženia ID dopytovateľa k lokálnemu záznamu

1.2.3 Diagram tried

Diagram na nasledujúcom obrázku ilustruje rozdelenie funkcionality vytvorenej aplikácie do tried. Triedy sú rozdelené na základe funkčných aj doménových závislostí. Dospeli sme tak ku stavu, kedy:

- databázová logika sa nachádza v samostatnom balíčku. Všetok kód týkajúci sa databázy a mapovania entít je tak sústredený na jednom mieste a je tak možné jednoducho vymeniť použitú databázu - dosiahli sme abstrakciu od použitej databázy
- metódy rozhrania, ktoré sú typu GET a POST a sú prístupné z vonka servletu sú taktiež oddelené v osobitnom balíčku. Je tak možné meniť za nimi stojacu implementáciu bez zmeny nimi poskytovaného rozhrania pre vonkajšie aplikácie.
- keďže každý z dvoch nami použitých citačných indexov komunikuje iným spôsobom (XML správy skrz SOAP vs. JSON správy skrz REST API), oddelili sme do samostatných balíčkov aj metódy špecifické pre jednotlivé domény. Triedy komunikujúce s rozhraním SCOPUS indexu tak nemusia napríklad implementovať mechanizmy pre vytváranie SOAP správ.



Obr. 1.5: Diagram tried systému

Kapitola 2

Moduly systému

2.1 Analýza

Z požiadaviek zákazníka vyplynulo, že cieľom projektu je vytvoriť webovú aplikáciu, ktorá bude schopná pristupovať k citačným databázam (minimálne k WOS a SCOPUS). Zároveň musí byť schopná ukladať vyhľadávané dáta a získavané výsledky do lokálnej databázy. Systém musí poskytovať rozhranie REST (Representational state transfer) pre používanie implementovanej funkcionality.

Vychádzajúc z vyššie uvedeného stručného opisu požiadaviek zákazníka sme pristúpili k analýze vhodnosti technológií a systémov, ktoré sme plánovali v rámci projektu použiť.

V tejto časti poskytujeme prehľad o jednotlivých fázach tejto analýzy.

2.1.1 Server

Systém (webová aplikácia) musí byť nasadený na server. Keďže hlavným implementačným jazykom použitým pre backend systému je JAVA, je potrebné použiť server, ktorý podporuje jazyk JAVA. V rámci analýzy sme zvažovali servery Tomcat a JBOSS AS 7.

Webovú aplikáciu vo forme WAR balíčka je možné nasadiť rovnako na JBOSS, aj na Tomcat. Pre WAR aplikácie, nie je potrebný JBOSS AS 7, ktorý je používaný skôr pre väčšie Enterprise aplikácie, ktoré potrebujú väčší výkon (napríklad EAR súbory). Pre potreby nami vyvíjaného servletu teda úplne postačuje server Tomcat.

2.1.2 REST rozhranie

Aby mohol server komunikovať s klientom a zároveň poskytovať rozhranie na komunikáciu, je potrebné nájsť a zadefinovať spôsob takejto komunikácie. Na základe požiadaviek zákazníka bolo zvolené pre komunikáciu REST rozhranie, ktoré je založené na HTTP. REST využíva základné metódy GET a POST.

V implementácií je potom typ každej konkrétnej metódy dafinovaný pomocou anotácií:

- @GET
- @POST

Zároveň je potrebné zadefinovať „cestu“, na ktorej môže používateľ alebo iný systém volať túto metódu.

- @Path(”cesta”)

Zadanie http linky http://server/cesta prípadne http://server/aplikacia/cesta následne na serveri vyvolá vykonanie konkrétnej metóda na uvedenej ceste a navrátenie výsledku, ktorý môže byť v rôznom formáte. (Response.status(200).entity(„params“).build();)

2.1.3 Formát pre výmenu dát

Ak predpokladáme, že existuje rozhranie, pomocou ktorého systém a klient komunikujú, je potrebné navyše zadefinovať „jazyk“ respektíve formát v ktorom budú komunikovať. V tejto časti analyzujeme dva najzákladnejšie formáty používane na výmenu dát cez internet. Sú nimi JSON a XML.

2.1.4 JSON

JSON je spôsob reprezentácie dát, ktorý je nezávislý na platforme. Umožňuje používanie polí alebo agregovaných dát. Vstupom môže byť akýkoľvek typ – boolean, čislo, string alebo pole. JSON je jednoduchý a rýchly spôsob na komunikáciu, ktorý je vo formáte kľúč – hodnota.

Nižšie je uvedený jednoduchý príklad dát vo formáte JSON (Zdroj: *w3schools.com*):

Listing 2.1: Príklad použitia formátu JSON

```
{"employees":[
    {"firstName":"John", "lastName":"Doe"},
    {"firstName":"Anna", "lastName":"Smith"},
    {"firstName":"Peter", "lastName":"Jones"}
]}
```

2.1.5 XML

XML je značkovací jazyk štandardizovaný konzorciom W3C. Umožňuje jednoduché vytváranie a reprezentáciu dát určených na prenos. Princíp značkovania, ktorý používa, je známy napríklad zo syntaxe jazyka HTML, ktorý je podmnožinou jazyka XML. XML musí byť rovnako ako JSON správne štruktúrované a na rozdiel od JSON je potrebné aby každý element mal začiatočnú a koncovú značku.

Nižšie je uvedená obdoba predchádzajúceho JSON príkladu vo formáte XML (Zdroj: *w3schools.com*):

Listing 2.2: Príklad použitia formátu XML

```
<employees>
    <employee>
        <firstName>John</firstName> <lastName>Doe</lastName>
    </employee>
    <employee>
        <firstName>Anna</firstName> <lastName>Smith</lastName>
    </employee>
    <employee>
        <firstName>Peter</firstName> <lastName>Jones</lastName>
    </employee>
</employees>
```

Pri porovnaní príkladov 2.1 a 2.2 vidíme, že formát JSON sa na prvý pohľad javí čitateľnejší. Keďže vymieňanie správ vo formáte JSON je navyše príznačné pre systémy poskytujúce REST rozhranie, rozhodli sme sa pre tento formát správ aj my.

2.1.6 Citačné databázy a ich API

V tejto časti analyzujeme základné citačné databázy, z ktorých bude vytváraný systém získavať dáta.

2.1.6.1 Web of Science

Web of Science (ďalej už len WOS) je služba prevádzkovaná spoločnosťou Thomson Reuters. Pre nás je dôležité, že poskytuje možnosť získavať dáta z ich databázy pomocou protokolu SOAP (Simple Object Access Protocol) pre transfer požiadaviek a odpovedí. Pre používanie ich API je však potrebné:

- Vlastniť IP adresu, z ktorej je povolený prístup k službám
- Vlastniť Cookie pre prístup k API

Získavanie cookie je možné pomocou SOAP dopytu opísaného vo WSDL (Web Service Definition Language), konkrétne služba s názvom $WOKMWSAuthenticate?wsdl$. Dopyt pre získanie cookie (zdroj: Dokumentácia wos – web guide):

Listing 2.3: SOAP cookie request

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:auth="http://auth.cxf.wokmws.thomsonreuters.com">
<soapenv:Header/>
<soapenv:Body>
<auth:authenticate/>
</soapenv:Body>
</soapenv:Envelope>
```

Následne získame odpoveď s novou cookie, ktorou je potrebné sa identifikovať pri každom ďalšom dopyte – napríklad pri hľadaní diel alebo citácií.

Listing 2.4: SOAP odpoveď na cookie request

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Body>
<ns2:authenticateResponse xmlns:ns2="http://auth.cxf.wokmws.thomsonreuters.com">
<return>ziskane COOKIE</return>
</ns2:authenticateResponse>
</soap:Body>
</soap:Envelope>
```

Je preto potrebné do ďalších dopytov pridať nasledujúcu hlavičku:

- Set-Cookie=SID= ziskanaCOOKIE

Vyhľadávanie pomocou služby WSDL: WokSearch?wsdl

Pri vyhľadávaní je možné používať boolovskú algebru použitím kľúčových slov OR, AND a pod. Podrobnejšie informácie je možné získať z dokumentácie WOS web services Premium guide.

Niektoré z polí, ktoré môžu byť užitočné pri získavaní dát z WOS API:

- AU=Author
- PY=Year Published
- AI=Author Identifiers
- TI=Title
- PD=Publication Date
- IS=ISSN/ISBN
- RID=ResearcherID
- ED=Editor
- TS=Topic

Odpoveď pri prehľadávaní je vo formáte XML. Bude teda potrebné použiť vhodný XML parser, pomocou ktorého takéto dáta vydolujeme.

2.1.6.2 SCOPUS

Pre používanie Scopus API musíme vlastniť IP adresu, ktorá má k nemu povolený prístup. Server, na ktorom bude naša služba nasadená nemá povolenú IP adresu. Musíme teda použiť proxy službu poskytovanú na adrese: https://library.sk/csp/arl1/util.proxy.ProxyRest.cls. V tomto prípade sa zadefinuje query a iné parametre.

Príklad dopytu do Scopus API pomocou proxyRest:

Listing 2.5: Príklad vyhľadávania v SCOPUSe cey proxyRest

```
https://library.sk/csp/arl1/util.proxy.ProxyRest.cls?server=
https://api.elsevier.com&path=content/search/index:SCOPUS
&params=query=REF((REFTITLE(DieloTitle))AND
(REFSRCTITLE(publikaciaTitle)))AND(REF(AutorMeno))
&=COMPLETE&key=PRISUPKOD
```

Jednotlivé parametre z príkladu 2.5:

- DieloTitle – dielo pre ktoré sa vyhľadajú citácie (REF – funkcia definovaná v Scopus dokumentácií)
- AutorMeno – meno Autora v Tvare „priezvisko Meno“ – Scopus API umožňuje hľadat len podľa 1. písmena mena
- PRISTUPKOD – prístupový kód pre používanie proxyRest

Dôležité je posielať všetko za params ako reťazec kódovaný pre použitie v URL. To znamená, že nie je možné poslať dopyt obsahujúci medzery alebo iné znaky, ktoré nie je možné jednoducho dekódovať.

API Scopus rovnako ako WOS API vracia výsledky dopytov vo formáte XML. Príklad takéhoto výstupu (získaneho z Scopus API cez ProxyRest) môžeme vidieť nižšie ako zjednodušenú štruktúru:

Listing 2.6: Odpoveď SCOPUSu na požiadavku vyhľadávania diel

```
<feed xmlns="http://www.w3.org/2005/Atom" xmlns:atom="ht...." >
<title>...</title>
<author>...</author>
<entry>...</entry>
<entry>...</entry>
</feed>
```

Výsledky vyhľadávania 2.6 sa nachádzajú medzi značkami <entry>.

2.2 Návrh

Po analýze možných riešení a preskúmaní vhodnosti použitia konkrétnych technológií sme pristúpili k návrhu architektúry. Počas celého procesu návrhu sme dbali na modularitu výsledného systému, udržiavateľnosť a čitateľnosť kódu a hlavne použitie čo možno najoptimálnejších riešení z hľadiska výkonnosti. Nasledujúca kapitola prezentuje návrhové rozhodnutia, ktoré boli vykonané v priebehu procesu návrhu, rozdelené do podkapitol podľa príslušnosti k niektorému z logických celkov aplikácie.

2.2.1 Servlet

Funkcionalita zodpovedná za beh samotného servletu je izolovaná v samostatnej časti aplikácie. Jedná sa o štandardný JAVA servlet. Obsahuje tým pádom direktívy,

ktoré mu predpisuje konvencia aplikácií spúšťaných na aplikačnom serveri Apache Tomcat. Jeho špecifická časť, ktoú priniesla povaha našej aplikácie je načítavanie konfiguračných hodnôt pri štarte servletu. V hlavnom priečinku aplikácie je vytvorený konfiguračný súbor, v ktorom sú uložené hodnoty používané naprieč aplikáciou - konštanty, kľúče a autentifikačné údaje.

2.2.2 Databáza

2.2.2.1 ORM

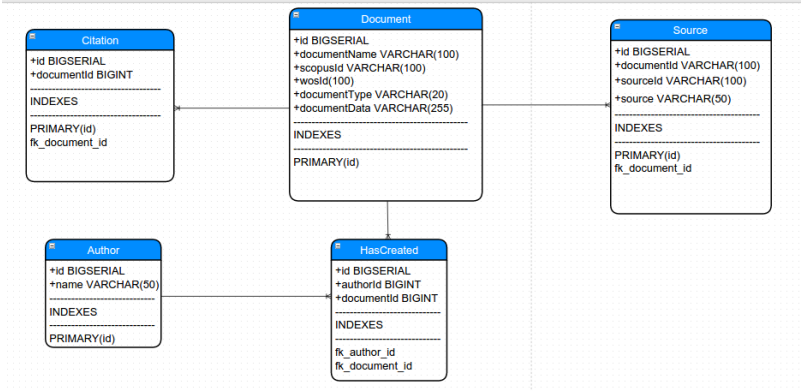
Funkcionalita riadiaca komunikáciu s databázou bola taktiež navrhnutá tak, aby stála ako samostatná časť, obaľujúca databázové SQL tabuľky do objektov, pomocou ktorých možno získavať konkrétne údaje. Štruktúra navrhnutých entít zodpovedá konvenciám požadovaným aplikačným rámcom JAVA Hibernate. Takto navrhnuté objektovo-relačné mapovanie je ďalej udržiavané na jednom mieste a naprieč aplikáciou používané v jednotnom tvare.

2.2.2.2 SQL tabuľky

Pri vytváraní logického modelu sme v prvej fáze riešenia projektu skonštatovali, že bude potrebné reprezentovať nasledujúce entity:

- Dokument
- Autor
- Inštitúcia
- Citácia
- Autorstvo

Prepojenie entít reprezentuje fyzický model vytvorený počas návrhu databázovej schémy:



Obr. 2.1: Fyzický model

2.2.3 Vyhľadávanie

Vytvorená aplikácia poskytuje jednotné rozhranie pre vyhľadávanie nad rôznymi citačnými indexami. Od začiatku návrhu jej fungovania bolo tak nutné myslieť na to, aby tento model bol rozšíriteľný. Štruktúra vyhľadávania bola tým pádom rozdelená do štyroch krokov, spracovávaných v troch oblastiach aplikácie. Sú nimi:

- **REST rozhranie** - osamostatnené triedy, tvoriace bránu do aplikácie, ktoré poskytujú re komunikáciu rozhranie podľa konvencií REST
- **Spracovanie požiadavky a vrátenie odpovede** - triedy spracovávajúce vstup a vyvolávajúce vyhľadávanie nad všetkými citačnými indexami, s ktorými je aplikácia momentálne schopná pracovať
- **Špecifické vyhľadávania** - samostatné implementácie vyhľadávaní nad konkrétnymi citačnými indexami, spĺňajúce konvencie vyžadované konkrétnym indexom

2.2.4 Výnimky

Počas vyhľadávania záznamov môže nastať hneď niekoľko situácií, kedy dopyt nedostane adekvátnu odpoveď. Zlyhanie môže nastať na strane citačných indexov alebo môže vzniknúť chyba vo vykonávaní v rámci aplikácie samotnej. Žiadne z týchto zlyhaní však nemôže mať za následok situáciu, kedy používateľ neobrží nijakú odpoveď. V takýchto situáciách aplikácia pracia chybovú odpoveď označenú kódom reflektujúcim štandardy HTTP aj čísla kódov používané citačnými indexami.

2.2.5 Testovacie rozhranie

Poslednou navrhovanou časťou aplikácie bolo webové rozhranie používané interne pre testovanie funkčnosti vytváraného servletu. Aby sme sa vyhli vytváraniu ďalšej hierarchie tried nad už existujúcim dátovým modelom, poskytujúcim aplikačné rozhranie, rozhodli sme sa využiť model SPA (Single Page Application), ktorý zavádza webový rámec pre jazyk JavaScript AngularJS. Navrhnuté rozhranie definuje iba vlastnú prezentačnú vrstvu. Prepínanie medzi jeho časťami riadi minimalistická aplikačná vrstva a ako dátová vrstva slúžia metódy servletu obalené do rozhrania služby pre získavanie dát.

2.3 Implementácia

Systém máme rozdelený do viacerých balíkov. Náš základný balík je sk.fiit.team15. Projekt ďalej obsahuje nasledujúce balíky:

- sk.fiit.team15.database
- sk.fiit.team15.engine
- sk.fiit.team15.entity
- sk.fiit.team15.exception
- sk.fiit.team15.exception.our
- sk.fiit.team15.exception.WoS
- sk.fiit.team15.exception.scopus
- sk.fiit.team15.scopus
- sk.fiit.team15.util
- sk.fiit.team15.WoS

2.3.1 Základný balík

V tomto balíku je obsiahnutá funkcionalita celého servletu. Obsahuje triedy:

- Configuration
- GenericServices
- GetService
- InitializationClass

2.3.1.1 Trieda Configuration

Slúži na načítanie konfigurácie zo súboru „config.properties“, rovnako do tohto súboru konfiguráciu zapisuje. Vykonáva sa to v metódach „readConfigurationEntries“ a „setConfigurationEntries“. Ďalej obsahuje metódu „getConfiguration“, ktorá vracia singleton objekt a metódu „getConfigurationEntry“, ktorá vracia jednu konkrétnu položku.

2.3.1.2 Trieda GenericServices

Obsahuje metódy na vyhľadávanie diel a citácií. Ďalej metódy na ošetrenie vstupov. Vyhľadanie diel sa vykonáva metódou „String search(JSONObject jsonObject)“. Táto metóda zoberie vstupný JSON objekt, overí štruktúru jeho dát a následne ho pošle službám vyhľadávajúcim v systéme WoS a Scopus. Nakoniec výsledky od týchto služieb znova poskladá do JSON objektu a ten vráti vo formáte String. Rovnako funguje aj metóda na vyhľadanie citácií „String findCitation(JSONObject jsonObject)“.

JSON na vyhľadanie diel má štruktúru:

Listing 2.7: Štruktúra JSONu používaného na vyhľadanie diel

```
{
        "request_type": "search",
        "db": ["WoS","scopus"],
        "title": "",
        "author": {
            "name": "",
            "surname": ""
        },
        "auth_token": ""
}
```

JSON na vyhľadanie citácií má štruktúru:

Listing 2.8: Štruktúra JSONu používaného na vyhľadanie citácií

```
{
        "request_type":"citations",
        "idWoS": "",
        "idScopus":""  ,
        "institution":"",
        "institutionidrecord":"STU",
        "auth_token":"valid"
}
```

Opis jednotlivých atribútov JSON dopytu pre vyhľadanie diel:

- request_type - definuje o aký druh požiadavky sa jedná
- db - definuje nad akými citačnými databázami sa bude dopyť vykonávať
- title - meno diela
- author - meno autora
- auth_token - definuje, či je používateľ oprávnený využívať službu
- idWoS - id diela v databáze WoS
- idScopus - id diela v databáze Scopus

- institution - id diela v databáze používateľa
- institutionidrecord - názov inštitúcie, z ktorej bol dopyt odoslaný

2.3.1.3 Trieda GetService

Obsahuje služby, ktoré môžu byť volané priamo pomocou webovej adresy. Sú identifikované pomocou dvoch anotácií. Jedna je @GET alebo @POST a určuje aký je typ vstupu. Druhá @Path(„názov“) určuje názov webovej služby. Samotná trieda má tiež anotáciu @Path, konkrétne @Path(„/get“). Takže výsledná adresa na kolanie konkrétnej služby vyzerá napríklad takto: http://127.0.0.1:8080/TP_team15/rest/get/jsonExample

V prvotnej fáze projektu sme implementovali viaceré GET metódy, ktoré sme neskôr nahradili jednou POST metódou, konkrétne metódou „createProductInJSON“, ktorá má anotáciu @Path("jsonExample"). Jej vstupom je String obsahujúci vstupný JSON objekt. Metóda podľa „request_typ“ zavolá vhodnú generickú službu. Nakoniec získany JSON vráti vo formáte http response.

2.3.1.4 Trieda InitializationClass

Táto trieda slúži na konfiguráciu servletu pred jeho štartom, prípadne jeho konfiguráciu pred ukončením. Implementuje „ServletContextListener“, kde prepisuje metódy „contextInitialized“ a „contextDestroyed“. Pri inicializácií načítava konfiguráciu, inicializuje hibernate a získava autentifikáciu.

2.3.2 Balík sk.fiit.team15.database

Tento balík obsahuje jedinú triedu DatabaseServices. Neobsahuje žiaden ďalší balík.

2.3.2.1 Trieda DatabaseServices

Táto trieda slúži na prácu s databázou a využíva na to Hibernate framework. Obsahuje viacero metód, ktoré sa dajú rozdeliť podľa návratového typu.

Prvé sú metódy bez návratového typu, čiže void a slúžia na ukladanie do databázy. Sú to:

- saveSourceToDatabase – ukladá zdroj. Vstupné parametre má:
  - sourceId - ID dokumentu v databaze zdroja dopytu
  - d - dokument na ktory sa zdroj pyta
- saveRecordsToDatabase – ukladá jedno dielo. Vstupné parametre má:
  - title - nazov diela
  - WoSId - ID diela vo WoSe
  - scopusId - ID diela v scopuse

  - documentType - typ dokumentu
  - data - ostatne data ktoé sme neparsovali
  - name - meno autorov

- saveCitationsToDatabase – ukladá citácie pre dané dielo. Vstupné parametre má:
  - d - dokument, pre ktorý sa hľadali citácie
  - newDocument - ohlas

Ďalej sú metódy, ktoré vracajú typ Document, tento typ je definovaný v balíčku sk.fiit.team15.entity. Sú to:

- getDocumentFromDatabaseByScopusId
- getDocumentFromDatabaseByWoSId

Obe tieto metódy vracajú dielo z našej databázy nájdené podľa vstupného parametra, ktorý je String. Pri prvej metóde to je scopusId a pri druhej WoSId.

Trieda obsahuje aj metódy metódy, ktoré vracujú typ boolean:

- isWoSDocumentInLocalDatabase – overuje či sa dokument s WoSId zadaným ako String parameter nachádza v našej databáze
- isScopusDocumentInLocalDatabase - overuje či sa dokument s scopusId zadaným ako String parameter nachádza v našej databáze
- checkIfCitacionIsAlreadyInOurDatabase – overuje či sa zadaná citácia nachádza v našej databáze. Citácia je zadaná pomocou 2 vstupných parametrov typu Document, rovnakých ako pri ukladaní citácie

Poslednou metódou je $getAuthorNameFromDatabaseByDocumentId$. Táto metóda vracia meno autora pre daný dokument. Typ návratovej hodnoty je String. Vstupný parameter je documentIt typu int.

2.3.3 Balík sk.fiit.team15.engine

Balík engine obsahuje tri triedy. Patria sem Authentication, Log4j, Timer.

2.3.3.1 Trieda Authentication

Táto trieda slúži na autentifikáciu zákazníkov v našom servlete, keďže možnosť využívať túto aplikáciu nebude mať každý. Na začiatku je implementovaný návrhový vzor singleton pre konfiguráciu.

Metódy v tejto triede sú:

- getAutheticatedUser - Metóda vracia singleton object. Ak objekt nie je vytvorený tak ho vytvorí a vráti.
- readConfigurationEntries - Metóda, ktorá číta hodnoty z konfiguračného súboru do premennej triedy. Metóda vracia typ boolean. Jej výstupom je rozhodnutie o tom, či sa singleton objekt vytvoril alebo nie.
- printAutheticaticatedUsers - Táto metóda tiež vracia typ boolean. Vytvára HashMapu pre udržiavanie zákaznikov.
- getConfigurationEntry – metóda vracia hodnotu konfiguračnej premennej, ktorá je typu String
- isAutheticatedUser – táto metóda slúži na vyhadzovanie výnimiek, či je potrebná autentifikácia a či autentifikácia nevypršala

2.3.3.2 Trieda Log4j

Trieda ktorá slúži na konfiguráciu pre logovanie v našom servlete. Obsahuje jednu metódu a tou je configure, ktorá je typu void. V nej sa volá konfigurátor, ktorý používa konfiguračný súbor log4j.xml.

2.3.3.3 Trieda Timer

Táto trieda vytvára objekt Timer. Je to Singleton objekt. Obsahuje premenné:

- instance typu Timer
- actTime typu long
- cookie typu String

2.3.4 Balík sk.fiit.team15.entity

Táto trieda obsahuje triedy, ktoré boli implementované ako entity v databáze s atribútmi, ktoré si do databázy ukladáme. Ide o triedy:

- Author
  - id – obsahuje získané ID autora
  - name – meno autora
  - vytvorí sa väzba medzi autorom a dielom
- Citation
  - id – je to primárny kľuč
  - citationId – ID citácie

  - vytvorí sa väzba na dokument, ktorý je citovaný

- Document
  - id – generovaná premenná ID dokumentu
  - documentName – názov dokumentu
  - WoSId – typu string, ide o ID, pod ktorým je dokument uložený v databáze WoS
  - scopusId – typu string, ide o ID, pod ktorým je dokument uložený v databáze Scopus
  - documentType – opäť stringová premenná, ktorá nám označuje či je dielo v databáze kniha, clanok a pod.
  - documentData – ostatné dáta dokumentu uložené v jednom reťazci
  - sourceTitle – zdrojový názov dokumentu diela, ktorý je takisto typu string

- HasCreated
  - je to väzobná entita medzi entitami Document a Autor
  - id – generované ID dvojíc Autor-Dielo
  - obsahuje ID autora prepojeného s dokumentom
  - obsahuje ID dokumentu prepojeného s autorom

- Source
  - id – generované ID primary key
  - sourceId – typu string a obsahuje ID zdroja
  - vytvorí sa väzba medzi zdrojom a dokumentom

2.3.5 Balíky výnimiek

Implementovali sme 4 balíčky výnimiek. Sú to balíčky exception, exception.our, exception.scopus a exception.WoS.

2.3.5.1 Balík sk.fiit.team15.exception

Obsahuje všetky výnimky, ktoré sú uložené v súbore v o formáte JSON. Takisto obsahuje triedu pre všetky nami definované výnimky. Načíta sa správa zo súboru, upraví sa, načíta sa chybová správa zo súboru podľa čísla chyby.

Príklad chybového JSONu:

Listing 2.9: Príklad chybového JSONu

```
{"number":101, "message":"Not found any document"}
```

Chybový JSON 2.9 obsahuje číslo chyby a správu o chybe.

2.3.5.2 Balík sk.fiit.team15.exception.our

Obsahuje triedy pre výnimky v našom servlete.

Sú tu napríklad triedy:

- OurExceptionAuthenticationExpired – nie je zadaný auth_token alebo je nesprávny
- OurExceptionEmptyTitleAndSurnameFields – nie je vyplnené pole pre názov dokumentu alebo meno autora
- OurExceptionInvalidRequestTypeField – prázdne alebo zlé vyplnené pole request_type, ktoré slúži na identifikáciu, či chceme vyhľadať dielo alebo chceme vyhľadať ohlasy
- OurExceptionInvalidTitleField – prázdne alebo zlé vyplnené pole title, pre názov diela
- OurExceptionNoResultScopus – nenájdené žiadne dielo v databáze Scopus
- OurExceptionNoResultWoS – nenájdené žiadne dielo v databáze WoS

2.3.5.3 Balík sk.fiit.team15.exception.scopus

Obsahuje triedy pre výnimky, ktoré sme dostali z databázy Scopus.

Triedy v tomto balíku:

- ScopusExceptionAuthorizationError – chýba alebo je neplatný auth_token alebo APIkey, nastala chyba pri autentifikovaní používateľa
- ScopusExceptionInvalidInput – zlé zadaná dotaz alebo neplatná http metóda alebo nepatný mime typ
- ScopusExceptionQuotaExceeded – prekročený počet kvót pre API
- ScopusExceptionSystemError – chyba v jadre služby elsevire

2.3.5.4 Balík sk.fiit.team15.exception.WoS

V tomto balíku sa nachádzajú triedy pre výnimky z WoSu.

Sú to napríklad triedy:

- WoSExceptionEmptyDatabaseId – prázdny databaseID element – prázdny string
- WoSExceptionInvalidDatabaseId – zadané databaseID je neplatné
- WoSExceptionTimeSpanConflict – timeSpan a symbolicTimeSpan sú vyplnené súčasne
- WoSExceptionNilDatabaseId – prázdny databseID element

2.3.6 Balík sk.fiit.team15.scopus

V tomto balíčku sa nachádza jedna trieda, a tou je ScopusServices, v ktorej sú implementované metódy na vyhľadanie citácií a diel. Táto trieda obsahuje služby scopusu na vyhľadanie diel a ohlasov na dielo.

Obsahuje metódy:

- getCitedReferences – Metóda ktorá vracia ohlasy na dielo. Oba vstupné parametre sú typu jsonObject.
  - vstupuje jsonObject – obsahuje atribúty, na základe ktorých sa vyhľadávajú ohlasy 2.7
  - vstupuje object – vytvorí sa query podľa predpisu Scopus API a odošle sa službe pomocou tohto parametra
  - výstupom jsonRecords – zoznam objektov typu JSON, ktoré obsahujú hľadanými ohlasy
- getScopusSearchDataFromScopusById – metóda na vyhľadanie diel v Scopuse podľa ID zo Scopusu
  - vstupuje scopusId – Scopus ID sa upraví na také, aby sa dalo vyhľadať v Scopuse
  - následne sa kóduje podľa UTF8 a posiela sa url pomnocou http klienta do služby Scopusu pre vyhľadanie diel
  - vytvorí sa JSONObject s požadovanými atribútmi, ktoré chceme získať
  - vystupuje jsonRecords – list JSON objektov, ktorý obsahuje zoznam diel zo scopusu nájdených podľa ID scopusu
- getRecords – metóda na vyhľadanie diel v databáze Scopus podľa názvu diela a mena autora
  - vstupuje jsonObject a object kde sú oba typu JSONObject
  - na začiatku sa kontroluje, či je vyplnený názov diela, ak áno treba prípadne zátvorky v názve diela upraviť, lebo API Scopusu ich berie ako špeciálny znak vo svojom query
  - potom kontrolujeme, či je vyplnené alebo vynechané
  - ak vstupný JSON obsahuje názov diela pridáme ho do query a vytvorené query podľa API Scopusu kódujeme podľa UTF-8
  - pošle sa url adresa aj s kódovaným query na službu Scopusu a tá nám vráti odpoveď so záznamami
  - vystupuje jsonRecords – osahuje zoznam diel zo Scopusu vyhľadaných podľa názvu diela a mena autora

2.3.7 Balík sk.fiit.team15.util

V tomto balíku sa nachádza trieda HibernateUtil, v ktorej je implementované vytváranie transakcii s databazou.

2.3.8 Balík sk.fiit.team15.WoS

V tomto balíku sú implementované služby WoSu na vyhľadanie diel a ohlasov na diela. Okrem triedy WoSServices obsahuje aj triedy AuthCookie a SOAPRequest.

2.3.8.1 Trieda AuthCookie

Táto trieda obsahuje implementáciu vytvorenia cookie session pre spojenie s WoS API cez WSDL, nastavenia cookie a uzatvorenia session.

Obsahuje nasledujúce metódy:

- getNewCookie – vytvorenie cookie session, ktoré je typu string, pre spojenie s WoS API cez WSDL
- getCookie – nastavenie cookie pre jej odoslanim v URL, vracia inicializovanú cookie typu string
- closeCookie – metóda volaná po expirácií aktuálnej cookie, ktorá uzatvára aktuálnu session

2.3.8.2 Trieda WoSServices

Trieda obsahujúca metódy na vyhľadanie indexov vo WoSe, vyhľadanie diel v databáze a vyhľadanie ohlasov na diela.

Metódy nachádzajúce sa v triede:

- getIndices – Metóda vyhľadáva indexy v databáze WoS na základe vstupného parametra WoSId. Jej výstupom je odpoveď vo formáte XML
- getCitedReferences – Metóda, ktorá vyhľadáva ohlasy v databáze WoS na základe vstupného parametra WoSId.
  - spúšťa sa timer a takisto sa vytvorí cookie, ktoré je potrebné na vyhľadávanie v databáze WoS
  - následne sa zo vstupného JSON objektu vyberú parametre a vytvorí sa URL adresa, ktorá sa spolu s dopytom, ktorý obsahuje vytvorené cookie, pošle vo forme SOAP správy
  - výsledné záznamy sa uložia do listu JSON objektov
  - výstupom je zoznam JSON objektov, ktoré reprezentujú získané ohlasy z WoSu

- getRecordsById – z vyplnených parametrov JSON objektu vytvoríme search query pre WoS a takisto ako pri vyhľadaní citácií vytvoríme cookie a pošleme SOAP správu
  - zo záznamov vyhľadaných podľa parametrov opäť vytvoríme JSON objekty a uložíme ich do listu objektov
  - výstupom je daný list JSON objektov
- getRecords – metóda na vyhľadanie diel podľa mena autora a názvu diela
  - na základe vstupného JSON objektu sa vytvorí tak ako vždy query podľa WoS API na vyhľadanie diela v databáze
  - opäť sa pošle SOAP správa s vytvorenou URL adresou a cookie
  - vyhľadané záznamy vytvoria JSON objekty, ktoré vytvoria list týchto JSON objektov
  - výstupom je zoznam diel z WoSu nájdených podľa mena autora a názvu diela
- createSOAPRequest – správa pre autentifikáciu
- getSOAPMessageFromString – metóda prevádzajúca XML string na SOAP message
  - na vstupe je xml správa vo forme stringu
  - na výstupe je SOAP správa

2.4 Testovanie

Spočiatku sme testovali vyhľadávanie diel a citácií vo WoSe a Scopuse manuálne, pomocou REST klienta v prehliadači Google Chrome s názvom Postman. Zadali sme vyhľadávací JSON a podľa neho sme sa dopytovali na tieto databázy. Následne sme výsledky porovnávali podľa testovacích dát získaných z manuálneho vyhľadávania na stránkach citačných databáz WoS a Scopus.

Pre skvalitnenie procesu testovania sme prešli na automatizované testovanie vyhľadávania diel a citácii. Napísali sme testovacie scenáre pre vyhľadávanie diel a citácií. Tieto scenáre pokrývajú generické služby, ktoré volajú ďalšie služby a metódy. Dáta do týchto scenárov sme získavali manuálnym vyhľadávaním v systémoch WoS a Scopus.

Príklad scénaru:

- Je zadané správne idWos aj idScopus (a správne sú zadané všetky ostatné parametre potrebné na vyhľadanie ohlasov).
- Vstup:

Listing 2.10: Príklad testovacieho JSONu

```
{
"request_type": "citations",
"idWos": "000074191800013",
"idScopus":"0031867993" ,
"institution":"",
"institutionidrecord":"STU",
"auth_token":"valid"
}
```

- Výstup: „recordsWosCount»= 10; "recordsScopusCount»= 13

Na základe týchto testovacích scenárov sme vytvorili súbory obsahujúce vstupné testovacie dáta. Toto nám umožňuje dáta jednoducho upraviť, alebo rozšíriť o ďalšie. Dáta sú zapísane vo forme JSONov, ktoré používame pre vyhľadávanie diel a ohlasov, no pribudli atribúty pre identifikáciu korektných a chybových testovacích vstupov.

Príklad testovacích dát:

Listing 2.11: Príklad testovacích dát uložených v súbore

```
"request_type":"citations",
"idWos": "",
"idScopus":"" ,
"institution":"",
"institutionidrecord":"STU",
"auth_token":"valid"
}, "output":{"code":116}
},
{"id":2, "typ":"ok","input":{
"request_type":"citations",
"idWos":"000074191800013",
"idScopus":"0031867993" ,
"institution":"",
"institutionidrecord":"STU",
"auth_token":"valid"
},
"output":{
"recordsWosCount":10,
"recordsScopusCount":13
}
},
```

Následne sme vytvorili JUnit testy, ktoré načítavajú vstupné dáta z vytvorených súborov. Testy sú rozdelené na testovanie vyhľadávania diel a na testovanie vyhľadávania citácií. Samotné testy sú realizované tak, že sa volá webové rozhranie, kde sa pomocou metódy POST posielajú vstupné dáta. Toto riešenie pokrýva všetky metódy, ktoré sú potrebné na vyhľadanie diela a citácií, ako aj na ich uloženie.

Na testovanie POST metódy sme použili manuálne testovanie pomocou restového clienta POSTMAN v prehliadači Google Chrome. Toto riešenie sme využili aj

na manuálne testy pri vývoji nových funkcií, oprave niektorých chýb alebo zmene štruktúry vstupných a výstupných JSON. Taktiež sme si ním overili správnu implementáciu JUnit testov.

Pri logike servera využívame manuálne testovanie, kde overujeme, či je na danej adrese dostupná webová služba.

Databázu testujeme pri manuálnom a automatickom testovaní webových služieb, kde sa chyby v databáze, ale pri spracovaní údajov databázy odzrkadlia na výstupe, ktorý vracia webová služba.