VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

EXTRAKCE INFORMACÍ Z WEBOVÝCH STRÁNEK

INFORMATION EXTRACTION FROM WEB PAGES

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. JAKUB BUKOVČÁK**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. RADEK BURGET, Ph.D.**
SUPERVISOR

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2018/2019

Zadání diplomové práce

Student: **Bukovčák Jakub, Bc.**
Program: Informační technologie Obor: Informační systémy
Název: **Extrakce informací z webových stránek**
Information Extraction from Web Pages
Kategorie: Web

Zadání:

1. Seznamte se se současnými metodami extrakce strukturované informace z HTML dokumentů.
2. Prostudujte technologie pro tvorbu informačních systémů s webovým rozhraním. Zaměřte se na platformu Java EE.
3. Navrhněte architekturu systému pro definici a správu úloh extrakce informací a získávání extrahovaných dat.
4. Po dohodě s vedoucím zvolte vhodnou metodu extrakce informací a implementujte navržený systém.
5. Proveďte testování systému na reálných webových stránkách.
6. Zhodnoťte dosažené výsledky.

Literatura:

- Burget, R.: Information Extraction from HTML Documents Based on Logical Document Structure, Brno University of Technology, 2004
- Burget R.: Information Extraction from the Web by Matching Visual Presentation Patterns. In: Knowledge Graphs and Language Technology: ISWC 2016 International Workshops: KEKI and NLP&DBpedia. Kobe: Springer International Publishing, 2017, s. 10-26. ISBN 978-3-319-68722-3.
- Bien. A.: Real World Java EE Patterns: Rethinking Best Practices, lulu.com, 2009

Při obhajobě semestrální části projektu je požadováno:

- Body 1 až 3

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Burget Radek, Ing., Ph.D.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 22. května 2019
Datum schválení: 23. října 2018

Abstrakt

Táto diplomová práca sa zaoberá súčasným stavom technológií používaných na sťahovanie webových stránok a extrakciou štruktúrovaných informácií z nich. Popisuje dostupné nástroje, ktoré umožňujú a zjednodušujú tento proces. Ďalej sa venuje základnému prehľadu technológií používaných pre vytváranie webových stránok. Nachádzajú sa tu informácie o tvorbe informačných systémov s webovým používateľským rozhraním v prostredí Java Enterprise Edition (Java EE). Hlavnou časťou je návrh a implementácia webovej aplikácie pre definíciu a správu extrakčných úloh. V závere je popísané testovanie aplikácie na reálnych webových stránkach a zhodnotenie dosiahnutých výsledkov.

Abstract

This master thesis is focused on current technologies that are used for downloading web pages and extraction of structured information from them. The paper describes available tools to make this process possible and easier. Another part of this document provides the overview of technologies that can be used for creating web pages. Also, there is an information about development of information systems with web user interface based on Java Enterprise Edition (Java EE) platform. The main part of this master thesis describes design and implementation of application used to specify and manage extraction tasks. The last part of this project describes application testing on real web pages and evaluation of achieved results.

Kľúčové slová

HLRT wrapper, extrakcia informácií z HTML, Java EE, Web Crawling, sťahovanie HTML dokumentov

Keywords

HLRT wrapper, information extraction from HTML, Java EE, Web Crawling, downloading HTML documents

Citácia

BUKOVČÁK, Jakub. *Extrakce informací z webových stránek*. Brno, 2019. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Radek Burget, Ph.D.

Extrakce informací z webových stránek

Prehlásenie

Prehlasujem, že som túto diplomovú prácu vypracoval samostatne pod vedením pána Ing. Radka Burgeta, Ph.D. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....
Jakub Bukovčák
21. mája 2019

Poďakovanie

Týmto by som chcel poďakovať môjmu vedúcemu práce, ktorým bol Ing. Radek Burget, Ph.D., za ochotu a odbornú pomoc poskytnutú pri práci na tomto zadaní. Ďalej sa chcem poďakovať svojej žene a rodine za podporu počas celého štúdia.

Obsah

Kapitola 1

Úvod

Od vzniku internetu už uplynulo niekoľko desaťročí a jeho vývoj stále napreduje. Veľkým míľnikom pre sprístupnenie internetu ľuďom bolo vytvorenie technológie World Wide Web. Vďaka tomu začalo postupne vznikať množstvo webových stránok dostupných pre ľudí cez webové prehliadače. V súčasnej dobe je na internete veľký počet webových stránok, na ktorých pribúda každým dňom obrovské množstvo dát a informácií. K tomuto prispievajú rôzne sociálne siete, blogy, informačné portály, spravodajské portály a pod.

Počas rozvoja internetu sa začali vyvíjať aj nástroje na získavanie dát a informácií z webových stránok. Dnešné využitie takýchto nástrojov spočíva v stiahnutí webovej stránky a získaniu konkrétnych informácií z nej. Takto získané dáta sa ďalej využívajú na porovnávanie cien elektronických obchodov, indexáciu webu pre vyhľadávače, získavanie kontaktných informácií a pod.

Témou tejto diplomovej práce je vytvoriť prehľad metód používaných na extrakciu informácií z HTML dokumentov. Popisuje tvorbu informačných systémov vytvorených na Java Enterprise Edition (Java EE) platforme. Cieľom práce je navrhnúť a implementovať webovú aplikáciu pre definíciu a správu extrakčných úloh v prostredí Java EE. Potom výslednú aplikáciu otestovať na reálnych webových stránkach a zhodnotiť výsledky.

Po splnení zadania tejto diplomovej práce, vznikne nová webová aplikácia na Java EE platforme pre definíciu a správu extrakčných úloh. Aplikácia bude umožňovať registráciu a prihlásenie používateľa. Po prihlásení bude môcť používateľ vytvárať nové extrakčné úlohy, ktoré budú sťahovať webové stránky a následne extrahovať dáta z nich. Extrahované dáta sa uložia do databázy. Používateľ bude môcť spravovať svoje úlohy a súbory s dátami.

Táto práca sa skladá zo siedmych kapitol, pričom prvá je tento úvod. V 2. kapitole sa popisujú základné technológie pre tvorbu a dizajn webových stránok. Popisuje sa význam správne štruktúrovaného HTML dokumentu a tiež sa táto časť venuje vývoju webových aplikácií na Java EE platforme. V 3. kapitole sa popisuje princíp sťahovania webových stránok. Ďalej sa venuje dostupným metódam pre extrakciu informácií z HTML dokumentov. Popisujú sa tu dostupné nástroje používané pre sťahovanie HTML dokumentov a následnú extrakciu dát z nich. Ďalšia, 4. kapitola sa venuje analýze a návrhu aplikácie. Obsahuje neformálnu špecifikáciu aplikácie a následnú analýzu požiadavkov, ktoré sú zachytené pomocou diagramu prípadov použitia. Kapitola popisuje návrh modelu databázy pre ukladanie dát a taktiež návrh diagramov tried. Obsahuje aj návrh grafického používateľského rozhrania. V 5. kapitole sa popisujú implementačné podrobnosti o celej aplikácii. Taktiež sa tu nachádza testovanie aplikácie počas vývoja, testovanie aplikácie na reálnych webových stránkach a zhodnotenie výsledkov. Predposledná 6. kapitola sa zaoberá testovaním aplikácie. Posledná 7. kapitola je záver, kde je zhodnotenie tejto diplomovej práce.

Kapitola 2

Súčasný stav webových technológií

Kapitola sa zaoberá súčasným stavom webových technológií týkajúcich sa tejto práce. Obsahuje informácie o základných technológiách pre tvorbu webových stránok ako sú HTML, XHTML, CSS a pod. Popisuje čo znamená správne štruktúrovaný HTML dokument a aký má význam pre extrakciu dát. Taktiež sa tu nachádzajú informácie o značkovacom jazyku XML a jeho využití. Ďalšiu časť tejto kapitoly tvorí popis vývoja webových aplikácií založených na Java EE platforme a popis aplikačného rámca Spring Framework.

2.1 Hypertext Markup Language

Hypertext Markup Language (HTML) je značkovací jazyk používaný na tvorbu webových stránok. Význam značkovania spočíva v pridávaní dodatočných informácií do textu, ktoré slúžia na definíciu štruktúry dokumentu. Taktiež umožňuje do webových stránok vkladať multimediálny obsah, tabuľky, formuláre, hypertextové prepojenia a mnohé ďalšie [9]. Takto vytvorené dokumenty sa dajú zobrazovať cez webové prehliadače.

HTML dokumenty sa skladajú z viacerých elementov, ktoré tvoria základné stavebné bloky dokumentu. Jednotlivé elementy sa môžu do seba viacnásobne vnárať. Každý element tvorí prvok, ktorý sa nazýva HTML tag a je tvorený znakmi $<$ a $>$. Tieto tagy majú svoj sémantický význam [20], napr.:

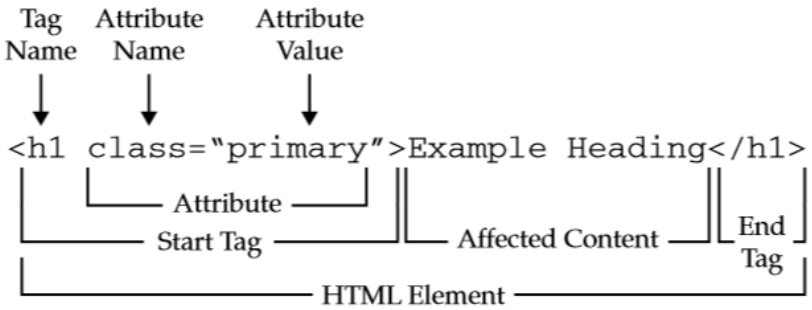
- *<p>* – definuje odstavec,
- *<b>* – definuje výrazný typ písma,
- ** – definuje typ písma ako kurzívu.

Elementy sa delia na párové a nepárové. Párové musia mať otvárací aj ukončovací tag, oproti nepárovým, ktoré majú iba otvárací tag. Príkladom párového je nastavenie písma na kurzívu, nepárový element je napr. vloženie obrázka do dokumentu. Takto môže vyzerať ich možná reprezentácia v HTML dokumente:

písmo s kurzívou
<img src="obr1.png" alt="Popis obr1" height="100" width="100">

Elementy môžu v sebe obsahovať aj rôzne atribúty, ktoré slúžia pre presnejšiu špecifikáciu významu a správania sa elementu. Na nasledujúcom obrázku 2.1 je znázornený konkrétny príklad HTML elementu, ktorého časti majú nasledovný význam:

- Názov tagu – v tomto prípade je názov *h1*, čo znamená že sa jedná o nadpis prvej úrovne.
- Atribút – názov atribútu je *class*, jedná sa o globálny atribút, ktorý sa použije napr. v CSS súbore na nastavenie kaskádového štýlu (viď ďalšia časť 2.5). Tento atribút sa skladá z dvoch častí – názov a hodnota. V rôznych elementoch je možné nastavovať odlišné atribúty a ich hodnoty.
- Obsah elementu – táto časť obsahu bude vykreslená a viditeľná vo webovom prehliadači. Bude naformátovaná ako nadpis prvej úrovne.
- Celý element je na konci ukončený koncovým tagom.



Obr. 2.1: Príklad HTML elementu a jeho častí[1]

Pre extrakciu dát z HTML dokumentov nás budú zaujímať informácie, ktoré sú umiestnené najmä v nasledujúcich elementoch a atribútoch:

- **Element *head*** – jedná sa o kontajner, v ktorom sa nachádzajú tieto elementy *<title>*, *<style>*, *<base>*, *<link>*, *<meta>*, *<script>* a *<noscript>* [9].

  Dôležitým elementom pre extrakciu informácií z vyššie uvedených je *<meta>*, ktorý obsahuje metadáta[2] o dokumente. Tieto metadáta sa nezobrazujú v prehliadači. Element je nepárový a zvyčajne býva použitý viackrát. Nachádzajú sa tam údaje o názve dokumentu a jeho obsahu, kľúčové slová a pod. Tieto informácie môžu vylepšiť optimalizáciu pre vyhľadávacie nástroje (Search Engine Optimization – SEO) [9].

  Z hľadiska extrakcie informácií zo štruktúrovaných dokumentov nás zaujíma hlavne atribút *charset*. Napr. *<meta charset="UTF-8">* znamená, že nastavuje znakové kódovanie pre celý HTML dokument na hodnotu atribútu, čo je v tomto prípade *UTF-8*. Toto je dôležité pre správnu interpretáciu znakov vo webovom prehliadači. Taktiež pri sťahovaní HTML dokumentov a následnom spracovaní je potrebné poznať aké kódovanie bolo použité, aby nedošlo k znehodnoteniu nesenej informácie a nesprávnej interpretácii dát. Mohli by tak vzniknúť pri extrahovaní dáta, ktoré nemajú pôvodnú výpovednú hodnotu.

[1] Obrázok prevzatý a upravený z: https://tutorial.techaltum.com/htmlTags.html, navštívené: 29. 11. 2018.

[2] Metadáta – sú dáta, ktoré v sebe nesú informácie o iných dátach.

- **Element *body*** – v tejto časti dokumentu sa nachádzajú informácie pre extrakciu dát. Tento element tvorí hlavné telo dokumentu a sú v ňom obsiahnuté všetky časti, ktoré sa budú spracovávať a získavať z nich informácie.

- **Atribút *lang*** – slúži na deklarovanie jazyka, ktorým je textová časť dokumentu písaná. Tento atribút môže byť nastavený na celý HTML dokument, alebo iba na určité jeho časti, napr.:

  <html lang="en"> ... </html>
  * Odstavec, ktorý je v napísaný slovenčine. *

  Prvý riadok ukážky znamená, že v dokumente sa bude používať anglický jazyk. Druhý riadok ukazuje ako sa dá nastaviť zmena jazyka iba na konkrétny odstavec.

  Pri extrakcii informácií, keď sa tam vyskytuje tento atribút s konkrétnym jazykom, treba prispôsobiť získavanie dát prípadným dodaním znakového kódovania podľa jazyka, aby nedošlo k stratám na dátach kvôli zlému kódovaniu.

- **Atribúty *id* a *class*** – pomocou týchto atribútov je možné nastavovať odkaz na príslušný element. Atribút *id* musí byť unikátny v rámci celého HTML dokumentu, naopak rovnaký atribút *class* sa môže vyskytovať vo viacerých elementoch v rámci dokumentu. Taktiež jeden element nemusí mať žiadny, alebo má práve jeden *id* atribút, pričom môže obsahovať ľubovoľné množstvo *class* atribútov [6].

  Tieto atribúty sa používajú najmä na tvorbu vizuálnej podoby vykresľovanej stránky (viď časť CSS). Taktiež sa cez programovací jazyk JavaScript môže pristúpiť k jednotlivým elementom pomocou *id* a *class*, pričom sa môžu takto získať dáta z elementu, prípadne meniť jeho atribúty, štýl a pod.

  Tieto atribúty sú zaujímavé z hľadiska extrakcie informácií, pretože špecifikujú konkrétne dáta v rámci HTML dokumentu. Keby chceme získať informácie napr. z internetového obchodu ponúkajúceho nejaký tovar, môže každá položka, ktorá sa tam predáva mať nasledovný element:

  * 10 € *,

  v ktorom je napísaná konkrétna suma. Keď sa potom pri spracovávaní dokumentu vyskytne element s touto triedou, tak sa z neho získajú textové dáta, ktoré reprezentujú cenu produktu.

HTML je základným programovacím jazykom pre tvorbu webových stránok dostupných online na internete. V dnešnej dobe je aktuálna verzia HTML 5.2 podľa W3C[3]. Od verzie HTML 5 pribudli viaceré nové elementy pre členenie častí (*header*, *footer*, *section*, *article*), pribudol kontajner pre prácu s grafikou (*canvas*). Dajú sa nastavovať nové atribúty pre formuláre a bola pridaná ďalšia nová funkcionalita [23].

[3]World Wide Web Consortium (W3C) – okrem iného tvoria aj slobodné štandardy pre HTML. Viď https://www.w3.org/TR/html52/, navštívené: 4. 12. 2019.

2.2 Správne štruktúrovaný HTML dokument

Správne štruktúrovaný HTML dokument (Well-Formed) sa nazýva dokument, ktorý spĺňa syntax definovanú štandardom pre HTML od W3C. Medzi základnú syntax takého dokumentu patrí, že všetky elementy, ktoré majú definovaný otvárací aj uzavierací tag, aby ich oba mali použité. To sú napr. *<h1>* a *</h1>*, * * a * * a pod. Výnimkou sú elementy, ktoré nepotrebujú uzavieracie tagy a nazývajú sa prázdne elementy[4] (void elements). Takéto elementy sú napr. *
*, *<img>*, *<a>* atď. Taktiež pre správne štruktúrovaný HTML dokument musí byť vnáranie všetkých elementov v správnom poradí. Povolený je iba jeden hlavný (koreňový) *html* element, v ktorom sa nachádzajú všetky ostatné elementy [22].

Na nasledujúcom príklade 2.2 je znázornený správne a nesprávne štruktúrovaný HTML dokument. Nesprávny dokument porušuje niektoré vyššie spomenuté vlastnosti: element *<h1>* nemá ukončovací tag a elementy *<b>* aj *<i>* sú ukončené v nesprávnom poradí v rámci vnorenia.

```
1  <!DOCTYPE html>
2  <html lang="sk">
3  <head>
4     <meta charset="UTF-8">
5     <title>Ukážka</title>
6  </head>
7  <body>
8     <h1>Nadpis 1. úrovne
9     <br>
10    <b>Text <i>s kurzívou.</b></i>
11 </body>
12 </html>
```

a) Nesprávna štruktúra

```
1  <!DOCTYPE html>
2  <html lang="sk">
3  <head>
4     <meta charset="UTF-8">
5     <title>Ukážka</title>
6  </head>
7  <body>
8     <h1>Nadpis 1. úrovne</h1>
9     <br>
10    <b>Text <i>s kurzívou.</i></b>
11 </body>
12 </html>
```

b) Správna štruktúra

Obr. 2.2: Ukážka správnej a nesprávnej štruktúry HTML dokumentu

Z hľadiska extrakcie informácií je správne štruktúrovaný HTML dokument dôležitý pre jeho vlastnosti, ktoré sú popísané vyššie. Z príkladu 2.2 v časti *Nesprávna štruktúra* by mohla nastať nasledujúca situácia. Úlohou by bolo implementovať extrakčnú metódu, ktorá by z dokumentu pomocou regulárnych výrazov mala získať všetky nadpisy prvej úrovne, teda získať obsah medzi *<h1>* a *</h1>*. Pri spracovávaní dokumentu by sa narazilo na začiatok elementu *<h1>*, z ktorého chceme získať obsah, avšak tento element nemá ukončovací tag, tak sa to nepodarí.

Pred extrakciou dát z HTML dokumentov je vhodné použiť nástroj, ktorý z dostupných dokumentov vytvorí správne štruktúrované dokumenty, ktoré následne dostane konkrétna extrakčná metóda na svoj vstup.

2.3 Extensible Markup Language

Extensible Markup Language (XML) je značkovací jazyk, ktorý patrí do podmnožiny SGML[5]. Jeho hlavnou úlohou je uchovávanie a prenášanie dát v štruktúrovanej podobe. Pri dodržaní

[4] Prázdny element (void element) – je element, ktorý nemôže obsahovať nasledovníka, prípadne obsahovať nejaký text.

[5] Standard Generalized Markup Language (SGML) – je štandard, ktorý umožňuje definíciu vlastných podriadených značkovacích jazykov.

určitých XML pravidiel je možné vytvoriť vlastný značkovací jazyk na ukladanie a prenášanie dát [12].

Výhodami tohto jazyka sú jeho textový formát a podpora univerzálneho kódovania (Unicode), vďaka čomu je možné pracovať s dátami v rôznych svetových jazykoch. Taktiež je to nezávislý jazyk, ktorý je prenositeľný medzi viacerými systémami a platformami bez prípadných konverzií. Umožňuje reprezentáciu najbežnejších dátových štruktúr ako sú listy, stromy a záznamy. Ďalšou výhodou je, že dáta uložené v XML formáte môžu byť hocikedy zmenené bez poškodenia ich formy.

Nevýhodou je redundancia potrebných informácií na štruktúru oproti iným textovým formátom na prenášanie dát. To sa môže odraziť na výkone aplikácie pri spracovaní, ukladaní a preposielaní väčších dokumentov.

2.4 Extensible Hypertext Markup Language

Extensible Hypertext Markup Language (XHTML) je značkovací jazyk podobný HTML, avšak má prísnejšiu syntax. V nasledujúcich bodoch sú zhrnuté hlavné rozdiely medzi HTML a XHTML [9]:

- HTML je aplikácia SGML – jedná sa o konkrétny jazyk, splňujúci SGML štandard. XHTML je aplikácia XML.
- V XHTML dokumente musí mať každý otvárací tag jemu odpovedajúci uzavierací tag. Uzatvorené musia byť aj prázdne elementy (void elements). Napr. v HTML je správne *
*, pričom v XHTML je správne *
*.
- Všetky elementy a ich atribúty v XHTML musia byť písané iba malými písmenami (lower case), naopak v HTML na tom nezáleží (je case insensitive – nerozlišuje malé a veľké písmená). Hodnoty atribútov a obsah elementov v XHTML už nemusí byť písaný iba malými písmenami.
- V XHTML musia byť ukončovacie tagy jednotlivých elementov v správnom poradí podľa vnorenia.

Dokumenty, ktoré sú typu HTML sa dajú pomocou určitých pravidiel previesť na XHTML dokumenty a naopak. Hlavné kroky prevodu HTML na XHTML dokument (pre viac informácií viď [8]):

- zmeniť všetky názvy elementov a atribútov na malé písmená (lower case),
- všetky hodnoty atribútov musia byť v úvodzovkách,
- uzavrieť všetky elementy vrátane prázdnych elementov,
- nastaviť typ dokumentu *XHTML <!DOCTYPE>* a pridať *xmlns* atribút do *html* elementu,
- upraviť vnorovanie elementov tak, aby boli v správnom poradí otváracie a ukončovacie tagy.

XHTML dokumenty majú oproti HTML dokumentom viaceré výhody. Jednou z nich je, že XHTML dokumenty sú vždy správne štruktúrované (Well-Formed) a neobsahujú žiadne chyby, čo môže čiastočne urýchliť zobrazovanie webových stránok v prehliadačoch. Taktiež to urýchľuje proces Web Crawlingu pre indexáciu webových stránok vyhľadávacími nástrojmi (viď časť 3.1).

2.5 Tvorba dizajnu stránok

Cascading Style Sheets (CSS) je technológia používaná pre vizuálne formátovanie, najmä HTML a XHTML dokumentov. Takto sa dosiahne oddelenie vzhľadu v súbore s kaskádovými štýlami od obsahu HTML dokumentu. Tým sa sprehľadní HTML dokument a je možné použiť jeden CSS súbor s definíciou dizajnu pre viaceré webové stránky. Kaskádové štýly definujú pravidlá pre formátovanie a nastavovanie vzhľadu HTML elementov [7].

CSS pravidlo sa skladá z viacero častí. Prvou časťou je selektor, ktorý bližšie špecifikuje element, na ktorý sa má dané pravidlo aplikovať. Takto sa môže vybrať napríklad element *h1*. Najviac používanými selektormi sú atribúty *id* a *class*, ktoré odkazujú na HTML elementy. Ďalšou časťou, ktorá tvorí pravidlo je deklarácia, ktorá sa skladá z parametra a hodnoty. Pravidlo, ktoré nastaví farbu písma nadpisu prvej úrovne na červenú môže vyzerať nasledovne: *h1 { color: red; }*.

Pridávanie kaskádových štýlov do HTML dokumentov je možné nasledujúcimi spôsobmi:

- Riadkové kaskádové štýly – vzťahujú sa na element v rámci ktorého sú použité. Nevýhodou je, že robí HTML dokument menej prehľadným. Taktiež to nie je efektívne pri formátovaní elementov, kde sa rovnaký vzhľad opakuje, prípadne sa trochu obmieňa. Ďalšou nevýhodou je, že sa tento štýl nedá znovu použiť pre iné dokumenty.
- Vložené kaskádové štýly – tieto štýly sa vkladajú na začiatok HTML dokumentu do elementu *head* a preto majú platnosť v rámci celého dokumentu. Sú vložené medzi element *<style>* a *</style>*. Nevýhodou je, že takto definované štýly sa dajú použiť iba v rámci jedného dokumentu. V prípade, že sú vo viacerých dokumentoch, tak sa pri zmene v jednom dokumente musia meniť aj v ostatných.
- Vonkajšie kaskádové štýly – do elementu *head* sa vloží odkaz na externý CSS súbor v ktorom sú definované pravidlá pre vzhľad dokumentu. Je možné takto vložiť aj viac odkazov na súbory, ktoré definujú vzhľad.

Kaskádové štýly sú v dnešnej dobe neodmysliteľnou súčasťou webových stránok. Definujú celkový dizajn stránok, pričom oddeľujú vzhľad od štruktúry dokumentu.

2.6 Java Enterprise Edition

Java Enterprise Edition (Java EE) je platforma pre vývoj podnikových a webových aplikácií. Java EE poskytuje vývojárom viaceré aplikačné programové rozhrania (Application Programming Interface, ďalej iba API), ktoré zmenšujú zložitosť výsledných aplikácií, urýchľujú vývoj a zvyšujú výkonnosť celej aplikácie. Táto časť čerpá informácie z [15].

Hlavným cieľom tejto technológie je zjednodušiť vývoj aplikácie. Umožňuje vývoj pomocou anotácií, vďaka čomu sa zmenšuje dodatočná potreba XML konfigurácie. Podporuje a zameriava sa na využívanie klasických POJO[6] objektov v Jave. Základom je programovací jazyk Java a virtuálny stroj pre beh Java aplikácií (Java Virtual Machine).

[6] Plain Old Java Object (POJO) – je objekt v programovacom jazyku Java vytvorený z klasickej Java triedy, ktorá nie je obmedzovaná napr. zo strany aplikačného rámca.

Java EE architektúra

Podniková aplikácia (enterprise application) znamená, že sa jedná o systém zložený z viacerých častí. Tieto jednotlivé časti sú od seba logicky oddelené a zvyčajne sa delia do viacvrstvovej architektúry. Každá vrstva zabezpečuje funkcionalitu pre danú časť aplikácie.

Na nasledujúcom obrázku 2.3 je znázornená viacvrstvová architektúra v Java EE. Táto architekúra sa nazýva *Model-View-Controller* (MVC) architektúra. Skladá sa z nasledujúcich vrstiev [15]:

- **Klientská vrstva (Client tier)** – táto vrstva je zvyčajne webový klient, alebo aplikačný klient. V prípade webového klienta je dynamický obsah tvorený zvyčajne HTML a XHTML dokumentami, ktoré sú generované na strane servera a zobrazované vo webovom prehliadači. Aplikačný klient je spustený na strane klienta a tvorí ho grafické používateľské rozhranie (GUI), ktoré umožňuje komunikáciu so serverom a vykonávanie jednotlivých akcií systému.

- **Webová vrstva (Web tier)** – používa technológie, ktoré sa starajú o generovanie vzhľadu výslednej aplikácie. V prípade webovej aplikácie sa na to najčastejšie používa *JavaServer Faces* (JSF) a *JavaServer Pages* (JSP). Keď je klient aplikácia, využívajú sa servlety[7].

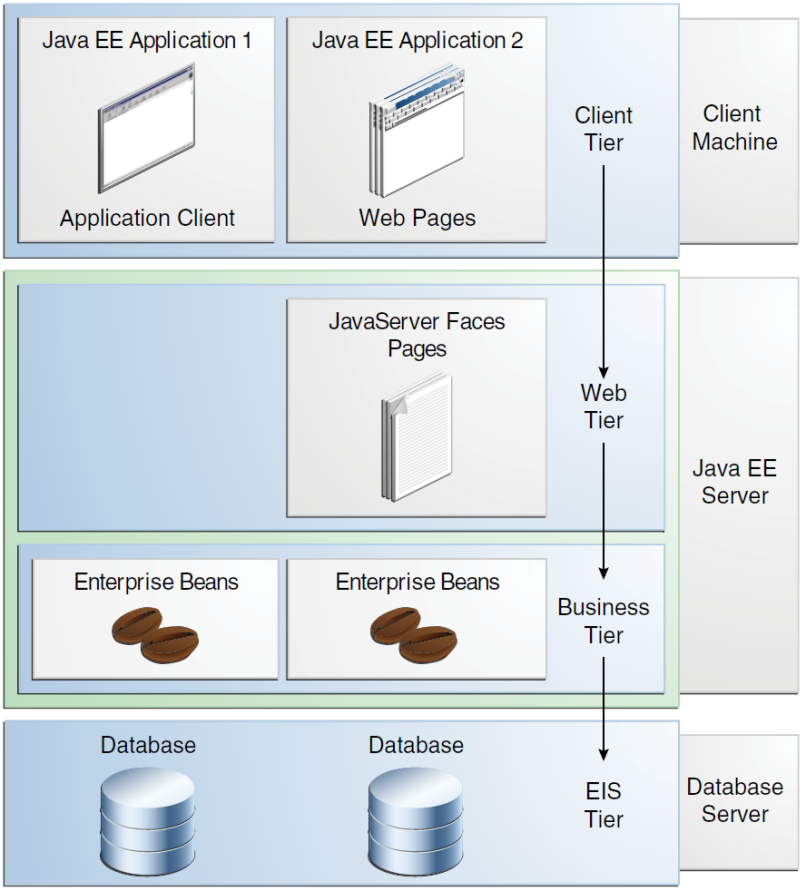
  JSP je technológia, ktorá obsahuje textové dokumenty majúce funkcionalitu ako servlety, ale zameriavajú sa na vytváranie statického obsahu pre webové stránky.

  JSF je technológia rozširujúca funkcionalitu JSP a servletov. Umožňuje vytváranie plnohodnotných webových aplikácií s grafickým používateľským rozhraním. Jedná sa o aplikačný rámec, ktorý obsahuje rôzne komponenty na tvorbu používateľského rozhrania, ako sú menu panely, tlačidlá, tabuľky, formuláre, multimediálny obsah a pod.

- **Business vrstva (Business tier)** – implementuje v sebe správanie sa celej aplikácie. Vrstva zabezpečuje spracovanie požiadaviek od klientov, požiadavky do databázy cez EIS vrstvu, prípadné spracovanie dát a preposlanie ich do klientskej aplikácie.

- **Databázová vrstva (Enterprise Information System tier)** – zabezpečuje dátovú časť informačného systému. Implementuje funkcionalitu pre prácu s databázou.

[7] Servlet – je trieda v programovacom jazyku Java, ktorá dynamicky spracováva požiadavky od klienta a vytvorí na ne odpoveď.



Obr. 2.3: Viacvrstvová architektúra Java EE[8]

Java EE kontajner

Beh aplikácie na servery zabezpečuje rozhranie medzi jednotlivými modulmi a ich funkcionalitou nazývané kontajner. Pred tým ako je aplikácia spustená sa musia jednotlivé moduly vložiť do kontajnera. Jednotlivé kontajnery sa delia na:

- EJB kontajner – umožňuje beh EJB modulov aplikácie. Tieto moduly bežia na Java EE servery.
- Webový kontajner – umožňuje beh webovej vrstvy. Teda má na starosti jednotlivé webové stránky, servlety a ďalšie komponenty z EJB potrebné pre beh aplikácie. Tieto moduly bežia na Java EE servery.
- Java EE server – taktiež nazývaný aplikačný server, poskytuje oba predchádzajúce kontajnery, teda webový a EJB kontajner.

Niektoré dostupné kontajnery

Pre technológiu Java EE sú dostupné viaceré kontajnery a aplikačné servery zabezpečujúce beh aplikácie. Niektoré z nich sú pod voľne šíriteľnou licenciou, iné sú komerčné. V nasledujúcej časti sú popísané niektoré z nich [21]:

[8]Obrázok prevzatý z: https://docs.oracle.com/javaee/7/JEETT.pdf, navštívené: 20. 12. 2018.

- **Apache Tomcat** – je kontajner s otvoreným zdrojovým kódom (open source[9]) od *Apache Software Foundation* poskytujúci implementáciu technológií Java Serlvets, JavaServer Pages (JSP), Java Expression Language a WebSocket. Je jedným z najpoužívanejších kontajnerov. Neposkytuje kompletnú špecifikáciu všetkých Java EE častí. Aplikačné servery so širšou podporou sú popísané ďalej.

- **Apache TomEE** – jedná sa rozšírenie *Tomcat* pre plnohodnotné využitie všetkých finkcií poskytujúcich platformou Java EE. Podporuje viaceré technológie ako sú JavaServer Faces (JSF), Java Transaction API (JTA), Java Persistence API (JPI) a pod. Je dostupný v dvoch verziách *TomEE* a *TomEE+*.

- **GlassFish** – je ďalším voľne dostupným riešením aplikačného servera. Pôvodne bol vyvíjaný firmou *Sun Microsystems*, teraz patrí pod *Oracle Corporation*. Poskytuje podporu pre moduly ako sú JSP, JMS, JPA, RMI a pod.

- **WildFly** – voľne dostupný aplikačný server patriaci pod licenciu *LGPL, verzia 2.1* a momentálne vyvíjaný firmou *Red Hat*. Poskytuje mnohé časti Java EE ako sú EJB, JSF, JPA, JTA, CDI a pod.

Aplikačný rámec Spring Framework

Spring Framework je aplikačný rámec s otvoreným zdrojovým kódom (open source) určený najmä pre Java EE platformu. Jeho hlavnou úlohou je zjednodušenie vývoja podnikových a webových aplikácií riešených na tejto platforme. Je založený na viacvrstvovej architektúre, ktorá bude popísaná neskôr [16].

Základnou vlastnosťou aplikačného rámca Spring je Inversion of Control (IoC), ktorá umožňuje aplikačnému rámcu kontrolu nad jednotlivými časťami programu a objektmi. Na to sa používa technika nazývaná vkladanie závislostí (Dependency Injection) – triedy sprístupnia závislosti na objekty cez metódy, ktoré implementujú, alebo ich konštruktory. Takto je umožnené aplikačnému rámcu zavolať metódu s konkrétnymi parametrami počas behu aplikácie, prácu s objektmi a ich závislosťami medzi sebou [16].

Spring Framework sa skladá z viacerých modulov, ktoré sú zobrazené na obrázku 2.4. Význam niektorých modulov [24]:

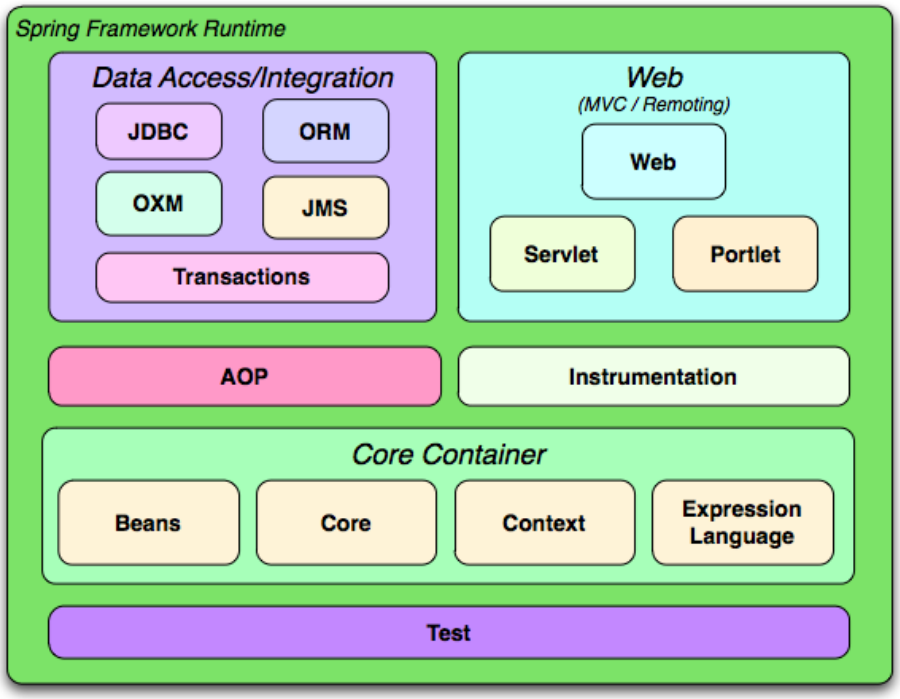
- **Hlavný kontajner (Core container)** – obsahuje moduly, ktoré tvoria základ aplikačného rámca Spring. Zabezpečuje implementáciu IoC a vkladanie závislostí (moduly *Core* a *Beans*). Ďalej sa tu nachádza modul *Expression Language*, ktorý zabezpečuje konfiguráciu objektov aplikácie.

- **Prístup k dátam a integrácia (Data Access/Integration)** – nachádzajú sa tu moduly pre prácu s databázou, objektovo-relačné mapovanie a pod. Taktiež tu je modul pre mapovanie objektov do XML a pre prácu s transakciami.

- **Webový kontajner (WEB)** – jednotlivé moduly, ktoré sa tu nachádzajú pokrývajú časť týkajúcu sa webovej aplikácie.

- **Test** – tento modul obsahuje časti používané pre testovanie aplikácie.

[9]Open Source softvér – je softvér, ktorý má verejne dostupné zdrojové kódy. Používatelia majú právo ho voľne používať, modifikovať a šíriť ďalej na nekomerčné, ako aj komerčné účely.



Obr. 2.4: Jednotlivé moduly používane v aplikačnom rámci Spring[10]

Aplikačný rámec Spring Framework je v dnešnej dobe najpoužívanejším spomedzi ostatných pre platformu Java EE. Výrazne uľahčuje prácu a urýchľuje vývoj podnikových aplikácií.

Zabezpečenie pomocou Spring Security

Dôležitou časťou podnikových aplikácií je implementácia zabezpečenia. V aplikačnom rámci Spring to je umožnené pomocou *Spring Security*. Takto je umožnené zabezpečenie Java EE aplikácií [1]. Hlavnými časťami zabezpečenia je overenie pravosti (authentication) a overenie oprávnenia (authorization). Pri overení pravosti sa zisťuje, či je používateľ ten, za ktorého sa vydáva. Overenie oprávnenia spočíva v zistení role používateľa a následné sprístupnenie akcií pre neho v rámci jeho role. Rolou sa rozumie napr. bežný používateľ, administrátor a pod., pričom každá rola má iné oprávnenia v rámci aplikácie.

Konfigruácia cez Spring Boot

Spring Boot robí jednoduchšiu prvotnú konfiguráciu Java EE aplikácií založených na aplikačnom rámci Spring Framework. Vďaka nemu je možné vytvoriť nezávislú spustiteľnú aplikáciu. Medzi niektoré jeho výhody patrí [13]:

- Umožnuje spustenie aplikácie so vstavaným aplikačným serverom ako sú napr. TomEE a GlassFish. Vďaka tomu nie je potrebné nasadzovanie WAR súborov (Deploy WAR files[11]).

[10] Obrázok prevzatý z: https://docs.spring.io/spring/docs/3.0.0.M4/reference/html/ch01s02.html, navštívené: 20. 12. 2018.

[11] Web application ARchive (WAR) súbory – používajú sa na distribúciu webových aplikácií v Jave. Obsahujú potrebné zdroje na beh aplikácie ako sú triedy, knižnice, servlety a pod.

- Vytvorenie aplikácie bez potreby konfigurácie pomocou XML súboru.
- Poskytuje rozhranie, ktoré sprístupňuje argumenty aplikácie, s ktorými bola spustená.
- Zjednodušuje nastavenie a používanie logovacích nástrojov.
- Podpora automatizačných nástrojov Maven a Gradle.

Spring Initializr

Spring Initializr je nástroj dostupný online[12] pre inicializáciu aplikácie založenej na Spring Boot. Umožnuje automatické vygenerovanie kostry projektu aj s potrebnými závislosťami na externé knižnice.

Na webovej stránke s nástrojom sa dajú nastaviť základné informácie o aplikácií ako sú názov, skupina, popis atď. Umožňuje nastavenie konkrétnej verzie Javy. Poskytuje výber automatizovaného nástroja pre správu závislostí Maven, alebo Gradle. Hlavnú časť tvorí pridávanie jednotlivých závislostí do projektu. Takto sa dá pridať napr. *Web* ktorý pridá závislosti na aplikačný server Tomcat a ďalšie potrebné časti. *JPA* pridá závislosti potrebné na objektovo-relačné mapovanie atď.

Po pridaní všetkých potrebných závislostí pre projekt je možné vygenerovať a stiahnuť výslednú kostru. Táto kostra sa dá následne importovať v integrovanom vývojovom prostredí (IDE) a vytvoriť tak nový projekt, ktorý obsahuje všetky zvolené závislosti a nastavené parametre. Pri tom sa vytvorí aj základná štruktúra projektu.

[12] Nástroj Spring Initializr je dostupný na adrese: https://start.spring.io/, navštívené: 20. 12. 2018.

Kapitola 3

Sťahovanie webových stránok a extrakcia informácií z nich

Kapitola sa zaoberá sťahovaním webových stránok a získavaniu im odpovedajúcim HTML dokumentov. Tento proces sťahovania sa nazýva Web crawling a bude tu popísaný. Ďalej sa tu nachádzajú viaceré dostupné metódy používané pre extrakciu štruktúrovaných dát z HTML dokumentov. Popisujú sa a ich kategórie a využitie. Taktiež tu sú popísané dostupné nástroje používané na sťahovanie webových stránok a následnú extrakciu dát z nich.

3.1 Web crawling – sťahovanie webových stránok

Web crawling (taktiež nazývané Web spidering) je technika slúžiaca na automatické sťahovanie obsahu webových stránok za účelom získania informácií z nich. Nástroj, ktorý sťahovanie realizuje, sa nazýva *crawler* (prípadne *spider*) [26].

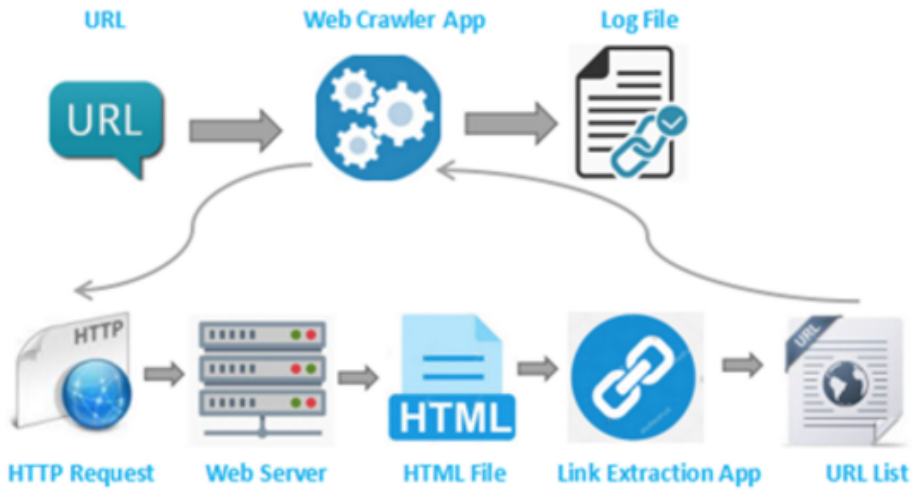
Crawler na začiatku dostane konkrétnu webovú adresu (URL[1]) taktiež nazývanú vstupný bod (seed page) [5]. Z tejto vstupnej adresy začne so sťahovaním HTML dokumentov. Po stiahnutí tejto vstupnej webovej stránky sa z nej pomocou nástroja na extrakciu informácií zo štruktúrovaného dokumentu získajú všetky dostupné hypertextové odkazy (URL). Následne sa tieto URL adresy zaradia do fronty. Vo fronte sú uložené všetky odkazy na webové stránky čakajúce na stiahnutie. Takto stiahnutý dokument sa môže ukladať do úložiska, ktorým je napr. súborový systém, kde je crawler spustený.

Implementácia crawlera býva s využitím viacerých vlákien prípadne procesov, aby sa urýchlil celý proces sťahovania. Každé vlákno tak spracováva niektoré odkazy, nachádzajúce sa vo fronte. Taktiež sa používa oneskorenie medzi jednotlivými požiadavkami na stiahnutie stránky, ktoré slúži napr. aby nebola IP adresa[2] zablokovaná zo strany servera pre veľký počet HTTP požiadavkov.

Na nasledujúcom obrázku 3.1 je znázornená architektúra crawlera, ktorá bola popísaná vyššie.

[1] Uniform Resource Locator (URL) – používa sa na unikátnu identifikáciu dokumentu umiestneného na internete.

[2] IP adresa – je unikátna adresa, ktorá jednoznačne identifikuje zariadenie na lokálnej sieti, alebo v rámci celého internetu.



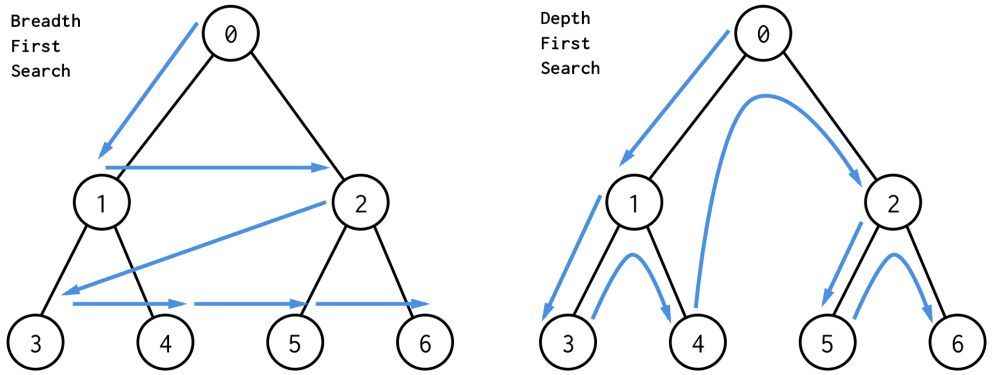
Obr. 3.1: Architektúra Web crawlera[3]

Podľa [5] je možné využiť nasledovné dva algoritmy na prehľadávanie a následné sťahovanie webových stránok:

1. **Prehľadávanie stránok do šírky (Breadth First Search – BFS):**
   - Crawler dostane vstupnú webovú stránku cez URL adresu (seed page), ktorú následne stiahne.
   - Po stiahnutí vstupnej webovej stránky sa z nej extrahujú všetky dostupné hypertextové prepojenia, ktoré sa tam nachádzajú a uložia sa do fronty. Tieto odkazy sú s hĺbkou 1.
   - Potom sa stiahnu všetky webové stránky s hĺbkou 2 (sú odkazované webovými stránkami s hĺbkou 1).
   - Tento proces sa opakuje pokiaľ nebude fronta prázdna, teda budú stiahnuté všetky odkazy, ktoré sa dali získať zo vstupnej URL a ďalšími stránkami dostupnými z nej v rozličných hĺbkach.
2. **Prehľadávanie stránok do hĺbky (Depth First Search – DFS)**
   - Crawler podobne ako v prehľadávaní stránok do šírky dostane vstupnú URL, z ktorej získa všetky odkazy na webové stránky, ktoré sa tam nachádzajú.
   - Získané odkazy sa uložia do fronty.
   - Následne crawler vyberie jeden odkaz z fronty a postupne prechádza a sťahuje všetky dostupné webové stránky (odkazy) z neho. Postupne sa vnoruje hlbšie a hlbšie. Keď sa takto spracujú všetky webové stránky dostupné z pôvodného odkazu, tak sa odstráni z fronty.
   - Potom vyberie z fronty ďalší odkaz a proces sa opakuje.
   - Crawler sa ukončí, keď je fronta prázdna.

[3]Obrázok prevzatý z: https://www.octoparse.com/blog/web-crawling-how-to-build-a-crawler-to-extract-web-data, navštívené: 3. 12. 2018.

Na nasledujúcom obrázku 3.2 sú znázornené dva vyššie popísané algoritmy prehľadávania HTML dokumentov pre získanie hypertextových odkazov.



Obr. 3.2: Algoritmy prehľadávania stránok crawlerom[4]

Teraz budú popísané niektoré možnosti a dôvody ukončenia sťahovanie webových stránok. Rozdelené sú do dvoch kategórií:

1. Možné ukončenia sťahovania zo strany crawlera:
   - fronta odkazov na webové stránky je prázdna,
   - bol dosiahnutý limit pre počet stiahnutých dokumentov,
   - bola dosiahnutá maximálna úroveň vnorenia pre algoritmy *BFS* a *DFS*,
   - bol dosiahnutý limit pre prenesené dáta a pod.
2. Ďalšie možné dôvody pre ukončenie sťahovania:
   - bola zablokovaná IP adresa zo strany servera, z ktorého sa sťahujú dáta,
   - bola obmedzená rýchlosť sťahovania zo strany servera,
   - server prerušil spojenie kvôli prekročeniu limitu prenesených dát,
   - server prerušil spojenie pre časté HTTP požiadavky a pod.

Web crawling sa používa na sťahovanie HTML dokumentov, metadát, obrázkov a pod. Túto techniku používajú najznámejšie vyhľadávacie nástroje ako sú Google, Bing, Yahoo pri indexácii a prehľadávaní webu.

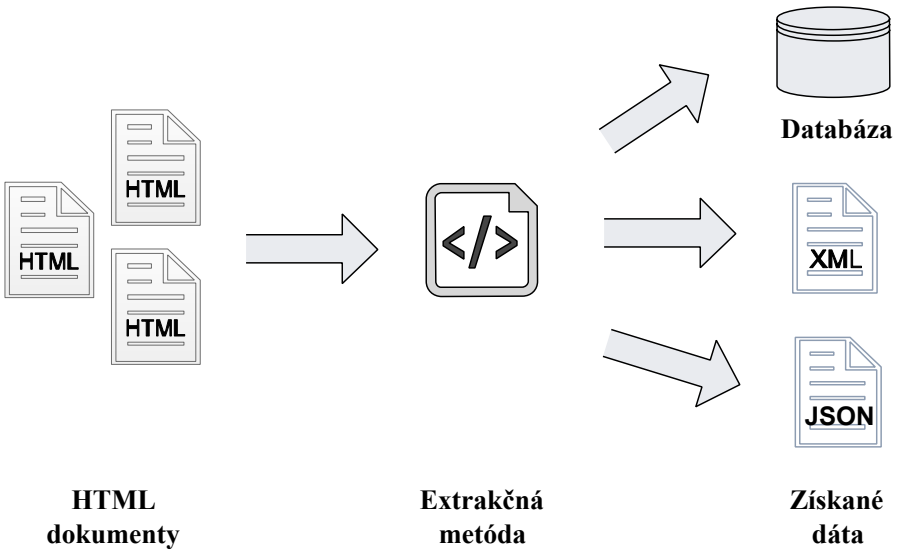
3.2 Extrakcia dát z HTML dokumentov

Extrakcia dát z HTML dokumentov (tiež nazývané Web Scraping, alebo Web Harvesting) je technika používaná na automatickú extrakciu informácií z webových stránok [4]. Pred touto extrakciou je použitý nástroj crawler (viď prechádzajúca časť) na stiahnutie požadovaných webových stránok. Tieto stiahnuté dokumenty dostane na vstup konkrétna extrakčná metóda tvoriaca extrakciu informácií, ktorá postupne prechádza všetky dokumenty a hľadá

[4]Obrázok prevzatý z: https://medium.com/@nonuruzun/algorithms-ceafa828f175, navštívené: 5. 12. 2018.

v nich výskyt požadovaných dát. Pri nájdení výskytu sa tieto dáta extrahujú a postupne sa ukladajú napr. do databázy.

Na nasledujúcom obrázku 3.3 je znázornená bloková schéma extrakcie dát z HTML dokumentov. Z architektúry je zrejmé, že vstupom pre konkrétnu extrakčnú metódu sú HTML dokumenty. Tieto dokumenty sa postupne spracovávajú a získavajú sa z nich dáta. Následne sa tieto dáta ukladajú buď do databázy, alebo do súborov pre uchovávanie a prenášanie dát ako je napr. XML.



Obr. 3.3: Bloková schéma pre extrakciu dát z HTML dokumentov

V nasledujúcej časti práce budú popísané konkrétne riešenia používané ako extrakčné metódy z obrázka 3.3.

Manuálna extrakcia

Pri manuálnej extrakcii HTML dokumentov nie je použitá žiadna automatická metóda, alebo aplikácia umožňujúca automatizované získavanie dát. Používateľ cez webový prehliadač pristúpi na webové stránky, z ktorých chce získať dáta. To je možné jednoduchým kopírovaním a vkladaním potrebných častí webových stránok do úložiska. Takto sa dajú získať textové dáta, obrázky a pod.

Táto metóda je efektívna iba v prípade potreby získania malého množstva dát z pár webových stránok. V opačnom prípade sa jedná o neefektívnu a časovo náročnú metódu. Napr. navštíviť všetky webové stránky celého elektronického obchodu, ktorý ponúka niekoľko tisíc produktov a získať z každého produktu dáta (názov, popis, cena a pod.) nie je efektívne a bolo by to časovo náročné.

Extrakcia pomocou Wrapperu

Wrapper je metóda, ktorá získava dáta z HTML dokumentov na vstupe, pričom spracováva jednotlivé elementy. Touto metódou sa získajú požadované dáta a nepotrebné časti ako sú zbytočné HTML tagy, reklamy a nepotrebný text sa ignorujú [19]. Tieto dáta môžu následne byť vstupom pre nejakú aplikáciu, alebo sa ukladajú do databázy, XML súborov a pod.

Implementácia wrappera môže byť rôzna. Jednou z možností je, že programátor prispôsobuje wrapper konkrétnej webovej stránke, pričom na to používa napr. regulárne výrazy pre získanie potrebných dát.

Ďalšou možnosťou je implementovať wrapper podľa niektorej zo šiestich možných tried, ktoré sú delené podľa zdroja [19]. Pre lepšie pochopenie týchto tried si uvedieme nasledujúci príklad, ktorý je znázornený na obrázku 3.4 a na ktorom je HTML dokument a jeho odpovedajúca webová stránka ako vyzerá v prehliadači. V dokumente a na stránke sú informácie o niektorých mestách na Slovensku a počte obyvateľov žijúcich v nich.

Populácia miest na Slovensku

- **Trenčín** - *54 916*
- **Nitra** - *79 125*
- **Žilina** - *81 094*
- **Trnava** - *65 382*

a) Stránka v prehliadači

```
1   <!DOCTYPE html>
2   <html lang="sk">
3   <head>
4      <meta charset="UTF-8">
5      <title>Mestá a obyvatelia</title>
6   </head>
7   <body>
8      <h3>Populácia miest na Slovensku</h3>
9      <ul>
10        <li><b>Trenčín</b> - <i>54 916</i></li>
11        <li><b>Nitra</b> - <i>79 125</i></li>
12        <li><b>Žilina</b> - <i>81 094</i></li>
13        <li><b>Trnava</b> - <i>65 382</i></li>
14     </ul>
15  </body>
16  </html>
```

b) Zdrojový HTML dokument

Obr. 3.4: Ukážka HTML kódu a webovej stránky pre počty obyvateľov v mestách

Ako si môžete všimnúť na predchádzajúcej ukážke, jednotlivé názvy miest sa nachádzajú vo vnútri elementu ** a **. Taktiež všetky počty obyvateľov v mestách sa nachádzajú v elemente ** a **. Cieľom pre wrapper bude získať všetky názvy miest a ich počty obyvateľov a uložiť ich do databázy. Ostatné informácie, ktoré sa nachádzajú na stránke nie sú potrebné, a preto ich wrapper bude ignorovať. Nasledujúca časť čerpá informácie zo zdroja [19].

Jednotlivé triedy wrapperov:

- **LR (Left-Right)** – je základná trieda, ktorá používa dva oddeľovače na extrakciu jednotlivých častí: *left* a *right*. Pomocou oddeľovačov sa vytvoria konkrétne pravidlá, ktoré určujú z ktorých elementov sa získajú dáta. Každé vytvorené pravidlo musí mať oba oddeľovače. Keď wrapper prechádza HTML dokument a narazí na oddeľovač *left*, tak získa všetky dáta, ktoré nasledujú za ním a skončí, keď narazí na oddeľovač *right*. Takto sa postupne prejde celý HTML dokument. Wrapper sa môže skladať z jedného a viac pravidiel, viď vysvetlenie nižšie na príklade.

  Na konkrétnom príklade z obrázka 3.4 budú extrakčné pravidlá pre LR wrapper vyzerať nasledovne:

  - $WRAP_{LR} = \{[< b >, < /b >], [< i >, < /i >]\}$
  - Prvé pravidlo P_1 má oddeľovače: $l_1 =$ ” $< b >$ ” , $r_1 =$ ” $< /b >$ ”
  - Druhé pravidlo P_2 obdobne: $l_2 =$ ” $< i >$ ” , $r_2 =$ ” $< /i >$ ”

Teda prvé pravidlo postupne extrahuje jednotlivé názvy miest, zatiaľ čo druhé pravidlo sa stará o získanie informácií o počte obyvateľov. Takto sa získajú potrebné dáta z oboch pravidiel:

- $P_1 = \{$"*Trenčín*" , "*Nitra*" , "*Žilina*" , "*Trnava*"$\}$
- $P_2 = \{$"*54 916*" , "*79 125*" , "*81 094*" , "*65 382*"$\}$

Výsledné dáta sa získajú postupným spájaním oboch množín P_1 a P_2. Takto sa spojí prvý prvok z množiny P_1 s prvým prvkom množiny P_2, druhý s druhým atď.

Keďže nie je bližšie špecifikované z ktorých častí má táto trieda wrappera extrahovať dáta, môže sa stať, že začne od nesprávnej časti, a preto získava nesprávne dáta. Toto rieši nasledujúca trieda wrappera HLRT popísaná nižšie.

- **HLRT (Head-Left-Right-Tail)** – ide o rozšírenie triedy LR pričom pridáva ďalšie dva oddeľovače *head* a *tail*. Tieto oddeľovače riešia problém triedy LR s extrakciou nepotrebných častí. Pomocou nich sa vymedzí konkrétna časť HTML dokumentu a v tejto časti už prebieha klasický LR wrapper. Oddeľovač *head* teda znamená odkiaľ sa má začať s extrakciou a oddeľovač *tail* určuje, kde sa má skončiť.

  Na konkrétnom príklade 3.4 budú prvé dve extrakčné pravidlá pre HLRT wrapper vyzerať rovnako ako pre LR triedu a pridajú sa ku nim dve nové:

  - $WRAP_{HLRT} = \{< ul >, < /ul >,\ [< b >, < /b >], [< i >, < /i >]\}$
  - Tretie pravidlo P_3 má oddeľovač: $h =$ " $< ul >$ "
  - Štvrté pravidlo P_4 má oddeľovač: $t =$ " $< /ul >$ "

  V tomto prípade HLRT wrapper vyhľadá prvý výskyt oddeľovača *<ul>* v HTML dokumente, ktorý znamená začiatok extrakcie. Potom začne postupne vyhľadávať oddeľovače pre pravidlá P_1 a P_2. Takto narazí na ** a začne extrahovať informácie, pokiaľ nenarazí na **. Potom sa snaží spracovať druhé pravidlo a obdobne získa informácie ktoré sú v elemente ** a **. Takto sa prechádza celý dokument a algoritmus sa ukončí, keď sa spracuje ukončujúci oddeľovač *
*.

- **OCLR (Open-Close-Left-Right)** – táto trieda zabraňuje extrahovaniu nepotrebných informácií medzi dvojicami oddeľovačov, pričom v jadre pracuje podobne ako LR wrapper. Pred každou dvojicou *left*, *right* oddeľovačov je na začiatku pridaný ďalší oddeľovač *open* a na konci každej dvojice je oddeľovač *close*. Keď sa spracováva HTML dokument, prvou časťou, ktorá sa začne spracovávať je prvý výskyt oddeľovača *open*. Extrahujú sa informácie medzi *left* a *right*, a potom sa preskočí až na ukončujúci oddeľovač *close*. Toto sa opakuje pokiaľ sú v dokumente oddeľovače *open* a *close*.

  Na konkrétnom príklade budú prvé dve extrakčné pravidlá pre OCLR wrapper vyzerať rovnako ako pre LR triedu a pridajú sa ku nim dve nové:

  - $WRAP_{OCLR} = \{< li >, < /li >,\ [< b >, < /b >], [< i >, < /i >]\}$
  - Tretie pravidlo P_3 má oddeľovač: $o =$ " $< li >$ "
  - Štvrté pravidlo P_4 má oddeľovač: $c =$ " $< /li >$ "

- **HOCLRT (Head-Open-Close-Left-Right-Tail)** – táto trieda kombinuje vyššie popísané triedy OCLR a HLRT. Obsahuje teda 6 druhov oddeľovačov *head*, *tail*, *open*, *close*, *left* a *right*, ktorých význam už bol vysvetlený.

- **N-LR (Nested-Left-Right)** – táto trieda sa zameriava na prácu s dokumentami, ktoré obsahujú vnorovanú štruktúru a sú organizované hierarchicky (anglicky nested documents). Tieto dokumenty nemusia byť typu HTML, pričom ich štruktúra sa dá vyjadriť stromom, ktorý reprezentuje obsah dokumentu. V jadre metódy sa využívajú oddeľovače *left* a *right*.
- **N-HLRT (Nested-Head-Left-Right-Tail)** – pracuje podobne ako N-LR trieda, ibaže je v kombinácii s HLRT triedou.

Vyššie uvedených šesť tried sa dá rozdeliť na tri podtriedy:

1. Triedy, ktoré pracujú s tabulárnymi dokumentami (LR, HLRT, OCLR, HOCLRT), alebo vnorovanými dokumentami (N-LR, N-HLRT).
2. Triedy, ktoré sú *HT*, alebo nie sú *HT*.
3. Triedy, ktoré sú *OC*, alebo nie sú *OC*.

Podľa zdroja [19] boli jednotlivé triedy wrapperov testované na reálnych webových stránkach, ktoré obsahovali rôzne typy informácií a taktiež boli z viacerých oblastí. Takto bolo vybratých 30 reprezentujúcich portálov, ktoré spracovávali wrappery implementované podľa jednotlivých tried. Postupne boli na týchto stránkach otestované všetky triedy vrátane ich jednotlivých kombinácií medzi sebou.

V nasledujúcej tabuľke 3.1 je vidieť akú mali percentuálnu úspešnosť jednotlivé triedy na testovacej sade tridsiatich webových portálov.

Trieda wrappera (prípadne ich kombinácia)	**Úspešnosť [%]**
LR	53
OCLR	53
LR ∪ OCLR	53
LR ∪ HLRT ∪ OCLR ∪ HOCLRT	60
LR ∪ HLRT ∪ OCLR ∪ HOCLRT ∪ N-LR ∪ N-HLRT	70
HLRT	57
HOCLRT	57
HLRT ∪ HOCLRT	57
N-LR	13
N-HLRT	50
N-LR ∪ N-HLRT	53
N-LR ∪ N-HLRT (ktoré nedokázali spracovať ostatné 4 triedy)	25

Tabuľka 3.1: Úspešnosť všetkých tried wrapperov a ich kombinácií[5]

Z vyššie uvedenej tabuľky vyplýva, že najviac sa podarilo získať dáta zo 70 % testovaných webových stránok s využitím všetkých šesť tried wrapperov spomenutých vyššie. Taktiež je možné vidieť, že triedy OCLR a HOCLRT sú rovnako úspešné ako triedy LR a HLRT, z čoho vyplýva, že triedy ktoré sú založené na oddeľovačoch *open* a *close* (*OC*) sú porovnateľné so zmienenými triedami, ktoré ich nepoužívajú. Najhoršie obstáli triedy založené na vnorovaciu štruktúru dokumentu N-LR a N-HLRT, preto nemá význam ich používať na tabulárnu štruktúru dokumentu (avšak dokážu spracovať niektoré webové stránky, čo ostatné triedy nie).

[5]Tabuľka prevzatá zo zdroja [19].

3.3 Dostupné nástroje pre sťahovanie a extrakciu dát

Táto časť sa venuje dostupným nástrojom pre sťahovanie webových stránok z internetu. Taktiež sa tu nachádzajú existujúce nástroje pre extrakciu dát z HTML dokumentov.

Knižnica pre crawler – Crawler4j

Knižnica *Crawler4j* je implementovaná pre platformu Java a je určená na sťahovanie webových stránok a ich obsahu – pracuje ako crawler. Jej zdrojové kódy sú dostupné online pod licenciou *Apache License Version 2.0*. Autorom knižnice *Crawler4j* je Yasser Ganjisaffar [11].

Crawler4j umožňuje viacvláknové sťahovanie webových stránok. Je možné nastaviť rôzne filtre pre sťahovanie, čím sa dajú zo sťahovania vylúčiť napr. súbory s kaskádovými štýlmi (CSS), JavaScript súbory, multimediálne súbory a pod. Tiež sa dá nastaviť, aby sa sťahovali iba súbory, ktoré sú zo špecifikovanej domény. Nastaviť sa dá aj počet vlákien pre paralelné sťahovanie a priečinok v súborovom systéme, kde sa majú ukladať stiahnuté dáta. Meniť sa dá maximálna hĺbka vnorenia a taktiež maximálny počet stiahnutých webových stránok. Dá sa nastaviť sťahovanie binárneho obsahu ako sú multimediálne súbory. Umožňuje meniť oneskorenie medzi jednotlivými HTTP požiadavkami na stiahnutie. Zo stiahnutého HTML dokumentu získa všetky hypertextové prepojenia, ktoré vyhovujú požadovaným filtrom a tie sa môžu ďalej spracovávať. Sťahovanie je možné aj zastaviť a neskôr v ňom pokračovať.

Na oficiálnej webovej stránke projektu *Crawler4j* [11] sa nachádzajú aj príklady použitia. Medzi nimi sa nachádza príklad pre základnú demonštráciu činnosti, ďalej tam je príklad crawlera pre sťahovanie iba obrázkov, viacvláknové použitie a ďalšie.

Knižnicu je možné pridať do projektu pomocou nástrojov pre správu, riadenie a automatizáciu nasadzovania aplikácie Maven a Gradle. Je možné ju využiť na sťahovanie dokumentov ako sú HTML, CSS, JS a pod. Taktiež sa dá použiť na sťahovanie binárneho obsahu webových stránok, ako sú multimediálne súbory.

Knižnica na extrakciu dát – Jsoup

Jednou z možností pre sťahovanie webových stránok a následné extrahovanie informácií z HTML dokumentov na platforme Java je knižnica *Jsoup*. Jedná sa o voľne šíriteľnú knižnicu so zverejneným zdrojovým kódom vydanú pod licenciou *MIT*. Jej autorom je Jonathan Hedley a je dostupná online na oficiálnych webových stránkach [14].

Medzi základnú funkcionalitu tejto knižnice patrí:

- sťahovanie webových stránok (HTML dokumentov) z internetu,
- extrahovanie informácií z HTML dokumentov cez URL, uložených v súborovom systéme, alebo ako predaný reťazec (string),
- extrakcia je umožnená postupným prechádzaním *Document Object Model* (DOM), alebo s využitím CSS selektorov,
- umožňuje manipulovanie s jednotlivými HTML elementami, atribútami týchto elementov a textov v dokumente,
- umožňuje vyčistenie HTML dokumentu,

- knižnicu je možné pridať do projektu pomocou nástrojov pre správu, riadenie a automatizáciu nasadzovania aplikácie Maven a Gradle.

Na oficiálnych stránkach je dostupná aj dokumentácia s príkladmi použitia nazývaná *jsoup cookbook*. Nachádzajú sa v nej príklady ako pracovať s knižnicou. Je tam ukázaná voľba vstupu HTML dokumentu (URL, súbor, reťazec). Ďalej ukazuje ako prebieha extrakcia pomocou *DOM*, alebo selektorov. Taktiež zobrazuje ako sa manipuluje s daným dokumentom – zmena atribútov, obsahu elementov a pod. Vďaka týmto ukážkam sa zjednodušuje práca s knižnicou a urýchľuje implementácia.

Knižnica je vhodná na implementáciu extrakčných metód pre HTML dokumenty, pričom môže nahrádzať napr. implementáciu pomocou regulárnych výrazov. Čo sa týka sťahovania HTML dokumentov nie je vhodná, pretože v sebe nemá podporujúcu funkcionalitu pre viacvláknové sťahovanie, oneskorenie medzi jednotlivými HTTP požiadavkami, obmedzenie na počet stiahnutých dokumentov a pod.

Kapitola 4

Analýza a návrh aplikácie

Kapitola sa zaoberá analýzou a návrhom aplikácie pre definíciu a správu extrakčných úloh v prostredí Java EE. Nachádza sa tu neformálna špecifikácia celej aplikácie, kde je zhrnutá funkcionalita aplikácie. Sú tu popísane funkčné požiadavky kladené na aplikáciu. Nasledujú diagramy prípadov použitia (Use Case Diagram), ktoré znázorňujú funkcionalitu a aké operácie môže používateľ v aplikácií vykonávať. Vybrané prípady použitia majú vytvorenú aj konkrétnu špecifikáciu prípadu. Časť sa venuje aj návrhu relačnej databáze pre uchovávanie dát na strane servera a obsahuje Entity Relationship Diagram (ER diagram), znázorňujúci model tejto databázy. Kapitola popisuje aj návrh diagramov tried pre aplikáciu. Nachádzajú sa tu požiadavky na grafické používateľské rozhranie a taktiež je vytvorený popísaný návrh tohto rozhrania. Ďalšiu časť tejto kapitoly tvorí bloková schéma aplikácie, na ktorej je znázornená a vysvetlená základná funkcionalita celej aplikácie.

4.1 Neformálna špecifikácia aplikácie

Webová aplikácia pre definíciu a správu extrakčných úloh bude navrhnutá a implementovaná pre Java EE platformu a bude prístupná cez webové používateľské rozhranie. Jej jednotlivé časti budú navrhnuté do viacvrstvovej architektúry. Bude umožňovať registráciu nového používateľa, a tak vytvorenie jeho účtu. Pod týmto účtom mu bude umožnené prihlásiť sa do aplikácie. Bez prihlásenia nebudú používateľovi sprístupnené žiadne funkcie aplikácie.

Po prihlásení používateľa do aplikácie mu budú sprístupnené všetky funkcie pre definíciu a správu extrakčných úloh. Používateľ si bude môcť meniť nastavenia účtu, ako sú kontaktné údaje, prípadne zmeniť heslo. Ďalej bude mať prístup do správy jeho extrakčných úloh. Tam mu bude umožnené vytvoriť novú úlohu, kde bude môcť nastaviť viaceré parametre ako je vstupná webová stránka (*seed page*), požadované dáta pre extrakciu, oneskorenie medzi HTTP požiadavkami a pod. Jednotlivé úlohy bude môcť vytvoriť, spustiť, zastaviť a zmazať. Pri každej úlohe budú dostupné podrobnejšie informácie. Používateľ bude mať možnosť zobraziť si záznamy spustení extrakčných úloh. Počas aktuálne prebiehajúcej úlohy sa budú dať zobraziť informácie o počte stiahnutých webových stránok, veľkosti prenesených dát, dĺžke sťahovania a pod. Podobné informácie budú dostupné aj o ukončenej, prípadne zastavenej úlohe.

Aplikácia bude umožňovať paralelné spracovanie viacerých extrakčných úloh jedného používateľa, pričom sa bude môcť prepínať medzi nimi a zobrazovať aktuálne informácie o priebehu sťahovania dokumentov a extrakcii dát z nich.

Ďalšou funkcionalitou bude správa uložených súborov s extrahovanými dátami. Tieto súbory sa budú ukladať do databázy a používateľ si ich bude môcť zobraziť, zmazať a stiahnuť. Jednotlivé súbory budú dostupné aj pomocou unikátnej URL adresy. Pristúpiť k jednotlivým súborom bude môcť prihlásený používateľ, alebo každý, kto má URL s unikátnou adresou daného súboru.

Z hľadiska bezpečnosti bude potrebné navrhnúť a implementovať zabezpečenie pre autentizáciu a autorizáciu používateľa podľa jednotlivých rolí. Dostupné role v aplikácii budú dve: bežný používateľ a administrátor. V aplikácii treba počítať s viacerými používateľskými účtami. Administrátorský účet bude iba jeden pre celú aplikáciu.

Administrátor bude mať dostupné všetky funkcie aplikácie pre bežného používateľa a okrem toho mu budú umožnené nasledovné operácie. Správa účtov všetkých používateľov ako je deaktivácia účtu a zmazanie účtu. Spravovať extrakčné úlohy a súbory s extrahovanými dátami, pričom bude mať prístup ku všetkým informáciám o úlohách, ich záznamoch spustení a súborom s dátami. Ďalej bude môcť zobraziť štatistický prehľad celej aplikácie, kde budú zobrazené informácie o počte používateľoch a ich aktivite, počte vytvorených úloh, koľko súborov používatelia majú, ich veľkosti a pod.

Zvolená metóda pre extrakciu dát z HTML dokumentov je HLRT wrapper, ktorý bol bližšie popísaný v časti 3.2. Ďalšie informácie o priebehu extrakcie budú ešte popísané v časti *Implementácia aplikácie*.

4.2 Analýza požiadaviek na aplikáciu

Z analýzy predchádzajúcich neformálnych požiadaviek na aplikáciu vzniknú funkčné požiadavky na ňu. Pod týmto pojmom sa rozumejú jednotlivé akcie, ktoré môže používateľ s aplikáciou vykonávať. Požiadavky budú delené na tri kategórie, ktoré sú popísané nižšie.

Funkčné požiadavky na aplikáciu:

1. **Neprihlásený používateľ**
   - Vytvorenie nového účtu – cez registračný formulár bude možné vytvoriť nový používateľský účet. Okrem kontaktných údajov sa tu bude zadávať email, ktorý slúži ako jednoznačná identifikácia používateľa a heslo.
   - Prihlásenie do aplikácie – pomocou používateľského emailu a hesla sa používateľ bude môcť prihlásiť do aplikácie a podľa typu účtu (bežný používateľ, alebo administrátor) mu budú pridelené oprávnenia.
2. **Prihlásený používateľ (nie administrátor)**
   - Správa účtu:
     - Zmena hesla – používateľ si môže zmeniť svoje prihlasovacie heslo.
     - Zmena kontaktných informácií – používateľ si môže zmeniť kontaktné údaje, ktoré má v profile. Taktiež je umožnená aj zmena emailu.
     - Odhlásenie z aplikácie.
   - Správa extrakčných úloh:
     - Vytvorenie novej úlohy – pri vytváraní novej úlohy používateľ zadá jej názov a vstupnú webovú stránku (*seed page*) ako URL. Bude môcť nastaviť

parametre pre sťahovanie HTML dokumentov: oneskorenie HTTP požiadaviek, maximálny počet stiahnutých súborov, maximálnu veľkosť stiahnutých dát, maximálnu dĺžku sťahovania, obmedzenie na doménu zo zadanej URL. Taktiež nastaví HLRT wrapper, ktorý bude extrahovať požadované dáta.
  - Spustenie úlohy – každá úloha môže byť spustená viackrát. Pri každom spustení sa vytvorí nový záznam o jej spustení.
  - Zastavenie úlohy.
  - Zmazanie úlohy – zmazať sa dá iba úloha, ktorá nie je práve aktívna.
  - Zobraziť úlohy – používateľ si môže prezerať svoje úlohy a podrobnejšie informácie o nich.
  - Editovať úlohu – budú sa dať zmeniť nastavenia extrakčnej úlohy podobne ako pri jej vytváraní.
  - Zobraziť záznamy spustenia – pri každom zázname budú dostupné informácie o priebehu úlohy. Zobrazí sa tak počet stiahnutých súborov, veľkosť stiahnutých dát a pod.
  - Zmazať záznam spustenia.
- Správa uložených súborov:
  - Zobraziť súbor – zobrazí sa súbor s extrahovanými dátami. Bude sa dať zobraziť aj cez unikátnu URL adresu.
  - Zmazať súbor.
  - Stiahnuť súbor.
- Štatistický prehľad:
  - Zobraziť štatistiky – zobrazia sa štatistiky o používateľových úlohách, súboroch atď.

3. Prihlásený administrátor

- Okrem funkcionality bežného používateľa môže:
  - Zobraziť používateľské účty – prehľad všetkých účtov v aplikácii.
  - Zmazať používateľský účet.
  - Deaktivovať používateľský účet – keď je účet deaktivovaný, používateľ sa pod ním nebude môcť prihlásiť do aplikácie.
  - Prístup k súborom – administrátor bude môcť pristúpiť ku všetkým súborom a ich URL odkazom.
  - Zobraziť štatistiky celej aplikácie.
  - Zobraziť všetky úlohy.
  - Zobraziť všetky záznamy spustení.

Z vyššie uvedených funkčných požiadaviek na aplikáciu vzniknú diagramy prípadov použitia aplikácie, ktoré sú popísané v nasledujúcej časti.

4.3 Diagramy prípadov použitia

Diagram prípadov použitia (Use Case Diagram) je diagram z jazyka UML, ktorý umožňuje modelovanie možných situácií, ktoré sa dajú v danom systéme vykonávať. Jeho hlavnými

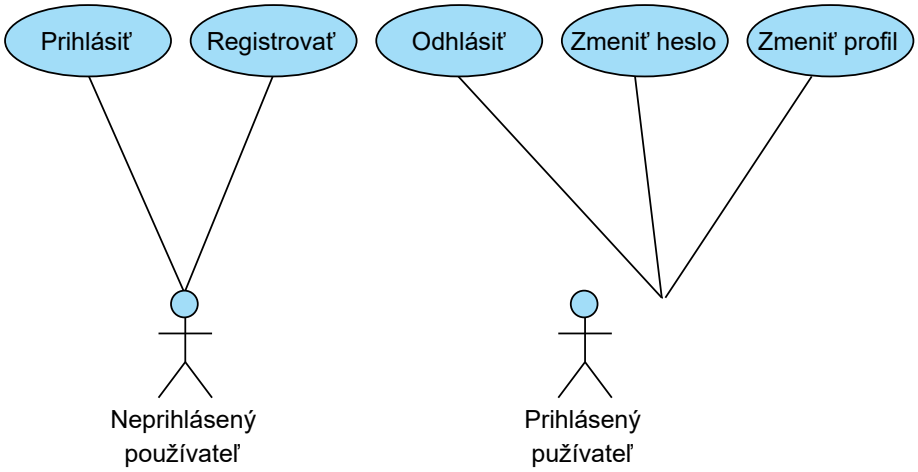
časťami sú účinkujúci v systéme, ktorý vykonávajú akcie a jednotlivé prípady použitia reprezentujúce funkcionalitu systému. Medzi účinkujúcimi a prípadmi použitia vznikajú asociácie (väzby). Tento diagram umožňuje zobrazenie systému na konceptuálnej úrovni a zachytáva tak jeho funkcionalitu. Dá sa z neho pochopiť základná funkcionalita systému bez ďalších potrebných informácií [3].

Celý návrh a potom aj implementácia aplikácie bude rozdelená na tri iterácie. Po dokončení každej iterácie vznikne funkčný prototyp, ktorý sa následne otestuje. Keď je prototyp otestovaný a funkčný, tak sa prejde na ďalšiu iteráciu. Po dokončení poslednej iterácie bude nasledovať testovanie celej aplikácie.

V nasledujúcej časti sú popísané jednotlivé iterácie. Ku každej iterácii je vytvorený diagram prípadov použitia a konkrétna špecifikácia vybraných prípadov použitia.

Prvá iterácia – správa účtu

Prvá iterácia modeluje správu používateľského účtu. Jedná sa o registráciu nového používateľa, prihlásenie používateľa do aplikácie a odhlásenie používateľa z aplikácie. Používateľovi bude umožnená zmena kontaktných informácií, emailu a hesla. Taktiež v tejto iterácii sa implementuje zabezpečenie aplikácie, ktoré sa skladá z autentizácie používateľa a jeho následnej autorizácii podľa jednotlivých rolí. Na nasledujúcom obrázku 4.1 je znázornený diagram prípadov použitia pre prvú iteráciu.



Obr. 4.1: Diagram prípadov použitia pre prvú iteráciu

Z diagramu prípadov použitia na vyššie uvedenom obrázku je zrejmá výsledná funkcionalita aplikácie po implementovaní prvej iterácie.

Nasleduje špecifikácia vybraných prípadov použitia z predchádzajúceho obrázku. Z tejto iterácie boli vybrané dva prípady, ktoré sú podrobnejšie popísané v nasledujúcich tabuľkách.

Prípad použitia	Registrácia používateľa
Identifikátor	UC1
Účastníci	Neregistrovaný používateľ Systém
Vstupné podmienky	
Kroky	1. Používateľ zvolí možnosť registrácie. 2. Používateľ vyplní registračný formulár. 3. Systém vytvorí nový používateľský účet podľa vyplnených dát. Nastaví tomuto účtu rolu bežného používateľa. Aktivuje tento nový účet.
Výnimky	Pre krok 2: 2.1 Neboli vyplnené povinné údaje vo formulári. 2.2 Vyplnené údaje nespĺňajú dané podmienky (dĺžka hesla a pod.). 2.3 Zadaný email sa už používa (existuje používateľský účet so zadaným emailom). 2.4 Zadané heslá sa nezhodujú. Dôsledok výnimiek pre krok 2: • Vytvorenie účtu neprebehlo. Používateľ je o tom informovaný.
Výstupný stav	Vytvoril sa nový používateľský účet a vložil sa do databázy.
Alternatívny tok	Používateľ môže kedykoľvek opustiť obrazovku vytvárania nového používateľského účtu.

Prípad použitia	Zmena hesla používateľa
Identifikátor	UC2
Účastníci	Registrovaný používateľ Systém
Vstupné podmienky	Používateľ je prihlásený do systému
Kroky	1. Používateľ zvolí možnosť úpravy profilu. 2. Používateľ vyplní formulár pre zmenu hesla. 3. Systém upraví používateľský účet podľa vyplnených dát.
Výnimky	Pre krok 2: 2.1 Vyplnené údaje nespĺňajú dané podmienky (dĺžka hesla a pod.). 2.2 Zadané heslá sa nezhodujú. Dôsledok výnimiek pre krok 2: • Zmena hesla neprebehla a používateľ je o tom informovaný.
Výstupný stav	Heslo pre používateľský účet bolo zmenené a uložené v databázy.
Alternatívny tok	Používateľ môže kedykoľvek opustiť obrazovku editácie profilu.

Druhá iterácia – správa extrakčných úloh a súborov

Druhá iterácia modeluje správu extrakčných úloh a súborov s extrahovanými dátami. Konkrétne sa jedná o vytvorenie novej extrakčnej úlohy, jej spustenie, zastavenie a zmazanie. Ďalej sa budú dať zobraziť informácie o úlohe, jej záznamoch spustení a priebehu. Umožnená bude editácia nastavení úlohy. Ďalej správa uložených súborov s extrahovanými dátami. Bude sa dať zobraziť súbor, zmazať a stiahnuť. Tiež pribudne štatistický prehľad koľko má používateľ úloh, súborov a pod.

Na nasledujúcom obrázku 4.2 je znázornený diagram prípadov použitia pre druhú iteráciu. Modrou farbou sú znázornené prípady použitia z prvej iterácie, zelenou farbou sú tie, ktoré pribudli v druhej iterácii.

Obr. 4.2: Diagram prípadov použitia pre druhú iteráciu

Z diagramu prípadov použitia na vyššie uvedenom obrázku je zrejmá výsledná funkcionalita aplikácie po implementovaní druhej iterácie.

Rovnako ako v prvej iterácii si teraz uvedieme dve špecifikácie vybraných prípadov použitia z predchádzajúceho obrázku. Tieto prípady sú podrobnejšie popísané v nasledujúcich dvoch tabuľkách.

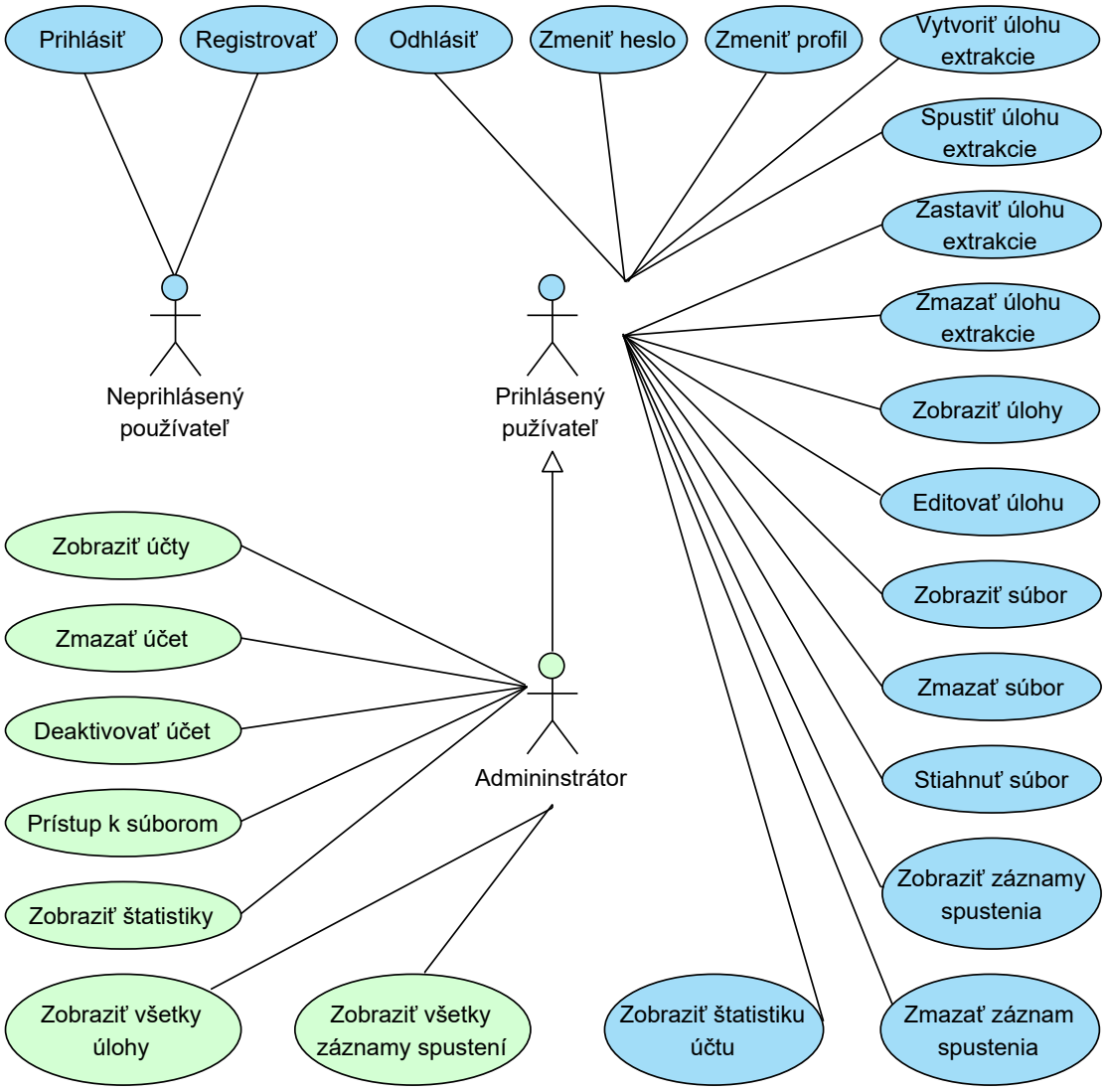
Prípad použitia	Vytvorenie novej extrakčnej úlohy
Identifikátor	UC3
Účastníci	Registrovaný používateľ Systém
Vstupné podmienky	Používateľ je prihlásený do systému
Kroky	1. Používateľ zvolí možnosť vytvorenia novej extrakčej úlohy. 2. Používateľ vyplní formulár pre jednotlivé nastavenia extrakčnej úlohy. 3. Systém vytvorí novú extrakčnú úlohu a podľa nastavenia spustenia: a) systém úlohu spustí. b) systém úlohu nespustí.
Výnimky	Pre krok 2: 2.1 Neboli vyplnené povinné údaje vo formulári. 2.2 Vyplnené údaje nespĺňajú dané podmienky. 2.3 Východzia URL adresa nie je dostupná. Dôsledok výnimiek pre krok 2: • Vytvorenie extrakčnej úlohy neprebehlo a používateľ je o tom informovaný.
Výstupný stav	Bola vytvorená nová extrakčná úloha a vložená do databázy. Podľa nastavenia bola po vytvorení spustená, alebo nebola.
Alternatívny tok	Používateľ môže kedykoľvek opustiť obrazovku vytvárania novej extrakčnej úlohy.

Prípad použitia	Zmazanie extrakčnej úlohy
Identifikátor	UC4
Účastníci	Registrovaný používateľ Systém
Vstupné podmienky	Používateľ je prihlásený do systému Používateľ má vytvorenú minimálne jednu extrakčnú úlohu
Kroky	1. Používateľ zvolí možnosť správy extrakčných úloh. 2. Používateľ nájde úlohu, ktorú chce vymazať. 3. Používateľ zvolí možnosť vymazania úlohy. 4. Systém overí, či extrakčná úloha nie je práve spustená a vymaže ju.
Výnimky	Pre krok 3: 2.1 Zvolená úloha je spustená. Dôsledok výnimiek pre krok 2: • Zmazanie extrakčnej úlohy neprebehlo a používateľ je o tom informovaný.
Výstupný stav	Bola vymazaná zvolená extrakčná úloha z databázy.
Alternatívny tok	Používateľ môže kedykoľvek opustiť obrazovku správy úloh.

Tretia iterácia – administrátorský účet

Tretia iterácia modeluje administrátorský účet. Jedná sa o správu používateľov, kde je možné používateľský účet aktivovať, prípadne deaktivovať a zmazať. Ďalšou časťou je správa úloh, kde je umožnené pristúpiť ku všetkým úlohám, ktoré sú v systéme a taktiež k súborom s extrahovanými dátami. Pri každej úlohe si administrátor môže pozrieť jednotlivé záznamy spustení. Poslednou časťou je štatistický prehľad týkajúci sa používateľov, úloh, počtu súborov a ich veľkostí.

Na nasledujúcom obrázku 4.3 je znázornený diagram prípadov použitia pre tretiu iteráciu. Podobne ako v druhej iterácií, sú modrou farbou znázornené prípady použitia získané spojením prvej a druhej iterácie. Zelenou farbou sú nové prípady, ktoré pribudli v tretej iterácii.



Obr. 4.3: Diagram prípadov použitia pre tretiu iteráciu

Na predchádzajúcom obrázku je znázornený celý diagram prípadov použitia po všetkých troch iteráciách, čo umožňuje získanie prehľadu o funkcionalite celej aplikácie.

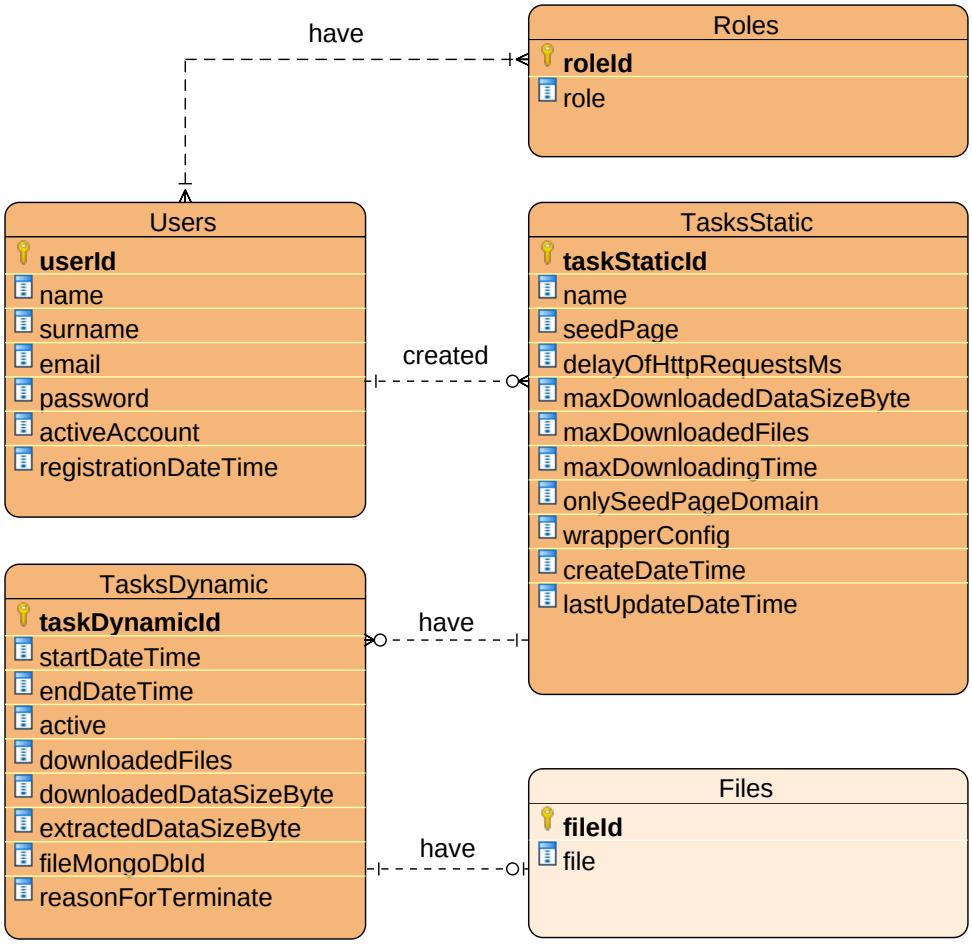
Aj pri tejto poslednej iterácii si uvedieme jednu špecifikáciu vybraného prípadu použitia z predchádzajúceho obrázku, ktorá sa nachádza v nasledujúcej tabuľke.

Prípad použitia	Deaktivácia používateľského účtu
Identifikátor	UC5
Účastníci	Administrátor Systém
Vstupné podmienky	Administrátor je prihlásený do systému V systéme je minimálne jeden používateľský účet
Kroky	1. Administrátor zvolí možnosť správy účtov. 2. Administrátor nájde používateľský účet, ktorý chce deaktivovať. 3. Administrátor zvolí možnosť deaktivácie účtu. 4. Ak má zvolený používateľ spustené nejaké extrakčné úlohy, tak ich systém zastaví a potom deaktivuje zvolený používateľský účet.
Výnimky	
Výstupný stav	Bol deaktivovaný zvolený používateľský účet a zmena sa uložila do z databázy.
Alternatívny tok	Administrátor môže kedykoľvek opustiť obrazovku správy účtov.

4.4 Návrh modelu relačnej databázy

Pre návrh modelu relačnej databázy sa používa Entity Relationship Diagram (ďalej iba ER diagram). Tento diagram modeluje význam dát na abstraktnej úrovni. Dáta sú definované a reprezentované v entitách (reláciách – tabuľkách) a vzťahmi medzi entitami. Entity obsahujú atribúty, ktoré reprezentujú stĺpce tabuľky [2].

Na nasledujúcom obrázku 4.4 je znázornený ER diagram. V aplikácií budú používané dve databázy: MySQL databáza a MongoDB databáza. Entity *Users*, *Roles*, *TasksStatic*, *TasksDynamic* budú relácie v MySQL databáze. Entita *Files* bude ako kolekcia v MongoDB databáze.



Obr. 4.4: ER diagram znázorňujúci uloženie dát v databázach

Z vyššie uvedeného ER diagramu je zrejmé aké informácie sa budú ukladať do oboch databáz. Nasleduje krátky popis každej relácie a kolekcie:

- Relácia *Users* – v tejto tabuľke sa budú ukladať informácie o jednotlivých používateľoch aplikácie. Okrem základných kontaktných informácií o používateľovi nás bude zaujímať aj, či je daný účet aktívny a kedy sa používateľ zaregistroval.
- Relácia *Roles* – v tejto tabuľke budú uložené dostupné role podľa predchádzajúcej analýzy požiadaviek na aplikáciu. Jedná sa o dve role *ROLE_USER* a *ROLE_ADMIN*.
- Relácia *TasksStatic* – táto tabuľka bude uchovávať informácie o vytvorenej extrakčnej úlohe. Jednotlivé atribúty tejto úlohy naznačujú svoj význam. Atribút *wrapperConfig* slúži ako konfigurácia HLRT wrappera pre extrakciu dát z HTML dokumentov.
- Relácia *TasksDynamic* – každá vytvorená úloha môže byť spustená viackrát. Pri každom spustení sa vytvorí nový záznam v relácií *TaskDynamic*, kde sa ukladajú podrobné informácie o úlohe a jej priebehu. Medzi tieto informácie patrí čas spustenia a ukončenia, dĺžka sťahovania a dĺžka extrakcie. Taktiež sa tam ukladajú informácie o veľkosti a počte stiahnutých dokumentov a či je daná úloha práve aktívna.

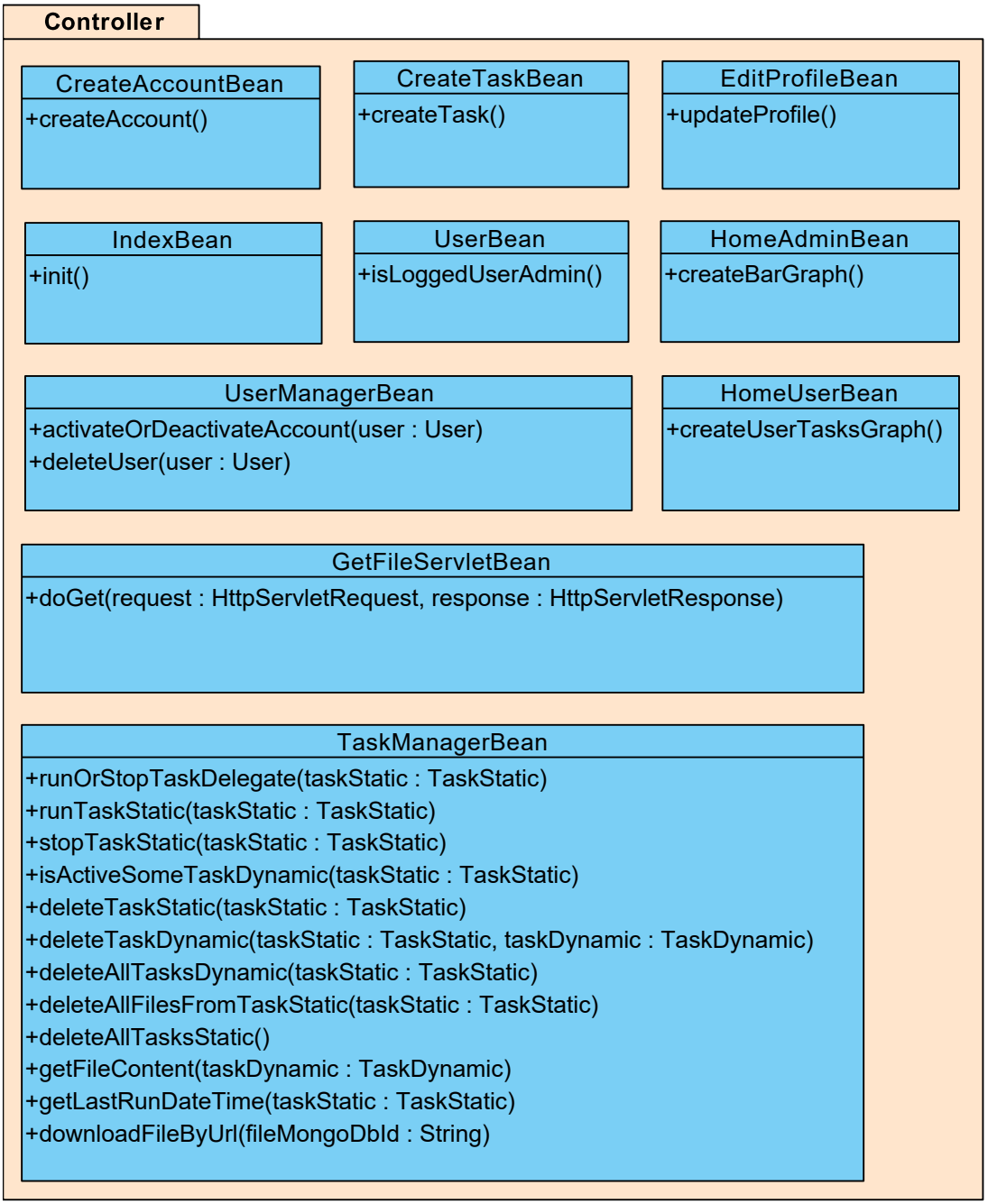
- Kolekcia *Files* – slúži na ukladanie súborov s extrahovanými dátami, ktoré sú výstupom jednotlivých extrakčných úloh. Každý súbor bude mať svoju unikátnu URL adresu na ktorej bude dostupný. Táto adresa sa získa z atribútu *fileId*. Táto kolekcia sa bude nachádzať v MongoDB databáze.

4.5 Diagramy tried

Diagram tried (Class Diagram) je diagram štruktúry patriaci do rodiny UML, ktorý znázorňuje triedy, ich atribúty a operácie. Trieda je základným prvkom tohto diagramu a každá má konkrétny názov. Medzi triedami vznikajú viaceré druhy asociácií, ktoré majú odlišné vlastnosti. Atribúty môžu a nemusia byť upresnené dátovými typmi. Tak isto aj pri operáciach sa môžu, ale nemusia uvádzať parametre, ich dátové typy a typ návratovej hodnoty. Diagram tried patrí medzi základné diagramy pre vytvorenie objektovo orientovaného návrhu systému [25].

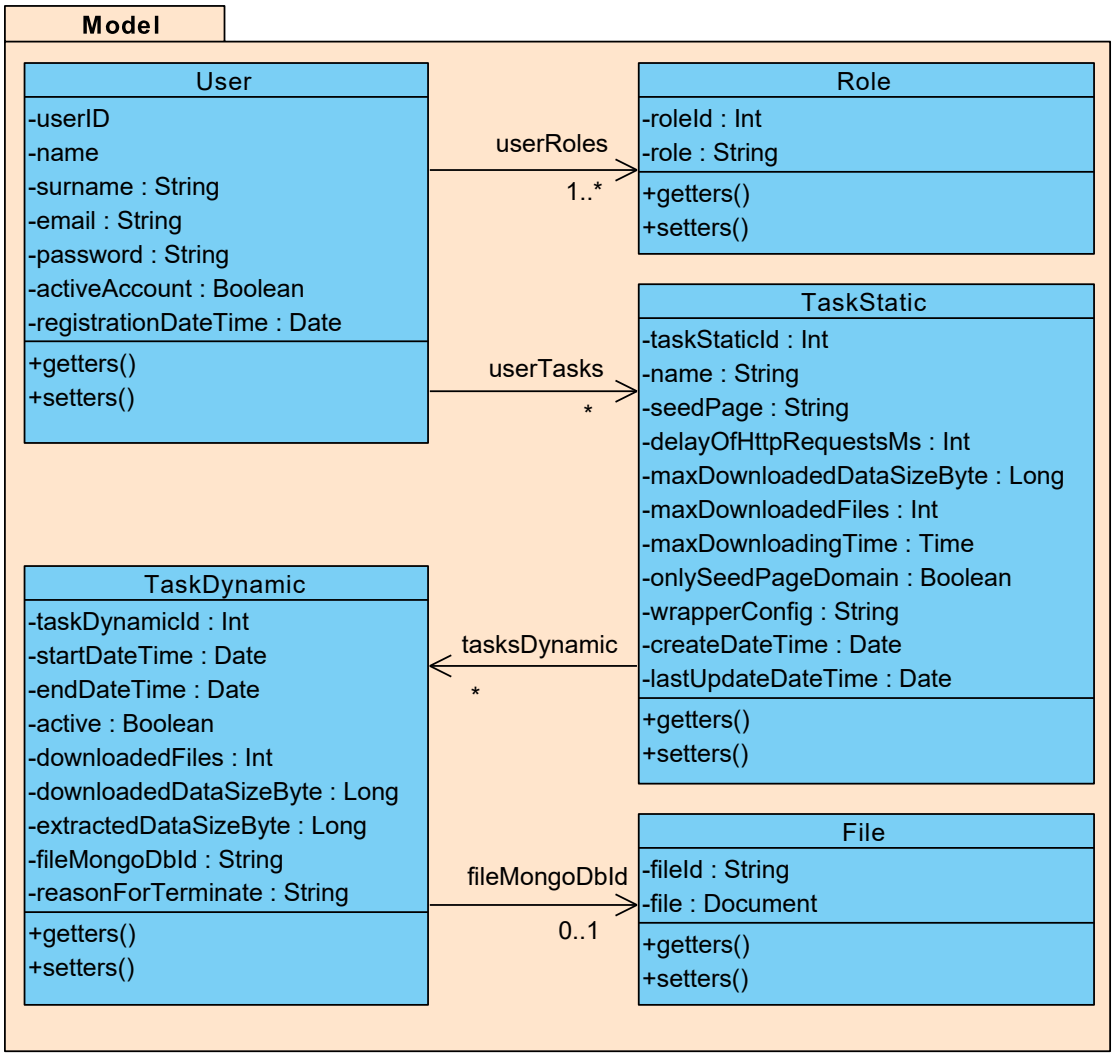
Nasledujú niektoré vybraté návrhy diagramov tried aplikácie pre definíciu a správu extrakčných úloh. Tieto diagramy nemajú uvádzané všetky atribúty a ich operácie. Zachytávajú hlavné a podstatné operácie, ktoré budú implementované. Samozrejme v implementácii budú aj mnohé ďalšie operácie.

Na nasledujúcom obrázku 4.5 je znázornený diagram tried pre balík (package) *Controller*. Balíkom sa rozumie reprezentácia jedného priečinka, ktorý sa nachádza v zdrojových súboroch aplikácie. V každom tomto balíku sú triedy, ktoré majú niečo spoločné. V balíku *Controller* sa nachádzajú všetky triedy majúce súvis s hlavnou funkcionalitou celej aplikácie.



Obr. 4.5: Diagram tried pre balík Controller

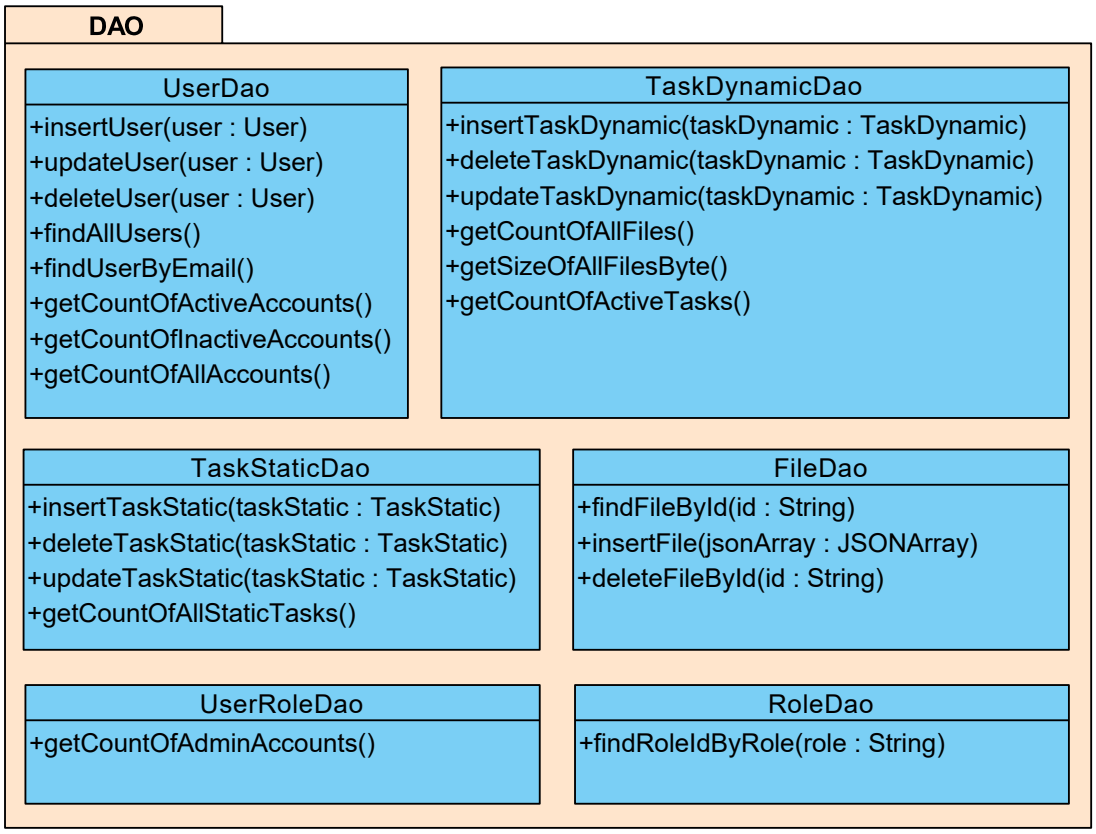
Na nasledujúcom obrázku 4.6 je znázornený diagram tried pre balík *Model*, v ktorom sa nachádzajú všetky triedy pre objektovo-relačné mapovanie (Object-Relational Mapping). Toto mapovanie je medzi uvedenými triedami v Jave a databázou. Každá entita z predchádzajúceho návrhu ER diagramu má vytvorenú jednu odpovedajúcu triedu.



Obr. 4.6: Diagram tried pre balík Model

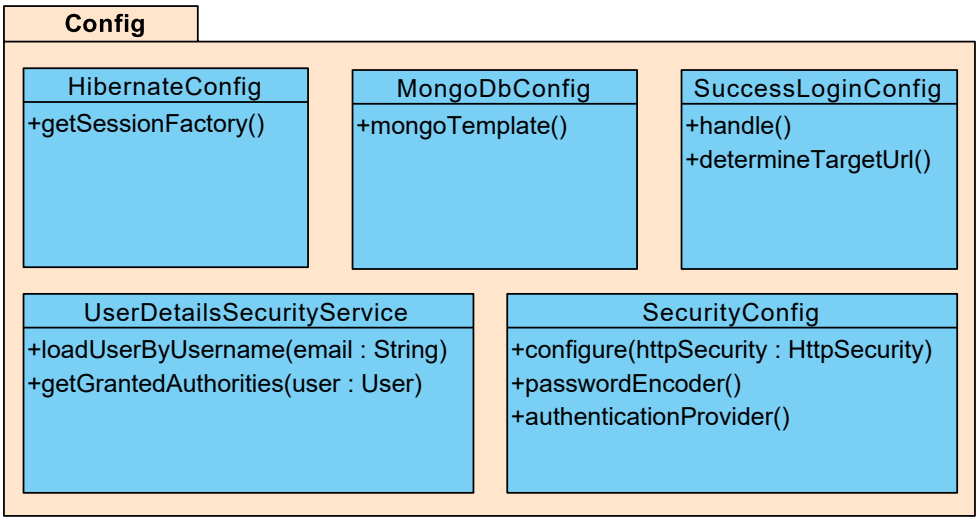
Na nasledujúcom obrázku 4.7 je znázornený diagram tried pre balík *DAO*[1], ktorý zjednocuje všetky triedy používané pre prístup do databázových relácií. Každá entita z predchádzajúceho návrhu ER diagramu má vytvorenú jednu odpovedajúcu DAO triedu.

[1]Data Access Object (DAO) – je objekt, ktorý poskytuje rozhranie pre prístup do databázy. Pomocou mapovaných operácií sa môžu vykonávať dotazy nad databázou.



Obr. 4.7: Diagram tried pre balík DAO

Na nasledujúcom obrázku 4.8 je znázornený diagram tried pre balík *Config*, v ktorom sa nachádzajú všetky konfiguračné triedy pre nastavenie Java EE aplikácie v aplikačnom rámci Spring.



Obr. 4.8: Diagram tried pre balík Config

Podľa vyššie uvedených diagramov tried sa pri implementácii vytvorí štruktúra balíkov zdrojových kódov a následne sa vytvoria triedy podľa návrhu, ich atribúty a metódy. Každá operácia je metóda v Jave, ktorá bude implementovaná.

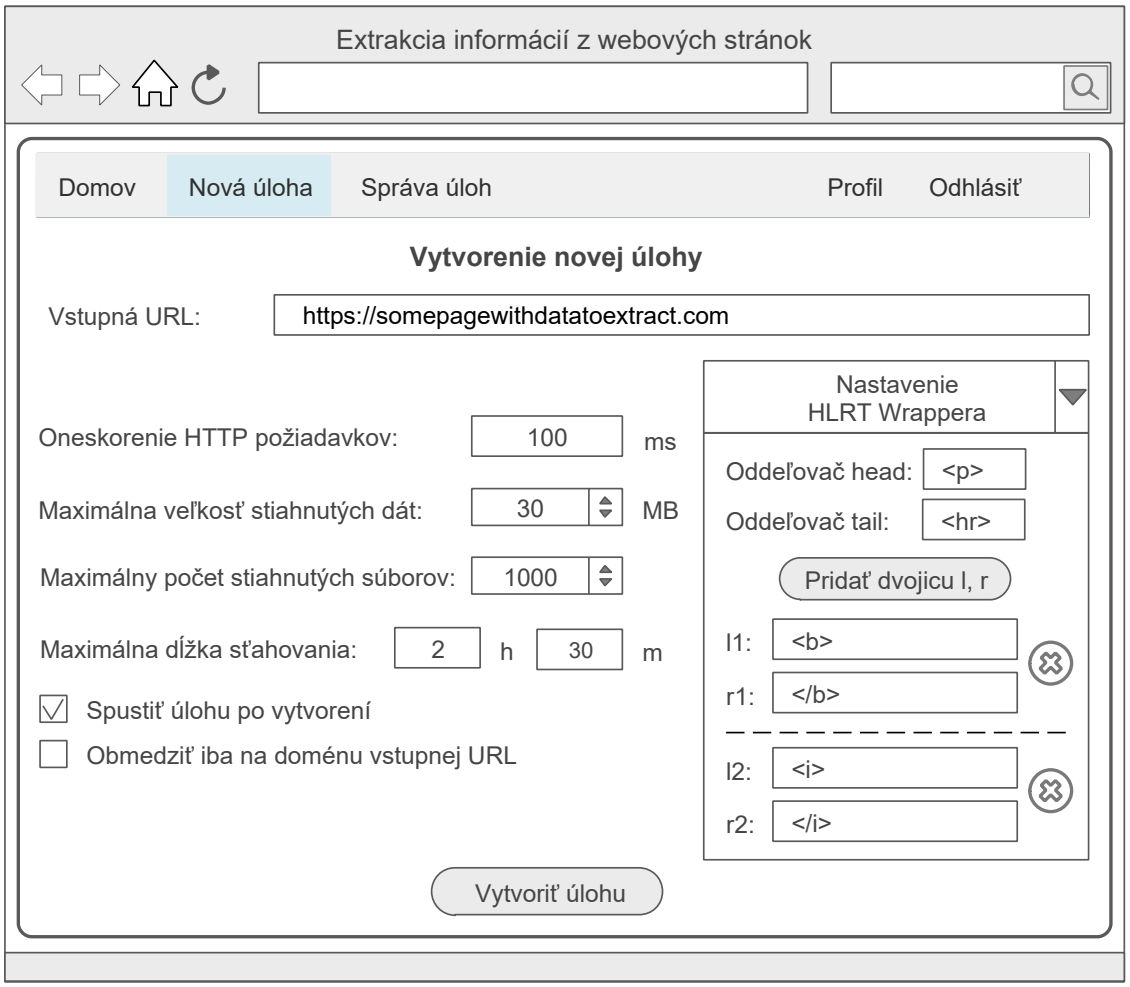
4.6 Návrh grafického používateľského rozhrania

Pri každej webovej aplikácii je dôležitou časťou práve návrh grafického používateľského rozhrania (Graphical User Interface – GUI). Jedným z dnešných typov návrhu webovej aplikácie je vytvorenie takzvanej jedno-stránkovej aplikácie (Single-Page Application – SPA). Tento návrh sa zameriava na vytvorenie intuitívneho rozloženia grafických prvkov na stránke, pričom splňuje naslednové veci. Hlavnou myšlienkou jedno-stránkovej aplikácie je zníženie počtu načítavania jednej webovej stránky. To znamená, že pri každej interakcii s používateľom sa nemusí znovu načítať celý obsah stránky. [18]. To sa docieli tak, že stránka sa celá načíta iba raz a potom sa podľa používateľovej interakcie na nej obnovuje iba určitá časť. Táto obnova sa vykonáva buď iba na strane klienta v prehliadači, alebo sa napr. pomocou technológie AJAX[2] získajú zo servera potrebné dáta na vykreslenie určitej časti stránky. Takto vytvorené webové aplikácie sú používateľsky prívetivejšie a dynamickejšie.

Aplikácia pre definíciu a správu extrakčných úloh bude navrhnutá a implementovaná podľa vyššie popísaného návrhu jedno-stránkovej aplikácie. Rozloženie jednotlivých grafických komponentov bude organizované do jednej hlavnej webovej stránky, z ktorej bude dostupná ostatná funkcionalita systému. Používateľ tak bude mať pri práci s aplikáciou všetky potrebné informácie zobrazené bez prípadného posúvania sa na stránke.

Na nasledujúcom obrázku 4.9 je návrh rozloženia používateľského rozhrania pre prihláseného používateľa pod rolou bežný používateľ. Obrázok zachytáva prípad použitia pre vytvorenie novej extrakčnej úlohy. Dajú sa tam nastavovať rôzne obmedzenia pre úlohu, ako už bolo spomenuté. Hlavnou časťou je práve nastavenie HLRT wrappera, ktorý ovplyvňuje extrahované dáta. Nastavujú sa oddeľovače *head*, *tail* a potom dvojice *left* a *right*. Je možné pridať ľubovoľné množstvo týchto dvojíc pomocou tlačidla *Pridať dvojicu l, r*. Taktiež je možné dvojicu vymazať príslušným tlačidlom.

[2] Asynchronous JavaScript and XML (AJAX) – je technológia využívaná pri webových aplikáciách na asynchrónne (neblokujúce) získanie dát zo servera, prípadne ich vloženie na server.



Obr. 4.9: Návrh GUI pre vytvorenie novej úlohy

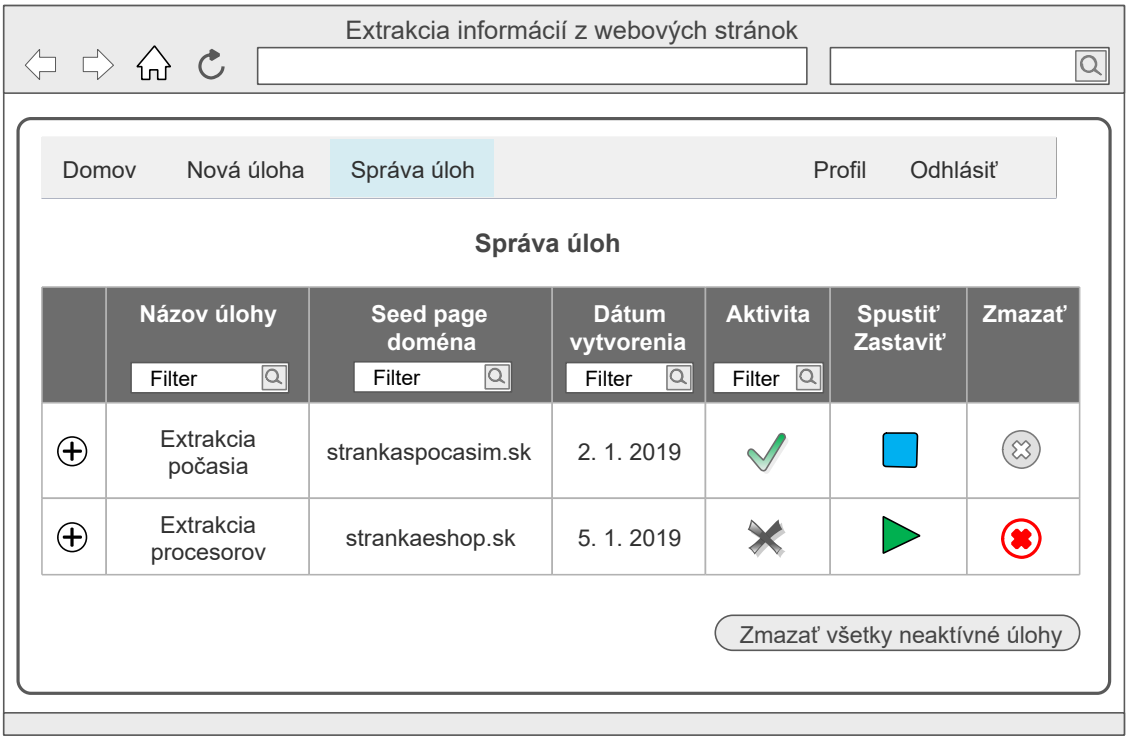
Navigácia po prihlásení je v podobe hlavného menu, ktoré je vždy zobrazené a nachádza sa na vrchnej časti webovej stránky. Z položiek tohto menu sa používateľ dostane do ďalších častí aplikácie.

Popis jednotlivých položiek z hlavného menu:

- Domov – táto časť sa zobrazí po prihlásení používateľa do aplikácie. Na tejto stránke budú zobrazené štatistické údaje týkajúce sa prihláseného používateľa. Medzi tieto údaje patrí počet vytvorených úloh, počet bežiacich úloh, počet uložených súborov, ich veľkosť a pod.
- Nová úloha – umožňuje vytvorenie novej extrakčnej úlohy. Pri vytváraní sa dajú meniť nastavenia viacerých parametrov, ktoré ovplyvňujú získané dáta.
- Správa úloh – zobrazuje prehľad všetkých používateľových úloh. Jednotlivé úlohy môžu byť aktívne a neaktívne. Umožňuje rýchle vyhľadávanie medzi úlohami a zobrazovanie informácií o nich.
- Profil – umožňuje používateľovi meniť kontaktné informácie, email a heslo.

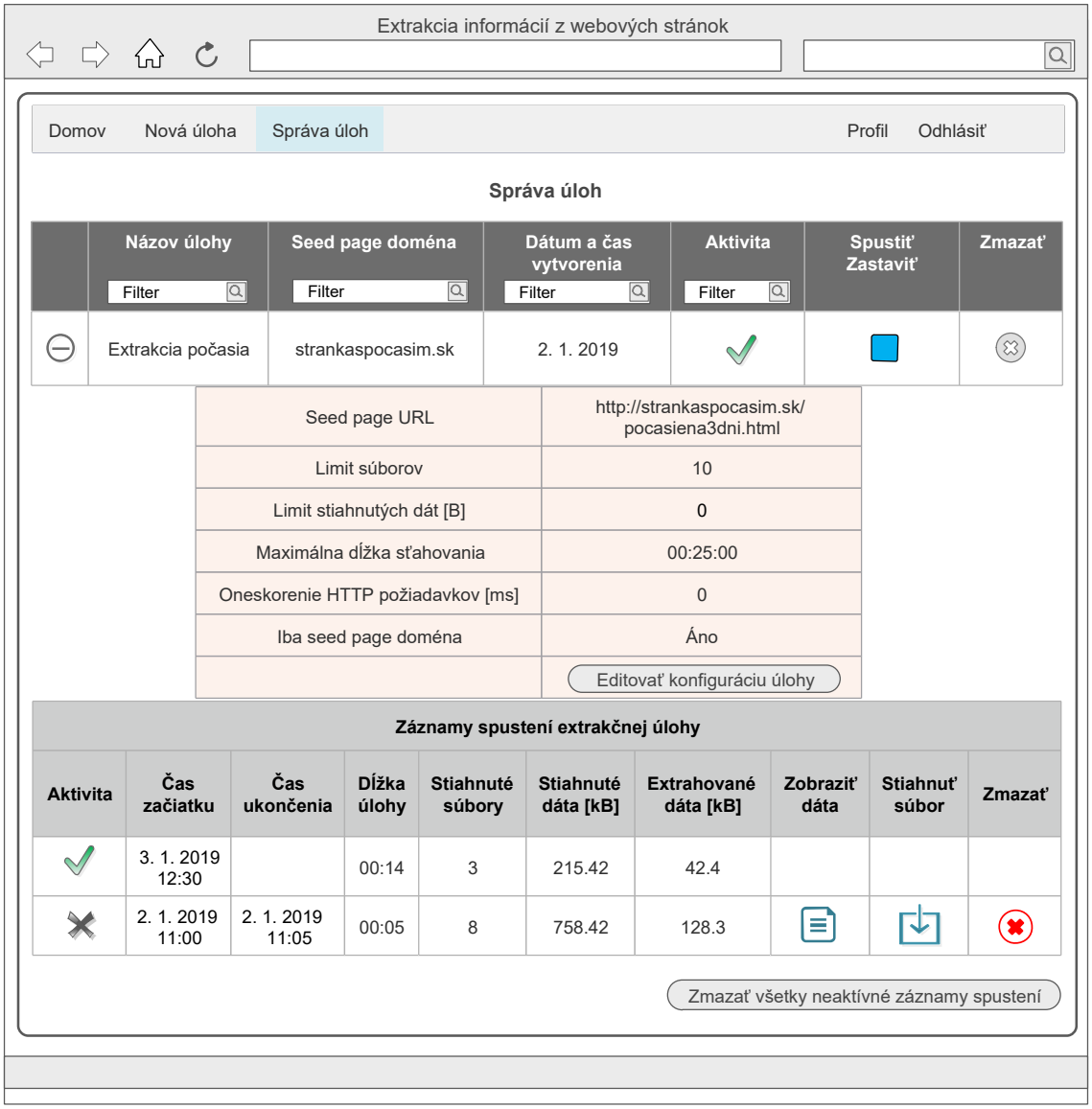
- Odhlásiť – odhlásenie používateľa zo systému.

Na nasledujúcom obrázku 4.10 je návrh rozloženia používateľského rozhrania pre správu používateľových úloh. Tu je možné vidieť iba základne informácie o úlohách. Záznamy v tabuľke sa dajú filtrovať a radiť podľa príslušných stĺpcov. Ak úloha nie je aktívna, tak sa dá spustiť. Naopak, aktívna úloha sa dá zastaviť. Úloha, ktorá nie je aktívna sa môže vymazať. Toto je odlíšené zvýrazneným červeným tlačidlom s ikonou krížika. Ktorá úloha je aktívna je vidieť podľa príslušnej zelenej ikony (✓) v stĺpci *Aktivita*.



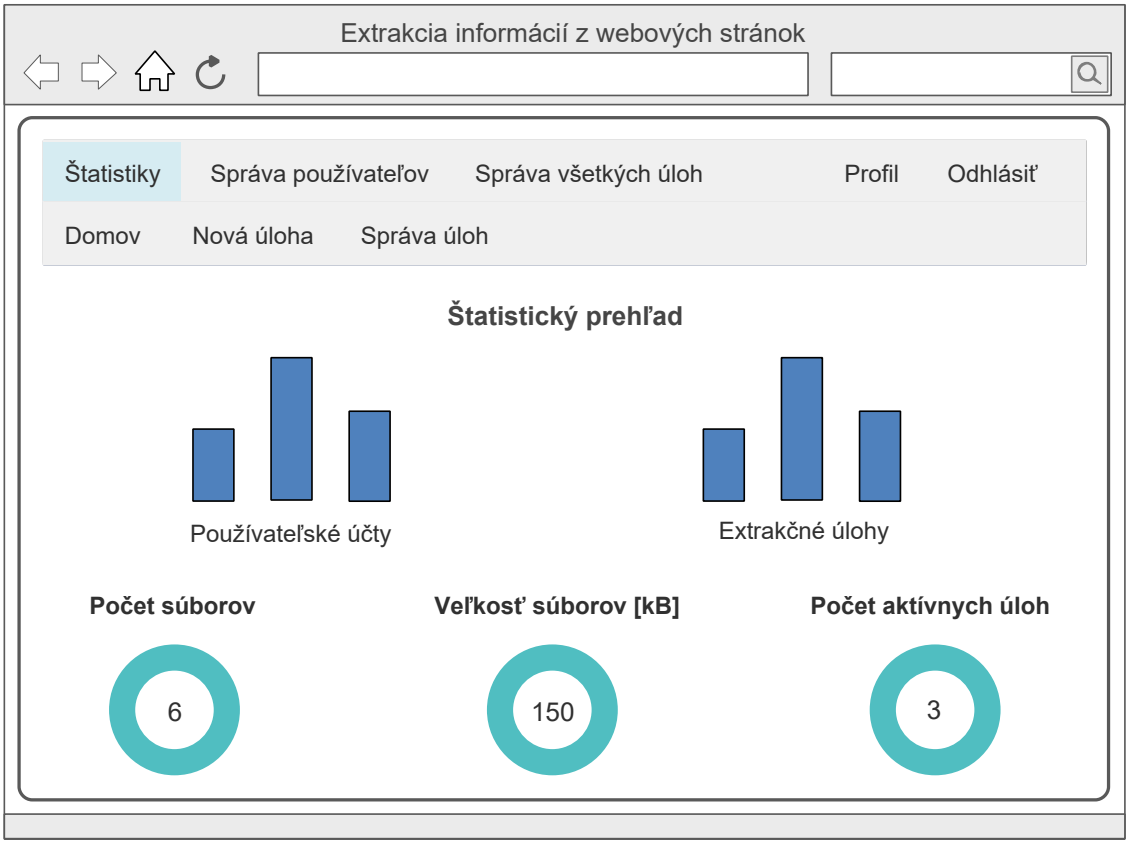
Obr. 4.10: Návrh GUI pre správu úloh

V ľavom stĺpci tabuľky na predchádzajúcom obrázku sa nachádza tlačidlo s ikonou plusu v kruhu (⊕). Po kliknutí na toto tlačidlo sa rozbalia podrobnejšie informácie o extrakčnej úlohe. Ak úloha nie je spustená je možná editácia jej konfigurácie. Taktiež sa po rozbalení zobrazia záznamy spustení tejto úlohy, viď obrázok 4.11. V záznamoch spustení je vidieť práve prebiehajúca úloha v prvom riadku. Údaje o sťahovaní a extrakcii sa tam budú aktualizovať. V druhom riadku je zobrazený záznam spustenia už ukončenej úlohy, kde sa dajú zobraziť extrahované dáta, prípadne stiahnuť súbor. Taktiež je možné vymazať neaktívny záznam. Všetky tieto podrobnejšie informácie sa dajú skryť opäť po kliknutí na tlačidlo v ľavom stĺpci, teraz s ikonou (⊖).



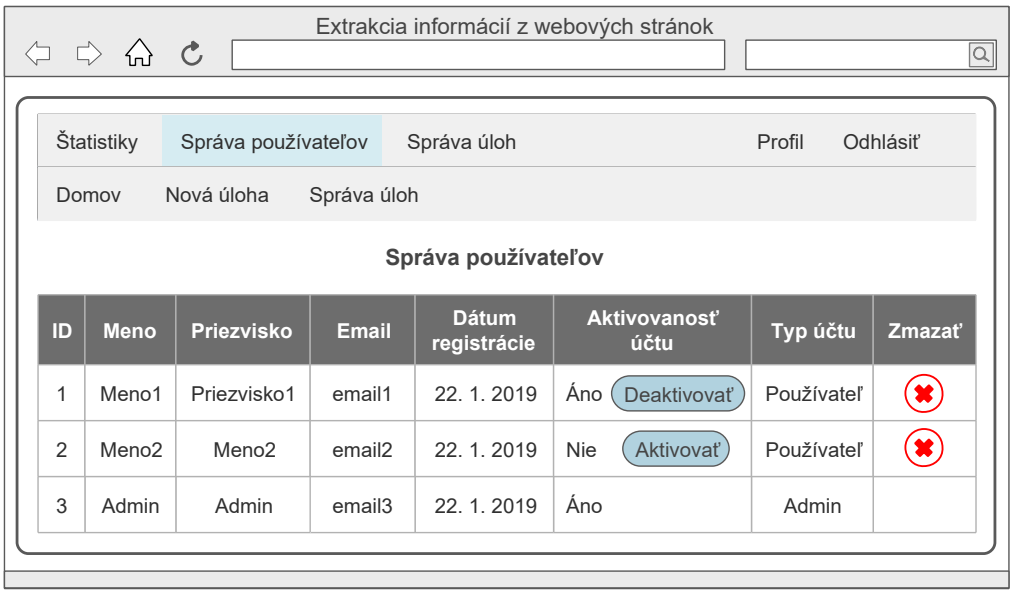
Obr. 4.11: Návrh GUI pre správu úloh – rozbalená úloha

Na nasledujúcom obrázku 4.12 je návrh používateľského rozhrania pre administrátora, ktorý sa nachádza v sekcii štatistiky. Na tejto stránke sa budú nachádzať štatistiky, ktoré boli popísane v časti funkčných požiadaviek na aplikáciu. Štatistiky pre používateľa budú vyzerať podobne.



Obr. 4.12: Návrh GUI pre administrátorské štatistiky

Na poslednom obrázku z návrhu GUI 4.13 je stránka pre administrátora, ktorý sa nachádza v sekcii správa používateľov. Tu je možné aktivovať účet, deaktivovať ho, alebo zmazať.



Obr. 4.13: Návrh GUI pre správu používateľov

Rozličné upozornenia systému pre používateľa budú zobrazované na viditeľných miestach. Na základe typu upozornenia sa bude odlišovať aj jeho zobrazenie a vzhľad. Napr. informačné hlásenia sa objavia po dobu niekoľkých sekúnd a potom sa skryjú, pričom ich vzhľadový štýl bude riešený zelenou farbou. Táto situácia nastane pri úspešnej registrácii používateľa, pri vytvorení novej úlohy a pod. Prípadné chybové hlásenia zostanú zobrazené pokiaľ používateľ chybu neodstráni a na ich vzhľadový štýl bude použitá červená farba. To môže nastať pri vyplňovaní formulárov s chybnými údajmi a pod.

Pri vyplňovaní formulárov budú používateľovi poskytnuté vhodné grafické prvky na uľahčenie a urýchlenie ich vyplnenia. Medzi tieto prvky patrí výber času pre nastavenie limitu maximálnej dĺžky úlohy pri jej vytváraní, prípadne editovaní. Tento výber bude umožnený pomocou vhodných posuvných nastavovačov hodnôt, pričom sa tak budú dať nastaviť zvlášť hodiny, minúty a sekundy. Ďalšími grafickými prvkami budú rôzne tlačidlá zľahčujúce vypĺňanie hodnôt. Niektoré polia formulárov budú prednastavené na východzie hodnoty, ktoré bude mať používateľ možnosť meniť.

Pre sprehľadnenie a zjednotenie vzhľadu sa budú využívať rôzne druhy ikon, ktoré sú zaužívané a používané v rôznych aplikáciách. Tieto ikony upresnia význam akcie, ktorú používateľ môže vykonať s daným tlačidlom, prípadne majú iba informatívny charakter.

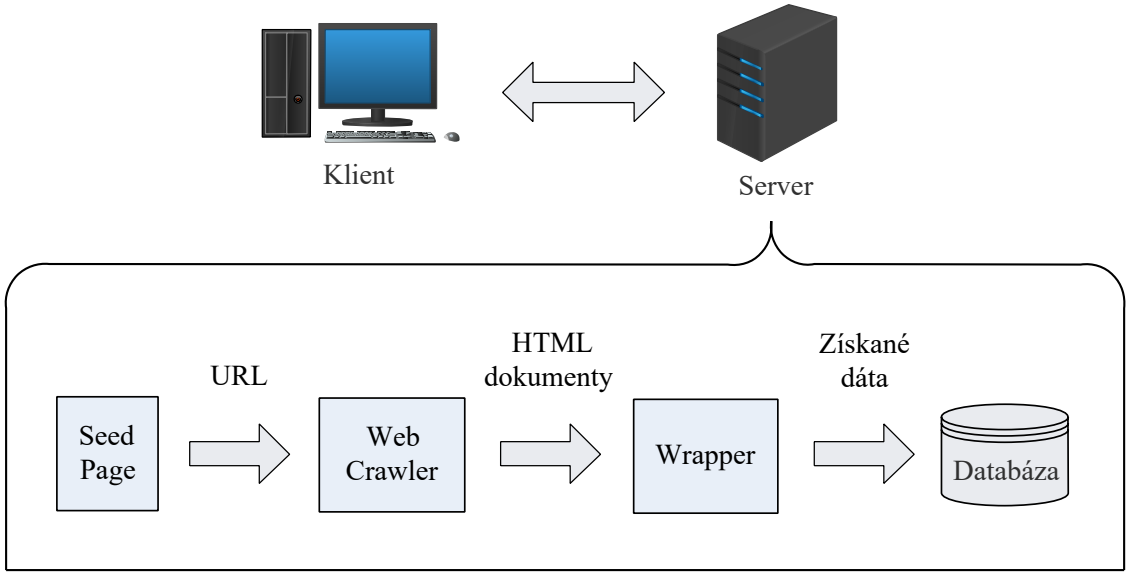
Kapitola 5

Implementácia aplikácie

Táto kapitola popisuje implementáciu aplikácie pre definíciu a správu extrakčných úloh. Nachádza sa tu popis konfigurácie a implementácie databáz pre ukladanie dát. Ďalej sú tu popísané implementačné podrobnosti, kde sa nachádza postup implementácie a prepojenie frontendu s backendom. Taktiež tu je popísaná implementácia crawlera pre sťahovanie webových stránok a implementácia HLRT wrappera, pre následnú extrakciu dát z nich. Nachádzajú sa tu snímky obrazovky výslednej aplikácie, ktoré sú aj popísané.

5.1 Bloková schéma aplikácie

Celá aplikácia pre definíciu a správu extrakčných úloh je založená na klient-server architektúre. Na nasledujúcom obrázku 5.1 je znázornená bloková schéma aplikácie znázorňujúca spracovanie novej extrakčnej úlohy.



Obr. 5.1: Bloková schéma aplikácie pre správu extrakčných úloh

Popis jednotlivých častí blokovej schémy:

- *Klient* – jedná sa o používateľa aplikácie, ktorým je bežný používateľ, alebo administrátor. So systémom pracuje pomocou webového prehliadača, na ktorom sa zobrazuje používateľské rozhranie.
- *Server* – serverová časť aplikácie poskytuje celú logiku systému a komunikáciu s klientom. Server je navrhnutý na viacvrstvovej Java EE architektúre a poskytuje celý systém pre správu extrakčných úloh.
- *Seed Page* – vstupná URL adresa pre konkrétnu extrakčnú úlohu.
- *Web Crawler* – táto časť systému sa stará o sťahovanie webových stránok. Na vstup dostane *Seed Page* URL a jej výstupom sú stiahnuté HTML dokumenty.
- *Wrapper* – táto časť systému tvorí jadro pre extrakciu štruktúrovaných informácií z HTML dokumentov. Na svoj vstup dostane HTML dokumenty, z ktorých extrahuje požadované dáta a jej výstupom je JSON súbor, ktorý sa uloží do databázy.
- *Databáza* – ukladajú sa do nej extrahované dáta. Taktiež slúži na ukladanie potrebných informácií pre chod celej aplikácie.

5.2 Úložisko dát

Pre ukladanie potrebných dát v rámci celej aplikácie budú vytvorené dve databázy. Jednou z nich bude relačná MySQL databáza pre ukladanie všetkých dát týkajúcich sa používateľov, ich rolí a extrakčných úloh. Druhou z nich bude nerelačná MongoDB databáza, v ktorej budú uložené všetky dokumenty s extrahovanými dátami. V nasledujúcej časti budú obe bližšie popísané.

Ukladanie informácií

Na ukladanie informácií o používateľoch, ich rolách a extračných úlohách je použítá relačná MySQL databáza. Schéma tejto databázy je znázornená ER diagramom a popísaná v kapitole 4.4. Pre každú entitu tohto ER diagramu je vytvorená odpovedajúca trieda v Jave. Týmito triedami je implementované objektovo-relačné mapovanie (ORM) pre aplikačný rámec Hibernate. Jednotlivé atribúty a metódy týchto tried sú znázornené v časti 4.5, konkrétne sa jedná o diagram tried pre balík *Model*.

Každá trieda vytvorená pre objektovo-relačné mapovanie má svoju odpovedajúcu triedu pre prístup k objektu databázy (Data Access Object – DAO). Tieto triedy v sebe implementujú metódy potrebné na vykonávanie operácií nad databázou. Jednotlivé metódy sú znázornené ako operácie v diagrame tried pre balík *DAO*, viď časť 4.5.

Konfigurácia aplikačného rámca Hibernate sa nachádza v triede *HibernateConfig* (balík *Config*). Táto trieda implementuje metódu *getSessionFactory()*, ktorá cez návratovú hodnotu vráti inštanciu nakonfigurovaného objektu z triedy *SessionFactory*. Práve v tomto objekte sa nastavujú všetky potrebné parametre pre prístup do MySQL databázy, ako sú názov databázy, číslo portu a pod. Taktiež sa do tejto konfigurácie pridávajú všetky triedy zabezpečujúce objektovo-relačné mapovanie modelu databázy. V triedach z balíku *DAO* sa potom volá metóda *HibernateConfig.getSessionFactory().openSession()*, čo následne umožní prístup do databázy a umožní vykonávať potrebné operácie nad ňou.

Pre chod aplikácie je potrebné vytvorenie MySQL databázy s názvom *iefwp_mysqldb* a spustenie inicializačného skriptu *iefwp_mysqldb.sql* nad touto databázou. Tento skript vykoná vytvorenie všetkých relácií (tabuľiek) a následné naplnenie vzorovými dátami. Podrobnejšie informácie o tomto sú uvedené v priloženej inštalačnej dokumentácii na prenosnom médiu.

Administrátor, ktorý je v celej aplikácií iba jeden musí mať nastavené obe dostupné role. Teda rolu bežného používateľa *ROLE_USER* a aj rolu administrátora *ROLE_ADMIN*.

Ukladanie extrahovaných dát

Na ukladanie dokumentov s extrahovanými dátami je použitá nerelačná MongoDB databáza. Jedná sa o databázu určenú na ukladanie JSON dokumentov. MongoDB databáza sa skladá z kolekcií (collections), ktoré sú ekvivalentom tabuliek v relačnej databáze. Každá kolekcia môže obsahovať v sebe viacero dokumentov. Dokument je v tejto databázy základným objektom a nemá preddefinovanú žiadnu schému. Dokument uložený v kolekcii je reprezentovaný ako pole vo formáte JSON, teda celý dokument sa ukladá v JSON formáte. Každý dokument má svoj unikátny identifikátor (ID) v rámci celej kolekcie [10].

Konfigurácia MongoDB databázy sa nachádza v triede *MongoDbConfig* (balík *Config*). Nastavuje sa tam názov databázy a jej umiestnenie. Metóda *mongoTemplate()*, ktorú táto trieda implementuje, vracia objekt triedy *MongoTemplate*. Tento objekt umožňuje prístup do databázy a následné vykonávanie operácií nad ňou. Využíva sa v triede *FileDao*, ktorá implementuje konkrétne metódy z diagramu tried *FileDao*, viď časť 4.5.

Extrahované dokumenty sa ukladajú do databázy s názvom *iefwp_mongodb*, ktorá obsahuje kolekciu *files*. Každý dokument tu je uložený vo formáte JSON a má svoj unikátny identifikátor (ID). Všetky dokumenty uložené v tejto kolekcii sú dostupné cez unikátne URL adresy (každý dokument má svoju unikátnu adresu). Prihlásený používateľ ma prístup iba ku svojim dokumentom s extrahovanými dátami a im odpovedajúcim URL adresám. Výnimkou je administrátor, ktorý má prístup ku všetkým dokumentom.

Prístup a zobrazenie dokumentu z databázy poskytuje trieda *GetFileServletBean*, ktorá sa nachádza v balíku *Controller*. Táto trieda implementuje servlet, ktorý sprístupňuje dokumenty cez metódu *doGet(. . .)*. Ukážka prístupu k dokumentu s extrahovanými dátami cez servlet:

http://localhost:8080/getFileServlet?id=5ca4b58f7e0beb2b0438d79d

Na predchádzajúcej ukážke pre prístup k dokumentu je zobrazená unikátna URL adresa, ktorej časti majú nasledovný význam:

- *http://localhost:8080/* – adresa servera, na ktorom je umiestnená aplikácia,
- *getFileServlet* – názov servletu, ktorý obsluhuje HTTP požiadavky,
- *id* – parameter *id* pre HTTP požiadavku typu *get*,
- *5ca4b58f7e0beb2b0438d79d* – unikátne ID dokumentu uloženého v kolekcii *files*.

Po zadaní uvedenej URL adresy vo webovom prehliadači servlet *getFileServlet* spracuje HTTP požiadavku *get* a ako odpoveď (response) vráti zobrazený JSON dokument z MongoDB databázy s príslušným ID. Keď sa dokument s uvedeným ID v kolekcii *files* nenájde, prípadne nastane nejaká chyba, tak je o tom používateľ informovaný.

V kolekcii *files* sa nenachádzajú žiadne prázdne dokumenty. To znamená, že keď extrakčná úloha zo všetkých stiahnutých HTML dokumentov nezíska pomocou HLRT wrappera žiadne dáta, tak sa do kolekcie *files* nevloží prázdny dokument.

5.3 Implementačné podrobnosti

Celá aplikácia je implementovaná na platforme Java EE, pričom bol použitý aplikačný rámec Spring Framework. Bol použitý viacvrstvový návrhový vzor MVC, ktorý je popísaný v časti 2.6.

Postup implementácie:

- Pomocou webových stránok nástroja *Spring Initializr* sa vygenerovala prvotná kostra aplikácie. Na stránkach sa nastavil názov projektu, artefaktu a balíka, ďalej programovací jazyk Java verzia 12 a verzia nástroja *Spring Boot*. Zvolený nástroj pre správu, riadenie a automatizáciu nasadzovania aplikácie bol Maven. Ďalej boli pridané závislosti ako sú: *spring-boot-starter-security*, *primefaces-spring-boot-starter*, *spring-boot-starter-data-mongodb* a pod. Po nastavení všetkých spomenutých parametrov sa stiahla vygenerovaná kostra pre aplikáciu a importovala sa pomocou IDE softvéru.

- Z ER diagramu popísaného v časti 4.4 sa vytvoril inicializačný skript v jazyku MySQL, ktorý vytvorí celú schému databázy podľa tohto diagramu. Tento skript, okrem vytvorenia relácií (tabuliek), nastaví aj integračné obmedzenia, väzby medzi reláciami a ich atribúty s dátovými typmi. Po vytvorení schémy naplní relácie testovacími dátami. Na MySQL serveri sa vytvorila nová databáza s názvom *iefwp_mysqldb*, na ktorej sa spustil spomínaný skript.

  Následne sa vytvorila nová databáza na MongoDB servery s názvom *iefwp_mongodb*. V tejto databáze sa vytvorila nová kolekcia s názvom *files*.

- Vytvorenie štruktúry balíkov zdrojových súborov a jednotlivých tried, ktoré do nich patria podľa návrhu diagramov tried z časti 4.5.

- Implementácia konfiguračných tried pre celú aplikáciu. Ďalej implementácia tried pre objektovo-relačné mapovanie aplikačného rámca Hibernate a im odpovedajúcim DAO triedam. Otestovanie funkčnosti prepojenia Hibernate s MySQL databázou. Keďže bol pri implementácii použitý aplikačný rámec Spring Framework, tak v zdrojových súboroch v balíku *resources* sa nachádza konfiguračný súbor *application.properties*. Nastavujú sa tam konfigurácie databáz, téma aplikačného rámca PrimeFaces a pod.

- Konfiguráciu zabezpečenia aplikácie pomocou *Spring security* implementuje trieda *SecurityConfig*. Poskytuje autentizáciu a autorizáciu používateľov podľa príslušných rolí. Pre každú rolu sú nastavené zdrojové súbory, kde má daný používateľ prístup. K ostatným mu bude prístup odmietnutý. Ukážka nastavenia prístupu, ktorý implementuje metóda *configure(HttpSecurity http)*:

  1. *http.authorizeRequests().antMatchers("/index.xhtml").permitAll()*
  2. *http.authorizeRequests().antMatchers("/user/**").hasRole("USER")*

  V ukážke prvý bod znázorňuje nastavenie autorizácie pre prístup k súboru *index.xhtml*, ktorý je dostupný pre všetkých používateľov. Druhý bod znázorňuje, že do balíka *user*

bude mať prístup iba používateľ, ktorý bude prihlásený a jeho účet bude mať nastavenú rolu *ROLE_USER*. Takýmto spôsobom sú definované všetky potrebné pravidlá prístupu.

- Implementácia jednotlivých iterácií podľa návrhu 4.3 a im odpovedajúcim diagramom tried, vrátane ich operácií z časti 4.5. Počas implementácie každej iterácie prebiehalo testovanie, ktorému bude venovaná samostatná časť neskôr.

Prepojenie frontendu a backendu

Frontend aplikácie je implementovaný pomocou technológie JavaServer Faces (JSF) a je na to použitý aplikačný rámec PrimeFaces. Táto nadstavba poskytuje funkcionalitu pre viac ako 100 grafických komponentov, ktoré je možné využiť. Nachádzajú sa tam rôzne typy formulárov, tlačidiel, modálne okná a pod. Taktiež poskytuje technológiu AJAX, ktorá je využívaná na funkcionalitu pri tvorbe jedno-stránkovej webovej aplikácie (SPA). Na oficiálnych webových stránkach sú dostupné ukážky aj so zdrojovými kódmi pre frontend programovaný v XHTML a backend programovaný v Jave[1]. Aplikačný rámec PrimeFaces je do aplikácie pridaný pomocou nástroja Maven.

Každá časť aplikácie ako je registrácia používateľa, prihlasovanie, vytvorenie novej úlohy, správa úloh, editácia profilu atď., má v zdrojových súboroch vytvorený XHTML dokument, ktorý implementuje príslušnú webovú stránku. Pre každý XHTML dokument existuje trieda v Jave, ktorá pre tento dokument implementuje celú logiku, chod aplikácie, overovanie formulárov a pod. Overovanie formulárov prebieha vždy na serverovej časti, čo zvyšuje zabezpečenie aplikácie.

V nasledujúcej tabuľke 5.1 je znázornená ukážka niektorých XHTML dokumentov a im odpovedajúcim triedam v Jave. V každom riadku tabuľky je uvedené, akú majú dané súbory funkciu.

[1]Ukážky aplikačného rámca PrimeFaces sú dostupné na adrese: https://www.primefaces.org/showcase/, navštívené: 25. 4. 2019.

XHTML dokument	Trieda v Jave	Popis
createAccount.xhtml	CreateAccountBean	Vytvorenie nového používateľského účtu počas registrácie.
homeUser.xhtml	HomeUserBean	Úvodná obrazovka po prihlásení používateľa. Nachádza sa tam štatistický prehľad.
editProfile.xhtml	EditProfileBean	Editovanie kontaktných informácií, zmena emailu a hesla.
createTask.xhtml	CreateTaskBean	Vytvorenie novej extrakčnej úlohy.
taskManager.xhtml	TaskManagerBean	Správa používateľových úloh. Zobrazenie úloh, editovanie úlohy a mazanie úlohy. Zobrazenie záznamov spustení a zmazanie záznamu spustenia. Zobrazenie a zmazanie súboru s extrahovanými dátami.
homeAdmin.xhtml	HomeAdminBean	Štatistiky pre administrátora.
userManager.xhtml	UserManagerBean	Správa používateľských účtov. Deaktivácia a mazanie účtu.

Tabuľka 5.1: Niektoré HTML dokumenty a im odpovedajúce triedy v Jave

Všetky ikony použité v aplikácií sú obsiahnuté vo fonte s názvom *FontAwesome*[2]. Jedná sa o voľne šíriteľný font s otvoreným zdrojovým kódom (open souce) vydaný pod licenciou GPL. Je dostupný v dvoch verziách, prvá je voľná s viac ako 1500 ikonami, druhá je platená s vyše 5000 ikonami. V tejto aplikácii bola použitá voľná verzia.

Ako už bolo spomínané, na správu a pridávanie závislostí na externé knižnice a aplikačné rámce do aplikácie bol použitý nástroj pre správu správu, riadenie a automatizáciu nasadzovania Maven. Jeho konfigurácia sa nachádza v súbore *pom.xml* (Project Object Model – POM). Tento konfiguračný XML súbor obsahuje informácie o projekte, ako je jeho názov, verzia použitej Javy a pod. Hlavnú časť tvorí pridávanie závislostí na externé knižnice, ktoré tu sú všetky uvedené. Pri prekladaní aplikácie tak nie je potrebné sťahovať tieto knižnice zvlášť, ale o všetko sa postará Maven automaticky.

V rámci backendu sa používa logovanie pomocou nástroja *SLF4J*[3]. Tento nástroj umožňuje nastavovanie viacerých úrovní logovania, ktoré sú v aplikácii využívané. Napr. pri úspešnom vytvorení novej extrakčnej úlohy a jej vložení do databázy sa zapíše do logovacieho súboru o tom informácia s nastavenou úrovňou *info* – teda má informačný charakter. Po výskyte nejakej chyby bude nastavená úroveň na *error*, čo znamená chybu.

5.4 Crawler a HLRT wrapper

Táto časť popisuje jadro celej aplikácie, ktorým je sťahovanie HTML dokumentov z webových stránok a následná extrakcia dát z nich.

[2]FontAwesome je dostupný online na oficiálnych stránkach projektu: https://fontawesome.com/, navštívené: 2. 5. 2019.

[3]Simple Logging Facade for Java (SLF4J) – nástroj pre logovanie v programovacom jazyku Java, viď https://www.slf4j.org/

V aplikácii je implementovaný správca všetkých aktívnych úloh. Aktívnou úlohou sa myslí práve prebiehajúca úloha. Tento správca je inštanciou triedy *ActiveTasksManager*, ktorá sa nachádza v balíku *activeTasksManager*. Po nasadení a spustení aplikácie na serveri sa vytvorí tento objekt, ktorý má nastavený rozsah platnosti v rámci celej aplikácie. V aplikačnom rámci Spring je na to použitá anotácia *@ApplicationScope*. Tento správca má ako svoj atribút list všetkých aktívnych úloh a ich ovládačov na crawler (každý crawler má ovládač na spustenie a zastavenie úlohy). Po spustení extrakčnej úlohy sa vloží do tohoto listu nový záznam spustenia (nová inštancia triedy *TaskDynamic*) a tiež ovládač na crawler (viď ďalšia časť). Po zastavení úlohy sa tento záznam z listu odstráni. Vďaka správcovi aktívnych úloh *activeTasksManager* sú stále dostupné všetky informácie o prebiehajúcich úlohách. Dostupné sú aj keď sa používateľ odhlási zo systému a znova sa prihlási. V nasledujúcich bodoch je zhrnutý životný cyklus extrakčnej úlohy:

1. Používateľ definuje novú extrakčnú úlohu (vytvorí sa inštancia triedy *TaskStatic*).
2. Úloha sa vloží do MySQL databázy – do relácie *TasksStatic*.
3. Používateľ spustí extrakčnú úlohu.
4. Vytvorí sa nový záznam spustenia (inštancia triedy *TaskDynamic*), nastaví sa na aktívny a vloží sa do MySQL databázy – do relácie *TasksDynamic*.
5. Úlohu zastaví používateľ, alebo sa ukončí sama po prekročení nejakého obmedzenia.
6. Keď úloha skončí, tak sa záznam spustenia nastaví na neaktívny a z aktuálnych štatistík úlohy sa aktualizujú dáta v relácií *TasksDynamic*. Ak boli extrahované nejaké dáta, tak sa s nimi vytvorí nový JSON dokument a vloží sa do MongoDB databázy.
7. V správcovi aktívnych úloh sa z listu odoberie príslušný záznam spustenia a tiež ovládač na crawler. Týmto je úloha úspešne ukončená.

V nasledujúcich dvoch častiach budú bližšie popísané implementačné podrobnosti sťahovania webových stránok a extrakcie informácií z nich.

Sťahovanie webových stránok

Na sťahovanie webových stránok sa používa knižnica Crawler4j, ktorá bola popísaná v časti 3.1. Knižnica je do projektu pridaná nástrojom Maven. Celý crawler je implementovaný v balíku *webCrawler*. Tento balík obsahuje nasledujúce tri triedy, ktoré v sebe implementujú všetko potrebné pre sťahovanie webových stránok:

- *CrawlingStats* – táto trieda implementuje dátovú štruktúru pre ukladanie informácií o aktuálnom priebehu sťahovania. Jej atribúty sú počet stiahnutých súborov, počet stiahnutých URL odkazov, dĺžka extrakcie, veľkosť stiahnutých súborov a pod.
- *CrawlerController* – každá úloha sťahovania má vytvorenú inštanciu z tejto triedy. Implementuje v sebe dve metódy:
  - *startCrawling(. . .)* metóda nakonfiguruje Crawler4j podľa parametrov zadávaných pri vytváraní úlohy. Nastavuje sa tu oneskorenie HTTP požiadaviek a všetky ostatné limity. Po tejto konfigurácii sa sťahovanie spustí. Táto operácia je neblokujúca, takže používateľ môže naďalej pracovať s aplikáciou a spúšťať aj ďalšie svoje úlohy.

  - *stopCrawling()* metóda zastaví proces sťahovania a extrakcie. Čo sa má po zastavení vykonať implementuje v sebe trieda *Crawler*.

- *Crawler* – táto trieda v sebe implementuje funkcionalitu sťahovania webových stránok. Začne sa stiahnutím vstupnej webovej stránky (seed page). Následne sa z nej extrahujú všetky dostupné URL adresy. Tieto adresy sa filtrujú podľa jednotlivých obmedzení nastavených pri vytváraní úlohy. Tie, ktoré vyhovujú požiadavkám sa zaradia do fronty, ostatné sa zahodia. Zo stiahnutej stránky sa získa jej veľkosť v *bytoch* a podobné informácie. Pokiaľ nie sú prekročené obmedzenia sťahovania, tieto informácie sa uložia do štatistiky sťahovania a nasleduje časť extrakcie. Táto časť je popísaná nižšie. Po extrakcii sa pokračuje stiahnutím a spracovávaním ďalšej webovej stránky uloženej vo fronte.

  Medzi hlavné metódy, ktoré implementuje táto trieda patria:

  - *shouldVisit(. . .)* metóda filtruje URL adresy podľa zadaných kritérií.
  - *visit(. . .)* metóda, ktorá získava zo stiahnutej stránky základné informácie. Kontroluje, či neboli prekročené limity. Volá metódu extrakcie pomocou HLRT wrappera. Aktualizuje štatistiky úlohy na nové hodnoty.
  - *onBeforeExit()* metóda implementuje funkcionalitu po zastavení sťahovania ovládačom, alebo po prekročení nejakého limitu. Pred ukončením sa vložia do MySQL databázy výsledné štatistiky záznamu spustenia. Ak boli extrahované nejaké dáta, tak sa do MongoDB databázy vloží dokument s nimi. Na záver sa zo správcu všetkých aktívnych úloh *activeTasksManager* vymaže táto spustená úloha a tiež ovládač na jej crawler.

Extrakcia informácií z HTML dokumentov

Na získavanie informácií z HTML dokumentov je použitý HLRT wrapper, ktorý bol popísaný v časti 3.2. Implementácia wrappera sa nachádza v balíku *wrapperHLRT*. Popis tried nachádzajúcich sa v tomto balíku:

- *WrapperTupleLR* – trieda reprezentuje dátovú štruktúru pre jednu dvojicu oddeľovačov *left* a *right* HLRT wrappera. Každá dvojica má navyše pomenovanie pre výsledné dáta, ktoré extrahuje. Napr. keď dvojica extrahuje cenu produktu, tak táto dvojica môže mať pomenovanie *price*. Toto pomenovanie sa bude ukladať aj do výsledného súboru s extrahovanými dátami. Nastavuje sa pri vytváraní novej extrakčnej úlohy v časti konfigurácie wrappera. Toto pomenovanie je voliteľné a nemusí byť zadané.

- *WrapperConfigHLRT* – trieda reprezentuje dátovú štruktúru pre celú konfiguráciu wrappera, ktorá vznikne pri vytvorení novej extrakčnej úlohy. Obsahuje atribúty pre oddeľovače *head* a *tail*. Taktiež má ako atribút list inštancií triedy *WrapperTupleLR*. Celá táto trieda sa mapuje pomocou JAXB[4] do XML reťazca a ten sa následne vkladá do databázy ako konfigurácia HLRT wrappera do relácie *TasksStatic* a jej atribútu *wrapperConfig*.

- *WrapperExtractor* – táto trieda v sebe implementuje samotný algoritmus extrakcie pomocou HLRT wrappera. Implementuje v sebe nasledovnú metódu pre extrakciu:

[4] Java Architecture for XML Binding (JAXB) – technológia, ktorá umožňuje konverziu Java objektov do XML a naopak.

extractDataFromHtml(String html, WrapperConfigHLRT wrapperConfigHLRT),

ktorá má dva parametre. Prvý parameter je reťazec HTML dokumentu na extrakciu a druhý parameter je konfigurácia HLRT wrappera.

Algoritmus extrakcie pomocou HLRT wrappera, ktorý je prevzatý zo zdroja [19], je implementovaný v uvedenej metóde. Na obrázku 5.2 je zobrazený pseudokód wrappera, pričom význam jednotlivých premenných je popísaný v časti 3.2:

1. procedure $\text{exec}_{\text{HLRT}}$(wrapper $\langle h, t, \ell_1, r_1, \ldots, \ell_K, r_K \rangle$, page P)
2. $m \leftarrow 0$
3. scan in P to the first occurrence of h
4. while the next occurrence of ℓ_1 in P occurs before the next occurrence of t
5. $m \leftarrow m + 1$
6. for each $\langle \ell_k, r_k \rangle \in \{\langle \ell_1, r_1 \rangle, \ldots, \langle \ell_K, r_K \rangle\}$
7. scan in P to the next occurrence of ℓ_k; save position as $b_{m,k}$
8. scan in P to the next occurrence of r_k; save position as $e_{m,k}$
9. return label $\{\ldots, \langle \langle b_{m,1}, e_{m,1} \rangle, \ldots, \langle b_{m,K}, e_{m,K} \rangle \rangle, \ldots\}$

Obr. 5.2: Algoritmus extrakcie pomocou HLRT wrappera[5]

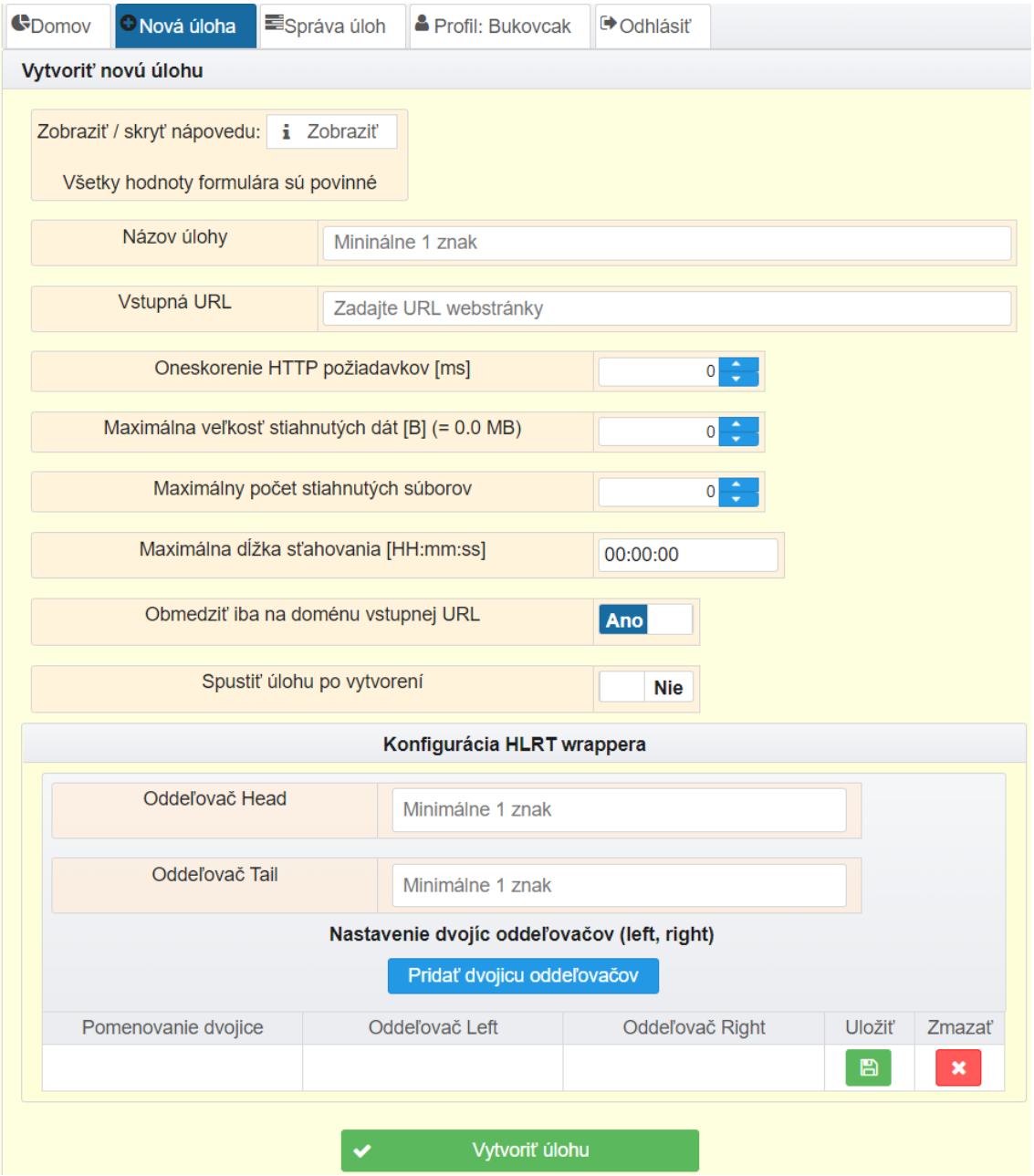
V metóde *extractDataFromHtml(...)* je algoritmus trochu pozmenený, avšak hlavné časti sú zachované. Zmenené je ukladanie extrahovaných dát, kde v originálnom algoritme z obrázka 5.2 sa ukladajú pozície (začiatočný a koncový index) v reťazci, medzi ktorými sa má získať časť reťazca. Tie sa na konci extrakcie vracajú ako dvojice týchto pozícií. Zmenené to je, na získavanie časti reťazca ktoré dvojica oddeľovačov extrahuje ihneď (neukladajú sa indexy). Po získaní časti reťazca sa vloží do výsledného JSON poľa, ktoré potom metóda vráti.

5.5 Výsledná aplikácia

Táto časť znázorňuje a popisuje niektoré časti výslednej webovej aplikácie pre definíciu a správu extrakčných úloh. Nachádzajú sa tu snímky obrazovky z aplikácie a ich popis. V niektorých častiach aplikácie je dostupná nápoveda, ktorú si môže používateľ zobraziť.

Na nasledujúcom obrázku 5.3 je snímka obrazovky pri vytváraní novej extrakčnej úlohy. Prihlásený je bežný používateľ. Z hlavného menu položka *Domov* reprezentuje štatistiky pre prihláseného používateľa o jeho úlohách a pod. Položka *Nová úloha* je práve aktívna. Ďalej je položka *Správa úloh* a *Profil*, ktoré sú popísané nižšie. Posledná možnosť *Odhlásiť* je pre odhlásenie používateľa z aplikácie. Ako je vidieť na obrázku, pri vytváraní úlohy je možné zadávať obmedzenia, ktoré boli popísané už v predchádzajúcich častiach. Pri zadaní vstupnej URL adresy (seed page) sa overí, či zadaná webová stránka je dostupná online a v prípade nedostupnosti je používateľ o tom informovaný. Podstatná časť je *Konfigurácia HLRT wrappera*, ktorá špecifikuje aké dáta sa majú získavať zo stiahnutých HTML dokumentov. Jednotlivé dvojice oddeľovačov *left* a *right* sa môžu pridávať, upravovať a mazať. Zadaná musí byť minimálne jedna dvojica.

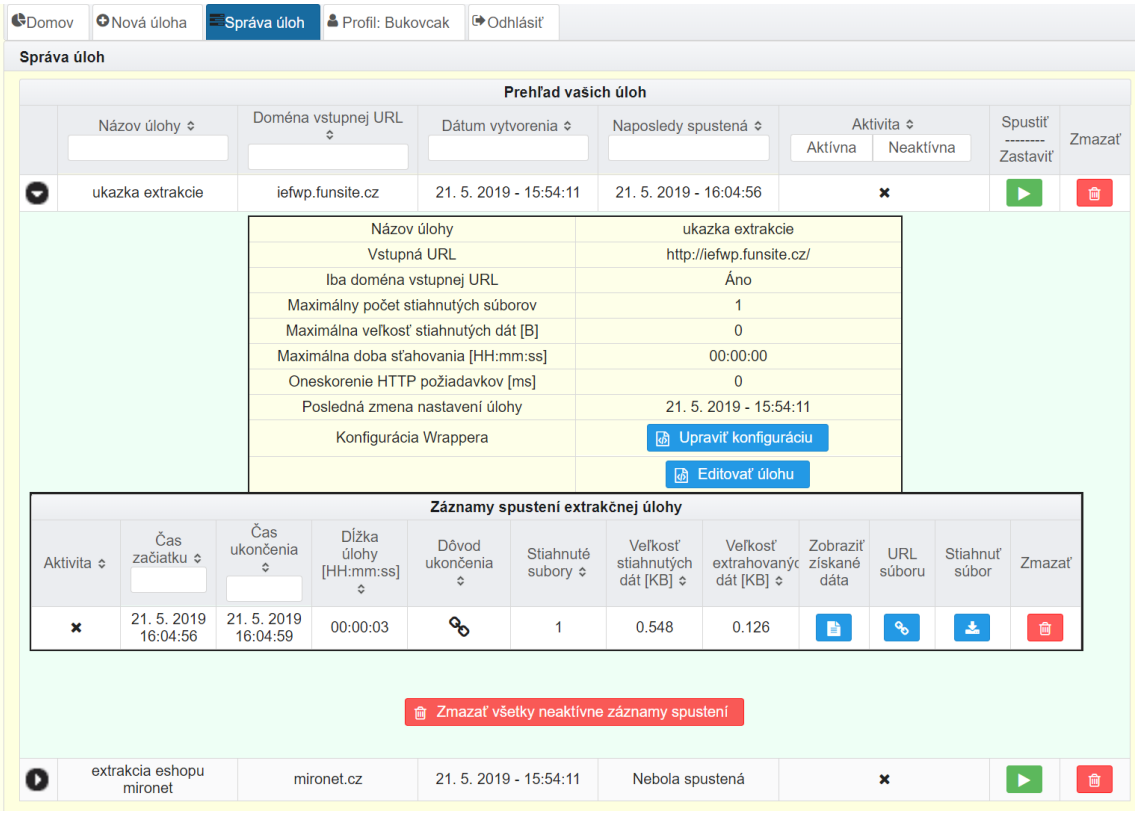
[5] Algoritmus prevzatý zo zdroja [19].



Obr. 5.3: Snímka obrazovky pri vytváraní novej úlohy

Na nasledujúcom obrázku 5.4 je snímka obrazovky zo správy extrakčných úloh, pričom je tiež prihlásený bežný používateľ. Práve je zobrazená a rozbalená konkrétna extrakčná úloha. Táto úloha je zastavená, takže je možné ju spustiť, editovať, alebo vymazať. Pri vymazaní úlohy sa taktiež odstránia z databázy jej záznamy spustení a súbory s extrahovanými dátami. Na obrázku je možné vidieť bližšie informácie o úlohe a taktiež o jej záznamoch spustení. V záznamoch spustení sú zobrazené informácie o priebehu úlohy a dôvode ukončenia. Dôvod ukončenia je znázornený príslušnou ikonou a popis sa zobrazí, keď je nad ňou kurzor myši. Dajú sa zobraziť získané dáta, získať URL adresu súboru s extrahovanými dátami a stiahnuť tento súbor vo formáte JSON. Taktiež je možné vymazať záznam

spustenia. Keď sa vymaže nejaký záznam spustenia, odstránia sa z MySQL databázy všetky informácie o ňom a taktiež sa z MongoDB databázy odstráni súbor s extrahovanými dátami. Následne sa zmaže aj príslušný riadok tabuľky v záznamoch o spustení zobrazený na frontende.

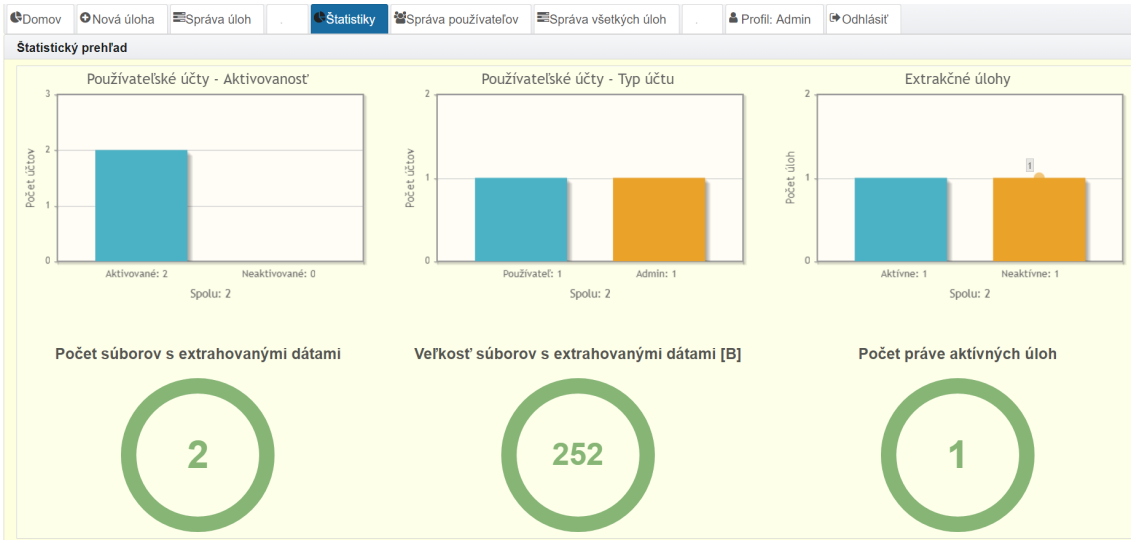


Obr. 5.4: Snímka obrazovky zo správy extrakčných úloh

Na nasledujúcom obrázku 5.5 je snímka obrazovky z editovania profilu, kde je prihlásený bežný používateľ. Používateľovi je umožnené zmeniť priezvisko, email a heslo. Zobrazuje sa tu aj dátum registrácie.

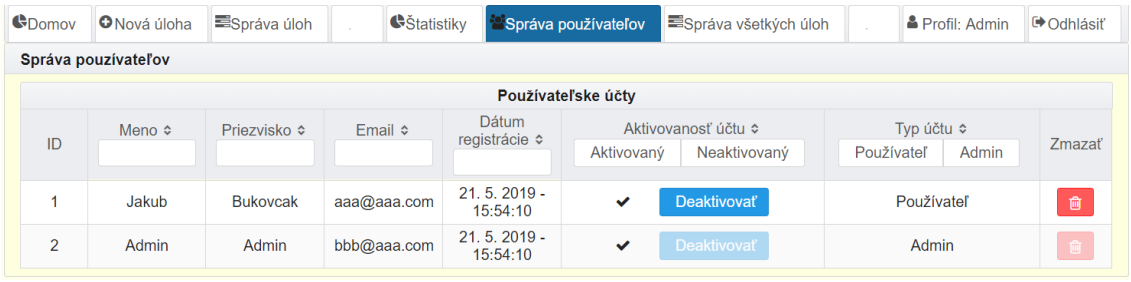
Obr. 5.5: Snímka obrazovky z editovania profilu používateľa

Na nasledujúcom obrázku 5.6 je snímka obrazovky zo štatistík pre prihláseného administrátora. Nachádzajú sa tu grafy zobrazujúce, koľko účtov je aktívnych, typy používateľských účtov a počet aktívnych/neaktívnych extrakčných úloh. Ďalej sú tu informácie o počte súborov a ich veľkosti.



Obr. 5.6: Snímka obrazovky z administrátorských štatistík

Na nasledujúcom obrázku 5.7 je snímka obrazovky zo správy používateľských účtov pre prihláseného administrátora. Účet bežného používateľa je možné aktivovať, deaktivovať a vymazať. Pri vymazaní používateľa sa odstránia všetky jeho extrakčné úlohy z MySQL databázy a taktiež všetky jeho súbory s extrahovanými dátami z MongoDB databázy. Administrátorský účet sa nedá deaktivovať, ani odstrániť, pretože je iba jeden v celej aplikácii. Po deaktivovaní účtu bežného používateľa sa mu nebude dať prihlásiť do aplikácie.



Obr. 5.7: Snímka obrazovky zo správy používateľských účtov

V tejto časti boli zachytené a popísané iba niektoré webové stránky výslednej aplikácie, pre získanie predstavy o grafickom používateľskom rozhraní aplikácie. Taktiež bola pri každom obrázku naznačená funkcionalita konkrétnej webovej stránky.

Kapitola 6

Testovanie aplikácie

Táto kapitola sa venuje testovaniu aplikácie a skladá sa z dvoch častí. Prvá časť sa zaoberá testovaniu aplikácie na rôzne chyby počas vývoja a implementácie. Druhou časťou je testovanie výslednej aplikácie pri sťahovaní reálnych webových stránok a následnej extrakcie informácií z nich. Nachádza sa tu aj zhodnotenie dosiahnutých výsledkov.

6.1 Testovanie počas vývoja

Testovanie tejto aplikácie bolo vykonávané programátorom a bolo zamerané na viaceré faktory a funkcionalitu. Počas testovania sa vychádzalo aj z nasledujúcich bodov, ktoré sú zo zdroja [17]:

- **Testovanie funkčnosti** – cieľom tohto testovania je zistiť, či sa aplikácia správa podľa špecifikácie a očakávania. Testuje sa funkčnosť všetkých odkazov, ktoré sa nachádzajú na stránkach. Podobne aj všetky tlačidlá na stránkach, či vykonávajú požadovanú funkcionalitu a pod.

  Ďalšou časťou je testovanie všetkých formulárov na stránkach. Kontroluje sa, či majú správne zadané predvolené možnosti. Formuláre sa testujú na zle zadané hodnoty, ktoré nemôžu byť prijaté a pod.

  Testovanie funkčnosti a konzistencie databázy počas vytvárania nových záznamov, mazania záznamov a ich editovania. Popri jednotlivých operáciách treba kontrolovať, či sa pracuje s aktuálnymi a správnymi dátami.

- **Testovanie použiteľnosti** – kontrola obsahu stránok z gramatického, štylistického a významového hľadiska. Skontrolovať, aby mali všetky obrázky, grafy, ikony a text správne veľkosti.

- **Testovanie rozhrania** – testuje sa komunikácia medzi viacvrstvovou architektúrou. Kontroluje sa, či sa správne zachytávajú výnimky a prípadné chyby vzniknuté počas behu aplikácie. Napr. keď nastane chyba počas vkladania nových údajov do databázy, overiť či boli podrobnosti o tom zapísané na príslušné miesta (logovanie, frontend).

- **Testovanie kompatibility** – kompatibilita webovej aplikácie je dôležitou časťou testovania. Overuje sa kompatibilita medzi viacerými webovými prehliadačmi, medzi odlišnými operačnými systémami a pod.

- **Testovanie zabezpečenia** – keďže aplikácia vyžaduje autentizáciu a atorizáciu, tak sa testuje, či používateľ môže pristúpiť iba k častiam aplikácie, kde má povolený prístup. Ďalej sa testuje zmena URL adresy v prehliadači, aby tak nebol umožnený prístup na zabezpečené miesto.

Počas implementácie bolo testovanie zamerané na predchádzajúce popísané body. Pri výskyte nejakej chyby, alebo neočakávaného správania sa aplikácie, boli tieto nedostatky ihneď zapracované a opravené.

Vývoj a testovanie aplikácie prebiehalo lokálne na operačnom systéme Windows 10 Education (64-bit). Lokálne boli spustené aj MySQL server a MongoDB server pre databázy. Po implementovaní všetkých iterácií, bola výsledná aplikácia otestovaná v nasledujúcich prehliadačoch:

- Google Chrome, verzia 74.0.3729.157 (64-bit),
- Mozilla Firefox, verzia 66.0.5 (64-bit),
- Opera, verzia 60.0.3255.95 (64-bit),
- Microsoft Edge, verzia 44.17763.1.0 (64-bit).

Na všetkých vyššie uvedených webových prehliadačoch bola aplikácia plne funkčná, pričom neboli zistené žiadne nekompatibilné časti.

6.2 Testovanie na reálnych webových stránkach

Táto časť popisuje testovanie výslednej aplikácie pri použití na reálnych webových stránkach, ktoré sú dostupné online. Testovanie aplikácie prebiehalo na viacerých webových portáloch. Teraz bude popísaná jedna ukážka extrakčnej úlohy.

Úloha: Extrakcia internetového obchodu

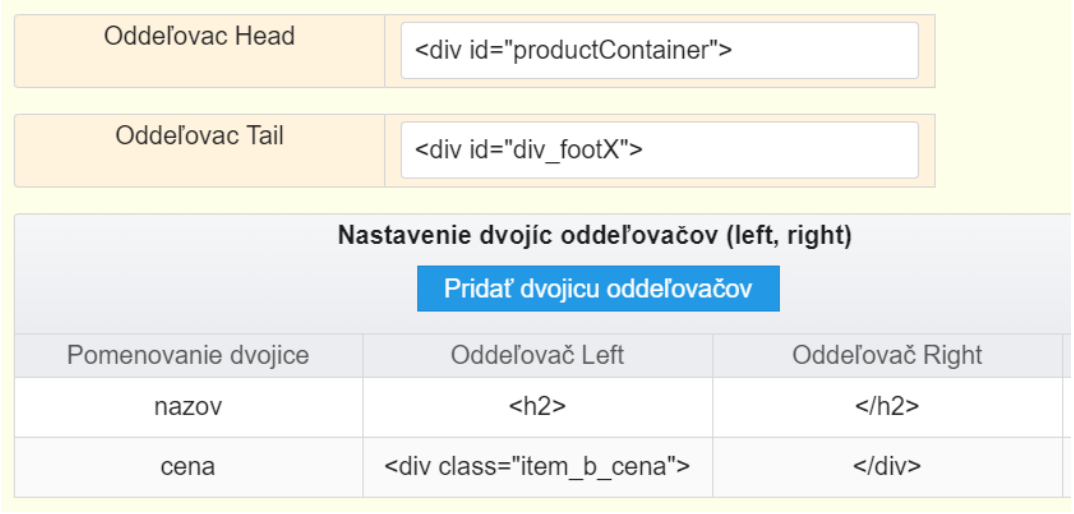
Extrakčná úloha postupne sťahuje webové stránky dostupné zo vstupnej URL adresy (seed page)[1]. Táto URL adresa odkazuje na internetový obchod do sekcie grafických kariet. Cieľom tejto úlohy je získať názov produktu a jeho cenu. Na nasledujúcom obrázku 6.1 je výrez zo snímky obrazovky zo správy úloh. Nachádzajú sa tam nastavenia úlohy a jej obmedzenia. Pri obmedzeniach hodnoty *0* a *00:00:00* znamenajú, že tento limit nie je nastavený. Takže úloha skončí buď zastavením používateľom, alebo po spracovaní všetkých dostupných webových stránok zo vstupnej URL adresy.

Názov úlohy	extrakcia eshopu mironet
Vstupná URL	https://www.mironet.cz/graficke-karty/nvidia+c14415/
Iba doména vstupnej URL	Áno
Maximálny počet stiahnutých súborov	0
Maximálna veľkosť stiahnutých dát [B]	0
Maximálna doba sťahovania [HH:mm:ss]	00:00:00
Oneskorenie HTTP požiadavkov [ms]	0

Obr. 6.1: Snímka obrazovky s konfiguráciou úlohy

[1] Vstupná URL adresa pre úlohu: https://www.mironet.cz/graficke-karty/nvidia+c14415/, navštívené: 11. 5. 2019.

Na nasledujúcom obrázku 6.2 je výrez zo snímky obrazovky pri vytváraní tejto úlohy. Je na ňom zobrazená konfigurácia HLRT wrappera pre túto extrakčnú úlohu.



Obr. 6.2: Snímka obrazovky s konfiguráciou HLRT wrappera úlohy

Ako je možné vidieť na predchádzajúcom obrázku, na extrakciu dát sa používajú dve dvojice oddeľovačov *left* a *right*. Prvá dvojica extrahuje názov produktu a druhá dvojica extrahuje cenu produktu.

Na nasledujúcom obrázku 6.3 je výrez zo snímky obrazovky zo správcu úloh. Je na ňom zobrazený záznam spustenia pre túto extrakčnú úlohu. V dôvode ukončenia tejto úlohy je ikona štvorca, čo znamená že bola zastavená používateľom.

Záznamy spustení extrakčnej úlohy

Aktivita	Čas začiatku	Čas ukončenia	Dĺžka úlohy [HH:mm:ss]	Dôvod ukončenia	Stiahnuté subory	Veľkosť stiahnutých dát [KB]	Veľkosť extrahovaných dát [KB]
✖	21. 5. 2019 16:17:37	21. 5. 2019 16:45:06	00:27:29	■	1264	510934.85	1044.695

Obr. 6.3: Snímka obrazovky zo záznamu spustenia

Na predchádzajúcom obrázku je vidieť, že za necelých 28 minút sa podarilo stiahnuť 1264 súborov a extrahovať z nich 1044.695 KB dát s názvami produktov a ich cenou. Výsledné extrahované dáta sa uložili do MongoDb databáze a používateľ si ich môže zobraziť a stiahnuť.

Testovanie aplikácie prebiehalo na viac prípadoch a odlišných webových portáloch. Niektoré z nich sú dostupné v inicializačnom skripte pre databázu, ktorý sa nachádza na priloženom vymeniteľnom médiu.

Aplikácia otestovaná na reálnych webových stránkach bola funkčná a po správnej konfigurácií HLRT wrappera sa podarilo získať požadované dáta.

Kapitola 7

Záver

Hlavným cieľom tejto diplomovej práce bolo navrhnúť a implementovať webovú aplikáciu pre definíciu a správu extrakčných úloh, ktorá by umožňovala sťahovanie webových stránok a získavanie informácií z nich. Ďalším cieľom bolo spraviť prieskum metód používaných na extrakciu informácií z HTML dokumentov. Taktiež spraviť prehľad technológií používaných pre návrh a implementáciu informačných systémov na Java EE platforme. Bolo potrebné vytvorenie návrhu architektúry pre celú aplikáciu. Úlohou bolo zvoliť extrakčnú metódu z prieskumu a potom celú aplikáciu implementovať. Na záver sa malo vykonať testovanie celej aplikácie na reálnych webových stránkach a zhodnotiť výsledky.

Všetky vyššie uvedené ciele pre diplomovú prácu boli splnené a sú podrobne popísané v predchádzajúcich kapitolách. Časť práce sa venovala základnému prehľadu webových technológií potrebných pre tvorbu webových stránok. Spomenuté bolo aj sťahovanie webových stránok (HTML dokumentov) z internetu a možné problémy, ktoré môžu pri tom nastať. Ďalej boli rozobraté a popísane viaceré prístupy pre získavanie dát z HTML dokumentov, ako sú wrappery. Práca sa venovala platforme Java EE a návrhu informačných systémov v tomto prostredí. Časť tejto práce bola venovaná návrhu aplikácie pre definíciu a správu extrakčných úloh. Tento návrh vychádzal z požiadaviek pre danú aplikáciu na platforme Java EE. Boli vytvorené príslušné návrhové diagramy, ako sú diagram prípadov použitia, diagram tried a ER diagram.

Implementácia aplikácie bola rozdelená do troch iterácií, ktoré boli popísané a znázornené príslušným diagramom. Zvolená extrakčná metóda pre získavanie dát z HTML dokumentov bol HLRT wrapper. Počas implementácie bola aplikácia stále testovaná programátorom. Taktiež sa testovala pri vzniknutom prototype z každej iterácie a výsledná aplikácia ako celok. Následne bola aplikácia otestovaná na reálnych webových stránkach a výsledky boli zhodnotené.

Aplikácia je implementovaná v programovacom jazyku Java s využitím technológií, ako sú Java EE, Spring Framework, Hibernate, PrimeFaces a mnohé ďalšie, ktoré boli tiež popísané. Vďaka aplikačnému rámcu Spring Framework je nasadenie a spustenie aplikácie jednoduché a vyžaduje minimálnu konfiguráciu.

Do budúcna by sa na tejto aplikácii mohlo vylepšiť grafické používateľské rozhranie, na ktoré teraz nebol kladený veľký dôraz. Taktiež by bolo možné pridať ďalšie extrakčné metódy, medzi ktorými by si používateľ mohol vyberať pri definícii novej extrakčnej úlohy. To by mu umožnilo širšie využitie aplikácie a viac možností pri extrakcii dát z HTML dokumentov. Architektúra aplikácie je navrhnutá na prípadné rozšírenia extrakčných metód.

Literatúra

[1] Alex, B.; Taylor, L.; Winch, R.; aj.: *Spring Security Reference*. 2017, [Online; navštívené 20.12.2018].
URL https://docs.spring.io/spring-security/site/docs/5.1.0.M1/reference/htmlsingle/

[2] Bagui, S.; Earp, R.: *Database Design Using Entity-Relationship Diagrams*. CRC Press, 2004, ISBN 9780203486054.

[3] Bittner, K.; Spence, I.: *Use Case Modeling*. Addison Wesley, 2003, ISBN 9780201709131.

[4] Broucke, S.; Baesens, B.: *Practical Web Scraping for Data Science: Best Practices and Examples with Python*. Apress, 2018, ISBN 9781484235829.

[5] Chakraborty, G.; Pagolu, M.; Garla, S.: *Text Mining and Analysis: Practical Methods, Examples, and Case Studies Using SAS*. SAS Institute, 2014, ISBN 9781612907871.

[6] Coyier, C.: *The Difference Between ID and Class*. Júl 2008, [Online; navštívené 30.11.2018].
URL https://css-tricks.com/the-difference-between-id-and-class/

[7] DeLoac, S.: *CSS to the Point*. ClickStart, Inc., 2013, ISBN 9780615212135.

[8] Doyle, M.: *Converting HTML to XHTML*. Október 2007, [Online; navštívené 3.12.2018].
URL https://www.elated.com/articles/converting-html-to-xhtml/

[9] Duckett, J.: *Beginning Web Programming with HTML, XHTML, and CSS*. Wiley, 2004, ISBN 9780764570780.

[10] Francia, S.: *MongoDB and PHP: Document-Oriented Data for Web Developers*. O'Reilly Media, 2012, ISBN 9781449324841.

[11] Ganjisaffar, Y.: *crawler4j*. [Online; navštívené 26.12.2018].
URL https://github.com/yasserg/crawler4j

[12] Goldberg, K. H.: *XML: Visual QuickStart Guide*. Peachpit Press, 2010, ISBN 9780132104319.

[13] Gutierrez, F.: *Pro Spring Boot*. Apress, 2016, ISBN 9781484214312.

[14] Hedley, J.: *jsoup: Java HTML Parser*. [Online; navštívené 26.12.2018].
URL https://jsoup.org/

[15] Jendrock, E.; Cervera-Navarro, R.; Evans, I.; aj.: *The Java EE 7 Tutorial*. Addison-Wesley, 2014, ISBN 9780133901580.

[16] Johnson, R.; Höller, J.; Arendsen, A.; aj.: *Professional Java Development with the Spring Framework*. Wiley, 2007, ISBN 9780471748946.

[17] Kumar, P.: *Website Testing Techniques*. November 2011, [Online; navštívené 2.5.2019].
URL
https://www.codeproject.com/Articles/291343/Website-Testing-Techniques

[18] Kunjumohamed, S.; Sattari, H.; Bretet, A.; aj.: *Spring MVC: Designing Real-World Web Applications*. Packt Publishing, 2016, ISBN 9781787125087.

[19] Kushmerick, N.: *Wrapper induction: Efficiency and expressiveness. Artificial Intelligence*, ročník 118, č. 1, 2000: s. 15 – 68, ISSN 0004-3702, [Online; navštívené 4.12.2018].
URL https://www.sciencedirect.com/science/article/pii/S0004370299001009

[20] Kyrnin, J.: *What Is An HTML Tag Versus an HTML Element?* November 2018, [Online; navštívené 29.11.2018].
URL https://www.lifewire.com/html-tag-vs-element-3466507

[21] Marshall, A.: *Top 10 Open Source Java and JavaEE Application Servers*. 2015, [Online; navštívené 25.12.2018].
URL https://blog.idrsolutions.com/2015/04/top-10-open-source-java-and-javaee-application-servers/

[22] Musciano, C.; Kennedy, B.: *HTML & XHTML: The Definitive Guide: The Definitive Guide*. O'Reilly Media, 2002, ISBN 9781449390853.

[23] Sanders, B.: *Smashing HTML5*. Wiley, 2010, ISBN 9780470977347.

[24] Sharma, J.; Sarin, A.: *Getting started with Spring Framework: covers Spring 5*. Createspace Independent Publishing Platform, 2017, ISBN 9781979962780.

[25] Unhelkar, B.: *Practical Object Oriented Design*. Cengage Learning, 2005, ISBN 9780170122993.

[26] Viscounty, P. J.; Barry, J. L.; Field, J.: *Spiders, Crawlers and Bots, Oh My: The Basics of Website Scraping*. Október 2012, [Online; navštívené 24.11.2018].
URL https://www.lw.com/thoughtLeadership/web-scraping-ip

Príloha A

Obsah priloženého CD

- /aplikacia – priečinok obsahujúci zdrojové súbory aplikácie
- /dokumentacia – priečinok obsahujúci zdrojové súbory (LaTeX) pre vytvorenie PDF súboru s textom práce. Taktiež obsahuje už vytvorený PDF súbor.
- /navod – priečinok obsahujúci návod na spustenie aplikácie
- /iefwp.war – zostavená aplikácia