Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Rok Burgar

Delno samodejna izdelava ovojnic za spletne vire

diplomsko delo
na univerzitetnem študiju

Mentor: doc. dr. Dejan Lavbič

Ljubljana 2012

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Univerza *v Ljubljani*
Fakulteta *za računalništvo in informatiko*

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 01809/2012
Datum: 15.03.2012

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **ROK BURGAR**

Naslov: **DELNO SAMODEJNA IZDELAVA OVOJNIC ZA SPLETNE VIRE**
SEMI-AUTOMATIC WEB SITE WRAPPER CONSTRUCTION

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

Podatki na spletnih straneh so pripravljeni za uporabnike s poudarkom na predstavitvi in privzeto niso v strukturirani obliki, primerni za računalniško obdelavo. Če želimo črpati podatke iz večih spletnih mest in jih tudi samodejno povezovati, moramo vpeljati strukturo in pripraviti meta podatke s pomočjo katerih lahko izvedemo takšno integracijo podatkov. V okviru diplomske naloge je potrebno raziskati različne možnosti luščenja podatkov iz spletnih strani in pripraviti prototip orodja, ki to podpira. Orodje naj bo namenjeno uporabnikom, ki niso tehnično podkovani in jim na ta način omogoča izdelavo ovojnic spletnih strani, s pomočjo katerih bi lahko povezali podatke večih spletnih strani.

Mentor:
doc. dr. Dejan Lavbič

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan:
prof. dr. Nikolaj Zimic

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Rok Burgar, z vpisno številko **63060045**, sem avtor diplomskega dela z naslovom:

Delno samodejna izdelava ovojnic za spletne vire

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Dejan Lavbič,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 11. september 2012 Podpis avtorja:

Rad bi se zahvalil moji družini, ki mi je stala ob strani skozi vsa leta študija.

Zahvalil bi se vsem učiteljem na FRI-ju za vse pridobljeno znanje, še posebej pa mentorju doc. dr. Dejanu Lavbiču za vse njegove nasvete in predloge.

Zahvalil bi se tudi vsem sošolcem in prijateljem, s katerimi smo si skozi pogovore izmenjali veliko znanja ter izkušenj, ne samo strokovnih, temveč tudi življenskih.

Zahvalil bi se tudi podjetju Hal Interactive in vsemu kolektivu, kjer sem skozi študentsko delo pridobil veliko izkušenj iz spletnega programiranja.

Kazalo

Seznam uporabljenih kratic in simbolov

HTML (ang. HyperText Markup Language) Označevalni jezik, ki se uporablja za opis strukture spletnih dokumentov.

PHP (ang. PHP: Hypertext Preprocessor) Splošnonamenski programski jezik, ki se večinoma uporablja za programiranje spletnih strežnikov.

CSS (ang. Cascading Style Sheets) CSS je označevalni jezik, s katerim definiramo grafično obliko spletne strani.

AJAX (ang. Asynchronous JavaScript and XML) Tehnika, ki se uporablja za pridobivanje podatkov, v brskalniku, brez osveževanja spletne strani.

DOM (ang. Document Object Model) Standardni način predstavitve XML dokumenta.

HTTP (ang. Hypertext Transfer Protocol) Protokol, ki se uporablja za prenos HTML dokumentov.

JSON (ang. JavaScript Object Notation) Format sporočila, ki se pogosto uporablja pri AJAX klicih.

ER (ang. Entity Relation) Je abstrakten model za opis podatkovnih baz.

XML (ang. Extensible Markup Language) Označevalni jezik, ki nam omogoča strukturiran opis podatkov.

API (ang. Application Programming Interface) Definiran vmesnik, preko katerega lahko komuniciramo.

SQL (ang. Structured Query Language) Jezik za poizvedovanje po podatkovnih bazah.

RDF (ang. Resource Description Framework) Standard, ki se uporablja za opis semantike.

URL (ang. Uniform Resource Locator) URL se uporablja za določanje lokacije dokumentov na spletu.

CSV (ang. Comma-Separated Values) Format za prenos podatkov.

SVN (ang. Subversion) Sistem za upravljanje z verzijami dokumentov.

Povzetek

V okviru diplomskega dela smo razvili orodje za podporo luščenju strukture podatkov iz delno strukturiranih spletnih strani. Splet je bil od vsega začetka zasnovan kot dokumentni sistem, vsak (HTML) dokument pa ima svoje metapodatke, ki opisujejo njegovo strukturo. Problem strani, ki nam jo prikaže brskalnik, je predvsem v opisovanju vizualnih lastnosti dokumenta, ne pa tudi njegove vsebine. Poleg tega so ti dokumenti narejeni po različnih standardih, hkrati pa večina spletnih strani ni 100% kompatibilnih s standardi.

Reševanje problema, kako na preprost in učinkovit način izluščiti potrebne informacije, je osrednji problem tega diplomskega dela. Ta opisuje zamisel in izdelavo dveh programov. Prvi je sestavljen iz vtičnika za brskalnik in skrbi za ročno označevanje lokacije podatkov. Drugi pa je aplikacija na strežniku, ki na podlagi označenih lokacij lahko pridobi podatke z določene spletne strani.

Ključne besede

ovojnica, luščenje podatkov, semantični splet, struktura, vtičnik za brskalnik, strežnik

Abstract

The paper describes the development of program for scraping data from partially structured web pages. Web is a document based system. Every documents have their metadata that describes their structure. Documents on the web are written in HTML. Problem with HTML is that it's primary purpose is to describe visual properties of the document and not its content. Besides that, web documents are made with different HTML standards and almost all web pages are not 100% compatible with web standards.

The paper describes how we can simply and effectivelly get the needed data from a web page. Implementation describes two main programs. The first one is a plugin for a web browser and takes care of marking the data locations. The second program runs on the web server and it gets the data from a web page based on data we marked with the first program.

Keywords

scraping template, web scraping, semantic web, structure, browser plugin, web server

Poglavje 1

Uvod

Preko svetovnega spleta je javno dostopnih ogromno podatkov. V letu 2005 je obstajalo 11,5 bilijona internetnih strani, ki so javno dostopne.[11] Vse te strani ustvarjajo veliko zalogo podatkov. Pri tem, kako priti do podatkov z interneta, obstajata dva problema:

- Kako najti podatke na internetu?
- Kako te podatke izluščiti iz dokumenta?

Iskalniki po svojih najboljših močeh rešujejo problem, kako najti podatke, ki jih potrebujemo. Če hočemo podatke samo prebrati, nam zadostuje že njihov prikaz v brskalniku. Podatki so po eni strani lahko dragoceni za odkrivanje zakonitosti, po drugi strani pa za obogatenje že obstoječih podatkov.

Obstaja več različnih rešitev za to, kako pridobiti te podatke. V diplomskem delu je predstavljen prototip programa, s katerim enostavno naredimo ovojnico z določene spletne strani in iz nje ob poljubnem času izluščimo podatke, ki nas zanimajo. Tako lahko pridobimo podatke iz več različnih strani in na podlagi teh podatkov ustvarimo skupno, obogateno stran.

Poudarek je tudi na inteligentnem zaznavanju spletne strani, tako da lahko nekatere informacije izluščimo avtomatsko, brez posega uporabnika. Poleg implementacije programa diplomsko delo vsebuje tudi opis testnega problema, ki je realiziran z uporabo programa za označevanje in združuje

podatke iz treh različnih spletnih strani. S temi podatki nadgradimo, obogatimo glavno funkcionalnost spletne strani z nepremičninami.

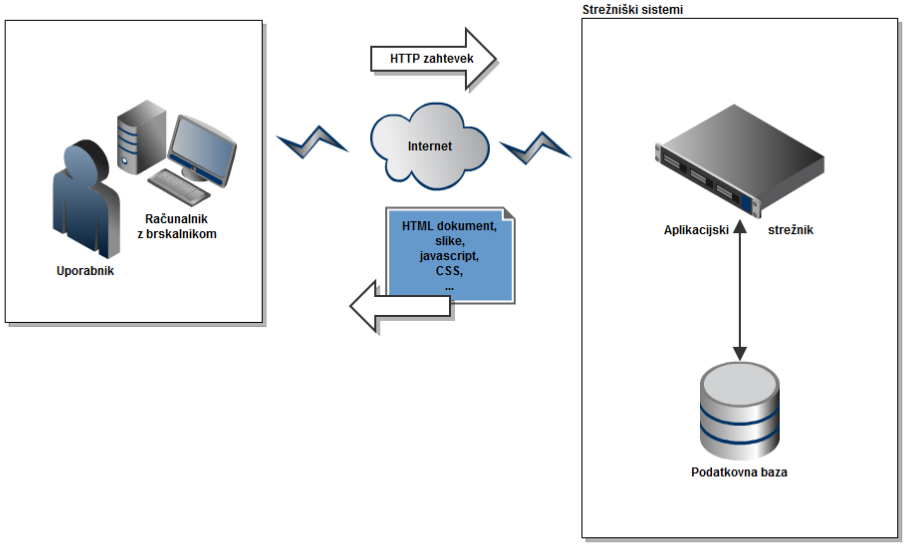
V poglavju 2 je opisan problem, kako najbolj učinkovito izluščiti podatke s trenutne spletne strani. Na podlagi opisanega problema je predstavljena zasnova in implementacija programa v poglavju 3.

Poglavje 2

Problem pridobivanja podatkov z določene spletne strani

2.1 Splošen pregled luščenja podatkov

Podmnožico interneta predstavlja splet. Predlog za zasnovo spleta je 1989 napisal Tim Berners-Lee[1]. Sestavljen je iz množice dokumentov, ki so medsebojno povezani. Vsak dokument ima določeno strukturo, ki jo imenujemo HTML. Z vpisom naslova v naš brskalnik le-ta pošlje HTTP zahtevo strežniku. Strežnik vrne dokument odjemalcu. Pri spletu je večinoma odjemalec brskalnik, ki interpretira HTML dokument (izračuna postavitev, barve, slike ...) in ga prikaže uporabniku.



Slika 2.1: Delovanje spleta

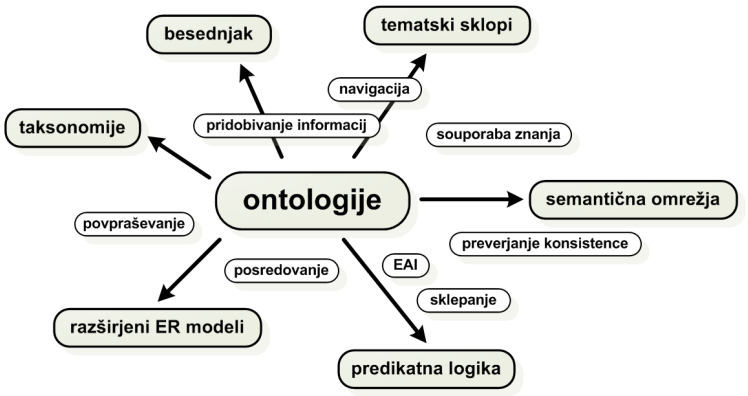
Pogosto obstaja potreba po avtomatskem zajemu podatkov iz dokumentov na določenem spletnem naslovu(npr. iskalniki, zbiranje podatkov, prikaz novic iz drugih strani ...). Podatke lahko izluščimo na dva načina:

- popolnoma avtomatizirano;
- s pomočjo ročnega označevanja, interakcije uporabnika.

Za kakšno rešitev se bomo odločili, je predvsem odvisno od tega, kakšen problem rešujemo. Primer avtomatiziranega programa je iskalnik. Program se bo sistematično sprehajal po spletu (ang. web crawler) in je popolnoma avtomatiziran, saj mora delovati hitro. Tako indeksira ogromne količine podatkov. Pri popolnoma avtomatiziranem postopku se lahko podatkom dodeli neka semantika, oziroma teža pri iskanju, ki sledi iz strukture dokumenta. Tako HTML sam po sebi, predvsem v novejših verzijah, določa zelo osnovno ontologijo podatkov.

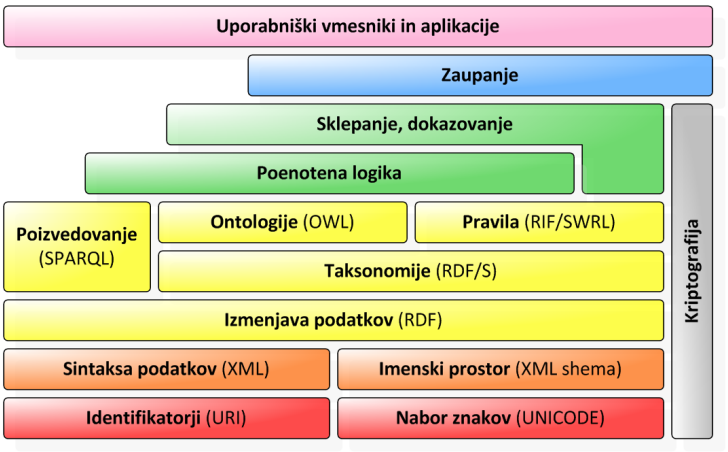
Ontologija v računalništvu pomeni skupno dojemanje besed in relacij med njimi na določenem področju, domeni. Obsega skupni slovar in nivojsko strukturo (ang. taxonomy) ločenih dokumentov. Ustvarja podlago za avtomatizacijo procesov, ki jih izvajajo računalniki. Je del semantičnega spleta, saj opredeljuje, kaj so določeni objekti.

Definicija:
Ontologijo lahko opredelimo kot formalno specifikacijo skupnih konceptov.[10]



Slika 2.2: Grafični prikaz obsega ontologije

Semantiko na višjem nivoju se lahko izlušči, če so spletne strani napisane z uporabo tehnologij, ki so bile sprejete za opis podatkov (metapodatki), ti metapodatki pa so namenjeni računalnikom. V tem primeru govorimo o semantičnem spletu, ki bo naslednja stopnja evolucije spleta. Semantični splet je W3C standard, ki promovira vključitev semantike v spletne strani. Tim Berners-Lee definira semantični splet kot splet podatkov, katerega lahko stroji procesirajo direktno ali indirektno[8]. Leta 2006 je Tim Barners-Lee izjavil, da ta preprosta ideja ostaja v veliki meri nerealizirana. [9].



Slika 2.3: Sklad tehnologij, ki se uporabljajo za semantični splet

HTML strani že od samega začetka v glavi vsebujejo tri oznake, ki se navezujejo na to, kaj se nahaja na strani.

Primer:

1. Ključne besede
   Množica besed, ki opisujejo trenutno spletno stran.
   <meta name="keywords" content="računalništvo, računalnik, prenosnik"/>

2. Opis
   Par stavkov o tem, kaj se nahaja na strani.
   <meta name="description" content="Poceni računalniška oprema."/>

3. Ime avtorja dokumenta
   <meta name="author" content="Rok Burgar"/>

Ključne besede nam ne povedo, kje v dokumentu se nahajajo podatki, ki jih predstavljajo. Tudi ni nujno, da sploh so na strani, zato si pri ročnem označevanju z njimi ne moremo pomagati. Lahko bi te ključne besede prikazali uporabniku, da dobi občutek, kaj se nahaja na trenutni strani. Če je

določen avtor, lahko ta podatek izvlečemo avtomatsko. Polja se uporabljajo pri spletnih iskalnikih, vendar imajo v zadnjem času manjšo težo kot včasih, saj iskalniki podatke raje izvlečejo s strani same. [13]

Pogosto so potrebni podatki neke strani za nadaljnjo uporabo oziroma obogatitev nekega drugega vira. V takem primeru moramo ročno označiti, katere podatke v HTML dokumentu potrebujemo. Primer tega je prikaz novic s sorodne strani na trenutni strani. Pri ročnem označevanju uporabnik označi, kje se kaj nahaja. To je veliko zamudnejše, vendar pa lahko pripiše podatkom neko semantiko, za katero ve uporabnik. Ta metoda bi bila neučinkovita pri iskanju podatkov na velikem številu strani, kot to npr. počnejo iskalniki. Uporabi se jo lahko v primeru, da rabimo podatke z natančno določene spletne strani.

Razdelitev glede na stopnje različne avtomatizacije:

1. **Izreži in prilepi**
   Včasih tudi najboljša tehnologija ne more nadomestiti pregleda spletne strani s pomočjo človeka, ki označi potrebne podatke. Seveda je tudi človek tukaj omejen s tem, kako dobro pozna tematiko strani. Pri določanju semantike enostavnim objektom je človek mnogo boljši kot današnji računalniški programi. Prav tako so strani grajene za ljudi, ljudje pa imamo priučeno skupno ontologijo.

2. **Ujemanje z regularnimi izrazi**
   Odjemalec prejme kopijo dokumenta. Dele dokumenta obdelamo z vnaprej določenimi regularnimi izrazi (oziroma s primerljivimi opisniki strukture), ki iščejo določeno strukturo z namenom, da ji pripišejo nek pomen.

3. **HTTP programiranje**
   Zahteve za dokument lahko programi pošljejo preko vtičnika (socket). Na zahtevanem dokumentu se izvede poljuben program.

4. **Izluščenje podatkov z DOM drevesom**
   Vgradnja pogona brskalnika, ki iz dokumenta zgradi DOM drevo. Po-

gon brskalnika (če to omogoča) lahko izvaja tudi odjemalčev del kode aplikacije, javascript.

5. **Algoritmi za podatkovno rudarjenje**
   Spletne strani se generirajo dinamično iz podatkovne baze. Podatki enakega tipa se praviloma zgenerirajo z enako predlogo (ang. template) strani. Algoritem lahko zazna predlogo in uporabi že zgrajeno ovojnico strani. Strani z enako predlogo se lahko identificira s podobno obliko dinamično generiranih URL-jev.

6. **XQuery**
   Z XQuery, ki uporablja XPATH, lahko iz dokumenta izluščimo določene podatke.

7. **Razni programi za izluščenje podatkov**
   So programi, ki kombinirajo pristope, opisane v tem seznamu. Nekateri tudi že vnaprej pogledajo za določeno ovojnico strani in lahko vsebujejo vtičnike za pisanje podatkov v lokalno podatkovno bazo.

8. **Vertikalno agregirane platforme**
   Specifično za nekatera podjetja.

9. **Razpoznavanje semantične anotacije**
   Tukaj imamo v mislih semantične dele HTML jezika in namenske razširitve HTML jezika. Predstavniki teh razširitev so Microdata, Microformat, RDF. Omogočajo razširitev HTML strukture z atributi, ki vsebujejo določeno semantiko. Spletni iskalnik Google trenutno uporablja Microdata za prikaz rezultatov iskanja[14]. Prednost tega je, da imamo neki standardni slovar in je zato globalno znan pomen določenih podatkov.

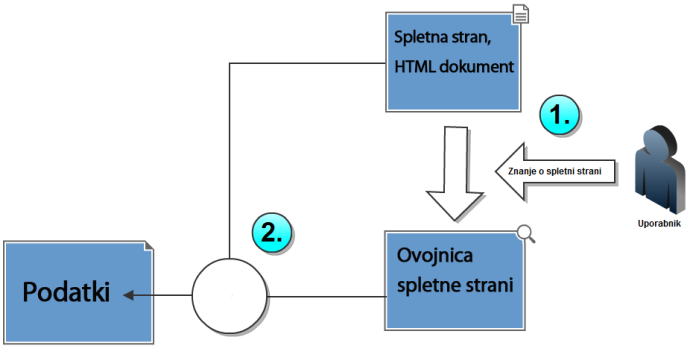
   Poleg zgoraj omenjenih razširitev je veliko bolj semantična tudi najnovejša verzija HTML-ja. HTML5 vsebuje veliko novih semantičnih elementov [5](primer: nav – navigacija, article – članek, footer – spodnji del strani, header – glava strani, meter – določena izmerjena velikost ...).

Naštete metode niso izključujoče. Za katero se bomo odločili, je odvisno predvsem od problema, ki ga moramo rešiti. Diplomsko delo se osredotoča predvsem na problem izdelave ovojnice določene spletne strani in osveževanje podatkov ovojnice v svoji podatkovni bazi. Proces naj bo čim bolj avtomatiziran. Označevanje naj poteka vizualno. Ko imamo podatke v naši bazi, lahko iz njih naredimo sestavljene spletne strani oziroma izvajamo določene akcije ob določenih pogojih. Pri problemu nam mora uporabnik povedati, kateri podatke na spletni strani so dejansko uporabni in kako naj se imenujejo. Ti se bodo preslikali iz dokumenta v podatkovno bazo. Podatki morajo biti sveži, vendar mora obstajati določena doba uporabnosti. Tako si zagotovimo, da ne preobremenimo strani, s katerih pridobivamo podatke. To bi se lahko zgodilo, če bi za vsako našo zahtevo preverili spremembo podatkov na strežniku. Zato si hranimo lokalno kopijo podatkov za naše potrebe (ang. cache).

2.2 Definicija problema

Definicija problema:

Preslikava potrebnih podatkov, na poljubni spletni strani, v določen izhodni format, s pomočjo znanja uporabnika o trenutni strani.



Slika 2.4: Grafična predstavitev definicije problema

Uporabnik z opredelitvijo lokacije in pomena elementov na strani kreira ovojnico spletne strani. To ovojnico si shranimo. Tako lahko korak 1, na sliki 2.4, naredimo enkrat, korak 2 pa lahko avtomatiziramo, tako da se lahko izvaja brez posega uporabnika.

Današnje spletne strani se generirajo dinamično. Temeljijo na podatkovni bazi, iz katere se generira in osvežuje množica spletnih strani. Če imamo ovojnico neke strani, lahko na njeni podlagi črpamo te podatke tudi v prihodnosti oziroma lahko črpamo podatke iz podobnih spletnih strani. Podobne spletne strani so spletne strani, ki imajo enako HTML strukturo.

Razčlenitev osnovnega problema na manjše naloge:

- Enostavno označevanje potrebnih podatkov[1].
- Shranjevanje ovojnice strani.
- Osveževanje podatkov na podlagi ovojnice.

2.3 Obstoječe rešitve

Na spletu obstaja množica rešitev za pridobivanje podatkov s spletnih strani. Tukaj bomo predstavili tiste, s katerimi se da rešiti zastavljeni problem in vsebujejo podobnosti z našim programom.

Obstoječe rešitve lahko razčlenimo po klasifikaciji na strani 7. Večina rešitev spada pod točko 7, saj je problem zastavljen dokaj splošno in zato uporablja celovite rešitve. Uporabljajo skoraj vse metode klasifikacije. Med programi obstajajo ogromne razlike predvsem v načinu uporabe in potrebnemu tehničnemu znanju uporabnika. Predstavljeni so po vseh zahtevanih treh korakih za rešitev problema, ki so opisani na strani 10.

2.3.1 Enostavno označevanje potrebnih podatkov

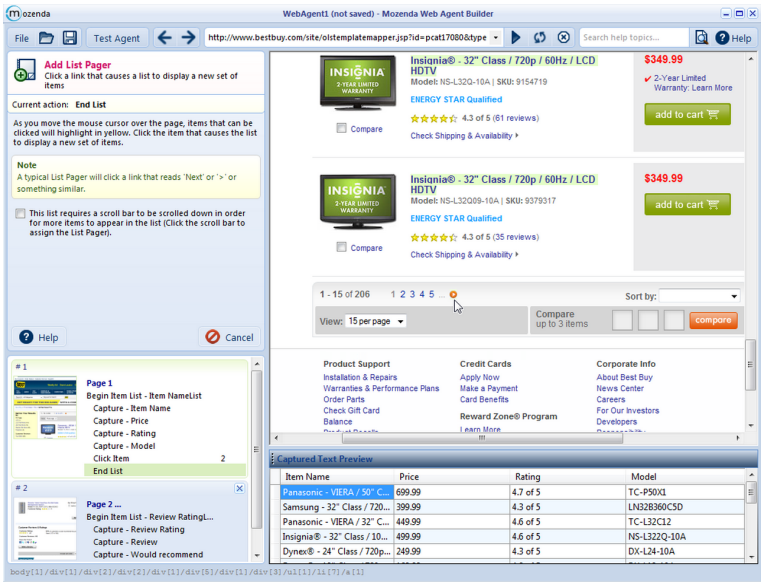
Označevanje podatkov imajo rešeno na tri različne načine:

[1]Enostavno pomeni, da je tehnično nezahtevno. To pomeni vizualno in ne pisanje programske kode, XPath, oziroma regularnih izrazov za stran

- **Preko vgrajenega brskalnika**

  Program ima integriran brskalnik, ki prikazuje strani. Prednost v tem pristopu je, da strežnik in odjemalec uporabljata isti HTML razčlenjevalnik (ang. parser). Prav tako je prednost tega pristopa, da lahko vgrajeni brskalnik izvede odjemalčev del kode dokumenta(javascript). Tako nikoli ne pride do nedoslednosti v označenih in dobljenih podatkih.

  Primer: Mozenda [19]

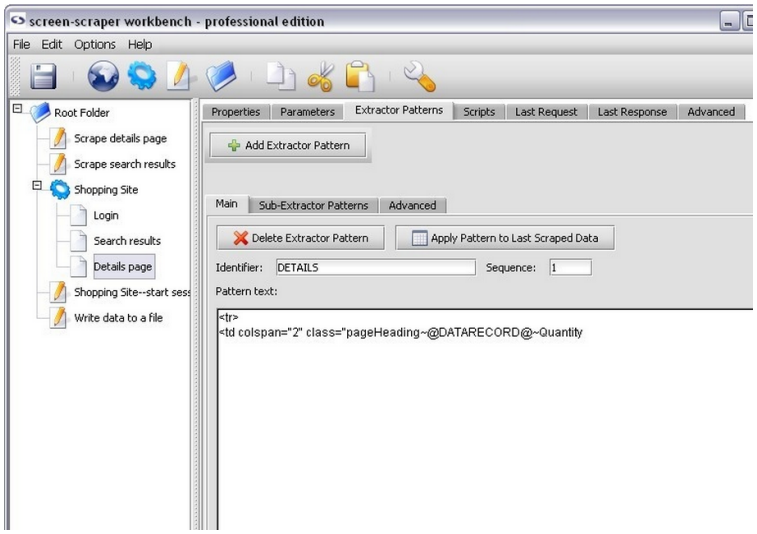


Slika 2.5: Zaslonska maska Mozende z vgrajenim brskalnikom

- **Preko proxy prestrezanja**

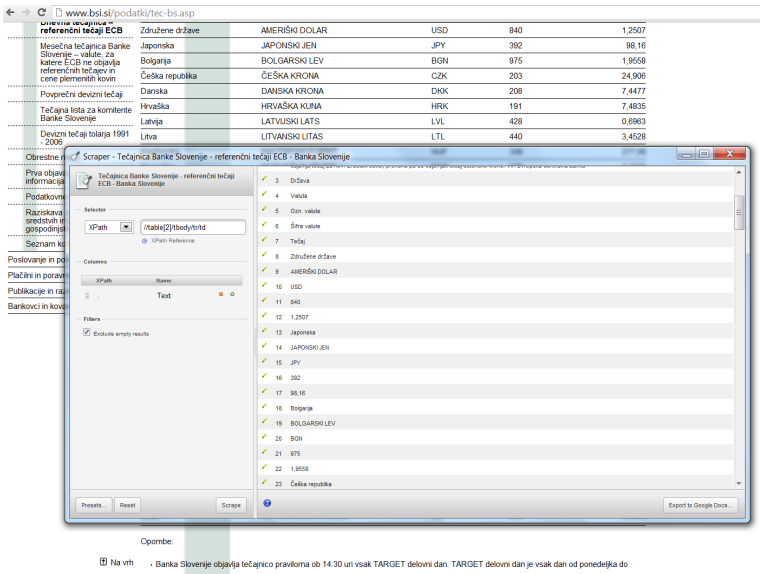
  Program preko proxy-ja prestreza ves HTTP promet brskalnika. Uporabnik izbere prestreženo spletno stran v programu. Nato uporabnik v brskalniku označi želeno besedilo in ga prilepi v program. Aplikacija razpozna, kje se nahaja tekst in iz tega izlušči ovojnico strani. Ta pristop odpravi neujemanje dokumenta med brskalnikom in aplikacijo, vendar pa aplikacija sama na izvaja javascripta, tako da lahko pride do nedoslednosti. Če želimo odpraviti še ta problem, lahko v aplikacijo vgradimo pogon brskalnika.

Primer: Screen Scrapper [22]



Slika 2.6: Zaslonska maska Screen Scraper s kopiranim tekstom

- **Vtičnika za brskalnik**
  Označevanje podatkov poteka v brskalniku, ki ga vsakodnevno uporabljamo. Praviloma se zažene z kombinacijo določenih tipk. Ta primer je najbolj enostaven, vendar pa lahko pride do nedoslednosti med aplikacijo, ki te podatke pridobi, in vtičnikom za brskalnik. Primer: Scrapy [20]



Slika 2.7: Zaslonska maska Scrapy

2.3.2 Shranjevanje ovojnice strani

Shranjevanje ovojnice imajo programi rešeno na tri načine:

- **Na oddaljenem strežniku** Primer: Mozenda
- **Aplikacija na našem računalniku** Primer: Screen Scrapper
- **V brskalniku** Primer: Scrapy

2.3.3 Osveževanje podatkov na podlagi ovojnice

Večinoma vsi programi podpirajo pridobivanje podatkov na podlagi ovojnic strani.

Naslednje rešitve ne izpolnjujejo prvega pogoja za rešitev našega problema, saj so tehnično bolj zahtevne in se predvsem uporabljajo kot del programov, oziroma vmesni nivo(ang. middleware), ki nam pomaga do rešitve.

- Knjižnica Scrapy[21] za python omogoča programerjem hitro izdelavo ovojnic in uporabo te možnosti v svojih programih.

- XQuery je deklarativni jezik, s katerim povemo, katere podatke v dokumentu rabimo. Uporablja XPath izraze.
- Na nižjem nivoju imamo programe, ki uporabljajo XPath. Večina popularnih programskih jezikov ima vgrajeno podporo za XPath.
- Na najnižjem nivoju lahko strukturo dokumenta preiskujemo z regularnimi izrazi, kar pomeni preiskovanje določene strukture direktno po tekstu. Večina programskih jezikov vsebuje podporo za regularne izraze.

Poglavje 3

Implementacija

3.1 Podlaga za predlagano implementacijo

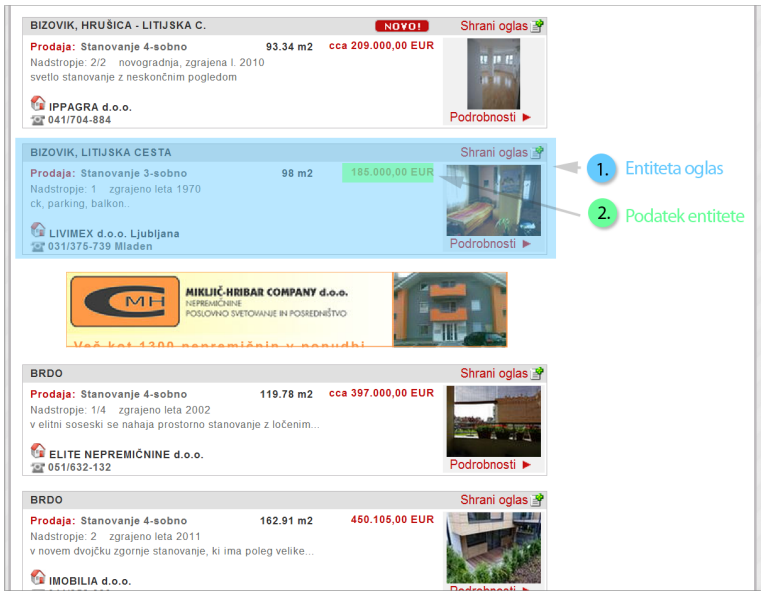
3.1.1 Sestava spletnega dokumenta

Namen programa je preslikava potrebnih podatkov HTML strani v izhodni format.

Dokument HTML je sestavljen iz HTML elementov. Podatke na strani označimo tako, da označimo HTML elemente s podatki, ki nas zanimajo. Na višjem semantičnem nivoju posamezni HTML elementi ali več HTML elementov tvorijo neko entiteto s podatki, ki nas zanimajo. Z označitvijo lokacije podatkov naredimo ovojnico spletne strani, dokumenta. Če na strani ne obstaja ponavljajočih elementov, lahko privzamemo, da je entiteta kar spletna stran sama.

Na eni spletni strani lahko obstaja več entitet enakega tipa. Za primer imamo seznam entitet oglasov za nepremičnine na sliki 3.1. To lahko zaznamo zaradi ponavljajočih HTML elementov in enake strukture kot pri prvi entiteti, katere podatke smo označili. Ponavljajoči HTML elementi se tipično nahajajo v HTML elementih, kot so <ul>, <ol>, <table>. Posamezne entitete so točka seznama(<li>- list item), oziroma vrstica tabele(<tr>- table row) Velikokrat imamo seznam entitet narejen z generičnimi HTML elementi,

kot je . Pri ogledu strukture vidimo, da se entitete še vedno nahajajo na enakem HTML nivoju (ang. siblings), enako kot pri seznamu, tabeli. Na eni spletni strani imamo lahko 0...n entitetnih tipov. Vsak entitetni tip ima lahko 1..n entitet. Ko uporabnik označi podatek ene entitete na strani, mora program zaznati, če se na strani nahaja še kakšna entiteta istega entitetnega tipa. Podatki ene entitete se na spletni strani večinoma nahajajo, tako da imajo skupnega prednika (ang. ancestor). Do tega prihaja predvsem zato, ker so vizualno podatki ene entitete na skupnem mestu.



Slika 3.1: Slika entitet oglasov za nepremičnine

Lastnosti entitet, ki se nahajajo v določenem HTML dokumentu:

1. Na strani se lahko nahaja 1...n entitet, katerega podatke smo označili.
2. Enake entitete imajo večinoma enako HTML strukturo.
3. Entitete se večinoma nahajajo na enakem nivoju v HTML hierarhiji (ang. siblings).
4. Podatki iste entitete imajo enakega prednika(ang. ancestor).

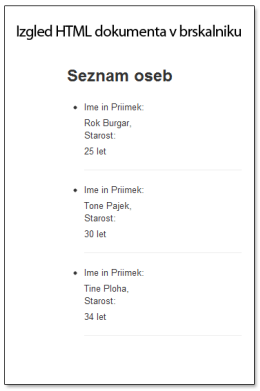
```
<h2>Seznam oseb</h2>

<br/>

<ul>
  <li>
        <label>Ime in Priimek:</label>
        <span>Rok Burgar</span>,
        <label>Starost:</label>
        <span>25 let</span>
        <hr/>
  </li>
  <li>
        <label>Ime in Priimek:</label>
        <span>Tone Pajek</span>,
        <label>Starost:</label>
        <span>30 let</span>
        <hr/>
  </li>
  <li>
        <label>Ime in Priimek:</label>
        <span>Tine Ploha</span>,
        <label>Starost:</label>
        <span>34 let</span>
        <hr/>
  </li>

  ...

<ul>
```



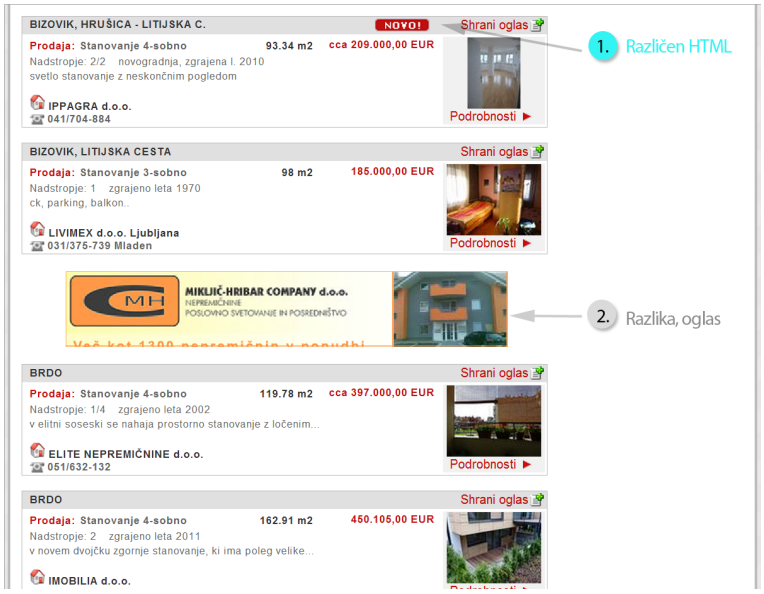
Slika 3.2: Slika lastnosti entitet

3.1.2 Algoritem za iskanje entitet

Če izkoristimo vse pogoje, lahko napišemo algoritem za iskanje entitetnega tipa. Označeni podatek je lahko del entitete. Ker imajo podatki iste entitete istega prednika, velja, da bo koren entitete HTML element med označenim podatkom in korenom HTML dokumenta. Da potrdimo, da je neko vozlišče koren entitete, moramo uporabiti lastnost 2 in 3. Ko se sprehajamo proti korenu HTML elementa, moramo preveriti vse sorojence (ang. siblings) trenutnega vozlišča. Pogoj, da potrdimo koren entitete, mora biti enaka HTML struktura sorojenca. Torej moramo primerjati HTML strukturo sorojenca s strukturo trenutnega vozlišča. Sorojencev je običajno več kot eden. Tako imamo možnost, da preverimo vse sorojence ali pa se zadovoljimo s preverjanjem prvega sorojenca. Pri slednji možnosti obstaja verjetnost, da napačno zavrnemo vozlišče (ang. FP – false positive), ki je kandidat za entiteto, zaradi rahlo spremenjene HTML strukture. To se lahko zgodi zaradi dveh logičnih razmislekov o HTML strukturi, ki lahko nastopijo v dokumentu:

- Sorojenec ne vsebuje podatka, ki smo ga označili. Primer: označili smo telefonsko številko, ampak, ker ni nujna, je sorejenec nima izpisane v HTML strukturi.

- Sorojenec, ki ga primerjamo, je z namenom drugačen po HTML strukturi, čeprav je del seznama. Primer: oglas med predmeti v spletni trgovini.



Slika 3.3: HTML razlike pri seznamu entitet

Druga metoda je tako manj zanesljiva, vendar pa je hitrejša.

```
function getHTMLRootEntity( selectedNode ) {

    node = selectedNode;
    entityRoot = null;
    do {
        siblings = node.siblings;

        foreach siblings as sibling:

            descendantsNode = node.descendants
            descendantsSiblings = sibling.descendants

            i = 0;
            doesMatch = true;
            foreach descendantsNode as dNode:
                if (descendantsSiblings[i].HTMLtype !== dNode.HTMLtype){
                    doesMatch = false;
                    break;
                }
                i++;
            endforeach;

            if (doesMatch) {
                return node; // če preverjamo samo ujemanje enega
            }

        endforeach;

        node = node.parent;

    } while(node !== rootHTML);

}
```

Slika 3.4: Psevdokoda algoritma za zaznavanje entitet

Časovna kompleksnost

Pregledamo do korena HTML dokumenta:
O(n)
Za vsako vozlišče pregledamo samo enega ali vse sorojence (ang. siblings):
O(1) proti O(m)
Za vsako vozlišče in sorojenca preverimo ujemanje vsakega HTML elementa (ang. descendants):
O(o)

Velja:
n ... število vozlišč od izbranega do korena HTML dokumenta
m ... število sorejencev trenutnega vozlišča (ang. siblings)
o ... število vseh HTML elementov trenutnega vozlišča (ang. descendants)

Končna komplekstnost:
Pregledamo vse:
O(n)*O(m)*O(o)
Pregledamo enega:
O(n)*O(1)*O(o) = O(m*o)

Iz HTML strukture torej lahko izluščimo, da je označeni podatek del ponavljajočih se entitet. To moramo preveriti pri vsakem označenem podatku na strani. Ker mora biti označevanje podatkov vizualno in enostavno, moramo uporabniku vizualno predstaviti, da se podatek ponavlja. Program mora zato klicati algoritem takoj, ko uporabnik označi nek podatek ter nato rezultate algoritma vizualno predstaviti.

Algoritem nam pove, da je določen označen podatek del entitete in da je na strani seznam teh entitet. Algoritem pa nam ne pove, kaj točno pomeni ta entitetni tip. To s tem algoritmom ne moremo odkriti, kar tudi ni njegov namen. Za avtomatizacijo tega dela bi morali implementirati napredne tehnologije semantičnega spleta. Problem tega pa je zanemarljiv odstotek strani, grajenih s temi tehnologijami. Vendar pa bistvo tega programa ni popolna avtomatizacija. Tako nam lahko entitetni tip pove uporabnik oziroma nam pove, kaj njemu določena entiteta predstavlja. Pri tem ni problema, če bo podatke uporabljal on, saj ve, kaj pomenijo.

Na strani 15 smo tudi omenili, da je lahko entiteta tudi celotna spletna stran sama. To situacijo lahko privzamemo, če ne najdemo entitetnega tipa podatka, ki smo ga označili. Spletne strani so grajene dinamično. Na podlagi predlog se iz podatkovne baze generira množica spletnih strani. Ker predloga vsebuje HTML strukturo strani, lahko na podlagi lastnosti 2 na strani 16, označene podatke dobimo tudi s strani, ki so generirane iz enake predloge. Podatke izluščimo tako, da uporabimo isto ovojnico za množico različnih URL naslovov. Dinamične naslove enakih predlog lahko pridobimo na tri načine:

- **Z ročnim vnosom**

Z malo znanja lahko ljudje prepoznamo, da gre za enako predlogo.

- **Preko generatorja naslovov**
  Za dinamične URL naslove običajno obstaja algoritem, po katerem se generirajo naslovi. V nekaterih primerih hoče imeti spletna stran neuganljive naslove (primer: stran za izmenjavo datotek, ki generira enkraten naslov). Takrat se uporabi naključne nize, katerih se ne da avtomatsko generirati. Zaradi vloge URL naslovov pri spletnih iskalnikih[13], dandanes ogromno strani generira spletni naslov na podlagi vsebine na njih (primer: URL naslov je naslov članka na strani). V teh primerih generatorji odpovedo in moramo uporabiti klasično ročno označevanje ali pa lahko dobimo seznam naslovov preko seznama strani.

- **Preko seznama strani iz enake predloge**
  Seznam naslovov, katerega naslovi vedno kažejo na stran, ki se generira iz enake predloge (ang. template). Eden od primerov, kjer je to lahko določiti, so gumbi za sprehajanje po straneh (ang. pager). Te se uporabljajo za sprehajanje po seznamu nekih entitet, razdeljenih po več straneh. Prepoznamo jih po enaki HTML strukturi. Gumbi so tudi sami seznam entitet, ker je določen gumb entiteta, in kot podatek vsebujejo naraščajoča števila strani (primer: 1, 2, ..., 4, 5, 6 ...). Če naredimo algoritem za zaznavanje teh gumbov, lahko iz spleta avtomatsko izluščimo celoten seznam entitet. Uporabnik lahko tudi ročno označi seznam naslovov na neki strani, ki kažejo na enako predlogo. Ti seznami pogosto obstajajo, saj se uporabljajo za usmerjanje iskalnikov na določene URL-je spletnega portala in zaradi vizualne predstavitve strukture portala uporabniku. Reče se jim zemljevid strani (ang. Site map)[12].

3.1.3 Detekcija osnovnih podatkovnih tipov

Uporabniško izkušnjo lahko še dodatno nadgradimo z avtomatiziranjem pridobivanja osnovne semantike iz označenih podatkov. Tako lahko z upo-

rabo regularnih izrazov določimo tip podatka. Podatke lahko razdelimo na osnovne podatkovne tipe:

- Tekst
- Celo število
- Število
- Da/Ne (ang. boolean)
- idr.

Zaznamo lahko tudi nekatere standardne strukture, ki jim lahko podamo nek pomen: datum, površina, dolžina, teža, cena ... Te podatke lahko avtomatsko razdelimo na posamezne komponente, kot so npr. enota in količina. Za te enote imamo lahko tudi skupni slovar, saj se večinoma označujejo vedno enako. To je tudi razlog, da jim lahko pripišemo neko semantiko z regularnimi izrazi.

3.1.4 Označevanje podatkov

Uporabnik bi lahko označil vse podatke na strani, ki jih potrebuje. To je lahko zelo zamudno, še posebno, če se podatki na strani ponavljajo. Zato je uporabniko veliko enostavneje, če program sam zazna entitete na strani. Zaradi večje enostavnosti ima zato lahko program vgrajena pravila in algoritme, kot je algoritem na sliki 3.4 na strani 19, ki izkoriščajo zakonitosti semantike v HTML dokumentih in s tem uporabniku pomagajo pri izdelavi ovojnice.

V poglavju 2, kjer smo definirali problem, smo na strani 10 omenili tri glavne potrebne naloge, ki jih mora znati izvajati program. Naštete naloge si ne sledijo vedno zaporedno. Če imamo ovojnico neke spletne strani oziroma jo dobimo od drugod, nam ni potrebno označiti podatkov, ki jih potrebujemo. Razčlenitev osnovnega problema nam predlaga ločitev in neodvisnost funkcionalnosti teh treh delov programa.

Enostavnost označevanja podatkov je bistvena za ljudi, ki nimajo potrebnega tehničnega znanja. Velikokrat se je izkazalo, da se neizkušeni uporabniki veliko bolje znajdejo, če je postopek predstavljen na vizualni način. Prav tako vizualizacija pomaga neizkušenim in izkušenim uporabnikom pri lažji predstavitvi problema in akcij, ki jih uporabniki izvajajo. Za prvi korak je tako bistvenega pomena, da se označevanje podatkov izvaja preko grafičnega vmesnika.

Uporabnik je pogosto vajen svojega brskalnika, saj ga uporablja vsakodnevno. Tako je za njega najenostavneje, če bi lahko označil podatke, ki jih potrebuje v svojem brskalniku. Vtičniki za brskalnike obstajajo odkar obstajajo brskalniki sami. V svoji moderni obliki pa izvajajo dve nalogi:

1. Nadgradijo delovanje brskalnika samega.

2. Izvajajo določene naloge na trenutni strani.

Tako lahko razvijemo vtičnik za poljuben moderni spletni brskalnik, s katerim bomo označili lokacijo podatkov, ki nas zanimajo. V brskalniku ne potrebujemo vedno orodij za izdelavo ovojnic spletne strani. Aktivacijo vtičnika se tako po potrebi sproži s poljubno kombinacijo tipk.

3.1.5 Shranjevanje ovojnice

Ko naredimo ovojnico strani (v brskalniku), jo mora določen program shraniti. Ta program bi lahko bil tudi vtičnik s katerim smo naredili ovojnico. To rešitev uporablja program Scrapy. Pri tem nastanejo težave, saj vtičniki primarno niso bili narejeni s tem namenom. Vtičnik bi moral periodično osveževati podatke. To bi bilo mogoče, če bi brskalnik neprestano tekel v ozadju. Ker je brskalnik primarno namenjen brskanju po spletu, si deli svoj prostor z drugimi opravili, ki jih izvaja uporabnik. To lahko privede do nestabilnosti samega brskalnika, kar ogroža tudi našo aplikacijo. Obstaja tudi omejitev, da lahko shranjujejo samo v lokalno bazo brskalnika. Tako bi moral uporabnik izvoziti podatke ročno v podatkovno bazo, saj ni dovoljeno izvajati zunanjih akcij brez privolitve uporabnika. Skrapy zato omogoča ročni

izvoz podatkov v googledocs, vendar mora uporabnik vedno sam izvesti to nalogo.

Rešitev tega problema je, da se ovojnice shranjujejo z drugim programom. Tako vtičnik našemu programu preda lokacijo podatkov. Z ločitvijo vtičnika za brskalnik, s katerim označimo podatke, od shranjevanja pridobimo tudi druge prednosti. Tako imamo lahko centraliziran strežnik za naše ovojnice. Poljubno število brskalnikov lahko naloži vtičnik, podatke pa pošljejo istemu strežniku. S tem razbremenimo uporabnika brskalnika in naredimo našo aplikacijo na uporabnikovi strani enostavnejšo, saj mora uporabnik imeti nameščen samo vtičnik za brskalnik. S tem tudi ločimo potrebo po dosegljivosti podatkov. Če bi program tekel na računalniku, kjer smo zajeli ovojnico, bi moral biti ta računalnik vedno dosegljiv, ko bi rabili podatke. Tako pa imamo strežnik, ki je vedno delujoč in so vsi podatki vedno dosegljivi. Obstajajo tudi slabosti takega sistema. Pri centraliziranem sistemu dobimo točko odpovedi (ang. Single point of failure), tako moramo poskrbeti za ustrezno redundanco, zanesljivost strežnika. Druga slabost je pri interpretaciji podatkov, opisana na strani 3.1.6, saj se lahko ta razlikuje na strežniku in brskalniku.

3.1.6 Komunikacija

Zaradi naše arhitekture odjemalec (vtičnik za brskalnik)–strežnik, je potrebna medsebojna komunikacija.

Problem komunikacije skriva več manjših problemov:

1. Po katerem standardu se bosta odjemalec in strežnik pogovarjala?
2. Kakšna bo oblika podatkov za komunikacijo?
3. Kakšen bo vmesnik API(ang. interface API)?
4. Določiti se mora, kateri jezik bosta uporabljala za iskanje podatkov v dokumentih.
5. Oba morata enako interpretirati dobljeni dokument.

Ko imamo arhitekturo strežnik–odjemalec, je potrebno strogo ločiti naloge. Kaj točno bo program omogočal ter katere naloge bo prevzel strežnik in katere odjemalec.

Odjemalec:

- Označevanje položaja podatkov
- Izvajanje vizualizacije
- Izvajanje algoritma za zaznavo entitet in vizualna predstavitev
- Avtomatsko zaznavanje osnovnih tipov
- Možnost klicanja določenih nalog strežnika

Strežnik:

- Shramba položaja podatkov.
- Shramba strani posamezne ovojnice.
- Kreiranje shem za izhodne podatke.
- Razbitje avtomatsko zaznanih podatkov.
- Osveževanje podatkov.

3.1.7 Osveževanje podatkov na podlagi ovojnice

Ko imamo shranjeno ovojnico, lahko ob poljubnem času zahtevamo osvežitev podatkov z določenega spletnega naslova. Ta možnost nam doda novo dimenzijo, saj lahko beležimo podatke skozi čas. Te podatke lahko program priskrbi v poljubni obliki (npr. podatkovna baza, CSV, XML, JSON ...). V izdelanem programu obstaja podpora za izvoz v podatkovno bazo. Na podlagi ovojnice spletne strani mora program zgraditi potrebne tabele za hranjenje podatkov. Vsaka tabela bo predstavljala entitetni tip, kamor bomo zapisovali entitete na določeni strani. Entitete zazna odjemalec z algoritmom, prikazanim na sliki 3.4, ki se nahaja na strani 19.

Na spletni strani se lahko nahaja več entitet istega entitetnega tipa naenkrat, zato nastane pri osveževanju podatkov s spletne strani do več vrstic v tabeli. Problem nastane, ker beležimo v isti tabeli podatke skozi daljše časovno obdobje. Tako ne moremo vedeti, ali je vrstica posledica podatkov skozi čas, ali pa posledica večih enakih entitet na strani. Rešitev je ta, da ima vsaka vrstica številko osveževanja. Tako imajo entitete, ki so se ob enakem času zapisale v določeno tabelo, enako številko osveževanja in jih lahko na podlagi tega ločimo od prejšnjih entitet, ki so nastale skozi čas. Številke osveževanja delujejo kot časovni žig.

Ko imamo shranjene ovojnice, se moramo vprašati, kako bomo osveževali podatke:

- **Ročno**
  Za uporabnika bi bilo zelo enostavno, če bi naredili kontrolno aplikacijo, oziroma spletno stran, kjer lahko izbere ovojnico in ročno sproži posodobitev podatkov. Poleg tega bi bilo dobro, če bi stran omogočala pregled trenutnih podatkov ovojnice. Tako lahko uporabnik preveri, če so podatki ažurni in celoviti. Realizacija nadzorne plošče je opisana na strani 48.

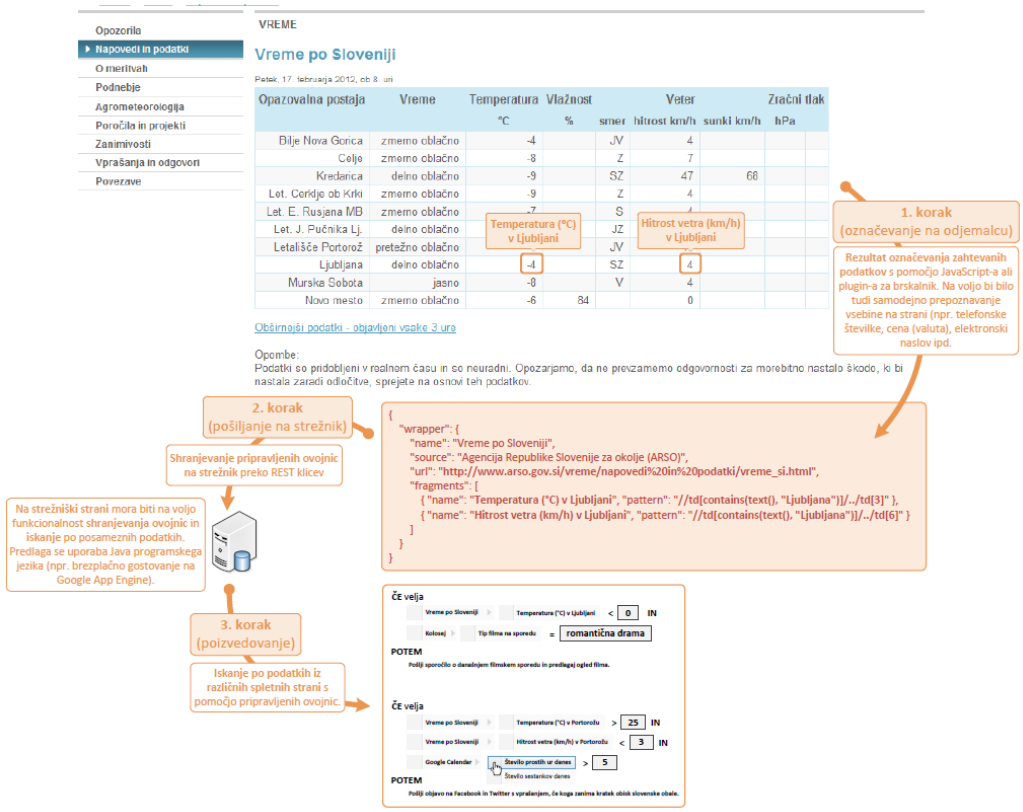
- **Avtomatsko intervalno**
  Osveževanje podatkov po preteku določenega časovnega intervala

- **Programsko**
  Neka aplikacija, ki deluje na podlagi zbranih podatkov, lahko sproži zahtevo za osvežitev. Ta možnost nam lahko pospeši aplikacijo, saj tako pridobimo možnost za leno nalaganje (ang. lazy loading) podatkov, oziroma naložimo podatke šele, ko jih uporabnik prvič potrebuje v aplikaciji. Če podatki že obstajajo, jih lahko aplikacija posodobi na podlagi časovnega žiga podatkov (ang. timestamp), upoštevajoč vnaprej definiran interval zapadlosti podatkov.

Pogoj: Osveževanje podatkov spletne strani lahko naredimo, če se struktura spletne strani ni spremenila.

3.1.8 Povzetek rešitve

Za boljšo organiziranost bomo ovojnice združili v projekte. Tako ustvarimo imenske prostore, kjer se nahajajo naše ovojnice. Projekt izberemo, ko v odjemalcu shranimo ovojnico. Priročno bi tudi bilo, da bi lahko pri shranjevanju vpisali intervalno osveževanje podatkov.



Slika 3.5: Grafični pregled zamisli, ki bo reševala naš problem.

3.2 Tehnične podrobnosti implementacije

3.2.1 Uporabljene tehnologije in orodja

HTML

HTML je označevalni jezik, s katerim pišemo dokumente za splet. Določa osnovne gradnike spletnih strani, na podlagi katerih se zgradi spletni dokument. Daje strukturo podatkom. Podrobno obliko strani se v današnjih časih določa s CSS-jem, včasih pa se je v ta namen uporabljal tudi HTML. Najnovejša različica je HTML5, ki še ni dokončna, vendar je že v veliki meri implementirana v trenutne brskalnike. HTML5 podpira veliko več semantičnih elementov kot prejšnje verzije jezika (primer: nav, footer, header ...).[5] Pri implementaciji ga uporabljamo skupaj s CSS-jem za izgradnjo grafičnega vmesnika odjemalca.

XHTML

Razlika med HTML in XHTML je ta, da je XHTML verzija XML-ja. HTML ni nujno XML dokument, saj po standardu ne rabi upoštevati vseh pravil(npr. zaključitev elementov in naštevanje atributov). To lahko pri HTML-ju pripelje do težav pri interpretaciji datoteke. Največ težav imajo majhni, preprosti programi, saj pogosto nimajo vseh pravil za razčlenjevanje HTML dokumenta. Poleg tega si večji programi, brskalniki, interpretirajo HTML pogosto vsak po svoje, tako da dobimo od primera do primera drugačne podatke, videz. Slaba strukturiranost spletnih dokumentov in nezapleten razčlenjevalnik na strežniku, sta v implementaciji pripeljala do težav v komunikaciji.

CSS

S CSS-jem se določa predvsem videz spletne strani. Spada med označevalne jezike. Definira stile in postavitev elementov, selektorjev in razredov. Poleg tega omogoča različen prikaz za različne izhodne potrebe. Pri implemen-

taciji se skupaj s HTML-jem uporablja za izgradnjo grafičnega vmesnika odjemalca.

PHP

Je najbolj razširjen splošnonamenski strežniški jezik. Koda je interpretirana, zato je pred zagonom ni potrebno prevesti. Od verzije 3 PHP podpira objekte. Pri odjemalec–strežnik internetni arhitekturi se ob HTTP zahtevku kliče PHP skripta, ki vrne dokument v HTML obliki. Razred za dostop do baze PDO je bil predstavljen v PHP-u z verzijo 5.1. V naši arhitekturi je strežnik napisan s PHP-jem. Z uporabo PDO razreda in parametriziranimi poizvedbami se izognemo SQL vrivanju.

MYSQL

Je sistem za upravljanje s podatkovno bazo (SUPB). Podatkovna baza je relacijska in je ena izmed najbolj razširjenih podatkovnih baz. V MySQL podatkovno bazo naš strežniški program shranjuje ovojnice in podatke določene spletne strani.

HTTP

HTTP je aplikacijski protokol za prenos HTML dokumenta. Je osnova za prenos podatkov svetovnega spleta.

javascript

Vtičniki za brskalnik Chrome so pisani v javascriptu. Tako je v njem pisan tudi naš celotni odjemalec. Javascript je dinamičen, objektni splošnonamenski programski jezik. Njegova priljubljenost izhaja iz tega, da imamo vedno večjo potrebo po internetnih aplikacijah, kjer je na odjemalčevi strani programska koda napisana izključno v javascriptu. Uporablja se predvsem za manipuliranje z DOM drevesom spletne strani, upravljanje z dogodki, ki se zgodijo na spletni strani, v zadnjem času pa tudi za asinhrono pogovarjanje s strežnikom

preko AJAX klicev. Vsak moderen spletni brskalnik vsebuje interpreter, ki omogoča poganjanje programov, napisanih v javascriptu. Poleg brskalnikov se vzpenja njegova uporaba tudi v drugih okoljih kot je strežniško programiranje preko projekta Node.js.

AJAX

V našem diplomskem delu smo AJAX uporabili za komuniciranje med odjemalcem in strežnikom. AJAX je tehnika, s katero se lahko odjemalec in strežnik pogovarjata brez osvežitve celotne spletne strani. Ta tehnika uporablja javascript, ki pošlje standardni HTTP zahtevek strežniku. Kratica pomeni asinhroni javascript in XML. Vendar ni nujno, da se zahtevki pošiljajo asinhrono. Format, ki nam ga vrne strežnik, prav tako ni nujno, da je XML. Velikokrat se uporablja JSON, lahko pa se uporablja tudi tekstovni format, saj dobljene podatke poljubno razčlenimo z javascriptom. Bistvo Ajax zahtev je v tem, da se za spremembo dela strani ne zahteva ponovno nalaganje celotne strani, ampak v ozadju neopazno javascript pošlje zahtevek ter dobi podatke od strežnika. Nato jih preko DOM manipulacije prikaže na strani.

jQuery

Javascript knjižnica jQuery je narejena z namenom normalizirati, popraviti ter poenostaviti ključne naloge pri programiranju v javascriptu. Najpomembnejša stvar, ki jo naredi jQuery (in podobne knjižnice) je abstrakcija interpreterja vseh spletnih brskalnikov. To pomeni, da izvajanje javascripta ni več odvisno od interpreterja posameznega spletnega brskalnika, ampak jQuery normalizira klice svojih funkcij preko vseh modernih brskalnikov. To je zelo pomembno predvsem zato, ker se nekateri brskalniki ne držijo ECMA (javascript) standardov. Vsebuje funkcije, ki močno olajšajo manipulacijo z DOM drevesom strani in izvajanje Ajax klicev. Poleg tega vsebuje uporabne funkcije, ki se uporabljajo vsakodnevno in jih ni v javascriptu (primer: .*each*in.trim). Zelo zgovoren je tudi podatek, da se uporablja v 55% od 10.000 najbolj obiskanih strani. [6]

JSON

Naša komunikacija med odjemalcem in strežnikom poteka z JSON formatom. Je predpisana struktura, ki se uporablja predvsem za prenos podatkov. Specifikacijo je napisal Douglas Crockford, ki je tudi eden od sodelujočih pri ECMA (javascript) standardu. Vpliv javascripta je tako velik, da je JSON sintaktično podmnožica javascripta. JSON lahko z javascript eval() funkcijo pretvorimo direktno v javascript objekt, kar pa zaradi varnosti ni priporočljivo. Sintaksa je bila določena tako, da je berljiva ter enostavna. Na spletu se pogosto uporablja pri AJAX zahtevkih.

XPATH

XPATH je jezik, s katerim pove odjemalec strežniku lokacijo podatkov. Je W3C standard, ki določa, kako se izbere določena vozlišča v XML datoteki. Uporablja se lahko tudi za poizvedovanje po XHTML datoteki, saj je XHTML dokument strukturiran kot XML dokument.

Regularni izrazi

Pri implementaciji se uporablja za zaznavo osnovnega tipa podatka. Regularni izrazi nam omogočajo, da ujemamo skupine teksta iz določenega večjega teksta. To nam omogoča, da razberemo določene dele formata teksta. Uporablja se tudi za sorodne namene, preverjanje, če je tekst določenega formata, in določanje možnega nabora črk. Implementirani so v vseh popularnih programskih jezikih.

Apache

Naša strežnikška aplikacija in nadzorna plošča tečeta na Apachiju. Apache je najbolj priljubljen spletni strežnik. Njegova naloga je, da streže HTTP zahtevke brskalnika. Tako apache preko PHP modula izvede PHP kodo, ki se nahaja v določeni datoteki. Poleg tako so ene izmed nalog Apachija nastavitve dostopa in razrešitev naslovov za določen URL.

Chrome

Je moderen spletni brskalnik, ki ga je razvil Google. Zasnovan je na odprtokodnem izrisovalniku Webkitu, ki skrbi za izračun položaja elementov. Omogoča pisanje vtičnikov, s katerimi si lahko izboljšamo funkcionalnost brskalnika ali spletne strani. Poleg tega ima serijsko vgrajena razvijalska orodja (dostopna s Control-shift-i), ki so nam močno pomagala pri razvoj vtičnika. Z razvijalskimi orodji lahko [15]:

- Pregledujemo HTML elemente in pripadajoč CSS. Prav tako lahko spreminjamo HTML in CSS na trenutni strani.
- Pregledujemo, katere vire je spletna stran naložila.
- Vsebuje razhroščevalnik (ang. debugger) za javascript. Lahko tudi spreminjamo javascript kodo direktno v orodjih.
- Analizo hitrosti spletne strani.
- Analizo hitrosti javascript kode (ang. profiler), ki nam pove, kje v javascriptu se brskalnik najdlje zadržuje. Na hitrosti pridobimo največ, če pohitrimo te dele.
- Priporoča, katere dele strani je treba izboljšati.
- Konzola. Z njo lahko izvajamo poljubno javascript kodo, ki se izvede na trenutni strani.

Handlebars.js

Javascript knjižnico smo uporabili pri programiranju odjemalca v javascriptu, ker nam omogoča HTML predloge (template). Naloga knižnice je, da skrbi za ločitev podatkov in oblike v javascriptu. Za maksimalno ločitev podatkov in oblike vsebuje določene dele programskega jezika, kot so pogojni stavki in zanke.

Skeleton CSS, Twitter bootstrap CSS

Knjižnici sta uporabni predvsem za izdelavo prototipov, saj vsebujeta osnovne CSS stile za HTML elemente. Poleg tega je pomembno tudi to, da vsebujeta normalizacijo CSS-ja za vse brskalnike. Twitter bootstrap CSS smo uporabili pri izdelavi primera. Skeleton CSS se uporablja za odjemalca in nadzorno ploščo.

SVN

Za upravljanje z verzijami smo uporabili SVN. SVN omogoča veliko večjo prilagodljivost, saj odpravlja veliko pomanjkljivosti direktnega urejanja datotek. Odpravi težave pri vzporednem spreminjanju datotek na projektu, saj se po potrditvi dokončanega dela primerja, da datoteke ni slučajno urejal še nekdo drug. Če je datoteko urejal še nekdo, ju program poskusi avtomatsko združiti. Če to ni mogoče, mora uporabnik ročno povedati programu, kako si sledijo deli dokumenta. Tako se izognemo prepisovanju podatkov. Prav tako zagotavlja obnovitev datotek na določeno verzijo in vsebuje zgodovino vseh spreminjanj datotek.

Sublime 2

Zelo prilagodljiv moderen urejevalnik teksta, ki smo ga uporabili za pisanje implementacije. Omogoča barvanje sintakse, avtomatsko dokončevanje, dokumentacijo za pogosto uporabljene funkcije, pomnenje svojih delov kode, ki so dostopni na določeni kombinaciji tipk (ang. snippet). Z vtičniki pa omogoča še upravljanje s SVN sistemom, generiranje komentarjev funkcij ter sinhronizacijo med brskalnikom in datoteko, kar odpravi ročno osveževanje strani v brskalniku.

Haidi SQL

Je program, ki omogoča upravljanje z Mysql bazo s pomočjo grafičnega vmesnika. Prav tako si lahko z njim naredimo sliko baze (ang. dump). To nam

pride prav pri obnavljanju stanja programa strežnika.

URI

URI je kazalec do dokumenta na spletu. Glede na namen jih ločimo na dva dela[2]:

- URN določa identiteto dokumenta.

- URL določa, kje se dokument nahaja.

DOM

Je konvencija za predstavitev HTML, XML podatkov. Z nevtralnostjo in ločitvijo vmesnika od določenega programskega jezika lahko s poljubnim programom dinamično dostopamo ter posodobimo vsebino, strukturo in stil dokumenta. [7] Ko brskalnik naloži stran, pretvori HTML tekst v drevo vozlišč. Drugi del je skupen vmesnik, s katerim se dostopa do drevesa. [3]

```
|-> Document
  |-> Element (<html>)
    |-> Element (<body>)
      |-> Element (<div>)
        |-> text node
        |-> Anchor
          |-> text node
      |-> Form
           |-> Text-box
           |-> Text Area
           |-> Radio Button
           |-> Check Box
           |-> Select
           |-> Button
```

Slika 3.6: Primer predstavitve DOM drevesa HTML dokumenta.

3.2.2 Arhitektura sistema

Za implementacijo sistema moramo napisati dva osnovna programa. Eden bo tekel kot vtičnik za brskalnik in bo namenjen označevanju trenutne spletne strani. Drugi bo tekel na strežniku in bo shranjeval ovojnice ter osveževal podatke. Poleg tega bomo implementirali še enostaven kontrolni program ter nadzorno ploščo za pregled in osveževanje podatkov ovojnice. Za demonstracijo implementacija vsebuje še program, ki izkorišča pridobljene podatke.

Odjemalec

Odjemalec je napisan kot vtičnik za brskalnik. Vtičnik obogati našo trenutno spletno stran. Sodobni vtičniki se pišejo z množico spletnih tehnologij: javascript, HTML in CSS. Javascript skrbi za logiko, HTML in CSS pa se uporabljata za dodatno obogatitev grafičnega vmesnika trenutne spletne strani.

Pri javascript vtičnikih moramo vedeti, da se stran ne sme osvežiti, drugače praviloma (če hranimo stanje v javascript spremenljivkah) izgubimo trenutno stanje aplikacije. HTML predpisuje strukturo dokumenta, redko pa se uporabi tudi za hrambo podatkov, medtem ko se CSS izkorišča za določanje grafičnih stilov HTML elementov. Dobra lastnost izkoriščanja teh tehnologij je, da dobimo prenosljiv program med sodobnimi brskalniki. Do tega pride, ker vsi vsebujejo javascript interpreter ter znajo izrisati (renderirati) HTML in CSS dokumente. To pa ne pomeni, da je koda avtomatsko prenosljiva, ampak da potrebuje minimalne popravke, saj se standardi in implementacija teh tehnologij v brskalnikih razlikujejo. Osredotočili smo se na vtičnik za brskalnik Chrome, saj vsebuje ogromno orodij za pomoč pri razvoju. Razvojno ime odjemalca je Scrappy in nesrečno sovpada s prej omenjenim testnim programom Scrapy[20].

Problemi prenosljivosti zaradi različnih implementacij tehnologij:

- Javascript
  Lahko privede do nedelovanja, če dva brskalnika različno implementirata javascript interpreter. Možnost napake minimiziramo z jQueryjem, ki je javascript knjižnica in poskrbi za abstrakcijo razlik različnih brskalnikov.

- HTML, CSS
  Verjetnost tega problema lahko zmanjšamo z upoštevanjem implementacije IE pogona in standardno HTML, CSS kodo, saj večina ostalih brskalnikov pravilno interpretira strani, napisane za IE. Ta problem ni tako kritičen, saj aplikacija še vedno deluje, vendar lahko povzroči vidno nedoslednost med brskalniki.

Strežnik

Strežnik lahko napišemo v katerem koli programskem jeziku. Implementacijo smo napisali s PHP. Za interpretiranje kode je na strežniku nameščen PHP interpreter, strežnik za HTTP zahteve je Apache, operacijski sistem pa

Ubuntu. Na strežniku tečeta tudi pomožni aplikaciji v obliki spletnih strani, program za nadzorno ploščo naše strežniške aplikacije ter demonstracijski program.

Komunikacija

Definirati moramo, na kakšen način se bosta odjemalec in strežnik pogovarjala. Seznam problemov pri komunikaciji smo določili na strani 24.

1. **Komunikacijski kanal**
   Ker je odjemalec napisan v javascriptu in se trenutna stran ne sme osveževati, morata komunicirati preko AJAX klicev. jQuery podpira funkcije, ki poskrbijo za abstrakcijo in poenostavi izvajanje teh klicev.

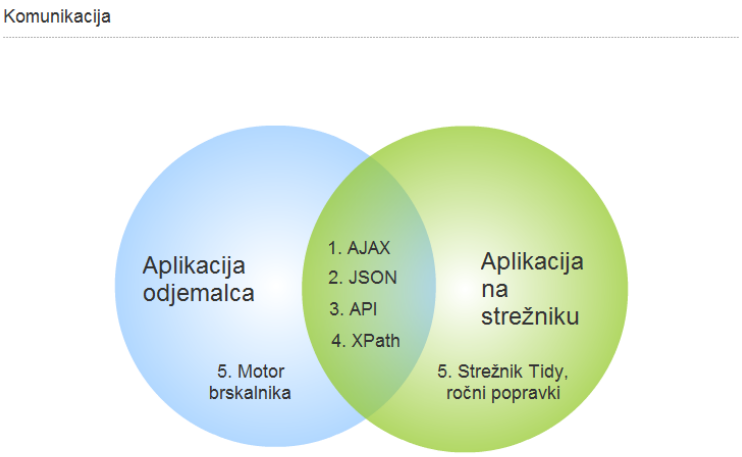
2. **Format sporočila**
   jQuery podpira mnogo dokumentov za komunikacijo z AJAX klici. Lahko je čisti tekst, JSON, JSONP, HTML dokument, XML. Izbrali smo JSON, saj je zelo enostaven in zadošča našim potrebam. JSONP se uporablja za komunikacijo med odjemalcem in strežnikom, ki se ne nahajata na isti domeni. Ta komunikacija je v brskalniku prepovedana. Ker ne izvajamo javascripta v kontekstu trenutne spletne strani, ampak vtičnika, se lahko s strežnikom pogovarjamo v JSON formatu.

3. **Vmesnik**
   Strežnik ima implementiran preprost usmerjevalnik z določenim vmesnikom in se bo odzival na izbranih URL naslovih. Za vsako akcijo moramo definirati format sporočila.

4. Položaj podatkov
   Ko strežnik dobi HTML dokument, mora poznati lokacijo podatkov. Za določanje položaja podatka v HTML dokumentu mora odjemalec strežniku poslati XPATH izraze. Privzeti javascript uporablja svoj opisnik lokacije v DOM drevesu brskalnika, vendar lahko to pot pretvorimo v XPATH izraz. S pretvorbo dobimo standardno obliko, skupni jezik, ki ga podpira tudi PHP.

5. **Interpretacija HTML dokumenta**

   Ker javascript teče v brskalniku, lahko prihaja do določenih razhajanj v interpretaciji HTML dokumenta v brskalniku in na strežniku. Brskalniki samodejno popravijo HTML na določenih delih ter ga nato izrišejo. To se dogaja ob nestandardno napisanem HTML dokumentu (ang. mallformed). V Chromu se tudi opazi, da samodejno doda v vsako tabelo element <tbody>. Problem lahko rešimo z implementacijo pogona brskalnika na strežniku. S to rešitvijo nastopi problem sinhronicije verzij med strežnikom im odjemalcem. Prav tako izgubimo na prenosljivosti programa v druge brskalnike. Druga rešitev je v programu Tidy. Tidy popravi HTML dokument v skladu s standardi, ki trenutno veljajo. Kliče se iz ukazne vrstice, prav tako pa obstajajo knjižnice za PHP. V implementaciji s klicanjem ukazne lupine in Tidy programa, nad trenutnim HTML dokumentom, popravimo napačno HTML strukturo. Za rešitev problema <tbody>smo vsakemu XPATH izrazu dodali možnost, da za <table>lahko obstaja <tbody>, lahko pa tudi ne. Možnosti za napačno interpretacijo XPATH izraza tukaj ni, saj se <tbody>vedno nahaja v enaki HTML obliki (<table><tbody>...</tbody</table>).



Slika 3.7: Komunikacija med odjemalcem in strežnikom

3.2.3 Implementacija odjemalca

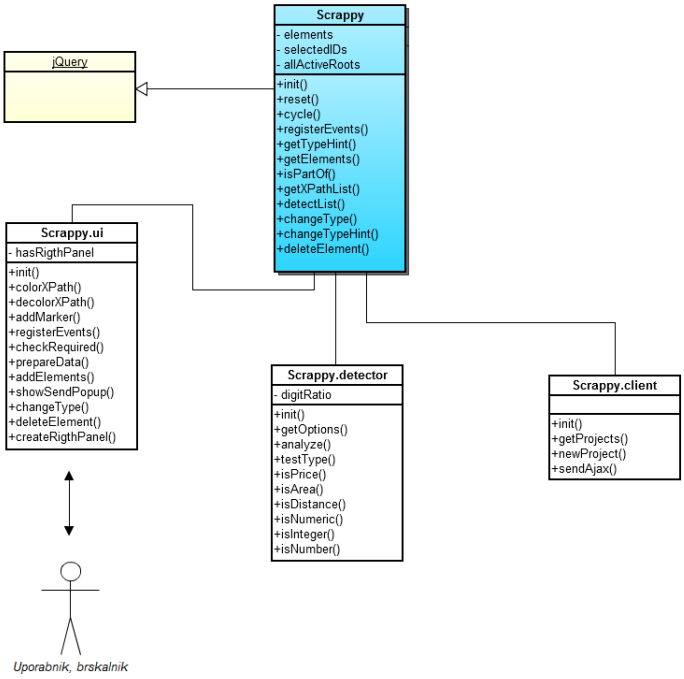
V splošnem smo že napisali nekaj o tehnologiji odjemalca. Odjemalec je napisan kot jQuery vtičnik (ang. plugin). To nam omogoča ponovno uporabljivost kode (ang. reusabillity). Ker nočemo, da je vtičnik vedno aktiven, ga lahko aktiviramo z uporabo tipke S. Pregled razdelitve na objekte:

1. Glavni del(scrappy.js)
   Vsebuje trenutno stanje programa, algoritme za iskanje entitet in združuje vse ostale dele sistema.

2. Grafični vmesnik(scrappy.ui.js)
   Vsebuje kodo za registracijo akcij ter grafični prikaz na zaslonu. Za ločitev strukture od podatkov uporablja predloge (ang. template) in javascript knjižnico Handlebars.js.

3. Objekt za komunikacijo s strežnikom(scrappy.client.js)

4. Zaznavanje osnovne semantike (scrappy.detector.js)
   Koda za določanje osnovne semantike na podlagi vnaprej določenih regularnih izrazov.



Slika 3.8: Struktura odjemalca

Osnovne možnosti

Slika 3.9: Prikaz zaslonske maske odjemalca.

1. Področje, kjer nam odjemalec prikazuje vse podatke ovojnice in omogoča izvajanje določenih akcij na označenih podatkih.

2. S premikanjem miške se nam z modro barvo označujejo spodaj ležeči HTML elementi spletne strani.

3. S klikom na HTML element se ta izbere z zeleno barvo in prestavi na desno platno. S pomočjo algoritma zaznamo, če je podatek del entitete na strani. Če je, se z zeleno barvo označijo vsi podatki istega entitetnega tipa.

4. Na podlagi regularnih izrazov se razbere osnovna semantika podatka. S klikom lahko spremenimo to lastnost podatka.

5. Odstranitev določenega podatka iz ovojnice.

6. S klikom povemo, da hočemo podatek dobiti samostojno in da ni del zaznanega entitetnega tipa.

7. S klikom se odpre okno za shranjevanje ovojnice na strežniku.

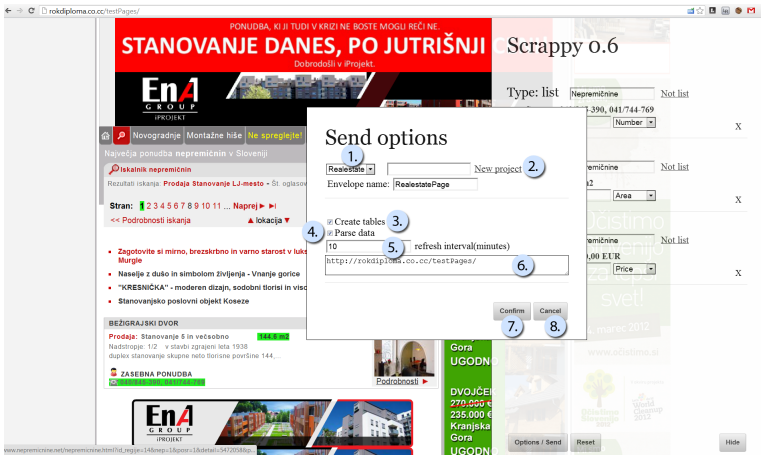
8. S klikom se program resetira. Trenutno implementirana samo enostavna osvežitev strani.

9. S klikom se zapre desno področje. Program še vedno teče. Tako lahko označimo podatke, ki jih desno področje zakriva.

10. Polje za vpis oznake, ki se bo uporabljala za določen podatek.

11. Če je algoritem zaznal nov entitetni tip, se pojavi polje za vpis imena entitetnega tipa. To polje se ob vpisu prekopira na vse podatke istega entitetnega tipa.

Pošiljanje ovojnice na strežnik



Slika 3.10: Zaslonska maska za pošiljanje podatkov na strežnik

1. Izberemo imenski prostor, projekt, pod katerim se bo nahajala ovojnica.

2. Če hočemo ustvariti nov projekt, vpišemo ime novega projekta in kliknemo "New Project".

3. Če označimo, se poleg shranjevanja ovojnice ustvarijo potrebne tabele za podatke.

4. Če označimo, se poleg shranjevanja ovojnice, in če so ustvarjene potrebne tabele, naredi prvo osveževanje podatkov.

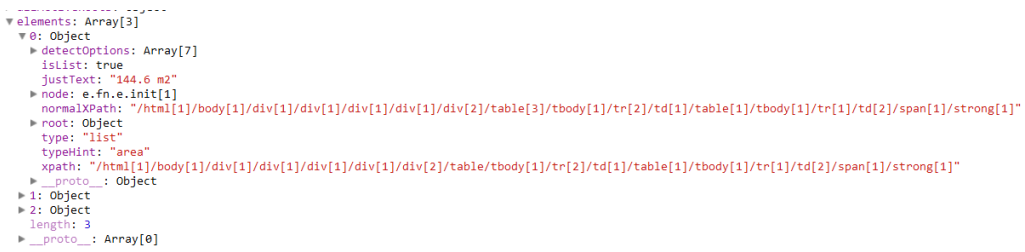
5. Vpišemo lahko interval avtomatskega osveževanja.

6. Trenutni spletni naslov, na katerega se nanaša ovojnica. Tukaj lahko spremenimo naslov preko predlog (primer na strani 47) ali dodamo poljubne nove.

7. S klikom pošljemo zahtevo na strežnik.

8. S klikom prekličemo pošiljanje ovojnice na strežnik.

Struktura stanja odjemalca, ovojnice



Slika 3.11: Slika strukture stanja odjemalca iz Chrome razvijalskih orodij

Kot vidimo na sliki 3.11 nam stanje predstavljata dva seznama. Seznam elementov ("elements") vsebuje vse označene elemente. Vsak element ima referenco na objekt korena seznama ("root"). Koreni seznama pa so skupaj povezani v svoj seznam. Obstaja še hash seznam šelectedIDs", kjer lahko hitro preverimo, če smo izbrani element že označili. Identifikator elementa je njegov XPath.

Detekcija semantike

Odjemalec skrbi za detekcijo tipa, saj lahko tako uporabnik še ročno popravi določen tip. Tukaj je potrebno predvsem paziti pri mešanici teksta in številk. Z uporabo samih regularnih izrazov bi tako v tekstu lahko razpoznali ceno, čeprav je drugotnega pomena. Tako moramo določiti neko statistiko, ki nam pove, kako bomo ločili števila od teksta. Ta statistika je:
Delež cifer = Število cifer/Dolžina teksta;
Pogoj: Dolžina teksta ne sme biti 0; če je 0, je delež cifer = 0
Pod cifre štejemo še ločili , in ., saj sta pogosto del števil.
Če je delež cifer >= 0,2, pomeni, da številke niso samo del teksta.

Primer:

1. Delež cifer: 1/5 = 0,200 = cena, "2 EUR"
2. Delež cifer: 1/6 = 0,167 = tekst, "2 EUR"
3. Delež cifer: 5/27 = 0,185 = tekst, "Povprečna cena je 30,5 EUR."
4. Delež cifer: 4/9 = 2,250 = tekst, "leta 1987"
5. Delež cifer: 3/3 = 1 = število, "0,3"

Za točko 2 bi lahko rekli, da ni tekst, temveč cena. Detektor vsakemu podatku odreže tudi prazen prostor (ang. trim), tako da bi bil v implementaciji primer enak Točki 1. Detekcija ni vedno zanesljiva, zato imamo možnost, da sami popravimo tip podatka. To je prikazano v točki 4 na strani 41. Implementacija upošteva pri detekciji samo izbrani podatek. Zanesljivost bi lahko še zvišali, če bi pogledali enake podatke enakih entitet.

3.2.4 Implementacija strežnika

Z .htaccess datoteko povemo HTTP strežniku Apache, da preusmeri vse prihajajoče zahteve na index.php. Tako lahko v index.php programsko implementiramo usmerjevalnik za prihajajoče zahteve.

Strežnik sestavlja 6 objektov:

- class.project.php
  Preprosta rešitev imenskih prostorov. Vsebuje ovojnice, ki jih potrebujemo pri določenem projektu. Vsebuje metodo za shranjevanje ovojnice.

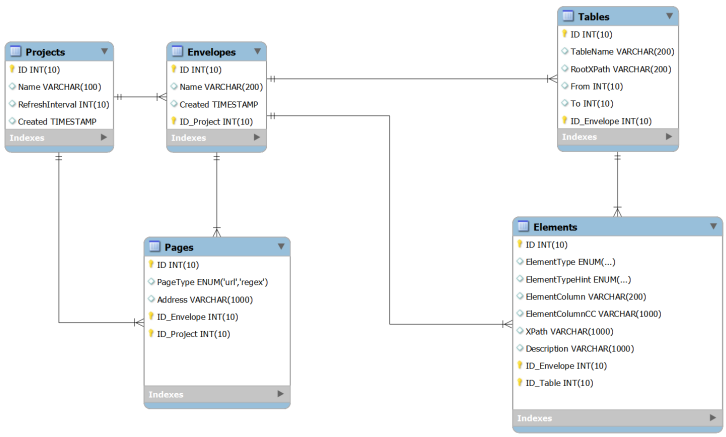
- class.envelope.php
  Vsebuje elemente ovojnice ter strani, na katere se ovojnica nanaša. Vsebuje metode za shranjevanje elementov, strani in tabel ovojnice. Hkrati vsebuje metode za osveževanje podatkov.

- class.page.php
  Naslovi, iz katerih naj ovojnica pridobi podatke.

- class.table.php
  Imena vseh tabel določene ovojnice in koren XPATH za določeno entiteto. Ker določeni tabeli pripadajo elementi, ima tabela metodo za kreiranje potrebne tabele za hranjenjenje podatkov.

- class.element.php
  Hrani pozicijo, XPATH podatka v HTML dokumentu. Vsak element pripada določeni ovojnici in tabeli.

- class.detector.php
  Statični razred, ki vsebuje metode za določanje osnovne semantike iz podatkov na podlagi regularnih izrazov. Regularni izrazi morajo biti sinhronizirani z javascript razredom, opisanim na strani 39.



Slika 3.12: ER diagram podatkovne baze.

Shranjevanje ovojnice

Za vsak označen entitetni tip se ustvari zapis v tabeli Table. Ta zapis pove, katero ime bo imela tabela, ki ga bo vsebovala, in koren XPath entitet v HTML dokumentu. Prav tako zapišemo vse strani, ki pripadajo določeni ovojnici in jo uvrstimo pod določen projekt. Shranimo tudi vse elemente, ki predstavljajo lokacije podatkov, ki nas zanimajo.

Osveževanje ovojnice

Program ima realizirane vse mehanizme osveževanja opisana na strani 26.

- **Ročno**
  V nadzorni plošči spletne aplikacije, opisane na strani 48.

- **Avtomatsko intervalno**
  Cron je program, s katerim lahko periodično izvajamo ukaze v komandni vrstici. PHP tako doda Cronu nalogo naj na določen časoven interval, preko vmesnika API na strežniku, pošlje zahtevo za osvežitev. Število minut lahko vpišemo v odjemalcu pri pošiljanju ovojnice, kot je opisano v točki 5 na strani 42.

- **Programsko**
  Enako kot stori Cron v prejšnji alineji, lahko program, preko vmesnika API na strežniku, pošlje zahtevo za osvežitev.

Ko sprožimo osveževanje ovojnice, strežnik prebere vse strani, ki pripadajo določeni ovojnici. Nato po potrebi sproži razširitev določenega naslova, ki pripada strani. Naslov je template za generiranje URL naslovov. Tako strežnik izvede preprosto nadomeščanje znakov in ustvari množico URL-jev, kot v primeru na strani 20. To nam omogoča pridobivanje podatkov s strani, ki uporabljajo enako predlogo. Generiramo lahko poljubna števila linearne funkcije v URL naslovu.

Formula: k*x+n x = (i...j) x bodo generirana števila od i do j
Primer:
Naslov[1]: http://www.nepremicnine.net/nepremicnine.html?offset=1*(1...3)+30

Generirani URL-ji:
http://www.nepremicnine.net/nepremicnine.html?offset=30
http://www.nepremicnine.net/nepremicnine.html?offset=60
http://www.nepremicnine.net/nepremicnine.html?offset=90

Z osnovnim URL-jem lahko dobimo podatke s trenutne strani, kjer je prikazano 30 entitet. Z generiranimi URL-ji, kjer se spreminja parameter offset, se sprehodimo po treh straneh in dobimo 90 entitet z isto ovojnico strani.

Če hočemo osvežiti stran z izbranim URL parametrom, lahko napišemo naslov oblike:
http://www.telefonskiimenik.si/ptis/Search.aspx?&q=:telephone

Pri klicu osveževanja moramo v URL ukaza prilepiti potreben parameter:
Za telefonsko število: 041/744-769 je naslov, ki sproži osveževanje podatkov:
http://rokdiploma.co.cc/project/
1/envelope/1/refreshJSON?telephone=%20041%2F744-769

[1]Odstranili, skrajšali celoten URL zaradi boljšega pregleda.

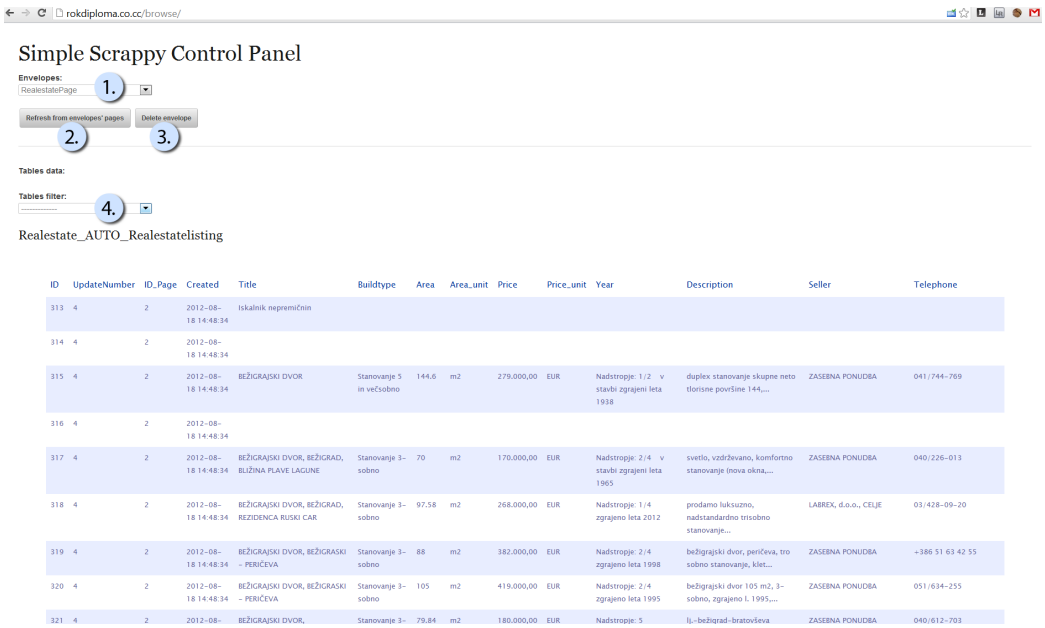
Seveda mora biti parameter URL zakodiran. Tako lahko s pomočjo ene ovojnice dobimo potrebne podatke, ki se generirajo iz enakih predlog, podobno kot v prejšnjem primeru z generiranjem števil.

Sledi pridobitev določenega HTML dokumenta z vsakega URL naslova s cUrl-om. To je program, ki prenese dokument na strežnik. HTML dokument shranimo na disk in nad njim sprožimo program Tidy. Ta popravi narobe strukturirane HTML dokumente (ang. mallformed). Korak je pomemben, ker zagotavlja enako interpretacijo dokumenta (razlaga problema na strani 37) na odjemalcu ter strežniku. V tej fazi imamo že določene tabele, v katere bomo zapisali podatke. Tabele vsebujejo XPath naslove korenov entitet. Algoritem te naslove uporabi in iz HTML dokumenta izlušči vsa vozlišča enakih entitet. XPath, ki je v objektu Element, pa nam pove, katere podatke rabimo iz vsakega vozlišča. Vsak podatek se z regularnim izrazom, po potrebi, razbije na več delov na podlagi tega, kateri tip je določil odjemalec. Vsak podatek ima svoj stolpec, v tabeli, kamor se zapisujejo podatki. Celoten postopek ponovimo za vse entitetne tipe na strani.

3.2.5 Nadzorna plošča

Trenutne naloge spletne aplikacije, ki nadzoruje strežnik:

- Pregled podatkov ovojnic
- Izbris ovojnice
- Ročna osvežitev podatkov ovojnice
  Implementira uporabniški vmesnik, ki je ena izmed treh mehanizmov za osveževanje podatkov, omenjenih na strani 26.



Slika 3.13: Slika zaslonske maske nadzorne plošče.

1. Izberi ovojnico

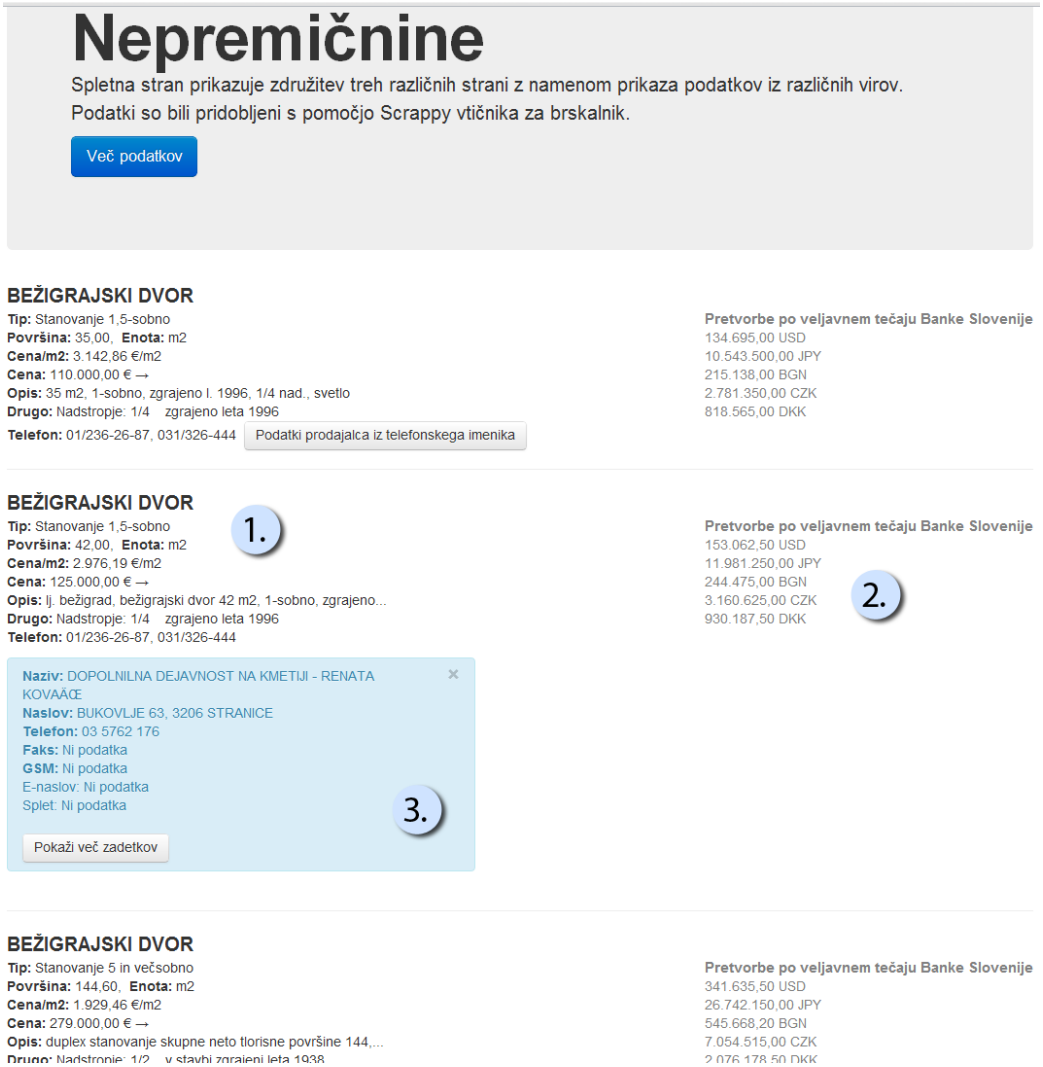
2. Pošlji zahtevo za ponovno osveževanje podatkov

3. Izbriši trenutno izbrano ovojnico

4. Izberi tabelo izbrane ovojnice

3.2.6 Testni primer

Za testni primer smo obogatili spletno stran z nepremičninami. Pri tem smo si pomagali z našo implementacijo programa za izdelavo ovojnic.



Slika 3.14: Slika zaslonske maske končnega rezultata testnega primera.

Naredili smo tri ovojnice:

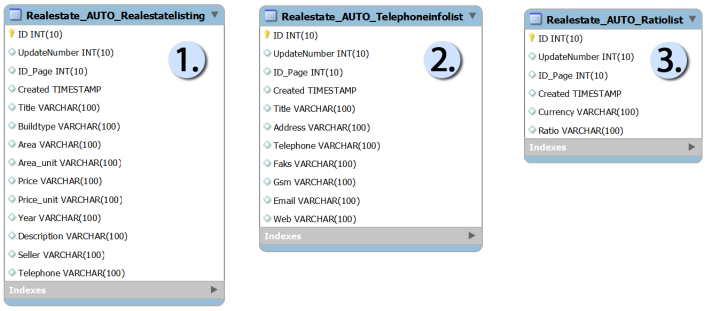
1. Stran z nepremičninami [16]
   Iz seznama smo pobrali oglase z nepremičninami.

2. Stran banke slovenije [17]
   Iz tabele smo pobrali trenutne tečaje svetovnih valut. S tem bomo lahko avtomatsko pretvorili ceno nepremičnin v poljubno valuto. Tako

lahko stran naredimo bolj privlačno za tujce, saj lahko pridobijo cene v svojih valutah.

3. Telefonski imenik [18]
   Na tej strani smo s pomočjo ovojnice iskali podatke preko telefonske številke, ki je napisana kot kontaktna številka pri nepremičninah. Tukaj lahko dobimo in preverimo dodatne podatke prodajalca nepremičnine.

V odjemalcu obkljukamo možnost, da aplikacija ustvari vse potrebne tabele za shranjevanje. To je prikazano na zaslonski maski v točki 3 na strani 42. Zahtevek se pošlje na strežnik, kjer se ustvarijo potrebne tabele.



Slika 3.15: ER slika tabel, ki se ustvarijo na strežniku za testni primer

Osveževanje podatkov telefonskega imenika se proži programsko. Tako nam ni potrebno osveževati vseh podatkov za vsako nepremičnino, kar lahko privede do velikega števila HTTP zahtevkov. Seveda bi bilo dobro, da si podatek, ko ga enkrat imamo, zapomnimo za naslednje poizvedbe in mu določimo nek rok uporabnosti.

Pridobitev podatkov telefonskega imenika se izvaja s pomočjo predlog naslova, kot v primeru na strani 47. Spletna stran se kliče z drugačnim URL parametrom, tako dobimo različne podatke s pomočjo ene ovojnice strani.

Poglavje 4

Razprava o predlagani implementaciji

Najbolj rizičen del aplikacije je problem komunikacije, omenjen v točki 5 na strani 3.1.6, ki nastane zaradi možnosti različne interpretacije HTML strukture dokumenta na strežniku in odjemalcu. Če pridobimo čisti tekst s spletnega mesta, bo drugačen od interpretacije brskalnika. Tako je lahko XPATH, ki ga označimo v brskalniku, različen od XPATH-a v čistem tekstu(neinterpretiranem). Rešitev bi bila, da bi v strežnik vgradili pogon brskalnika. S tem bi izgubili možnost prenosa vtičnika v drug brskalnik, hkrati pa bi morali biti verziji pogona brskalnika sinhronizirani. Prvi problem bi lahko rešili z vgradnjo več pogonov različnih brskalnikov v strežnik. Rešitev, ki smo jo izbrali in ki močno popravi ta problem, je uporaba programa Tidy. S pomočjo Tidy-a in dodajanjem <tbody>za <table>, je problem praktično izginil. Seveda ne moremo točno vedeti, kako se bosta brskalnik in strežnik obnašala za vsako spletno stran.

Ta implementacija brskalnik–strežnik ima zgoraj opisani bistveni problem napram ostalim programom, opisanim v poglavju 2.3. Ima pa ravno iz enakih razlogov bistveno prednost. Zaradi svoje arhitekture tipa brskalnik–strežnik je zelo enostavna za uporabo.

4.1 Izboljšave

Možnost luščenja podatkov bi povečali, če bi dodali podporo za seje. Večina spletnih strani dandanes za sejo uporablja piškotke. Tako bi strežnik v svojo HTTP zahtevo lahko dodal piškotek, ki se uporablja za prijavo na določeno stran. Seveda obstajajo tudi drugi načini vzdrževanja seje. Tako bi trenutno delovale strani, ki vodijo sejo v URL naslovu.

Poleg parametrov v URL naslovu, primer na strani 47, bi koristilo, če bi lahko izvedli HTTP POST zahtevek na določen URL naslov. S tem bi lahko zajeli podatke množice strani na spletu, ki se generirajo na podlagi teh zahtevkov.

Ker je bil implementiran prototip programa, varnost ni bila prva skrb. Ima pa program implementirano HTTP avtentikacijo. Vse poizvedbe se izvajajo preko PHP PDO knjižnice s parametriziranimi poizvedbami, ki poskrbijo, da ne more priti do SQL vrivanja. Prav tako lahko neka druga stran vrine HTML, javascript, CSS kodo v našo stran, ki se napaja z njenimi podatki. Tako moramo pri izdelavi ali shranjevanju spletne strani paziti oziroma uiti (ang. escape) določenim znakom.

Na višjem nivoju bi bilo potrebno razdeliti bazo na uporabnike, saj je namen vtičnika, da ga v svojem brskalniku uporablja več različnih uporabnikov. Vsak uporabnik bi bil povezan s svojimi projekti. Lahko bi obstajala možnost uporabe ovojnice drugega uporabnika.

Naslednja logična izboljšava kode bi bila: [4]

- Testi programa(ang. Unit testing)
  Napišejo se testi programske kode, ki se avtomatsko izvajajo po vsaki dopolnitvi programa. Tako lažje zaznamo, če je zaradi spremembe kode nastala kakšna napaka v delovanju in zaradi tega program ni več v skladu s specifikacijo programa.

- Izboljšava hitrosti(ang. Performance testing)
  Pohitrimo tiste dele programa, kjer se porabi največ procesorskega časa.

Poglavje 5

Zaključek

V okviru diplomskega dela smo se ukvarjali s problemom pridobivanja podatkov iz določene spletne strani, na način, da ročno označimo, kje na strani se nahajajo. Z razmislekom v poglavju 2 smo prišli do arhitekture tipa brskalnik–strežnik, ki rešuje naš problem. Prednost tega je, da označimo lokacijo podatkov v našem brskalniku preko vtičnika. Tako postane označevanje enostavno, saj smo vajeni svojega brskalnika, hkrati pa se ovojnice in podatki nahajajo na oddaljenem strežniku.

Našli smo tudi pomanjkljivosti takšne arhitekture. Zaradi nastale brskalnik–strežnik arhitekture pride do komunikacijskih problemov. Obstaja možnost, da strežnik in brskalnik drugače interpretirata spletno stran. Pri iskanju rešitev smo se odločili za program Tidy in popravek HTML strukture dobljenega dokumenta strežnika, kar smo opisali v podpoglavju 3.2.2 v točki 5. Tako smo močno zmanjšali možnost napake, saj je v testnem okolju nismo več zaznali.

V okviru diplomskega dela smo razvili prototip programa na strani odjemalca in strežnika. Za dokaz uporabnosti našega novega orodja smo implementirali testni primer. Naredili smo novo spletno stran z nepremičninami, ki smo jo obogatili z dodatnimi podatki. Dodali smo ji pretvorbo cene s tečajem, s strani Banke Slovenije, ter dodali prikaz podatkov s strani poslovnega telefonskega imenika. Tako smo pokazali, kako lahko z razvito aplika-

cijo poenostavimo zajem podatkov s spletne strani, hkrati pa s kombinacijo različnih strani nadgradimo uporabniško izkušnjo.

Literatura

[1] Tim Berners-Lee's proposal, http://info.cern.ch/Proposal.html

[2] D. Fensel, F. M. Facca, E. Simperl, I. Toma, Semantic Web Services, str. 91.

[3] A. Burgess, Getting good with javascript str. 115–140, 2011.

[4] A. Burgess, Getting good with javascript str. 97–108, 2011.

[5] J. Way, Decoding HTML5 str. 35–62, 2012.

[6] jQuery Usage Statistics, 12. 1. 2012, Dostopno na: http://trends.builtwith.com/javascript/JQuery

[7] Document Object Model, "The Document Object Model is a platform- and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents.", Dostopno na: http://www.w3.org/DOM/#what

[8] T. Berners-Lee, J. Hendler, O. Lassila, The Semantic Web, Dostopno na: http://www.scientificamerican.com/article.cfm?id=the-semantic-web

[9] N. Shadbolt, W. Hall, T. Berners-Lee, The Semantic Web Revisited, http://eprints.soton.ac.uk/262614/1/Semantic_Web_Revisted.pdf, "This simple idea ... remains largely unrealized."

[10] T. R. Gruber, (June 1993)., "A translation approach to portable ontology specifications", Dostopno na: http://tomgruber.org/writing/ontolingua-kaj-1993.pdf

[11] A. Gulli, A. Signorini, The Indexable Web is more than 11.5 billion pages, Dostopno na: http://homepage.cs.uiowa.edu/ asignori/papers/the-indexable-web-is-more-than-11.5-billion-pages/

[12] J. Nielsen, Sitemap Usability, 2008, Dostopno na: http://www.useit.com/alertbox/sitemaps.html

[13] Search Engine Optimization Starter Guide, Dostopno na: http://static.googleusercontent.com/external_content/untrusted_dlcp/www.google.com/sl//webmasters/docs/search-engine-optimization-starter-guide.pdf

[14] Rich snippets (microdata, microformats, and RDFa) http://support.google.com/webmasters/bin/answer.py?hl=en&answer=99170&topic=21997&ctx=topic

[15] Chrome Developer Tools: Overview, Dostopno na: https://developers.google.com/chrome-developer-tools/docs/overview

[16] Nepremičnine.net, Dostopno na: http://www.nepremicnine.net/

[17] Banka Slovenije, Dnevna tečajnica – referenčni tečaji ECB, Dostopno na: http://www.bsi.si/podatki/tec-bs.asp

[18] Poslovni telefonski imenik Slovenije, Dostopno na: http://www.telefonskiimenik.si/ptis.aspx

[19] Mozenda, Dostopno na: http://www.mozenda.com/

[20] Scraper, Dostopno na: https://chrome.google.com/webstore/detail/mbigbapnjcgaffohmbkdlecaccepngjd

[21] Python Scrapy Tutorial, Dostopno na: http://doc.scrapy.org/en/latest/intro/tutorial.html

[22] Screen-Scraper, Dostopno na: http://www.screen-scraper.com/

Slike