SEFRA - Web-based Framework Customizable for Serbian Language Search Applications

Mioljub Jovanović[*], Goran Šimić[**], Milan Čabarkapa[*], Dragan Ranđelović[***], Vojkan Nikolić[****], Slobodan Nedeljković[****], Petar Čisar[***]

[*]Department for Postgraduate Studies, Singidunum University, Danielova 32, 11000 Belgrade, Serbia, mioljub.jovanovic.12@singimail.rs, mcabarkapa@singidunum.ac.rs

[**]Research Centre for Simulations, University of Defense, Generala Pavla Jurišića Šturma 33, Banjica, 11000 Belgrade, Serbia, goran.simic@va.mod.gov.rs

[***]Department for Informatics and Computing, Criminalistics and Police University, Cara Dušana 196, 11070 Belgrade, Serbia, dragan.randjelovic@kpa.edu.rs, petar.cisar@kpa.edu.rs

[****]Ministry of Interior of the Republic of Serbia, Kneza Miloša 101, 11000 Belgrade, Serbia, vojkan.nikolic@mup.gov.rs, slobodan.nedeljkovic@mup.gov.rs

Abstract: This paper presents SEFRA – a web-based framework for searching Web content written in Serbian. SEFRA is an easily customizable hybrid solution that can be a platform for new search applications and/or a service for already existing ones. The proposed architecture solves the problems of indexing, searching and displaying search results adjusted for Serbian. It unifies several web technologies and services into one product suitable for use in the Western Balkan's countries for helping e-Government citizens' services and other public-sector services, private company administration, solving specific search problems for academic institutions and scientific literature publishers, etc. The proposed solution uses advanced Serbian language services accessible over the Web. It is also implementable for any other language where the target language morphology service exists. In other words, architecture is also customizable in this direction. It should be noted that the proposed architecture is optimized from both backend and web front-end perspective. The source code can be pulled from https://bitbucket.org/mjovanov/pretraga/. The one application of the proposed architecture is experimentally demonstrated through the search of crime law documents of Serbia. The experimental usage of this implementation shows that the problem of search relevance, is well-solved and easily customizable.

Keywords: web-based architecture; Serbian language text search; software implementation; search results

1 Introduction

An accelerated development of the Internet as a platform and WWW (Web) as the most frequently used service of the Internet, brought access to a huge number of documents on the global network. Moreover, the documents' content is distributed in the same way. The page can also consist of fragments that originate from different hosts. Considering such a complex situation, advanced search application developers face many challenges, such as: collecting all available pages, analyzing the content of the collected material and enabling a quick query, as well as, display relevant documents based on the specified search criteria.

Since it is a fact that English is the most commonly used language on the Web [1], there are many representative search applications and services specialized for English content (e.g. Google, Bing…). In other words, we are in a position where the problem of collecting, analyzing documents, and finding the search results is solved for the English language. Since there are significant differences between the languages of the Western Balkans and the English language, then the question arises – if the data search problem is thoroughly resolved in English, is the problem of indexing and searching documents in our local languages also solved?

This paper suggests the possible approach, through the example of the realization of modern web architecture, to provide the answer to the above question of indexing, searching and displaying the results adjusted for the Serbian language. This work certainly would not be possible without going through various research documents and reports which are mentioned in Section 2. The architecture of the proposed solution along with its used components are discussed in Section 3, while implementation details and most code excerpts are covered in Section 4 of the document. Evaluation of results is given in Section 5, followed by conclusions and potential future work.

2 Related Works

Nikolic et al. [2] presented one e-Government services to get quick responses. This service enables citizens to receive answers, in the form of documents in the Serbian language, at any time and in any place, to the questions in the criminal law domain. This service has developed a Question and Answer (Q&A) system, based on Bag of Words (BoW) and Bag of Concepts (BoC), for categorizing text and incorporating background knowledge. The automatic mapping of relevant documents stands out as an important application for automatic question-documents classification strategies. This research presents a contribution to the identification concepts in text comprehension in unstructured documents as a significant step towards clarifying the role of explicit concepts in retrieval information in general. These authors introduce a new approach to create concept-based text representations and apply it to a text categorization collection in order to create predefined classes in the case of a short document analysis document. In the revolutions of this Q&A system, is a classification-based algorithm for a question matching topic model. The results obtained proved to be satisfactory based on the "golden rule".

In article Martinovic et al. [3] is presented an information retrieval system for Serbian language. Approaches designed and adopted to handle them are depicted and illuminated in this article. As a backbone of this system, they used a SMART retrieval system which they augmented with features necessary to deal with the specifics of the Serbian alphabet. Serbian language is a morphologically rich language that leads to specific implications of the text prefix. During the development a SMART retrieval system, the authors developed two algorithms which increased retrieval precision by 14% and 27%, respectively. Complete testing was conducted using two gigabyte EBART collection of Serbian newspaper articles.

Considering the existing solutions that depend on e-Government requirements, Šimić et al. in [4] proposed focusing on testing in different conditions and improving the ability of adaptation in the next research phases. One of the objectives pursued in this work is to find solutions for the functioning of such a system in multilingual environments and increasing content complexity concerning grammar and dictionaries of different languages, regardless of the area of use.

Kolomiyets et al. [5] represent the Question Answering method as a comprehensive approach that provides a qualitative way for information retrieval. This approach is a system of queries and documents in relation to the possible functions of search to find an answer. This research discusses general questions contained in a complex architecture with increasing complexity and the level of frequency of questions and information objects. These authors represent here a method of how natural language roots are reduced on keyword for search, while knowledge databases, and resources, obtained from natural language questions and answers, are made intelligible.

In addition, there are now research efforts where the authors try to solve a specific language searching problem [6] - [13], but there is no complete software architecture easily customizable for different search applications. In [6] author’s give one optimization of the method proposed in [2] where selection of the similarity measure is performed using the principles of redundancy and fault tolerance, in [7] is described one search engine using MySQL as one of cheap option, work [8] presents one architecture which uses different semantic web technologies and builds one prototype of semantic web mashup possibility, paper [9] proposes one novel Italian Sign Language Multi Word Net using process of integration the Multi Word Net lexical database and the Italian Sign Language, paper [10] describes a novel LInSTSS approach which is suitable for using to create a software tool which is capable to determine the semantic similarity of two presented no large texts, in paper [11], authors propose the use of smoothed n-gram language models to classify tweets as a typical short texts from Twitter in both Portuguese languages - Brazilian and European variants, paper [12] deals with the software architecture which establishing electronic services for searching and presentation in an information system on scientific activities of the Ministry of Education, Science and Technological Development of the Republic of Serbia and work [13] has objective to give a lexicon based algorithm which is able to perform different natural language identification using minimal training data in the obligatory process of machine learning because this step is often the first step in many natural language processing tasks which is normally necessary to make in the shortest possible time. Therefore, we have a strong motive for designing the SEFRA framework – hybrid solution based on existing Web services and technologies (framework source code is available at: https://bitbucket.org/mjovanov/pretraga/). Additionally, there is a search application developed for demonstration and testing SEFRA (the implementation available online: http://88.99.175.85/pretraga/).

3 Proposed Web Architecture

According to previous researches and already existing implementations [14] [15], there are four processes necessary to obtain relevant search results (Figure 1). During processing, targeted content passes through the two stages: collecting, preparing and indexing belong to the preparation stage while query processing, searching and presenting belongs to the production stage.

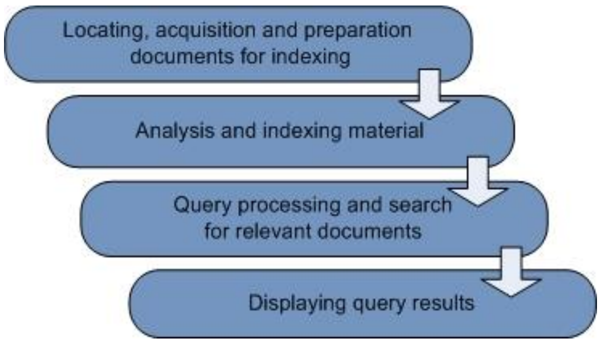

Locating, acquisition and preparation documents for indexing
Analysis and indexing material
Query processing and search for relevant documents
Displaying query results

Figure 1

Four processes necessary for Web content searching

Additionally, entering the production stage, the system must run these four processes simultaneously. This is a consequence of permanent changes of the content. For instance, constantly adding, removing, updating and replacing of documents and their references (URI) is a common case on the global network. Such challenges as well as a complex nature of Serbian language directed SEFRA design (Figure 2) to be modular solution based on open components.

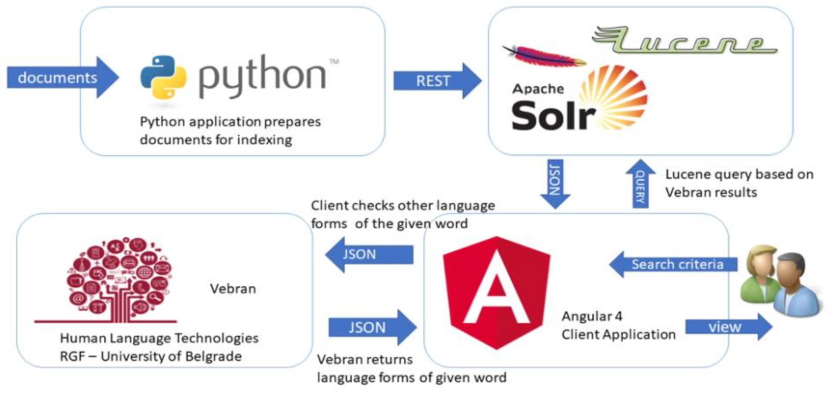

documents
python
Python application prepares documents for indexing
REST
Lucene
Apache Solr
JSON
QUERY
Lucene query based on Vebran results
Client checks other language forms of the given word
JSON
Vebran
Human Language Technologies RGF – University of Belgrade
Search criteria
Angular 4 Client Application
JSON
view
Vebran returns language forms of given word

Figure 2
SEFRA architecture - modular solution based on open components

The developed solution is an open and modular framework which consists of four components. For preparation stage, SEFRA uses SolrClient – Python library (solrclient.readthedocs.io) for locating, collecting, and preparing documents. Further documents' analyzing and indexing SEFRA performs by using Apache's Solr and Lucene libraries (lucene.apache.org/solr/).

Solr is a popular fast – search platform based on Lucene technology. Both are developed under Apache Foundation [16] as platforms for full text search written in the Java programming language. Lucene [19] represents a framework with high – performances in full-text search. Designed as a system centered solution, Lucene API is complex for implementing special requests and search customization. For this reason, Solr is a solution dedicated to enable simpler interface and better customization abilities for Lucene with resources accessible locally as well as remotely (through the REST API).

For production stage, SEFRA uses a reach Web client application based on Angular 4 (angular.io) [18] and Bootstrap 4 (v4-alpha.getbootstrap.com) [19] libraries. SEFRA client communicates with the end users as well as with the external services. In concrete scenario, SEFRA uses the Vebran – Serbian language service (hlt.rgf.bg.ac.rs/VeBran) to obtain lemmatization of query (transforming its words into normal form). Then, it sends a prepared query, as a

REST request to the Solr search service. After receiving search results, the SEFRA Client prepares the representation for delivering to the end user.

4 Implementation

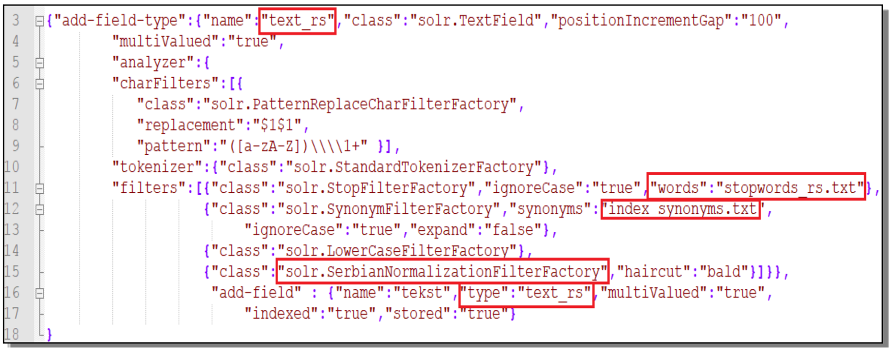
4.1 Preparation Stage Processes

As mentioned, SEFRA uses *SolrClient* – Python library for locating, collecting, and preparing documents. *Python* is a modern, easy-to-use programming language, which contains many libraries useful for acquiring documents, analyzing them and creating fields and schemas for indexing as well. In SEFRA the *SolrClient* represents a module which works together with Solr server acting as an interface between Solr and rest of the system as well as with the outer world. In other words, SEFRA uses *SolrClient* instance to retrieve local or remote documents and to prepare them for later indexing and searching. It leverages the potential of the Solr server quickly and easily from the *Python* environment, facilitating the use of the REST interface of the *Solr* platform.

For analyzing of acquired documents *Solr* server by default tries to find out fields and their types based on the content. This approach is interesting when objective is to index text in English as *Solr* uses built in functions (and thesaurus) designed for English language in this process. This automatized indexing unfortunately produces unexpected results for content written in other languages (e.g. Serbian). In this case, Solr recognizes the sentences in the Serbian language in the wrong way – meaningless strings often considered as a single string instead at least, a list of words. Therefore, adding new pre-defined fields is necessary for making improvements of indexing process.

SolrClient allows adding of new fields or metadata to a document, specifying the type of field(s) and schema on the server, preparing it for further document processing. There are several *SolrClient* functions for this purpose. In the concrete scenario, *solr.schema.create_field* and *solr.index* methods are used to add desired fields in the scheme and to add indexes to documents**.** To enable proper processing of words and sentences in Serbian, the default Solr behavior has been adapted to manage specific language requirements such as grammatical cases, synonyms, stop words and processing of diacritics.

There is a REST service designed for this purpose. It is accessible over *Solr*'s URI *solr/sd/schema*. It enables remote setup of *Solr* server by using simple JSON formatted messages (Figure 3) sent as a HTTP request (*application/json* type).



```
{"add-field-type":{"name":"text_rs","class":"solr.TextField","positionIncrementGap":"100",
        "multiValued":"true",
        "analyzer":{
        "charFilters":[{
           "class":"solr.PatternReplaceCharFilterFactory",
           "replacement":"$1$1",
           "pattern":"([a-zA-Z])\\\\1+" }],
        "tokenizer":{"class":"solr.StandardTokenizerFactory"},
        "filters":[{"class":"solr.StopFilterFactory","ignoreCase":"true","words":"stopwords_rs.txt"},
                   {"class":"solr.SynonymFilterFactory","synonyms":"index_synonyms.txt",
                        "ignoreCase":"true","expand":"false"},
                   {"class":"solr.LowerCaseFilterFactory"},
                   {"class":"solr.SerbianNormalizationFilterFactory","haircut":"bald"}]}},
                    "add-field" : {"name":"tekst","type":"text_rs","multiValued":"true",
                        "indexed":"true","stored":"true"}
}
```

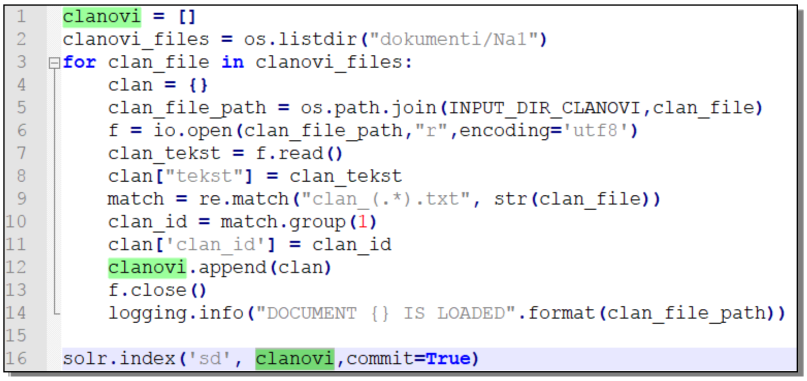
Figure 3

Solr server setup for indexing documents written in Serbian

The above figure shows the complete process of creating a *text_rs* field type. This field is based on similar solutions for other languages that are supported in *Solr*, and it has been defined including new entities into *Solr* core: stop-words filter (*stopwords_rs.txt*), filter for synonyms (*index_sinonimus.txt*), filter for upper / lower cases (already built-in resource) and filter for the processing of diacritical signs (*SerbianNormalizationFilterFactory*).

SerbianNormalizationFilter is Java based Lucene library which is implemented and leveraged for use in Solr custom field “text_rs”, as depicted on Figure 3. Since Serbian language uses Cyrillic alphabet as well as Latin it’s possible to perform several language-specific steps in order to render both indexing and querying process more effective. Example implementation in SEFRA uses haircut=”bald” which removes all diacritics from letters such as 'č', 'ć', 'š', 'ž' or 'đ' from the search text which reduces the precision of the query results. However, since Vebran is invoked as a backend service before Solr query, text in query field entered with diacritics will be correctly returned in all shapes and forms. Finally, these settings are stored in *Solr* server configuration file ready for use.

After creating fields and setup *Solr*, the documents are ready to be loaded into index. Next Python code demonstrates this process (Figure 4). In this example, the origin of documents to be indexed is in *dokumenti/Na1* folder. The system opens each file to read its content and put it in the previously created field named *tekst* for further analysis. After the collection is completed, document is sent to a server by forwarding it as a parameter of *SolrClient* (*solr*) *index()* method. Example depicted on Figure 4 demonstrates use of Python code to enrich index by additional fields which may be relevant for such – such as “clan_id”, which is the exact article id referenced in the original text search corpus. Python code provides capability to easily extend given code to any number of additional parameters or fields which may be required to properly index the text.



```
clanovi = []
clanovi_files = os.listdir("dokumenti/Na1")
for clan_file in clanovi_files:
    clan = {}
    clan_file_path = os.path.join(INPUT_DIR_CLANOVI,clan_file)
    f = io.open(clan_file_path,"r",encoding='utf8')
    clan_tekst = f.read()
    clan["tekst"] = clan_tekst
    match = re.match("clan_(.*).txt", str(clan_file))
    clan_id = match.group(1)
    clan['clan_id'] = clan_id
    clanovi.append(clan)
    f.close()
    logging.info("DOCUMENT {} IS LOADED".format(clan_file_path))

solr.index('sd', clanovi,commit=True)
```

Figure 4

Preparing documents for indexing

4.2. Production Stage Processes

SEFRA performs production stage processes in both of entities – *Solr* server and *Angular* clients. The distribution of functionalities reduces stress to the server side by using powerful client technology. The main reasons in favor of using the Angular 4 as a client platform [20] are:

- Separating of the user interface from the business logic
- Modularity (Enabling flexible design of low coupled forms and logic)
- Asynchronous calls support (Easy to code client-side multithreading)
- Rich user interface support (forms based on templates as well as program-generated forms)
- Reusable and responsive components (support for Angular Material and Bootstrap 4)

The list of features and advantages embedded in *Angular 4* is a quite long. TypeScript is language of choice in our implementation. There are several reasons for choosing TypeScript: modular development support (object oriented language semantic), easy compiling (*transpiling*) into *JavaScript* code and compilation time error detection, etc. Moreover, *Angular* is written in *TypeScript*, which is obviously a huge advantage if one develops application in the same programming language as the platform itself. Typescript is a super-set of JavaScript that will be transpiled into pure JavaScript code [21].

The core production – stage functionality is to process searching queries, prompted by the users, based on existing services specialized for Serbian language. This client-side module provides integration of *Solr* indexing &

searching services with *Vebran – Delafs* services (Vocabulary of word forms with all their morphological properties) [3]. Vebran is Serbian linguistic web-based service offering different linguistic capabilities for semantic and morphologic extension of the given phrase. Implementation of Verbran service invocation is explained in page 69 and Figure 4. As an example, for input search text "delo", getDelafs method will invoke Vebran API "/Vebran/api/delafs/delo" which returns all other morphological shapes of the given input text, which is:

```
<string xmlns="http://schemas.microsoft.com/2003/10/Serialization/">
dela;delima;delo;delom;delu;дела;делима;дело;делом;делу</string>
```

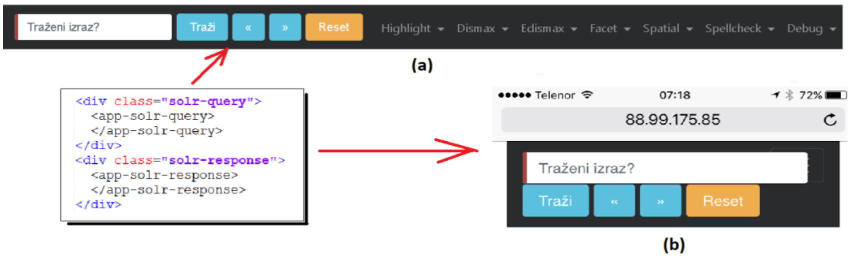
With Verbran it is also possible to leverage other linguistic services, such as synonyms using API "Vebran/api/sinonimi/", which for given term "delo" also gives it is synonyms:

```
<string xmlns="http://schemas.microsoft.com/2003/10/Serialization/">
delo;opus;podvig;rad;duhovnatvorevina;дело;опус;подвиг;рад;духовна
творевина</string>
```

On one hand, with the Web services used on the server side, SEFRA client communicates asynchronously. On the other hand, there has to be synchronization between Solr and *Vebran* services during this process. Therefore, SEFRA uses *RxJS Observable Library* (*Observer* design pattern) [22] [23] as appropriate one for handling multiple service requests and responses simultaneously. This way it becomes possible to extend the number of services used. Enabling full control of concurrent execution, Observable also simplifies the termination of running asynchronous calls if timeout is expired. In SEFRA example implementation Vebran Delafs API has been leveraged, yet all other services can be easily involved and processed using Observable design pattern as described in the remainder of the paper.

4.3. User Interface

SEFRA uses *Bootstrap Navbar* component (getbootstrap.com) for creating end – user interface. It enables responding to the device size (Figure 5), easy navigation and intuitive interface. There is one menu bar with drop down menus, modelled through the appropriate classes.

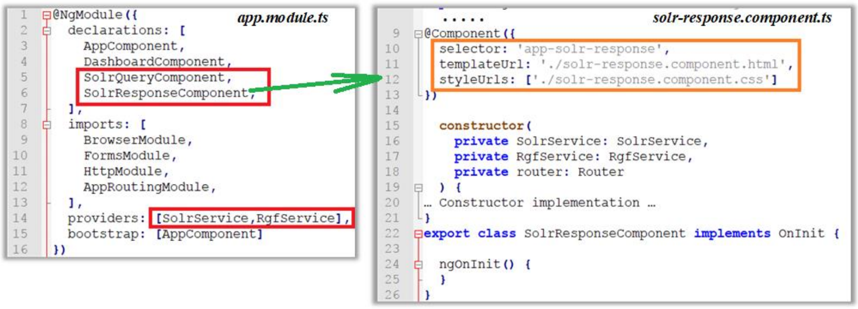


```
<div class="solr-query">
  <app-solr-query>
  </app-solr-query>
</div>
<div class="solr-response">
  <app-solr-response>
  </app-solr-response>
</div>
```

Figure 5

Responding GUI – the same code produces different appearance for desktop (a) and smartphone (b) display

Angular enables clear separating of components and easy arranging of elements on the web page. It also reduces the content of HTML elements combined them with the ones defined in *Angular*. Previous picture illustrates it – GUI contains only *div* HTML elements, while *app-solr-query* and *app-solr-response* are user-defined Angular components. Their definitions are split in three parts - three files generated for each active GUI element in Angular (Figure 6): HTML, CSS and *TypeScript* (*ts*) files (there is additional *spec.ts* file only for testing purposes).



```
@NgModule({
  declarations: [
    AppComponent,
    DashboardComponent,
    SolrQueryComponent,
    SolrResponseComponent,
  ],
  imports: [
    BrowserModule,
    FormsModule,
    HttpModule,
    AppRoutingModule,
  ],
  providers: [SolrService,RgfService],
  bootstrap: [AppComponent]
})
```

```
  .....
@Component({
  selector: 'app-solr-response',
  templateUrl: './solr-response.component.html',
  styleUrls: ['./solr-response.component.css']
})

  constructor(
    private SolrService: SolrService,
    private RgfService: RgfService,
    private router: Router
  ) {
… Constructor implementation …
}
export class SolrResponseComponent implements OnInit {

  ngOnInit() {
  }
}
```

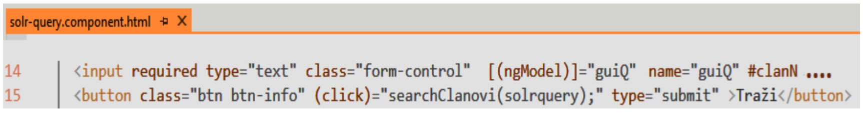
Figure 6

Modular design of Angular application

HTML and CSS files contain details of design, while the *ts* files implement the application logic (interactivity). In addition, a separate file (i.e. *app.module.ts*) contains declarations for each GUI component, imports of built-in library modules and other specifications necessary for running the application. Consequently, the described approach produces low coupling between presentation and functionality behind it enabling if–necessity, or on–demand, easy replacing any of these two.

4.4. The Search Query Processing

App-solr-query is the frontend Angular component (previous section) which receives searching criteria entered by the user (Figure 7) putting it in the variable named *guiQ* (line 14). There is bounded variable of the same name defined in *SolrQueryComponent* class. SEFRA starts searching process when the user clicks the button labeled *Traži* (line 15). This event triggers *searchClanovi* method defined in background class (*SolrQueryComponent* class).


solr-query.component.html

```
14   <input required type="text" class="form-control"  [(ngModel)]="guiQ"  name="guiQ" #clanN ....
15   <button class="btn btn-info" (click)="searchClanovi(solrquery);" type="submit" >Traži</button>
```

Figure 7
Fragment of app-solr-query component

Method *SolrQueryComponent.searchClanovi* starts the *RgfService* firstly (Figure 8), calling its observable method *getDelafs* forwarding the user search criteria as created *query* instance to normalize it by finding the basic form of each word in a query. In more details, the system consequently calls additional three observable operators: *map* – calls the function for each (query) element found, *mergeMap* – calls *SolrService* method named *getClanovi* enabling the use of more than one service at the same time and merging their results, and *subscribe* – that enables an observer object to receive items emitted by an observable instance. In the concrete case, the *SolrQueryComponent* is a subscriber. That means that any change in a query string produces a new request to the Solr service and updates the results presented to the user.

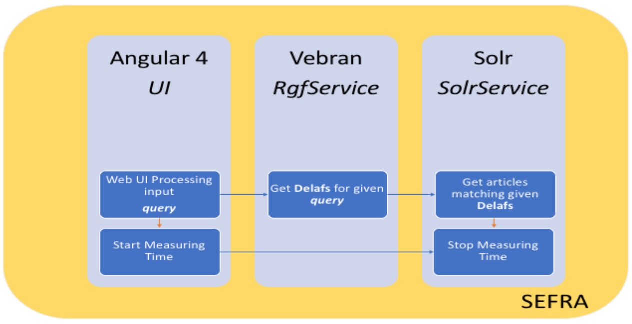

Angular 4
UI
Vebran
RgfService
Solr
SolrService
Web UI Processing input query
Get Delafs for given query
Get articles matching given Delafs
Start Measuring Time
Stop Measuring Time
SEFRA

Figure 8
The core function of SolrQueryComponent

In the last statement, *SolrQueryComponent* uses RxJS synchronization to propagate the information through the rest of the system that initiates the search of specified query (*SolrService.announceQueryStart*).

As mentioned, SEFRA uses *Vebran* service to pre-process the search criteria originally sent by the user to obtain regular query expression. It happens through the *RgfService* class (Figure 9). More precisely, running in separate process thread, *getDelafs* method implements it. Due to *getDelafs* method, instances of *RgfService* class become observable for consumer class instance.

```
getDelafs(parameters: query)
    Set q to query value entered in Web UI
    Initialize params object as empty instance of URLSearchParams class
    Set helper array paramsArr to contain all query keys,values used in UI
    Populate all keys,values from paramsArr into params object
    Get Delafs response from Vebran service and map it to response object
    Return response
```

Figure 9

Angular service implementation for Vebran query

SolrQuery is a class that contains everything necessary for performing a search. SEFRA communicates with the Vebran service through the HTTP GET request sending a query as JSON formatted string. The method *getDelafs* returns content received from the Vebran service to the observer (*SolrQueryComponent*).

In the same manner, *SolrQueryComponent* leverages the other service encapsulated in *SolrSevice* class. After query normalization, *SolrSevice*'s method *getClanovi* forwards query as a JSON string through the HTTP GET request to the *Solr* server (

```
getClanovi(parameters: query)
    Set q to query value entered in Web UI
    Initialize params object as empty instance of URLSearchParams class
    Set helper array paramsArr to contain all query keys,values used in UI
    Populate all keys,values from paramsArr into params object
    Get Articles response from Solr service and map it to response object
    Return response
```

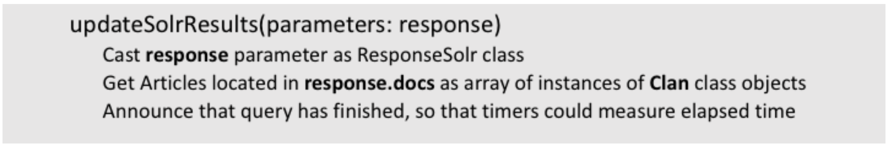
Figure 10).

```
getClanovi(parameters: query)
    Set q to query value entered in Web UI
    Initialize params object as empty instance of URLSearchParams class
    Set helper array paramsArr to contain all query keys,values used in UI
    Populate all keys,values from paramsArr into params object
    Get Articles response from Solr service and map it to response object
    Return response
```

Figure 10

Sending of search query to the Solr server

Since *getClanovi* is an observable method, it returns content received from the service to the observer (*SolrQueryComponent*). More precisely, it triggers *updateSolrResults* method subscribed to wait this response (Figure 8, line 138) encapsulated in *ResponseSolr* instance for search results (Figure 11). It contains a set of highlighted document fragments that fit the criteria, the links and similarity scores as well.



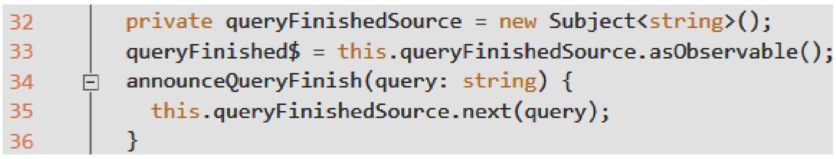
```
updateSolrResults(parameters: response)
    Cast response parameter as ResponseSolr class
    Get Articles located in response.docs as array of instances of Clan class objects
    Announce that query has finished, so that timers could measure elapsed time
```

Figure 11
The last preparations before presenting search results

After the preparation, *SolrQueryComponent* calls the *SolrService* to announce that the searching is finished, and results are ready for presenting. Further, the system broadcast this information for updating GUI components that present the results.

4.5. Presentation of Search Results

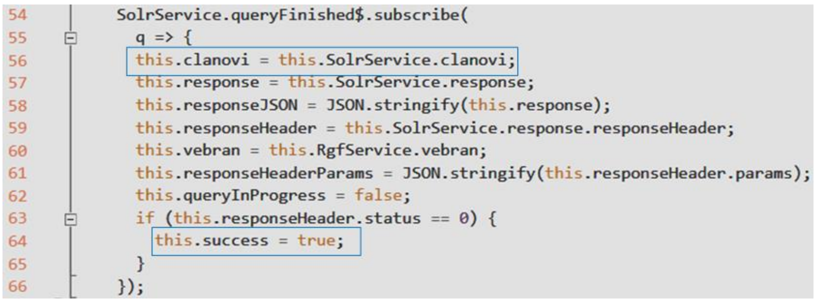
An appropriate binding between GUI components and needed features of the SEFRA provides handling of the user requests and system responses separately because there are several services and asynchronous calls used for this purpose. *SolrService* class is responsible for a mutual synchronization in this complex process. It uses broadcasted events (*observables*) for triggering appropriate component functions. After completing the query task, *SolrQueryComponent* forces *SolrService* to emit this information to the all subscribed objects (Figure 11, line 89). It performs this task by using observable string subject *queryFinishedSource* and observable string stream *queryFinished$* (Figure 12).



```
private queryFinishedSource = new Subject<string>();
queryFinished$ = this.queryFinishedSource.asObservable();
announceQueryFinish(query: string) {
  this.queryFinishedSource.next(query);
}
```

Figure 12
Broadcasting the event when the search has finished

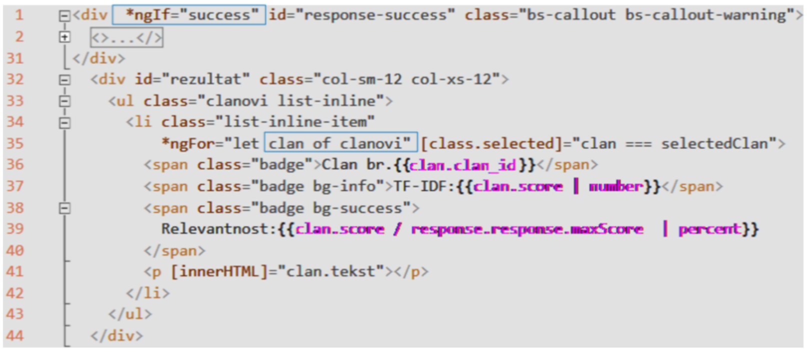
SolrResponseComponent is a class responsible for presenting searching results. Subscribing the stream named *queryFinished$* (Figure 13), it receives the event when the search has finished and prepares the results for rendering. SolrResponseComponent extracts all the information necessary to perform mentioned task.



```
    SolrService.queryFinished$.subscribe(
      q => {
      this.clanovi = this.SolrService.clanovi;
      this.response = this.SolrService.response;
      this.responseJSON = JSON.stringify(this.response);
      this.responseHeader = this.SolrService.response.responseHeader;
      this.vebran = this.RgfService.vebran;
      this.responseHeaderParams = JSON.stringify(this.responseHeader.params);
      this.queryInProgress = false;
      if (this.responseHeader.status == 0) {
        this.success = true;
      }
    });
```

Figure 13
SolrResponseComponent subscribed for event that the search is finished

Angular GUI component *app-solr-response* is responsible for presenting the search results to the user (Figure 14). This component is acting along with *SolrResponseComponent* class and they share the same scope. In other words, the variables defined in *SolrResponseComponent* are visible to *app-solr-response* and vice versa.



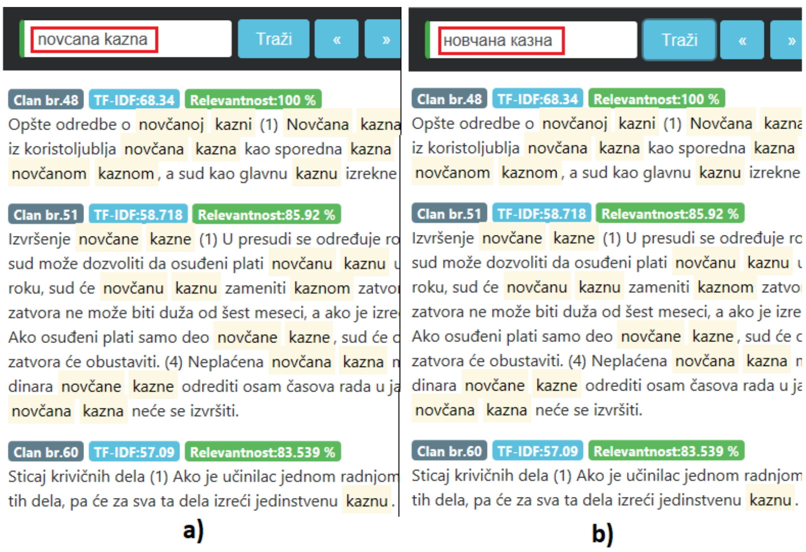
```
<div  *ngIf="success" id="response-success" class="bs-callout bs-callout-warning">
  <>...</>
</div>
  <div id="rezultat" class="col-sm-12 col-xs-12">
    <ul class="clanovi list-inline">
      <li class="list-inline-item"
          *ngFor="let clan of clanovi" [class.selected]="clan === selectedClan">
        <span class="badge">Clan br.{{clan.clan_id}}</span>
        <span class="badge bg-info">TF-IDF:{{clan.score | number}}</span>
        <span class="badge bg-success">
          Relevantnost:{{clan.score / response.response.maxScore  | percent}}
        </span>
        <p [innerHTML]="clan.tekst"></p>
      </li>
    </ul>
  </div>
```

Figure 14
GUI component

This way a variable *success* is examined by using *ngIf* directive to check the conditions if there are quantitative details of the search process (this part of code is collapsed) which should be shown. The collection *clanovi* represents the search result that is iterated through by using *ngFor* directive. The component represents each element in this collection by using temporary variable *clan* showing its id, absolute and relative score, and textual content.

5 Evaluation

For evaluation purposes, the set of criminal low documents written in Serbian Latin is used as a searching content. SEFRA framework shows a flexible behavior for different test cases. The following examples illustrate it. In the first one, irrespectively whether the search criteria have been written in different alphabets – Latin (Figure 15a) or Cyrillic (Figure 15b), SEFRA obtained the identical results in both cases. The ranking, similarity measures and relevance were the same. Moreover, the example shows that SEFRA responses appropriately on inaccurate written query – *novcana kazna* (eng. *Fine/monetary penalty*). Instead of correcting word *novčana*, word *novcana* is used. In other words, it compensates the case in which the user cannot use specific letters of national alphabet (e.g. mobile devices or keyboard without this kind of support).



Clan br.48 TF-IDF:68.34 Relevantnost:100 %
Opšte odredbe o novčanoj kazni (1) Novčana kazna
iz koristoljublja novčana kazna kao sporedna kazna
novčanom kaznom, a sud kao glavnu kaznu izrekne

Clan br.51 TF-IDF:58.718 Relevantnost:85.92 %
Izvršenje novčane kazne (1) U presudi se određuje ro
sud može dozvoliti da osuđeni plati novčanu kaznu u
roku, sud će novčanu kaznu zameniti kaznom zatvo
zatvora ne može biti duža od šest meseci, a ako je izre
Ako osuđeni plati samo deo novčane kazne, sud će o
zatvora će obustaviti. (4) Neplaćena novčana kazna n
dinara novčane kazne odrediti osam časova rada u ja
novčana kazna neće se izvršiti.

Clan br.60 TF-IDF:57.09 Relevantnost:83.539 %
Sticaj krivičnih dela (1) Ako je učinilac jednom radnjom
tih dela, pa će za sva ta dela izreći jedinstvenu kaznu.

a)

Clan br.48 TF-IDF:68.34 Relevantnost:100 %
Opšte odredbe o novčanoj kazni (1) Novčana kazna
iz koristoljublja novčana kazna kao sporedna kazna
novčanom kaznom, a sud kao glavnu kaznu izrekne

Clan br.51 TF-IDF:58.718 Relevantnost:85.92 %
Izvršenje novčane kazne (1) U presudi se određuje ro
sud može dozvoliti da osuđeni plati novčanu kaznu u
roku, sud će novčanu kaznu zameniti kaznom zatvo
zatvora ne može biti duža od šest meseci, a ako je izre
Ako osuđeni plati samo deo novčane kazne, sud će c
zatvora će obustaviti. (4) Neplaćena novčana kazna n
dinara novčane kazne odrediti osam časova rada u ja
novčana kazna neće se izvršiti.

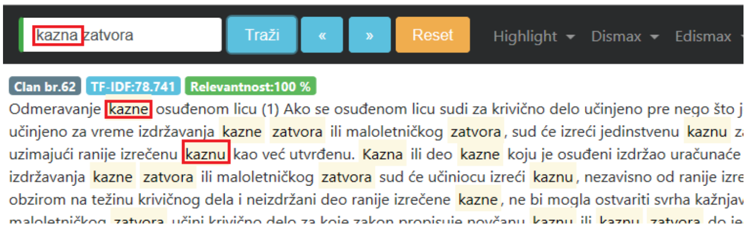
Clan br.60 TF-IDF:57.09 Relevantnost:83.539 %
Sticaj krivičnih dela (1) Ako je učinilac jednom radnjom
tih dela, pa će za sva ta dela izreći jedinstvenu kaznu.

b)

Figure 15

The same query written in different alphabets produces the same result

SEFRA also has a flexible search for different word forms found in documents. For concrete searching term *kazna zatvora* (eng. *Prison sentence*), it responses with document ranking that shows the terms are counted regardless of their forms (singular/plural, tenses, grammatical cases etc.). Consequently, there is a minor influence of word forms on a final documents' rank.

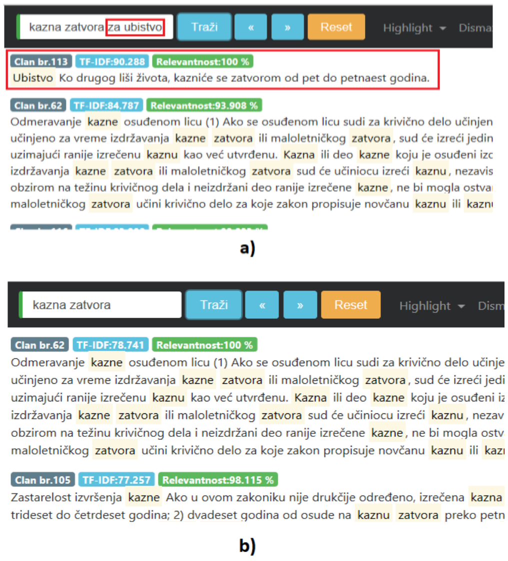


Clan br.62 TF-IDF:78.741 Relevantnost:100 %
Odmeravanje kazne osuđenom licu (1) Ako se osuđenom licu sudi za krivično delo učinjeno pre nego što j
učinjeno za vreme izdržavanja kazne zatvora ili maloletničkog zatvora, sud će izreći jedinstvenu kaznu z
uzimajući ranije izrečenu kaznu kao već utvrđenu. Kazna ili deo kazne koju je osuđeni izdržao uračunaće
izdržavanja kazne zatvora ili maloletničkog zatvora sud će učiniocu izreći kaznu, nezavisno od ranije izre
obzirom na težinu krivičnog dela i neizdržani deo ranije izrečene kazne, ne bi mogla ostvariti svrha kažnjav
maloletničkog zatvora učini krivično delo za koje zakon propisuje novčanu kaznu ili kaznu zatvora do je

Figure 16

Flexible response on different word forms

For more details in search criteria SEFRA produces better results in document selection and ranking. Next example shows expanded criteria *kazna zatvora za ubistvo* (eng. *Prison sentence for murder*, Figure 17a) related with the previous one *kazna zatvora* (eng. *Prison sentence*, Figure 17b). The best-fit (100% relevancy) document explains a *murder* as a term and time range the jury can punish the accused with, while previously first-ranked document is shifted down the list.



a)

Clan br.113 TF-IDF:90.288 Relevantnost:100 %
Ubistvo Ko drugog liši života, kazniće se zatvorom od pet do petnaest godina.

Clan br.62 TF-IDF:84.787 Relevantnost:93.908 %
Odmeravanje kazne osuđenom licu (1) Ako se osuđenom licu sudi za krivično delo učinjen
učinjeno za vreme izdržavanja kazne zatvora ili maloletničkog zatvora, sud će izreći jedin
uzimajući ranije izrečenu kaznu kao već utvrđenu. Kazna ili deo kazne koju je osuđeni izc
izdržavanja kazne zatvora ili maloletničkog zatvora sud će učiniocu izreći kaznu, nezavis
obzirom na težinu krivičnog dela i neizdržani deo ranije izrečene kazne, ne bi mogla ostva
maloletničkog zatvora učini krivično delo za koje zakon propisuje novčanu kaznu ili kazn

b)

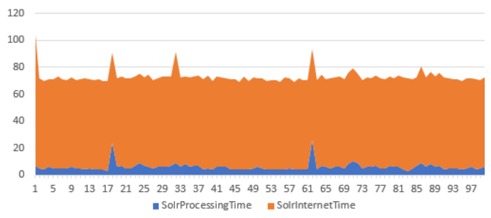
Clan br.62 TF-IDF:78.741 Relevantnost:100 %
Odmeravanje kazne osuđenom licu (1) Ako se osuđenom licu sudi za krivično delo učinje
učinjeno za vreme izdržavanja kazne zatvora ili maloletničkog zatvora, sud će izreći jedi
uzimajući ranije izrečenu kaznu kao već utvrđenu. Kazna ili deo kazne koju je osuđeni iz
izdržavanja kazne zatvora ili maloletničkog zatvora sud će učiniocu izreći kaznu, nezav
obzirom na težinu krivičnog dela i neizdržani deo ranije izrečene kazne, ne bi mogla ostv
maloletničkog zatvora učini krivično delo za koje zakon propisuje novčanu kaznu ili kazı

Clan br.105 TF-IDF:77.257 Relevantnost:98.115 %
Zastarelost izvršenja kazne Ako u ovom zakoniku nije drukčije određeno, izrečena kazna
trideset do četrdeset godina; 2) dvadeset godina od osude na kaznu zatvora preko petn

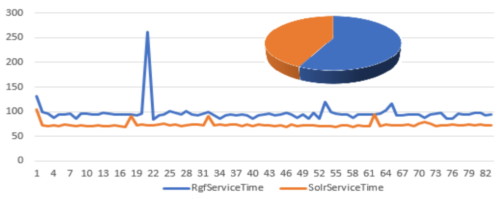
Figure 17

The more details in search criteria the better results in response

Additionally, there is a quantitative analysis performed through two types of measurement. The first one shows the *Solr* service response time (Figure 18a). As expected, service-processing time is 5 to 10 times less than service-delivering time. A satisfactory fact is that the *Solr* aggregate response time varies from 10^{-2} to 10^{-1} seconds.



a)



b)

Figure 18

SEFRA quantitative analysis

As SEFRA includes using of outside services, its performances depend on their response time. In the concrete study, *Vebran* and *Solr* services are used (see Section 3). The second chart (Fig. 18b) presents the performance of these two services used together. It shows timelines (x-axis values are in hours) of their responses. It is obvious that (excepting one remarkable peak produced by *Vebran*) there are minor differences between them. On the other hand, pie chart shows that *Solr* consumes 43% while *Vebran* consumes 57% of aggregate response time which varies from 156 to 236 ms with the average response time of 170 ms. Also, in comparison with previous work [2], when consider precision, recall and accuracy metrics, SEFRA has obtained better validity results performance in considerably more advanced testing conditions but in comparison with work [6] which is one optimization of algorithm described in [2] it has poorer results. These results are an excellent starting point for further development.

Table 1

Validity performance results

PERFORMANCE	Work [2]	Work [6]	SEFRA
Precision	75.71%	49.67%	78.3%
Recall	57%	49.67%	57.76%
Accuracy	46.66%	74.83%	48.21%

Conclusions

SEFRA represents one of the rare solutions focused on improving search capabilities of specific (*non-English*) language(s). Regardless of a significant progress made in natural language processing (NLP) of Serbian and other western Balkan languages (i.e. south Slavic languages group), neither commercial solutions, nor similar public services exist yet. Fortunately, there are Web services (as *Vebran*) that enable further NLP development offering their capacities to the researchers and developers. Moreover, SEFRA brought important contribution as new TypeScript libraries for Angular 4 framework both for Solr as well as Vebran backend services. Therefore, new Solr and Vebran libraries written as a part of SEFRA can be easily reusable by research and development community, so that powerful language and search services can be leveraged by simple instantiation of Classes defined in SEFRA libraries.

From design prospective, SEFRA is a hybrid Web framework with processing power balanced between advanced application delivered to the platforms on the client side and different Web services on server side. At the same time, it represents a proof of concept that it is possible to make reliable and efficient synchronization of different Web services on the client side. This approach which is very similar to *observer* design pattern, enables easy subscribing of new services in the roles of observables (e.g. for pre-processing the search queries, or for displaying results to the end users), or observers (e.g. different Web search services that can be used). This flexibility is also useful to relieve search engines, as SEFRA will not engage search services if there are no results returned from query pre-processing service(s).

During the evaluation, SEFRA satisfied the expectations of various searching tasks preformed. The collection of criminal low documents written in Serbian Latin enabled us a full control during this process (comparison of obtained and expected results). Domain experts agreed almost with all search results, which means SEFRA made reliable selection and ranking of documents. Inability to handle search queries that include different forms, cases and tenses and delivering nothing, or unexpected results, represent the main weaknesses of already existing searching engines for Serbian content. Therefore, such problems became also high-priority evaluation tasks. SEFRA uses *Vebran* service as a part of solution. Specific set up of *Solr* server as a search engine represents the other service being used. The Solr service was prepared by adding new fields, configuring the stop-words set and set of synonyms, and modifying indexing schema. In other words, the issues described above became solvable by using different services combined and synchronized in the joint solution.

As a modular and flexible framework built of low-coupled and easy-to-change components that interact with each other through the standardized services, SEFRA provides conditions for making modifications and improvements permanently. The core component is *Angular* multithread application (SEFRA client) that can manage any number of services involved in the search process. Holding services in the separate threads, SEFRA client synchronizes the API calls and mutual information exchange making them to act as a whole. Alternatively, it delivers the results to the end user in seconds, hiding a lot of processing performed on various distributed platforms.

There are several ways for future development. Improving the search quality by including new services is one of them. For instance, there are many foreign companies running their business in Serbia. Including Bi/Multi-lingual services can significantly improve SEFRA usability. On the other hand, it is necessary to index as much available content as possible. Increasing quantity of the content results in the need for a content categorization (clustering). Moreover, it implies separate, domain-specific dictionaries for this purpose. Finally, every new (domain-specific) collection requires resetting of search engine(s). As Web content is constantly changing, the researchers and developers face the challenges in the same manner. SEFRA provides well-formed infrastructure for such efforts.

References

[1] Miniwatts Marketing Group, “Internet World Stats”, Miniwatts Marketing Group, [Online]. Available: http://www.internetworldstats.com/stats7.htm [Accessed August 2017]

[2] V. Nikolić, B. Markoski, K. Kuk, D. Randjelović, P. Čisar, “Modelling the System of Receiving Quick Answers for e-Government Services: Study for the Crime Domain in the Republic of Serbia”, *Acta Polytechnica Hungarica*, Vol. 14, No. 8, pp. 143-163, 2017

[3] M. Martinović, S. Vesić and G. Rakić, “Building an Information Retrieval System for Serbian - Challenges and Solutions”, *Springer International Publishing*, 2015

[4] G. Šimić, Z. Jeremić, E. Kajan, D. Randjelović and A. Presnall, “A Framework for Delivering e-Government Support”, *Acta Polytechnica Hungarica*, Vol. 11, No. 1, pp. 79-96, 2014

[5] O. Kolomiyets and M.-F. Moens, “A Survey on Question Answering Technology from an Information Retrieval Perspective”, *Information Sciences*, No. 181, pp. 5412–5434, 2011

[6] S. Nedeljković, V. Nikolić, M. Čabarkapa, J. Mišić, D. Randjelović, “An Advanced Quick-Answering System Intended for the e-Government Service of the Republic of Serbia”, *Acta Polytechnica Hungarica*, paper accepted for publishing in 2019

[7] C. Gyorodi, R. Gyorodi, G. Pecherle and G. Mihai Cornea, “Full-Text Search Engine Using MySQL”, *Int. J. of Computers*, Vol. V, pp. 735-743, 2010

[8] S.-C. Necula, “Implementing the Main Functionalities Required by Semantic”, *International Journal of Computers Communications & Control*, Vol. 7, No. 5, 2012

[9] U. Shoaib, N. Ahmad, P. Prinetto and G. Tiotto, “Integrating MultiWordNet with Italian Sign Language Lexical Resources”, *Expert Systems with Applications,* Vol. 41, No. 5, pp. 2300-2308, 2014

[10] B. Furlan, V. Batanović and B. Nikolić, “Semantic Similarity of Short Texts in Languages with a Deficient Natural Language Processing Support”, *Decision Support Systems,* Vol. 55, No. 3, pp. 710-719, 2013

[11] D. W. Castro, E. Souza, D. Vitório, D. Santos and A. L. Oliveira, “Smoothed n-gram Based Models for Tweet Language Identification: A Case Study of the Brazilian and European Portuguese National Varieties”, *Applied Soft Computing,* pp. 1568-4946, 2017

[12] D. Ivanović, D. Surla, M. Trajanović, D. Misić and Z. Konjović, “Towards the Information System for Research Programmes of the Ministry of Education, Science and Technological Development of the Republic of Serbia”, *Procedia Computer Science,* Vol. 106, pp. 122-129, 2017

[13] A. Selemat and N. Akosu, “Word-Length Algorithm for Language Identification of Under-Resourced Languages”, *Journal of King Saud University - Computer and Information Sciences,* Vol. 28, No. 4, pp. 457-469, 2016

[14] G. Kowalski and M. Maybury, Information Storage and Retrieval Systems, Springer US, 2002

[15] H. Bast and B. Buchhold, “An Index for Efficient Semantic Full-Text Search”, in *International Conference on Information and Knowledge Management, Proceedings*, 2013

[16] Apache Solr, “Overview of Searching in Solr”, 2017 [Online] Available: https://lucene.apache.org/solr/guide/6_6/

[17] M. White, Enterprise Search, 2nd Edition ed., O'Reilly Media, Inc., 2015

[18] Google, Inc, “Angular Fundamentals – Overview”, 2017. [Online]. Available: https://angular.io/guide/architecture

[19] Twitter Bootstrap, “Bootstrap”, 2017 [Online] Available: http://getbootstrap.com/

[20] Y. Fain and A. Moiseev, Angular 2 Development with Typescript, Manning Publications, 2016

[21] “Microsoft Typescript project on GitHub”, 2017 [Online] Available: https://github.com/Microsoft/TypeScript

[22] SourceMaking.com, “Design Patterns”, 2017 [Online] Available: https://sourcemaking.com/design_patterns

[23] S. Salehi, Angular Services, Packt Publishing, 2017