# Upravljanje digitalnim dokumetima (UDD)

Predlog projekta, Ivan Mršulja R2-30/2022

## Arhitektura Sistema i *High-Level* pregled

Sitem za pretragu aplikacija korisnika i uvid u izvještaje će biti realizovan kao poseban servis. Glavni razlog za ovo je nemogućnost izmjene *SpringBoot* aplikacije generisane od strane *Bonita BPM*-a kojeg koristimo za izradu *web* aplikacije agencije za zapošljavanje. Tehnologije koje će se koristiti su *SpringBoot* za backend i *Vue3* za frontend. *Elasticsearch*, *Logstash* i *Kibana* će biti pokrenute u *Docker* kontejnerima zarad jednostavnije instalacije dok će se sva potrebna konfiguracija (.conf i .yml fajlovi) prosleđivati preko environment varijabli. Ukoliko ovaj vid konfiguracije bude proizvodio neočekivane bagove, alterativa je lokalna instalacija (jedne ili sve tri) komponente *ELK stack*-a i malo drugačija konfiguracija koja će biti objašnjena u nastavku. Iicijalna ideja kako će ovaj sistem biti integrisan u postojeću web aplikaciju je sledeća:

- Ukoliko zaposleni želi pretragu aplikacija, CV dokumenata i propratnih pisama, klikom na dugme biva redirektovan sa sajta aplikacije na sajt servisa za pretragu gdje ima pristup svim funkcioalnostima navedenim u specifikaciji projekta
- U momentu kada šef tima za premium zapošljavanje dobije zadatak da bira zaposlenog koji će obraditi zahtjev, klikom na dugme "*Reports*" biva redirektovan na sajt servisa za pretragu gdje ima uvid u sve izvještaje navedene u specifikaciji
- Ukoliko u bilo kom drugom trenutku, zaposleni žele da imaju pristup ovim funkcionalnostima, potrebno je da ručno odu na sajt servisa gdje će im poslije neophodne prijave biti omogućene sve funkcionalnosti za njihovu rolu, shodno tome, aplikacija će biti povezana na PostgreSQL bazu kako bi skladištila *login* kredencijale, role i permisije.

## Indeksiranje i osnovna pretraga kolekcije aplikacija, dostavljenih CV dokumenata i propratnih pisama

Indeksiranje aplikacije i dostavljenih dokumenata se vrši sa osloncem na *ElasticsearchRepository* interfejs. Da bi ovo bilo omogućeno potrebo je izvršiti *SpringBoot* konfiguraciju gdje ćemo specificirati pakete koji sadrže *Elasticsearch* repozitorijume kao i ručno dodavanje *bean*-ova pomoću *@ComponentScan* anotacije.

```
@EnableElasticsearchRepositories(basePackages
        = "com.rs.elasticsearchservice.repository")
@ComponentScan(basePackages = { "com.rs.elasticsearchservice" })
```

Nakon toga, potrebno je odraditi svojevrsu konfiguraciju *RestHighLevelClient*-a kako bi aplikacija znala da pročita *respons*-e sa *Elasticsearch* servisa.

```
Ivan Mrsulja
@Bean(destroyMethod = "close")
public RestHighLevelClient restClient() {

    final CredentialsProvider credentialsProvider = new BasicCredentialsProvider();
    credentialsProvider.setCredentials(AuthScope.ANY, new UsernamePasswordCredentials(userName, password));

    RestClientBuilder builder = RestClient.builder(new HttpHost(host, port, protocol))
            .setHttpClientConfigCallback(httpClientBuilder -> httpClientBuilder.setDefaultCredentialsProvider(credentialsProvider))
            .setDefaultHeaders(compatibilityHeaders());

    return new RestHighLevelClient(builder);
}

1 usage  Ivan Mrsulja *
private Header[] compatibilityHeaders() {
    return new Header[]{new BasicHeader(HttpHeaders.ACCEPT,  value: "application/vnd.elasticsearch+json;compatible-with=7"),
            new BasicHeader(HttpHeaders.CONTENT_TYPE,  value: "application/vnd.elasticsearch+json;compatible-with=7")};
}
```

Bitno je napomenuti da iako je *RestHighLevelClient* sada već *deprecated*, zajednica podržava njegovo korišćenje zbog *backward* kompatibilnosti sa *Elasticsearch* 7.X.X verzijama. Moderniji pandan bi bila direktna konfiguracija *ElasticsearchOperations bean*-a s tim što mu ne bi prosleđivali *RestClient* već noviji *ElasticsearchClient*. Ostatak konfiguracije izgledao bi identično.

Indeksiranje se vrši na isti način kao i na primjerima sa Vašeg *GitHub*-a. Tekstualna polja se inderksiraju na trivijalan način, dok se PDF dokumenta prvo čuvaju lokalno na servisu (u fajl sistemu), a zatim parsiraju uz pomoć *pdfbox* biblioteke i potom se indeksiraju kao tekstualni sadržaj u *Elasticsearch*-u. Izgled klase DTO objekta kao i klase indeksirajućeg objekta možete vidjeti u nastavku.

```
2 usages  Ivan Mrsulja
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
public class CandidateApplicationIndexDTO {

    private String name;

    private String surname;

    private String education;

    private String address;

    private MultipartFile cv;

    private MultipartFile motivationLetter;
}
```

```java
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
@Document(indexName = "applications")
public class CandidateApplication {

    @Id
    private String id;

    1 usage
    @Field(type = FieldType.Text, store = true, name = "name")
    private String name;

    1 usage
    @Field(type = FieldType.Text, store = true, name = "surname")
    private String surname;

    1 usage
    @Field(type = FieldType.Text, store = true, name = "education")
    private String education;

    1 usage
    @Field(type = FieldType.Text, store = true, name = "cv")
    private String cv;

    1 usage
    @Field(type = FieldType.Text, store = true, name = "letter")
    private String letter;

    @Field(type = FieldType.Text, store = true, name = "cvPath")
    private String cvPath;

    @Field(type = FieldType.Text, store = true, name = "letterPath")
    private String letterPath;

    1 usage
    @GeoPointField
    @Field(store = true, name = "location")
    private GeoPoint location;
```

Odavde možemo vidjeti da je indeksiran i *GeoPoint* koji predstavlja adresu kandidata. Iz obične tekstualne adrese uz pomoć *LocationIQ API*-a se mogu veoma lako dobiti geografska širina i dužina koji se kasnije mogu iskoristiti za instanciranje *GeoPoint* klase koja nam omogućava geoprostornu pretragu. Atributi *cvPath* i *letterPath* predstavljaju putanju do lokalno sačuvanih dokumenata i evetualni način za realizaciju dodatne funkcioalnosti preuzimanja  i pregleda dokumenata za određenu aplikaciju. Kako ovo nije traženo u specifikaciji projekta, u zavisnosti od mojih obaveza zavisi i sudbina ove funkcionalnosti.

Osnovna pretraga po sadržaju tekstualnih polja kao i sadržaju dokumenata je moguće kompletno odraditi pomoću *ElasticsearchRepository* interfejsa. Dovoljno je deklarisati ovu metodu i u *Query* anotaciji navesti *Elasticsearch* upit.

```java
1 usage   Ivan Mrsulja *
@Query("{\"bool\" : " +
        "{\"should\" : [ " +
        "{\"field\" : {\"name\" : \"?\"}}, " +
        "{\"field\" : {\"surname\" : \"?\"}}, " +
        "{\"field\" : {\"education\" : \"?\"}} " +
        "{\"field\" : {\"cv\" : \"?\"}} " +
        "{\"field\" : {\"letter\" : \"?\"}} " +
        "]}}")
Page<CandidateApplication> findByNameOrSurnameOrEducation(String name, String surname, String education, String cv, String letter, Pageable pageable);
```

## Kombinacija prethodnih parametara pretrage (BooleanQuery + operatori + PhrazeQuery)

S obzirom da je u *ElasticSearchRepository* nemoguće dinamički kreirati upit ova funkcionalnost će biti realizovana uz oslonac na *ElasticsearchOperations* klasu (moderniji pandan *ElasticsearchTemplate* koji podržava sve iste funkcionalnosti i ima skoro identičan API). Inicijalna zamisao je da se upit dobija sa *frontend*-a kao niz tokena gdje svaki token odgovara jednom operatoru/operandu u *Lucene query* jeziku.

```
"expression": ["name:Ivan", "NOT", "surname:Mrsulja", "OR", "surname:popovic", "AND", "education:\"PhD\"", "OR", "education:MSC", "OR", "cv:aaaaa"]
```

Za unos ovakvog *query*-a, na *frontend*-u, se može koristiti neki *discord-style search bar*. Kada smo na *backend*-u dobili niz tokena, koristi se jedostavan parser zasnovan na *stack* mšinama. Upit se prvo prevodi u postfiksnu notaciju a zatim uz pomoć *stack* mašine pretvara u odgovarajući *query* uz pomoć *QueryBuilder*-a. U zavisosti da li je operand zadat pod znakovima navoda, u *QueryBuilder*-u je moguće specificirati da li koristimo *MatchPhrazeQuery* ili standardan *MatchQuery* (ne koristi se *TermQuery* jer se u praksi pokazalo da se bolji kvalitet pretrage dobije kada upit ne mora biti *case sesitive*). Ovakva implementacija pogodna je i za lako proširenje koje bi podržavalo i *FuzzyQuery* kao i *PrefixQuery*. Još jedan benefit ove implementacije je laka promjena prioriteta operatora kao i dodavanje novih i podržavanje redefinicije prioriteta pomoću zagrada. U inicijalnoj implementaciji podržao sam AND, NOT i OR operator sa poredkom kao u C programskom jeziku (NOT > AND > OR). Konfiguracija prioriteta operatora se implementira pomoću standardne matrice prioriteta.

```
3 usages
private final Map<String, Integer> priorities = Map.of( k1: "AND", v1: 2,  k2: "OR",  v2: 1,  k3: "NOT",  v3: 3);
```

## Pretprocesiranje upita pomoću *SerbianAnalyzer*-a

S obzirom da je u novijim verzijama *Elasticsearch*-a *SerbianAnalyzer* podržan kao ugrađeni plugin, potrebno ga je samo specificirati u *query*-u u *analyzer* sekciji.

```
GET my-index-000001/_search
{
  "query": {
    "match": {
      "message": {
        "query": "Quick foxes",
        "analyzer": "serbian"
      }
    }
  }
}
```

Alternativno, ako ovo ne bude radilo, moguće je instalirati *SearbianAnalyzer* kako je specificirano ovdje. U tom slučaju *SerbianAalyzer* bi bio korišćen za pretprocesiranje svakog zahtjeva.

## Geoprostorna pretraga

U sekciji posvećeoj indeksiranju objašnjeno je kako se lokacija može indeksirati. Što se pretrage tiče, sa *frotend*-a uz *query* dobija se i ime grada i radijus u kilometrima. A na *backend*-u se uz pomoć, prethodno pomenutog, *LocationIQ* API-ja dobijaju koordinate početne tačke pretrage i uz pomoć *QueryBuilder*-a moguće je u prethodo kreiran *query* ubaciti i dodatni *geoDistanceQuery*.

```
finalQuery = QueryBuilders.boolQuery()
        .must(finalQuery)
        .must(QueryBuilders.geoDistanceQuery( name: "location")
                .distance( distance: 100, DistanceUnit.KILOMETERS)
                .point( lat: 45.25, lon: 19.81));
```

Napomena: U ovom primjeru parametri su zakucani što će, naravno, biti promijenjeno prilikom finalne odbrane.

## Dinamički sažetak

Za kreiranje dinamičkog sažetka može se koristiti *NativeSearchQueryBuilder* kojem se prosleđuje koja polja da *highlight*-uje prilikom upita kao i koliko okolnih karaktera da uključi u sažetak.

```
var searchQuery : NativeSearchQuery = new NativeSearchQueryBuilder()
        .withQuery(query)
        .withHighlightFields(
                new HighlightBuilder.Field("name").fragmentSize(200),
                new HighlightBuilder.Field("surname").fragmentSize(200),
                new HighlightBuilder.Field("education").fragmentSize(200),
                new HighlightBuilder.Field( name: "cv").fragmentSize(200),
                new HighlightBuilder.Field( name: "letter").fragmentSize(200))
        .build();
```

Dinamički sažetak se odnosi samo na CV i propratno pismo s obzirom da nema smisla *highlight*-ovati polja koja sadrže jednu do dvije riječi.

## *Logstash* i *Kibana*

*Logstash* je alat za vođenje evidencije koji prihvata unose iz različitih izvora, izvršava različite transformacije i izvozi podatke u različite ciljeve. Predstavlja dinamički *pipeline* za prikupljanje podataka i posjeduje snažnu koheziju sa *Elasticsearch*-om. Inicijalna zamisao je da se logovi *SpringBoot* aplikacije čuvaju u zaseban .log fajl. Prilikom svakog podnošenja zahtjeva za premium zapošljavanje na servisu za zapošljavanje slaće se log na servis za pretragu i izvještavanje koji će taj log upisati u prethodo pomeuti fajl. Zatim će se uz pomoć *logback* biblioteke sa prethodno definisanim *grok* filterom u *logstash.conf* fajlu izvršiti slanje logova bitnih za izvještaje iz fajla na *Elasticsearch*. Format logova bi bio “Grad-Kandidat-

Kompanija”. Moguće je odraditi proširenje da se za grad ne koristi adresa kandidata već da se on dobavi preko IP adrese uz pomoć *IPGeolocationAPI*-a ukoliko iz *Bonita BPM*-a bude moguće poslati IP adresu trenutnog korisnika koji podnosi zahtjev. Čitav postupak je detaljno opisan ovdje.

*Kibana* predstavlja vizualizacioni alat za *Elasticsearch*. U *Kibani* je moguće defiisati *custom report* iz *Logstash datastream*-a uz pomoć *Kibana Query Language*-a (KQL) po sledećem uputstvu. Kada je jednom definisan način za njegovo generisanje, moguće ga je preko *IFrame*-a ubaciti u *frontend* moje aplikacije.