Upravljanje digitalnim dokumentima

Predlog projekta, Nevena Gligorov E2 7/2024

Arhitektura sistema i high-level pregled

Sistem za monitoring i pretragu sigurnosnih incidenata će biti realizovan kao *SpringBoot* aplikacija na *backend*-u i *Angular* aplikacija na *frontend*-u. *Elasticsearch*, *Logstash* i *Kibana* koji predstavljaju osnovu *ELK stack*-a, zajedno sa *MinIO*-om, biće pokrenuti preko *Docker*-a uz pomoć *docker-compose.yml* fajla i odgovarajućih konfiguracionih fajlova. Za sve komponente *ELK stack*-a će se koristiti verzija 8.15, a ukoliko se naiđe na problem prilikom korišćenja ove verzije, preći će se na neku drugu verziju, gde bi poslednja solucija bila lokalna instalacija komponenti koje to budu zahtevale. Osim pomenutih, koristiće se i *PostgreSQL* baza koja će lokalno čuvati informacije vezane za korisnike i koja će predstavljati mesto prezistentnosti podataka kada je reč o sigurnosnim incidentima.

Komponente arhitekture

Elasticsearch je distribuirani, *open source* mehanizam za pretragu i analitiku. *Dependency* i konfiguracija koja je potrebna da bi se *Elasticsearch* koristio u okviru aplikacije su dati na slikama 1 i 2.

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-elasticsearch</artifactId>
    <version>3.4.0</version>
</dependency>

<dependency>
    <groupId>org.elasticsearch</groupId>
    <artifactId>elasticsearch</artifactId>
    <version>8.15.2</version>
</dependency>
```

Slika 1. Elasticsearch dependency

```java
@Configuration  no usages
@EnableElasticsearchRepositories(basePackages = "repositoryIndex.SecurityIncidentIndexRepository")
public class ElasticsearchConfiguration extends org.springframework.data.elasticsearch.client.elc.ElasticsearchConfiguration {
    @Override  no usages
    public ClientConfiguration clientConfiguration() {
        return ClientConfiguration.builder().connectedTo( hostAndPort: "localhost:9200")
                .withBasicAuth( username: "",  password: "").build();
    }
}
```

Slika 2. Elasticsearch konfiguracija

Logstash je *open source pipeline* za obrađivanje podataka koji prihvata podatke sa jednog ili više izvora, transformiše ih i prosleđuje ka jednom ili više izlaza, odnosno odredišta. U slučaju sistema za monitoring i pretragu sigurnosnih incidenata, *Logstash* će podatke primati sa samo jednog izvora, koji predstavlja *.log* fajl u kome će se nalaziti logovi *SpringBoot* aplikacije. Nakon pristizanja, sigurnosni incidenti će biti sačuvani u *PostgreSQL*-u, prilikom čega će servis tu informaciju upisati i u *.log* fajl u formatu: *Timestamp LogLevel Attacked Org: AOName where Security Org: SOName with Member of incident team: fullName and severity: severity*. *Grok* filter podatke sa ulaza obrađuje i strukturirane ih šalje ka izlazu definisanom u konfiguracionom fajlu, odnosno ka *Elasticsearch*-u.

Kibana omogućava vizualizaciju indeksiranih podataka, njihovo analiziranje i kreiranje različitih grafika. Povezivanje sa *Elasticsearch*-om je realizovano uz pomoć *docker-compose.yml* fajla. Definisanje *custom report*-ova je moguće uz pomoć *Kibana Query Language*-a (*KQL*), jer omogućava definisanje pravila za filtriranje i pretragu dokumenata, a za prikaz na *frontend*-u će se koristiti *IFrame*.

MinIO predstavlja *open source* distribuirano skladište za objekte u okviru koga će se *PDF* dokumenti čuvati u sirovom obliku. *Dependency* koji će biti korišćen, kao i povezivanje sa SpringBoot aplikacijom su prikazani na slikama 3 i 4.

```xml
<dependency>
    <groupId>io.minio</groupId>
    <artifactId>minio</artifactId>
    <version>8.5.5</version>
</dependency>
```

Slika 3. MinIO dependency

```java
@SpringBootApplication
public class UddApplication {

    public static void main(String[] args) {
        MinioClient minioClient = demo();
        try {
            List<Bucket> buckets = minioClient.listBuckets();
            System.out.println("Connection success, total buckets: " + buckets.size());
        } catch (MinioException e) {
            System.out.println("Connection failed: " + e.getMessage());
        } catch (Exception e) {
            e.printStackTrace();
        }

        SpringApplication.run(UddApplication.class, args);
    }

    private static MinioClient demo() {  1 usage
        MinioClient minioClient = MinioClient.builder()
                .endpoint("http://localhost:9000")
                .credentials( accessKey: "x3FTNltxbbOrxKcIkrWO",  secretKey: "qnB6aJlejdCIIrxB7LqUp6KW4vuHhzx7MglMhG8s")
                .build();

        return minioClient;
    }
}
```

Slika 4. MinIO konfiguracija

Indeksiranje i pretrage

SecurityIncident klasa sadrži polja pobrojana u specifikaciji, kao što je prikazano na slici 5, i biće korišćena za čuvanje podataka u okviru *PostgreSQL* baze.

```java
@Entity  2 usages
@Table(name = "securityIncident")
public class SecurityIncident {
    @Id  3 usages
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String fullName;  3 usages
    private String securityOrganizationName;  3 usages
    private String attackedOrganizationName;  3 usages
    private IncidentSeverity incidentSeverity;  3 usages
    private String location;  3 usages
```

Slika 5. SecurityIncident

Kako je potrebno omogućiti **geolokacijsku pretragu**, umesto da u indeksu bude sačuvano polje za lokaciju pogođene organizacije tipa *String*, čuvaće se polje tipa *GeoPoint*, pošto je uz pomoć *LocationAPI*-a na osnovu imena grada, odnosno adrese, i specificiranog radjiusa moguće dobiti informacije o geografskoj širini i dužini. Prema tome, na slici 7 se nalazi primer klase sa poljima koja će se naći u indeksu. Primer ovog tipa pretrage, preuzetog sa Vašeg predloga projekta, dat je na slici 6.

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

Slika 6. GeoDistanceQuery

Osim geolokacijske, potrebno je omogućiti i izvršavanje ***Approximate KNN* pretrage**, čemu je namenjeno polje *vectorizedContent*, a sama pretraga bi se vršila korišćenjem *CustomQuery*-a.

DatabaseId predstavlja id entiteta sačuvanog u *PostgreSQL* bazi, kako bi postojala povezanost sa perzistentnom verzijom podataka. *@Document* anotacijom se definiše naziv indeksa u okviru koga će se čuvati podaci, a koji će se kreirati ukoliko već ne postoji. *@Setting* anotacija definiše putanju na kojoj se nalazi *analyzer*. *Analyzer* definisan nad čitavom klasom znači da će biti korišćen za svako polje definisano u klasi.

```
@Document(indexName = "security_incident_index")  14 usages
@Setting(settingPath = "/configuration/analyzerConfig.json")
public class SecurityIncidentIndex {
    @Id  3 usages
    private String id;

    @Field(type = FieldType.Text, store = true, name = "full_name")  3 usages
    private String fullName;
    @Field(type = FieldType.Text, store = true, name = "security_organization_name")  3 usages
    private String securityOrganizationName;
    @Field(type = FieldType.Text, store = true, name = "attacked_organization_name")  3 usages
    private String attackedOrganizationName;
    @Field(type = FieldType.Text, store = true, name = "incident_severity")  3 usages
    private IncidentSeverity incidentSeverity;
    @Field(type = FieldType.Integer, store = true, name = "database_id")  3 usages
    private Integer databaseId;
    @Field(store = true, name = "location")  3 usages
    private GeoPoint location;
    @Field(type = FieldType.Dense_Vector, dims = 384, similarity = "cosine")  3 usages
    private float[] vectorizedContent;
```

Slika 7. SecurityIncidentIndex

Struktura *analyzera* je prikazana na slici 8. Koriste se filteri koji će čitav tekst konvertovati u mala slova, na isti način će biti tretirana slova sa i bez ,,kvačica'', poput č ć i c (*icu_folding*), ukoliko je tekst pisan na ćirilici, vršiće se konvertovanje u latinicu, izbacivaće se suvišne reči (*stop words*) i reči će biti skraćivane na njihov koren, ali ne nužno na pravi koren reči (*stemming*).

```
{
  "analysis": {
    "analyzer": {
      "serbian_simple": {
        "type": "custom",
        "tokenizer": "standard",
        "filter": [
          "lowercase",
          "icu_folding",
          "serbian_cyrillic_to_latinic",
          "stop",
          "serbian_stemmer"
        ]
      }
    },
    "filter": {
      "serbian_cyrillic_to_latinic": {
        "type": "icu_transform",
        "id": "Any-Latin; NFD; [:Nonspacing Mark:] Remove; NFC"
      },
      "serbian_stemmer": {
        "type": "stemmer",
        "language": "serbian"
      }
    }
  }
}
```

Slika 8. Analyzer konfiguracija

Da bi aplikacija mogla da koristi osnovne operacije nad indeksom, potrebno je kreirati repozitorijum koji nasleđuje *ElasticsearchRepository*, kao što je prikazano na slici 9.

```
@Repository  2 usages
public interface SecurityIncidentIndexRepository
        extends ElasticsearchRepository<SecurityIncidentIndex, String> {
}
```

Slika 9. SecurityIncidentIndexRepository

Osnovna pretraga

Potrebno je implementirati metodu u okviru *SecurityIncidentIndexRepository*-a. Ta metoda bi bila anotirana *@Query* anotacijom, u okviru koje bi bio definisan konkretan upit. Ovo je pogodno za implementaciju *full-text* pretrage. U slučaju **pretrage sadržaja *PDF***-a je potrebno prvo izvući sadržaj *PDF*-a, što je moguće uz pomoć *Apache Tika*, a zatim izvršiti *full-text* pretragu.

Kombinovana *boolean semi-structured* pretraga (uz podršku za *AND*, *NOT* i *OR* operatore)

Za kombinovanu *boolean semi-structured* pretragu će se koristiti *boolean query* koji ima podršku za korišćenje *AND*, *NOT* i *OR* operatora. Korisnik će za ovaj tip pretrage koristiti *search bar* koji će izgledati slično onim koje se nalaze u okviru *Stack Overflow*-a ili *Discord*-a. Sa *front*-a se korisnikov upit šalje na *backend*, parsira se i od njega se kreira *boolean query*.

Da bi se obezbedilo izvršavanje operatora kao u *C-like* jeziku, potrebno je konfigurisati poredak uz pomoć standardne matrice prioriteta. Poredak bi bio sledeći: *NOT* > *AND* > *OR*, a svakom od njih bi se dodeljivao odgovarajući prioritet od jedan do tri.

Phraze query

Elasticsearch omogućava pretragu po ključnim rečima, odnosno frazama, i logičkim operatorima uz pomoć *lucene query*-a. Zahvaljujući tome, moguće je realizovati pretragu tipa *phraze query*, gde bi korisnici u *search bar*-u pod znacima navoda naveli za koje pojmove, u redosledu u kojem su ti pojmovi navedeni, žele da pronađu podudaranje u dokumentima.

Prikaz rezultata

Dinamički prikaz sažetka sa istaknutim ključnim pojmovima je moguće realizovati korišćenjem *Elasticsearch Highligting API*-a. *Elasticsearch* nudi opciju za definisanje *highlight*-ovanih sažetaka na osnovu jednog ili više polja rezultata. Na takav način je moguće pokazati korisnicima gde postoje poklapanja između otkucanog i ponuđenog.

Za realizaciju dinamičkog prikaza sažetka će se koristiti *NativeSearchQueryBuilder* kojem se prosleđuju polja za *highlight*-ovanje i broj okolnih karaktera koji će se naći u sažetku.