Predlog projekta iz predmeta Upravljanje digitalnim dokumentima

Vujić Milica E2 19/2024

• Arhitektura Sistema

Sistem za monitoring i pretragu sigurnosnih incideneta će biti realizovan kao monolitna aplikacija sa klijentskim (frontend) slojem za čiji će razvoj biti korišćen Angular i severskim (backend) slojem koji će biti implementiran u Spring Boot-u. Komunikacija između frontend-a i backend-a će biti zasnovana na HTTP protokolu upotrebom HTTP zahteva. MinIO skladište će se koristi za čuvanje sirovih PDF dokumenata koji su povezani sa sigurnosnim incidentima. Elasticsearch će se koristiti kao sistem za pretragu i analizu, Logstash za prikupljanje i obradu podataka, a Kibana za vizualizaciju tih podataka. ELK Stack komponente će biti pokretane preko Docker kontejnera, kako se ne bi morale pojedinačno instalirati. U *docker-compose.yml* fajl-u će biti konfiguracija za pokretanje ELK stack-a i MinIO u Docker okruženju. Fajl koristi Docker Compose za orkestraciju ovih kontejnera koji rade zajedno. Svaka od komponenata je definisana kao servis unutar services sekcije. Parametri kao što su konfiguracija za Logstash (putanje fajlova) prosleđuju se putem environment varijabli i Docker volumes. U koliko bi sa ovom konfiguracijom došlo do problema, instalirala bih pojedinačno potrebne komponente.

• Autentifikacija i autorizacija

Za obezbeđivanje autentifikacije i autorizacije koristiće se Spring Security framework. Proces bi počinjao unosom kredencijala i slanjem zahteva na endpoint za login. Ovaj zahtev preuzima Spring Security, koji koristi AuthenticationManager za autentifikaciju. Kredencijali se prosleđuju komponenti UserDetailsService, koja pretražuje bazu podataka (relacionu PostgresSQL) kako bi pronašla korisnika, dok se unesena lozinka poredi sa hashovanom lozinkom u bazi pomoću BCryptPasswordEncoder. Ako su podaci ispravni, generiše se JSON Web Token (JWT) koji se vraća korisniku, koji ga čuva u local storage-u na frontend strani. Prilikom svakog narednog zahteva, token se šalje u Authorization header-u, gde ga Spring Security prepoznaje pomoću JWT filtera. Na osnovu informacija iz tokena, poput korisničkih uloga, Spring Security odlučuje da li korisnik ima pristup traženom resursu, a pravila pristupa se definišu u klasi konfiguracije sigurnosti koristeći metode kao što su .antMatchers. Korisnici bez validnog tokena imaju pristup samo enpointu za login. Na frontend strani bih dodatno postavila canActivate AuthGurad na rute koje nisu stranica za login (stranice za pretragu).

• MinIO

Kao open-source skalabilno objektno skladište koristi se u svrhe čuvanja nestruktuiranih PDF dokumenata o bezbednosnim događajima. Pokreće se putem docker-compose.yml fajla koristeći najnoviju verziju slike (:latest). Glavne operacije MinIO servisa su dostupne na portu 9000, dok je konzola dostupna na portu 9090. Pristup sistemu je omogućen pomoću environment varijabli kao što su MINIO_ROOT_USER i MINIO_ROOT_PASSWORD, koje predstavljaju korisničko ime i lozinku. U aplikaciji, klasa

MinioConfig je odgovorna za konfiguraciju MinioClient-a, koji se instancira kao Bean koristeći vrednosti iz fajla application.properties. Kada je MinioClient dostupan kao Bean, može se ubaciti u različite servise radi izvršavanja operacija poput dodavanja, preuzimanja ili brisanja dokumenata. (Implementacija kao sa primera projekta sa vežbi)

- ## Elasticsearch konfiguracija, Indeksiranje

Konfiguracija Elasticsearch-a omogućava vezu između Spring aplikacije i Elasticsearch servera pomoću Spring Data Elasticsearch modula. Ovaj modul koristi prilagođene anotacije za definisanje repozitorijuma indeksa i klijentske konfiguracije. Klijent se postavlja sa osnovnim informacijama kao što su URL, port i autentifikacija, dok @EnableElasticsearchRepositories omogućava skeniranje repozitorijuma u definisanom paketu.

```java
@Configuration
@EnableElasticsearchRepositories(basePackages = "com.example.udd_security_incidents.indexrepository")
public class ElasticsearchConfiguration
    extends org.springframework.data.elasticsearch.client.elc.ElasticsearchConfiguration {

    @Value("${ES_HOST:localhost}")
    private String host;

    @Value("${ES_PORT:9200}")
    private int port;

    @Value("${elasticsearch.username}")
    private String userName;

    @Value("${ES_PASSWORD:}")
    private String password;

    no usages
    @Override
    public ClientConfiguration clientConfiguration() {
        return ClientConfiguration.builder().connectedTo( hostAndPort: host + ":" + port)
            .withBasicAuth(userName, password).build();
    }
}
```

Tekstualna polja se inderksiraju na trivijalan način, dok se PDF dokumenta prvo čuvaju u miniIO skladištu, a zatim parsiraju uz pomoć Apache pdfbox biblioteke i potom se indeksiraju kao tekstualni sadržaj u Elasticsearch-u. @Document anotacija se koristi za definisanje Elasticsearch indeksa, @Field anotacija mapira svaki atribut klase na specifičan tip u Elasticsearch-u. Izgled klase indeksirajućeg objekta možete videti u nastavku. Atribut filePath predstavlja putanju (filename) od sačuvanog dokumenta koji bi služio za dohvatanje i preuzimanje dokumenta nakon uspešne pretrage.

```java
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
@Document(indexName = "incident_documents")
@Setting(settingPath = "/configuration/serbian-analyzer-config.json")

public class IncidentIndex {

    @Id
    private String id;

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

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

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

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

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

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

    @Field(type = FieldType.Keyword, store = true, name = "file_path")
    private String filePath; // Путања у MinIO

    @Field(type = FieldType.Dense_Vector, dims = 384, similarity = "cosine")
    private float[] vectorizedContent;

    @GeoPointField
    @Field(store = true, name = "organization_location")
    private GeoPoint organizationLocation;

}
```

- Geoprostorna pretraga

Indeksiran je i GeoPoint koji predstavlja adresu afektovane organizacije. Za konverziju tekstualne adrese u geografske koordinate koristiće se LocationIQ forward geocoding https://docs.locationiq.com/docs/search-forward-geocoding. Dobijene koordinate se zatim koriste kao ulaz u QueryBuilder, koji omogućava dodavanje geoDistanceQuery parametra u kreirani upit.

```java
public GeoDistanceQueryBuilder createGeoQuery(String address, double distanceInKm) {
    GeoPoint geoPoint = locationIQService.getCoordinates(address);

    return QueryBuilders.geoDistanceQuery( name: "organizationLocation")
            .point(geoPoint.getLat(), geoPoint.getLon())
            .distance(distanceInKm, org.elasticsearch.common.unit.DistanceUnit.KILOMETERS);
}
```

- ## Osnovna pretraga

Osnovna pretraga u Elasticsearchu omogućava dohvat dokumenata na osnovu jednostavnih upita. Ovo se obično postiže korišćenjem ElasticsearchRepository i metoda koje vrše pretragu spram naziva metode ili metoda sa @Query anotacijom gde bi se ručno napisao upit za tražene parametre, što je šira varijanta pomoću koje se mogu definisati custom upiti, ili upotrebom NativeSearchQueryBuilder-a za dinamičko kreiranje upita.

```
2 usages
public interface IncidentIndexRepository extends ElasticsearchRepository<IncidentIndex, String> {


    1 usage
    Page<IncidentIndex> findIncidentIndexByEmployeeNameOrEmployeeSurnameOrSeverity(
            String name, String surname, String severity);

    1 usage
    Page<IncidentIndex> findIncidentIndexByAffectedOrganizationOrSecurityOrganization(
            String affectedOrganization, String securityOrganization);


}
```

- ## Dinamički sažetak

Ako se upit postavi dinamički upotrebom NativeSearchQueryBuilder-a, onda je moguće highlight-ovati specifična polja definisana u upitu kako bi se istakli relevantni delovi teksta u rezultatima pretrage. Highlightovanje je moguće realizovati korišćenjem HighlightBuilder.Field, gde su definisani parametri poput veličine fragmenta (fragmentSize) i HTML oznaka za formatiranje (preTags i postTags), čime se omogućava prikaz samo onih delova teksta koji sadrže ključne pojmove iz pretrage, u jasno naglašenom obliku.

```
public SearchHits<IncidentIndex> searchWithHighlight(String affectedOrganization, String securityOrganization, Pageable pageable) {
    var query = QueryBuilders.boolQuery()
            .should(QueryBuilders.matchQuery( name: "affectedOrganization", affectedOrganization))
            .should(QueryBuilders.matchQuery( name: "securityOrganization", securityOrganization));

    var searchQuery = new NativeSearchQueryBuilder()
            .withQuery(query)
            .withHighlightFields(
                    new HighlightBuilder.Field( name: "affectedOrganization").fragmentSize(50).preTags("<em>").postTags("</em>"),
                    new HighlightBuilder.Field( name: "securityOrganization").fragmentSize(50).preTags("<em>").postTags("</em>"))
            .withPageable(pageable)
            .build();

    assert elasticsearchTemplate != null;
    return elasticsearchTemplate.search(searchQuery, IncidentIndex.class);
}
}
```

- ## Boolean query

Sistem za pretragu će da koristi kombinaciju Shunting-yard algoritma i dinamičkog kreiranja Elasticsearch upita kako bi podržao složene pretrage sa logičkim operatorima AND, OR i NOT. Uneti izraz se obrađuje u tri ključna koraka: tokenizacija, konverzija u postfiksnu notaciju i generisanje BoolQuery upita u skladu sa operatorima i operandima. Tokenizacija razdvaja operatore, operande i zagrade, dok Shunting-yard algoritam omogućava pravilno raspoređivanje operatora prema prioritetu (NOT > AND > OR), uz mogućnost redefinisanja prioriteta zagradama. Rezultat algoritma je niz u postfiksnoj notaciji, što omogućava jednostavnu evaluaciju izraza pomoću steka. Generisanje upita za Elasticsearch oslanja se na korišćenje odgovarajućih tipova upita za svaki token. Za osnovne termine koristi se MatchPhraseQuery, koji pretražuje polje u dokumentima. Ovo polje sadrži indeksirani sadržaj tekstualnih dokumenata koji se pretražuju. Logički operatori se mapiraju na Elasticsearch BoolQuery: AND se prevodi u must, OR se prevodi u should, NOT se prevodi u must_not.

Primer

Unos: ("A:employeeName " OR " B:employeeSurname ") AND NOT " C:affectedOrganization " na frontu u input polju

Rezultat tokenizacije: ["A:employeeName", "OR", "B:employeeSurname", "AND", "NOT", "C:affectedOrganization"].

Rezultat transformacije u postfiksnu notaciju (RPN): ["A:employeeName","B:employeeSurname", "OR", "C:affectedOrganization","NOT", "AND"].

Rezultat generisanja Elasticsearch upita: Postfiksni niz se obrađuje stek mašinom:

-"A:employeeName": Kreira MatchPhraseQuery za employeeName polje.
-"B: employeeSurname ": Kreira MatchPhraseQuery za employeeSurname polje.
-"malver:description": Kreira MatchPhraseQuery za description polje.
-"OR": Kombinuje prva dva uslova sa should.
-"NOT": Primenjuje must_not na "malver:description".
-"AND": Kombinuje prethodne uslove sa must.

```
private Query buildSearchQuery(List<String> tokens) {
    Stack<Query> stack = new Stack<>();
    Map<String, Integer> operatorPriority = Map.of( k1: "NOT", v1: 3, k2: "AND", v2: 2, k3: "OR", v3: 1);

    for (String token : tokens) {
        if (operatorPriority.containsKey(token)) {
            Query right = stack.pop();
            Query left = token.equals("NOT") ? null : stack.pop();
            stack.push(combineQueries(left, right, token));
        } else {
            String[] parts = token.split( regex: ":");
            String phrase = parts[0];
            String field = parts.length > 1 ? parts[1] : "employeeName"; // Podrazumevano polje "employeeName"
            stack.push(buildMatchQuery(phrase, field));
        }
    }
    return stack.pop();
}
1 usage
private Query combineQueries(Query left, Query right, String operator) {
    return BoolQuery.of(b -> {
        switch (operator) {
            case "AND": b.must(left).must(right);
                break;
            case "OR": b.should(left).should(right);
                break;
            case "NOT": b.mustNot(right);
                break;
        }
        return b;
    })._toQuery();
}
```

- Phrase query

Elasticsearch implementira Phrase Query pomoću MatchPhraseQuery. Ovaj upit se koristi u poljima koja indeksiraju tekstualni sadržaj i pruža opciju za određivanje slobode u kretanju reči pomoću slop parametra, koji omogućava da reči u frazi budu razdvojene do određenog broja drugih reči.

```
1 usage
private Query buildMatchQuery(String phrase, String field) {
    return MatchPhraseQuery.of(m -> m
            .field(field)
            .query(phrase)
            .slop( value: 2)
    )._toQuery();
}
```

- KNN

Approximate KNN pretraga koristi vektorsku reprezentaciju teksta kako bi pronašla dokumente koji su semantički slični unosu u pretraživačkom polju. Iskoristila bih kod koji ste dali u materijalima sa vežbi za realizaciju. Prvo bi se vršila vektorizacija teksta - tekstualni sadržaj pretraživačkog upita se prvo transformiše u vektorski format (koristeći NLP model), zatim koverzija float[] u List<Float> radi kompatibilnosti sa Elasticsearch KNN

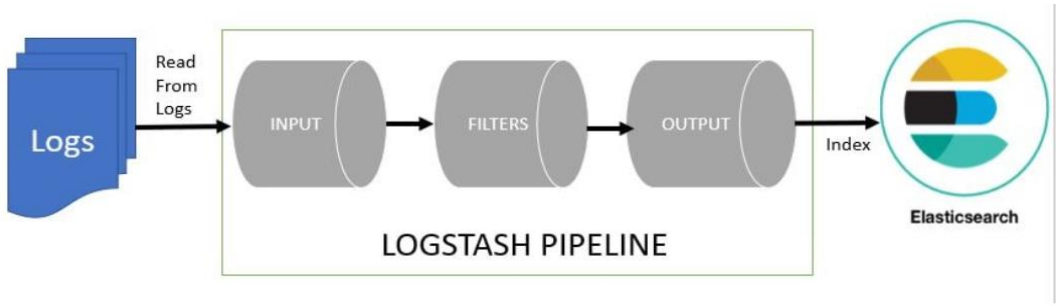
API-jem. Kreirao bi se KNN upit sa poljem vectorizedContent, koje sadrži unapred generisane vektore dokumenata. Elasticsearch pretražuje indeks i vraća dokumente sa najvišim kosinusnim sličnostima.

- ## Pretprocesiranje upita pomoću SerbianAnalyzer-a

Analyzer u Elasticsearch-u služi za obradu tekstualnih podataka pre nego što se oni indeksiraju, omogućavajući pretrage koje su preciznije i relevantnije. Svrha analizatora je da transformiše ulazni tekst u tokene i da primeni različite filtere kako bi optimizovao pretragu. Serbian analyzer sam preuzela sa Language analyzers | Elasticsearch Guide [8.16] | Elastic. Koristi standardni tokenizator koji deli tekst na osnovne jedinice na osnovu razmaka i interpunkcije. Zatim, primenjuje filter lowercase koji konvertuje sve tokene u mala slova kako bi pretraga bila neosetljiva na velika/mala slova. Filter serbian_stop uklanja česte reči u srpskom jeziku, poput "i", "ali", "da", koje ne doprinose relevantnosti rezultata. Sledeći filter, serbian_keywords, štiti određene ključne reči od daljeg obrade, poput stemming-a, kako bi se osiguralo da ostanu netaknute. Filter serbian_stemmer sprovodi stemming, odnosno redukuje reči na njihov osnovni oblik, što omogućava pronalaženje rezultata bez obzira na gramatički oblik unosa. Na kraju, filter serbian_normalization dodatno prilagođava tekst standardima srpskog jezika, poput zamene latiničnih "dj" sa "đ" ili uklanjanja nepotrebnih dijakritičkih znakova. Na primer, korisnik koji pretražuje reč "primer" može dobiti odgovarajuće rezultate čak i ako je reč u indeksu u drugom gramatičkom obliku, poput "primerima" ili "primera".

- ## Logstash

Omogućava prikupljanje, transformaciju i prenos podataka ka odredišnim sistemima, ovde konkretno ka Elasticsearch-u. U sistemima za praćenje i analizu sigurnosnih incidenata, koristi se za filtriranje i formatiranje logova pre njihovog skladištenja. U Docker okruženju, Logstash se definiše kao poseban servis u okviru *docker-compose.yml* fajla, dok se njegova funkcionalnost konfiguriše u datoteci *logstash.config*. Ova konfiguracija sadrži tri glavne sekcije: input, za definisanje izvora podataka; filter, za primenu pravila transformacije; i output, gde se obrađeni podaci prosleđuju odredištu, u ovom slučaju Elasticsearch-u. Log zapisi generisani tokom različitih procesa, poput kreiranja sigurnosnih incidenata, čuvaju se u *.log* fajlovima, koji služe kao ulaz za Logstash. Parametar *path* u *file* input sekciji precizno definiše lokaciju ovih fajlova. S obzirom na to da su logovi nestruktuirani, neophodna je transformacija u struktuirani format pre nego što se proslede Elasticsearch-u. Za ovu svrhu koristi se Grok filter, alat koji omogućava izdvajanje relevantnih informacija iz nestruktuiranog teksta. Ovaj filter se prilagođava specifičnom formatu podataka kako bi izvukao ključne vrednosti, poput nivoa greške ili korisničkog ID-a. Nakon što podaci budu transformisani i struktuirani, Logstash ih prosleđuje Elasticsearch-u kroz *output* sekciju, čime se omogućava efikasna analiza i pretraga sigurnosnih incidenata. Logovanje bih dodala u servise. Postupak je detaljno opisan na [ELK + Spring Boot: A Guide to Local Configuration | Cloud Native Daily].



- ## Kibana

Kibana je alat za vizualizaciju podataka koji omogućava jednostavnu integraciju sa Elasticsearch-om, pružajući interaktivan prikaz podataka smeštenih u indeksima. Koristi JSON DSL za formiranje upita, kao i Kibana Query Language (KQL), koji olakšava pretragu i filtriranje podataka. Kibana podržava dinamičko osvežavanje sadržaja kako bi u realnom vremenu prikazala najnovije informacije iz Elasticsearch-a. Jedan od ključnih elemenata Kibane su prilagodljivi izveštaji i dashboard-i, koji se mogu kreirati iz Logstash-ovih datastream-ova uz KQL. Ovi dashboard-i se mogu jednostavno integrisati u front-end aplikacije putem HTML iFrame elemenata. Kibana se, poput ostalih servisa u ELK Stack-u, konfiguriše kao zaseban servis unutar *docker-compose.yml* fajla, gde se definišu portovi i povezanost sa Elasticsearch-om. Konfiguracija uključuje mapiranje porta 5601 za pristup korisničkom interfejsu. Vizuelizacije u Kibani bih realizovala po uputstvu sa [Kibana - Create Visualization].